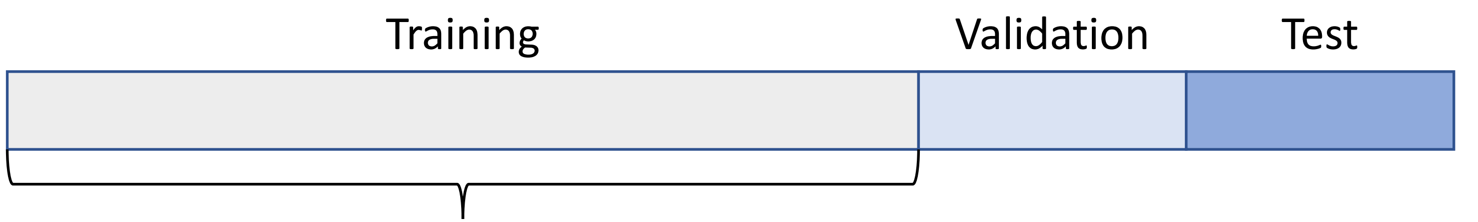
Machine Learning Workshop: Model Evaluation

Selecting data for model evaluation

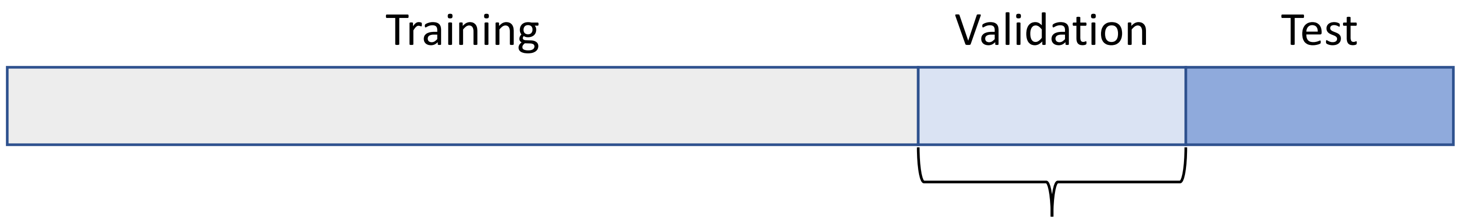
Selecting data for model evaluation



Use the majority of the data to train/fit the model, providing many examples to “learn” from.

<u>Never</u> use these examples to evaluate model performance!
Reason: Observations used to train the model will have higher model performance simply because the model has already seen this data.

Selecting data for model evaluation

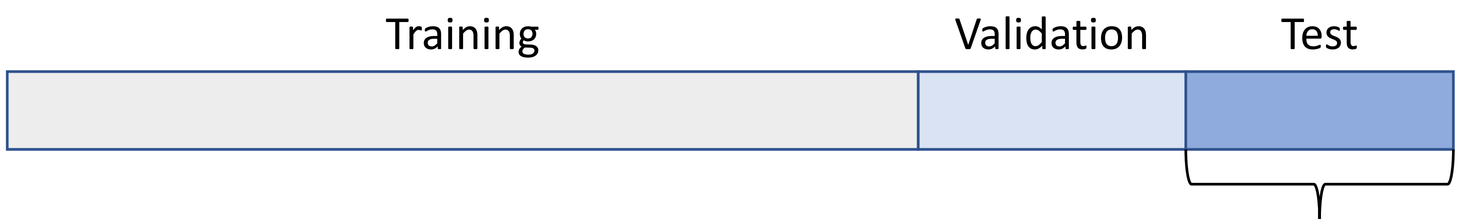


Leave out a proportion of the data to adjust or ”tune” model parameter settings (e.g., the number of nearest neighbors).

Note: We only use this to evaluate the effect of adjusting these parameters, and performances based on these points.

Similar to training data, by continuously adjusting the model for the best results, we indirectly provide the model the validation labels. The model parameters are become biased toward the validation set.

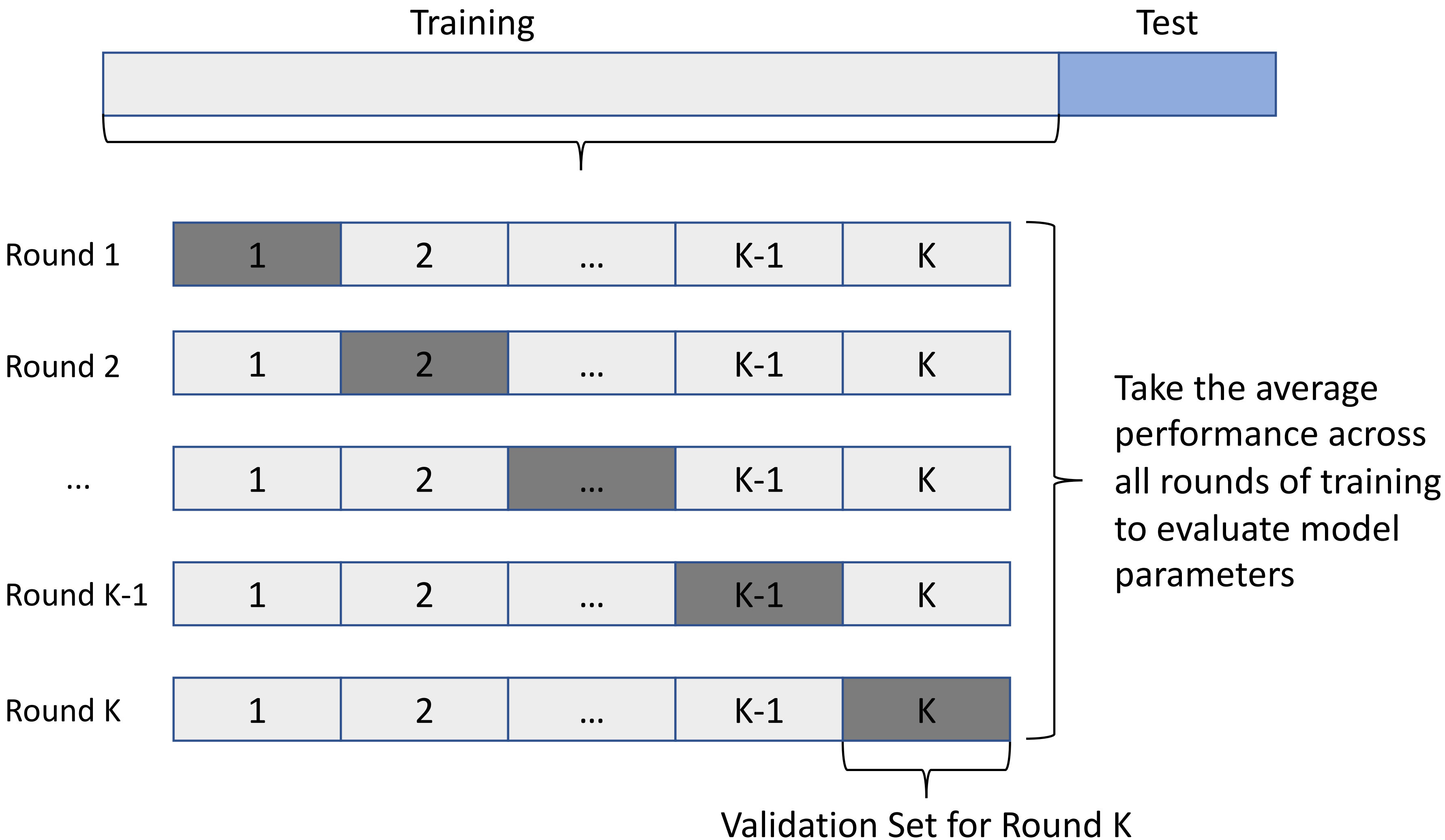
Selecting data for model evaluation



The left out test set informs us how well the models perform.

This data has not been seen by the models (either directly or indirectly) and therefore will be a fair assessment of how well the model can perform for the classification task.

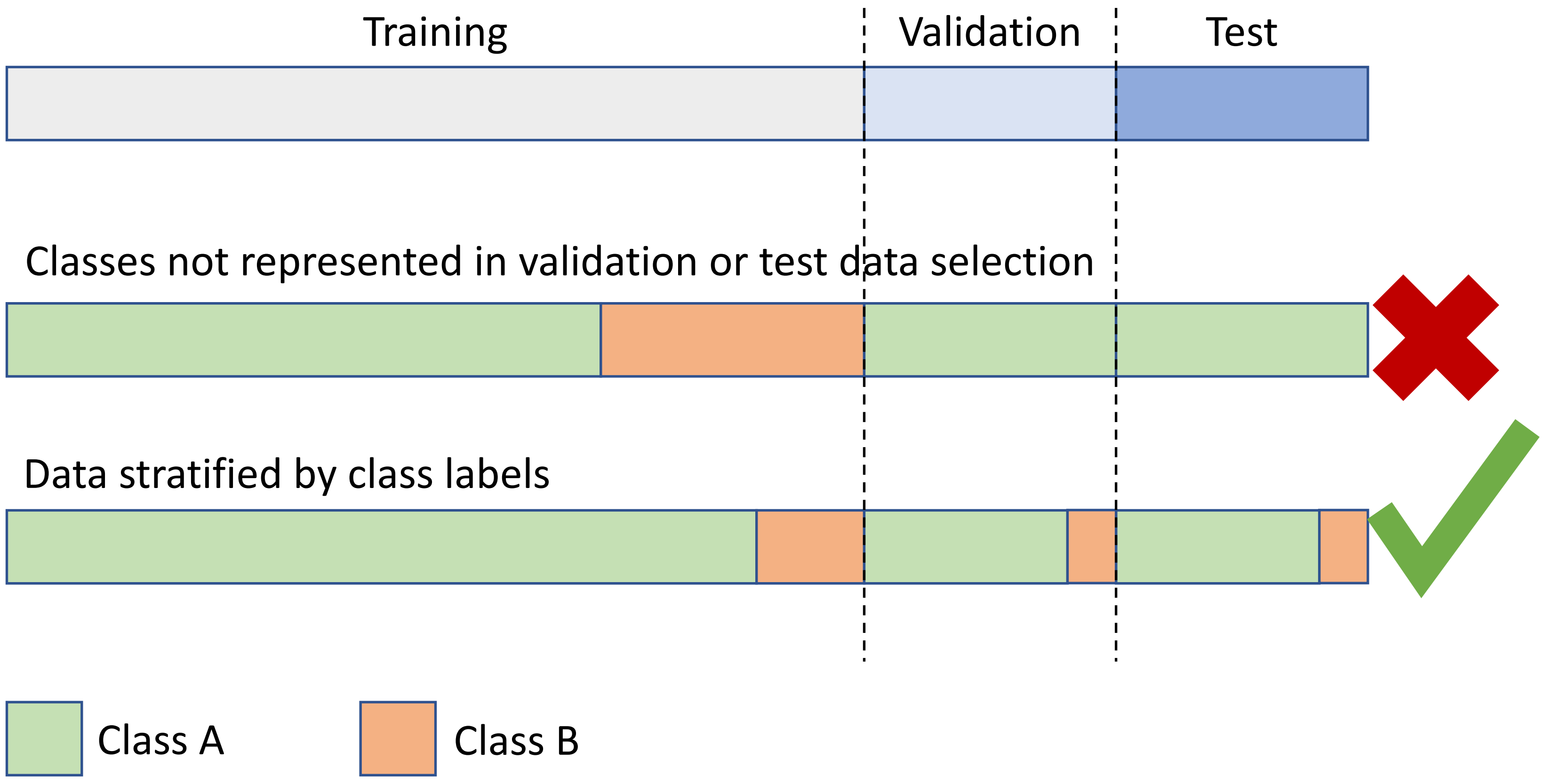
K-Fold Cross-Validation



Scikit Function for Applying K-Fold

```
import numpy as np
from sklearn.model_selection import KFold

X = np.array([[0, 11],
              [1, 22],
              [2, 33],
              [3, 44]])

kf = KFold(n_splits=4, shuffle=True)

for X_train, X_test in kf.split(X):
    #Train the model (i.e., model.fit(X[X_train])
    #Test the model (i.e., model.predict(X[X_test])
    #Try print(X[X_Train]) and print(X[X_test])!
```

Stratifying Class Labels



Stratified K-Fold Function in Scikit

```
import numpy as np
from sklearn.model_selection import StratifiedKFold

X = np.array([[0, 11],
              [1, 22],
              [2, 33],
              [3, 44]])
y = np.array([0, 0, 1, 1])

skf = StratifiedKFold(n_splits=2, shuffle=True)

for X_train, X_test in skf.split(X,y):
    #Train the model (i.e., model.fit(X[X_train])
    #Test the model (i.e., model.predict(X[X_test])
    #Try print(X[X_Train]) and print(X[X_test])!
```

Train Test Split Function

```
import numpy as np
from sklearn.model_selection import train_test_split

X = np.array([[0, 11],
              [1, 22],
              [2, 33],
              [3, 44]])
y = np.array([0, 0, 1, 1])

X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.5, stratify=y)

print(X_train)
```

Parameter Tuning

Machine learning models have parameters which can change classification results.

Example: K-Nearest Neighbor (sklearn.neighbors.KNeighborsClassifier)

- **n_neighbors:** The number of neighbors used by the classifier to make a decision.
- **weights:** Parameter for applying weights to the nearest neighbors.
- **p:** The power parameter for the distance metric.

Grid Search

Number of Neighbors

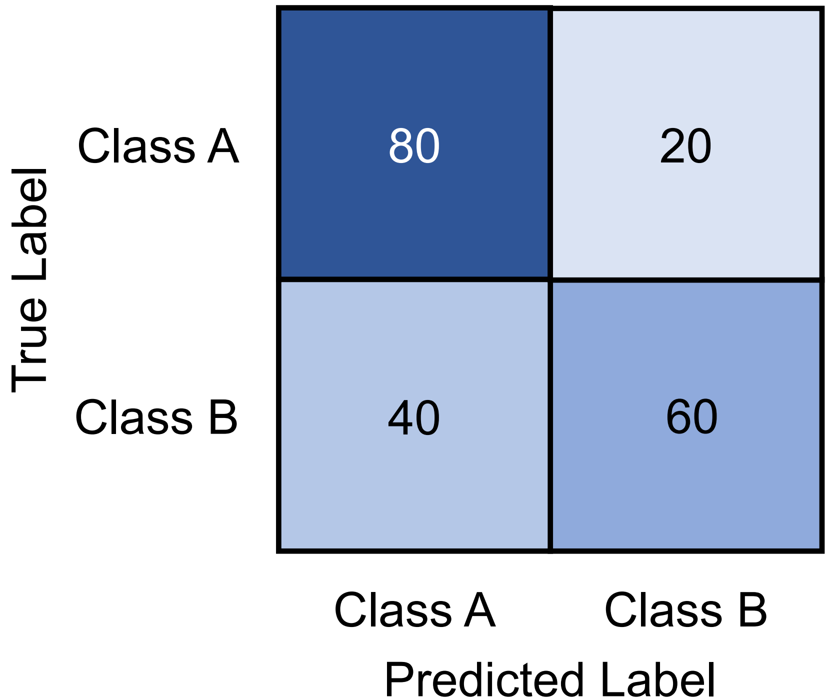
p	3	5	7	11	13	15	17
1							
2							
3							
4							
5							

Basic Idea: Run all model parameter combinations (i.e., brute force) and choose the best one.

Grid Search with Scikit

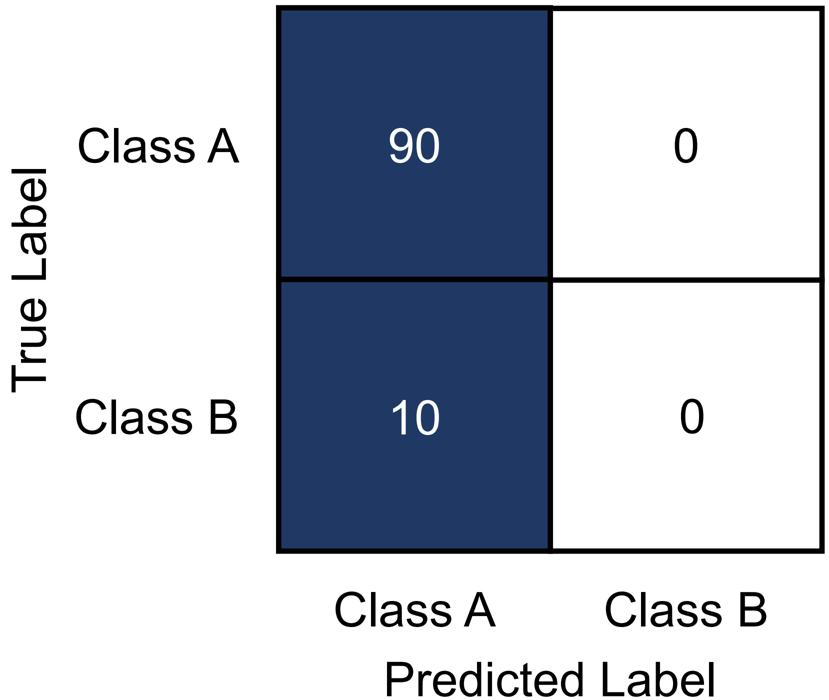
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

parameters = {'p':[1,2,3,4,5],
              'n_neighbors':[3,5,7,11,13,15,17]}

knn = KNeighborsClassifier()

gsc = GridSearchCV(knn, parameters)
gsc.fit(gs_data, gs_target)

gsc.cv_results_['mean_test_score']
gsc.cv_results_['params']
```

Confusion Matrix



Accuracy = 80+60/(80+20+40+60) = 140/200 = 0.70 = **70%**

Accuracy Can Be Misleading



Accuracy = 90+0/(90+0+10+0) = 90/100 = 0.90 = **90%**

Other Metrics

True Label		Predicted Label: Class A	Predicted Label: Class B
	Class A	True Positive (TP)	False Negative (FN)
	Class B	False Positive (FP)	True Negative (TN)

Accuracy (ACC): $\frac{TP+TN}{TP+FP+TN+FN}$

True Positive Rate (TPR): $\frac{TP}{TP+FN}$

False Positive Rate (FPR): $\frac{FP}{FP+TP}$

Precision (PPV): $\frac{TP}{TP+FP}$

Recall: $\frac{TP}{TP+FN}$

F1 Score: $\frac{PPV*TPR}{PPV+TPR} = \frac{2TP}{2TP+FP+FN}$

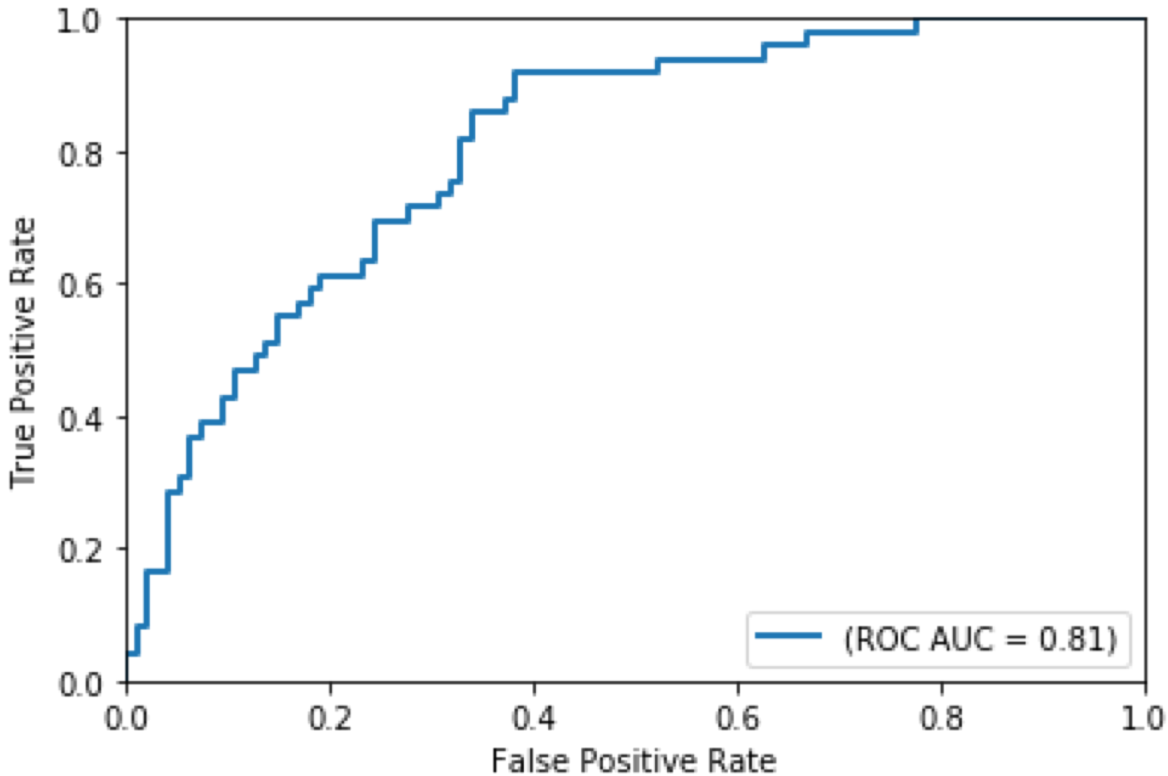
Matthew's Correlation Coefficient:

$$\frac{TP * TN - \mathrm{FP} * \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$

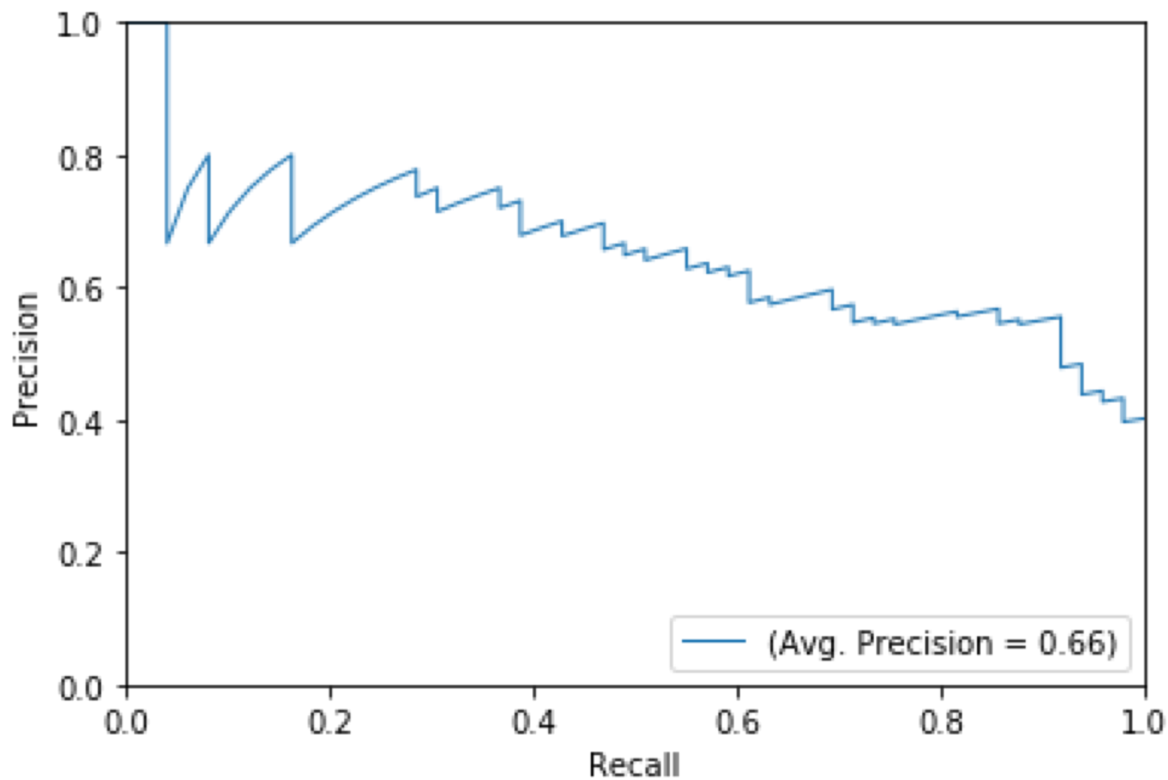
Metrics with Scikit

```
y_pred = [0, 1, 0, 1]
y_true = [0, 1, 0, 0]

from sklearn.metrics import accuracy_score
acc = accuracy_score(y_true, y_pred)
acc

from sklearn.metrics import f1_score
f1 = f1_score(y_true, y_pred)
f1

from sklearn.metrics import matthews_corrcoef
mcc = matthews_corrcoef(y_true, y_pred)
mcc
```

Receiver Operating Characteristic (ROC) Curve



Precision Recall (PRC) Curve



Exercise

Using the mouse protein dataset (MiceProtein_2f2c.csv):

1. Split the data into training and testing sets, stratifying by class labels
2. Run grid search cross validation on the training set using KNN classifier for *n_neighbors* and *p*
3. Plot ROC and PRC curves using a KNN classifier with the best parameters on the test dataset

Hint: *Use one KNN classifier to fit a grid search model to find the best parameters, then create a new KNN classifier setting the best parameters observed, fit the training data, and predict the probabilities from the test data*