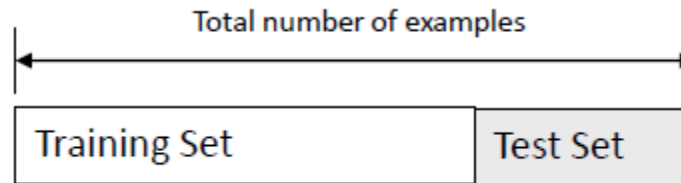


Cross Validation

The holdout method

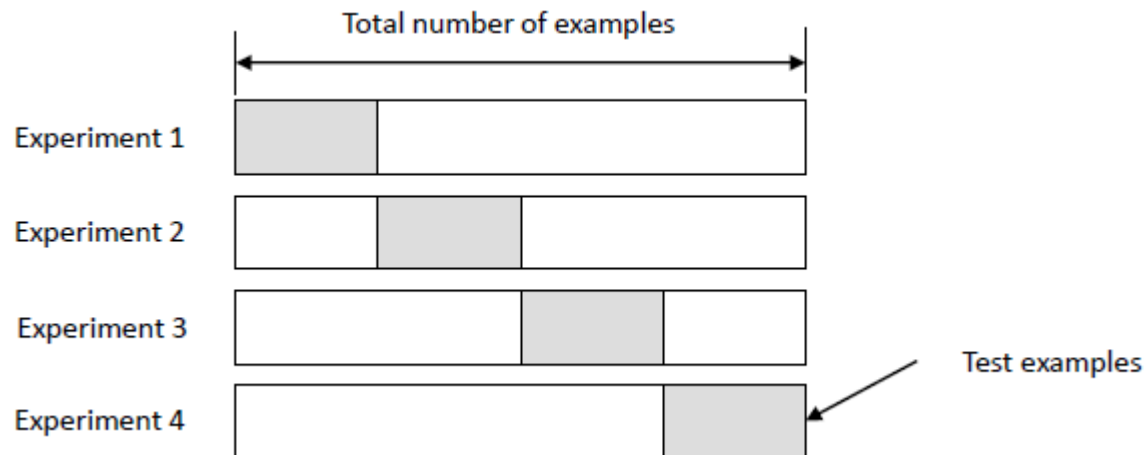
- Split dataset into two groups
 - Training set: used to train the classifier
 - Test set: used to estimate the error rate of the trained classifier



- Drawbacks
 - Since it is a single train-and-test experiment, the holdout estimate of error rate will be misleading if we happen to get an “unfortunate” split
 - In problems where we have a sparse dataset we may not be able to afford the “luxury” of setting aside a portion of the dataset for testing

K-fold cross validation (KFCV)

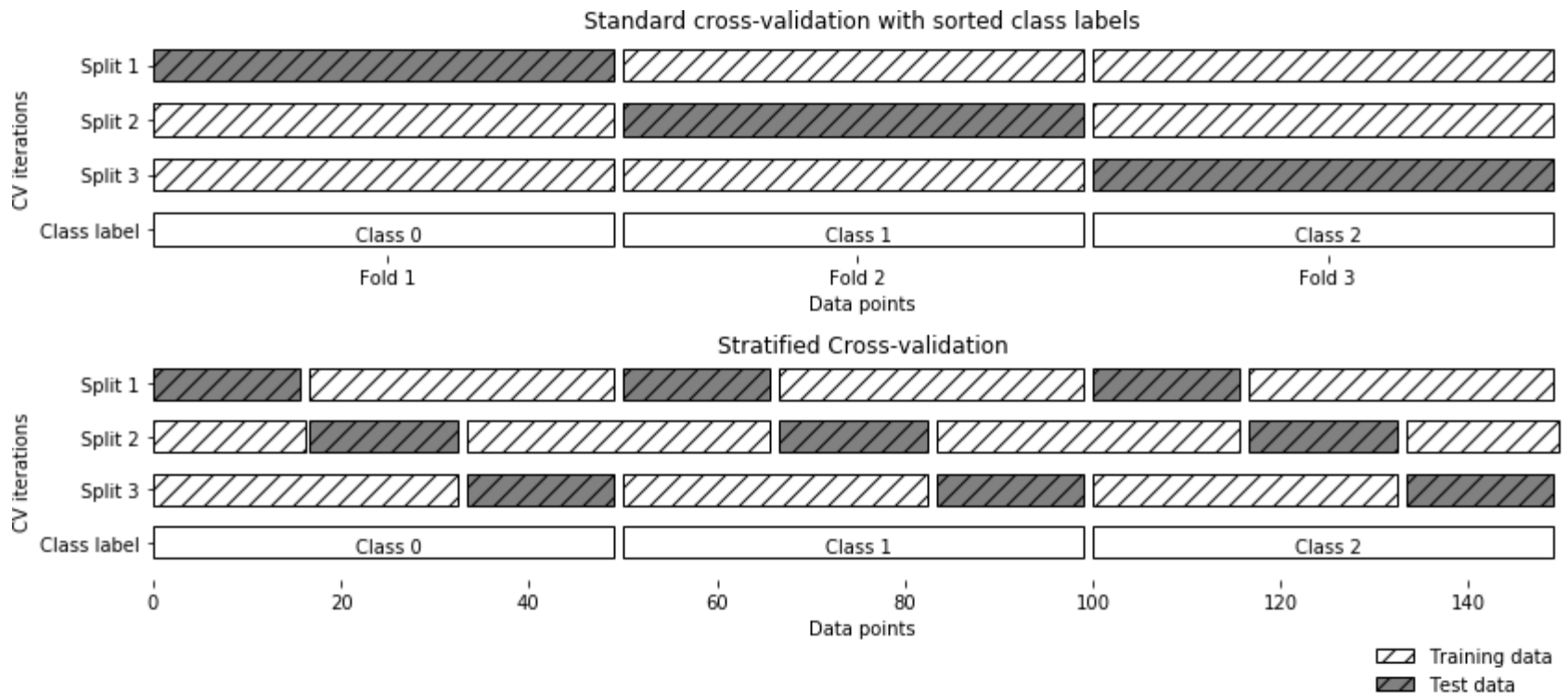
- Create a K-fold partition of the dataset
 - For each of K experiments, use $K-1$ folds for training and a different fold for testing
 - This procedure is illustrated in the following figure for $K=4$



- Advantages
 - The advantage of KFCV is that all the examples in the dataset are eventually used for both training and testing
 - The error is estimated as the average error rate on test examples

Stratified KFCV

- **Stratified KFCV** rearranges the data as to ensure each fold is a good representative of the whole
- It is generally a better scheme, both in terms of bias and variance, when compared to standard cross-validation



```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree, X_train, y_train); scores

array([ 0.93555556,  0.93541203,  0.93303571])
```

```
# Change k
scores = cross_val_score(tree, X_train, y_train, cv=5); scores

array([ 0.92962963,  0.93703704,  0.92962963,  0.94052045,  0.9291
0448])
```

```
print("Mean: {:.3f}\nMin: {:.3f}\nMax: {:.3f}".format(
    scores.mean(), scores.min(), scores.max()))
```

```
Mean: 0.933
Min: 0.929
Max: 0.941
```

```
# Change performance measure
cross_val_score(tree, X_train, y_train, cv=5, scoring='roc_auc')

array([ 0.8243408 ,  0.86625514,  0.87768633,  0.92283951,  0.8773
8398])
```



Other cross validation methods

- Leave-one-out cross-validation (LOOCV)
- Shuffle-split cross-validation
- Cross-validation with groups
- Nested cross-validation

For more information, refer to :

- http://scikit-learn.org/stable/modules/cross_validation.html
- *"Introduction to Machine Learning", pp.312–316, pp.332–334*

Model Tuning: Hyperparameter Optimization



Hyperparameter Optimization: 개념 및 기법

- **Hyperparameter Optimization**

학습을 수행하기 위해 사전에 설정해야 하는 값인 hyperparameter의 최적값을 탐색하는 문제. 여기서, 최적값이란 학습이 완료된 러닝 모델의 일반화 성능을 최고 수준으로 발휘하도록 하는 hyperparameter 값을 의미

- **Manual Search**

- **Grid Search**

- 탐색의 대상이 되는 특정 구간 내의 후보 hyperparameter 값들을 일정한 간격(grid)을 두고 선정하여, 이들 각각에 대하여 성능 결과를 측정한 후 가장 높은 성능을 발휘했던 hyperparameter 값을 최적값으로 선정하는 방법

- **Random Search**

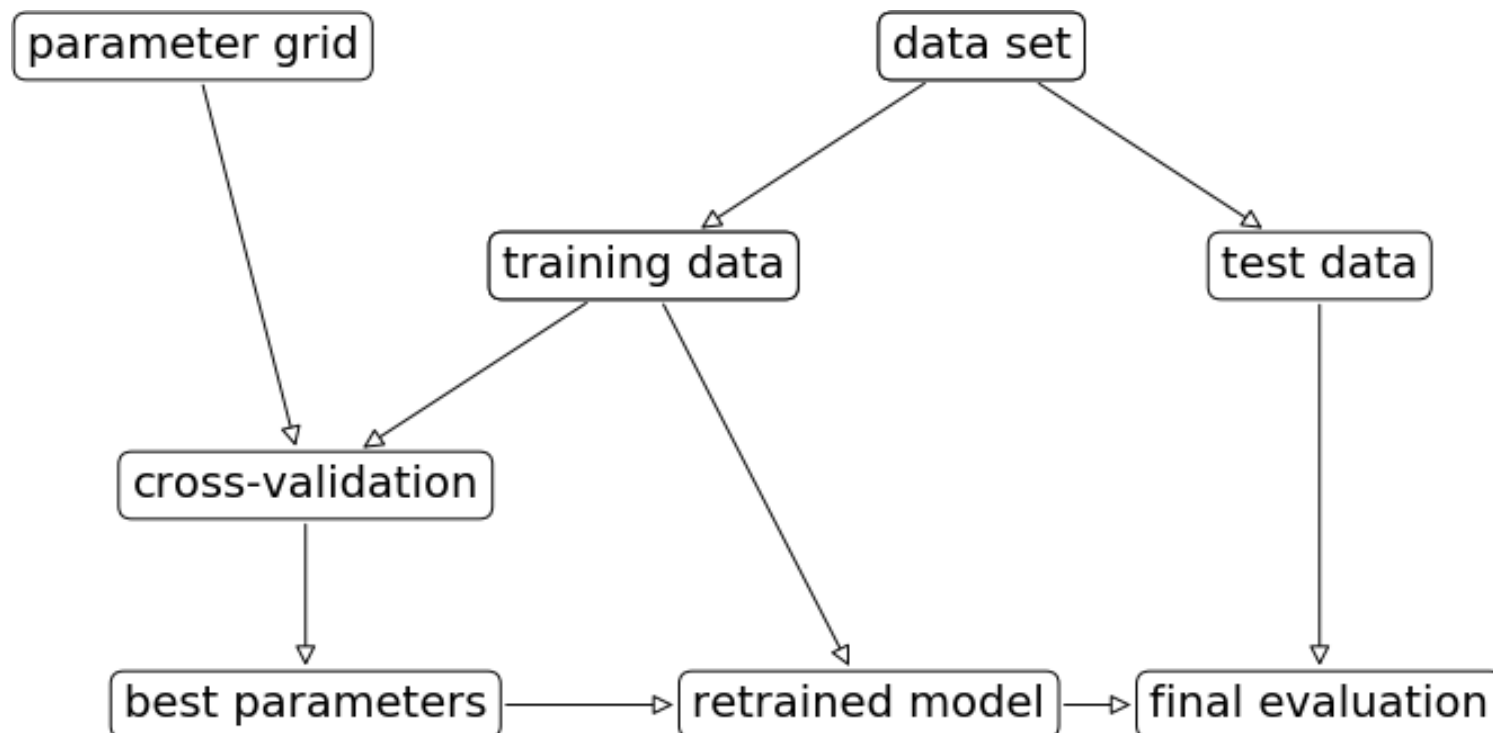
- 탐색 대상 구간 내의 후보 hyperparameter 값들을 랜덤 샘플링을 통해 선정. Grid Search에 비해 불필요한 반복 수행 횟수를 대폭 줄이면서 동시에 정해진 간격 사이에 위치한 값들에 대해서도 확률적으로 탐색이 가능하므로, 최적값을 더 빨리 찾을 수 있는 것으로 알려져 있음

- **Bayesian Optimization**

- 매 회 새로운 hyperparameter 값에 대한 조사를 수행할 때에 "사전 지식"을 충분히 반영 하면서 동시에 전체적인 탐색 과정을 좀 더 체계적으로 수행하는 방법

Hyperparameter Optimization Workflow

- Parameter estimation using grid search with cross-validation
 - grid search: an exhaustive searching through a manually specified subset of the hyper-parameter space of a learning algorithm
 - using `sklearn.model_selection.GridSearchCV`



Set the parameters for grid search

```
# param_grid: dictionary with parameters names as keys and
# lists of parameter settings to try as values
param_grid = {'n_estimators': [100, 200, 300],
              'max_features': range(7,10)}
param_grid

{'max_features': range(7, 10), 'n_estimators': [100, 200, 300]}
```

Grid search with cross-validation

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(max_depth=5, random_state=0)
grid_search = GridSearchCV(rf, param_grid, cv=5, n_jobs=-1)
```

```
# grid search is very time-consuming
grid_search.fit(X_train, y_train)
```

Evaluate the model with best parameters

```
grid_search.score(X_test, y_test)
```

```
0.9666666666666667
```

```
print("Best parameters: {}".format(grid_search.best_params_))  
print("Best CV score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'max_features': 9, 'n_estimators': 200}
```

```
Best CV score: 0.97
```

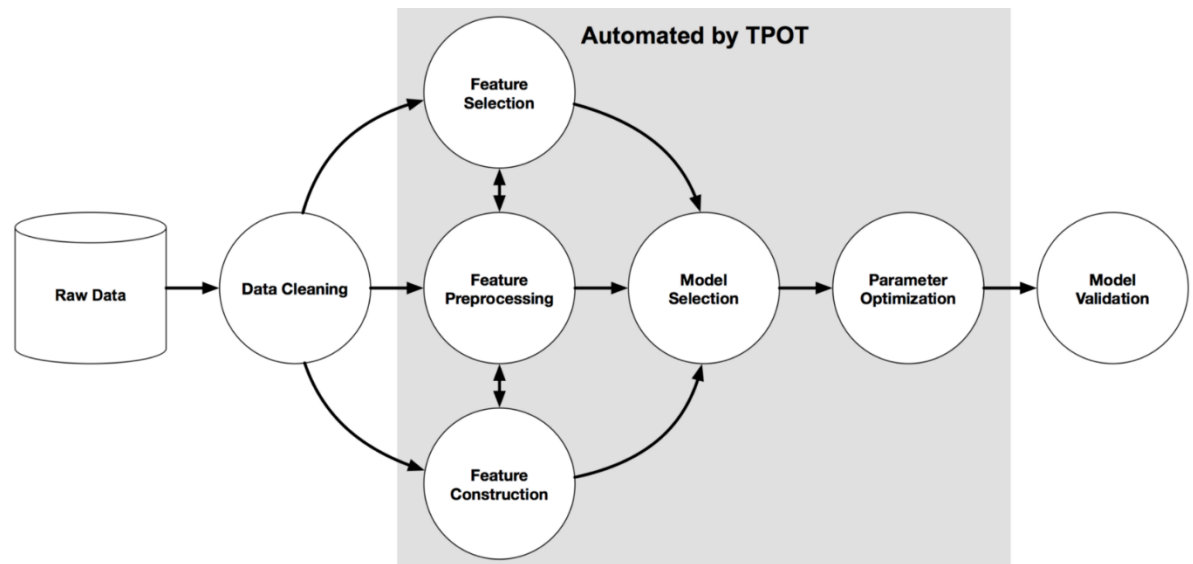
```
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

```
Best estimator:
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',  
                        max_depth=5, max_features=9, max_leaf_nodes=None,  
                        min_impurity_decrease=0.0, min_impurity_split=None,  
                        min_samples_leaf=1, min_samples_split=2,  
                        min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=1,  
                        oob_score=False, random_state=0, verbose=0, warm_start=False)
```

AutoML

- The goal is to automate the building of ML pipelines
- Open-source AutoML packages
 - The Tree-Based Pipeline Optimization Tool (TPOT)
 - Hyperopt
 - scikit-optimize



Evaluation Metrics: Measuring Model Performance



모형평가의 개념 및 고려사항

■ 모형평가란

- 고려된 서로 다른 모형들 중 어느 것이 가장 우수한 예측력을 보유하고 있는지, 선택된 모형이 '임의의 모형(random model)' 보다 우수한지 등을 비교하고 분석하는 과정을 말한다.
- 이 때 다양한 평가지표와 도식을 활용하는데, 머신러닝 애플리케이션의 목적이나 데이터 특성에 따라 적절한 성능지표(performance measure)를 선택해야 한다.

■ 모형 선택 시 고려사항

- (일반화 가능성) 같은 모집단 내의 다른 데이터에 적용하는 경우 얼마나 안정적인 결과를 제공해 주는가?
- (효율성) 얼마나 적은 feature를 사용하여 모형을 구축했는가?
- (정확성) 모형이 실제 문제에 적용될 수 있을 만큼 충분한 성능이 나오는가?

Confusion Matrix

		Predicted		
		Negative (0)	Positive (1)	
Actual	Negative (0)	True Negative TN	False Positive FP (Type I error)	Specificity $= \frac{TN}{TN + FP}$
	Positive (1)	False Negative FN (Type II error)	True Positive TP	Recall, Sensitivity, True positive rate (TPR) $= \frac{TP}{TP + FN}$
Accuracy $= \frac{TP + TN}{TP + TN + FP + FN}$			Precision, Positive predictive value (PPV) $= \frac{TP}{TP + FP}$	F1-score $= 2 \times \frac{Recall \times Precision}{Recall + Precision}$

For more information, refer to https://en.wikipedia.org/wiki/Confusion_matrix

Confusion Matrix

		Predicted		
		납입 정상 (0)	납입 연체 (1)	
Actual	납입 정상 (0)	401	2	Specificity $= \frac{401}{401 + 2}$
	납입 연체 (1)	8	39	Recall, Sensitivity, True positive rate (TPR) $= ?$
		Accuracy $= \frac{440}{440 + 10}$		Precision, Positive predictive value (PPV) $= ?$
				F1-score $= ?$



Accuracy vs. Precision vs. Recall

■ Accuracy의 한계

- 오류 중에서 FN 오류(*ex: 연체를 정상으로 예측 / 암환자를 건강한 사람으로 예측*)를 줄이는 것이 FP 오류(*ex: 정상을 연체로 예측 / 건강한 사람을 암환자로 예측*)를 줄이는 것보다 훨씬 중요한 경우 \Rightarrow accuracy는 두 오류의 정도 차이를 구분할 수 없기 때문에 적절한 성능지표가 되지 못함
- 두 클래스 중 하나(*ex: 납입 정상 / 건강한 사람*)가 다른 것(*ex: 납입 연체 / 암환자*) 보다 훨씬 많은 경우(imbalanced datasets) \Rightarrow random model 조차도 높은 정확도를 보이기 때문에 accuracy로는 random model과 진짜로 성능이 우수한 모델을 구분하기 어려움

■ Precision vs. Recall

- FP를 줄이는 것이 목표일 때(*ex: 임상실험을 통한 신약 치료효과 예측*)는 precision을 주로 사용
- FN을 줄이는 것이 목표일 때는 recall을 주로 사용
- precision과 recall은 trade-off의 관계이기 때문에, 클래스가 불균형인 경우에는 이 둘을 조화 평균한 값인 F1-score를 많이 사용

```
from sklearn.datasets import load_digits
digits = load_digits()
y = digits.target == 9 # 숫자 9를 positive class로 설정
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    digits.data, y, random_state=0)
```

Training Models

```
from sklearn.dummy import DummyClassifier
dummy = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
pred_dummy = dummy.predict(X_test)
```

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
pred_tree = tree.predict(X_test)
```

Accuracy

```
from sklearn.metrics import accuracy_score
print("Dummy model:")
print(accuracy_score(y_test, pred_dummy))
print("Decision tree:")
print(accuracy_score(y_test, pred_tree))
```

Dummy model:
0.89555555555556
Decision tree:
0.9177777777778

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
print("Dummy model:")
print(confusion_matrix(y_test, pred_dummy))
print("Decision tree:")
print(confusion_matrix(y_test, pred_tree))
```

Dummy model:
[[403 0]
 [47 0]]
Decision tree:
[[390 13]
 [24 23]]

Classification Report

```
from sklearn.metrics import classification_report
print("Dummy model:")
print(classification_report(y_test, pred_dummy,
                           target_names=["not 9", "9"]))
print("\nDecision tree:")
print(classification_report(y_test, pred_tree,
                           target_names=["not 9", "9"]))
```

Dummy model:

	precision	recall	f1-score	support
not 9	0.90	1.00	0.94	403
9	0.00	0.00	0.00	47
avg / total	0.80	0.90	0.85	450

Decision tree:

	precision	recall	f1-score	support
not 9	0.94	0.97	0.95	403
9	0.64	0.49	0.55	47
avg / total	0.91	0.92	0.91	450

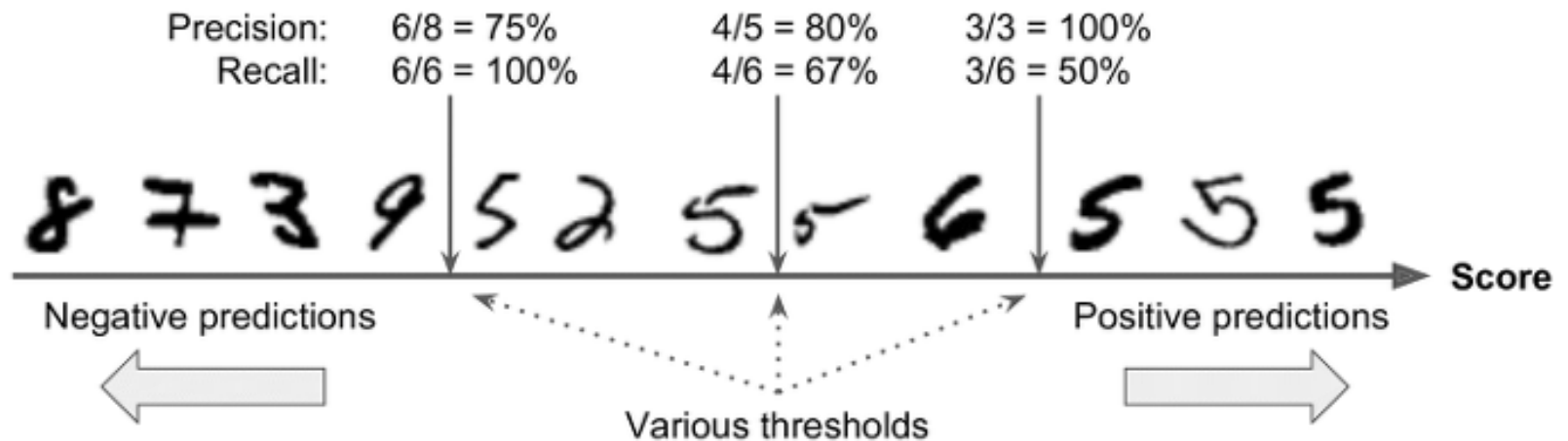


불확실성을 고려하여 예측성능 높이기

- confusion matrix에 나타나는 예측 값은 모형에 담긴 많은 정보가 이미 손실된 상태에서 제공되는 것
- scikit-learn에서 구현된 대부분의 classifier는 예측의 확실성(certainty)을 표현하기 위해 `decision_function`이나 `predict_proba` 메소드를 제공
- 이러한 메소드가 제공하는 출력 값에 임의의 임계 값을 적용하여 예측 값이 결정되는데, 이진 분류에서 `decision_function`은 0, `predict_proba`는 0.5를 default 임계 값으로 사용
 - `predict_proba`: 0.5 이상이면 positive class
 - `decision_function`: 0 보다 크면 positive class
- 분석목적에 따라 이러한 임계 값을 조정함으로써(\Rightarrow FP와 FN이 달라짐) 원하는 평가지표 (*ex: precision 또는 recall*)를 개선할 수 있음

For more details, refer to "Introduction to Machine Learning", pp.347-355

Recall precision trade-off



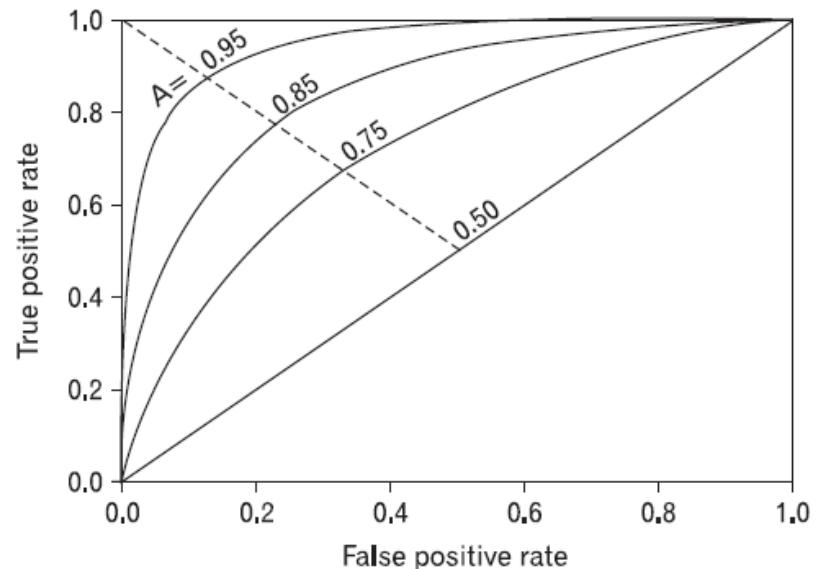
ROC & AUC

■ ROC curve

- Receiver Operating Characteristic curve
- false positive rate($1 - \text{specificity}$)를 x축으로, true positive rate(recall)를 y축으로 하여 둘 간의 관계를 표현한 그래프

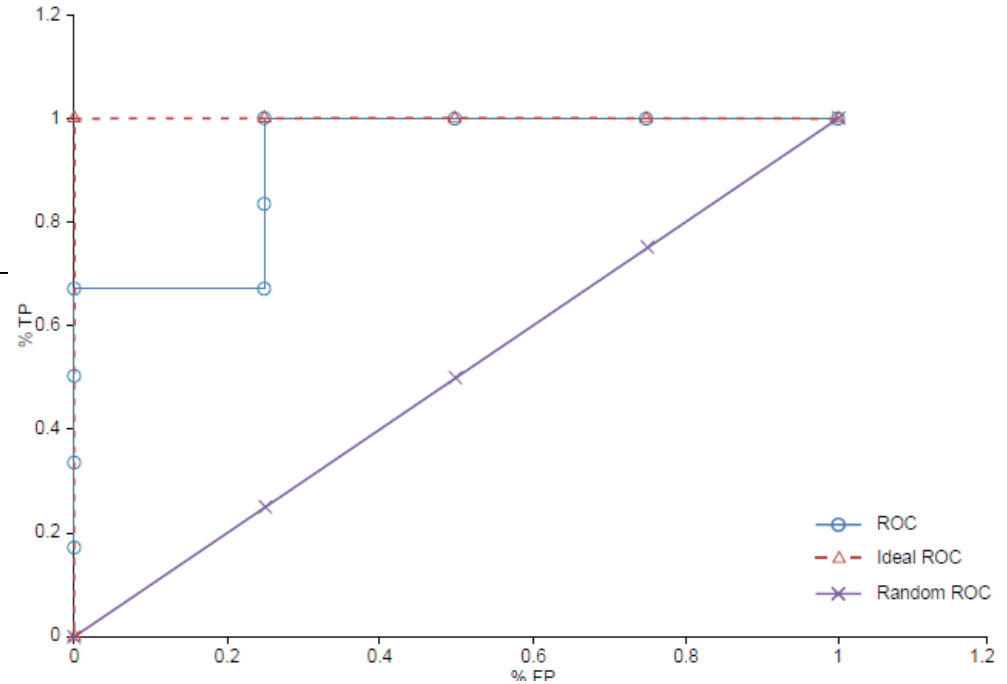
■ AUC

- ROC curve의 밑부분 면적(area under the ROC curve; AUC)이 넓을수록 모형 성능이 높아짐
- Thumb rules:
 - Poor model (0.5 ~ 0.7)
 - Fair model (0.7 ~ 0.8)
 - Good model (0.8 ~ 0.9)
 - Excellent model (0.9 ~ 1.0)



ROC curve

Actual Class	Predicted Class	Confidence of "response"	Type?	Number of TP	Number of FP	X Fraction of FP	Y Fraction of TP
response	response	0.902	TP	1	0	0	0.167
response	response	0.896	TP	2	0	0	0.333
response	response	0.834	TP	3	0	0	0.500
response	response	0.741	TP	4	0	0	0.667
no response	response	0.686	FP	4	1	0.25	0.667
response	response	0.616	TP	5	1	0.25	0.833
response	response	0.609	TP	6	1	0.25	1
no response	response	0.576	FP	6	2	0.5	1
no response	response	0.542	FP	6	3	0.75	1
no response	response	0.530	FP	6	4	1	1
no response	no response	0.440	TN	6	4	1	1
no response	no response	0.428	TN				
no response	no response	0.393	TN				
no response	no response	0.313	TN				
no response	no response	0.298	TN				
no response	no response	0.260	TN				
no response	no response	0.248	TN				
no response	no response	0.247	TN				
no response	no response	0.241	TN				
no response	no response	0.116	TN				



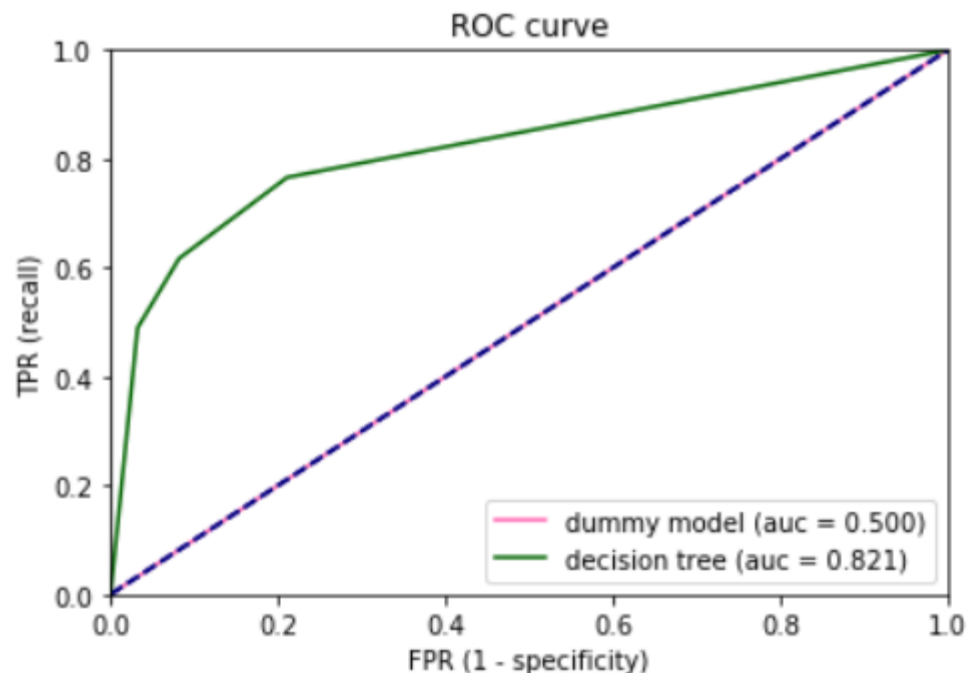

```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
%matplotlib inline
```

Define ROC curve drawing function

```
def plot_roc_curve(fpr, tpr, model, color=None) :
    model = model + ' (auc = %0.3f)' % auc(fpr, tpr)
    plt.plot(fpr, tpr, label=model, color=color)
    plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
    plt.axis([0,1,0,1])
    plt.xlabel('FPR (1 - specificity)')
    plt.ylabel('TPR (recall)')
    plt.title('ROC curve')
    plt.legend(loc="lower right")
```

Plot multiple ROC curves

```
fpr_dummy, tpr_dummy, _ = roc_curve(y_test,  
                                     dummy.predict_proba(X_test)[: ,1])  
plot_roc_curve(fpr_dummy, tpr_dummy, 'dummy model', 'hotpink')  
fpr_tree, tpr_tree, _ = roc_curve(y_test,  
                                   tree.predict_proba(X_test)[: ,1])  
plot_roc_curve(fpr_tree, tpr_tree, 'decision tree', 'darkgreen')
```

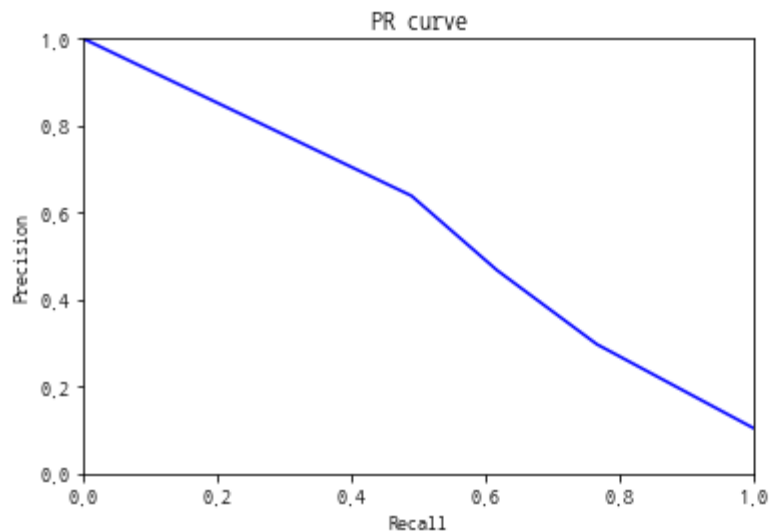


Precision-recall curve

```
from sklearn.metrics import precision_recall_curve
```

```
def plot_precision_recall_curve(precisions, recalls) :  
    plt.plot(recalls, precisions, color='blue')  
    plt.axis([0,1,0,1])  
    plt.xlabel('Recall')  
    plt.ylabel('Precision')  
    plt.title('PR curve')
```

```
precisions, recalls, _ = precision_recall_curve(y_test,  
                                                tree.predict_proba(X_test)[:,-1])  
plot_precision_recall_curve(precisions, recalls)
```





Other evaluation charts

- Gains Chart
 - Refer to http://mlwiki.org/index.php/Cumulative_Gain_Chart
- Lift chart
- Response Chart
- Profit Chart
- ROI Chart