Lecture 3. Core ML concepts (part 1): Model evaluation using Scikit-Learn

COMP 551 Applied machine learning

Yue Li
Assistant Professor
School of Computer Science
McGill University

September 8, 2022

Outline

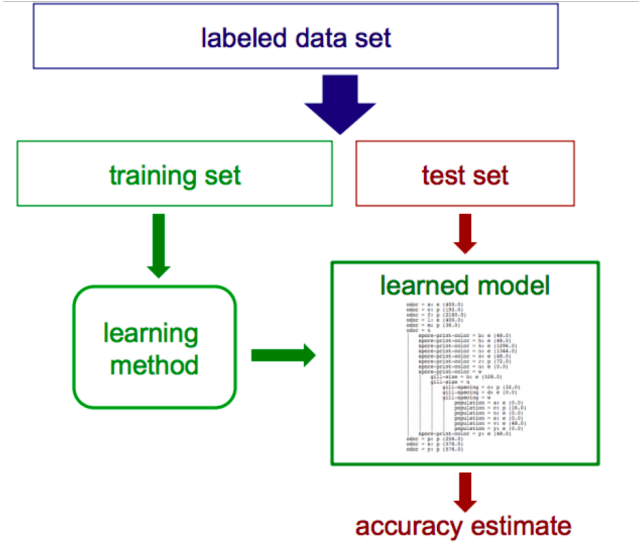
Learning objectives

Understanding the following concepts

- Model evaluation
- Model selection
- Receiver operating characteristics curve
- Cross-validation
- Method comparison

Outline

A typical workflow to evaluate a classification model



Generalization error (or generalization accuracy)

- What we really care about is the performance of the model on *new data*.
- In other words, we want to see how our model *generalizes* to unseen data.
- An assumption that justifies deployment of our model on unseen data is the fact that our training and unseen data come from the **same** distribution.
- In fact very often we assume that there exists some distribution $p(x, y)$ over our features and labels, such that our training data is composed of independent samples from this distribution – that is $x^{(n)}, y^{(n)} \sim p(x, y)$ for all $x^{(n)}, y^{(n)} \in \mathcal{D}$.
- We assume that unseen data are also samples from *the same* distribution.

Generalization error is the **expected error** of our model $f : x \mapsto y$ under this distribution:

$$Err(f) = \mathbb{E}_{x,y\sim p}[\ell(f(x), y)].$$

Here ℓ is some *loss function* such as classification error $\ell(y, \hat{y}) = \mathbb{I}(y \neq \hat{y})$ or square loss $\ell(y, \hat{y}) = (y - \hat{y})^2$ that we often use in regression.

Test set

- Unfortunately, we don't have access to the true data distribution, we only have samples from this dataset.
- We can estimate the generalization error by setting aside a portion of our dataset that **we do not use in any way in learning or selecting the model**.
- This part of the dataset is called the **test set**. Let's use $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ to this partitioning of our original dataset $\mathcal{D}$.

The **test error** is

$$\widehat{Err}(f) = \mathbb{E}_{x,y\sim\mathcal{D}_{\text{test}}}[\ell(f(x),y)] = \frac{1}{|\mathcal{D}_{\text{test}}|}\sum_{x,y\in\mathcal{D}_{\text{test}}}\ell(f(x),y).$$

Prostate cancer prediction problem

Suppose you want to learn to predict if a person has a prostate cancer based on two easily-measured variables obtained from blood sample: Complete Blood Count (CBC) and Prostate-specific antigen (PSA). We have collected data from patients known to have or not have prostate cancer:

CBC	PSA	Status
142	67	Normal
132	58	Normal
178	69	Cancer
188	46	Normal
183	68	Cancer
...		

Goal: Train classifier to predict the class of new patients, from their CBC and PSA.

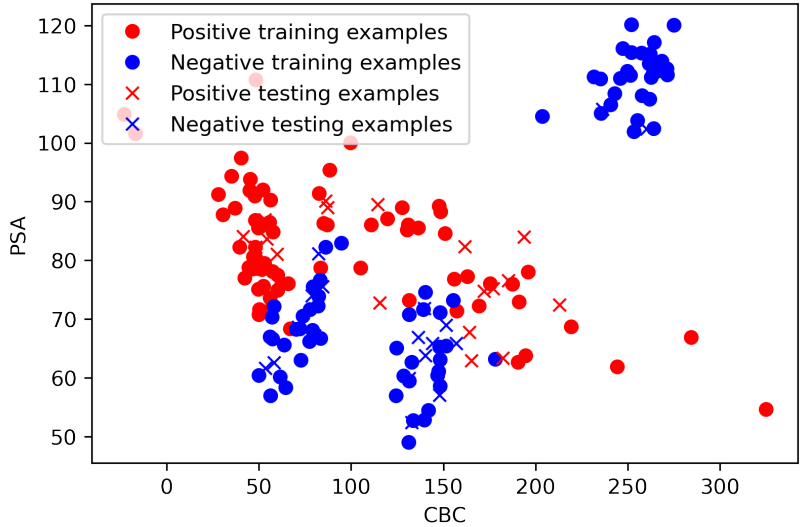
Split the dataset into training and testing datasets

We split the data into 80% training and 20% testing. In Python, this is easy to do:

```
1 from sklearn import model_selection
2
3 X_train, X_test, y_train, y_test =
  ↪  model_selection.train_test_split(X, y, test_size = 0.2,
  ↪  random_state=1, shuffle=True)
```

- `random_state` set to a fixed number for reproducibility
- `shuffle` by default is `True` to randomly permute the orders of the rows to avoid splitting examples of the same class into training or testing set. For example, if rows are ordered by classes.

Prostate cancer data



Model prediction

Model training:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier() # n_neighbors=5 (default)
fit = knn.fit(X_train, y_train)
```

Model prediction:

```
y_train_pred = fit.predict(X_train)
y_test_pred = fit.predict(X_test)
```

- We then apply the trained model `fit` to predict survivor: `y_train_pred=fit.predict(X_train)`, `y_test_pred=fit.predict(X_test)`
- Our prediction is binary 0 (healthy) or 1 (cancer) based on whether the predicted probabilities are greater than 0.5.

Classification Accuracy

```
acc_train = np.sum(y_train_pred==y_train)/len(y_train)
acc_test = np.sum(y_test_pred==y_test)/len(y_test)
```

- We then evaluate the prediction accuracy:

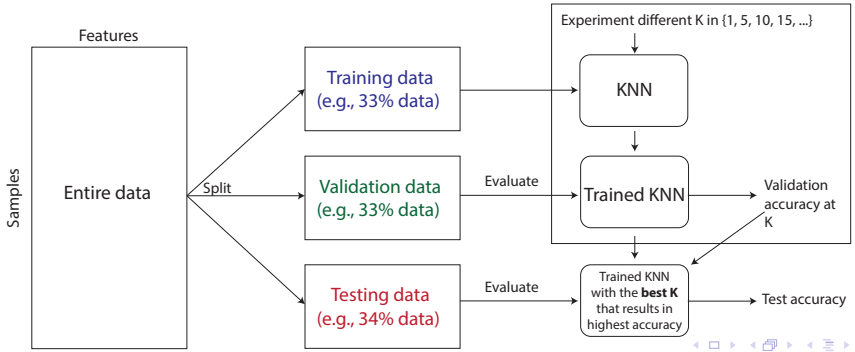
$$Accuracy = \frac{\text{Correctly classified passengers}}{\text{Total number of classified passengers}}$$

- The accuracy for predicting cancer in the training dataset (93.75%) is a bit lower than the accuracy in predicting cancer in the testing dataset (97.5%).
- In practice, the accuracy for the training tends to be higher than the accuracy on the testing
- When the training accuracy is much higher than the testing accuracy, the model is *overfitting* the data (more to this in Lec 4)

Outline

Choosing K using a validation set (Recall from Lec2)

- The goal is to accurate predict unseen data that are not in the training set
- The accuracy can be approximated by the accuracy on the **test set**.
- To this end, we split the entire data into training, validation, and testing data.
- We use training to train the model, validation data to choose the hyperparameter from a finite set of them (i.e., $K \in \{1, \ldots, 10\}$) that result in the highest validation accuracy, and finally we evaluate the chosen model on the test set.



Classification Accuracy

We can split the data twice:

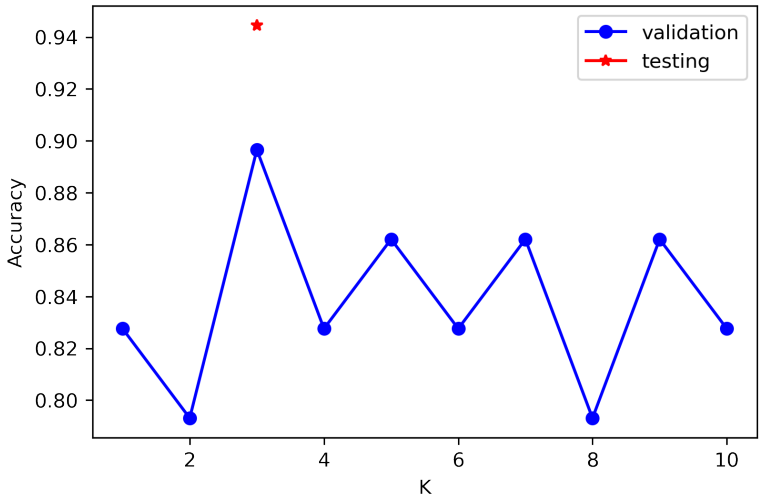
- first split the *entire* data into training and testing set
- then split the *training* data into training and validation set

```
1 X_train, X_test, y_train, y_test = train_test_split(X, Y,
  ↪  test_size=0.2, random_state=11, shuffle=True)
2 X_train, X_valid, y_train, y_valid = train_test_split(X_train,
  ↪  y_train, test_size=0.2, random_state=11, shuffle=True)
```

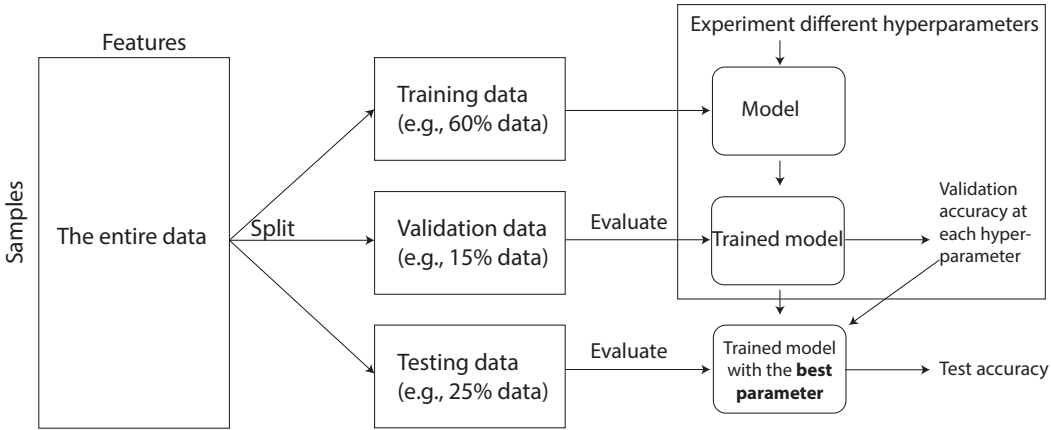
Using validation set to choose $K \in \{1, 10\}$

```
model_choices=[]
valid_acc = []

for k in range(1,11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_valid_pred = knn.predict(X_valid)
    accuracy = np.sum(y_valid_pred == y_valid)/y_valid.shape[0]
    model_choices.append(k)
    valid_acc.append(accuracy)

# use the best K to predict test data
best_valid_K = valid_acc.index(max(valid_acc)) + 1
knn = KNeighborsClassifier(n_neighbors=best_valid_K)
knn.fit(X_train, y_train)
y_test_pred = knn.predict(X_test)
test_accuracy = np.sum(y_test_pred == y_test)/y_test.shape[0]
print(test_accuracy)
```

KNN's prediction accuracy in validation set of a function of K



The training-validation-testing is the most general ML experiment design



Outline

True/false positives and negatives

True positive (TP)
Positive example that is predicted to be positive

- A person who is predicted to have cancer and in fact has cancer

False positive (FP)
Negative example that is predicted to be positive

- A person who is predicted to have cancer but in fact is healthy

True negative (TN)
Negative example that is predicted to be negative

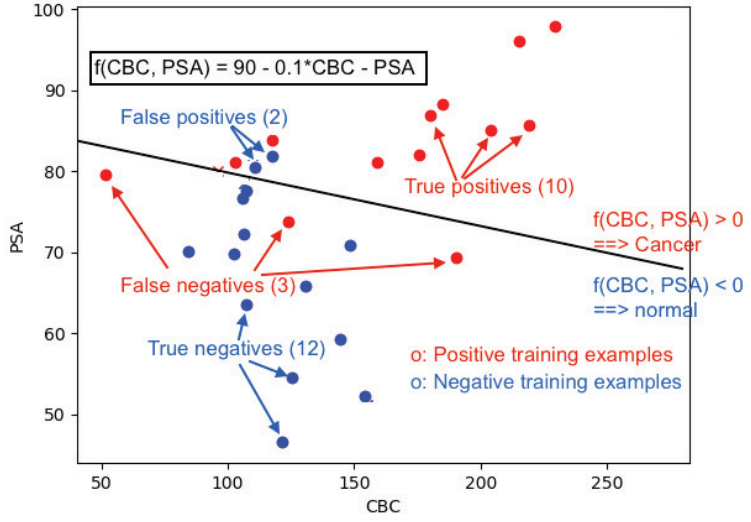
- A person who is predicted to be healthy and in fact is healthy

False negative (FN)
Positive example that is predicted to be negative

- A person who is predicted to be healthy but in fact has cancer

Classification errors

Here: TP = 10, TN = 12, FP = 2, FN = 3.



Confusion matrix

Confusion matrix: A table of counts for TPs, FPs, TNs, and FNs

	Predicted negative	Predicted positive
Actual negative	TN = 12	FP = 2
Actual positive	FN = 3	TP = 10

In scikit-learn, we can get the confusion matrix for the KNN by:

```
from sklearn.metrics import confusion_matrix

X_train, X_test, y_train, y_test = \
train_test_split(X, y, test_size=0.5, random_state=100)

knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_test_pred = knn.predict(X_test)

cm = confusion_matrix(y_test, y_test_pred)
```

Sep 13 lecture resumes from Sep 8 lecture

True and false positive rates

At a specific threshold, we can calculate TPR and FPR:

True positive rate (TPR) (aka sensitivity)

The proportion of positive examples that are predicted positive

- Fraction of cancer who are predicted to have cancer

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

False positive rate (FPR) (aka 1 - specificity)

The proportion of negative examples that are predicted to be positive

- Fraction of non-cancer who are predicted to have cancer

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

True/false positive rates

	Predicted negative	Predicted positive
Actual negative	TN = 12	FP = 2
Actual positive	FN = 3	TP = 10

True-positive rate (TPR): Fraction of cancer patients who are predicted as cancer

$$Sensitivity = \frac{TP}{TP + FN} = \frac{10}{10 + 3} = 77\%$$

Specificity: Proportion of negative examples that are predicted to be negative

- Fraction of healthy patients who are predicted to be healthy

$$Specificity = \frac{TN}{FP + TN} = \frac{12}{2 + 12} = 86\%$$

False-positive rate (FPR): Proportion of negative examples that are predicted to be positive

- Fraction of healthy patients who are predicted to have cancer

$$FPR = \frac{FP}{FP + TN} = 1 - specificity = \frac{2}{2 + 12} = 14\%$$

True/false positive rates (pop quiz)

	predicted negative	predicted positive
actual negative	1	2
actual positive	3	7

True positive rate (TPR) (or sensitivity)

The proportion of positive examples that are predicted positive

- Fraction of true cancer who are predicted to be cancer

$$TPR = \frac{TP}{TP + FN} = \frac{?}{? + ?} = ?$$

False positive rate (FPR)

The proportion of negative examples that are predicted to be positive

- Fraction of non-cancer who are predicted to be cancer

$$FPR = \frac{FP}{FP + TN} = \frac{?}{? + ?} = ?$$

A classification model often produces probabilities instead of hard decision

Recall KNN from Lecture 2. When $K > 1$, we set our predicted class to be the class label c supported by the majority of the K neighbours for a new data point $\mathbf{x}^{(*)}$:

$$y^* = \arg\max_c \sum_{n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})} \mathbb{I}(y^{(n)} = c) \tag{1}$$

Alternatively, we can calculate *the class probabilities* of each class:

$$p(y^{(*)} = c | \mathbf{x}^{(*)}) = \frac{1}{K} \sum_{n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})} \mathbb{I}(y^{(n)} = c) \tag{2}$$

In binary classification (i.e., cancer or normal), we can reduce the formula to predicting only the positive class:

$$p(y^{(*)} = 1 | \mathbf{x}^{(*)}) = \frac{1}{K} \sum_{n \in \mathcal{N}_K(\mathbf{x}^{(*)}, \mathcal{D})} \mathbb{I}(y^{(n)} = 1) \tag{3}$$

If we want to set the class to be 0 or 1 as in Eq. (1), what rule should we use?

Classification threshold: how to decide who are cancer patients?

patient index	$p(y = 1\|x)$	true_label
0	0.1	0
1	0.4	0
2	0.5	1
3	0.8	1

- By default, `fit.predict(X_test)` uses 0.5 as threshold. What does this imply in context of KNN algorithm?

```
1         if pred_prob > 0.5:
2             cancer = 1
3         else:
4             normal = 0
```

- What accuracy do we get with a different threshold say 0.6?
- Can we evaluate the model without setting an arbitrary threshold?

True/false positive rates (pop quiz)

Suppose there are 2 cancer and 2 non-cancer. What will be the TPR and FPR if **the threshold is 1**?

patient index	$p(y=1\|x)$	pred_label	true_label
0	0.1		0
1	0.4		0
2	0.5		1
3	0.8		1

	PN	PP
AN		
AP		

True positive rate (TPR):

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

False positive rate (FPR):

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

True/false positive rates (pop quiz)

What will be the TPR and FPR if **the threshold is 0.6**?

patient index	$p(y = 1\|x)$	pred_label	true_label
0	0.1		0
1	0.4		0
2	0.5		1
3	0.8		1

	PN	PP
AN		
AP		

True positive rate (TPR):

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

False positive rate (FPR):

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

True/false positive rates (pop quiz)

What will be the TPR and FPR if **the threshold is 0.3**?

patient index	$p(y = 1\|x)$	pred_label	true_label
0	0.1		0
1	0.4		0
2	0.5		1
3	0.8		1

	PN	PP
AN		
AP		

True positive rate (TPR):

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

False positive rate (FPR):

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

True/false positive rates (pop quiz)

What will be the TPR and FPR if **the threshold is -1**?

patient index	$p(y = 1\|x)$	pred_label	true_label
0	0.1		0
1	0.4		0
2	0.5		1
3	0.8		1

	PN	PP
AN		
AP		

True positive rate (TPR):

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

False positive rate (FPR):

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

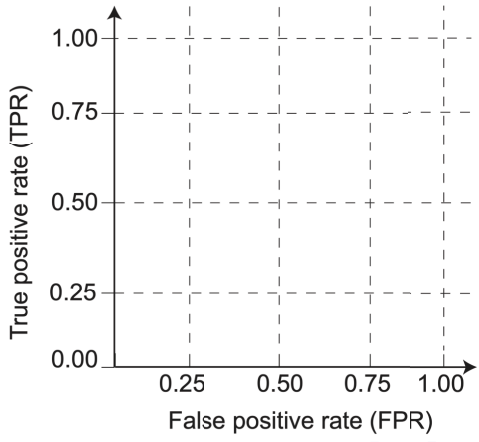
Receiver Operating Characteristic (ROC) curve

- We can create a table for TPR and FPR at each Threshold.
- ROC curve plots TPR (y-axis) versus FPR (x-axis)
- *Area under the curve (AUC)* is a metric common used to evaluate the model.

Threshold	TPR	FPR
1		
0.6		
0.3		
0		

Consider three extreme cases:

1. What does ROC look like for a dummy model that predicts everything 0.5?
2. What does ROC look like for a perfect model?
3. What does ROC look like for a model opposite to perfect?



Dummy model for thresholds -1, 0.6, 0.3, 1

patient index	$p(y=1\|x)$	pred_label				true_label
0	0					0
1	0					0
2	0					1
3	0					1

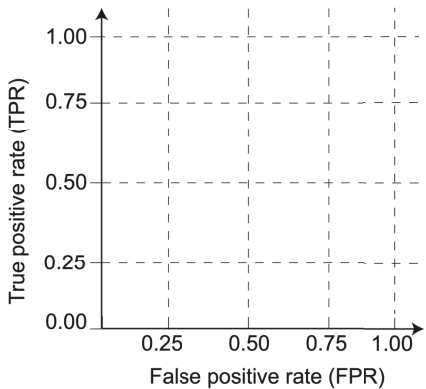
	PN	PP
AN		
AP		

True positive rate (TPR):

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

False positive rate (FPR):

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



Perfect model for thresholds -1, 0.6, 0.3, 1

patient index	$p(y=1\|x)$	pred_label	true_label
0	0		0
1	0		0
2	1		1
3	1		1

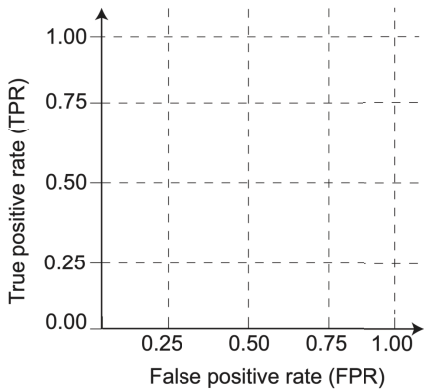
	PN	PP
AN		
AP		

True positive rate (TPR):

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

False positive rate (FPR):

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



Opposite to perfect model for thresholds -1, 0.6, 0.3, 1

patient index	$p(y=1\|x)$	pred_label				true_label
0	1					0
1	1					0
2	0					1
3	0					1

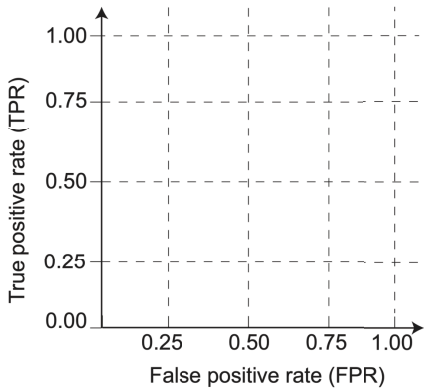
	PN	PP
AN		
AP		

True positive rate (TPR):

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

False positive rate (FPR):

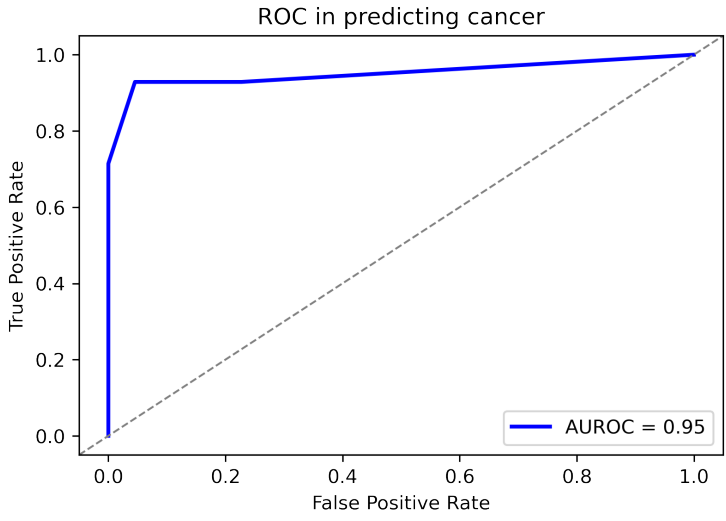
$$FPR = \frac{FP}{FP + TN} =$$



Computing ROC and AUROC using Scikit-learn function

```
from sklearn.metrics import roc_curve, roc_auc_score

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_test_prob = knn.predict_proba(X_test)[:,1] # column 0 is healthy,
↪  column 1 is cancer
fpr, tpr, thresholds = roc_curve(y_test, y_test_prob)
roc_auc = roc_auc_score(y_test, y_test_prob)
plt.clf()
plt.plot(fpr, tpr, "b-", lw=2, label="AUROC = %0.2f"%roc_auc)
plt.axline((0, 0), (1, 1), linestyle="--", lw=1, color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC in predicting cancer')
plt.legend(loc="best")
```

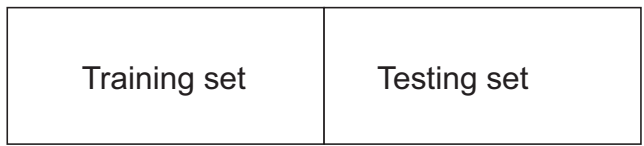
The resulting ROC



Outline

K-fold Cross Validation

- In our above example, we split the data into 50% training and 50% testing
- We train the model using only half of the data and evaluate the model using the other half



- This is quite wasteful. How can we evaluate our model on *every data point* while training on the rest of the data points?
- Answer: K-fold cross-validation

Five-fold cross validation

Step 1. Randomly split the data $\mathcal{D}$ into 5 folds

$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$

Step 2. Training and prediction

Fold 1	$\mathcal{F}_1$					Train on $\mathcal{D}$ - $\mathcal{F}_1$, predict on $\mathcal{F}_1$
Fold 2		$\mathcal{F}_2$				Train on $\mathcal{D}$ - $\mathcal{F}_2$, predict on $\mathcal{F}_2$
Fold 3			$\mathcal{F}_3$			Train on $\mathcal{D}$ - $\mathcal{F}_3$, predict on $\mathcal{F}_3$
Fold 4				$\mathcal{F}_4$		Train on $\mathcal{D}$ - $\mathcal{F}_4$, predict on $\mathcal{F}_4$
Fold 5					$\mathcal{F}_5$	Train on $\mathcal{D}$ - $\mathcal{F}_5$, predict on $\mathcal{F}_5$

- How many times each data point is trained?
- How many times each data point is predicted?

Evaluate on all K folds

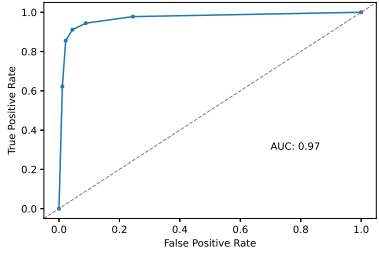
Step 3. Evaluate predictions on all 5 folds by ROC

Predicted probabilities	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$

versus

True labels	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$

ROC curve of KNN predicted on ALL data points



Cross validation in Python scikit-learn

```
def cross_validate(model, X_input, Y_output):
    kf = KFold(n_splits=5, random_state=1, shuffle=True)
    true_labels = np.array([0] * X_input.shape[0])
    pred_scores = np.array([0.0] * X_input.shape[0])
    for train_index, test_index in kf.split(X_input):
        model.fit(X_input[train_index], Y_output[train_index])
        true_labels[test_index] = Y_output[test_index]
        pred_scores[test_index] =
        ↪ model.predict_proba(X_input[test_index])[:,1]
    return true_labels, pred_scores
true_labels,pred_scores = cross_validate(model, X, y)
```

Outline

Method comparisons

- There are many machine learning methods implemented in scikit-learn
- How do we know which one performs the test on *our data set*?
- To get the answer, we will need to compare these methods using cross validation
- Let's compare three machine learning methods namely
  - K-nearest neighbours (KNN)
  - Decision tree classifier (DT) (Lec 4)
  - Logistic regression (LR) (Lec 6)
- Note: for each method (or class), we create an *object* of the method using their initializer method defined under that class
- Training and prediction follows the *generic* syntax

Method comparisons using scikit-learn

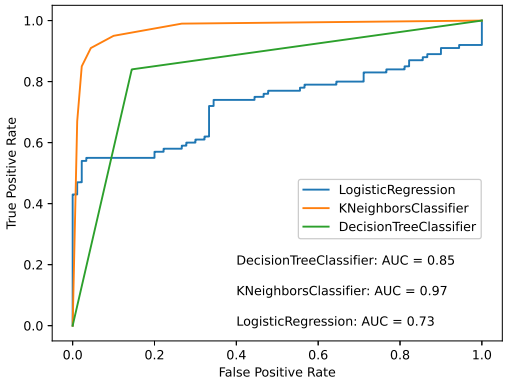
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

models = [LogisticRegression(),
          KNeighborsClassifier(),
          DecisionTreeClassifier()]

perf = {}

for model in models:
    model_name = type(model).__name__
    print(model_name)
    label,pred = cross_validate(model, X, y)
    fpr, tpr, thresholds = roc_curve(label, pred)
    auc = roc_auc_score(label, pred)
    perf[model_name] = {'fpr':fpr,'tpr':tpr,'auc':auc}
```

Plot the ROC curves for all method in one plot

```
import matplotlib.pyplot as plt

i = 0
for model_name, model_perf in perf.items():
    plt.plot(model_perf['fpr'], model_perf['tpr'],
             label=model_name)
    plt.text(0.4, i, model_name + ': AUC = '+
             str(round(model_perf['auc'],2)))
    i += 0.1

plt.legend(loc='upper center',
           bbox_to_anchor=(0.75, 0.5))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")

plt.savefig('roc_multimethods.eps')
```

ROC curves and AUC for all of the four methods

- KNN (K=5) performs the best with 0.97 AUC
- DT achieves 0.85 AUC
- LR did worse (AUC = 0.73) because our data are not linearly separable
- In contrast, DT and KNN are non-linear methods



Summary

- Common model training and testing to approximate generalization performance
- Use validation set to select hyperparameter (but not training the model)
- ROC is an effective way to test overall model performance without a threshold
- Cross-validation makes use of the full data for both training and evaluation
- Generic model implementation in Scikit-learn enables efficient method comparison