



# 분류 모델

1. 로지스틱 회귀
2. 의사결정 트리
3. 앙상블 학습
4. 랜덤 포레스트

# 로지스틱 회귀(Logistic Regression)

- 로지스틱 회귀(Logistic Regression)

회귀(Regression)를 사용하여 데이터가 어떤 범주에 속할 확률을 0에서 1 사이의 값으로 예측  
확률에 따라 가능성이 더 높은 범주에 속하는 것으로 분류해 주는 지도 학습 알고리즘

메일에 대해 스팸일 확률이 0.5 이상이면 spam으로 분류하고, 확률이 0.5보다 작은 경우 ham으로 분류  
데이터가 2개의 범주 중 하나에 속하도록 결정하는 것을 2진 분류(binary classification)라고 함

로지스틱 회귀에서 데이터가 특정 범주에 속할 확률을 예측하기 위한 과정

- ① 모든 속성(feature)들의 계수(coefficient)와 절편(intercept)을 0으로 초기화
- ② 각 속성들의 값(value)에 계수(coefficient)를 곱해서 log-odds를 계산
- ③ log-odds를 sigmoid 함수에 넣어서  $[0,1]$  범위의 확률 계산

# 로지스틱 회귀(Logistic Regression)

- Odds 란?

특정 사건이 발생할 확률을 발생하지 않을 확률로 나누어 준 값

$$Odds = \frac{P(event\ occurring)}{P(event\ not\ occurring)}$$

- log- Odds 란?

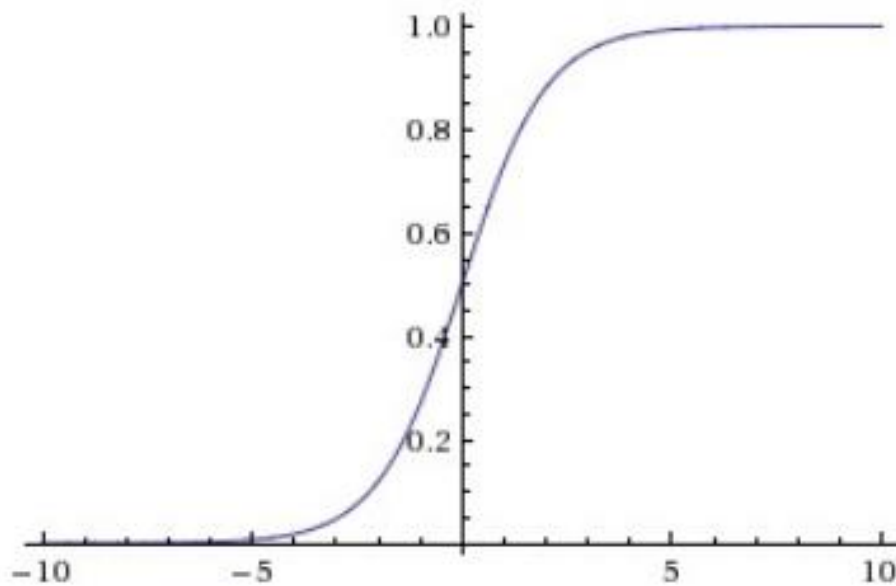
Odds에 로그를 취한 값

$$\log \left( Odds = \frac{P(event\ occurring)}{P(event\ not\ occurring)} \right)$$

# 로지스틱 회귀(Logistic Regression)

- 시그모이드(Sigmoid) 함수

어떠한 값이 입력되더라도 0 ~ 1 사이의 값 만을 출력하는 함수



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

# 로지스틱 회귀(Logistic Regression)

- 로그 손실(log loss)

- 로지스틱 회귀에 대한 손실 함수는 Log Loss(로그 손실)
- 로지스틱 함수를 구성하는 계수와 절편에 대해 Log Loss(로그 손실)을 최소화하는 값을 찾는 것
- 경사하강법(Gradient Descent)을 사용하여 모든 데이터에서 로그 손실(Log Loss)을 최소화하는 계수를 찾을 수 있음

- Classification Threshold (임계값)

- 대부분의 알고리즘에서 기본 임계 값은 0.5
- 필요에 따라 모델의 임계값을 변경
- 암 진단과 같은 민감한 모델의 경우 0.3이나 0.4로 임계값을 낮춰 모델의 민감도를 높임

# 로지스틱 회귀(Logistic Regression)

- 로지스틱 회귀 모델 fitting

```
from sklearn.linear_model.logistic import LogisticRegression
```

```
clf = LogisticRegression()
```

```
from sklearn.model_selection import GridSearchCV
```

```
params = {'C': [0.01, 0.1, 1], "penalty": ["l1", "l2"]}  
grid_search = GridSearchCV(clf, params, n_jobs=-1, cv=3, scoring="roc_auc")  
grid_search.fit(X_train, y_train)
```

```
print(grid_search.best_params_)
```

```
{'C': 1, 'penalty': 'l1'}
```

```
clf_best = grid_search.best_estimator_  
y_pred = clf_best.predict(X_test)
```

# 로지스틱 회귀(Logistic Regression)

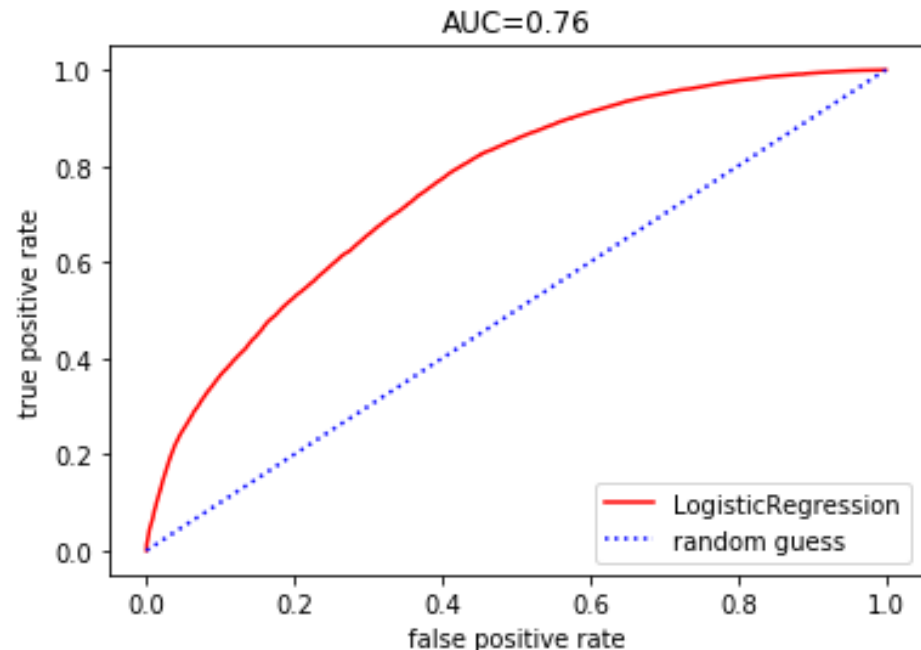
## ■ 모델 성능 평가

```
: from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve
accuracy = accuracy_score(y_test, y_pred)
confusion_matrix = confusion_matrix(y_test, y_pred)
print(accuracy, confusion_matrix)
y_pred_proba = clf_best.predict_proba(X_test)
```

```
0.83197 [[80838 1672]
        [15131 2359]]
```

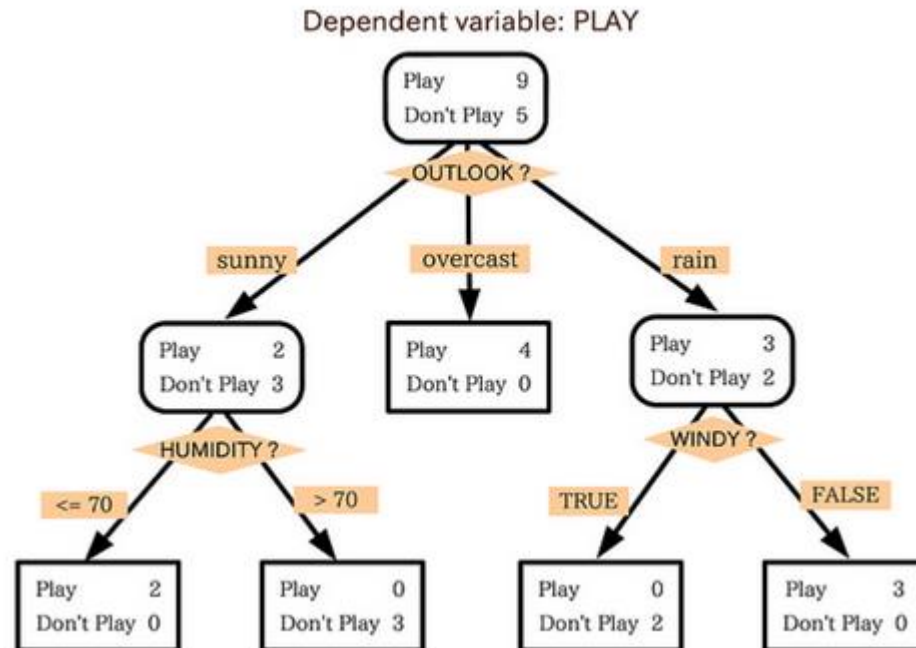
```
: import matplotlib.pyplot as plt

fpr, tpr, _ = roc_curve(y_test, y_pred_proba[:, 1])
auc = roc_auc_score(y_test, y_pred_proba[:, 1])
plt.plot(fpr, tpr, "r-", label="LogisticRegression")
plt.plot([0, 1], [0, 1], "b:", label="random guess")
plt.xlabel("false positive rate")
plt.ylabel("true positive rate")
plt.title("AUC={0:.2f}".format(auc))
plt.legend(loc="lower right");
plt.show()
```



# 의사결정 트리(Decision Tree)

- 의사결정 트리(Decision Tree)
- 분류, 회귀 작업에 사용되며, 복잡한 데이터 세트도 학습 가능
- 설명하기도 쉽고 해석하기도 쉬우면서 나이브 베이즈와 같이 범주형 데이터도 적합
- 의사결정 트리는 강력한 머신러닝 알고리즘 가운데 하나인 랜덤 포레스트의 기본 구성 요소
- ID3, C4.5, CART, CHAID와 같은 알고리즘이 있음



출처 : <https://ratsgo.github.io/machine%20learning/2017/03/26/tree/>



# 의사결정 트리(Decision Tree)

- 의사결정 트리(Decision Tree)의 생성
  - 데이터를 분할하기 위해 가장 중요한 특징이 무엇인지 선택하는 로컬 최적화 방법을 계속 적용하는 탐욕적 방식으로 트리를 생성함
  - 의사결정 트리는 학습 샘플을 서브 셋으로 분할하면서 생성되며, 분할의 과정은 각 서브 셋에 대해 재귀 형태로 진행
  - 각 노드에서의 분할은 특징 값을 기반으로 조건 검사를 통해 진행
  - 서브 셋이 동일한 클래스 레이블을 가지는 경우나 분할을 통한 클래스 분류가 더 이상 의미가 없을 경우 트리 분할 작업을 종료

# 의사결정 트리(Decision Tree) – CART 알고리즘

- CART(Classification and Regression Tree)란?  
불순도를 지니계수(Gini Index)로 계산하는 의사결정 트리 알고리즘
- 노드를 왼쪽, 오른쪽 자식 노드로 분할 확장하면서 트리를 생성
- 분할 단계에서 가장 중요한 특징과 해당 값의 모든 가능한 조합을 측정 함수를 이용해 탐욕적으로 탐색
- 범주형 특징의 경우 해당 특징 값을 가진 샘플들을 오른쪽 자식 노드에 할당
- 수치형 특징의 경우 해당 값보다 큰 값을 가진 샘플들을 오른쪽 자식 노드에 할당

CART는 scikit-learn의 의사결정 트리, 랜덤 포레스트가 사용하는 내부 알고리즘

# 의사결정 트리(Decision Tree) – CART 알고리즘

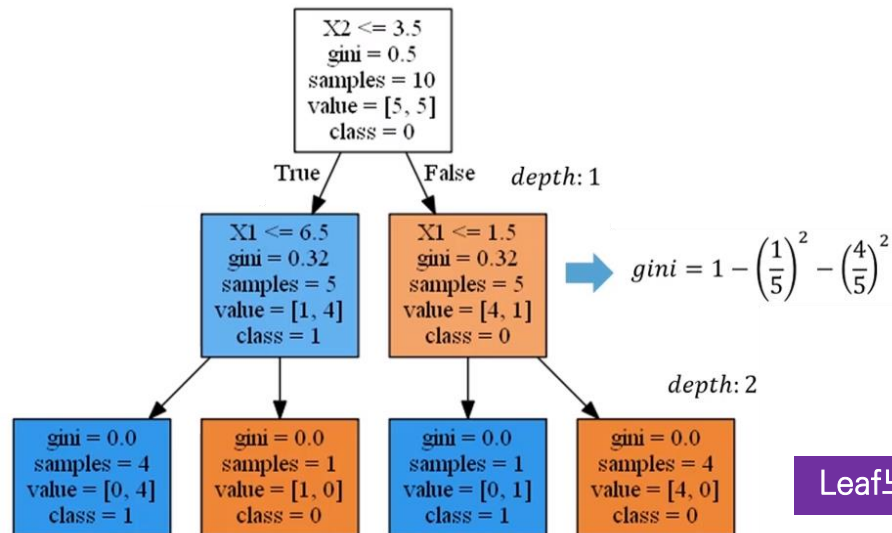
- 지니계수(Gini Index) 란?

불순도(impurity) 특정 지표로서, 데이터의 통계적 분산정도를 정량화해서 표현한 값

- 지니 불순도(Gini Impurity)

$$G_i = 1 - \sum_{k=1}^n P_{ik}^2$$

$P_{i,k}$  : i번째 노드에 있는 훈련 샘플 중 k 클래스에 속한 샘플의 비율  
depth: 0



Leaf노드(자식노드가 없는 노드)에서 지니계수는 0이 됨

# 의사결정 트리(Decision Tree) – CART 알고리즘

## ▪ Iris 데이터 로드 및 Scaling

```
from sklearn.datasets import load_iris
```

```
iris = load_iris()
```

```
X = iris.data  
y = iris.target
```

```
print(X.shape, y.shape)
```

```
(150, 4) (150,)
```

```
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import StandardScaler
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
sc = StandardScaler()  
sc.fit(X_train)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
X_train_std = sc.transform(X_train)  
X_test_std = sc.transform(X_test)
```

# 의사결정 트리(Decision Tree) – CART 알고리즘

- Decision Tree 모델 fitting 및 성능 측정

```
from sklearn.tree import DecisionTreeClassifier
iris_tree = DecisionTreeClassifier(max_depth=5, random_state=0)
iris_tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
                        max_features=None, 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, presort=False,
                        random_state=0, splitter='best')
```

```
from sklearn.metrics import accuracy_score
```

```
y_pred_tr = iris_tree.predict(X_test)
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred_tr))
```

Accuracy: 0.98

# 의사결정 트리(Decision Tree) – CART 알고리즘

- Decision Tree 시각화
  - Graphviz 설치 : Anaconda Prompt에서 conda install graphviz

```
#!/pip install pydot
```

```
from sklearn.tree import export_graphviz
```

```
export_graphviz(iris_tree, out_file="iris.dot", feature_names=iris.feature_names,  
                class_names=iris.target_names, rounded=True, filled=True, impurity=True)
```

```
import pydot
```

```
graph = pydot.graph_from_dot_file("iris.dot")[0]  
graph
```

```
<pydot.Dot at 0x23b6f0072b0>
```

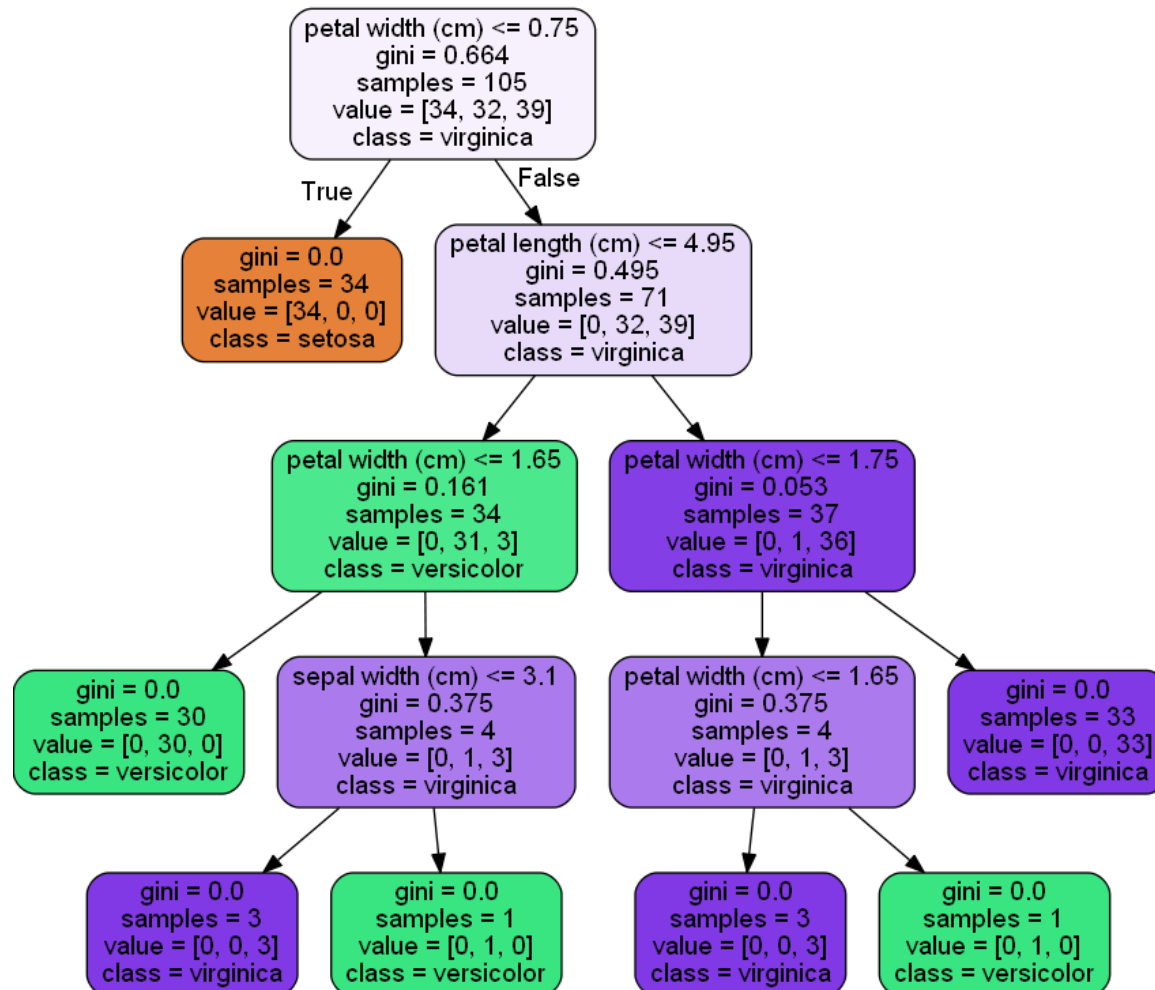
```
iris_png = graph.create_png()
```

```
from IPython.core.display import Image
```

```
Image(iris_png)
```

# 의사결정 트리(Decision Tree) – CART 알고리즘

## Decision Tree 시각화



# 의사결정 트리(Decision Tree) 적용 – 온라인 광고 클릭 예측

- 온라인 광고 클릭 데이터 활용
- CTR : Click-Through Rate. 전체 페이지 뷰 횟수 대비 특정 광고를 클릭한 횟수의 비율
- 클릭 스루 예측이란 어떤 사용자가 현재 보고 있는 웹 페이지에 노출된 광고를 클릭할지를 예측하는 이진 분류 문제
- 캐글 온라인 광고 클릭 예측 모델을 위한 데이터 세트
- <https://www.kaggle.com/c/avazu-ctr-prediction/download/train.gz>
- train.csv : 시간 순으로 정렬된 10일 간의 클릭 스루 예측을 위한 데이터



# 의사결정 트리(Decision Tree) 적용 – 온라인 광고 클릭 예측

## ▪ 데이터 변수 구성

- id : 광고 아이디
- click : 클릭하지 않은 경우 0, 클릭한 경우 1
- hour : YYMMDDHH 포맷
- C1: 익명처리된 범주형 변수
- banner\_pos: 배너 위치 0, 1
- site\_id
- site\_domain
- site\_category
- app\_id
- app\_domain
- app\_category
- device\_id
- device\_ip
- device\_model
- device\_type
- device\_conn\_type
- C14-C21 -- 익명처리된 범주형 변수

# 의사결정 트리(Decision Tree) 적용 – 온라인 광고 클릭 예측

- 데이터 로드 및 전처리

```
import pandas as pd
```

```
train_df = pd.read_csv("data/click/train.csv", nrows=100000)
```

```
# train_df.head()
```

```
unused_columns, label_column = ["id", "hour", "device_id", "device_ip"], "click"  
train_df = train_df.drop(unused_columns, axis=1)
```

```
y_train = train_df[label_column]
```

```
test_df = pd.read_csv("data/click/train.csv", header=0, skiprows=(1, 100000), nrows=100000)
```

```
# test_df.head()
```

```
test_df = test_df.drop(unused_columns, axis=1)  
y_test = test_df[label_column]
```

# 의사결정 트리(Decision Tree) 적용 – 온라인 광고 클릭 예측

- BoW(Bag of Words) 생성
- Bag of Words란 단어들의 순서에 상관없이 단어들의 출현 빈도(frequency)만 고려하는 텍스트 데이터의 수치화 표현 방법
- 해당 문서 내에서 특정 단어가 N번 등장했다면, Bag에는 그 특정 단어가 N개 존재
- BoW를 생성 과정
  - ① 각 단어에 고유한 정수 인덱스를 부여
  - ② 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터 생성

# 의사결정 트리(Decision Tree) 적용 – 온라인 광고 클릭 예측

- DictVectorizer를 이용한 BoW 생성

```
from sklearn.feature_extraction import DictVectorizer
```

```
X_dict_train = list(train_df.drop(label_column, axis=1).T.to_dict().values())  
X_dict_test = list(test_df.drop(label_column, axis=1).T.to_dict().values())
```

```
vectorizer = DictVectorizer(sparse=True)  
X_train = vectorizer.fit_transform(X_dict_train)  
X_test = vectorizer.fit_transform(X_dict_test)
```

# 의사결정 트리(Decision Tree) 적용 – 온라인 광고 클릭 예측

- Grid Search를 이용한 Hyper Parameter 최적화 및 모델 fitting

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

```
params = {"max_depth": [3, 10, None]}
```

```
tree = DecisionTreeClassifier(criterion="gini", min_samples_split=30)
```

```
grid_search = GridSearchCV(tree, params, n_jobs=-1, cv=3, scoring="roc_auc")
```

```
grid_search.fit(X_train, y_train)
```

```
grid_search.best_params_
```

```
{'max_depth': 10}
```

```
tree_best = grid_search.best_estimator_
```

```
y_pred = decision_tree_best.predict(X_test)
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

# 의사결정 트리(Decision Tree) 적용 - 온라인 광고 클릭 예측

- 분류 모델 성능 측정

		Actual Value (as confirmed by experiment)	
		positives	negatives
Predicted Value (predicted by the test)	positives	<b>TP</b> True Positive	<b>FP</b> False Positive
	negatives	<b>FN</b> False Negative	<b>TN</b> True Negative

출처 : <https://medium.com/@awabmohammedomersks>

- 정확도(accuracy) : 전체 샘플에서 정확하게 예측한 샘플 수의 비율

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- 정밀도(precision) : Positive 클래스로 예측한 샘플에서 실제 Positive 클래스에 속하는 샘플 수의 비율

$$Precision = \frac{TP}{TP + FP}$$

# 의사결정 트리(Decision Tree) 적용 – 온라인 광고 클릭 예측

- 재현율(recall) 실제 Positive 클래스에 속한 샘플에서 Positive 클래스에 속한다고 예측한 샘플 수의 비율 (참 긍정률 : True Positive rate, 민감도: Sensitivity)

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

- 특이도(specificity) : 실제 Negative 클래스에 속한 샘플에서 Negative 클래스에 속한다고 예측한 샘플 수의 비율 (1-False Positive Rate)

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

- F1 score : 정밀도와 재현율의 조화평균, score에 상수 2를 곱해 정밀도와 재현율이 모두 1일 경우, score가 1이 되도록 함

$$F1-Score = \frac{2 * Precision * Recall}{Precision + Recall}$$

# 의사결정 트리(Decision Tree) 적용 - 온라인 광고 클릭 예측

- 분류 모델 성능 측정

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

0.83248

```
from sklearn.metrics import confusion_matrix  
confusion_matrix(y_test, y_pred)
```

```
array([[81151, 1359],  
       [15393, 2097]], dtype=int64)
```

```
from sklearn.metrics import recall_score  
recall_score(y_test, y_pred)
```

0.11989708404802744

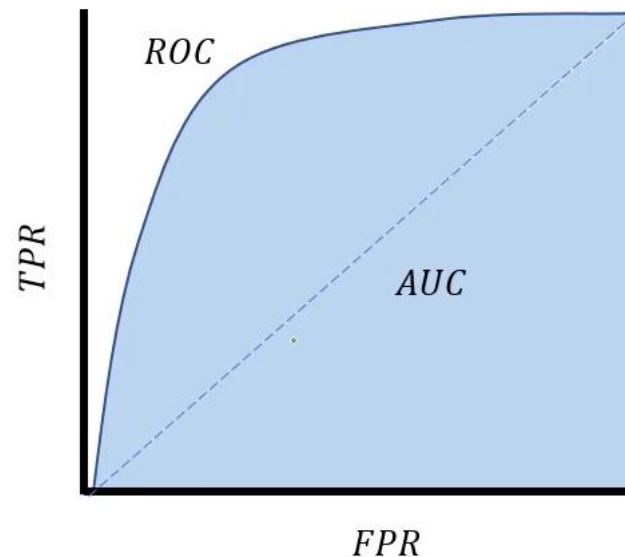
```
from sklearn.metrics import f1_score  
f1_score(y_test, y_pred)
```

0.20022916069894015



# 의사결정 트리(Decision Tree) 적용 - 온라인 광고 클릭 예측

- ROC(Receiver Operating Characteristic) curve
- AUC(Area Under The Curve) 곡선하 면적
- 참 긍정률(TPR)과 거짓 부정률(FPR) 사이를 표현하기 위해 ROC(Receiver Operating Characteristics) 커브를 사용
- 예측된 확률로부터 여러 클래스로 분류를 수행하는 데 활용



모든 케이스에 대해 정확히 분류할 경우( $TPR=1, FPR=0$ )  
AUC 면적은 1이 됨(사각형 모양)

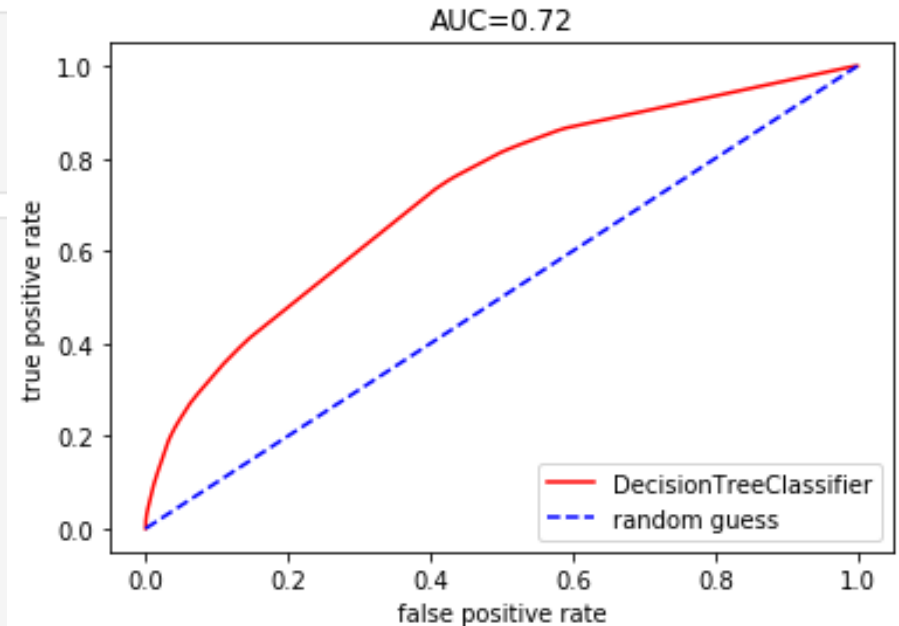
# 의사결정 트리(Decision Tree) 적용 – 온라인 광고 클릭 예측

- ROC(Receiver Operating Characteristic) curve

```
from sklearn.metrics import roc_auc_score, roc_curve
y_pred_proba = decision_tree_best.predict_proba(X_test)[: , 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)
```

```
import matplotlib.pyplot as plt
```

```
plt.plot(fpr, tpr, "r-", label="DecisionTreeClassifier")
plt.plot([0, 1], [0, 1], "b--", label="random guess")
plt.xlabel("false positive rate")
plt.ylabel("true positive rate")
plt.title("AUC={0:.2f}".format(auc))
plt.legend(loc="lower right");
plt.show()
```



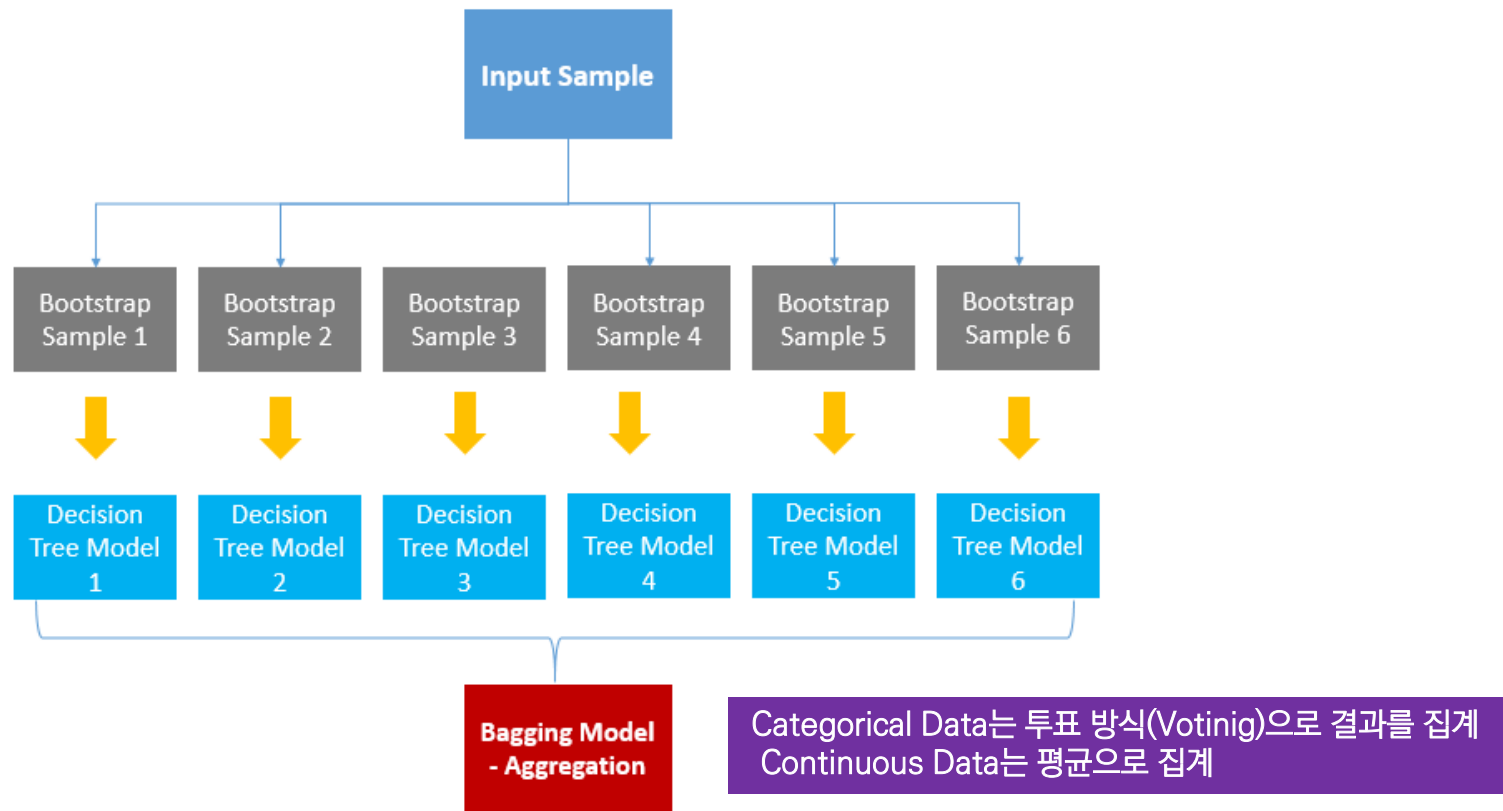
# 앙상블 학습 (Ensemble Learning)

- 앙상블(Ensemble)
  - 여러 개의 의사결정 트리(Decision Tree)를 결합하여 하나의 의사결정 트리를 사용하는 것 보다 더 좋은 성능을 내도록 하는 머신러닝 기법
  - 앙상블 학습의 핵심은 여러 개의 약 분류기 (Weak Classifier)를 결합하여 강 분류기(Strong Classifier) 를 생성
  - 앙상블 학습 기법에는 배깅(Bagging)과 부스팅(Boosting)

# 앙상블 학습 (Ensemble Learning)

## ▪ 배깅(Bagging)

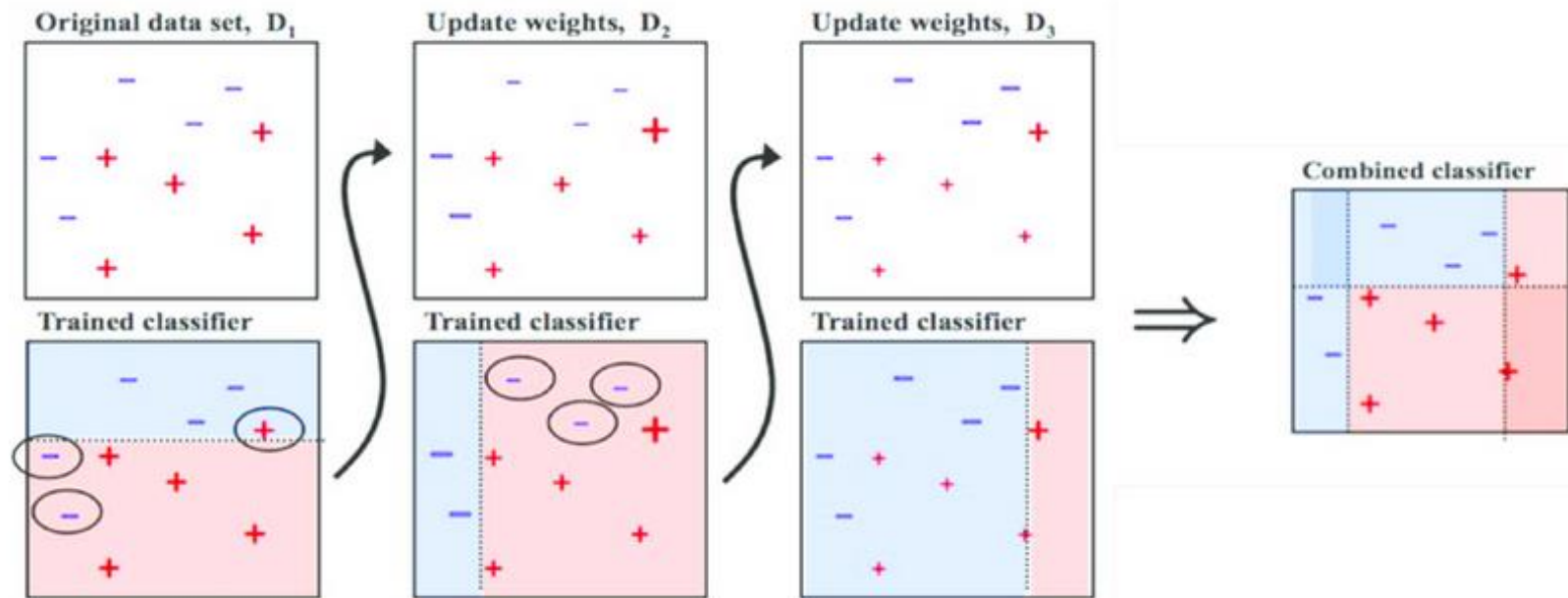
Bagging은 Bootstrap Aggregation의 약자로 데이터로부터 부트스트랩(복원 랜덤 샘플링)을 수행 부트스트랩한 데이터로 모델을 학습, 학습된 모델의 결과를 집계하여 최종 결과 값 도출



# 앙상블 학습 (Ensemble Learning)

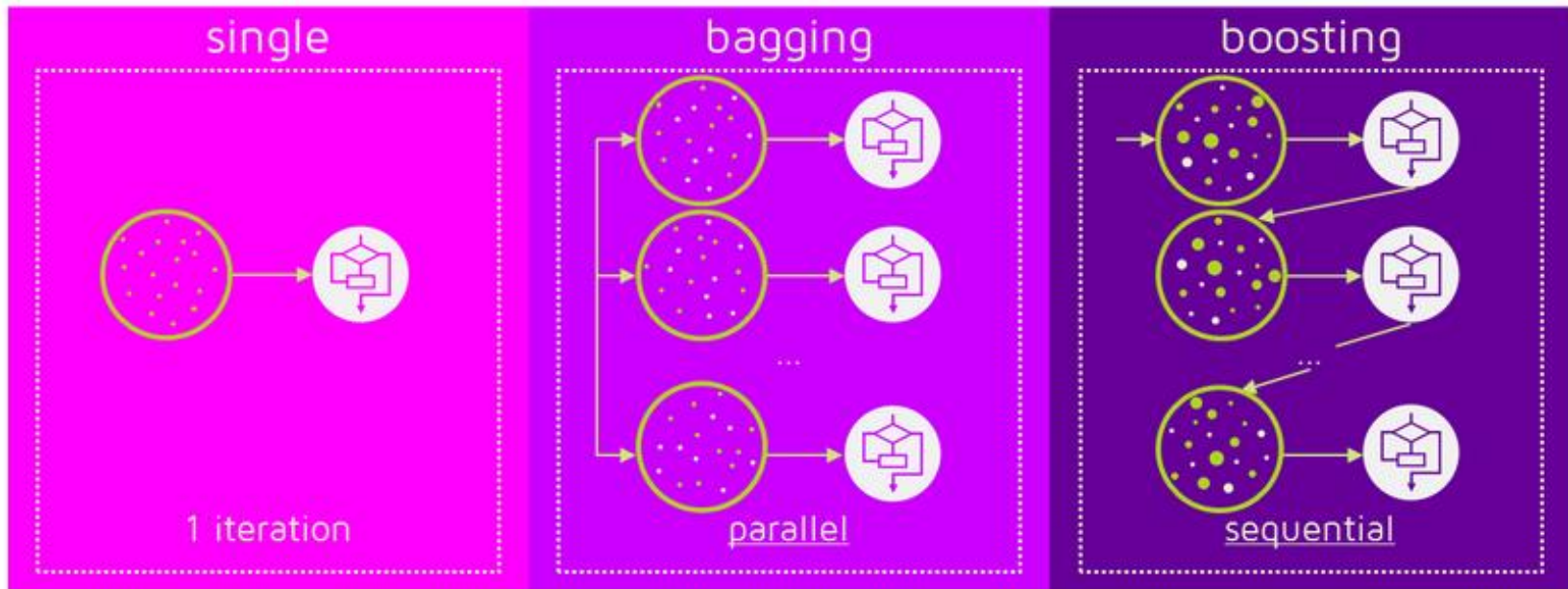
## ■ 부스팅(Boosting)

- 부스팅은 가중치를 활용하여 weak classifier로 strong classifier를 생성하는 과정
- 오분류 데이터에 높은 가중치 부여, 정분류 데이터는 낮은 가중치 부여
- 선행 모델의 오분류 데이터에 가중치를 부여하여 후행 모델에 반영



# 앙상블 학습 (Ensemble Learning)

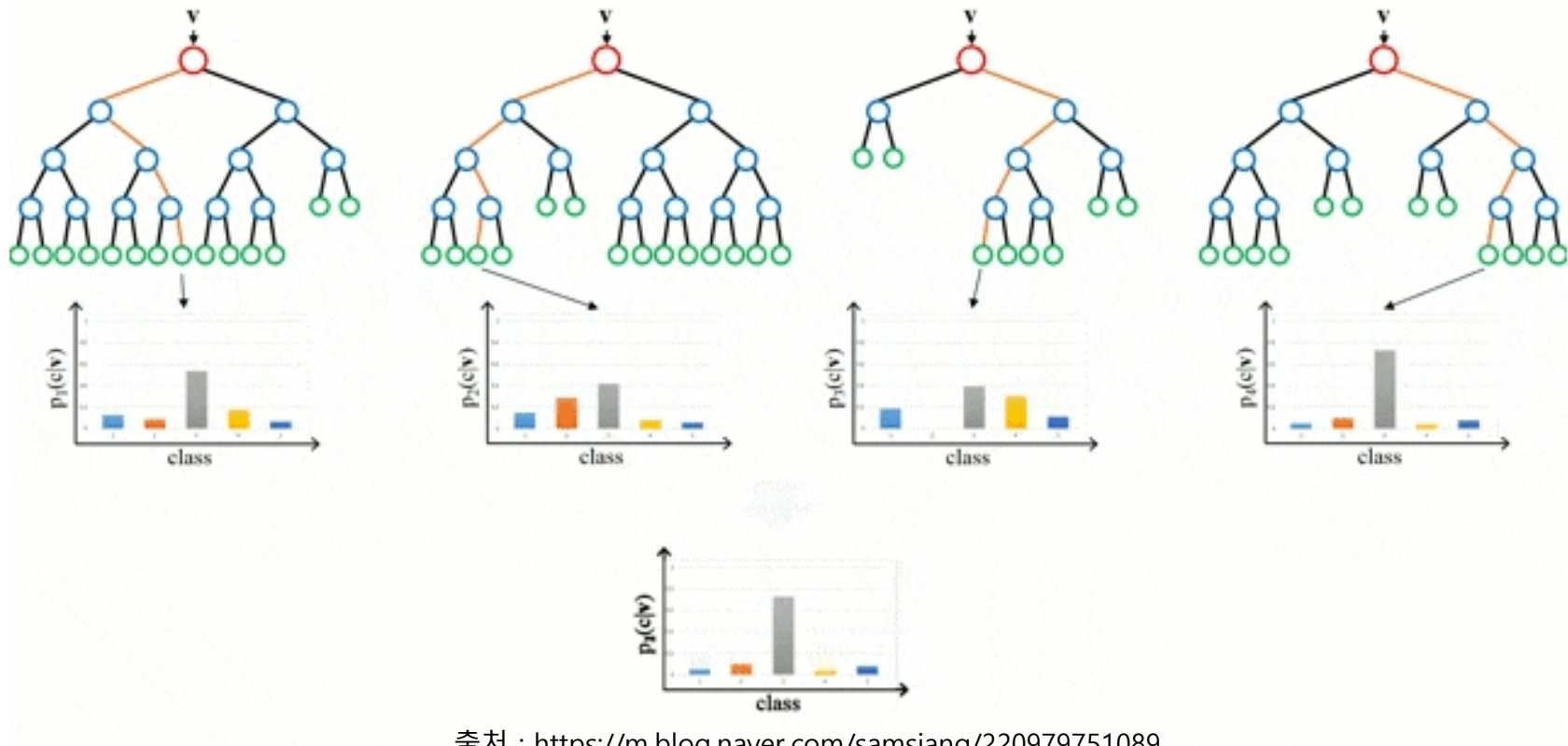
- 배깅(Bagging)과 부스팅(Boosting)
  - 배깅은 병렬로 학습하는 반면, 부스팅은 순차적으로 학습
  - 부스팅은 배깅에 비해 error가 적고 성능이 좋으나 속도가 느리고 오버피팅 가능성 높음
  - 개별 결정 트리의 낮은 성능이 문제라면 부스팅이 적합하고, 오버 피팅이 문제라면 배깅이 적합



출처: [swallow.github.io](https://swallow.github.io)

# 랜덤 포레스트(Random Forest)

- 랜덤 포레스트는 특징 기반 배깅 기법을 적용한 의사결정 트리의 앙상블
- 트리 배깅은 의사결정 트리 모델의 단점 중 하나인 고분산을 줄여 단일 트리보다 고성능
- 랜덤 포레스트의 무작위적 특성은 트리를 보다 다양하게 생성하고, 분산을 축소



출처 : <https://m.blog.naver.com/samsjang/220979751089>

# 랜덤 포레스트(Random Forest)

- 랜덤 포레스트 성능 개선을 위한 하이퍼 파라미터
  - max\_features: 최적의 분할 지점을 찾기 위해 검토할 특징의 개수로  
일반적으로 n차원의 데이터 세트의  $\sqrt{n}$ 의 반올림 값을 설정('auto')
  - n\_estimators : 트리의 개수가 많을수록 성능이 더 좋지만 계산 시간 장시간 소요  
일반적으로 100, 200, 500을 설정
  - min\_sample\_splits : 노드에서 추가 분할을 위해 필요한 샘플의 최소 개수  
숫자가 너무 작으면 오버피팅, 너무 크면 언더피팅 발생가능  
일반적으로 10, 30, 50으로 설정



Q&A

A 3D blue 'Q&A' text is centered on a white surface. A computer mouse with a grey cord is positioned in front of the text, with the cord looping around the base of the letters.

Thank you

A hand-drawn 'Thank you' note is shown on a white background. The text 'Thank you' is written in a cursive script, and a pen is visible at the end of the line.