ŷhat

PRODUCTS CUSTOMERS COMPANY BLOG RSS Follow @YhatHQ

Email Address

Get Updates

BACK TO BLOG

ROC Curves in Python and R

by Greg

June 15, 2015

Learn More Tweet 174

Ever heard people at your office talking about AUC, ROC, or TPR but been too shy to ask what the heck they're talking about? Well lucky for you we're going to be diving into the wonderful world of binary classification evaluation today. In particular, we'll be discussing ROC curves.

ROC curves are a great technique that have been around for a while and is still one of the tried and true industry standards. So sit back, relax, and get ready to dive into a world of 3 letter acronyms!

What is it?

Receiving Operating Characteristic, or ROC, is a visual way for inspecting the performance of a binary classifier (0/1). In particular, it's comparing the rate at which your classifier is making correct predictions (True Positives or TP) and the rate at which your classifier is making false alarms (False Positives or FP). When talking about True Positive Rate (TPR) or False Positive Rate (FPR) we're referring to the definitions below:

$$TPR = TruePositives/(TruePositives + FalseNegatives)$$

$$FPR = FalsePositives/(FalsePositives + TrueNegatives)$$

You might have heard of True Positives and True Negatives referred to as

Sensitivity and Specificity. No matter what you call it, the big point here is we're measuring the trade off between the rate at which you can correctly predict something, with the rate at which you make an embarrassing blunder and predict something that doesn't happen.

10 Worst Ceremonial First Pitches

Don't let your classifier embarrass you.

A brief history primer

ROC curves were first used during WWII to analyze radar effectiveness. In the early days of radar, it was sometimes hard to tell a bird from a plane. The British pioneered using ROC curves to optimize the way that they relied on radar for detecting incoming German planes.

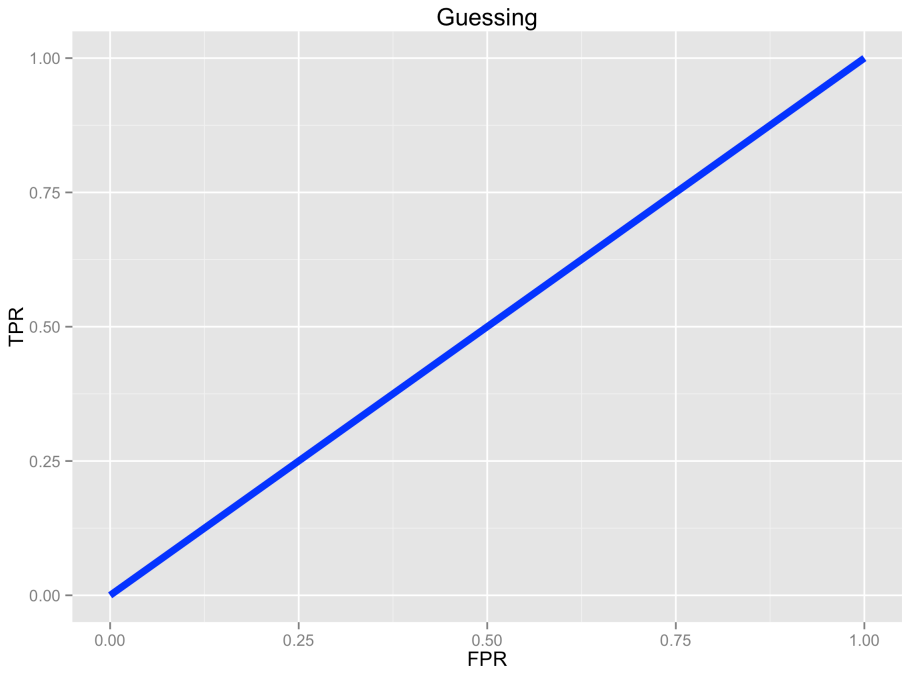
Blast, birds again.

ROC curves: the basics

Sound confusing? Let's go over a few simple rules for reading an ROC curve. To do this, I'm going to present some "pre-canned" charts that will show extreme situations that should make it easier to understand what other ROC curves are "saying".

Guessing

The first example is the simplest: a diagonal line. A diagonal line indicates that the classifier is just making completely random guesses. Since your classifier is only going to be correct 50% of the time, it stands to reason that your TPR and FPR will also be equal.

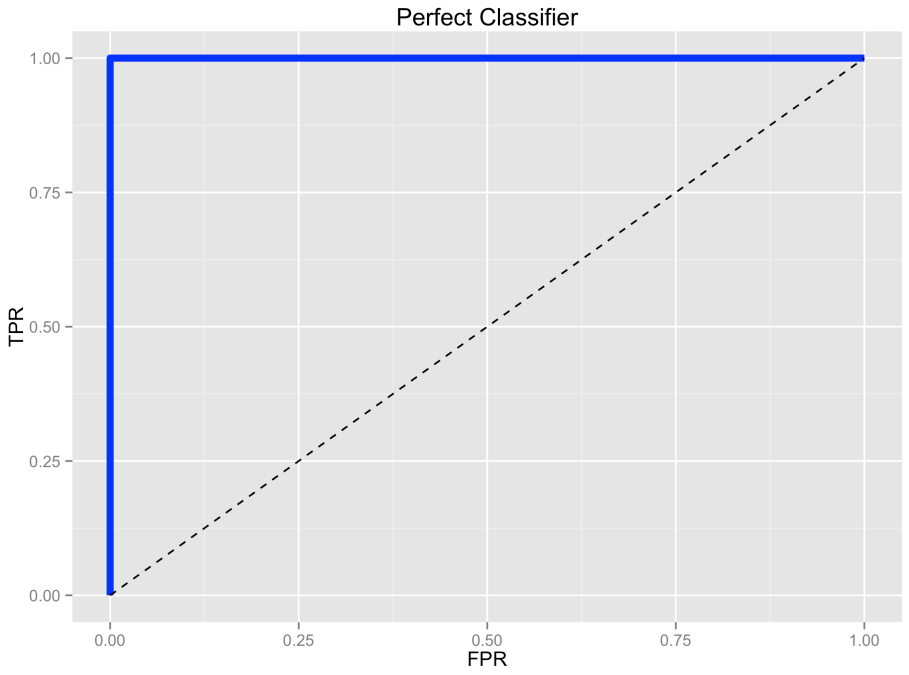


Often times, ROC charts will include the random ROC curve to provide the user with a benchmark for what a naive classifier would do. Any curves above the line are better than guessing, while those below the line...well you're better off guessing.

A Perfect Classifier

We know what a totally random classifier looks like, but what about a PERFECT classifier--i.e. something that makes every prediction correctly.

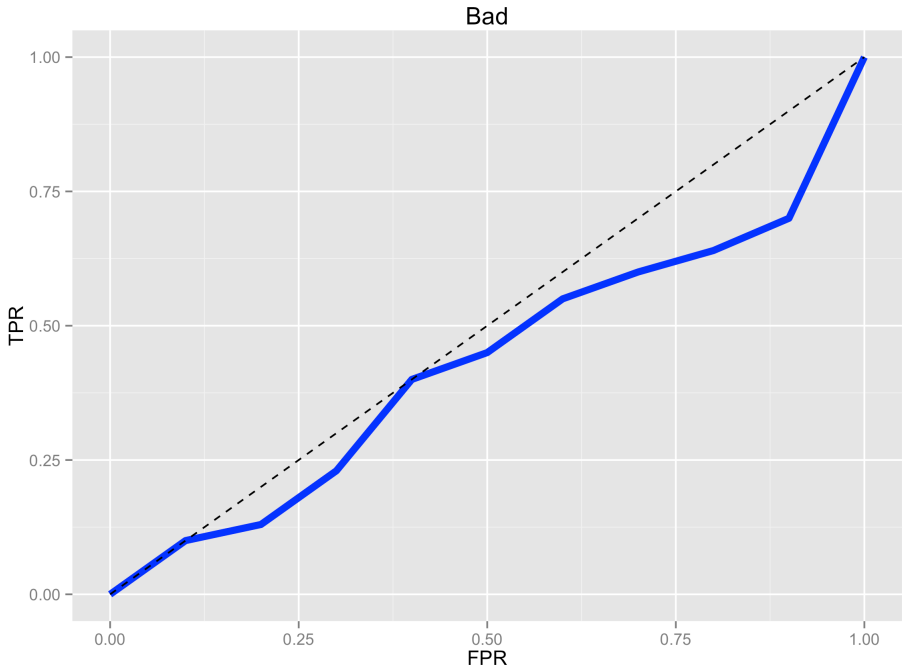
Well if you're lucky enough to have a perfect classifier, then you'll also have a perfect trade-off between TPR and FPR (meaning you'll have a TPR of 1 and an FPR of 0). In that case, your ROC curve looks something like this.



Note the "random curve" is included as a benchmark as a dotted line.

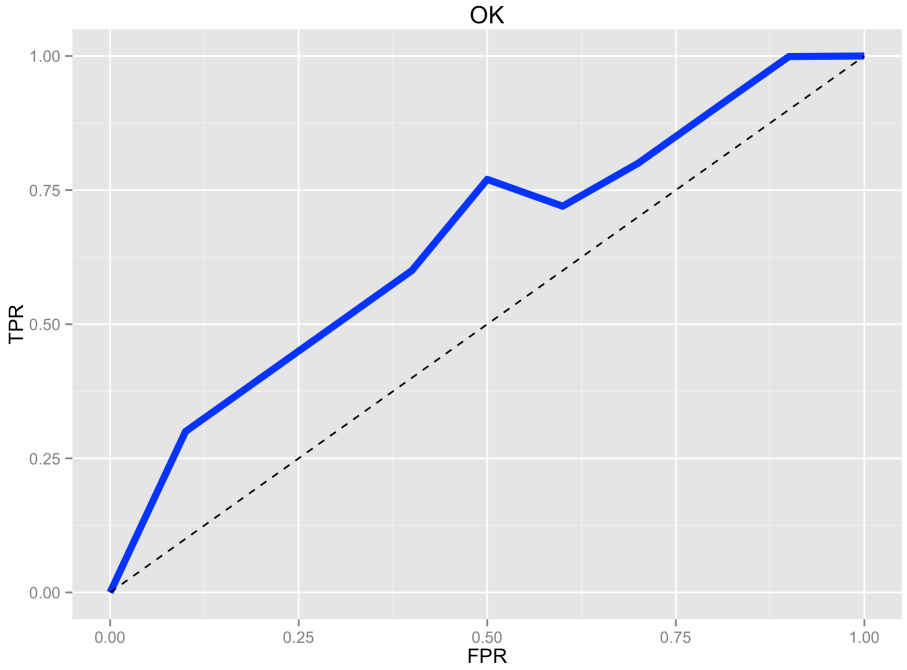
Worse than guessing

So we know what a random classifier looks like and what a perfect classifier looks like, but what about a bad classifier? A bad classifier (i.e. something that's *worse than guessing*) will appear below the random line. This, my friend, is absolute garbage. Throw it away...now!



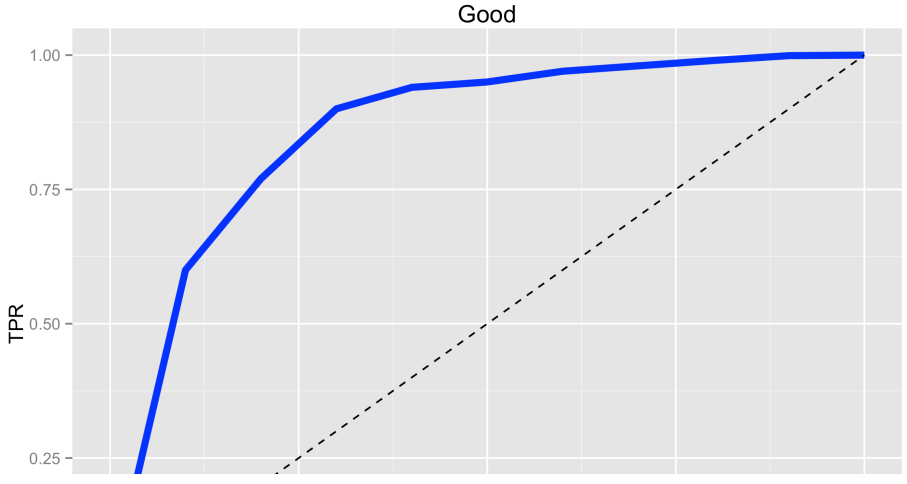
Better than guessing

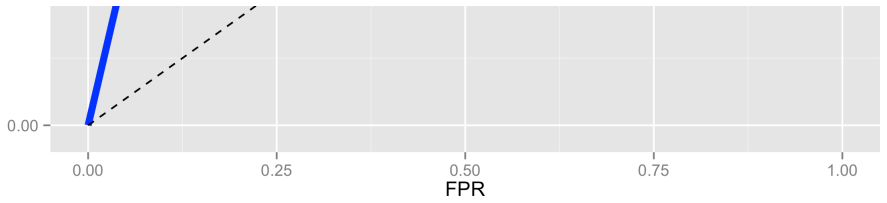
A much more interesting activity is attempting to decipher the difference between an "OK" and a "Good" classifier. The chart below shows an example of a very mediocre classifier. Context is everything of course, but there's not much lift here. In addition, be *very wary* of lines that dip or are very geometric looking. I've found that in practice, this can mean that there's an irregularity with your data, or you're making a very bad assumption in your model.



Pretty good

Ahh this is looking a little better. Below you can see a nice "hump shaped" (it's a technical term) curve that's continually increasing. It sort of looks like it's being yanked up into that top left (the perfect) spot of the chart.





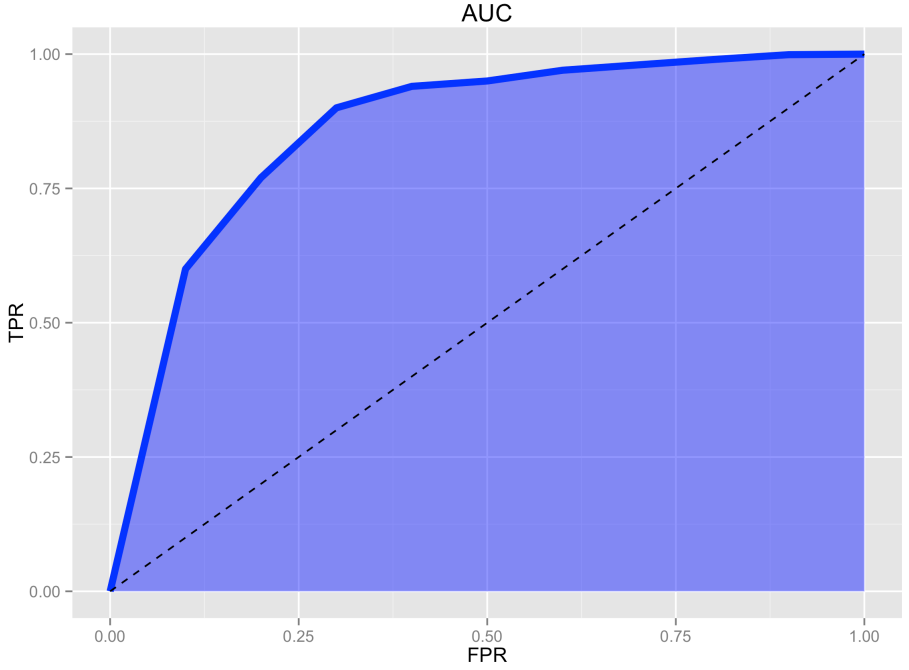
Area under the curve (AUC)

So it turns out that the "hump shaped-ness" actually has a name: AUC or Area Under the Curve. One can't really give an overview of ROC curves without mentioning AUC. The good news is it's exactly what it sounds like--the amount of space underneath the ROC curve. You can think of the AUC as sort of a holistic number that represents how well your TPR and FPR is looking in aggregate.

To make it super simple:

- AUC=0 -> BAD
- AUC=1 -> GOOD

So in the context of an ROC curve, the more "up and left" it looks, the larger the AUC will be and thus, the better your classifier is. Comparing AUC values is also really useful when comparing different models, as we can select the model with the high AUC value, rather than just look at the curves.



Calculating an ROC Curve in Python

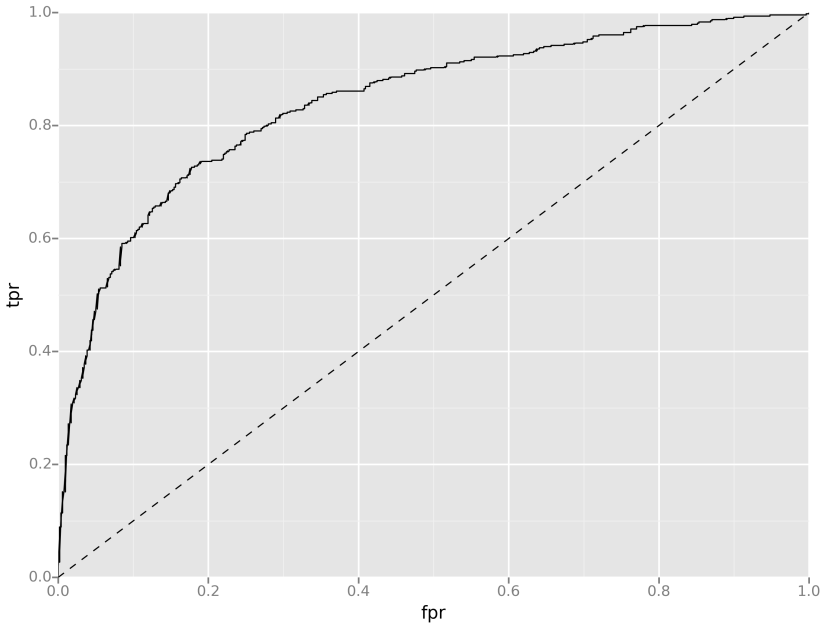
`scikit-learn` makes it super easy to calculate ROC Curves. But first things first: to make an ROC curve, we first need a classification model to evaluate. For this example, I'm going to make a synthetic dataset and then build a logistic regression model using `scikit-learn`.

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression

X, y = make_classification(n_samples=10000, n_features=10, n_classes=2, n_inf
ormative=5)
Xtrain = X[:9000]
Xtest = X[9000:]
ytrain = y[:9000]
ytest = y[9000:]

clf = LogisticRegression()
clf.fit(Xtrain, ytrain)
```

Ok, now that we have our model we can calculate the ROC curve. Pretty easy--from scikit-learn import `roc_curve`, pass in the actual y values from our test set and the predicted probabilities for those same records.

The results will yield your FPR and TPR. Pass those into a `ggplot` and BAM! You've got yourself a nice looking ROC curve.

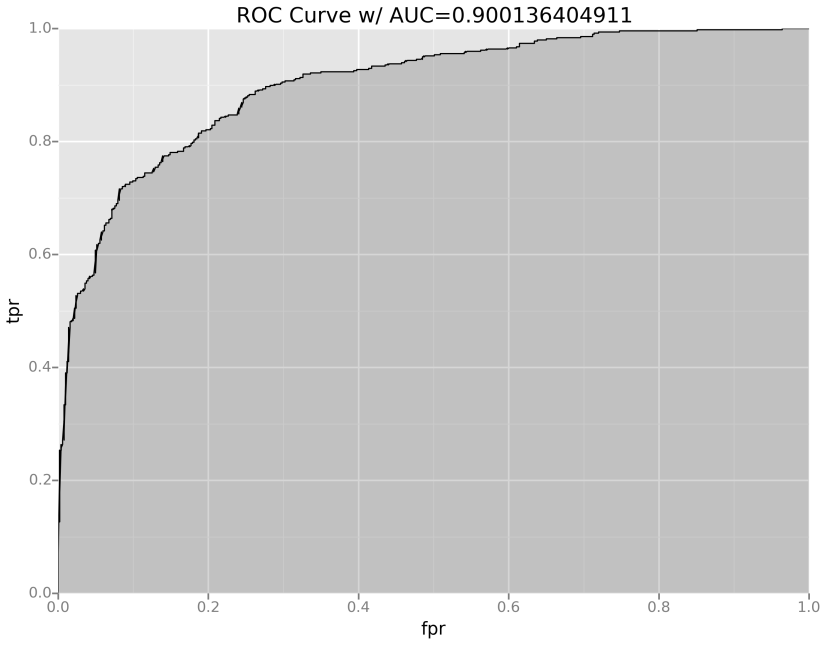
```
from sklearn import metrics
import pandas as pd
from ggplot import *

preds = clf.predict_proba(Xtest)[:,1]
fpr, tpr, _ = metrics.roc_curve(ytest, preds)

df = pd.DataFrame(dict(fpr=fpr, tpr=tpr))
ggplot(df, aes(x='fpr', y='tpr')) +\
    geom_line() +\
    geom_abline(linetype='dashed')
```



Voilà! Your very own ROC curve

Finally to calculate the AUC:

```
auc = metrics.auc(fpr,tpr)
ggplot(df, aes(x='fpr', ymin=0, ymax='tpr')) +\
    geom_area(alpha=0.2) +\
    geom_line(aes(y='tpr')) +\
    ggtitle("ROC Curve w/ AUC=%s" % str(auc))
```



We get 0.900. Recalling from earlier, AUC is bounded between 0 and 1, so

this is pretty good.

Calculating an ROC Curve in R

Making ROC curves in R is easy as well. I highly recommend using the ROCR package. It does all of the hard work for you and makes some pretty nice looking charts.

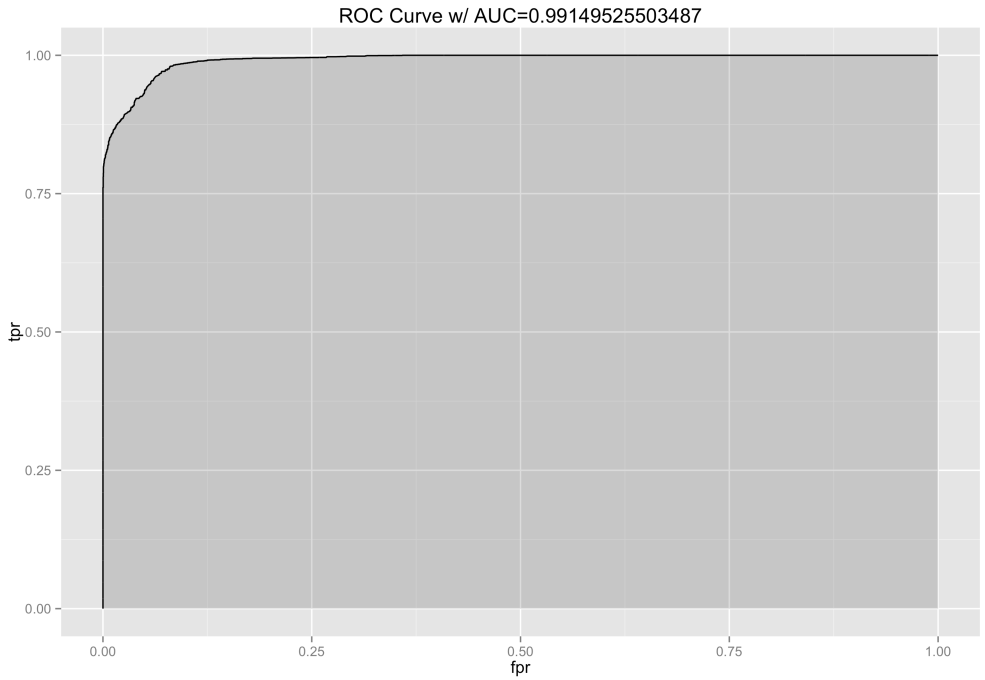
For the model, we're going to build a classifier that uses a logistic regression model to predict if a record from the `diamonds` dataset is over $2400.

```
library(ggplot2)

diamonds$is_expensive <- diamonds$price > 2400
is_test <- runif(nrow(diamonds)) > 0.75
train <- diamonds[is_test==FALSE,]
test <- diamonds[is_test==TRUE,]

summary(fit <- glm(is_expensive ~ carat + cut + clarity, data=train))
```

Using ROCR, making the charts is relatively simple.

```
library(ROCR)

prob <- predict(fit, newdata=test, type="response")
pred <- prediction(prob, test$is_expensive)
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
# I know, the following code is bizarre. Just go with it.
auc <- performance(pred, measure = "auc")
auc <- auc@y.values[[1]]

roc.data <- data.frame(fpr=unlist(perf@x.values),
                       tpr=unlist(perf@y.values),
                       model="GLM")
ggplot(roc.data, aes(x=fpr, ymin=0, ymax=tpr)) +
    geom_ribbon(alpha=0.2) +
    geom_line(aes(y=tpr)) +
```

```
    geom_line(aes(y=tpr)) +
    ggtitle(paste0("ROC Curve w/ AUC=", auc))
```



Closing Thoughts

Well that about does it for ROC curves. For more reading material, check out these resources below:

- UGA: Receiver Operating Characteristic Curves
- Intro to ROC Curves
- The Area Under an ROC Curve
- ROC curves and Area Under the Curve explained (video)

Our Products

scienceCLUSTER

Harness the power of distributed computing to run computationally intensive tasks on a cluster of servers.

LEARN MORE

A platform for productionizing, scaling, and monitoring predictive models in production applications.

LEARN MORE

A Python IDE built for doing data science directly from on desktop.

DOWNLOAD IT NOW!

ŷhat

Yhat (pronounced Y-hat) provides data science and decision management solutions that let data scientists create, deploy and integrate insights into any business application without IT or custom coding.

With Yhat, data scientists can use their preferred scientific tools (e.g. R and Python) to develop analytical projects in the cloud collaboratively and then deploy them as highly scalable real-time decision making APIs for use in customer- or employee-facing apps.

Contact Us

info@yhat.com
support@yhat.com
(646) 918-7342

Our Products

ScienceOps
ScienceCluster

Learn More

About
Blog
Terms of Service

Newsletter

Email Address

Get Updates

Connect With Us

Made in New York City • © 2015 ŷhat