Classify Handwritten Digits Using the Famous MNIST Data

Lingzhe Teng and Hochul Cho

1. Problem definition

The goal of this project is to take an image of a handwritten single digit, and determine what that digit is. From the provided dataset, each row in training dataset means the label and 28x28 pixels data of an image. The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems, and through the home page of this dataset, we found that it provides many kinds of solution implemented by different machine learning algorithms without random forest. Random Forest [1] uses multiple classifiers for bagging, boosting and random subspaces, and there is a growing interest to use this kind of multiple classifier system in pattern recognition field. Based on previously works, we want to implement random forest by coding so as to have a deeper understand about how a machine learning algorithm works in real life problem.

Additionally, we use third-part toolbox, like WEKA, to train and evaluate the same dataset with random forest and other machine learning algorithms, so that we can compare the performances between our implement and existing solutions. The evaluation metric for this project is the categorization accuracy, or the proportion of test images that are correctly classified. Besides, some performance results will be presented to evaluate this project.

2. Background

2.1 Handwritten Digits Recognition Problem

Handwriting Recognition is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents and other devices. Narrowing the problem domain often helps increase the accuracy of handwriting recognition systems. The dataset from MNIST would contain only the characters 0-9. This fact would reduce the number of possible identifications, by which it's more suitable for machine learning beginners to conduct some researches. The main process [2] of handwritten digits recognition includes dataset preprocessing, feature extraction, feature selection and test by classifier, and the accuracy rate for this problem varies by using different machine learning algorithm.

2.2 Algorithms

A lot of machine learning [3] is developed for solving data science problems. However, there is no perfect algorithm for all problems because they have different characteristic. The type of algorithms which are frequently used can be classified as Tree, Neural Network, Bayes Theory, and Ensemble. The representative algorithms of each type and characteristics are as follows.

- Decision Tree (Tree): decision tree is one of the classification methods, which uses a tree-like graph. The model can be expressed visually, so it is easy to understand the model. This has high performance for numerical data.
- Multi-Layer Perceptron (Neural Network): This is one of the Neural Network algorithms. It takes lots of training time, but it has good performance in complex problems.

- Naïve Bayes (Probability): This is probabilistic classification algorithm based on Bayes Theory. Because it includes strong independence assumptions between the features, it takes less training time than other Bayes algorithms.
- Random Forest (Ensemble): Random Forest uses ensemble approach, which makes many models by training data and combines the results of the models to improve the accuracy rate. Decision Tree is used for basic model.

3. Dataset Description

The MNIST (Modified National Institute of Standards and Technology) database [4] gives a lot of images of handwritten digits. It is constructed based on NIST (National Institute of Standards and Technology) which gives data set of over 800,000 images of handwritten digits from 3,600 writers. The MNIST data have been size-normalized and centered in a fixed size image, so we can use this data and apply our learning algorithms to real-world problem without additional effort on pre-process procedure. We use the dataset which is organized for Keggle competition [5].

The data contains gray-scale images of hand-written digits (0~9). The each image consists of total 784 pixels (28 * 28) and pixel-value (0~255) which determines the degree of darkness or lightness. The image data is pre-processed and combined to one CSV type (Figure 2). The column of CSV data means pixels of an image data and the row means each image data. Training data set contains label, but test data set doesn't contain the label.

000	001	002	003	...	026	027
028	029	030	031	...	054	055
056	057	058	059	...	082	083
|	|	|	|	...	|	|
728	729	730	731	...	754	755
756	757	758	759	...	782	783

Figure 1. One Image Data

Data 1	001	002	…	783
Data 2	001	002	…	783
Data 3	001	002	…	783
…	001	002	…	783
Data 60000	001	002	…	783

Figure 2. CSV Data

Features of the dataset are as below.

- Real hand-printed data prepared by individuals (approximately 500 writers)
- Amount of data: 40,000 examples (71.4MB)
- Amount of features: 784 (pixel)
- CSV data type

4. Methods

4.1 Implement Random Forest by Code

In this project, we tried to implement our own Random Forest without any machine learning toolbox provided by MATLAB, which is a famous a high-level language and interactive environment for numerical computation, visualization, and programming. In this part, we will discuss how to implement a Random Forest by MATLAB_R2015a to predict label values in MNIST dataset.

4.1.1 Making a Single Tree

Random Forest includes many individually developed trees, and the Random Forest algorithm [6] developed by Dr. Leo Breiman in 2001 choose to CART algorithm to train each tree and the predictions of all trees are subjected to a voting procedure which aggregates the results. The voting determines the prediction of the final class of the algorithm.

CART algorithm based on the classification and regression methodology, and it uses an information measure technique, Gini index, to determine the best split at each level. The pseudo code for GenerateTree is shown as follow, and after recursively call GenerateTree method, we can finally generate a decision tree.

```
treeNode
: a struct variable for each node, and it stores data like dataset will be processed in current node, chosen feature to split and the split position, etc.

treeNodeNo
: a global variable to record the number of current node

treeNodes
: a global variable to record all tree nodes for a tree

MaxGiniGain(samples)
: return max gini gain, split feature index and split position

GenerateTree(treeNode)
: train tree node based on data from treeNode variable

Load data from treeNode variable;
[maxGain splitFeatureIdx splitPosition] = MaxGiniGain(samples)
if ( maxGain < threshold || num of sample less than 0){
    it’s a leaf node, create tree node as leaf node and return;
}
get split feature from splitFeatureIdx variable;
create current tree node;

get sample index for left and right branches by compare the value of split position;
get samples for left and right branches;
create tree node variable for left and right branches;
GenerateTree(leftTreeNode); GenerateTree(rightTreeNode);
```

Figure 3. Pseudo code for the generate a single tree

4.1.2 Making a Random Forest

To make a Random Forest, three parameters have to be set at appropriate values in consideration of time and system resources. The three parameters are the number of trees, the random number seed and the number of selected features. The value of random seed is selected to control the constructor of each tree, because the random value used in our

implement is to select the samples and features for training. In Random Forest, a larger number of trees mean a better classification rate. However, since large data sets are used in this project, the number of trees developed is limited to the available system resources. The number of features which is the subset of variables selected from all variables in the dataset, and the author of Random Forest, Breiman, suggest in his paper that $\log_2[M+1]$ for this value will provide the best overall performance. The pseudo code for make a Random Forest is shown as
follow.

```
randomSeed
: control the constructor of each tree

numOfSelFtr
: determine the number of features will be selected to train a tree

treeNumber
: larger number may be get be better accuracy rate

trees
: storage the trees created by GenearteTree method

RandomSubset(dataset, random seed)
: bagging dataset to get bootstrap dataset for training and out-of-bag dataset for evaluating

GenerateTree(treeNode)
: train tree node based on data from treeNode variable

for ti=1 to ti=treeNumber{
    in-of-bag and out-of-bag dataset = RandomSubset(dataset, randomSeed);
    m = get the number of features for training based on the value of numOfSelFtr;
    subdataset = randomly select feature columns from dataset;
    treeNode = create a new tree node with calculated data;
    GenerateTree(treeNode);
    forest = trees(ti);
}
```

Figure 4. Pseudo code for generating a Random Forest

4.1.3 Preprocessing Dataset

The dataset from MNIST without any feature of the image, and the dimension for this dataset is very large. Therefore, preprocessing dataset and extract image feature from the raw is becoming an acceptable way to improve the performance of our project. To do this, we need binarize, crop, find and merge the profile for the raw data [2] and then conduct some feature extraction jobs like circle recognition, row subsection, and side contour. After doing this, we may get more reliable feature vectors for our project. The details about the work of preprocessing are shown in the source code.

4.1.4 The Way tried to Improve Performance

Based on the work of Breiman, researchers conduct a lot of works and make some processes. Through these paper, we tried to improve the performance of our implement by adding weights [8] [9] . The strategies to do that are shown as follow.

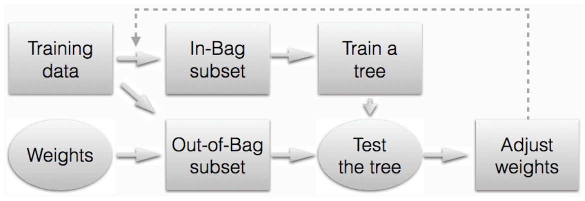


Figure 5. the process to the generate a random forest

Through the paper, the author mainly provides two ways to improve the performance. The first way is to calculate the accuracy of each tree and sort the accuracy of trees, then just select 80% trees with higher accuracy to generate to forest. The other ways are that recalculate weight of each training data, and use the weighted data to form the next tree.

4.2 Applying Machine Learning Algorithms by Weka

Weka [10] is a famous machine learning tool written in Java. This is open source software under the GPL. It helps to solve data science problem like classification, cluster and associate by giving a lot of useful machine learning algorithms library.

As we explained in chapter 2.2, representative algorithms in the data science field are like Decision Tree, MLP, Naïve Bayes and Random Forest. Each algorithm has distinct characteristics because they are based on different theory as Tree, Neural Network, Bayes Theory (Probability) and Ensemble. In this paper, we show the difference of these algorithms in the handwritten digits recognition problem. For comparing the performance of these algorithms, we use the algorithms implemented in Weka. The basic setting of parameters of the algorithms is as follows.

- Decision Tree (J48) -> Confidence Factor: 0.25, Minimum number of objects: 2 and Number of Folds: 3
- MLP -> Hidden Layer: 1, Learning rate: 0.3, Momentum: 0.2, Training time (The number of epochs): 100 and Validation Threshold: 20
- Naïve Bayes-> None
- Random Forest-> Maximum depth: 10, Number of Random Features: 10 and Number of Trees: 100

5. Experiment: performance analysis and results

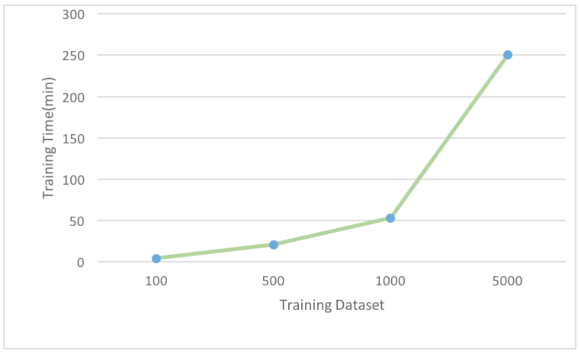
5.1 Experiment Environment for Analysis of our Random Forest

The performance of random forest can be influenced by key parameters we set previously, and thus we can easily analyze the difference of the results by control these parameters. We use a machine with Intel® CPU Core i5 @ 2.70GHz running on Mac OS X 10.10.2 Edition. It has 8 GB DDR3 RAM, and the MATLAB version is R2015a. In this experiment, we use the version without preprocessing, because currently the version with preprocessing dataset doesn't bring a lot improvement for our project.

In this project, we use 10-fold cross validation to train and evaluate our random forest. Meanwhile, during the process to build a single tree, we use out-of-bag dataset as testing dataset to evaluate the accuracy or error rate after the training process of each tree. Using 10-fold cross validation, the random forest could be trained 10 times and get the accuracy or error rate each time, by which we can use the average value as the result of accuracy or error rate.

5.1.1 Controlling Data Size

For controlling the size of training dataset, we keep the same number of trees for each random forest and use $\log_2[M+1]$ to randomly select features to train, and thus we can try to control the size of training dataset to test running time and accuracy. In this case, we don't use whole dataset to evaluate and add weights tried to improve the accuracy, because with the bigger dataset we used, the running time was increased obviously, and we need to reduce the time cost by following works.



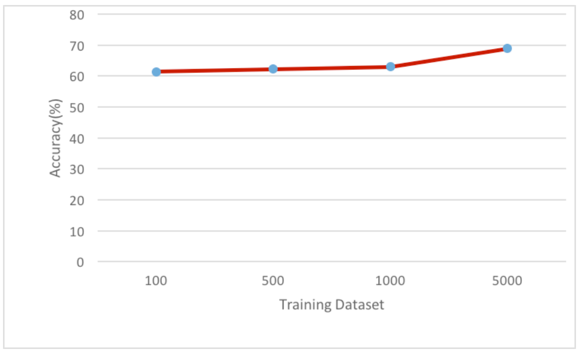


Chart 1. running time for random forest with different size of training Dataset

Chart 2. accuracy for random forest with different size of training Dataset

Through these two charts, we could know that the running time for our random forest will increase with the bigger size of training dataset. However, the more dataset we used to train cannot improve the accuracy rate obviously when training dataset.

5.1.2 Controlling Parameters

Based on previous experiment, we try to control parameter like the number of selected features or the number of trees to evaluate the accuracy performance of our random forest. To control the number of trees, for example, we keep the same number of selected features, which is 10, and monitor the changes when using different sizes of training dataset.

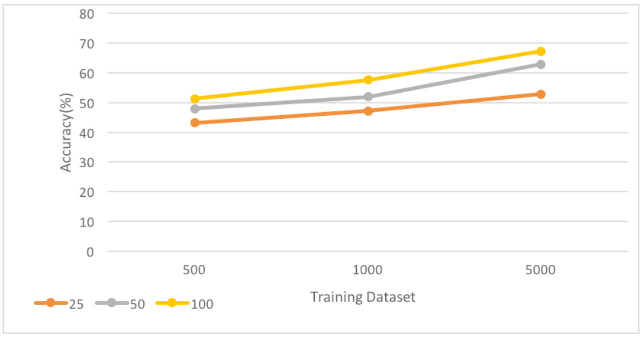


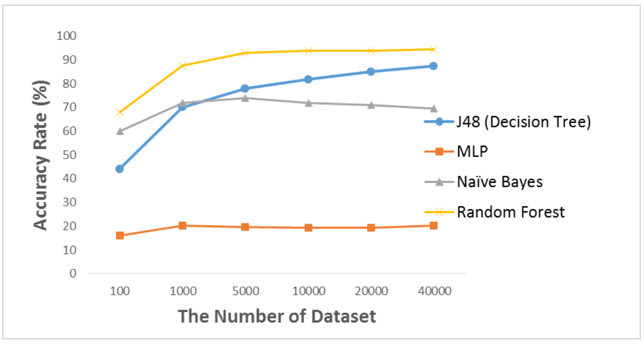
Chart 3. Accuracy for random forest with different number of trees and data size

From Chart 3, we can find when the size of training dataset is not big enough, like 500 to 1,000, the accuracy for different number of trees doesn't changed a lot. Meanwhile, from the data for 5,000 training dataset, we could assume that the more trees used to train can bring faster increase for accuracy, and through the whole charts, we can see that tree number is a factor can influence the accuracy.

5.2 Experiment Environment for Analysis of the Machine Learning Algorithms

To determine an algorithm's performance, it is important to have a common measure for all the data mining models. The most common measure is to calculate the accuracy rate for each algorithm and compare them for finding the best model of this problem. In this experiment, the accuracy rate is calculated by 10 cross-validation. We use a machine with Intel® Xeon® CPU E5-2690 @ 2.90GHz running on Windows 7 Professional Edition. It has 32 GB DDR3 RAM.

5.2.1 Accuracy Rates and Training Time by the Data Size



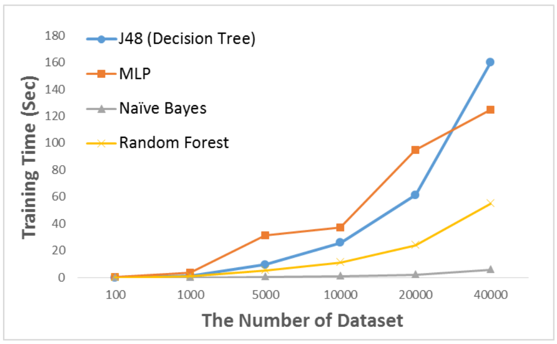


Chart 4. Accuracy Rate of the Algorithms **Chart 5. Training Time of the Algorithms**

In the Chart 4, the accuracy rate of MLP (Multi-Layer Perceptron) is very low. This is shown in the all number of the dataset, so it means MLP is not suitable for this problem or parameter setting is incorrect. We set very low number of hidden layers and the number of epochs in order to reduce training time. It can bring about low performance of this MLP. On the other hand, Random Forest shows very high performance. A notable point is that Random Forest has the accuracy rate over 90% even though the number of dataset is only 1000.

Chart 5 shows the change of training time as the number of dataset increase. It is very interesting that Naïve Bayes has few training time compared to other algorithms and it takes only 6 seconds even when using 40,000 dataset. The reason of this small amount of time is Naïve Bayes strong independence assumptions between the features. As MLP is famous for taking a lot of training time, it takes most training time except when using 40,000 dataset.

6. Observation and Conclusion

Through the process to improve the performance of our random forest, we read a lot of papers, and tried to find some useful solutions to improve the accuracy. However, the ways we tried doesn't bring markedly improvement, and it makes the running time of our implement much longer. The best value of accuracy we got during experiments was approaching 73%. By observing parameters and searching solutions from existing solutions, we think over some reasons for that. The first reason is that the method we chose to generate random forest is not suitable to this dataset, and the second reason we think it's the features we selected to build a tree. In current implement, we didn't add any solution to optimize the logic of selecting features, instead, we just determine the number of feature needed to select and randomly select them from features pool. Other reasons may be the algorithm we chose to generate tree nodes or the way to get in-of-bag dataset for a tree. Therefore, the following works for our implement of random forest is to reduce the running time and increase the accuracy without any performance lost.

In the second experiment, we show the difference of machine learning algorithms. It shows, although there are many machine learning algorithms, data scientist should choose proper algorithm and parameter for the problem. For classification of handwritten digits, we uses representative machine learning algorithms which stem from other concepts like traditional algorithm (tree), probability theory and Neural Network, mixing algorithms. Our experiment shows very interesting results about accuracy rate and training time by algorithms. The overall of Chart 4 and Chart 5 is that MLP is not suitable for this problem because accuracy rate is very low with taking a lot of training time. Random Forest fits this problem because it shows very high accuracy rate and normal training time.

7. Reference

[1] Bernard, S, "Using Random Forests for Handwritten Digit Recognition", ICDAR , 2007

[2] Pandey, Akhilesh. "A Study of Structural Feature Extraction of Handwritten Numerals". *International journal of electronics and computer science engineering* (2277-1956)

[3] Bishop, Christopher M. Pattern recognition and machine learning. Vol. 4. No. 4. New York: springer, 2006.

[4] http://yann.lecun.com/exdb/mnist/

[5] https://www.kaggle.com/c/digit-recognizer

[6] Leo Breiman, "Random forests," *Machine Learning*, 2001.

[7] Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard and V. Vapnik: Comparison of learning algorithms for handwritten digit recognition, in Fogelman, F. and Gallinari, P. (Eds), *International Conference on Artificial Neural Networks*, 53-60, EC2 & Cie, Paris, 1995.

[8] Florian Baumann, "Sequential Boosting for Learning a Random Forest Classifier", *Winter Conference on Applications of Computer Vision*, 2015.

[9] Baoxun Xu and Yunming Ye , An Improved Random Forest Classifier for Image Classification, *IEEE International Conference on Information and Automation*, 2012

[10] M. Hall *et al*. "The WEKA data mining software: an update." *ACM SIGKDD explorations newsletter,* 2009.

[11] Pal, Sankar K., and Sushmita Mitra. "Multilayer perceptron, fuzzy sets, and classification." *IEEE Transactions on Neural Networks* 3.5 (1992): 683-697.

[12] Yi Yao and Gianfranco Doretto, "Boosting for transfer learning with multiple sources," in *CVPR*, 2010, pp. 1855 – 1862.

[13] J. Ross Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.