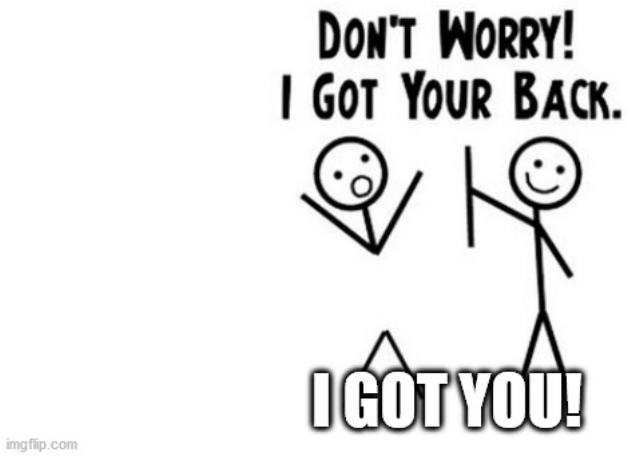
AUC-ROC Curve in Machine Learning Clearly Explained

BEGINNER PYTHON TECHNIQUE

AUC-ROC Curve – The Star Performer!

You've built your machine learning model – so what's next? You need to evaluate it and validate how good (or bad) it is, so you can then decide on whether to implement it. That's where the AUC-ROC curve comes in.

The name might be a mouthful, but it is just saying that we are calculating the "Area Under the Curve" (AUC) of "Receiver Characteristic Operator" (ROC). Confused? I feel you! I have been in your shoes. But don't worry, we will see what these terms mean in detail and everything will be a piece of cake!



For now, just know that the AUC-ROC curve helps us visualize how well our machine learning classifier is performing. Although it works for only binary classification problems, we will see towards the end how we can extend it to evaluate multi-class classification problems too.

We'll cover topics like sensitivity and specificity as well since these are key topics behind the AUC-ROC curve.

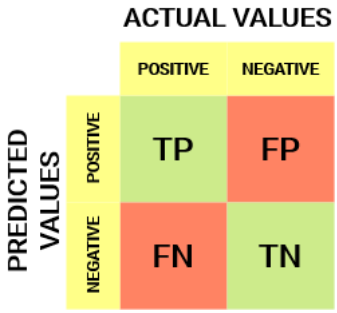
I suggest going through the article on Confusion Matrix as it will introduce some important terms which we will be using in this article.

Table of Contents

- AUC-ROC in Python
- AUC-ROC for Multi-Class Classification

What are Sensitivity and Specificity?

This is what a confusion matrix looks like:



From the confusion matrix, we can derive some important metrics that were not discussed in the previous article. Let's talk about them here.

Sensitivity / True Positive Rate / Recall

$$Sensitivity = \frac{TP}{TP + FN}$$

Sensitivity tells us what proportion of the positive class got correctly classified.

A simple example would be to determine what proportion of the actual sick people were correctly detected by the model.

False Negative Rate

$$FNR = \frac{FN}{TP + FN}$$

False Negative Rate (FNR) tells us what proportion of the positive class got incorrectly classified by the classifier.

A higher TPR and a lower FNR is desirable since we want to correctly classify the positive class.

Specificity / True Negative Rate

$$Specificity = \frac{TN}{TN + FP}$$

Specificity tells us what proportion of the negative class got correctly classified.

Taking the same example as in Sensitivity, Specificity would mean determining the proportion of healthy people who were correctly identified by the model.

False Positive Rate

$$FPR = \frac{FP}{TN + FP} = 1 - Specificity$$

FPR tells us what proportion of the negative class got incorrectly classified by the classifier.

A higher TNR and a lower FPR is desirable since we want to correctly classify the negative class.

Out of these metrics, **Sensitivity** and **Specificity** are perhaps the most important and we will see later on how these are used to build an evaluation metric. But before that, let's understand why the probability of prediction is better than predicting the target class directly.

Probability of Predictions

A machine learning classification model can be used to predict the actual class of the data point directly or predict its probability of belonging to different classes. The latter gives us more control over the result. We can determine our own threshold to interpret the result of the classifier. This is sometimes more prudent than just building a completely new model!

Setting different thresholds for classifying positive class for data points will inadvertently change the Sensitivity and Specificity of the model. And one of these thresholds will probably give a better result than the others, depending on whether we are aiming to lower the number of False Negatives or False Positives.

Have a look at the table below:

ID	Actual	Prediction Probability	>0.6	>0.7	> 0.8	Metric
1	0	0.98	1	1	1	
2	1	0.67	1	0	0	
3	1	0.58	0	0	0	
4	0	0.78	1	1	0	
5	1	0.85	1	1	1	
6	0	0.86	1	1	1	
7	0	0.79	1	1	0	
8	0	0.89	1	1	1	
9	1	0.82	1	1	1	
10	0	0.86	1	1	1	
			0.75	0.5	0.5	TPR
			1	1	0.66	FPR
			0	0	0.33	TNR
			0.25	0.5	0.5	FNR

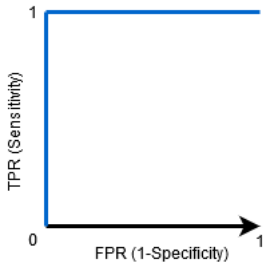
The metrics change with the changing threshold values. We can generate different confusion matrices and compare the various metrics that we discussed in the previous section. But that would not be a prudent thing to do. Instead, what we can do is generate a plot between some of these metrics so that we can easily visualize which threshold is giving us a better result.

The AUC-ROC curve solves just that problem!

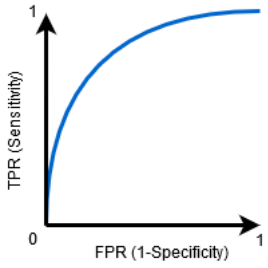
What is the AUC-ROC curve?

The **Receiver Operator Characteristic (ROC)** curve is an evaluation metric for binary classification problems. It is a probability curve that plots the **TPR** against **FPR** at various threshold values and essentially **separates the 'signal' from the 'noise'**. The **Area Under the Curve (AUC)** is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

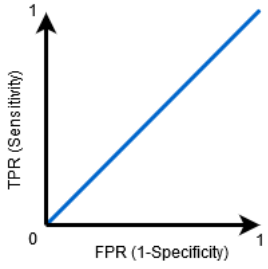
> The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.



When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.



When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.



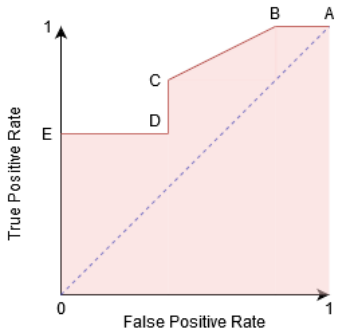
When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning either the classifier is predicting random class or constant class for all the data points.

So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.

How Does the AUC-ROC Curve Work?

In a ROC curve, a higher X-axis value indicates a higher number of False positives than True negatives. While a higher Y-axis value indicates a higher number of True positives than False negatives. So, the choice of the threshold depends on the ability to balance between False positives and False negatives.

Let's dig a bit deeper and understand how our ROC curve would look like for different threshold values and how the specificity and sensitivity would vary.



We can try and understand this graph by generating a confusion matrix for each point corresponding to a threshold and talk about the performance of our classifier:

		Actual	
		+ve	-ve
Predicted	+ve	5	5
	-ve	0	0

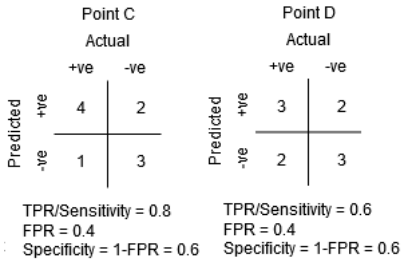
Point A is where the Sensitivity is the highest and Specificity the lowest. This means all the Positive class points are classified correctly and all the Negative class points are classified incorrectly.

In fact, any point on the blue line corresponds to a situation where True Positive Rate is equal to False Positive Rate.

All points above this line correspond to the situation where the proportion of correctly classified points belonging to the Positive class is greater than the proportion of incorrectly classified points belonging to the Negative class.

		Actual	
		+ve	-ve
Predicted	+ve	5	4
	-ve	0	1

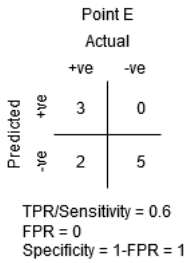
Although Point B has the same Sensitivity as Point A, it has a higher Specificity. Meaning the number of incorrectly Negative class points is lower compared to the previous threshold. This indicates that this threshold is better than the previous one.



Between points C and D, the Sensitivity at point C is higher than point D for the same Specificity. This means, for the same number of incorrectly classified Negative class points, the classifier predicted a higher number of Positive class points. Therefore, the threshold at point C is better than point D.

Now, depending on how many incorrectly classified points we want to tolerate for our classifier, we would choose between point B or C for predicting whether you can defeat me in PUBG or not.

> *"False hopes are more dangerous than fears."–J.R.R. Tolkein*



Point E is where the Specificity becomes highest. Meaning there are no False Positives classified by the model. The model can correctly classify all the Negative class points! We would choose this point if our problem was to give perfect song recommendations to our users.

Going by this logic, can you guess where the point corresponding to a perfect classifier would lie on the graph?

Yes! It would be on the top-left corner of the ROC graph corresponding to the coordinate (0, 1) in the cartesian plane. It is here that both, the Sensitivity and Specificity, would be the highest and the classifier would correctly classify all the Positive and Negative class points.

Understanding the AUC-ROC Curve in Python

Now, either we can manually test the Sensitivity and Specificity for every threshold or let sklearn do the job for us. We're definitely going with the latter!

Let's create our arbitrary data using the sklearn make_classification method:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# generate two class dataset
```

```
X, y = make_classification(n_samples=1000, n_classes=2, n_features=20, random_state=27)

# split into train-test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=27)
```

view raw

AUC-ROC1.py hosted with ❤ by GitHub

I will test the performance of two classifiers on this dataset:

```
# train models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# logistic regression
model1 = LogisticRegression()
# knn
model2 = KNeighborsClassifier(n_neighbors=4)

# fit model
model1.fit(X_train, y_train)
model2.fit(X_train, y_train)

# predict probabilities
pred_prob1 = model1.predict_proba(X_test)
pred_prob2 = model2.predict_proba(X_test)
```

view raw

AUC-ROC2.py hosted with ❤ by GitHub

Sklearn has a very potent method roc_curve() which computes the ROC for your classifier in a matter of seconds! It returns the FPR, TPR, and threshold values:

```
from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, pred_prob2[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)
```

view raw

AUC-ROC3.py hosted with ❤ by GitHub

The AUC score can be computed using the roc_auc_score() method of sklearn:

```
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(y_test, pred_prob1[:,1])
auc_score2 = roc_auc_score(y_test, pred_prob2[:,1])

print(auc_score1, auc_score2)
```

view raw

AUC-ROC4.py hosted with ❤ by GitHub

```
0.9761029411764707 0.9233769727403157
```

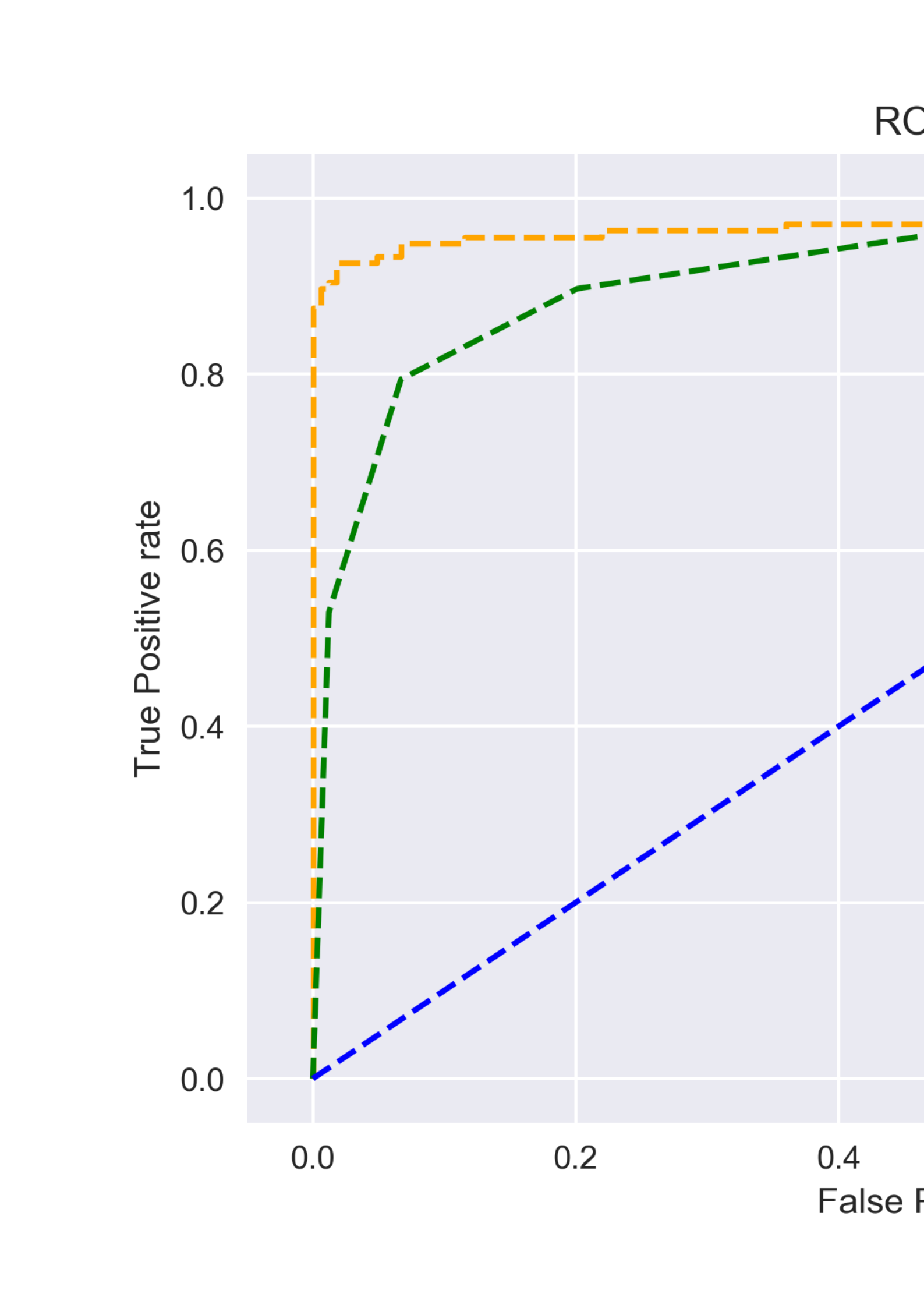
We can also plot the ROC curves for the two algorithms using matplotlib:

```
# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
```

```
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

view raw

AUC-ROC5.py hosted with ❤ by **GitHub**


RO
1.0
0.8
0.6
0.4
0.2
0.0
True Positive rate
0.0
0.2
0.4
False P

It is evident from the plot that the AUC for the Logistic Regression ROC curve is higher than that for the KNN ROC curve. Therefore, we can say that logistic regression did a better job of classifying the positive class in the dataset.

AUC-ROC for Multi-Class Classification

Like I said before, the AUC-ROC curve is only for binary classification problems. But we can extend it to multiclass classification problems by using the One vs All technique.

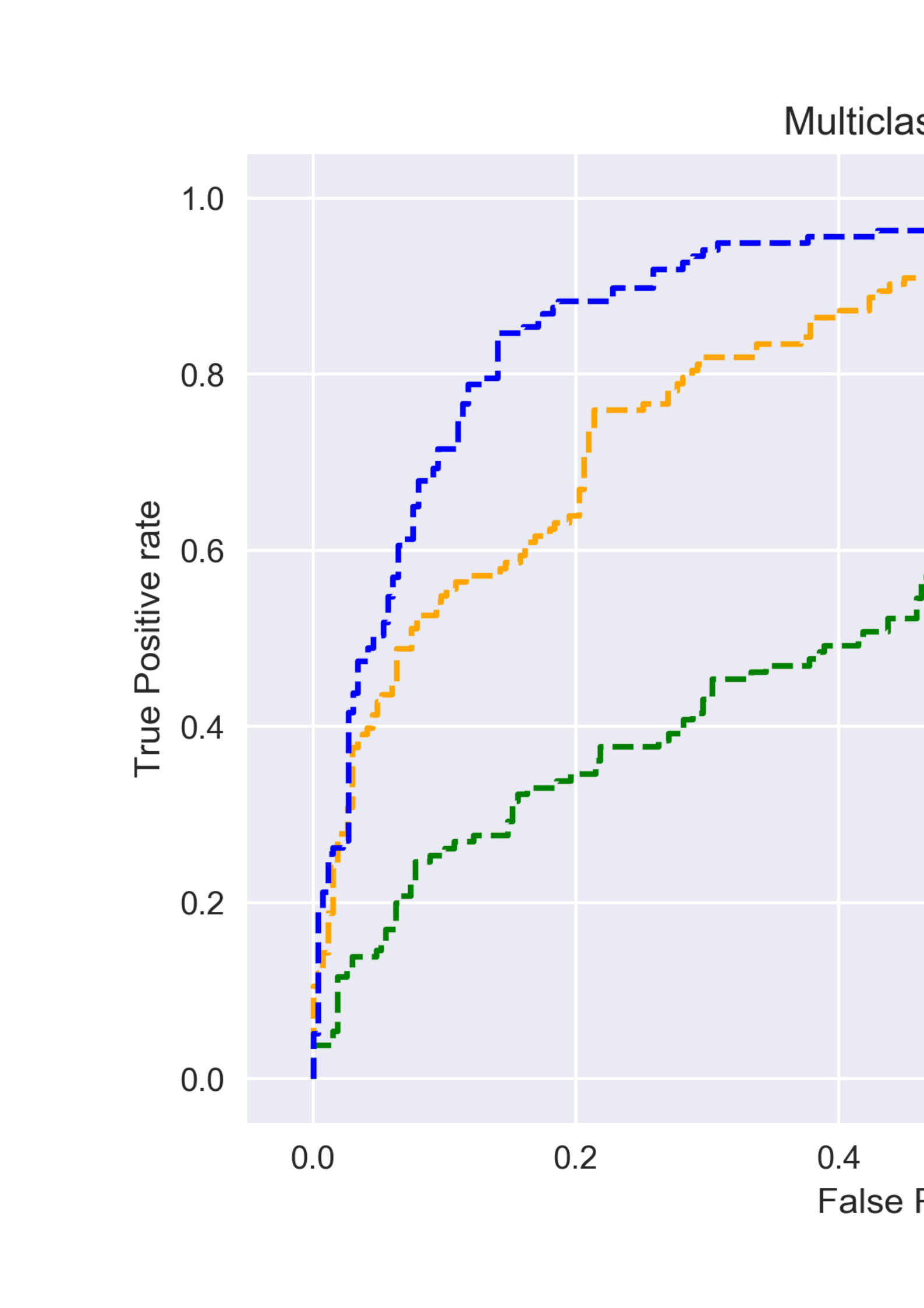
So, if we have three classes 0, 1, and 2, the ROC for class 0 will be generated as classifying 0 against not 0, i.e. 1 and 2. The ROC for class 1 will be generated as classifying 1 against not 1, and so on.

The ROC curve for multi-class classification models can be determined as below:

```
# multi-class classification
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=3, n_features=20, n_informative=3, random_state=42)

# split into train/test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

# fit model
clf = OneVsRestClassifier(LogisticRegression())
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
pred_prob = clf.predict_proba(X_test)

# roc curve for classes
fpr = {}
tpr = {}
thresh ={}

n_class = 3

for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, pred_prob[:,i], pos_label=i)

# plotting
plt.plot(fpr[0], tpr[0], linestyle='--',color='orange', label='Class 0 vs Rest')
plt.plot(fpr[1], tpr[1], linestyle='--',color='green', label='Class 1 vs Rest')
plt.plot(fpr[2], tpr[2], linestyle='--',color='blue', label='Class 2 vs Rest')
plt.title('Multiclass ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.savefig('Multiclass ROC',dpi=300);
```

view raw

AUC-ROC6.py hosted with ❤ by GitHub


Multiclas
True Positive rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
False P

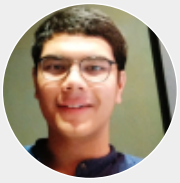
End Notes

I hope you found this article useful in understanding how powerful the AUC-ROC curve metric is in measuring the performance of a classifier. You'll use this a lot in the industry and even in data science or machine learning hackathons. Better get familiar with it!

Going further I would recommend you the following courses that will be useful in building your data science acumen:

- Introduction to Data Science
- Applied Machine Learning

Article Url - https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/



Aniruddha Bhandari

I am on a journey to becoming a data scientist. I love to unravel trends in data, visualize it and predict the future with ML algorithms! But the most satisfying part of this journey is sharing my learnings, from the challenges that I face, with the community to make the world a better place!