Korea Advanced Institute of Science and Technology
School of Electrical Engineering
EE488 Intro. to Machine Learning, Spring 2018

Issued: Apr. 6, 2018
Due: Apr. 13, 2018

Assignment II-part2

Policy

Group study is encouraged; however, assignment that you hand-in must be of your own work. Anyone suspected of copying others will be penalized. The homework will take considerable amount of time so start as soon as possible..

1. **LDA**: Consider data generated in the following manner: (1) toss a coin with probability of heads equal to p; (2) when the coin toss results in "head" then $y = 1$ otherwise $y = 0$; (3) sample $x|y = 0 \sim N(\mu_0, \Sigma_0)$ and $x|y = 1 \sim N(\mu_1, \Sigma_1)$. Let $\theta = p, \Sigma_1, \Sigma_2, \mu_0, \mu_1$.

   (i) Show that $p(y = 1|\mathbf{x}; \theta) = \frac{1}{1+\exp(f(\theta,\mathbf{x}))}$. Find an explicit form for $f(\theta, \mathbf{x})$.
   (solution):

$$\begin{aligned} p(y=1|\mathbf{x};\theta) &= \frac{p(\mathbf{x}, y=1;\theta)}{\sum_{y\in\{0,1\}} p(\mathbf{x}, y;\theta)} \\ &= \frac{p(y=1|\mathbf{x};\theta)p(\mathbf{x}|\theta)}{p(y=1|\mathbf{x};\theta)p(\mathbf{x}|\theta) + p(y=0|\mathbf{x};\theta)p(\mathbf{x}|\theta)} \\ &= \frac{1}{1 + \frac{p(y=0|\mathbf{x};\theta)}{p(y=1|\mathbf{x};\theta)}} \end{aligned}$$

$$\frac{p(y=1|\mathbf{x};\theta)}{1 - p(y=1|\mathbf{x};\theta)} = \exp(f(\theta, \mathbf{x}))$$

$$\therefore p(y=1|\mathbf{x};\theta) = \frac{1}{1+\exp(f(\theta,\mathbf{x}))}$$

   Find an explicit form for $f(\theta, \mathbf{x})$.

$$\begin{aligned} f(\theta,\mathbf{x}) &= \log\frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} \\ &= \log\frac{p(\mathbf{x}|y=1)p(y=1)}{p(\mathbf{x}|y=0)p(y=0)} \\ &= \log\frac{1/(2\pi)^{D/2}1/|\mathbf{\Sigma_1}|^{\frac{1}{2}}\exp\left[-1/2(\mathbf{x}-\mu_\mathbf{1})^T\mathbf{\Sigma_1}^{-1}(\mathbf{x}-\mu_\mathbf{1})\right]p}{1/(2\pi)^{D/2}1/|\mathbf{\Sigma_0}|^{\frac{1}{2}}\exp\left[-1/2(\mathbf{x}-\mu_\mathbf{0})^T\mathbf{\Sigma_0}^{-1}(\mathbf{x}-\mu_\mathbf{0})\right](1-p)} \\ &= \log\frac{|\mathbf{\Sigma_0}|^{\frac{1}{2}}p}{|\mathbf{\Sigma_1}|^{\frac{1}{2}}(1-p)} + \frac{1}{2}(\mathbf{x}-\mu_\mathbf{0})^T\mathbf{\Sigma_0}^{-1}(\mathbf{x}-\mu_\mathbf{0}) - \frac{1}{2}(\mathbf{x}-\mu_\mathbf{1})^T\mathbf{\Sigma_1}^{-1}(\mathbf{x}-\mu_\mathbf{1}) \end{aligned}$$

(ii) The above represents Linear Discriminant Analysis (LDA) studied in class and it is in similar form as logistic regression. What are the difference and similarity between the two?

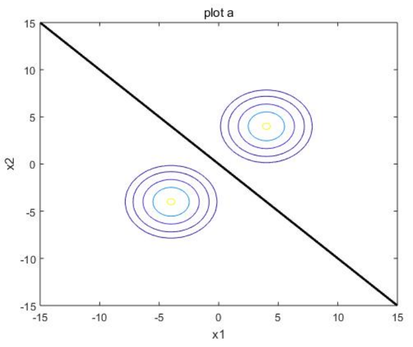
(solution): The similarity between the two is that they predict the class probability of a data point.
The difference between the two is that the logistic regression don't assume anything about $p(\mathbf{x}|y)$ but LDA assumes $p(\mathbf{x}|y)$ as Gaussian.

a. $\mu_0 \neq \mu_1$ but $\Sigma_0 = \Sigma_1 = I$

(solution):

$$f(\theta, \mathbf{x}) = \log \frac{p}{1-p} + \frac{1}{2}(\mu_{\mathbf{0}}^{\mathbf{T}}\mu_{\mathbf{0}} - \mu_{\mathbf{1}}^{\mathbf{T}}\mu_{\mathbf{1}}) + \mathbf{x}^{\mathbf{T}}(\mu_{\mathbf{1}} - \mu_{\mathbf{0}})$$

$\therefore$ Decision boundary is straight line.

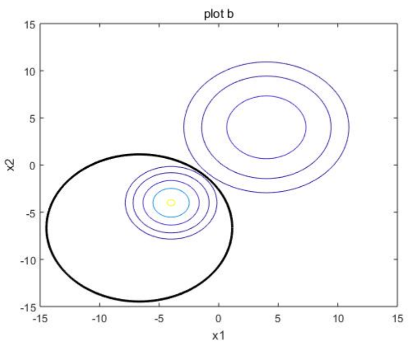


b. $\mu_0 \neq \mu_1$ and $\Sigma_0 = I, \Sigma_1 = 4 * I$

(solution):

$$f(\theta, \mathbf{x}) = \log \frac{p}{2(1-p)} + \frac{1}{2}(\mu_{\mathbf{0}}^{\mathbf{T}}\mu_{\mathbf{0}} - \frac{\mathbf{1}}{\mathbf{4}}\mu_{\mathbf{1}}^{\mathbf{T}}\mu_{\mathbf{1}}) + \mathbf{x}^{\mathbf{T}}(\frac{\mathbf{1}}{\mathbf{4}}\mu_{\mathbf{1}} - \mu_{\mathbf{0}}) + \frac{3}{8}\mathbf{x}^{\mathbf{T}}\mathbf{x}$$

$\therefore$ Decision boundary is circle.

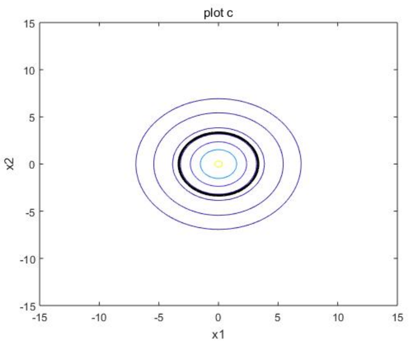


c. $\mu_0 = \mu_1$ and $\Sigma_0 = I, \Sigma_1 = 4 * I$

(solution):

$$f(\theta, \mathbf{x}) = \log \frac{p}{2(1-p)} + \frac{3}{8}(\mu_\mathbf{0}^\mathbf{T}\mu_\mathbf{0}) - \frac{3}{4}\mathbf{x^T}\mu_\mathbf{0} + \frac{3}{8}\mathbf{x^T x}$$

$\therefore$ Decision boundary is circle.



d. $\mu_0 = \mu_1 = (0,0)$ and $\Sigma_0 = \begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}$

(solution):

$$\Sigma_0^{-1} = \frac{1}{15}\begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}, \Sigma_1^{-1} = \frac{1}{15}\begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}$$

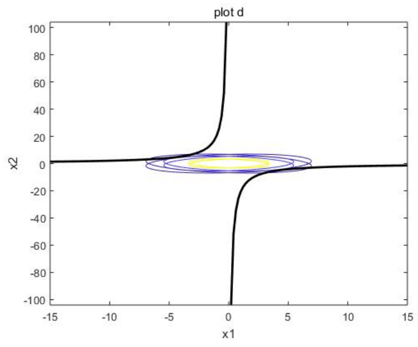
$$\begin{aligned} f(\theta, \mathbf{x}) &= \log \frac{p}{1-p} + \frac{1}{2}\mathbf{x^T}\left(\frac{1}{15}\begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix} - \frac{1}{15}\begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}\right)\mathbf{x} \\ &= \log \frac{p}{1-p} - \frac{1}{15}\mathbf{x^T}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\mathbf{x} \end{aligned}$$

Because $\mathbf{x}$ is two dimensional point by seeing the covariance matrix, we can set

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\therefore f(\theta, \mathbf{x}) = \log \frac{p}{1-p} - \frac{1}{15}x_1 x_2$$

$\therefore$ Decision boundary is hyperbola.



2. **Kernel SVM**: In class, Gaussian radial basis function (RBF) Kernel was introduced where $k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{1}{2}\|\mathbf{x}-\mathbf{z}\|^2\right) = \phi(\mathbf{x})^T\phi(\mathbf{z})$. Show that nth element of $\phi(x_i)_n = \frac{1}{\sqrt{n!}}\exp\left(-\frac{x_i^2}{2}\right)x_i^n$.

   (solution): using Taylor expansion of RBF kernel with $\gamma = \frac{1}{2\sigma^2}$

$$\begin{aligned}
K(x_i, z_j) &= \exp(-\frac{1}{2\sigma^2}||x_i - z_j||^2) \\
&= \exp(-\gamma||x_i - z_j||^2) = \exp(-\gamma x_i^2 + 2\gamma x_i z_j - \gamma z_j^2) \\
&= \exp(-\gamma x_i^2 - \gamma z_j^2)\left(1 + \frac{2\gamma x_i z_j}{1!} + \frac{(2\gamma x_i z_j)^2}{2!} + \frac{(2\gamma x_i z_j)^3}{3!} + \cdots\right) \text{ where } \exp(x_i z_j) = \sum_{n=0}^{\infty}\frac{(x_i z_j)^2}{n!} \\
&= \exp(-\gamma x_i^2 - \gamma z_j^2)\left(1\cdot 1 + \sqrt{\frac{(2\gamma)^1}{1!}}(x_i)^1 \cdot \sqrt{\frac{(2\gamma)^1}{1!}}(z_j)^1 + \sqrt{\frac{(2\gamma)^2}{2!}}(x_i)^2 \cdot \sqrt{\frac{(2\gamma)^2}{2!}}(z_j)^2 + \cdots\right) \\
&= \exp(-\gamma x_i^2)\left(1, \sqrt{\frac{(2\gamma)^1}{1!}}(x_i)^1, \sqrt{\frac{(2\gamma)^2}{2!}}(x_i)^2, \cdots\right)^T \exp(-\gamma z_j^2)\left(1, \sqrt{\frac{(2\gamma)^1}{1!}}(z_i)^1, \sqrt{\frac{(2\gamma)^2}{2!}}(z_i)^2, \cdots\right) \\
&= \phi(x_i)^T\phi(z_j)
\end{aligned}$$

   If $\sigma = 1$, then $\gamma = \frac{1}{2}$. Therefore the nth element of $\phi(x_i)_n = \frac{1}{\sqrt{n!}}\exp\left(-\frac{x_i^2}{2}\right)x_i^n$.

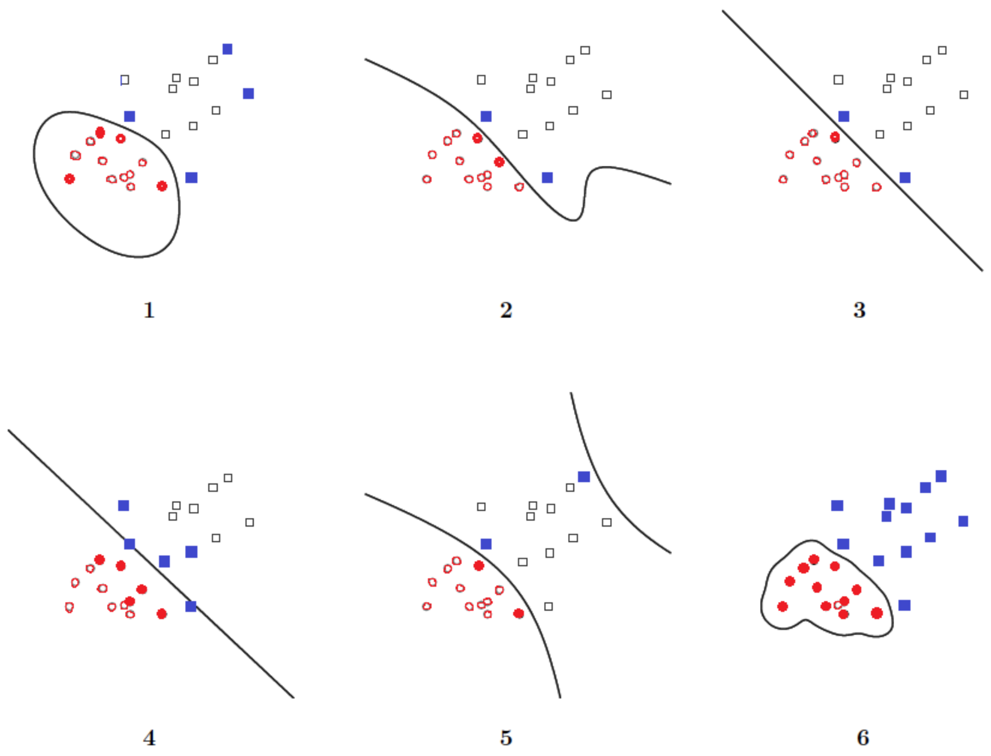
3. **Kernel SVM**: Consider the constrained optimization problem for obtaining soft-SVM

$$\min \frac{1}{2}\|\theta\|^2 + C\sum_{i=1}^{n}\xi_i$$

   such that $y_i(\theta \cdot \mathbf{x_i} + \theta_0) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for $i = 1, \ldots, n$.

   The following figures represents plots of Kernel SVM decision boundaries obtained using different kernels and/or different slack penalties. In the figure, there are two classes of training data with labels $y_i \in \{-1, 1\}$, represented by red circles and blue squares. The filled-in circles and squares represent support vectors. Find from figure $1-6$, a figure that presents the following conditions.

   (i) Soft-margin linear SVM with trade-off factor of $C = 0.1$
   [**Solution**] Fig. 4. The decision boundary has to be linear because it is linear SVM. However it has large margin.



(ii) Soft-margin linear SVM with trade-off factor of $C = 10$

[**Solution**] Fig. 3. The decision boundary has to be linear because it is linear SVM. However it has small margin.

(iii) Hard-margin kernel machine with $k(\mathbf{x}, \mathbf{z}) = \mathbf{x} \cdot \mathbf{z} + (\mathbf{x} \cdot \mathbf{z})^2$

[**Solution**] Fig. 5. Since the kernel is second order, it has to have shape of ellipse or hyperbolic.

(iv) Hard-margin kernel machine with $k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{1}{4}\|\mathbf{x} - \mathbf{z}\|^2\right)$.

[**Solution**] Fig. 1. Let $k(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma\|\mathbf{x} - \mathbf{z}\|^2\right)$. The kernel classfier will be in the form given as $y(\mathbf{x}) = \text{sign}(\sum \alpha_i y_i k(x_i, \mathbf{x}))$ and for given $k(\mathbf{x}, \mathbf{z})$, it has a circular contour, and therefore, the derived classifier will either be figure 1 or 6. Notice that the kernel value becomes larger as $\mathbf{x}$ moves closer to $\mathbf{z}$ and with a larger value of γ, $\mathbf{x}$ has to be even closer to $\mathbf{z}$ to have the same effect. Therefore, to define the decision boundry with large γ, you would need more support vectors for defining the decision boundry.

(v) Hard-margin kernel machine with $k(\mathbf{x}, \mathbf{z}) = \exp\left(-4\|\mathbf{x} - \mathbf{z}\|^2\right)$

[**Solution**] Fig. 6. See (iv) for explanation.

4. **Kernel SVM (Matlab Programming Assignment)**: In this problem, we implement a very simple algorithm for solving the following optimization problem of Kernel SVM in the

feature spaces $\phi(\mathbf{x})$,

$$\min_{\theta}\Big(\frac{\lambda}{2}||\theta||^2 + \frac{1}{m}\sum_{i=1}^{m}\max\{0, 1 - y_i\langle\theta, \phi(\mathbf{x}_i)\rangle\}\Big) = \min_{\theta}(\frac{\lambda}{2}||\theta||^2 + \mathcal{L}_s(\theta)).$$

while only using kernel evaluations where a kernel function is defined as $K(\mathbf{x}_i, \mathbf{x}) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x})$. The basic observation is that the vector $\boldsymbol{\theta}$ maintained by the SGD procedure is always in the linear span of $\{\phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2), ..., \phi(\boldsymbol{x}_m)\}$. Therefore, rather than maintaining $\boldsymbol{\theta}^{(t)}$, we can maintain the corresponding coefficients $\boldsymbol{\alpha}$. The algorithm for solving above optimization is given below.

(i) Implement the algorithm in Matlab code.

(ii) Explain line 5 of the algorithm.

(solution)
In kernel-SVM, $f(\boldsymbol{x}_i) = y_i\sum_{j=1}^{m}\alpha_j^{(t)}K(\boldsymbol{x}_j, \boldsymbol{x}_i)$. Based on the hing loss $\mathcal{L}_s(\boldsymbol{\theta})$ as convex optimizer, it is not differentiable, but has a subgradient with respect to the model parameters $\boldsymbol{\theta}$. In $\mathcal{L}_s(\boldsymbol{\theta})$, the loss occurs if $f(\boldsymbol{x}_i) < 1$. Then the parameter ($\boldsymbol{\alpha}$) is updated by the subgradient :$\beta_i^{(t+1)} = \beta_i^{(t)} + y_i$. In addition, $f(\boldsymbol{x}_i) \geq 1$ where no loss occurs, then the subgradient is equal to 0. Therefore $\beta^{(t+1)} = \beta^{(t)}$.

Algorithm 1: SGD for Solving Soft-SVM with Kernels

```
   Goal          : Solve Equation
   Parameter     : $T$
   Initialization: $\boldsymbol{\beta}^{(1)} = 0$
1  for $t = 1, ..., T$ do
2      Let $\boldsymbol{\alpha}^{(t)} = \frac{1}{\lambda t}\boldsymbol{\beta}^{(t)}$
3      Choose $i$ uniformly at random from $[m]$($m$ is the number of data)
4      For all $j \neq i$ set $\beta_j^{(t+1)} = \beta_j^{(t)}$
5      if $(y_i\sum_{j=1}^{m}\alpha_j^{(t)}K(\boldsymbol{x}_j, \boldsymbol{x}_i) < 1)$ then
6          Set $\beta_i^{(t+1)} = \beta_i^{(t)} + y_i$
7      else
8          Set $\beta^{(t+1)} = \beta^{(t)}$
9      end
10 end
   Output        : $\bar{\boldsymbol{\theta}} = \sum_{j=1}^{m}\bar{\alpha}_j\phi(\boldsymbol{x}_j)$ where $\bar{\boldsymbol{\alpha}} = \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\alpha}^{(t)}$
```

For implementing the RBF kernel svm algorithm, a skeleton code is provided in the "kernel_svm" folder. The main function `main_svm_rbf.m` calls the following 5 functions:
(1) `[x_train,y_train,x_test,y_test] = createDataset('train.csv','test.csv')`,
(2) `[avg_alpha], = train_rbf(x_train,y_train,T)`,
(3) `[acc] = test_rbf(x_test,y_test,x_train,avg_alpha)` and
(4) `plot_svm_rbf(x_train,y_train,x_test,avg_alpha)`
(5) `[K] = svm_kernel(x_i,x_j,kernel)` of radial basis function (RBF) kernels.
(6) `[prediction] = estimate_svm_rbf(x_input,x_train,avg_alpha)` of radial basis function (RBF) kernels.

(i) `[x_train,y_train,x_test,y_test] = createDataset('train.csv','test.csv')` loads both the training and test dataset contained respectively in `train.csv` and `test.csv`.

```
function [x_train,y_train,x_test,y_test] = createDataset(train_data,test_data)
%% Arguments %%
% train_data : name of train data
% test_data : name of test data
% x_train : train data
% y_train : train label
% x_test : test data
% y_test : test label
%% Codes %%
train = load(train_data);
test = load(test_data);
x_train = train(:,1:2);
y_train = train(:,3);
x_test = test(:,1:2);
y_test = test(:,3);
end
```

(ii) `[avg_alpha] = train_rbf(x_train,y_train,T)` takes both the training set {`x_train,y_train`} where `x_train` is the data array while `y_train` is the corresponding label array of training dataset as inputs to the svm and outputs an averaged alpha `avg_alpha`.

```
function [avg_alpha] = train_rbf(x_train,y_train,T)
%% Arguments %%
% x_train: training data
% y_train: training label
% T: the number of iteration
% theta: weight parameter
%% Your code here %%
% Set lambda value
lambda = 0.015;

% Calculate dimension of data
x_dim = 100;

% Initialization
beta = zeros(x_dim, 1);
alpha = zeros(x_dim, 1);

sum_alpha = alpha;
% Training
for t = 1:T
% let alpha set as follows as
alpha = 1/(lambda*t)*beta;
sum_alpha = sum_alpha + alpha;

% Choose i uniformly at random from [num of training data]
i = randi(length(x_train));
x_i = x_train(i, :);
y_i = y_train(i, :);

% Estimate feature map
y_sum = 0;
for j = 1:length(x_train)
if j ~= i
x_j = x_train(j,:);
k_ij = svmkernel(x_j, x_i,'rbf');
y_sum = y_sum + (alpha(j) * k_ij);
end
end

if y_sum * y_i < 1
beta(i) = beta(i) + y_i;
else
```

```
beta(i) = beta(i);
end

loss = y_i - y_sum;
yp = sign(y_sum);
% Display logs
log = ['step:',num2str(t),' loss:', num2str(loss), ' label:' , num2str(y_i), ' yp:', num2str(yp)];
disp(log);
end

avg_alpha = sum_alpha / T;
end
```

(iii) **[acc]=test_rbf(x_test,y_test,x_train, avg_alpha)** takes test set {**x_test,y_test**} the estimated averaged alpha **avg_alpha** and training set {**x_train**} and estimates the classification accuracy **acc**.

```
function [acc] = test_rbf(x_test,y_test,x_train,alpha)
%% Argurments %%
% x_test: test data
% y_test: test label
% alpha : average alpha
% acc: accuracy
%% Your code here %%
% Evaluate x_test
num_test = length(x_test);
num_train = length(x_train);
y_test_result = length(x_test);
for i=1:num_test

% ith data sample
x_i = x_test(i, :);
y_i = y_test(i, :);

% Estimate feature map
y_sum = 0;
for j = 1:num_train
x_j = x_train(j,:);
k_ij = svmkernel(x_i, x_j,'rbf');
y_sum = y_sum + (alpha(j) * k_ij);
end

loss = y_i - y_sum;
yp = sign(y_sum);

log = ['sample:',num2str(i),' loss:', num2str(loss), ' label:' , num2str(y_i), ' prediction:', num2
disp(log);

y_test_result(i) = yp;
end
% Calculate acc
acc = sum(y_test_result' == y_test) / length(y_test);

disp(['test accuracy:', num2str(acc)])
end
```

(iv) **plot_svm_rbf(x_train,y_train,x_test,y_test,avg_alpha)** takes the training set {**x_train,y_train**} to scatter plots the loaded data and makes the use of the averaged alpha **avg_alpha** to draw a decision boundary on the scatter plot. The positive samples should be represented as red dots while the negative samples should be in blue dots.

```
function plot_svm_rbf(x_train,y_train,x_test,y_test,alpha)
%% Arguments %%
% x_train: training data
% y_train: training label
% x_test: test data
% y_test: test label
% alpha: average alpha
%% Your code here %%
% Divide data according to label
x_train_pos = x_train(y_train==1,:);
x_train_neg = x_train(y_train==-1,:);
x_test_pos = x_test(y_test==1,:);
x_test_neg = x_test(y_test==-1,:);

% Plot decision boundary
max_x = max([x_train; x_test]);
min_x = min([x_train; x_test]);
[x1, x2] = meshgrid(linspace(min_x(1),max_x(1)), linspace(min_x(2),max_x(2)));
x_grid = [reshape(x1,[],1) reshape(x2,[],1)];

% Sum of support vectors
% x_grid : [10000 x 2]
% x_train : training data
% alpha: average alpha vector
y = estimate_svm_rbf(x_grid,x_train,alpha');

% decision boudary and margin
v = [0,1,-1];

% plot with cotour
figure('pos',[100 300 1200 500])
subplot(1,2,1)
hold on
contour(x1,x2,reshape(y,size(x1)),v);
plot(x_train_pos(:,1), x_train_pos(:,2), 'or', x_train_neg(:,1), x_train_neg(:,2), 'ob')
title('Train data')
subplot(1,2,2)
hold on
contour(x1,x2,reshape(y,size(x1)),v);
plot(x_test_pos(:,1), x_test_pos(:,2), 'or', x_test_neg(:,1), x_test_neg(:,2), 'ob');
title('Test data')

end
```

(v) `[K] = svm_kernel(x_i, x_j,kernel)` takes the randomly chosen ith sample and $j \neq i$th sample and kernel type, {`x_i, x_j, kernel`} and outputs a feature map `kernel` is defined by {`'RBF'`}.

```
function K = svmkernel(x_i, x_j,kernel_type)
%% Arguments %%
% x_i: ith random data from [m]
% x_j: j ~= i data from [m]
% kernel_type: kernel types : 'linear' or 'rbf'
%% Your code here %%
switch kernel_type
case 'linear'
%linear kernel type
K = x_i*x_j';
case 'rbf'
%rbf kernel type
K = exp(-(x_i-x_j)*(x_i-x_j)'/2);
otherwise
error('Unknown kernel function');
```

```
K = double(K);
end
end
```

(vi) `[y] = estimate_kernel(x_grid, x_train,avg_alpha)` takes the `x_grid` data samples, `x_train, avg_alpha` and outputs prediction scores `y`. In the skeleton code, the additional information on `x_grid` given.

```
function prediction = estimate_svm_rbf(x_input,x_train,alpha)
%% Arguments%%
% x_input: grid of data
% x_train : training data
% alpha : average alpha vector
% prediction : sum of support vector algorithm

[d, N] = size(alpha);
grid_size=length(x_input(:,1));
prediction=[];

for i=1:grid_size
y_sum =0;
for j=1:N
K=svmkernel(x_train(j,:),x_input(i,:),'rbf');
y_sum= y_sum + K*alpha(j);
end

%returns sum of support vector algorithm.
prediction(i,1) = y_sum;
end

end
```

Submit Instructions for Programming Assignment

- Please submit in .zip file to KLMS named **ee488_assignment2_student#.zip**, for example, "ee488_assignment2_20181234.zip".

- This file should contain the following folder - **kernel_SVM** and each folder contains document file for the result with analysis.

- In matlab code, the comment explaining your code **must be** included, or you will not get a full grade even if your code works fine. Please also include all the files that are required to run the code in the zip file. Do not change the name of the folder and comments should be written in English. Additionally submitting unexecutable code will receive no points.