PyROC

A Python Package for Binary Classifier Evaluation

Version 1.0

May 2, 2008

Nikhil Ketkar

nikhilketkar@mac.com

Table of Contents

1. Introduction

This document is intended to serve both as a tutorial and a reference manual for PyROC which is a Python package for binary classifier evaluation. The motivation behind PyROC is to provide a powerful and flexible framework for binary classifier evaluation.

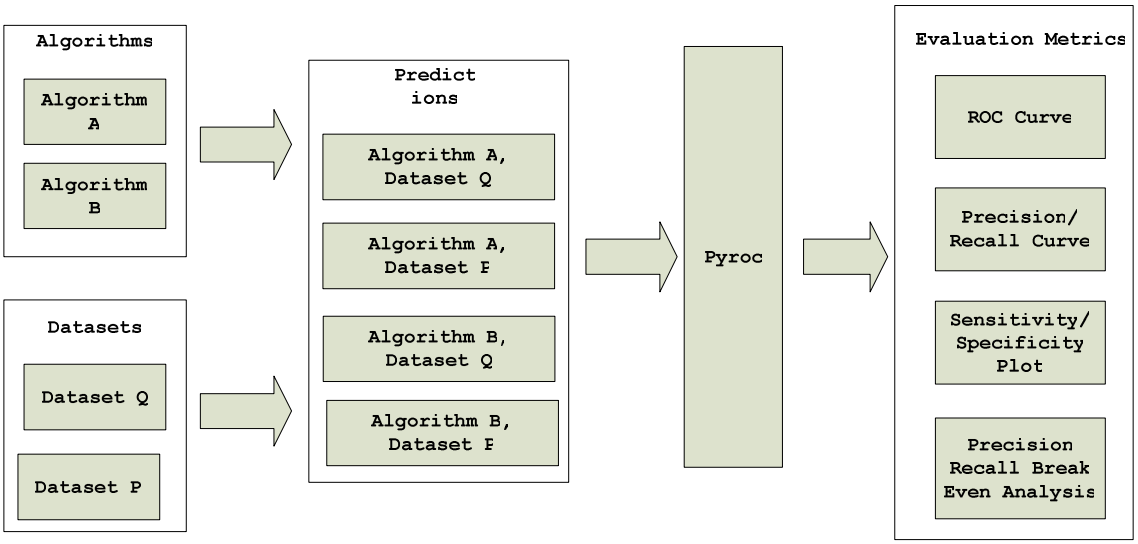


Figure 1. Binary Classifier Evaluation with PyROC

Researchers in machine learning, data mining and bioinformatics often have to compare the performance of binary classification algorithms, the metrics for which have become increasingly sophisticated over the period of years. The use of accuracy as a metric for comparing classifier performance has been found to be biased to class skew and unequal misclassification costs. A number of new metrics such as ROC curves, Precision/Recall curves, Sensitivity/Specificity plots and Lift charts have been introduced in literature as metrics for binary classification which give an unbiased estimate of the performance. Computing these metrics becomes a non-trivial task and software to perform such analysis often has to be written by researchers at a significant cost in terms of time and effort. The

goal of PyROC is to address this need by providing a general purpose framework for evaluating binary classification algorithms as illustrated in Figure 1. PyROC implements all the major threshold based metrics introduced in literature which can be combined and plotted to provide for a versatile analysis of binary classification algorithms.

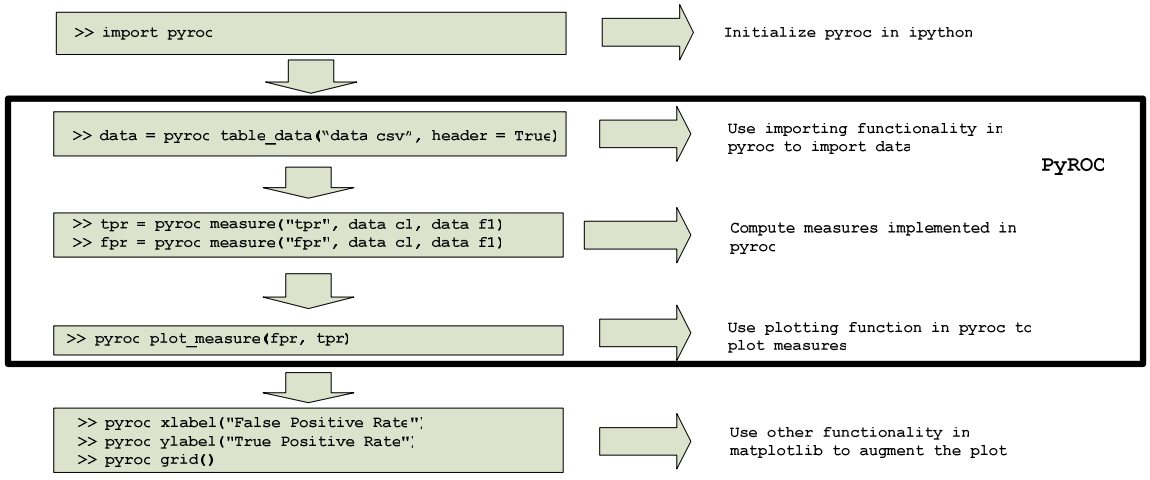
2. Installation

Pyroc requires that you have ipython and matplotlib installed. Once this is done perform the following steps.

1. Unzip the folder `pyroc.zip`. This should give you a folder `pyroc`.
2. Open a terminal and traverse (change directory) to inside the `pyroc` folder.
3. Start ipython with the pylab option `ipyton –pylab`
4. At the ipython prompt type `import pyroc`
5. You should now be able to use pyroc and all the examples in this document. The files simple.csv and data.csv have been provided (in the folder) to run the examples in this document.

Overall Usage

PyROC is designed to seamlessly integrate with ipython and matplotlib providing an interactive command line interface for analysis like MATLAB or R. Figure 2. illustrates a PyROC session.



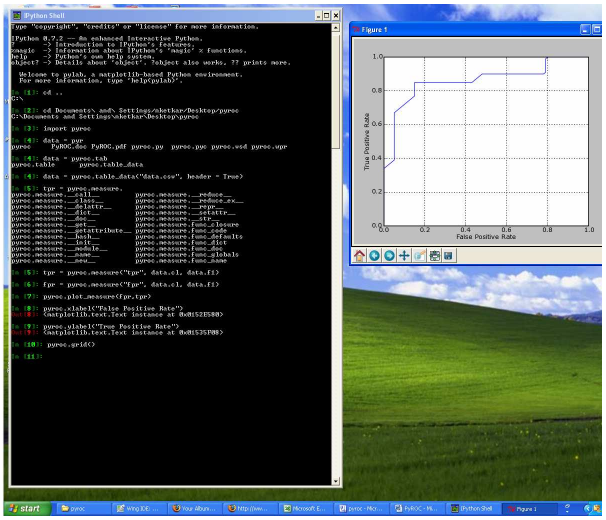


Figure 2. A typical PyROC session

4. Reading Data

One of the important steps in performing in performance analysis is reading the predictions and class labels into memory. PyROC provides a very easy and intuitive mechanism for this process which we describe with an example. Let us say that we have a file called “test.csv” with data as illustrated in Figure 2. Note that this is a typical comma delimited file with 5 columns and 5 rows, the first row being the names of the columns.

```
a, b, c, d, e
1, 2, 3, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 5
```

Figure 3. Contents of the file “test.csv”

In order to read this data into memory, we issue the command illustrated in Figure 3. The file has now been read in the memory and can be accessed. We can access the names of the columns and their individual values as illustrated in Figure 3. It is important to note that the names of the columns in the file become the attributes of the object.

```
a, b, c, d, e
1, 2, 3, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 5
1, 2, 3, 4, 5
```

```
>> t = pyroc.table_data("simple.csv", header = True)
>> t.columns
     [a,b,c,d,e]
>> t.a
    [1,1,1,1,1]
>> t.b
    [2,2,2,2,2]
```

Figure 4. Accessing the file “test.csv”

It is possible in some cases that the delimiter may be something other that a comma. Also, it is sometimes possible that the file does not have column names as the first row. In these cases we can specify the correct options and get the desired result as illustrated in Figure 4. Note that if the column names are absent, PyROC introduces names for them.

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

```
>> t = pyroc.table_data("simple.txt", delimiter = " ")
>> t.columns
     [C0, C1, C2, C3, C4]
>> t.C0
    [1,1,1,1,1]
>> t.C1
    [2,2,2,2,2]
```

Figure 5. Accessing files with other delimiters and no headers.

5. Measures

PyROC implements a number of threshold based measures that can be used to perform classifier evaluation. Computing a specific measure involves executing the function "measure" as illustrated in Figure 5.

```
>> tpr = pyroc.measure("tpr",labels, predictions)
```

Figure 6. Computing threshold measures.

Here, "tpr" referes to the specific measure in question, in this case it is the True Positive Rate. The labels correspond to the list of class labels, predictions refer to the list of corresponding predictions and threshold_count refers to the number of thresholds.

The threshold metrics implemented in PyROC and their abbreviations to be used in calling the measure function are illustrated in Table 1.

Table 1. Threshold measures implemented in PyROC

No.	Measure	Abbreviation
1	Accuracy	acc
2	Error	err
3	True Positive Rate	tpr
4	Sensitivity	sens
5	False Positive Rate	fpr
6	Fallout	fall
7	Recall	rec
8	False Negative Rate	fnr
9	Miss	miss
10	True Negative Rate	tnr
11	Specificity	spec
12	Positive Predictive Value	ppv
13	Precision	prec
14	Negative Predictive Value	nvp
15	Prediction Conditioned Fallout	pcfall
16	Prediction Conditioned Miss	pcmiss
17	Rate of Positive Predictions	rpp
18	Rate of Negative Predictions	rnp
19	Phi Correlation Coefficient	phi
20	Matthew's Correlation Coefficient	mat
21	Odds Ratio	odds
22	Hit	hit
22	Lift	lift

6. Averaging

Typically, during performance evaluation of any machine learning algorithm, results are collected not over a single run but over a set of runs. The set of runs may be either a k-fold cross validation or a bootstrap. In order to analyze such results, it is necessary to combine the performance measures on each run. PyROC provides for such analysis by implementing functionality to combine threshold based metrics over multiple runs. Figure 5 illustrates how this functionality can be accessed by the user.

```
>> tpr_mean = pyroc.combine("mean", tpr1, tpr2 ...)
```

Figure 7. Combining threshold measures.

The function "combine" accepts as input the particular way in which to combine the performance measure and two or more performance objects and returns a performance object that can be used to plot the result of combining the measure over multiple runs. The various ways in which performance measures can be combined and the corresponding parameters to be used are illustrated in Table 1.

Table 2. Combining Measures in PyROC

No.	Combination	Abbreviation
1	Sum	sum
2	Mean	mean
3	Variance	var
4	Standard Deviation	stddev
5	Standard Error	stderr

7. Typical Use Cases

In this section, we illustrate a few typical use cases for PyROC. These use cases are by no means comprehensive as PyROC implements a total of 22 threshold based metrics which can be freely combined and plotted.

7.1 Accuracy/Error/TPR/FPR on Threshold

A typical use case is the visualization of Accuracy/Error/TPR/FPR with respect to the threshold. This visualization is generally carried out for the purposes of calibrating the model learned by the classification algorithm. Figures 8, 9,10 and 11 illustrate the code and the generated plots for visualizing the Accuracy, Error, TPR and the FPR versus the threshold.

It is important to note here that although we illustrate the procedure for each case from scratch which takes about 6-7 lines of code, the process of loading PyROC and the data are common to each process and if a user is conducting such an analysis there is an overlap between these operations. Typically, any plot takes about 3-4 lines of code.

```
>> import pyroc
>> data = pyroc.table_data("data.csv", header = True)
>> thresh = pyroc.generate_thresholds()
>> acc = pyroc.measure("acc", data.cl, data.f1)
>> pyroc.plot_measure(thresh, acc)
>> pyroc.xlabel("Threshold")
>> pyroc.ylabel("Accuracy")
>> pyroc.grid()
# Use pyroc.cla() to clear.
```

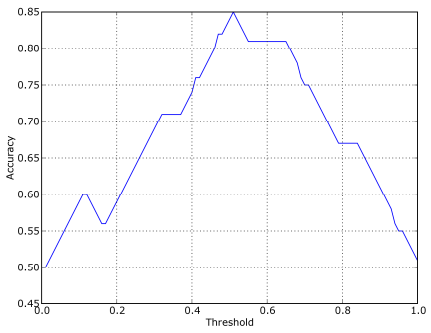


Figure 8. Visualizing Accuracy vs. Threshold

Close observation of each of these illustrations will allow the user to get familiar with the overall process of analysis and plotting. The key steps are generating the thresholds and the particular measure (Accuracy, Error, TPR and the FPR) after which they are plotted and the plot is augmented with the axis labels and the grid.

```
>> import pyroc
>> data = pyroc.table_data("data.csv", header = True)
>> thresh = pyroc.generate_thresholds()
>> err = pyroc.measure("err", data.cl, data.f1)
>> pyroc.plot_measure(thresh, err)
>> pyroc.xlabel("Threshold")
>> pyroc.ylabel("Error")
>> pyroc.grid()
```

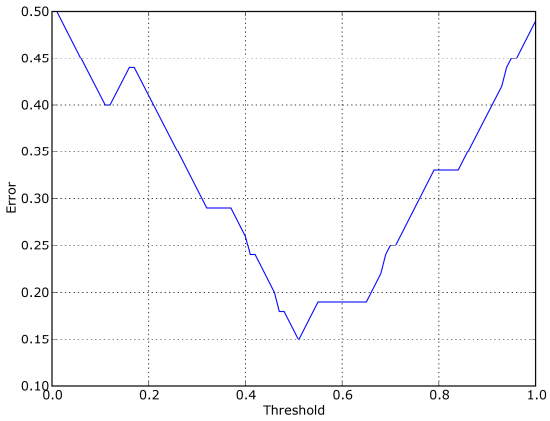


Figure 9. Visualizing Error vs. Threshold

```
>> import pyroc
>> data = pyroc.table_data("data.csv", header = True)
>> thresh = pyroc.generate_thresholds()
>> tpr = pyroc.measure("tpr", data.cl, data.f1)
>> pyroc.plot_measure(thresh, tpr)
>> pyroc.xlabel("Threshold")
>> pyroc.ylabel("True Positive Rate")
>> pyroc.grid()
```

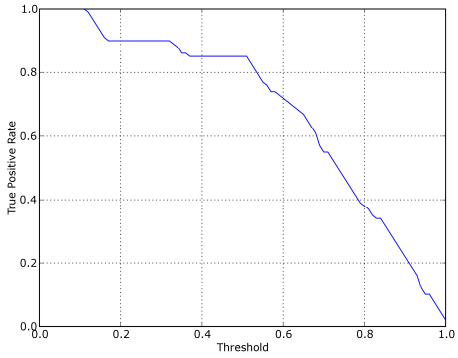


Figure 10. Visualizing True Positive Rate vs. Threshold

```
>> import pyroc
>> data = pyroc.table_data("data.csv", header = True)
>> thresh = pyroc.generate_thresholds()
>> fpr = pyroc.measure("fpr", data.cl, data.f1)
>> pyroc.plot_measure(thresh, fpr)
>> pyroc.xlabel("Threshold")
>> pyroc.ylabel("False Positive Rate")
>> pyroc.grid()
```

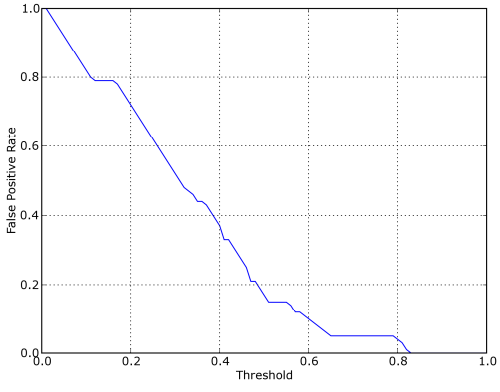


Figure 11. Visualizing False Positive Rate vs. Threshold

7.2 ROC Curve

The ROC curve is perhaps the most popular visualization of performance. The procedure for visualizing the ROC curve is similar to the previously discussed cases except that in the case of the ROC curves, TPR and FPR are plotted against each other.

```
>> import pyroc
>> data = pyroc.table_data("data.csv", header = True)
>> tpr = pyroc.measure("tpr", data.cl, data.f1)
>> fpr = pyroc.measure("fpr", data.cl, data.f1)
>> pyroc.plot_measure(fpr, tpr)
>> pyroc.xlabel("False Positive Rate")
>> pyroc.ylabel("True Positive Rate")
>> pyroc.grid()
```

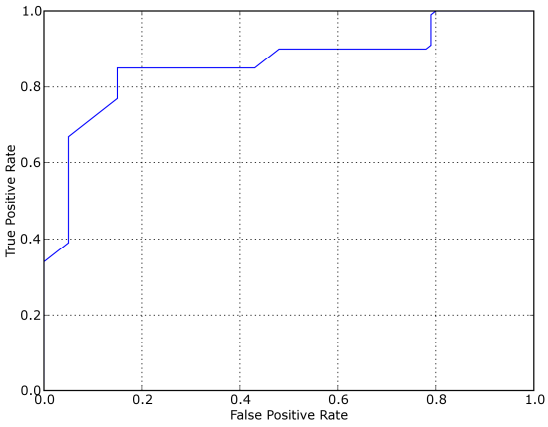


Figure 12. Visualizing ROC Curve

7.3 Precision/Recall Curve

The Precision/Recall curve is another popular visualization of performance. The procedure for visualizing the ROC curve is similar to the previously discussed cases except that in the case of the Precision/Recall curves, Precision and Recall are plotted against each other.

```
>> import pyroc
>> data = pyroc.table_data("data.csv", header = True)
>> prec = pyroc.measure("prec", data.cl, data.f1)
>> rec = pyroc.measure("rec", data.cl, data.f1)
>> pyroc.plot_measure(rec, prec)
>> pyroc.xlabel("Recall")
>> pyroc.ylabel("Precision")
>> pyroc.grid()
```

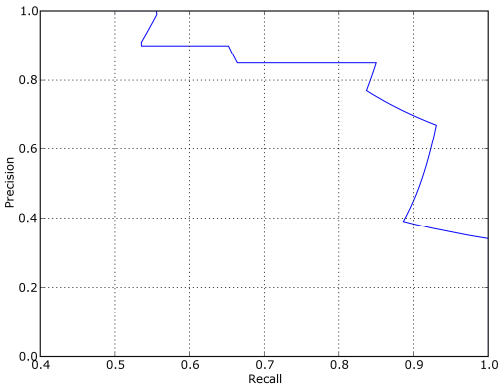


Figure 13. Visualizing Precision/Recall Curve

7.4 Sensitivity/Specificity Plot

The Sensitivity/Specificity curve is another popular visualization of performance. The procedure for visualizing the ROC curve is similar to the previously discussed cases except that in the case of the Sensitivity/Specificity curves, Sensitivity and Specificity are plotted against each other.

```
>> import pyroc
>> data = pyroc.table_data("data.csv", header = True)
>> sens = pyroc.measure("sens", data.cl, data.f1)
>> spec = pyroc.measure("spec", data.cl, data.f1)
>> pyroc.plot_measure(spec, sens)
>> pyroc.xlabel("Specificity")
>> pyroc.ylabel("Sensitivity")
>> pyroc.grid()
```

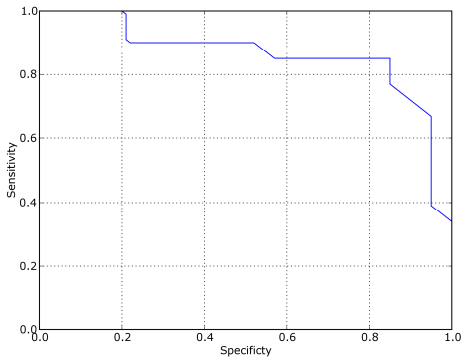


Figure 14. Visualizing Sensitivity/Specificity Curve

7.5 Precision Recall Break Even Point

Visualizing the Precision Recall Break Even Point is another common use case. It involves visualizing threshold values where Precision equals Recall. Visualizing this involves computing and plotting the Precision and Recall versus the threshold. It must be noted here that the function plot_measure can be passed additional parameters which are passes to the underlying plot function from matplotlib to augment the plot.

```
>> import pyroc
>> data = pyroc.table_data("data.csv", header = True)
>> prec = pyroc.measure("prec", data.cl, data.f1)
>> rec = pyroc.measure("rec", data.cl, data.f1)
>> thresh = pyroc.generate_thresholds()
>> pyroc.plot_measure(tresh, prec, 'r-')
>> pyroc.plot_measure(tresh, rec, 'b-')
>> pyroc.xlabel("Threshold")
>> pyroc.ylabel("Measure")
>> pyroc.grid()
```

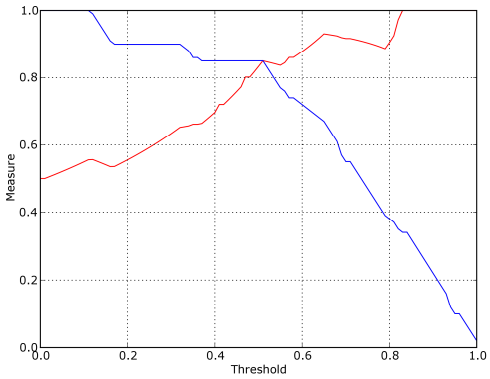
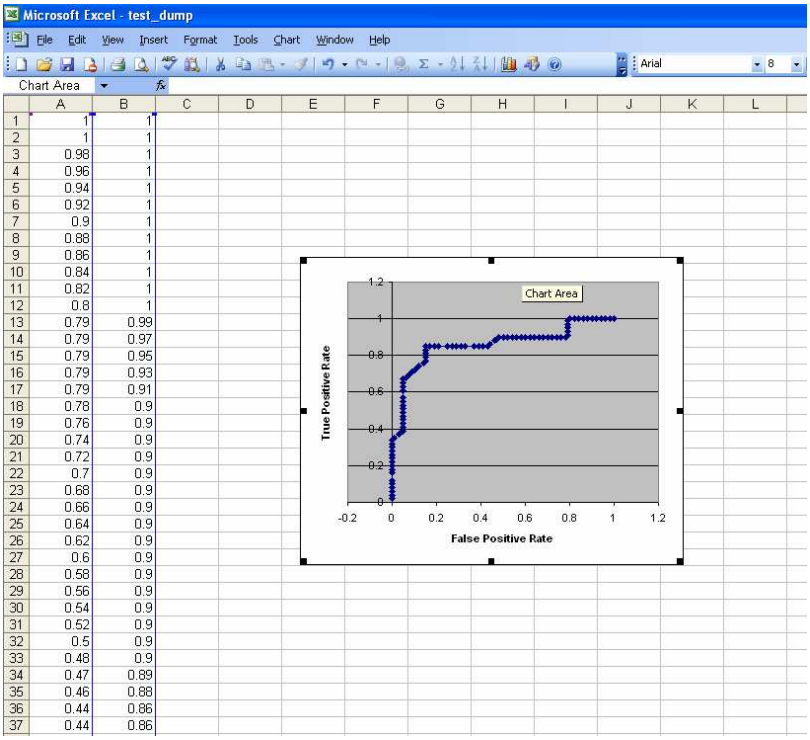


Figure 15. Visualizing Precision Recall Break Even Point

9. Dumping Analysis Results

PyROC implements functionality which allows for the user to dump the results of the analysis in comma delimited file format. This is useful in the case where the results of the analysis have to be loaded in a different software package for analysis. Figure 16. illustrates the table_dump function which takes as input an output file and a set of measure objects.

```
>> pyroc.table_dump("dump.csv", tpr, fpr, ...)
```



	A	B
1	1	1
2	1	1
3	0.98	1
4	0.96	1
5	0.94	1
6	0.92	1
7	0.9	1
8	0.88	1
9	0.86	1
10	0.84	1
11	0.82	1
12	0.8	1
13	0.79	0.99
14	0.79	0.97
15	0.79	0.95
16	0.79	0.93
17	0.79	0.91
18	0.78	0.9
19	0.76	0.9
20	0.74	0.9
21	0.72	0.9
22	0.7	0.9
23	0.68	0.9
24	0.66	0.9
25	0.64	0.9
26	0.62	0.9
27	0.6	0.9
28	0.58	0.9
29	0.56	0.9
30	0.54	0.9
31	0.52	0.9
32	0.5	0.9
33	0.48	0.9
34	0.47	0.89
35	0.46	0.88
36	0.44	0.86
37	0.44	0.86

Figure 16. Dumping results of analysis and importing into any other software package

10. Extending PyROC

PyROC implements functionality that allows the user to add new threshold based measures by providing a standard interface. Figure 16. illustrates how a new threshold based measure which is referred to as “new_phi” (which is the same as the phi) can be added PyROC. After this, in the current session, the user can use “new_phi” as if it is an originally implemented measure. Note that this change lasts only for the current session and needs to be specified in terms of P,N, TP, FP, FN and TN.

```
>> pyroc.add_measure("new_phi", "((TP*TN) - (FP*FN))/ sqrt((TP+FN) * (TN+FP)* (TP+FP) * (TN+FN))")
>> new_phi = pyroc.measure("new_phi",labels, predictions)
```

Figure 16. Extending PyROC by adding new threshold based measures

11. Reference

Function Name	**Parameters**	**Documentation**
table_data	filename delimiter = "," header = True	Reads a file into memory. The columns can be accessed as attributes of the object.
measure	measure_name class_labels predictions threshold_count = 100	Computes measures as specified in Table 1.
combine	name measure1, measure2 …	Combines measures as specified in Table 2.
plot_measure	x_measure y_measure …	Plots the given measures. Other orguments may be provided which will be forwarded to the plot function in matplotlib.
generate_thresholds	threshold_count	Generates thresholds for the specified integer value.
add_measure	name formula	Allows user to add new measures. Must be specified in terms of P, N, TP, TN, FP, FN
table_dump	file_name measure1, measure2 …	Dumps the given measures as a comma delimited file