Data Science classification metrics in Sci-kit Learn

*Background

In project, let's explore a few classifications metrics in Python's scikit-learn and write our own functions from scratch to understand the math behind a few of them.

One major area of predictive modeling in data science is classification.

Classification consists of trying to predict which class a particular sample from a population comes from. For example, if we are trying to predict if a particular patient will be re-hospitalized, the two possible classes are hospital (positive) and not-hospitalized (negative). The classification model then tries to predict if each patient will be hospitalized or not hospitalized. In other words, classification is simply trying to predict which bucket (predicted positive vs predicted negative) a particular sample from the population should be placed as seen below.

As we train our classification predictive model, we will want to assess how good it is. Interestingly, there are many different ways to evaluate the performance. Most data scientists that use Python for predictive modeling use the Python package called scikit-learn.

Scikit-learn contains many built-in functions for analyzing the performance of models.

This project will cover the following metrics functions from sklearn.metrics:

- confusion_matrix

- accuracy_score

- recall_score

- precision_score

- f1_score

- roc_curve

- roc_auc_score

*Objective

Explore a few classifications metrics in Python's scikit-learn and write our own functions from scratch (assuming a two-class classification) to understand the math behind a few of them.

*Data Source

We will be a using a dummy data provided by the professor. This dataset has has the actual labels (actual_label) and the prediction probabilities for two models (model_RF and model_LR), where the probabilities are the probability of being class 1.

In [85]:
```
# Import libraries

import pandas as pd
```

In [86]:
```
# Load dataset

df = pd.read_csv('data.csv')
df.head()
```

Out[86]:

	actual_label	model_RF	model_LR
0	1	0.639816	0.531904
1	0	0.490993	0.414496
2	1	0.623815	0.569883
3	1	0.506616	0.443674
4	0	0.418302	0.369532

In [87]:
```
# Define a threshold to define which prediction probabilities are labeled as predicted positive vs predicted negative

thresh = 0.5

# For now let's assume the threshold is 0.5 & let's add 2 additional columns that convert the probabilities to predicted labels

df['predicted_RF'] = (df.model_RF >= 0.5).astype('int')
df['predicted_LR'] = (df.model_LR >= 0.5).astype('int')
df.head()
```

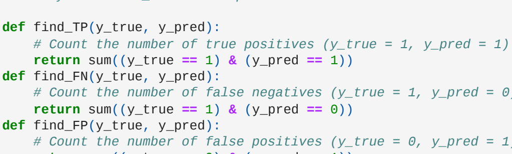
Out[87]:

	actual_label	model_RF	model_LR	predicted_RF	predicted_LR
0	1	0.639816	0.531904	1	1
1	0	0.490993	0.414496	0	0
2	1	0.623815	0.569883	1	1
3	1	0.506616	0.443674	1	0
4	0	0.418302	0.369532	0	0

Confusion Matrix

In [88]:
```
# Load confusion matrix table for visualization

!pip install pillow
from PIL import Image
import matplotlib.pyplot as plt
image = Image.open('C:/Users/Harvey/Documents/Harvey/Personal/DS Personal Projects/Data Science Classification Metrics in Sci-kit Learn/conf_matrix.
plt.imshow(image)
plt.axis('off') # hide axis
plt.show()
```

```
Requirement already satisfied: pillow in c:\users\harvey\anaconda3\lib\site-packages (9.4.0)
```



Given an actual label and a predicted label, the first thing to do is divide the samples in 4 buckets:

- True positive - actual = 1, predicted = 1

- False positive - actual = 1, predicted = 0

- False negative - actual = 0, predicted = 1

- True negative - actual = 0, predicted = 0

In [89]:
```
# Get the confusion matrix (as a 2x2 array) from scikit learn, which takes as inputs the actual labels and the predicted labels

from sklearn.metrics import confusion_matrix
confusion_matrix(df.actual_label.values, df.predicted_RF.values)
```

Out[89]:
```
array([[5519, 2360],
       [2832, 5047]], dtype=int64)
```

There are 5,047 true positives, 2,832 false negatives, 2,360 false positives, and 5,519 true negatives

In [90]:
```
# Verify confusion_matrix and print

def find_TP(y_true, y_pred):
    # Count the number of true positives (y_true = 1, y_pred = 1)
    return sum((y_true == 1) & (y_pred == 1))
def find_FN(y_true, y_pred):
    # Count the number of false negatives (y_true = 1, y_pred = 0)
    return sum((y_true == 1) & (y_pred == 0))
def find_FP(y_true, y_pred):
    # Count the number of false positives (y_true = 0, y_pred = 1)
    return sum((y_true == 0) & (y_pred == 1))
def find_TN(y_true, y_pred):
    # Count the number of true negatives (y_true = 0, y_pred = 0)
    return sum((y_true == 0) & (y_pred == 0))

print('TP:',find_TP(df.actual_label.values, df.predicted_RF.values))
print('FN:',find_FN(df.actual_label.values, df.predicted_RF.values))
print('FP:',find_FP(df.actual_label.values, df.predicted_RF.values))
print('TN:',find_TN(df.actual_label.values, df.predicted_RF.values))
```

```
TP: 5047
FN: 2832
FP: 2360
TN: 5519
```

In [91]:
```
# Calculate all 4 buckets (TP, FN, FP, TN)

import numpy as np
def find_conf_matrix_values(y_true,y_pred):
    # Calculate TP, FN, FP, TN
    TP = find_TP(y_true,y_pred)
    FN = find_FN(y_true,y_pred)
    FP = find_FP(y_true,y_pred)
    TN = find_TN(y_true,y_pred)
    return TP,FN,FP,TN
```

In [92]:
```
# Duplicate confusion_matrix

def my_confusion_matrix(y_true, y_pred):
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return np.array([[TN,FP],[FN,TP]])
my_confusion_matrix(df.actual_label.values, df.predicted_RF.values)
```

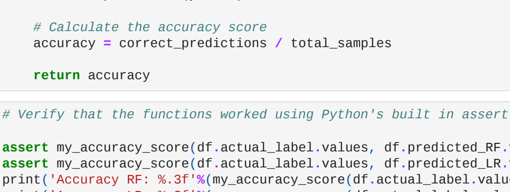
Out[92]:
```
array([[5519, 2360],
       [2832, 5047]])
```

In [93]:
```
# Verify that the functions worked using Python's built in assert function and numpy's array_equal functions

assert np.array_equal(my_confusion_matrix(df.actual_label.values, df.predicted_RF.values),\
                      confusion_matrix(df.actual_label.values, df.predicted_RF.values) ),'my_confusion_matrix() is not correct for RF'

assert np.array_equal(my_confusion_matrix(df.actual_label.values, df.predicted_LR.values),\
                      confusion_matrix(df.actual_label.values, df.predicted_LR.values) ),'my_confusion_matrix() is not correct for LR'
```

Accuracy Score

In [94]:
```
# Load accuracy image for visualization

!pip install pillow
from PIL import Image
import matplotlib.pyplot as plt
image = Image.open('C:/Users/Harvey/Documents/Harvey/Personal/DS Personal Projects/Data Science Classification Metrics in Sci-kit Learn/accuracy.png
plt.imshow(image)
plt.axis('off') # hide axis
plt.show()
```

```
Requirement already satisfied: pillow in c:\users\harvey\anaconda3\lib\site-packages (9.4.0)
```



This is the most common metric for classification is accuracy, which is the fraction of samples predicted correctly as shown above.

In [102…
```
# Obtain the accuracy score from scikit learn, which takes as inputs the actual labels and the predicted labels

from sklearn.metrics import accuracy_score
accuracy_score(df.actual_label.values, df.predicted_RF.values)
```

Out[102]:
```
0.6705165630156111
```

Duplicate accuracy_score def my_accuracy_score(y_true, y_pred): # Calculate the fraction of samples predicted correctly TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred) return np.array([[TN,FP],[FN,TP]]) my_accuracy_score(df.actual_label.values, df.predicted_RF.values)

In [99]:
```
# Calculate the accuracy score

def my_accuracy_score(y_true, y_pred):
    # Calculate the number of correctly predicted samples
    correct_predictions = np.sum(y_true == y_pred)

    # Calculate the total number of samples
    total_samples = len(y_true)

    # Calculate the accuracy score
    accuracy = correct_predictions / total_samples

    return accuracy
```

In [104…
```
# Verify that the functions worked using Python's built in assert function

assert my_accuracy_score(df.actual_label.values, df.predicted_RF.values) == accuracy_score(df.actual_label.values, df.predicted_RF.values), 'my_accu
assert my_accuracy_score(df.actual_label.values, df.predicted_LR.values) == accuracy_score(df.actual_label.values, df.predicted_LR.values), 'my_accu
print('Accuracy RF: %.3f'%(my_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Accuracy LR: %.3f'%(my_accuracy_score(df.actual_label.values, df.predicted_LR.values)))
```

```
Accuracy RF: 0.671
Accuracy LR: 0.616
```

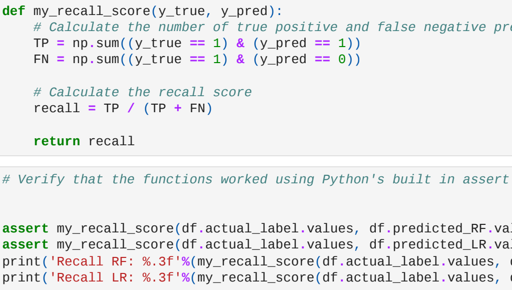
Using accuracy as a performance metric, the RF model is more accurate than the LR model.

So should we stop here and say RF model is the best model? No! Accuracy is not always the best metric to use to assess classification models. For example, let's say that we are trying to predict something that only happens 1 out of 100 times. We could build a model that gets 99% accuracy by saying the event never happened. However, we catch 0% of the events we care about. The 0% measure here is another performance metric known as recall.

Recall Score / Sensitivity

In [49]:
```
# Load recall image for visualization

!pip install pillow
from PIL import Image
import matplotlib.pyplot as plt
image = Image.open('C:/Users/Harvey/Documents/Harvey/Personal/DS Personal Projects/Data Science Classification Metrics in Sci-kit Learn/recall.png')
plt.imshow(image)
plt.axis('off') # hide axis
plt.show()
```

```
Requirement already satisfied: pillow in c:\users\harvey\anaconda3\lib\site-packages (9.4.0)
```



Recall (also known as sensitivity) is the fraction of positives events that you predicted correctly as shown above.

In [50]:
```
# Obtain the accuracy score from scikit-learn, which takes as inputs the actual labels and the predicted labels

from sklearn.metrics import recall_score
recall_score(df.actual_label.values, df.predicted_RF.values)
```

Out[50]:
```
0.6405635232897576
```

Duplicate recall_score def my_recall_score(y_true, y_pred): # Calculate the fraction of positive samples predicted correctly TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred) return np.array([[TN,FP],[FN,TP]]) my_recall_score(df.actual_label.values, df.predicted_RF.values)

In [105…
```
# Calculate recall score

def my_recall_score(y_true, y_pred):
    # Calculate the number of true positive and false negative predictions
    TP = np.sum((y_true == 1) & (y_pred == 1))
    FN = np.sum((y_true == 1) & (y_pred == 0))

    # Calculate the recall score
    recall = TP / (TP + FN)

    return recall
```

In [109…
```
# Verify that the functions worked using Python's built in assert function


assert my_recall_score(df.actual_label.values, df.predicted_RF.values) == recall_score(df.actual_label.values, df.predicted_RF.values), 'my_recall_s
assert my_recall_score(df.actual_label.values, df.predicted_LR.values) == recall_score(df.actual_label.values, df.predicted_LR.values), 'my_recall_s
print('Recall RF: %.3f'%(my_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall LR: %.3f'%(my_recall_score(df.actual_label.values, df.predicted_LR.values)))
```

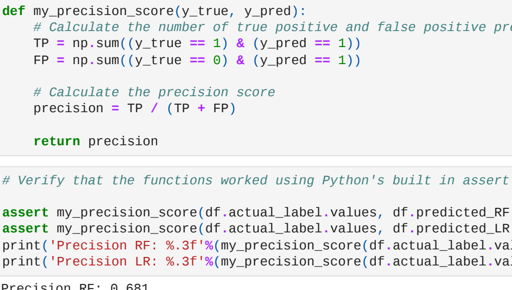
```
Recall RF: 0.641
Recall LR: 0.543
```

One method to boost the recall is to increase the number of samples that we define as predicted positive by lowering the threshold for predicted positive. Unfortunately, this will also increase the number of false positives. Another performance metric called precision takes this into account.

Precision Score

In [56]:
```
# Load recall image for visualization

!pip install pillow
from PIL import Image
import matplotlib.pyplot as plt
image = Image.open('C:/Users/Harvey/Documents/Harvey/Personal/DS Personal Projects/Data Science Classification Metrics in Sci-kit Learn/precision.pr
plt.imshow(image)
plt.axis('off') # hide axis
plt.show()
```

```
Requirement already satisfied: pillow in c:\users\harvey\anaconda3\lib\site-packages (9.4.0)
```



Precision is the fraction of predicted positives events that are actually positive as shown above

In [57]:
```
# Obtain the accuracy score from scikit-learn, which takes as inputs the actual labels and the predicted labels

from sklearn.metrics import precision_score
precision_score(df.actual_label.values, df.predicted_RF.values)
```

Out[57]:
```
0.681382476036182
```

Duplicate precision_score def my_precision_score(y_true, y_pred): # Calculate the fraction of predicted positives samples that are actually positive TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred) return np.array([[TN,FP],[FN,TP]]) my_precision_score(df.actual_label.values, df.predicted_RF.values)

In [107…
```
# Calculate precision score

def my_precision_score(y_true, y_pred):
    # Calculate the number of true positive and false positive predictions
    TP = np.sum((y_true == 1) & (y_pred == 1))
    FP = np.sum((y_true == 0) & (y_pred == 1))

    # Calculate the precision score
    precision = TP / (TP + FP)

    return precision
```

In [108…
```
# Verify that the functions worked using Python's built in assert function

assert my_precision_score(df.actual_label.values, df.predicted_RF.values) == precision_score(df.actual_label.values, df.predicted_RF.values), 'my_pr
assert my_precision_score(df.actual_label.values, df.predicted_LR.values) == precision_score(df.actual_label.values, df.predicted_LR.values), 'my_pr
print('Precision RF: %.3f'%(my_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision LR: %.3f'%(my_precision_score(df.actual_label.values, df.predicted_LR.values)))
```

```
Precision RF: 0.681
Precision LR: 0.636
```

In this case, it looks like RF model is better at both recall and precision.

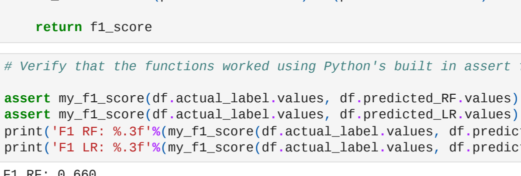
But what if one model was better at recall and the other was better at precision.

Another method to use is called the F1 score.

F1 Score

In [61]:
```
# Load recall image for visualization

!pip install pillow
from PIL import Image
import matplotlib.pyplot as plt
image = Image.open('C:/Users/Harvey/Documents/Harvey/Personal/DS Personal Projects/Data Science Classification Metrics in Sci-kit Learn/f1_score.png
plt.imshow(image)
plt.axis('off') # hide axis
plt.show()
```

```
Requirement already satisfied: pillow in c:\users\harvey\anaconda3\lib\site-packages (9.4.0)
```



The f1 score is the harmonic mean of recall and precision, with a higher score as a better model, calculated using the above formula

In [62]:
```
# Obtain the f1 score from scikit-learn, which takes as inputs the actual labels and the predicted labels

from sklearn.metrics import f1_score
f1_score(df.actual_label.values, df.predicted_RF.values)
```

Out[62]:
```
0.660342797330891
```

Duplicate f1_score def my_f1_score(y_true, y_pred): # Calculate the F1 score recall = my_recall_score(y_true,y_pred) precision = my_precision_score(y_true,y_pred) TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred) return np.array([[TN,FP],[FN,TP]]) my_f1_score(df.actual_label.values, df.predicted_RF.values)

In [110…
```
# Calculate f1 score

def my_f1_score(y_true, y_pred):
    # Calculate the recall and precision scores
    recall = my_recall_score(y_true, y_pred)
    precision = my_precision_score(y_true, y_pred)

    # Calculate the F1 score
    f1_score = 2 * (precision * recall) / (precision + recall)

    return f1_score
```

In [111…
```
# Verify that the functions worked using Python's built in assert function

assert my_f1_score(df.actual_label.values, df.predicted_RF.values) == f1_score(df.actual_label.values, df.predicted_RF.values), 'my_f1_score failed
assert my_f1_score(df.actual_label.values, df.predicted_LR.values) == f1_score(df.actual_label.values, df.predicted_LR.values), 'my_f1_score failed
print('F1 RF: %.3f'%(my_f1_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 LR: %.3f'%(my_f1_score(df.actual_label.values, df.predicted_LR.values)))
```

```
F1 RF: 0.660
F1 LR: 0.586
```

So far, we have assumed that we defined a threshold of 0.5 for selecting which samples are predicted as positive. If we change the threshold, the performance metrics will change as well (see below).

In [114…
```
print('Scores with Threshold = 0.5')
print(' ')
print('Accuracy RF:  %.3f'%(my_accuracy_score(df.actual_label.values, df.predicted_RF.values)))
print('Recall RF:    %.3f'%(my_recall_score(df.actual_label.values, df.predicted_RF.values)))
print('Precision RF: %.3f'%(my_precision_score(df.actual_label.values, df.predicted_RF.values)))
print('F1 RF:        %.3f'%(my_f1_score(df.actual_label.values, df.predicted_RF.values)))
print(' ')
print('Scores with Threshold = 0.25')
print(' ')
print('Accuracy RF:  %.3f'%(my_accuracy_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Recall RF:    %.3f'%(my_recall_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f'%(my_precision_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
print('F1 RF:        %.3f'%(my_f1_score(df.actual_label.values, (df.model_RF >= 0.25).astype('int').values)))
```

```
Scores with Threshold = 0.5

Accuracy RF:  0.671
Recall RF:    0.641
Precision RF: 0.681
F1 RF:        0.660

Scores with Threshold = 0.25

Accuracy RF:  0.502
Recall RF:    1.000
Precision RF: 0.501
F1 RF:        0.668
```

How do we assess a model if we haven't picked a threshold?

One very common method is using the receiver operating characteristic (ROC) curve.

roc_curve and roc_auc_score

ROC curves are VERY help with understanding the balance between true-positive rate and false positive rates. The inputs to these functions (roc_curve and roc_auc_score) are the actual labels and the predicted probabilities (not the predicted labels). Both roc_curve and roc_auc_score are both complicated functions, so let's use sci-kit learn's functions and explain the key points.

In [115…
```
# Use roc_curve to make the ROC plot

from sklearn.metrics import roc_curve

fpr_RF, tpr_RF, thresholds_RF = roc_curve(df.actual_label.values, df.model_RF.values)
fpr_LR, tpr_LR, thresholds_LR = roc_curve(df.actual_label.values, df.model_LR.values)
```

The roc_curve function returns three lists:

In [117…
```
# thresholds = all unique prediction probabilities in descending order
thresholds_RF
```

Out[117]:
```
array([1.93052053, 0.93052053, 0.82363091, ..., 0.25654616, 0.25587275,
       0.17142947])
```

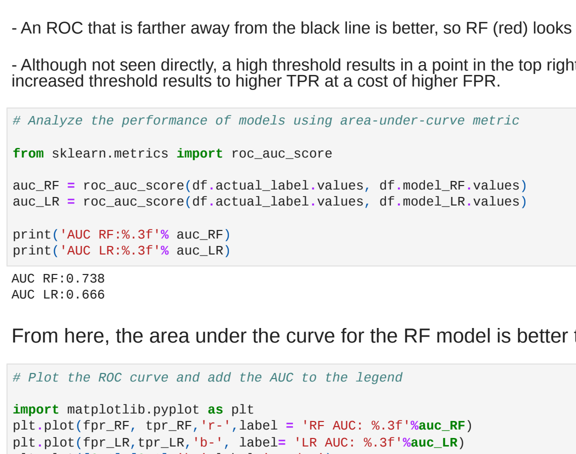
In [118…
```
# fpr = the false positive rate (FP / (FP+TN)) for each threshold
fpr_RF
```

Out[118]:
```
array([0.        , 0.        , 0.        , ..., 0.9941617, 0.9941617,
       1.        ])
```

In [119…
```
# tpr = the true positive rate (TP / (TP+FN)) (i.e. recall) for each threshold
tpr_RF
```

Out[119]:
```
array([0.00000000e+00, 1.26919660e-04, 5.33062571e-03, ...,
       9.99873080e-01, 1.00000000e+00, 1.00000000e+00])
```

In [120…
```
# Plot the ROC curve for each model

import matplotlib.pyplot as plt

plt.plot(fpr_RF, tpr_RF,'r-',label = 'RF')
plt.plot(fpr_LR,tpr_LR,'b-', label= 'LR')
plt.plot([0,1],[0,1],'k-',label='random')
plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



There are a couple things that we can observe from this figure.

- A model that randomly guesses the label will result in the black line (good models should have a curve above this black line).

- An ROC that is farther away from the black line is better, so RF (red) looks better than LR (blue).

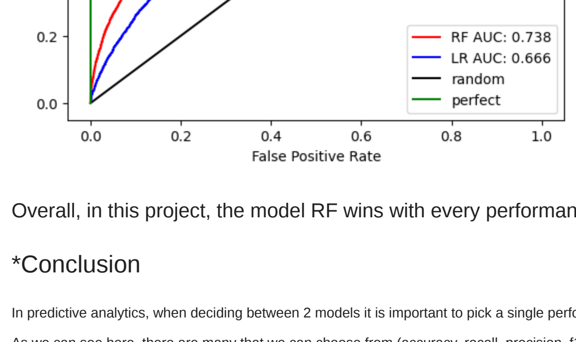
- Although not seen directly, a high threshold results in a point in the top right and a low threshold results in a point in the bottom left. This means increased threshold results to higher TPR at a cost of higher FPR.

In [121…
```
# Analyze the performance of models using area-under-curve metric

from sklearn.metrics import roc_auc_score

auc_RF = roc_auc_score(df.actual_label.values, df.model_RF.values)
auc_LR = roc_auc_score(df.actual_label.values, df.model_LR.values)

print('AUC RF:%.3f'% auc_RF)
print('AUC LR:%.3f'% auc_LR)
```

```
AUC RF:0.738
AUC LR:0.666
```

From here, the area under the curve for the RF model is better than the LR

In [122…
```
# Plot the ROC curve and add the AUC to the legend

import matplotlib.pyplot as plt
plt.plot(fpr_RF, tpr_RF,'r-',label = 'RF AUC: %.3f'%auc_RF)
plt.plot(fpr_LR,tpr_LR,'b-', label= 'LR AUC: %.3f'%auc_LR)
plt.plot([0,1],[0,1],'k-',label='random')
plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



Overall, in this project, the model RF wins with every performance metric.

*Conclusion

In predictive analytics, when deciding between 2 models it is important to pick a single performance metric.

As we can see here, there are many that we can choose from (accuracy, recall, precision, f1-score, AUC, etc).

Ultimately, we should use the perfomance metric that is most suitable for the business problem at hand.

Many prefer to use the AUC because it does not require selecting a threshold and helps balance true positive rate and false positive rate.

End