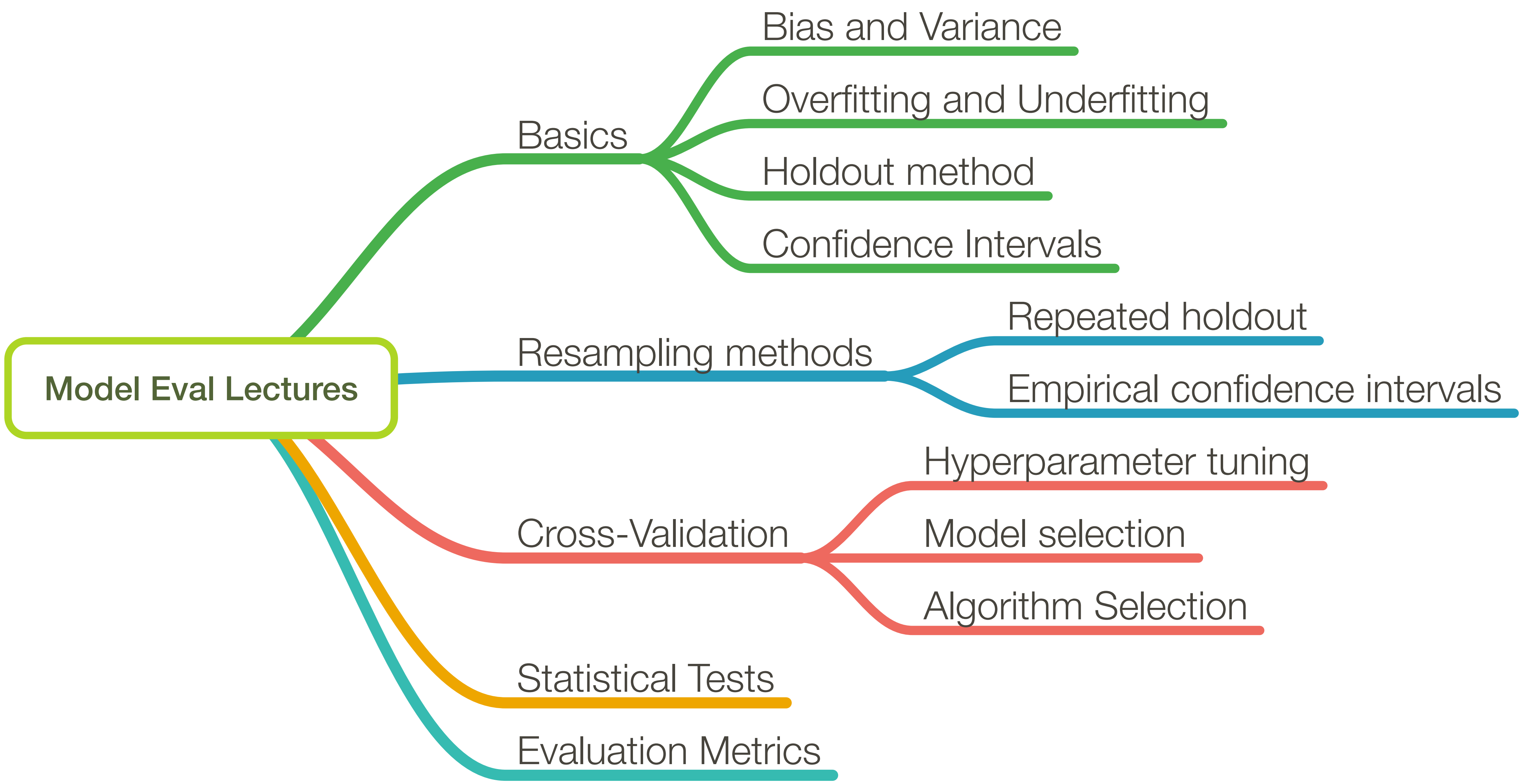
Lecture 12

Model Evaluation 5:
Performance Metrics

STAT 451: Machine Learning, Fall 2021
Sebastian Raschka


Model Eval Lectures
Basics
Bias and Variance
Overfitting and Underfitting
Holdout method
Confidence Intervals
Resampling methods
Repeated holdout
Empirical confidence intervals
Cross-Validation
Hyperparameter tuning
Model selection
Algorithm Selection
Statistical Tests
Evaluation Metrics

1. Confusion Matrix

2. Precision, Recall, and F1 Score

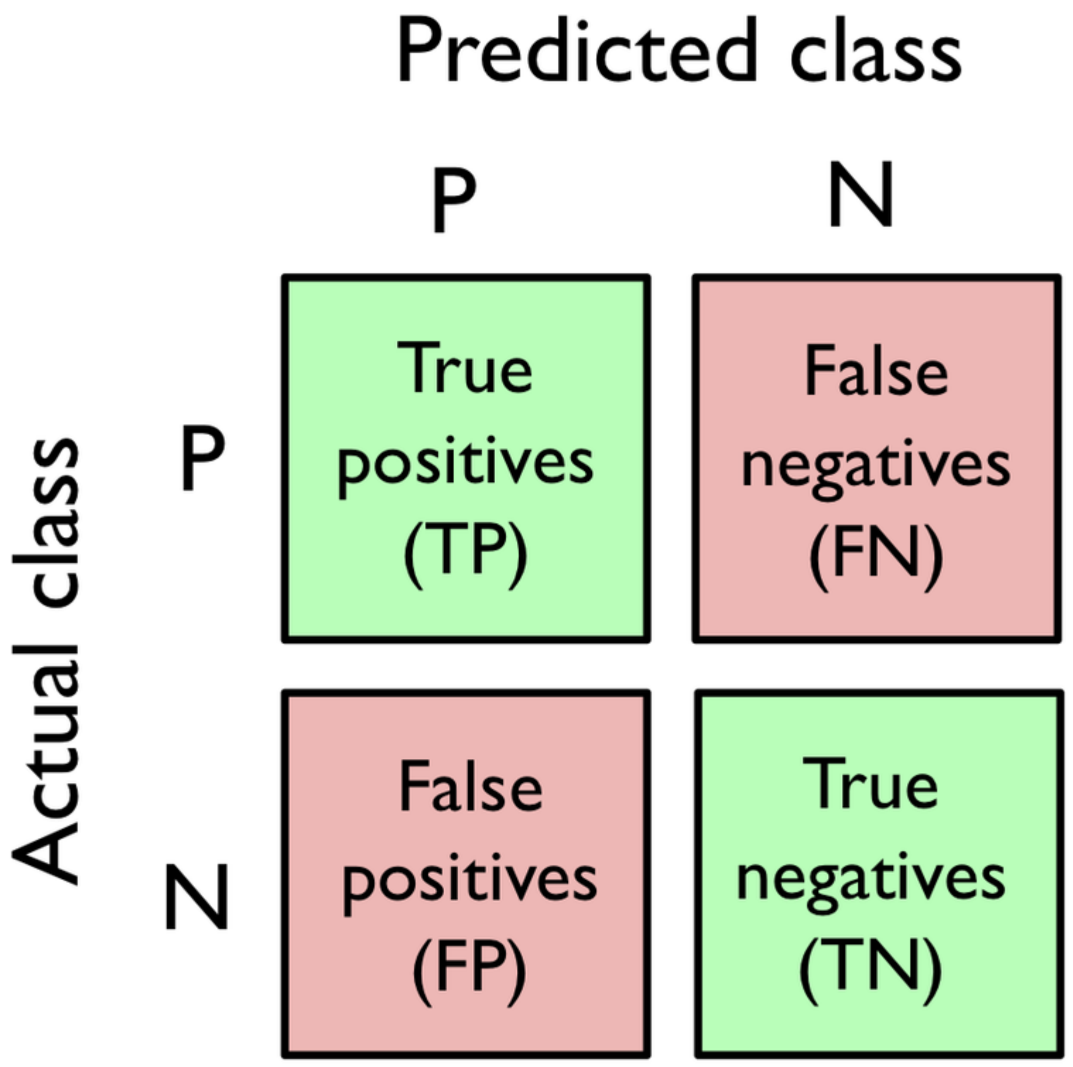
3. Matthews Correlation Coefficient

4. Balanced Accuracy

5. ROC

6. Extending Binary Metrics to Multi-class Settings

2x2 Confusion Matrix



$$ERR = \frac{FP + FN}{FP + FN + TP + TN} = 1 - ACC$$

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

Loading the Breast Cancer Wisconsin dataset

- In the Breast Cancer Wisconsin dataset, the firt column in this dataset stores the unique ID numbers of patients
- The second column stores the corresponding cancer diagnoses (M = malignant, B = benign)
- Columns 3-32 contain features that were extracted from digitized images of the nuclei of the cancer cells, which can be used to build a model to predict whether a tumor is benign or malignant.
- The Breast Cancer Wisconsin dataset has been deposited in the UCI Machine Learning Repository, and more detailed information about this dataset can be found at https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

[1]:
```
import pandas as pd

df = pd.read_csv('https://archive.ics.uci.edu/ml/'
                 'machine-learning-databases'
                 '/breast-cancer-wisconsin/wdbc.data', header=None)

df.head()
```

[1]:

	0	1	2	3	4	5	6	7	8	9	...	22	23	24	25
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.0
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.0
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.7
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.0

5 rows × 32 columns

[2]:
```
df.shape
```

[2]:
```
(569, 32)
```

- First, we are converting the class labels from a string format into integers

```
[3]: from sklearn.preprocessing import LabelEncoder

     X = df.loc[:, 2:].values
     y = df.loc[:, 1].values
     le = LabelEncoder()
     y = le.fit_transform(y)
     le.classes_
```

```
[3]: array(['B', 'M'], dtype=object)
```

- Here, class "M" (malignant cancer) will be converted to class 1, and "B" will be converted into class 0 (the order the class labels are mapped depends on the alphabetical order of the string labels)

```
[4]: le.transform(['M', 'B'])
```

```
[4]: array([1, 0])
```

- Next, we split the data into 80% training data and 20% test data, using a stratified split

```
[5]: from sklearn.model_selection import train_test_split

     X_train, X_test, y_train, y_test = \
         train_test_split(X, y,
                          test_size=0.20,
                          stratify=y,
                          random_state=1)
```

1) Confusion Matrix

More examples at

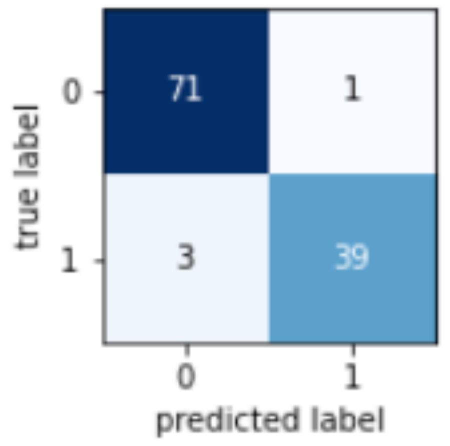
- http://rasbt.github.io/mlxtend/user_guide/evaluate/confusion_matrix/
- and http://rasbt.github.io/mlxtend/user_guide/plotting/plot_confusion_matrix/

```
[6]: from sklearn.preprocessing import StandardScaler
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.pipeline import make_pipeline

     from mlxtend.evaluate import confusion_matrix
     #or
     #from sklearn.metrics import confusion_matrix



     pipe_knn = make_pipeline(StandardScaler(),
                              KNeighborsClassifier(n_neighbors=5))

     pipe_knn.fit(X_train, y_train)

     y_pred = pipe_knn.predict(X_test)

     confmat = confusion_matrix(y_test, y_pred)

     print(confmat)
```

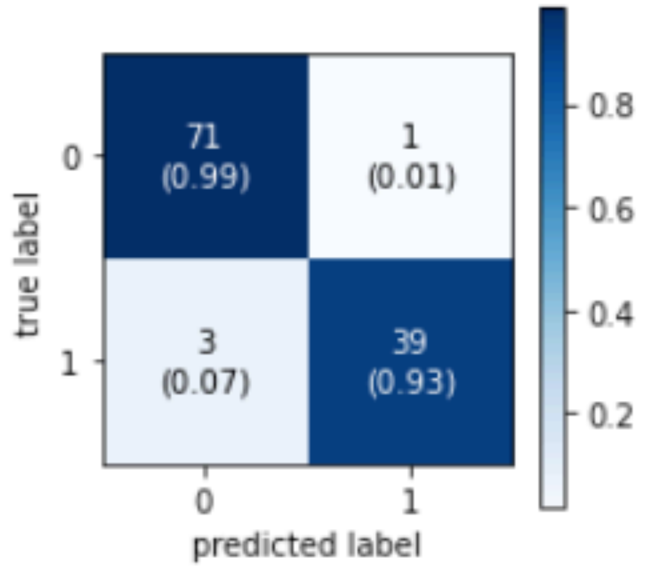
```
[[71  1]
 [ 3 39]]
```

Visualizing a Confusion Matrix

```
[9]: from mlxtend.plotting import plot_confusion_matrix
     import matplotlib.pyplot as plt

     fig, ax = plot_confusion_matrix(conf_mat=confmat, figsize=(2, 2))
     plt.show()
```



```
[10]: fig, ax = plot_confusion_matrix(conf_mat=confmat,
                                      show_absolute=True,
                                      show_normed=True,
                                      colorbar=True,
                                      figsize=(3, 3))
      plt.show()
```



False Positive Rate and False Negative Rate

* Relevant later for ROC

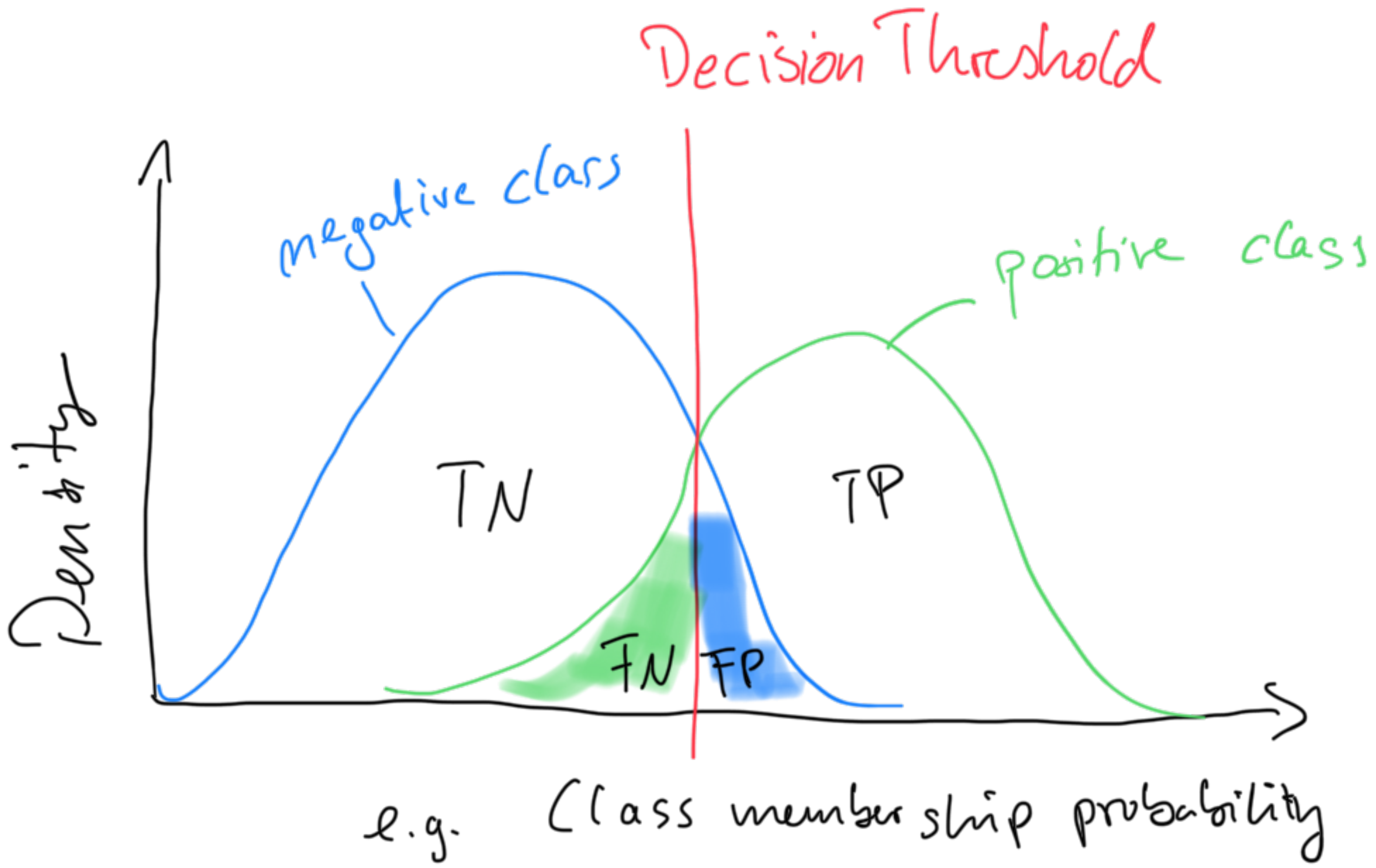
$$\text{TPR}^{*} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}} = 1 - \text{FNR}$$

$$\text{FPR}^{*} = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP} + \text{TN}} = 1 - \text{TNR}$$

$$\text{FNR} = \frac{\text{FN}}{\text{P}} = \frac{\text{FN}}{\text{FN} + \text{TP}} = 1 - \text{TPR}$$

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

False Positive Rate and False Negative Rate



Confusion Matrix for Multi-Class Settings

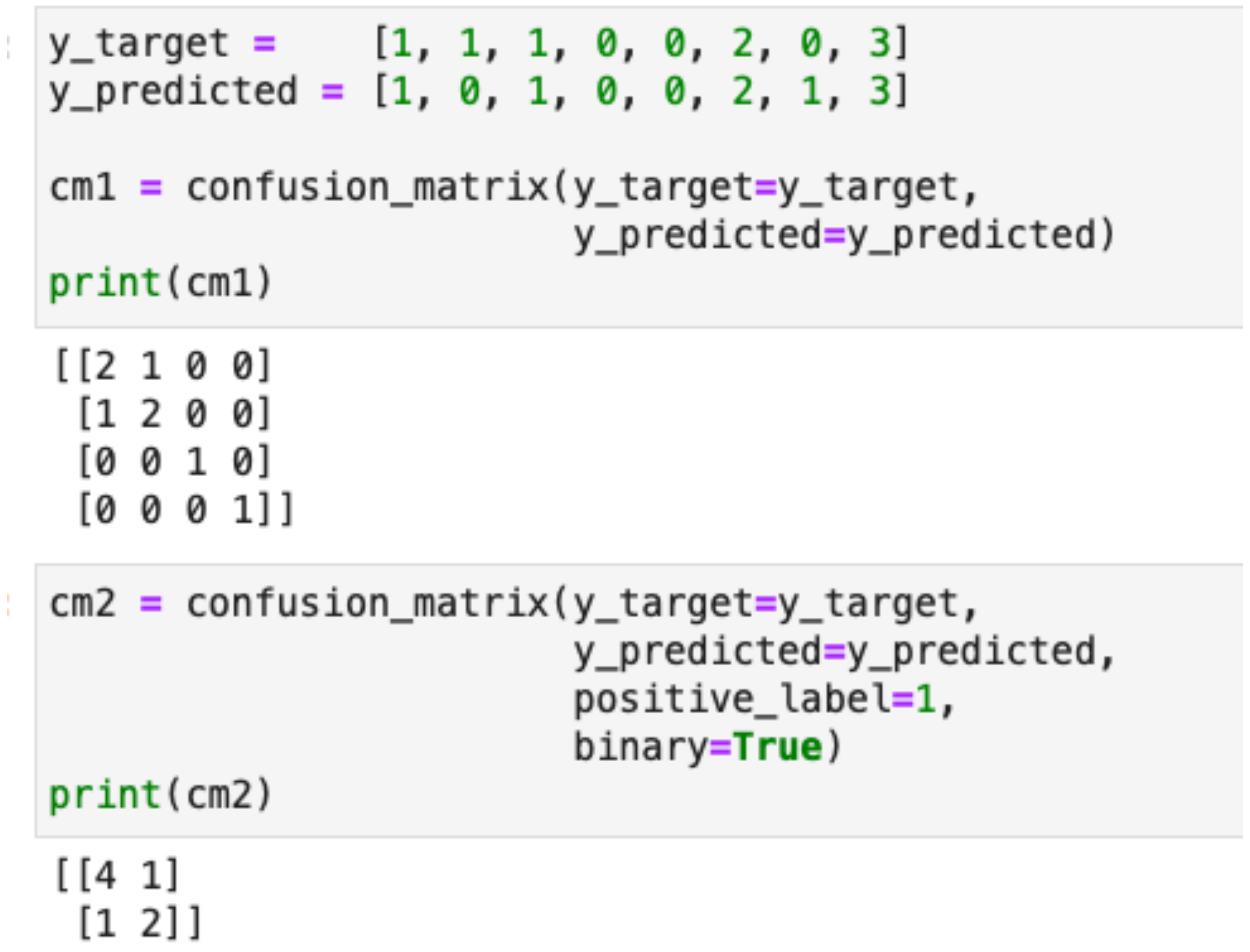
Predicted Labels

True Labels

	Class 0	Class 1	Class 2
Class 0	T(0,0)		
Class 1		T(1,1)	
Class 2			T(2,2)

Confusions matrices are traditionally for binary class problems but we can be readily generalized it to multi-class settings

Multiclass to Binary



```
y_target =    [1, 1, 1, 0, 0, 2, 0, 3]
y_predicted = [1, 0, 1, 0, 0, 2, 1, 3]

cm1 = confusion_matrix(y_target=y_target,
                       y_predicted=y_predicted)
print(cm1)
```

```
[[2 1 0 0]
 [1 2 0 0]
 [0 0 1 0]
 [0 0 0 1]]
```

```
cm2 = confusion_matrix(y_target=y_target,
                       y_predicted=y_predicted,
                       positive_label=1,
                       binary=True)
print(cm2)
```

```
[[4 1]
 [1 2]]
```

1. Confusion Matrix

2. Precision, Recall, and F1 Score

3. Matthews Correlation Coefficient

4. Balanced Accuracy

5. ROC

6. Extending Binary Metrics to Multi-class Settings

Precision, Recall, and F1 Score

$$PRE = \frac{TP}{TP + FP}$$

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

$$F_1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC}$$

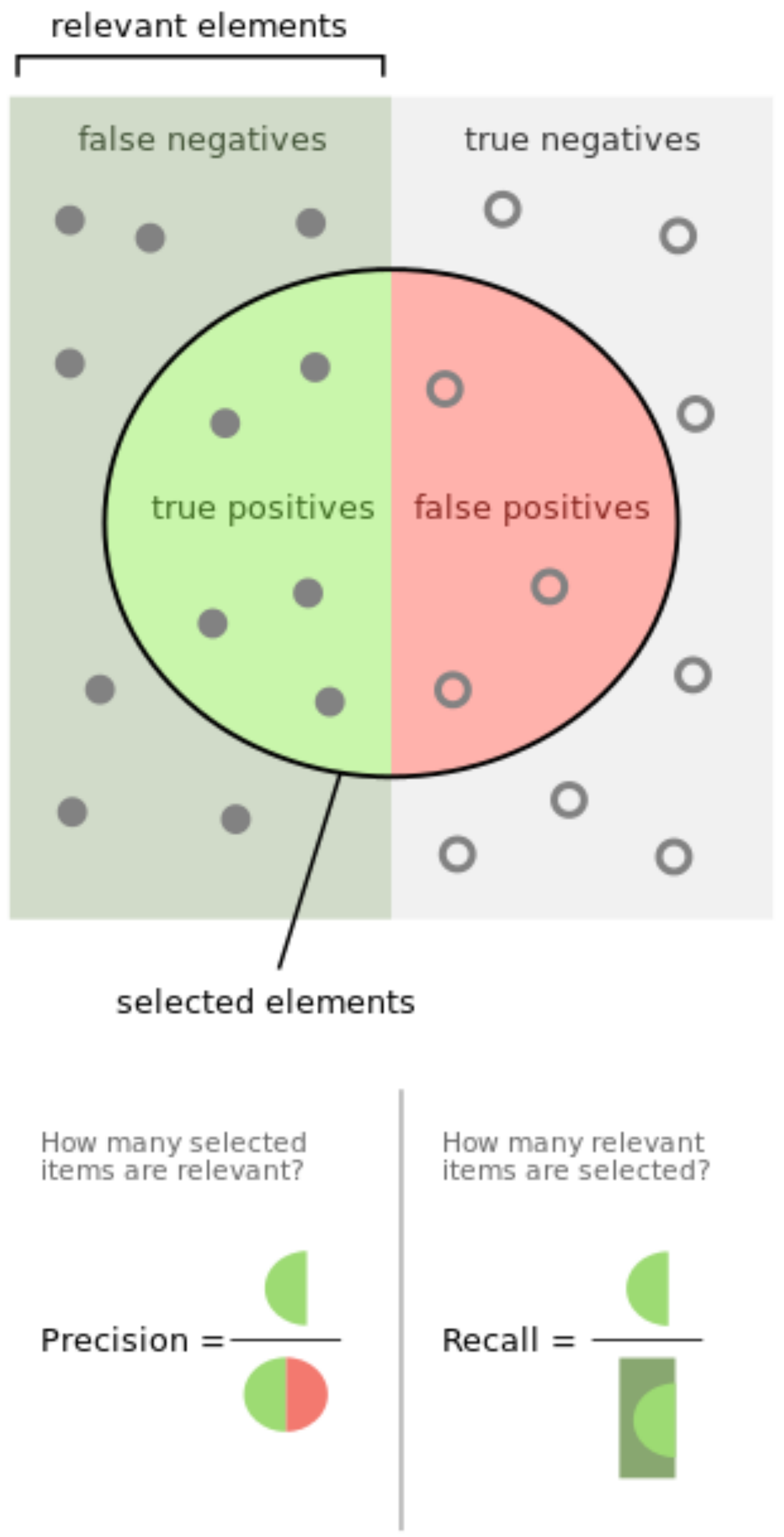
- Terms that are more popular in Information Technology
- Recall is actually just another term for True Positive Rate (or "sensitivity")

Precision and Recall

$$PRE = \frac{TP}{TP + FP}$$

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

$$F_1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC}$$



https://en.wikipedia.org/wiki/Precision_and_recall

Sensitivity and Specificity

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

$$SPC = TNR = \frac{TN}{N} = \frac{TN}{FP + TN}$$

Sensitivity (SEN) measures the recovery rate of the Positives and complimentary, *Specificity (SPC)* measures the recovery rate of the Negatives.

2) Precision, Recall, F1 Score

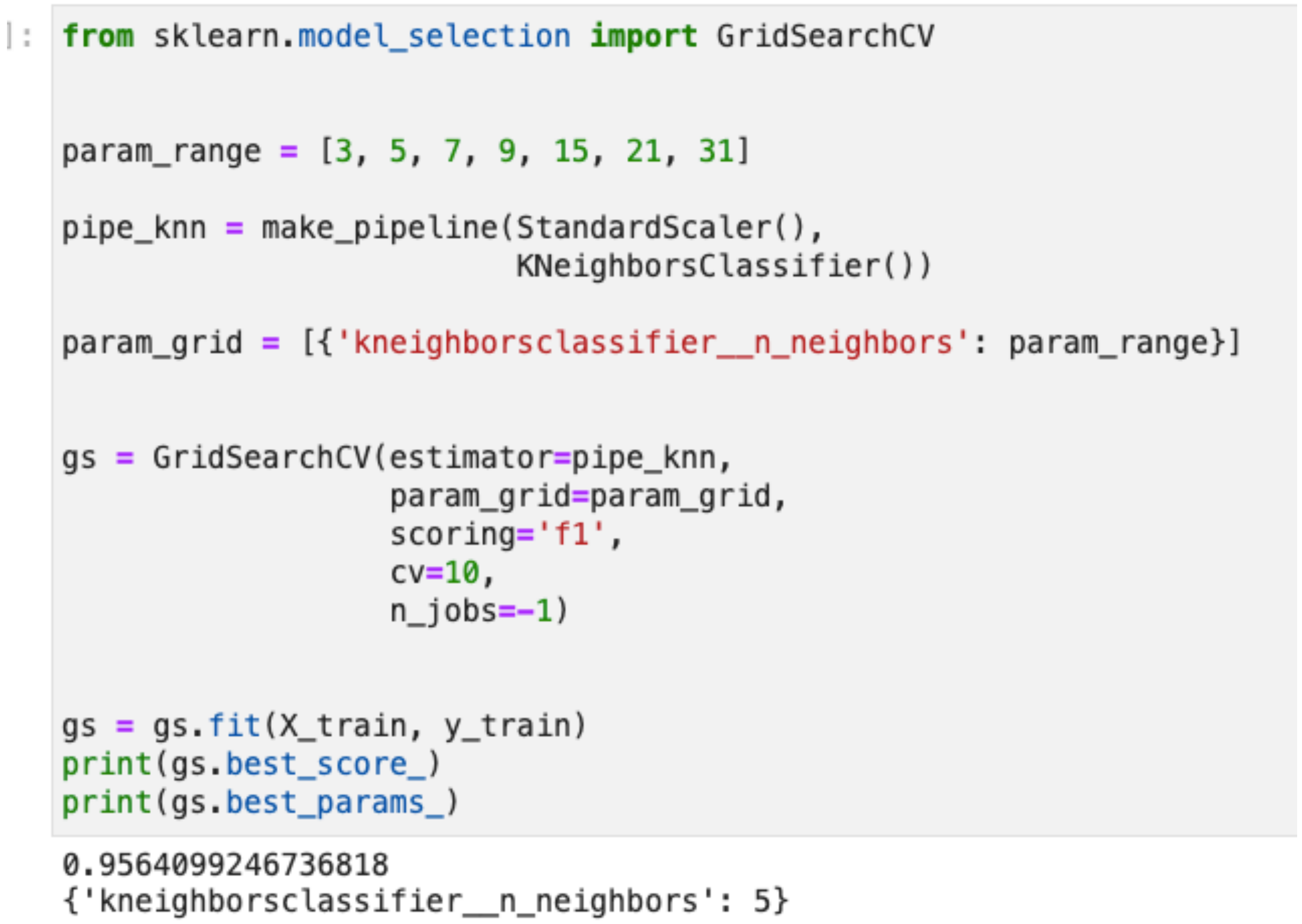
```python
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from mlxtend.evaluate import confusion_matrix


pipe_knn = make_pipeline(StandardScaler(),
                         KNeighborsClassifier(n_neighbors=5))

pipe_knn.fit(X_train, y_train)

y_pred = pipe_knn.predict(X_test)

confmat = confusion_matrix(y_test, y_pred)

print(confmat)
```

```
[[71  1]
 [ 3 39]]
```

```python
from sklearn.metrics import accuracy_score, precision_score, \
                            recall_score, f1_score, matthews_corrcoef


print('Accuracy: %.3f' % accuracy_score(y_true=y_test, y_pred=y_pred))
print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
print('MCC: %.3f' % matthews_corrcoef(y_true=y_test, y_pred=y_pred))
```

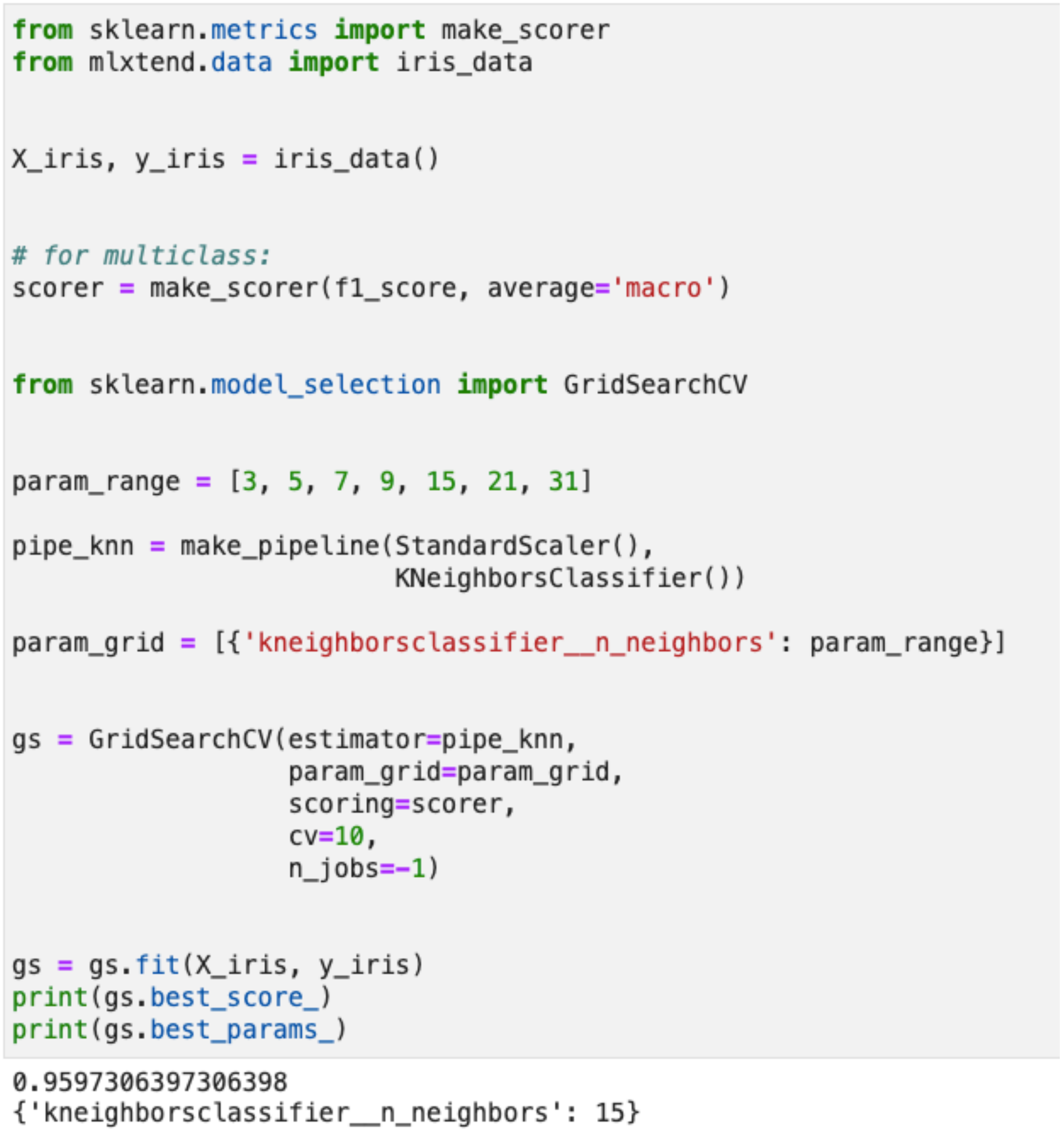
```
Accuracy: 0.965
Precision: 0.975
Recall: 0.929
F1: 0.951
```

3) Using those Metrics in GridSearch



```
[ ]: from sklearn.model_selection import GridSearchCV


     param_range = [3, 5, 7, 9, 15, 21, 31]

     pipe_knn = make_pipeline(StandardScaler(),
                              KNeighborsClassifier())

     param_grid = [{'kneighborsclassifier__n_neighbors': param_range}]


     gs = GridSearchCV(estimator=pipe_knn,
                       param_grid=param_grid,
                       scoring='f1',
                       cv=10,
                       n_jobs=-1)


     gs = gs.fit(X_train, y_train)
     print(gs.best_score_)
     print(gs.best_params_)
```

```
0.9564099246736818
{'kneighborsclassifier__n_neighbors': 5}
```



```python
from sklearn.metrics import make_scorer
from mlxtend.data import iris_data


X_iris, y_iris = iris_data()


# for multiclass:
scorer = make_scorer(f1_score, average='macro')


from sklearn.model_selection import GridSearchCV


param_range = [3, 5, 7, 9, 15, 21, 31]

pipe_knn = make_pipeline(StandardScaler(),
                         KNeighborsClassifier())

param_grid = [{'kneighborsclassifier__n_neighbors': param_range}]


gs = GridSearchCV(estimator=pipe_knn,
                  param_grid=param_grid,
                  scoring=scorer,
                  cv=10,
                  n_jobs=-1)


gs = gs.fit(X_iris, y_iris)
print(gs.best_score_)
print(gs.best_params_)
```

```
0.9597306397306398
{'kneighborsclassifier__n_neighbors': 15}
```

1. Confusion Matrix

2. Precision, Recall, and F1 Score

3. **Matthews Correlation Coefficient**

4. Balanced Accuracy

5. ROC

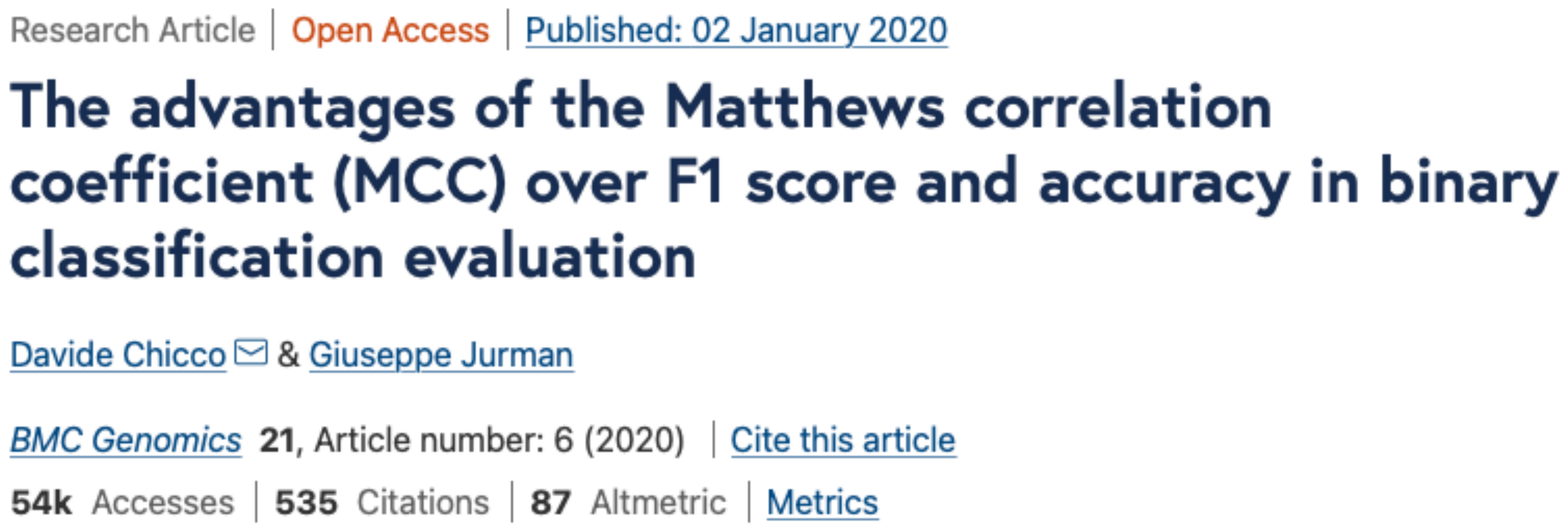
6. Extending Binary Metrics to Multi-class Settings

Matthew's Correlation Coefficient

- Matthews correlation coefficient (MCC) was first formulated by Brian W. Matthews [1] in 1975 to assess the performance of protein secondary structure predictions
- The MCC can be understood as a specific case of a linear correlation coefficient (Pearson r) for a binary classification setting
- Considered as especially useful in unbalanced class settings
- The previous metrics take values in the range between 0 (worst) and 1 (best)
- The MCC is bounded between the range 1 (perfect correlation between ground truth and predicted outcome) and -1 (inverse or negative correlation) — a value of 0 denotes a random prediction.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{10}$$

[1] Brian W Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta (BBA)- Protein Structure, 405(2):442–451, 1975.



Research Article | Open Access | Published: 02 January 2020

The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation

Davide Chicco & Giuseppe Jurman

BMC Genomics **21**, Article number: 6 (2020) | Cite this article

54k Accesses | **535** Citations | **87** Altmetric | Metrics

https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-019-6413-7

Precision, TP / (TP+FP), tells you something about how many out of the predicted positives are actually positives.

Recall, TP / (TP+FN), tells you how many of the actual positives you captured.

The F1 score, 2(Pre x Rec) / (Pre + Rec) provides a harmonic mean.

	Predicted positive	Predicted negative
Actual positive	True positives (TP)	False negatives (FN)
Actual negative	False positives (FP)	True negatives (TN)

Neither F1, precision, or recall takes the true negatives into account.

Accuracy, (TP+TN) / (TP+TN+FP+FN) actually does this, but the problem with acc is that it is defunct for imbalanced data.

That is, acc may be very close to the no information rate (majority class prediction)

	Predicted positive	Predicted negative
Actual positive	True positives (TP)	False negatives (FN)
Actual negative	False positives (FP)	True negatives (TN)

Another aspect worth noting about F1 score is that it is not symmetric, that is, if you swap positive and negative class, results can differ. Although, this can be addressed via micro/macro averaging

Matthews correlation coefficient (MCC) is a measure that takes all elements of a confusion matrix into account and doesn't require micro/macro averaging in order to be symmetric. Also, it doesn't suffer from class imbalance like accuracy

$$\mathrm{MCC} = \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP}) \cdot (\mathrm{TP}+\mathrm{FN}) \cdot (\mathrm{TN}+\mathrm{FP}) \cdot (\mathrm{TN}+\mathrm{FN})}}$$

	Predicted positive	Predicted negative
Actual positive	True positives (TP)	False negatives (FN)
Actual negative	False positives (FP)	True negatives (TN)

The MCC ranges between -1 and 1, which can be interpreted as follows:

-1: total misclassification
0: random prediction (coin tossing classifier)
+1: perfect classification.

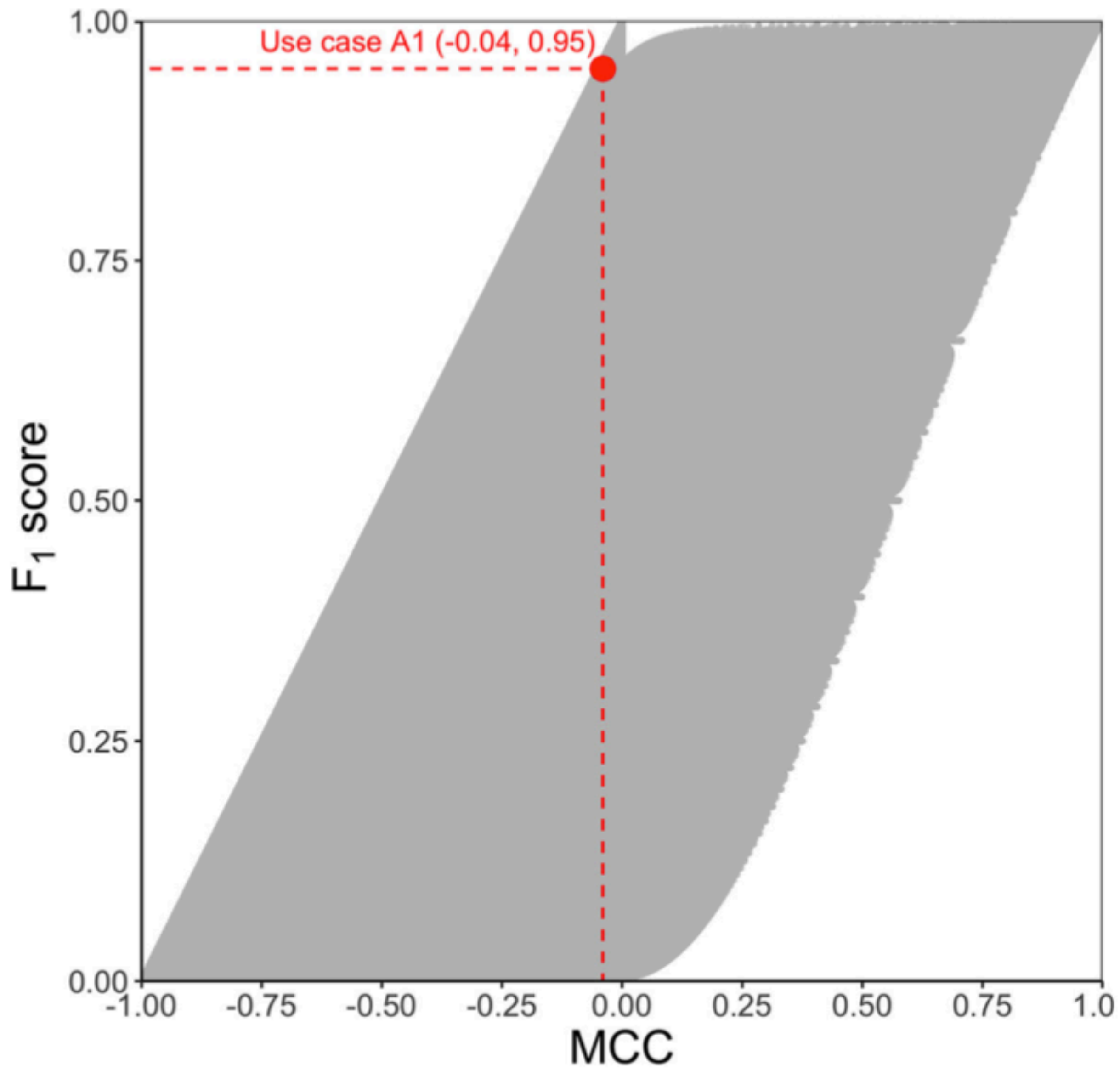
$$\mathrm{MCC} = \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP}) \cdot (\mathrm{TP} + \mathrm{FN}) \cdot (\mathrm{TN} + \mathrm{FP}) \cdot (\mathrm{TN} + \mathrm{FN})}}$$

You can rescale it, too, e.g., $MCC_{scaled} = \frac{MCC + 1}{2}$

Then, it will range between 0 and 1

The researchers did an interesting experiment, where they created a scatterplot of 21,084,251 possible confusion matrices from a dataset of 500 examples. Interestingly, for a given MCC value, the F1 score can vary widely (actually, by its whole range, [0, 1])

https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-019-6413-7



The authors compiled a list of "special" cases. In the list below, in scenarios

A2: only 5 out of 70 sick patients predicted
B2: only 10 out of 50 sick patients
C1: only 1 healthy out of 90 recognized

ACC and F1 can also detect the low performance.

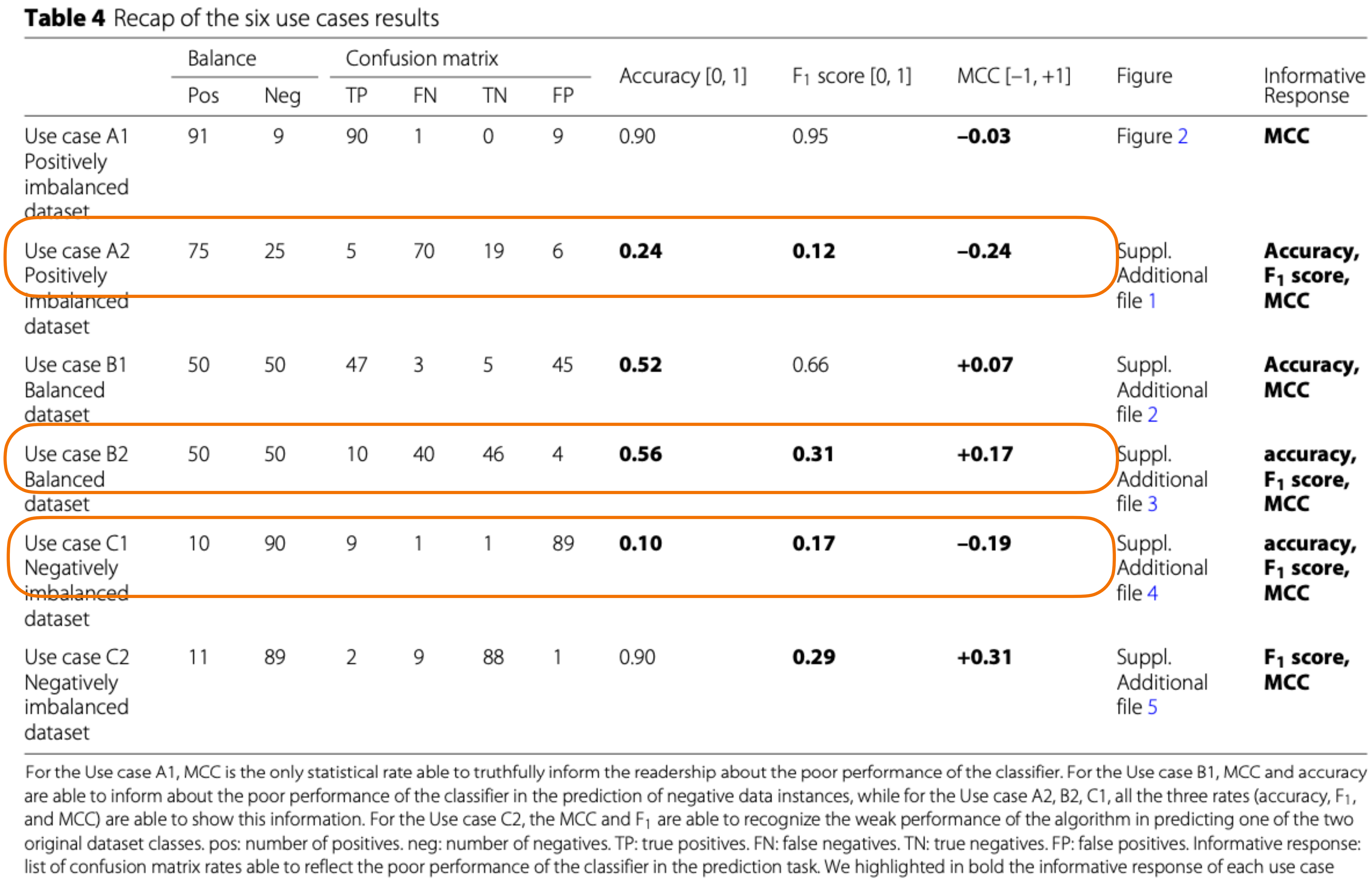


Table 4 Recap of the six use cases results

	Balance		Confusion matrix				Accuracy [0, 1]	F_1 score [0, 1]	MCC [−1, +1]	Figure	Informative Response
	Pos	Neg	TP	FN	TN	FP					
Use case A1 Positively imbalanced dataset	91	9	90	1	0	9	0.90	0.95	**−0.03**	Figure 2	**MCC**
Use case A2 Positively imbalanced dataset	75	25	5	70	19	6	**0.24**	**0.12**	**−0.24**	Suppl. Additional file 1	**Accuracy, F_1 score, MCC**
Use case B1 Balanced dataset	50	50	47	3	5	45	**0.52**	0.66	**+0.07**	Suppl. Additional file 2	**Accuracy, MCC**
Use case B2 Balanced dataset	50	50	10	40	46	4	**0.56**	**0.31**	**+0.17**	Suppl. Additional file 3	**accuracy, F_1 score, MCC**
Use case C1 Negatively imbalanced dataset	10	90	9	1	1	89	**0.10**	**0.17**	**−0.19**	Suppl. Additional file 4	**accuracy, F_1 score, MCC**
Use case C2 Negatively imbalanced dataset	11	89	2	9	88	1	0.90	**0.29**	**+0.31**	Suppl. Additional file 5	**F_1 score, MCC**

For the Use case A1, MCC is the only statistical rate able to truthfully inform the readership about the poor performance of the classifier. For the Use case B1, MCC and accuracy are able to inform about the poor performance of the classifier in the prediction of negative data instances, while for the Use case A2, B2, C1, all the three rates (accuracy, F_1, and MCC) are able to show this information. For the Use case C2, the MCC and F_1 are able to recognize the weak performance of the algorithm in predicting one of the two original dataset classes. pos: number of positives. neg: number of negatives. TP: true positives. FN: false negatives. TN: true negatives. FP: false positives. Informative response: list of confusion matrix rates able to reflect the poor performance of the classifier in the prediction task. We highlighted in bold the informative response of each use case

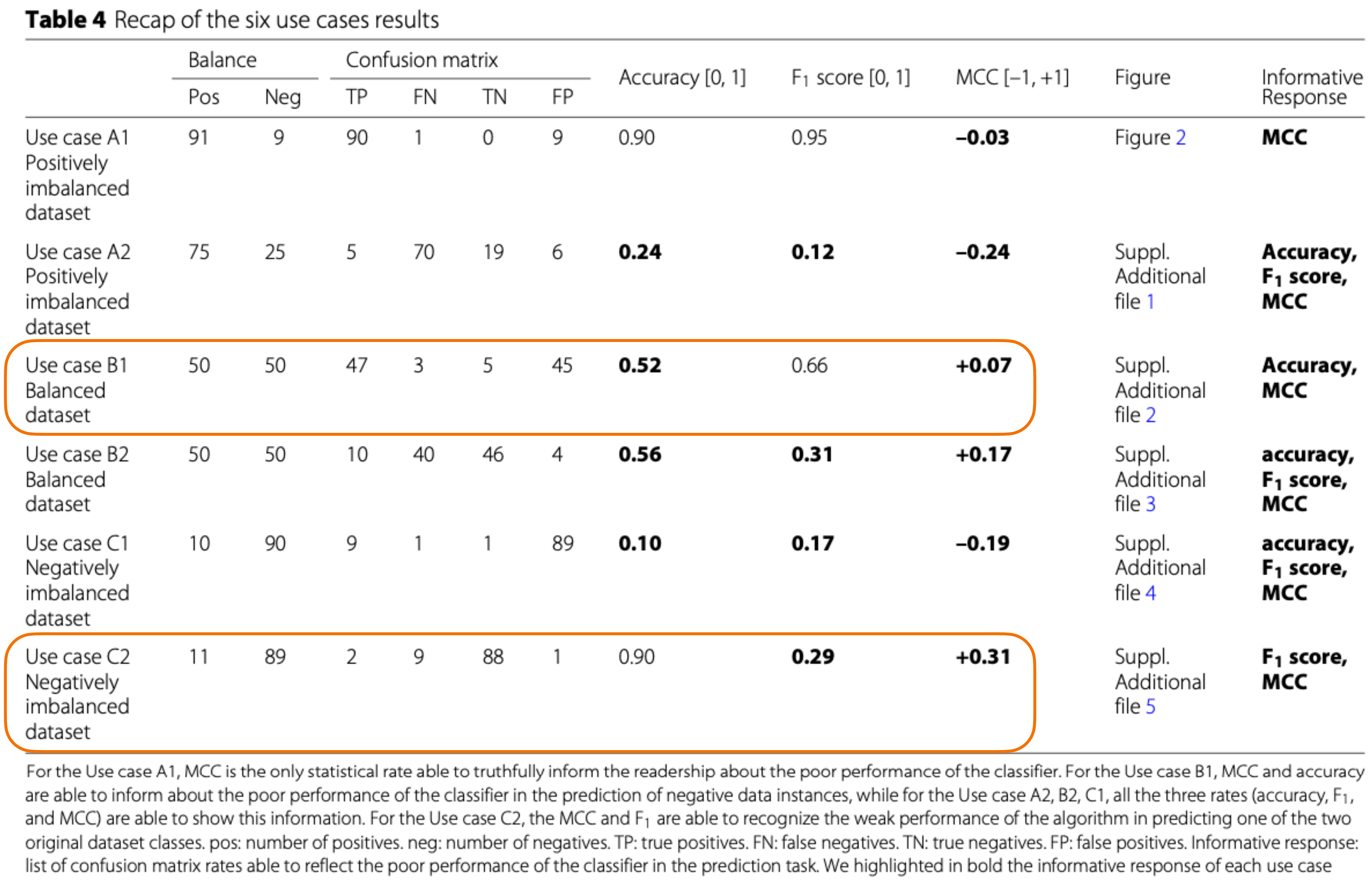


Table 4 Recap of the six use cases results

	Balance		Confusion matrix				Accuracy [0, 1]	F_1 score [0, 1]	MCC [−1, +1]	Figure	Informative Response
	Pos	Neg	TP	FN	TN	FP					
Use case A1 Positively imbalanced dataset	91	9	90	1	0	9	0.90	0.95	**−0.03**	Figure 2	**MCC**
Use case A2 Positively imbalanced dataset	75	25	5	70	19	6	**0.24**	**0.12**	**−0.24**	Suppl. Additional file 1	**Accuracy, F_1 score, MCC**
Use case B1 Balanced dataset	50	50	47	3	5	45	**0.52**	0.66	**+0.07**	Suppl. Additional file 2	**Accuracy, MCC**
Use case B2 Balanced dataset	50	50	10	40	46	4	**0.56**	**0.31**	**+0.17**	Suppl. Additional file 3	**accuracy, F_1 score, MCC**
Use case C1 Negatively imbalanced dataset	10	90	9	1	1	89	**0.10**	**0.17**	**−0.19**	Suppl. Additional file 4	**accuracy, F_1 score, MCC**
Use case C2 Negatively imbalanced dataset	11	89	2	9	88	1	0.90	**0.29**	**+0.31**	Suppl. Additional file 5	**F_1 score, MCC**

For the Use case A1, MCC is the only statistical rate able to truthfully inform the readership about the poor performance of the classifier. For the Use case B1, MCC and accuracy are able to inform about the poor performance of the classifier in the prediction of negative data instances, while for the Use case A2, B2, C1, all the three rates (accuracy, F_1, and MCC) are able to show this information. For the Use case C2, the MCC and F_1 are able to recognize the weak performance of the algorithm in predicting one of the two original dataset classes. pos: number of positives. neg: number of negatives. TP: true positives. FN: false negatives. TN: true negatives. FP: false positives. Informative response: list of confusion matrix rates able to reflect the poor performance of the classifier in the prediction task. We highlighted in bold the informative response of each use case

either ACC or F1 score failed to recognize the poor performance. The bottom line is. MCC is relatively robust as it was able to predict the poor performance in all 6 scenarios.

Regular Accuracy (Binary Classification)

$$ACC = \frac{TP + TN}{n}$$

	Predicted positive	Predicted negative
Actual positive	**True positives (TP)**	False negatives (FN)
Actual negative	False positives (FP)	**True negatives (TN)**

Regular Accuracy (Multiclass)

$$ACC = \frac{T}{n}$$

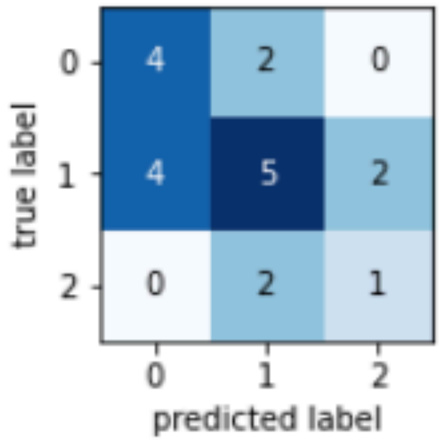
	Class 1	Class 2	Class 3
Class 1	T(1, 1)		
Class 2		T(2, 2)	
Class 3			T(3, 3)

One way of computing the balanced accuracy ...

... is by considering one class as the positive class and the remaining classes as the negative class.

This is called "average per-class accuracy"

```
from mlxtend.evaluate import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix


true =      [2, 2, 0, 0, 1, 2, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,]
predicted = [1, 1, 1, 1, 1, 2, 0, 0, 0, 0, 0, 0, 1, 0, 2, 2, 0, 1, 1, 1]

cm = confusion_matrix(y_target=true,
                      y_predicted=predicted)
fig, ax = plot_confusion_matrix(conf_mat=cm, figsize=(2, 2))
```



	0	1	2
0	4	2	0
1	4	5	2
2	0	2	1

```
Acc = 10/20
Acc
```

```
0.5
```

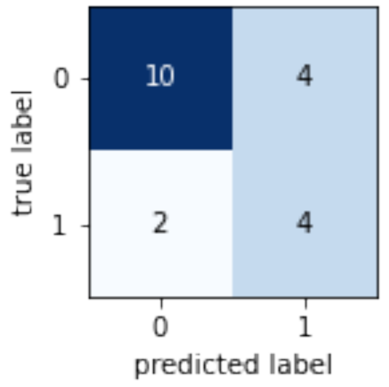
```
from mlxtend.evaluate import accuracy_score

accuracy_score(true, predicted)
```

```
0.5
```

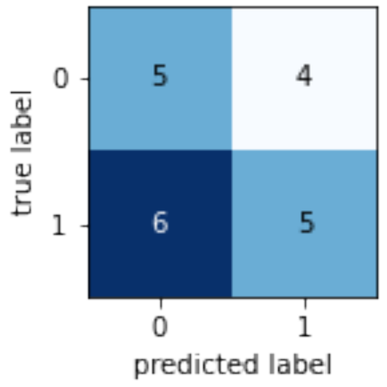
```
accuracy_score(true, predicted, method='average')
```

```
0.6666666666666666
```

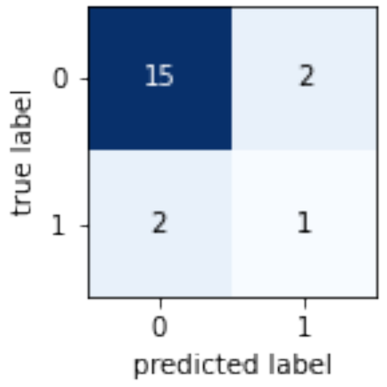
Binarize:



	0	1
0	10	4
1	2	4



	0	1
0	5	4
1	6	5



	0	1
0	15	2
1	2	1

positive class = 0 positive class = 1 positive class = 2

Average per class accuracy: (14/20 + 10/20 + 16/20) / 3 = 0.6666666666666666

The more common way to compute the balanced accuracy ...

... is as the arithmetic mean of the true positive rate (TPR) and true negative rate (TNR).

For the multi-class case we average over the TPRs.

Binary case:

$$\text{balanced accuracy} = \frac{1}{2}(TPR + TNR)$$

$$= \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

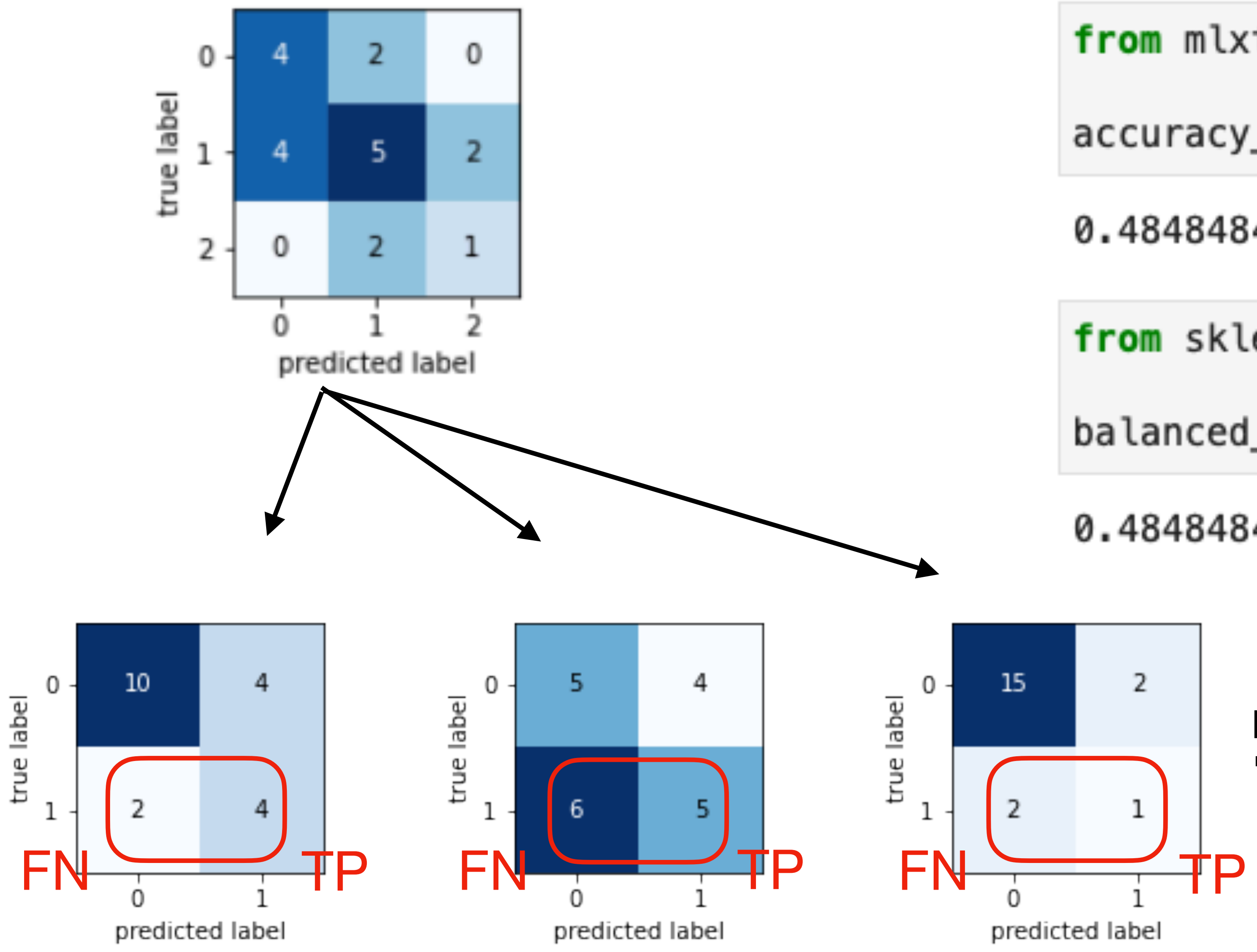
Multi-class case:

```
from mlxtend.evaluate import accuracy_score

accuracy_score(true, predicted, method='balanced')
```

```
0.4848484848484848
```

```
from sklearn.metrics import balanced_accuracy_score

balanced_accuracy_score(true, predicted)
```

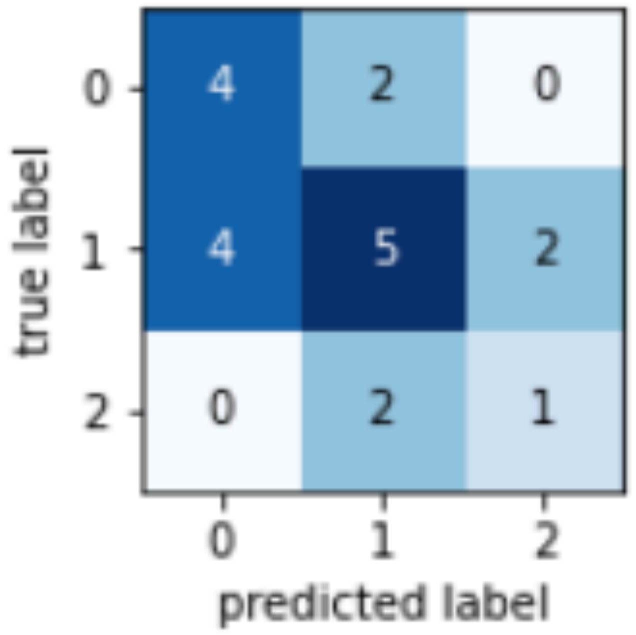
```
0.4848484848484848
```

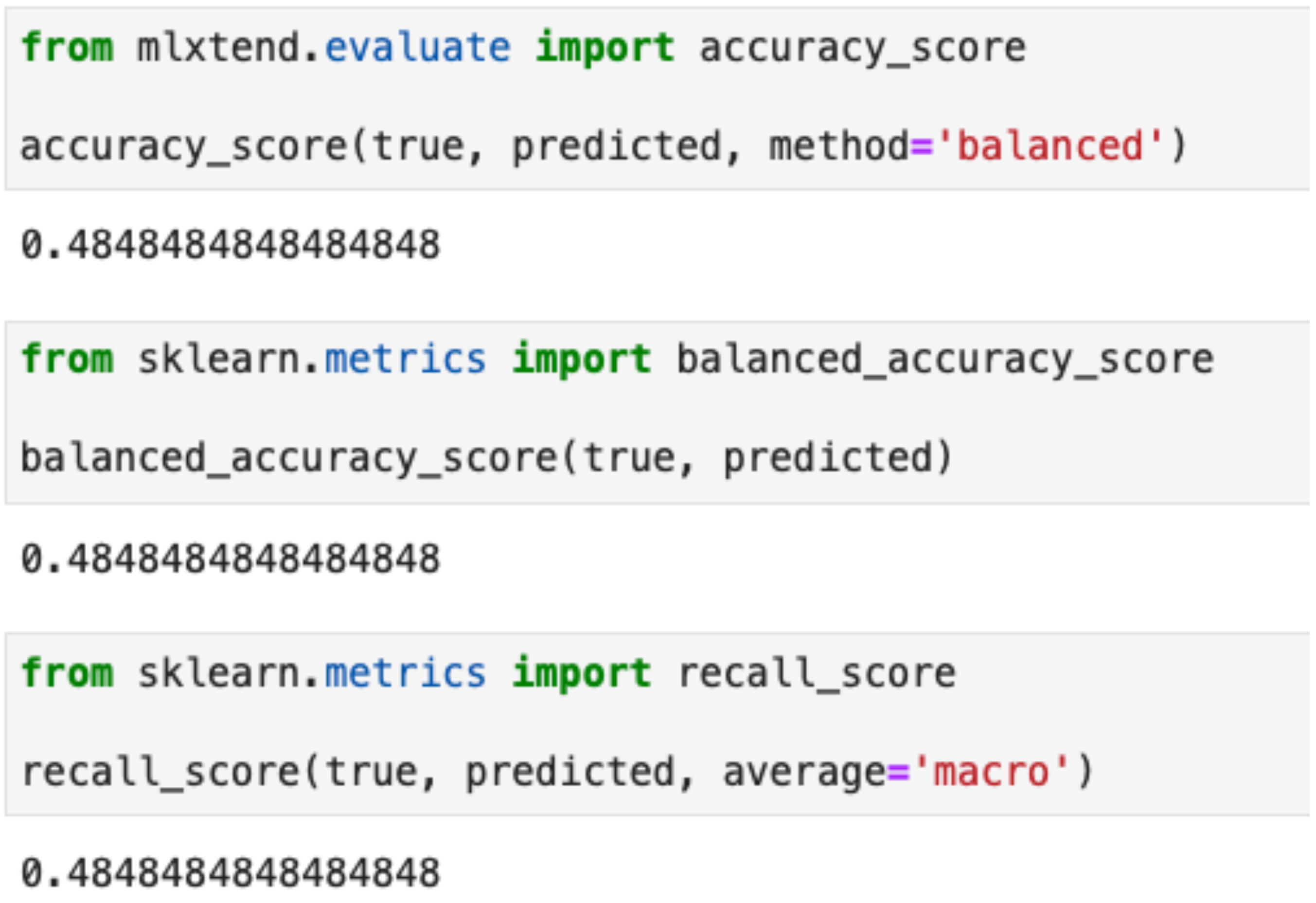


Note that for the calculation of FP and FN, "1" is the positive label

((4/(2+4)) + (5/(5+6)) + (1/(1+2)))/ 3 = 4848484848

On a side note, the true positive rate (TPR) is also known as "sensitivity" or "recall." So, in this case, we can think of the balanced accuracy as the macro-averaged recall:





```
from mlxtend.evaluate import accuracy_score

accuracy_score(true, predicted, method='balanced')
```

```
0.48484848484848484848
```

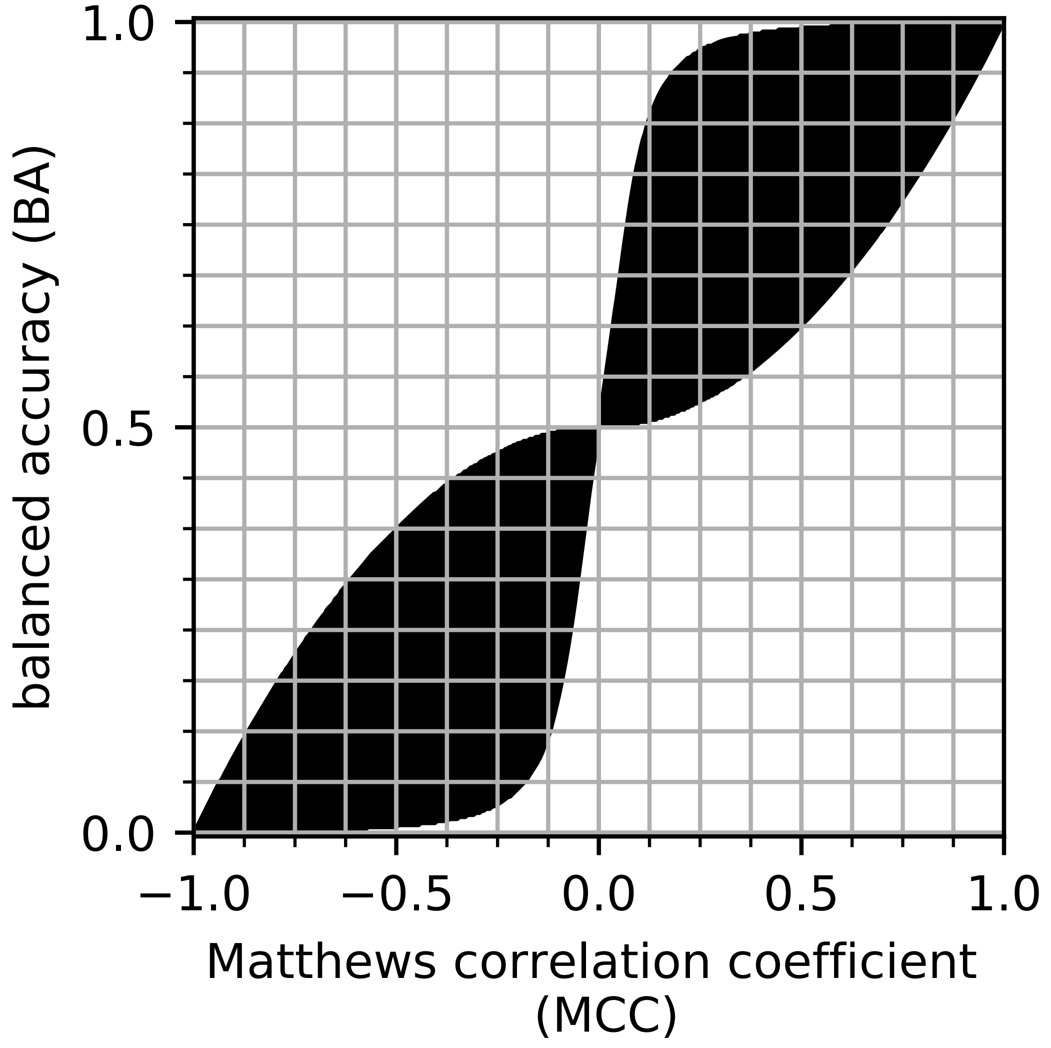
```
from sklearn.metrics import balanced_accuracy_score

balanced_accuracy_score(true, predicted)
```

```
0.48484848484848484848
```

```
from sklearn.metrics import recall_score

recall_score(true, predicted, average='macro')
```

```
0.48484848484848484848
```

An interesting tidbit is that a BA > 0.5 corresponds to a positive MCC, and a BA < 0.5 corresponds to a negative MCC.



(https://biodatamining.biomedcentral.com/articles/10.1186/s13040-021-00244-z

The following is based on the follow-up paper that the authors sent me: "The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation" (https://biodatamining.biomedcentral.com/articles/10.1186/s13040-021-00244-z)

The bottom line is that MCC is only high if all 4 rates are high: True positive R, True negative R, Positive predicted value (precision): TP / (TP+FP), and Negative predicted value: TN / (TN+FN).

However, BA gives us a better idea about whether a classifier is much better than random guessing.

1. Confusion Matrix

2. Precision, Recall, and F1 Score

3. Matthews Correlation Coefficient

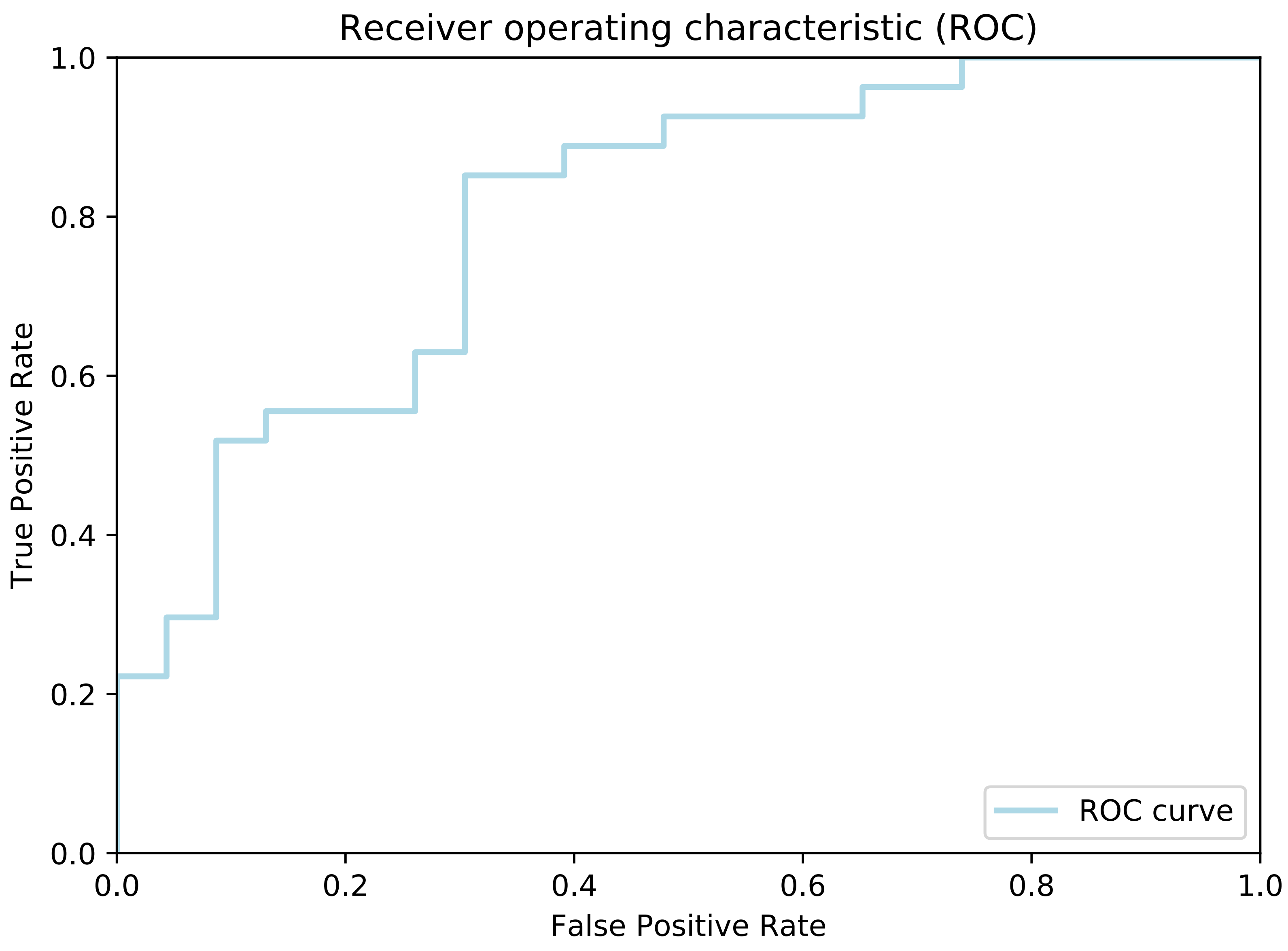
4. Balanced Accuracy

5. ROC

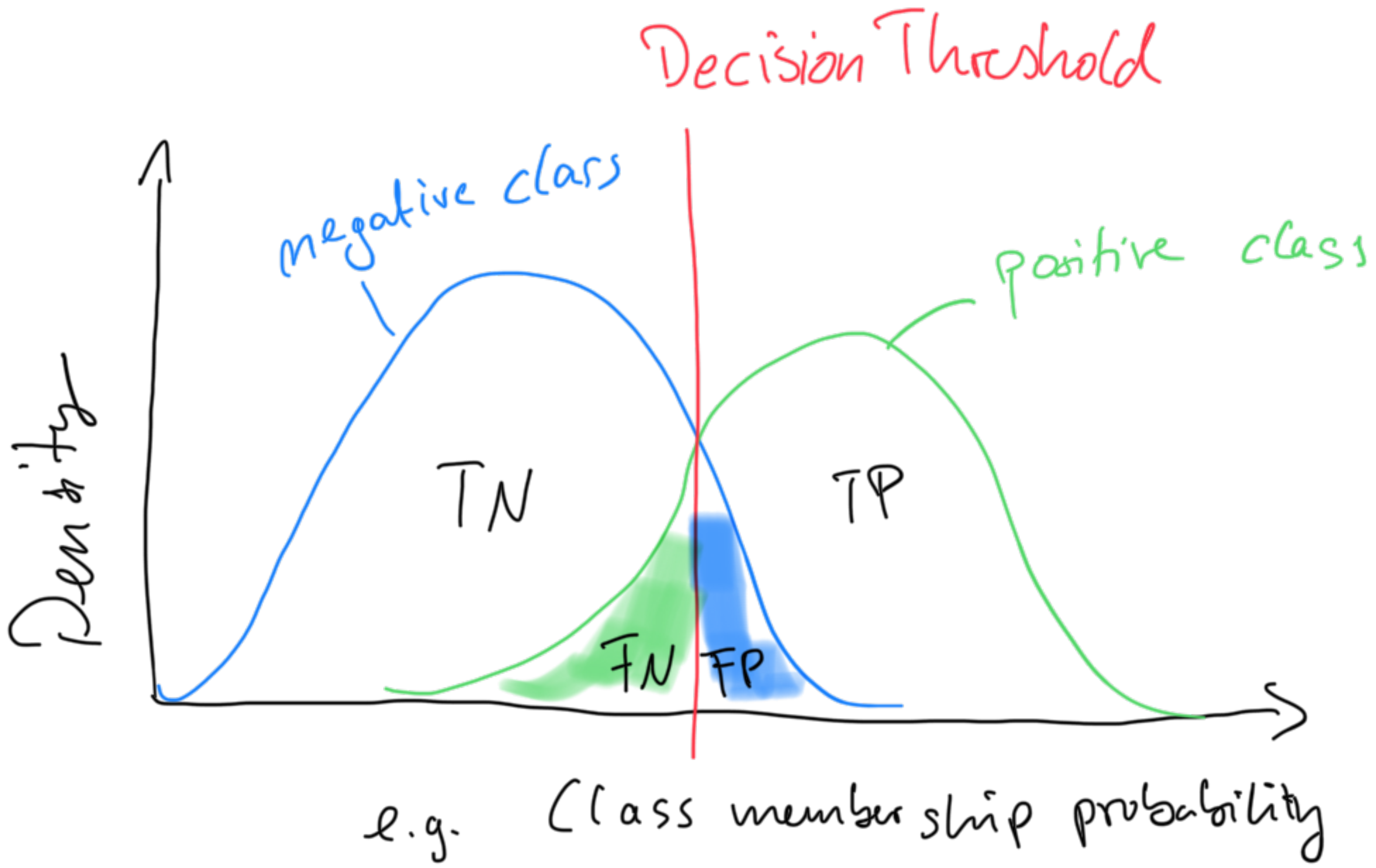
6. Extending Binary Metrics to Multi-class Settings

Receiver Operating Characteristic curve (ROC curve)

- Trade-off between True Positive Rate and False Positive Rate
- ROC can be plotted by changing the prediction threshold
- ROC term comes from "Radar Receiver Operators"
  (analysis of radar [**RA**dio **D**irection **A**nd **R**anging] images)

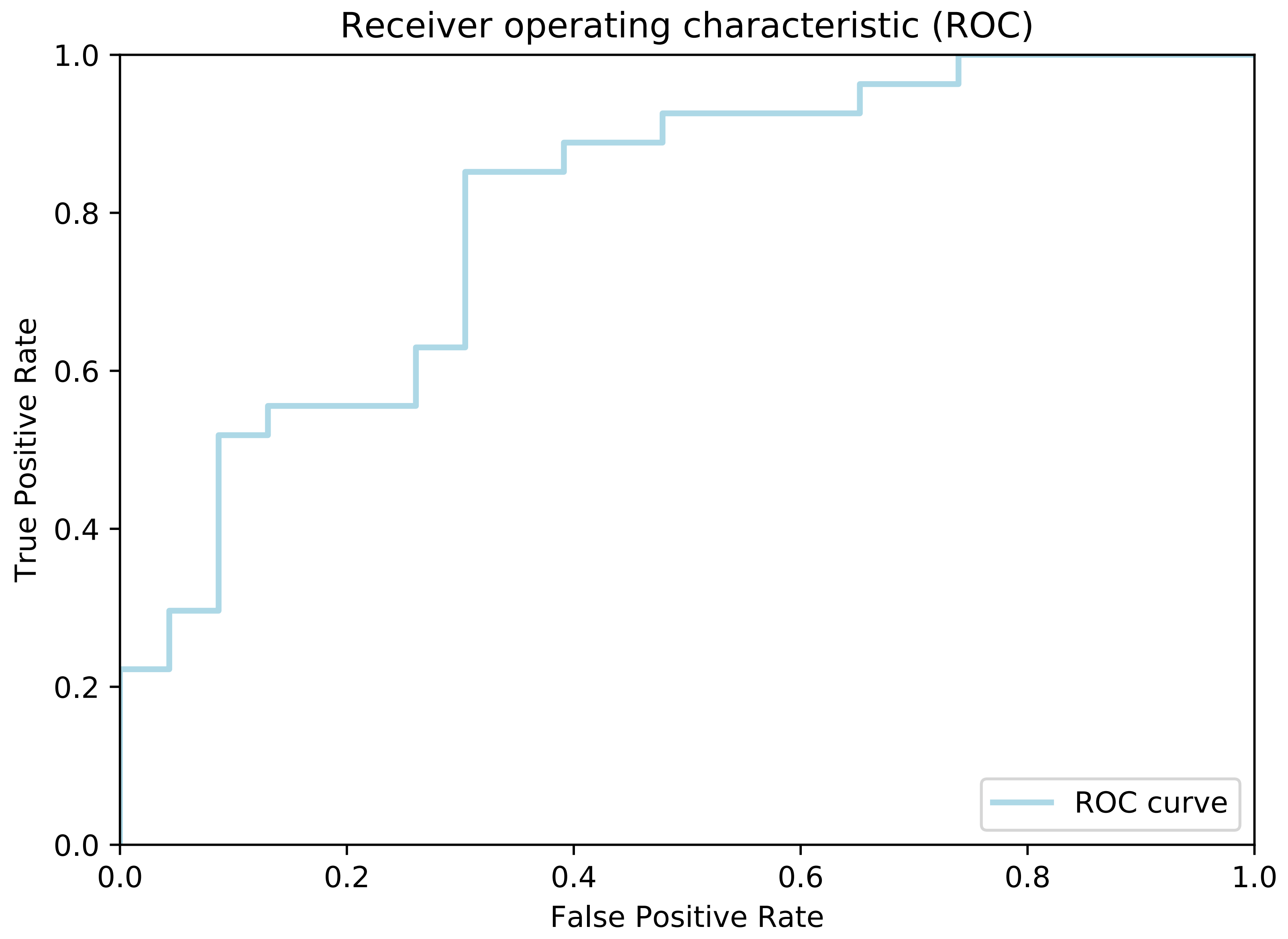


RECAP: False Positive Rate and False Negative Rate

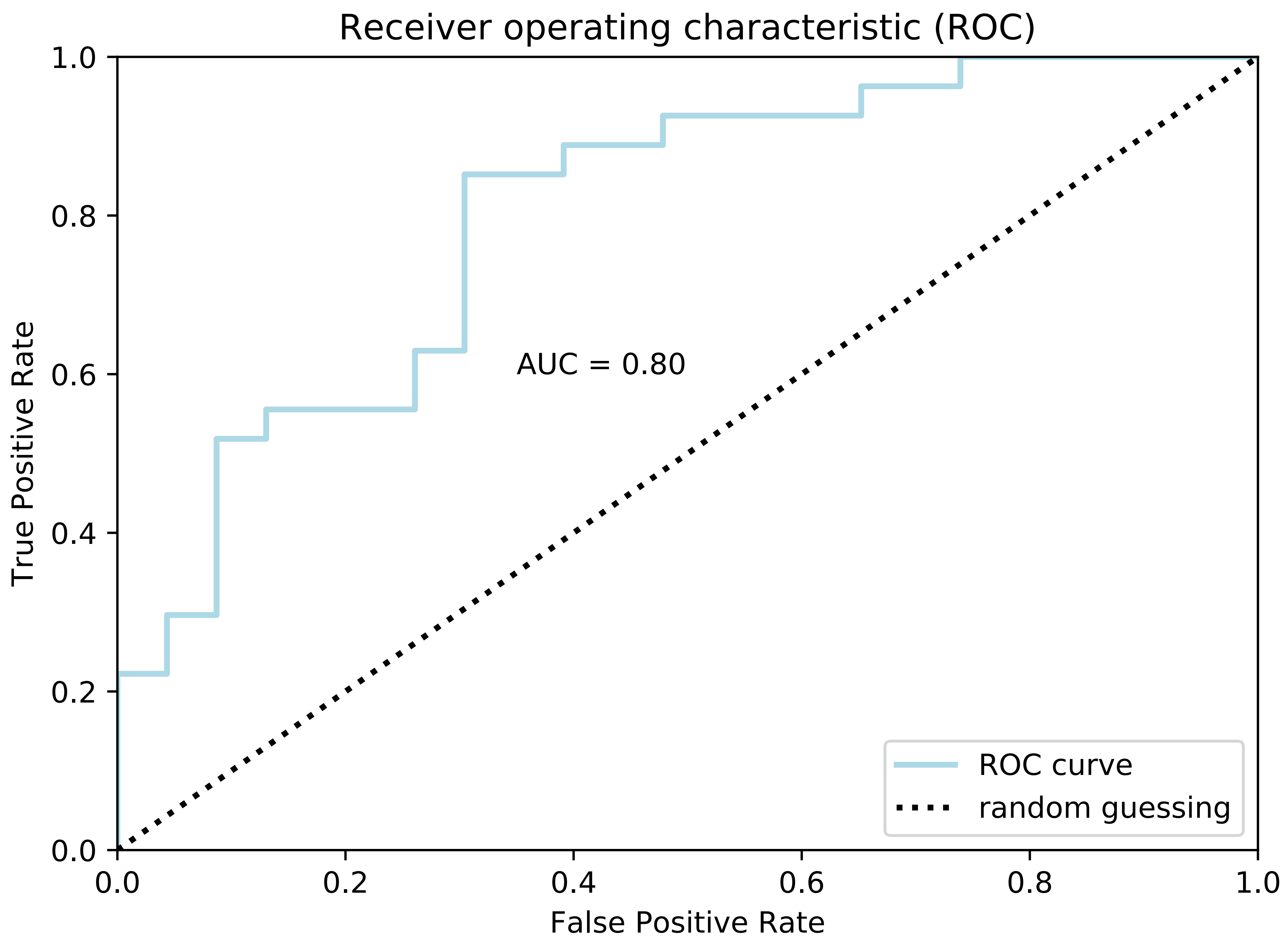


Receiver Operating Characteristic curve (ROC curve)

- ?.? = Perfect Prediction
- ?.? = Random Prediction

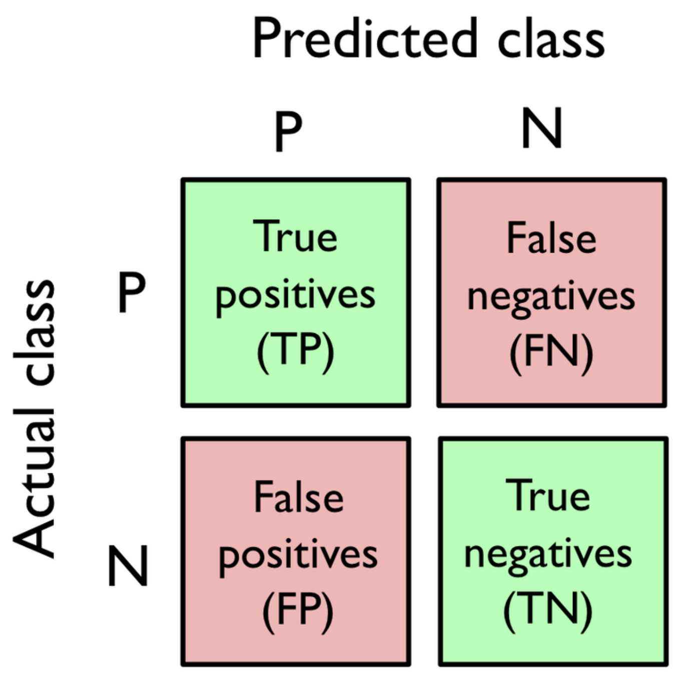


ROC Area Under the Curve (AUC)



$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{FNR}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR}$$



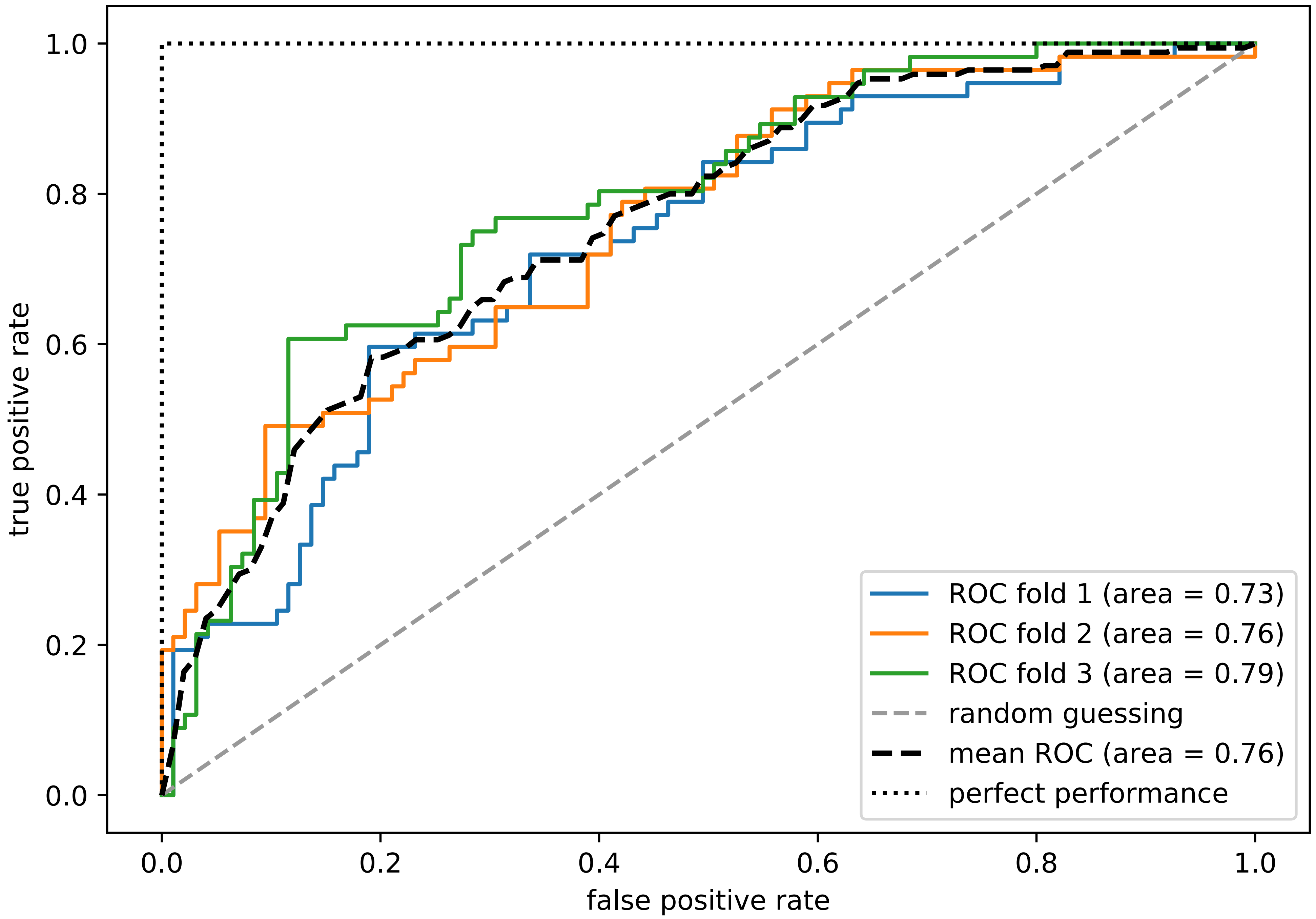
Balanced case: $\begin{array}{cc} 100 & 100 \\ 100 & 100 \end{array}$

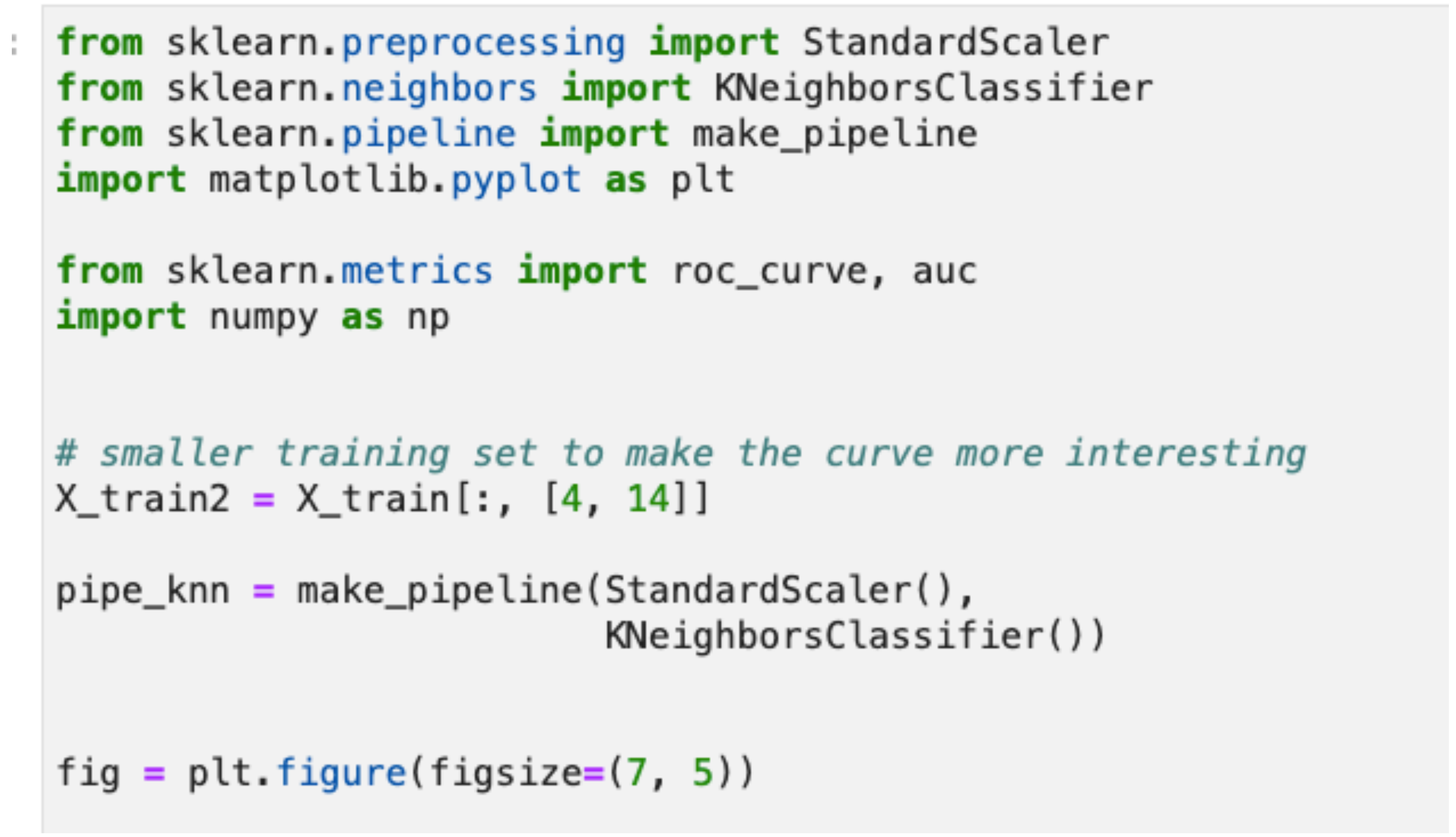
TPR = 100/200 = 0.5
FPR = 100/200 = 0.5

Imbalanced case: $\begin{array}{cc} 200 & 200 \\ 50 & 50 \end{array}$

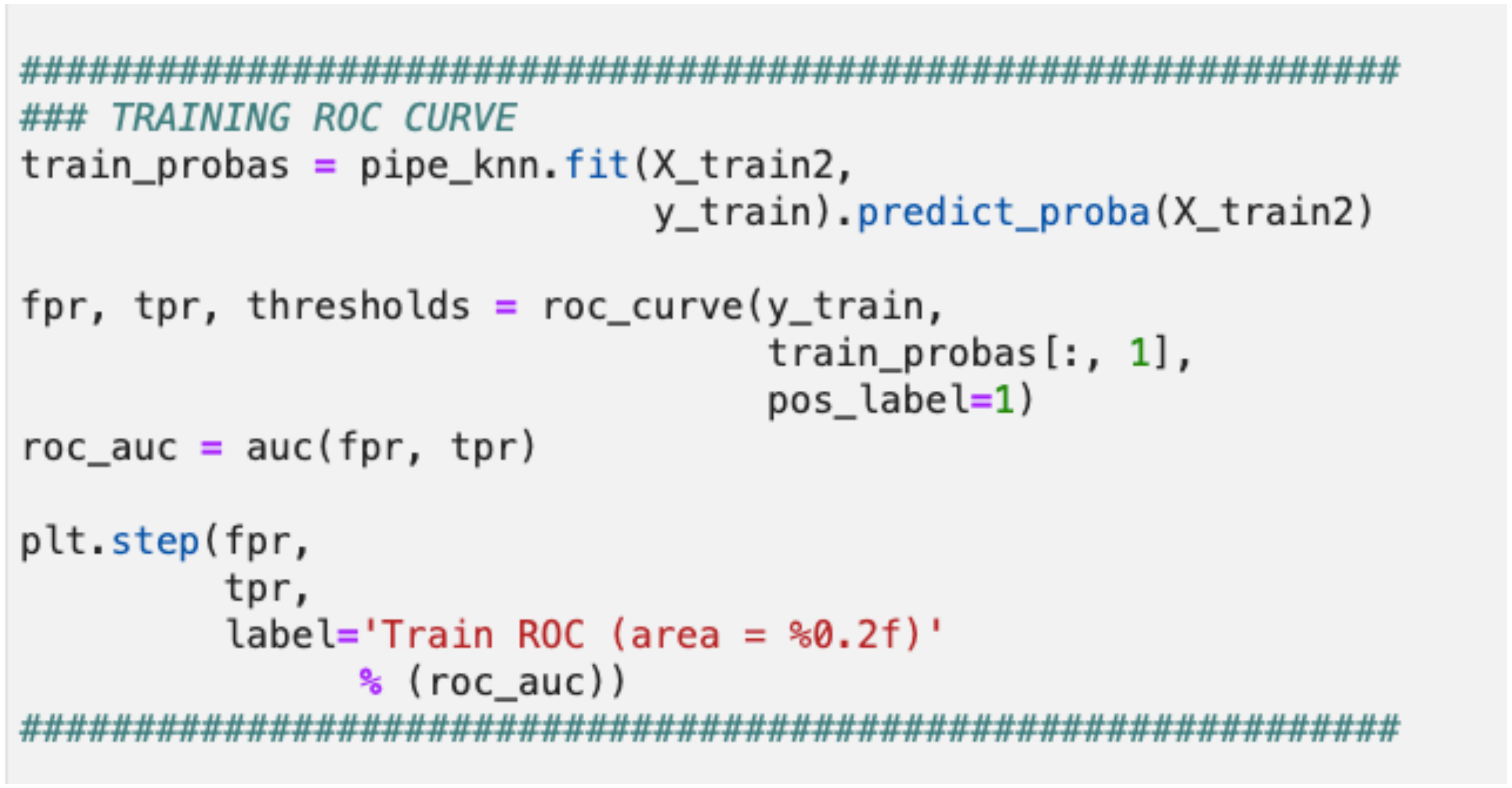
TPR = 200/400 = 0.5
FPR = 50/100 = 0.5

ROC and k-Fold Cross-Validation



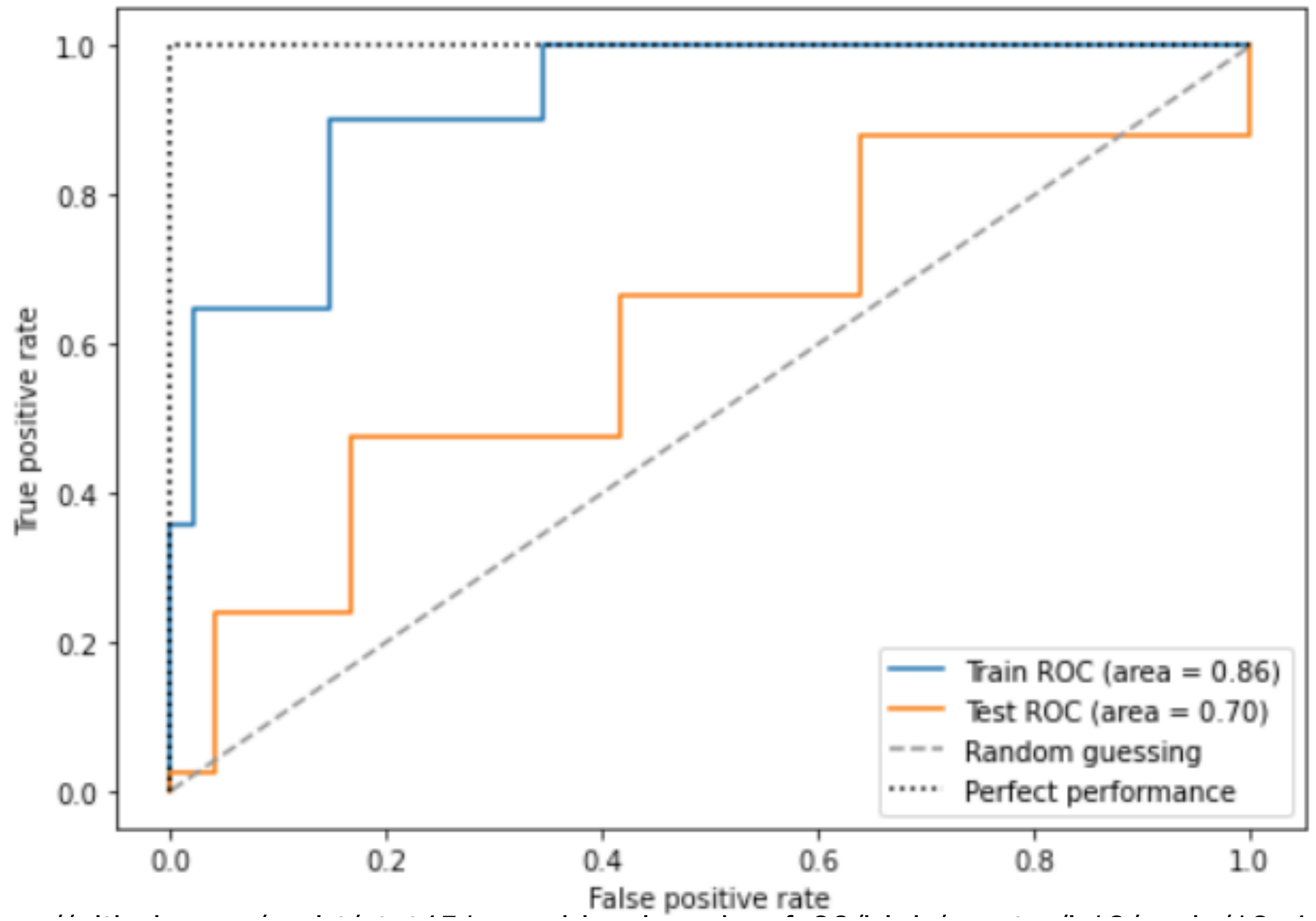


```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt

from sklearn.metrics import roc_curve, auc
import numpy as np


# smaller training set to make the curve more interesting
X_train2 = X_train[:, [4, 14]]

pipe_knn = make_pipeline(StandardScaler(),
                         KNeighborsClassifier())


fig = plt.figure(figsize=(7, 5))
```

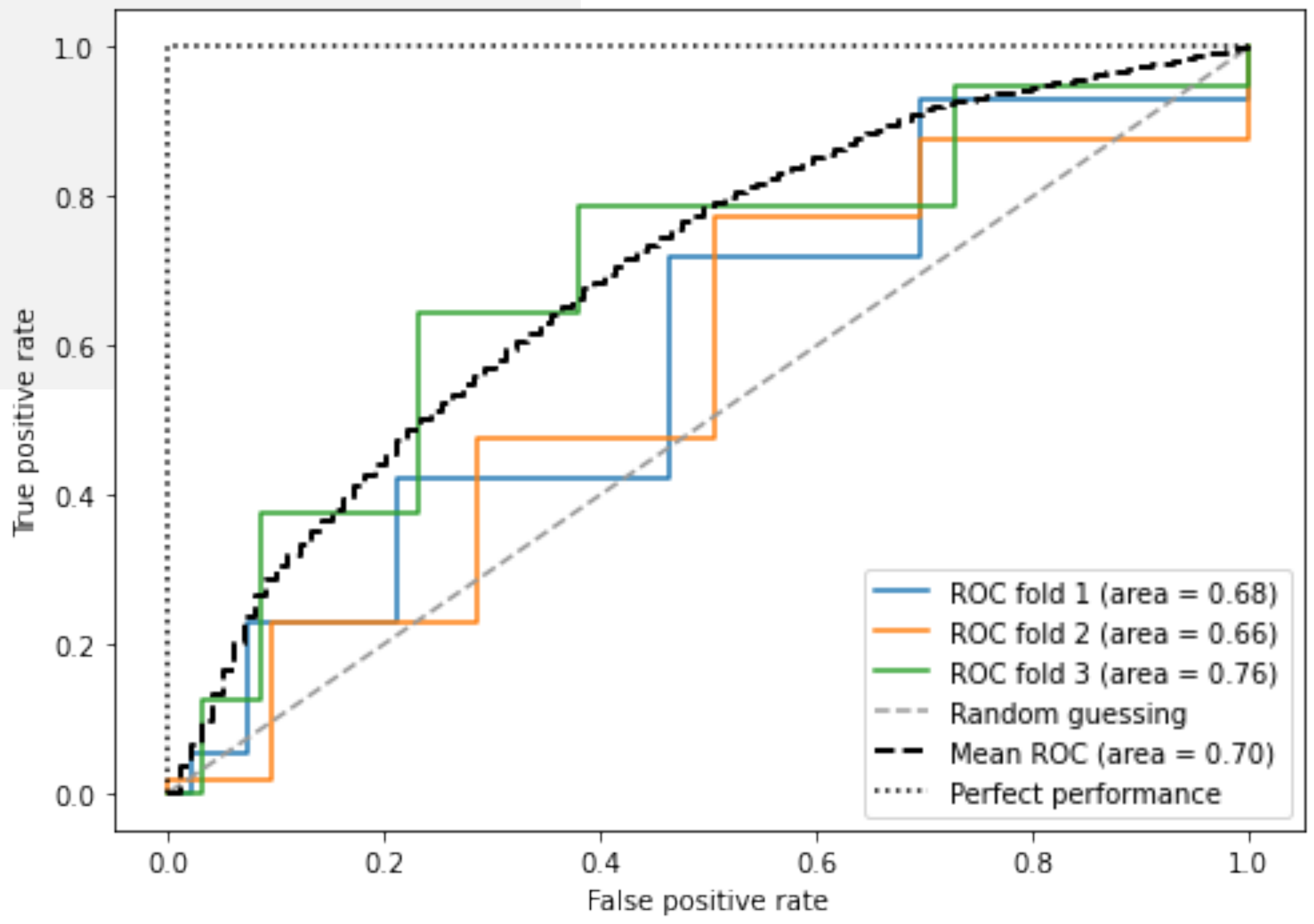


```
#############################################################
### TRAINING ROC CURVE
train_probas = pipe_knn.fit(X_train2,
                            y_train).predict_proba(X_train2)

fpr, tpr, thresholds = roc_curve(y_train,
                                 train_probas[:, 1],
                                 pos_label=1)
roc_auc = auc(fpr, tpr)

plt.step(fpr,
         tpr,
         label='Train ROC (area = %0.2f)'
               % (roc_auc))
#############################################################
```

```
##################################################################
### TEST ROC CURVE
test_probas = pipe_knn.predict_proba(X_test[:, [4, 14]])

fpr, tpr, thresholds = roc_curve(y_test,
                                 test_probas[:, 1],
                                 pos_label=1)
roc_auc = auc(fpr, tpr)

plt.step(fpr,
         tpr,
         where='post',
         label='Test ROC (area = %0.2f)'
              % (roc_auc))
##################################################################
```



```
cv = list(StratifiedKFold(n_splits=3,
                          shuffle=True,
                          random_state=1).split(X_train, y_train))

fig = plt.figure(figsize=(7, 5))

mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)
all_tpr = []

for i, (train, test) in enumerate(cv):
    probas = pipe_knn.fit(X_train2[train],
                          y_train[train]).predict_proba(X_train2[test])

    fpr, tpr, thresholds = roc_curve(y_train[test],
                                     probas[:, 1],
                                     pos_label=1)
    mean_tpr += np.interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = auc(fpr, tpr)
    plt.step(fpr,
             tpr,
             label='ROC fold %d (area = %0.2f)'
                   % (i+1, roc_auc), where='post')
```



1. Confusion Matrix

2. Precision, Recall, and F1 Score

3. Matthews Correlation Coefficient

4. Balanced Accuracy

5. ROC

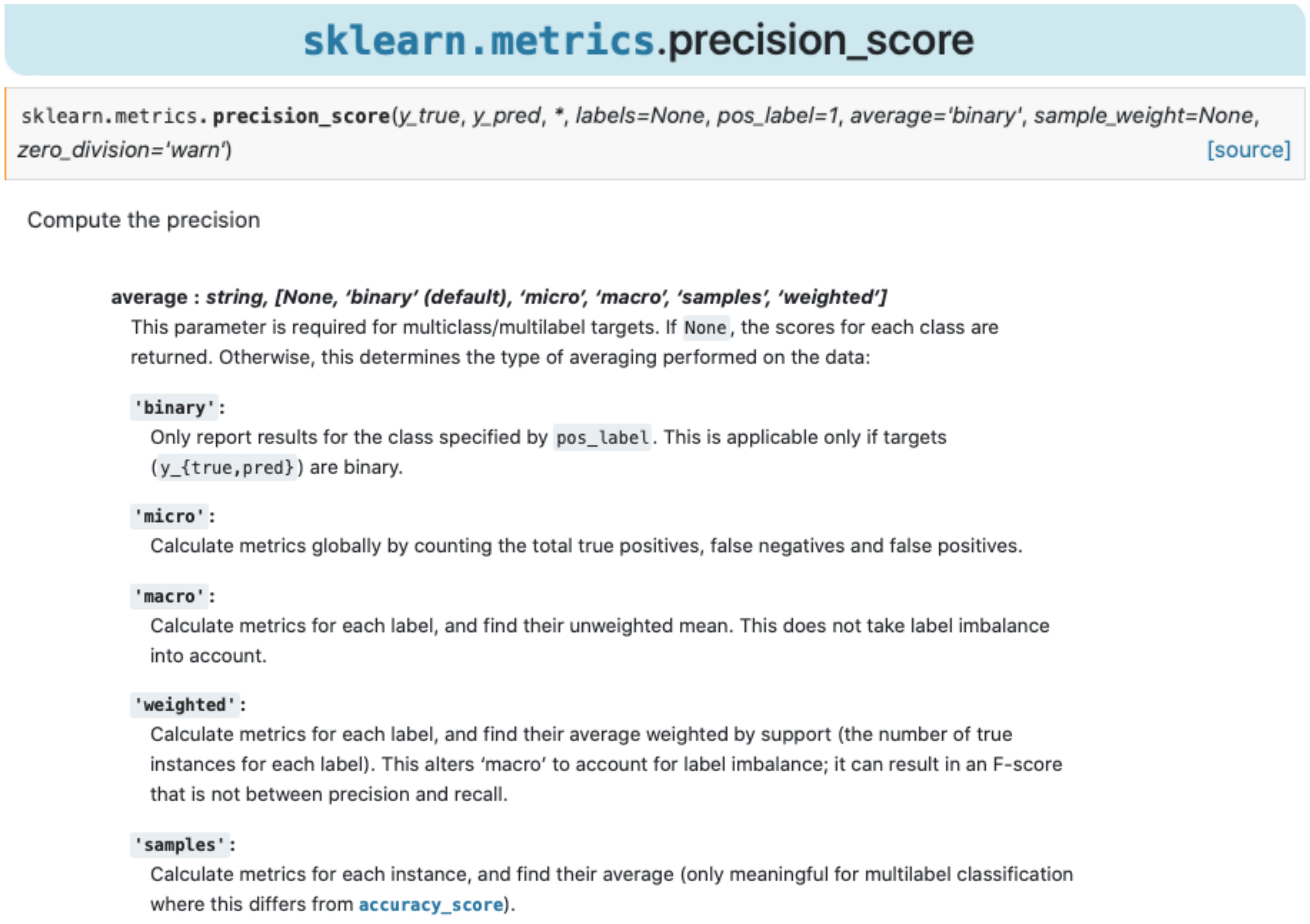
6. Extending Binary Metrics to Multi-class Settings

Macro and Micro Averaging

$$PRE_{micro} = \frac{TP_1 + \ldots + TP_c}{TP_1 + \ldots + TP_c + FP_1 + \ldots + FP_c}$$

$$PRE_{macro} = \frac{PRE_1 + \ldots + PRE_c}{c}$$

Micro-averaging is useful if we want to weight each instance or prediction equally, whereas macro-averaging weights all classes equally to evaluate the overall performance of a classifier with regard to the most frequent class labels.



sklearn.metrics.precision_score

`sklearn.metrics.precision_score`(*y_true, y_pred, *, labels=None, pos_label=1, average='binary', sample_weight=None, zero_division='warn'*) [source]

Compute the precision

average : *string, [None, 'binary' (default), 'micro', 'macro', 'samples', 'weighted']*

This parameter is required for multiclass/multilabel targets. If `None`, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data:

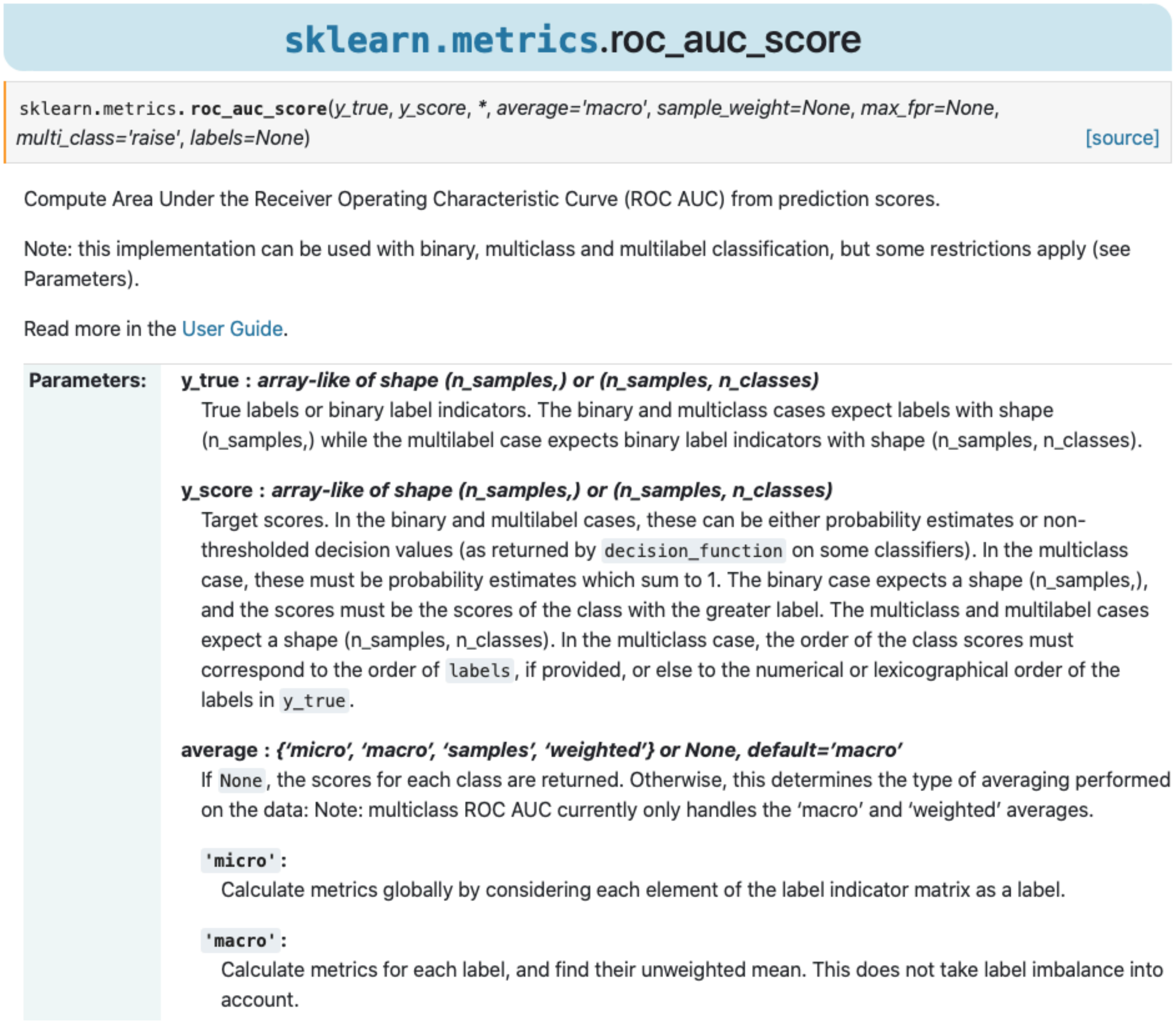
`'binary'`:
Only report results for the class specified by `pos_label`. This is applicable only if targets (`y_{true,pred}`) are binary.

`'micro'`:
Calculate metrics globally by counting the total true positives, false negatives and false positives.

`'macro'`:
Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

`'weighted'`:
Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall.

`'samples'`:
Calculate metrics for each instance, and find their average (only meaningful for multilabel classification where this differs from `accuracy_score`).



sklearn.metrics.roc_auc_score

sklearn.metrics.**roc_auc_score**(*y_true*, *y_score*, *, *average='macro'*, *sample_weight=None*, *max_fpr=None*, *multi_class='raise'*, *labels=None*) [source]

Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

Note: this implementation can be used with binary, multiclass and multilabel classification, but some restrictions apply (see Parameters).

Read more in the User Guide.

Parameters:

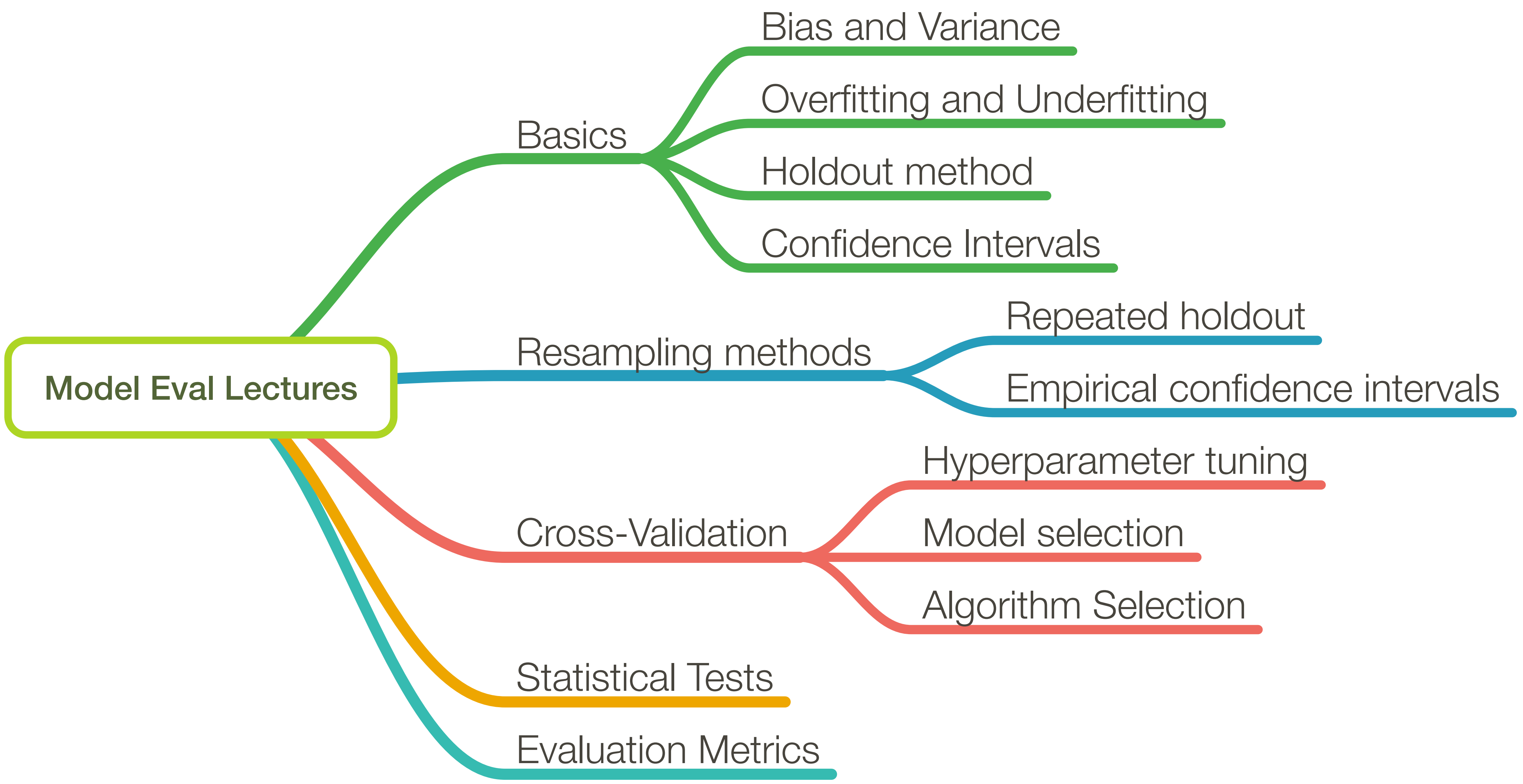
y_true : *array-like of shape (n_samples,) or (n_samples, n_classes)*
True labels or binary label indicators. The binary and multiclass cases expect labels with shape (n_samples,) while the multilabel case expects binary label indicators with shape (n_samples, n_classes).

y_score : *array-like of shape (n_samples,) or (n_samples, n_classes)*
Target scores. In the binary and multilabel cases, these can be either probability estimates or non-thresholded decision values (as returned by `decision_function` on some classifiers). In the multiclass case, these must be probability estimates which sum to 1. The binary case expects a shape (n_samples,), and the scores must be the scores of the class with the greater label. The multiclass and multilabel cases expect a shape (n_samples, n_classes). In the multiclass case, the order of the class scores must correspond to the order of `labels`, if provided, or else to the numerical or lexicographical order of the labels in `y_true`.

average : *{'micro', 'macro', 'samples', 'weighted'} or None, default='macro'*
If `None`, the scores for each class are returned. Otherwise, this determines the type of averaging performed on the data: Note: multiclass ROC AUC currently only handles the 'macro' and 'weighted' averages.

`'micro'`:
Calculate metrics globally by considering each element of the label indicator matrix as a label.

`'macro'`:
Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.


Model Eval Lectures
Basics
Bias and Variance
Overfitting and Underfitting
Holdout method
Confidence Intervals
Resampling methods
Repeated holdout
Empirical confidence intervals
Cross-Validation
Hyperparameter tuning
Model selection
Algorithm Selection
Statistical Tests
Evaluation Metrics