Math 521 Classification: Cats and Dogs

Due on Tuesday, May 15, 2018

Kristin Holmbeck and Debbie Tonne

Contents

List of Figures

Introduction

The project we present in this report involves properly classifying two data sets successfully. In this context, the data sets are images of dogs and cats, but the same ideas and algorithms can be successfully applied to other data sets, such as sound waves. Since we are working with images, some preprocessing methods will be explored to add uniformity or variance to the data sets.

Preprocessing

Image *preprocessing* typically involves filtering or computing the Fourier transform of an image prior to analysis. To this end, we will discuss some basics, beginning with filtering.

Filtering

Image filtering uses a *mask* matrix on subsets of an image to perform operations. One filter example is the averaging filter: Given an $m \times n$ mask size, the mask $m_a$will be

$$m_a = \frac{1}{mn} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

where the filtering operation involves an $m \times n$ neighborhood around each pixel of the original image matrix A.

Dimension Reduction

Linear Discriminant Analysis

Background and Motivation

Linear discriminant analysis (LDA) is the method of classifying sets of data. Given a training set whose groups are given *a priori*, we apply LDA to determine which points of a testing (or probe) set can be categorized into each group. The method comes from Fisher's linear discriminant paper [1], but LDA has been developed under some statistical assumptions about the data. Although LDA can be used on n number of groups, we will present the method in the context of a two-class data set. An illustration of the idea behind LDA is shown in Figure 1.

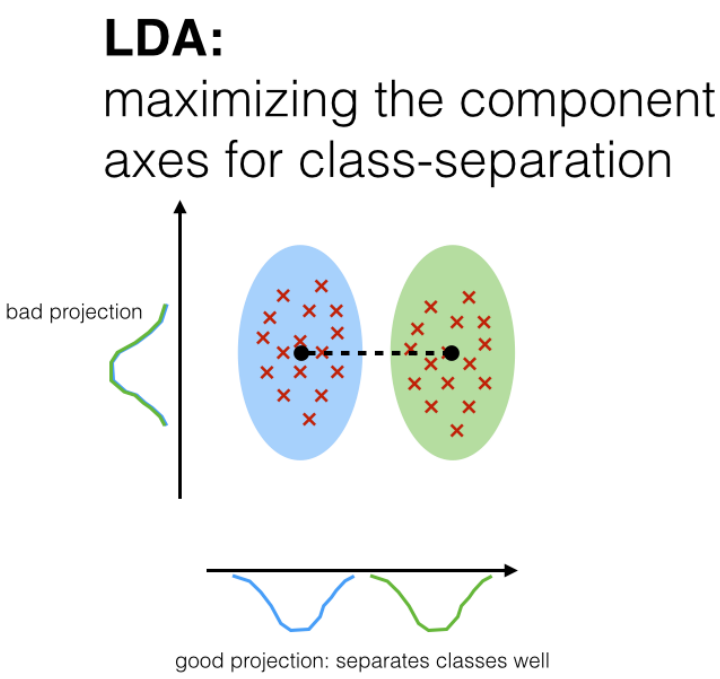

Figure 1: LDA Motivation [4]

Method / Algorithm

As shown in Figure 1, the goal of LDA is to find the optimal projection vector w that maximizes

$$J(w) = \frac{w^T S_B w}{w^T S_W w} \tag{1}$$

where S_B and S_W define the between-class and within-class *scatter* matrices, respectively. S_B and S_W are covariance matrices used to represent the difference between classes.

Given a set of data $X \in \mathbb{R}^{m \times n}$ separated *a priori* into two classes ($X = X_1 \cup X_2$), where $X_1 \in \mathbb{R}^{m \times n_1}$ and $X_2 \in \mathbb{R}^{m \times n_2}$, we define the classwise mean m_i and total mean m:

$$m_i = \frac{1}{n_i} \sum_{y \in X_i} y, \quad m = \frac{1}{n} \sum_{x \in X} x.$$

Then the between-class and within-class scatter matrices

$$S_B = \sum_{i=1}^{2} n_i (m - m_i)(m - m_i)^T, \quad S_W = \sum_{i=1}^{2} \sum_{x \in X_i} (x - m_i)(x - m_i)^T$$

Setting $\nabla J(w) = 0$, we obtain the optimal w^* as the vector that solves the generalized eigenvalue problem

$$S_B w^* = \lambda_{\max} S_W w^* \tag{2}$$

in other words, the w^* associated with the maximum eigenvalue of (2).

Kernel Discriminant Analysis

Before detailing the kernel classification method, we will provide an intuitive example for explaining why one might want to use KDA over LDA. First, consider the toy problem of two concentric circles of data (Figure 2).

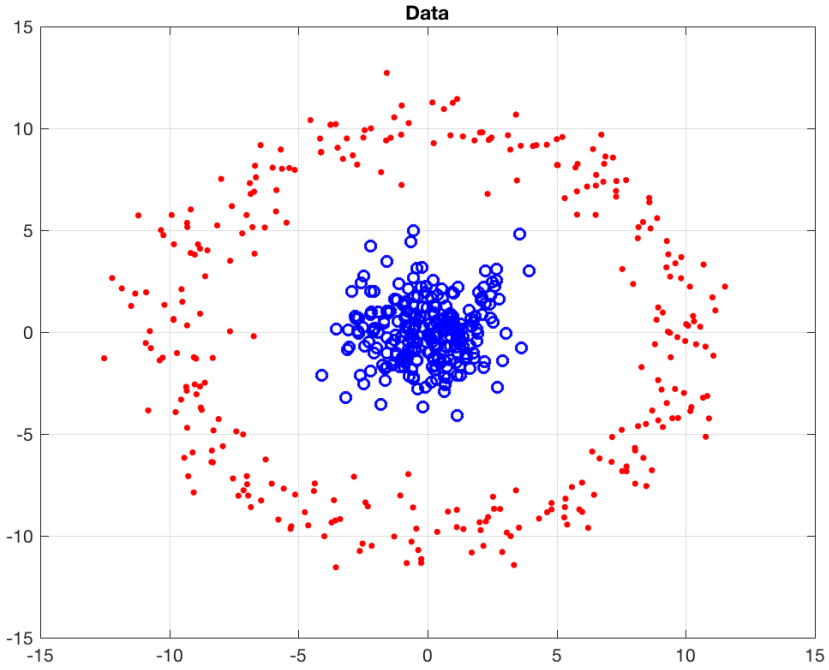

Figure 2: Concentric Data

There is no linear way to separate the data, but the classes clearly have a defined separation. Kernel Discriminant Analysis (KDA) utilizes the ideas of LDA but with the data set X mapped onto a new *feature* space $\mathcal{F}$ where the data has a linear relationship in $\mathcal{F}$. As with LDA, we will present KDA in the context of two classes, although it can be generalized to n classes.

Following the LDA method, suppose our two-class data is given by $X = X_1 \cup X_2$ where $X_1 \in \mathbb{R}^{m \times n_1}$ and $X_2 \in \mathbb{R}^{m \times n_2}$. Now, let Φ be a nonlinear mapping to some feature space $\mathcal{F}$, that is, we take a vector $x \in X$ and map it using $\Phi(x) \in \mathcal{F}$. From the LDA algorithm, we need to maximize

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

However, using the new space, we must maximize

$$J(w) = \frac{w^T S_B^\Phi w}{w^T S_W^\Phi w} \tag{3}$$

where

$$S_B^\Phi = (m_1^\Phi - m_2^\Phi)(m_1^\Phi - m_2^\Phi)^T \qquad \text{and}$$
$$S_W^\Phi = \sum_{i=1}^{2} \sum_{x \in X_i} (\Phi(x) - m_i^\Phi)(\Phi(x) - m_i^\Phi)^T$$

are the between-class and within-class scatter matrices, respectively, in the $\mathcal{F}$ space, and $m_i = \frac{1}{n_i} \sum_{x \in X_i} \Phi(x)$, the mean of the i^{th} class. Furthermore, note that from (3) we are now finding the projection $w \in \mathcal{F}$.

Kernel Trick

Explicitly computing the mappings of a function $\Phi(x)$ onto $\mathcal{F}$ is often inefficient. To that end, we instead compute the inner products between the images (images in the linear algebra sense) of all pairs of data in the *implicit* feature space. This is referred to as the kernel trick or kernel method.

A feature map is a map $\Phi : X \to \mathcal{F}$, where $\mathcal{F}$ is what we call the feature space. Every feature map defines a kernel

$$\kappa(x, y) = \langle \Phi(x), \Phi(y) \rangle$$

where κ is clearly symmetric and positive-definite. In the statistical context, the kernel is used as a measure of similarity. In particular, the kernel function κ defines the distribution of similarities of points around a given point x, $\kappa(x, y)$ denotes the similarity of point x with another given point y.

Choosing a kernel

The most commonly used kernels are polynomial kernels $\kappa(x, y) = \langle x, y \rangle^d$ (for some $d > 0$) and the Gaussian radial basis kernel $\kappa(x, y) = \exp(-||x - y||^2/c)$ (for some $c > 0$).

Since the kernel can be thought of as a similarity measure, we want to make sure to choose a kernel where points in the same class have a high kernel value (very similar) and points in different classes have a low kernel value (dissimilar).

KDA using the Kernel Trick

Noting that $w \in \mathcal{F}$, and using the theory of reproducing kernels [3], w lies in the span of all training samples in $\mathcal{F}$. Hence,

$$w = \sum_{i=1}^{n} \alpha_i \Phi\left(x_i\right)$$

Multiplying w^T by the mean,

$$\begin{aligned} w^T m_i^\Phi &= \frac{1}{n_i} \sum_{j=1}^{n} \alpha_j \Phi^T \left(x^{(j)}\right) \sum_{y \in X_i} \Phi\left(y\right) \\ &= \frac{1}{n_i} \sum_{j=1}^{n} \sum_{y \in X_i} \alpha_j \Phi^T \left(x^{(j)}\right) \Phi\left(y\right) \\ &= \frac{1}{n_i} \sum_{j=1}^{n} \sum_{y \in X_i} \alpha_j \kappa \left(x^{(j)}, y\right) \\ &= \alpha^T M_i \end{aligned}$$

where $\left(M_i\right)_j := \frac{1}{n_i} \sum_{y \in X_i} \kappa\left(x^{(j)}, y\right)$. The number of (3) can then be rewritten as

$$w^T S_B^\Phi w = w^T (m_1^\Phi - m_2^\Phi)(m_1^\Phi - m_2^\Phi)^T w = \alpha^T M \alpha$$

where $M = (M_1 - M_2)(M_1 - M_2)^T$.

Following the same logic, the denominator $w^T S_W^\Phi w$ can be written as $\alpha^T N \alpha$ where

$$N := \sum_{j=1}^{2} K_j (I - \mathbf{1}_{n_j}) K_j^T$$
$$(K_j)_{nm} := \kappa\left(x_n, x_m^{(j)}\right), K_j \in \mathbb{R}^{n \times n_j}$$
$$\mathbf{1}_{lj} := \frac{1}{n_j} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{n_j \times n_j}$$

Then, (3) can be rewritten as (4) and a *new* projection of x onto w is given by (5).

$$J(\alpha) = \frac{\alpha^T M \alpha}{\alpha^T N \alpha} \tag{4}$$

$$w^T \Phi(x) = \sum_{i=1}^{n} \alpha_i \kappa(x_i, x) \tag{5}$$

Just as in LDA, (4) can be optimized via the generalized eigenvalue problem

$$M\alpha^* = \lambda_{\max} N \alpha^*$$

Example on concentric circle data

Let us revisit the concentric data problem present in Figure 2, and compare the classification of LDA to KDA. As discussed, the data is not linearly separable and so we expect LDA to be a poor classifier. For LDA, we obtain the failed classification (Figure 3) and the projection vector along with the data in Figure 4.

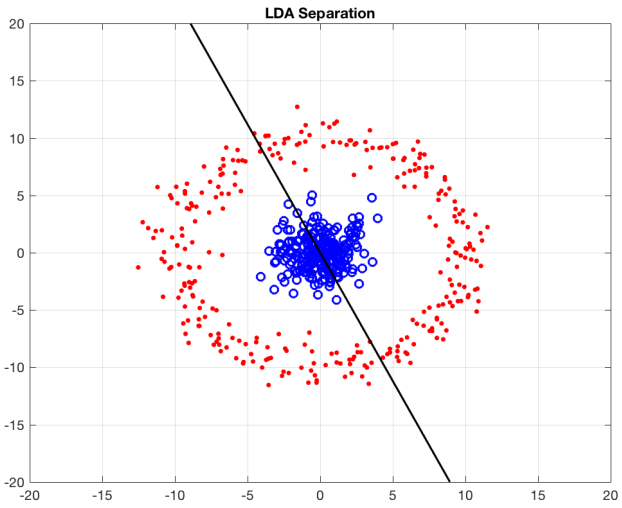

Figure 3: Figure 2 data with LDA projection vector

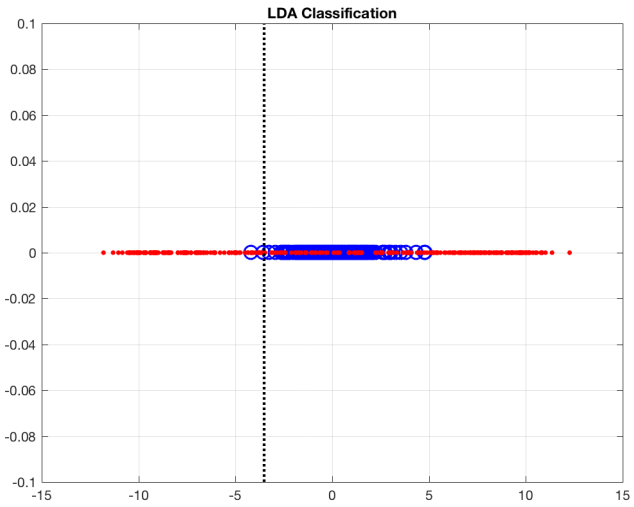

Figure 4: Figure 2 LDA classification

Because of the kernel trick, we cannot plot the projection "vector" directly. Nonetheless, the successful classification is shown in Figure 5.

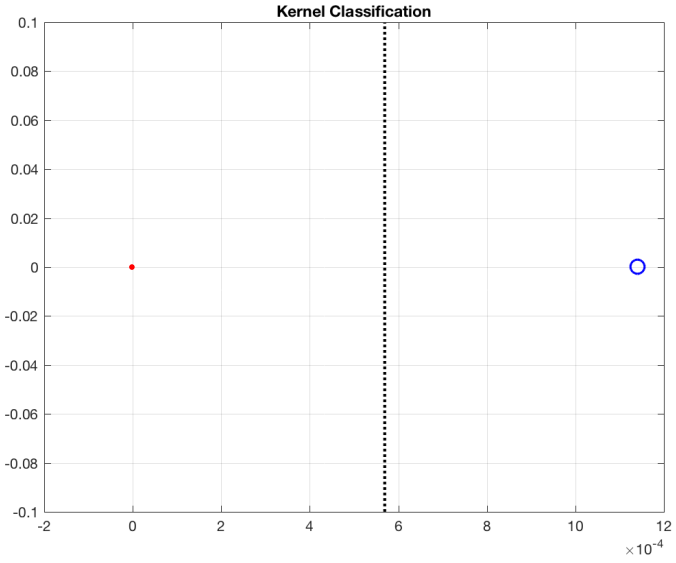

Figure 5: Figure 2 KDA classification

Example on parabolically-separable data

Another data set is shown in Figure 6. Again, this is easily separable, but it's clear that the separation is nonlinear. Again, for LDA, we obtain the failed classification (Figure 7) and the successful classification in Figure 9.

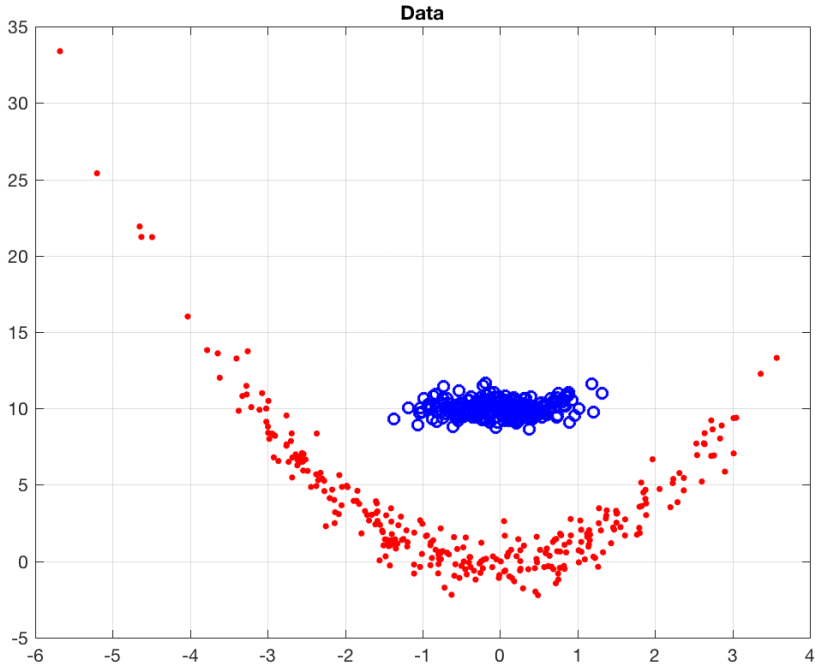

Figure 6: Parabolically-separable data

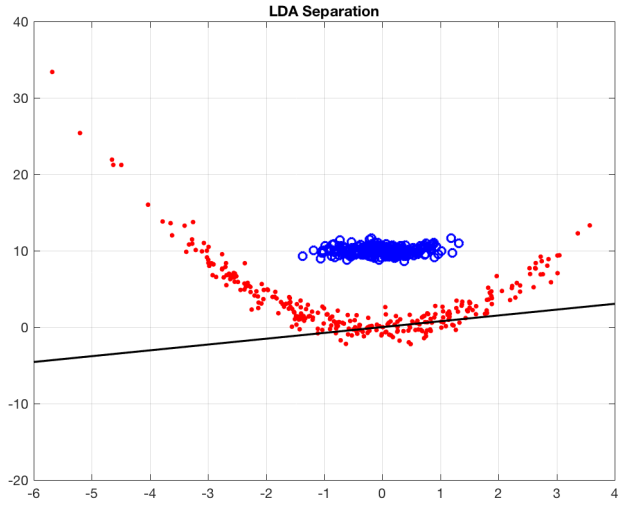

Figure 7: Figure 6 data with LDA projection vector

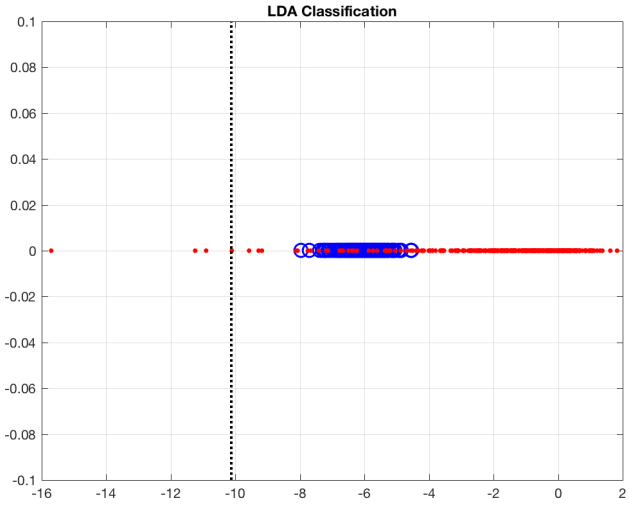

Figure 8: Figure 6 LDA classification

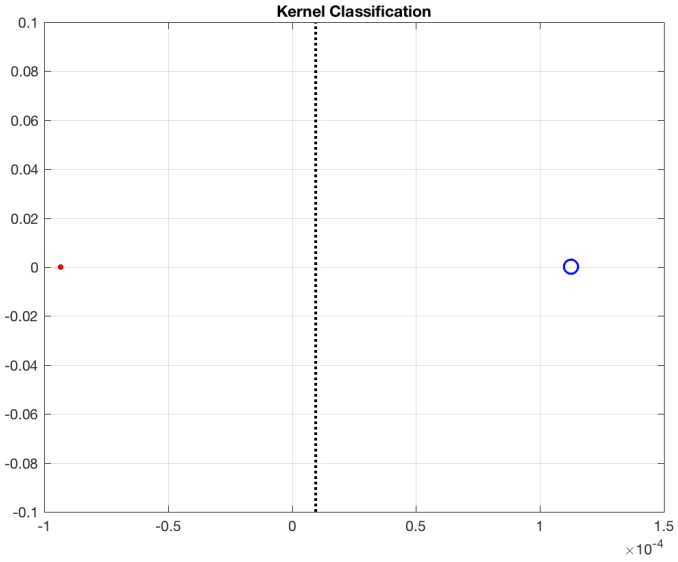

Figure 9: Figure 6 KDA classification

Results

Dogs and Cats

Classification Types

References

[1] R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179188, 1936.

[2] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K. Muller. Fisher discriminant analysis with kernels. In *Proc. IEEE Neural Networks for Signal Processing Workshop*, pages 4148. IEEE Computer Society Press, 1999.

[3] Aronszajn, N. "Theory of Reproducing Kernels." *Transactions of the American Mathematical Society* 68, no. 3 (1950): 337-404. doi:10.2307/1990404.

[4] Raschka, Sebastian. Cropped illustration of PCA vs LDA. *Sebastian Raschka*. 3 August 2014. `http://sebastianraschka.com/Articles/2014_python_lda.html`.

Code

Linear Discriminant Analysis

```
function [Xproj, Xproj0, alpha, w] = LDA(DATA, classes, TestData, opts)
% LDA
% Linear Discriminant Analysis
%
% Syntax:
%   w = LDA(X, class)
%
% where X is the entire data set and class indicates the class of each
% column of X. The function returns the projection vector, w, a vector
% whose length equals the number of columns of X, and the projected data,
% Xproj.
%

% Author: Kristin Holmbeck

if nargin == 3
    opts.doPCA = false;
    opts.tol = 0.95;
end

allClasses  = unique(classes);
nClasses    = length( allClasses );
ni          = zeros(nClasses,1);
for ii = 1:nClasses
    ni(ii) = sum(classes == allClasses(ii));
end

% approximate the data with lower dimensions (PCA)
% and transform into a new space (X)
if opts.doPCA
    [U,S,V] = svd(DATA,0);
    % [U,S,V] = svd(bsxfun(@minus, DATA, sum(DATA,2))/size(DATA,2), 0);
    E       = cumulative_energy(diag(S), rank(DATA));
    k       = find(E>opts.tol, 1);          % 99% rank approximation
    X       = S(1:k,1:k)*V(:,1:k)';     % use this new space
else
    X = DATA;
    k = size(DATA,1);
end

```

```
nDat    = size(X,2);
m       = sum(X,2) / nDat;          % total mean

SB = zeros(k);
SW = zeros(k);
for ii = 1:nClasses
    thisClass   = ( classes==allClasses(ii) );
    classData   = X(:,thisClass);
    mu          = sum(classData,2) / ni(ii);   % current classwise mean
    SB          = SB + ni(ii)*(m-mu)*(m-mu)';

    for jj = 1:size(classData,2)
        SW = SW + (classData(:,jj) - mu)*(classData(:,jj) - mu)';
    end
end

% A       = pinv(SW)*SB;
% [V,D]   = eig(A);
% D       = abs(diag(D));
% [D,ndx] = max(D);

[U1,V1,X1,C1,S1] = gsvd(SB, SW, 0);
[v,ndx] = max(diag(S1).*diag(C1));
w       = V1(:,ndx);

Xproj0  = w'*X;

if opts.doPCA
    TestData = U(:,1:k)'*TestData;
end

Xproj   = w'*TestData;

alpha = mean(Xproj0);

% alpha   = w'*m;          % alpha is only valid for nClasses=2

if nClasses == 2
    x1 = sort(Xproj0(classes == allClasses(1)));
    x2 = sort(Xproj0(classes == allClasses(2)));
    alpha = (max(x1) + min(x2))/2;
end

end
```

Kernel Discriminant Analysis

```
function [yproj, yproj0, alpha] = KLDA(DATA, classes, TestData, options)
% KLDA
% Kernel Linear Discriminant Analysis
%
% Syntax:
%   yproj = KLDA(TrainData, TrainClassLabel, TestData)
%   yproj = KLDA(TrainData, TrainClassLabel, TestData, options)
%
% where TrainData is the entire data set and TrainClassLabel indicates the
% class of each column of TrainData.
%
% The function returns the vector yproj, which is the projection of
% TestData onto the space.
%
% The options input is a structure that controls whether or not we perform
```

```
% PCA.
%

% Author: Kristin Holmbeck

if nargin == 3
    options.doPCA = 1;
    options.tol = 0.999;
    options.kernType = 0;
end

allClasses  = unique(classes);
nClasses    = length( allClasses );
ni          = zeros(nClasses,1);
for ii = 1:nClasses
    ni(ii) = sum(classes == allClasses(ii));
end

% meanDat = sum(DATA,2) / size(DATA,2);
% DATA = bsxfun(@minus, DATA, meanDat);
% TestData = bsxfun(@minus, TestData, meanDat);

if options.doPCA
    % approximate the data with lower dimensions (PCA)
    % and transform into a new space (X)
    X       = DATA;
    [U,S,V] = svd(DATA,0);
    % [U,S,V] = svd(bsxfun(@minus, DATA, sum(DATA,2))/size(DATA,2), 0);
    E       = cumulative_energy(diag(S), rank(DATA));
    k       = find(E>options.tol, 1);
    DATA    = S(1:k,1:k)*V(:,1:k)';     % use this new space
end

npts = size(DATA,2);

M = zeros(npts, 2);
N = zeros(npts, npts);
for ii = 1:nClasses
    thisClass   = ( classes==allClasses(ii) );
    classData   = DATA(:,thisClass);

    Ki = zeros(npts, ni(ii));
    for jj = 1:npts
        for kk = 1:ni(ii)
            Ki(jj,kk) = kernel(DATA(:,jj), classData(:,kk), options.kernType);
        end
    end
    I = eye(ni(ii));
    L = ones(ni(ii)) * (1/ni(ii));
    N = N + Ki*(I - L)*Ki';

    for jj = 1:ni(ii)
        for kk = 1:npts;
%             M(:,ii) = M(:,ii) + kernel(DATA(:,kk), classData(:,jj), options.kernType);
            M(kk,ii) = M(kk,ii) + kernel(DATA(:,kk), classData(:,jj), options.kernType);
        end
    end
    M(:,ii) = M(:,ii) / ni(ii);

%     M(:,ii) = sum(Ki,2) / ni(ii);
end
% N = N + 1e-3*eye(npts);  % regularization term
```

```
78
79  % ASSUME WE ONLY HAVE 2 CLASSES
80  M1 = M(:,1);
81  M2 = M(:,2);
82  M  = (M1-M2)*(M1-M2)';
83
84  % [V,D]   = eig(M,N);
85  % D       = abs(diag(D));
86  % [D,ndx] = max(D);
87  % alpha   = V(:,ndx);
88
89  [U1,V1,X1,C1,S1] = gsvd(M,N,0);
90  [v,ndx] = max(diag(C1).*diag(S1));
91  alpha = V1(:,ndx);
92
93  % project testing data
94  if options.doPCA
95      TestData = U(:,1:k)'*TestData;
96  end
97
98  % project training data
99  yproj0 = zeros(size(DATA,2), 1);
100 for jj = 1:size(DATA,2)
101     for ii = 1:size(DATA,2)
102         yproj0(jj) = yproj0(jj) + alpha(ii)*kernel(DATA(:,ii), DATA(:,jj), options.
                kernType);
103     end
104 %     yproj0(jj) = kernel(DATA, DATA(:,jj), options.kernType) * alpha;
105 end
106
107 yproj = zeros(size(TestData,2), 1);
108 for jj = 1:size(TestData,2)
109     for ii = 1:size(DATA,2)
110         yproj(jj) = yproj(jj) + alpha(ii)*kernel(DATA(:,ii), TestData(:,jj), options.
                kernType);
111     end
112 %     yproj(jj) = kernel(DATA, TestData(:,jj), options.kernType) * alpha;
113 end
114
115 end
116
117 function k = kernel(x,y, kernType)
118
119 switch kernType
120     case 0
121         k = y'*x;
122     case 1
123         k = exp(-norm(x-y).^2 / 10.5); return;
124         tmp = bsxfun(@minus, x, y);
125         k = exp( -(tmp(1,:).^2 + tmp(2,:).^2) / 0.5 );
126     otherwise
127         error('Invalid kernel type');
128 end
129
130 end
```

KDA testing code for data in Figures 2 and 6

```
1  clear;
2
3  rng(1);
4  n1 = 250;
```

```
n2 = 300;

folder = 'tex/kda_test/';

% Note: use the Gaussian kernel instead
options.doPCA    = 0;
options.kernType = 1;   % gaussian


SAVEFIGS = 1;

if SAVEFIGS
    newfig = figure;
    fullPos = get(gca, 'Position');
end

switch 1
    case 0      % parabola data
        fname = 'parab';
        x = 0.5*randn(1,n1);
        y = 0.5*randn(1,n1) + 10;

        x1 = linspace(-3,2,n2) + randn(1,n2);
        y1 = x1.^2 + randn(1,n2);
    case 1      % concentric circles data
        fname = 'circles';

        x = 1.5*randn(1,n1);
        y = 1.5*randn(1,n1);

        theta = linspace(0, 2*pi, n2);
        x1 = 10*cos(theta) + randn(1,n2);
        y1 = 10*sin(theta) + randn(1,n2);
    otherwise
        error('Invalid');
end

c1_style = 'ob';
c2_style = '.r';

figure;
axh = subplot(321);
ax1 = plot(x,y, c1_style); hold on;
ax2 = plot(x1,y1, c2_style, 'MarkerSize',10); hold off;
% axis tight;
XL = xlim; YL = ylim;

title 'Data';
h = legend([ax1(1),ax2(1)], 'Class 1', 'Class 2', 'Location', 'EastOutside');
pos = get(h, 'Position');
set(h,'Position', [pos(1)+0.2, pos(2), pos(3)+0.05, pos(4)+0.05]);

if SAVEFIGS
    clf(newfig); h = copyobj(axh, newfig); set(h, 'Position', fullPos);
    saveas(newfig, [folder, fname, '_data.png']);
end

DATA = [x, x1; y, y1];
classes = [zeros(1,n1), ones(1,n2)];

[yproj0] = KLDA(DATA, classes, DATA, options);

```

```
67  c1 = (classes==0); c2 = (classes==1);
68
69  if max(yproj0(c1)) < max(yproj0(c2))
70      alpha = max(yproj0(c1)) + min(yproj0(c2));
71  else
72      alpha = min(yproj0(c1)) + max(yproj0(c2));
73  end
74  alpha = alpha / 2;
75
76  axh = subplot(323);
77  plot(x,y, c1_style);
78  hold on; plot(x1,y1, c2_style, 'MarkerSize',10); xlim(XL); ylim(YL);
79  hold off; title 'Kernel Separation';
80
81  if SAVEFIGS
82      clf(newfig); h = copyobj(axh, newfig); set(h, 'Position', fullPos);
83      saveas(newfig, [folder, 'KDA_', fname, '.png']);
84  end
85
86  axh = subplot(324);
87  plot(yproj0(c1),0, c1_style, 'MarkerSize',10); hold on;
88  plot(yproj0(c2),0, c2_style, 'MarkerSize',10);
89  plot(alpha*[1,1], 0.1*[-1,1], ':k', 'LineWidth',2);
90
91  % xlim(1.1*[1,1].*[min(yproj0)-1e-5, max(yproj0)])
92
93  hold off;
94  title 'Kernel Classification';
95
96  if SAVEFIGS
97      clf(newfig); h = copyobj(axh, newfig); set(h, 'Position', fullPos);
98      saveas(newfig, [folder, 'KDA_proj_', fname, '.png']);
99  end
100
101 [yproj, yproj0, alpha, w] = LDA(DATA, classes, DATA, options);
102
103 axh = subplot(325);
104 plot(x,y, c1_style);
105 hold on; plot(x1,y1, c2_style, 'MarkerSize',10);
106 slope = w(2)/w(1); xtmp = linspace(XL(1), XL(2), 100);
107 plot(xtmp, slope*xtmp, 'k');      % line that spans projection vector w
108 xlim(XL); ylim(YL);
109 hold off;
110 title 'LDA Separation'; figure(gcf);
111
112 if SAVEFIGS
113     clf(newfig); h = copyobj(axh, newfig); set(h, 'Position', fullPos);
114     saveas(newfig, [folder, 'LDA_', fname, '.png']);
115 end
116
117 axh = subplot(326);
118 plot(yproj0(c1),0, c1_style, 'MarkerSize',10); hold on;
119 plot(yproj0(c2),0, c2_style, 'MarkerSize',10);
120 plot(alpha*[1,1], 0.1*[-1,1], ':k', 'LineWidth',2);
121 hold off;
122 title 'LDA Classification';
123
124 if SAVEFIGS
125     clf(newfig); h = copyobj(axh, newfig); set(h, 'Position', fullPos);
126     saveas(newfig, [folder, 'LDA_proj_', fname, '.png']);
127 end
```