[2주차]
CH3. 분류

1기 김다선
1기 오수진
1기 홍문경

EURON

목차

3.1 MNIST

MNIST 데이터셋에서 추출한 숫자 이미지

MNIST 데이터셋?

1) 손으로 직접 쓴 숫자 이미지

2) 70,000개, 28 x 28

3) 머신러닝, 딥러닝의 “Hello World”

4) 사이킷런 fetch_openml

```
[ ]  1 from sklearn.datasets import fetch_openml
     2 mnist = fetch_openml('mnist_784', version=1, as_frame=False)
     3 mnist.keys()
```

ID, VERION

#as_frame=False 가 기본
True면 데이터 프레임으로 반환

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'DESCR', 'details', 'categories', 'url'])
```

mnist[‘DESCR’] = 데이터셋 설명
mnist[‘url’] = 데이터셋 url

3.1 MNIST

data -> X(이차원배열)
Target -> y

```
[34]  1 X, y = mnist["data"], mnist["target"]
      2 X.shape
```

(70000, 784) 행 70000개 (샘플), 열 784개 (특성)
특성 : 0~255 픽셀 강도, 28*28

```
[35]  1 y.shape
```

(70000,) 70000개 샘플 구분(0~9) 1차원 배열

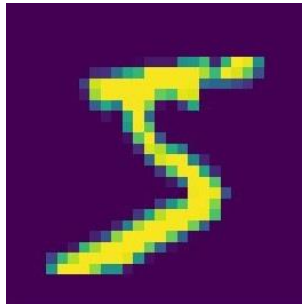
```
1 %matplotlib inline
2 import matplotlib as mpl
3 import matplotlib.pyplot as plt
4
5 some_digit = X[0]
6 some_digit_image = some_digit.reshape(28, 28)
7 plt.imshow(some_digit_image, cmap=mpl.cm.binary) # =cmap="binary"
8 plt.axis("off")
9
10 save_fig("some_digit_plot")
11 plt.show()
```

X[0] : 1행 (784개) 배열

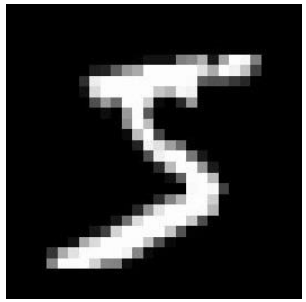
사진출력

그림 저장: some_digit_plot

설정X



cmap = 'gray'



3.1 MNIST

①

```
1 y[0]
```

```
'5'
```

문자열

↓

```
1 y = y.astype(np.uint8)
2 y[0]
```

```
5
```

unsigned integer

③

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

훈련세트 : 앞 60000
테스트 세트: 뒤 10000개

②

```
def plot_digit(data):
    image = data.reshape(28, 28)
    plt.imshow(image, cmap = mpl.cm.binary,
               interpolation="nearest")
    plt.axis("off")
```

'5' 흑백이미지 출력 과정과 동일

최근접 보간법 (가장 가까운 화소값 사용)

```
# 숫자 그림을 위한 추가 함수
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size,size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap = mpl.cm.binary, **options)
    plt.axis("off")
```

여러 개(100개) 숫자를 이미지 한 이미지에 표현

```
plt.figure(figsize=(9,9))
example_images = X[:100]
plot_digits(example_images, images_per_row=10)
save_fig("more_digits_plot")
plt.show()
```

3.2 이진 분류기 훈련

이진 분류 – 2 개의 클래스 구분
다중 분류 – 3 개 이상 클래스 구분

```
1 y_train_5 = (y_train == 5)
2 y_test_5 = (y_test == 5)
```

50이면 True
5가 아니면 False

```
1 from sklearn.linear_model import SGDClassifier
2
3 sgd_clf = SGDClassifier(max_iter=1000, tol=1e-3, random_state=42)
4 sgd_clf.fit(X_train, y_train_5)
```

확률적 경사 하강법(4강

최대 반복 횟수 지정

```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=42, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

```
1 sgd_clf.predict([some_digit])
```

첫번째 샘플 '5'

```
array([ True])
```

Predict 함수 이차원배열 기본

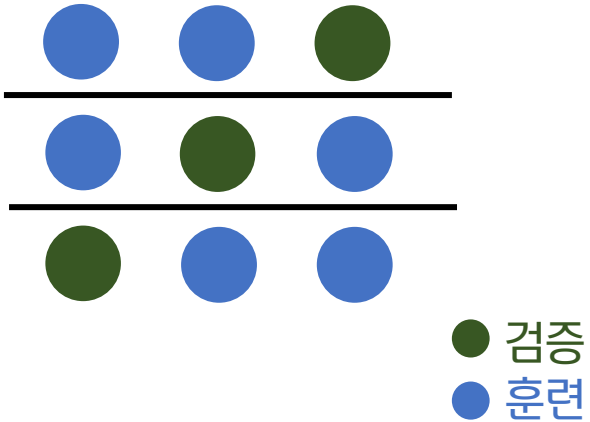
Some_Digit은 일차원 - > [] 이차원 처럼 전달

3.3.성능측정

1) 교차 검증을 사용한 정확도 측정

(CH2. 교차검증)

K – 겹 교차 검증
K – 폴드 교차 검증




```
1 from sklearn.model_selection import cross_val_score
2 cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

```
array([0.95035, 0.96035, 0.9604 ])
```

모두 False 반환 – 90%이상 정확도

```
1 from sklearn.base import BaseEstimator
2 class Never5Classifier(BaseEstimator):
3     def fit(self, X, y=None):
4         pass
5     def predict(self, X):
6         return np.zeros((len(X), 1), dtype=bool)
```

```
1 never_5_clf = Never5Classifier()
2 cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
```

```
array([0.91125, 0.90855, 0.90915])
```

3.3.성능측정

2) 오차 행렬

클래스 A - > 클래스 B 분류 횟수

Ex) 5 -> 3 : 잘못 분류 횟수 (오차 행렬 5행 3열)

예측값 생성

```
1 from sklearn.model_selection import cross_val_predict
2
3 y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

```
1 from sklearn.metrics import confusion_matrix
2
3 confusion_matrix(y_train_5, y_train_pred)
```

```
array([[53892,   687],
       [ 1891,  3530]])
```

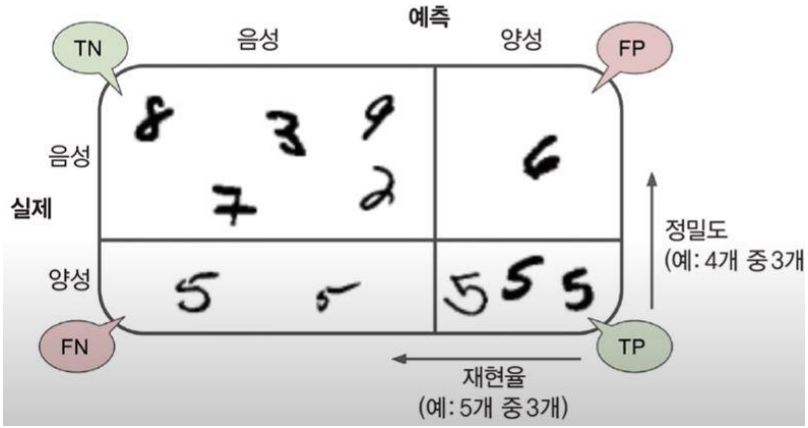
Cross_val_predict:
K-겹 교차 검증(o), 평가점수반환(x), 예측값 반환

Ch2. 복습<Confusion Matrix>

		실제 정답	
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

3.3.성능측정

2) 오차 행렬




```
1 y_train_perfect_predictions = y_train_5
2 confusion_matrix(y_train_5, y_train_perfect_predictions)
```

```
array([[54579,     0],
       [    0,  5421]])
```

Ch2. 복습

1. Precision ; 정밀도
   (positive predictive value; PPV)

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

2. Recall ; 재현율
   (Sensitivity, hit rate)

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

3.3.성능측정

3) 정밀도와 재현율

정밀도

```
1 from sklearn.metrics import precision_score, recall_score
2
3 precision_score(y_train_5, y_train_pred)
```

```
0.8370879772350012
```

```
1 cm = confusion_matrix(y_train_5, y_train_pred)
2 cm[1, 1] / (cm[0, 1] + cm[1, 1])
```

```
0.8370879772350012
```

실제 연산 값과 비교

재현율

```
recall_score(y_train_5, y_train_pred)
```

```
0.6511713705958311
```

```
1 cm[1, 1] / (cm[1, 0] + cm[1, 1])
```

```
0.6511713705958311
```

실제 연산 값과 비교

F1 score

```
1 from sklearn.metrics import f1_score
2
3 f1_score(y_train_5, y_train_pred)
```

```
0.7325171197343846
```

```
1 cm[1, 1] / (cm[1, 1] + (cm[1, 0] + cm[0, 1]) / 2)
```

```
0.7325171197343847
```

실제 연산 값과 비교

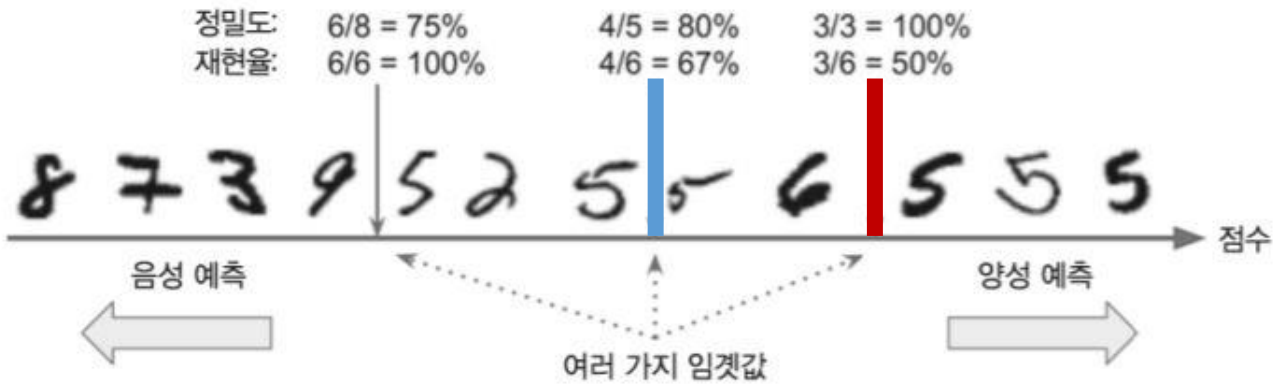
CH2. 복습

4. F1 score : Precision과 Recall의 조화평균

$$(F1\text{-}score) = 2\times\frac{1}{\frac{1}{Precision}+\frac{1}{Recall}} = 2\times\frac{Precision \times Recall}{Precision + Recall}$$

3.3.4 정밀도/재현율 트레이드오프

정밀도/재현율 트레이드오프 : 정밀도와 재현율은 서로 상호 보완적인 평가 지표이기 때문에, 어느 하나를 높이면 다른 하나의 수치는 떨어지게 되는 것.



임계값: 정밀도 80% 재현율 67%

임계값: 정밀도 100% 재현율 50%

➔임계값을 올리면 정밀도 높아지고 재현율 낮아짐

반대로 임계값을 내리면 재현율 높아지고 정밀도 낮아짐

3.3.4 정밀도/재현율 트레이드오프

```
y_scores = sgd_clf.decision_function([some_digit])
y_scores
```

decision_function()을 이용해 각 샘플의 점수를 얻음

```
array([2164.22030239])
```

```
threshold = 0
y_some_digit_pred = (y_scores > threshold)
```

임계값=0

```
y_some_digit_pred
```

```
array([ True])
```

```
threshold = 8000
y_some_digit_pred = (y_scores > threshold)
y_some_digit_pred
```

임계값 ↑

```
array([False])
```

➔ 임계값을 높이면 ,
재현율이 줄어든다!

3.3.4 정밀도/재현율 트레이드오프

적절한 임계값 정하기!

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

cross_val_predict() 함수를 이용해 훈련 세트에 있는 모든 샘플의 점수를 구함.

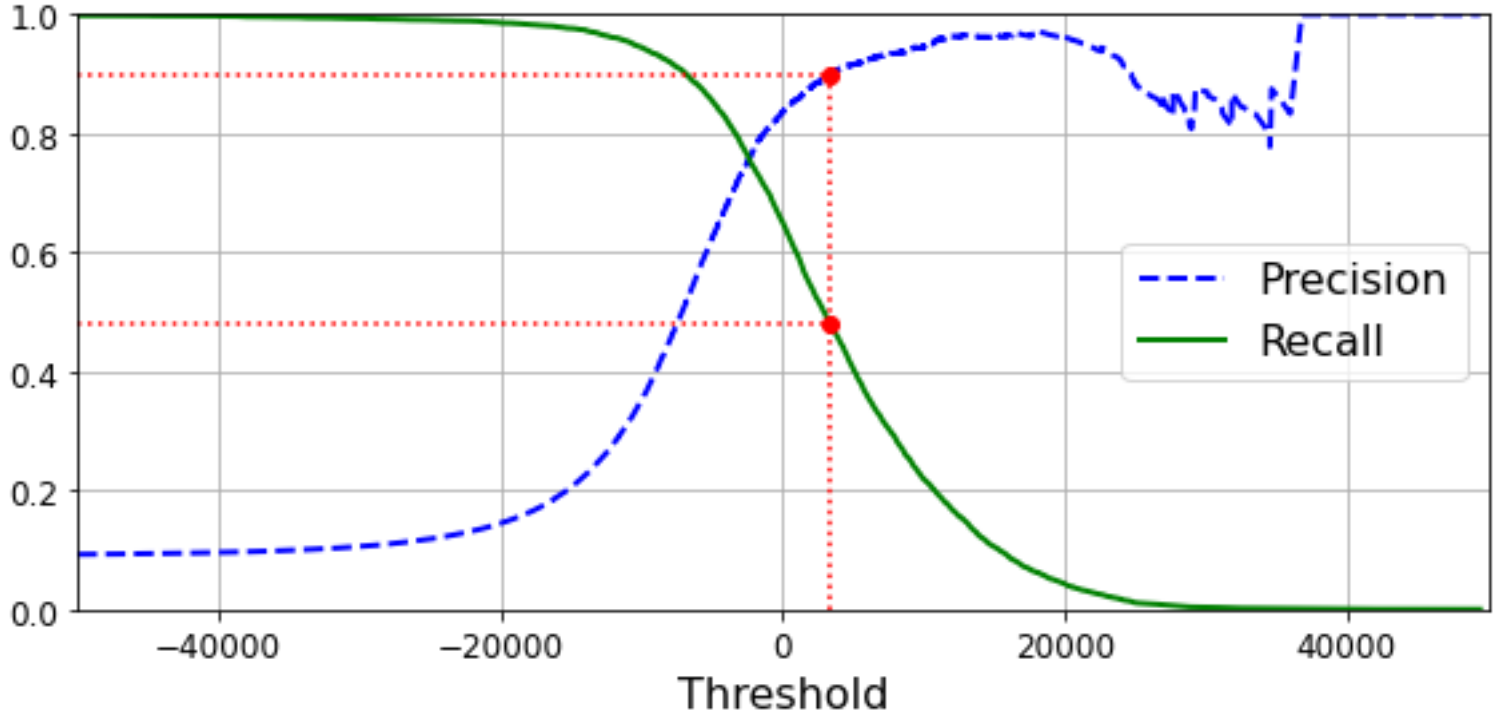
```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

precision_recall_curve() 함수를 이용해
가능한 모든 임계값에 대한 정밀도와 재현율을 계산.

3.3.4 정밀도/재현율 트레이드오프

결정 임계값에 대한 정밀도와 재현율

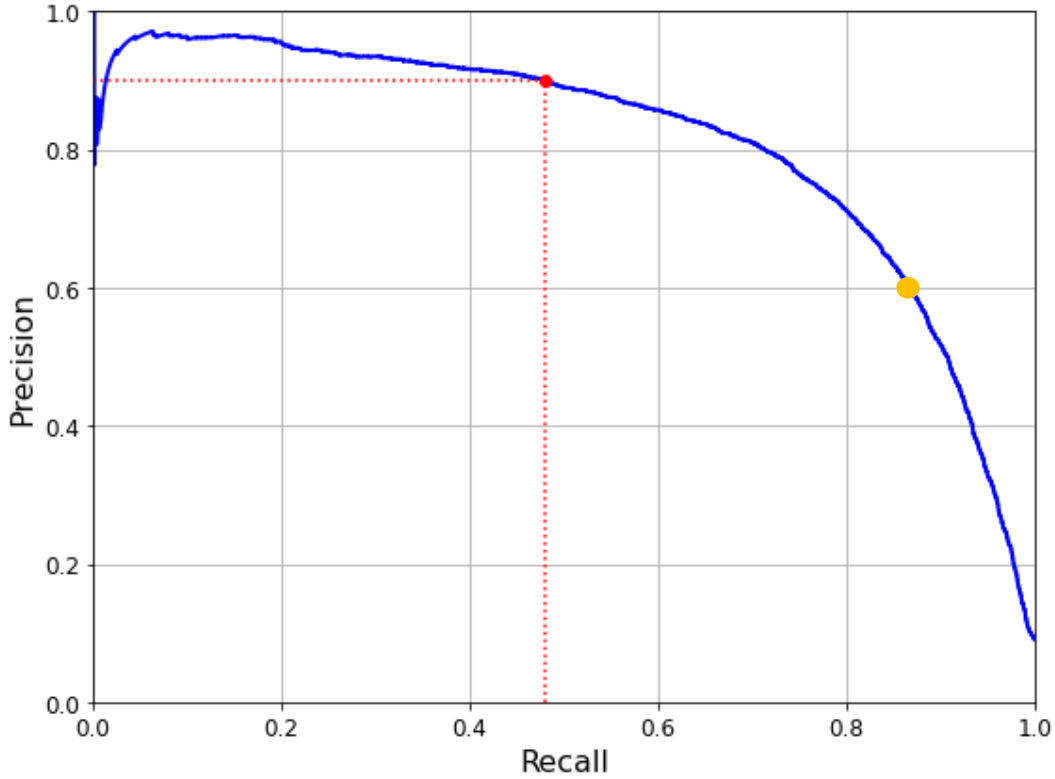
```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
```



3.3.4 정밀도/재현율 트레이드오프

정밀도와 재현율

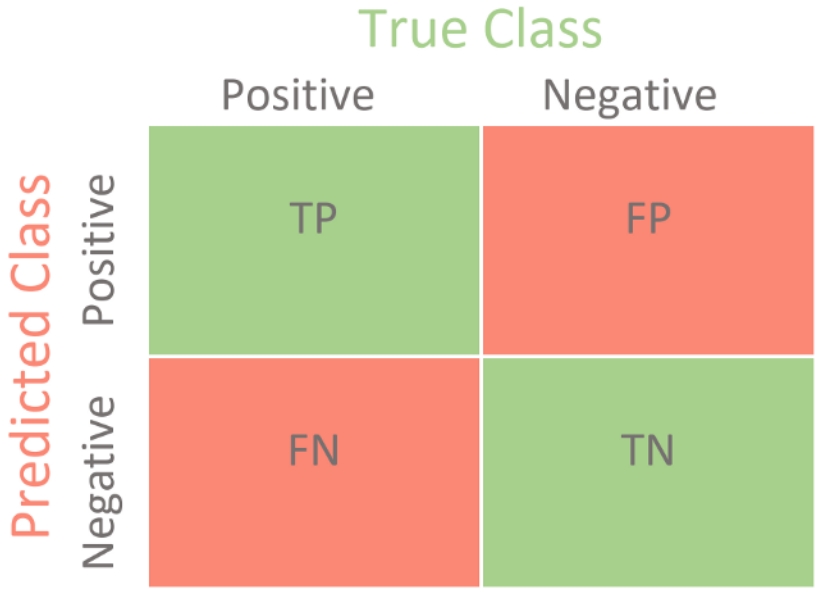
```
def plot_precision_vs_recall(precisions, recalls):
```



3.3.5 ROC 곡선

ROC 곡선(수신기 조작 특성): 민감도(재현율)에 대한 1-특이도 그래프

AUC(area under the curve): ROC 곡선의 밑넓이



<Confusion Matrix>

민감도(재현율)
(true positive rate; TPR)
= TP/(FN+TP)

특이도
(true negative rate; TNR)
= TN/TN+FP

출처: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

3.3.5 ROC 곡선

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

roc_curve() 함수를 이용해
여러 임계값에서
fpr과 tpr계산

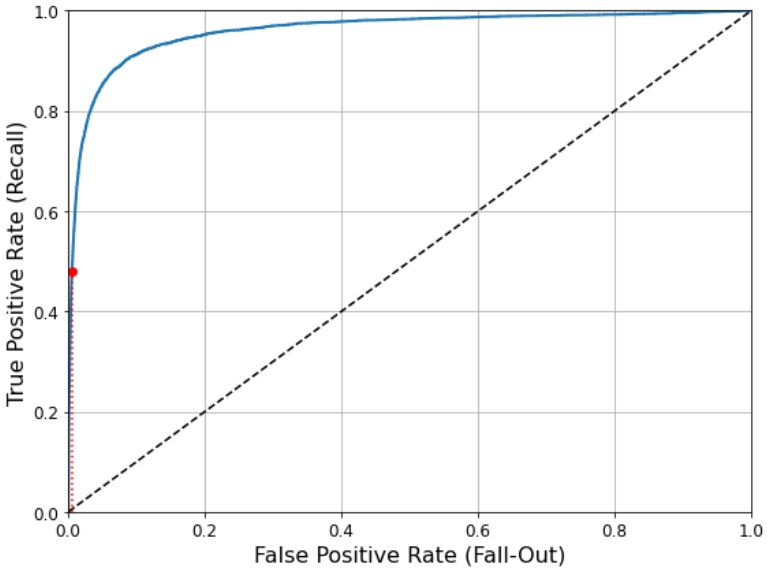
```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_train_5, y_scores)
```

```
0.9604938554008616
```

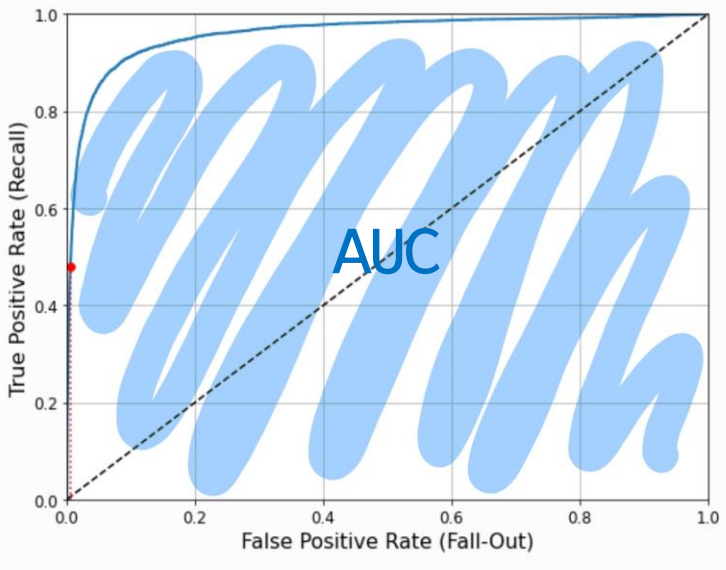
roc_auc_score 함수를 이용해
AUC 계산

3.3.5 ROC 곡선

ROC 곡선



AUC(area under the curve): ROC 곡선의 밑넓이

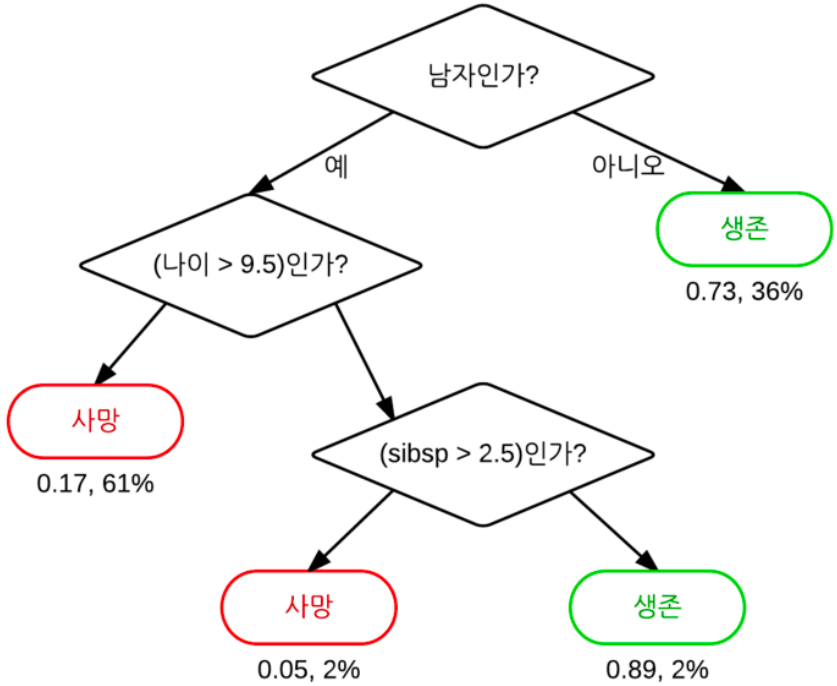


→ROC 곡선이 왼쪽 위 모서리와 가까울수록, AUC가 1에 가까울 수록 성능이 좋음

결정트리(Decision Tree)

결정트리(Decision Tree):

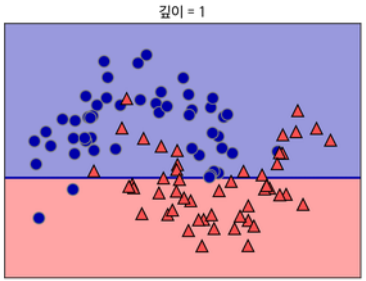
분류와 회귀 모두 가능한 지도 학습 모델 중 하나로,
특정 기준이나 질문에 따라 데이터를 분류하고 예측하여 일련의 규칙을 찾는 알고리즘

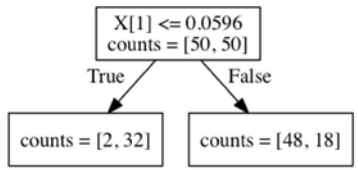


출처https://wooono.tistory.com/104

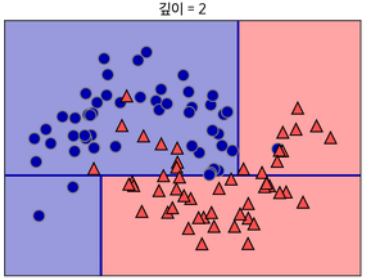
결정트리(Decision Tree)

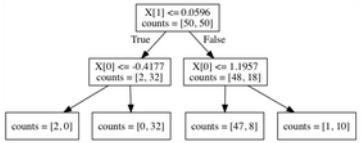
출처https://bkshin.tistory.com/entry/%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-4-%EA%B2%B0%EC%A0%95-%ED%8A%B8%EB%A6%ACDecision-Tree



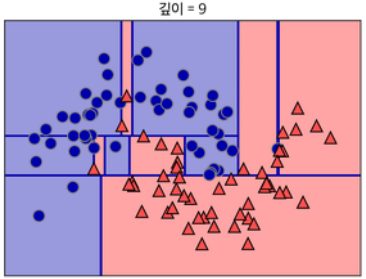


데이터를 잘 구분할 수 있는
질문을 기준으로 나눔





나뉜 범주에서 또다시 데이터를
일련의 질문을 기준으로 나눔

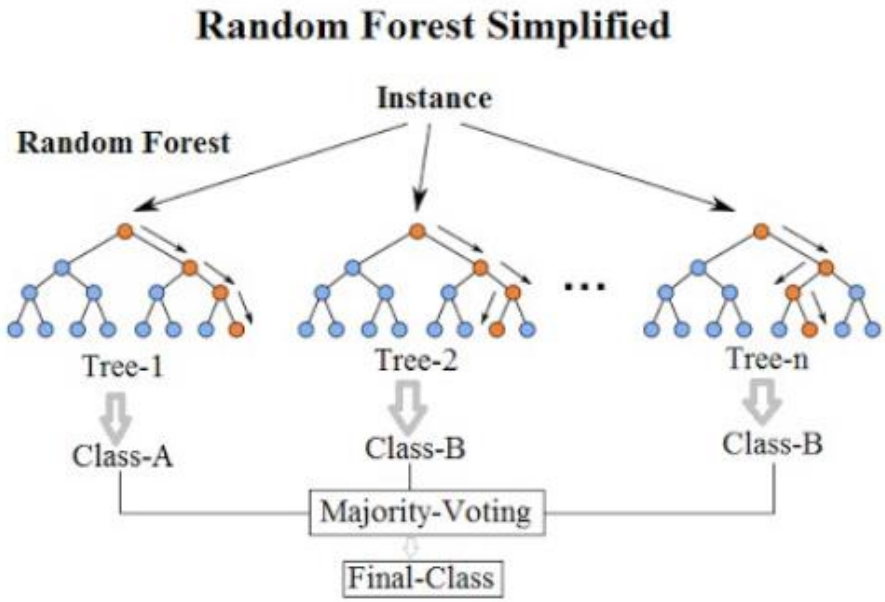


이 과정을 지나치게 많이 하게
되면 과대적합(overfitting) 문제!

랜덤 포레스트(Random Forest):

랜덤 포레스트(Random Forest):

결정 트리의 과대 적합 현상을 개선하기 위해 앙상블 기법으로 고안된 것으로,
여러 개의 결정 트리의 결과들을 모아 최종 결론을 도출하는 알고리즘.



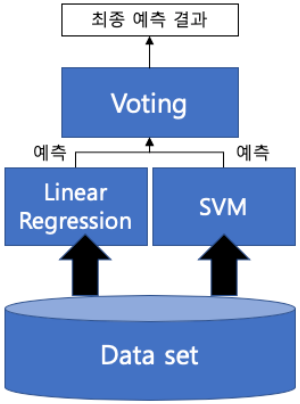
출처https://blog.naver.com/PostView.nhn?blogId=sbd38&logNo=221373436623

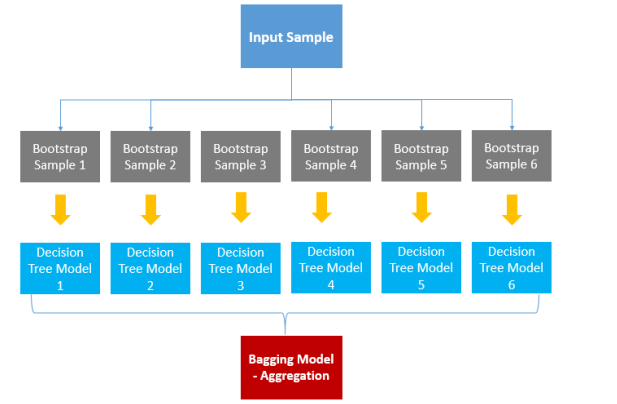
앙상블 기법(Ensemble Learning)

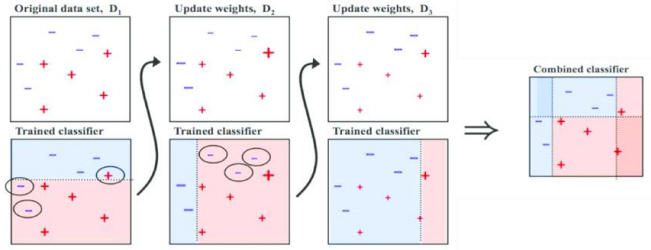
출처http://www.dinnopartners.com/__trashed-4/, swallow.github.io, Medium (Boosting and Bagging explained with examples)

앙상블 기법(Ensemble Learning):

여러 개의 분류기를 생성하고 그 예측을 결합해서 보다 정확한 예측을 도출하는 기법.







보팅(Voting):

여러 개의 분류기에서 출력된 결과를 종합하여 다수결로 최종 결과를 집계하는 방법

배깅(Bagging):

샘플을 여러 번 뽑아 각 모델을 학습시켜 결과물을 집계하는 방법

부스팅(Boosting):

여러 개의 분류기에 가중치를 부가하면서 학습을 진행시키는 방법

3.3.5 ROC 곡선

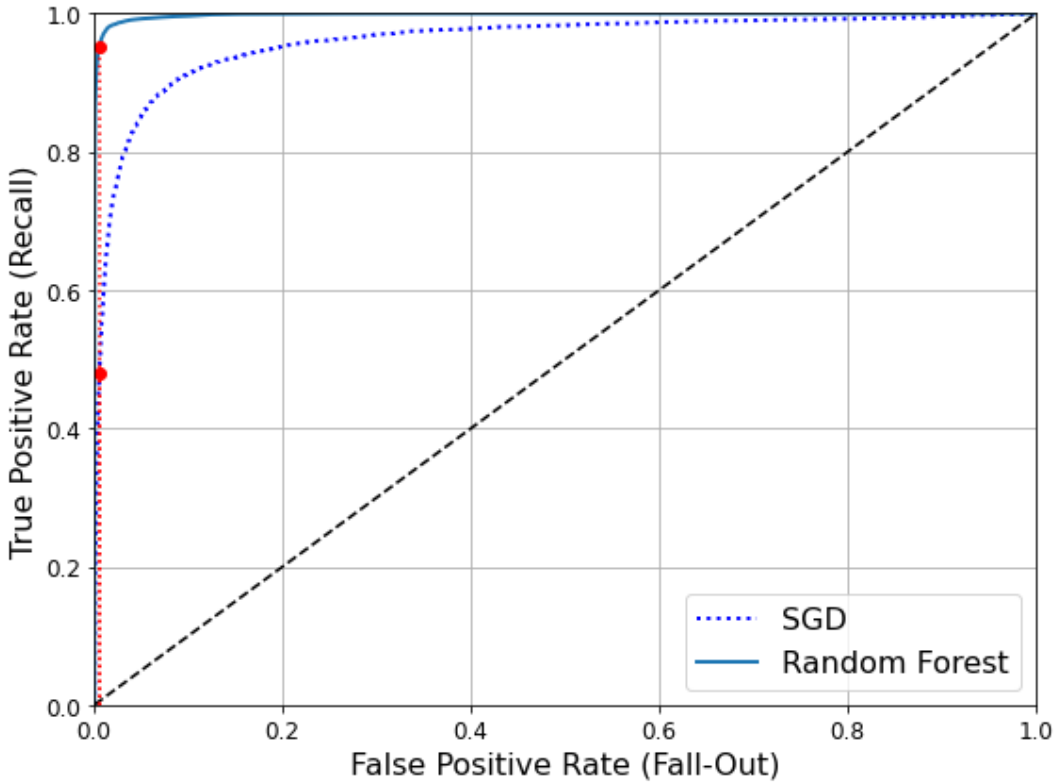
```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(n_estimators=100, random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method="predict_proba")
```

predict_proba()를 통해
랜덤포레스트 샘플 점수 구함

```
y_scores_forest = y_probas_forest[:, 1] # 점수 = 양성 클래스의 확률
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5,y_scores_forest)
```

```
recall_for_forest = tpr_forest[np.argmax(fpr_forest >= fpr_90)]

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, "b:", linewidth=2, label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.plot([fpr_90, fpr_90], [0., recall_90_precision], "r:")
plt.plot([0.0, fpr_90], [recall_90_precision, recall_90_precision], "r:")
plt.plot([fpr_90], [recall_90_precision], "ro")
plt.plot([fpr_90, fpr_90], [0., recall_for_forest], "r:")
plt.plot([fpr_90], [recall_for_forest], "ro")
plt.grid(True)
plt.legend(loc="lower right", fontsize=16)
save_fig("roc_curve_comparison_plot")
plt.show()
```

3.3.5 ROC 곡선

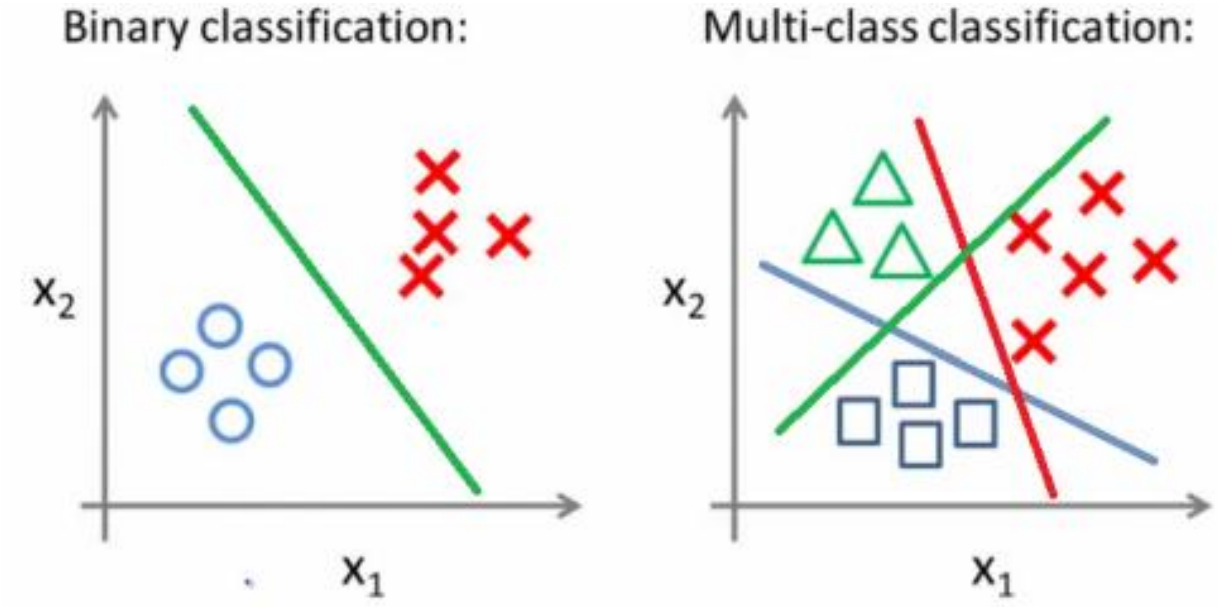


→ RandomForestClassifier의 ROC 곡선이 왼쪽 위 모서리에 더 가까워 SGDClassifier보다 좋음

(AUC 점수도 RandomForestClassifier가 더 높음)

3.4 다중분류

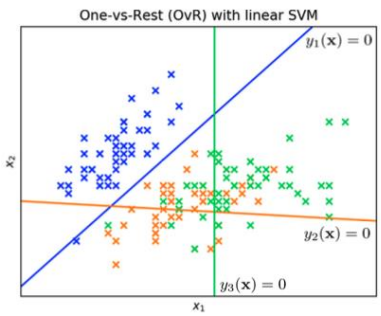
이진분류(Binary Classification): 두 가지 클래스 구별
;로지스틱 회귀, 서포트 벡터 머신 분류기
다중분류(Multiclass Classification): 둘 이상의 클래스 구별
;SGD 분류기, 랜덤 포레스트 분류기, 나이브 베이즈 분류기

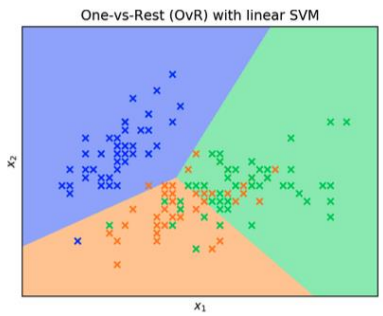


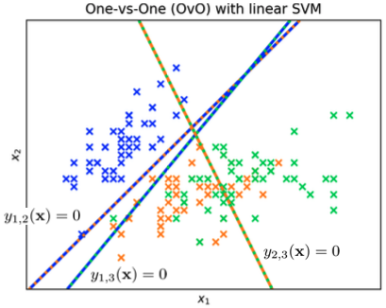
출처 https://medium.com/@b.terryjack/tips-and-tricks-for-multi-class-classification-c184ae1c8ffc

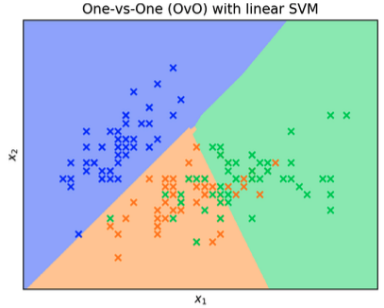
3.4 다중분류

이진분류 여러 개 사용해 다중 클래스 분류:
OvR(one-versus-the-rest), OvO(one-versus-one)









OvR(one-versus-the-rest)

클래스마다 이진 분류기를 만들어서 가장 높은 점수를 낸 클래스 선택

OvO(one-versus-one)

모든 가능한 두 개의 클래스 조합에 대해 이진 분류기를 만들어서, 가장 많이 예측된 클래스 선택

출처 https://medium.com/analytics-vidhya/ml06-intro-to-multi-class-classification-e61eb7492ffd

3.4 다중분류

OvO(one-versus-one)

```
from sklearn.svm import SVC

svm_clf = SVC(gamma="auto", random_state=42)
svm_clf.fit(X_train[:1000], y_train[:1000])
svm_clf.predict([some_digit])
```

```
array([5], dtype=uint8)
```

```
some_digit_scores = svm_clf.decision_function([some_digit])
some_digit_scores
```

```
array([[ 2.81585438,  7.09167958,  3.82972099,  0.79365551,  5.8885703 ,
         9.29718395,  1.79862509,  8.10392157, -0.228207  ,  4.83753243]])
```

```
np.argmax(some_digit_scores)
```

```
5
```

np.argmax()
배열에서 가장 큰 값의 인덱스

OvR(one-versus-the-rest)

```
from sklearn.multiclass import OneVsRestClassifier
ovr_clf = OneVsRestClassifier(SVC(gamma="auto", random_state=42))
ovr_clf.fit(X_train[:1000], y_train[:1000])
ovr_clf.predict([some_digit])
```

```
array([5], dtype=uint8)
```

```
len(ovr_clf.estimators_)
```

len(): 객체의 길이/항목 수

```
10
```

3.4 다중분류

SGD분류기 훈련

```
sgd_clf.fit(X_train, y_train)
sgd_clf.predict([some_digit])
```

```
array([3], dtype=uint8)
```

```
sgd_clf.decision_function([some_digit])
```

```
array([[-31893.03095419, -34419.69069632,  -9530.63950739,
          1823.73154031, -22320.14822878,  -1385.80478895,
        -26188.91070951, -16147.51323997,  -4604.35491274,
        -12050.767298  ]])
```

분류기 평가

```
cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
```

```
array([0.87365, 0.85835, 0.8689 ])
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
```

```
array([0.8983, 0.891 , 0.9018])
```

3.5 에러분석

[사례]
고양이 앱 사용 중 강아지를 고양이로 잘못 분류 오류 발견
⇒ 개 이미지를 잘못 분류 (ex. 현재 시스템 90% 정확도)

-> 시스템이 잘못 분류한 100개의 데이터 수집
-> 개발 데이터 셋 형성
(개 이미지 잘못 분류 + 나머지 오류 분류 모두 포함)
-> 수작업으로 살펴보고, 오류 비율 확인

잘못 분류된 이미지 중 5%만 개 이미지일 경우
: 개 알고리즘 향상 시켜도
최대 전체 오류의 5% 줄여
ex. 향상 후 최대 정확도 90.5%

VS

잘못 분류된 이미지 중 50%가 개 이미지일 경우
: 개 알고리즘 향상 시키면
최대 전체 오류의 50% 줄여
ex. 향상 후 최대 정확도 95%

프로젝트에 대한 투자가 합당한가 : 정량적 기준

3.5 에러분석

에러분석이란,
알고리즘이 잘못 분류하는
개발 데이터셋의 데이터(여기서는 이미지)를 검사하는 프로세스

가정: 가능성이 높은 **모델 선정 후** 모델 **성능 향상 방법** 모색

1. **오차 행렬** 살펴보기

   cross_val_predict( ) -> 예측 생성
   confusion_matrix( ) -> 오차 행렬 함수 호출

2. matshow( ) 함수로 **이미지 시각화**

3. 오차행렬 분석 후 분류기의 성능 향상 방안에 대한 인사이트 얻기

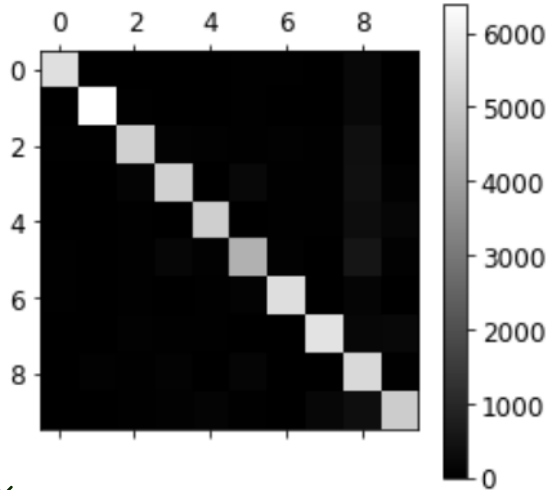
3.5 에러분석

[교재 149pg코드]

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
conf_mx = confusion_matrix(y_train, y_train_pred)
conf_mx
```

```
배열 ([[5577,    0,   22,    5,    8,   43,   36,    6,  225,    1],
       [   0, 6400,   37,   24,    4,   44,    4,    7,  212,   10],
       [  27,   27, 5220,   92,   73,   27,   67,   36,  378,   11],
       [  22,   17,  117, 5227,    2,  203,   27,   40,  403,   73],
       [  12,   14,   41,    9, 5182,   12,   34,   27,  347,  164],
       [  27,   15,   30,  168,   53, 4444,   75,   14,  535,   60],
       [  30,   15,   42,    3,   44,   97, 5552,    3,  131,    1],
       [  21,   10,   51,   30,   49,   12,    3, 5684,  195,  210],
       [  17,   63,   48,   86,    3,  126,   25,   10, 5429,   44],
       [  25,   18,   30,   64,  118,   36,    1,  179,  371, 5107]],
      dtype = int64)
```

```
# 보기 쉬운 형태로 변환
plt.matshow(conf_mx, cmap=plt.cm.gray)
plt.colorbar()
plt.show()
```



실제값 예측값 비교에서는
값이 **클수록, 밝을 수록** 좋은 것

3.5 에러분석

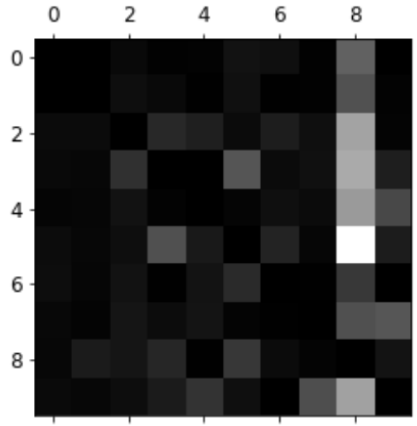
[교재 149pg코드]

```
# 그래프의 에러 부분에 초점을 맞추기
# 오차 행렬의 각 값을 대응되는 클래스의 이미지 개수로 나누어 에러 비율 비교
np.set_printoptions(formatter={'float_kind': lambda x: "{0:0.3f}".format(x)})
```

```
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx=conf_mx/row_sums
np.diagonal(norm_conf_mx) #array([0.942, 0.949, 0.876, 0.853, 0.887, 0.820, 0.938, 0.907, 0.928, 0.858])
np.fill_diagonal(norm_conf_mx,0)
np.diagonal(norm_conf_mx) # array([0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000])
```

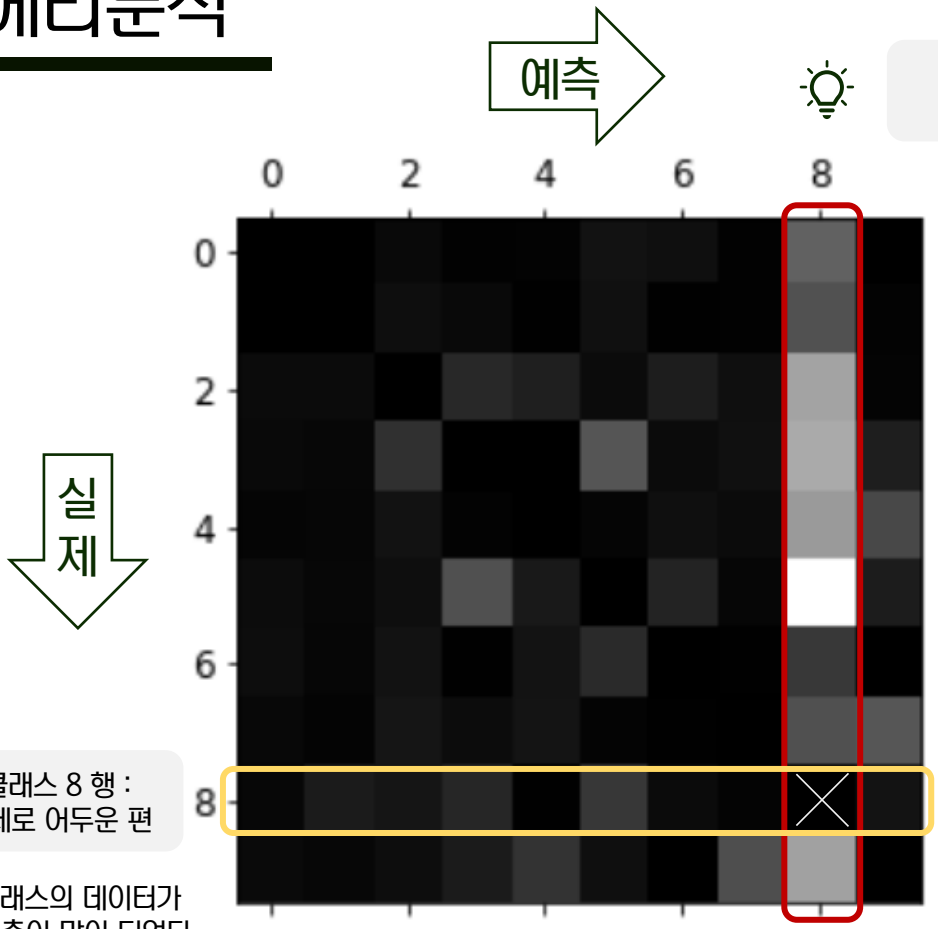
```
[[0.000 0.000 0.004 0.001 0.001 0.007 0.006 0.001 0.038 0.000]
 [0.000 0.000 0.005 0.004 0.001 0.007 0.001 0.001 0.031 0.001]
 [0.005 0.005 0.000 0.015 0.012 0.005 0.011 0.006 0.063 0.002]
 [0.004 0.003 0.019 0.000 0.000 0.033 0.004 0.007 0.066 0.012]
 [0.002 0.002 0.007 0.002 0.000 0.002 0.006 0.005 0.059 0.028]
 [0.005 0.003 0.006 0.031 0.010 0.000 0.014 0.003 0.099 0.011]
 [0.005 0.003 0.007 0.001 0.007 0.016 0.000 0.001 0.022 0.000]
 [0.003 0.002 0.008 0.005 0.008 0.002 0.000 0.000 0.031 0.034]
 [0.003 0.011 0.008 0.015 0.001 0.022 0.004 0.002 0.000 0.008]
 [0.004 0.003 0.005 0.011 0.020 0.006 0.000 0.030 0.062 0.000]]
```

```
plt . matshow ( norm_conf_mx , cmap = plt . cm . gray )
plt.show()
```



에러의 비율
비교에서는
값이 작을 수록,
어두울 수록 좋은 것

3.5 에러분석



클래스 8 열 :
밝은 부분 多

-> 8이 아닌 클래스의 데이터가
8로 예측이 많이 되었다.

클래스 8 행 :
대체로 어두운 편

-> 8인 클래스의 데이터가
8로 예측이 많이 되었다.

인사이트

- 8로 잘못 분류되는 경우 줄여야
- 8처럼 보이는(실제 8은 아닌)
  숫자 훈련데이터로 학습
- 학습할 특성 추가
  ex.
  동심원 count 알고리즘,
  동심원 같은 패턴
  드러나도록 이미지 전처리

3.5 에러분석

(추가) 동심원 count 알고리즘

Detecting Circles in Images using OpenCV and Hough Circles

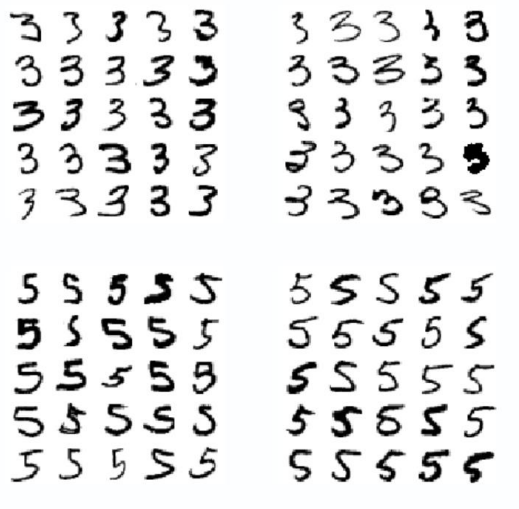
by **Adrian Rosebrock** *on* July 21, 2014

A few days ago, I got an email from a PyImageSearch reader asking about circle detection. See below for the gist:

- `image`: 8-bit, single channel image. If working with a color image, convert to grayscale first.
- `method`: Defines the method to detect circles in images. Currently, the only implemented method is `cv2.HOUGH_GRADIENT`, which corresponds to the **Yuen et al.** paper.
- `dp`: This parameter is the inverse ratio of the accumulator resolution to the image resolution (see Yuen et al. for more details). Essentially, the larger the `dp` gets, the smaller the accumulator array gets.
- `minDist`: Minimum distance between the center *(x, y)* coordinates of detected circles. If the `minDist` is too small, multiple circles in the same neighborhood as the original may be (falsely) detected. If the `minDist` is too large, then some circles may not be detected at all.

3.5 에러분석

```
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]

plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
```



3과 5로 분류된 두 이미지 비교 후 원인 분석

- SGD Classifier 사용

  클래스 점수=
  클래스마다 픽셀에 가중치 할당 후
  새로운 이미지에 대한
  픽셀 강도의 가중치 합

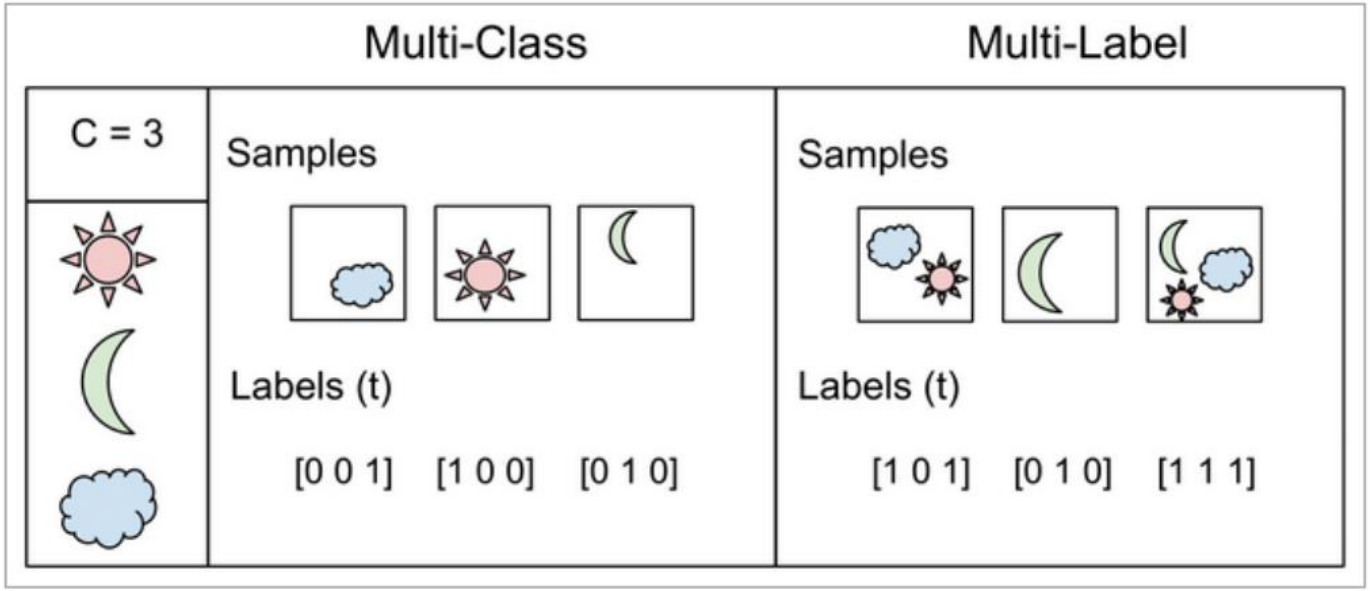
- 문제는 3과 5가 픽셀 수의 유사성

해결방법

분류기가 이미지 위치,회전 방향에 민감한점 이용

->이미지 중앙에 위치시키고
회전 없도록 전처리

3.6 다중 레이블 분류



각 샘플이
하나의 클래스에만 할당됨

각 샘플이
여러 개의 이진 꼬리표 출력

3.6 다중 레이블 분류

```
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train>=7)  # 7이상인 값 (7,8,9)을 large train set으로
y_train_odd = (y_train % 2 ==1 )  # 홀수인 값을 odd train set로 지정
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf=KNeighborsClassifier
knn_clf.fit(X_train, y_multilabel)

knn_clf.predict([some_digit])
# array([[False, True]])
```

올바르게 분류!
숫자 5는 크지 않고
홀수 입니다.

```
# 다중 레이블 분류기의 평가
# 예) 각 레이블의 F1 점수를 구하고 간단하게 평균 점수 계산

y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
f1_score(y_multilabel, y_train_knn_pred, average="macro")

# 0.976410265560605
```

3.6 다중 출력 분류

다중출력분류

다중 레이블 분류의 확장 개념

한 레이블이 다중 클래스가 되도록 일반화 한 것

(즉, 값을 두 개 이상 가질 수 0)

이미지에서 잡음을 제거하는 시스템 예시 [교재 153pg]

- input : 잡음이 많은 숫자 이미지
- output : 깨끗한 숫자 이미지→ 픽셀 강도 배열 출력
- 각 레이블의 값 : 0~255 개의 값 가져

3.6 다중 출력 분류

```
noise = np.random.randint(0,100, (len(X_train), 784))  #noise 생성
X_train_mod = X_train + noise  # 기존 X_train 에 noise 추가

noise = np.random.randint (0, 100, len(X_test), 784))
X_test_mod = X_test+noise # 기존 X_test에 noise 추가

y_train_mod = X_train # 깨끗한 X_train의 상태가 이제 y 값

y_test_mod = X_test   # 깨끗한 X_test의 상태가 이제 x 값
```

```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
```

감사합니다

Q&A