Table of Contents

Distance Metric Learning for Large Margin Nearest Neighbor Classification

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Course Project Final Report (ECE 602: Introduction to Optimization)
% NAME: Bharat Venkitesh, ID: 20676494
% Date: April 24,2017
% University of Waterloo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all
close all
```

Introduction

The k-Nearest Neighbour(kNN) rule classifies each unlabeled example by the majority label of its k-nearest neighbors(based on distance measure) in the training data. Though the idea is simple, it yields competetive results. The performance of the kNN depends on the way the distances are computed between examples in the data. More often when no prior knowledge is available euclidean distance between examples. Euclidean distance may not be the right distance metric and can not capture all the properties in the data. In this project, Mahalanobis distance metric, which is a linear transformation of the input space, that precedes the kNN classification is learnt. The distance metric learning is modeled as a problem in convex optimization.

Problem Formulation

The algorithm is based on the observation that kNN will accurately classifiy if its k nearest neighbour belong to the same class label. The algorithm increases the number of samples from the training data with this property by learning a linear transformation of the input space that precedes kNN classification using Euclidean distances. The linear transfomration is derived by minimizing a loss function, The first term penalizes large distances between samples in the same class desired as k-nearest neighbors and the second term penalizes small distances between samples with different labels. The euclidean distance in transformed space is equivalent to mahalanobis distance in the input space.

Basic Notations

First a few basic formulae and notations are introduced which is used later on in the convex optimization. Let $\vec{x}$ be the d-dimensional samples. Let the linear transformation be $\mathbf{L}$, which is a dxd dimensional matrix. The tranformed or projected samples are then $\hat{x}$= $\mathbf{L}\vec{x}$. The squared euclidean distance in the transformed domain is

$$D_L(\hat{x}_i, \hat{x}_j) = ||\mathbf{L}(\vec{x_i} - \vec{x_j})||_2^2$$

It is common to express squared distance metrics as square matrix as follow

$$\mathbf{M} = \mathbf{L}^T\mathbf{L}$$

Any matrix $\mathbf{M}$ formed from real valued matrix \textbf{L} is guaranteed to be a positive semi-definite matrix. The squared distances interms of \textbf{M} is

$$D_M(\hat{x}_i, \hat{x}_j) = (\vec{x_i} - \vec{x_j})^T M(\vec{x_i} - \vec{x_j})$$

For each sample $\vec{x_i}$, a target nighbour $\vec{x_j}$ is defined as one of the k closest neighbour of the same class. This is denoted as $j \rightarrow i$, $\vec{x_j}$ is a target neighbour to $\vec{x_i}$. For each sample there are k target neighbours(same class). Note that $j \rightarrow i$ does not imply the reverse is true. For kNN classification to succeed, target neighbors of each input $\vec{x_i}$ should be closer than all differently labeled inputs. For each input a perimeter can be imagined around the target neighbours. A diffferently labeled input should not inavde this perimeter. These inputs are defined as impostors. The important goal of the learning is to redice the number of impostors. A large distance is to be maintained between the impostors and the perimeter defined by the target neighbours. By ensuring a margin of safety, the algorithm is robust to noise in the training data. Hence the name Large Margin Nearest Neighbour(LMNN) classification. For an input $\vec{x_i}$ with label y_i and target neighbor $\vec{x_j}$, an impostor is any input $\vec{x_l}$ with a different label y_l such that

$$D_L(\hat{x}_i, \hat{x}_l) <= D_L(\hat{x}_i, \hat{x}_j) + 1$$

Loss Function

The loss function consists of two terms, one which pulls target neighbors closer together, and another which pushes differently labeled examples further apart. The two terms have competing effects, since the first is reduced by shrinking the distances between examples while the second is generally reduced by magnifying them. The first term in the loss function penalizes large distances between each input and its target neighbors. In terms of the linear transformation $\mathbf{L}$ of the input space, the sum of these squared distances is given by

$$\mathbf{E}_{pull} = \Sigma_{j \rightarrow i}||\mathbf{L}(\vec{x_i} - \vec{x_j})||_2^2$$

It does not penalize all the samples with same labels, but only the target neighbours. The second term in the loss function penalizes small distances between differently labeled examples. To simplify notation, we introduce a new indicator variable y_{il} =1 if and only if $y_i = y_l$, and y_{il} =0. The second term is

$$\mathbf{E}_{push} = \Sigma_{i,j\to i}\Sigma_l(1-y_{il})[1+||\mathbf{L}(\vec{x_i}-\vec{x_j})||_2^2 - ||\mathbf{L}(\vec{x_i}-\vec{x_l})||_2^2]_+$$

where the term $[z]_+ = max(z,0)$, denotes the standard hinge loss. The final loss function is

$$\mathbf{E(L)} = (1-u)\mathbf{E}_{pull} + u\mathbf{E}_{push}$$

The above can be rewritten in terms of matrix $\textbf{M} as follows,

$$\mathbf{E(M)} = (1-u)\Sigma_{j\to i}D_M(\dot{x}_i,\dot{x}_j) + u\Sigma_{i,j\to i}\Sigma_l(1-y_{il})[1+D_M(\dot{x}_i,\dot{x}_j) - D_M(\dot{x}_i,$$

With this substitution, the loss function is now expressed over positive semidefinite matrices $\mathbf{M} \succeq 0$, as opposed to real-valued matrices $\mathbf{L}$. The loss function is a piecewise linear, convex function of the elements in the matrix $\mathbf{M}$. The first term(penalizing large distances between target neighbors) is linear in the elements of M, while the second term (penalizing impostors) is derived from the convex hinge loss. The loss function is brought to a standard form, formulated as an semidefinite programming(SDP) by introducing slack variables e_{ijl} for all triplets of target neighbours $j \to i$ and impostots $\vec{x_l}$. The slack variable $e_{ijl} \geq 0$, is used to measure the amount by which the large margin inequality is violated. The SDP is as follows

$$\textbf{Minimize}_{M,e_{ijl}}(1-u)\Sigma_{j\to i}(\vec{x_i}-\vec{x_j})^T M(\vec{x_i}-\vec{x_j}) + u\Sigma_{i,j\to i}\Sigma_l(1-y_{il})e_{ijl}$$

$$\textbf{Subject to}$$

$$(\vec{x_i}-\vec{x_l})^T M(\vec{x_i}-\vec{x_l}) - (\vec{x_i}-\vec{x_j})^T M(\vec{x_i}-\vec{x_j}) \geq 1 - e_{ijl}$$

$$M \succeq 0$$

$$e_{ijl} \geq 0$$

Code Run through

The data used here is 4 dimensional data with 429 training samples. The slack variable used above is a combination of 3 incides leading to a total of 22787 inequalities. For computational sake the data is sampled. The nearest neighbour is set to 3.

```
load bal.mat % xtr yTr xTe yTe
sample=150;
xTr=xTr(:,35:end);
yTr=yTr(35:end);
y=randsample(size(xTr,2),sample); %radom sampling
y=sort(y);
K=3; % nearest neighbour
xTr=xTr(:,y);
yTr=yTr(:,y);
% test_sample=randsample(length(yTe),50);
% test_sample=sort(test_sample);
```

The target neighbours are calculated for each sample and their corresponding distances.

```
[d,n]=size(xTr);
un=unique(yTr);
NN=zeros(K,n);
for c=un
    i=find(yTr==c);
    nn=LSKnn(xTr(:,i),xTr(:,i),2:K+1); % gives the indices of k nearest neighbours
    NN(:,i)=i(nn);
end
Ni=zeros(K,n);
dis=zeros(K,n);
%target neighbours
 for nnid=1:K
        Ni(nnid,:)=sum((xTr-xTr(:,NN(nnid,:))).^2,1)+1; % euclidean distance
        disvector{nnid}=xTr-xTr(:,NN(nnid,:));
 end;
```

The impostors are calcuated for each sample $\vec{x}_i$ and target neighbour $\vec{x}_j$, the samples that violate the perimeter defined by the target neighbours.

```
for c=un

   i=find(yTr==c);
   index=find(yTr~=c);
   X1=xTr(:,i);
   X2=xTr(:,index);
   distmat=zeros(4,length(i),length(index));
   for a=1:length(i)
       for b=1:length(index)

       distmat(:,a,b)=X1(:,a)-X2(:,b);
       end
   end
   distancevector{c}=distmat;
end
for i=1:sample
distmat=[];
index=find(yTr~=yTr(i));
dist=distance(xTr(:,i),xTr(:,index));
id=find(dist<=Ni(3,i)+1); %violating the perimeter
% for j=1:length(index)
%     distmat(:,j)=xTr(:,i)-xTr(:,j);
% end
distancevector2{i}=index(id)';
end
```

The total number of impostors are calculated for each sample

```
total=0;
for i=1:sample
    len(i)=size(distancevector2{i},1);

end
```

```
total=sum(len);
num_imp=zeros(1,length(len)+1);
num_imp(2:end)=len(1:end);
cc=cumsum(num_imp);
```

CVX Package

The cvx is run for variables $\mathbf{M(d,d)}$ and e_{iji}

```
cvx_begin
 variable M(d,d) symmetric
 variable e(total*3) nonnegative
 f_x=0;
 for i=1:sample
     for j=1:K
         f_x=f_x+(xTr(:,i)-xTr(:,NN(j,i)))'*M*(xTr(:,i)-xTr(:,NN(j,i)));
     end
 end
 for i=1:sample

     for l=1:len(i)
         for j=1:K
             f_x=f_x+e((cc(i))+(l-1)*3+j);
         end
     end
 end
 minimize f_x
 subject to
 for i=1:sample

      for l=1:len(i)
         for j=1:K
             (xTr(:,i)-xTr(:,distancevector2{i}(l)))'*M*(xTr(:,i)-xTr(:,distanceve
         end
     end
 end
 M>=0
 e>=0
 cvx_end
```

```
        Calling SDPT3 4.0: 74944 variables, 24988 equality constraints
           For improved efficiency, SDPT3 is solving the dual problem.
        ------------------------------------------------------------

         num. of constraints = 24988
         dim. of linear var  = 74944
        *******************************************************************
           SDPT3: Infeasible path-following algorithms
        *******************************************************************
         version  predcorr  gam  expon  scale_data
            NT      1      0.000   1        0
        it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj    cputime
```

```
 -----------------------------------------------------------------
  0|0.000|0.000|2.8e+04|4.7e+02|9.2e+09|-1.125357e+07  0.000000e+00| 0:0:00
  1|0.986|0.989|3.9e+02|5.5e+00|1.4e+08|-1.617086e+05 -6.722362e+06| 0:0:01
  2|0.311|0.925|2.7e+02|5.5e-01|1.6e+08|-1.126235e+05 -9.279932e+06| 0:0:01
  3|0.576|1.000|1.1e+02|4.5e-02|1.0e+08|-4.973859e+04 -1.068478e+07| 0:0:02
  4|0.713|1.000|3.3e+01|1.4e-02|4.2e+07|-1.588769e+04 -7.951765e+06| 0:0:02
  5|0.882|0.885|3.8e+00|2.8e-03|6.7e+06|-2.809887e+03 -2.362185e+06| 0:0:02
  6|0.569|0.856|1.7e+00|5.1e-04|4.3e+06|-1.335572e+03 -1.247294e+06| 0:0:02
  7|0.541|0.737|7.6e-01|1.4e-04|2.7e+06|-8.329418e+02 -8.292659e+05| 0:0:03
  8|0.231|0.585|5.8e-01|6.1e-05|2.6e+06|-7.346886e+02 -6.895367e+05| 0:0:03
  9|0.243|0.570|4.4e-01|2.6e-05|2.4e+06|-6.621047e+02 -6.279435e+05| 0:0:03
 10|0.344|0.869|2.9e-01|3.5e-06|2.4e+06|-5.839804e+02 -6.580309e+05| 0:0:03
 11|0.361|1.000|1.9e-01|1.3e-09|2.3e+06|-5.337265e+02 -6.883679e+05| 0:0:03
 12|0.728|1.000|5.0e-02|1.4e-10|9.5e+05|-4.675752e+02 -4.280027e+05| 0:0:04
 13|0.946|1.000|2.7e-03|1.0e-02|1.7e+05|-4.480507e+02 -1.358312e+05| 0:0:04
 14|0.971|0.813|7.9e-05|2.4e-03|2.9e+04|-4.548411e+02 -2.883139e+04| 0:0:04
 15|0.538|0.858|3.7e-05|3.6e-04|1.7e+04|-4.779317e+02 -1.403700e+04| 0:0:04
 16|0.403|0.611|2.2e-05|1.5e-04|1.4e+04|-4.979192e+02 -9.981313e+03| 0:0:05
 17|0.357|0.503|1.4e-05|7.8e-05|1.2e+04|-5.198594e+02 -8.134095e+03| 0:0:05
 18|0.190|0.639|1.1e-05|3.1e-05|1.2e+04|-5.328629e+02 -6.697082e+03| 0:0:05
 19|0.503|0.762|5.6e-06|9.6e-06|8.9e+03|-5.662790e+02 -5.371553e+03| 0:0:05
 20|0.116|0.579|5.0e-06|5.2e-06|9.6e+03|-5.759594e+02 -4.940691e+03| 0:0:05
 21|0.157|0.692|4.2e-06|2.6e-06|1.0e+04|-5.881332e+02 -4.796957e+03| 0:0:06
 22|0.237|0.541|3.2e-06|2.0e-06|1.1e+04|-5.947629e+02 -5.106477e+03| 0:0:06
 23|0.159|0.472|2.7e-06|1.7e-06|1.2e+04|-5.947208e+02 -5.387889e+03| 0:0:06
 24|0.433|0.978|1.5e-06|5.8e-07|1.1e+04|-5.998757e+02 -5.424105e+03| 0:0:06
 25|0.708|0.997|4.5e-07|3.1e-07|6.8e+03|-6.251604e+02 -4.704686e+03| 0:0:07
 26|0.260|0.577|3.3e-07|2.2e-07|6.4e+03|-6.343023e+02 -4.070812e+03| 0:0:07
 27|0.252|0.726|2.5e-07|1.3e-07|5.9e+03|-6.534449e+02 -3.331455e+03| 0:0:07
 28|0.172|0.380|2.0e-07|1.3e-07|6.0e+03|-6.736778e+02 -3.345386e+03| 0:0:07
 29|0.345|1.000|1.3e-07|4.0e-08|5.7e+03|-7.020926e+02 -3.067355e+03| 0:0:08
 30|0.172|0.514|1.1e-07|4.6e-08|6.1e+03|-7.091686e+02 -3.196543e+03| 0:0:08
 31|0.547|0.749|5.0e-08|3.6e-08|4.9e+03|-7.176236e+02 -3.265880e+03| 0:0:08
 32|0.339|0.886|3.3e-08|1.4e-08|4.7e+03|-7.352838e+02 -2.949555e+03| 0:0:08
 33|0.441|0.937|1.9e-08|6.1e-09|4.4e+03|-7.630268e+02 -2.909296e+03| 0:0:08
 34|0.475|1.000|9.7e-09|2.8e-09|3.8e+03|-7.795720e+02 -2.618199e+03| 0:0:08
 35|0.248|0.824|7.3e-09|1.9e-09|3.9e+03|-7.983813e+02 -2.481542e+03| 0:0:09
 36|0.133|0.589|6.3e-09|1.9e-09|4.4e+03|-7.994047e+02 -2.554034e+03| 0:0:09
 37|0.134|0.585|5.5e-09|1.7e-09|5.1e+03|-7.957531e+02 -2.772835e+03| 0:0:09
 38|0.742|0.939|1.4e-09|9.3e-10|3.1e+03|-8.061164e+02 -2.709359e+03| 0:0:09
 39|0.481|0.976|7.4e-10|2.3e-10|2.4e+03|-8.573249e+02 -2.143747e+03| 0:0:10
 40|0.045|0.177|7.0e-10|3.0e-10|2.5e+03|-8.613016e+02 -2.147011e+03| 0:0:10
 41|0.291|0.603|5.0e-10|2.2e-10|2.5e+03|-8.842667e+02 -2.116990e+03| 0:0:10
 42|0.094|0.572|4.5e-10|1.7e-10|2.8e+03|-8.855393e+02 -2.068511e+03| 0:0:10
 43|0.357|0.475|2.9e-10|1.6e-10|2.5e+03|-9.016986e+02 -2.118163e+03| 0:0:10
 44|0.534|0.965|1.4e-10|4.9e-11|2.1e+03|-9.256663e+02 -2.009882e+03| 0:0:10
 45|0.272|0.904|9.8e-11|2.5e-11|2.1e+03|-9.447330e+02 -1.902725e+03| 0:0:11
 46|0.303|0.738|6.9e-11|2.1e-11|2.1e+03|-9.544977e+02 -1.884574e+03| 0:0:11
 47|0.334|0.910|4.6e-11|1.2e-11|2.2e+03|-9.530983e+02 -1.919064e+03| 0:0:11
 48|0.344|0.934|3.0e-11|7.6e-12|2.2e+03|-9.613494e+02 -1.884168e+03| 0:0:11
 49|0.624|0.901|1.1e-11|5.2e-12|1.4e+03|-1.005026e+03 -1.752550e+03| 0:0:11
 50|0.068|0.136|1.1e-11|6.2e-12|1.4e+03|-1.003833e+03 -1.741732e+03| 0:0:12
 51|0.208|0.496|8.3e-12|4.7e-12|1.5e+03|-1.013455e+03 -1.725056e+03| 0:0:12
 52|0.482|0.694|4.3e-12|2.7e-12|1.2e+03|-1.033308e+03 -1.658162e+03| 0:0:12
```

```
53|0.160|0.987|3.6e-12|7.8e-13|1.4e+03|-1.042188e+03 -1.601415e+03| 0:0:12
54|0.584|0.665|1.5e-12|1.0e-12|1.1e+03|-1.029864e+03 -1.623859e+03| 0:0:12
55|0.246|1.000|1.1e-12|7.5e-13|1.2e+03|-1.042596e+03 -1.568923e+03| 0:0:13
56|0.424|0.878|6.6e-13|8.4e-13|1.0e+03|-1.067948e+03 -1.550425e+03| 0:0:13
57|0.307|0.807|4.6e-13|9.1e-13|1.0e+03|-1.080329e+03 -1.521279e+03| 0:0:13
58|0.339|0.808|3.1e-13|9.2e-13|9.7e+02|-1.088900e+03 -1.518378e+03| 0:0:13
59|0.185|0.535|2.5e-13|1.2e-12|1.0e+03|-1.087641e+03 -1.524802e+03| 0:0:13
60|0.922|1.000|2.3e-14|7.5e-13|5.0e+02|-1.094396e+03 -1.517631e+03| 0:0:14
61|0.278|0.626|4.0e-14|1.0e-12|4.1e+02|-1.126963e+03 -1.428732e+03| 0:0:14
62|0.162|0.491|3.5e-14|1.3e-12|4.3e+02|-1.139622e+03 -1.412291e+03| 0:0:14
63|0.080|0.365|3.1e-14|1.6e-12|4.6e+02|-1.143048e+03 -1.404214e+03| 0:0:14
64|0.143|0.515|2.2e-14|1.5e-12|4.9e+02|-1.150564e+03 -1.400285e+03| 0:0:14
65|0.616|0.717|3.9e-14|1.2e-12|3.5e+02|-1.173779e+03 -1.390539e+03| 0:0:15
66|0.222|0.476|2.2e-14|1.4e-12|3.5e+02|-1.182389e+03 -1.382105e+03| 0:0:15
67|0.178|0.382|2.1e-14|1.6e-12|3.6e+02|-1.186295e+03 -1.377106e+03| 0:0:15
68|0.287|0.638|3.2e-14|1.3e-12|3.4e+02|-1.195583e+03 -1.367126e+03| 0:0:15
69|0.889|0.929|6.2e-14|8.4e-13|1.7e+02|-1.215859e+03 -1.358746e+03| 0:0:15
70|0.358|0.504|9.3e-14|1.2e-12|1.5e+02|-1.231163e+03 -1.341562e+03| 0:0:16
71|0.254|0.114|2.3e-13|1.8e-12|1.3e+02|-1.236882e+03 -1.339813e+03| 0:0:16
72|0.190|0.475|3.3e-13|1.7e-12|1.3e+02|-1.242043e+03 -1.329797e+03| 0:0:16
73|0.067|0.027|8.8e-13|2.4e-12|1.3e+02|-1.243811e+03 -1.329605e+03| 0:0:16
74|0.656|0.465|3.5e-13|2.0e-12|8.9e+01|-1.256837e+03 -1.324922e+03| 0:0:16
75|0.417|0.420|9.1e-13|1.9e-12|7.4e+01|-1.264144e+03 -1.318924e+03| 0:0:17
76|0.413|0.390|5.6e-13|1.9e-12|6.0e+01|-1.270554e+03 -1.314539e+03| 0:0:17
77|0.514|0.397|1.3e-12|1.9e-12|4.7e+01|-1.275862e+03 -1.311117e+03| 0:0:17
78|0.310|0.537|1.1e-12|1.6e-12|4.2e+01|-1.278801e+03 -1.306989e+03| 0:0:17
79|0.565|0.918|8.5e-13|8.8e-13|2.7e+01|-1.284254e+03 -1.301448e+03| 0:0:17
80|0.465|0.872|5.0e-13|8.6e-13|2.3e+01|-1.286640e+03 -1.300338e+03| 0:0:18
81|0.860|0.927|1.4e-13|8.1e-13|1.0e+01|-1.290747e+03 -1.298911e+03| 0:0:18
82|0.932|0.390|1.6e-13|1.2e-12|5.0e+00|-1.293472e+03 -1.298282e+03| 0:0:18
83|0.930|0.935|4.8e-14|8.3e-13|6.8e-01|-1.295505e+03 -1.296156e+03| 0:0:18
84|0.976|0.981|9.7e-14|7.6e-13|1.5e-02|-1.295861e+03 -1.295876e+03| 0:0:18
85|0.989|0.989|4.3e-12|7.6e-13|2.9e-04|-1.295870e+03 -1.295870e+03| 0:0:18
86|0.989|0.989|5.5e-14|7.6e-13|4.9e-06|-1.295870e+03 -1.295870e+03| 0:0:19
  stop: max(relative gap, infeasibilities) < 1.49e-08
-------------------------------------------------------------------
 number of iterations   = 86
 primal objective value = -1.29587015e+03
 dual   objective value = -1.29587015e+03
 gap := trace(XZ)       = 4.93e-06
 relative gap           = 1.90e-09
 actual relative gap    = 1.85e-09
 rel. primal infeas (scaled problem)   = 5.50e-14
 rel. dual     "        "       "      = 7.56e-13
 rel. primal infeas (unscaled problem) = 0.00e+00
 rel. dual     "        "       "      = 0.00e+00
 norm(X), norm(y), norm(Z) = 2.1e+02, 2.6e+19, 3.6e+19
 norm(A), norm(b), norm(C) = 3.8e+02, 2.8e+02, 1.6e+02
 Total CPU time (secs)  = 18.67
 CPU time per iteration = 0.22
 termination code       =  0
 DIMACS: 9.3e-13  0.0e+00  6.0e-11  0.0e+00  1.8e-09  1.9e-09
-------------------------------------------------------------------
```

```
------------------------------------------------------------
Status: Solved
Optimal value (cvx_optval): +1295.87
```

Results

The projection matrix $\mathbf{L}$ is calculated for the proposed technique and the results are compared with four algorithms. 1)kNN before LMNN, 2) kNN after LMNN, 3) kNN after PCA. We see that kNN performs well after LMNN compared to the other two methods

```
L_lmnn=sqrt(M);
L_knn=[];
[L_pca,score,latent,tsquared,explained]= pca(xTr','Algorithm','eig','Centered',fal
L_pca=L_pca';
Pca_err=knncl(L_pca,xTr,yTr,xTe,yTe,1,'train',0);
knn_err=knncl(L_knn,xTr,yTr,xTe,yTe,1,'train',0);
lmnn_err=knncl(L_lmnn,xTr,yTr,xTe,yTe,1,'train',0);
fprintf('Bal data set\n');
fprintf('\n\nTesting error LMNN: %2.2f%%\n',100*lmnn_err);
fprintf('Testing error kNN: %2.2f%%\n',100*knn_err);
fprintf('Testing error PCA: %2.2f%%\n',100.*Pca_err);
```

```
 Progress:[**********]0.00/16.98 Bal data set


Testing error LMNN: 13.21%
Testing error kNN: 16.98%
Testing error PCA: 16.98%
```

Conclusion and Discussion

A new framework for large margin nearest neighbor (LMNN) classification is introduced. From labeled training examples, a Mahalanobis distance metric is learned for kNN classification. The required optimization is formulated as a semidefinite programming. The framework makes no parametric assumptions about the structure or distribution of the data, though it does not scale well for large data sets. The accuracy of kNN classification is imporved significantly by learning a metric in this way.

References

1)Kilian Q. Weinberger and Lawrence K. Saul. 2009. Distance Metric Learning for Large Margin Nearest Neighbor Classification. J. Mach. Learn. Res. 10 (June 2009), 207-244.

Published with MATLAB® R2013a