Digits Recognition

Ziyi Feng, Xingjian Jiang, Yimin Yang, Qiubao Ye

Mechanical Engineering Department

University of Washington

Abstract

The purpose of this project is to apply several machine learning algorithms on digits recognition. The training set has 33600 data and test set consists of 8400 data, both of which are downloaded from kaggle.com. After we load these two data sets, we use the several machine learning algorithms based on the training data set to exam the test data. Then, there is a comparison between the original data set from test data set after using these several algorithms. Finally, the accuracy was obtained and then the reasons of feasibility and failure about these algorithms were analyzed.

Sec. I. Introduction and Overview

Nowadays, people usually use laptops to type document, and if we can recognize and convert handwritten into document editor, we can save a lot of time instead of typing them again. Thus, the goal of this project is to determine what the digit is by proper algorithms. To achieve this goal, we use several different machine learning algorithms, including K-means, Naive Bayes, BP Neural Networks, Linear Discriminant Analysis, Support vector machines and K-nearest Neighbors.

First, we apply the k-means algorithm to the digit recognizer. In the first place, the relationship between the accuracy and the percentage of the training data is studied. And it is found that the accuracy is stable around 80% and increases slightly with the increase of the percentage of training data, which is not ideal enough. Also we observe that the k-means algorithm is an effective algorithm because it only takes a short period time to do the calculation.

Secondly, we apply the k nearest-neighbors algorithm to the digit. The k-nearest neighbors (KNN) rule is one of the oldest and simplest methods for pattern classification. The function of

KNN is only approximated locally and all computation is deferred until classification. It classifies each unlabeled example by the majority label among its k-nearest neighbors in the training set [1]. 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). If k = 1, then the object is simply assigned to the class of its nearest neighbor.

Thirdly, we apply the BP neural networks algorithm to the digit recognizer. Artificial neural networks are families of models inspired by biological neural networks and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. And back-propagation algorithm is a well-known method of training multilayer Feed-forward artificial neural networks [2].

Moreover, naïve Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite sets.

Also, linear Discriminant Analysis (LDA) is a method used in machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination would be used as a linear classifier in our problem.

Finally, support vector machines (SVM) offers one of the most robust and accurate methods among all well-known algorithms. It has a sound theoretical foundation, requires only a dozen examples for training, and is insensitive to the number of dimensions. In addition, efficient methods for training SVM are also being developed at a fast pace.

Sec. II. Theoretical Background

K-Means:

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. The description of the k-means algorithm is shown as follows. Given a

set of observations (x_1, x_2,⋯, x_n), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k (⩽ n) sets S = {S_1, S_2,⋯, S_k} so as to minimize the inter-cluster sum of squares (ICSS) (sum of distance functions of each point in the cluster to the k center). In other words, its objective is to find:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} \left\| x - \mu_i \right\|_2^2 \tag{1}$$

where μ_i is the mean of points in S_i. Commonly used initialization methods are Forgy and Random Partition. [3] The Forgy method randomly chooses k observations from the data set and uses these as the initial means. The Random Partition method first randomly assigns a cluster to each observation and then proceeds to the update step, thus computing the initial mean to be the centroid of the cluster's randomly assigned points. The Forgy method tends to spread the initial means out, while Random Partition places all of them close to the center of the data set. According to Hamerly et al. the Random Partition method is generally preferable for algorithms such as the k-harmonic means and fuzzy k-means. For expectation maximization and standard k-means algorithms, the Forgy method of initialization is preferable. A demonstration of the standard algorithm is shown as follows.

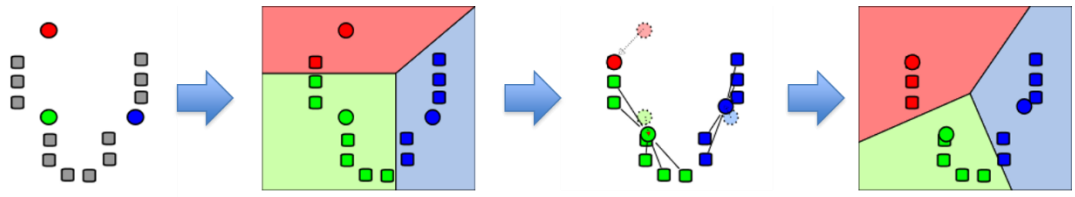

Figure 1. A demonstration of the standard algorithm

The steps of the standard algorithm are shown as follows. In the first step, k initial "means" (k=3 in Figure 5) are randomly generated within the data domain (shown in color in Figure 5). Secondly, k clusters are created by associating every observation with the nearest mean. Then the centroid of each of the k clusters becomes the new mean. At last, steps 2 and 3 are repeated until convergence has been reached. As it is a heuristic algorithm, there is no guarantee that it will converge to the global optimum, and the result may depend on the initial clusters. As the algorithm is usually very fast, it is common to run it multiple times with different starting conditions. However, in the worst case, k-means can be very slow to converge: in particular it has been shown that there exist certain point sets, even in 2 dimensions, on which k-means takes

exponential time, that is $2^{\Omega(n)}$, to converge.[4] These point sets do not seem to arise in practice: this is corroborated by the fact that the smoothed running time of k-means is polynomial. [5]

K nearest neighbor:

Officially explain KNN is that given a training data set, for every new input data, we can find K data which are close to the input data. Then to classify an unlabeled object, the distance of this object to the labeled objects is computed, its k-nearest neighbors are identified, and the class labels of these nearest neighbors are then used to determine the class label of the object. Since the points are in feature space, they have a notion of distance – This need not necessarily be Euclidean distance although it is the one commonly used. From Euclidean distance, if x = (x_1,...,x_n) and y = (y_1,...,y_n) are two points in Euclidean n-space, then the distance from x to y, or from y to x is given by the Pythagorean formula:

$$d(x,y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2}=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2} \tag{2}$$

The Euclidean norm of a vector measures the length of the vector:

$$\left\|\vec{x}\right\|_2=\sqrt{|x_1|^2+\cdots+|x_n|^2} \tag{3}$$

Given a training set D and a test object x = (x′, y′), the algorithm computes the distance (or similarity) between z and all the training objects (x, y) $\in$ D to determine its nearest-neighbor list, Dz. (x is the data of a training object, while y is its class. Likewise, x′ is the data of the test object and y′ is its class.) Once the nearest-neighbor list is obtained, the test object is classified based on the majority class of its nearest neighbors:

$$\textit{Majority Voting}: \; y'=\arg\max_{v}\sum_{(x_i,y)\in D_z} I(v=y_i) \tag{4}$$

where v is a class label, y_i is the class label for the i nearest neighbors, and I(•) is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

BP neural network:

Back propagation neural network is one kind of neural networks with most wide application. It is based on gradient descent method which minimizes the sum of the squared errors between the actual and the desired output values. The structure is showed in Figure 2.

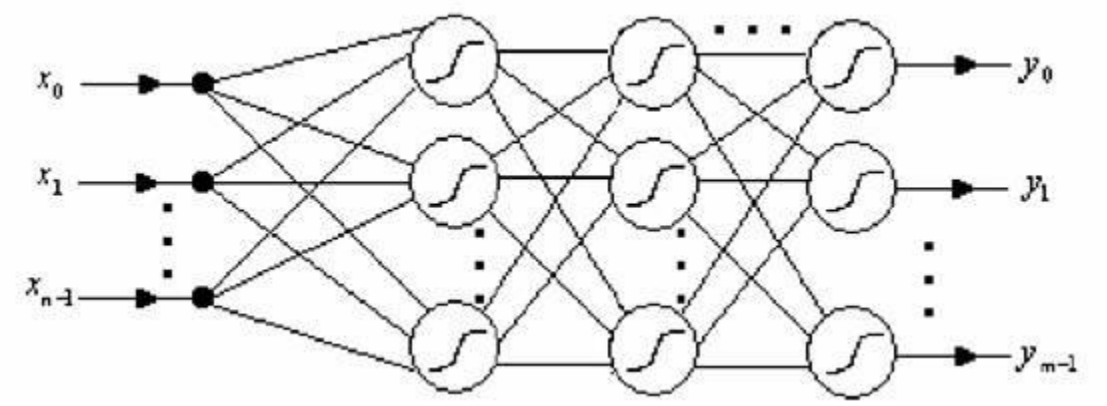

Figure 2. Structure of BP neural networks.

The basic formula of BP algorithm is:

$$W(n) = W(n-1) - \Delta W(n)$$
$$\Delta W(n) = \eta \frac{\partial E}{\partial W}(n-1) + \alpha \Delta W(n-1) \tag{5}$$

In the formula: W means weight, η means learning rate, E means gradient of error function, $\Delta W(n-1)$ means weight incremental quantity according to Kolmogorov theorem and BP fix quantify. Three layers BP network with nonlinear function as excitation function can approach any continual function in any precision.

Naïve Bayes:

Naïve Bayes is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. We would assume two classes, labeled i = 0, 1. The aim is to construct a score model that larger scores are corresponding to class 1 objects and smaller scores to class 0 objects. If we define $P(i|x)$ to be the probability that an object with measurement vector $x = (x_1, \ldots, x_p)$ belongs to class i, then any monotonic function of $P(i|x)$ would make a suitable score. The ratio $P(1|x)/P(0|x)$ can be rewritten as following

$$\frac{P(1|x)}{P(0|x)} = \frac{f(x|1)P(1)}{f(x|0)P(0)} \tag{6}$$

Given the independence assumption, the ratio above becomes

$$\frac{P(1|x)}{P(0|x)} = \frac{\prod_{j=1}^{p} f(x_j|1)P(1)}{\prod_{j=1}^{p} f(x_j|0)P(0)} = \frac{P(1)}{P(0)} \prod_{j=1}^{p} \frac{f(x_j|1)}{f(x_j|0)} \tag{7}$$

Now, recalling that our aim was merely to produce a score which was monotonically related to P(i|x), we can take logs of equation above—log is a monotonic increasing function. This gives an alternative score:

$$\text{In} \frac{P(1|x)}{P(0|x)} = \text{In} \frac{P(1)}{P(0)} + \sum_{j=1}^{p} \text{In} \frac{f(x_j|1)}{f(x_j|0)} \tag{8}$$

So that the classifier has a particularly simple structure.

Linear Discriminant Analysis:

For a two-class LDA, construct a projection w such that

$$w = \arg\max \frac{w^T S_B w}{w^T S_W w} \tag{9}$$

where the scatter matrices for between-class S_B and within-class S_w data are given by

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \tag{10}$$

$$S_w = \sum_{j=1}^{2} \sum_{x} (x - \mu_j)(x - \mu_j)^T \tag{11}$$

These quantities essentially measure the variance of the data sets as well as the variance of the difference in the means. The generalized Rayleigh quotient whose solution can be found via the generalized eigenvalue problem

$$S_B w = \lambda S_w w \tag{12}$$

where the maximum eigenvalue λ and its associated eigenvector gives the quantity of interest and the projection basis. Thus once the scatter matrices are constructed, the generalized eigenvectors can easily be constructed with MATLAB.

Support vector machines:

In a two-class learning task, the aim of SVM is to find the best classification function to distinguish between members of the two classes in the training data. The metric for the concept of the "best" classification function can be realized geometrically. For a linearly separable dataset, a linear classification function corresponds to a separating hyperplane f(x) that passes

through the middle of the two classes, separating the two. Once this function is determined, new data instance x_n can be classified by simply testing the sign of the function $f(x_n)$; xn belongs to the positive class if $f(x_n) > 0$. Because there are many such linear hyperplanes, what SVM additionally guarantee is that the best such function is found by maximizing the margin between the two classes. Intuitively, the margin is defined as the amount of space, or separation between the two classes as defined by the hyperplane. Geometrically, the margin corresponds to the shortest distance between the closest data points to a point on the hyperplane. To ensure that the maximum margin hyperplanes are actually found, an SVM classifier attempts to maximize the following function with respect to $\vec{w}$ and b:

$$L_p = \frac{1}{2}\|w\| - \sum_{j=1}^{t} \alpha_i y_i (w \cdot x_i + b) + \sum_{i=1}^{t} \alpha_i \qquad (13)$$

where t is the number of training examples, and α_i, i = 1, . . . , t , are non-negative numbers such that the derivatives of L_P with respect to α_i are zero. αi are the Lagrange multipliers and L_P is called the Lagrangian. In this equation, the vectors $\vec{w}$ and constant b define the hyperplane.

Sec. III. Algorithm Implementation and Development

The K means algorithm can be summarized as:

Since we have already known the label of the training data, it is unnecessary for us to determine the mean of each cluster with tens of iterations any more. Instead, we can calculate the mean of each cluster directly by dividing the training data into ten clusters according to the labels and averaging the data in each cluster. After that we can use the formula (1) to calculate the distance from a data in test data to the mean of each cluster. Then this data should belong the cluster which has the nearest distance to it. And we can apply this algorithm to every point of the test data to determine what number is this data indicates.

The K nearest neighbor algorithm can be summarized as:

1. A positive integer k is specified, along with a new sample
2. Selecting the k entries in our database which are closest to the new sample
3. Calculating the distance

4. To compare the data. If the input data larger than max distance, the go find next one. If the input data less than max distance, then delete the largest distance and store the training data instead.

5. Find the most common classification of these entries

6. This is the classification we give to the new sample

To make this algorithm more detail, following is a spread of red circles and green squares:

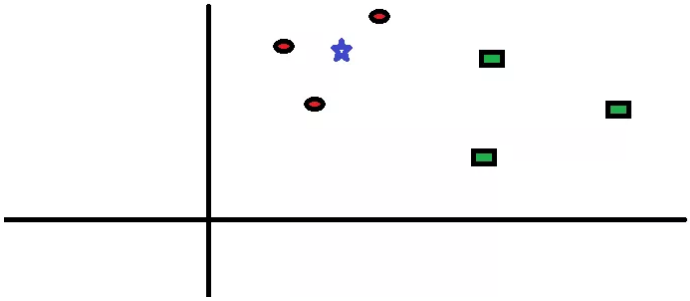

Figure 3. Demonstrate new data classification

From Figure 3, you intend to find out the class of the blue star. blue star can either be red circles or green squares and nothing else. The "K" is KNN algorithm is the nearest neighbors we wish to take vote from. Let's say K = 3. Hence, we will now make a circle with blue star as center just as big as to enclose only three data points on the plane. Refer to following diagram for more details:

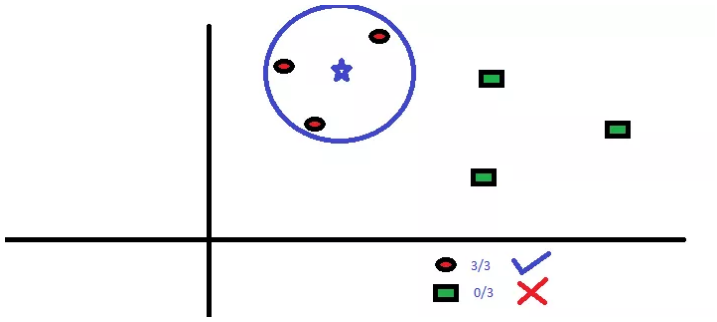

Figure 4. Demonstrate new data classification by using knn

From Figure 4, the three closest points to blue star is all red circles. Hence, with good confidence level we can say that the blue star should belong to the class red circles. Here, the choice became very obvious as all three votes from the closest neighbor went to red circles. The choice of the parameter K is very crucial in this algorithm. Next we will understand what are the factors to be considered to conclude the best K. [6]

The BP neural network algorithm can be summarized as:

Step1. Load the training input data and make normalization between 0 and 1.

Step2. Load the training output data and reconstruct it:

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ \vdots \\ 9 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & & & 0 \\ 0 & & 1 & & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Step3. Build the neural network with input/output data, hidden layers (2 layers), nodes ([300 50]), excitation function (hyperbolic tangent sigmoid transfer function and logarithmic sigmoid transfer function) and training method (traingdx).

Step4. Add training parameters: learning rate (0.001), maximum iterations (500), goal error (0.0000001).

Step5 Training the BP neural network.

Step6. Load the training input data and make normalization between 0 and 1

Step7. Simulate and get the output

Step8. Calculate the accuracy

The Naive Bayes, Linear Discriminant Analysis and Support vector machines algorithm can be summarized as:

Preprocessing: Every digits image in our training data set consists of 784 pixels. So, before applying multiple machine learning algorithm to the data set, we would like to employ the Singular Value Decomposition (SVD) to extract the features. And we pick up the features ordering from 2 to 50 as the training "input".

Main program: Then, Naïve Bayes, Linear Discriminant Analysis and Support Vector Machines are used to construct the classification models with the corresponding "input". The MATLAB implementation is shown below:

```
[u,s,v] = svd(A1,'econ');
xtrain = v(8401:end,2:50);  % pick 49 features
xtest = v(1:8400,2:50);
ctrain2 = ctrain(8401:end); % training label
ctest = ctrain(1:8400);     % test label
```

```
%%naive bayes
nb = fitNaiveBayes(xtrain,ctrain2);
pre = nb.predict(xtest);
%% LDA
pre = classify(xtest,xtrain,ctrain2);
%% SVM
svm = fitcecoc(xtrain,ctrain2);
pre = predict(svm,xtest);
```

The variable “pre” stores the predictions given by our classification model with the test data set.

Sec. IV. Computational Results

For the result of k-means, it is not ideal. We use different percentages of the training data, and the relationship between the percentage of training data and the accuracy rate is shown in Figure 5.

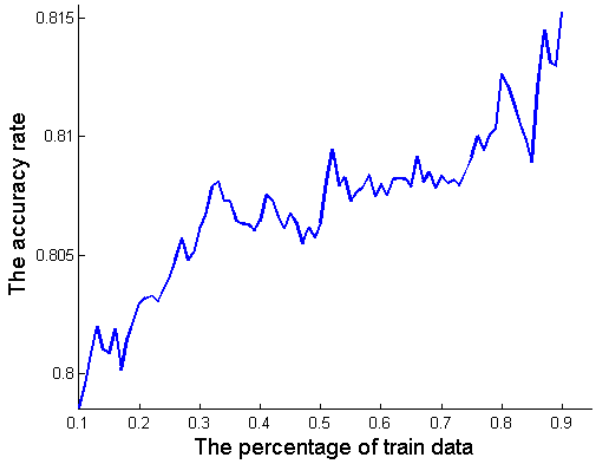

Figure 5. The relationship between the percentage of training data and the accuracy rate

It can be found that the accuracy rate is stable around 80% and increase slightly with the increase of the percentage of training data. This result is not that ideal because of the following reason. As we have shown in Fig.1, for the two-dimensional data, the partitions of the k-means algorithm are always ‘lines’ (‘planes’ if the data is three-dimensional). However, if the data of different clusters jumble together, the ‘lines’ or ‘planes’ cannot divide clusters accurately, which leads to the unsatisfactory results in digit recognizer. Instead, curves or curved surface should be used as

the partitions in this problem. However, it is found that the k-means is an effective algorithm. In this problem, it only takes 7.12 seconds to obtain the calculation results, which is extremely fast comparing with other algorithms.

For the result of K nearest neighbor in our project, 80% of the data were used to train the system and rest 20% to test the K-NN Algorithm. By using this method, for k = 100, the error is 7.82%, which means it can recognize around 92% of digits but it takes 1956seconds to classify the entire data.

Table 1. The cooperation between test labels and test results

Test Labels			Test Results		
Value	Count	Percent	Value	Count	Percent
0	852	10.14%	0	873	10.39%
1	950	11.31%	1	1209	14.39%
2	801	9.54%	2	687	8.18%
3	882	10.50%	3	858	10.21%
4	811	9.65%	4	757	9.01%
5	747	8.89%	5	719	8.56%
6	831	9.89%	6	856	10.19%
7	884	10.52%	7	897	10.68%
8	802	9.55%	8	663	7.89%
9	840	10.00%	9	881	10.49%

But if we choose k = 3, the error is 3.91%, which means it can recognize around 96% of digits but it takes 999 seconds to classify the entire data. From the table, it is obviously see that the percentage are pretty much same, because the error percentage is very low.

The result of BP neural network, the overall data is 42000 and we use 41000 of it to train the system and use left 1000 to test the BP neural network. After training, the BP neural network can recognize 87.7% of digital number. The training result is showed in Figure 6.

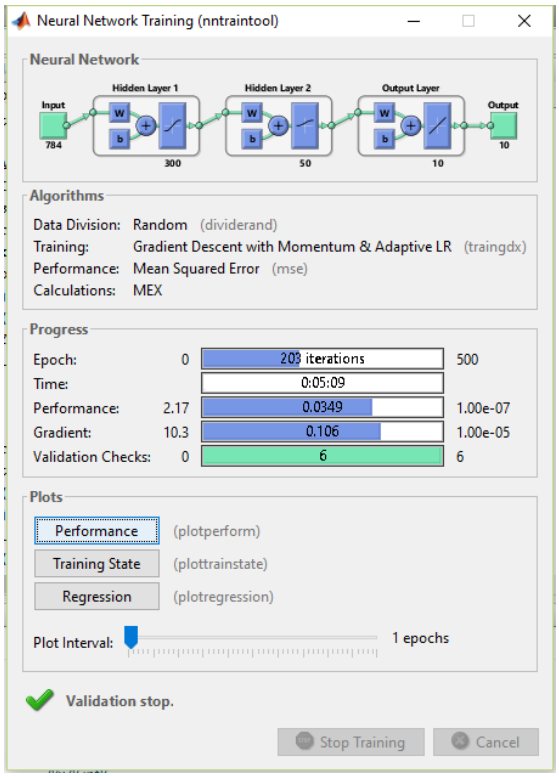

Figure 6. The training result of BP neural network

Compared with KNN method which has highest accuracy, BP neural network is not so good in this case. There may be several reasons: firstly, the training data is not big enough and especially for those number not in normal style, for example, the number showed in Figure 7. It's hard for us to recognize those numbers even based on images.

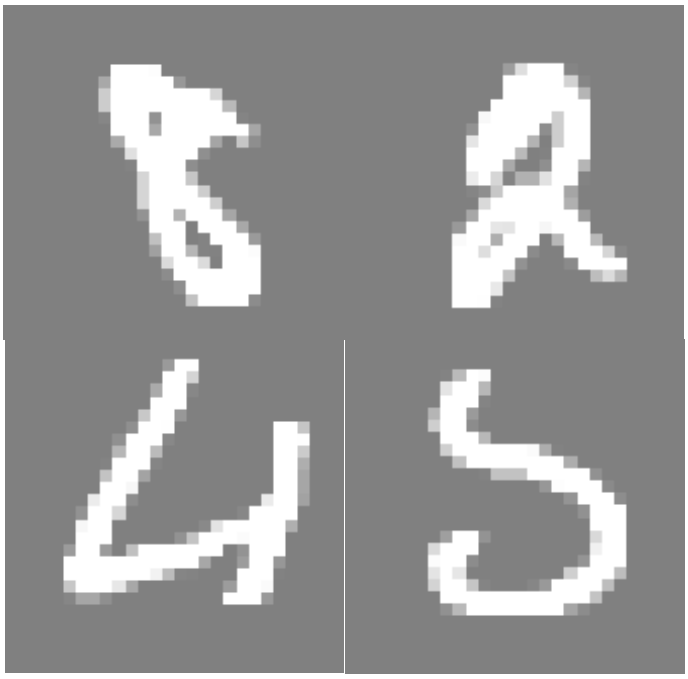

Figure 7. Unrecognized numbers

Secondly, the excitation function is not perfect. There are many other types of excitation functions and some of them may behave better in this case. Thirdly, we use the existing function and training method in MATLAB, they are general for all case.

For the result of Naïve Bayes, the recognition accuracy given by Naïve Bayes in our digit recognition problem is 86.76%, which is surprisingly well since we did not even optimize the algorithm for this certain problem and its data set. The reason why Naïve Bayes works in this problem is that Naïve Bayes is proved to be robust and can be applied to many machine learning problems.

For the result of LDA, the recognition accuracy given by LDA in our digit recognition problem is 85.87%. In this case, every feature used for training is independent from other features and this is why the LDA works for our reduced dimensional training data set.

For the result of SVM, the error rate given by the SVM algorithm is as high as 50%, which is not expected from us before we apply it to our training data set. The reason might lies on that SVM shows unstable properties when dealing with multiclass model problems.

Sec. V. Summary and Conclusions

To sum up k means algorithm, it is found that the accuracy of the digit recognizer is stable around 80% and increase slightly with the increase of the percentage of training data, which is not ideal enough. This is so because if the data of different clusters jumble together, the 'lines' or 'planes' cannot divide clusters accurately. Also we can observe that the k-means algorithm is an effective algorithm because it only takes a short period time to do the calculation. Thus, we can come to the conclusion that although the k-means is an effective algorithm, it cannot perform well in the problem of digit recognizer because of the low accuracy.

K nearest neighbor is a simple algorithm for pattern classification, and has a high correctness and stable in our project, which can achieve correctness around 96%. To compare with other methods, the advantage of K nearest neighbor is easy to understand and achieve the algorithm conveniently. The disadvantage is that we make the algorithm more complicated, which is K*M(train data size)*N(test data size). Also, it takes very longer time than other methods, because every new data are need to calculate the distance with training data, which is 33600 in our case. In a big data problem, it is not an efficient way to choose K nearest neighbor algorithm.
BP neural network gets 87.7% accuracy and is a good one overall. Besides, there are many aspects of it, like training methods and excitation function which can be optimized to better the training result.

The Naïve Bayes is very simple while powerful algorithm used in various applications. And LDA is also widely used in statistics, pattern recognition and machine learning problems. The SVM is definitely a powerful tool and shows high accuracy for certain kind of problem. However, in other cases SVM might not be as suitable as the other high performance algorithm.
Several machine learning algorithms are applied to this digit recognizer project, which as k-means, Naïve Bayes, BP neural networks, Linear Discriminant Analysis, Support vector machines and K-nearest neighbors. Some of the algorithms can recognize the digits accurately but take a long time to run, while some of others are efficient, but the accuracy is not satisfactory. Thus, it is necessary for us to choose an appropriate algorithm according to its accuracy and efficiency. All of the algorithms we present here have acceptable accuracy rate, which indicates

that they can be applied to the recognizer. For further application, the recognizer can recognize not only digits, but also words, characters and symbols.

Reference

[1] Zhang, Hao, et al. "SVM-KNN: Discriminative nearest neighbor classification for visual category recognition." *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 2. IEEE, 2006.
[2] Newff: net=newff (PR,[S1 S2… SN-1], {TF1 TF2…TFN-1}, BTF, BLF, PF). It creates a feed-forward back propagation network.
[3] Hamerly, G. and Elkan, C. (2002). "Alternatives to the k-means algorithm that find better clusterings". Proceedings of the eleventh international conference on Information and knowledge management (CIKM).
[4] Vattani., A. (2011). "k-means requires exponentially many iterations even in the plane". Discrete and Computational Geometry 45 (4): 596–616.
[5] Arthur, D.; Manthey, B.; Roeglin, H. (2009). "k-means has polynomial smoothed complexity". Proceedings of the 50th Symposium on Foundations of Computer Science (FOCS).
[6]Srivastava, T. (2014, October 10). Introduction to k-nearest neighbors: Simplified. Retrieved from http://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/

Appendix A MATLAB functions used and brief implementation explanation

repmat: Replicate and tile an array. B = repmat(A,M,N) creates a large matrix B consisting of an M-by-N tiling of copies of A. If A is a matrix, the size of B is [size(A,1)*M, size(A,2)*N].

sum: Sum of elements, which is the sum of the elements of the vector X.

sort: Sort in ascending or descending order.

tabulate: Frequency table. TABLE = tabulate(X) takes a vector X and returns a matrix, TABLE. The first column of TABLE contains the unique values of X. The second is the number of instances of each value. The last column contains the percentage of each value.

newff: net=newff (PR,[S1 S2… SN-1], {TF1 TF2…TFN-1}, BTF, BLF, PF). It creates a feed-forward back propagation network.

train: [net, tr, realoutput, error]=train(net, input, output). It trains a network net according to input and output.

sim: Y=sim (net, input). It simulates neural networks and return estimated output Y.

Appendix B MATLAB codes

```
%%K-means
close all; clear all; clc;

tic
train = csvread('train.csv');

m = size(train,1);
n = size(train,2);

h1 = 0.1:0.01:0.9;
h = 0.1:0.01:0.9;
len = length(h);
p = zeros(1,len);
k = 1;
for h = 0.1:0.01:0.9
train1 = train(1:(m*h),2:end);
trainlabel = train(1:(m*h),1);
test = train((m*h+1):end,2:end);
testlabel = train((m*h+1):end,1);

means = zeros(10,n-1);
for j = 1:10
    means(j,:) = findmean(train1,trainlabel,j-1);
end

l = length(test);
testresult = zeros(1,l);
for j = 1:l;
    testresult(j) = find_min_distance(test(j,:),means);
```

```
end
testresult = testresult';
s = 0;
for j = 1:l;
    if (testlabel(j) == testresult(j))
        s = s+1;
    end
end
p(k) = s/l;
k = k+1
end
axis([0.1 0.95 0.7985 0.8155]);
hold on
plot(h1,p,'b','Linewidth',[2]);
xlabel('The percentage of train data','Fontsize',14);
ylabel('The accuracy rate','Fontsize',14);
toc

function Y = findmean(X1,X2,n)

% X1: the train data
% X2: the train label
% n: the label we would like to calculate

m = size(X1,1);
s = 0;
c = 0;
for j = 1:m
    if X2(j,1) == n
        s = s+X1(j,:);
        c = c+1;
    end
end
Y = s/c;
End

function k = find_min_distance(X1,means)

% X1: the point we give
% means: the ten means

n = size(means,1);
distance = zeros(n);
for j = 1:n
    distance(j) = norm(X1-means(j,:),2);
end
m = distance(1);
for j = 1:n
    if (m >= distance(j))
        m = distance(j);
        k = j-1;
    end
end
end


%% k nearest neighbor
```

```
clear all; close all; clc

A = csvread('train.csv');

B = A(1:33600,2:785)'; %Train
C = A(33601:end,2:785)';%Test 42000


trainImages = B;
trainLabels = A(1:33600,1);

N = 784;
K = 100; % can be any other value precentage
testImages = C;
testLabels = A(33601:end,1);

trainLength = length(trainImages);
testLength = size(testImages,2);
testResults = linspace(0,0,length(testImages));
compLabel = linspace(0,0,K);
tic;

for i=1:testLength
    curImage = repmat(testImages(:,i),1,trainLength);
    curImage = abs(trainImages-curImage);
    comp=sum(curImage);
    [sortedComp,ind] = sort(comp);
    for j = 1:K
        compLabel(j) = trainLabels(ind(j));
    end
    table = tabulate(compLabel);
    [maxCount,idx] = max(table(:,2));
    testResults(i) = table(idx);

    disp(testResults(i));
    disp(testLabels(i));
end

% Compute the error on the test set
error=0;
for i=1:testLength
  if (testResults(i) ~= testLabels(i))
    error=error+1;
  end
end

%Print out the classification error on the test set
error/testLength
toc;
disp(toc-tic);

%%BP neutral network
clear all; clean all;clc
A=train_notitle;
AA=zeros(4200,784);
for k=1:42000
for j=1:784
```

```
AA(k,j)=A(k,j)/255;
end
end
AA1=AA(1:41000,:);
input= AA1';
M=train1(:,1);
M1=M(1:41000,1);
output=zeros(41000,10);
for j=1:41000
if M(j,:)==0
    output(j,:)=[1 0 0 0 0 0 0 0 0 0];
elseif M(j,:)==1
    output(j,:)=[0 1 0 0 0 0 0 0 0 0];
elseif M(j,:)==2
    output(j,:)=[0 0 1 0 0 0 0 0 0 0];
elseif M(j,:)==3
    output(j,:)=[0 0 0 1 0 0 0 0 0 0];
elseif M(j,:)==4
    output(j,:)=[0 0 0 0 1 0 0 0 0 0];
elseif M(j,:)==5
    output(j,:)=[0 0 0 0 0 1 0 0 0 0];
elseif M(j,:)==6
    output(j,:)=[0 0 0 0 0 0 1 0 0 0];
elseif M(j,:)==7
    output(j,:)=[0 0 0 0 0 0 0 1 0 0];
elseif M(j,:)==8
    output(j,:)=[0 0 0 0 0 0 0 0 1 0];
elseif M(j,:)==9
    output(j,:)=[0 0 0 0 0 0 0 0 0 1];
end
end
net = newff( input,output', [300 50] , { 'tansig' 'logsig'} , 'traingdx' );
net.trainparam.show = 50 ;
net.trainparam.epochs = 500 ;
net.trainparam.goal = 0.0000001 ;
net.trainParam.lr = 0.001 ;
[net,tr,realoutput,error] = train( net, input , output' );
AA2=AA(41001:42000,:);
testInput = AA2' ;
Y = sim( net , testInput );
Z=zeros(1000,1);
for k=1:1000
    big=0;
for j=1:10
if Y(j,k)-big>=0
big=Y(j,k);
Z(k,1)=j-1;
end
end
end
E=zeros(1000,1);
count=0;
M2=M(41001:42000,1);
for j=1:1000
E(j,1)=Z(j,1)-M2(j,1);
if E(j,1)==0
    count=count+1;
```

```
end
end
count

clear; close all; clc;
M1 = csvread('train.csv',1,0);
M2 = csvread('test.csv',1,0);

% A = M1(1,2:end); A = reshape(A,28,28);
% imshow(A)

ctrain = M1(:,1);   % label
M1 = M1(:,2:end);   % remove the label column
A1 = M1';           % transpose the matrix (like dotcat problem)
A2 = M2';           % transpose test matrix

% extract the feature
[u,s,v] = svd(A1,'econ');
%[u1,s1,v1] = svd(A2,'econ');
figure()
plot(diag(s),'ko','Linewidth',[2])
title('singular value','FontSize',15)

figure()
semilogy(diag(s),'ko','Linewidth',[2])
title('singular value','FontSize',15)

figure()
for j = 1:4
    subplot(4,1,j),plot(v(8401:8501,j),'LineWidth',[3])
    if j == 1
        title('Projection of first 100 images onto the first 4 principal
component','FontSize',15)
    end
end

% principal component
figure()
for j = 1:9
    subplot(3,3,j)
    pc = reshape(u(:,j),28,28);
    pcolor(pc),axis off, shading interp,colormap(hot)
    if j == 2
        title('Nine Principal Component of MNIST Database','FontSize',15)
    end
end

xtrain = v(8401:end,2:50);  % pick 49 features
xtest = v(1:8400,2:50);
ctrain2 = ctrain(8401:end); % training label
ctest = ctrain(1:8400);     % test label


% naive bayes
tic
nb = fitNaiveBayes(xtrain,ctrain2);
pre = nb.predict(xtest);
```

```
toc

num = 0;

for jj = 1:8400
    if ctest(jj) == pre(jj)
        num = num + 1;
    end
end

rate1 = num/8400;

figure()
subplot(2,1,1), bar(ctest);
title('test data','FontSize',20)
ylabel('digit number','FontSize',15)
subplot(2,1,2), bar(pre);
title('prediction data','FontSize',20)
xlabel('Naive Bayes algorithm, accuracy rate: 86.76%','FontSize',20)
ylabel('digit number','FontSize',15)

% LDA
pre = classify(xtest,xtrain,ctrain2);

num = 0;

for jj = 1:8400
    if ctest(jj) == pre(jj)
        num = num + 1;
    end
end

rate2 = num/8400;

figure()
subplot(2,1,1), bar(ctest);
title('test data','FontSize',20)
ylabel('digit number','FontSize',15)
subplot(2,1,2), bar(pre);
title('prediction data','FontSize',20)
xlabel('Linear Discriminant Analysis algorithm, accuracy rate:
85.87%','FontSize',20)
ylabel('digit number','FontSize',15)


% SVM cannot train an SVM model for more than 2 classes
% svm = fitcsvm(xtrain,ctrain2);
% pre = ClassificationSVM(svm,xtest);
svm = fitcecoc(xtrain,ctrain2);
pre = predict(svm,xtest);

num = 0;

for jj = 1:8400
    if ctest(jj) == pre(jj)
        num = num + 1;
    end
```

```
end

rate2 = num/8400;

figure()
subplot(2,1,1), bar(ctest);
title('test data','FontSize',20)
ylabel('digit number','FontSize',15)
subplot(2,1,2), bar(pre);
title('prediction data','FontSize',20)
xlabel('Support Vector Machines algorithm, accuracy rate:
51.1%','FontSize',20)
ylabel('digit number','FontSize',15)
```