**Naïve Bayes Classifier, LDA, QDA**

**Dividing The Data Nfold**

function data\_nfold = divide\_nfold\_data(feature, label, N)

% This is to split a dataset into N-fold for cross-valiation purpose

% feature: the data matrix, each row is a smaple, each column is an attribute

% label: class label of the samples

% N: divide dataset into N parts with equal size

C = unique(label);

for iC = 1:length(C)

cl = C(iC);

idx = find(label==cl);

data = feature(idx,:);

L = length(idx);

feat\_nfold = nfold\_set(data, N);

eval(['data\_nfold.class', num2str(cl), '=feat\_nfold;']);

end

%% The function to find points to seperate dataset to N-fold

function feat\_nfold = nfold\_set(feat, N)

% Determin the size of each subset

L = size(feat,1); % number of samples

n = floor(L/N); % basic subset size

rem = mod(L, N); % Modulus after division, there are some extra samples to assign

a = n\*ones(N,1);

if rem>0

b = nchoosek(1:N,rem);

c = ceil(rand\*size(b,1));

idx = b(c,:);

a(idx)= a(idx) + 1;

end

nfoldpt =[0; cumsum(a)];

nint = [nfoldpt(1:end-1)+1, nfoldpt(2:end)];

for i = 1:N

dsub = feat(nint(i,1):nint(i,2), :);

eval(['feat\_nfold.fold', num2str(i), '=dsub;']);

end

**Bayes, LDA, QDA with kfold cross validation**

**Opt=1,2,3**

data = heartdiseasedata;

feat = data(:,1:13); % feature matrix

label = data(:,14); % class label vector

C = unique(label); %extract label information from label vector

%-----2. Prepare N-fold dataset for classification----------%

N = 5; % N-fold cross validation

data\_nfold = divide\_nfold\_data(feat, label, N);

%-----3. Perform N-fold Cross-Validation using KNN Function-----------------%

ACC\_SUM = [];

for K = [3 5 7]

for Dorder = [1 2 5]

acc\_nfold = [];

Lpred\_nfold =[];

Ltest\_nfold =[];

confusion\_nfold = zeros(3,3);

for ifold = 1:N

%----prepare cross-validation training and testing dataset---%

idx\_test = ifold; % index for testing fold

idx\_train = setdiff(1:N, ifold); % index for training folds

Dtest = []; Ltest = []; % initialize testing data and label

Dtrain = []; Ltrain = []; % initialize testing data and label

%---construct the training and testing dataset for the ith fold cross validatoin

for iC = 1:length(C)

cl = C(iC);

dtest = eval(['data\_nfold.class',num2str(cl), '.fold', num2str(ifold)]);

Dtest = [Dtest; dtest];

Ltest = [Ltest; cl\*ones(size(dtest,1), 1)];

for itr = 1:length(idx\_train)

idx = idx\_train(itr);

dtrain = eval(['data\_nfold.class',num2str(cl), '.fold', num2str(idx)]);

Dtrain = [Dtrain; dtrain];

Ltrain = [Ltrain; cl\*ones(size(dtrain,1), 1)];

end

end

opt = 1;

Lpred = myBayesPredict(Dtrain, Ltrain, Dtest, opt);

%---Calculate Classification Accuracy-----%

acc = sum(Lpred==Ltest)/length(Ltest);

if ifold ==1

Lpred1 =Lpred;

Ltest1 =Ltest;

elseif ifold ==2

Lpred2 =Lpred;

Ltest2 =Ltest;

elseif ifold ==3

Lpred3 =Lpred;

Ltest3 =Ltest;

elseif ifold ==4

Lpred4 =Lpred;

Ltest4 =Ltest;

elseif ifold ==5

Lpred5 =Lpred;

Ltest5 =Ltest;

end

acc\_nfold(ifold, 1) = acc;

end

acc\_ave = mean(acc\_nfold);

ACC\_SUM = [ACC\_SUM; K, Dorder, acc\_ave];

end

end

function Lpred = myBayesPredict(Dtrain, Ltrain, Dtest, opt)

% This is for multi-class classification using Bayesian Decision Theory

% Function Input:

% Dtrain: training dataset, each row is a feature vector of a training sample

% Ltrain: class labels of training samples

% Dtest: testing dataset

% opt: classification options

% if opt==1, use Naïve Bayes

% if opt==2, use posterior probability as discriminant function

% if opt==3, use the derived formula based on multivariate normal

% distribution

%

% Function Output:

% Lpred: predicted class labels for the testing samples in Dtest

%% 1. Use Naive Bayes Function to Make classification

% Assume the features are independent, then we can use Naive Bayes for prediction

if opt==1

NB = fitcnb(Dtrain,Ltrain); % construct a Naive Bayes model NB

Lpred = predict(NB, Dtest); % apply the trained model NB to predict class of test samples in Dtest

end

%% 2. Use the discriminant function G(x) = likelihood\*prior for classification

% In a general case with correlated features, we can assume the features

% follows multivariate normal distribution, then we can use function "mvnpdf"

% to calculate the likelihood P(X|Wj) directly

% Decision Rule: select the class that maximizes P(X|Wj)P(Wj) - likelihood\*prior

if opt==2

C = unique(Ltrain);

Lpred = [];

for iC = 1:length(C) % For each class i, calculate P(X|Wj)P(Wj) for all testing samples

cl = C(iC);

idx = find(Ltrain==cl);

data = Dtrain(idx,:);

mu = mean(data); % feature mean vector

sigma = cov(data); % feature covariance matrix

P = length(idx)/length(Ltrain);

% For each testing sample, calculate P(X|Wj)P(Wj) = likelihood of class i \* prior of class i

for j = 1:size(Dtest,1)

x = Dtest(j, :);

likelihood = mvnpdf(x,mu,sigma); % likelihood of the current class i

prior = P; % prior of the current class i

% Record values of the discriminat function G(X)

% In the following matrix G, each row represent a class, and

% each column represent a testing sample

G(iC, j) = likelihood\*prior; % P(X|Wj)P(Wj)

end

end

% For each testing sample, find the index of the class that have maximum

% value of likelihood\*prior

[~, pred] = max(G);

Lpred = C(pred);

end

%% 3. Use the derived discriminant function G(x) for classification

% based on the the assumption of Multivariate Normal Distribution for features

if opt==3

C = unique(Ltrain);

Lpred = [];

for iC = 1:length(C)

cl = C(iC);

idx = find(Ltrain==cl);

data = Dtrain(idx,:);

mu = mean(data)';

E = cov(data);

P = length(idx)/length(Ltrain);

W = -0.5\*inv(E);

w = inv(E)\*mu;

w0 = -0.5\*mu'\*inv(E)\*mu-0.5\*log(det(E))+log(P);

for j = 1:size(Dtest,1)

x = Dtest(j, :)';

% The closed form of the derived discriminant function G(X)

G(iC, j) = x'\*W\*x + w'\*x + w0;

end

end

[~, pred] = max(G);

Lpred = C(pred);

end

**Random Forest Classifier:-**

function BaggedEnsemble = generic\_random\_forests(X,Y,iNumBags,str\_method)

BaggedEnsemble = TreeBagger(iNumBags,X,Y,'OOBPred','On','Method',str\_method)

% plot out of bag prediction error

oobErrorBaggedEnsemble = oobError(BaggedEnsemble);

figID = figure;

plot(oobErrorBaggedEnsemble)

xlabel 'Number of grown trees';

ylabel 'Out-of-bag classification error';

print(figID, '-dpdf', sprintf('randomforest\_errorplot\_%s.pdf', date));

oobPredict(BaggedEnsemble)

% view trees

view(BaggedEnsemble.Trees{1}) % text description

view(BaggedEnsemble.Trees{1},'mode','graph') % graphic description

**Neural Networks:-**

Solve a Pattern Recognition Problem with a Neural Network

% Script generated by NPRTOOL

% Sample Data Sets Avaiable in Matlab for Model Training

% https://www.mathworks.com/help/deeplearning/gs/deep-learning-toolbox-sample-data-sets.html

% This script used Breast cancer dataset

A =heartdiseasedata';

inputs =A(1:13,:);

targets =A(14,:);

% Create a Pattern Recognition Neural Network

% Construct a multilayer feedforward neural network

% Two hidden layers, the 1st hidden layger has 10 hidden unit

% the 2nd hidden layer has 20 hidden unit

hiddenLayerSize = [10 20];

net = patternnet(hiddenLayerSize);

% visualize the constructed neural network

view(net);

% Set up Division of Data for Training, Validation, Testing

% More Information on Training, Validation, Testing:

% https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

net.divideParam.trainRatio = 70/100;

net.divideParam.valRatio = 15/100;

net.divideParam.testRatio = 15/100;

% Train the Network

rng('default');

[net,tr] = train(net,inputs,targets);

% Test the Network

outputs = net(inputs);

errors = gsubtract(targets,outputs);

performance = perform(net,targets,outputs)

% Plot Confusion Matrix

figure, plotconfusion(targets,outputs)

% Plot Network Performance for Training, Validation, Testing

figure, plotperform(tr)

**Predictor Importance**

t = templateTree('MaxNumSplits',1,'Surrogate','on');

ens = fitrensemble(heartdiseasedata(:,1:13),heartdiseasedata(:,14),'Method','LSBoost','Learners',t);

[imp,ma] = predictorImportance(ens);

imp = predictorImportance