Comparison of Handwriting Recognition Algorithms AdaBoost, Support Vector Machines and Neural Network

By Shiyu Wang, Yi Xing

Introduction, Background, Motivation

Introduction

The human visual system is one of the mysteries in the world. Normally, most people effortlessly recognize any digits from 0 to 9. However, that ease is deceptive. In each hemisphere of our brain, humans have a primary visual cortex, also known as V1, containing 140 million neurons, with tens of billions of connections between them. (Kégl & Busa-Fekete, 2009) If we Consider human brains as supercomputers, the software that human used to recognize handwritten digits are amazing and incredible. Recognizing handwritten digits is not easy, but humans can do it within a few milliseconds. And most of the times, human will not make mistakes on recognizing them, and all that work is done unconsciously. Hence, we do not appreciate how tough this problem is for computers.

Comparing with human, writing a computer program to recognize handwriting is extremely difficult. Understanding a shape of digit can be easily confused for computers. For instance, a simple description for number eight can be: "An 8 has two circles, one on top and another one on the bottom." This depiction seems to be clear enough, but troubles can be various in the real world. For example, the circles of 8 will not always be round or same size in handwriting. Moreover, the two circles may not stay vertical. Also, it is easy to mix up the 8 and 9. Therefore, scientists in the machine learning field come out a method that other than only describing the digits, we provide abundant sample of handwritten digits to train the computer firstly. Then the computer can form a model based on the training data, and recognize digits with the model. Within this logic, many algorithms are used for handwritten digit recognition. The main purpose of this project is to show and compare three representative algorithms in handwritten digit recognition, AdaBoost, Support Vector Machines and Neural Network.

Background

Handwriting recognition is not only the first lesson of machine learning, but also an important challenge in the tech industry, which has a variety of uses. The handwriting has entered everyone's life. One of the first uses is from USPS zip code reading. A zip code consists of 5 digits, which is the most important parts used to find the correct location. Also, comparing with typing on the keypad, writing with fingers on mobile devices is another instance which is much more convenient. After that, there are other uses such as deciphering forms and evaluating exams are related to handwriting recognition. Some of these tasks are even impossible or extremely inefficient for human.

Motivation

In pattern recognition, handwriting recognition is a challenging task through many years. Efficiently and accurately reading handwritten digits is tough for computers. Algorithms of machine learning like neural networks, SVM or AdaBoost have been applied to solve this problem. Hence, which algorithm is more efficient and accurate in handwriting recognition interest us. Therefore, three representative algorithms are chosen to compare in this project.

Problem Statement

This handwriting recognition project deals with classifying digits data from the MNIST("Modified National Institute of Standards and Technology") datasets. The data contains 60,000 photos of 28x28 pixel handwritten digits. In the current century, many machine learning algorithms can be used to classify and recognize handwritten digits with high rate. In this project, we would like to implement and compare the efficiency of different algorithms in digit recognition, including Support Vector Machines, Adaboost and Neural Networks.

Methodology

In this project, we researched and implemented three algorithms: Adaboost, Support Vector Machines and Neural Network.

Adaboost (Adaptive boost)

Adaboost is a meta-algorithm that combine one or more types of weak learners with weights on them to be a classifier. It is a continuous step that each step update the weight of weak learner so it could force on those instances that misclassified by previous learners. The Adaboost classifier:

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

At each iteration t, the sum training error be updated as:

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

Support Vector Machines

SVM (Support Vector Machines) is a widely used technique in data classification. The goal of a support vector machine is to find the optimal separating hyperplane which maximizes the margin of the training data. Transferring in math term: Given a training set of instance-label pairs (xi, yi), i = 1, … , l where $x_i \in R^n$ and $y \in \{1, -1\}^1$, the support vector machines (SVM) require the solution of the following optimization problem (Hsu & Chang & Lin, 2003):

$$\min_{w, b, \varepsilon} \frac{1}{2} w^T w + C \sum_{i=1}^{l} \varepsilon_i$$

$$\text{Subject to } y_i(w^T \O(x_i) + b) \geq 1 - \varepsilon_i,$$

$$\varepsilon_i \geq 0.$$

Neural Network

Neural Network in machine learning and cognitive science refers to artificial neural network(ANN). This model can be defined as a class of functions $f: X \rightarrow Y$ which determined by vary parameters and connected weights. In this project, we chose one of the models - convolutional neural network(CNN). This model is a type of feed-forward neural network with the connective pattern of neurons is inspired by the animal visual cortex. One of the distinguishing feature of CNN is shared weight in convolutional layers, which decrease the memory and improve the performance.

Novelty

In this project, we use the same database for all three algorithms in order to further compare the training speed, test accuracy and implementation of them. Moreover, other than using package, we research and implement Adaboost and SVM by ourselves to understand and applying advanced theories and methods which will be introduced in "The AI algorithms/methods used" part.

Analysis and Design of Project

The train and test dataset of this project is MNIST, can be downloaded from MNIST website (http://yann.lecun.com/exdb/mnist/). This dataset is binary files that store the image pixels as matrices. It could be read by standard method provide by Gabe Johnson to be pixel matrices. Also, it could be handled to be a .mat file.

There are three packages in the project, Adaboost, SVM and neural network to implement three machine learning algorithm separately. The steps of all the three algorithms are the same. The first step is to use the train dataset and algorithm to train a model. Then use the train and test dataset and the model to generate predicted labels. Finally, compare the predict labels with original labels and calculate the accuracy.

The result is an array to store the predicted train and test dataset labels. To evaluate the algorithm performance, we compare the predicted labels with original labels. If related labels are not the same, it should be an error prediction. To statistic these errors, we can get the error rate:

Number of error predict labels / Number of dataset instances

Also, the accuracy is the percentage of correct labels. Furthermore, performance should consider overfit. Overfit means that an algorithm has very high accuracy when predict train dataset, while has very low accuracy in test dataset.

To compare the performance of machine learning algorithm, we need to consider the accuracy and the overfit. With higher accuracy and without overfit is a better algorithm. The efficiency is another important factor. The shorter running time with higher accuracy is better. In this project, we will use above factors to compare our algorithms.

The AI algorithms/methods used

Adaboost - Yi Xing

Adaboost (Freund & Schapire, 1997) is a classifier with other weak learner and updating the weight of them to be a strong learner. One of Adaboost application is to use in handwriting digit recognition. HAAR image features combine with the stump (Balázs & Busa-Fekete, 2009) has been shown with high performance and accuracy. In this project, I choose HAAR image features as features and decision stump as a weak classifier in Adaboost. Tending to handle multi-class labels in the dataset, I use Error-Correcting Output Coding (ECOC). The architecture diagram shows in Figure 1.

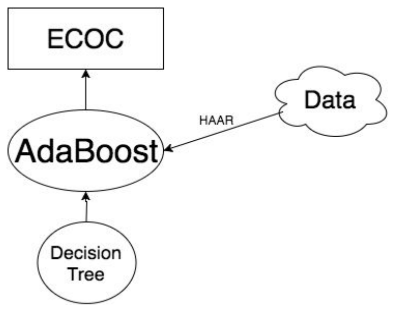
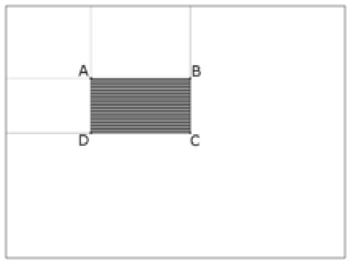


Figure 1. Adaboost Implementation

Haar-like features are digital image features used in object recognition. It considers adjacent rectangular regions and sums up the pixel intensities of these regions in a detection window, and then calculates the difference between these sums. These differences used as features and reduce the number of image features. In this project, I extracted 200 Haar features from MNIST dataset as the features of train and test dataset. The sum formulates and Figure 2 shown in below:



$$sum = I(C) + I(A) - I(B) - I(D)$$

Figure 2 The sum of shaded in rectangle area in HAAR

Decision Stump is a machine learning model which is a one-level decision tree. It is widely used in Adaboost as a weak learner. During training the Adaboost model, in each iterative step, a stump is built to classify train data with weighting error rate on the misclassified training sample. Therefore, later stumps building tends to focus on harder-to-classify examples. In this project, due to the limitation of computer, the steps of each Adaboost is 200.

Error-Correcting Output Coding(ECOC) is a recipe for solving multi-class problems. It decomposes the multi-class problem into binary classification. The first step is to create binary string to represent the class independently. The second stage is to using Euclidean distance to decide the result belongs to which class. In the project, due to the number of digit class is 10, I implemented a 50 length binary string to represent each class. For each ECOC function, I built the Adaboost that for each instance, the train label change to the relative label from ECOC function. After the ECOC loop, each instance label will be a binary string constructed by each Adaboost model result. Calculating the Euclidean distance between the ECOC code book label string and instance string to determine which class it belongs to.

Support Vector Machines - Shiyu Wang

The SVM algorithm was created by Vladimir Vapnik in 1963. In 1992, Bernhard Boser, Isabelle Guyon and Vladimir Vapnik improved it by applying the kernel trick to optimal hyperplanes for creating nonlinear classifiers. In order to implement SVM, we need to understand the goal of SVM is to find the optimal separating hyperplane which maximizes the margin of the training data, because larger margin can give the lower generalization error of the classifier.

The hardest part of SVM is the math behind it. Following the optimization function mentioned above, A kernel function is used to map training vectors into higher dimensional space. Kernel function is a similarity function. It is a function that provides to SVM, which takes two inputs and spits out how similar they are. According to “A Practical Guide to Support Vector Classification”, there are four basic kernels, including:

- Linear: $K(x_i, x_j) = x_i^T x_j$
- Polynomial: $K(x_i, x_j) = (\gamma \cdot x_i^T x_j + r)^d, \gamma > 0$
- Radial Basis Function (Gaussian): $K(x_i, x_j) = \exp(-\gamma \cdot ||x_i - x_j||^2, \gamma > 0$
- Sigmoid: $K(x_i, x_j) = \tanh(\gamma \cdot x_i^T x_j + r)$

Here, γ, r, and d are kernel parameters. (Hsu & Chang & Lin, 2003)

In this project, Gaussian kernel is the best choice. It maps the training data into higher dimensional space nonlinearly. Therefore, it can handle cases of the relation between class labels and attributes is nonlinear. Furthermore, linear kernel and sigmoid kernel can be considered as a special case of Gaussian kernel, because the linear kernel and the sigmoid kernel has the same performance as the Gaussian kernel with some parameters (C, γ) (Hsu & Chang & Lin, 2003) . Moreover, sigmoid kernel can’t be applied with some parameters in this condition. On the other hand, The polynomial kernel will make the model selection much more difficult because it has more parameters than the Gaussian kernel. Finally, the Gaussian kernel has fewer numerical difficulties. One key point is $0 < K_{ij} <= 1$ in contrast to polynomial kernels of which kernel values may go to infinity ($\gamma \cdot x_i^T x_j + r > 1$) or zero ($x_i^T x_j + r < 1$) while the degree is large (Hsu & Chang & Lin, 2003) .

After kernel function, interior point method is picked to make the model. It applies Newton-like iterations to make the Karush–Kuhn–Tucker (KKT) system into linear equation. By using this method, I do not need to solve a sequence of problems, but to directly solve all the problem at once. With interior point method, I come out with function to minimize sequence of logarithmic barrier:

$$\min \Phi\mu(w, \xi, t, b) \triangleq$$
$$\sum_{j=1}^{m} t_j + C \sum_{i=1}^{n} \xi_i - \mu \sum_{j=1}^{m} [log(t_j^2 - w_j) + log(t_j^2 + w_j) + log(t_j)] - \mu \sum_{i=1}^{n} (log\, p_i + log\, \xi_i)$$
$$\text{Subject to } t_j^2 - w_j > 0,\ t_j^2 + w_j > 0,\ t_j > 0,\ j = 1,2, \ldots, m,$$
$$P_i = y_i(w^T x_i + b) + \xi_i - 1 > 0,\ \xi_i > 0,\ i = 1, 2, \ldots, n.$$

* μ is the barrier parameter.

After that, the KKT system can be formed into linear equations:

$$\lambda^{(1)} - \lambda^{(2)} - X^T Y^T \lambda^{(4)} = 0,$$
$$C \cdot e_n - \lambda^{(4)} - \lambda^{(5)} = 0,$$
$$e_m - 2T\lambda^{(1)} - 2T\lambda^{(2)} - \lambda^{(3)} = 0,$$
$$y^T \lambda^{(4)} = 0,$$
$$(T^2 - W)\lambda^{(1)} - \mu e_m = 0,$$
$$(T^2 + W)\lambda^{(2)} - \mu e_m = 0,$$
$$T\lambda^{(3)} - \mu e_n = 0,$$
$$P\lambda^{(4)} - \mu e_n = 0,$$
$$\Xi\lambda^{(5)} - \mu e_n = 0,$$

* λ’s are Lagrangian multipliers

T, W, Ξ and P are diag(t), diag(w) and diag(ξ); P = diag(Y(Xw + b ･ e_n) + ξ - e_n) are formed diagonal matrices. Beside it, $e_m \in R^m$ and $e_n \in R^n$ are the unit vectors. Then we just need to apply newton's method to solve the equations:

$$M = \begin{pmatrix} \Delta w \\ \Delta \xi \\ \Delta t \\ \Delta b \\ \Delta \lambda^{(1)} \\ \Delta \lambda^{(2)} \\ \Delta \lambda^{(3)} \\ \Delta \lambda^{(4)} \\ \Delta \lambda^{(5)} \end{pmatrix} = - \begin{pmatrix} \lambda^{(1)} - \lambda^{(2)} - X^T Y^T \lambda^{(4)} \\ C^* e_n - \lambda^{(4)} - \lambda^{(5)} \\ e_m - 2T\lambda^{(1)} - 2T\lambda^{(2)} - \lambda^{(3)} \\ e_n^T Y \lambda^{(4)} \\ \left(T^2 - W\right) \lambda^{(1)} - \mu e_m \\ \left(T^2 + W\right) \lambda^{(2)} - \mu e_m \\ T\lambda^{(3)} - \mu e_m \\ P\lambda^{(4)} - \mu e_n \\ \Xi \lambda^{(5)} - \mu e_n \end{pmatrix}$$

The result of it brings the matrix to train the model with data. The more data training, the longer it takes and more accurate in test.

Neural network - Yi Xing

In this project, we implemented a convolutional neural network (CNN) by TensorFlow. The CNN is a very efficient neural network for classifying images. It reduces the numbers of neurons in the hidden layers and helps us to train more deep and more layers network with shorter training time. The three important features of CNN are local receptive fields, shared weights and pooling.

For local receptive fields, in hidden layer, we connect the neuron a localized regions of the input image. In the project, each neuron in the first hidden layer will be connected to a 5×5 region, which is 25 input pixels. We then create the local receptive fields by slide 1 pixel from left-top to right-bottom crossing the entire input image. Therefore, for this 28×28 pixels image, it turns to be a 24×24 neurons hidden layer.

For shared weights and bias, all neurons in the first hidden layers share the same weights and bias, so they will detect the same features. To do handwriting digit recognition, we need more than one feature. In the project, the code creates 32 features map.

For pooling layers, they usually immediately after convolutional layers. The pooling layers take the output from convolutional layer to be a condensed feature map. The pooling unit in this project is maximum 2×2 input region. After pooling, the layer becomes a 12×12 neurons. Also, for each feature map, there is a pooling layer.

In the project, we trained a 2 layers convolutional neural network. The final layer, after the 2 convolutional layer-pooling layer, is a fully connected layer. This layer connects neurons from the lasted max pooled layer to 10 neurons. In this part, we use softmax to handle this multi-class problems.

Design and implementation tools

- Adaboost is implemented by Java in Intellij IDE
- SVM is implemented by python 2.7.12 in Pycharm IDE with Anaconda interpreter.
- Neural Network is run using TensorFlow with python 2.7.12 in terminal.

Discussion

During this project, we observed and learned more about machine learning. The most interesting part is that there are many algorithms can be used to solve one problem, handwritten digits Recognition. Beside it, there exists many different ways and theories to be selected to implement each algorithm. The result of the experiments in this project can show the best way to solve this modern problem.

Conclusion and future work

Here is a table of result of all three algorithms:

Algorithms\aspects	Accuracy (database size is 6000)	Training Time	Implementation Obstacle
AdaBoost	90.75%	$\geq$12 hr	Long Time
SVM	97.79%	$\approx$ 30min	Heavy Math
Neural Network	99.29%	$\approx$ 10 min	Complex Function

From the result, it shows Neural network has the highest accuracy and lowest training time among all 3 algorithms. However, it is really hard to implement. After that, SVM also has a high accuracy, 97.79%. It only takes about 30 minutes to calculate matrix and train data, and also not too hard to implement. However, it requires a strong math background. AdaBoost comes after other algorithms not because the accuracy, but the training time is too long. In theory, the Adaboost could have a high accuracy with Haar and stump. However, the high performance based on the high volume computation such 10000 iterations. Considering the limitation with our computers, it had only implemented on 200 iterations and the accuracy is not good as other algorithms. Therefore, it only suitable for a project with no time limitation.

In the future, we will research and implement more algorithms in this field. Algorithms, such as Decision Tree and Deep Learning can also be used to recognize handwritten digits. Moreover, we will spend more time to run the code multiple times on the same machine to make results more accurate, since it takes too long to run the code; and the training time is much depending on the machine.

User's' manual

Adaboost (Adaptive Boost)

1. Download the dataset from http://yann.lecun.com/exdb/mnist/
2. Run ImageExtraction.java to extract Haar features from image files to be new train and test features. The new train and test dataset produced as trainImage.txt and testImage.txt.
3. Run Main.java to run adaboost with ECOC.

Notes: In ECOC.java class, it could change the size of train and test dataset

Support Vector Machines

1. Install Anaconda 2 with python 2.7.12.
2. Add python.exe in Anaconda to System variables.
    a. Open control panel>System and Security>System>Advanced system settings>Environment Variables
    b. Add Anaconda folder into path
3. Open command prompt
4. Run python main.py in the implementation

Neural Network

1. Install TensorFlow followed by Download and Setup
2. In terminal, command ‘python’ to open python 2.7.12 in Mac
3. Imported TensorFlow and download dataset followed by nn.py
4. Command rest of script in nn.py to run CNN model.

References

Freund, Yoav, and Robert E. Schapire. "A desicion-theoretic generalization of on-line learning and an application to boosting." European conference on computational learning theory. Springer Berlin Heidelberg, 1995.
Kégl, Balázs, and Róbert Busa-Fekete. "Boosting products of base classifiers." Proceedings of the 26th Annual International Conference on Machine Learning. ACM, 2009.
Abadi, Martın, et al. "TensorFlow: Large-scale machine learning on heterogeneous systems, 2015." Software available from tensorflow. org 1 (2015).
Chih-Wei Hsu, Chih-Chung Chang and Chih-Jen Lin. “a practical guide to support vector classification.” Department of Computer Science, National Taiwan University, 2003.

Team information

This team consists of two members, Yi Xing and Shiyu Wang.

- Yi Xing is a CCIS master student. She is responsible for the research and implementation of Adaboost and using TensorFlow implement Neural Network.
- Shiyu Wang is a CCIS undergraduate student with a bachelor and master combined major in computer science and a math minor. He is responsible for the research and implementation of SVM.

Acknowledgement

- The MNIST Reader class (MNISTReader.java), which is a standard method to read the binary file to be pixel matrix, is provided by Gabe Johnson in the Adaboost package.
- The neural network code cited from TensorFlow tutorial website.