Receiver Operator Characterisitc (ROC) Curves: Classifier Performance

Table of Contents

Background

Often we have an established reference standard and wish to evaluate a new diagnostic test. For example, the reference standard might be a label of Positive or Negative from a histology slide. The new test might measure the concentration of a blood marker, or an intenisty in an image (the 'score') and it could be cheaper, easier to perform, have fewer side-effects etc. We need to determine the performance of the new test, and be able to set a threshold score that will classify patients as Positive or Negative.

The ROC curve is a mechanism for evaluating the new test. The new test classifies patients by measuring some quantity (the test score) and then choosing a threshold for this score, with all scores on one side of the threshold classed as Positive, and those on the other side as Negative. The ROC curve shows what happens to the classification as the threshold is changed. It is important to realise that the interesting part of the ROC curve is where the test scores overlap for cases classed by the reference as Positive or Negative. This notebook takes a look at what lies behind the ROC curve, how to plot informative figures and examine confidence intervals.

```
% Set up a simulation of data

% Mean test scores for reference-standard Positive cases and Negative cases
meanTestPositive = 20 ;
meanTestNegative = 10 ;

% Standard deviations of these test scores
stdTestPositive = 8 ;
stdTestNegative = 5 ;

% Size of sample
numSubjects = 150 ;

% Prevalence: the percentage of the total in the test set that are positive
prevalencePercent = 20 ;
```

```
% Actual number of subjects Positive and Negative (accounting for rounding).
numPositive = round( prevalencePercent/100 * numSubjects ) ;
numNegative = numSubjects - numPositive ;

% Assign Labels for reference-standard Positive and Negative classes
posClassName = "Positive" ; negClassName = "Negative" ;

% Assign colours for Positive (red) and Negative (blue)
posCol = [1 0 0] ; negCol = [0 0 1] ;
```

Now simulate the test scores.

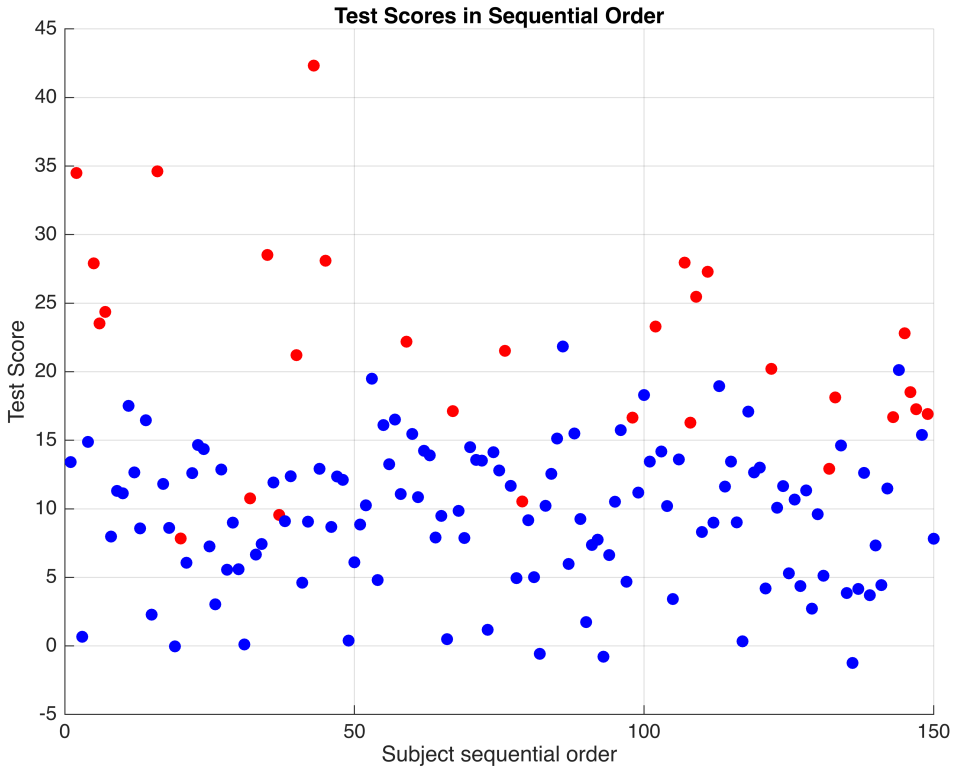
```
% randn draws from a normal distribution with zero mean and std of 1.
% randn([numPositive 1]) returns a column vector with numPositive values
% multiply by required std and add the required mean
scoresOfPositives = ( randn([numPositive 1]) * stdTestPositive ) +
meanTestPositive ;
scoresOfNegatives = ( randn([numNegative 1]) * stdTestNegative ) +
meanTestNegative ;

% Group all the positives then all the negatives, then randomly permute
% to simulate no particular order for the data. Do this for the test
% scores, the reference labels and their associated colours.

p = randperm(numSubjects) ;  % determine a random order of the subjects

scores = cat(1, scoresOfPositives, scoresOfNegatives ) ; % concatenate
scores
scores = scores(p) ; % apply the random order determined above

Labels = cat(1, repmat(posClassName,[numPositive 1]), repmat(negClassName,
[numNegative 1])) ;
Labels = Labels(p) ;

refColours = cat(1, repmat(posCol,[numPositive 1]), repmat(negCol,
[numNegative 1])) ;
refColours = refColours(p,:) ;

locNegative = (Labels == negClassName) ; % The location of the Negatives in
the Labels
locPositive = (Labels == posClassName) ;
```

Visualise Your Data: swarmchart, histogram and boxchart

Before doing any numerical analysis, it is very useful to visualise your data. A simple thing is to just plot the scores using a `scatter` plot.

```
% Plot the values
```

```
figure(Name="Data Points")
scatter(1:numSubjects, scores, [], refColours,"filled")
grid on
xlabel('Subject sequential order'), ylabel('Test Score')
title('Test Scores in Sequential Order')
```



The scatter plot gives a good idea of the means and spreads, but is a bit cluttered. Generally the Positive cases (red) have a higher score, but there is some overlap. If there was a problem with the data acquisition, it might be evident from a plot like this.

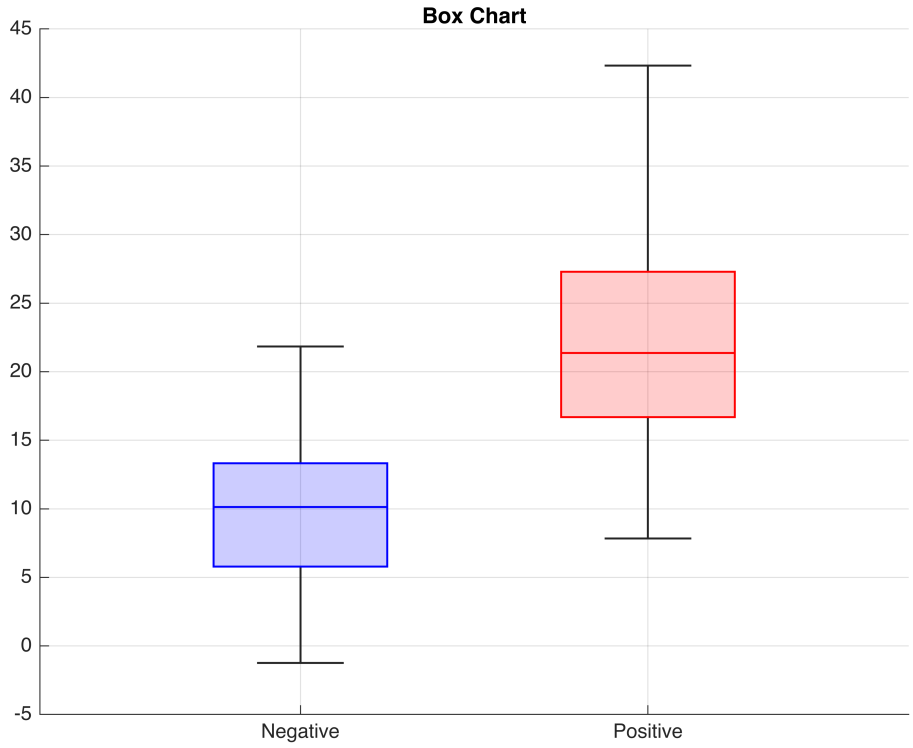
Alternativly, use a `swarmchart`. This plots the data points offset (jittered) in the x-direction to indicate the local denisty of points. It is similar to a violin plot except that it shows the actual data points. Ideally we want the Test Scores for all the red Positives to be well above the scores for the blue Negatives. If that were the case, we could choose a Test Score value between the two groups as the threshold operating point. All scores above would be correctly classed by the test as positives, and all below would be negatives. In reality, there is usually overlap and the `swarmchart` visualises how many points are in the overlapping region and gives a visual a guide to the distributions and amount of data.

```
figure(Name='Test Scores')
catLabels = categorical(Labels) ; % The Labels as a variable of type
categorical
swarmchart(catLabels, scores, [], refColours, 'filled', ...
```

```
    'MarkerFaceAlpha',0.5,'MarkerEdgeAlpha',0.5, 'XJitter','density',
'XJitterWidth',0.3)
title('Test Scores Grouped by Reference Class')
ylabel('Test score'), xlabel('Class label from reference standard')
grid on
```

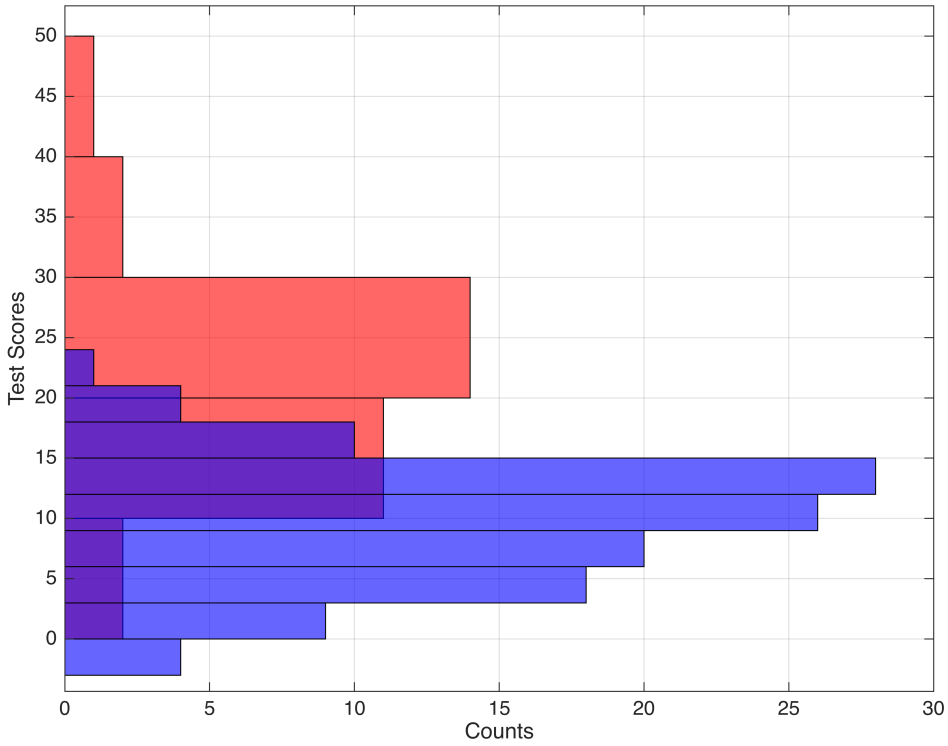
It is also common to see box and whisker plots in scientific papers. These have the advantage of showing values such as median and interquartile range. They sometimes don't provide a very good representation of the overlap region, compared to the `swarmchart` above, or histograms below.

```
figure(Name="Box Chart")
b = boxchart(catLabels(locNegative),scores(locNegative));
b.BoxFaceColor = negCol ;

hold on, grid on
b = boxchart(catLabels(locPositive),scores(locPositive)) ;
b.BoxFaceColor = posCol ;

title("Box Chart")
```



Finally, let's visualise histograms of the data. These show the shapes of the distributions of data points in the sample. Again, we can see the overlap region. The `swarmchart` has the advantage of showing individual data points and not bins, but the jittering in the x direction can be hard to interpret numerically. If we had a lot of data points, a histogram might be a better representation than a `swarmchart`.

```
% Show horizontal histograms
figure(Name="Histograms")
hPositive =
histogram(scoresOfPositives,'Orientation','horizontal','FaceColor',posCol,'F
aceAlpha',0.6) ;
hold on, grid on
hNegative =
histogram(scoresOfNegatives,'Orientation','horizontal','FaceColor',negCol,'F
aceAlpha',0.6) ;
ylabel('Test Scores'), xlabel('Counts')
```

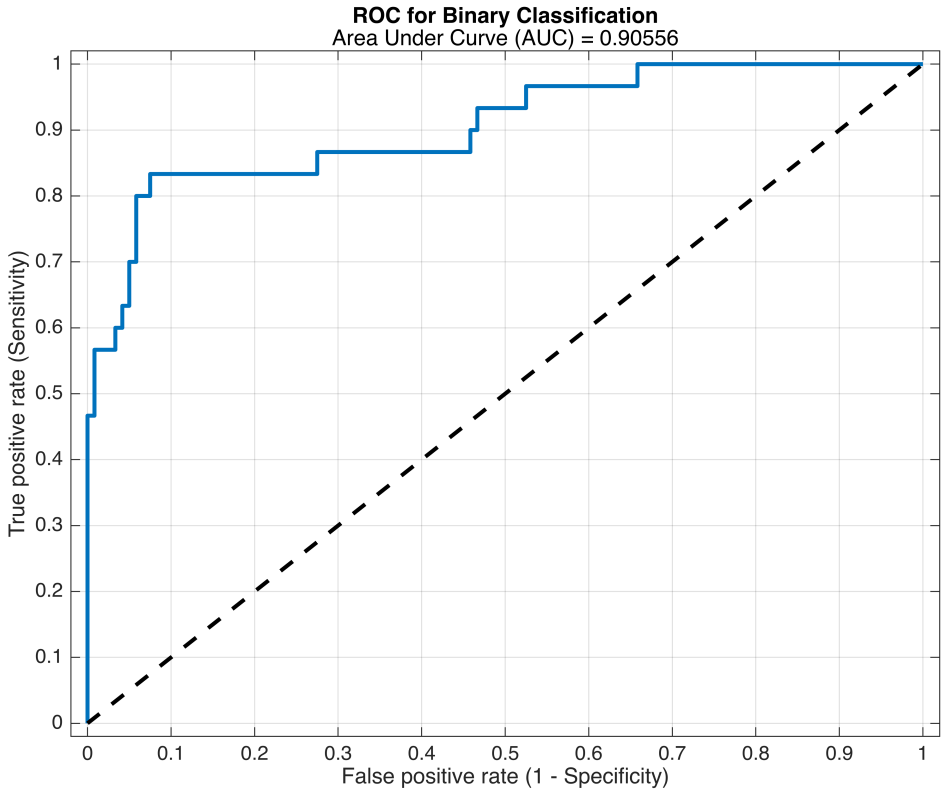


ROC Curve Plotting

From the previous figures, we can see that there is an overlap region where subjects with the same test score can be either reference-standard Positive or Negative. It is this overlap region that is of interest. The performance of the test at a set score ('threshold' or 'operating point') can be characterised by its Sensitiviy and Specificity. The way these change with the threshold can be seen in an ROC curve.

To plot an ROC curve, we can use `perfcurve` or `rocmetrics`.

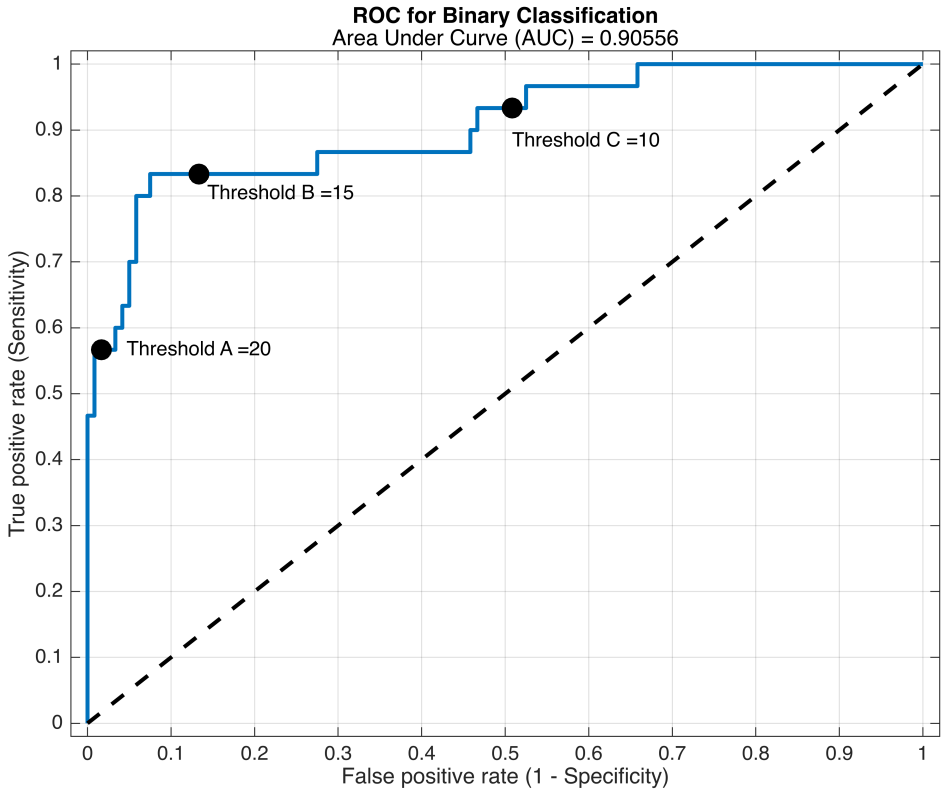
```
[X,Y,T,AUC,OPTROCPT] = perfcurve(Labels, scores, posClassName);
figure(Name="ROC curve")
plot(X,Y, LineWidth=2), hold on, grid on
plot([0 1],[0 1],'k--', LineWidth=2)
xlim([-0.02 1.02]), ylim([-0.02 1.02])
xlabel('False positive rate (1 - Specificity)')
ylabel('True positive rate (Sensitivity)')
title("ROC for Binary Classification","Area Under Curve (AUC) = "+AUC)
```



Add markers at three different thresholds to highlight that the curved part of the ROC curve depends only on the thresholds where test scores from the Positive and Negative classes overlap. If there is a small number of points in that region, the ROC curve will have obvious steps,

```
thresholdA = meanTestPositive  ;
thresholdB = ( meanTestNegative + meanTestPositive ) / 2 ;
thresholdC = meanTestNegative  ;

% Find locations of these thresholds on the ROC curve
[~, iThreshA] = min(abs(T-thresholdA)) ; % location of threshold closest to
thresholdA
[~, iThreshB] = min(abs(T-thresholdB)) ;
[~, iThreshC] = min(abs(T-thresholdC)) ;

markerArgs = { 'MarkerSize',10, 'MarkerEdgeColor','k',
'MarkerFaceColor','k'} ;
plot(X(iThreshA(1)), Y(iThreshA(1)),'ko', markerArgs{:})
text(X(iThreshA(1))+0.03, Y(iThreshA(1)), ['Threshold A
=',num2str(thresholdA)])

plot(X(iThreshB(1)), Y(iThreshB(1)),'go',  markerArgs{:})
text(X(iThreshB(1))+0.01, Y(iThreshB(1))-0.03, ['Threshold B
=',num2str(thresholdB)])
```

```
plot(X(iThreshC(1)), Y(iThreshC(1)),'ro',  markerArgs{:})
text(X(iThreshC(1)), Y(iThreshC(1))-0.05, ['Threshold C
=',num2str(thresholdC)])
```



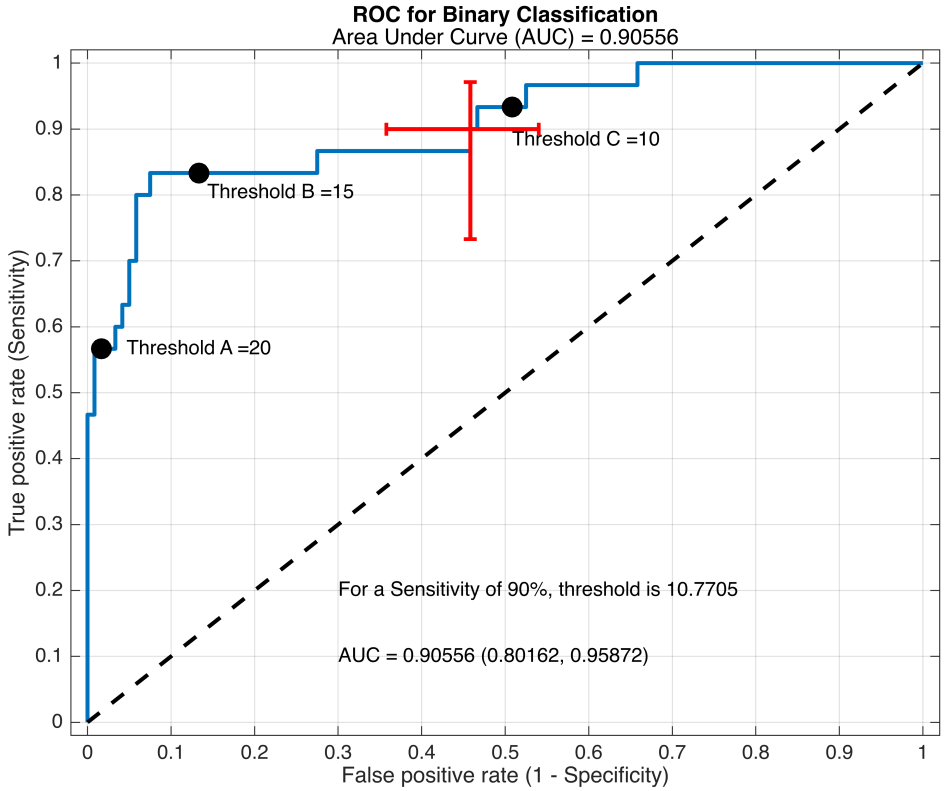
Point Confidence Interval

This simulation script can be re-run many times to give an indication of the variability in the curve shape. In reality, we measure only one sample of limited size. A method to estimate confidence intervals is bootstrapping where an analysis is run multiple times, leaving out different permutations of data.

Typically a threshold (operting point) is chosen and we can assess our confidence in the Sensitvity and Specificity at that point. In this example, we pick the threshold that gives a Sensitivity closest to 90% as the operating point (i.e. 9 out of 10 positive cases are correctly identified). In practice, the choice of operating point depends upon the risks and costs of under/over diagnosis/treatment.

```
% Use Bootstrapping in perfcurve. Note Xb,Yb and AUCb outputs have 3
columns -
% the value, its lower limit and its upper limit.
[Xb,Yb,Tb,AUCb] = perfcurve(Labels,scores,posClassName,'NBoot', 1000);

% Put text information on graph
text(0.3,0.1,"AUC = "+AUCb(1)+" ("+AUCb(2)+", "+AUCb(3)+")")
```

```
% At a pre-determined Sensitivity (y value) of 90%
yPreset = 0.9 ;
[~,itpr] = min(abs( Yb(:,1) - yPreset)) ; % itpr is index into Yb, the true
positive rate

ind = itpr(1) ; % In case there is more than one point equi-distant from
yPreset

text(0.3,0.2,"For a Sensitivity of "+Yb(ind,1)*100 +"%, threshold is
"+Tb(ind))

% Plot the confidence intervals using errorbar(x,y,yneg,ypos,xneg,xpos)
eb = errorbar(Xb(ind,1), Yb(ind,1), Yb(ind,1)-Yb(ind,2), Yb(ind,3)-
Yb(ind,1), ...
                                    Xb(ind,1)-Xb(ind,2), Xb(ind,3)-
Xb(ind,1) ) ;

eb.LineWidth = 2 ; eb.Color = [1 0 0] ;
```



2x2 Truth Table

This shows a 2x2 truth table. Note that if using the MATLAB `confusion` matrix, the axes are transposed.

```
% Display a 2x2 truth table and calculate Sensitivity and Specificty at a
set
% threshold
```

```
TTPR90 = Tb(ind) ; % Threshold at which Sensitivity (TPR) is 90% (see above)

% isTP etc are arrays with 0 where the expression is false, and 1 where true
isTP = scores >= TTPR90 & Labels==posClassName ; % True Positives
isTN = scores <  TTPR90 & Labels==negClassName ; % True Negatives
isFP = scores >= TTPR90 & Labels==negClassName ; % False Positives
isFN = scores <  TTPR90 & Labels==posClassName ; % False Negatives

TP = sum(isTP) ; TN = sum(isTN) ; FP = sum(isFP) ; FN = sum(isFN) ;

% Table uses variable names to label the columns and rows.
RowNames = {'Test_Positive','Test_Negative'} ;
Reference_Positive = [ TP ; FN];
Reference_Negative = [ FP ; TN] ;
T2x2 = table(Reference_Positive, Reference_Negative ,'RowNames',RowNames)
```

T2x2 = 2×2 table

	Reference_Positive	Reference_Negative
1 Test_Positive	27	55
2 Test_Negative	3	65

```
disp("Sensitivity: TPR = TP/(TP+FN) = " + TP + " / ( " + TP +" + " + FN +
" ) = " + TP/(TP+FN))
```

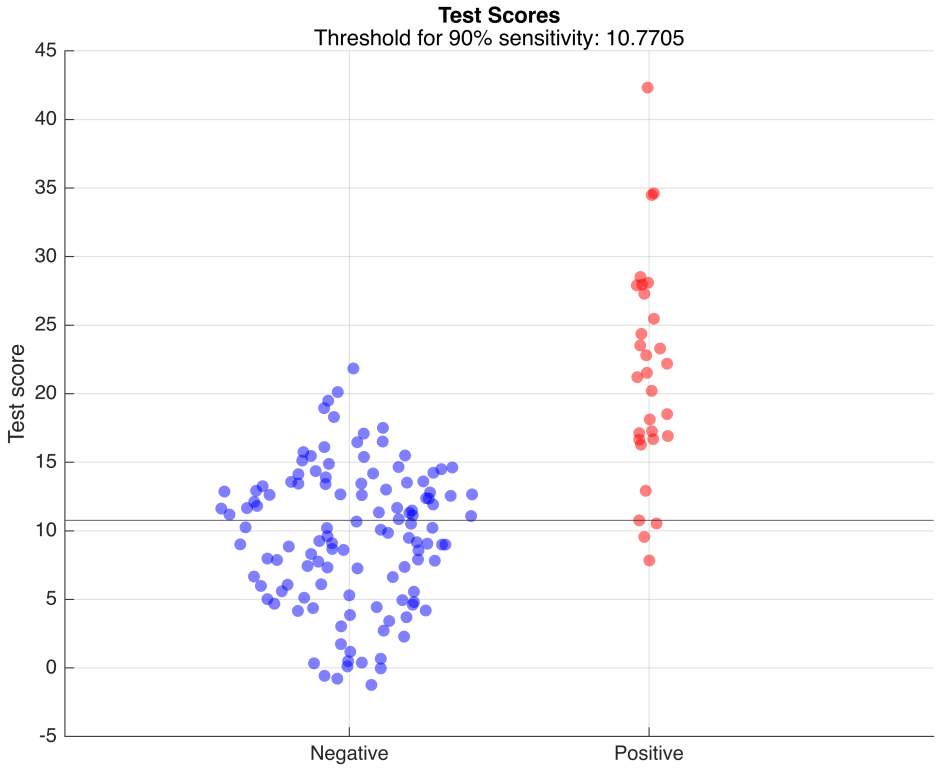
```
Sensitivity: TPR = TP/(TP+FN) = 27 / ( 27 + 3 ) = 0.9
```

```
disp("Specificity: TNR = 1-FPR = TN/(TN+FP) = " + TN + " / ( " + TN +" + "
+ FP + " ) = " + TN/(TN+FP))
```

```
Specificity: TNR = 1-FPR = TN/(TN+FP) = 65 / ( 65 + 55 ) = 0.54167
```

Re-plot the swarmchart with the above threshold as a horizontal line.

```
figure(Name='swarmchart Test Scores')
swarmchart(catLabels, scores, [], refColours,
'filled','MarkerFaceAlpha',0.5,'MarkerEdgeAlpha',0.5)
title("Test Scores", "Threshold for 90% sensitivity: "+TTPR90)
ylabel('Test score')
grid on

yl = yline(TTPR90) ;
```



Note, because we specified a high sensitivity of 90%, the threshold is "forced" down so that most of the red (Positive) points are above the threshold. Red points below the line are False Negatives, blue points above the line are False Positives.

Additional Comments

Scores that *decrease* with increasly more likely Positive Class

This demonstration assumed that as the test score increases, it is more likely that a subject is Positive. If the the test score decreases with increasingly more likely positive cases, we need to make adjustments. An example of decreasing scores is in MRI diffusion Apparent Diffusion Coefficient (ADC) measures where darker pixels indicate a greater chance of cancer. To cope with this, we pass the negative of the scores, i.e. `-scores` into `perfcurve` or `rocmetrics`. For either of these functions, you will need to negate any threshold values output by the function (if you are using a datatip, see below).

rocmetrics

The `rocmetrics` function can also be used to generate ROC curves from either binary or multiclass data. It can also return quantities at user-provided fixed metric values. For example,

```
rocObj = rocmetrics(Labels, scores, posClassName, "AdditionalMetrics", ...
    ["TP","TN","FP","FN"],"FixedMetric","tpr","FixedMetricValues",0.9) ;
```

```
disp("90% Sensitivity threshold here is: " + rocObj.Metrics.Threshold + "
with Specificity of " + ...
    (1-rocObj.Metrics.FalsePositiveRate))
```

```
90% Sensitivity threshold here is: 10.6699 with Specificity of 0.53333
```

```
disp("TP = " + rocObj.Metrics.TruePositives + ", TN = " +
rocObj.Metrics.TrueNegatives)
```

```
TP = 27, TN = 64
```

Note the numbers can differ a bit from the previous determination due to where the threshold is placed between data points.

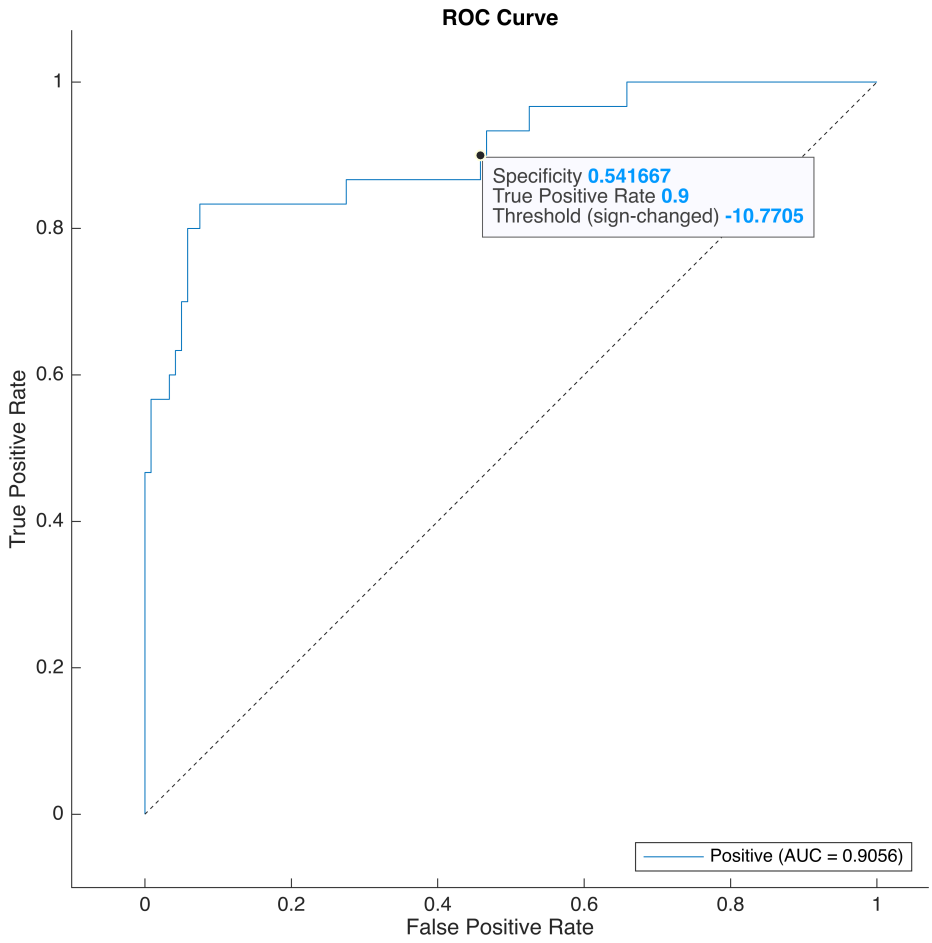
`rocmetrics` with `scores` passed in as a vector (as here) expects `scores` to represent probabilites and sets the threshold to 0.5, leading to a non-sense operating point when the scores are not probabilites. You can turn off the operating point using `ShowModelOperatingPoint=false` when plotting.

```
rocObj = rocmetrics(Labels, scores, posClassName) ;
figure(Name="rocmetrics plot")
[curveObj, graphicsObjs] = plot(rocObj, ShowModelOperatingPoint=false) ;
```

DataTip

`rocmetrics` also sets a useful datatip. The user can change the datatip template, for example

```
% Reset datatip first row from default of False Positive Rate to
Specificity (1-FPR)
curveObj.DataTipTemplate.DataTipRows(1).Label = 'Specificity' ;
curveObj.DataTipTemplate.DataTipRows(1).Value = @(FPR)(1-FPR) ;

% If -scores was passed in, set UserData to negative of internal Thresholds
and then
% display this UserData so that the user sees sign-changed threshold values.
curveObj.UserData = -curveObj.Thresholds ;
curveObj.DataTipTemplate.DataTipRows(3).Label = 'Threshold (sign-changed)' ;
curveObj.DataTipTemplate.DataTipRows(3).Value = 'UserData' ;

dt = datatip(curveObj, 'DataIndex', ind, 'Location', 'southeast' ) ;
```

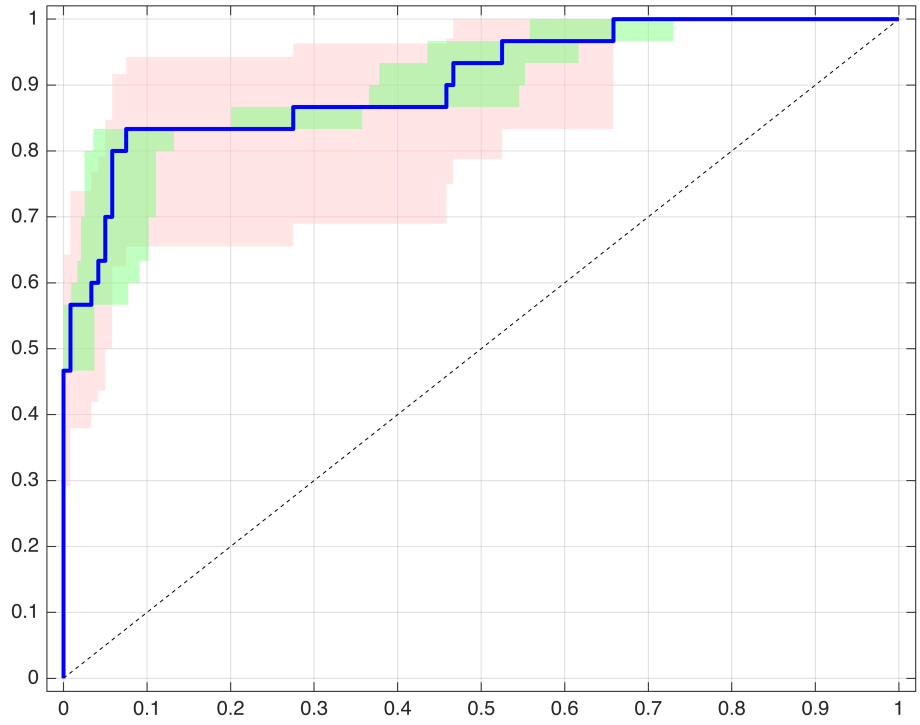


Confidence Interval Plotting

In the `perfcurve` example above we showed how to plot a point confidence interval. `rocmetrics` can plot TPR confidence intervals but we may wish to also have the FPR confidence intervals. Both `perfcurve` and `rocmetrics` can compute the intervals using bootstrapping and these can then be plotted manually. Note that I think these can be hard to see clearly and the errorbar red cross in the example above is probably more useful.

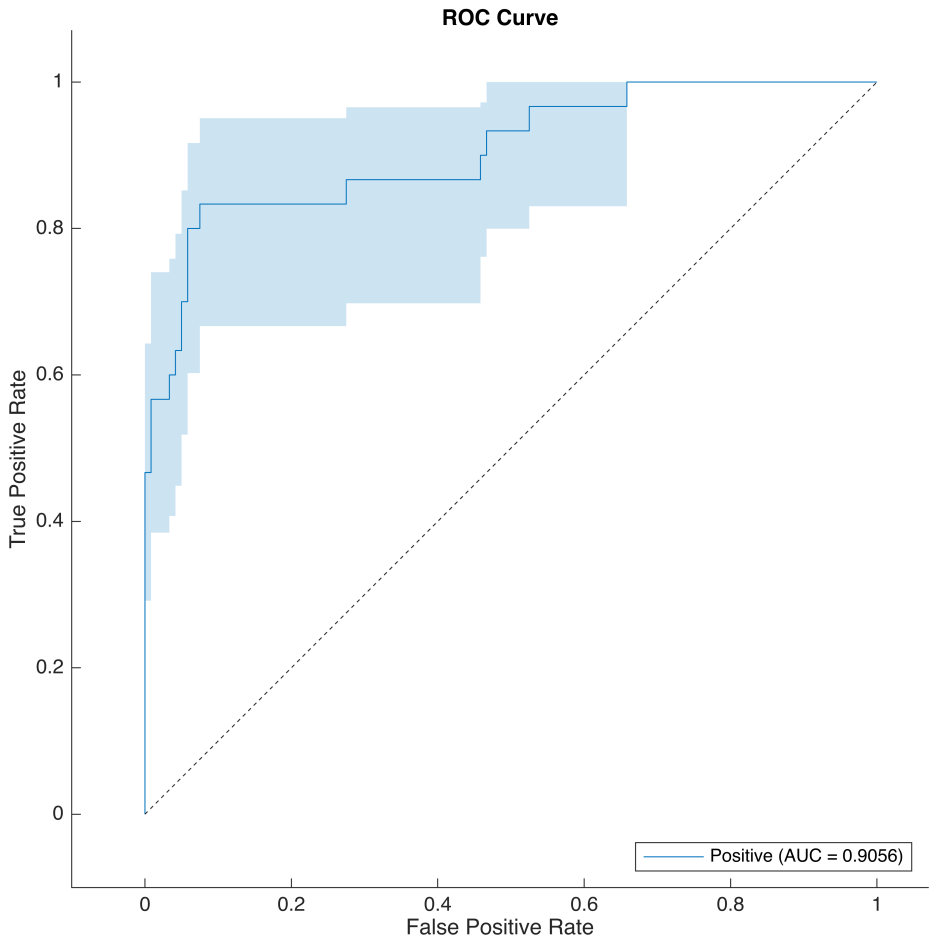
```
[Xb,Yb,Tb,AUCb] = perfcurve(Labels,scores,posClassName,'NBoot', 1000);
figure
plot(Xb(:,1),Yb(:,1),'LineWidth',2)
hold on, grid on
plot([0 1],[0 1],'k--')
xlim([-0.02 1.02]), ylim([-0.02 1.02])

% TPR Confidence
xconf = [Xb(:,1)' Xb(end:-1:1,1)'] ;
yconf = [Yb(:,2)' Yb(end:-1:1,3)'] ;
```

```
hf = fill(xconf, yconf, 'red') ;
hf.FaceColor = [1 0.8 0.8] ;
hf.EdgeColor = 'none' ;
hf.FaceAlpha = 0.5 ;

% FPR Confidence
xconf = [Xb(:,2)' Xb(end:-1:1,3)'] ;
yconf = [Yb(:,1)' Yb(end:-1:1,1)'] ;
hf = fill(xconf, yconf, 'green') ;
hf.FaceColor = [0.5 1 0.5] ;
hf.EdgeColor = 'none' ;
hf.FaceAlpha = 0.5 ;

plot(Xb(:,1),Yb(:,1),'b','LineWidth',2)
```



The TPR confidence intervals can be plotted easily using `rocmetrics`:

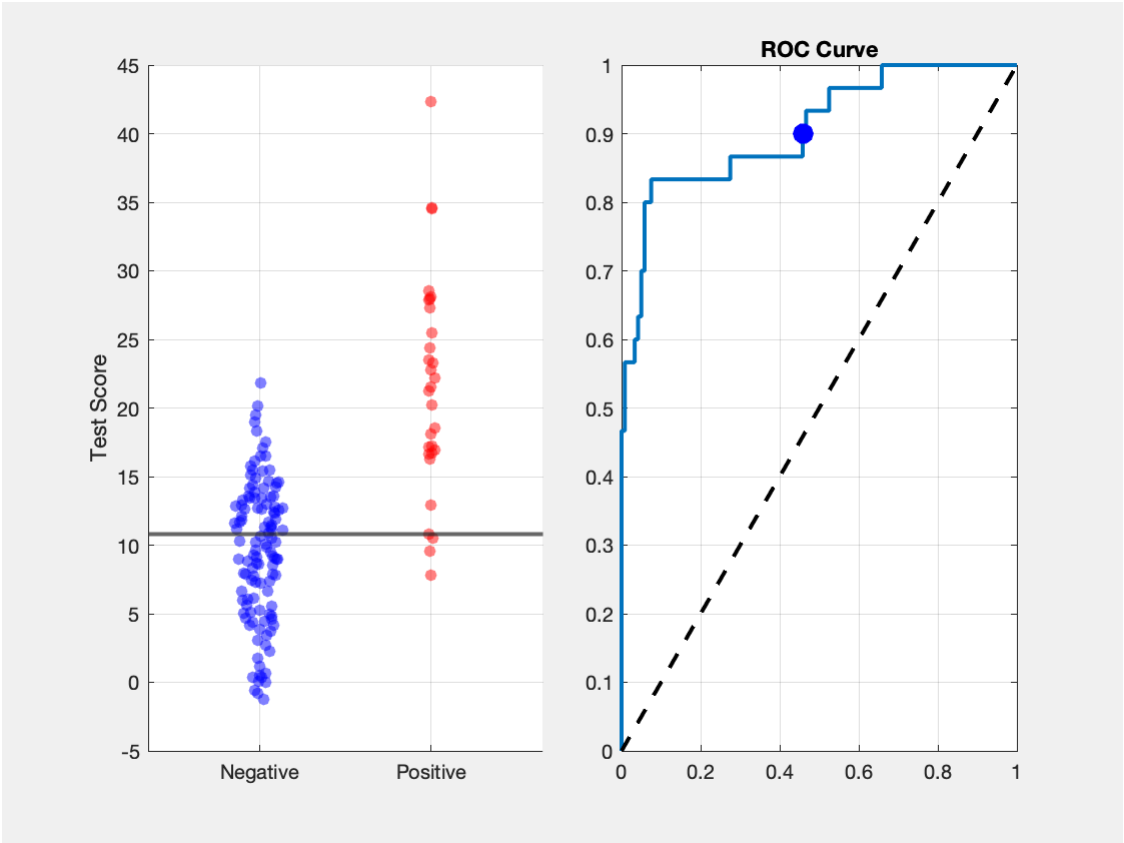
```
rocObj = rocmetrics(Labels, scores, posClassName, NumBootstraps=1000) ;
figure
plot(rocObj,ShowConfidenceIntervals=true, ShowModelOperatingPoint=false)
```



Animation of Threshold Changes

```
hf = figure(Name="animation");
set(hf,'Visible','on') % Force out of Live Script
tiledlayout("horizontal","TileSpacing","compact")
axSwarm = nexttile;

swarmchart(axSwarm, catLabels, scores, [], refColours, 'filled', ...
    'MarkerFaceAlpha',0.5,'MarkerEdgeAlpha',0.5, 'XJitter','density',
'XJitterWidth',0.3)

yl = yline(axSwarm,T(1),'LineWidth',2) ;
grid on
ylabel('Test Score')

axROC = nexttile ;
plot(X, Y,'LineWidth',2)
hold(axROC,"on"), grid(axROC,"on")
plot([0 1],[0 1],'k--','LineWidth',2)
hp = plot(X(1),Y(1),'bo','MarkerSize',10,'MarkerFaceColor','b') ;
```

```
title('ROC Curve')

for ithresh = 1:length(T)
    yl.Value = T(ithresh) ;
    hp.XData = X(ithresh) ;
    hp.YData = Y(ithresh) ;
    drawnow
    pause(0.1)
end

% Leave on the point ind (90% sens, determined earlier)
yl.Value = T(ind) ;
hp.XData = X(ind) ;
hp.YData = Y(ind) ;
```



Further examples used for figures in the presentation (including making an animated gif) are in the folder `figures/ROC`.

David Atkinson, 2023-2025.