Documentation: Project Markov

1 Data Pre-Processing

1.1 Format of Raw Data:

Data obtained from the sensor can be either fused or non-fused. Non fused data has been used as there is no much use using fused data. The Raw data obtained from the BNO -055 Sensor is byte encoded and has the data in the following format.

Packet Sno	Lax	Hax	Lay	Hay	Laz	Haz	Lmx	Hmx	Lmy	Hmy	Lmz	Hmz	Lgx	Hgx	Lgy	Hgy	Lgz	Hgz

L- Lower Significant Byte, H-Higher Significant Byte.

ax,ay,az- Accelerometer data gx,gy,gz- Gyroscope data mx,my,mz- Magnetometer data

Hence it is decoded and processed to get the actual values using.

Example: 21 2 -57 2 -125 -22 -105 6 -61 -28 6 122 -1 -126 -4 -67 -5 40 -1

1.2 Decoding Raw Data:

Raw data as discussed in previous section is Byte Encoded hence must be encoded this is done by shaping the matrix and type casting the data as shown in the code snippet such that the absolute value is obtained.

Code Snippet:

```
load Clip1.txt; % Load the Corresponding text file

nfu = typecast(int8(Clip1),'uint8');
% Replace Clip1 With  Corresponding text file
nfumat = reshape(nfu,[19,length(nfu)/19]);
sno = nfumat(1,:);

TotalLoss=sum(PacketLost(:,2))
ax = typecast(uint16(hex2dec([dec2hex(nfumat(3,:))
dec2hex(nfumat(2,:))])),'int16');
ay = typecast(uint16(hex2dec([dec2hex(nfumat(5,:))
dec2hex(nfumat(4,:))])),'int16');
az = typecast(uint16(hex2dec([dec2hex(nfumat(7,:))
dec2hex(nfumat(6,:))])),'int16');

mx = typecast(uint16(hex2dec([dec2hex(nfumat(9,:))
dec2hex(nfumat(8,:))])),'int16');
my = typecast(uint16(hex2dec([dec2hex(nfumat(11,:))
dec2hex(nfumat(10,:))])),'int16');
mz = typecast(uint16(hex2dec([dec2hex(nfumat(13,:))
dec2hex(nfumat(12,:))])),'int16');

gx = typecast(uint16(hex2dec([dec2hex(nfumat(15,:))
dec2hex(nfumat(14,:))])),'int16');
gy = typecast(uint16(hex2dec([dec2hex(nfumat(17,:))
dec2hex(nfumat(16,:))])),'int16');
gz = typecast(uint16(hex2dec([dec2hex(nfumat(19,:))
dec2hex(nfumat(18,:))])),'int16');

clear nfu nfumat
disp('NFU Data ready!');
```

1.3 SIGNAL CORRECTION AND PACKET LOSS:

The signal received from the BNO-055 sensor node suffered had packet loss due to which there was discrepancies in time stamps of the signal and corresponding videos. This critical issue has to be addressed before proceeding further because the classifiers must have proper ground Truths no data gets misclassified. Hence, Correction was done by padding zeros at location where the data was lost.

Code snippet:

Packet Loss Estimation:

```
function [PacketLost]= packetloss(sno) % Function to Find the location and
number of packet Lost
loc=[]; loss=[];
sno=single(sno); % Serial Number a.k.a packet number 0-255
for i=1:size(sno,2)-1
    if sno(i+1)== sno(i)+1 || (sno(i)== 255 && sno(i+1)== 0)
        loc=[loc];
    else
        loc=[loc; i];
    end
end
for i=1:size(loc,1)
    if sno(loc(i)+1)-sno(loc(i))<= 0
        loss(i)=uint8(255-abs(sno(loc(i)+1)-sno(loc(i))));
    else
        loss(i)=uint8((sno(loc(i)+1)-sno(loc(i))));
    end
end
PacketLost=[loc double(loss')];
end
```

Signal Correction:

```
function [ SignalCorrected ] = SignalCorrect( Signal,PacketLost ) % Function
to Correct the Signal and Nullify Packet Loss
Loc=PacketLost(:,1); % Location of Packet Lost
NumPackets=PacketLost(:,2); % Number of Packets Lost at Corresponding loca-
tions
Total=sum(NumPackets);  % Total number of Packets Lost
SignalCorrected=[];
for j=1:size(Signal,2)
    flag=0; % Intitally assume that no packet was lost
    for i=1:size(Signal,1)+Total
        if any(Loc == i)
            idx=find(Loc == i);
            SignalCorrected(i+flag,j)=Signal((i-flag),j);
            SignalCorrected(i+1:i+NumPackets(idx)-1,j)=Signal((i-flag),j);
            flag=flag+NumPackets(idx);
        else
            flag=flag;
            SignalCorrected(i+flag,j)=Signal((i-flag),j);
        end
    end
end
end
```

1.4 DETERMINING SHOT LOCATION:

Location of the shots is crucial task when shot class classification is to be done. To segment the shot from the real time signal and perform further processing, we find the point where the Root-Mean Square(RMS) value changes instantaneously (i.e. During Impact) above a predefined threshold (6g). By performing RMS Thresholding we remove the locations where there where only empty swings and unwanted activity by the player. This avoids the unwanted computations that are performed. To ensure that we don't have

multiple peaks detected for a single shot (Some Shot might have large impact time- Improper Shots) we filter out all shots within its vicinity (20 samples before and after). Once the location of the shots are identified it has to be segmented for further processing. This Segmentation task is discussed in the next section.

Code Snippet:

```
%% Finding the location of Shots based on RMS Threshold
function[NShots1,Location1]=Findshots(rms,WindowSize,Thresholdrms)
NShots1=0; Location1=[];Lred=[];
ptemp=(diff(rms));
for i=1:length(ptemp)
    if (ptemp(i)>=Thresholdrms || ptemp(i)<=-Thresholdrms  );
        Location1=[Location1;i];
    end
end
for i=1:length(Location1)-1
    if Location1(i+1)-Location1(i)<=(0.6*WindowSize)
        Lred=[Lred;i+1];
    end
end
Location1(Lred) = []; % Remove Adjacent Peaks in Same Window
NShots1= size(Location1,1);% Number of Shots
end
```

1.5 WINDOWING:

After the Location of the shot has been identified, we consider a constant window of size 70 samples (0.7s). It was observed that shots had impact time of almost 0.2s and hence a window with 0.4s and 0.3s before and after the impact was considered to ensure that we get enough information to classify the shot. It was observed that for all shots collected in the dataset the mean value was around 60 Samples. Hence, we chose to fix the window size to be 70, 40 samples before and 30 Samples after the impact. Future improvements can be done in development of adaptive window based on the signal.

Code Snippet:

```
%% Window Extraction Function
function [Sigparsed]=SignalExtractdata(data,Start,End)
Sigparsed=[];
LocShots=[single(Start) single(End)];
for i=1:size(Start,1)
    for j=1:size(data,2)
       Signal{i,j}=[data(LocShots(i,1):LocShots(i,2),j)];
    end
end
Sigparsed=Signal;
end
```

2 FEATURE EXTRACTION

Feature extraction refers to process of extraction of meaningful information from the data sample that can represent the sample itself at lower dimensionality. We usually start from an initial set of measured data and derive features (values) intended to be informative and non-redundant, facilitating the subsequent processing and classification. Feature extraction is a kind of indirect dimensionality reduction

where instead of considering the entire 70 sample window we consider only 24 features for each window(Shot).

Features Considered: Mean and Variance of Acceleration along all axis (6), Mean and Variance of Gyroscope data along all axis (6), Minimum of Acceleration and Gyroscope data in the segmented window (3) and Acceleration and Gyroscope data during the impact (Peak) (3).

The above mentioned features were considered because they are global features that do not have depend on the locality and represent the characteristics of the type of shot played.

Feature Vector Organisation:

Mean ax	Mean ay	Mean az	Var ax	Var ay	Var az	min(ax)	min(ay)

min(az)	ax(Impact)	ay(Impact)	az(Impact)	Meangx	Meangy	Meangz	Vargx

Var gy	Var gz	min(gx)	min(gy)	min(gz)	gx(Impact)	gy(Impact)	gz(Impact)

Code Snippet:

```
function [Features] = FeatureExtract(Sigparsed) % Function
to Extract Features
% Sigparsed - Cell Type Contains all Shots that are Segmented
Features=[]; % Variable to Store Features
for i=1:size(Sigparsed,1)
    Meanax=mean(cell2mat(Sigparsed(i,1)));
Meangx=mean(cell2mat(Sigparsed(i,4))); % Mean Acceleration and Gyroscope data
    Meanay=mean(cell2mat(Sigparsed(i,2)));
Meangy=mean(cell2mat(Sigparsed(i,5)));
    Meanaz=mean(cell2mat(Sigparsed(i,3)));
Meangz=mean(cell2mat(Sigparsed(i,6)));
    Sigax=range(cell2mat(Sigparsed(i,1)));
Siggx=range(cell2mat(Sigparsed(i,4)));% Variance Acceleration and Gyroscope
data
    Sigay=range(cell2mat(Sigparsed(i,2)));
Siggy=range(cell2mat(Sigparsed(i,5)));
    Sigaz=range(cell2mat(Sigparsed(i,3)));
Siggz=range(cell2mat(Sigparsed(i,6)));
    ax=cell2mat(Sigparsed(i,1)); ay= cell2mat(Sigparsed(i,2));
az=cell2mat(Sigparsed(i,3));
    gx=cell2mat(Sigparsed(i,4)); gy= cell2mat(Sigparsed(i,5));
gz=cell2mat(Sigparsed(i,6));
    rmstmp=cell2mat(Sigparsed(i,10));
    Peak=max(rmstmp);
    ind = find(rmstmp==Peak);
    loc=max(ind);  % Location of Peak (Impact Point)
    Features=[Features; Meanax Meanay Meanaz Sigax Sigay Sigaz min(ax)
min(ay) min(az) ax(loc,1) ay(loc,1) az(loc,1) Meangx Meangy Meangz Siggx
Siggy Siggz min(gx) min(gy) min(gz) gx(loc,1) gy(loc,1) gz(loc,1)];
end
end
```

3 SUPPORT VECTOR MACHINE (SVM)

3.1 INTRODUCTION TO SUPPORT VECTOR MACHINE:

Vladimir Vapnik along with his co-workers invented Support Vector Machines in 1979. A linear SVM is a Hyperplane that separates positive examples from negative examples with a maximum margin (see figure 1). The margin is defined by the distance of the Hyperplane to the nearest of the positive and negative examples. The output of a linear SVM is

$$u = \vec{w}.\vec{x} - b$$

Output is based on the location of the test input (Either above or below) with respect to the Hyperplane and usually class labels are assigned based on the sign of the output variable.

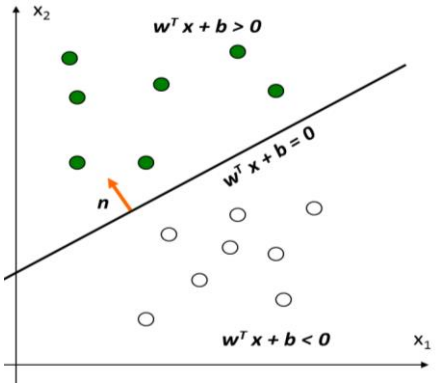

Figure 1: Linear SVM Model

(1)

Where *w* is the normal vector to the hyperplane and *x* is the input vector. The separating hyperplane is the plane *u*=0. The nearest points lie on the planes $u = \pm 1$. The margin $m = \frac{1}{\|w\|_2}$

Maximizing margin can be expressed via the following optimization problem:

$\underbrace{min}_{\vec{w},b} \frac{1}{2}\|w\|^2$ Subject to $y_i(\vec{w}.\vec{x} - b) \geq 1.$

Training of SVM needs solution of a very large quadratic programming (QP) optimization problem. Sequential Minimization Optimization breaks this QP problem into a series of smallest possible QP problems. These small QP problems are solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop.

3.2 APPLYING SVM TO SHOT CLASSIFICATION PROBLEM:

Support vector machines (SVM) are powerful binary classifiers based on optimal hyper planes with either hard or soft margins. Generally SVMs classifiers are fast, have capability to use kernels and solutions obtained are sparse in nature. Optimizing margin can be controlled easily and it doesn't have any local maxima. SVM overcomes various traditional issues such as the "curse of dimensionality", "over-fitting" and etc. SVM has strong theoretical foundation and well established method for implementation, gaining rapid development and popularity due to attractive features like good mathematical illustrations, geometrical descriptions, good generalization capabilities and promising performance.

SVMs have been successfully applied and extensively used in Activity Recognition for detection of various human activities like walking, Jogging, Running etc. Hence we apply SVMs to classify the type of shot which is similar to Activity recognition.

3.3 SVM IMPLEMENTATION ON MATLAB:

SVM has been implemented using various methods of solving the quadratic problem discussed above. Sequential Minimal Optimization is the state of art technique that solves the convex optimization problem efficiently. LIBSVM Library uses SMO to train SVM Models and is can be developed in various languages like Python, Java, C++, MATLAB, OCTAVE etc.

LIBSVM Library Link: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Once the Zip File is downloaded extract the file to the location where the project code is being developed. Add this Folder and its Sub Folder to Path, then open MATLAB type mex –setup

If you don't see Microsoft Visual C/C++ compiler install visual Studio to proceed further, if you have the Visual C/C++ Compiler already installed choose that option.Once the Compiler is set, Run Male file to Setup the LIBSVM Library

Svmtrain Command Syntax:

Model = svmtrain (training_label_vector, training_instance_matrix [, 'libsvm_options']);

-training_label_vector:
An m by 1 vector of training labels (type must be double).

-training_instance_matrix:
An m by n matrix of m training instances with n features.
It can be dense or sparse (type must be double).

-libsvm_options: A string of training options in the same format as that of LIBSVM.

Svmpredict Command Syntax
[predicted_label] = svmpredict (testing_label_vector, testing_instance_matrix, model [, 'libsvm_options']);

-testing_label_vector:
An m by 1 vector of prediction labels. If labels of test
Data are unknown, simply use any random values. (type must be double)
-testing_instance_matrix:
An m by n matrix of m testing instances with n features.
It can be dense or sparse. (Type must be double)
-model:
The output of svmtrain.
-libsvm_options:
A string of testing options in the same format as that of LIBSVM.

libsvm_options:

-s svm_type : set type of SVM (default 0)
0 -- C-SVC
1 -- nu-SVC
2 -- one-class SVM
3 -- epsilon-SVR
4 -- nu-SVR
-t kernel_type : set type of kernel function (default 2)
0 -- linear: u'*v
1 -- Polynomial: (gamma*u'*v + coef0)^degree
2 -- Radial basis function: exp(-gamma*|u-v|^2)
3 -- Sigmoid: tanh(gamma*u'*v + coef0)
-d degree: set degree in kernel function (default 3)
-g gamma: set gamma in kernel function (default 1/num_features)
-r coef0: set coef0 in kernel function (default 0)
-c cost: set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu: set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon: set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cache size: set cache memory size in MB (default 100)
-e epsilon: set tolerance of termination criterion (default 0.001)
-h shrinking: whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability estimates: whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
-wi weight: set the parameter C of class i to weight*C, for C-SVC (default 1)
The k in the -g option means the number of attributes in the input data.

For the Classification Problem we need to build multi class classifier which can be done In two ways:

One vs. one: requires N (N-1) / 2 Models; Computations will be More; Generally Not Preferred; (LIBSVM by Default Implements one vs. one).

One vs. all: Required only N Models; Fewer Computations;

Options Chosen for SVM Model: -s 0 (i.e. C-SVC rbf kernel –c Hyperplane parameter –g Gamma parameter)

Final model: -s 0 -t 2 -c best C-g bestGamma -b 0 bestC and bestGamma are obtained by fine tuning the parameter and optimizing the cost using cross validation.

Please refer to LIBSVM FAQ and Readme if any discrepancies arise:

Link: http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html#

3.4 PARAMETER SELECTION AND CROSS VALIDATION:

Cross validation is used for selecting good parameters. After finding them, you want to re-train the whole data without the -v option. Due to random partitions of the data, on different systems CV accuracy values may be different.

Choice of Kernel: When RBF is used with model selection, then there is no need to consider the linear kernel. The kernel matrix using sigmoid may not be positive definite and in general its accuracy is not better than RBF. Polynomial kernels are ok but if a high degree is used, numerical difficulties tend to happen (thinking about dth power of (<1) goes to 0 and (>1) goes to infinity).

Enabling Probability outputs do not guarantee you better accuracy. The purpose of this option is to provide you the probability estimates and not to boost prediction accuracy. Parameter selections, in general with and without -b have similar accuracy.

Code Snippet : Estimate BestC and BestGamma;

```
folds = 10; % N Fold Cross Validaiton
[C,gamma] = meshgrid(-5:2:15, -15:2:3);
%# grid search, and cross-validation
cv_acc = zeros(numel(C),1);
for i=1:numel(C)
    cv_acc(i) = svmtrain(labels, data, ...
                    sprintf('-s 0 -t 2 -c %f -h 0 -g %f -v %d ',
2^C(i), 2^gamma(i), folds));
end
%# pair (C,gamma) with best accuracy
[~,idx] = max(cv_acc); % Index where the CrossVal Accuracy Maximum

%# contour plot of paramter selection
contour(C, gamma, reshape(cv_acc,size(C))), colorbar
hold on
plot(C(idx), gamma(idx), 'rx')
text(C(idx), gamma(idx), sprintf('Acc = %.2f %%',cv_acc(idx)), ...
    'HorizontalAlign','left', 'VerticalAlign','top')
hold off
xlabel('log_2(C)'), ylabel('log_2(\gamma)'), title('Cross-Validation
Accuracy')

%# now you can train you model using best_C and best_gamma
best_C = 2^C(idx);
best_gamma = 2^gamma(idx);
```

The point marked on Contour generated must not lie on the edges, if they lie on the edges change the mesh grid suitably

4 DECISION TREES LEARNING (DT)

4.1 INTRODUCTION TO DECISION TREES LEARNING

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

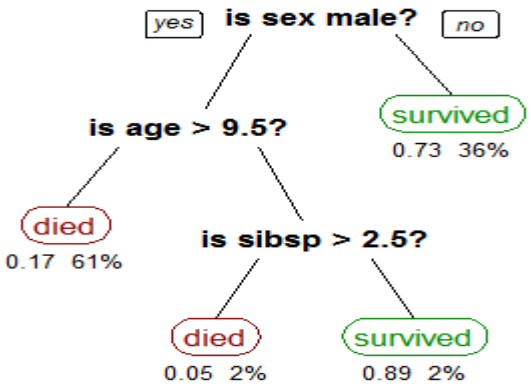

Figure 2: Example of Decision Tree

4.1.1 Some advantages of decision trees are:

- **Simple** to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable.
- Able to handle multi-output problems.
- Uses a **white box model**. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

4.1.2 The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called **overfitting**. Mechanisms such as **pruning** (removing the branches of the tree to improve regularisation), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be **unstable because small variations** in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an **ensemble (Combinations of Trees/Classifiers)**.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the **greedy algorithm where locally optimal decisions** are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners **create biased trees if some classes dominate**. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

4.2 DT IMPLEMENTATION ON MATLAB

This Library can be used to construct Decision trees and perform pruning as per user's requirement. It has interactive GUI and has support for handling Categorical Variables. Level of Importance of the features can also be obtained.

Usage: Data must be Stored into an Excel File (.xls) format in the following order; First row must have the details of the corresponding columns; First column must be the class label followed by the feature.

To start learning from the data use 80% of dataset as Train data and remaining can be used as testdata. The Decision tree must be trained after having a perfect dataset as the structure might change when a new data is added into the training data.

Steps to train DT:

1. >> Tree('Traindata.xls')
2. A GUI will appear for selection of features to be considered. Select the appropriate features and press continue.
3. In This Step, A new GUI will appear to select the features that are categorical (like High, Low, and Medium etc.). Press continue once the selection is done.
4. The Tree will be displayed along with a performance metric (cross validation and re-substitution error) which also suggest the level of pruning for optimal performance and to avoid overfitting of the data. Corresponding ROC for respective classes will also be displayed and area under the curve will give the accuracy.

5. Pruning can be done by entering Y and entering the level of pruning. The Decision tree and corresponding ROC Curve will be updated.
6. Once the Tree has been obtained testing can be done by using a simple if else program written according to the decision tree obtained.

The Code can be downloaded From: Link (Math works Site).

Example :

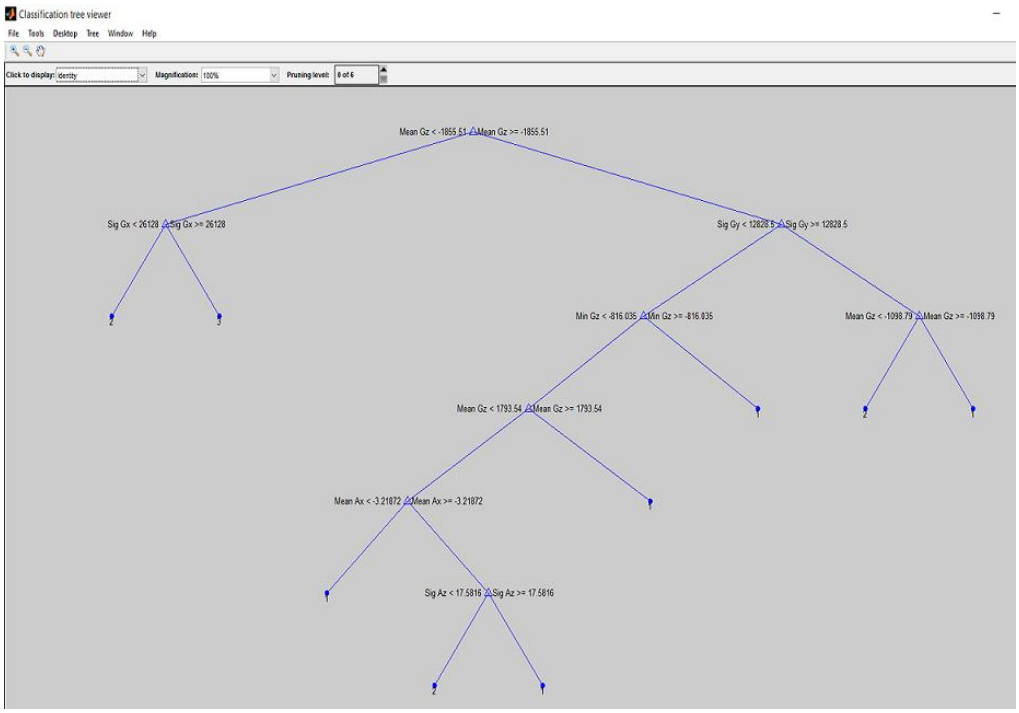

Figure 3 Decision Tree Before Pruning

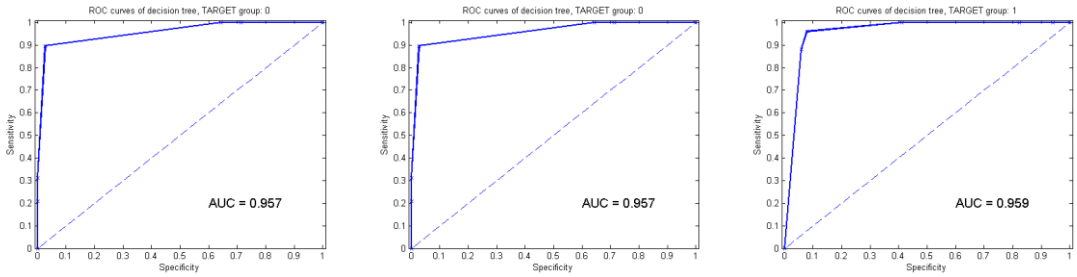

Figure 4 ROC Curves

Functions: Tree('inputfile.xls') – Trains a decision tree based on the input file given with a user friendly GUI interface.

5 RANDOM FOREST (RF)

5.1 INTRODUCTION TO RANDOM FOREST

Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but *with replacement*, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

Forest error rate depends on two things:

- The *correlation* between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The *strength* of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. Using the oob error rate (see below) a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

5.1.1 Features of Random Forests

- It is unexcelled in accuracy among current algorithms.
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.

- Prototypes are computed that give information about the relation between the variables and the classification.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

5.2 RF IMPLEMENTATION ON MATLAB

Library Link: https://codeload.github.com/erogol/Random_Forests/zip

Usage:

```
load train1new.mat
load test1new.mat
train_x=trdata(:,2:end);
train_y=trdata(:,1);
test_x=Testdata(:,2:end);
test_y=Testdata(:,1);
% train_x is nxm matrix where rows are instances and columns are the
variables
% train_y is nx1 matrix where each row is the label of the mathing instance
% For other parameters refer to source code
model = train_RF(train_x, train_y,'ntrees',
100,'oobe','y','nsamtosample',25,'method','c','nvartosample',2);
pred = eval_RF(test_x, model, 'oobe', 'y');
accuracy = cal_accuracy(test_y,pred)
```

The library used seems to be premature and needs lot of development another library that supports different weak learners can be used. Link: https://github.com/karpathy/**Random-Forest-Matlab**

Weak random learner can train

1. Decision stump: look along random dimension of data, choose threshold that maximizes information gain in class labels

2. 2D linear decision learner: same as decision stump but in 2D.

3. Conic section learner: second order learning in 2D. I.e. x*y is a feature in addition to x, y and offset (as in 2.)

4. Distance learner. Picks a data point in train set and a threshold. The label is computed based on distance to the data point.