ML-C5

22.07.14

모델평가와 성능향상

비지도학습의 평가는 굉장히 정성적인 일이므로

지도학습 분류와 회귀에 모델 평가를 진행.

Classification – Score Method

```
X, y = make_blobs(random_state=0)

X_train, X_test, y_train , y_test = train_test_split(X,y,random_state=0)
logreg = LogisticRegression().fit(X_train, y_train)
print("test_ac : {:.2f}".format(logreg.score(X_test,y_test)))
```

```
test_ac : 0.88
```

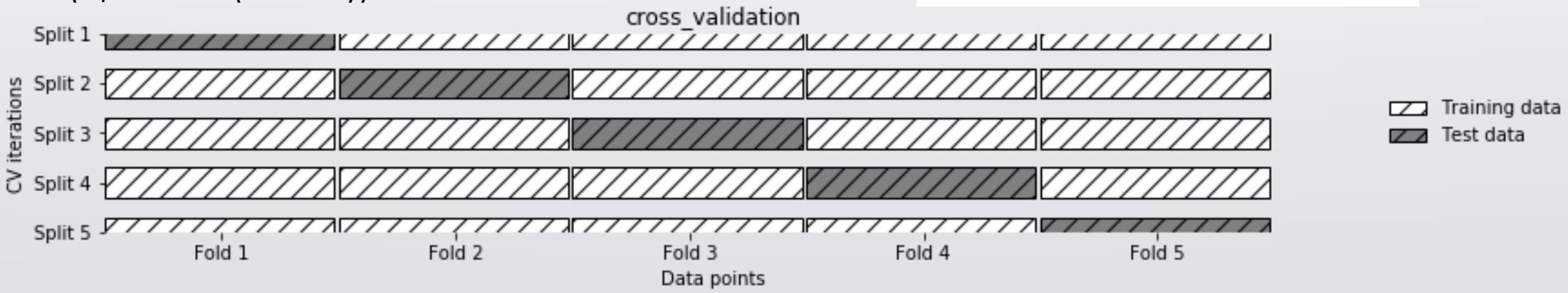
Classification – Cross-Validation(교차검증)

K-겹 교차검증(k-fold cross-validation)
Ex)5일경우. 전체데이터를 5개의 부분집합으로 나누고 1세트는 1번test 2,3,4,5(train)
2세트는 2번test 1,3,4,5(train) … 이런식으로 총 5개의 정확도 값을 얻는다.

```
iris = load_iris()
logreg = LogisticRegression()
scores = cross_val_score(logreg, iris.data,iris.target)
print(scores)
print(np.mean(scores))
```

```
[0.96078431 0.92156863 0.95833333]
#폴드의수는 cv= 매개변수를 통해 변경가능하다. 기본은 3
```

```
0.9600000000000002
```



Classification – Cross-Validation(교차검증)

```
iris = load_iris()
logreg = LogisticRegression()
res= cross_validate(logreg,
iris.data,iris.target,cv=5,return_train_score=True)
res_df = pd.DataFrame(res)
display(res_df)

print(res_df.mean())
```

	fit_time	score_time	test_score	train_score
0	0.003001	0.000000	1.000000	0.950000
1	0.002001	0.000998	0.966667	0.966667
2	0.003001	0.000000	0.933333	0.966667
3	0.001953	0.001000	0.900000	0.975000
4	0.005000	0.000000	1.000000	0.958333

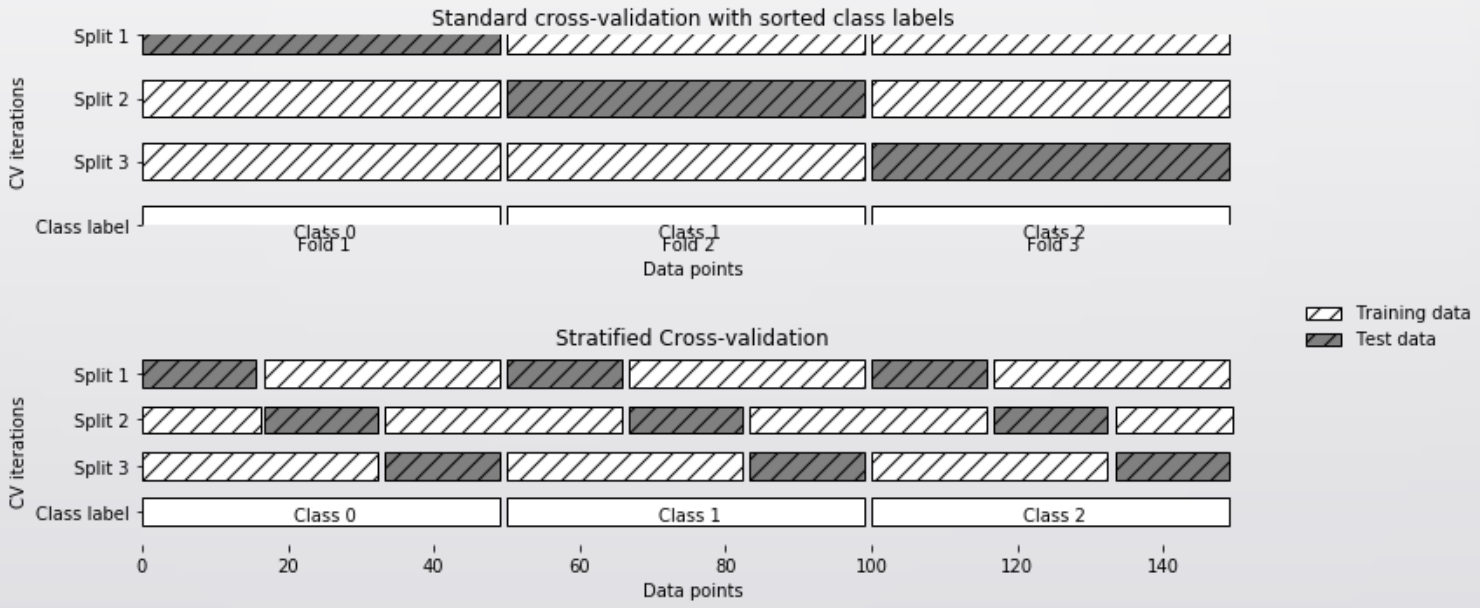
```
fit_time 0.002991 score_time 0.000400 test_score 0.960000 train_score 0.963333 dtype: float64
```

Classification – Cross-Validation(교차검증) 문제점

```
array([0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2])
```

Iris_dataset의 target은 다음과 같이 정렬되어 있어, 교차검증시 같은 클래스만 학습되는 경우가 생길 수 있다. 그래서 나온것이 계층별 k-겹 교차검증 이다.

폴드안의 클래스 비율이 전체 데이터셋의 클래스 비율과 같도록 데이터를 나눈다.



Classification – Cross-Validation(교차검증) 상세옵션

교차검증시 교차 검증 분할기를 전달함으로써 더 세밀하게 분할 할 수 있다.

```
from sklearn.model_selection import KFold
iris = load_iris()
logreg = LogisticRegression()
res= cross_validate(logreg, iris.data,iris.target,cv=5,return_train_score=True)
res_df = pd.DataFrame(res)

kfold = KFold(n_splits=5)
cross_val_score(logreg,iris.data,iris.target,cv=kfold)
```

```
array([1. , 0.93333333, 0.43333333, 0.96666667, 0.43333333])
```

```
from sklearn.model_selection import KFold
iris = load_iris()
logreg = LogisticRegression()
res= cross_validate(logreg, iris.data,iris.target,cv=5,return_train_score=True)
res_df = pd.DataFrame(res)

kfold = KFold(n_splits=3)
cross_val_score(logreg,iris.data,iris.target,cv=kfold)
```

```
array([0., 0., 0.])
```

```
kfold = KFold(n_splits=3,shuffle=True,random_state=0)
cross_val_score(logreg,iris.data,iris.target,cv=kfold)
```

```
array([0.9 , 0.96, 0.96])
```

LOOCV

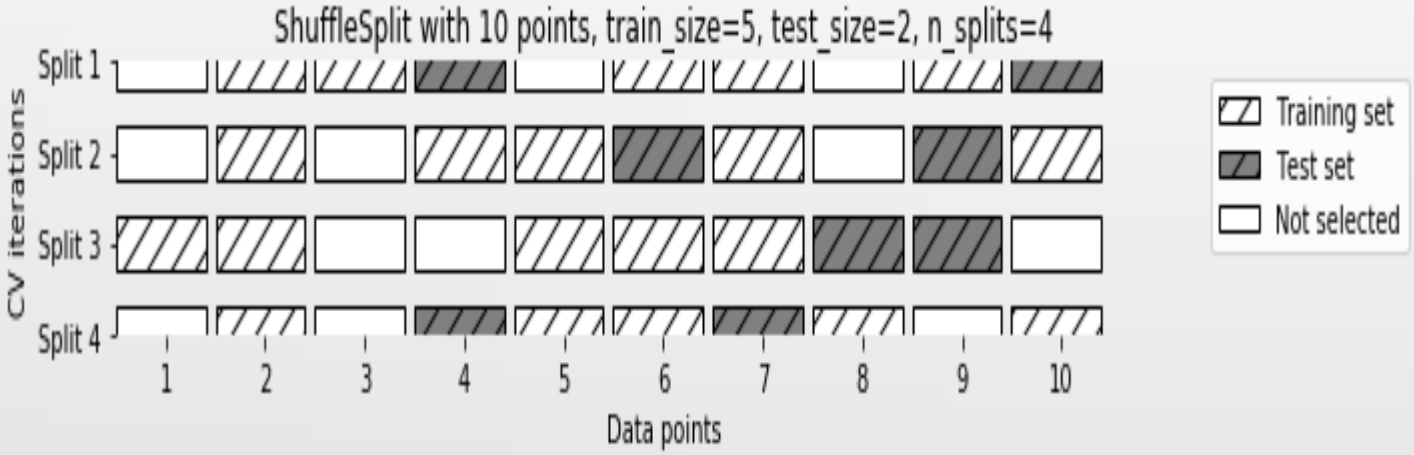
폴드 하나에 샘플 하나만 들어있는 k겹 교차 검증.
각 반복에서 하나의 데이터포인트를 선택해 테스트 세트로 사용.

```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
scores =
cross_val_score(logreg,iris.data,iris.target,cv=loo)
print(len(scores))
print(np.mean(scores))
```

```
150 0.9533333333333334
```

임의 분할 교차 검증(Shuffle-split cross_validation)

Train_size 만큼 훈련세트, test_set만큼 테스트세트로 분할 하는데, n_splits 횟수 만큼 반복됨. (실수로 입력시 비율이고, 정수로는 절대 개수를 지정)
비율지정을 통한 부분샘플링으로 대규모 데이터셋에서 종종 도움이됨.




```
from sklearn.model_selection import ShuffleSplit
shuffle_split = ShuffleSplit(test_size=.5,train_size=.5,n_splits=10)
scores = cross_val_score(logreg,iris.data,iris.target,cv=shuffle_split)
print(scores)
```

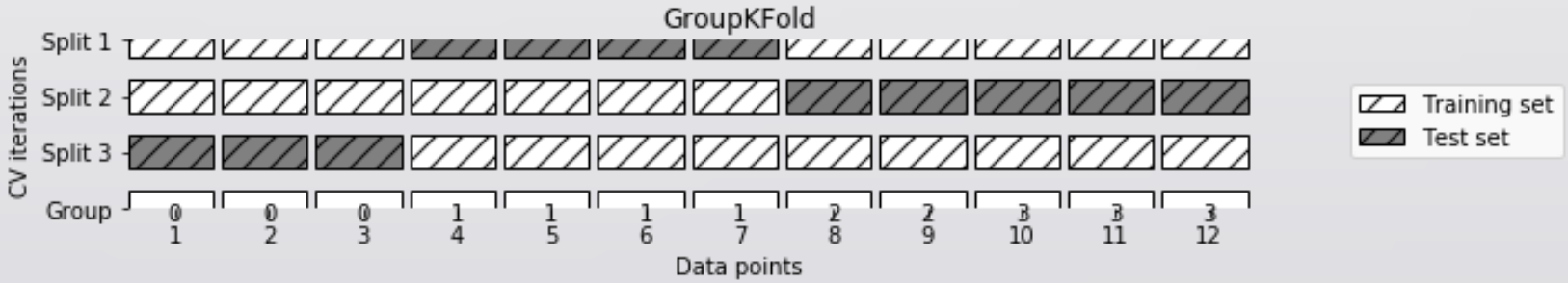
```
[0.96 0.96 0.93333333 0.89333333 0.92 0.97333333 0.92 0.97333333 0.94666667 0.94666667]
```

그룹별 교차 검증

서로다른 그룹으로 학습 및 테스트를 해야하는 경우 , (음성인식, 표정인식등) 사용.

```
from sklearn.model_selection import GroupKFold
X, y = make_blobs(n_samples=12,random_state=0)

groups=[0,0,0,1,1,1,1,2,2,3,3,3]
scores = cross_val_score(logreg,
X,y,groups=groups,cv=GroupKFold(n_splits=3))
print(scores)
```

```
[0.75 0.8 0.66666667]
```



반복 교차 검증

데이터셋의 크기가 크지 않을 경우 안정된 검증 점수를 얻기 위해 교차 검증을 반복하여 여러 번 수행하는 경우가 있다. -> RepeatedKFold(회귀) / RepeatedStratifiedKFold(분류) 분할기가 있다.

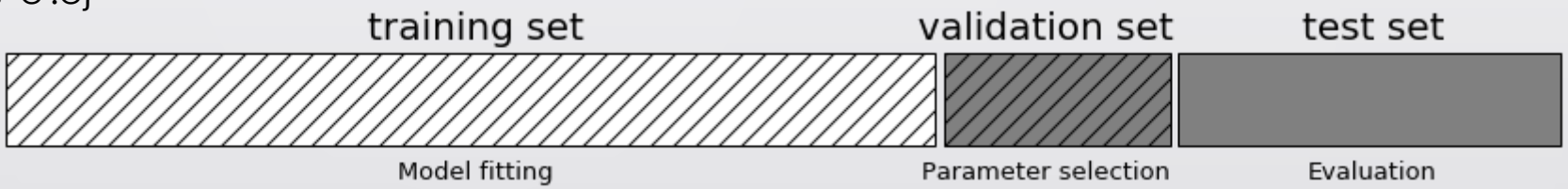
```
from sklearn.model_selection import RepeatedStratifiedKFold
rskfold = RepeatedStratifiedKFold(random_state=42)
scores = cross_val_score(logreg, iris.data,iris.target,cv=rskfold)
np.mean(scores)
```

```
0.9566666666666666
```

그리드 서치

매개변수 별로 가능한 조합을 시도해보는것이다.
데이터셋을 나눌때 훈련, 검증(파라미터찾기), 테스트이렇게 나눈다.

```
best_score =0
X_train, X_test, y_train,y_test =
train_test_split(iris.data,iris.target,random_state=0)
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for c in [0.001, 0.01, 0.1, 1, 10, 100]:
        svm = SVC(gamma=gamma, C=c)
        svm.fit(X_train, y_train)
        score=svm.score(X_test,y_test)
        if score > best_score :
            best_score = score
            best_p = {'gamma ':gamma, 'c':c}

print(best_score)
print(best_p)
```




```
0.9736842105263158 {'gamma ': 0.001, 'c': 100}
```

그리드 서치

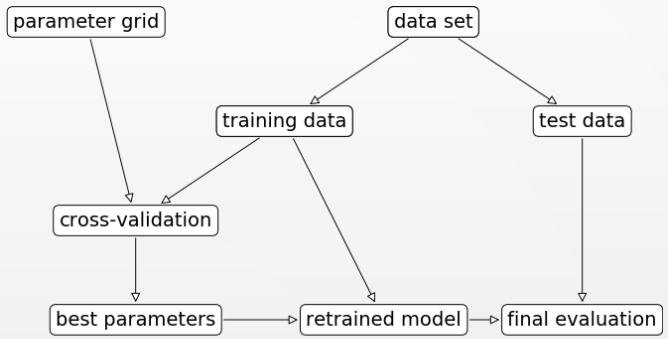
매개변수 별로 가능한 조합을 시도해보는것이다.
데이터셋을 나눌때 훈련, 검증(파라미터찾기), 테스트이렇게 나눈다.

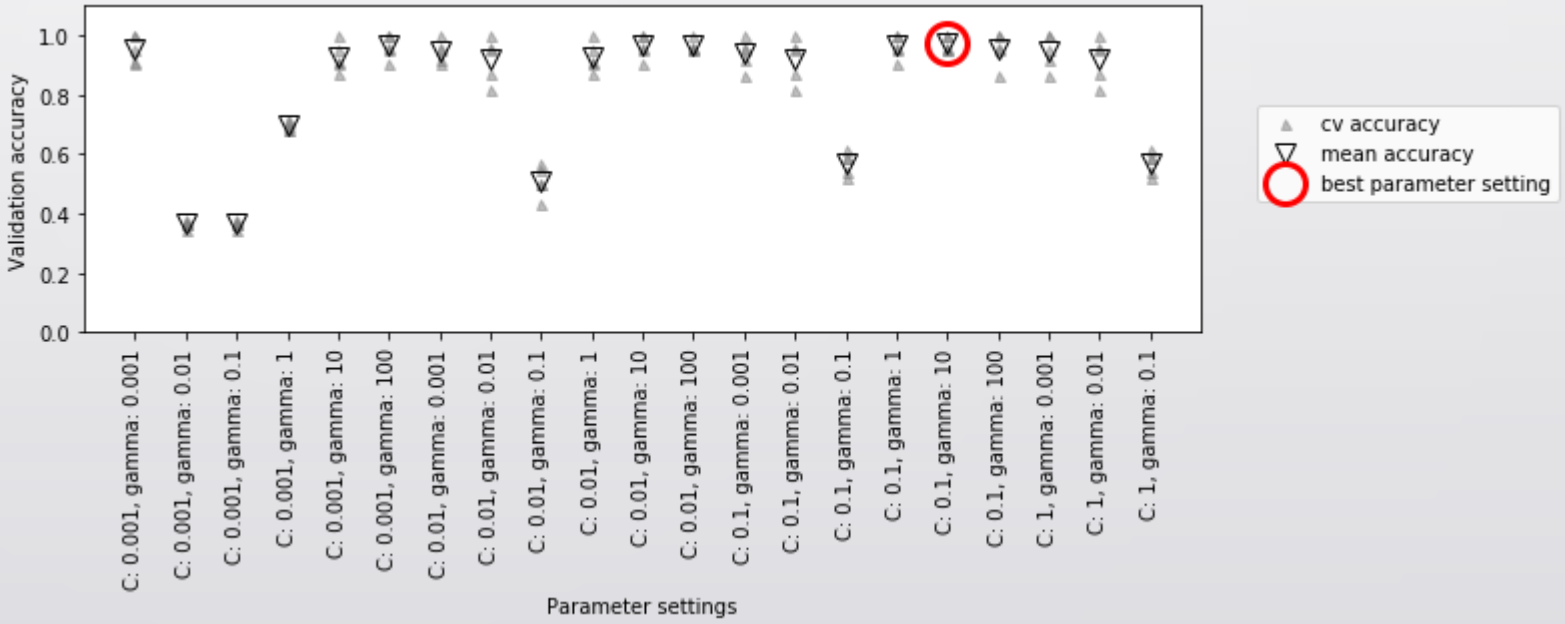
```
X_trainval, X_test, y_trainval, y_test = train_test_split(iris.data,iris.target,random_state=0)
X_train, X_valid, y_train, y_valid = train_test_split(X_trainval,y_trainval,random_state=1)

best_score =0
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        score=svm.score(X_valid,y_valid)
        if score > best_score :
            best_score = score
            best_p = {'C':C,'gamma':gamma }

svm = SVC(** best_p) # 딕셔너리 언패킹
svm.fit(X_trainval, y_trainval)
test_score=svm.score(X_test,y_test)

print("validation_best : {:.2f}".format(best_score))
print("best_combination : ",best_p)
print("optimal_parameter_test_Set_score : {:.2f}".format(test_score))
```

```
validation_best : 0.96 best_combination :
{'C': 10, 'gamma': 0.001}
optimal_parameter_test_Set_score : 0.92
```

교차검증을 사용한 그리드 서치(=교차검증 한다.)

```
X_trainval, X_test, y_trainval, y_test =
train_test_split(iris.data,iris.target,random_state=0)
X_train, X_valid, y_train, y_valid =
train_test_split(X_trainval,y_trainval,random_state=1)

best_score =0
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        score=cross_val_score(svm,X_trainval,y_trainval,cv=5)
        score=np.mean(scores)
        if score > best_score :
            best_score = score
            best_p = {'C':C,'gamma':gamma }

svm = SVC(** best_p) # 딕셔너리 언패킹
svm.fit(X_trainval, y_trainval)
print(best_score)
print(svm.score(X_test,y_test))
```





교차검증을 사용한 그리드 서치(=교차검증 한다.)

Sklearn -> GridSearchCV가 구현되어있다. 파라미터로, 매개변수의 딕셔너리 형태를 받기 때문에 미리 딕셔너리형태로 선언해야한다.

```
param_grid = {'gamma' : [0.001, 0.01, 0.1, 1, 10,
100],'C' : [0.001, 0.01, 0.1, 1, 10, 100]}
grid_search =
GridSearchCV(SVC(),param_grid,cv=5,return_train_s
core=True)

X_train , X_test, y_train , y_test =
train_test_split(iris.data, iris.target,random_state=0)

grid_search.fit(X_train,y_train )
print(grid_search.score(X_test,y_test))
```

```
0.9736842105263158
```

```
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma=0.01,
kernel='rbf', max_iter=-1, probability=False,
random_state=None, shrinking=True, tol=0.001,
verbose=False)
```

교차검증 결과 분석

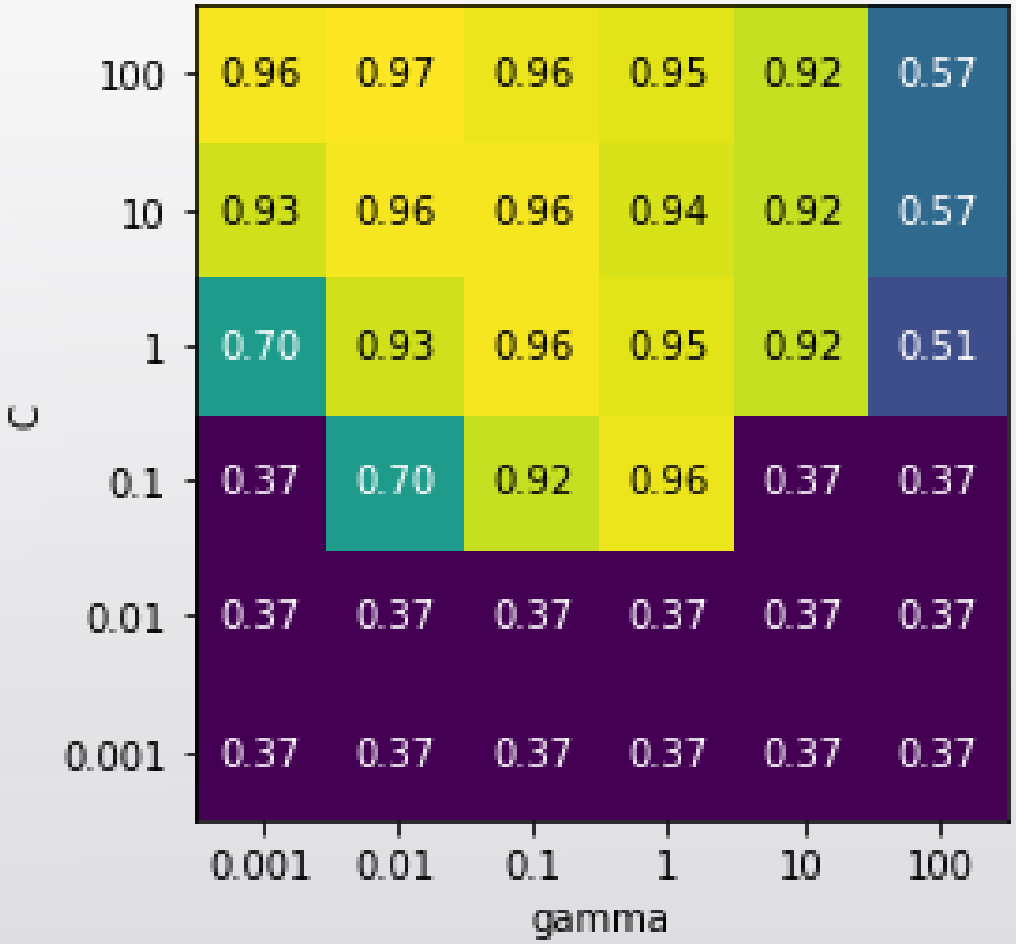
Sklearn -> GridSearchCV가 구현되어있다. 파라미터로, 매개변수의 딕셔너리 형태를 받기 때문에 미리 딕셔너리형태로 선언해야한다.

```
pd.set_option('display.max_columns',None)
results = pd.DataFrame(grid_search.cv_results_)
display(np.transpose(results.head()))
```

	0	1	2	3	4	
mean_fit_time	0.00120368	0.00139656	0.000399733	0.00079999	0.000803041	
std_fit_time	0.000755131	0.000796128	0.000489571	0.000399996	0.000401558	
mean_score_time	0.000796843	0.00020442	0.000797462	0.00019722	0.000399542	
std_score_time	0.000398577	0.00040884	0.00074475	0.00039444	0.000489337	
param_C	0.001	0.001	0.001	0.001	0.001	
param_gamma	0.001	0.01	0.1	1	10	
params	{'C': 0.001, 'gamma': 0.001}	{'C': 0.001, 'gamma': 0.01}	{'C': 0.001, 'gamma': 0.1}	{'C': 0.001, 'gamma': 1}	{'C': 0.001, 'gamma': 10}	
split0_test_score	0.375	0.375	0.375	0.375	0.375	
split1_test_score	0.347826	0.347826	0.347826	0.347826	0.347826	
split2_test_score	0.363636	0.363636	0.363636	0.363636	0.363636	
split3_test_score	0.363636	0.363636	0.363636	0.363636	0.363636	
split4_test_score	0.380952	0.380952	0.380952	0.380952	0.380952	
mean_test_score	0.366071	0.366071	0.366071	0.366071	0.366071	
std_test_score	0.0113708	0.0113708	0.0113708	0.0113708	0.0113708	
rank_test_score	22	22	22	22	22	
split0_train_score	0.363636	0.363636	0.363636	0.363636	0.363636	
split1_train_score	0.370787	0.370787	0.370787	0.370787	0.370787	
split2_train_score	0.366667	0.366667	0.366667	0.366667	0.366667	
split3_train_score	0.366667	0.366667	0.366667	0.366667	0.366667	
split4_train_score	0.362637	0.362637	0.362637	0.362637	0.362637	
mean_train_score	0.366079	0.366079	0.366079	0.366079	0.366079	
std_train_score	0.00285176	0.00285176	0.00285176	0.00285176	0.00285176	

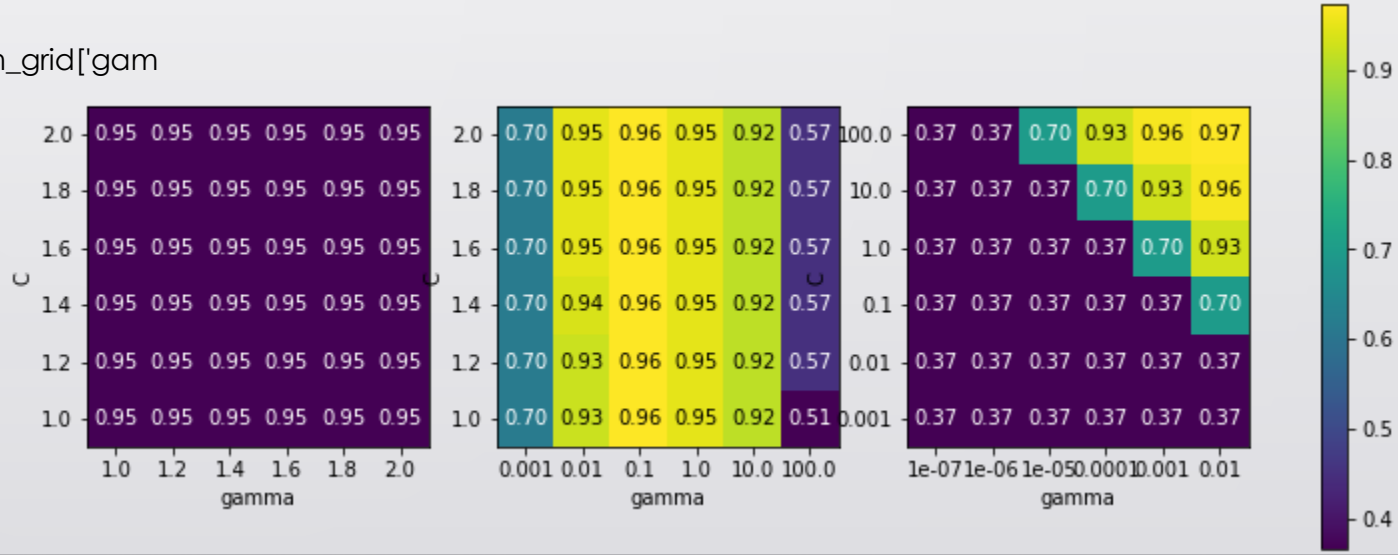
교차검증 결과 분석 파라미터 히트맵 띄우기

```
pd.set_option('display.max_columns',None)
results = pd.DataFrame(grid_search.cv_results_)
scores = np.array(results.mean_test_score).reshape(6,6)

mglearn.tools.heatmap(scores,xlabel='gamma',xticklabels=param_g
rid['gamma'],ylabel='C'
                      ,yticklabels=param_grid['C'],cmap="viridis")
```



교차검증 결과 분석 파라미터 히트맵 띄우기

```
pd.set_option('display.max_columns',None)
results = pd.DataFrame(grid_search.cv_results_)
scores = np.array(results.mean_test_score).reshape(6,6)

fig, axes = plt.subplots(1,3,figsize=(13,5))

param_grid_linear = {'C' : np.linspace(1,2,6),'gamma' : np.linspace(1,2,6)}
param_grid_one_log = {'C' : np.linspace(1,2,6),'gamma' : np.logspace(-3,2,6)}
param_grid_range={'C':np.logspace(-3,2,6),'gamma':np.logspace(-7,-2,6)}

for param_grid, ax in zip([param_grid_linear,
param_grid_one_log,param_grid_range],axes):
    grid_search = GridSearchCV(SVC(),param_grid,cv=5)
    grid_search.fit(X_train, y_train)
    scores = grid_search.cv_results_['mean_test_score'].reshape(6,6)

    scores_image =
mglearn.tools.heatmap(scores,xlabel='gamma',ylabel='C',xticklabels=param_grid['gam
ma'],yticklabels=param_grid['C'],cmap="viridis",ax=ax)
plt.colorbar(scores_image, ax=axes.tolist())
```



비대칭 매개변수 그리드 탐색

SVC 같은 경우 Kernel매개변수에 따라 하이퍼파라미터 종류가 달라진다.
Linear -> C / RBF -> C, gamma : 이는 딕셔너리안의 리스트로 만들어주면 된다.

```
[{'kernel': ['rbf'],
'gamma': [0.001, 0.01,
0.1, 1, 10, 100], 'C':
[0.001, 0.01, 0.1, 1, 10,
100]}, {'kernel':
['linear'], 'C': [0.001,
0.01, 0.1, 1, 10, 100]}]
```

```
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'} 0.9732142857142857
```

그리드서치에 다양한 교차검증적용 ->중첩교차검증

훈련셋과 데이터셋을 5번 분할하여 각각의 테스트성능의 평균을 알 수 있다.

Test / Train : 5번 분할
Train : 5번 분할 및 GridSearch(36)

Train에서의 분할해서 검증세트의 성능을 구한 것은 GridSearch의 목적인 best_parameter를 얻기 위함.

```
param_grid = {'gamma' : [0.001, 0.01, 0.1, 1, 10, 100],'C' : [0.001, 0.01, 0.1, 1, 10, 100]}
```

```
scores =
cross_val_score(GridSearchCV(SVC(),param_grid,cv=5),iris.data,iris.target,cv=5)

print(scores)
print(scores.mean())
```

```
[0.96666667 1. 0.96666667 0.96666667 1. ] 0.9800000000000001
# iris데이터셋에서 SVC는 평균 교차 검증 정확도가 98퍼센트이다.
```

평가지표와 측정

지금까지 분류 성능 평가에 정확도를 사용했고, 회귀 성능평가에는 R^2를 사용.

그외 성능평가 방법돈 존재한다.

최종목표를 기억

이진분류의 평가지표 : TruePositive FP Fn TrueNegative

불균형 데이터셋 : 클릭데이터 예측 -> 100개광고 뜨는데 1번클릭이면 클릭안함이 99개.

평가지표와 측정

```
from sklearn.dummy import DummyClassifier
digits = load_digits()
y = digits.target==9

X_train , X_test , y_train ,y_test = train_test_split(digits.data, y,random_state=0)
dummy_majority =
DummyClassifier(strategy='most_frequent').fit(X_train,y_train)
pred_most_fre = dummy_majority.predict(X_test)
print(np.unique(pred_most_fre))
print(dummy_majority.score(X_test,y_test))
```

```
[False] 0.8955555555555555
```

평가지표와 측정

```
from sklearn.dummy import DummyClassifier
digits = load_digits()
y = digits.target==9

X_train , X_test , y_train ,y_test = train_test_split(digits.data, y,random_state=0)
dummy_majority = DummyClassifier(strategy='most_frequent').fit(X_train,y_train)
pred_most_fre = dummy_majority.predict(X_test)
print(np.unique(pred_most_fre))
print(dummy_majority.score(X_test,y_test))
```

```
[False] 0.8955555555555555
```

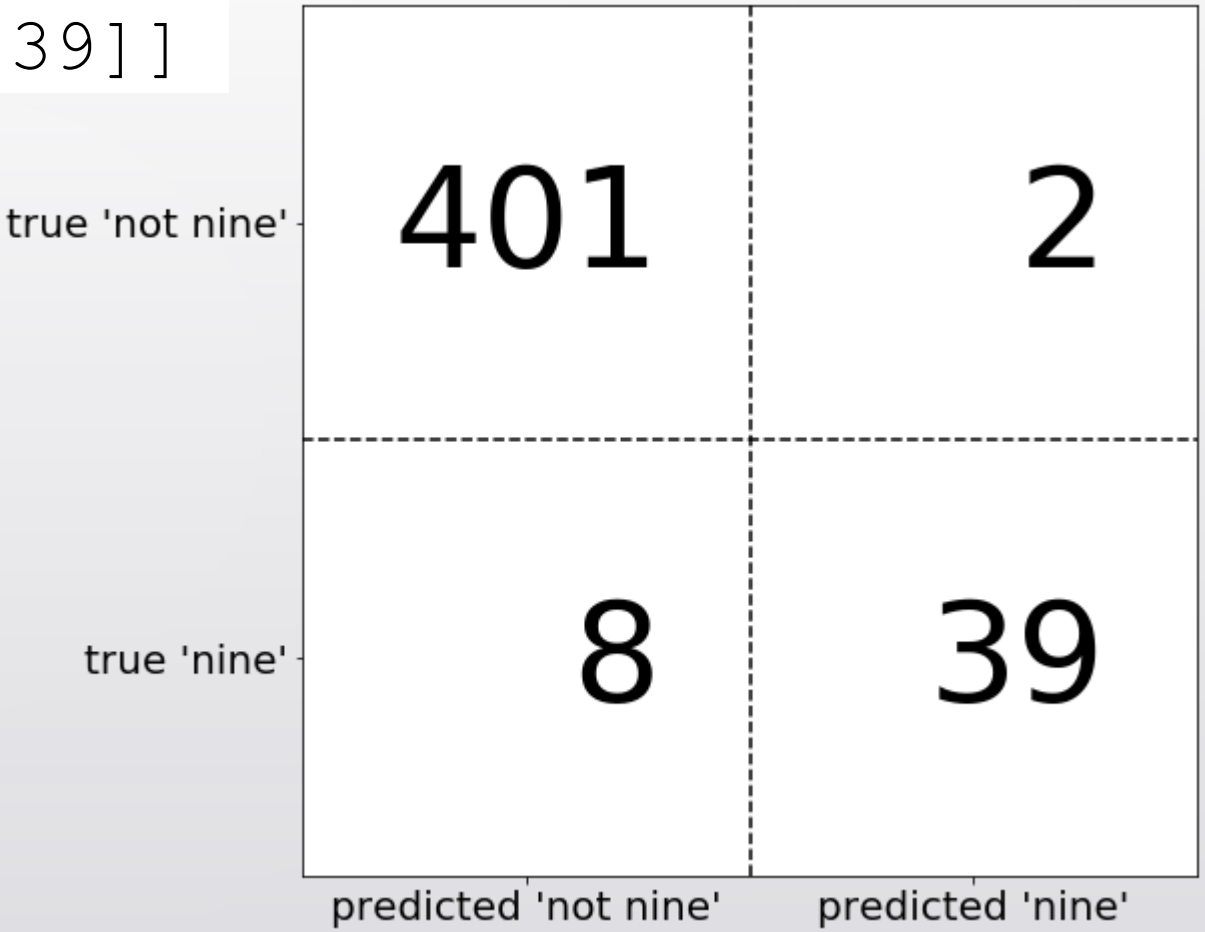
```
digits = load_digits()
y = digits.target==9

X_train , X_test , y_train ,y_test = train_test_split(digits.data, y,random_state=0)
tree = DecisionTreeClassifier(max_depth=2).fit(X_train,y_train)
print("tree :",tree.score(X_test,y_test))
dum = DummyClassifier().fit(X_train,y_train)
print("dummy :",dum.score(X_test,y_test))
logi = LogisticRegression(C=0.1).fit(X_train,y_train)
print("LR :",logi.score(X_test,y_test))
```

```
tree : 0.9177777777777778 dummy : 0.8266666666666667 LR : 0.9777777777777777
```

오차행렬(Confusion_Matrix)

```
digits = load_digits()
y = digits.target==9

X_train , X_test , y_train ,y_test = train_test_split(digits.data,
y,random_state=0)

logi = LogisticRegression(C=0.1).fit(X_train,y_train)
pred_lr = logi.predict(X_test)
confusion = confusion_matrix(y_test, pred_lr)
print(confusion)
```

```
[[401  2]
[  8 39]]
```




```
most_frequent [[403 0] [ 47 0]]
dum [[361 42] [ 42 5]]
logi [[401 2] [ 8 39]]
```

오차행렬(Confusion_Matrix) -> 정확도, 정밀도, 재현율

정확도와의 관계

정확도 = (TP + TN) / (TP + TN+FP+FN)

정밀도 , 재현율

정밀도 = TP/(TP + FP) # 진짜 양성인 친구들의 비율

->FP의 비중을 줄이고 자할 때 사용한다.

재현율 = TP/(TP + FN)

->모든 양성 샘플을 식별해야할 때 성능지표로 사용

정밀도와 재현율은 반비례관계이다.

오차행렬(Confusion_Matrix) -> f-점수

F = 2 * ((정밀도 * 재현율) / (정밀도 + 재현율))

```
digits = load_digits()
y = digits.target==9

X_train , X_test , y_train ,y_test = train_test_split(digits.data, y,random_state=0)
dummy = DummyClassifier(strategy='most_frequent').fit(X_train,y_train)
most_freq_pred=dummy.predict(X_test)
print("freq : ",f1_score(y_test,most_freq_pred))
y_pred_tree = tree.predict(X_test)
print("tree : ",f1_score(y_test,y_pred_tree))
dum = DummyClassifier().fit(X_train,y_train)
y_pred_dum = dum.predict(X_test)
print("dum : ",f1_score(y_test,y_pred_dum))

logi = LogisticRegression(C=0.1).fit(X_train,y_train)
y_pred_logi = logi.predict(X_test)
print("logistic : ",f1_score(y_test,y_pred_logi))
```

```
precision recall f1-score support 9x 0.90
1.00 0.94 403 9 0.00 0.00 0.00 47 accuracy
0.90 450 macro avg 0.45 0.50 0.47 450
weighted avg 0.80 0.90 0.85 450
```

```
freq : 0.0 tree : 0.5542168674698795 dum : 0.13636363636363635 logistic : 0.8863636363636364
```

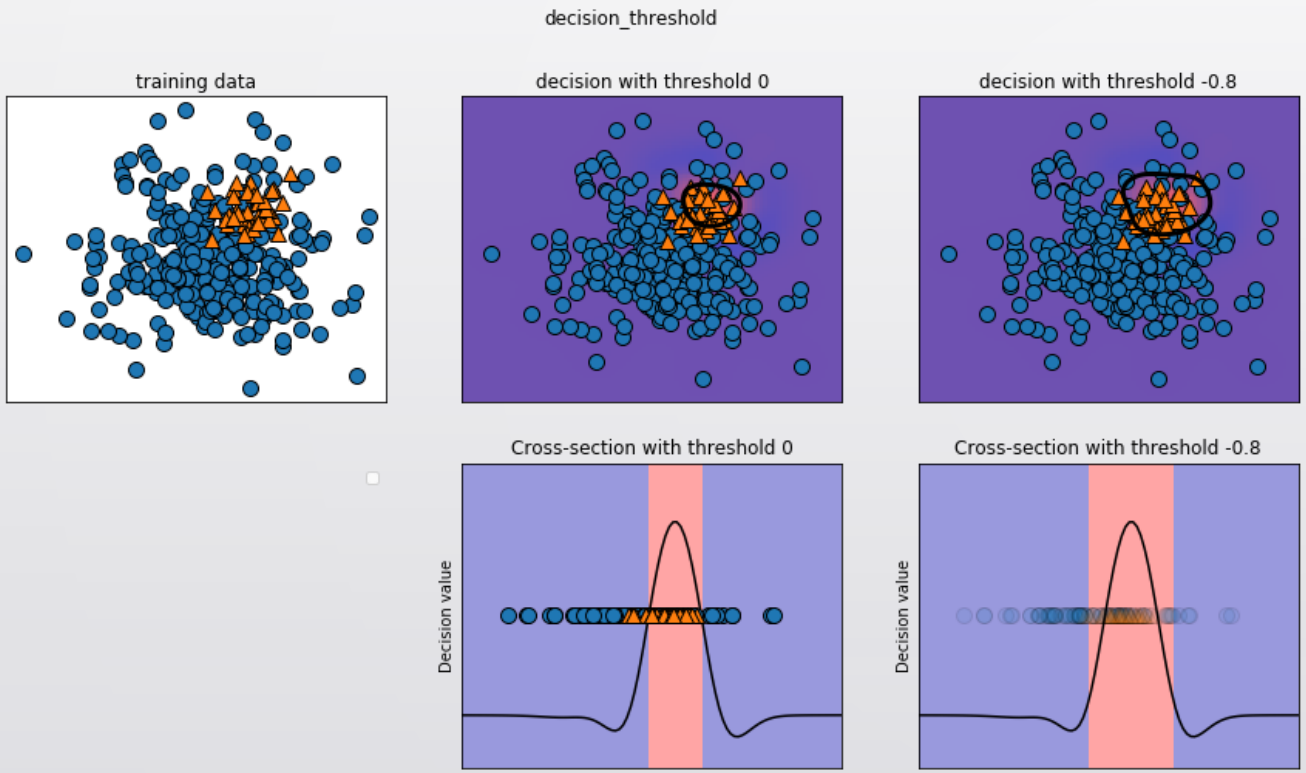
오차행렬(Confusion_Matrix) -> classification_report

```
dum :                precision    recall  f1-score   support

          9x       0.91      0.91      0.91       403
           9       0.21      0.21      0.21        47

    accuracy                           0.84       450
   macro avg       0.56      0.56      0.56       450
weighted avg       0.84      0.84      0.84       450


tree :                precision    recall  f1-score   support

          9x       0.94      0.97      0.95       403
           9       0.64      0.49      0.55        47

    accuracy                           0.92       450
   macro avg       0.79      0.73      0.75       450
weighted avg       0.91      0.92      0.91       450


logistic :                precision    recall  f1-score   support

          9x       0.98      1.00      0.99       403
           9       0.95      0.83      0.89        47

    accuracy                           0.98       450
   macro avg       0.97      0.91      0.94       450
weighted avg       0.98      0.98      0.98       450
```

```
from sklearn.metrics import classification_report
print("dum : ",classification_report(y_test,y_pred_dum,target_names=["9x","9"]))
print()
print("tree : ",classification_report(y_test,y_pred_tree,target_names=["9x","9"]))
print()
print("logistic : ",classification_report(y_test,y_pred_logi,target_names=["9x","9"]))
```

오차행렬(Confusion_Matrix) -> 불확실성 고려

Ex) 음성 400개와 양성 50개의 불균형 이진 데이터 분률 셋.




```
X, y = make_blobs(n_samples=(400,50),cluster_std=[7,2],random_state=22)
X_train , X_test , y_train, y_test = train_test_split(X,y,random_state=0)
svc = SVC(gamma=.05).fit(X_train,y_train)

y_pred_lower_th = svc.decision_function(X_test) >-.8
print(classification_report(y_test,y_pred_lower_th))
```

	precision	recall	f1-score	support
0	1.00	0.82	0.90	104
1	0.32	1.00	0.49	9
accuracy			0.83	113
macro avg	0.66	0.91	0.69	113
weighted avg	0.95	0.83	0.87	113

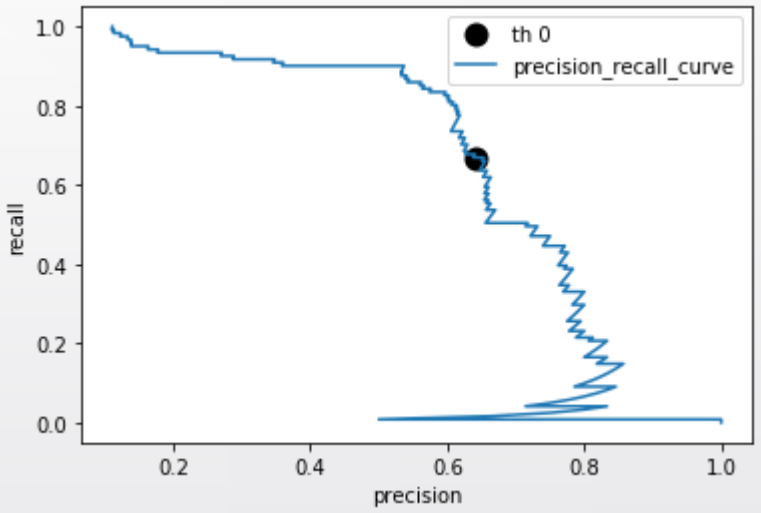
정밀도-재현율 곡선과 ROC곡선(Support_Vector_Classification

모델 분류작업을 결정하는 임계값을 바꾸는 것은 분류기의 정밀도와 재현율의 상충관계를 조정하는 것이다.

90% 재현율을 맞추어라! → 운영 포인트(Operating Point)를 지정한다고 말한다.

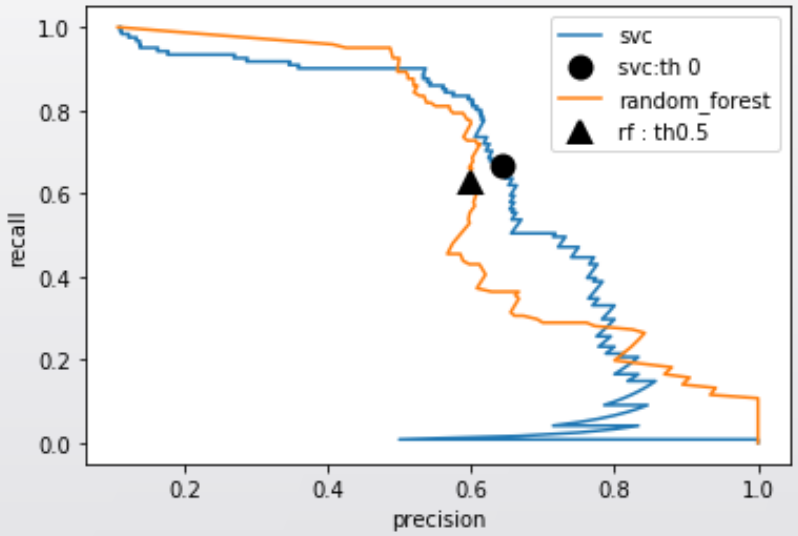
하지만 새로운 모델을 만들때는 운영포인트라는 것이 없기 때문에 일일히 확인 해야한다.
→ 정밀도-재현율 곡선(Precision-Recall Curve)를 사용한다.

```
from sklearn.metrics import precision_recall_curve
X, y = make_blobs(n_samples=(4000,500),cluster_std=[7,2],random_state=22)
X_train , X_test , y_train, y_test = train_test_split(X,y,random_state=0)
svc = SVC(gamma=.05).fit(X_train,y_train)
precision, recall, thresholds =
precision_recall_curve( y_test,svc.decision_function(X_test))
close_zero = np.argmin(np.abs(thresholds))
plt.plot(precision[close_zero],recall[close_zero],'o',markersize=10,label="th
0",c='k',mew=2)
plt.plot(precision, recall,label="precision_recall_curve")
plt.xlabel("precision")
plt.ylabel("recall")
plt.legend(loc="best")
```



정밀도-재현율 곡선과 ROC곡선(Random_forest)

```
rf = RandomForestClassifier(n_estimators=100,random_state=0,max_features=2)
rf.fit(X_train, y_train)

svc = SVC(gamma=.05).fit(X_train,y_train)
precision_rf, recall_rf,thresholds_rf =
precision_recall_curve(y_test,rf.predict_proba(X_test)[:,1])

precision, recall, thresholds =
precision_recall_curve( y_test,svc.decision_function(X_test))
close_zero = np.argmin(np.abs(thresholds))
plt.plot(precision, recall,label="svc")
plt.plot(precision[close_zero],recall[close_zero],'o',markersize=10,label="svc:th
0",c='k',mew=2)
plt.plot(precision_rf,recall_rf,label="random_forest")
close_default_rf = np.argmin(np.abs(thresholds_rf-0.5))
plt.plot(precision_rf[close_default_rf],recall_rf[close_default_rf],'^',markersize=10,lab
el="rf : th0.5",c='k',mew=2)
plt.xlabel("precision")
plt.ylabel("recall")
plt.legend(loc="best")
```



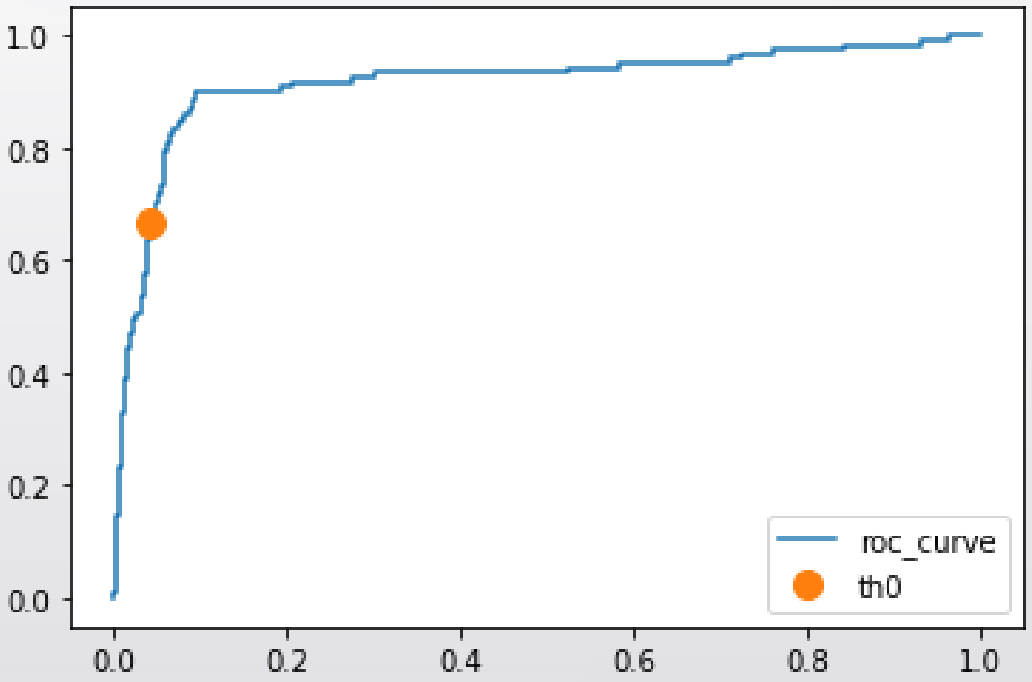
Random_forest f1_score

```
X, y = make_blobs(n_samples=(4000,500),cluster_std=[7,2],random_state=22)
X_train , X_test , y_train, y_test = train_test_split(X,y,random_state=0)
rf = RandomForestClassifier(n_estimators=100,random_state=0,max_features=2)
rf.fit(X_train, y_train)

svc = SVC(gamma=.05).fit(X_train,y_train)
precision_rf, recall_rf,thresholds_rf =
precision_recall_curve(y_test,rf.predict_proba(X_test)[:,1])

precision, recall, thresholds =
precision_recall_curve( y_test,svc.decision_function(X_test))

print("random_forest_f1_score : ",f1_score(y_test,rf.predict(X_test)))
print("svc_forest_f1_score : ",f1_score(y_test,svc.predict(X_test)))
```

```
random_forest_f1_score : 0.6097560975609757 svc_forest_f1_score : 0.6558704453441295
```

ROC and AUC

```
from sklearn.metrics import roc_curve
X, y = make_blobs(n_samples=(4000,500),cluster_std=[7,2],random_state=22)
X_train , X_test , y_train, y_test = train_test_split(X,y,random_state=0)
svc = SVC(gamma=.05).fit(X_train,y_train)
fpr, tpr , ths = roc_curve(y_test,svc.decision_function(X_test))

plt.plot(fpr,tpr,label="roc_curve")
print(ths.shape)
close_zero = np.argmin(np.abs(ths))
plt.plot(fpr[close_zero],tpr[close_zero],'o',markersize=10,label='th0')
plt.legend(loc=4)
```



다중분류의 평가지표

```
digits = load_digits()
X_train , X_test , y_train, y_test =
train_test_split(digits.data,digits.target,random_state
=0)
lr =
LogisticRegression(solver='liblinear',multi_class='ovr').fi
t(X_train,y_train)
pred = lr.predict(X_test)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
```

```
0.9533333333333334
[[37 0 0 0 0 0 0 0 0 0]
[ 0 39 0 0 0 0 2 0 2 0]
[ 0 0 41 3 0 0 0 0 0 0]
[ 0 0 1 43 0 0 0 0 0 1]
[ 0 0 0 0 38 0 0 0 0 0]
[ 0 1 0 0 0 47 0 0 0 0]
[ 0 0 0 0 0 0 52 0 0 0]
[ 0 1 0 1 1 0 0 45 0 0]
[ 0 3 1 0 0 0 0 0 43 1]
[ 0 0 0 1 0 1 0 0 1 44]]
```