Confusion Matrix Visualization

Dennis T Follow
Jul 25, 2019 · 5 min read

How to add a label and percentage to a confusion matrix plotted using a Seaborn heatmap. Plus some additional options.

One great tool for evaluating the behavior and understanding the effectiveness of a binary or categorical classifier is the **Confusion Matrix**. A wise data science professor once said:

> "Good confusion matrix usage separates a good data scientist from a hack."

Ahh… good memories from study group.

Side note: I absolutely love the name confusion matrix because it reminds me of Giosue Cozzarelli — the Panamanian beauty pageant contestant who, when asked to explain the Confucius quote: "*Reading without meditating is a useless occupation.*" proclaimed that "*Confucius was one of the men who invented confusion… Because of this, he was one of the most old… who was one of the oldest. Thank you.*"

When I hear confusion matrix, I always think of this and then laugh to myself…. ***Anyway***, enough about my strange sense of humor, let's get back to Confusion Matrices.

The confusion matrix is a 2 dimensional array comparing **predicted** category labels to the **true** label. For binary classification, these are the *True Positive, True Negative, False Positive* and *False Negative* categories.

Assuming that you've already fit a logistic regression model, the confusion matrix can be calculated manually, or if you are lazy (*aka smart*)… you can use the **confusion_matrix** function from **sklearn**.

The code below fits a Logistic Regression Model and outputs the confusion matrix. *X* is a data frame of my predictors while *y* contains the data for the target category (I'm ignoring train test split for simplicity since it is not relevant to this blog post).

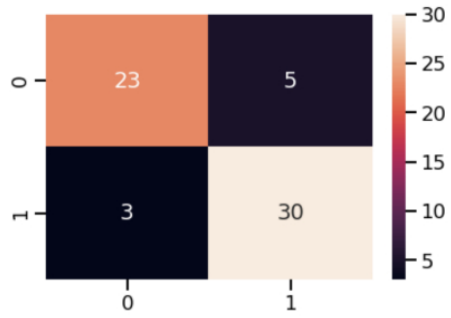
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

#Fit the model
logreg = LogisticRegression(C=1e5)
logreg.fig(X,y)

#Generate predictions with the model using our X values
y_pred = logreg.predict(X)

#Get the confusion matrix
cf_matrix = confusion_matrix(y, y_pred)
print(cf_matrix)
```

Output:

```
array([[23,  5],
       [ 3, 30]])
```

The output is meaningful, but looks like absolute garbage. Luckily, we can make it beautiful with a *heatmap* from the **Seaborn** library.

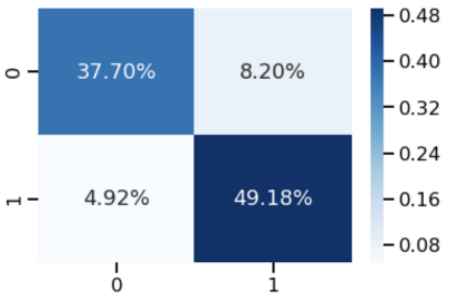
```
import seaborn as sns
sns.heatmap(cf_matrix, annot=True)
```



Oooooh how neat. But wouldn't it be nice if I could see what percentage of my data is represented in each quadrant?

It can be done easily as follows:

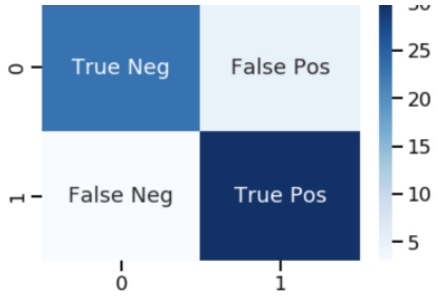
```
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Blues')
```



This is cool too. I even changed the color to something more appealing with the cmap attribute… but what if I want to see both count and percentage at once? What if I want to see a label also? Luckily the seaborn heatmap has the ability to accept text labels for the *annot* field.

Making a heatmap with labels:

```
labels = ['True Neg','False Pos','False Neg','True Pos']
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

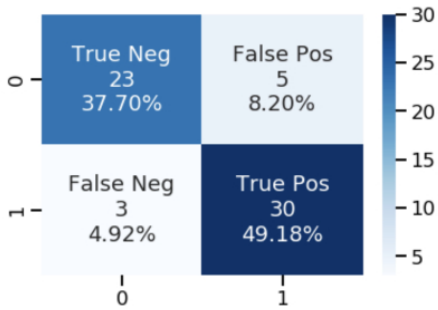


This is cool. The *fmt* field was added in order to prevent formatting being applied to the manual label. …but with this visualization, I've lost all detailed information that makes it useful.

With the ability to add a custom label with the annotation string, I realized that I could **create custom labels** that contained all of the information I desired.

If I can create strings that contain all of the information that I want, I can apply them to the heatmap and show everything at once.

```
group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)

sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```



This is so awesome! I've got some group names, counts, and percentages.

So… what if I put it all into a function and included options to show or not-show certain parameters, and also pass through some other seaborn options like the colormap, or showing the colorbar? What if I included some summary statistics to display such as **Accuracy**, **Precision**, **Recall** and **F-Score**? That would be incredibly convenient. With these thoughts in mind, I created a function that does just that. Feel free to visit the repository below:

DTrimarchi10/confusion_matrix

Contains cf_matrix.py file with a function to make a pretty visualization of a confusion matrix. ...

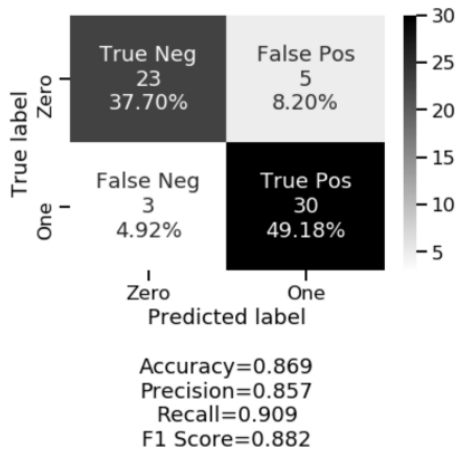
github.com

The function will take in a 2-D Numpy array representing a confusion matrix. It has many options to change the output. The defaults are to show (not hide) things. The function contains a docstring showing all options. I have included the docstring below for convenience:

```
This function will make a pretty plot of an sklearn Confusion Matrix
cm using a Seaborn heatmap visualization.

Arguments
---------
cf:            confusion matrix to be passed in

group_names:   List of strings that represent the labels row by row
               to be shown in each square.

categories:    List of strings containing the categories to be
               displayed on the x,y axis. Default is 'auto'

count:         If True, show the raw number in the confusion matrix.
               Default is True.

normalize:     If True, show the proportions for each category.
               Default is True.

cbar:          If True, show the color bar. The cbar values are
               based off the values in the confusion matrix.
               Default is True.

xyticks:       If True, show x and y ticks. Default is True.

xyplotlabels:  If True, show 'True Label' and 'Predicted Label' on
               the figure. Default is True.

sum_stats:     If True, display summary statistics below the figure.
               Default is True.

figsize:       Tuple representing the figure size. Default will be
               the matplotlib rcParams value.

cmap:          Colormap of the values displayed from
               matplotlib.pyplot.cm. Default is 'Blues'
```

Here are some examples with outputs:

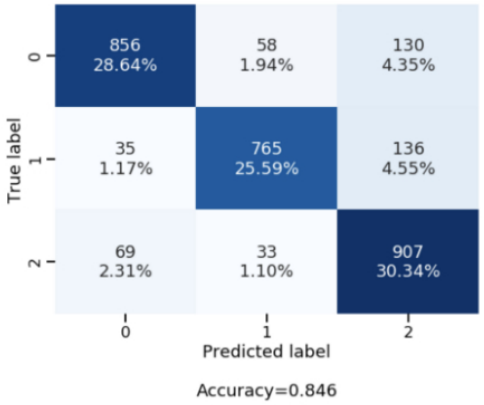
```
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Zero', 'One']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
```



Accuracy=0.869
Precision=0.857
Recall=0.909
F1 Score=0.882

Here is an example using a non-binary classifier (3x3 in this case).

```
make_confusion_matrix(cf_matrix_3x3, figsize=(8,6), cbar=False)
```



You can really do anything with it. I hope that you enjoyed this quick demo on improving confusion matrix visualization. It's always great to have a nice way to visualize your data. Feel free to copy my code or to make suggestions on how to update the functionality. **Cheers**!

Machine Learning | Python | Confusion Matrix

120 claps

WRITTEN BY
Dennis T

Follow

Write the first response

More From Medium

Related reads

3 Things You Need To Know Before You Train-Test Split

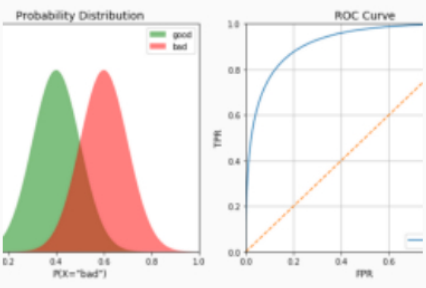
Mayukh Bhattacharyya in Towar...
Dec 4, 2019 · 5 min read
157

Related reads

Having an Imbalanced Dataset? Here Is How You Can Fix It.

Will Badr in Towards Data Science
Feb 22, 2019 · 5 min read ★
983

Related reads



Receiver Operating Characteristic Curves Demystified (in Python)

Syed Sadat Nazrul in Towards Da...
Jun 29, 2018 · 5 min read
1K

Discover Medium
Welcome to a place where words matter. On Medium, smart

Make Medium yours
Follow all the topics you care about, and we'll deliver the

Become a member
Get unlimited access to the best stories on Medium — and