Lab#05

Classification with Machine Learning

CIS492/593 Big Data

Dataset Selection:

The selected dataset for this lab is has cancerous lesion features for benign 'B' or malignant 'M'. The dataset has 569 data rows each corresponds a sample, and 32 columns correspond to different features. The first column corresponds to patient ID that has got no statistical significance and hence dropped. The second one corresponds to diagnosis of the lesion having two values, 'B' and 'M', representing benign and malignant cancer severity and is considered as target or output variable. The rest of the columns are different shape features of the lesion and are considered as input variables.

Data Preprocessing:

The input and output data are extracted from the dataset. Since the output variable has 'B' and 'M' ASCII data, so, to make it statistically significant, 'B' and 'M' are represented by 0 and 1, respectively. The column associated to patient ID is dropped since it has no linkage with lesion feature.

The rest of 30 features are considered as input as it can be used to predict the lesion diagnosis. After summarizing the input data, it is concluded that each feature has a specific mean and variance. To make the variable invariant to scale (variance) and mean, batch normalization is used, in which mean of each column is subtracted from the respective column and then each columns is divided by it respective variance. Generally, batch normalization is given by:

$$X \leftarrow \frac{X - \mu}{\sigma^2}$$

Where X is column vector, μ and σ^2 are mean and variance of X and $\leftarrow$ represents the assignment operator. Batch normalization gives equal weight to all of the variables as it all of columns gets unit scale.

Training Classifiers:

Different classification models are created to have as accurate prediction as possible. In our case, we used 4 different classifiers with its default hyperparameters:

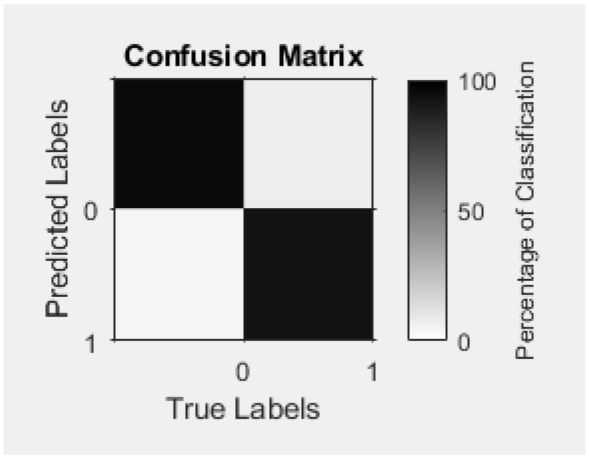
i. Support Vector Machine (SVM)
ii. Decision Tree (DT)
iii. Artificial Neural Network (ANN)
iv. Ensemble Learning Model

Author: **Hayat Ullah**
Email | LinkedIn | GitHub

Support Vector Machine (SVM):

Support Vector Machine (SVM) is a classifier that draws hyper-plane between two classes by maximizing margin between them. SVM with its default hyper-parameters use hyper-plane as a separator.

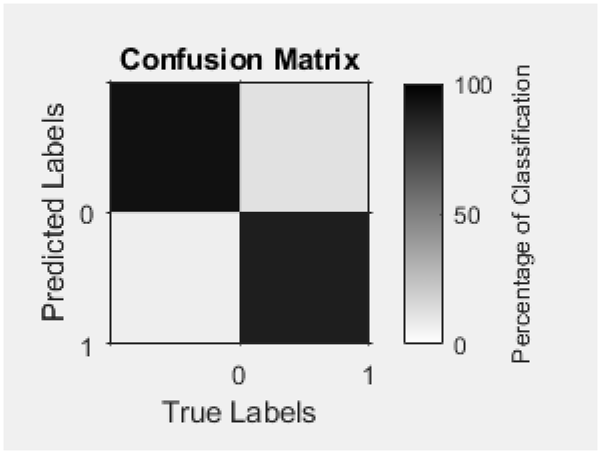
We trained SVM model with it default parameters and confusion matrix and accuracy reported for 5 fold cross-validation.



Decision Tree (DT) Classifier:

DT is also of a classifier that constructs a tree based on the training data, learn thresholds and classify data based on those hard thresholds.

We trained DT model with it default parameters and confusion matrix and accuracy reported for 5 fold cross-validation.

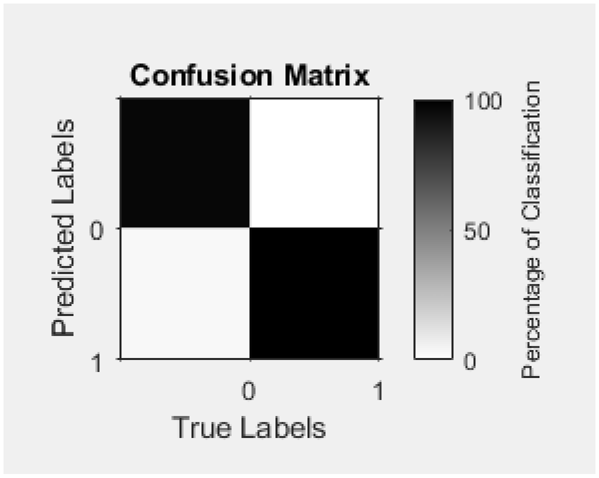


Artificial Neural Network (ANN):

ANN is a model of interconnected artificial neurons. An artificial neuron (AN) is a mathematical model inspired from biological neuron that maps a number of inputs onto a single output (weighted sum of input) followed by an activation function. An activation function decides when to trigger depending upon the input and activation function itself. ANN is a dense network of interconnected ANs. Each AN learns weights from training data.

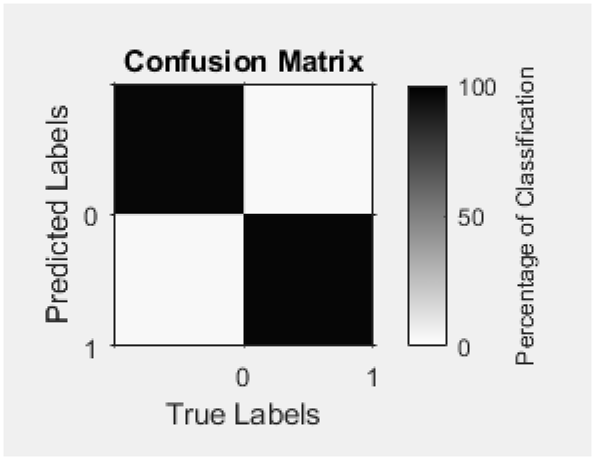
Here we used two layered neural network each having ten ANs. The first layer is activated by symmetric sigmoid function while the second is activated by Positive sigmoid function (generic sigmoid).

We trained an ANN model with it default parameters and confusion matrix and accuracy reported for 5 fold cross-validation.



Ensemble Learning Model:

It consists of 100 boosting trees and uses LogitBoost for binary classification and AdaBoostM2 for multiclass classification. We consider this model in MATLAB with its default parameters and the model is evaluated through confusion matrix and accuracy reported for 5 fold cross validation.



Evaluation Scores for all of the Models:

The average accuracy scores of 5 fold cross-validation test data for aforementioned models are listed in the table below:

Models	SVM	DT	ANN	Ensemble Learning
Accuracy	0.9456	0.9122	0.9824	0.9701
Precision	0.9364	0.8998	0.9758	0.9868
Recall	0.9523	0.9237	0.9983	0.9619
F1 Score	0.9435	0.9139	0.9871	0.9716

From the above table it is clear that ANN performed best, then Ensemble Learning, then SVM, and Decision Tree performed worst, yet it accuracy is above 90%.

Appendix:

Here is MATLAB code:

code.m file: This file acts a main file,

```
clc; clear; close all;

%% Load the Data:
filename = 'data.csv';
opts = detectImportOptions(filename);
loadNames;
opts.VariableNames = names;
data = readtable(filename, opts);

%% Extracting Input and Output from the table:
% Extracting features as X
X = table2array(data(1:end, 3:end));

% Extracting Labels as y from the table.
y = double((cell2mat(table2array(data(:, {'diagnosis'})))=='M'));

%% Preprocessing
% Batch normalization of the data.
% for making the data mean and scale invariant..
summary(data);
X_mean = mean(X);
X_var  = var(X);

X = (X - X_mean)./X_var;

%% SVM
% k-fold cross validation

k = 5;
indices = crossvalind('Kfold', y, k);
accSVM = zeros(1,k);
confmatSVM = zeros(2,2,k);

for i = 1:k
    disp(['SVM Training: Fold: ', num2str(i), ' out of ', num2str(k)]);
    testIdx = (indices==i);
    trainIdx= ~testIdx;
    mdl = fitcsvm(X(trainIdx, :), y(trainIdx));
    trueLab = y(testIdx);
    predLab = predict(mdl, X(testIdx, :));
    accSVM(i) = sum(trueLab == predLab)/length(trueLab);
    confmatSVM(:, :, i) = confusionmat(trueLab, predLab);
end
accSVM_mean = mean(accSVM);
confmatSVM_mean = round(mean(confmatSVM, 3));
figure, plotConf(confmatSVM_mean);



%% Decision Tree
accDT = zeros(1,k);
confmatDT = zeros(2,2,k);

for i = 1:k
    disp(['DecisionTree Training: Fold: ', num2str(i), ' out of ', num2str(k)]);
    testIdx = (indices==i);
    trainIdx= ~testIdx;
    mdl = fitctree(X(trainIdx, :), y(trainIdx));
    trueLab = y(testIdx);
    predLab = predict(mdl, X(testIdx, :));
    accDT(i) = sum(trueLab == predLab)/length(trueLab);
    confmatDT(:, :, i) = confusionmat(trueLab, predLab);
end
accDT_mean = mean(accDT);
confmatDT_mean = round(mean(confmatDT, 3));
figure, plotConf(confmatDT_mean);


%% Artificial Neural Network (ANN)
accANN = zeros(1,k);
confmatANN = zeros(2,2,k);

for i = 1:k
    disp(['ANN Predicting: Fold: ', num2str(i), ' out of ', num2str(k)]);
    testIdx = (indices==i);
    trueLab = y(testIdx);
    predLab = myANN(X(testIdx, :));
    accANN(i) = sum(trueLab == round(predLab))/length(trueLab);
    confmatANN(:, :, i) = confusionmat(trueLab, round(predLab));
end
accANN_mean = mean(accANN);
confmatANN_mean = round(mean(confmatANN, 3));
figure, plotConf(confmatANN_mean);



%% Ensemble Learning:
accEns = zeros(1,k);
confmatEns = zeros(2,2,k);

for i = 1:k
    disp(['Ensemble Model Training: Fold: ', num2str(i), ' out of ', num2str(k)]);
    testIdx = (indices==i);
    trainIdx= ~testIdx;
    mdl = fitcensemble(X(trainIdx, :), y(trainIdx));
    trueLab = y(testIdx);
    predLab = predict(mdl, X(testIdx, :));
    accEns(i) = sum(trueLab == round(predLab))/length(trueLab);
    confmatEns(:, :, i) = confusionmat(trueLab, round(predLab));
end
accEns_mean = mean(accEns);
```

Author: **Hayat Ullah**
Email | LinkedIn | GitHub

```
confmatEns_mean = round(mean(confmatEns, 3));
figure, plotConf(confmatEns_mean);
```

loadNames.m file:

```
names = {'id',  'diagnosis',    'radius_mean',  'texture_mean', ...
    'perimeter_mean',    'area_mean',    'smoothness_mean',  'compactness_mean', ...
    'concavity_mean',    'concave_points_mean',  'symmetry_mean',    'fractal_dimension_mean',
...
    'radius_se',    'texture_se',    'perimeter_se', 'area_se',  'smoothness_se', ...
    'compactness_se',    'concavity_se', 'concave_points_se',    'symmetry_se', ...
    'fractal_dimension_se', 'radius_worst', 'texture_worst',    'perimeter_worst', ...
    'area_worst',    'smoothness_worst', 'compactness_worst',    'concavity_worst', ...
    'concave_points_worst', 'symmetry_worst',   'fractal_dimension_worst'
```

myANN.m file:

```
function [y1] = myANN(x1)
%MYNEURALNETWORKFUNCTION neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 10-May-2021 21:46:21.
%
% [y1] = myNeuralNetworkFunction(x1) takes these arguments:
%   x = Qx30 matrix, input #1
% and returns:
%   y = Qx1 matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Pre-trained weights:
% Input 1
x1_step1.xoffset = [-0.575435840263102;-0.51784938627214;-0.0815984592230846;-
0.00412931546181952;-221.083657532062;-30.4608375075232;-13.9725966962376;-32.4901510830517;-
100.010086983807;-257.531369335064;-3.81876560435621;-2.81501324417232;-0.515927829486905;-
0.0162049711446021;-591.005161837264;-72.4226060908735;-35.0019512994888;-309.834188434758;-
185.273844569839;-414.200460044631;-0.356982439215611;-0.361527124750992;-0.0503495346068426;-
0.00214513600150147;-117.391647518914;-9.16894146533184;-6.25374316687479;-26.524670585524;-
34.8981465195523;-88.6112391324418];
x1_step1.gain =
[1.17553316574629;1.25119436449452;8.16032726863064;105.063460714894;0.00357135867604772;0.017
110529418061;0.0297809179963596;0.0149668068505322;0.00759134162799309;0.00209977770843357;0.0
556960723347617;0.134510232088553;0.385232609687679;7.73044196231115;0.000612918652688961;0.00
481723926398622;0.00460201130213684;0.0014424102719857;0.00192302859942953;0.00048378947548279
8;1.66205792779634;2.01367178937935;11.2468832804659;159.342993070276;0.00688529132307726;0.04
80344049125438;0.0695273010531322;0.0296958122274897;0.0150900198679516;0.00427927821393446];
x1_step1.ymin = -1;

% Pre-trained weights:
% Layer 1
b1 = [-1.6406606640665317709;-
1.1740640417282686148;0.7524793380555223620 8;0.7630620669096720432;0.5246563211091562761 7;0.19
775670836730888102;0.6730956053930012839;1.4878539022799490699;1.9176862977026338797;-
1.4905809896225388211];
```

Author: **Hayat Ullah**
Email | LinkedIn | GitHub

```
IW1_1 = [0.0081922966951132412255 -0.05328067004046827676 -0.35690525632096414821
0.30313545151551013479 0.17248841838847545072 0.283085920879297126 -0.0072967404036492228975 -
0.015315542914106087397 0.26242830629154911826 0.35951279066054964151 -0.10603662151994804053
0.57117540339034278407 -0.38854199847430087589 -0.013286094053727933589 -
0.10618929515080943737 0.011958816974146592102 -0.22523380069504661538 -0.21349769836308521431
-0.52442967477103952234 0.0090152701252611665977 0.32291050852534786131 -
0.067076633500003840505 0.14587556231119125361 -0.27998325849402344589 -
0.030682134102189718289 -0.40666600230478783251 0.15127725364908725658 -0.30155868605238000946
0.46954985136185301009 0.020834695688666426266;0.17174154636599855972 0.13442104648969532832 -
0.34576831588126100137 0.061026212102552157557 0.20950577177405793616 0.39295403036161341559 -
0.22180180218027409356 -0.14079325636343920092 -0.25241621821540832782 -0.1512887068401674584
-0.31500820024609976766 0.32610005740510622507 0.32145980514554511887 -0.039359387706786062855
0.35967647988499890088 0.13561063356358923904 0.062608885217135257228 -0.20930829994190885968
-0.31026121204074952331 -0.42307029834259424206 0.52800810392016150896 -0.25075941749437002937
0.39763568301188295662 0.0066681898971956265337 -0.062535955395563433412
0.19808508302022095893 -0.11950553081198404815 -0.24549936834344213143 0.16624775292641427726
0.34744691849138048534;-0.19501388887669141026 0.39369487506163619583 0.21402030809438182812
0.13470260572198947124 -0.48985350085783396779 0.13140792177314594702 0.13001201438659670817 -
0.12648210618388264215 0.046403866744225877317 -0.37777647628582333716 0.31206516731829359035
-0.24805200475994143572 -0.063477621612271009632 -0.38095642737312601644 -
0.16356687966603050688 0.0527432365525382571 -0.33008930669997410545 0.019561056078100699795 -
0.037261225326241799449 -0.35610977154163181435 0.22096514787814452996 -0.39148276050886859911
0.2369614464286968758 0.47215771285481850095 0.15722674270242587014 -0.32545940934816652845
0.12589894447794083843 -0.59870191817131124257 0.050156694037893842508 -
0.14143115703361894275;-0.22396456150604088231 -0.086367768086980961639
0.028855313820044647261 0.020012499036744832437 -0.2124457890439458152 -0.34600632298878658544
-0.16550693630604282336 0.046377562137678220167 -0.36488276669159663834 -
0.12707986310650060124 -0.47767946418618262294 0.286325296644580507 0.13061742235353468455 -
0.42050373593818329843 0.08814039217018714012 -0.18999512165248305662 -0.50236011029878835377
-0.23095941473066824989 -0.42238003645743071868 -0.53192597586538925558 -0.4925565959642553282
-0.31402650028935025217 -0.25798871752699059634 -0.27833208330287106946 0.30532230880083766067
-0.26939196188520220243 -0.14872588062630121009 -0.259072350266487339 -0.015224491106428907072
-0.14587187059133641109;0.36302165650513001927 -0.075519744781128106403 0.30356266043462415416
0.14513506303474413284 -0.071479671719340567848 -0.05149903528016219062 0.26667693766687577961
0.81169000881202546527 0.42147306554960251024 -0.053675039521459022573 -0.12028050226355029018
-0.27730818985568739254 -0.087796022358964734278 -0.30492743825078749698 -
0.13148114158923829287 -0.31588853766721558536 -0.80534124417957975162 0.13427094610982387435
0.11570841742385103157 -0.30220483936699438221 1.0330975351058981904 1.1632901930141832381
0.39294210555664854656 0.60957351658375558134 0.21600885640792061304 0.40959924320051555036
0.45846631086882189132 1.2953514299314166092 0.44395455456866061983
0.35354897196662837811;0.36013825039855396071 -0.038443382024422399768 0.10895740345173769714
-0.38155717316904225189 0.34336492431108167622 0.34226539110220466355 0.39583967517434509586 -
0.25430808315243541573 0.056526319110721111316 0.34817673485262901067 -0.20431791883102756735
-0.25035375081476757275 0.26677878949578182066 -0.12508443573257571613 -0.41918740092327538838
-0.19139043593577881319 -0.17938598833680294287 -0.20969113643644565759 -0.198687517056187396
0.36989059992680412003 -0.40667148525107371659 -0.35589660850186571883 -
0.087447653598819549692 -0.23317793068837897041 -0.37373370437399289212
0.067213058675839015832 -0.37695720005280419773 -0.15173332891083846086 -
0.31239682796138895648 -0.13039662993690159798;0.065177343294589420997 0.013357402574153708075
0.20180458385708829883 -0.33757727801706155502 -0.24785909851923618175 -0.39245096887448255307
0.19365288168965297255 0.018570395299732422867 -0.44714944470358214401 -0.41618862410040446953
-0.2844307121497348656 0.03587499768545233253 0.12541821379039935769 -0.47025632629453212319
0.082709655616599625505 0.35425124044645922572 0.17805466419404361589 0.26360955397419483237
0.12709719014228973433 -0.22113617691907194729 -0.47375968134926921271 -0.41011215065253719958
0.07637597930722170303 -0.34737156974329364001 -0.13827709745071520286 -0.18824660495248765613
-0.45311535897699345421 0.16430781187252091469 -0.30159311103891106587
0.022637221920085427024;0.24532215875139243288 -0.24682427563082517952 0.48266383743899943237
-0.32841884692705142212 -0.43531145297431794727 -0.32278664178749505442 0.86735327032382081303
0.65518305300239187972 0.32340273012730424984 -0.13753117965057248817 0.50240122331491166019 -
0.52425119765881245115 0.19098803857191454236 -0.2132347575435865894 -0.51888881225701821709 -
0.67605273332027315814 -0.28637613206018552869 -0.12295285309401060569 0.21720513063189755831
```

Author: **Hayat Ullah**
Email | LinkedIn | GitHub

```
-0.824526860511519466 0.959602659448389849 75 1.4460320098517747134 0.79317577536784444892
```

```
-0.824526860511519466 0.95960265944838984975 1.4460320098517747134 0.79317577536784444892
0.78457817174597832555 0.8613994673057810969 -0.17457214313182609078 0.42843531576642168712
0.71540154855414672674 0.80413262799392770663 0.61302651393648499578;0.07064914386616996666
0.24592757945435766986 -0.04832647948837291707 0.10836277801442828483 0.23910120253505659504 -
0.4797587787419003158 0.59965387520440116553 0.19476224219977006369 0.22563158920197515234 -
0.57692434169809880817 1.7588609603086451383 0.11868241556869052467 0.73583848941213336747
0.20087400069756880527 0.13018746106037801247 -1.1044009409058608284 -0.13436734642640951742
0.35864341088082385411 -0.64231113054099953086 -0.87728787120363704854 0.61617241392551946877
0.61501595337666437047 0.67689868249808771115 0.57753796264529588811 0.44308148363753169 23 -
0.53698418872700581783 0.16195519348787315583 0.56384630050226813847 -0.11432696699064737178
0.38088616136329334072;-0.10406926442788146103 0.17759397806100657435 -0.33875686454081460619
0.020131757794295750824 0.1433579344503987385 -0.27863501907348919 85 0.18577028288841843096
0.064858833763424703078 0.25587456368930255879 0.22320158984376903266 0.47086913301357602446 -
0.29382683815849502951 0.038803793463825009269 0.29905564261065259446 0.33420699685020127667 -
0.17370600775368111313 0.42173314751243634602 -0.14842744191226023287 -0.14969676027324030354
0.24107939093361552274 0.043517420826226782538 -0.31880331943404005557 -0.32548843365280566431
0.13520470245314955 56 0.45371188019247116197 -0.40072304024485411178 -0.39843072284398456384
0.25585477209683882283 0.36159192477 2744433 -0.24513191644956144888];

% Layer 2
b2 = 0.71783525723917851469;
LW2_1 = [-0.68625554726056092747 0.2115999474131095015 -0.41235092088722707926
1.2778884197750792762 2.290552920376476731 0.062823765817620835783 0.35866548548658810924
3.3229927485069010729 3.6523361480113947231 0.16596780454250939507];

% ===== SIMULATION ========

% Dimensions
Q = size(x1,1); % samples

% Input 1
x1 = x1';
xp1 = mapminmax_apply(x1,x1_step1);

% Layer 1
a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

% Layer 2
a2 = logsig_apply(repmat(b2,1,Q) + LW2_1*a1);

% Output 1
y1 = a2;
y1 = y1';
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
y = bsxfun(@minus,x,settings.xoffset);
y = bsxfun(@times,y,settings.gain);
y = bsxfun(@plus,y,settings.ymin);
end

% Sigmoid Positive Transfer Function
function a = logsig_apply(n,~)
a = 1 ./ (1 + exp(-n));
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
```

Author: **Hayat Ullah**
Email | LinkedIn | GitHub

```
a = 2 ./ (1 + exp(-2*n)) - 1;
end
```

plotConf.m file: for plotting confusion matrix C

```
function plotConf(C)
img = (C./sum(C,1));
z = 15;
img = imresize(img, z, 'nearest');
imshow(img)
img(isnan(img)) = 0;
fig = imshow(1-img);
colormap hot
c = colorbar;
c.YDir = 'reverse';
axis on
fig.YData = [0 1];
fig.XData = [0 1];
set(gca, 'xticklabels', {'','0','1'});
set(gca, 'yticklabels', {'','0','1'});
xlim([0 1]);
ylim([0 1]);
c.TickLabels = 100:-50:0;
xlabel('True Labels')
ylabel('Predicted Labels')
ylabel(c, 'Percentage of Classification')
title('Confusion Matrix');
grid on
```

Author: **Hayat Ullah**
Email | LinkedIn | GitHub