Handwritten Digit Classification

Using Numerical Algorithm

Supervised by

Cho Hong, Min

Ewha W, University

Mathematic Dep.

Written by

Hye Mee, Sohn

Hye Yun, Oh

Yu Jin, Jung

Ewha W. University

Computational Science Dep.

**1.Introduction**

The MNIST database is a large dataset of handwritten digits. The MNIST database is widely used for training and testing in the field of machine learning. In this paper, we consider image datasets of 10 digits by university students and try to classify those handwritten digits numerically. There are various numerical methods for predicting patterns of handwritten digits. Above all, we chose two methods. This paper explores the patterns of digits by using weighted graph Laplacian method and Singular Value Decomposition (SVD) method. The weighted graph Laplacian method is first presented and SVD method is discussed in next section. We want to find efficient method to show better performance compared to the machine learning algorithm which is widely used.

**2. Weighted Graph Laplacian**

**2.1 Theory and Algorithm**

Graph Laplacian is well-known approach which has been used widely in many data analysis and machine learning problems. Let be a set of points in and be a subset of . Let be a function on the point set and the value of on is given as a function over , i.e. , .

To minimize the following energy functional,

with the constraint

we use Gaussian weight, , is a parameter, is the Euclidean norm in . This model is widely used in many applications. By deriving the Euler-Lagrange equation of the above optimization problem, we can get

This is a graph Laplacian approach. However, the solution given by this graph Laplacian approach is often non-continuous. This is because of the fact that one important boundary term is dropped. Hence, we add a weight to balance the energy on the labeled and unlabeled sets on . Separate the into two terms, one over the unlabeled set and the other over the labeled set.

To assure the continuity on the labeled set, put a weight ahead of the summation over the labeled set.

Since this method is obtained by modifying the graph Laplacian to add weight inside, it is called weighted graph Laplacian, WGL for short. For , in this case, choose the inverse of the sample rate, . Then we get

with the constraint

, are the number of points in and , respectively. When is high, WGL above becomes ordinary graph Laplacian and when is low, the large weight in the weighted graph Laplacian makes solution to be continuous near the labeled set.

By solving linear system, we can get

This linear system is symmetric and positive definite.

**2.2 Methodology for Digit Classification**

Assume we have a point set, , and labels . A subset is labeled,

is the labeled set with the label . The size of the labeled set is much smaller than the size of the data set . MNIST data set contains gray scaled digit images. View ten digits 0~9 as ten classes. Our goal is to extend the label assignment to the entire set and infer the labels for the unlabeled points. We can obtain by solving the following linear system with conjugate gradient method.

The weight function is constructed as

, where is chosen to be the distance between and its 10th nearest neighbor.

Actual programming code will be in appendix.

**Algorithm** Semi-Supervised Learning

**Require:** A point set and a partial labeled set

**Ensure:** A complete label assignment

**for** **do**

Compute on , with the constraint

**end for**

**for** **do**

Label as following

, where .

**end for**

**2.3 Results**

Two image sets were used to test the Weighted Graph Laplacian method. One of the sets contains 200 test images and the other set contains 600 test images. We did not use other MNIST Database which is opened to everyone. Instead, all of these test sets were made by us. We used 30 image samples for each digit. By using our handwritten test sets, we got two results as follows.

|  |  |
| --- | --- |
| 200 test images | 600 test images |
| 9.0% | 12.3% |

This result table indicates that for 200 test images 9.0% of the digits are classified correctly and for 600 test images 12.3% of the digits are classified correctly. Normally, Weighted Graph Laplacian method is numerically reasonable and guarantees over 90% accuracy with total MNIST database. However, with only 30 samples for each digit, we could not get high accuracy. Therefore, we decided to find out a better method to classify digits.

**3. Singular Value Decomposition**

**3.1 Theory and Algorithm**

**3.1.1 SVD**

The singular value decomposition (SVD) of matrix can be written as where is a matrix and is an matrix. and should be a unitary matrix which means . Also, is a rectangular diagonal matrix with non-negative real numbers on the diagonal and the other elements of are all zero. The diagonal entries of are non-negative singular values of . The singular values can be listed in a descending order such that

Since and are unitary, the columns of and form a set of orthonormal vectors. By the definition of a unitary matrix, the same is true for the conjugate transposes and .

The choices of SVD are various and the three ways of SVD are as follows.

*A .*

In this paper, we especially use the formula, *.* Thes from the formula are the left singular vectors of which are unit eigen vectors of . Also, the s are the right singular vectors of .

**3.1.2 Matrix Formation**

A vector norm can be thought of as the length or magnitude of a vector . There are many examples of vector norms such that

(L1- norm)

(Euclidean norm or L2 norm).

Generally, Euclidean norm is widely used. In this paper, we regard all norms as Euclidean norms. Considering this, the two norm of a matrix is determined from the singular values with

To achieve SVD form, matrix can be written in summation form as

Our goal is to reduce a large matrix forming digit and thus we can approximate with one of a lower rank. In this summation form, there are the zero singular values which give no extra information to the matrix. The summation form can be reduced to

, where r is the rank of the matrix . Also, means the number of nonzero singular values. In other words, there can be a rank- matrix with so that is minimized. Using the previous way of formation, is denoted as , where . Then,

Using that minimization process, we will discuss a method of ten handwritten digit classification in the next section.

**3.2 Methodology for Digit Classification**

All handwritten digits can be considered as a matrix. Each of the digit matrices is compressed to a large matrix called and so the columns of each image are stacked to form a column vector of size, . Then, a matrix is in a space, with n being the number of training images for a digit and.

To find the best approximation of an unknown digit, we first need to find SVD of considering s which are the left singular vectors of one digit matrix. The s form an orthonormal basis in a digit. Therefore, the best operation can be done by finding the basis differences.

An unknown digit can be approximated by calculating the residual between the unknown digit and singular images. A minimization formula can be written as

, where is a matrix of one unknown digit and s are bases of each other digits. In this formula, is the number of bases and thus s form a large matrix . The best approximation is computed by reducing the minimization formula to zero. Then, the ideal solution is to get which can be expressed as . The differences are then as follows:

Therefore, handwritten ten digit classification can be explained in two steps: training and classification. First of all, we compute SVD of a training set of each digit. Then, an unknown digit is classified against ten differences by using the minimization formula mentioned above.

In this paper, we consider each digit image as a matrix. Also, we give 10 training images of each digit. An unknown digit is also denoted as matrix and then classified with ten digits which are classified as members of the basis forming the smallest differences.

**3.3 Results**

Two image sets were used to test the SVD classification method. Here as elsewhere we did not use other MNIST Database which is opened to everyone. Instead, test set for SVD was made by us. This test set contains 900 images which were used for the previous method, Weighted Graph Laplacian. Also, we used 30 image samples for each digit. By using our handwritten test sets, we got a result table as follows.

|  |  |
| --- | --- |
| Digit | Accuracy |
| *1* | *0.67* |
| *2* | *0.90* |
| *3* | *0.65* |
| *4* | *0.63* |
| 5 | 0.62 |
| 6 | 0.72 |
| 7 | 0.65 |
| 8 | 0.42 |
| 9 | 0.62 |
| 0 | 0.62 |
| Total | 0.65 |

For every digits, 60 test images were used and there are some differences for each digit. The accuracy table above shows that 90% of digit 2 sets were classified correctly and it is the highest accuracy among digit sets. Also, for other digits, more than 60% percent of accuracy rate was reached normally. It can be considered as good performance since the sample data set contains only 300 images, smaller than 0.5% of MNIST data set and 3% of MINST training data set.

**4. Conclusion**

We introduced two methods of finding unknown digits, the weighted graph Laplacian and Singular Value Decomposition. In the previous sections, we got the correct classification percentage, 9.0% for 200 test images and 12.3% for 600 test images by using the weighted graph Laplacian and 90 % as the highest accuracy rate by using SVD classification method. Comparing the two results respectively, the second result seems to be much better than the first one. We conclude that the first method needs more training images than we used. However, our goal is to find more efficient way compared to widely-used MNIST method. Thus, the first method seems to be inappropriate. Instead, we conclude that the second method, SVD classification method, is more convenient and accurate method for classifying handwritten digits.

We used only 300 training images for the second method. This number implies that larger training set will bring about better results. Normally, MNIST database contains 60,000 training images and 10,000 testing images. Considering the 60,000 training images, our correct classification percentage is accepted. Thus, this means that SVD classification method is the most efficient and stable method to classify handwritten digits.

**5. Summary and Future work**

Based on the weighted graph Laplacian, we add a weight to correctly approximate the unknown digits. We extend the label assignment to the entire set and infer the labels. Then, we obtain by solving the following linear system with conjugate gradient method.

In SVD method, we computed singular value decomposition and minimizing process for approximation in the second section. We separate into singular values and approximate with one of a lower rank. Then, an unknown digit is classified against ten differences by using the minimization formula.

These days, there are many uses of artificial intelligence using deep learning algorithms. We expect that the classification methods of handwritten digits to be used in various fields. Since a lot of data can be converted into a matrix form, this classification method can be expanded to various fields.

**6. References**

[1] WEIGHTED GRAPH LAPLACIAN AND IMAGE INPAINTING, Zuoqiang shi, Stanley osher, and Wei zhu

[2] Handwritten Digit Classification and Reconstruction of Marred Images Using Singular Value Decomposition, Andy Lassiter

[3] Numerical mathematics and computing, 6th edition, David Kincaid

**7. Appendix**

**7.1 Programming code for WGL method**

|  |
| --- |
| // -----------------------------------------------  // Image loading  // -----------------------------------------------  ARRAY< ARRAY2D<double> > P;  load\_images(IMNUM, P); // image loading function  // -----------------------------------------------  // Sigma calculation  // -----------------------------------------------  ARRAY <double> SIGMA(IMNUM);  ARRAY2D<double> WEIGHT(IMNUM,IMNUM);  ARRAY <double> resultSIG(IMNUM);  for(int i=0;i<IMNUM;i++)  { ARRAY<int> ngbdSet;  SIGMA[i] = sigma(P,i,10,ngbdSet);//sigma calculation function }  for (int i = 0; i < IMNUM; i++)  for (int j = 0; j < IMNUM; j++)  WEIGHT[i][j] = weight(P, SIGMA, i, j); //weight calculation function  // -----------------------------------------------  // loading sample img for test  // -----------------------------------------------  ARRAY< double > U(IMNUM); U=0;  for (int i = 0; i < SAMPLE; i++) U[i] = ((i/(SAMPLE / DIGIT))+1) % 10 ;  for (int i = SAMPLE; i < IMNUM; i++) U[i] = (((i - SAMPLE) / ((IMNUM-SAMPLE)/ DIGIT)) + 1) % 10;  //-------------------------------------------------  // create A,b  //-------------------------------------------------  ARRAY2D<double> A(IMNUM - SAMPLE,IMNUM - SAMPLE); A=0;  ARRAY <double> b(IMNUM - SAMPLE); b=0;  double r = IMNUM / SAMPLE + 1;  for (int m = 0; m < IMNUM - SAMPLE; m++)  {  // A 대각원소  for (int n = 0; n< SAMPLE; n++)  A[m][m] = A[m][m] + r \* WEIGHT[m][n];  for (int n = SAMPLE; n< IMNUM; n++)  A[m][m] = A[m][m] + 2 \* WEIGHT[m + SAMPLE][n];  // A 그 외 원소  for (int n = 0; n < IMNUM - SAMPLE; n++)  {  if (m != n)  A[m][n] = A[m][n] - 2 \* WEIGHT[m + SAMPLE][n + SAMPLE];  }  //b  for (int n = 0; n < SAMPLE; n++)  b[m] = b[m] + r \* WEIGHT[m + SAMPLE][n] \* U[n];  }  //-------------------------------------------------  // solve AX=B by CG  //-------------------------------------------------  ARRAY<double> X;  X = ConjugatGradient(A,b); // function solving AX=B by Conjugated Gradient method  X.print("U"); // Result of digit regocnition |

**7.2 Programming code for WGL method (except declaration part)**

|  |
| --- |
| //----------------------------------------------------------------------  // load Image  //----------------------------------------------------------------------  oad\_sample(IMG);  //----------------------------------------------------------------------  // Calculatin U  //----------------------------------------------------------------------  initialize3D(U);  {  ARRAY2D<double> u(W\*H, W\*H);  ARRAY2D<double> w(SAMNUM, SAMNUM);  ARRAY<double> lambda(SAMNUM);  ARRAY<double> dist(SAMNUM);  ARRAY2D<double> vw(SAMNUM, SAMNUM);  ARRAY2D<double> Residual(SAMNUM, SAMNUM);  double dist\_fin;  double temp;  for (int d = 0; d < DIGIT; d++) { // DIGIT LOOP START  A = IMG[d];  At = t(A);  //----------------------------------------------------------------------  // initial random V  //----------------------------------------------------------------------  dist\_fin = 0;  for (int k = 0; k < SAMNUM; k++)  for (int j = 0; j < SAMNUM; j++)  V[k][j] = ((double)rand()) / RAND\_MAX;  //----------------------------------------------------------------------  // power method - calculating V, w  //----------------------------------------------------------------------  vw = initialize(vw, 0);  temp = 0;  for (int k = 0; k < SAMNUM; k++) dist[k] = 1E8;  do {  for (int k = 0; k < SAMNUM; k++) ArrInsert(w, k, multi1D(At, multi1D(A, creVector(V, k))));  for (int k = 0; k < (SAMNUM - 1); k++) {  for (int j = k + 1; j < SAMNUM; j++) {  vw[k][j] = ip1D(creVector(w, j), creVector(V, k));  }  }  for (int j = 1; j < SAMNUM; j++) {  for (int k = 0; k < j; k++) {  for (int l = 0; l < SAMNUM; l++) w[l][j] -= vw[k][j] \* V[l][k];  }  }  for (int k = 0; k < SAMNUM; k++) {  lambda[k] = ip1D(creVector(w, k), creVector(V, k));  if (lambda[k] < 0) lambda[k] = abs(lambda[k]);  }  for (int k = 0; k < SAMNUM; k++) for (int j = 0; j < SAMNUM; j++) Residual[j][k] = w[j][k] - lambda[k] \* V[j][k];  for (int k = 0; k < SAMNUM; k++) dist[k] = norm(Residual, k);  dist\_fin = ArrSum(dist);  V = w;  for (int k = 0; k < SAMNUM; k++) {  temp = norm(V, k);  for (int j = 0; j < SAMNUM; j++) V[j][k] /= temp;  }  } while (dist\_fin >(1E-8));  //----------------------------------------------------------------------  // making D with lambda  //---------------------------------------------------------------------  D = initialize(D, 0);  for (int k = 0; k < SAMNUM; k++) D[k][k] = sqrt(lambda[k]);  //----------------------------------------------------------------------  // making U with lambda and V  //----------------------------------------------------------------------  u = initialize(u, 0);  for (int j = 0; j < SAMNUM; j++) {  for (int k = 0; k < W\*H; k++) {  u[k][j] = (1 / D[j][j])\*multi1D(A, creVector(V, j))[k];  }  }  U[d] = u;  } // DIGIT LOOP END  }  ////======================================  ////4. test img load  ////======================================  int true\_index;  int min\_digit;  int temp = 0;  ARRAY<double> res(W\*H);  ARRAY<double> res\_digit(DIGIT);  for (int d = 1; d <= TEST; d++) { // TEST = 600  true\_index = ((d / 10) % 10);  Z = load\_testimg(d);  for (int n = 0; n<DIGIT; n++) {  for (int i = 0; i<W\*H; i++) {  res[i] = Z[i];  for (int j = 0; j<BASISNUM; j++)  for (int k = 0; k<W\*H; k++)  res[i] -= U[n][i][j] \* U[n][k][j] \* Z[k];  }  }  min\_digit = arr\_min(res\_digit);  if (true\_index == (min\_digit + 1)) temp++;  }  printf("\n Caculation Finished. %d images out of %d test img are correctly recognized.", TEST, temp); |