11. Evaluating Binary Classification

Bruce E. Shapiro

Getting Started in Machine Learning

Last revised: March 27, 2019

Image you are a Radar Operator During WW II

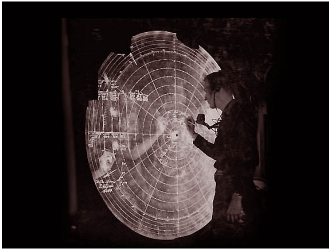


Image from Yank Magazine, Oct. 5, 1945.

The "blips" you don't recognize (and classify as enemy aircraft) may be

- Enemy aircraft
  - We call these TRUE POSITIVES
- False alarms (birds, weather, noise, etc.)
  - We call these FALSE POSITIVES

Image you are a Radar Operator During WW II

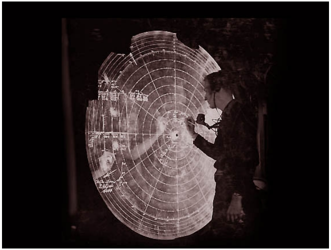


Image from Yank Magazine, Oct. 5, 1945.

The "blips" you don't recognize (and classify as enemy aircraft) may be

- Enemy aircraft
  - We call these TRUE POSITIVES
- False alarms (birds, weather, noise, etc.)
  - We call these FALSE POSITIVES

Other blips whose radar signatures you think you recognize could be

- Friendly aircraft, birds, weather, etc., correctly identified.
  - We call these TRUE NEGATIVES
- Enemy aircraft that not identified as threats
  - We call these FALSE NEGATIVES

Confusion Matrix

- count **true positive** (TP) if $y = 1$ and $y_p = 1$
- count **false positive** (FP) if $y = 0$ and $y_p = 1$
- count **true negative** (TN) if $y = 0$ and $y_p = 0$
- count **false negative** (FN) if $y = 1$ and $y_p = 0$

		Predicted Category y_p 0	Predicted Category y_p 1
Actual Category y	0	True Negative TN	False Positive FP
	1	False Negative FN	True Positive TP

Measures of Success

- Let $N = TN + FP$ = total number of $y = 0$'s (negative examples)
- Let $P = TP + FN$ = total number of $y = 1$'s (positive examples)

- **True Positive Rate**, **Recall**, **Sensitivity**

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

- **specificity** $= \frac{TN}{N} = \frac{TN}{FP + TN}$
- **precision** $= \frac{TP}{TP + FP}$
- **accuracy** $= \frac{TP + TN}{P + N}$

- **False Positive Rate**, **Fallout**

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$
$$= 1 - \text{specificity}$$

- **error rate** $= 1 -$ accuracy

Pythonic Calculation of TP, TN, FP, FN

- Python stores **`booleans`** as **`1`** for **`True`** and **`0`** for **`False`**
- The expression **`True and False`** evaluates to **`False`** and is stored as a **`0`**
- The expression **`False and False`** evaluates to **`False`** and is stored as a **`0`**
- The expression **`True and True`** evaluates to **`True`** and is stored as a **`1`**
- Similarly, the expression **`X==1 and Y==0`** will evaluate to **`True`** (i.e., **`1`**) only when **`X`** is 1 and **`Y`** is 0, and will evaluate to **`False`** (i.e., **`1`**) otherwise

To count the TP, TN, FP, FN manually:

```
OPS=list(zip(Y,YP))
TP = sum([(OB==1) and (PR==1) for OB,PR in OPS])
TN = sum([(OB==0) and (PR==0) for OB,PR in OPS])
FP = sum([(PR==1) and (OB==0) for OB,PR in OPS])
FN = sum([(PR==0) and (OB==1) for OB,PR in OPS])
```

here **YP** is the output of a **predict** method and **Y** is the array of exemplars (such as **YTEST**)

To count the TP, TN, FP, FN manually:

```
OPS=list(zip(Y,YP))
TP = sum([(OB==1) and (PR==1) for OB,PR in OPS])
TN = sum([(OB==0) and (PR==0) for OB,PR in OPS])
FP = sum([(PR==1) and (OB==0) for OB,PR in OPS])
FN = sum([(PR==0) and (OB==1) for OB,PR in OPS])
```

here **YP** is the output of a **predict** method and **Y** is the array of exemplars (such as **YTEST**)

```
confusionMatrix=np.array([[TN,FP],[FN,TP]])
```

To count the TP, TN, FP, FN manually:

```
OPS=list(zip(Y,YP))
TP = sum([(OB==1) and (PR==1) for OB,PR in OPS])
TN = sum([(OB==0) and (PR==0) for OB,PR in OPS])
FP = sum([(PR==1) and (OB==0) for OB,PR in OPS])
FN = sum([(PR==0) and (OB==1) for OB,PR in OPS])
```

here **YP** is the output of a **predict** method and **Y** is the array of exemplars (such as **YTEST**)

```
confusionMatrix=np.array([[TN,FP],[FN,TP]])
```

```
P = sum(OBS)
N = len(OBS)-P
TPR = TP/P # True Positive Rate or Recall
TNR = TN/N
Specificity = TN/(FP+TN)
Accuracy = (TP+TN)/(P+N)
Precision = TP/(TP+FP)
```

Equivalent Metrics in Sklearn

```
sklearn.metrics.confusion_matrix(YTEST,YP)
sklearn.metrics.recall_score(YTEST,YP)
sklearn.metrics.accuracy_score(YTEST,YP)
sklearn.metrics.precision_score(YTEST,YP)
```

ROC Curve

- Plots **true positive rate** as a function of **false positive rate** at different thresholds
- The threshold is varied by sorting the data in decreasing order by probability

ROC Curve

- Plots **true positive rate** as a function of **false positive rate** at different thresholds
- The threshold is varied by sorting the data in decreasing order by probability

```
def ROC(Y,Prob):
    if (len(Y)!=len(Prob)):
        print("Length mismatch")
        return([])

    P=sum(Y); N=len(Y)-P

    if (P<1) or (N<1):
        print("There must be both postive and negative example
        return([])

        # code continued on next page ...
```

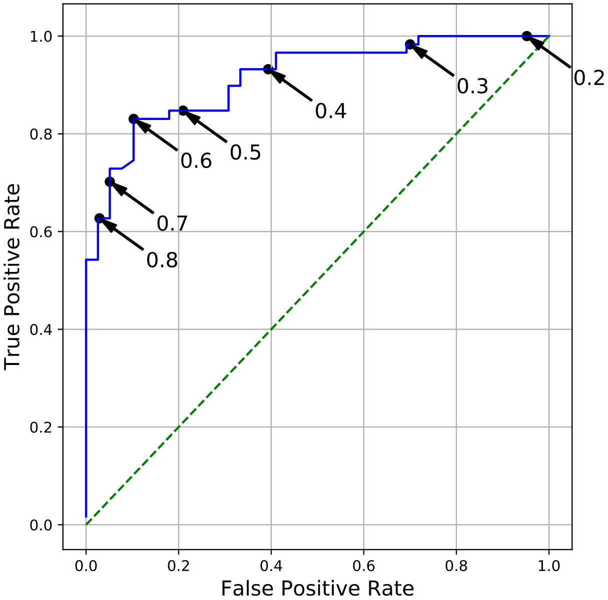
ROC (continued)

```
py_pairs = sorted(zip(Prob, Y), reverse=True)

FP=0; TP=0
ROC_CURVE=[]
pprev=float("-inf")

for p,y in py_pairs:
    if p != pprev:
        ROC_CURVE.append([FP/N, TP/P])
        pprev=p
    if y>0:
        TP+=1
    else:
        FP+=1

ROC_CURVE.append([FP/N, TP/P])
return(ROC_CURVE)
```



True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.8
0.7
0.6
0.5
0.4
0.3
0.2

Code for Plotting ROC Curve With Sklearn

After reading data, and creating test and training set: (this example is for Logistic regression but it works for any method with two classes.)

```
model=LR().fit(XTRAIN,YTRAIN)
probs=model.predict_proba(XTEST)[:,1]
fpr, tpr, threshold = roc_curve(YTEST,probs)
#
# <-- Insert code for threshold annotations here
#      (see next slide)
#
plt.plot(fpr,tpr, c="blue")
plt.plot([0,1],[0,1],c="green",ls="--")
plt.xlabel("False Positive Rate", fontsize=14)
plt.ylabel("True Positive Rate", fontsize=14)
plt.grid()
```

Code for Annotating ROC Curve with Thresholds

```
from scipy import interpolate
# --------- Create an interpolating functions
x=threshold              # domain of f
y=np.vstack([fpr,tpr])   # range of f
f=interpolate.interp1d(x,y)
# --------- Interpolate at interesting threshold points
tvals=np.arange(.2,.9,.1)
xyvals=f(tvals)
xvals,yvals=xyvals
# --------- Add scatter plot to the figure
for t,x,y in zip(tvals,xvals,yvals):
    plt.annotate(str(round(t,1)),(x,y),
      xytext=(x+.1,y-.1), fontsize=14,
      arrowprops={"width":1.0, "facecolor":"black",
                  "headwidth":6})
```

Citations

- Radar image from Yank Magazine, Oct 1945, Copyright unclear, posted at http://www.oldmagazinearticles.com/WW2_radar_history_article_development_of_radar_in_world_war_two#.XHPr-HWQE5k
- UCI MPG data set https://archive.ics.uci.edu/ml/datasets/auto+mpg; relevant paper: Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
- Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.
- ROC Curve Algorithm: Tom Fawcett (2006) An Introduction to ROC analysis, Pattern Recognition Letters, 27:861-874.