Final Project Report
Classifying Images of Handwritten Digits

Jiaqi Chen

1. Introduction

This project is aimed to use three different types of classification algorithms to classify images that contain binary images of handwritten digits. The classification algorithms are ridge regression, SVM, and Neural Network with three layers. The dataset is a benchmark dataset of MNIST. In the data set, each image has 28*28 pixels of which the value is from 0 to 255 and a label from 0 to 9, which can be used to train the model. The pixel value is used to be the feature. The testing data set in MNIST is used to evaluate the performance of the different algorithms. By comparing the classification accuracy of the algorithms, we can obtain the final performance estimate. The test results are analyzed and the performance of different algorithms are compared. The link to the project github page is https://github.com/Jiaqichengood/ECE532-Project.

2. Dataset

In this project, the MNIST dataset is used to do the classification. The link to the dataset is http://yann.lecun.com/exdb/mnist/. The MNIST data set is generated from images of handwritten digits shown in Fig.1. The MNIST training set has 60,000 examples. The testing set has 10,000 examples. Each example has a 28 by 28 pixel grayscale handwritten digital image and corresponding labels from 0 to 9. The image pixel value is from 0 to 255. The 60,000 pattern training set contained examples from approximately 250 writers. And the testing set is written by disjoint writers. The first 5000 testing data is cleaner and easier than the last 5000. In this project, we use the first 10000 training data examples and the first 1000 test data examples of the MNIST dataset as the training and testing data set. The goal of the project is to use the training data set with different classification algorithms to train the classifier, to let to classifier can classify the handwritten digital image with an output of 0 to 9.

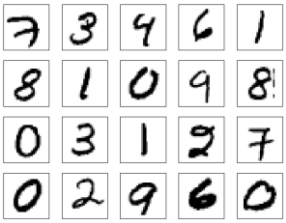

Fig.1 Images in MNIST data set

3. Methods

3.1 Data Processing

First of all, in the MNIST data set, there were 60,000 samples in the training set and 5,000 in the testing set, we need to read the data and use part of them in our project. For the MNIST data set, all images are normalized to 28*28 pixels, with pixel values

ranging from 0 to 255, with 0 representing the background and 255 representing the foreground. Here I use the unpack function in the struct module and the Numpy in Python. The struct module can process binary data stored in files, and the transformation of binary files can be achieved through the unpack function of this module. np.fromfile is a function that reads binary data using a known data type. It returns an array. After I read the file by np. fromfile, it returns a one-dimensional array, which means the data has 28*28=784 features. We need to take out every image, so I use the reshape function. Then I choose the first 10000 training data examples and the first 1000 test data examples of the MNIST dataset as the training and testing data set. Then I normalize the pixel value of the image to 0.0~1.0 by dividing 255.

3.2 Ridge Regression

To achieve the multiple classification, I define a function to convert the label to one hot encoding. That is for each data, if it belongs to class i, the i^{th} element is 1 and other elements are 0. For example, for the number 1, its one hot encoding is [0 1 0 0 0 0 0 0 0 0]. Then I use the linear regression method ridge regression to achieve the classification. To point out that, in the initial proposal, I planned to use the least squares method to do the classification. But when I applied the least squares method to train the model, it showed the feature matrix is singular. So I turned to use the ridge regression to train the model. Here I achieved the multiclass classification by train the weight k for class k by one- vs- rest. The label is 1 if the data is in class k, otherwise is 0. Then I used the ridge regression with 10000 training data to calculate the weight matrix w (since there are 10 classes). The lambda is chosen by cross validation. The training time is recorded. Next, I calculated the predicted label by test feature*w with 1000 test data. The class with the largest value is the predicted class. Then I calculated the accuracy by comparing the predicted class and the correct class (it has been provided in the test data set, which is the test data label). In the testing cases, I changed the lambda in the ridge regression to analyze the impact of the lambda parameter.

3.3 SVM

In this program, I used sklearn to complete the classification. First I defined two functions to read the labels and the features. The training data set includes 10000 data examples. The features is 10000*784, and the label is 10000*1, which value is 0~9. In the code, I used GridSearchCV to introduce N_jobs to the CPU for multithreading. When N_jobs =-1, the number of parallelism of the program will be the same as the number of cores of the CPU, thus greatly accelerating the running of the program. Here I use the parameters to add the corresponding parameter. The settings are default settings. Then I used the fit function to train the model. The training time is recorded. The predict function is used to predict the test labels. Then I compared the predicted labels with the test labels to test the accuracy. In the testing cases, I changed the type of kernel and the parameters in SVM method to analyze their impact.

3.4 Neural Networks

In this section, I used the neural network to do the classification. First, I read the data

as the method in section 3.2 and converted the labels to one hot encoding. Then I designed the neural network with 3 layers (one input layer, one hidden layer, and one output layer). Since the data has 784 features and 10 labels, the input layer has 784 nodes and the output layer has 10 nodes. The number of nodes in the hidden layer is determined by cross validation. I use the backpropagation to determine the weight values. The step size and epoch number are also determined by cross validation. Next, I used the training data set to train the model. The training time is recorded. The accuracy neural network model was tested using the testing data set. I calculated the predicted label by test feature*w with 1000 test data. The class with the largest value is the predicted class. Then I calculated the accuracy by comparing the predicted class with the correct class. In the testing cases, I changed the number of nodes in the hidden layer, step size, and epoch number to analyze the impact of these parameters.

4. Results

4.1 Ridge Regression

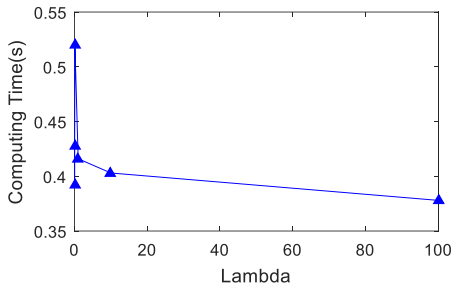

Fig.2 Computing time of ridge regression

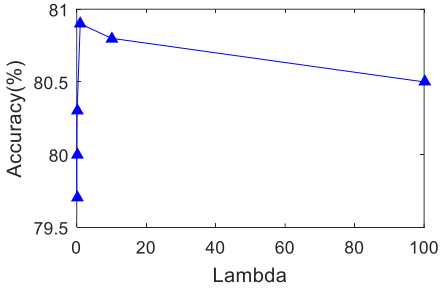

Fig.3 Accuracy of ridge regression

In the testing cases, I changed the lambda in the ridge regression to analyze the impact of the lambda parameter. Fig.2 illustrates the computing time of different cases. It can be observed that the computing time of ridge regression is short since here I just used the matrix calculation to calculate the weights. The computing time is not different too much between different lambdas since the computational complexity is not changed. From Fig.3, we can observe that the accuracy is the highest when lambda is 1. The highest classification accuracy is 80.9%.

4.2 SVM

TABLE.I Computing Time and Accuracy of SVM

	rbf	Linear	poly(degree 3)	poly(degree 5)	sigmod
Computing Time (s)	96.078	54.659	97.65	139.447	98.876
Accuracy (%)	94.7	90.7	93.6	87.2	81.3

In the testing cases, I changed the kernel function in the SVM to analyze the impact of different kernels. TABLE.I illustrates the computing time and the accuracy of different cases. It can be observed that the computing time of the linear kernel is the shortest. The accuracy is the highest when the kernel is a Gaussian function. The highest

classification accuracy is 94.7%. Especially, the accuracy with the poly function of degree 5 is lower than that of degree 3. Since there might be an overfitting problem when the degree is high. The computing time is increasing when the degree increases. Since the computational complexity increases.

4.3 Neural Networks

In the testing cases, I changed the number of nodes in the hidden layer, step size, and epoch number to analyze the impact of these parameters.

Fig.4 and Fig.5 illustrate the computing time and accuracy with different step sizes. The hidden node number is 200 and the epoch number is 5. It can be observed that the computing time is not different too much since the computational complexity is not changed. When the step size is between 0.1 and 0.3, the classifier has good performance. The highest classification accuracy is 95.3% with a step size of 0.2.

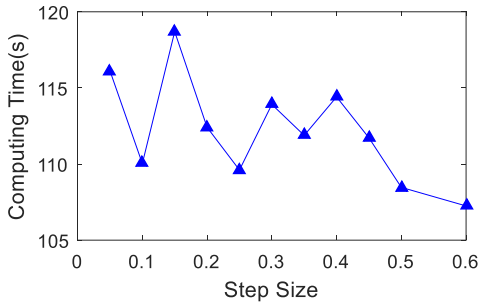

Fig.4 Computing time of neural networks with different step sizes

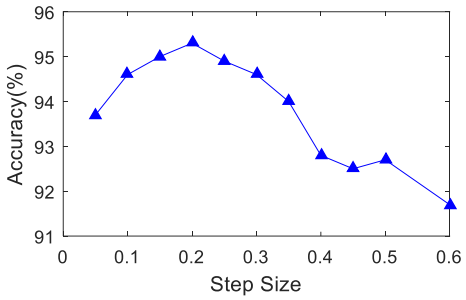

Fig.5 Accuracy of neural networks with different step sizes

Fig.6 and Fig.7 illustrate the computing time and accuracy with different epoch numbers. The hidden node number is 200 and the step size is 0.2. It can be observed that the computing time increases as the epoch number increases. The performance of the training model can be improved by increasing the epoch number, so the accuracy in the test cases increases. However, too much epoch number might lead to the overfitting problem, thus reducing the classification accuracy of the trained model. Among these test cases, when the epoch number is 9, the classification accuracy is the highest with 96.0%. (Note the randomness of stochastic gradient descent will lead the variance among different test running).

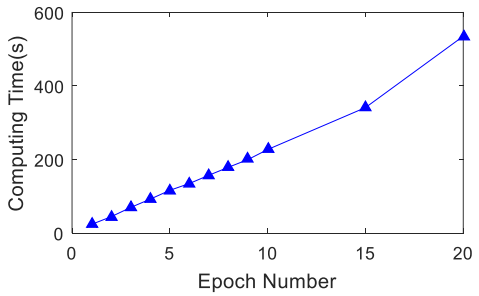

Fig.6 Computing time of neural networks with different epoch numbers

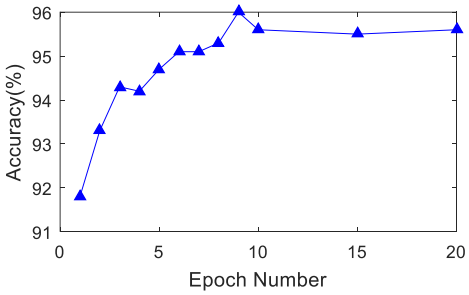

Fig.7 Accuracy of neural networks with different epoch numbers

Fig.8 and Fig.9 illustrate the computing time and accuracy with different hidden node numbers. The epoch number is 9 and the step size is 0.2. It can be observed that the computing time increases as the hidden node numbers increases. The performance of the training model can be improved by increasing the hidden node number, so the accuracy in the test cases increases. Among these test cases, when the hidden node number is 200, the classification accuracy is the highest with 95.5%. However, if too many hidden nodes are selected, it is difficult to train the network because there are too many paths to choose. Therefore, more epochs need to be set to train the network, which will greatly increase the computing power and time consuming. Therefore, we need to select a certain number of hidden layer nodes for training within tolerable running time. In our project, we can choose the hidden node number with 200.

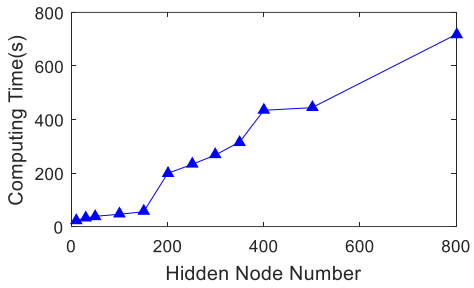

Fig.8 Computing time of neural networks with different hidden node numbers

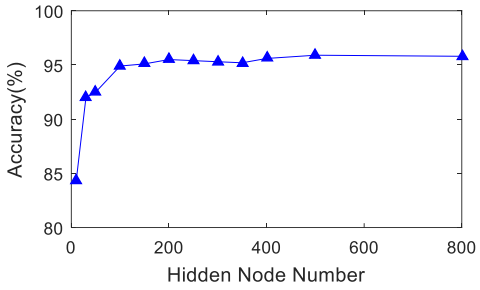

Fig.9 Accuracy of neural networks with different hidden node numbers

5. Conclusion

In this project, I use ridge regression, SVM, and neural network methods to train the classifiers to classify the images with handwritten digits. The computing time and accuracy are compared in the test cases using the testing data set. The test results illustrate that the SVM and neural networks have better performance compared with the ridge regression. However, the parameters and the kernel type in SVM and the parameters in the neural network method can affect the accuracy of the classification. The computing time is high when there are too many iterations or the computational complexity is high. So we need to select an appropriate model and parameters for training within tolerable running time. Moreover, the overfitting problem should also be considered when designing classifiers.