Digit recognizer

Qian Zhang Karan Dharni
qzhang348@wisc.edu dharni@wisc.edu
Computer Sciences department
University of Wisconsin-Madison

Abstract

In this project we trained and tested a set of classifiers for pattern analysis in solving handwritten digit recognition problems, using MNIST database. The classifying methods implemented and tested include Ridge Regression, Support Vector Machine (with Principal Component Analysis), K-Nearest Neighbors, Neural Network and Random Forests.

1 Introduction

Description of Dataset The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, which is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Visualization of Dataset A bunch of examples are plotted as Figure 1 below to help in visualizing the data set. Data distribution is shown in Figure 2, which shows that the samples are uniformly distributed.

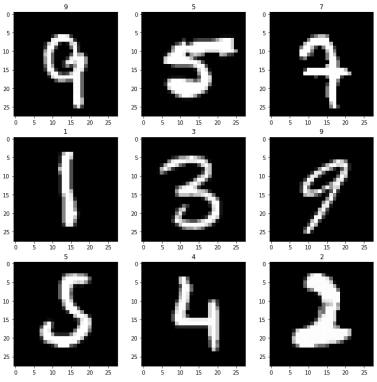

Figure 1: Visualization of samples

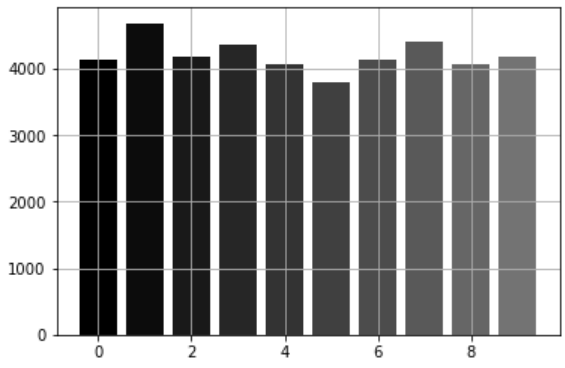

Figure 2: Data distribution

Methodology Various models will be compared to each other based on the accuracy of prediction. The accuracy will be based on error-rates generated by the models. For each model, the error rate has been calculated by dividing the dataset into training and test datasets. The training data is fed to each model to calculates respective weights/parameters. Then based on the number of correct predictions on the test data, the accuracy is calculated.

2 Models

2.1 Ridge Regression & Multi-class Classification

Ridge regression can be used in binary classification problem, and in this classification problem, the number of classes is 10, so we can design 10 binary ridge regression classifiers to solve this classification problem. Take 3 classifiers as example, our idea of ridge regression and multi-class classification is shown in Figure 3.

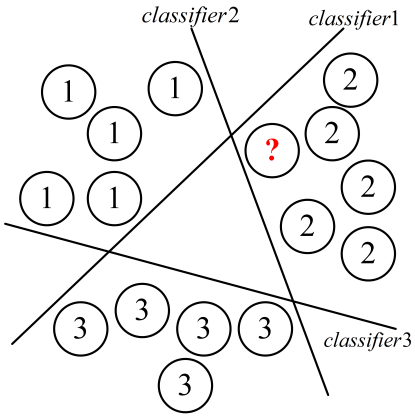

Figure 3: Ridge regression & Multi-class clas-sification

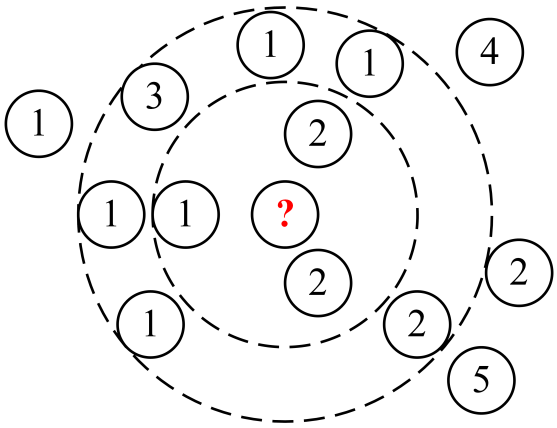

Figure 4: K-Nearest Neighbors algorithm

Take classifier 0 as an example, for all training samples, if their labels are 0, then we set their labels to +1, otherwise, we set their labels to -1, and then we can formulate least problem from these classifier.

$$\hat{w} = \arg\min_{w} \|X \cdot w - y\| + \lambda \cdot \|w\|^2 = (X^T \cdot X + \lambda \cdot I)^{-1} \cdot X^T \cdot y$$

Where X is our training data, and y is the label, and $\hat{w}$ is the weight solved. And we do the same thing for classifier 1-9, and when we have a new sample, we use 10 different w from 10 classifiers to calculate the labels of the new sample, and we use the largest label calculated to decide the label of new sample.

2.2 K-Nearest Neighbors Algorithm

The k-nearest neighbors algorithm (k-NN) is one of the non-parametric methods used for classification. The idea of k-NN is straight forward. In k-NN classification, an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). Our idea of using k-NN in this classification problem is shown in Figure 4. If we choose k=3, then we choose 3 training samples, which are closer to new sample than other training samples, as shown in the first circle. In the first circle, we let the closest training samples vote for new sample, and because label 2 is the majority, we assign label 2 to the new sample. And if we choose k=8, then we choose 8 training samples, which are closer to new sample than other training samples, as shown in the second circle. In the second circle, we let the closest training samples vote for new sample, and because label 1 is the majority, we assign label 1 to the new sample. In this project, since the data are in 784 dimensional space, the computational cost is relatively high. It is sensible to pick a small k value. we run the algorithm for k from 1 to 30 with 2 as step size and compute the classification accuracy on the training data, and choose k which achieves best classification performance.

Algorithm 1 K-Nearest Neighbors algorithm

Input: Training samples, new sample, k **Output:** Label of new sample

1: Calculate distances between new sample with each training sample.
2: Order training samples according to distances in ascending order.
3: Choose first k training samples.
4: Choose the label of majority of chosen samples.
5: **return** Label chosen.

2.3 Support Vector Machine

Support vector machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. In this multiclass classification problem, similar to Ridge regression, we need to build 10

binary SVM classifiers. The difference is the loss function is hinge loss instead of squared loss. Take one classifier as an example, the model is shown as follows.

$$\hat{w} = \arg\min_{w} \sum_{i=1}^{N} (1 - y_i \cdot x_i^T \cdot w)_+ + \lambda \cdot \|w\|^2$$

Where N is the number of samples. And we can use stochastic gradient descent to solve this model. The process of determining label of new sample is same as that of ridge regression.

Our implementation of SVM is based on LibSVM(LIBSVM is an integrated software for support vector classification, regression and distribution estimation. It supports multi-class classification. It has been developed and maintained by Chih-Chung Chang and Chih-Jen Lin)[1]. And we used cross validation to choose the best parameter.

SVM & Principle Component Analysis Principal component analysis (PCA) is a very powerful technique to reduce the dimension. It will extract the features with high variance and compress the data by ignoring components which are not meaningful. To show the effects of PCA, the first column in the Figure 5 is the original image. The images in the second column are obtained by just keeping one features with the largest variance. From the second column to the very right column, the retained number of component is increasing by 5, such that the images in the last column are reconstructed by keeping 46 features. By using PCA, the dimension is reduced dramatically without losing too much information. Variance captured when we choose different number of PCA dimensions is shown in Figure 6. It is exciting to note that 99% of variance has been captured using approximately half the number of dimensions we initially had in the data set!

Figure 5: Effects of PCA

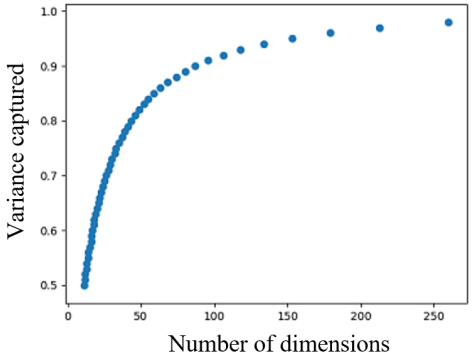

Figure 6: Variance captured

we'll be performing SVM analysis on a dimensionally compressed dataset using PCA, without compromising on the accuracy of the SVM model.

2.4 Neural Network

We used neural network on this classification task. The neural network we built consists of 784 nodes in input layer, 500 nodes in first hidden layer, 300 nodes in second layer, and 10 nodes in output layer, which is shown in Figure 7.

The input is $x_i, i = 1...784$ and x_0 is constant bias. The output of first hidden layer is $h_j^1, j = 1...500$ and h_0^1 is constant bias. The output of second hidden layer is $h_m^2, m = 1...300$ and h_0^2 is constant bias. The output of output layer is $\hat{y}_n, n = 1...10$. The weights between input layer and first hidden layer are $w_{i,j}$. The weights between first hidden layer and second hidden layer are $v_{j,m}$. The weights between second hidden layer and output layer are $u_{m,n}$.

Data preprocessing We normalized data dividing sample data by 255 to avoid error caused by precision problem of computer. Then, we need to implement one hot encoding for labels of our data. One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. Because we have 10 classes in our labels, if a label is 1, then it's one hot encoding is $[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$. [5] showed that elastic distortion can improve the performance of neural network. So we used elastic transformation to preprocess our data. The results are shown in Figure 8.

Activation functions We used Relu (Rectifier Linear Unit) as the activation function of the hidden layers. This function is similar to the functioning of the biological neuron, and allow us to run the neural network model more fast. $R(x)$ and its derivative are as follows.

$$Relu : R(x) = max(x, 0); \;\; Derivative : R^{'}(x) = \mathrm{I}(x)$$

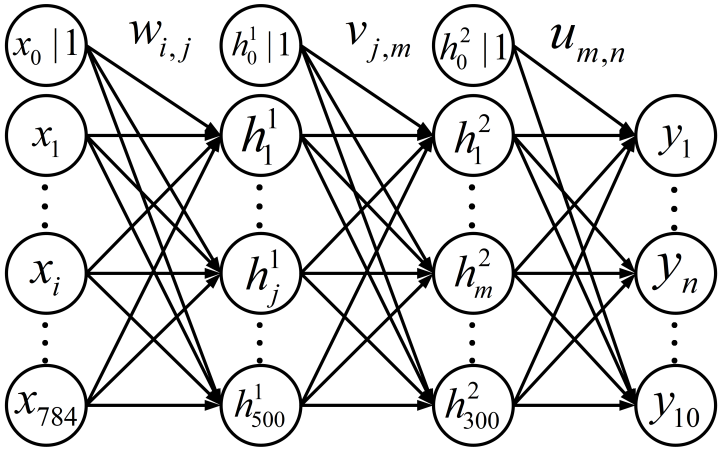

Figure 7: Neural Network

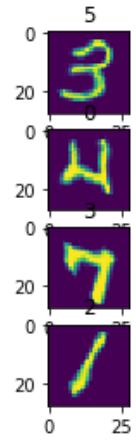

Figure 8: Elastic transformation

Where I is a indicator, it means if x is greater than 0, and the value of indicator is 1, otherwise, it's value is 0.

We used Softmax function as the activation function of the output layer. In probability theory, the output of the Softmax function can be used to represent a categorical distribution. So it's used in various multiclass classification. $S(\mathbf{x})_i$ and its derivative of are as follows.

$$Softmax : S(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{10} e^{x_j}}, i = 1...10; \;\; Derivative : S^{'}(\mathbf{x})_i = S(\mathbf{x})_i \cdot (1 - S(\mathbf{x})_i)$$

Dropout Dropout is technique for addressing overfitting problem in neural network. The key idea is to randomly drop units (along with their connections) from the neural network during training[6,7]. We implemented dropout by randomly setting outputs of some units of hidden layers to 0 when training and testing the model.

Metrics We chose Cross entropy loss function as our loss function.

$$Cross\ entropy\ loss : L(y|\hat{y}) = -\frac{1}{N} \cdot \sum_{i=1}^{N} [y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})]$$

Where N is the number of samples, and y is the correct label, and $\hat{y}$ is the predicted label. And the derivative of $S(\mathbf{x})_i$ is as follows. And we use accuracy rate to evaluate the performance of our neural network.

$$Derivative : L^{'}(y|\hat{y}) = -\frac{1}{N} \cdot \frac{y - \hat{y}}{\hat{y} \cdot (1 - \hat{y})}$$

Forward propagation When we move forward through the network from inputs to outputs, we have these calculations.

$$h_j^1 = R(\sum_{i=0}^{784} x_i \cdot w_{i,j}), j = 1...500;$$

$$h_m^2 = R(\sum_{j=0}^{500} h_j^1 \cdot v_{j,m}), m = 1...300; \;\; \hat{y}_n = S([\sum_{m=0}^{300} h_m^2 \cdot u_{m,n}]_{n=1...10})_n, n = 1...10$$

Stochastic gradient descent We used Stochastic gradient descent to train our neural network. Take one sample for example. Firstly, we calculate the gradient of loss function to $u_{m,n}$. Because the weight is only relevant with y_n,so we only care about loss caused by $\hat{y}_n$.

$$\begin{aligned} \nabla_{u_{m,n}} L(y_n|\hat{y}_n) &= \frac{dL}{d\hat{y}_n} \cdot \frac{d(\hat{y}_n)}{dI(\hat{y}_n)} \cdot \frac{dI(\hat{y}_n)}{u_{m,n}} \\ &= \frac{y_n - \hat{y}_n}{\hat{y}_n \cdot (1 - \hat{y}_n)} \cdot \hat{y}_n \cdot (1 - \hat{y}_n) \cdot h_m^2 \\ &= (y_n - \hat{y}_n) \cdot h_m^2 \end{aligned}$$

And then we calculate the gradient of loss function to $v_{j,m}$.

$$\begin{aligned}\nabla_{v_{j,m}} L(y|\hat{y}) &= \sum_{n=1}^{10} \frac{dL}{d\hat{y}_n} \cdot \frac{d(\hat{y}_n)}{dI(\hat{y}_n)} \cdot \frac{dI(\hat{y}_n)}{dh_m^2} \cdot \frac{dh_m^2}{dI(h_m^2)} \cdot \frac{dI(h_m^2)}{dv_{j,m}} \\ &= \sum_{n=1}^{10} \frac{y_n - \hat{y}_n}{\hat{y}_n \cdot (1-\hat{y}_n)} \cdot \hat{y}_n \cdot (1-\hat{y}_n) \cdot u_{m,n} \cdot \mathrm{I}(I(h_m^2)) \cdot h_j^1 \\ &= \sum_{n=1}^{10} (y_n - \hat{y}_n) \cdot u_{m,n} \cdot \mathrm{I}(I(h_m^2)) \cdot h_j^1\end{aligned}$$

And then we calculate the gradient of loss function to $w_{i,j}$.

$$\begin{aligned}\nabla_{w_{i,j}} L(y|\hat{y}) &= \sum_{m=1}^{300} \sum_{n=1}^{10} \frac{dL}{d\hat{y}_n} \cdot \frac{d(\hat{y}_n)}{dI(\hat{y}_n)} \cdot \frac{dI(\hat{y}_n)}{dh_m^2} \cdot \frac{dh_m^2}{dI(h_m^2)} \cdot \frac{dI(h_m^2)}{dh_j^1} \cdot \frac{dh_j^1}{dI(h_j^1)} \cdot \frac{dI(h_j^1)}{w_{i,j}} \\ &= \sum_{m=1}^{300} \sum_{n=1}^{10} \frac{y_n - \hat{y}_n}{\hat{y}_n \cdot (1-\hat{y}_n)} \cdot \hat{y}_n \cdot (1-\hat{y}_n) \cdot u_{m,n} \cdot \mathrm{I}(I(h_m^2)) \cdot v_{j,m} \cdot \mathrm{I}(I(h_j^1)) \cdot x_i \\ &= \sum_{m=1}^{300} \sum_{n=1}^{10} (y_n - \hat{y}_n) \cdot u_{m,n} \cdot \mathrm{I}(I(h_m^2)) \cdot v_{j,m} \cdot \mathrm{I}(I(h_j^1)) \cdot x_i\end{aligned}$$

The SGD in training have a lot of oscillations, so the momentum term was invented. The momentum term is used for soft the oscillations and accelerates of the convergence. Take weights between input layer and the first hidden layer as an example. $M_{i,j}$ is the momentum term, so the gradient descent process is like this.

$$\begin{aligned}M_{i,j}(k+1) &= \gamma \cdot M_{i,j}(k) + \eta \cdot \nabla_{w_{i,j}(k)} L(y|\hat{y}) \\ w_{i,j}(k+1) &= w_{i,j}(k) - M_{i,j}(k+1)\end{aligned}$$

2.5 Random forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes[4]. For every tree in the forest, we used bootstrapping to choose samples randomly to do the training, and choose f features from all features randomly to do the splitting. And when we get new samples, we used every tree to decide distribution of classes of the new sample, and make the final decision based on decisions of all trees. Let f be 2 (the number of features in experiment won't be so small), and an example of random forest is shown as Figure 9.

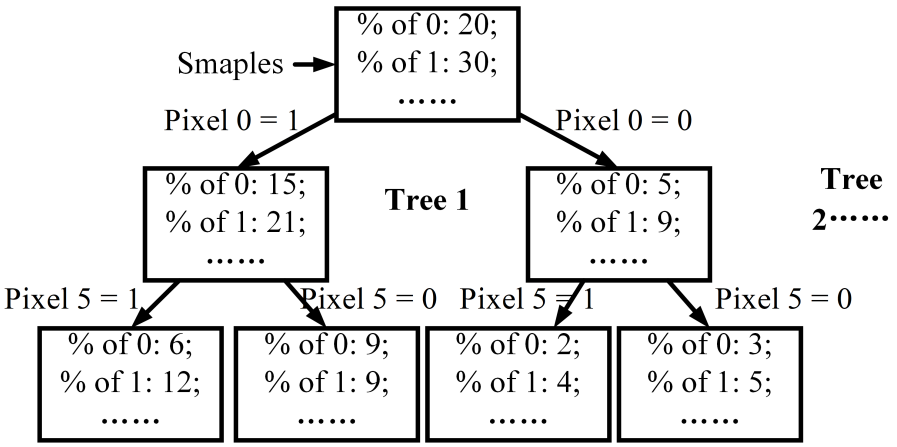

Figure 9: Random forest

3 Experiment results and Conclusion

We implemented every model we talked about before and solve the classification problem of MNIST data set. Please find our code here `https://github.com/PencilZQ/CS532-final-project.git`. The results of all the discussed models have been summarized in Table 1 and some very interesting conclusions can be draw.

i) The Neural Network model has the highest accuracy rate (Accuracy: NN>k-NN>SVM>Random forest>Ridge regression). This result is because: (1) Loss function of Ridge regression is squared loss, which may have high loss even when the classification is correct, so it's not good for classification. Compared to that, loss function of SVM is much more suitable for classification and hence achieved higher accuracy(2) The other models are non-linear classifiers, which are good for multi-class classification problem like this one, and hence achieved higher accuracy (except Random forest).

ii) The training time of Ridge regression is attractive, but its low accuracy makes it not useful in classification problem. SVM, k-NN and NN can achieve top 3 highest accuracy, but the their training costs are too high, especially k-NN. Compared to the models mentioned before, although Random forest has a little lower accuracy, its big advantage is its training cost. So if our first objective is the accuracy and we don't want to spend a lot of time training, NN is our first choice, and if we don't need our model to be the most accurate and want to train the model as fast as possible, then Random forest is our first choice.

iii) PCA is good way to improve efficiency and reduce training cost. When implemented with PCA, efficiency of SVM is improved without deteriorating performance too much. When we used 100 dimensions of PCA, the efficiency of model training was improved by 6 times, and the accuracy only got 0.3% lower.

Table 1: Experiment results

1.a Comparison of models

Model	Accuracy (%)	Time(s)
Ridge Regression	84.17	7.93
SVM	95.21	306.44
k-NN clustering	96.45	674.65
Neural Network	98.65	287.54
Random Forest	94.07	2.74

1.b SVM with PCA

Dimension	Accuracy(%)	Time(s)
50	88.23	42.87
100	94.89	57.44
200	95.15	131.46
250	95.17	181.22
Without PCA	95.21	306.44

Learning outcomes and connection to CS 532 Some of the methods we have implemented (NN, SVM, Ridge Regression) were covered in the class. We were able to successfully apply them to a huge data set with multiclasses, and obtain tangible results and knowledge. And we used some training techniques like cross validation covered in class. Lastly, we were able to combine the concepts learned, like PCA for dimensionality reduction followed by SVM, and greatly improved efficiency of model training.

In NN part, we succeeded in increasing the number of hidden layers to 2 instead of 1, and we used some more advanced training technique like dropout, momentum term and elastic transformation based on some papers and textbooks we read beyond the scope of this class.

Other methods(k-NN, Random forest) were beyond the scope of the class, but because these models are also supervised learning, and building upon the concepts we learned throughout the course; we were able to develop a strong mathematical intuition and successfully implement these models on a large data set.

References

[1] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training SVM[J]. Journal of Machine Learning Research, 2005, 6:1889-1918.
[2] Matrix Methods in Data Mining and Pattern Recognition by Lars Elden.
[3] Pattern Recognition and Machine Learning by Christopher M. Bishop.
[4] Bernard S, Adam S, Heutte L. Using Random Forests for Handwritten Digit Recognition[C]. International Conference on Document Analysis and Recognition. IEEE, 2007:1043-1047.
[5] Simard P Y, Steinkraus D, Platt J C. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis[C]. International Conference on Document Analysis and Recognition, 2003. Proceedings. IEEE, 2003:958.
[6] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. Journal of Machine Learning Research, 2014, 15(1):1929-1958.
[7] Wan L, Zeiler M, Zhang S, et al. Regularization of neural networks using dropconnect[C]. International Conference on Machine Learning. 2013:1058-1066.
[8] Zhu X, Suk H I, Lee S W. Subspace Regularized Sparse Multi-Task Learning for Multi-Class Neurodegenerative Disease Identification[J]. IEEE transactions on bio-medical engineering, 2016, 63(3):607.
[9] Zhang H R, Min F. Three-way recommender systems based on random forests[J]. Knowledge-Based Systems, 2016, 91(C):275-286.
[10] Biau G. Analysis of a Random Forests Model[J]. Journal of Machine Learning Research, 2010, 13(2):1063-1095.