Lecture 12

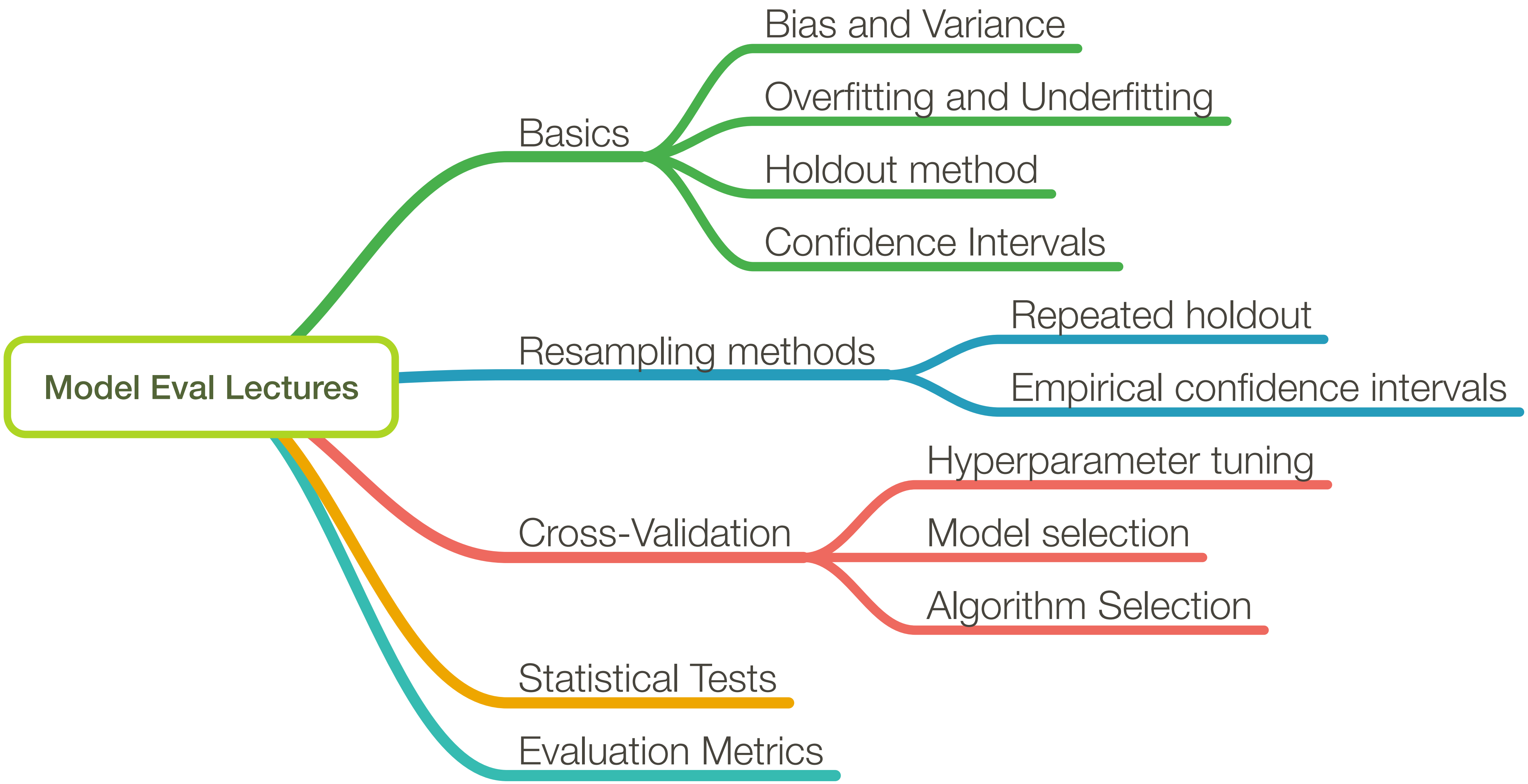
Model Evaluation 5:

Performance Metrics

STAT 451: Machine Learning, Fall 2020

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat451-fs2020/



- **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. Balanced Accuracy
4. ROC
5. Extending Binary Metrics to Multi-class Settings

1. **Confusion Matrix**

2. Precision, Recall, and F1 Score

3. Balanced Accuracy

4. ROC

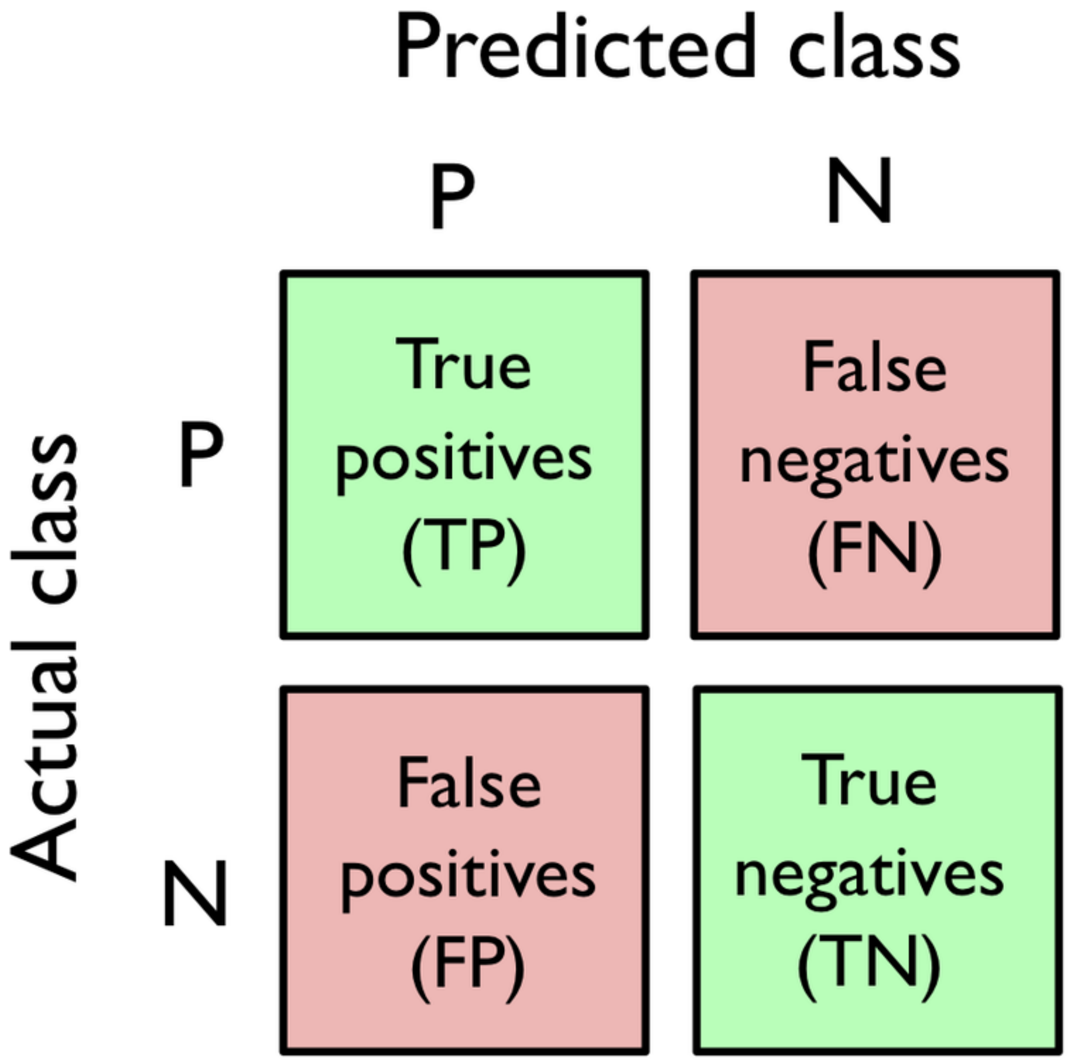
5. Extending Binary Metrics to Multi-class Settings

Based on

Raschka & Mirjalili 2019: *Python Machine Learning, 3rd Edition*
Chapter 6: Learning Best Practices for Model Evaluation and Hyperparameter Tuning

(no lecture notes)

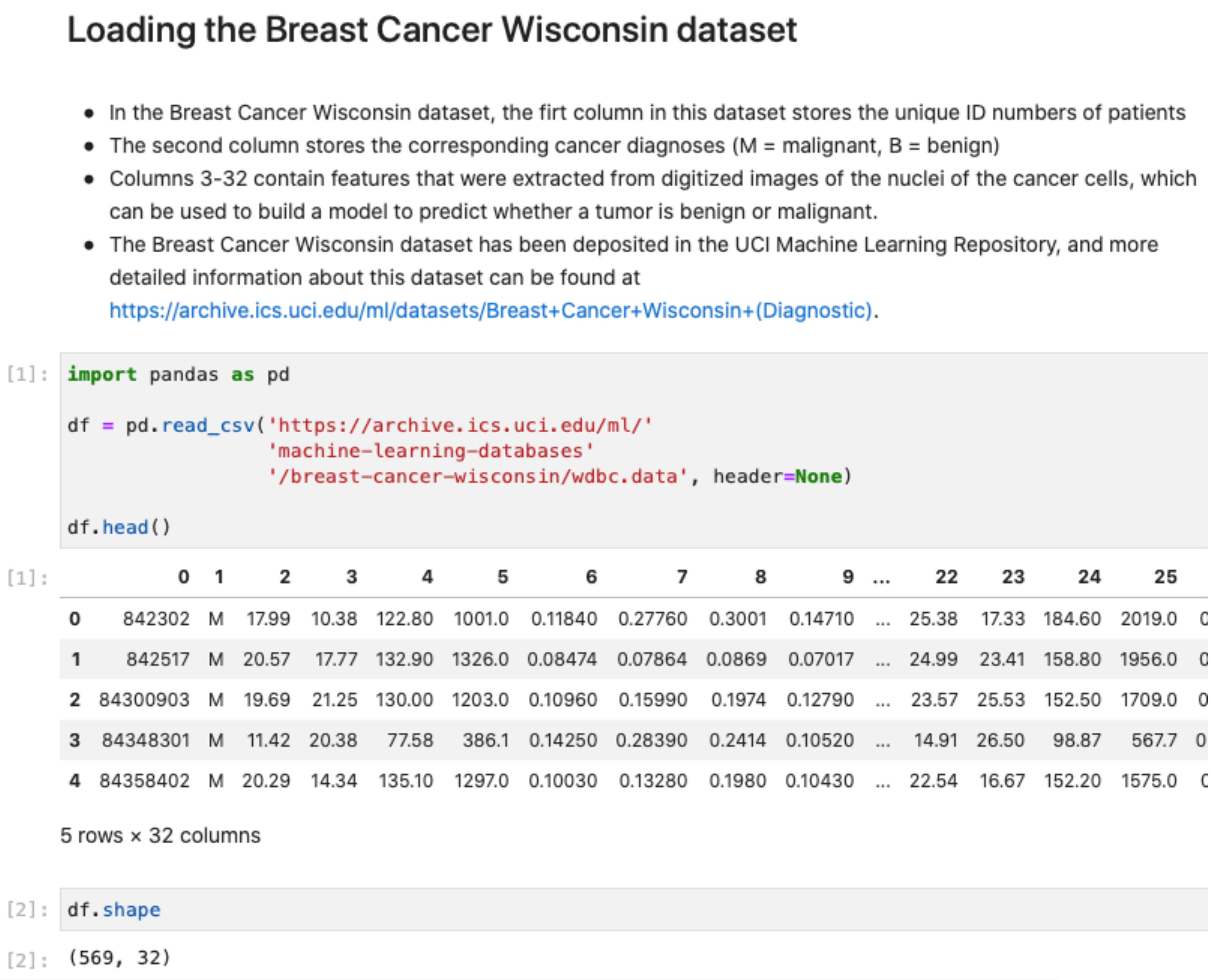
2x2 Confusion Matrix



		Predicted class	
		P	N
Actual class	P	True positives (TP)	False negatives (FN)
	N	False positives (FP)	True negatives (TN)

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

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



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()
```

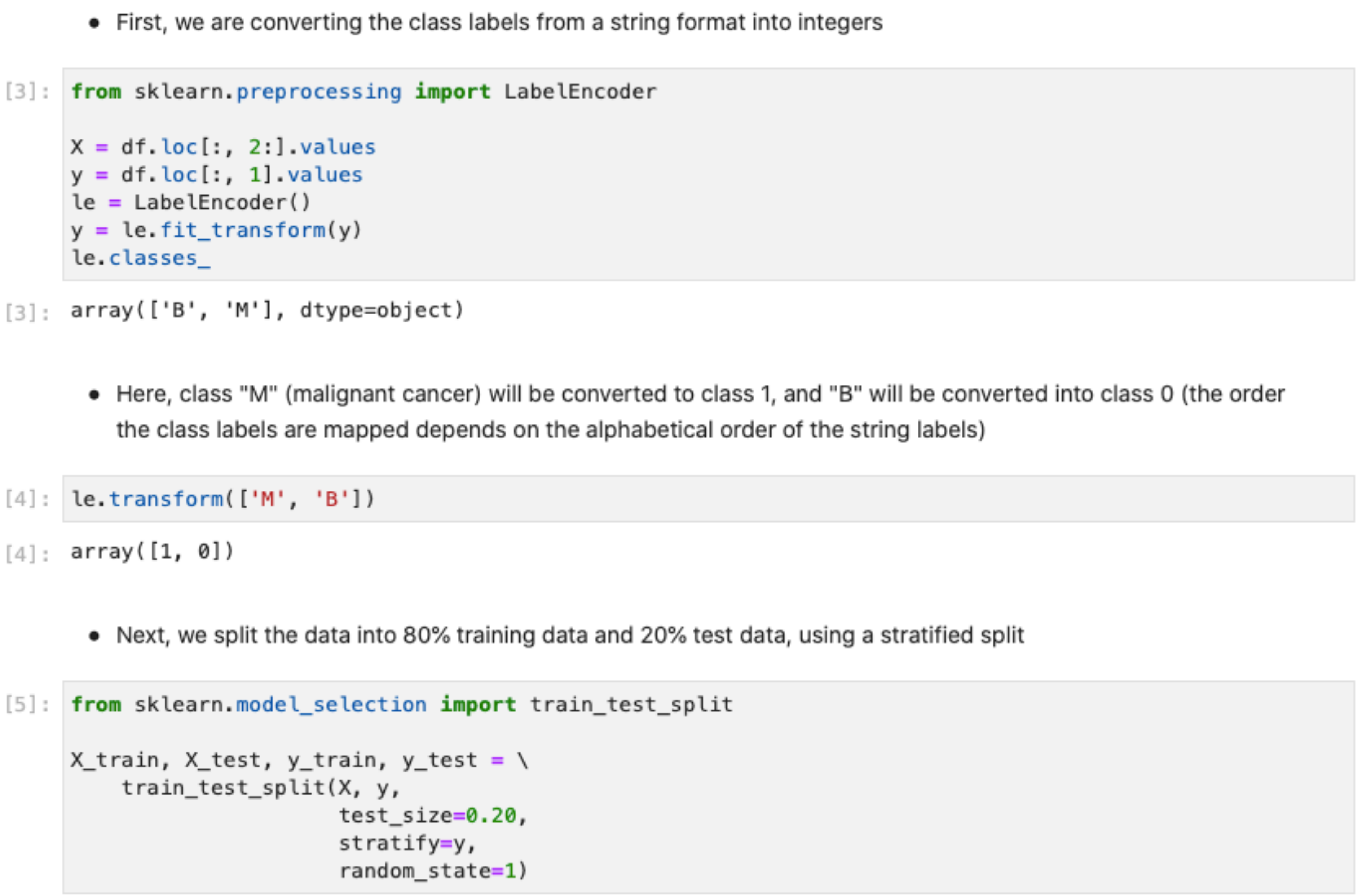
[1]:

	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	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	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	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	0
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	0

5 rows × 32 columns

```
[2]: df.shape

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

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_1_confusion-matrix.ipynb



- 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)

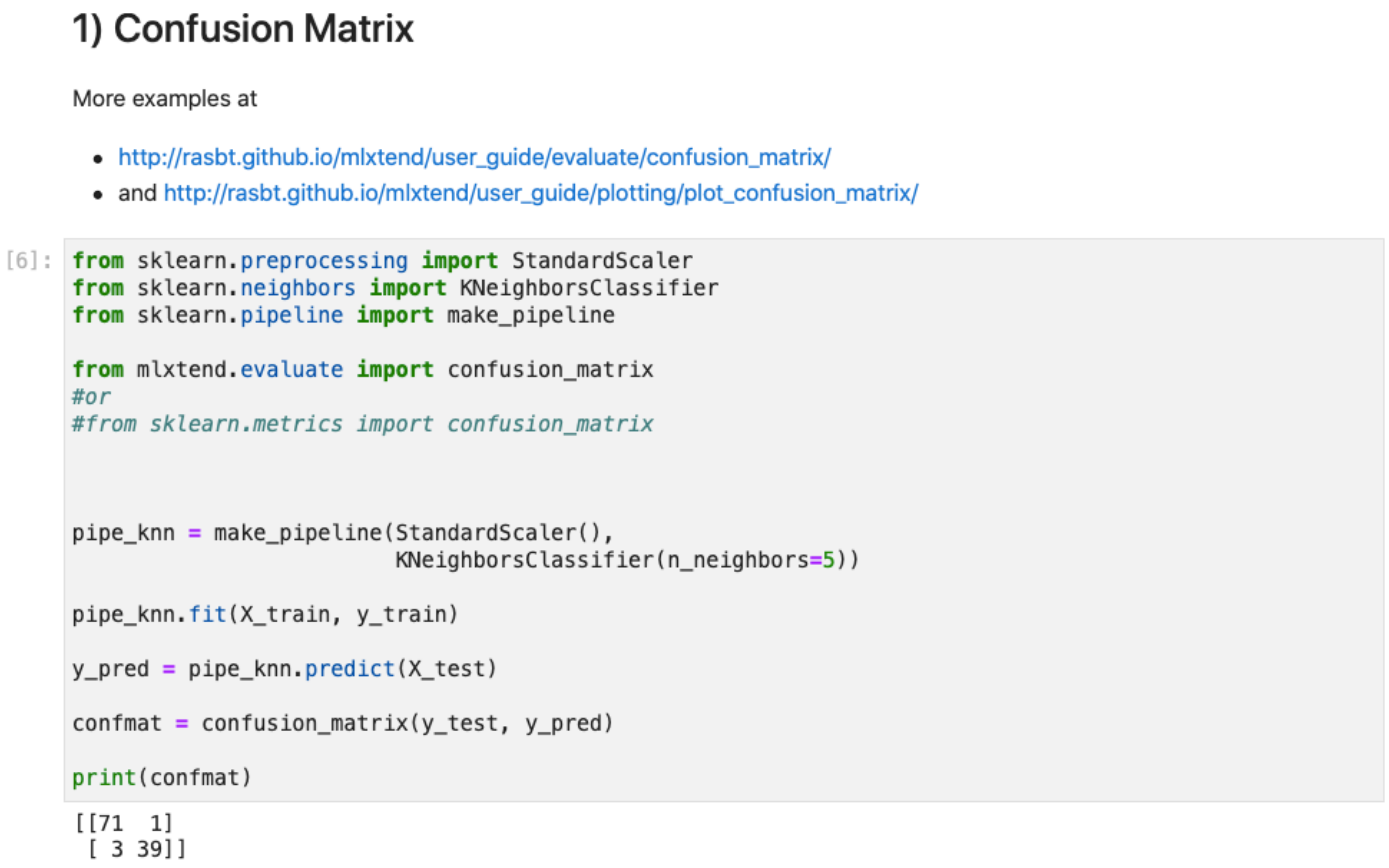
```
[4]: le.transform(['M', 'B'])
```

```
[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)
```

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_1_confusion-matrix.ipynb



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/

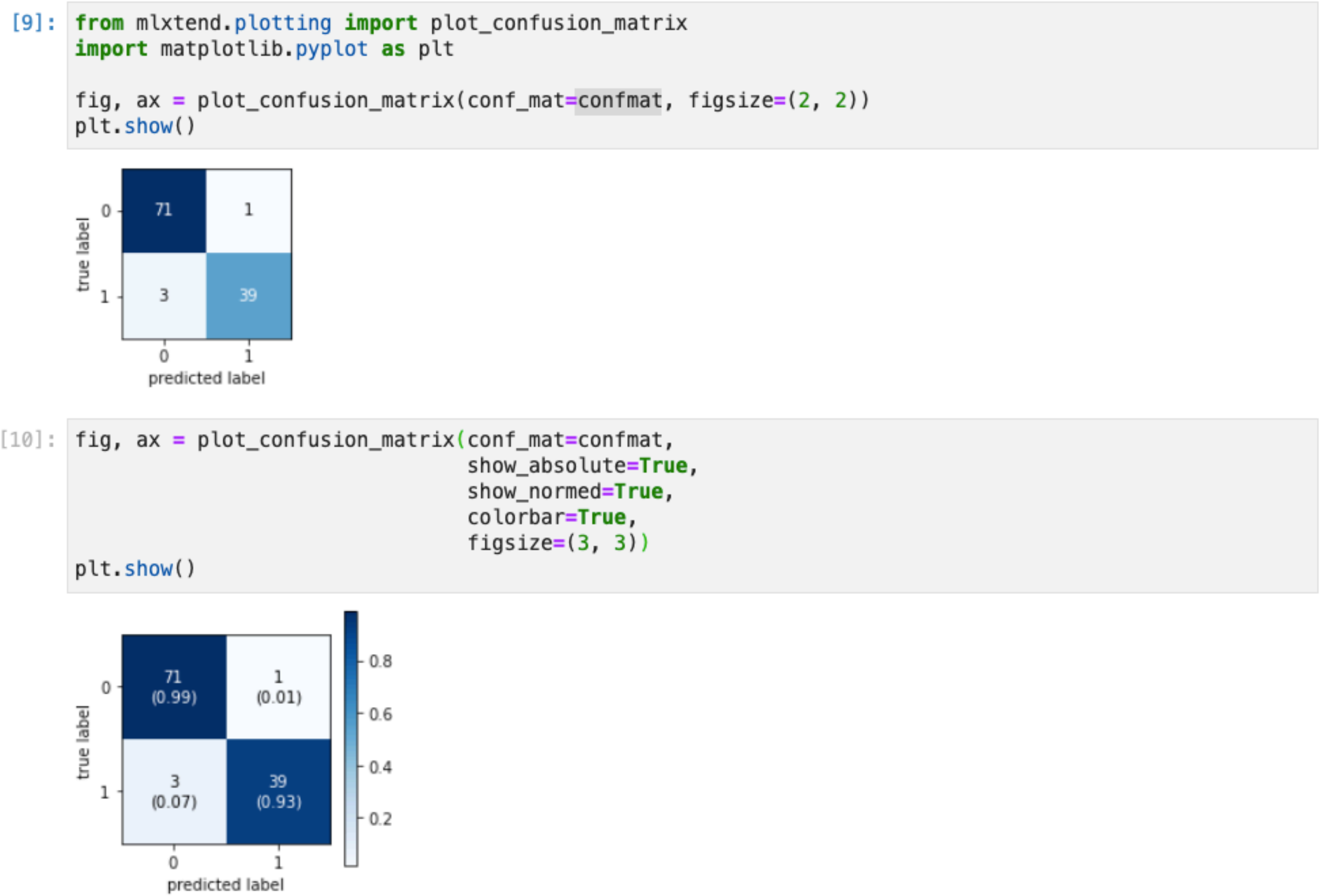
[6]:

```
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]]
```

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_1_confusion-matrix.ipynb

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()
```

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_1_confusion-matrix.ipynb

False Positive Rate and False Negative Rate

* Relevant later for ROC

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

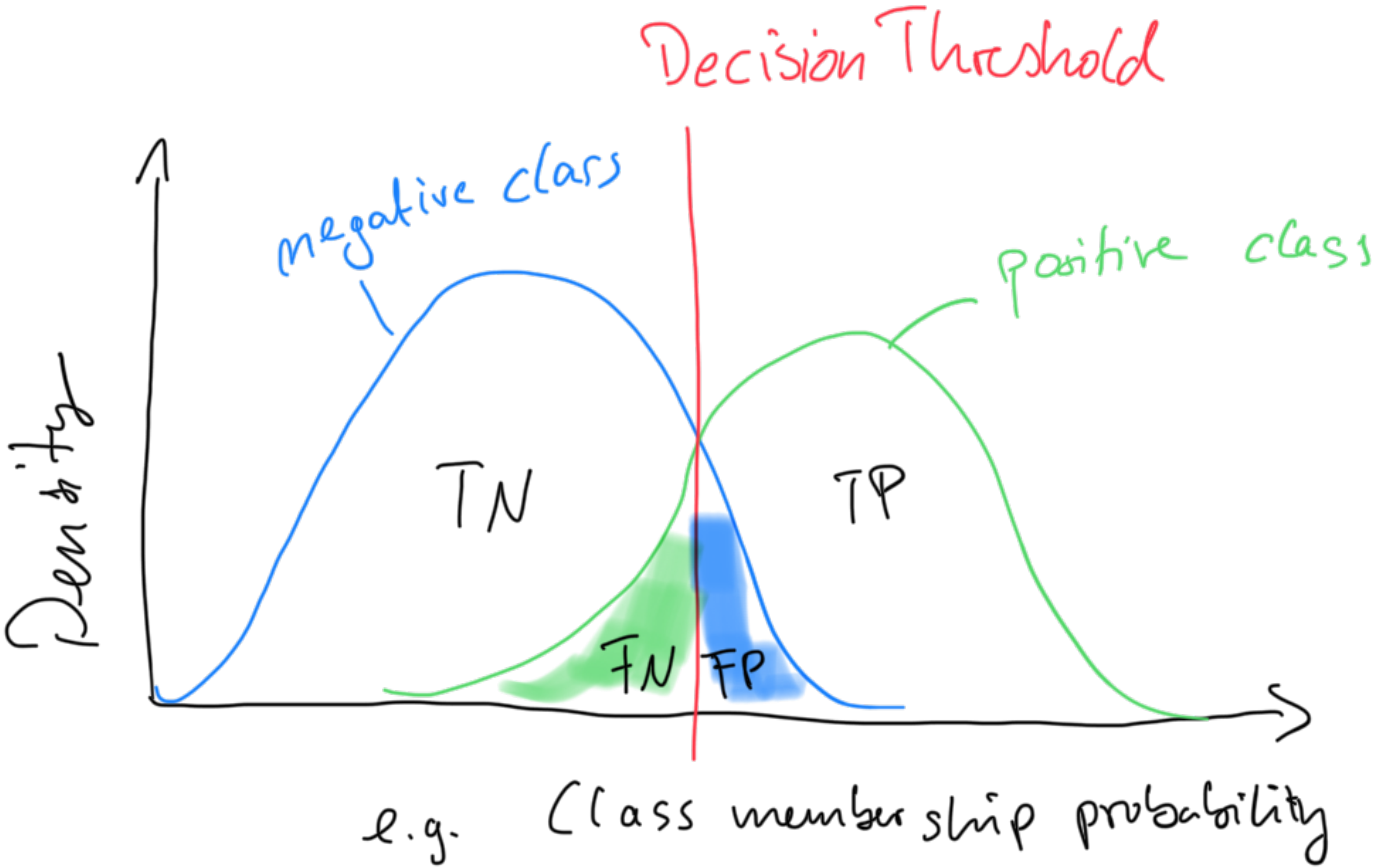
$$\mathrm{FPR}^{*} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

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

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

- Think of it in a spam classification problem (what are true positives, and if you had to pick one at the expense of the other: would you rather decrease the FPR or increase the TPR?)

False Positive Rate and False Negative Rate



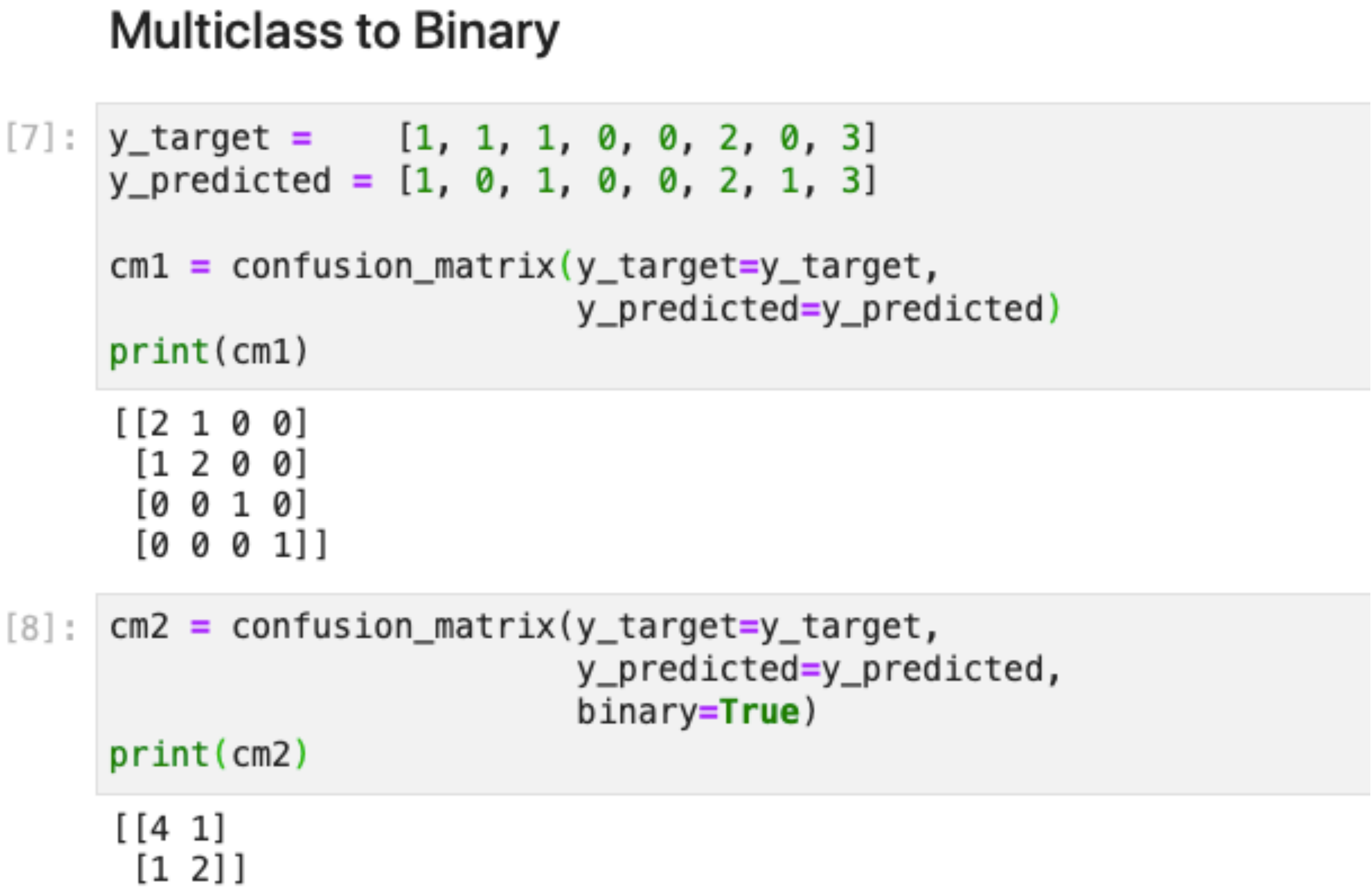
Confusion Matrix for Multi-Class Settings

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

```
[7]: 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]]
```

```
[8]: cm2 = confusion_matrix(y_target=y_target,
                            y_predicted=y_predicted,
                            binary=True)
     print(cm2)
```

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

Code: https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_1_confusion-matrix.ipynb

1. Confusion Matrix

2. **Precision, Recall, and F1 Score**

3. Balanced Accuracy

4. ROC

5. Extending Binary Metrics to Multi-class Settings

Precision, Recall, and F1 Score

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

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

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

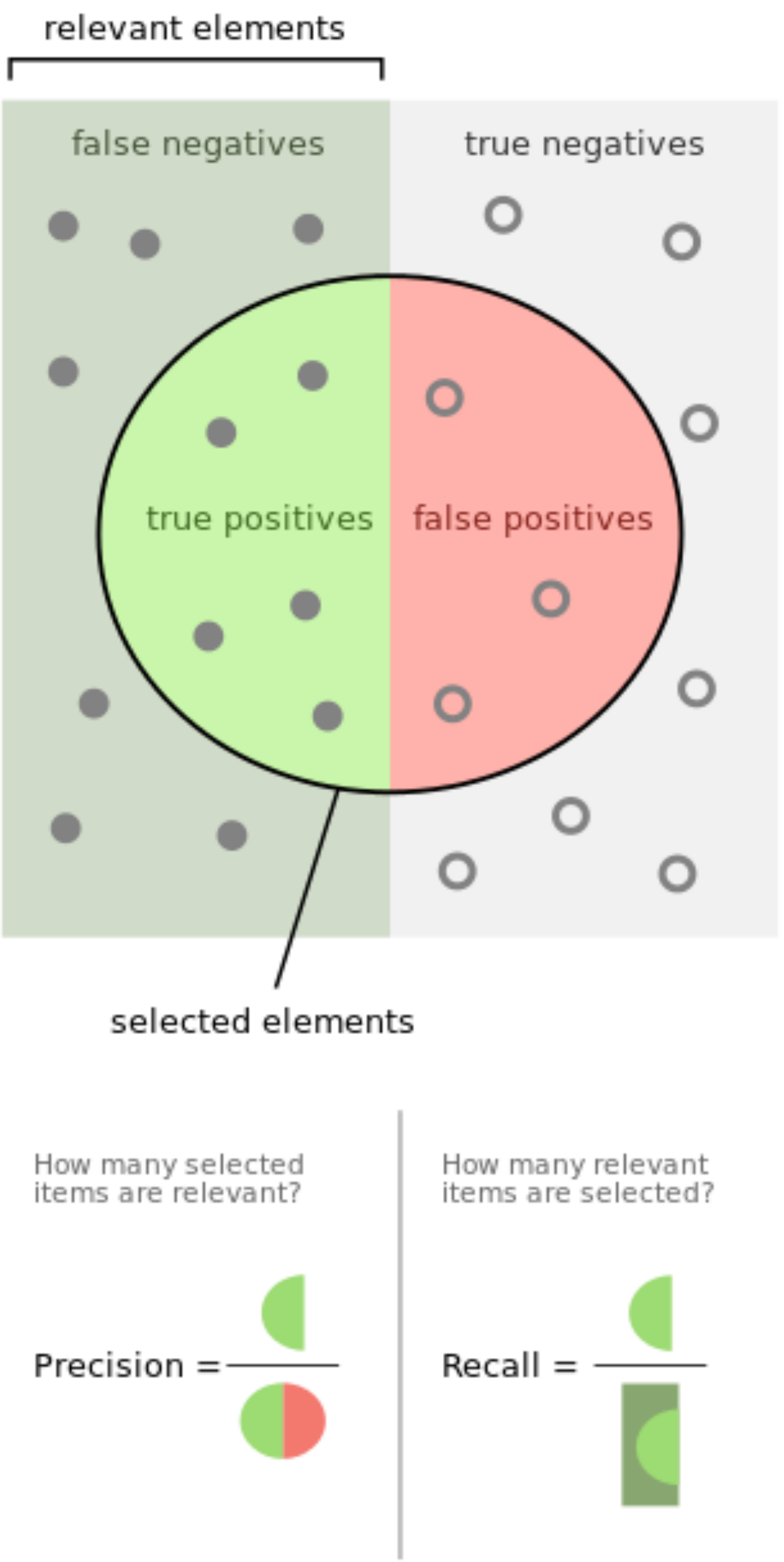
- Terms that are more popular in Information Technology
- Recall is actually just another term for True Positive Rate (or "sensitivity")

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.

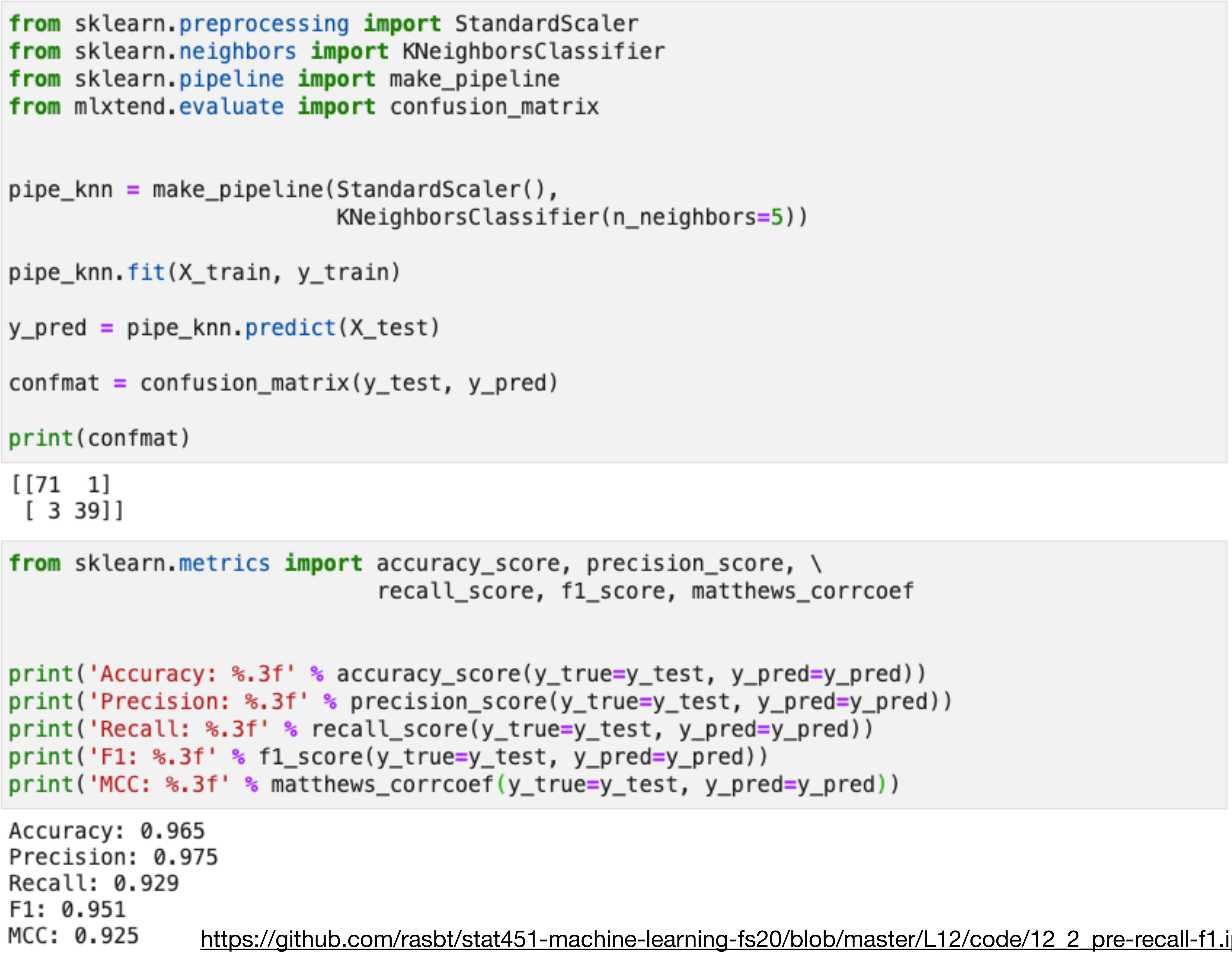
Matthew's Correlation Coefficient

- 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.

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)
```

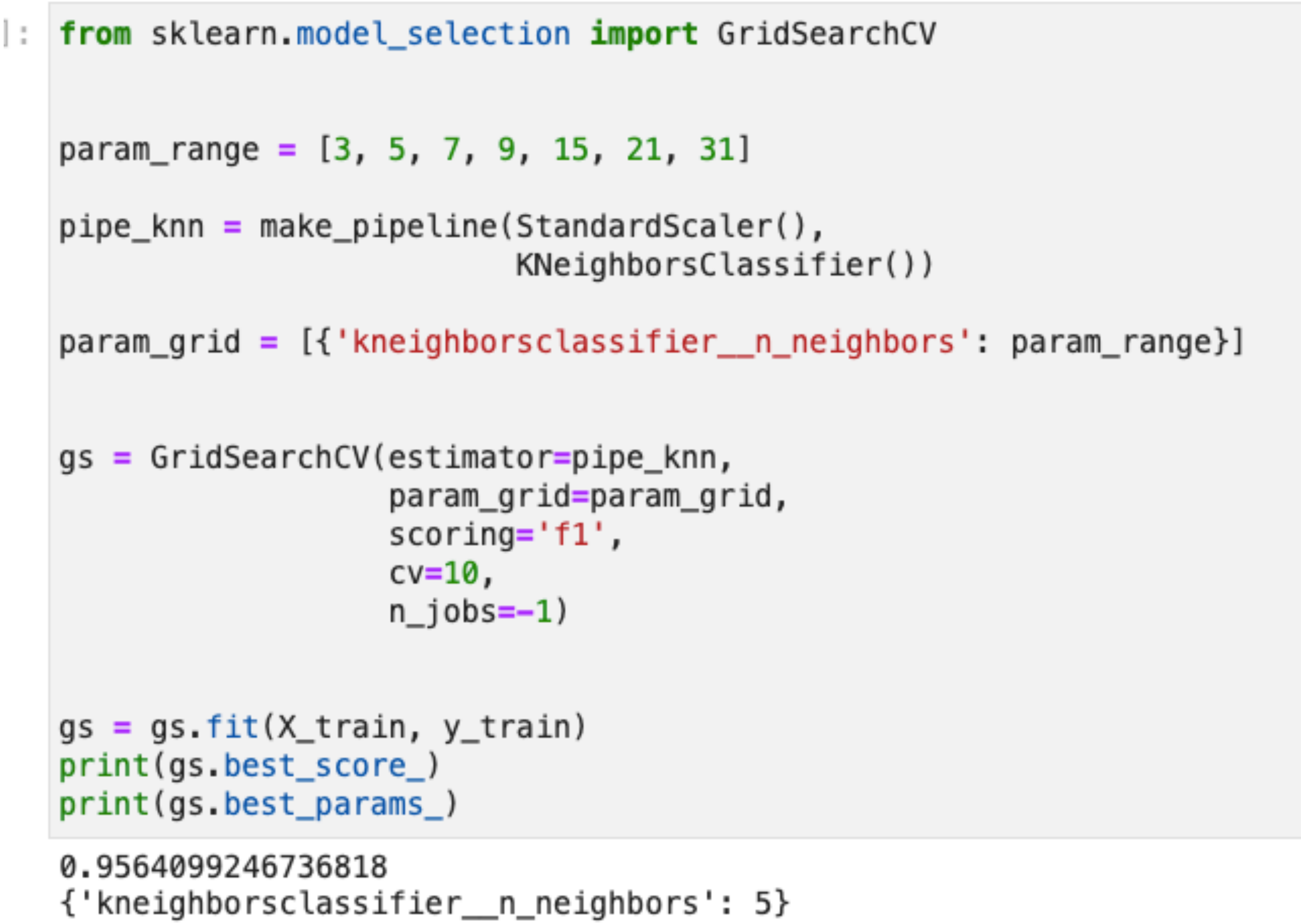
```
[[71  1]
 [ 3 39]]
```

```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
MCC: 0.925
```

https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_2_pre-recall-f1.ipynb

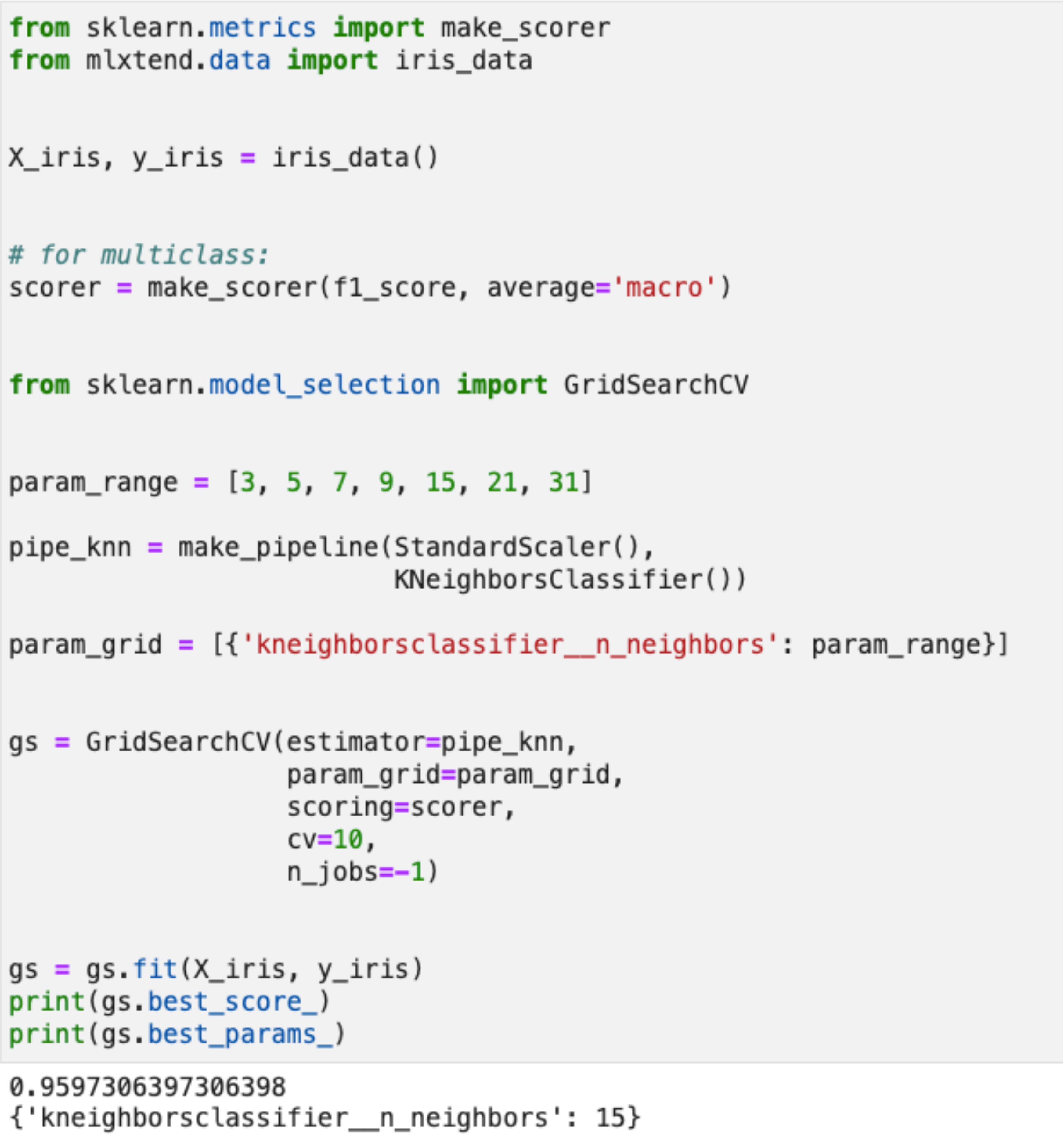
3) Using those Metrics in GridSearch



```
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}
```

https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_2_pre-recall-f1.ipynb



```
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}
```

https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_2_pre-recall-f1.ipynb

1. Confusion Matrix

2. Precision, Recall, and F1 Score

3. **Balanced Accuracy**

4. ROC

5. Extending Binary Metrics to Multi-class Settings

Balanced Accuracy / Average Per-Class (APC) Accuracy

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)

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

Predicted Labels

True Labels

	Class 0	Class 1	Class 2
Class 0	3	0	0
Class 1	7	50	12
Class 2	0	0	18

$$ACC = \frac{3 + 50 + 18}{90} \approx 0.79$$

$$APC\ ACC = \frac{83/90 + 71/90 + 78/90}{3} \approx 0.86$$

Balanced Accuracy / Average Per-Class Accuracy

Predicted Labels

True Labels

	Class 0	Neg Class
Class 0	3	0
Neg Class	7	80

Predicted Labels

True Labels

	Class 1	Neg Class
Class 1	50	19
Neg Class	0	21

Predicted Labels

True Labels

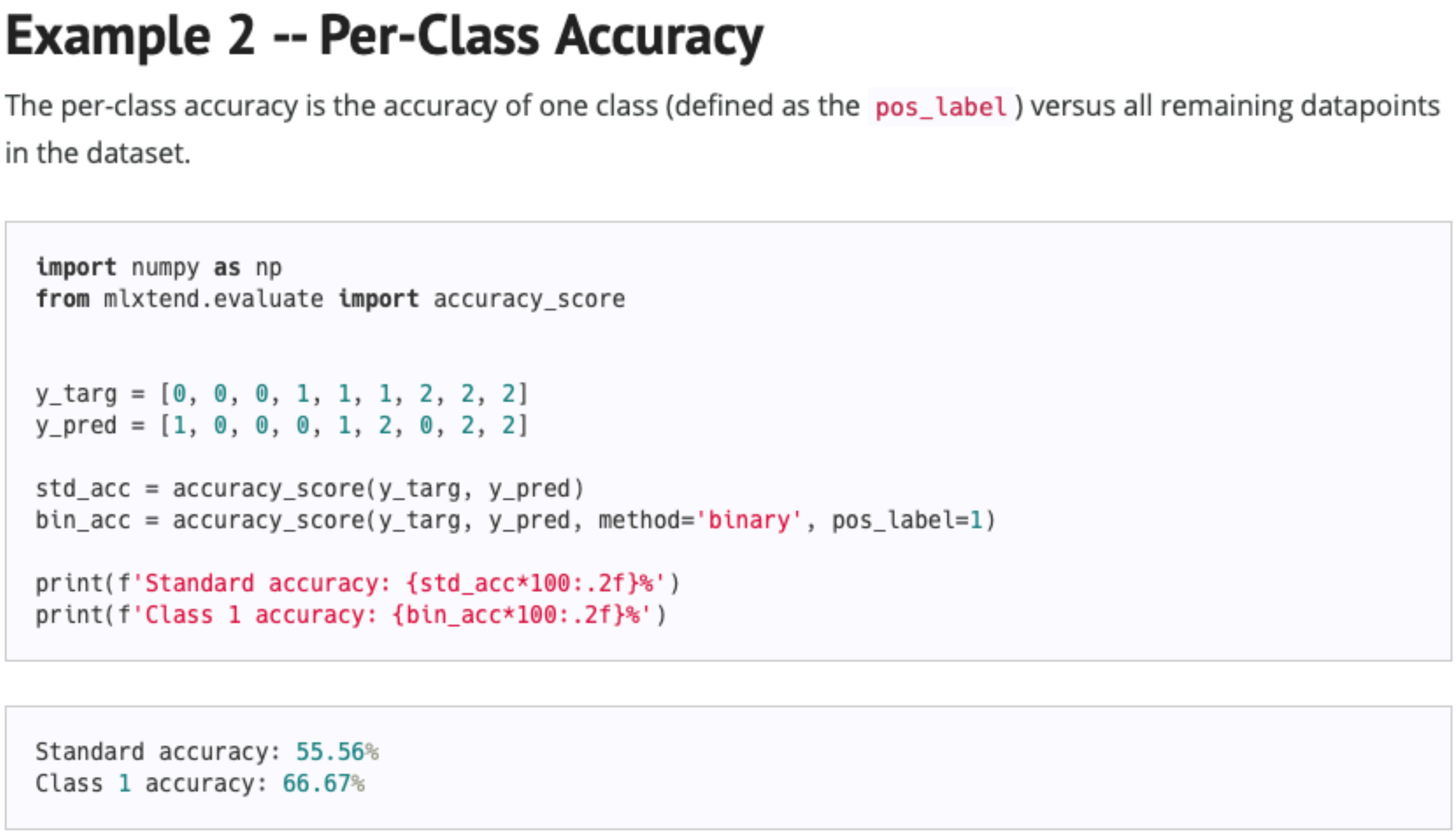
	Class 2	Neg Class
Class 2	18	0
Neg Class	12	60

Predicted Labels

True Labels

	Class 0	Class 1	Class 2
Class 0	3	0	0
Class 1	7	50	12
Class 2	0	0	18

$$APC\ ACC = \frac{83/90 + 71/90 + 78/90}{3} \approx 0.86$$



Example 2 -- Per-Class Accuracy

The per-class accuracy is the accuracy of one class (defined as the `pos_label`) versus all remaining datapoints in the dataset.

```
import numpy as np
from mlxtend.evaluate import accuracy_score


y_targ = [0, 0, 0, 1, 1, 1, 2, 2, 2]
y_pred = [1, 0, 0, 0, 1, 2, 0, 2, 2]

std_acc = accuracy_score(y_targ, y_pred)
bin_acc = accuracy_score(y_targ, y_pred, method='binary', pos_label=1)

print(f'Standard accuracy: {std_acc*100:.2f}%')
print(f'Class 1 accuracy: {bin_acc*100:.2f}%')
```

```
Standard accuracy: 55.56%
Class 1 accuracy: 66.67%
```

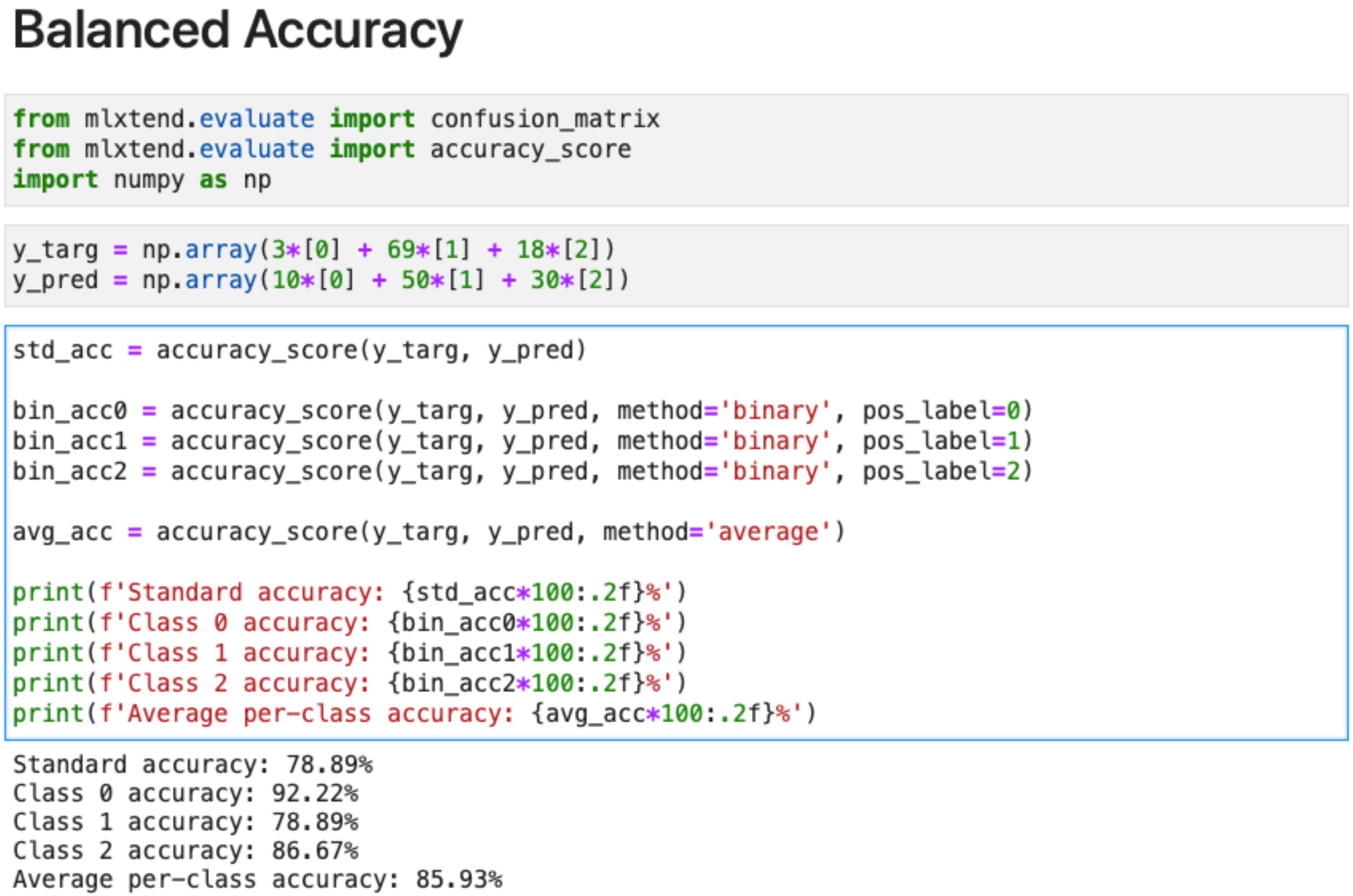
Predicted Labels

True Labels

	Class 0	Class 1	Class 2
Class 0	3	0	0
Class 1	7	50	12
Class 2	0	0	18

$$ACC = \frac{3 + 50 + 18}{90} \approx 0.79$$

$$APC\ ACC = \frac{83/90 + 71/90 + 78/90}{3} \approx 0.86$$



Balanced Accuracy

```
from mlxtend.evaluate import confusion_matrix
from mlxtend.evaluate import accuracy_score
import numpy as np
```

```
y_targ = np.array(3*[0] + 69*[1] + 18*[2])
y_pred = np.array(10*[0] + 50*[1] + 30*[2])
```

```
std_acc = accuracy_score(y_targ, y_pred)

bin_acc0 = accuracy_score(y_targ, y_pred, method='binary', pos_label=0)
bin_acc1 = accuracy_score(y_targ, y_pred, method='binary', pos_label=1)
bin_acc2 = accuracy_score(y_targ, y_pred, method='binary', pos_label=2)

avg_acc = accuracy_score(y_targ, y_pred, method='average')

print(f'Standard accuracy: {std_acc*100:.2f}%')
print(f'Class 0 accuracy: {bin_acc0*100:.2f}%')
print(f'Class 1 accuracy: {bin_acc1*100:.2f}%')
print(f'Class 2 accuracy: {bin_acc2*100:.2f}%')
print(f'Average per-class accuracy: {avg_acc*100:.2f}%')
```

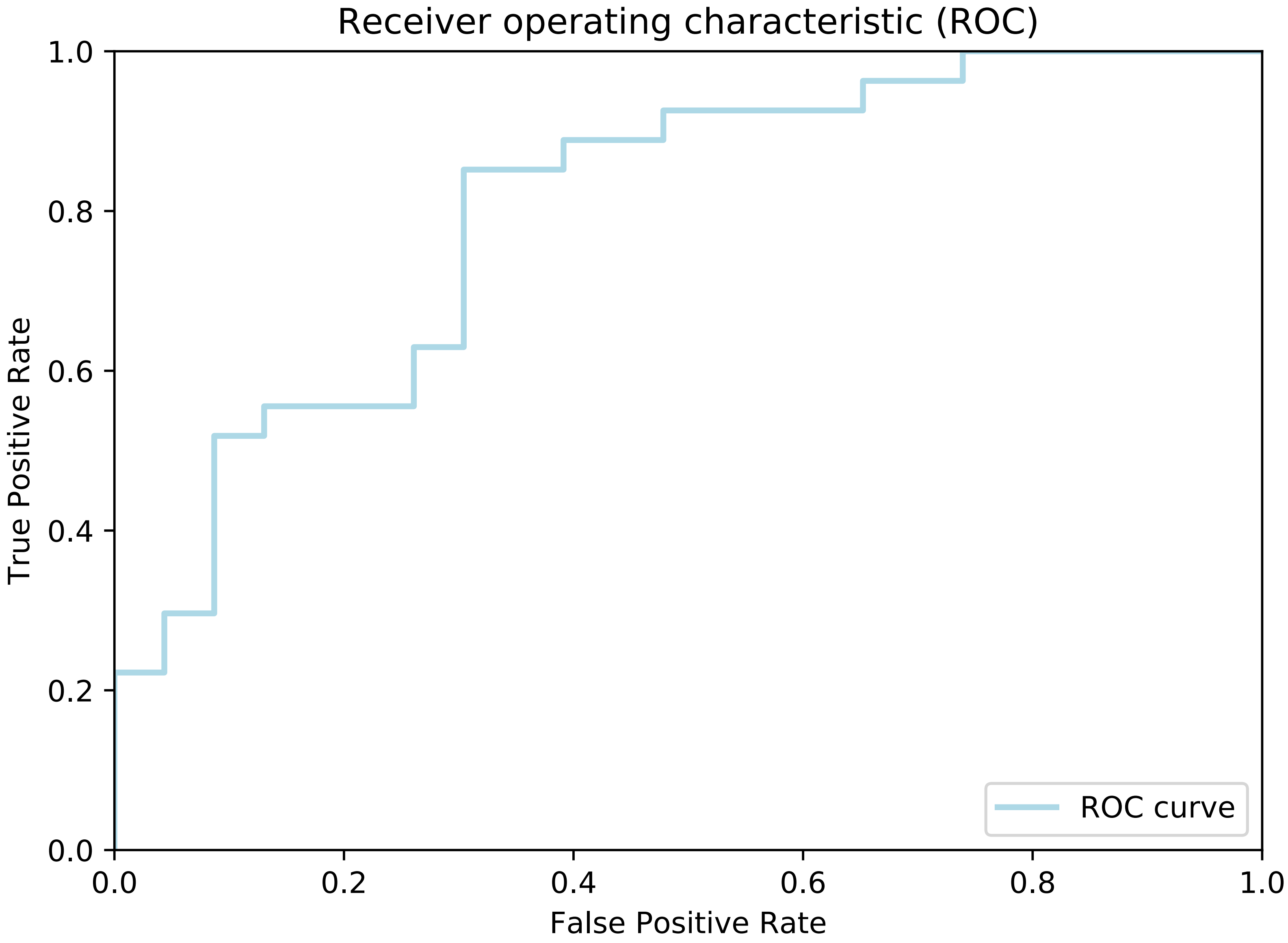
```
Standard accuracy: 78.89%
Class 0 accuracy: 92.22%
Class 1 accuracy: 78.89%
Class 2 accuracy: 86.67%
Average per-class accuracy: 85.93%
```

https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_3_balanced-acc-Copy1.ipynb

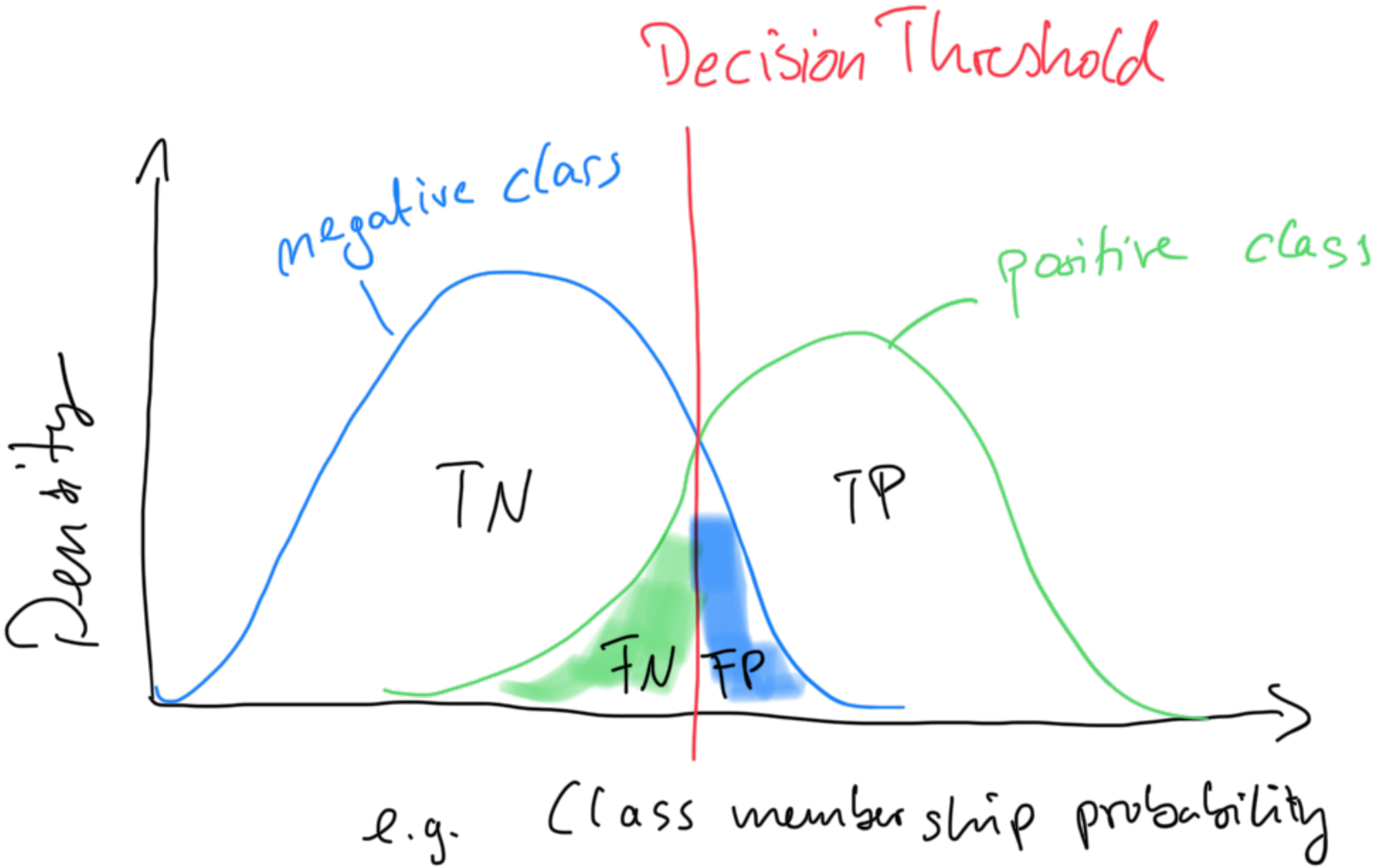
1. Confusion Matrix
2. Precision, Recall, and F1 Score
3. Balanced Accuracy
4. **ROC**
5. Extending Binary Metrics to Multi-class Settings

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)

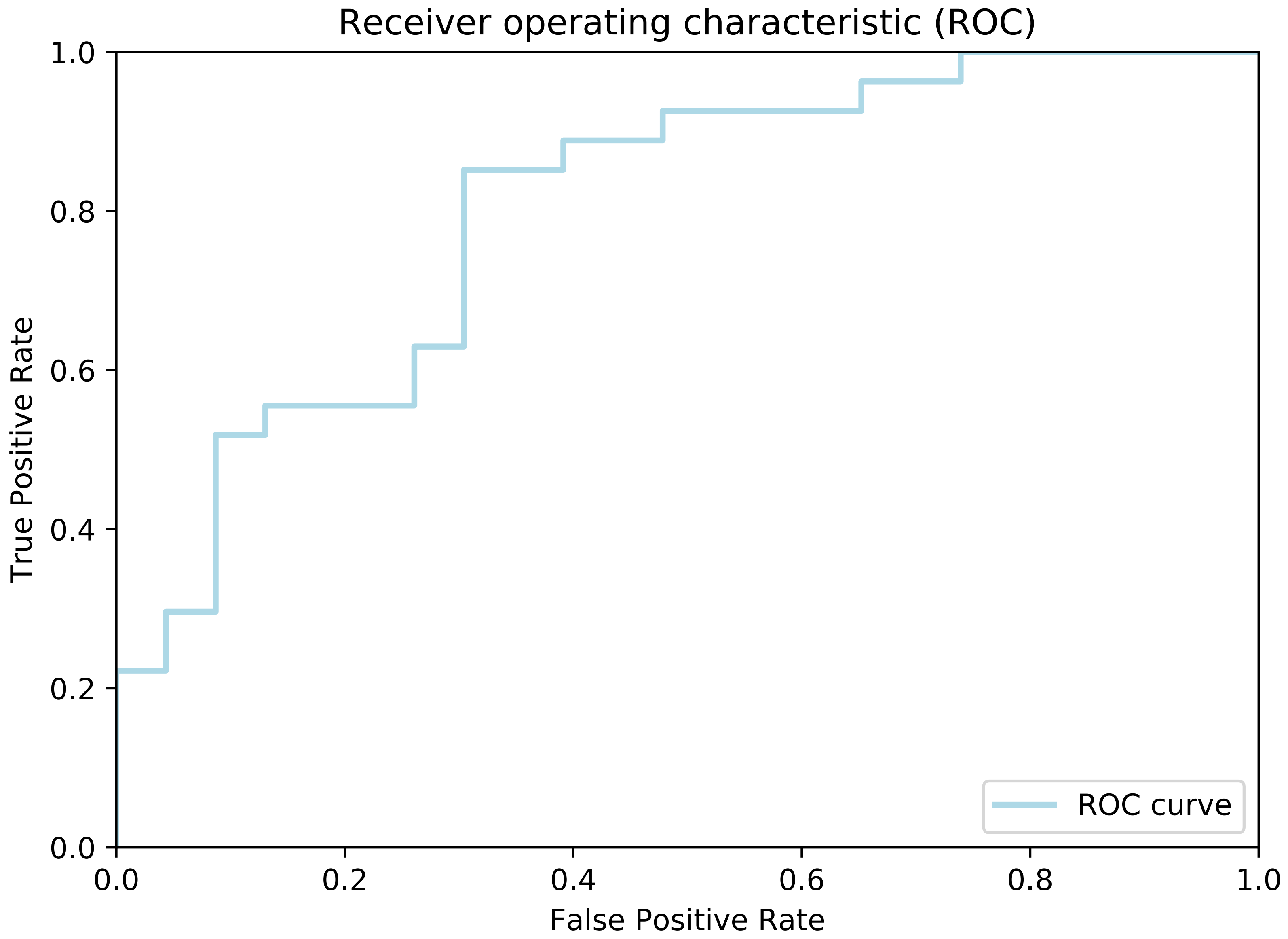


RECAP: False Positive Rate and False Negative Rate

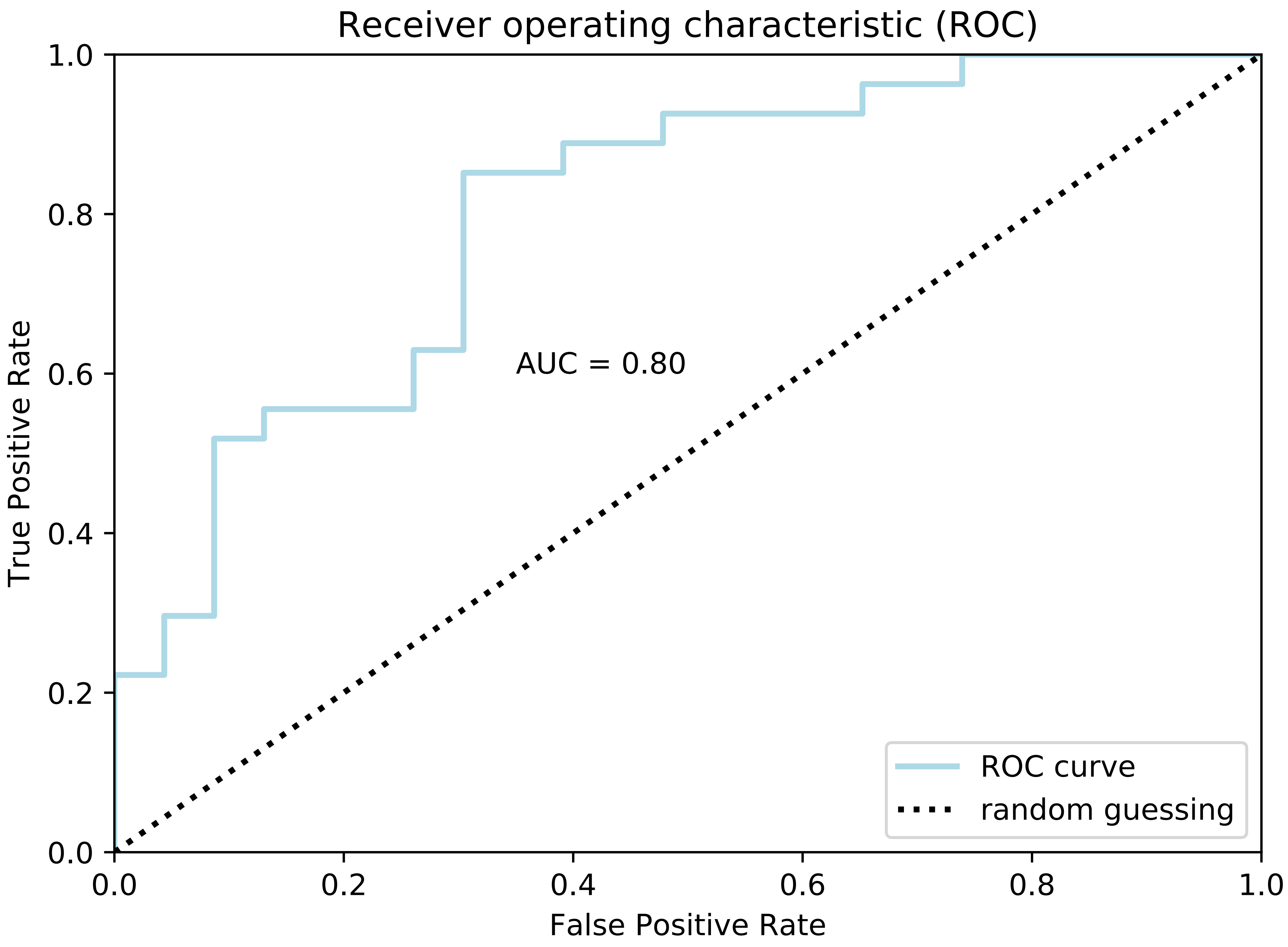


Receiver Operating Characteristic curve (ROC curve)

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

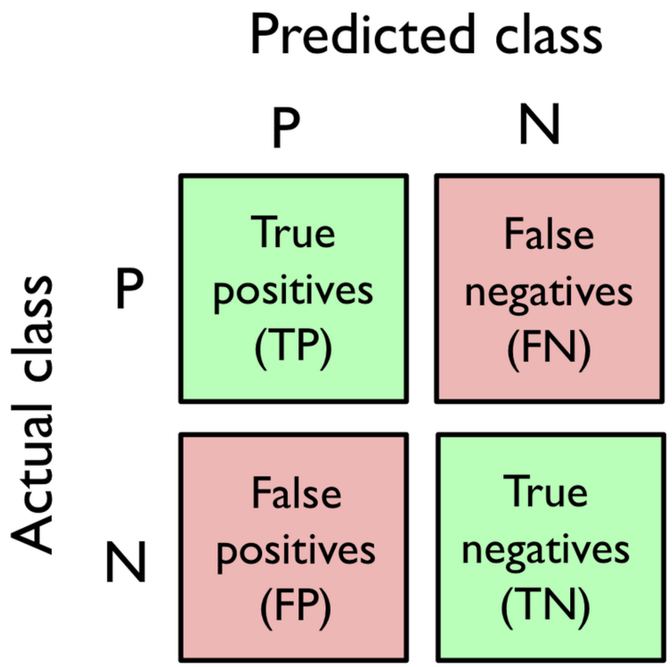


ROC Area Under the Curve (AUC)



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

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



Predicted class

Actual class	P	N
P	True positives (TP)	False negatives (FN)
N	False positives (FP)	True negatives (TN)

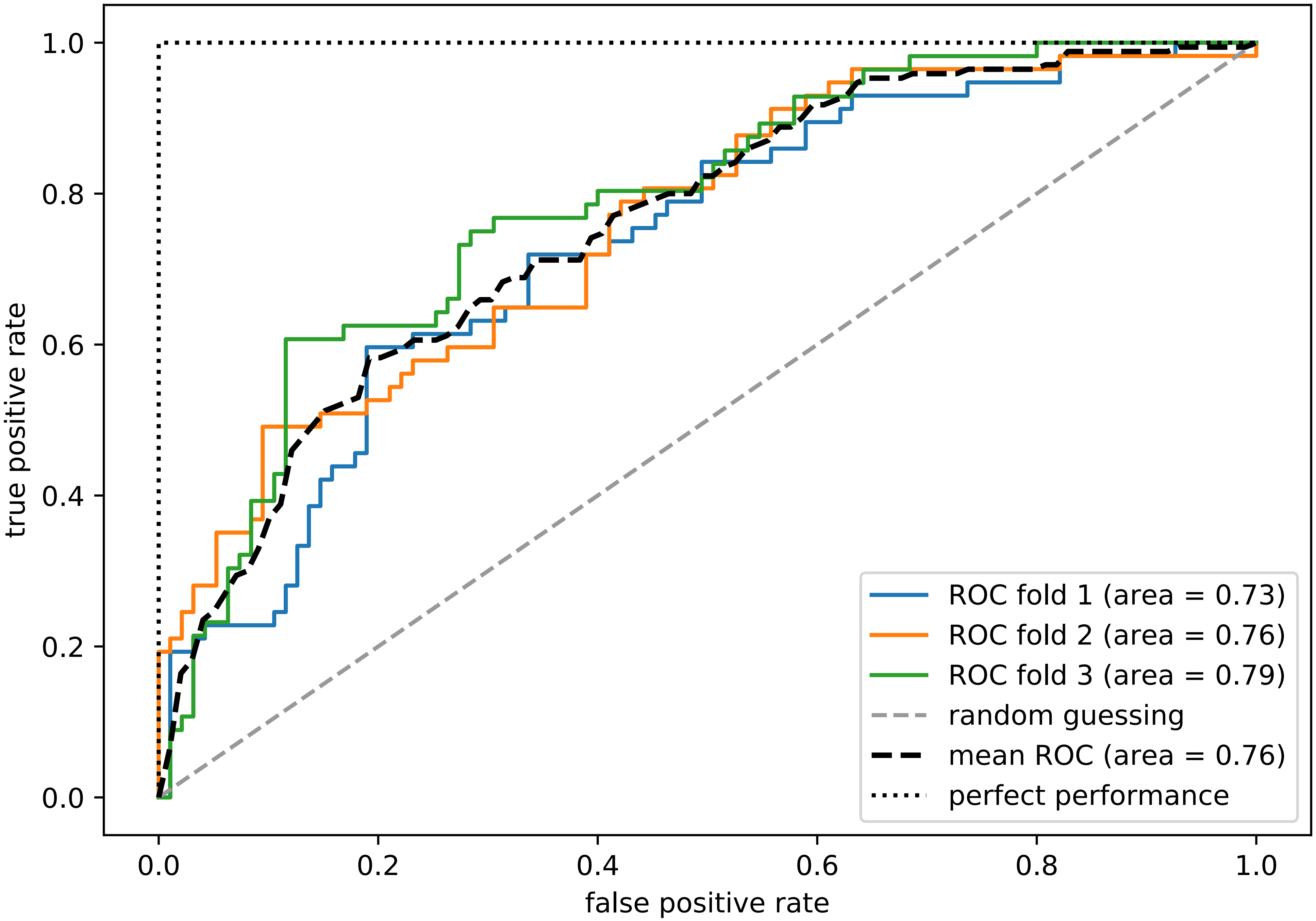
Balanced case: 100 100
100 100

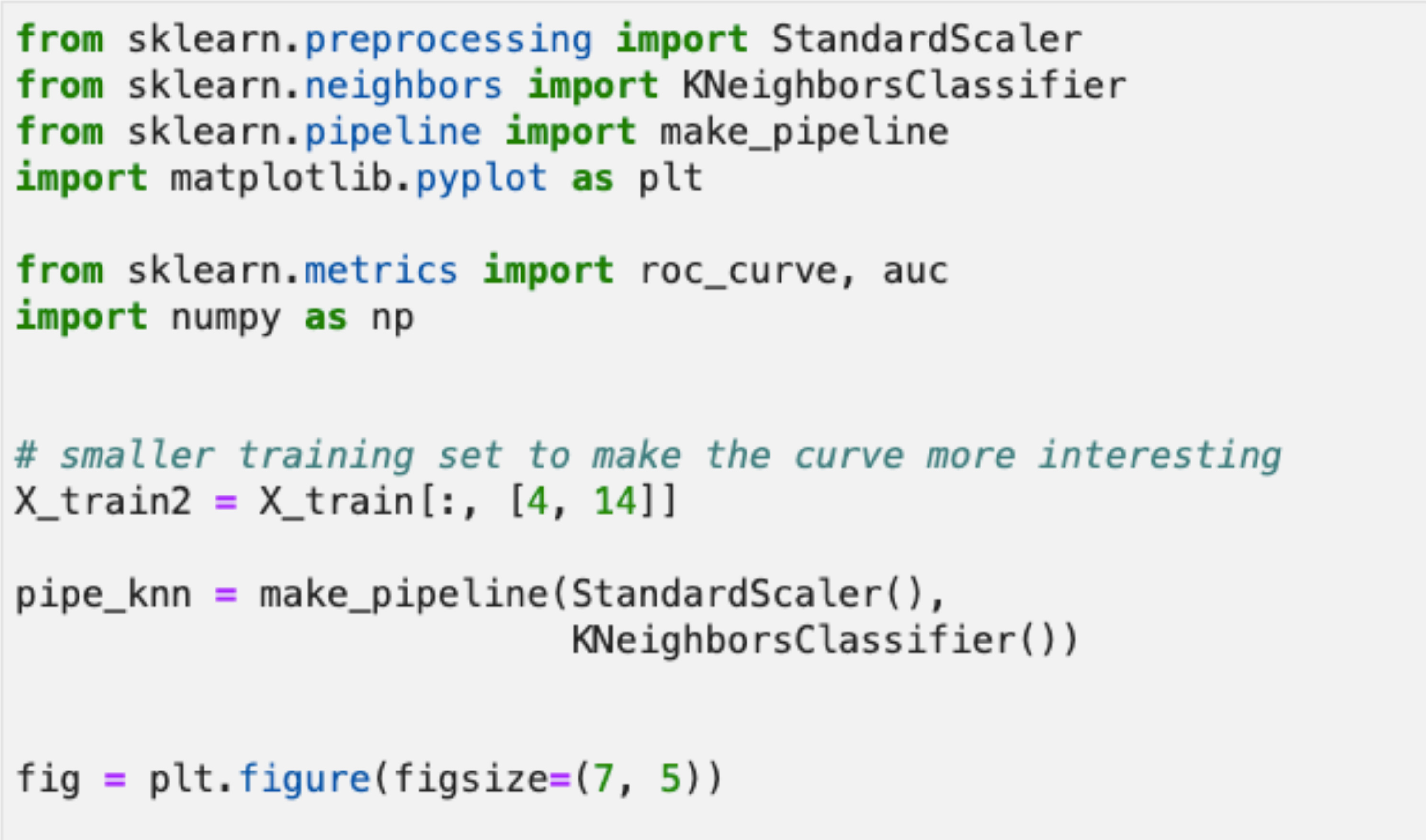
TPR = 100/200 = 0.5
FPR = 100/200 = 0.5

Imbalanced case: 200 200
50 50

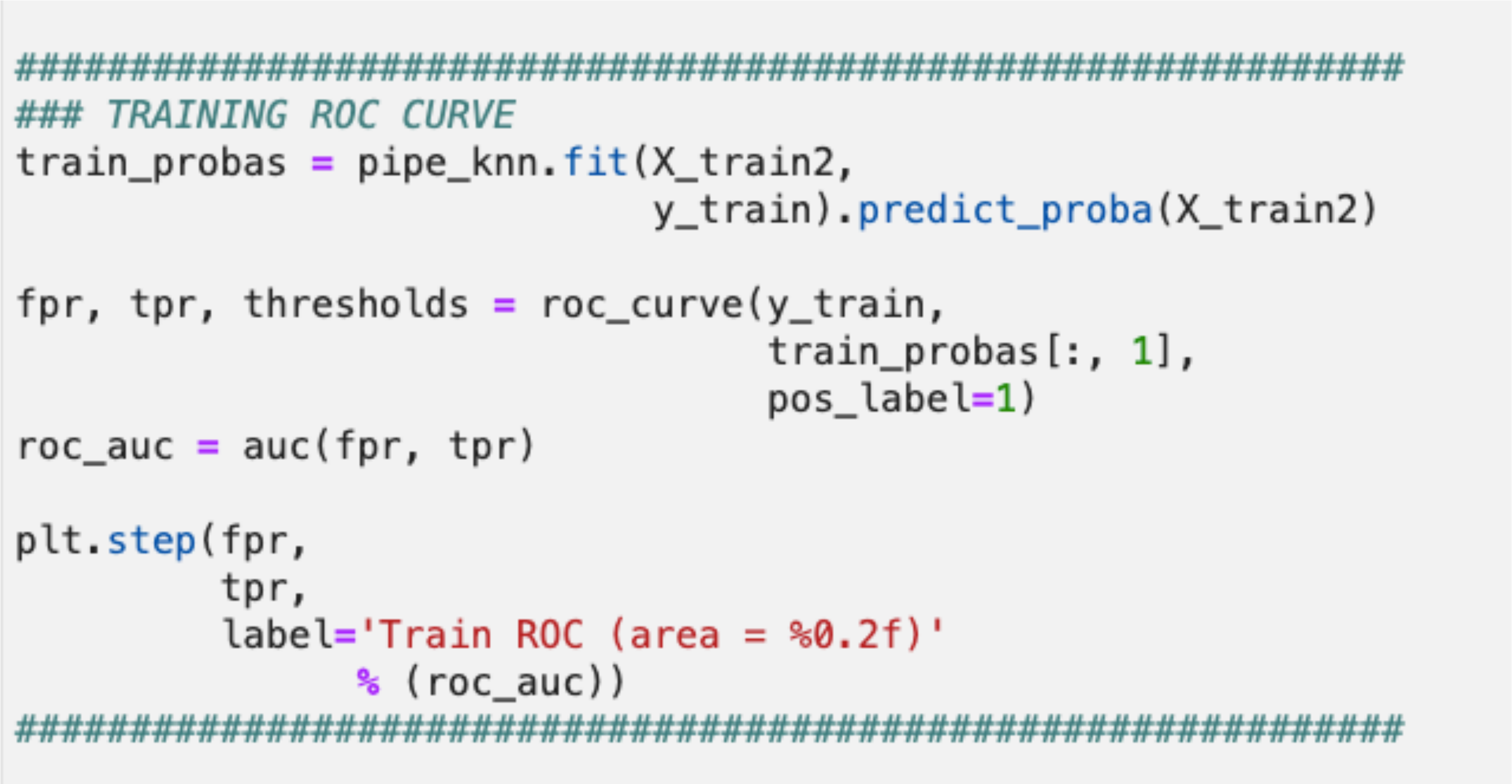
TPR = 200/400 = 0.5
FPR = 50/100 = 0.5

ROC and k-Fold Cross-Validation



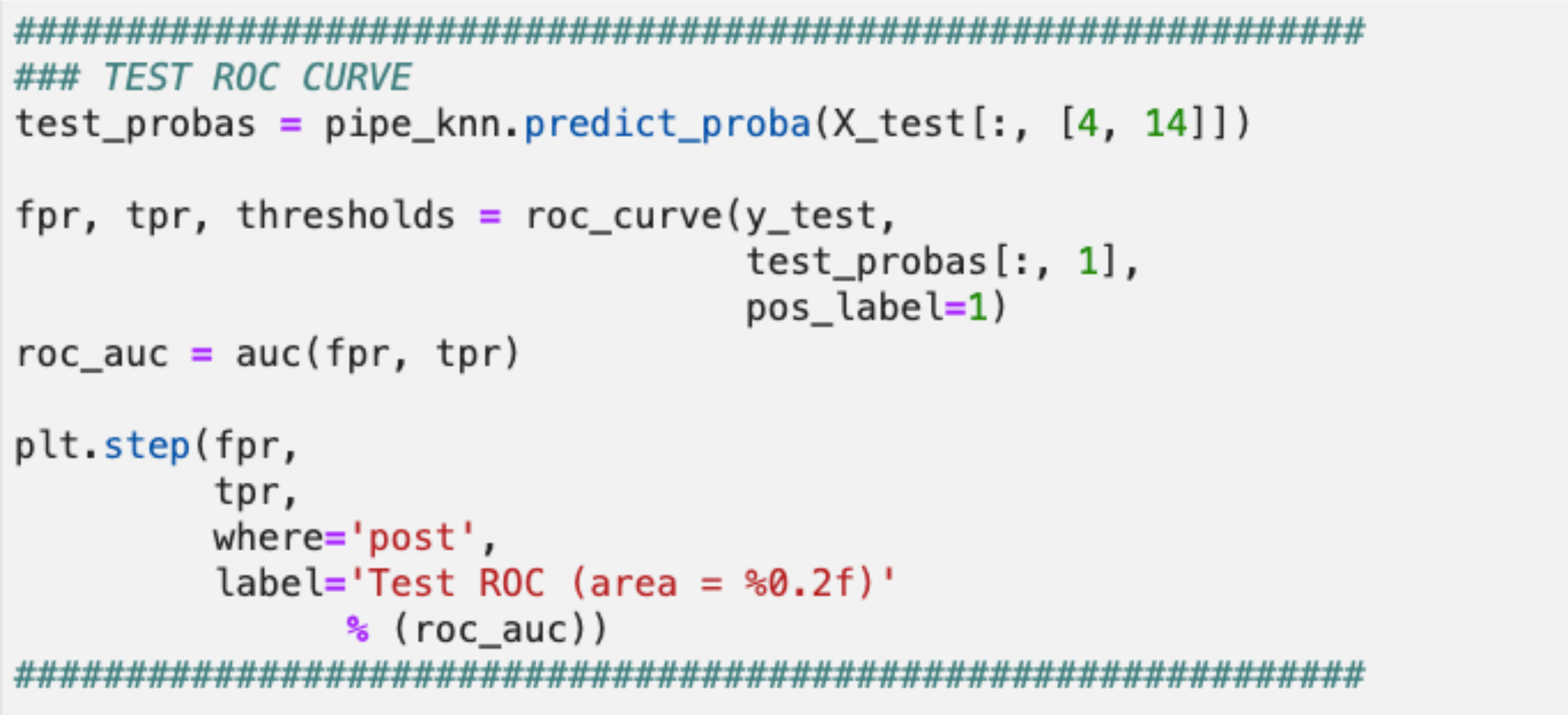


```python
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
import matplotlib.pyplot as plt

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))
```

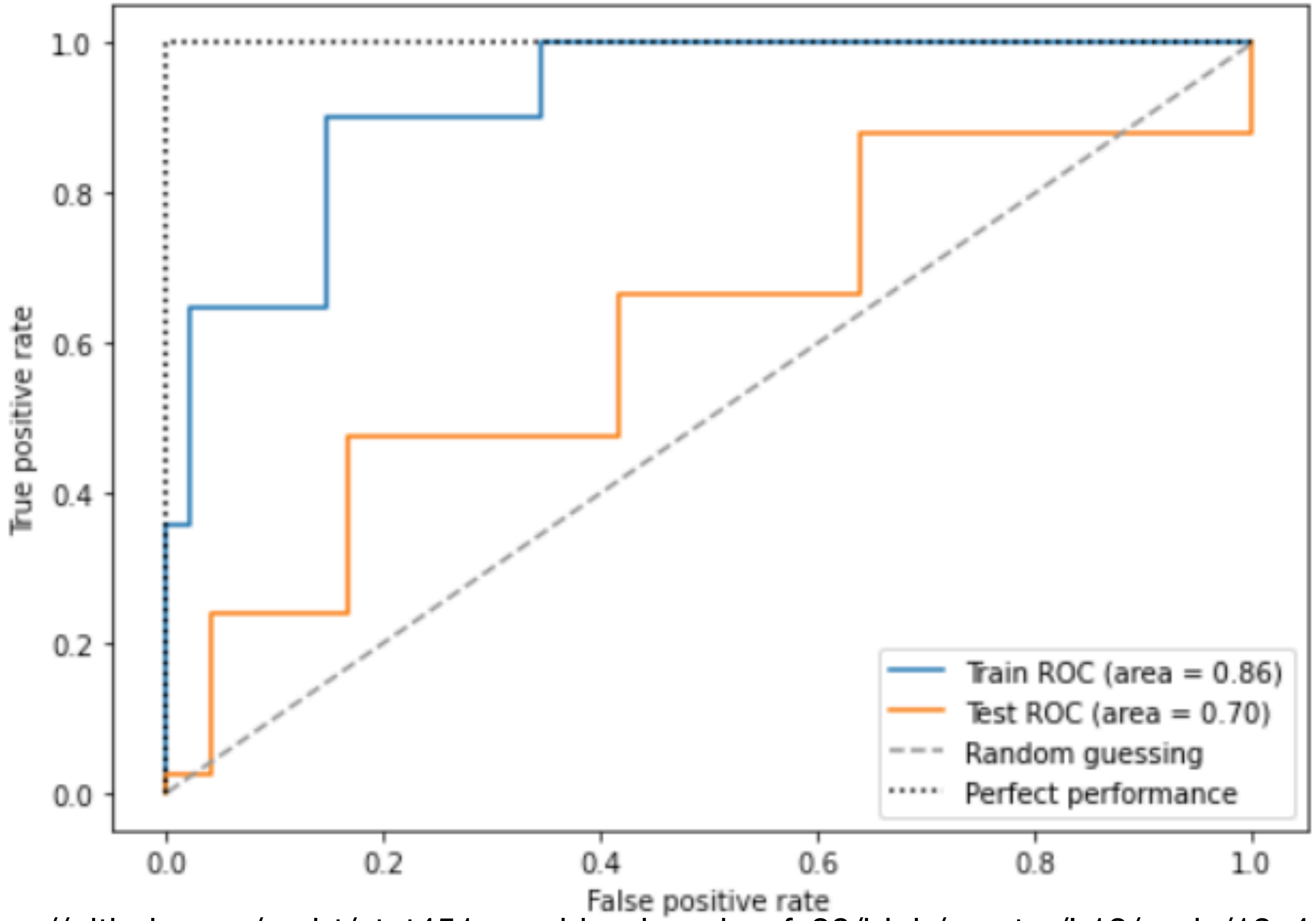


```python
##################################################################
### TRAINING ROC CURVE
train_probas = pipe_knn.fit(X_train2,
                            y_train).predict_proba(X_train2)

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))
##################################################################
```

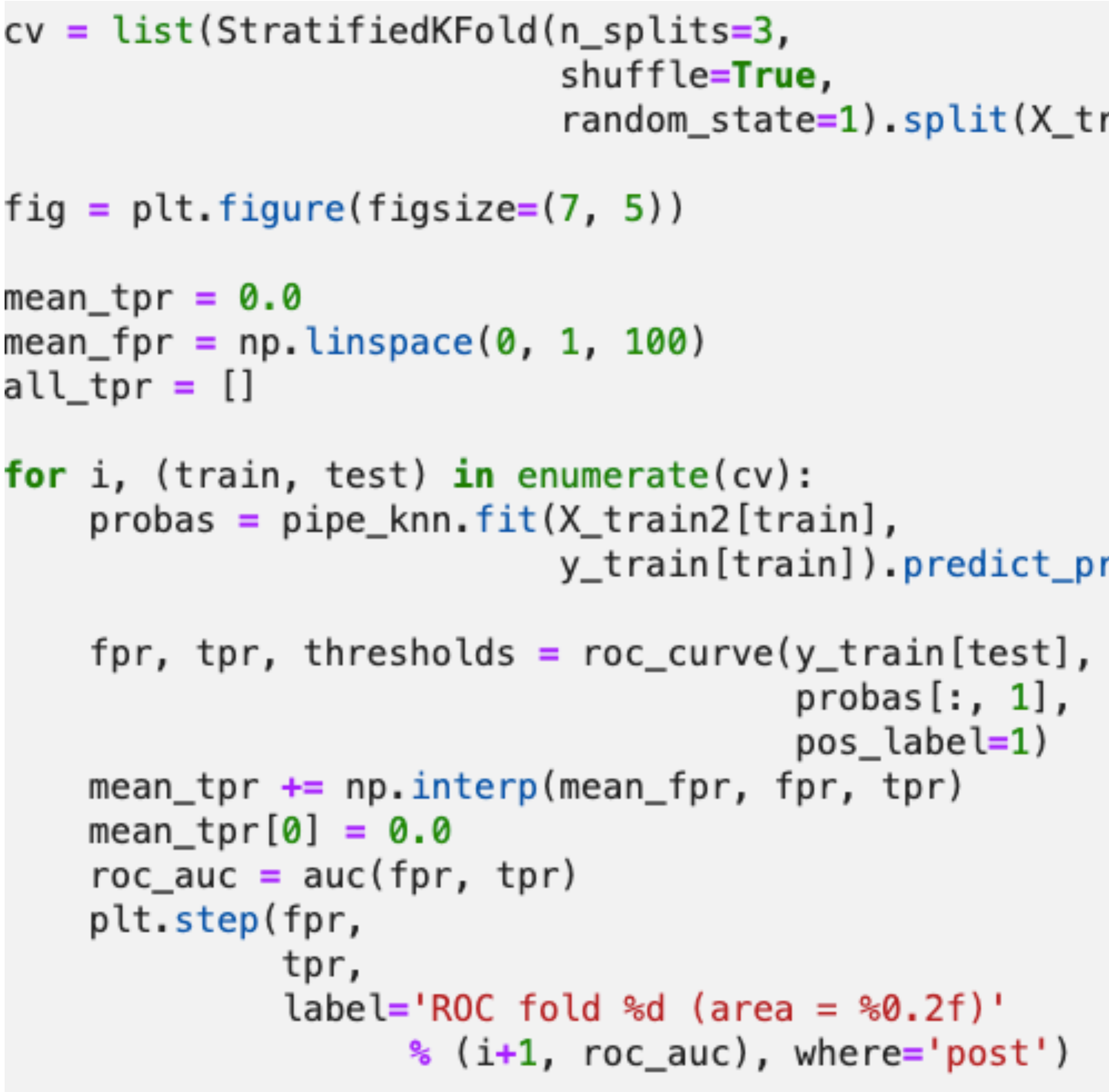
https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_4_roc.ipynb



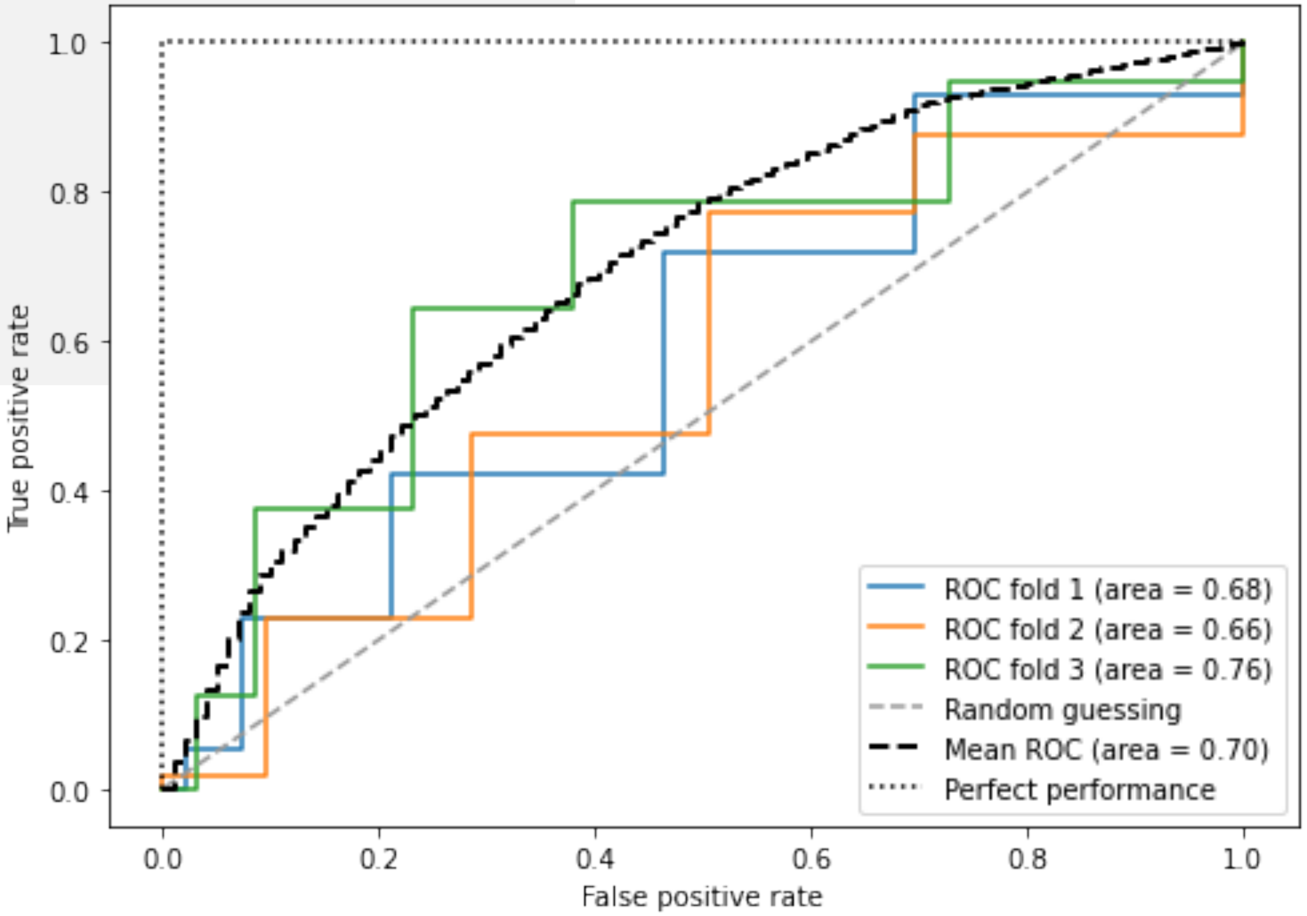
```
##############################################################
### TEST ROC CURVE
test_probas = pipe_knn.predict_proba(X_test[:, [4, 14]])

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))
##############################################################
```



https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_4_roc.ipynb



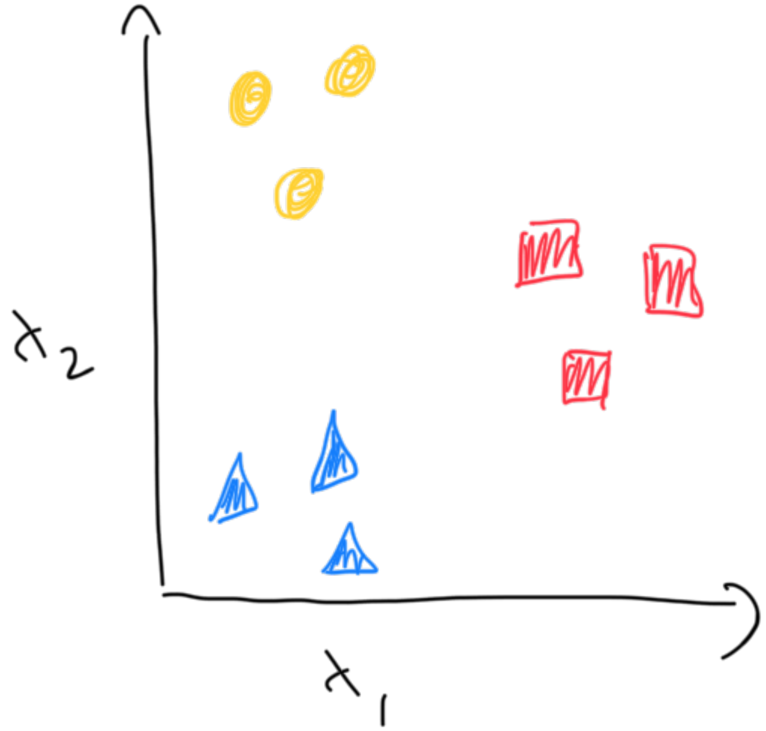
```
cv = list(StratifiedKFold(n_splits=3,
                          shuffle=True,
                          random_state=1).split(X_train, y_train))

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')
```

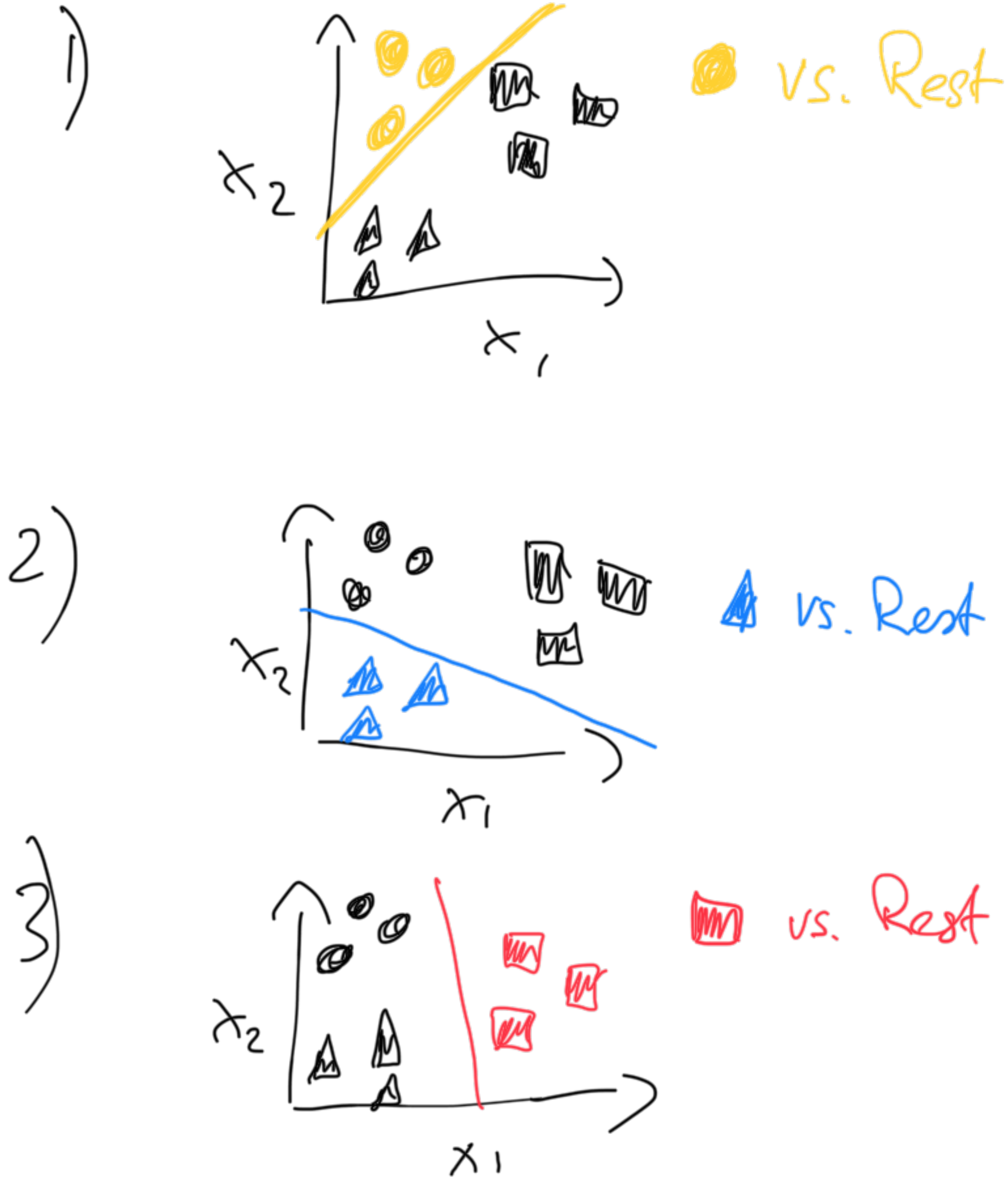


https://github.com/rasbt/stat451-machine-learning-fs20/blob/master/L12/code/12_4_roc.ipynb

1. Confusion Matrix
2. Precision, Recall, and F1 Score
3. Balanced Accuracy
4. ROC
5. **Extending Binary Metrics to Multi-class Settings**

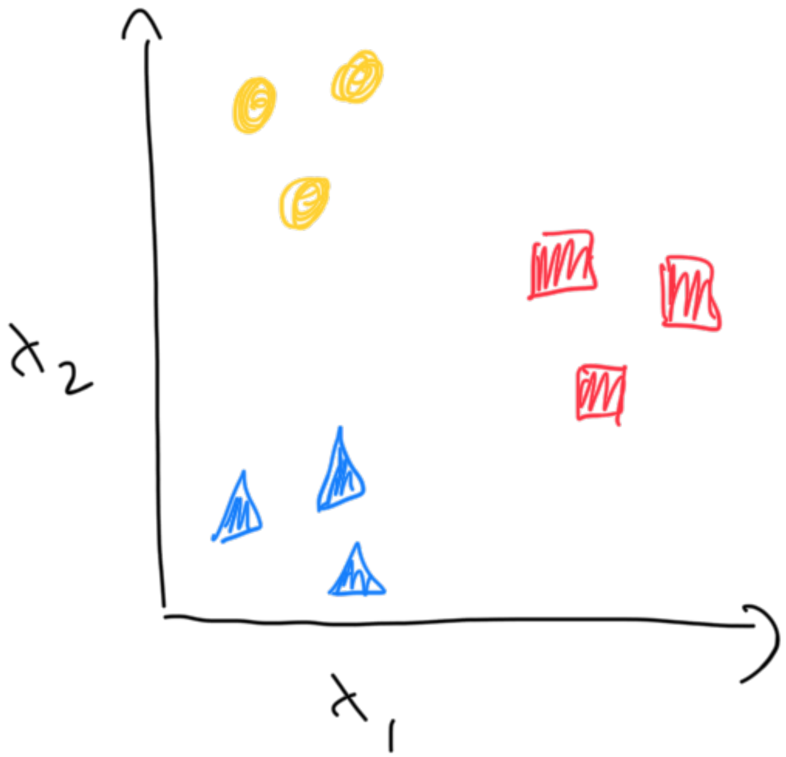
Binary Classifiers and One-vs-Rest (OvR) / One-vs-All (OvA)

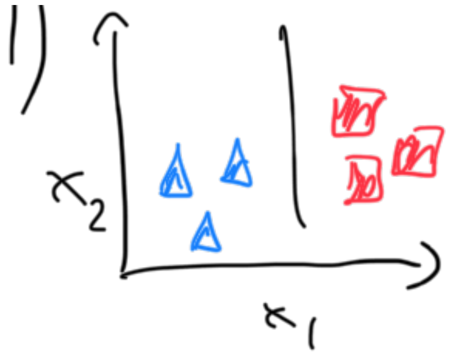


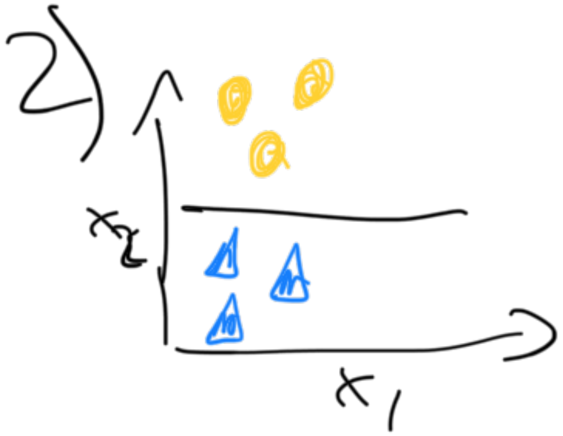


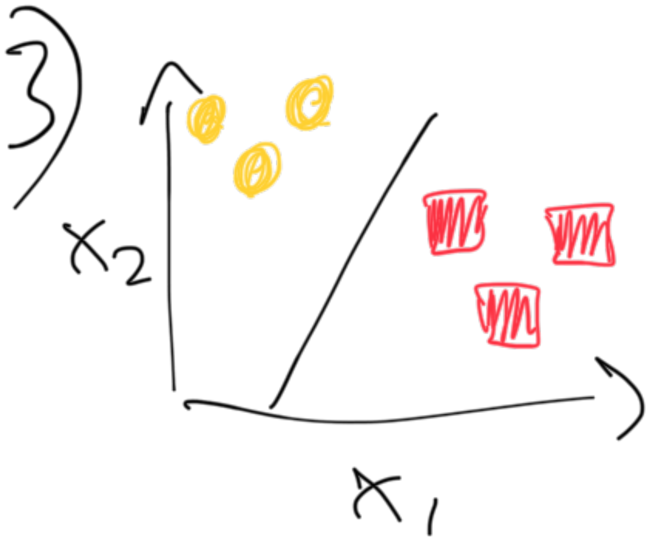
Then, choose the class with the highest confidence score

Binary Classifiers and One-vs-One (OvO) / All-vs-All (AvA)









num_classes x (num_classes - 1) / 2

Big O: O(?)

Select the class by majority vote (and use confidence score in case of ties)

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.

Balanced Accuracy / Average Per-Class Accuracy

Predicted Labels

		Class 0	Neg Class
True Labels	Class 0	3	0
	Neg Class	7	80

Predicted Labels

		Class 1	Neg Class
True Labels	Class 1	50	19
	Neg Class	0	21

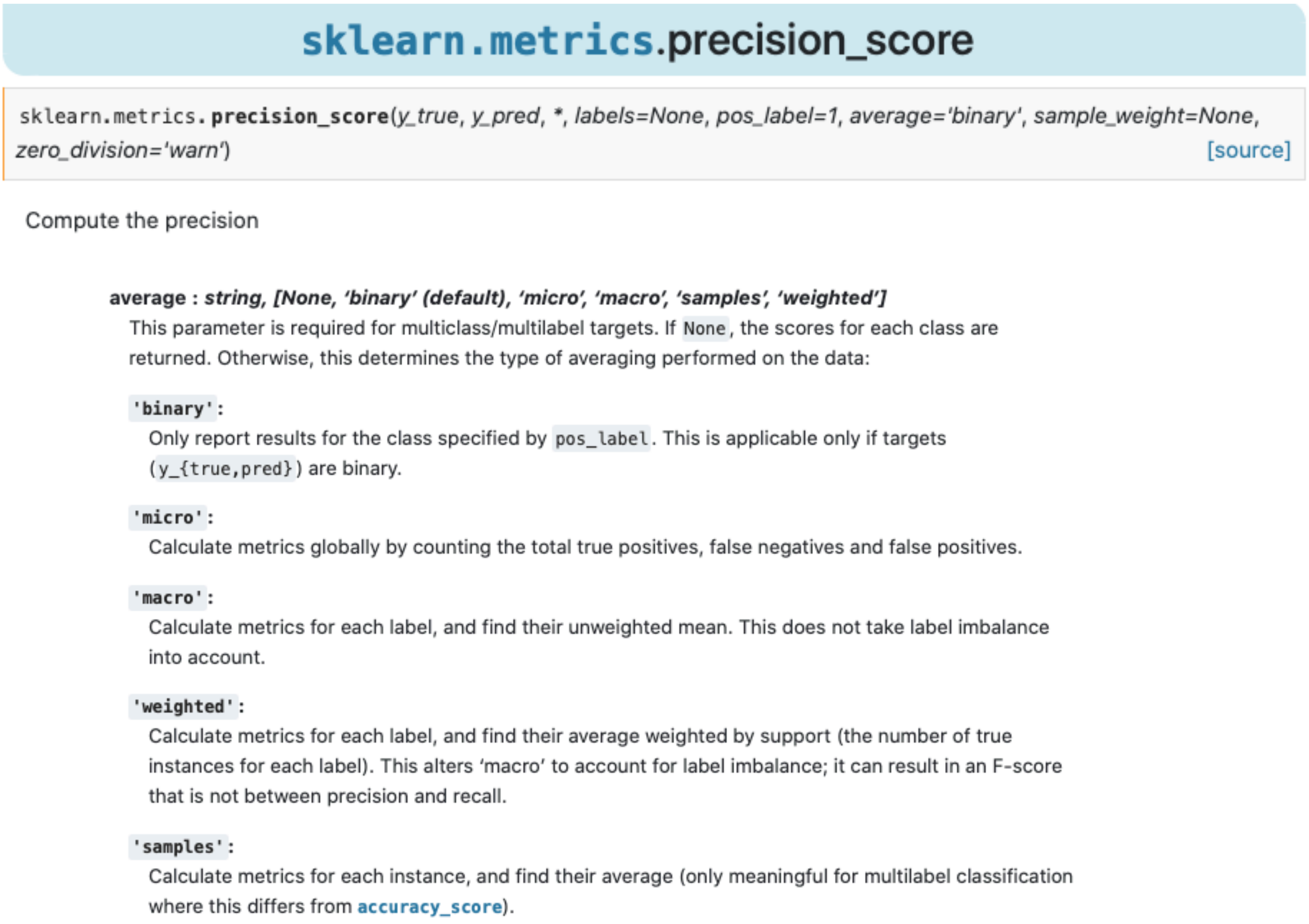
Predicted Labels

		Class 2	Neg Class
True Labels	Class 2	18	0
	Neg Class	12	60

Predicted Labels

		Class 0	Class 1	Class 2
True Labels	Class 0	3	0	0
	Class 1	7	50	12
	Class 2	0	0	18

$$APC\ ACC = \frac{83/90 + 71/90 + 78/90}{3} \approx 0.86$$



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:

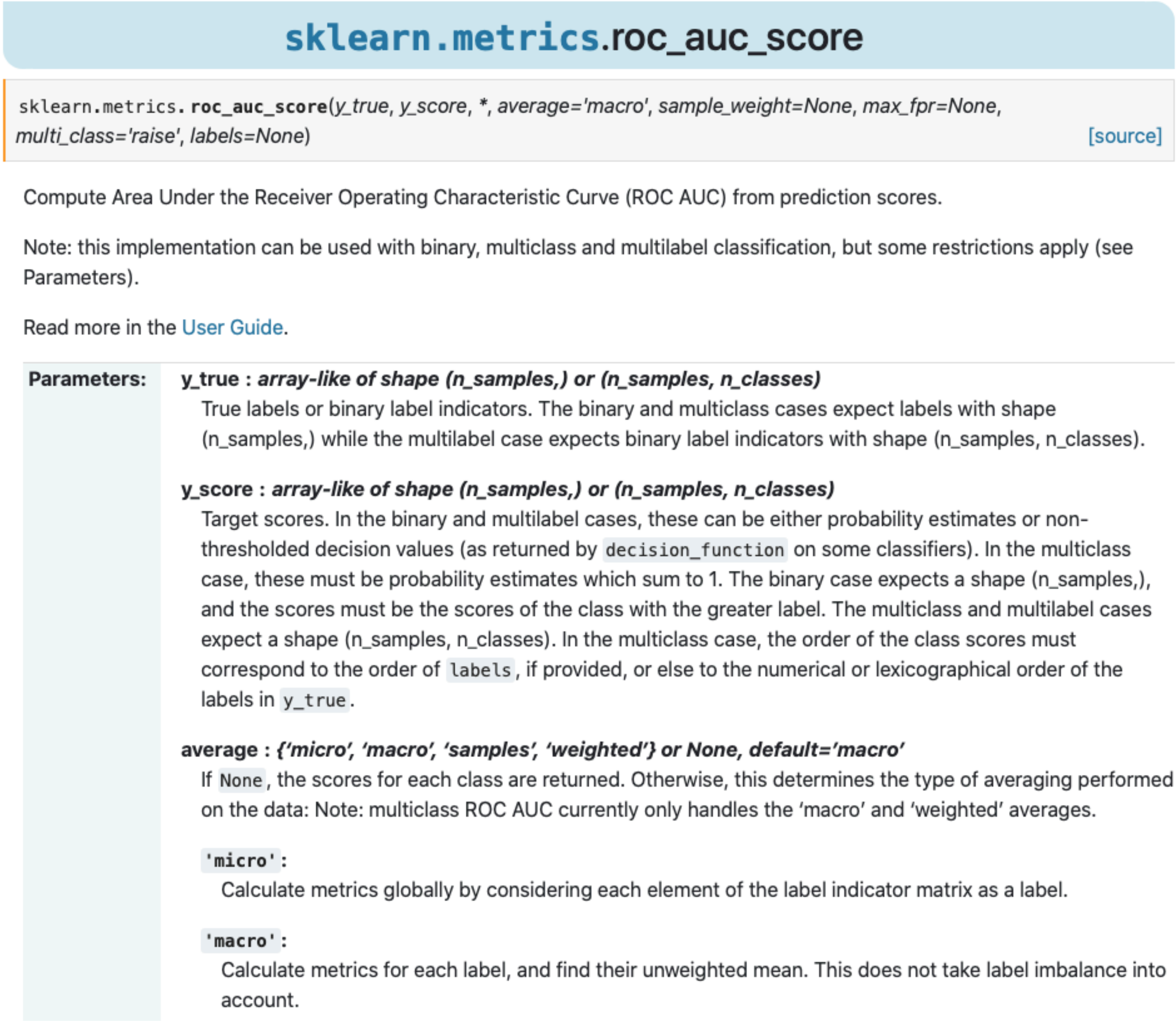
`'binary'`:
Only report results for the class specified by `pos_label`. This is applicable only if targets (`y_{true,pred}`) are binary.

`'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.

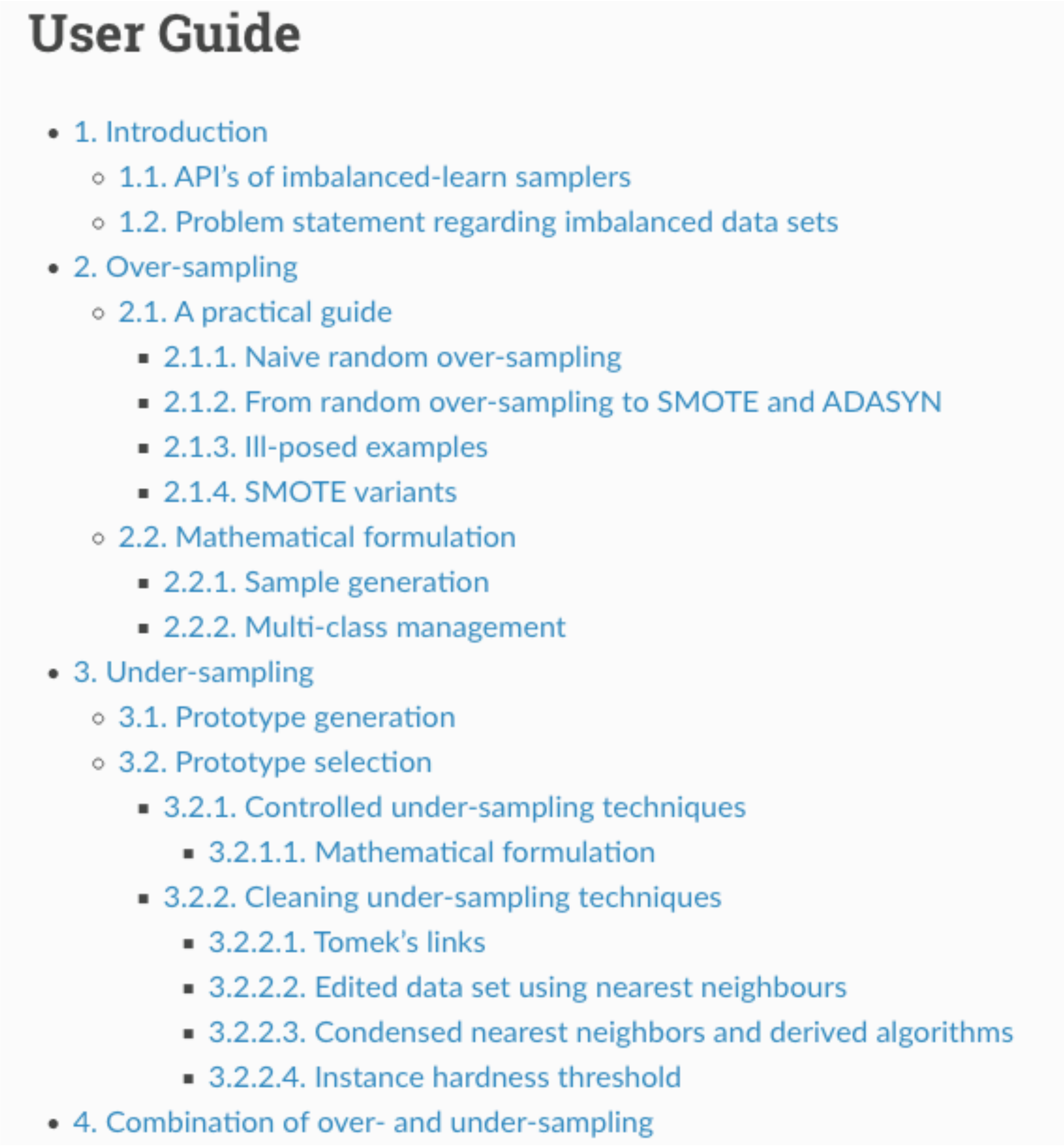
`'micro'`:

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.

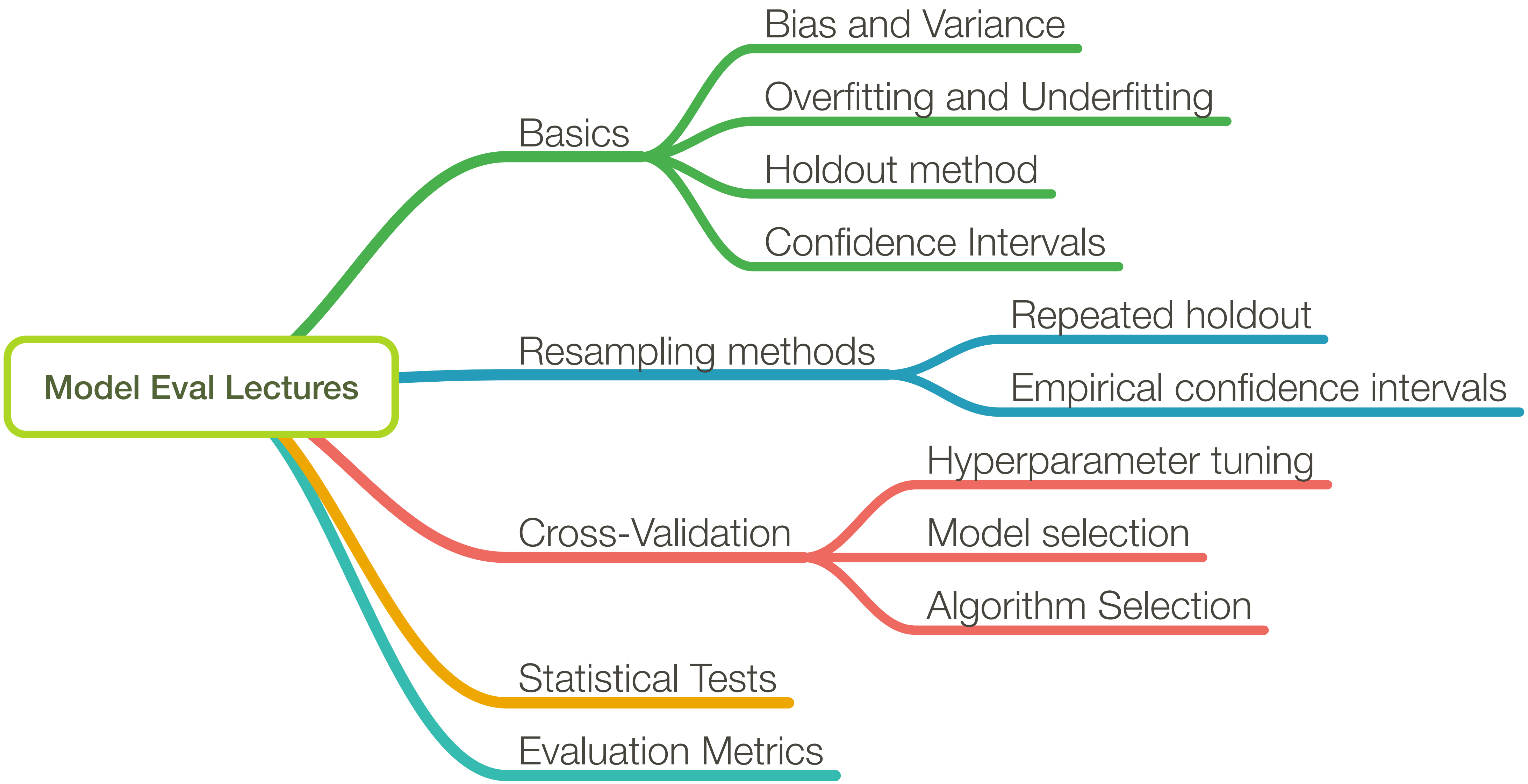
Dealing with Class Imbalance



User Guide

- 1. Introduction
  - 1.1. API's of imbalanced-learn samplers
  - 1.2. Problem statement regarding imbalanced data sets
- 2. Over-sampling
  - 2.1. A practical guide
    - 2.1.1. Naive random over-sampling
    - 2.1.2. From random over-sampling to SMOTE and ADASYN
    - 2.1.3. Ill-posed examples
    - 2.1.4. SMOTE variants
  - 2.2. Mathematical formulation
    - 2.2.1. Sample generation
    - 2.2.2. Multi-class management
- 3. Under-sampling
  - 3.1. Prototype generation
  - 3.2. Prototype selection
    - 3.2.1. Controlled under-sampling techniques
      - 3.2.1.1. Mathematical formulation
    - 3.2.2. Cleaning under-sampling techniques
      - 3.2.2.1. Tomek's links
      - 3.2.2.2. Edited data set using nearest neighbours
      - 3.2.2.3. Condensed nearest neighbors and derived algorithms
      - 3.2.2.4. Instance hardness threshold
- 4. Combination of over- and under-sampling

https://imbalanced-learn.readthedocs.io/en/stable/user_guide.html


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