SMAI ASSIGNMENT 4

Romil Aggarwal
201330112

QUES1)

To extend LDA to non-linear mappings, the data can be mapped to a new feature space, via some function ϕ. In this new feature space, the function that needs to be maximized is

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{S}_B^{\phi}\mathbf{w}}{\mathbf{w}^{\mathrm{T}}\mathbf{S}_W^{\phi}\mathbf{w}},$$

where

$$\mathbf{S}_B^{\phi} = (\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi})(\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi})^{\mathrm{T}}$$

$$\mathbf{S}_W^{\phi} = \sum_{i=1,2}\sum_{n=1}^{l_i}(\phi(\mathbf{x}_n^i) - \mathbf{m}_i^{\phi})(\phi(\mathbf{x}_n^i) - \mathbf{m}_i^{\phi})^{\mathrm{T}},$$

and

$$\mathbf{m}_i^{\phi} = \frac{1}{l_i}\sum_{j=1}^{l_i}\phi(\mathbf{x}_j^i).$$

Further, note that $\mathbf{w} \in F$. Explicitly computing the mappings $\phi(\mathbf{x}_i)$ and then performing LDA can be computationally expensive, and in many cases intractable. For example, F may be infinitely dimensional. Thus, rather than explicitly mapping the data to F, the data can be implicitly embedded by rewriting the algorithm in terms of dot products and using the kernel trick in which the dot product in the new feature space is replaced by a kernel function $k(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y})$.

LDA can be reformulated in terms of dot products by first noting that $\mathbf{w}$ will have an expansion of the form

$$\mathbf{w} = \sum_{i=1}^{l}\alpha_i\phi(\mathbf{x}_i).$$

Then note that

$$\mathbf{w}^{\mathrm{T}}\mathbf{m}_i^{\phi} = \frac{1}{l_i}\sum_{j=1}^{l}\sum_{k=1}^{l_i}\alpha_j k(\mathbf{x}_j, \mathbf{x}_k^i) = \alpha^{\mathrm{T}}\mathbf{M}_i,$$

where

$$(\mathbf{M}_i)_j = \frac{1}{l_i}\sum_{k=1}^{l_i} k(\mathbf{x}_j, \mathbf{x}_k^i).$$

The numerator of $J(\mathbf{w})$ can then be written as:

$$\begin{aligned}\mathbf{w}^{\mathrm{T}}\mathbf{S}_B^{\phi}\mathbf{w} &= \mathbf{w}^{\mathrm{T}}(\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi})(\mathbf{m}_2^{\phi} - \mathbf{m}_1^{\phi})^{\mathrm{T}}\mathbf{w} \\ &= \alpha^{\mathrm{T}}\mathbf{M}\alpha,\end{aligned}$$

where $\mathbf{M} = (\mathbf{M}_2 - \mathbf{M}_1)(\mathbf{M}_2 - \mathbf{M}_1)^{\mathrm{T}}$. Similarly, the denominator can be written as
$\mathbf{w}^{\mathrm{T}}\mathbf{S}_W^{\phi}\mathbf{w} = \alpha^{\mathrm{T}}\mathbf{N}\alpha,$
where

$$\mathbf{N} = \sum_{j=1,2}\mathbf{K}_j(\mathbf{I} - \mathbf{1}_{l_j})\mathbf{K}_j^{\mathrm{T}},$$

with the $n^{\mathrm{th}}, m^{\mathrm{th}}$ component of $\mathbf{K}_j$ defined as $k(\mathbf{x}_n, \mathbf{x}_m^j)$, $\mathbf{I}$ is the identity matrix, and $\mathbf{1}_{l_j}$ the matrix with all entries equal to $1/l_j$. This identity can be derived by starting out with the expression for $\mathbf{w}^{\mathrm{T}}\mathbf{S}_W^{\phi}\mathbf{w}$ and using the expansion of $\mathbf{w}$ and the definitions of $\mathbf{S}_W^{\phi}$ and $\mathbf{m}_i^{\phi}$

$$\begin{aligned}\mathbf{w}^{\mathrm{T}}\mathbf{S}_W^{\phi}\mathbf{w} &= \left(\sum_{i=1}^{l}\alpha_i\phi^{\mathrm{T}}(\mathbf{x}_i)\right)\left(\sum_{j=1,2}\sum_{n=1}^{l_j}(\phi(\mathbf{x}_n^j) - \mathbf{m}_j^{\phi})(\phi(\mathbf{x}_n^j) - \mathbf{m}_j^{\phi})^{\mathrm{T}}\right)\left(\sum_{k=1}^{l}\alpha_k\phi(\mathbf{x}_k)\right) \\
&= \sum_{j=1,2}\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\alpha_i\phi^{\mathrm{T}}(\mathbf{x}_i)(\phi(\mathbf{x}_n^j) - \mathbf{m}_j^{\phi})(\phi(\mathbf{x}_n^j) - \mathbf{m}_j^{\phi})^{\mathrm{T}}\alpha_k\phi(\mathbf{x}_k) \\
&= \sum_{j=1,2}\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i k(\mathbf{x}_i, \mathbf{x}_n^j) - \frac{1}{l_j}\sum_{p=1}^{l_j}\alpha_i k(\mathbf{x}_i, \mathbf{x}_p^j)\right)\left(\alpha_k k(\mathbf{x}_k, \mathbf{x}_n^j) - \frac{1}{l_j}\sum_{q=1}^{l_j}\alpha_k k(\mathbf{x}_k, \mathbf{x}_q^j)\right) \\
&= \sum_{j=1,2}\left(\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i\alpha_k k(\mathbf{x}_i, \mathbf{x}_n^j)k(\mathbf{x}_k, \mathbf{x}_n^j)\right.\right. \\
&\quad \left.\left. - \frac{2\alpha_i\alpha_k}{l_j}\sum_{p=1}^{l_j}k(\mathbf{x}_i, \mathbf{x}_n^j)k(\mathbf{x}_k, \mathbf{x}_p^j) + \frac{\alpha_i\alpha_k}{l_j^2}\sum_{p=1}^{l_j}\sum_{q=1}^{l_j}k(\mathbf{x}_i, \mathbf{x}_p^j)k(\mathbf{x}_k, \mathbf{x}_q^j)\right)\right) \\
&= \sum_{j=1,2}\left(\sum_{i=1}^{l}\sum_{n=1}^{l_j}\sum_{k=1}^{l}\left(\alpha_i\alpha_k k(\mathbf{x}_i, \mathbf{x}_n^j)k(\mathbf{x}_k, \mathbf{x}_n^j) - \frac{\alpha_i\alpha_k}{l_j}\sum_{p=1}^{l_j}k(\mathbf{x}_i, \mathbf{x}_n^j)k(\mathbf{x}_k, \mathbf{x}_p^j)\right)\right) \\
&= \sum_{j=1,2}\alpha^{\mathrm{T}}\mathbf{K}_j\mathbf{K}_j^{\mathrm{T}}\alpha - \alpha^{\mathrm{T}}\mathbf{K}_j\mathbf{1}_{l_j}\mathbf{K}_j^{\mathrm{T}}\alpha \\
&= \alpha^{\mathrm{T}}\mathbf{N}\alpha.\end{aligned}$$

With these equations for the numerator and denominator of $J(\mathbf{w})$, the equation for J can be rewritten as

$$J(\alpha) = \frac{\alpha^{\mathrm{T}}\mathbf{M}\alpha}{\alpha^{\mathrm{T}}\mathbf{N}\alpha}.$$

Then, differentiating and setting equal to zero gives

$$(\alpha^{\mathrm{T}}\mathbf{M}\alpha)\mathbf{N}\alpha = (\alpha^{\mathrm{T}}\mathbf{N}\alpha)\mathbf{M}\alpha.$$

Since only the direction of $\mathbf{w}$, and hence the direction of α, matters, the above can be solved for α as

$$\alpha = \mathbf{N}^{-1}(\mathbf{M}_2 - \mathbf{M}_1).$$

Note that in practice, $\mathbf{N}$ is usually singular and so a multiple of the identity is added to it

$$\mathbf{N}_\epsilon = \mathbf{N} + \epsilon\mathbf{I}.$$

Given the solution for α, the projection of a new data point is given by

$$y(\mathbf{x}) = (\mathbf{w} \cdot \phi(\mathbf{x})) = \sum_{i=1}^{l} \alpha_i k(\mathbf{x}_i, \mathbf{x}).$$

QUES2)

Code(PCA)

```
clear all; clc;

load arcene_train.data
a = arcene_train;
load arcene_train.labels
load arcene_valid.data
train_labels = arcene_train;
c = arcene_valid;
load arcene_valid.labels
valid_labels = arcene_valid;

sigma = 10000;
d=[a;c];
%-----------------------------------------------------------
%kernel on train+valid data
K1 = zeros(size(d,1),size(d,1));
for i=1:size(d,1)
        for j=1:size(d,1)
        K1(i,j) = exp(-norm(d(i,:)-d(j,:))^2/sigma^2);
        end
end
temp1 = ones(size(d,1),size(d,1))/(size(d,1));
Ker1 = K1 - temp1*K1 - K1*temp1 + temp1*K1*temp1;
[eigenvec eigenval] = eig(Ker1);
eigenval = diag(eigenval);
for i=1:size(d,1)
        eigenvec(:,i) = eigenvec(:,i)/eigenval(i);
end
%---------------------------------------------------------



accuracies1 = zeros(1,100);
accuracies2 = zeros(1,100);

for t=1:100
        v = eigenvec(:,1:t);
        train_set = Ker1(1:100,:)*v;
        validate_set = Ker1(101:200,:)*v;
        %linear svm
        trainmodel = svmtrain(train_set,train_labels);

accuracies1(t)=100*(size(find(svmclassify(trainmodel,validate_set)==valid_labels),1)/size(train_
set,1));
        %rbf svm
        trainmodel = svmtrain(train_set,train_labels,'kernel_function','rbf','rbf_sigma',5);

accuracies2(t)=100*(size(find(svmclassify(trainmodel,validate_set)==valid_labels),1)/size(train_
set,1));
end
figure
plot(accuracies1)
figure
plot(accuracies2)
```

<u>Code(LDA)</u>

```
clear all; clc
load arcene_train.data
a = arcene_train;
load arcene_train.labels
load arcene_valid.data
b = arcene_train;
c = arcene_valid;
load arcene_valid.labels
d = arcene_valid;


%------------------------------------------------------------
% Kernel on train data
sigma1 = 10000;
K11 = zeros(size(a,1),size(a,1));
for i=1:size(a,1)
        for j=1:size(a,1)
        K11(i,j) = exp(-norm(a(i,:)-a(j,:))^2/sigma1^2);
        end
end
temp1 = ones(size(a,1),size(a,1))/(size(a,1));
Ker1 = K11 - temp1*K11 - K11*temp1 + temp1*K11*temp1;
%------------------------------------------------------------



%------------------------------------------------------------
%Kernel on valid data
K12 = zeros(size(c,1),size(c,1));
for i=1:size(c,1)
        for j=1:size(c,1)
        K12(i,j) = exp(-norm(c(i,:)-c(j,:))^2/sigma1^2);
        end
end
temp2 = ones(size(c,1),size(c,1))/(size(c,1));
Ker2 = K12 - temp2*K12 - K12*temp2 + temp2*K12*temp2;
%------------------------------------------------------------


m1indices = find(b==1);
m2indices = find(b==-1);
M1 = mean(Ker1(m1indices,:));   % mean for class label 1
M2 = mean(Ker1(m2indices,:));   % mean for class label 2

bm1indices = find(d==1);
bm2indices = find(d==-1);
bM1 = mean(Ker2(bm1indices,:));   % mean for class label 1
```

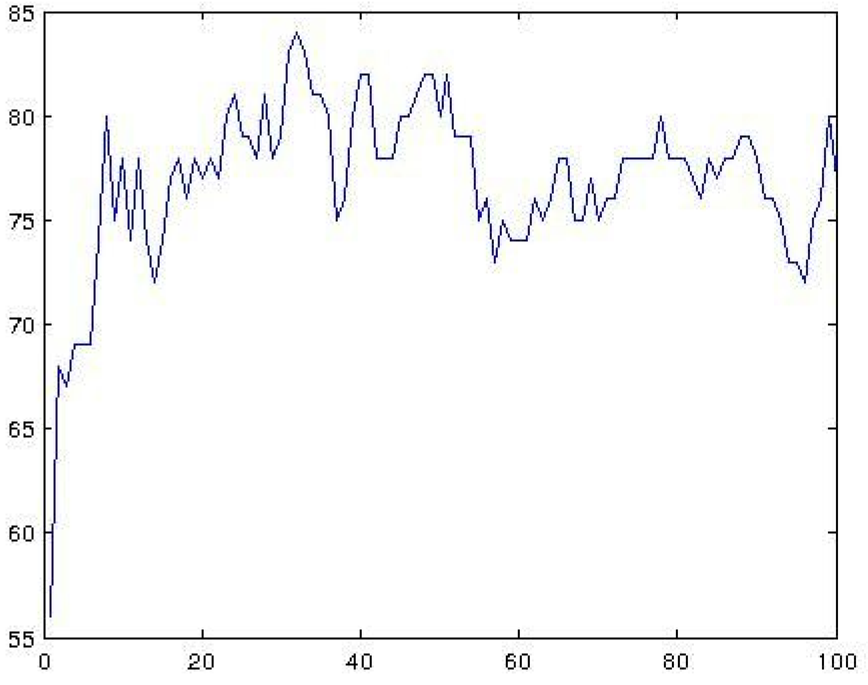
```
bM2 = mean(Ker2(bm2indices,:));   % mean for class label 2

N = Ker1(m1indices,:)'*(eye(size(m1indices,1))-(1/size(m1indices,1)))*Ker1(m1indices,:) +
Ker1(m2indices,:)'*(eye(size(m2indices,1))-(1/size(m2indices,1)))*Ker1(m2indices,:);
N1 = Ker2(bm1indices,:)'*(eye(size(bm1indices,1))-(1/size(bm1indices,1)))*Ker2(bm1indices,:) +
Ker2(bm2indices,:)'*(eye(size(bm2indices,1))-(1/size(bm2indices,1)))*Ker2(bm2indices,:);
N1 = N1 + 8000*eye(size(Ker1,1));
N = N + 640*eye(size(Ker2,1));
train = Ker1*inv(N)*(M1-M2)';
test = Ker2*inv(N1)*(bM1-bM2)';
%test = Ker1*inv(N)*(M1-M2)';

%train1 = Ker1*inv(N1)*(M1-M2)';
%test1 = Ker2*inv(N1)*(M1-M2)';

trainmodel = svmtrain(train,b);
accuracy = size(find(svmclassify(trainmodel, test)==d),1);
trainmodel = svmtrain(train,b,'kernel_function','rbf');
accuracy1 = size(find(svmclassify(trainmodel, test)==d),1);
accuracy
accuracy1
```

Accuracy reported - >

for linear svm (LDA) -> 60%

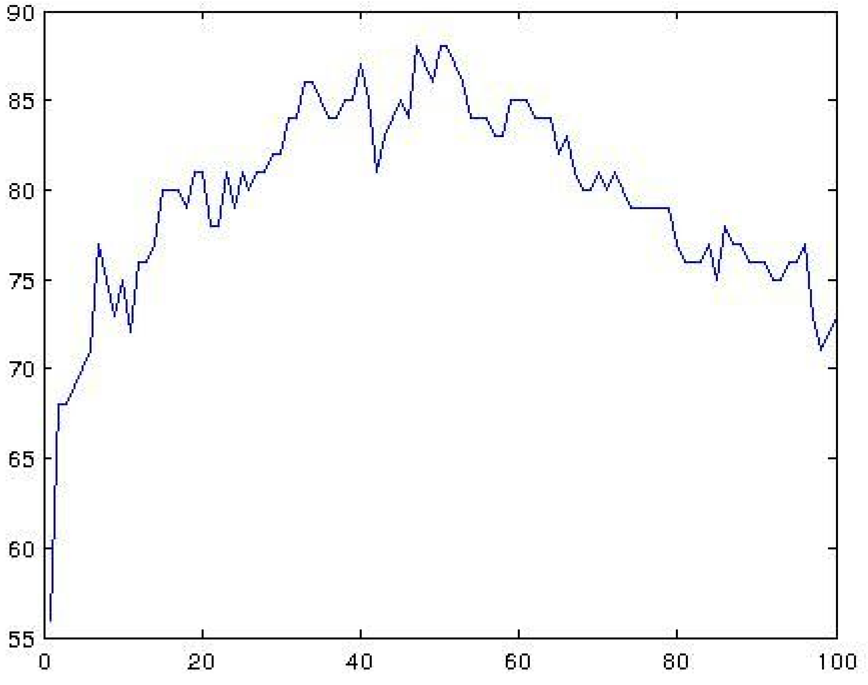
for svm using rbf (LDA)-> 64%

<u>Accuracy for linear SVM with kernel PCA over 1-100 eigenvectors</u>



mean accuracy = 76.93%

Accuracy for SVM (using RBF) with kernel PCA over 1-100 eigenvectors



mean accuracy = 79.61%

- Different values of sigma gave different accuracy results. A value of sigma = 1000 gave accuracy results of 58%. While the one used in the example (sigma = 10000) gave accuracy results 76%.
- I have trained my svm model on train+validation dataset (200 instances) which further gave better results because of better estimation of eigenvectors as compared to only training set (100 instances)
- Using RBF for SVM gave slightly better results as compared to linear SVM

------------------------------------ END---