Evaluating Classification Methods

Evaluating and Comparing Classification Methods

- Predictive accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

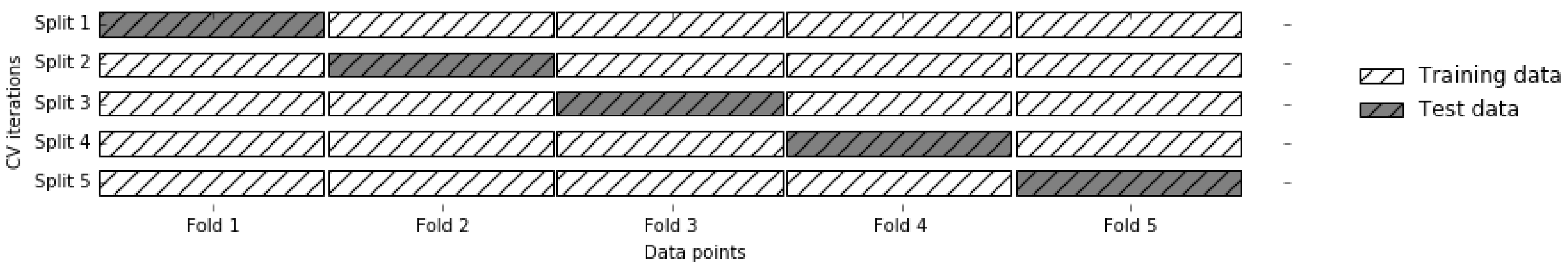
- Efficiency
- Robustness
- Scalability
- Interpretability
- Compactness of the model

Methods for Evaluating a Classifier's Accuracy

- **Holdout Method**
  - The data set is randomly partitioned into two disjoint sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - The test set is also called the holdout set
  - This method is mainly used when the data set *D* is large
- **Random Sampling**
  - a variation of holdout
  - repeat holdout k times
  - accuracy = avg. of the accuracies obtained

Evaluation Methods (cont)

- **K-Fold Cross Validation:** (k = 10 or 5 are popular)
  - Partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use Fold_i as test set and others as training set



Evaluation Methods (cont)

- **Leave One Out:** k folds where k = # of tuples, for small sized data
- **Stratified Cross Validation:**
  - For unbalanced data
  - Folds are stratified so that class distribution in each fold is approximately the same as that in the initial data

Scikit-learn Zone

- We have been using the holdout method

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.2, **stratify=cancer.target**, random_state=66)

Scikit-learn Zone – Cross Validation

- Cross-validation is implemented in model_selection.cross_val_score
- Parameters of cross_val_score include the model we want to evaluate, the data matrix, and the labels

Example– Using Cross Validation to Evaluate a DT on the Iris Dataset

- from sklearn.model_selection import cross_val_score
  from sklearn.tree import DecisionTreeClassifier
  from sklearn import datasets

- iris = datasets.load_iris()
  dt = DecisionTreeClassifier(random_state=0)

- scores = cross_val_score(dt, iris.data, iris.target)
  print("Cross-validation scores: ", scores)
  Cross-validation scores: [0.98039216   0.92156863   1.]

Example – Pipelines & Kfold

#StratifiedKFold

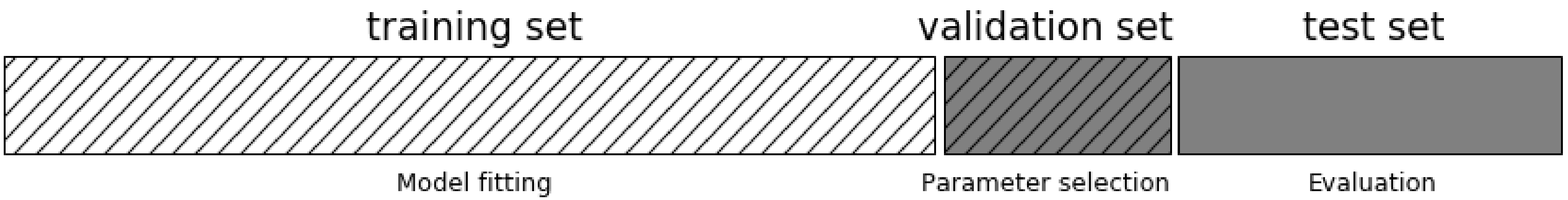
- from sklearn.**model_selection** import **KFold**, **cross_val_score**
- from sklearn.**pipeline** import **make_pipeline**
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.preprocessing import StandardScaler
- from sklearn import datasets; digits = datasets.load_digits()
- features = digits.data # Create features matrix
- target = digits.target # Create target vector
- standardizer = StandardScaler() # Create standardizer
- dt = DecisionTreeClassifier(random_state=0) # Create DT object
- **pipeline = make_pipeline(standardizer, dt) # Create a pipeline that standardizes, then runs dt**
- **kf = KFold(n_splits=10, shuffle=True, random_state=1) # Create k-Fold cross-validation**
- **# Conduct k-fold cross-validation**
- **cv_results = cross_val_score(pipeline, # Pipeline**
- **features, # Feature matrix**
- **target, # Target vector**
- **cv=kf, # Cross-validation technique**
- **scoring="accuracy", # Evaluation function**
- **n_jobs=-1) # Use all CPU scores**
- cv_results.**mean()** # Calculate mean
  #0.856

Example on cv = LeaveOneOut()

- from sklearn.model_selection import LeaveOneOut
- loo = LeaveOneOut()
- scores = cross_val_score(dt, iris.data, iris.target, cv=loo)
- print("Number of cv iterations: ", len(scores))
- print("Mean accuracy: ", scores.mean())
  #Number of cv iterations: 150
  #Mean accuracy: 0.95

Validation Set & Hyperparameters

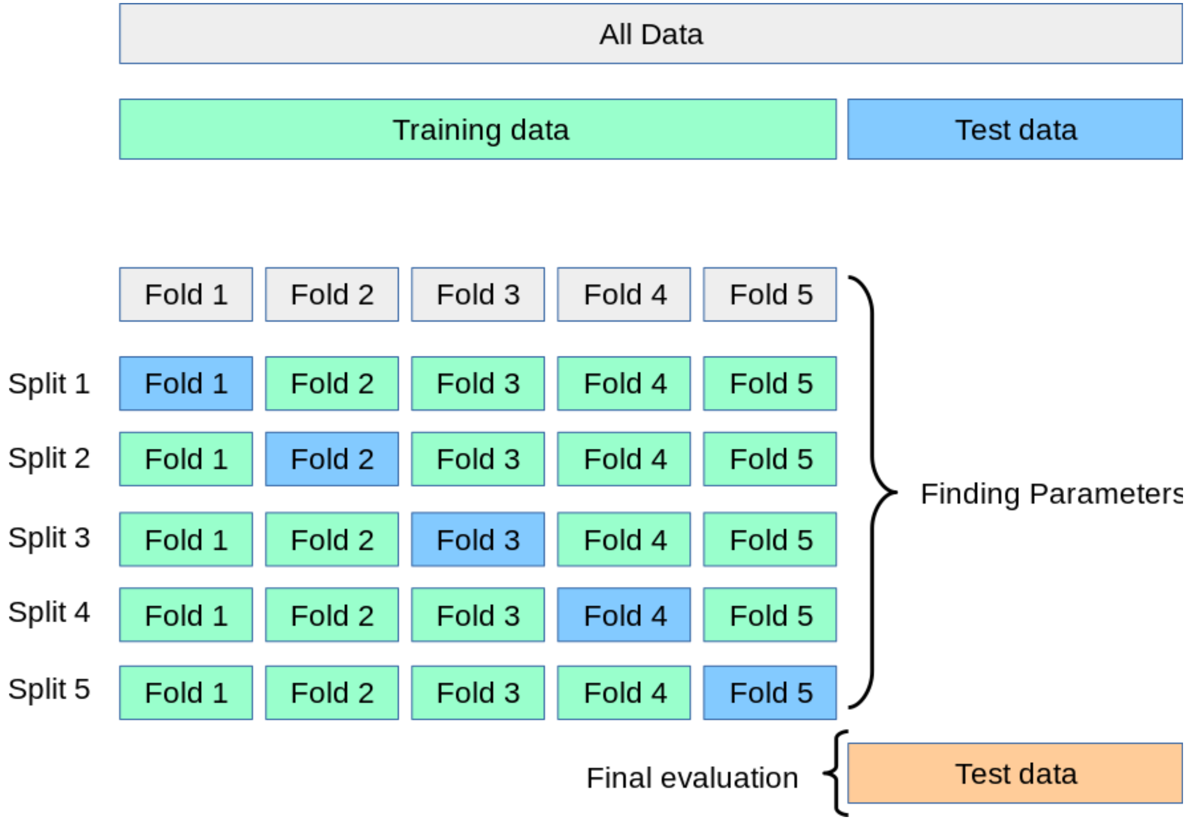
- A validation set is used frequently for estimating **hyperparameters** in learning algorithms
- **Validation set**: the available data is divided into three subsets,
  - a training set,
  - a validation set and
  - a test set



- The values that give the best accuracy on the validation set are used as the final hyperparameter values

Validation Set & Hyperparameters (cont)

- Cross-validation can be used for hyperparameter estimating



Source: https://scikit-learn.org/stable/modules/cross_validation.html

Classification Measures

- Accuracy is only one measure (error = 1 - accuracy)
- Accuracy is not suitable in some applications
- In classification involving skewed or highly imbalanced data, e.g.,
  - Network intrusion and financial fraud detections
  - Predicting documents of a particular topic
  - Predicting the presence of a rare disease such as cancer
  - We are interested only in the minority class
  - E.g., 1% intrusion. Achieve 99% accuracy by doing nothing
  - High accuracy does not mean any intrusion is detected
- The class of interest is commonly called the **positive class**, and the rest **negative classes**

Limitation with Accuracy

- Consider a 2-class problem
  - Number of –ve class examples = 9990
  - Number of +ve class examples = 10

- If model predicts everything to be –ve, accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect any +ve examples

Confusion Matrix

- Confusion Matrix:

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Confusion Matrix

- Used in information retrieval and text classification

	PREDICTED CLASS		
ACTUAL CLASS		Class=+ve	Class=–ve
	Class=+ve	a (TP)	b (FN)
	Class=–ve	c (FP)	d (TN)

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P’	**N’**	**All**

Entry, CM_{ij} gives the number of tuples of class i that were labeled by the classifier as class j

TP: the number of correct classifications of the positive examples (**true positive**),

FN: the number of incorrect classifications of positive examples (**false negative**),

FP: the number of incorrect classifications of negative examples (**false positive**), and

TN: the number of correct classifications of negative examples (**true negative**).

Accuracy

	PREDICTED CLASS	
ACTUAL CLASS	Class=Yes	Class=No
Class=Yes	a (TP)	b (FN)
Class=No	c (FP)	d (TN)

- Most widely-used metric:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Alternative Measures – Recall & Precision

	PREDICTED CLASS	
	Class=Yes	**Class=No**
ACTUAL CLASS: Class=Yes	a (TP)	b (FN)
ACTUAL CLASS: Class=No	c (FP)	d (TN)

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

TPR is fraction of positive samples that are correctly classified

$$\text{TPR} = \frac{TP}{P} = \frac{TP}{TP+FN} = \frac{a}{a+b} = \text{recall } (r) = \text{sensitivity}$$

Precision is the fraction of correct **predictions** of the positive class

$$\text{Precision (p)} = \frac{TP}{P'} = \frac{TP}{TP+FP} = \frac{a}{a+c} = \text{PPV}$$

Imbalanced Data

- The skew in the data is defined as $\alpha = \frac{P}{(P+N)}$
- Recall = TP/P, is not sensitive to the skew in the data
- $\text{Precision (p)} = \frac{TP}{P'} = \frac{TP}{TP+FP} = \frac{a}{a+c}$ is sensitive to the skew
- Precision is a useful measure for skewed data
- Recall, TP/(TP+**FN**), and precision, TP/(TP+**FP**), either capture the effect of FP or FN but not both
- If a classifier can optimize one of these measures, it can end up with a low FN but high FP, or vice-versa
  - A classifier that predicts every instance as positive will have perfect recall (as FN = 0), but poor precision
  - A classifier that is very conservative in classifying an instance as positive (thus FP = 0) will have high precision but low recall

F1- Score

- Aka F1 measure
- F1-score accounts for both FP and FN
- It is the harmonic mean of recall and precision

$$\text{F1}-\text{measure (F1)} = \frac{2}{\frac{1}{r} + \frac{1}{p}} = \frac{2rp}{r + p}$$

- The harmonic mean of two numbers tends to be closer to the smaller of the two
- For F1-score to be large, both *p* and *r* must be large

Example

	PREDICTED CLASS	
	Class=Yes	Class=No
ACTUAL CLASS Class=Yes	10	0
ACTUAL CLASS Class=No	10	980

$$\text{Precision (p)} = \frac{10}{10+10} = 0.5$$

$$\text{Recall (r)} = \frac{10}{10+0} = 1$$

$$\text{F1-measure (F1)} = \frac{2*1*0.5}{1+0.5}$$

$$= 0.67$$

$$\text{Accuracy} = \frac{990}{1000} = 0.99$$

	PREDICTED CLASS	
	Class=Yes	Class=No
ACTUAL CLASS Class=Yes	1	9
ACTUAL CLASS Class=No	0	990

$$\text{Precision (p)} = \frac{1}{1+0} = 1$$

$$\text{Recall (r)} = \frac{1}{1+9} = 0.1$$

$$\text{F1-measure (F1)} = \frac{2*0.1*1}{1+0.1}$$

$$= 0.18$$

$$\text{Accuracy} = \frac{991}{1000} = 0.991$$

Scikit-learn Zone – Classification Measures

- Set the scoring parameter for the cross_val_score function
- Possible values include accuracy| precision| recall |f1

```
bc = datasets.load_breast_cancer()
dt = DecisionTreeClassifier(random_state=0)
scores = cross_val_score(dt, bc.data, bc.target, cv = 5,
                         scoring ="recall")
print("Cross-validation scores: ", scores)
#Cross-validation scores:  [0.88888889 0.94444444 0.92957746
0.95774648 0.88732394]
print("average score:", scores.mean())
average score: 0.9215962441314554
```

Scikit-Learn Zone – Classification Measures (cont)

```
cross_val_score(dt, bc.data, bc.target, cv = 5,
                scoring ="precision")
```

```
# array([0.95522388, 0.93150685, 0.92957746,
0.95774648, 0.95454545])
```

```
cross_val_score(dt, bc.data, bc.target, cv = 5,
                scoring ="f1")
```

```
array([0.92086331, 0.93793103, 0.92957746,
0.95774648, 0.91970803])
```

A different Way to Get f1-score

- **from sklearn.metrics import** f1_score
- **print**("f1 score: ",
  f1_score(y_test, predicted_targets))

#f1 score: 0.9659090909090908

You could also use:
metrics.precision_score(y_true, y_pred)
metrics.recall_score(y_true, y_pred)
metrics.accuracy_score(y_true, y_pred)

Scikit-Learn – Confusion Matrix

- #Confusion Matrix for a DT classifier on a two-class dataset
- from sklearn import datasets

  bc = datasets.load_breast_cancer()

  features = bc.data

  target = bc.target

- from sklearn.model_selection import train_test_split

  X_train, X_test, y_train, y_test = train_test_split(
  features, target, random_state=0)

Scikit-Learn – Confusion Matrix (cont)

- from sklearn.tree import DecisionTreeClassifier

```
dt = DecisionTreeClassifier(max_depth = 2, random_state=0)
trained_dt = dt.fit(features, target)
predicted_targets = dt.predict(X_test)
predicted_targets[0:10] #array([0, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

- from sklearn.metrics import confusion_matrix

```
confusion = confusion_matrix(y_test, predicted_targets)
print(confusion)
  [[52  1]
   [ 5 85]]
```

A different Way to Get f1-score

- **from sklearn.metrics import** f1_score
- **print**("f1 score: ",
  f1_score(y_test, predicted_targets))

  #f1 score: 0.9659090909090908

- Likewise, we can call precision_score(), recall_score() and accuracy_score()

Receiver Operating Characteristics Curve

- It is commonly called the ROC curve
- Used to evaluate and compare binary classifiers
- It is a plot of the true positive rate (TPR) against the false positive rate (FPR).
- True positive rate:

$$TPR = \frac{TP}{TP + FN} = \frac{a}{a+b} = \frac{TP}{P}$$

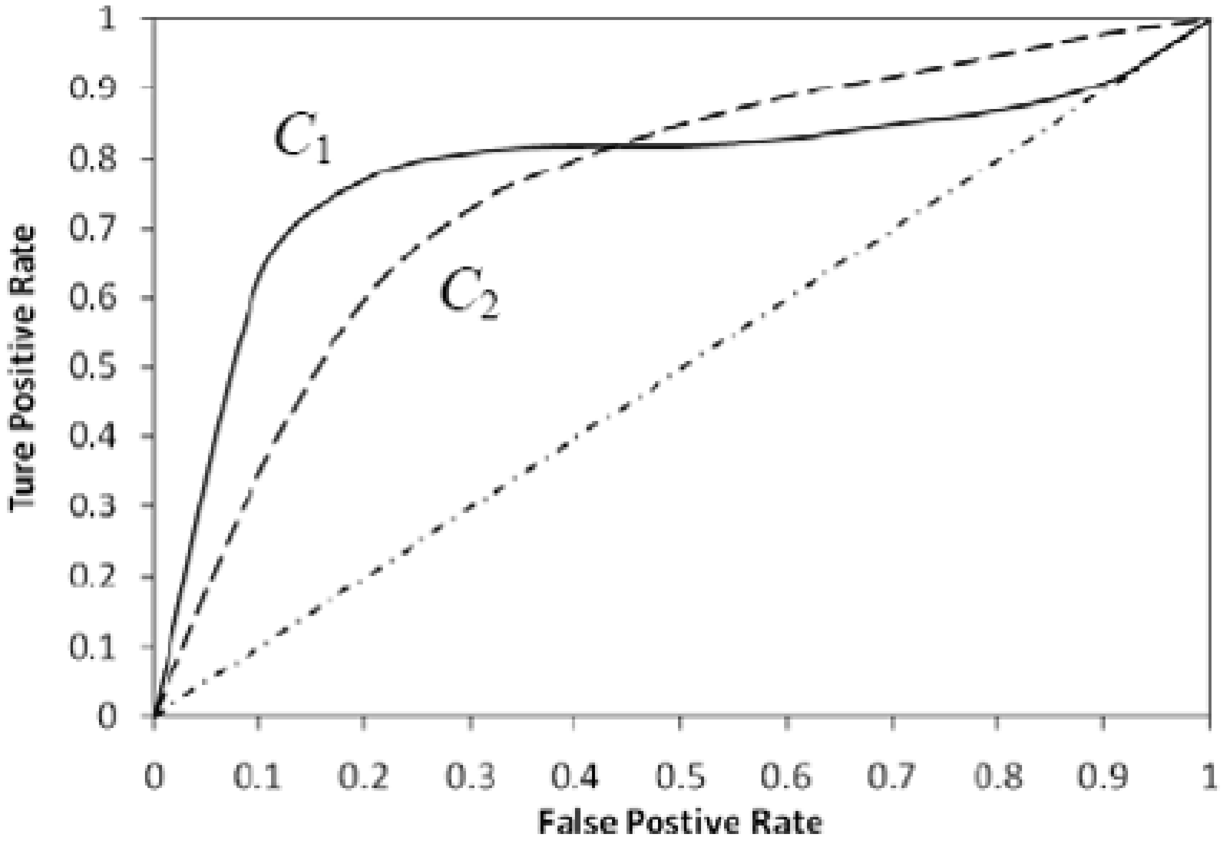
TPR is the proportion of the positive samples that are correctly classified

- False positive rate:

$$FPR = \frac{FP}{TN + FP} = \frac{c}{c+d} = \frac{FP}{N}$$

FPR is the proportion of the negative samples that are misclassified

Example ROC Curves



ROC curves for two classifiers (C_1 and C_2) on the same data

Area Under the Curve (AUC)

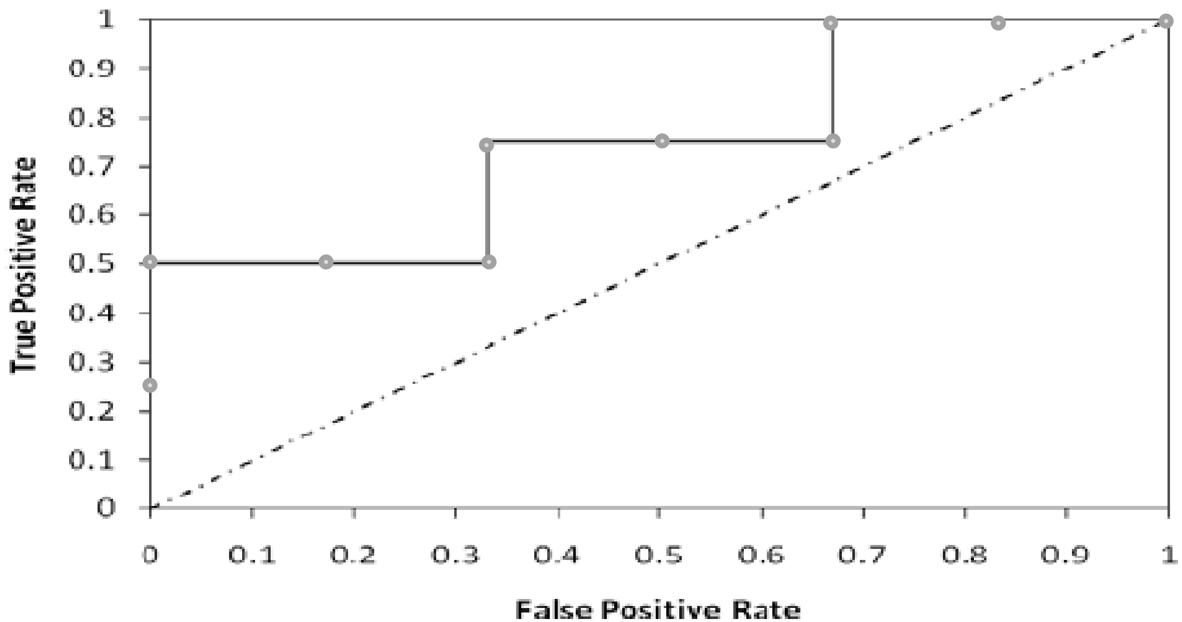
- Which classifier is better, C_1 or C_2?
  - It depends on which region you talk about
- Can we have one measure?
  - Yes, we compute the area under the curve (AUC)
- If AUC for C_i is greater than that of C_j, it is said that C_i is better than C_j
  - If a classifier is perfect, its AUC value is 1
  - If a classifier makes all random guesses, its AUC value is 0.5

Drawing a ROC curve

Rank		1	2	3	4	5	6	7	8	9	10
Actual class		+	+	–	–	+	–	–	+	–	–
TP	0	1	2	2	2	3	3	3	4	4	4
FP	0	0	0	1	2	2	3	4	4	5	6
TN	6	6	6	5	4	4	3	2	2	1	0
FN	4	3	2	2	2	1	1	1	0	0	0
TPR	0	0.25	0.5	0.5	0.5	0.75	0.75	0.75	1	1	1
FPR	0	0	0	0.17	0.33	0.33	0.50	0.67	0.67	0.83	1

$$TPR = \frac{TP}{P}$$

$$FPR = \frac{FP}{N}$$



Scikit-learn Zone – ROC Curve

- # Use NB to classify the breast_cancer dataset, then draw ROC curve and find auc
- from sklearn import datasets
- from sklearn.model_selection import train_test_split
- from sklearn.naive_bayes import GaussianNB
- bc = datasets.load_breast_cancer()
- X = bc.data
- y = bc.target
- X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)

- classifer = GaussianNB() # Create Gaussian Naive Bayes object
- model = classifer.fit(X_train, y_train) # Train model
- y_pred = model.predict(X_test) # Test the model

- from sklearn.metrics import accuracy_score
- accuracy_score(y_test, y_pred)

Getting Predicted Probabilities

- target_probabilities = model.predict_proba(X_test)
- print("target_probabilities\n", target_probabilities[0:3])

```
target_probabilities
 [[9.99995956e-01 4.04353322e-06]
 [9.92851058e-01 7.14894159e-03]
 [2.81653862e-12 1.00000000e+00]]
```

- print("The classes:", model.classes_) # Get the classes

```
The classes: [0 1]
```

- # We care about predicted probabilities over the +ve class
- target_probabilities = model.predict_proba(X_test)[:,1]

Getting TPR, FPR and AUC

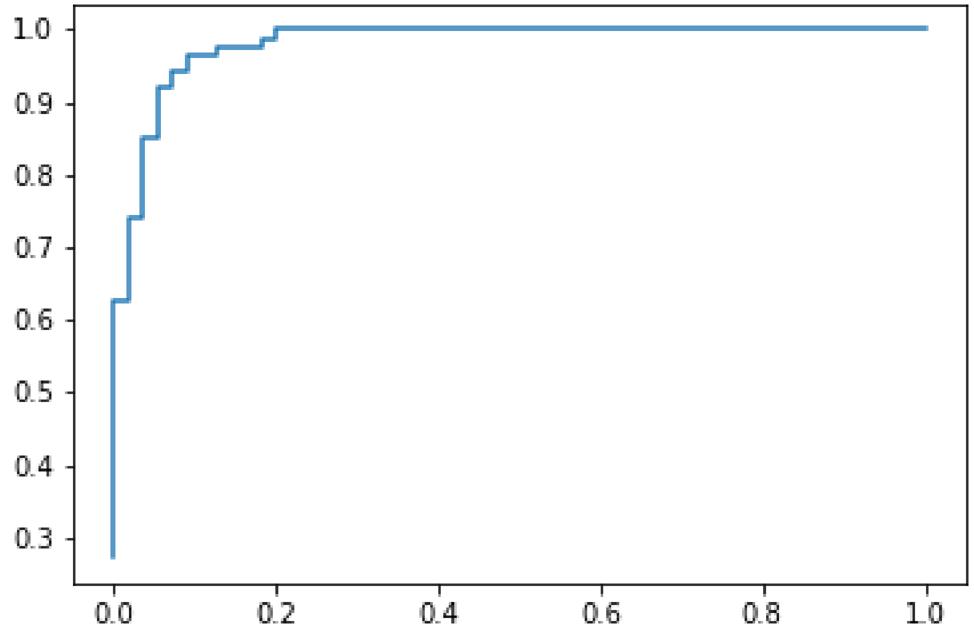
- # Create true and false positive rates
- from sklearn.metrics import roc_curve, roc_auc_score

- fpr, tpr, threshold = roc_curve(y_test, target_probabilities)

- # Calculate area under curve
- roc_auc_score(y_test, target_probabilities) # 0.9805785123966944

Visualizing the ROC Curve

- import matplotlib.pyplot as plt
- plt.plot(fpr, tpr)
- plt.show()



Better ROC Visualization

- import matplotlib.pyplot as plt
- plt.plot(fpr, tpr)
- plt.plot([0, 1], ls="--") # diagonal line
- plt.plot([0, 0], [1, 0] , c=".7") # left vertical line
- plt.plot([1, 1] , c=".7") # top horizontal line
- plt.ylabel("True Positive Rate")
- plt.xlabel("False Positive Rate")
- plt.show()