**Threshold Labeling on ROC Curves Generated by Several Different Classification Models**

Joseph E. Shymanski

College of Engineering & Natural Sciences, University of Tulsa

**Abstract**

This paper presents research into the definitions of threshold values for a number of commonly used classification models as well as methods for labeling these values on their respective ROC curves in R.

*Keywords:* ROC curves, threshold labeling, R packages, cutoff and alpha values, interactive plots, threshold definitions, threshold summaries, ROCR, plotROC, ROCit

**Threshold Labeling on ROC Curves Generated by Several Different Classification Models**

A receiver operating characteristic (ROC) curve is a curve in two-dimensional space which displays how the true positive rate (TPR) and false positive rate (FPR) of a classifier change as a deterministic threshold is varied. Traditionally, the FPR is drawn on the horizontal axis while the TPR lies on the vertical axis. The curve’s endpoints lie at (0, 0) and (1, 1), and the curve tends to bulge outward toward the upper-left corner of the graph. The ROC curve serves as a performance measure for classifiers; more accurate models have curves that bulge out more, while worse models have curves that lie closer to the diagonal. Thus, classifiers whose ROC curve exhibits a higher area under the curve (AUC) are typically more desired.

There are a couple problems with most ROC curves. The first is that the threshold values mentioned above are not uniformly defined for every single classifier. Each model is built differently and has different types of outputs, so a threshold defined for one classifier may not make any sense for a different classifier. The second problem is that the threshold values are usually never presented on the graph. This is an issue if you are trying to determine what TPRs and FPRs you want your classification model to output.

All of the data mentioned in this paper is obtained from sections 8.2.2 “Example: High-earners in the 1994 United States Census” and onward of *Modern Data Science with R* (Baumer et al., 2017). All plots were created using R within RStudio and Jupyter Notebooks (R Core Team, 2019; RStudio Team, 2019; Kluyver et al., 2016).

**Threshold Summary for Different Classifiers**

Certain binary classifiers inherently return class probabilities or scores. For these probabilistic classifiers, no special process is needed to obtain thresholds for their ROC curves, since the probabilities or scores themselves can serve as the thresholds. Other classifiers do not inherently return probabilities or scores. For these discrete classifiers, we need ad hoc methods to obtain appropriate thresholds. Since it can be difficult to remember how these thresholds are generated among the different classifiers, a summary would be quite useful. Fawcett (2006), Majnik, and Bosnić (2013) have presented similar summaries and threshold suggestions.

***Decision Tree (DT)***

Decision Trees are discrete classifiers, but it is still possible to build ROC curves for them. Thresholds can be determined by using the percentages of a given class at each leaf node. Since a decision tree likely does not have hundreds or thousands of leaf nodes, the curve will look a bit jagged. As you can see in Figure 1, the percentage of instances that are “>50K” (a true classification) at each leaf node are given by the rightmost number on the second row: .05, .96, .30, .98, and .72. These will act as our ROC curve thresholds. The model will look at a test instance and filter it down to its appropriate leaf node like normal. If the percentage of training instances that are “>50K” at that leaf node is .05 or greater, then the test instance will receive a true classification. Once every test instance has been classified, the true positive rate (TPR) and false positive rate (FPR) are calculated and the point is plotted in the ROC space. This is repeated for every threshold value and all the points are connected, producing a particularly rough ROC curve, as you can see in Figure 2. Since there are five leaf nodes in Figure 1, there are five points with finite thresholds in Figure 2, and you can see how their values line up.

Moving along a Decision Tree ROC curve is simple but can be misleading. According to Figure 2, if you want an FPR of about .05, then set your threshold to be .72. If you can tolerate an FPR of about .33, then you can set your threshold to be .3. However, setting a threshold value of .5 will not give you an FPR between .05 and .33; it will only ever give you an FPR of .05. The threshold values on each line segment of the curve map to the TPR and FPR values of its leftmost endpoint.

In order to make predictions on future testing data whose classifications are unknown, a threshold should be chosen based on the FPR and TPR values that are most favorable for the situation. Once a threshold has been selected, the ROC curve can be ignored altogether for the rest of the prediction process. Our new set of testing data uses the previously constructed tree and the specified threshold value in order to make a prediction. Each test instance falls to a certain leaf node of the tree and looks at the proportion of training instances in that leaf node that are classified as true. If this proportion is greater than or equal to the threshold value you had set earlier, then the test instance is classified as true, and false otherwise. The same threshold is used for every test instance in the set.

**Threshold Definition.** A Decision Tree threshold is the minimum proportion of true-classified training instances in any leaf node necessary to classify a test instance in that same leaf node as true.

**Figure 1**

*Decision Tree Diagram*

![A close up of a map

Description automatically generated]()**Figure 2**

*Decision Tree ROC Curve*

![A screenshot of a cell phone

Description automatically generated]()

***K-Nearest Neighbors (KNN)***

KNN does not inherently return probabilities or scores. Instead, you can utilize the voting aspect of this classifier to create a ROC curve. Each test instance receives *k* votes that reflect the classifications of its *k*-nearest neighbors, which are training instances. We can take these varying proportions of votes for a given true class as our thresholds. This curve, like the Decision Tree curve, can be quite jagged; you will likely end up plotting *k* points with finite thresholds, and *k* is often not in the hundreds or thousands. However, some implementations of KNN have rules for breaking ties in which more than *k* votes are allowed. Allowing this rule can greatly increase the number of thresholds and, subsequently, points on the curve in an effort to make it smoother. This can be seen in Figure 3, which is the KNN ROC curve obtained by setting *k* equal to ten and allowing the aforementioned tiebreaking rule.

After selecting the appropriate threshold from the KNN ROC curve, the curve can be ignored. Every new test instance will have a certain proportion of its *k­*-nearest training set neighbors classified as true. If this proportion is greater than or equal to the chosen threshold, then the test instance is classified as true, and false otherwise. The same threshold is used for every test instance.

**Threshold Definition.** A KNN threshold is the minimum proportion of votes in favor of the true class in order to classify a test instance as true.

**Figure 3**

*K-Nearest Neighbors ROC Curve*

![A close up of a map

Description automatically generated]()

***Naïve Bayes (NB)***

The Naïve Bayes classifier inherently returns probabilities. Thus, thresholds are simply set to each one of these resulting probabilities.

**Threshold Definition.** A Naïve Bayes threshold is the minimum true-class probability needed to classify a test instance as true.

***Neural Network (NN)***

Neural Network inherently returns scores for classification. Thus, thresholds are simply set to each one of these resulting scores.

**Threshold Definition.** A Neural Network threshold is the minimum true-class score needed to classify a test instance as true.

***Random Forest (RF)***

Random Forest is simply a collection of randomly generated Decision Trees. Each test instance falls through all of the trees, receiving a class vote with each pass. This class vote is determined by which class had the highest proportion of training instances in the leaf node into which the test instance fell. Then the votes are tallied and, usually, the class with the highest vote count wins. However, we can utilize the proportions of votes like we did in the KNN classifier to create our thresholds. Thus, given a forest with *n* trees, the Random Forest ROC curve will have at most *n* points with finite thresholds. Since *n* is usually in the hundreds, the curve will likely appear smooth. Figure 4 demonstrates this smoothness, which contrasts the roughness of the Decision Tree curve generated in Figure 2.

Once a threshold has been chosen from the ROC curve, the curve can be ignored altogether for further predictions. You can send each instance of a new set of testing data inti the same forest generated from before. Once a test instance has filtered entirely through the forest, it will have a certain proportion of votes in favor of a true classification. If this proportion is greater than or equal to the selected threshold value, then the test instance will be classified as true, and false otherwise. The same threshold is used for every instance in the set of testing data.

**Threshold Definition.** A Random Forest threshold is the minimum proportion of votes in favor of the true class in order to classify a test instance as true.

**Figure 4**

*Random Forest ROC Curve*

A close up of a map

Description automatically generated

***Support Vector Machine (SVM)***

SVM inherently returns a class score for each test instance. Thus, thresholds are simply set to these various scores.

**Threshold Definition.** An SVM threshold is the minimum true-class score needed to classify a test instance as true.

**ROC Curve Packages in R**

I researched a number of different ROC packages in R, many of which were summarized by Rickert (2019), according to a few criteria. First, and most importantly, the package must at least be able to generate the necessary data for a labeled ROC curve: true positive rates, false positive rates, and threshold values for each individual point on the curve. The less abstracted and more easily accessible this data is, the better. Second, the package should be able to easily produce an accurate AUC value for a given ROC curve. Finally, the package should provide something interesting or useful which is not readily available in any of the other packages.

With these criteria in mind, I found three packages worth mentioning: **ROCR**, **plotROC**, and **ROCit** (Sing et al., 2005; Sachs, 2017; Khan & Brandenburger, 2020). The handful of other packages not discussed are still viable in their own ways. However, they were either too similar to one of the three aforementioned packages, unnecessarily complicated in terms of syntax, or were better suited for jobs other than creating ROC curves with labeled thresholds. That being said, these three packages together provide nearly all of the ROC curve-plotting features one could ask for.

***ROCR***

**ROCR** is one of the oldest and most reliable packages for the purposes of creating ROC curves. On top of that, it is capable of calculating AUC as well as other performance-measuring plots, such as precision-recall. **ROCR** has two different methods of labeling thresholds on its ROC curves. First, it has a feature called *colorize* within the base R plotting function that colors the curve according to threshold, using the whole color spectrum as a gradient. This offers a wider color range than the default gradient within **ggplot2** (Wickham, 2016). On top of that, **ROCR** allows the user to label specified thresholds on any ROC curve by using the *print.cutoffs.at* feature. You can also stack multiple **ROCR**-generated curves on top of each other by specifying *add = TRUE* in the base R plotting function. These plots can also be made interactive using a package called **plotly**, but due to a bug, you cannot add a color gradient.

**The Good.** This package includes two different ways to label thresholds which may be used separately or in a combination. First, a color gradient can be applied along a given ROC curve to represent the range of threshold values for that curve. Second, a specified sequence of threshold values can be plotted and labeled on the ROC curve.

**The Bad.** Differentiating overlaid curves with a color gradient is quite difficult, as you can see in Figure 5. A method of overlaying ROC curves with labels, then, would be to color code the curves instead, as seen in Figure 6. It is much easier to discern which curve belongs to which classification model, but not as easy to see the relative changes in threshold values along each curve.

**Figure 5**

*Overlaid ROCR Curves with Color Gradient*

A close up of a map

Description automatically generated

**Figure 6**

*Overlaid ROCR Curves with Threshold Labels*

![A close up of a map

Description automatically generated]()

***plotROC***

The next package, **plotROC**, is a bit newer than **ROCR** and not quite as popular. However, it creates sleeker threshold labels than **ROCR**, and it can automatically generate labels for you. Unlike **ROCR**, **plotROC** uses **ggplot2** to create the ROC curves as opposed to the base plotting function. This allows one to overlay several ROC curves quite easily using a single data frame. These overlaid curves are more easily differentiable than the overlaid curves using **ROCR** since the labels match the color of their corresponding curve. On top of that, calculating AUC is easy with **plotROC**. Unfortunately, the result occasionally differs from the results obtained using **ROCR** and **ROCit** starting as early as four decimal places down. Finally, **plotROC** offers interactive plot capabilities built within the package.

**The Good.** Figure 7 displays the great properties that **plotROC** brings to the table. It can easily label ROC curves with a set of threshold values and overlay multiple of these curves onto the same graph, clearly distinguished the curves and labels with a color code.

**The Bad.** A color gradient cannot be easily added to the ROC curves produced by **plotROC**. This means that observing how threshold values are distributed along each curve is much more difficult than, say, with **ROCR**.

**Figure 7**

*Overlaid plotROC Curves with Threshold Labels*

A close up of a map

Description automatically generated

***ROCit***

**ROCit** is the newest of the three packages and, although it is less popular than the first two, has grown in popularity immensely. This package creates ROC curves using the base plotting function and, by default, labels a single point on each curve corresponding to that curve’s Youden Index, which can be seen in Figure 8. **ROCit** does not inherently label thresholds along the curve, but since it uses the same plotting function as **ROCR**, the labels from **ROCR** can be overlaid onto the **ROCit**-generated curve, as demonstrated in Figure 9. Thus, you could obtain a labeled ROC curve with an additional Youden Index label, displaying a (perhaps imperfect) measure of optimality.

**The Good.** The Youden Index calculated in **ROCit** can provide a helpful, additional point for labelling and comparing thresholds along ROC curves.

**The Bad.** This package does not inherently label thresholds on its own and needs to be coupled with another package, such as **ROCR**, in order to accomplish this goal.

**Figure 8**

*ROCit Curve with Youden Index*

![A close up of a map

Description automatically generated]()

**Figure 9**

*ROCit Curve with ROCR Threshold Labels*

![A close up of a map

Description automatically generated]()

**References**

Baumer, B., Kaplan, D., & Horton, N. (2017). *Modern Data Science with R* (1st ed.). CRC Press.

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861–874. <https://doi.org/10.1016/j.patrec.2005.10.010>

Khan, M. R. A., & Brandenburger, T. (2020, June 14). *Performance Assessment of Binary Classifier with Visualization [R package ROCit version 2.1.1]*. <https://CRAN.R-project.org/package=ROCit>.

Kluyver T., Ragan-Kelley B., Pérez F., Granger B., Bussonnier M., Frederic J., Kelley K., Hamrick J., Grout J., Corlay S., Ivanov P., Avila D., Abdalla S., Willing C., & Jupyter Development Team. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. Stand Alone, 0(Positioning and Power in Academic Publishing: Players, Agents and Agendas), 87–90. <https://doi.org/10.3233/978-1-61499-649-1-87>

Majnik, M., & Bosnić, Z. (2013). ROC analysis of classifiers in machine learning: A survey. Intelligent Data Analysis, 17(3), 531–558. <https://doi.org/10.3233/IDA-130592>

R Core Team (2019). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. <https://www.R-project.org/>.

Rickert, J. (2019, March 1). Some R Packages for ROC Curves. R Views. <https://rviews.rstudio.com/2019/03/01/some-r-packages-for-roc-curves/>.

RStudio Team (2019). *RStudio: Integrated Development for R*. RStudio, Inc. <https://rstudio.com/>.

Sachs, M. C. (2017). plotROC: A Tool for Plotting ROC Curves. Journal of Statistical Software, 79(Code Snippet 2). <https://doi.org/10.18637/jss.v079.c02>

Sachs, M. C. (2018, June 23). *Generate ROC Curve Charts for Print and Interactive Use*. The Comprehensive R Archive Network. <https://cran.r-project.org/web/packages/plotROC/vignettes/examples.html>.

Sing, T., Sander, O., Beerenwinkel, N., & Lengauer, T. (2005). ROCR: visualizing classifier performance in R. *Bioinformatics*, 21(20):3940–3941. [http://rocr.bioinf.mpi-sb.mpg.de](http://rocr.bioinf.mpi-sb.mpg.de/).

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. [https://ggplot2.tidyverse.org](https://ggplot2.tidyverse.org/).