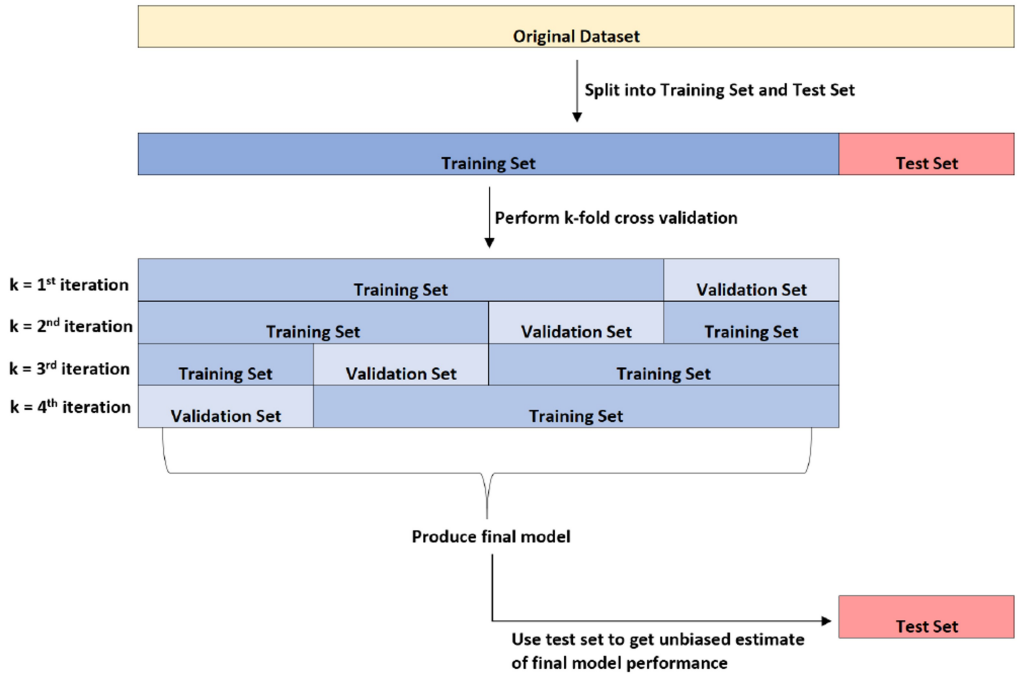
Lecture 9

Dataset Partitioning

```
clear all;
close all;

rng(1);
N = 15;
X1 = -3+randn(1,N);
C1 = ones(1,N);  % labels

X2 = 3+randn(1,N);
C2 = 2*ones(1,N);

X = [X1 X2];
C = [C1 C2];

cv = cvpartition(C, 'HoldOut', 0.2)

cv.test+cv.training % all 1 vector

final_test_label = C(cv.test)
final_test_data = X(cv.test)

>> % Remaining data samples and
corresonding labels used for training and
% validation (TV)
X_TV = X(cv.training);
C_TV = C(cv.training);

>> cv_TV = cvpartition(C_TV, 'KFold', 4)

>> % For training (1st fold, etc.)
X_V_1 = X_TV(cv_TV.training(1));
C_V_1 = C_TV(cv_TV.training(1));

>> % For validation
X_T_1 = X_TV(cv_TV.test(1));
C_T_1 = C_TV(cv_TV.test(1));
```

```
>> final_test_label = C(cv.test)
final_test_data = X(cv.test)

final_test_label =

    1     1     1     2     2     2


final_test_data =

  -1.8188   -3.8456   -4.5094    1.7299    1.1349    2.7075


cv_TV =

K-fold cross validation partition
  NumObservations: 24
     NumTestSets: 4
       TrainSize: 18  18  18  18
        TestSize: 6  6  6  6

>> X_T_1

X_T_1 =

  -3.7585   -3.5587   -3.1969    4.1752    2.7248    2.5826

>> C_T_1

C_T_1 =

    1     1     1     2     2     2
```




```
>> clear all;
N = 2;
X1 = -3+randn(1,N);
C1 = ones(1,N);  % labels

X2 = 3+randn(1,N);
C2 = 2*ones(1,N);

X = [X1 X2];
C = [C1 C2];
>> X

X =

  -3.3490   -2.8358    1.3634    3.5814
```

```
>> cv = cvpartition(C, 'LeaveOut')

cv =

Leave-one-out cross validation partition
   NumObservations: 4
      NumTestSets: 4
        TrainSize: 3  3  3  3
         TestSize: 1  1  1  1
>> [cv.test(1) cv.test(2) cv.test(3) cv.test(4)]

ans =

  4×4 logical array

   0   0   0   1
   0   0   1   0
   1   0   0   0
   0   1   0   0
```

```
>> cv =
cvpartition(C,'resubstitution')

cv =

Resubstitution (no partition of
data)
   NumObservations: 4
      NumTestSets: 1
        TrainSize: 4
         TestSize: 4

>> cv.test'

ans =

  1×4 logical array

   1   1   1   1
```

sklearn.model_selection.StratifiedKFold

class sklearn.model_selection.**StratifiedKFold**(*n_splits=5*, ***, *shuffle=False*, *random_state=None*)[source]
Stratified K-Folds cross-validator.
Provides train/test indices to split data in train/test sets.
This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.

>

cross_val_score.py

sklearn.model_selection.cross_val_score

sklearn.model_selection.**cross_val_score**(*estimator*, *X*, *y=None*, ***, *groups=None*, *scoring=None*, *cv=None*, *n_jobs=None*, *verbose=0*, *fit_params=None*, *pre_dispatch='2*n_jobs'*, *error_score=nan*)[source]
Evaluate a score by cross-validation.

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()

from sklearn.datasets import load_iris
train_samples = load_iris()
X = train_samples.data
Y = train_samples.target

from sklearn.model_selection import StratifiedKFold
KFold1 = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)

from sklearn.model_selection import KFold
KFold2 = KFold(n_splits=10, random_state=1, shuffle=True)

from sklearn.model_selection import cross_val_score
```

```
scores1 = cross_val_score(clf, X, Y, cv = KFold1, scoring = 'accuracy')
import numpy as np
print(np.mean(scores1))

print(np.mean(scores1))
0.9600000000000002

scores2 = cross_val_score(clf, X, Y, cv = KFold2, scoring = 'accuracy')
print(np.mean(scores2))

print(np.mean(scores2))
0.9533333333333334
```

```
% kfoldloss_demo.m

load fisheriris


rng(1);     % For reproducibility
cv = cvpartition(species,'KFold', 5)

>> Model_cv = fitcnb(meas, species, 'CVPartition', cv);

% Equivalently
% Model_cv = fitcnb(meas, species,'CrossVal','on','KFold',5);


>> error_rate_1 = kfoldLoss(Model_cv,'LossFun','ClassifErr')

error_rate_1 =

   0.0533
```

```
>> K = 5;
rng(1);     % For reproducibility
cv_part = cvpartition(species,'KFold',K);

Errors = zeros(1, K);

for i = 1: K
   training_index = cv_part.training(i);
   testing_index = cv_part.test(i);

   Model = fitcnb(meas(training_index,:),
species(training_index,:));
   % Then predict using the testing data
   label = predict(Model, meas(testing_index,:));
   diff = strcmp (label, species(testing_index,:));
   Errors(i) = length(find(diff==0))/length(label);
end
>> Errors

Errors =

   0.0333    0.1000    0.0333    0.0667    0.0333

>> error_rate_2 = mean(Errors)

error_rate_2 =

   0.0533
```

Metrics to Evaluate the
Quality of Prediction

- TP, TN, FP, FN
- Sensitivity (Recall)
- Specificity
- Precision
- Accuracy
- F1 score
- Confusion Matrix
- Receiver Operator Characteristic (ROC) curve

```
>> Mdl = fitcnb(meas, species);
>> label = predict(Mdl, meas);
>> Pred_Class = resubPredict(Mdl);
>> isequal(label, Pred_Class)

ans =

  logical

   1

>> confusionchart(species, Pred_Class)
```