LETTER_RECOGNITION

Report

Abstract: This data set includes 20000 data and the feature of these data were summarized 16 different numerical attributes, and different combination of these 16 attributes can represent different English capital letters， and my training model can predicate the letters by the 16 different numerical attributes. And when comparing the model trained by Pavlov, Popescul, Pennock and Ungar(2003), it can be easily found that my model is worse than theirs as my model accuracy can not reach that high value as theirs.

Keywords：data normalization, activation function, optimizer, batch gradient descent

1 Introduction

With the development of modern science and technology, data statistics and analysis techniques have been widely applied to various fields. According to Paliouras and S.Bree(2005), quantities of empircM concept learning algo-rithms have been improved since two decades ago. Also, when human experts faced with difficult situation, they always can treat these problems as special cases of familiar examples by classifying and analyzing them and then apply known solutions to work it out (de Groot, Chase & Simon, cited in W.Fery, J.Slate, 1991). As a result, I planned to find a suitable data set to check the effect of using digital features in the training set, which could help me realize the power of analytics in some degree. Finally, the data set I chose is called 'Letter Recognition Data Set', has 20000 characters which were produced by 20 randomly distorted and different fonts. And these 20 fonts could make up different character images and each of them would be identified as one of 26 English capital letters. During the whole process, I need to select 16000 data as training data and the rest of the data set as testing data, and after I finish training model, I can use the testing data to check the accuracy of my model.

2 Method

1) randomly extract data and divide it into training and test sets
   The dataset I chose has only one overall data, so as the title requires, I need to select 16000 data as training data and the rest of the data as testing data which will be helpful to examine the accuracy of my model. In this section, I used the random sampling method to extract data to make the data predicted by my model more representative. In addition, I chose different data as training and test sets to prevent

data overfitting.

```
my_matrix = data.as_matrix()
x = my_matrix[:,1:17]
y = my_matrix[:,0]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
traindata = np.column_stack((y_train,x_train))
traindata = pd.DataFrame(traindata)
testdata = np.column_stack((y_test, x_test))
testdata = pd.DataFrame(testdata)
```

2) change the column name and convert string target values to numeric values

First, I will show the attribute information of this datset.

Letter	capital letter	(string)
x-box	horizontal position of box	(integer)
y-box	vertical position of box	(integer)
width	width of box	(integer)
high	height of box	(integer)
onpix	total # on pixels	(integer)
x-bar	mean x of on pixels in box	(integer)
y-bar	mean y of on pixels in box	(integer)
x2bar	mean x variance	(integer)
y2bar	mean y variance	(integer)
xybar	mean x y correlation	(integer)
x2ybr	mean of x * x * y	(integer)
xy2br	mean of x * y * y	(integer)
x-ege	mean edge count left to right	(integer)
xegvy	correlation of x-ege with y	(integer)
y-ege	mean edge count bottom to top	(integer)
yegvx	correlation of y-ege with x	(integer)

According this form, the type of target values is string, I should convert them to numeric values at first, then the outputs can be hold by Tensor in Variables.

3) data preprocessing

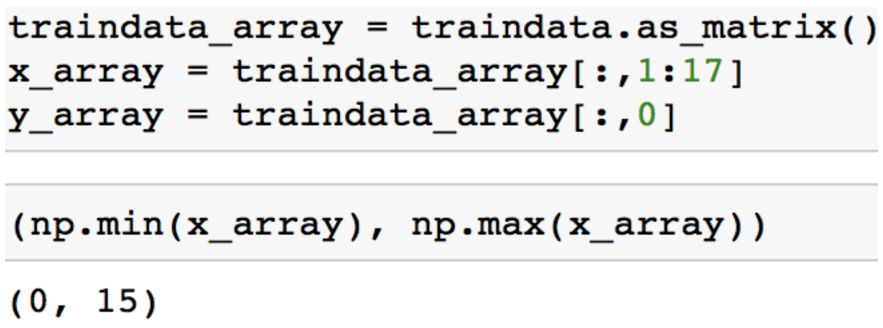
Here I will show my trial process.

- Firstly I want to apply stratified sampling method to my first step, extracting data, to train a more accurate model. However, after I counted the number of each values of target columns, I found the number of different class almost the same both in training and testing sets, hence I realized that stratified sampling methods could make little attributes to improving testing accuracy.

```
count_t = traindata['l_capital_letter'].value_counts()
count_t
```

- Then I read one pdf file called DecryptGISData camera form Papers for NN4, it told us that normalizing data over the range 0 - 1 for the network from logistic aspect can help to deal with the unreliable data and get more accurate prediction, so I tried to normalize training input data by columns. Importantly, we cannot normalize the target data because if we normalize the target data, we will get a Tensor hold outputs whose values are changing to 0 or 1 as a result of .long() function, and in that case our model will be become extremely inaccurate. However, after I normalized my training input data and testing input data, I found my model accuracy has not increase d, but dropped. Next part will show testing results.

- Also, this pdf file also told me to remove bias of lowest and highest values by using statistical Z function to reduce noise. I thought this method is feasible until I check the minimum value and the maximum value of the values of input data in training set and testing set.



```
traindata_array = traindata.as_matrix()
x_array = traindata_array[:,1:17]
y_array = traindata_array[:,0]
```

```
(np.min(x_array), np.max(x_array))
```

```
(0, 15)
```

4) I used different activation function (sigmoid, tanh, relu) for hidden layer to train my model, and examine my model accuracy.
5) Also, I tried to use different optimizer to train my model, which could help me to get a more accurate model.
6) After reading HeurPatRed pdf file, I thought I could try to apply batch gradient descent method to training my model. And batch gradient descent is a method to calculate error for each data in the training set, and the model will be updated after all the training data is calculated, which could help us get a more stable error gradient and a more stable convergence point. Also this method reached the purpose of separating error calculation and model update process, which is conducive to the implementation of parallel algorithms. However, I found testing accuracy of my model representing a extremely low value after I use this method, and I was very confused about it and cannot figure out why.

3 Result and Discussion

1. Test with a few different simple parameters

- Results

Test number	Data preprocessing	Number of neurons for hidden layer	Learning rate	Number of epoch on training	Testing accuracy
1	No	10	0.01	500	5.95%

2	No	10	0.01	2000	19.23%
3	No	14	0.01	2000	25.3%2
4	No	14	0.01	5000	42.83%
5	No	100	0.01	5000	63.52%
6`	Yes(normalize data)	100	0.01	5000	16.98%

- Discussion

From the results of above six tests, we can conclude that for my training model, with the increasing of the number of neurons for hidden layer, the testing accuracy of my model show an increasing trend as well. Also, the number of epoch on training is bigger, the testing accuracy will be higher. As a result, the number of neurons for hidden layer and epoch on training can affect the model accuracy. However, we can find an interesting fact that after I normalized data before training them, my model accuracy showed a decreasing trend, which made me confused. While the other day I read some information related to this data set, I found that every stimulus was converted into 16 primitive numerical attributes and then scaled to fit into a range of integer values from 0 through 15, so I thought that maybe lead to the situation in test 6.
When comparing to the results from paper called Mixtures of Conditional Maximum Entropy Models, we can find with the rising number of attributes, the testing accuracy would represent a increasing trend, and I have to admit their model is better than me as most of the testing accuracy of their model are higher than mine, which could reach 82.2% (Palov, popescul, Pennock, Ungar 2003). To improve my model, I still have lots of work to finish.

2. Test with different activation function and different optimizer (all without data preprocessing)

- Results

Test number	Number of neurons for hidden layer	Learning rate	Number of epoch on training	Type of activation function	Type of optimizer	Testing accuracy
7	100	0.01	2000	sigmoid	SGD	53.83%
8	100	0.01	2000	tanh	SGD	70.67%
9	100	0.01	2000	relu	SGD	76.9%

10	100	0.01	2000	sigmoid	ASGD	51.52%
11	100	0.01	2000	sigmoid	Adam	95.2%
12	100	0.01	2000	sigmoid	Adamax	94.88%

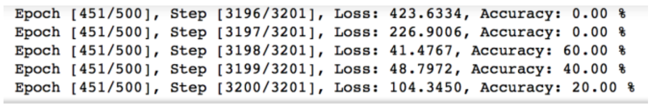
● Discussion

From above 6 tests we can find that the activation function for hidden layer which called tanh or relu can make more attributes to training a more accurate model. Also, the optimizer called Adam or Adamax can make our model becaome extremely accurate, however, I thought these situation happened because these two optimizer could make my training data become overfitting.

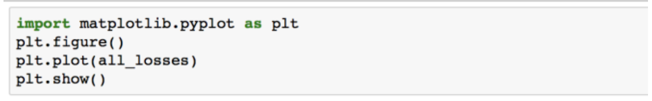
3. Using batch gradient descent

Define input_size = 16, hidden_size = 100 , num_classes = 26, num_epochs = 500, batch_size = 5, earning_rate = 0.01

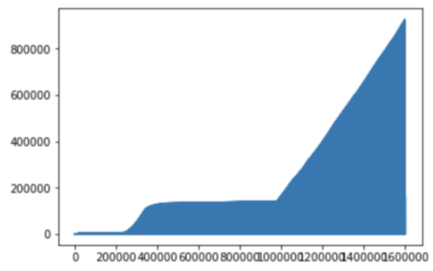
Followiong pictures show the results

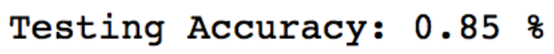


```
Epoch [451/500], Step [3196/3201], Loss: 423.6334, Accuracy: 0.00 %
Epoch [451/500], Step [3197/3201], Loss: 226.9006, Accuracy: 0.00 %
Epoch [451/500], Step [3198/3201], Loss: 41.4767, Accuracy: 60.00 %
Epoch [451/500], Step [3199/3201], Loss: 48.7972, Accuracy: 40.00 %
Epoch [451/500], Step [3200/3201], Loss: 104.3450, Accuracy: 20.00 %
```



```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(all_losses)
plt.show()
```





```
Testing Accuracy: 0.85 %
```

4 Conclusion and Feature Work

I changed different parameters like the number of neurons for hidden layer and the number of epoch on training to train my model, and I changed the activation function for hidden layer to train my model to, and also I changed the type of optimizer to train model as well. I applied all above methods to obtaining a more accurate model and some operations really did work. During my trial process, I found that normalizing data before training is useless to make training model predicate values more accurate. And also I thought I was failed to apply batch gradient descent method to training my model. So as for my future work, I should and I will work ahead on figuring out this problem and moreover I will try some other methods to improve my model. Moreover, in the future work, I want to apply bimodal distribution removal method successfully to training model, I found it is very interesting but sadly this time I am failed to turn theory method into real code and make it work.

- Reference list

1. Gedeon, T. D. (1995, November). Indicators of hidden neuron functionality: the weight matrix versus neuron behaviour. In *Artificial Neural Networks and Expert Systems, 1995. Proceedings, Second New Zealand International Two-Stream Conference on* (pp. 26-29). IEEE.
2. Gedeon, T. D., & Bowden, T. G. (1992). Heuristic pattern reduction. In *International Joint Conference on Neural Networks* (Vol. 2, pp. 449-453).
3. Frey, P. W., & Slate, D. J. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine learning*, *6*(2), 161-182.
4. Pavlov, D., Popescul, A., Pennock, D. M., & Ungar, L. H. (2003). Mixtures of conditional maximum entropy models. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)* (pp. 584-591).