

180709

머신러닝 ~ 파이썬

-----목적-----

1. 범주형 자료 '분류'

2. 연속형 자료 '예측'

descriptive vs predictive

descriptive -- 기술통계 ~ clustering도 포함

predictive -- 예측(설명과 추론과는 다름)

scikit-learn

- NumPy, Scipy, matplotlib, IPython 등 필요 (아나콘다 설치 권장)
- 알고리즘마다 매개변수가 다름

Scipy

- 과학 계산을 함수를 모아놓은 파이썬 패키지
- 고성능 선형 대수, 함수 최적화, 신호 처리, 특수한 수학 함수와 통계 분포 등을 포함한 많은 기능 제공
- scikit-learn 은 알고리즘을 구현할 때 SciPy 의 여러 함수를 사용

mglearn

- 그래프나 데이터 적재와 관련한 세세한 코드를 일일이 쓰지 않아도 되게끔 만든 유틸리티 함수집합
- pip install mglearn 으로 설치 필요 (in cmd 창)

(+ pip install msgpack ~ 업그레이드 / 인스톨 관련 문제 발생 --> 창에 나와있는 명령어 / 필요 라이브러리 설치)

붓꽃데이터

- 클래스 : 출력되는 값으로 붓꽃의 종류를 의미 - y 의 범주
- 레이블 : 데이터 포인트 하나에 대한 기대 출력으로 특정 데이터 각각의 품종을 의미 - 하나의 row (level)

scikit-learn 의 datasets 모듈에 포함된 iris 데이터를 호출

- load_iris 함수 사용
- 반환한 iris 객체는 딕셔너리와 유사한 brunch 클래스의 객체로 키 / 값으로 구성

```
In [192]: from sklearn.datasets import load_iris
```

```
In [193]: iris_dataset=load_iris()
```

```
In [194]: print("iris_dataset 의 키 : %n{}".format(iris_dataset.keys()))
```

```
# print("{0} and {1} and {2}".format(a,b,result))
```

```
print("%s and %s and %f" % (a, b, result))   프린트 형식!
```

iris_dataset 의 키 :

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

```
#DESCR 키 : 데이터셋에 대한 간략한 설명
```

```
In [196]: print(iris_dataset['DESCR'][:193] + "\n ...")
```

```
#      + : 문자결합
```

Iris Plants Database

=====

Notes

Data Set Characteristics:

:Number of Instances: 150 (50 in each of three classes)

:Number of Attributes: 4 numeric, predictive att

...

```
# target_names 키 : 실제 y 이름 (수행은 factor화된 target 키로!)
```

```
In [197]: print("타깃의 이름 : {}".format(iris_dataset['target_names']))
```

```
타깃의 이름 : ['setosa' 'versicolor' 'virginica']
```

feature_names 키 : 각 특성을 설명하는 문자열 리스트

```
In [199]: print("특성의 이름 : {0}".format(iris_dataset['feature_names']))
```

특성의 이름 :

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

data 키 : x 에 대한 값(특성)만 가지고 있음!

```
In [200]: iris_dataset['data']
```

Out[200]:

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       ....
```

target 키 : y 값(범주형)을 숫자(팩터)형 데이터로 가지고 있음!! 범주형 분류하려면 꼭 필요

```
In [202]: iris_dataset['target']
```

Out[202]:

```
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, 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])
```

훈련 데이터와 테스트 데이터

모델 평가

- 머신러닝 모델을 만들었다면 새 데이터를 적용시켜 예측을 하기 전 모델을 평가해야 함
- 모델을 만들 때 쓴 데이터는 이미 학습되어 있으므로 모델 평가 목적으로 재사용 불가 (overfit 확인을 위해 train 데이터 / test 데이터 예측도 함께 그래프로 그리기도 함! 적당한 neighbor number 알기 위해서)
- 모델 평가를 위해 레이블을 알고 있는 새 데이터를 모델에 적용한 예측도 측정 필요
- scikit-learn 은 데이터 셋을 섞어서 제공할 경우 자동으로 훈련 데이터 셋과 테스트 데이터 셋으로 나눠주는 **train_test_split** 함수 제공
- 기본 값은 훈련데이터 75% : 테스트데이터 25%
 - => test_size 옵션 = 0.25 기본값, 조정 가능함. 정수가 아닌 경우 반올림한 숫자 사용

2019 추가

by data

- split test // train data + test data(=hold-out set)
- cross validation // 섞어서 검증
- bootstrap
- Leave One Out Cross validation

by 변수 평가 척도

regression

- coefficient of determination
- Root mean squared error
- Mean absolute error
- Akaike or Bayesian information criterion

classification

- Area under ROC
- confusion matrix
- gain or lift charts
- concordant and discordant ratio

Unsupervised

- contingency tables
- Sum of squared errors between clustering objects and cluster centers or centroids
- Silhouette value
- Rand index
- Matching index
- Pairwise and adjusted pairwise precision and recall (primarily used in NLP)

confusion matrix

		예측치		합계
		True	False	
실제값	True	TP	FN	P
	False	FP	TN	N
합계		P'	N'	P+N

-False Positive : 1종 오류--알파값

-False Negative : 2종 오류

Precision(정확도, 정밀도) = $\frac{TP}{TP+FP}$ / True로 예측한 값 중 실제로 True인 정도

Recall(재현율) = $\frac{TP}{TP+FN}$ / 실제값 True 중 True로 예측된 값의 정도 (= sensitivity(민감도))

F1 score = $2 \times \frac{Precision \times Recall}{Precision + Recall}$

accuracy(정확도) = $\frac{TP+TN}{P+N}$ / 전체 관측치 중 실제값과 예측치가 일치한 정도

error rate = $1 - accuracy = \frac{FP+FN}{P+N}$

sensitivity(민감도) = $\frac{TP}{TP+FN}$

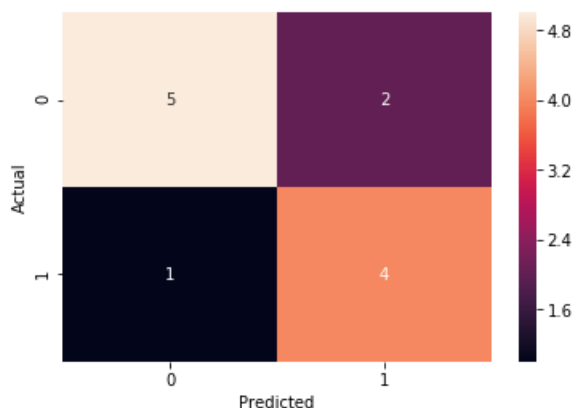
specificity(특이도) = $\frac{TN}{TN+FP}$ / 실제값이 False인 경우 False로 예측치가 적중한 정도

```
In [86]: data={'y_Actual':[1,0,0,1,0,1,0,0,1,0,1,0],
              'y_Predicted':[1,1,0,1,0,1,1,0,1,0,0,0]}
df=DataFrame(data)
```

```
In [80]: confusion_matrix=pd.crosstab(df.y_Actual, df.y_Predicted,
                                     rownames=['Actual'],colnames=['Predicted'])
```

```
In [81]: import seaborn as sns
sns.heatmap(confusion_matrix, annot=True)
```

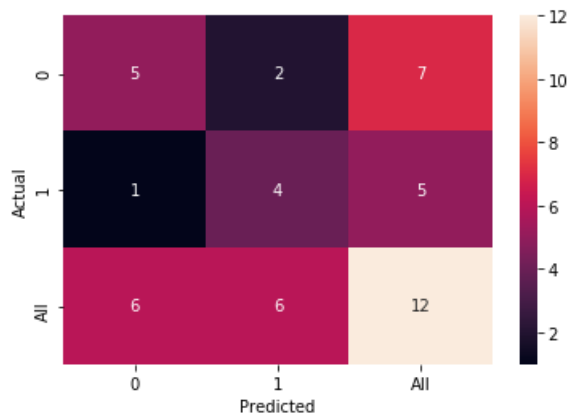
```
Out [81]: <matplotlib.axes._subplots.AxesSubplot at 0x21032758438>
```



```
In [84]: confusion_matrix=pd.crosstab(df.y_Actual, df.y_Predicted,
                                     rownames=['Actual'],colnames=['Predicted'],margins=True)
```

```
In [85]: import seaborn as sns
sns.heatmap(confusion_matrix, annot=True)
```

Out [85]: <matplotlib.axes._subplots.AxesSubplot at 0x210328132e8>



sklearn.metrics에서 confusion_matrix를 통해서도 가능

Recall, Precision 사용 이유?(Accuracy 말고)

→ 데이터 분포가 불균형할 때 Accuracy만 높게 나오는 경우가 있음.

→ 의료에서는 Precision이 높은 모델을 학습하는 경우가..! 더 바람직 / 목적에 맞게 타겟 설정

```
In [92]: from pandas_ml import ConfusionMatrix # !pip install pandas_ml
```

```
In [93]: ConfusionMatrix(df.y_Actual, df.y_Predicted).print_stats()
```

```
population: 12
P: 5
N: 7
PositiveTest: 6
NegativeTest: 6
TP: 4
TN: 5
FP: 2
FN: 1
TPR: 0.8
TNR: 0.7142857142857143
PPV: 0.6666666666666666
NPV: 0.8333333333333334
FPR: 0.2857142857142857
FDR: 0.3333333333333333
FNR: 0.2
ACC: 0.75
F1_score: 0.7272727272727273
MCC: 0.50709255283711
informedness: 0.5142857142857142
markedness: 0.5
prevalence: 0.4166666666666667
LAP: 2.8000000000000003
LRN: 0.28
DOR: 10.0
FOR: 0.16666666666666666
```

다시 2018년

1. train_test_split

In [203]: `from sklearn.model_selection import train_test_split`

In [204]: `x_train, x_test, y_train, y_test = train_test_split(iris_dataset['data'], iris_dataset['target'], random_state=0)` # X / Y 분리해서 입력

`random_state` 옵션 : 데이터 셋을 `random` 으로 제공하면서 여러 번 실행해도 결과 값이 똑같이 나오도록 0 을 매개변수로 전달 (샘플 고정)

#(in R ~ seed) for 문을 통한 TEST 등 train / test 자료를 사용할 때 변동없이 똑같이 사용하기 위해 고정시키기!

2. 산점도 행렬

- 각 속성간 상관관계 분석 시 필요

In [208]: `iris_dataframe = pd.DataFrame(x_train, columns=iris_dataset.feature_names)`

`x_train` 데이터를 사용하여 데이터프레임을 만든다.

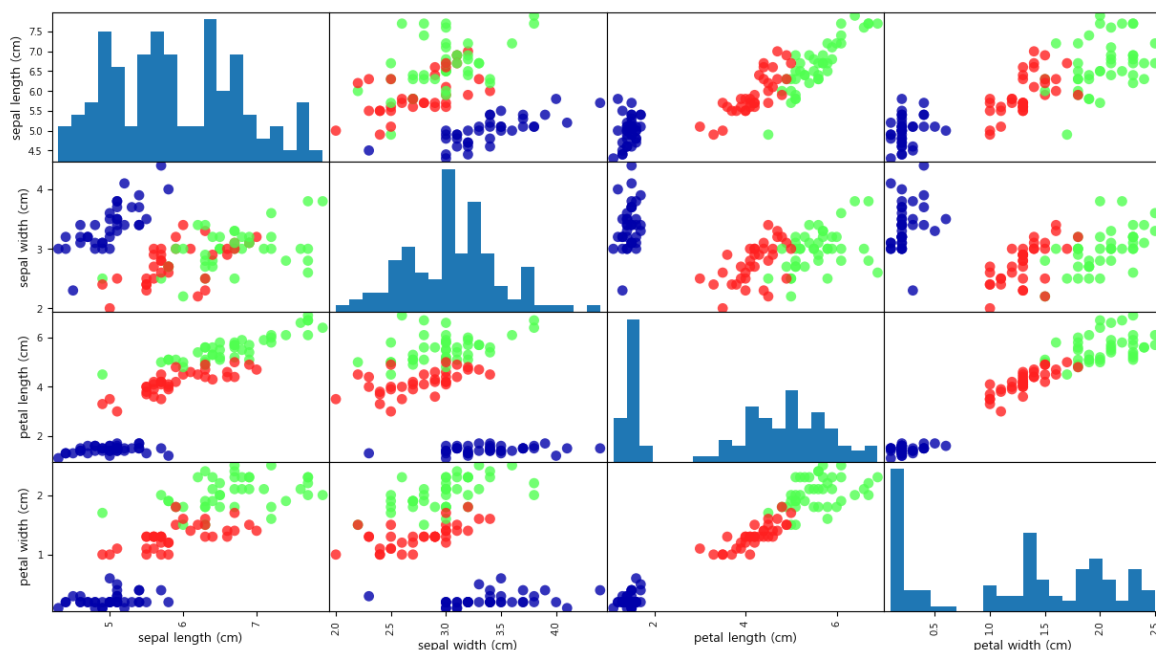
열의 이름은 `iris_dataset.feature_names` 에 있는 문자열을 사용한다.

In [209]: `pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15,15), marker='o', hist_kws={'bins' : 20}, s=60, alpha=.8, cmap=mglearn.cm3)`

`c` 옵션 : `y_train` 에 따라 색으로 구분함

#`s` 옵션..?

#`cmap` ~ `mglearn` ㄸ



- R 과 다르게 자신과의 산점도 --> 분포로 표현 => 정규분포 등 시각적으로 가볍게 확인 가능.
(그나마 2nd col- 정규분포 근사,
나머지는 log 화, 지수화, 제곱 등 변수 변경 통해 정규성 맞도록 변화! or 비모수적 접근방식
사용)

kNN : 알고리즘 (범주형 자료) -- 분류

예측 알고리즘 : 새 데이터 포인트에서 가장 가까운 훈련 데이터 포인트의 레이블을 새 데이터 포인트 레이블로 예측 --> 스케일 조정 필요 (거리기반)

- scikit-learn 의 neighbors 모듈의 KNeighborsClassifier 클래스 사용
- KNeighborsClassifier 에 n_neighbors 로 이웃의 개수(k)를 전달

1. KNeighborsClassifier 함수

```
In [211]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [212]: knn=KNeighborsClassifier(n_neighbors=1)
```

- knn 객체는 훈련 데이터로 모델을 만들고 새로운 데이터 포인트에 대한 예측을 수행
- 매개변수 : n_neighbors

2. .fit # 분류 모델 설정

```
In [213]: knn.fit(x_train, y_train)
```

#x, y 순서

```
Out[213]:
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', # 유클리드 거리
                    metric_params=None, n_jobs=1, n_neighbors=1, p=2,
                    weights='uniform')
```

3. .score # 예측력 확인

```
In [214]: knn.score(x_test, y_test) # x_test / y_test 따로 만들기
```

```
Out[214]: 0.9736842105263158
```

4. .prediction

cf. in R

```
(In [215]: y_pred=knn.predict(x_test)
```

```
In [216]: np.mean(y_pred==y_test)
```

```
Out[216]: 0.9736842105263158 )
```

---- k를 1 부터 10 까지 변화시키면서 훈련 세트와 테스트 세트의 정확도를 확인

- > 과대적합과 과소적합의 차이를 확인할 수 있음
- > 과대적합과 과소적합의 적절한 균형을 이루는 k 선택

```
In [218]: x_train, x_test, y_train, y_test = train_test_split(iris_dataset['data'], iris_dataset['target'],
random_state=66)
```

for 문 밖에 있는 이유 : data 고정!

```
In [219]: training_accuracy=[]
```

```
In [220]: test_accuracy=[]
```

```
In [12]: k=range(1,11)
```

```
In [22]: for i in k :
```

```
....:     knn=KNeighborsClassifier(n_neighbors=i)
```

```
....:     knn.fit(x_train, y_train)
```

```
....:     training_accuracy.append(knn.score(x_train, y_train))
```

```
....:     test_accuracy.append(knn.score(x_test, y_test))
```

```
In [26]: plt.rc("font",family="Malgun Gothic")
```

```
In [25]: plt.plot(k, training_accuracy, label="훈련 정확도")
```

```
Out[25]: [<matplotlib.lines.Line2D at 0xbbd5080>]
```

```
In [29]: plt.plot(k, test_accuracy, label="테스트 정확도")
```

```
Out[29]: [<matplotlib.lines.Line2D at 0xbc57630>]
```

```
In [30]: plt.ylabel("정확도")
```

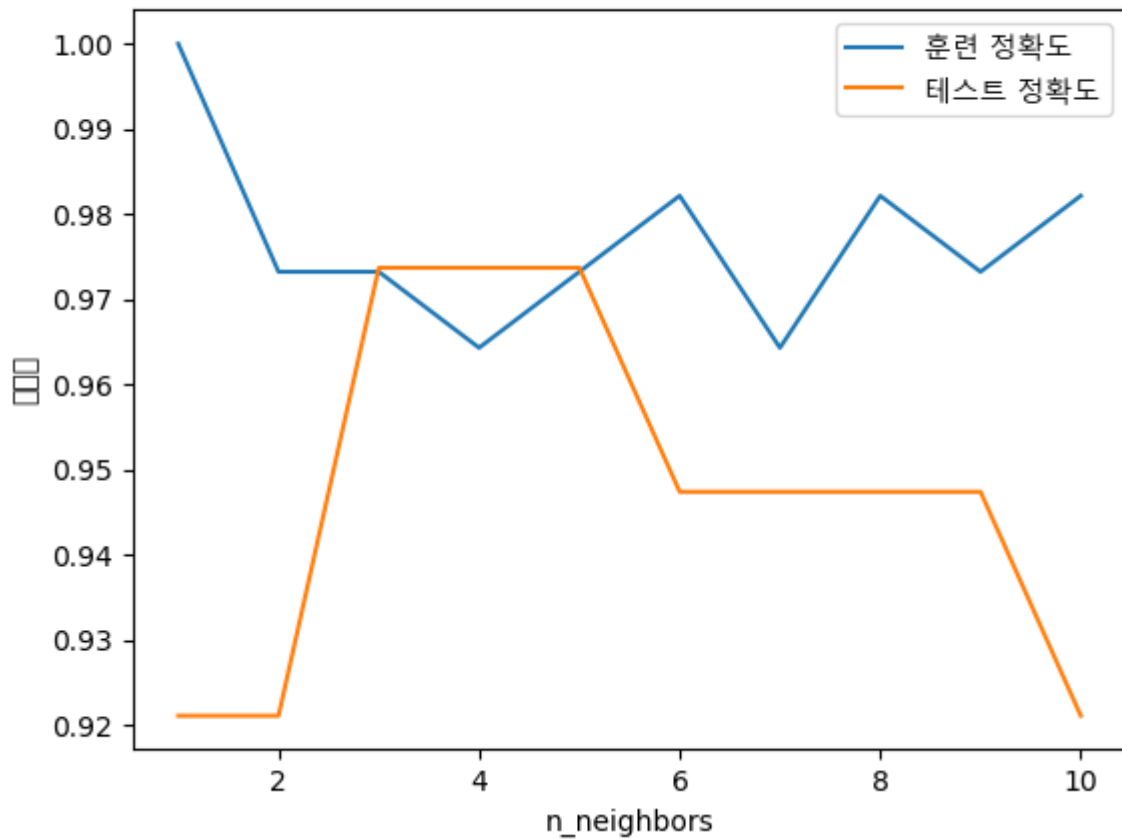
```
Out[30]: Text(33.8472,0.5,'정확도')
```

```
In [31]: plt.xlabel("n_neighbors")
```

```
Out[31]: Text(0.5,23.4122,'n_neighbors')
```

```
In [32]: plt.legend()
```

```
Out[32]: <matplotlib.legend.Legend at 0xbc370f0>
```



검증용 데이터는 정확도가 비교적 낮게 나타나고, 훈련 데이터는 이미 모델링할 때 적용시켜서 정확도가 높게 나타남

k=3, 5정도에서 gap 차이가 크게 나타나지 않음 + 두 정확도 모두 높은 수준

원래는 테스트용 정확도만 확인하는데 training + test 모두 확인 => 과적합 확인

아예 다른, 새로운 데이터로 평가하는게 더 score 변수의 과적합을 막을 수 있다.

실습

유방암 데이터 세트

```
In [51]: from sklearn.datasets import load_breast_cancer
```

```
In [52]: cancer=load_breast_cancer()
```

```
In [53]: cancer.keys()
```

```
Out[53]: dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

```
In [54]: cancer.data.shape
```

```
Out[54]: (569, 30) # 총 569행, 변수 개수 30개
```

```
In [58]: list(zip(cancer.target_names,np.bincount(cancer.target)))
```

```
#np.bincount : 빈도수 count method
```

```
Out[58]: [('malignant', 212), ('benign', 357)]
```

모델 생성

```
In [38]: x_train, x_test, y_train, y_test = train_test_split(cancer['data'], cancer['target'],  
    ...: random_state=0)
```

```
In [39]: knn=KNeighborsClassifier(n_neighbors=1) # k = 1
```

```
In [40]: knn.fit(x_train,y_train)
```

```
Out[40]:
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
    metric_params=None, n_jobs=1, n_neighbors=1, p=2,  
    weights='uniform')
```

```
In [41]: knn.score(x_train,y_train)
```

```
Out[41]: 1.0
```

```
In [42]: knn.score(x_test,y_test)
```

```
Out[42]: 0.916083916083916
```

적절한 K 수 구하기

In [43]:

```
x_train,x_test,y_train,y_test=train_test_split(cancer['data'],cancer['target'],random_state=66)
```

In [44]: **train_accuracy=[]**

In [45]: **test_accuracy=[]**

In [46]: **k=arange(1,11)**

In [47]: **for i in k :**

```
...     knn=KNeighborsClassifier(n_neighbors=i)
...     knn.fit(x_train,y_train)
...     train_accuracy.append(knn.score(x_train,y_train))
...     test_accuracy.append(knn.score(x_test,y_test))
...:
```

In [48]: **plt.rc('font', family='Malgun Gothic')**

In [51]: **plt.plot(k, train_accuracy, label="훈련 정확도")**

Out[51]: [

In [52]: **plt.plot(k, test_accuracy, label="test 정확도")**

Out[52]: [

In [53]: **plt.ylabel("정확도")**

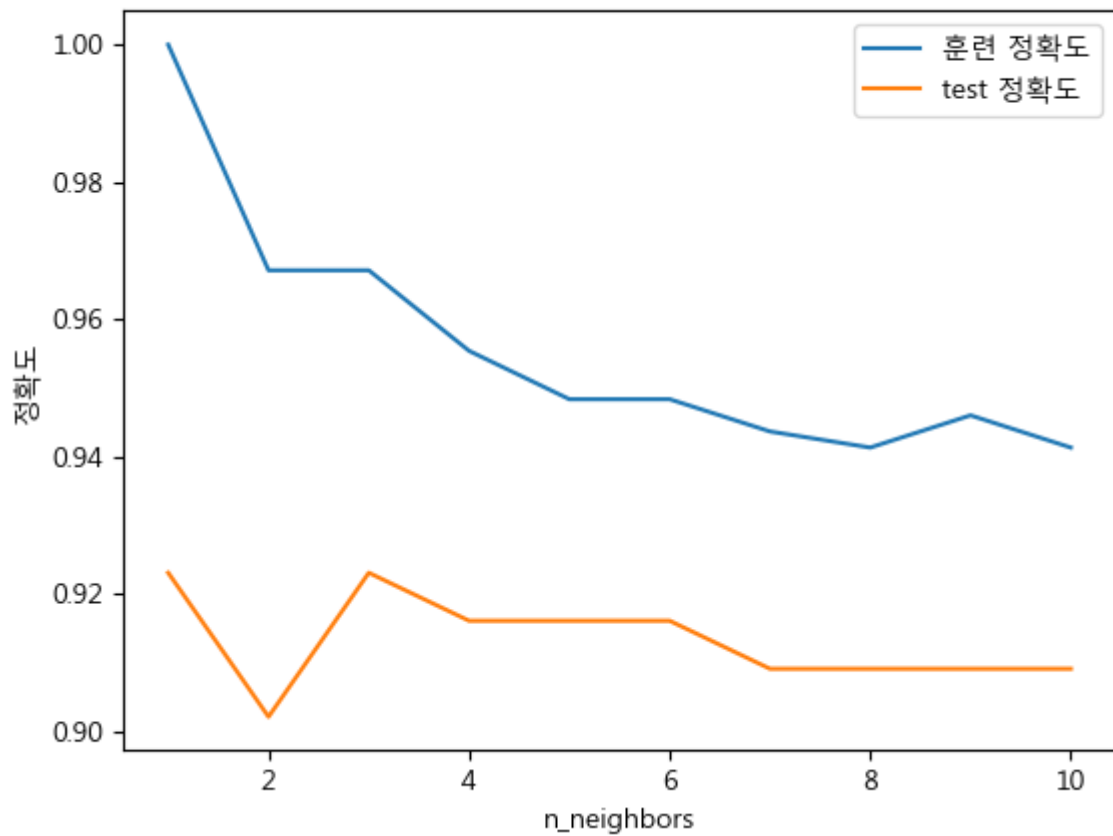
Out[53]: Text(38.7222,0.5,'정확도')

In [54]: **plt.xlabel("n_neighbors")**

Out[54]: Text(0.5,24.4122,'n_neighbors')

In [55]: **plt.legend()**

Out[55]: <matplotlib.legend.Legend at 0xa71e6a0>



k=3일 때 말하기엔 너무 gap이 큼. gap이 작은 k=5일 때 훈련, 테스트 정확도도 모두 높은 편..!
(여러 개인 경우 평균을 내기도)
그래도 훈련정확도가 테스트 정확도보다는 높아야 함 !!(모델이 학습하지 못했다는 뜻)

k-최근접 이웃 회귀 (연속형 자료) -- 예측 !=회귀분석

kNeighborsRegressor

- scikit-learn 에서 제공하는 KNeighborsRegressor 알고리즘 활용
- wave 데이터 셋 활용한 k-최근접 이웃 회귀 분석 예제

```
In [56]: from sklearn.neighbors import KNeighborsRegressor
```

```
In [58]: x,y=mglearn.datasets.make_wave(n_samples=40)
```

```
# cf. 그냥 연속형 데이터 예시 자료. 두 개로 분리된 배열 형태
```

```
In [59]: x_train, x_test, y_train, y_test = train_test_split(x,y,random_state=0)
```

```
In [60]: reg=KNeighborsRegressor(n_neighbors=3)
```

```
# 이웃관측치 3 일 때 평균 -->연속형변수는 k 가 클수록 좋은 편
```

```
In [61]: reg.fit(x_train,y_train)
```

```
Out[61]:
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',  
                    metric_params=None, n_jobs=1, n_neighbors=3, p=2,  
                    weights='uniform')
```

```
In [62]: reg.score(x_test,y_test) #score 메서드를 사용한 모델 평가
```

```
Out[62]: 0.8344172446249604
```

```
-----  
## score 메서드는 회귀분석 시 결정계수( $R^2$  - 회귀선이 설명해주는 정도)를 리턴함.
```

```
## 정확도라는 단어 사용 불가. 연속형 변수여서 class 가 없고, 평균으로 예측. 그래서 일치  
판단이 아닌, 모델로 설명되는 예측 값 나타냄.
```

```
In [63]: reg.predict(x_test)
```

```
Out[63]:
```

```
array([-0.05396539,  0.35686046,  1.13671923, -1.89415682, -1.13881398,  
       -1.63113382,  0.35686046,  0.91241374, -0.44680446, -1.13881398])
```

실습 -3 에서 3 의 값을 갖는 테스트 데이터 셋을 이용한 회귀 예측

```
In [64]: fig, axes=plt.subplots(1, 3, figsize=(15,4))
```

```
In [65]: line=np.linspace(-3,3,1000).reshape(-1,1)
```

```
# 균등한 크기로 -3 ~ 3 1000 개 생성 - 임의생성 for test
```

```
In [86]: for n_neighbors, ax in zip([1,3,9], axes) : #axes 가 axes[0], [1], [2] 순으로 ax 에 전해짐
```

```
...:     reg=KNeighborsRegressor(n_neighbors=n_neighbors)
```

```
...:     reg.fit(x_train, y_train)
```

```
...:     ax.plot(line, reg.predict(line))
```

```
...:     ax.plot(x_train, y_train, '^', c=mglearn.cm2(0), markersize=8)
```

```
...:     ax.plot(x_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)
```

```
...:     ax.set_title("{} 이웃의 훈련 스코어: {:.2f} 테스트 스코어: {:.2f}".format(n_neighbors, reg.score(x_train, y_train), reg.score(x_test, y_test)))
```

```
...:     ax.set_xlabel("특성")
```

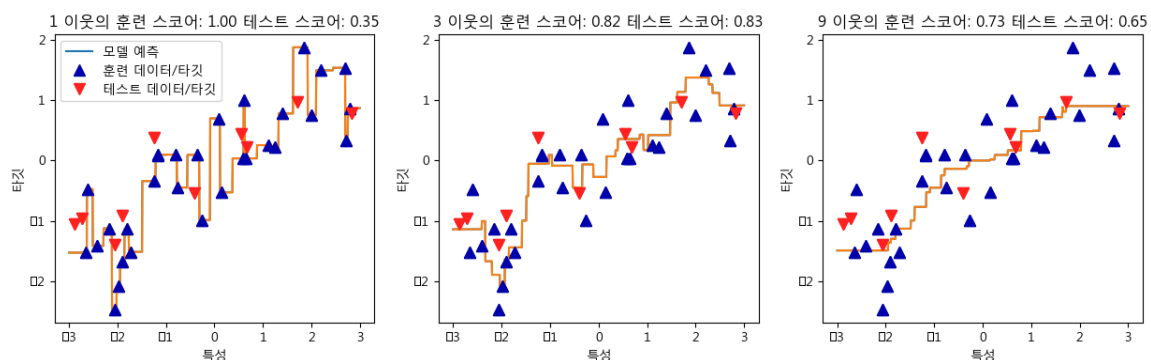
```
...:     ax.set_ylabel("타겟")
```

```
...:
```

```
In [87]: axes[0].legend(["모델 예측", "훈련 데이터/타겟", "테스트 데이터/타겟"])
```

```
#순서대로 1st, 2nd, 3rd 에 대한 label 을 리스트로 전달
```

```
Out[87]: <matplotlib.legend.Legend at 0xc208b70>
```



- k=1 인 경우 훈련 세트의 각 데이터 포인트가 예측에 주는 영향이 커서 예측값이 훈련 데이터 포인트를 모두 지나가는 형태 (테스트 데이터는 매우 엇나감)

- 이웃이 늘어나면 훈련데이터에는 잘 안 맞을 수 있지만, 테스트 데이터는 더 안정된 예측을 얻게 된다.

- k=3 일 때가 가장 적절한듯 !

k-NN 의 장단점과 매개변수

k-NN 분류에서 중요한 매개변수

- 데이터 포인트 사이의 거리를 재는 방법 (주로 유클리디안 거리 방식 사용)
- 파이썬에서의 거리는 민코프스키 거리를 의미하는 `minkowski` 가 기본값
- 거듭제곱의 크기를 정하는 매개변수인 $p=2$ 이면 유클리디안 거리 (기본값)
- 이웃간의 거리를 계산할 때 각 특성마다 범위가 다르면 정규화 필요
- 이웃의 수(주로 3-5 개 정도가 적당)

장점

- 이해하기 쉬운 모델
- 많은 조정 필요 없이 좋은 성능 발휘

단점

- 훈련 데이터 셋이 많아질수록 예측이 느린 경향
- 수백 개 이상의 많은 특성을 가진 데이터 셋에는 잘 동작하지 않음 (비정형 데이터)
- 가중치 x , 가장자리보다 가운데가 더 중요한 경우(이미지) 의미 없음

ex. 보스턴 주택가격 데이터셋

```
In [88]: from sklearn.datasets import load_boston
```

```
In [89]: boston=load_boston()
```

```
In [90]: boston.data.shape
```

```
Out[90]: (506, 13)    # 총 506 건의 데이터와 13 개의 속성 제공
```

```
In [91]: x, y = mglearn.datasets.load_extended_boston()
```

```
# 각 특성끼리 곱하여 상호작용을 표현하는 새로운 특성으로 확장된 데이터 셋 제공
```

```
In [92]: x.shape
```

```
Out[92]: (506, 104)
```

```
In [93]: y.shape
```

```
Out[93]: (506,)
```

#회귀모델 추정 및 예측

```
In [94]: from sklearn.neighbors import KNeighborsRegressor
```

```
In [95]: x_train, x_test, y_train, y_test = train_test_split(boston['data'],boston['target'],
random_state=0)
```

```
In [96]: reg=KNeighborsRegressor(n_neighbors=3)
```

```
In [97]: reg.fit(x_train, y_train)
```

```
Out[97]:
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=3, p=2,
weights='uniform')
```

```
In [98]: y_pred=reg.predict(x_test)
```

```
In [99]: y_pred
```

```
Out[99]:
```

```
array([21.23333333, 39.26666667, 24.06666667, 12.86666667, 21.43333333,
       21.86666667, 21.46666667, 27.8        , 33.23333333, 14.16666667,
       12.        , 11.56666667, 16.66666667, 10.16666667, 40.9        ,
       28.46666667, 23.2        , 26.03333333, 25.56666667, 24.06666667,
       23.53333333, 19.76666667, 19.6        , 32.5        , 20.96666667,
       12.33333333, 16.73333333, 22.06666667, 21.5        , 16.96666667,
       ...
```

모델의 예측률 확인

```
In [100]: reg.score(x_test,y_test)
```

```
Out[100]: 0.5123918703499843
```

적절한 k 수 확인

```
In [101]: x_train, x_test, y_train, y_test = train_test_split(boston['data'],boston['target'],r
...: andom_state=66)
```

```
In [102]: train_accuracy, test_accuracy=[], []
```

```
In [103]: neighbors_settings=range(1,11)
```

```
In [104]: for n_neighbors in neighbors_settings :  
...:     reg=KNeighborsRegressor(n_neighbors=n_neighbors)  
...:     reg.fit(x_train,y_train)  
...:     train_accuracy.append(reg.score(x_train,y_train))  
...:     test_accuracy.append(reg.score(x_test,y_test))  
...:
```

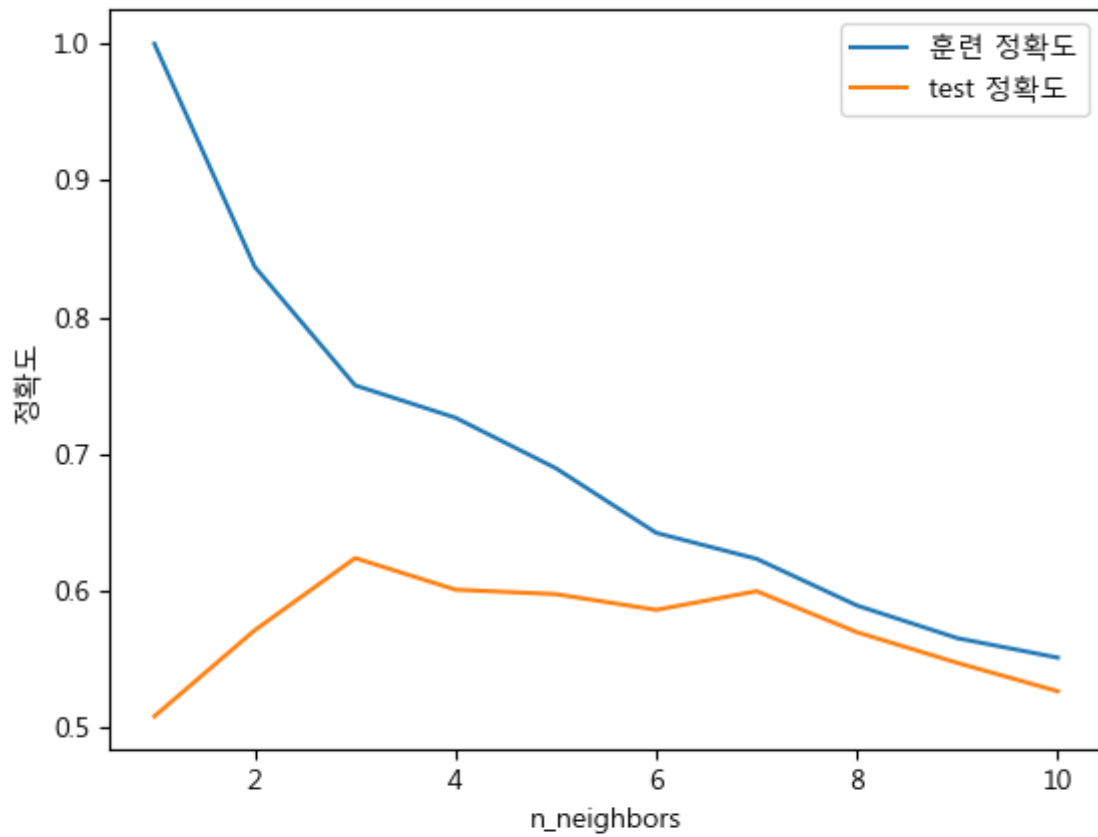
```
In [108]: plt.plot(neighbors_settings, train_accuracy, label="훈련 정확도")  
Out[108]: [<matplotlib.lines.Line2D at 0xc497128>]
```

```
In [109]: plt.plot(neighbors_settings, test_accuracy, label="test 정확도")  
Out[109]: [<matplotlib.lines.Line2D at 0xbe54080>]
```

```
In [111]: plt.xlabel("n_neighbors")  
Out[111]: Text(0.5,24.4122,'n_neighbors')
```

```
In [112]: plt.ylabel("정확도")  
Out[112]: Text(46.3472,0.5,'정확도')
```

```
In [113]: plt.legend()  
Out[113]: <matplotlib.legend.Legend at 0xc497748>
```



-- 초반에 두드러지는 overfit 현상!

- 그나마 $k=7$ 이 제일 gap 이 작고 test 정확도도 높은 지점인 듯

- R^2 값이 낮음. FNN 회귀모델은 별로 좋아보이지 않음(0.8 이상 희망)

(cf. 이미지 모델은 워낙 예측력이 낮아서 기준이 낮은 편)

선형 회귀

-단순 선형 회귀 모델

모델

w : 기울기 파라미터로 가중치 또는 계수로 표현, coef_속성으로 확인 가능

b : 절편을 나타내는 파라미터로 intercept_속성으로 확인 가능

파이썬을 활용한 분석 방법(scikit-learn 알고리즘 활용 방법)

```
In [114]: from sklearn.linear_model import LinearRegression
```

```
In [115]: x, y = mglearn.datasets.make_wave(n_samples=60)
```

```
In [119]: lr=LinearRegression().fit(x_train,y_train) # 적용 + 훈련
```

결과 확인

```
In [120]: lr.coef_
```

```
Out[120]: array([0.39390555])
```

```
In [121]: lr.intercept_
```

```
Out[121]: -0.03180434302675973
```

```
In [122]: lr.score(x_train, y_train) # R^2 값
```

```
Out[122]: 0.6700890315075756
```

```
In [123]: lr.score(x_test, y_test)
```

```
Out[123]: 0.6593368596863701
```

- test 결정계수가 0.66 이므로 비교적 좋은 모델이라고 볼 수 없음.

다만 과대적합 문제는 발생하지 않음 (단순 선형 회귀 모델)

(대체적으로 모델이 간단하면 overfit 발생 X)

cf. vs KNN

- 회귀분석에서는 매개변수 선택과정 X, 바로 알스퀘어 값 비교

-다중 선형 회귀 모델

- 보스턴 주택가격 데이터 셋을 이용한 선형 회귀

```
In [125]: x,y=mglearn.datasets.load_extended_boston()
```

```
In [126]: x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0)
```

```
In [127]: lr=LinearRegression().fit(x_train,y_train)
```

```
In [128]: lr.score(x_train,y_train)
```

```
Out[128]: 0.9523526436864241
```

```
In [129]: lr.score(x_test,y_test)
```

```
Out[129]: 0.6057754892935703
```

=> 과대 적합 현상-훈련 set만 높은 점수- (복잡도 제어 필요성 제기)

overfit 현상 -> 릿지, 라쏘, 주성분분석(PCA) for 변수조절

- training 데이터가 작거나 모델이 복잡할수록 회귀분석에서 overfit현상 두드러짐
- 데이터를 잘 모를 때 설명력이 높은 방법으로 변수조절해주는 세 가지 방법
릿지, 라쏘, 주성분분석-(for 이미지 인식)

- 공통점 : 설명변수간 영향력을 계산하여 서로 다른 가중치 부여
- 차이점 : 릿지, 주성분분석 - 제거되는 설명변수 없음
 라쏘 - 제거되는 설명변수 있음 (너무 많은 변수를 차원축소 할 때 사용)

cf. 군집분석(kmeans) - 다름! 별개!!! kNN : 답 있는 자료를 분류

릿지(Ridge) 회귀

- 기존 선형 회귀의 과대 적합을 일부 해소시킬 수 있는 모델
- 회귀를 위한 선형 모델이므로 최소적합법에서 사용한 것과 같은 예측 함수를 사용
- 릿지 회귀에서의 가중치(w) 선택 시 고려사항
: 훈련 데이터의 예측 + 과대적합의 최소화

규제(Regularization) - 가중치 부여 과정

- 과대적합이 되지 않도록 모델을 강제로 제한
- 가중치(w)의 절댓값을 가능한 작게 만들어 모든 특성이 모델에 주는 영향을 최소화
- L2방식 사용 (제거X)

alpha 매개변수

- 훈련 세트의 성능 대비 모델의 단순화 강도
- 기본값은 1.0
- 값이 높을수록 계수를 0에 가깝게 일반화시킴
= 모델을 단순화시킴 = 복잡도가 낮아짐

예 : 보스턴 주택 가격 분석

```
In [134]: from sklearn.linear_model import Ridge
```

```
In [135]: ridge=Ridge().fit(x_train, y_train)
```

```
In [136]: ridge.score(x_train,y_train)
```

```
Out[136]: 0.8860578560395835
```

```
In [137]: ridge.score(x_test,y_test)
```

```
Out[137]: 0.7527139600306948
```

=> 선형회귀보다 훈련 세트는 낮아지고 테스트 점수가 높아짐. 둘 간 gap이 줄어들고 test점수가 높아졌으므로 더 우수하다고 볼 수 있다.(과적합X)

alpha값에 따른 정확도의 차이

```
In [140]: x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0)
```

```
In [138]: train_ac=[]
```

```
In [139]: test_ac=[]
```

```
In [141]: k=range(1,11)
```

```
In [142]: for i in k :
```

```
...:     ridge=Ridge(alpha=i).fit(x_train,y_train)
```

```
...:     train_ac.append(ridge.score(x_train,y_train))
```

```
...:     test_ac.append(ridge.score(x_test,y_test))
```

```
...:
```

```
In [143]: plt.plot(k,train_ac, label="훈련 점수")
```

```
Out[143]: [<matplotlib.lines.Line2D at 0xc066080>]
```

```
In [144]: plt.plot(k,test_ac, label="테스트 점수")
```

```
Out[144]: [<matplotlib.lines.Line2D at 0xc066518>]
```

```
In [145]: plt.xlabel("alpha")
```

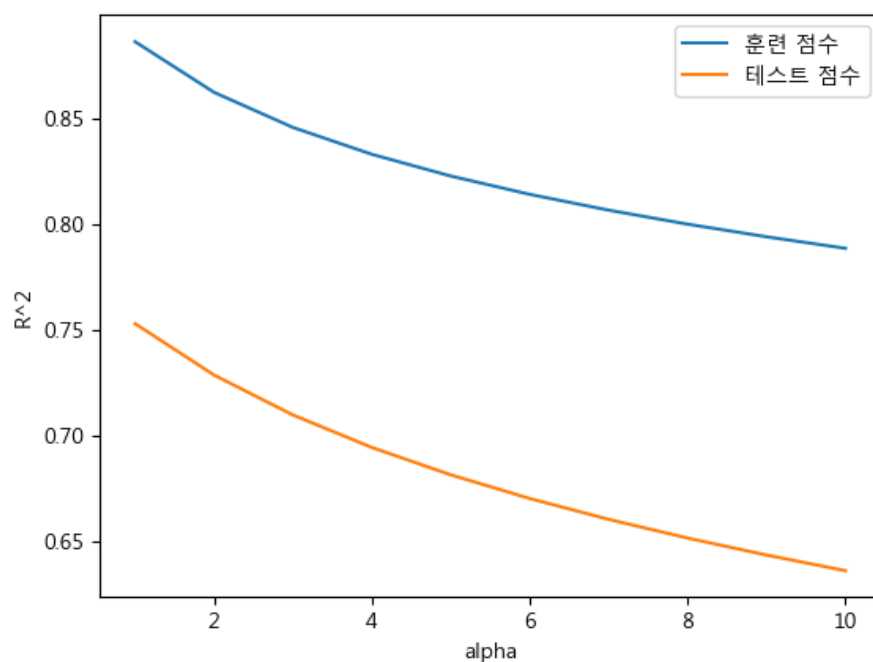
```
Out[145]: Text(0.5,24.4122,'alpha')
```

```
In [146]: plt.ylabel("R^2")
```

```
Out[146]: Text(0.5,24.4122,'R^2')
```

```
In [148]: plt.legend()
```

```
Out[148]: <matplotlib.legend.Legend at 0xc076ba8>
```



alpha값이 같을 경우에 n 의 크기에 따른 선형 회귀와 릿지회귀 비교
릿지가 소표본임에도 회귀분석에 비해 꽤 높은 결과, 유리한 예측률을 가졌다.

cf. 시험문제

릿지 : 차원이 축소된다? (X) ---> 라쏘가 차원이 축소된다. (변수 삭제)

0710

라쏘(Lasso)

- 선형회귀에 규제를 적용하는데 릿지의 대안
 - 릿지 회귀에서와 같이 계수를 0에 가깝게 만드는 것이 목적
 - L1규제 채택
 - L1규제 결과로 라쏘를 사용할 때 어떤 계수는 0이 되어 모델에서 제거될 수 있음
- => 자동으로 **feature selection**이 이루어짐 (=차원축소) // 라쏘 특징

- boston 주택 가격에 라쏘 적용 파이썬 분석

```
In [10]: from sklearn.linear_model import Lasso
```

```
In [32]: lasso=Lasso().fit(x_train,y_train)
```

```
In [33]: lasso.score(x_train,y_train)
```

```
Out[33]: 0.29323768991114607
```

```
In [34]: lasso.score(x_test,y_test)
```

```
Out[34]: 0.20937503255272294
```

```
In [36]: sum(lasso.coef_!=0)
```

```
Out[36]: 4    # 과소적합 (선택된 특성이 4개로 너무 작음)
```

- alpha 매개변수를 조절하여 모델의 복잡도 선택 가능 (alpha=0.01)

```
In [37]: lasso001=Lasso(alpha=0.01, max_iter=100000).fit(x_train,y_train)
```

```
# max_iter 기본값을 증가시키지 않으면 max_iter값을 늘리라는 경고 발생
```

```
In [42]: lasso001.score(x_train,y_train)
```

```
Out[42]: 0.8965069559751287
```

```
In [43]: lasso001.score(x_test,y_test)
```

```
Out[43]: 0.7656489887843517
```

```
In [44]: sum(lasso001.coef_!=0)
```

```
Out[44]: 33
```

- alpha 매개변수를 조절하여 모델의 복잡도 선택 가능 (alpha=0.001)

In [45]: `lasso0001=Lasso(alpha=0.001, max_iter=100000).fit(x_train,y_train)`

In [47]: `lasso0001.score(x_train,y_train)`

Out[47]: 0.9396620850927064

In [46]: `lasso0001.score(x_test,y_test)`

Out[46]: 0.7396519336121237

In [48]: `sum(lasso0001.coef_!=0)`

Out[48]: 69

alpha값이 너무 낮으면 모델의 복잡도가 높아져 선형회귀와 비슷한 결과 (과대 적합 발생)

alpha값에 따른 정확도 차이

```
In [49]: train_ac=[]
```

```
In [50]: test_ac=[]
```

```
In [51]: k=range(1,11)
```

```
In [52]: for i in k :^M
```

```
...:      ...:      lasso=Lasso(alpha=i).fit(x_train,y_train)^M
...:      ...:      train_ac.append(lasso.score(x_train,y_train))^M
...:      ...:      test_ac.append(lasso.score(x_test,y_test))^M
...:
...:
...:
```

```
In [53]: plt.plot(k,train_ac, label="훈련 점수")
```

```
Out[53]: [<matplotlib.lines.Line2D at 0x85f9748>]
```

```
In [54]: plt.plot(k,test_ac, label="테스트 점수")
```

```
Out[54]: [<matplotlib.lines.Line2D at 0x8539320>]
```

```
In [55]: plt.xlabel("alpha")
```

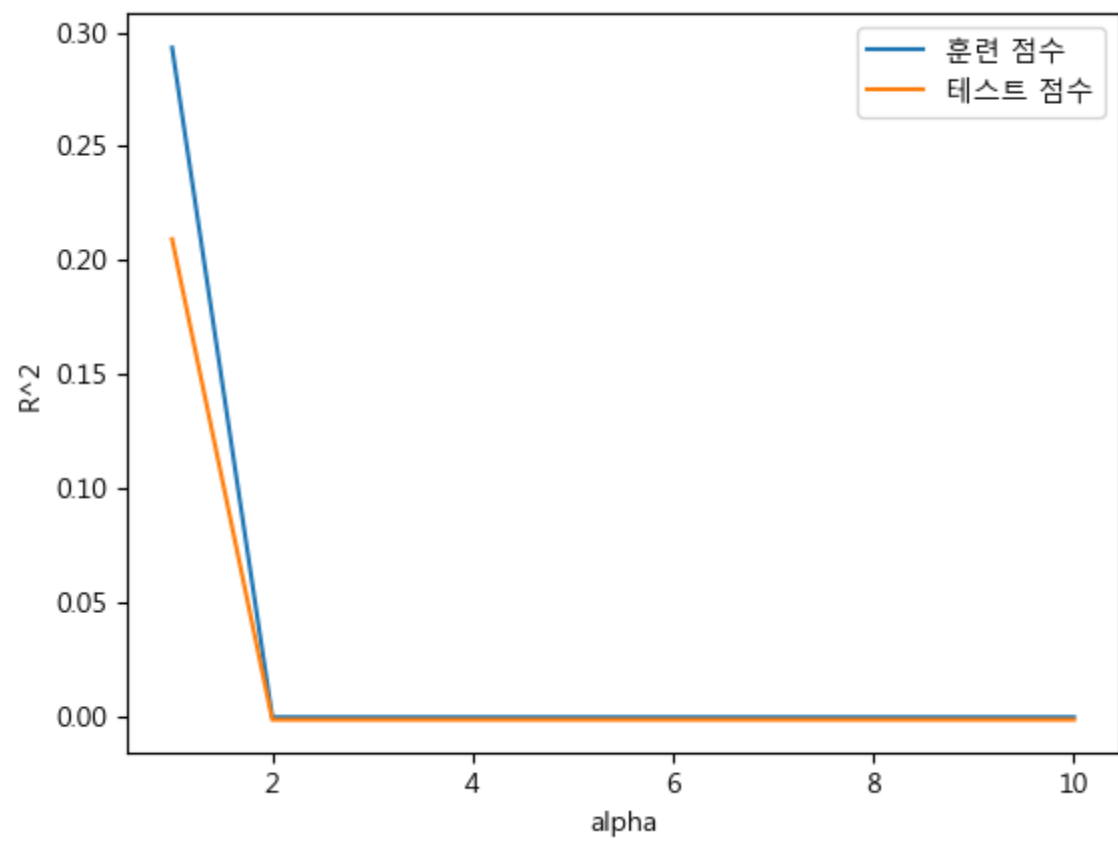
```
Out[55]: Text(0.5,23.4122,'alpha')
```

```
In [56]: plt.ylabel("R^2")
```

```
Out[56]: Text(33.8472,0.5,'R^2')
```

```
In [57]: plt.legend()
```

```
Out[57]: <matplotlib.legend.Legend at 0x85343c8>
```



|

StatsModels 패키지

StatsModels

-기초 통계, 회귀 분석, 시계열 분석 등 다양한 통계 분석 기능을 제공하는 패키지 (sklearn 에 없는 기능 제공 -- 통계적 검정 절차)

(홈페이지 <http://www.statsmodels.org>)

나이브 베이즈 분류기(naive bayes)

+ 통계적 분류기

/ 주어진 데이터가 특정 클래스에 속하는지를 확률을 통해서 예측 (조건부 확률 이용)

+ 베이즈 확률 이론을 적용한 기계학습 방법

- 두 확률 변수(사전 확률과 사후 확률) 사이의 관계를 나타내는 이론
- 사전확률 : 사건이 발생하기 전에 알려진 확률
- 사후확률 : 베이즈 이론에 근거한 확률

+ 텍스트 데이터처럼 희소한 고차원인 경우 높은 정확도와 속도 제공

(차원이 낮을 땐 효과적 X, 설명변수가 많이 필요 - 고차원!) cf. kNN 고차원 부적합(거리기반)

+ 적용 분야

- 스팸 메일 분류, 문서(주제) 분류, 비 유무
- 컴퓨터 네트워크에서 침입자 분류(악성코드 유무)

~ 토큰화 ; 분절화!

in python

- GaussianNB : 연속적인 어떤 데이터도 적용 가능 ~ 연속형 잘 안 쓰임.. 낮은 예측
 - BernoulliNB : 이진데이터 적용
 - MultinomialNB : 카운트 데이터(단어의 횟수 등)
- /텍스트 데이터의 분류는 주로 BernoulliNB와 MultinomialNB를 사용

scikit-learn의 나이브 베이즈 분류기 알고리즘의 계산 방법

- GaussianNB : 클래스별로 각 특성의 표준편차와 평균을 계산
- BernoulliNB : 각 클래스의 특성 중 0이 아닌 것이 몇 개인지를 계산
- MultinomialNB : 클래스별로 특성(설명변수)의 평균을 계산

alpha값 조정으로 모델의 복잡도 조절 가능

ex. 아이디어 익명의 작가 ~ 여러 작가들 데이터 가지고 분석..! 분류

설명변수(단어) ~ 각 확률 조사(빈도수 기반) ~ 가장 높은 확률을 기반으로 분류

(내 기준 ex. 50%이상시 분류 -->알고리즘 내에서 불가능. 알고리즘과 별개로 조건을 달아야 함!)

아이디어 자소서 단어 ~ 합격여부

-20뉴스그룹 데이터

```
In [67]: from sklearn.datasets import fetch_20newsgroups
```

```
In [68]: from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [69]: from sklearn.naive_bayes import MultinomialNB
```

```
In [72]: from sklearn.pipeline import Pipeline
```

모델을 여러 개 적용하고 싶을 때 Pipeline 사용

토큰화된 데이터를 Multinomial을 전달하고 싶어서..?

```
In [73]: news=fetch_20newsgroups(subset="all")
```

```
In [80]: model=Pipeline([('vect' , TfidfVectorizer(stop_words="english")), ('nb', MultinomialNB
...: ()),]) #english 단어 제외
```

Pipeline을 통한 연속적인 전달 -> vect, nb 알고리즘 이름 + 그에 대한 내용 튜플 형태로 묶어서 동시 적용 + 전달.

#vect -- 1. 토큰화(단어 분절)

nb -- 2. MultinomialNB 적용 -- 데이터 count~ 각각 확률 구함

```
In [81]: model
```

```
Out[81]:
```

```
Pipeline(memory=None,
       steps=[('vect', TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
lowercase=True, max_df=1.0, max_features=None, min_df=1,
ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True, ...True,
vocabulary=None)), ('nb', MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))])
```

#multinomialNB 옵션 -> alpha

(잘 사용은 안 함. 릿지랑 비슷. 제거되는 설명변수 없이 가중치 조절)

```
In [86]: model.fit(news.data, news.target)
```

분류 모델을 이용한 예측 (첫 번째 뉴스 문서 분류 예측해보기)

```
In [87]: n=1
```

```
In [88]: x=news.data[n:n+1]
```

```
In [94]: y=model.predict(x)[0]
```

cf

```
In [92]: model.predict(x)[0]
```

```
Out[92]: 3
```

```
In [93]: model.predict(x)
```

```
Out[93]: array([3])
```


나이브 베이즈 분류기를 이용한 뉴스 분류 시각화

```
In [21]: plt.subplot(211)
```

```
#subplot(nrows,ncols,index,**kwargs), subplot(2,1,1)
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0xb273208>
```

```
In [22]: plt.bar(model.classes_, model.predict_proba(x)[0])
```

```
Out[22]: <BarContainer object of 20 artists>
```

```
In [23]: plt.xlim(-1,20)
```

```
Out[23]: (-1, 20)
```

```
In [24]: plt.gca().xaxis.grid(False)
```

```
In [24]: plt.xticks(model.classes_)
```

```
In [21]: plt.subplot(212)
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0xb273208>
```

```
In [22]: plt.bar(model.classes_, model.predict_log_proba(x)[0]) #값이 낮아서 로그화
```

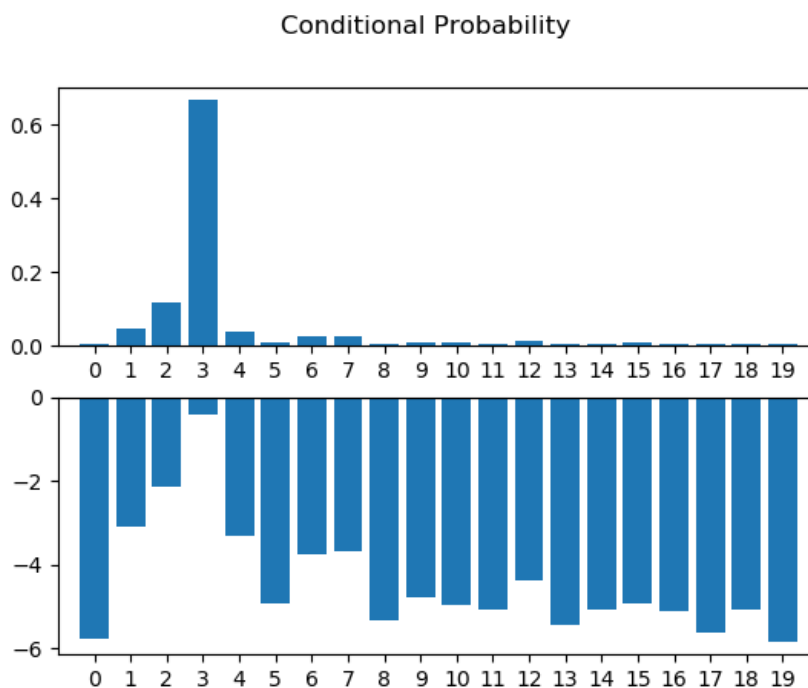
```
Out[22]: <BarContainer object of 20 artists>
```

```
In [23]: plt.xlim(-1,20)
```

```
Out[23]: (-1, 20)
```

```
In [24]: plt.gca().xaxis.grid(False)
```

```
In [24]: plt.xticks(model.classes_)
```



결정 트리

- 분류모델 나무구조

1. Decision Tree(기반)

장점 ; 시각화가 쉽다.

단점 ; 불순도로 구분 - 통계적인 유의 수치 확인 불가능.

복잡한 모델 : 예측력은 좋지만 overfit 현상도 같이 발생함 --> 반드시 조절해야 함

2. Random Forest

3. Gradient Regression Tree (가장 권장/ 그러나 고사양)

- DecisionTreeRegressor(예측)와 **DecisionTreeClassifier(분류)** 모델 제공

Tree -> 모델 복잡도 제어, 매개변수가 많음

cp

max_depth

...

-유방암 데이터 셋의 결정 트리 분석

```
In [35]: from sklearn.tree import DecisionTreeClassifier
```

```
In [36]: tree=DecisionTreeClassifier(max_depth=4, random_state=0)
```

```
In [37]: tree.fit(x_train,y_train)
```

```
Out[37]:
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=4,  
                        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')
```

```
In [41]: tree.score(x_train,y_train)
```

```
Out[41]: 0.9929577464788732
```

```
In [43]: tree.score(x_test,y_test)
```

```
Out[43]: 0.9020979020979021
```

#도표 그리기

```
In [44]: from sklearn.tree import export_graphviz
```

```
In [45]: export_graphviz(tree, out_file="tree.dot", class_names=["악성","양성"], feature_names=  
        ...: cancer.feature_names, impurity=False, filled=True)
```

#out_file : 내보낼 파일

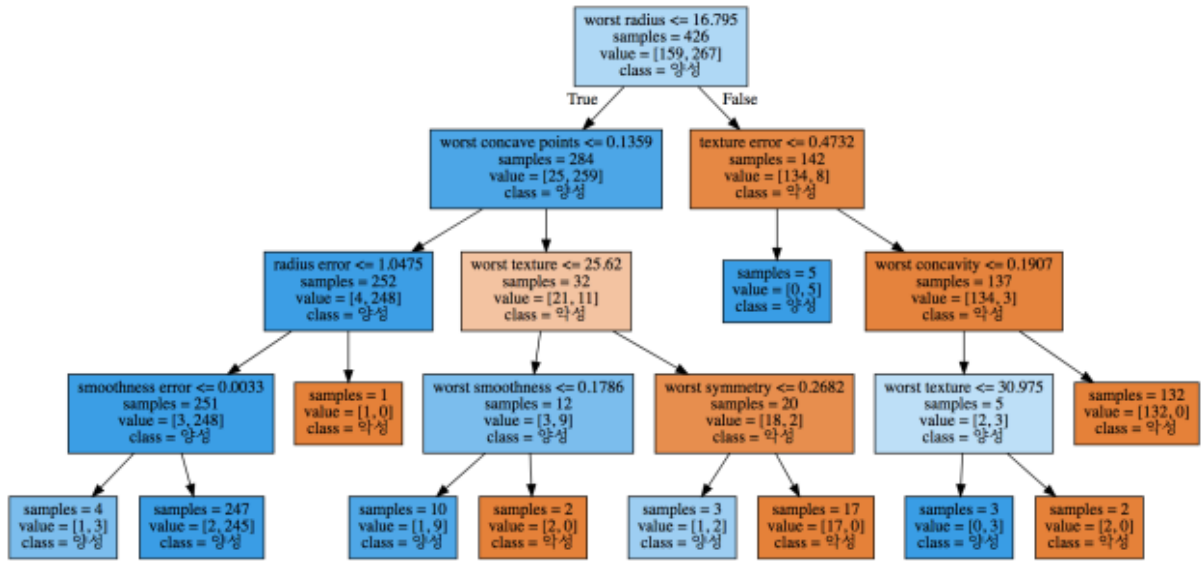
class_names : 분류 클래스

feature_names : 적용시킬 설명변수 data

impurity : 지니불순도 출력

filled : 채워진 형태 graph

```
import graphviz  
with open("tree.dot") as f:  
    dot_graph = f.read()  
display(graphviz.Source(dot_graph))
```



랜덤 포레스트(Random Forest)

- 앙상블
- : 여러 머신러닝 모델을 연결하여 더 강력한 모델을 만드는 기법
- : 분류와 회귀 문제의 다양한 데이터 셋 예측에 효과적 (우수한 예측력)
- : 과대적합을 해결할 수 있는 매개변수의 사용
- : 랜덤 포레스트(Random Forest)와 그래디언트 부스팅 기법(gradient boosting)이 주로 사용

랜덤 포레스트 ; 여러 결정트리 모델 연결 = 앙상블 기법 적용

- 결정 트리의 과대적합 보완 모델로 사용
- 조금씩 다른 여러 결정 트리의 묶음을 통해 평균적 예측을 수행하는 기법
- 각 트리는 비교적 예측을 잘 할 수 있지만 데이터의 일부에 과대적합 경향
- 트리 모델의 예측 성능이 유지되면서 과대적합을 줄일 수 있는 모델
- RandomForestRegressor 나 RandomForestClassifier 모델 제공

장점

- 성능이 매우 뛰어나고 매개변수 튜닝을 많이 하지 않아도 잘 작동됨
- 데이터의 스케일을 맞추는 필요 없음(cf. 회귀분석, 거리기반 -> 스케일 조정 꼭 필요)
- 분산 시스템에 유리 (n_jobs 옵션으로 cpu 코어 수 지정, 병렬 수행 가능)

단점

- 텍스트 데이터 같이 매우 차원이 높고 희소한 데이터에는 부적합 (-> 나이브 베이즈 모델)
- 메모리 사용률이 높아 훈련과 예측이 선형모델보다 느림

주요 매개변수

1. n_estimators : 트리의 수, 클수록 과대적합을 줄여 더 안정적인 모델 학습, 메모리 사용률 고려 필요
2. max_features : 각 트리의 random 생성을 결정하는 변수로, 클수록 과대적합 해소.
그러나 max_features=설명변수 개수인 경우 random tree 와 크게 다를 바 없음.
3. max_depth : 사전 가지치기 옵션, 연속된 질문을 제한. 작을수록 과대적합 해소

2 번과 3 번으로 overfit 조절 !

랜덤 포레스트의 알고리즘

1. 랜덤으로 만들 트리의 개수 지정(**n_estimators** 매개변수로 기본값은 100)
 - 각 트리들은 완전히 독립적이어야 함. 트리가 고유하게 생성되도록 random sampling 수행
2. bootstrap sample 생성 : 트리를 만들 때 사용하는 데이터 포인트를 무작위로 선택 (독립적인 data 만들 ~ sample 추출)
 - 원래의 데이터 셋과 동일한 크기의 샘플 생성
 - 어떤 데이터 포인트는 누락될 수도 있고 중복될 수도 있음

참 ; RandomForest - 각 관측치들이 중복될 수 없다? (X)
3. 결정 트리 생성
4. 각 노드에서 후보 특성을 무작위로 선택 (**max_features** 로 지정된 숫자만큼) ~ 후보군
 - max_features 의 크기가 원래 설명변수 개수와 같으면 단 하나의 Decision Tree 로 하는 것과 큰 차이가 없다.
 - 오히려 굉장히 작을 때 random 효과가 크게 나타난다. (모델이 완전 다름). 그러나 max_features 가 작으면 모델이 복잡하여 더 복잡해진다.

[무작위로 만든 모델 - 좋은 설명변수 생략 가능성이 높아 depth 가 높아질 가능성 높음->overfit]
5. 선택된 후보의 features 중 최선의 테스트 수행
6. 각 노드마다 반복 수행
 - 트리의 각 노드는 다른 후보 특성들을 사용하여 테스트 수행

2번과 4번의 과정이 서로 다른 tree들을 만드는 데에 중요하다.

max_features 옵션

- n_features 로 지정하면 각 분기에서 모든 특성을 고려하므로 특성 선택에 무작위성 배제
- 값이 클수록 트리들은 서로 비슷해지고 가장 두드러진 특성을 이용해 데이터에 적용
- 1 로 지정하면 트리의 분기는 테스트할 특성을 고를 필요 없이 무작위 선택된 특성 사용
- 값이 낮아질수록 서로 다른 트리들이 생성, 각 데이터를 맞추기 위해 depth 가 커짐

사용되는 곳 ~ 아이디어

: 신용불량도 / 고객 이탈률 / 사고방식 System (분류 선택시)
/ 사고방지시스템-정상거래 + 비정상거래 분류(for 빠른 처리)

make_moons test data 사용

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_moons

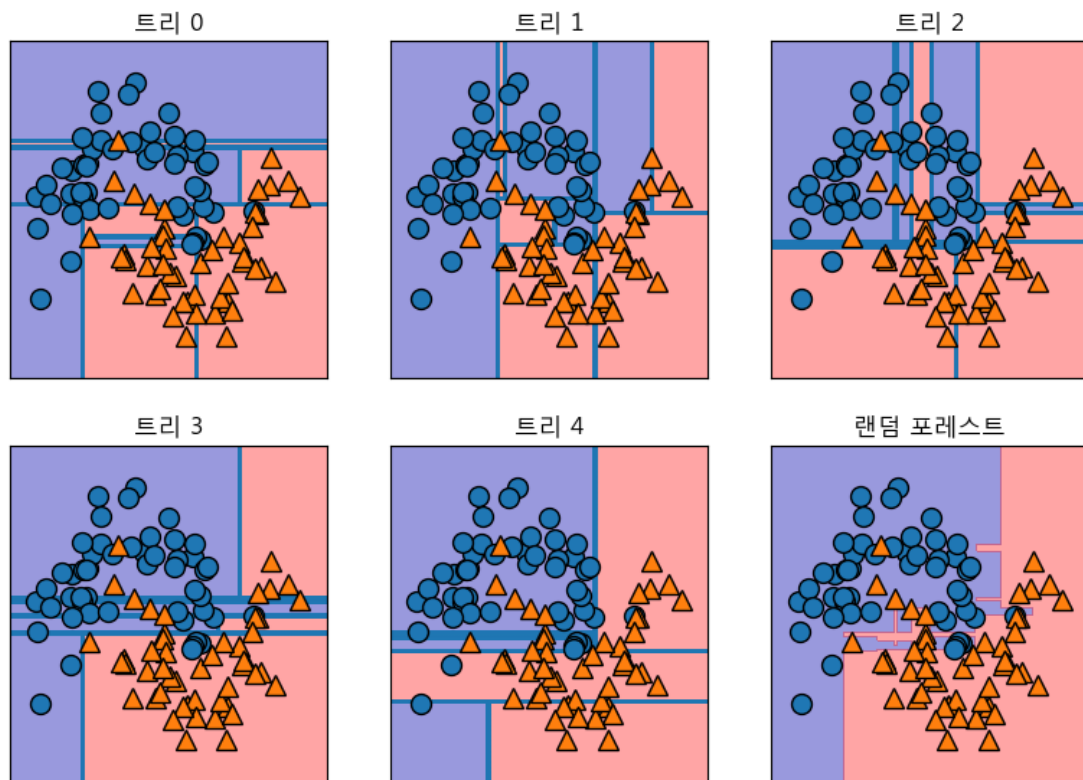
X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    stratify=y,
                                                    random_state=42)

forest = RandomForestClassifier(n_estimators=5,
                               random_state=2)
forest.fit(X_train, y_train)
```

트리의 결정 경계 시각화

```
fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for i, (ax, tree) in enumerate(zip(axes.ravel(),
                                   forest.estimators_)):
    ax.set_title("트리 {}".format(i))
    mglearn.plots.plot_tree_partition(X, y, tree, ax=ax)

mglearn.plots.plot_2d_separator(forest, X, fill=True,
                                ax=axes[-1, -1], alpha=.4)
axes[-1, -1].set_title("랜덤 포레스트")
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
```



분류에서 2 차원 경계확인 시각화기법..!

`#enumerate` : 순서대로 번호 할당

`#mgelarn.plots.plot_2d_separator` (#데이터를 model로 분할한 평면을 만드는 그래프

`forest, #model`

`X, #train data`

`fill=True` #평면 칠함

`...)`

-축의 의미를 직접적으로 알기는 어렵다.

cf. SVM ~ 3,4 차원

랜덤포레스트 유방암 데이터 셋 분석

```
In [79]: x_train, x_test, y_train, y_test=train_test_split(cancer.data, cancer.target, random_s
...: state=0)
```

```
In [86]: forest=RandomForestClassifier(n_estimators=100, random_state=0).fit(x_train,y_train)
```

```
In [87]: forest.score(x_train,y_train)
```

```
Out[87]: 1.0
```

```
In [88]: forest.score(x_test,y_test)
```

```
Out[88]: 0.972027972027972
```

n_estimators, max_features, n_estimator, cp 매개변수 값이 달라질 때 각 훈련 세트와 테스트 세트에 대한 score? 해보기 !

x_test와 y_test에서 마지막 값을 제외하고 train set 적용, 나머지 마지막 한 개 data는 예측 시도 forest.predict (예측용 데이터)

```
In [89]: x_test=x_test[:-1]
```

```
In [90]: y_test=y_test[:-1]
```

```
In [92]: x_sample=x_test[-1]
```

```
In [93]: y_sample=y_test[-1]
```

```
In [94]: forest.predict(x_sample.reshape(1,-1))
```

한 개만 선택하는 과정에서 차원 변경됨 -- 설명변수를 한 행에 담긴 형태로 전달하려고 reshape(1,-1) -- 행별 전달

```
Out[94]: array([1])
```

```
In [95]: forest.predict(x_sample.reshape(1,-1))[0]
```

```
Out[95]: 1
```

```
In [96]: y_hat=forest.predict(x_sample.reshape(1,-1))[0]
```

```
In [98]: cancer['target_names'][y_hat] # 예측
```

```
Out[98]: 'benign'
```

```
In [99]: cancer['target_names'][y_sample] # 실제
```

```
Out[99]: 'benign'
```

---다시 복습

1. max_features의 변화에 따른 정확도 차이 비교

: 각 트리의 random 생성을 결정하는 변수로, 클수록 과대적합 해소.

(그러나 random의미 떨어짐)

<https://tensorflow.blog/%ED%8C%8C%EC%9D%B4%EC%8D%AC-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D/2-3-6-%EA%B2%B0%EC%A0%95-%ED%8A%B8%EB%A6%AC%EC%9D%98-%EC%95%99%EC%83%81%EB%B8%94/>

```
In [2]: from sklearn.datasets import load_breast_cancer
```

```
In [3]: cancer=load_breast_cancer()
```

```
In [4]: from sklearn.model_selection import train_test_split
```

```
In [5]: x_train,x_test,y_train,y_test = train_test_split(cancer.data, cancer.target, random_sta
...: te=3)
```

```
In [6]: train_acc, test_acc=[], []
```

```
In [7]: feature_setting=range(1,20)
```

```
In [8]: from sklearn.ensemble import RandomForestClassifier
```

```
In [11]: for max_feature in feature_setting :
...:     forest=RandomForestClassifier(max_features=max_feature).fit(x_train,y_train)
...:     train_acc.append(forest.score(x_train,y_train))
...:     test_acc.append(forest.score(x_test,y_test))
...:
```

```
In [12]: plt.plot(feature_setting, train_acc, label="훈련 정확도")
```

```
Out[12]: [<matplotlib.lines.Line2D at 0xa5c4470>]
```

```
In [13]: plt.rc("font", family="Malgun Gothic")
```

```
In [14]: plt.plot(feature_setting, train_acc, label="훈련 정확도")
```

```
Out[14]: [<matplotlib.lines.Line2D at 0xbe62400>]
```

```
In [15]: plt.plot(feature_setting, test_acc, label="test 정확도")
```

```
Out[15]: [<matplotlib.lines.Line2D at 0xbe96be0>]
```

```
In [16]: plt.ylabel("정확도")
```

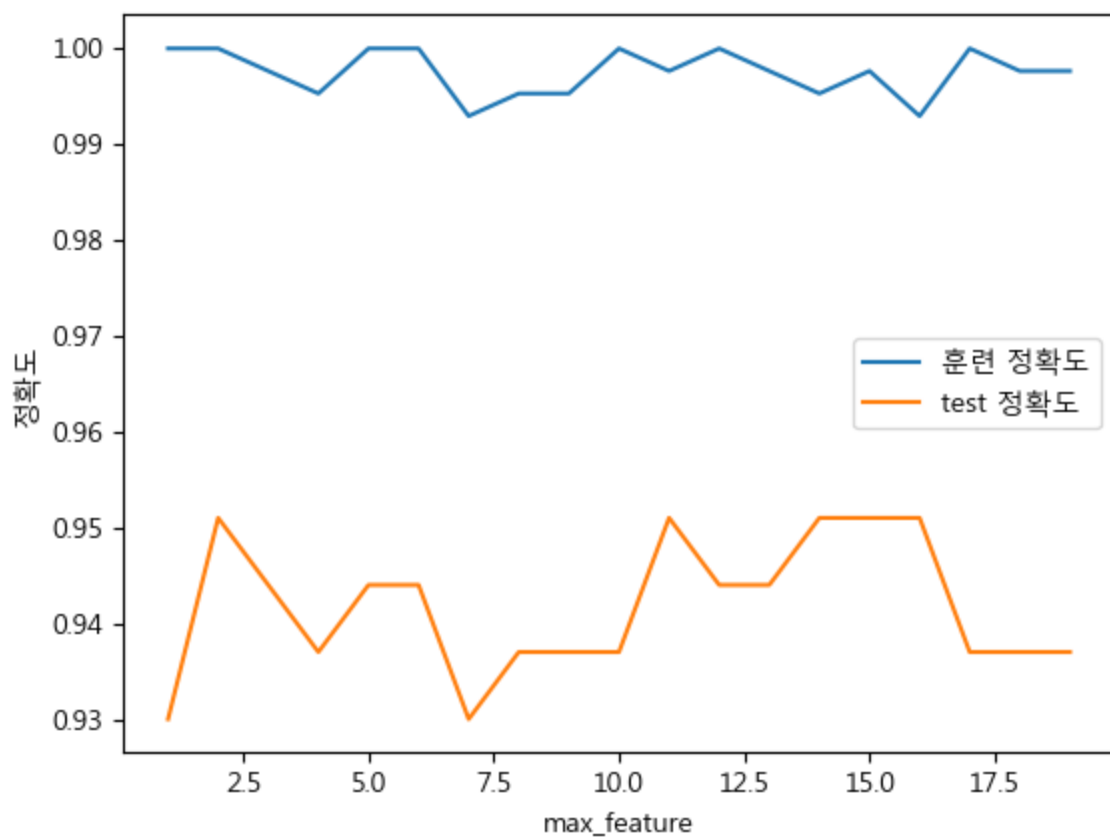
```
Out[16]: Text(38.7222,0.5,'정확도')
```

```
In [17]: plt.xlabel("max_feature")
```

```
Out[17]: Text(0.5,24.4122,'max_feature')
```

```
In [18]: plt.legend()
```

```
Out[18]: <matplotlib.legend.Legend at 0xbcef978>
```



-max_features=1 --> 불순도 고려 X(무작위로 설정된 하나의 설명변수->모델 복잡, depth 커짐)

--> 현재 결과로는 16,14,11,15,2 좀 낮은 정도!

cf.

```
In [30]: (Series(train_acc)-Series(test_acc)).sort_values()
```

```
Out[30]:
```

```
15    0.041909
```

13	0.044256
10	0.046604
14	0.046604
1	0.048951
2	0.053597
12	0.053597
4	0.055944
5	0.055944
11	0.055944
3	0.058242
7	0.058242
8	0.058242
18	0.060590
17	0.060590
6	0.062888
9	0.062937
16	0.062937
0	0.069930

dtype: float64

+ 2. max_depth

: 사전 가지치기 옵션, 연속된 질문을 제한, 작을수록 과대적합 해소

```
In [31]: train_acc, test_acc=[], []
```

```
In [32]: for max_feature in feature_setting :
```

```
...:     forest=RandomForestClassifier(max_depth=max_feature).fit(x_train,y_train)
```

```
...:     train_acc.append(forest.score(x_train,y_train))
```

```
...:     test_acc.append(forest.score(x_test,y_test))
```

```
...:
```

```
In [33]: plt.plot(feature_setting, train_acc, label="훈련 정확도")
```

```
Out[33]: [<matplotlib.lines.Line2D at 0x930bc18>]
```

```
In [34]: plt.plot(feature_setting, test_acc, label="test 정확도")
```

```
Out[34]: [<matplotlib.lines.Line2D at 0xa356e80>]
```

```
In [36]: plt.ylabel("정확도")
```

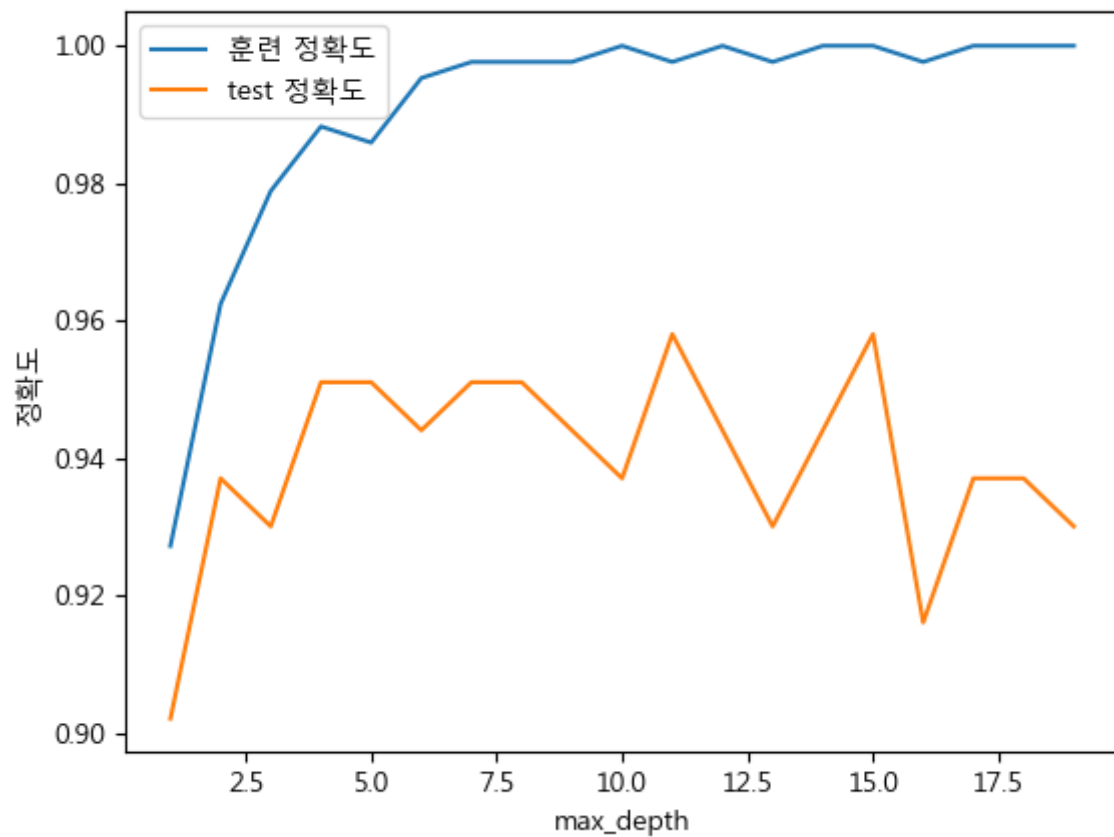
```
Out[36]: Text(38.7222,0.5,'정확도')
```

```
In [37]: plt.xlabel("max_depth")
```

```
Out[37]: Text(0.5,24.4122,'max_depth')
```

```
In [38]: plt.legend()
```

```
Out[38]: <matplotlib.legend.Legend at 0xa407f98>
```



```
In [44]: (Series(train_acc)-Series(test_acc)).sort_values()
```

```
Out[44]:
```

```
0    0.025132
1    0.025378
4    0.034867
3    0.037214
10   0.039611
14   0.041958
6    0.046604
7    0.046604
2    0.048803
5    0.051249
8    0.053597
11   0.055944
13   0.055944
16   0.062937
9    0.062937
17   0.062937
```

12	0.067583
18	0.069930
15	0.081569

dtype: float64

3. n_estimators

: 트리의 수, 클수록 과대적합을 줄여 더 안정적인 모델 학습, 메모리 사용률 고려해야 함

```
In [31]: train_acc, test_acc=[], []
```

```
In [49]: n_estimator=range(10,201,10)
```

```
In [50]: for max_feature in n_estimator :
```

```
...:     forest=RandomForestClassifier(n_estimators=max_feature).fit(x_train,y_train)
...:     train_acc.append(forest.score(x_train,y_train))
...:     test_acc.append(forest.score(x_test,y_test))
...:
```

```
In [53]: plt.plot(n_estimator, train_acc, label="훈련 정확도")
```

```
Out[53]: [<matplotlib.lines.Line2D at 0xd5404e0>]
```

```
In [54]: plt.plot(n_estimator, test_acc, label="test 정확도")
```

```
Out[54]: [<matplotlib.lines.Line2D at 0xd540cc0>]
```

```
In [55]: plt.ylabel("정확도")
```

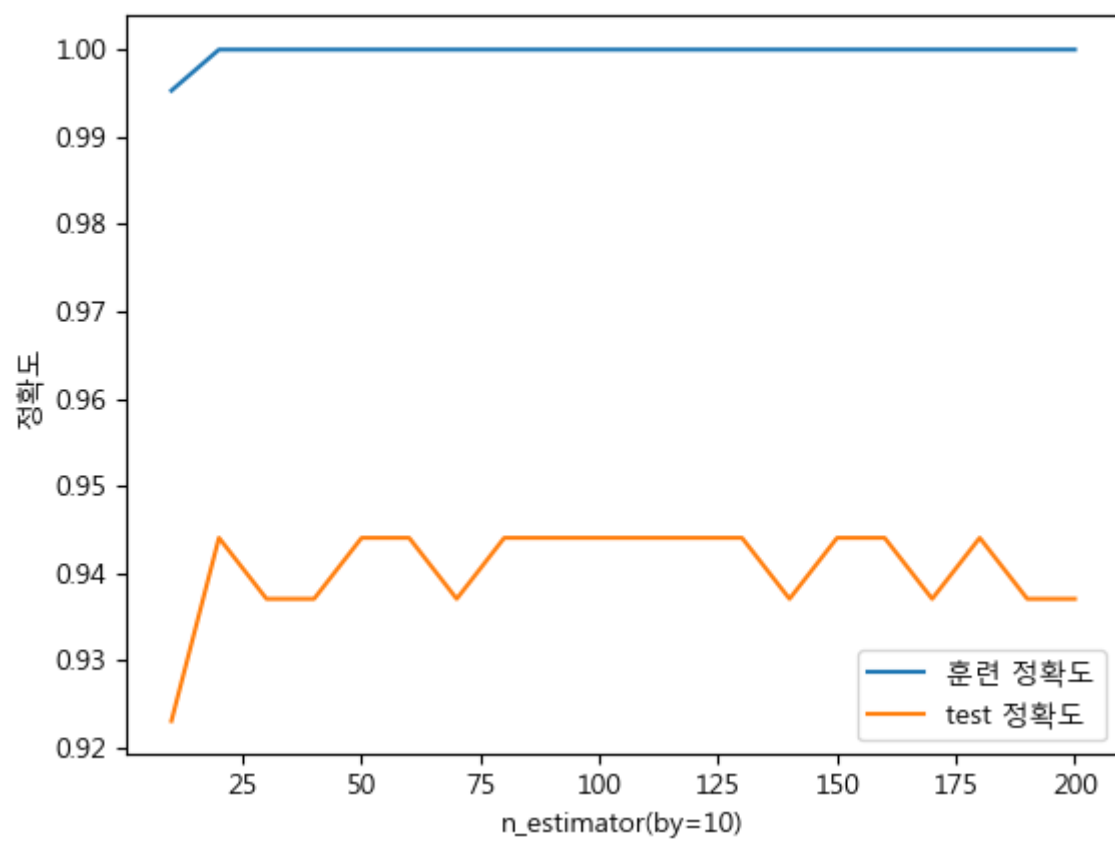
```
Out[55]: Text(38.7222,0.5,'정확도')
```

```
In [58]: plt.xlabel("n_estimator(by=10)")
```

```
Out[58]: Text(0.5,24.4122,'n_estimator(by=10)')
```

```
In [57]: plt.legend()
```

```
Out[57]: <matplotlib.legend.Legend at 0xa3e1d30>
```



0711

그래디언트 부스팅 회귀 트리 (주목적 : 분류)

- 부스팅 기법

: 잘못 분류된 객체들에 집중하여 새로운 분류규칙을 생성하는 단계를 순차적으로 반복하는 알고리즘

(depth 낮게 시작)

->(반복)오분류 된 객체에 가중치 부여 -> (반복)잘 식별, 분류하도록 새로 구성

그래디언트 부스팅 회귀 트리

- 여러 개의 결정 트리를 묶어 강력한 모델을 만드는 또 다른 앙상블 방법
- 회귀와 분류 모두에 사용 (주목적 : 분류)
- 랜덤 포레스트와는 달리, 이전 트리의 오차를 보완하는 방식으로 순차적으로 트리 생성
- 트리 생성의 무작위성이 존재하지 않는 대신, 강력한 사전 가지치기 사용
- 보통 하나에서 다섯 정도의 깊이 얕은 트리를 사용(최소), 적은 메모리 사용과 빠른 예측 수행
- 얕은 트리 같은 간단한 모델(weak learner)을 연결하는 알고리즘 (by learning_rate)
- 각 트리는 데이터의 일부에 대해서만 예측을 잘 수행하므로, 트리가 많이 추가될수록 예측력이 높음 (but overfit 현상 발생)
- random forest에 비해 적은 트리로도 높은 예측력을 보임

장점

: 특성의 스케일을 조정하지 않아도 되고, 이진 특성이거나 연속적인 특성에도 잘 동작함

단점

- : 매개 변수에 대한 조정 필요
- : 훈련 시간이 다소 길어지는 현상
- : 트리 기반 모델의 특성상 희소한 고차원 데이터에는 잘 작동하지 않음

★ 중요 매개변수 ★ -

- max_depth : 각 트리의 복잡도를 낮추는 매개변수 (보통 5보다 작은 값 사용)
- learning_rate : (기본값 0.1) 이전 트리의 오차 보정 정도(가중치), 값이 클수록 모델의 복잡도가 커짐(depth ↑, overfit)

이전 트리의 오차를 보정하는 정도를 조정

- n_estimators : 트리의 수, 값이 클수록 모델의 복잡도가 커짐(러닝레이트 때문, 기준 ↑ -> overfit)
=> 예측력에 대한 설명 X (cf. 랜덤 포레스트)

매개변수 조절 후 test 데이터와의 예측률 확인 필요! + overfit도 확인

-- 매개변수간 상관관계--

- learning_rate를 낮추면 비슷한 복잡도의 모델을 만들기 위해 더 많은 트리를 추가해야 함
- n_estimators가 클수록 좋은 랜덤 포레스트와는 달리, 그래디언트 부스팅에서는 n_estimators를

크게 하면 모델이 복잡해지고 과대적합 될 가능성이 높아짐

예시 : 유방암 데이터 셋

```
In [51]: from sklearn.datasets import load_breast_cancer
```

```
In [52]: cancer=load_breast_cancer()
```

```
In [7]: from sklearn.ensemble import GradientBoostingClassifier
```

```
In [8]: x_train,x_test,y_train,y_test = train_test_split(cancer.data, cancer.target, random_state=0)
```

```
In [9]: gbrt=GradientBoostingClassifier(random_state=0)
```

```
In [10]: gbrt.fit(x_train,y_train)
```

```
Out[10]:
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           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, n_estimators=100,
                           presort='auto', random_state=0, subsample=1.0, verbose=0,
                           warm_start=False)
```

```
In [11]: gbrt.score(x_train,y_train)
```

```
Out[11]: 1.0
```

```
In [12]: gbrt.score(x_test,y_test)
```

```
Out[12]: 0.958041958041958
```

1. 과대 적합 경향이 있는가? 2. 있다면 해소 방안은?

1. 때에 따라 다르다!

2. 만일 있다면 learning_rate 낮추기 / n_estimators 낮추기 / max_depth..?

1. max_depth(변수 기준으로 재사용횟수) 조절

In [13]: **gbrt=GradientBoostingClassifier(random_state=0, max_depth=1)**

In [14]: **gbrt.fit(x_train,y_train)**

Out[14]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=1,
                           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, n_estimators=100,
                           presort='auto', random_state=0, subsample=1.0, verbose=0,
                           warm_start=False)
```

In [15]: **gbrt.score(x_train,y_train)**

Out[15]: 0.9906103286384976

In [16]: **gbrt.score(x_test,y_test)**

Out[16]: 0.972027972027972

2. learning_rate 조절

In [17]: **gbrt=GradientBoostingClassifier(random_state=0, learning_rate=0.01)**

In [18]: **gbrt.fit(x_train,y_train)**

Out[18]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.01, loss='deviance', max_depth=3,
                           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, n_estimators=100,
                           presort='auto', random_state=0, subsample=1.0, verbose=0,
                           warm_start=False)
```

In [19]: **gbrt.score(x_train,y_train)**

Out[19]: 0.9882629107981221

In [21]: **gbrt.score(x_test,y_test)**

Out[21]: 0.965034965034965

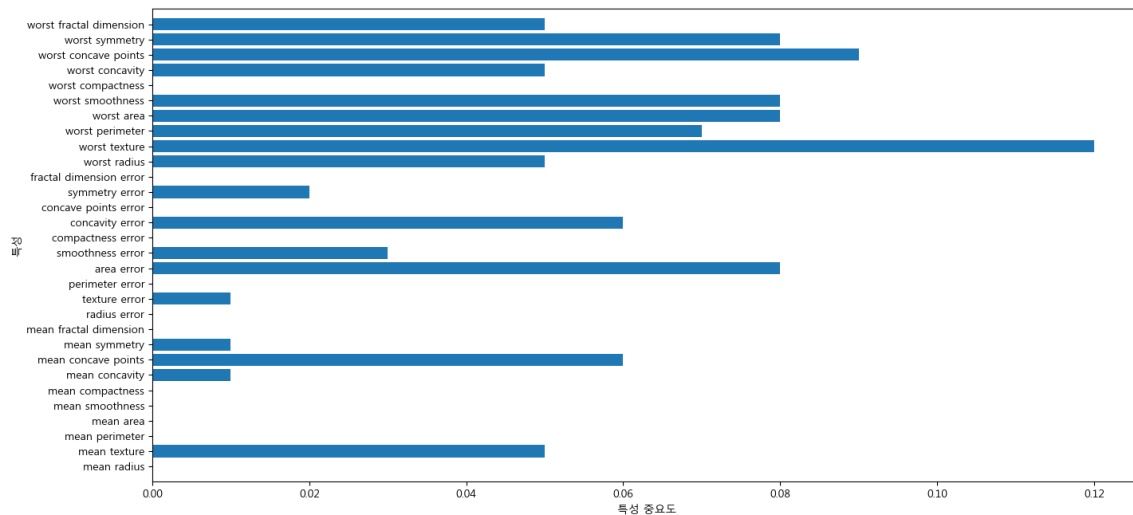
##해석

learning_rate를 낮추는 것은 테스트 셋 성능을 조금밖에 개선하지 못했지만,
max_depth를 낮추는 것은 모델 향상에 크게 기여

시각화 결과

```
def plot_feature_importances_cancer(model):  
    n_features = cancer.data.shape[1]  
    plt.barh(range(n_features), model.feature_importances_,  
align='center')  
    plt.yticks(np.arange(n_features), cancer.feature_names)  
    plt.xlabel("특성 중요도")  
    plt.ylabel("특성")  
    plt.ylim(-1, n_features)
```

In [121]: **plot_feature_importances_cancer(gbrt)**



for문으로 적합 값 찾기 방법

1. max_depth

In [22]: **train_acc, test_acc=[], []**

In [23]: **k =range(1,21)**

In [24]: **x_train, x_test, y_train, y_test=train_test_split(cancer.data, cancer.target, random_s
...: tate=0)**

In [25]: **for i in k :**

```
...:     gbrt=GradientBoostingClassifier(random_state=0, max_depth=i).fit(x_train,y_train)  
...:     train_acc.append(gbrt.score(x_train,y_train))  
...:     test_acc.append(gbrt.score(x_test,y_test))  
...:
```

In [30]: **plt.plot(k, train_acc, label="훈련 정확도")**

Out[30]: [

In [31]: **plt.plot(k,test_acc,label="테스트 정확도")**

Out[31]: [

In [35]: **plt.rc("font",family="Malgun Gothic")**

In [32]: **plt.xlabel("max_depth")**

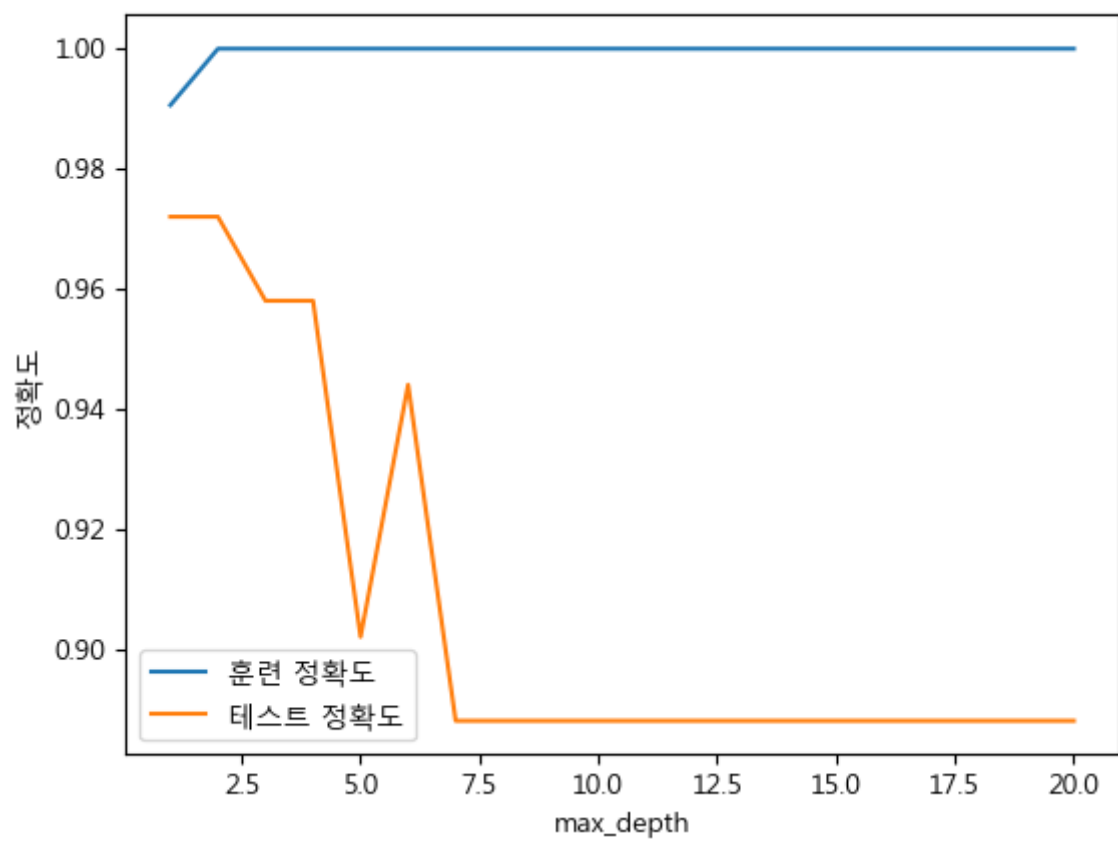
Out[32]: Text(0.5,23.4122,'max_depth')

In [33]: **plt.ylabel("정확도")**

Out[33]: Text(33.8472,0.5,'정확도')

In [34]: **plt.legend()**

Out[34]: <matplotlib.legend.Legend at 0xb2d0f98>



2. learning_rate 조절

In [45]: **k=np.arange(0.001,0.051,0.001)**

In [46]: **train_acc, test_acc=[], []**

In [48]: **for i in k :**

```
...:                                     gbrt=GradientBoostingClassifier(random_state=0,  
learning_rate=i).fit(x_train,y_train)  
...:     train_acc.append(gbrt.score(x_train,y_train))  
...:     test_acc.append(gbrt.score(x_test,y_test))  
...:
```

In [49]: **plt.plot(k, train_acc, label="훈련정확도")**

Out[49]: [<matplotlib.lines.Line2D at 0xa299320>]

In [50]: **plt.plot(k, test_acc, label="테스트정확도")**

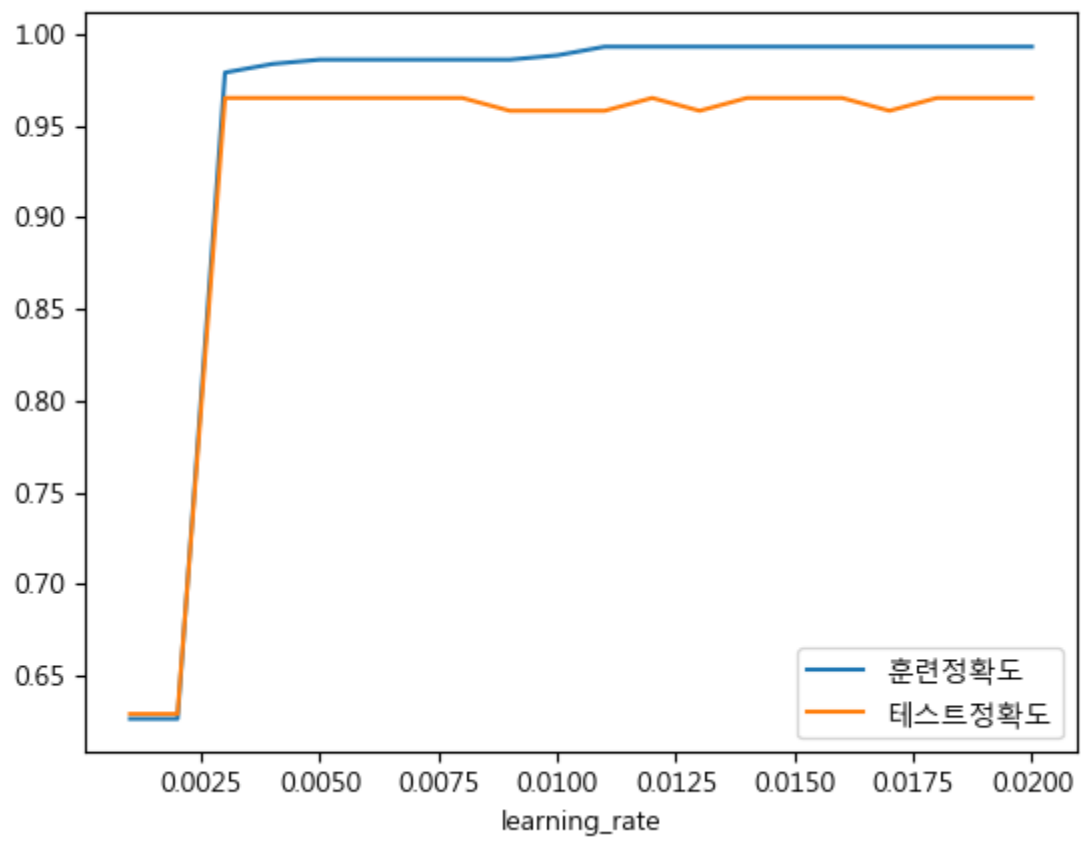
Out[50]: [<matplotlib.lines.Line2D at 0xcd35908>]

In [51]: **plt.xlabel("learning_rate")**

Out[51]: Text(0.5,24.4122,'learning_rate')

In [52]: **plt.legend()**

Out[52]: <matplotlib.legend.Legend at 0xb304a58>



3. n_estimators 조절

In [62]: `k=arange(10,251,10)`

In [63]: `train_acc, test_acc=[], []`

In [48]: `for i in k :`

`gbrt=GradientBoostingClassifier(random_state=0, n_estimators=i).fit(x_train,y_train)`

`train_acc.append(gbrt.score(x_train,y_train))`

`test_acc.append(gbrt.score(x_test,y_test))`

In [65]: `plt.plot(k, train_acc, label="훈련정확도")`

Out[65]: [`<matplotlib.lines.Line2D at 0xafaa320>`]

In [66]: `plt.plot(k, test_acc, label="테스트정확도")`

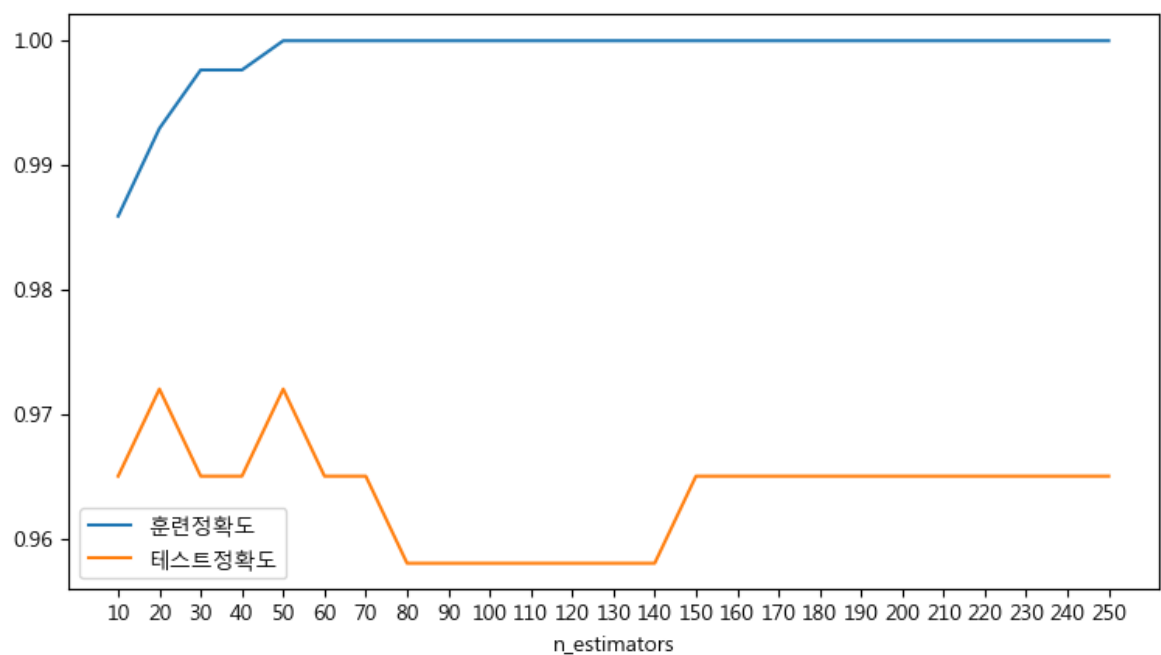
Out[66]: [`<matplotlib.lines.Line2D at 0xb0f4588>`]

In [67]: `plt.xlabel("n_estimators")`

Out[67]: `Text(0.5,24.4122,'n_estimators')`

In [68]: `plt.legend()`

Out[68]: `<matplotlib.legend.Legend at 0xb0fd400>`



cf. n_estimators 개수는 index+1

```
In [107]: (Series(train_acc)-Series(test_acc)).sort_values()
```

Out[107]:

0	0.020881
1	0.020930
4	0.027972
2	0.032618
3	0.032618
22	0.034965
21	0.034965
20	0.034965
19	0.034965
18	0.034965
17	0.034965
16	0.034965
15	0.034965
14	0.034965
24	0.034965
23	0.034965
6	0.034965
5	0.034965
11	0.041958
10	0.041958
9	0.041958
8	0.041958
7	0.041958
13	0.041958
12	0.041958

dtype: float64

독버섯 데이터 셋

<https://archive.ics.uci.edu/ml/datasets/Mushroom>에서 데이터 설명 확인 가능

- 첫 번째 열 : 독성 유무, 독성(p) / 식용(e)

- 두 번째 열 : 버섯의 머리모양

(b: 벨형태 / c: 원뿔 / x: 볼록한 형태 / f: 평평한 형태 / k: 혹 형태/ s:오목한 형태)

- 네 번째 열 : 버섯의 머리 색

(n: 갈색 / b: 황갈색 / c: 연한 갈색 / ...)

[--> kNN, RandomForest, 그래디언트부스팅 해보기](#)

----데이터 셋 가공 및 모델 생성---

```
import urllib.request as req
local= "mushroom.csv"
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-
lepiota.data"
req.urlretrieve(url, local)
print("ok")
```

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

데이터 loading

```
mr = pd.read_csv("mushroom.csv", header=None)
```

데이터 내부의 기호를 숫자로 변환하기

```
target = []
```

```
data = []
```

```
attr_list = []
```

```
for row_index, row in mr.iterrows():
```

```
    target.append(row.ix[0]) #첫 번째 열 : 독 / 식용
```

```
    row_data = []
```

```
    for v in row.ix[1:]:
```

```
        row_data.append(ord(v))
```

```
    data.append(row_data)
```

data split

```
X_train, X_test, y_train, y_test = train_test_split(data, target)
```

#일단 target은 factor화 아니긴 함 ...!

cf. iterrows -- index와 row 분리해서 한 행으로 저장

In [97]: Series(mr.iloc[:4,:4].iterrows())

Out[97]:

0 (0, [p, x, s, n])

1 (1, [e, x, s, y])

2 (2, [e, b, s, w])

3 (3, [p, x, y, w])

dtype: object

1. 그래디언트 부스팅

```
In [104]: gbrt=GradientBoostingClassifier(random_state=0).fit(x_train,y_train)
```

```
In [105]: gbrt.score(x_train,y_train)
```

```
Out[105]: 1.0
```

```
In [106]: gbrt.score(x_test,y_test)
```

```
Out[106]: 1.0
```

-----조절해볼 필요가 있나?0?

각 특성의 중요도 분석

```
In [107]: n_features=np.array(data).shape[1]
```

```
In [110]: plt.barh(range(n_features), gbrt.feature_importances_, align='center')
```

```
Out[110]: <BarContainer object of 22 artists>
```

```
In [111]: plt.yticks(np.arange(n_features))
```

```
In [112]: plt.xlabel("특성 중요도")
```

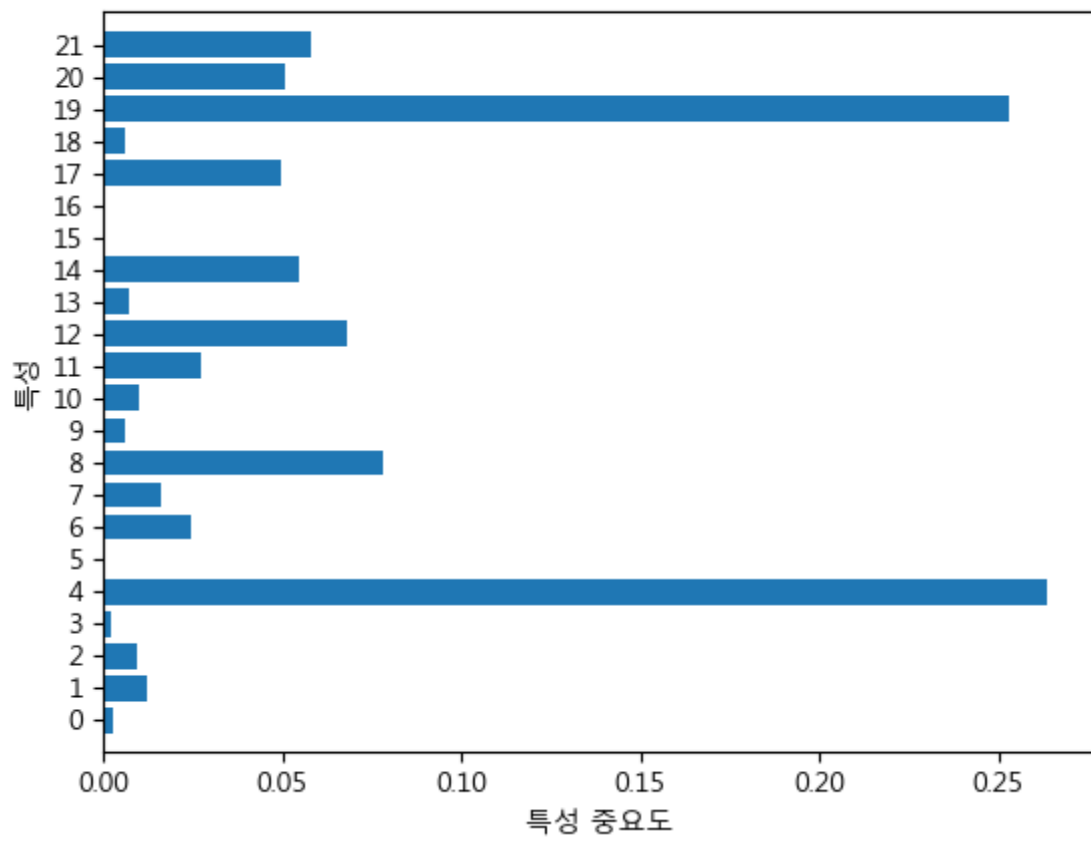
```
Out[112]: Text(0.5,24.4122,'특성 중요도')
```

```
In [113]: plt.ylabel("특성")
```

```
Out[113]: Text(49.3472,0.5,'특성')
```

```
In [115]: plt.ylim(-1,n_features) # 미세조정
```

```
Out[115]: (-1, 22)
```



2.random forest 하기

```
In [155]: from sklearn.ensemble import RandomForestClassifier
```

```
In [156]: forest=RandomForestClassifier().fit(x_train,y_train)
```

```
In [158]: forest.score(x_train,y_train)
```

```
Out[158]: 1.0
```

```
In [159]: forest.score(x_test,y_test)
```

```
Out[159]: 1.0
```

각 특성의 중요도 분석

```
In [160]: n_features=np.array(data).shape[1]
```

```
In [163]: plt.barh(range(n_features), forest.feature_importances_, align='center')
```

```
Out[163]: <BarContainer object of 22 artists>
```

```
In [164]: plt.yticks(np.arange(n_features))
```

```
In [165]: plt.xlabel("특성 중요도")
```

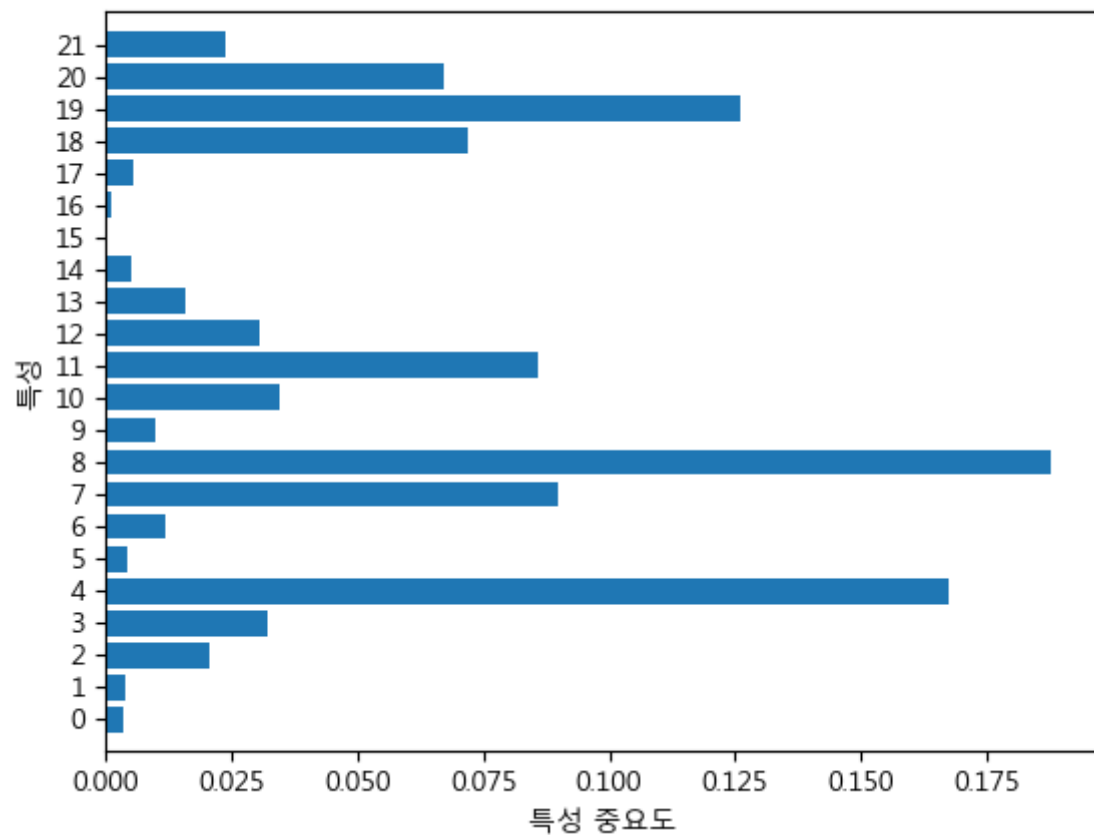
```
Out[165]: Text(0.5,24.4122,'특성 중요도')
```

```
In [166]: plt.ylabel("특성")
```

```
Out[166]: Text(49.3472,0.5,'특성')
```

```
In [167]: plt.ylim(-1,n_features)
```

```
Out[167]: (-1, 22)
```



특성중요도 :

in 독버섯

-> 특성중요도 값이 높을수록 독버섯인지 식용버섯인지 분류하는 데에 끼치는 영향력이 높음!

분류 할 때 상위에 배치될 가능성 높음--확실한 기준 (in RandomForest)

// 해석이 확실할 때엔 특성 중요도를 빼고 새로 Random Forest 하는 편이 모델의 복잡도를 낮추므로 확실하게 뺄 수 있는 정보면 빼는 편이 좋다.

3. kNN하기

-> 0 / 1 나눠져야 할 것 같아서 위의 결과 target / data 사용하되

```
In [139]: f1=lambda x : int(x.replace('e', '0').replace('p', '1'))
```

```
In [138]: target1=list(map(f1, target))
```

#target1로 변경! 팩터화

+ scale 조정도 필요했다...! 여기선 안 했지만.. π - π

StandardScaler().fit(data).transform(data) 인가?

```
In [140]: x_train,x_test,y_train,y_test=train_test_split(data,target1)
```

```
In [141]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [145]: train_acc,test_acc=[],[]
```

```
In [146]: k=range(1,11)
```

```
In [148]: for i in k :^M
```

```
...:     knn=KNeighborsClassifier(n_neighbors=i)
```

```
...:     knn.fit(x_train, y_train)
```

```
...:     train_acc.append(knn.score(x_train, y_train))
```

```
...:     test_acc.append(knn.score(x_test, y_test))
```

```
In [150]: plt.plot(k, train_acc, label="훈련 정확도")
```

```
Out[150]: [<matplotlib.lines.Line2D at 0xe37b940>]
```

```
In [151]: plt.plot(k, test_acc, label="테스트 정확도")
```

```
Out[151]: [<matplotlib.lines.Line2D at 0xd16bc50>]
```

```
In [152]: plt.ylabel("정확도")
```

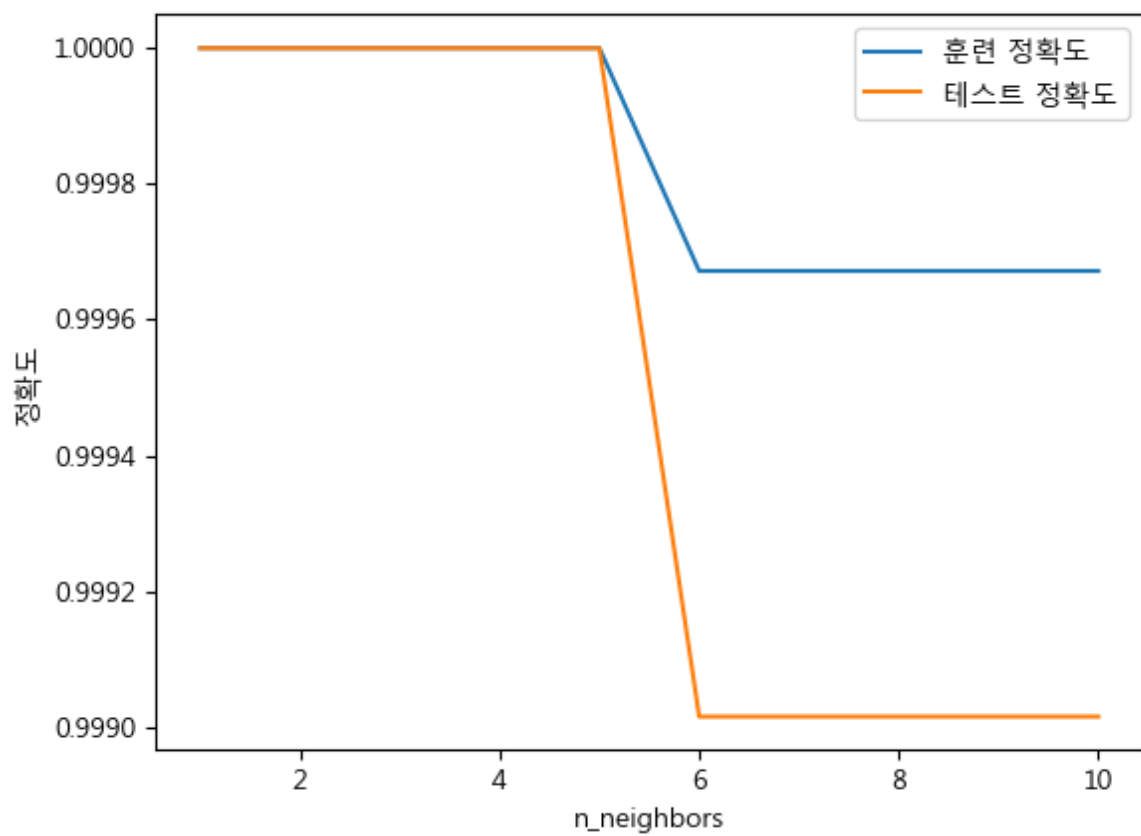
```
Out[152]: Text(23.4722,0.5,'정확도')
```

```
In [153]: plt.xlabel("n_neighbors")
```

```
Out[153]: Text(0.5,24.4122,'n_neighbors')
```

```
In [154]: plt.legend()
```

```
Out[154]: <matplotlib.legend.Legend at 0xe3986d8>
```



SVC 모델 - 커널 서포트 벡터 머신

- 훈련 시간이 오래 걸림
- 비선형 의사결정 영역을 모형화 가능
- 과대적합 되는 경향이 매우 낮아 주로 분류 연구에 사용

→ 비선형SVM : 실제 현실에서 발생하는 분류 문제는 단순한 이진화로 해결하기 어려움

→ 커널함수 : 클래스간의 경계를 고차원의 새로운 공간으로 매핑하여 선형분리 가능

- Linear SVC(Classifier) / non-LinearSVC(kernel=linear)

non-LinearSVC(kernel=polynomial) / non-LinearSVC(kernel = rbf)

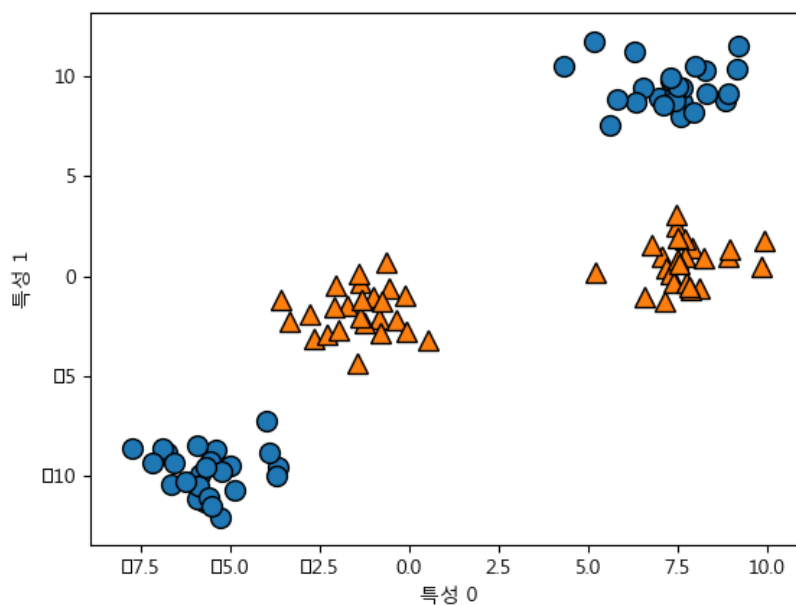
- 인위적 데이터 셋(이진 분류 데이터 셋) 연습

In [195]: from sklearn.datasets import make_blobs

In[77]:

```
X, y = make_blobs(centers=4, random_state=8)
y = y % 2

mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("특성 0")
plt.ylabel("특성 1")
```

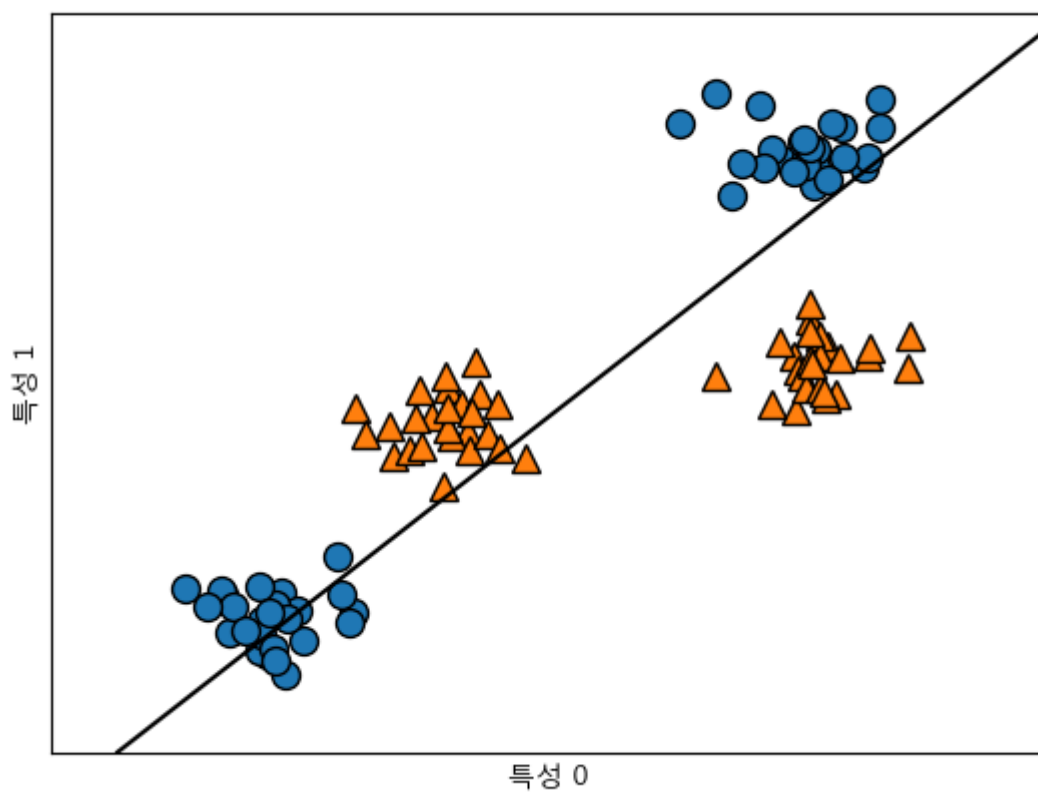


#선형으로 구분되지 않는 클래스의 선형 분류 모델적용

In[78]:

```
from sklearn.svm import LinearSVC
linear_svm = LinearSVC().fit(X, y)

mglearn.plots.plot_2d_separator(linear_svm, X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("특성 0")
plt.ylabel("특성 1")
```



#2 차원(선형)으로 분류가 안 됨 --> 3 차원으로 확장 필요

#특성을 추가하여 고차원 데이터로 확장

```
In [14]: X_new=np.hstack([X,X[:,1:]**2])
```

두 번째 특성을 제공하여 추가한다.

```
In [16]: from mpl_toolkits.mplot3d import Axes3D, axes3d
```

```
In [17]: figure=plt.figure()
```

```
In [18]: ax=Axes3D(figure,elev=-152,azim=-26)
```

Axes3D(figure_name, elev(고도), zim(좌우각도))

```
In [19]: mask=y==0
```

y==0 인 포인트를 먼저 그리고 다음 y==1 인 포인트를 그림.

```
In [20]:
```

```
ax.scatter(X_new[mask,0],X_new[mask,1],X_new[mask,2],c='b',cmap=mglearn.cm2,s=60,edgec
...: olor='k')
```

```
Out[20]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0xbf962b0>
```

```
In [21]: ax.scatter(X_new[~mask,0],X_new[~mask,1],X_new[~mask,2],c='r',marker='^',cmap=mglearn.
...: cm2,s=60,edgecolor='k')
```

```
Out[21]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0xbf405c0>
```

```
In [22]: ax.set_xlabel("특성 0")
```

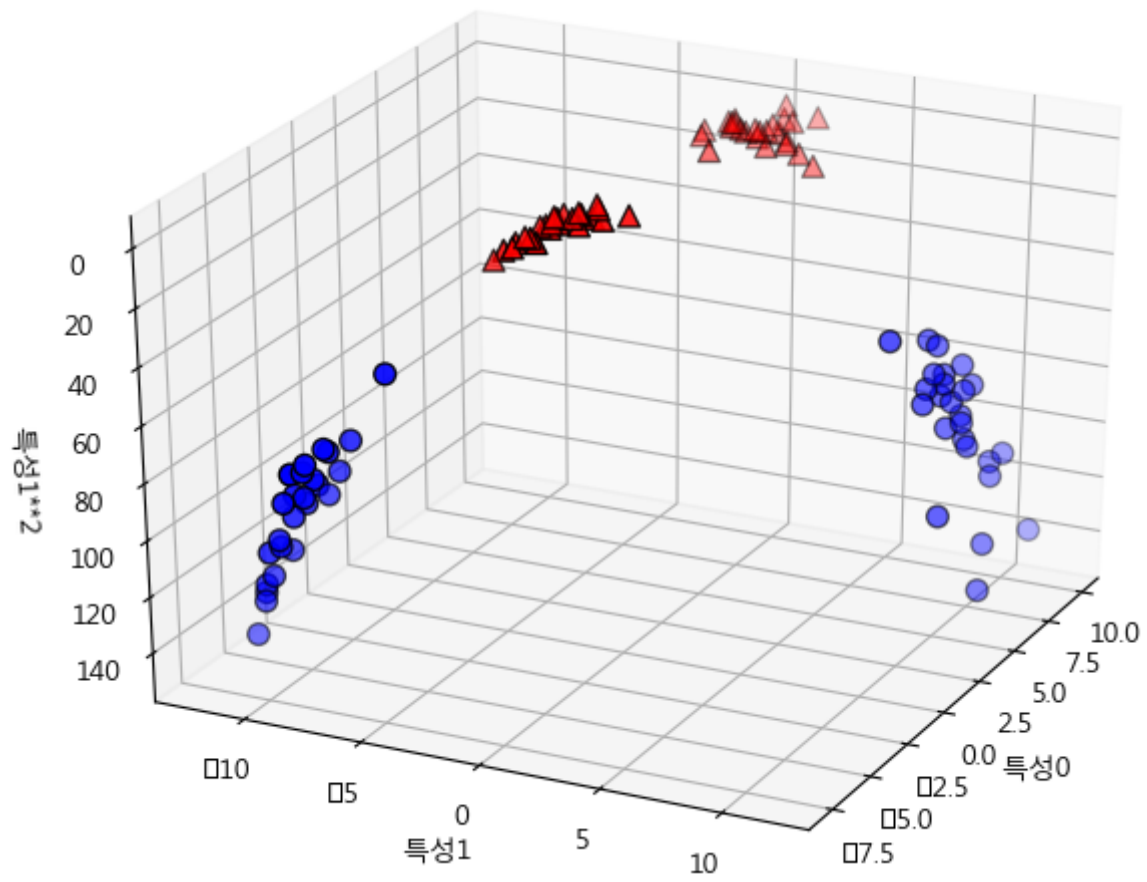
```
Out[22]: Text(0.0576614,-0.0698479,'특성 0')
```

```
In [23]: ax.set_ylabel("특성 1")
```

```
Out[23]: Text(-0.0288765,-0.085957,'특성 1')
```

```
In [25]: ax.set_zlabel("특성 1**2")
```

```
Out[25]: Text(-0.0885628,-0.0143164,'특성 1**2')
```



#선형모델과 3 차원 공간의 평면을 사용한 클래스 분류

```
In [37]: linear_svm_3d=LinearSVC().fit(X_new,y)
```

```
In [38]: coef, intercept=linear_svm_3d.coef_.ravel(), linear_svm_3d.intercept_
```

#선형 결정 경계 그리기

```
In [39]: figure=plt.figure()
```

```
In [40]: ax=Axes3D(figure,elev=-152, azim=-26)
```

```
In [41]: xx=np.linspace(X_new[:,0].min() - 2, X_new[:,0].max() + 2, 50)
```

```
In [42]: yy=np.linspace(X_new[:,1].min() - 2, X_new[:,1].max() + 2, 50)
```

```
In [43]: XX, YY=np.meshgrid(xx,yy)
```

```
In [44]: ZZ=(coef[0]*XX+coef[1]*YY+intercept)/-coef[2]
```

```
In [51]: ax.plot_surface(XX,YY,ZZ, rstride=8, cstride=8, alpha=0.3)
```

```
Out[51]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0xbdc3080>
```

```
In [53]:
```

```
ax.scatter(X_new[mask,0],X_new[mask,1],X_new[mask,2],c='b',cmap=mglearn.cm2,s=60,edgec
...: olor='k')
```

```
Out[53]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0xbddfc50>
```

```
In [54]: ax.scatter(X_new[~mask,0],X_new[~mask,1],X_new[~mask,2],c='r',marker='^',cmap=mglearn.
```

```
...: cm2,s=60,edgecolor='k')
```

```
Out[54]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0xc17c160>
```

```
In [55]: ax.set_xlabel("특성 0")
```

```
Out[55]: Text(0.057537,-0.0699981,'특성 0')
```

```
In [56]: ax.set_ylabel("특성 1")
```

```
Out[56]: Text(-0.0291631,-0.0858793,'특성 1')
```

```
In [57]: ax.set_zlabel("특성 1 ** 2")
```

```
Out[57]: Text(-0.0886773,-0.0141067,'특성 1 ** 2')
```

#linspace(x,y,z) : x~y 범위의 등간격 z 개의 벡터 출력

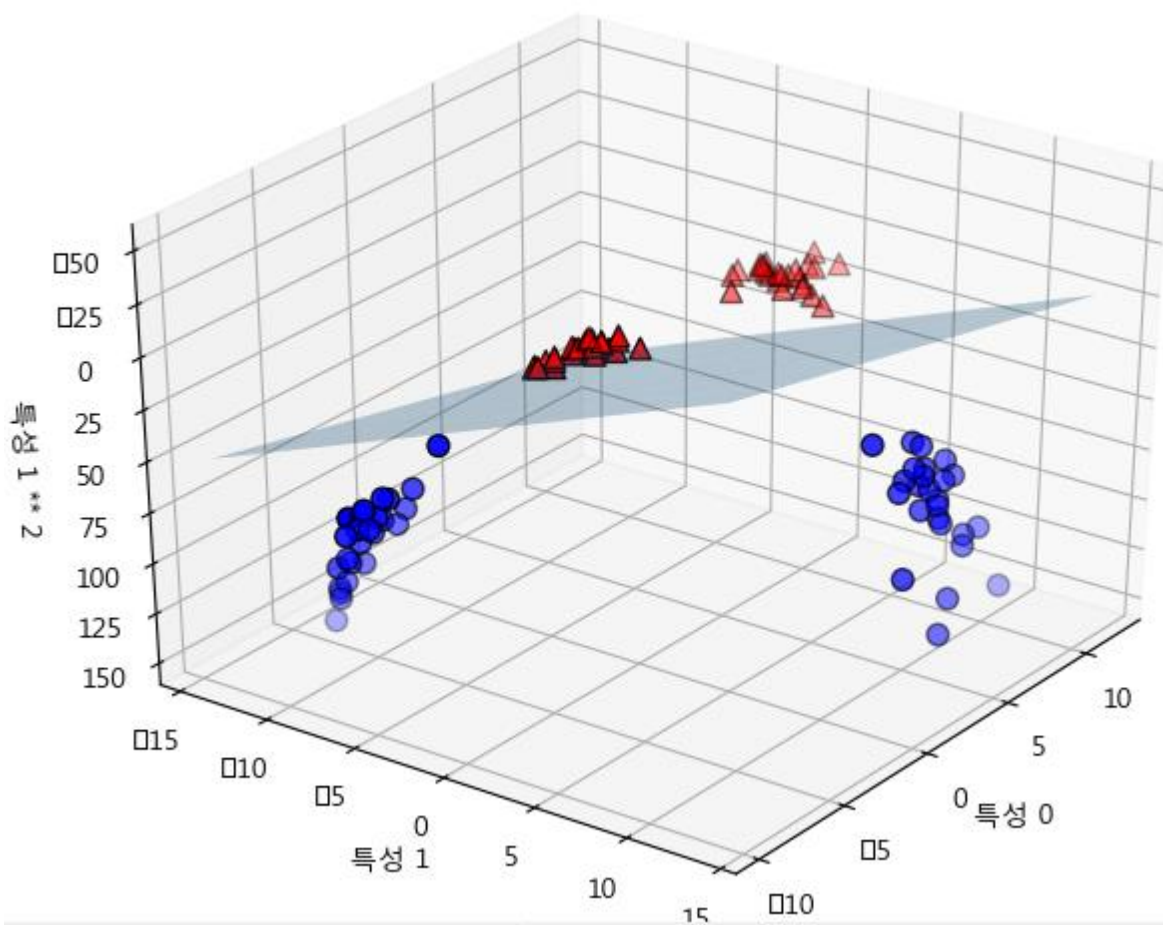
#meshgrid : 벡터 기준 브로드캐스팅 함수

Examples

```
>>> nx, ny = (3, 2)
>>> x = np.linspace(0, 1, nx)
>>> y = np.linspace(0, 1, ny)
>>> xv, yv = np.meshgrid(x, y)
>>> xv
array([[ 0.,  0.5,  1. ],
       [ 0.,  0.5,  1.]])
>>> yv
array([[ 0.,  0.,  0.],
       [ 1.,  1.,  1.]])
>>> xv, yv = np.meshgrid(x, y, sparse=True) # make sparse output arrays
>>> xv
array([[ 0.,  0.5,  1.]])
>>> yv
array([[ 0.],
       [ 1.]])
```

`meshgrid` is very useful to evaluate functions on a grid.

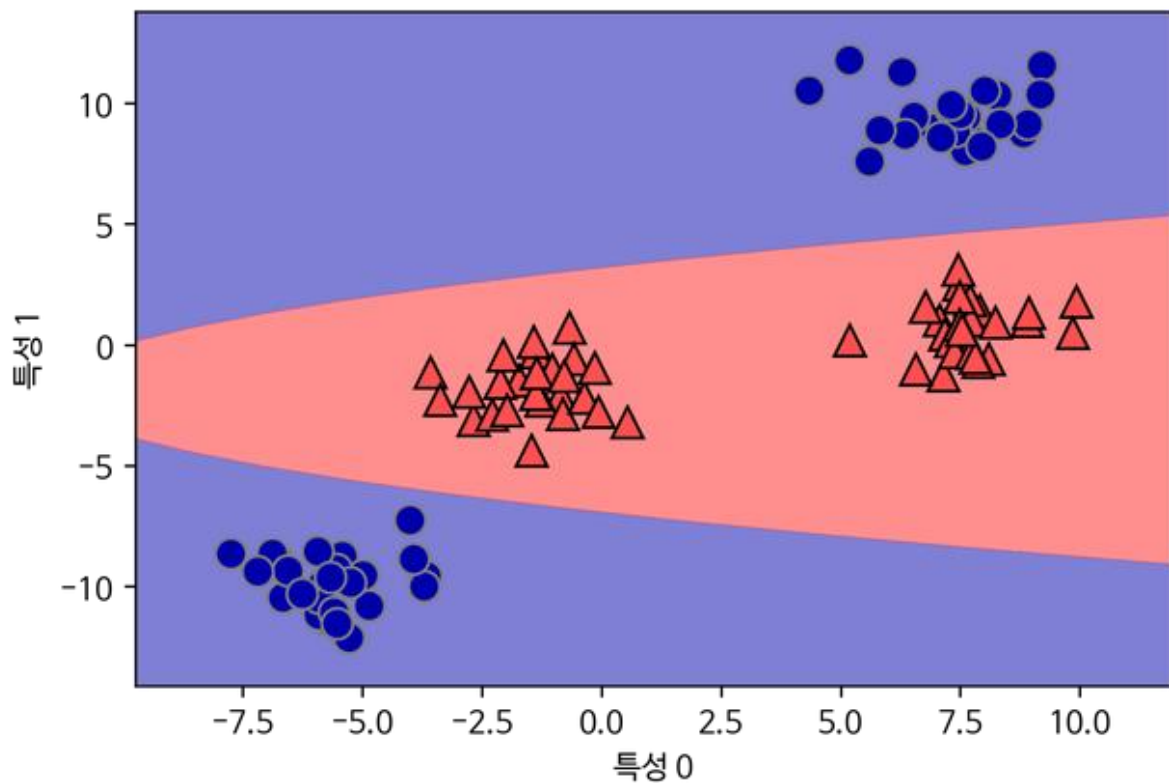
```
>>> x = np.arange(-5, 5, 0.1)
>>> y = np.arange(-5, 5, 0.1)
>>> xx, yy = np.meshgrid(x, y, sparse=True) #xx.shape : (1,100) yy.shape : (100,1)
>>> z = np.sin(xx**2 + yy**2) / (xx**2 + yy**2) #z.sahpe : (100,100)
>>> h = plt.contourf(x,y,z) #??
```



원래 특성으로 투영해보면 이 선형 SVM 모델은 더 이상 선형이 아닙니다. 직선보다 타원에 가까운 모습을 [그림 2-40]에서 확인할 수 있습니다.

In[81]:

```
ZZ = YY ** 2
dec = linear_svm_3d.decision_function(np.c_[XX.ravel(),
YY.ravel(), ZZ.ravel()])
plt.contourf(XX, YY, dec.reshape(XX.shape), levels=[dec.min(),
0, dec.max()],
cmap=mglearn.cm2, alpha=0.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("특성 0")
plt.ylabel("특성 1")
```



커널 기법

앞에서는 데이터셋에 비선형 특성을 추가하여 선형 모델을 강력하게 만들었습니다. 하지만 많은 경우 어떤 특성을 추가해야 할지 모르고 특성을 많이 추가하면 (예를 들면, 100 개의 특성에서 가능한 모든 조합) 연산 비용이 커집니다. 다행히 수학적 기교를 사용해서 새로운 특성을 많이 만들지 않고서도 고차원에서 분류기를 학습시킬 수 있습니다.

이를 **커널 기법** kernel trick 이라고 하며 실제로 데이터를 확장하지 않고 확장된 특성에 대한 데이터 포인트들의 거리(더 정확히는 스칼라 곱)를 계산합니다.

서포트 벡터 머신에서 데이터를 고차원 공간에 매핑하는 데 많이 사용하는 방법은 두 가지입니다. 원래 특성의 가능한 조합을 지정된 차수까지 모두 계산(예를 들어 특성 1 ** 2 × 특성 2 ** 5)하는 다항식 커널이 있고 가우시안 Gaussian 커널로도 불리는 RBF radial basis function 커널이 있습니다. 가우시안 커널은 차원이 무한한 특성 공간에 매핑하는 것으로, 설명하기가 좀 더

어렵습니다. 가우시안 커널은 모든 차수의 모든 다항식을 고려한다고 이해하면 좋습니다. 하지만 특성의 중요도는 고차항이 될수록 줄어듭니다. ²

실제로는 커널 SVM 이면의 수학적 이론은 중요하지 않지만, RBF 커널을 사용한 SVM 이 결정을 만드는 방법은 비교적 쉽게 요약할 수 있습니다. 다음 절에서 이를 살펴보겠습니다.

SVM 이해하기

학습이 진행되는 동안 SVM 은 각 훈련 데이터 포인트가 두 클래스 사이의 결정 경계를 구분하는 데 얼마나 중요한지를 배우게 됩니다. 일반적으로 훈련 데이터의 일부만 결정 경계를 만드는 데 영향을 줍니다. 바로 두 클래스 사이의 경계에 위치한 데이터 포인트들입니다. 이런 데이터를 포인트를 **서포트 벡터** support vector 라 하며, 여기서 서포트 벡터 머신이란 이름이 유래했습니다.

새로운 데이터 포인트에 대해 예측하려면 각 서포트 벡터와의 거리를 측정합니다. 분류 결정은 서포트 벡터까지의 거리에 기반하며 서포트 벡터의 중요도는 훈련 과정에서 학습합니다(SVC 객체의 `dual_coef_` 속성에 저장됩니다).

데이터 포인트 사이의 거리는 가우시안 커널에 의해 계산됩니다.

$$k_{rbf}(x_1, x_2) = \exp(-\gamma ||x_1 - x_2||^2)$$

여기에서 x_1 과 x_2 는 데이터 포인트이며 $||x_1 - x_2||$ 는 유클리디안 거리이고 γ 감마, gamma는 가우시안 커널의 폭을 제어하는 매개변수입니다.

[그림 2-41]은 두 개의 클래스를 가진 2 차원 데이터셋에 서포트 벡터 머신을 학습시킨 결과를 보여줍니다. 결정 경계는 검은 실선으로, 서포트 벡터는 굵은 테두리로 크게 그렸습니다. 다음은 `forge` 데이터셋에 SVM 을 학습시켜 이 그래프를 그리는 코드입니다.

In[82]:

```
from sklearn.svm import SVC
X, y = mglearn.tools.make_handcrafted_dataset()
svm = SVC(kernel='rbf', C=10, gamma=0.1).fit(X, y)
mglearn.plots.plot_2d_separator(svm, X, eps=.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
# 서포트 벡터
sv = svm.support_vectors_
# dual_coef_의 부호에 의해 서포트 벡터의 클래스 레이블이 결정됩니다.
sv_labels = svm.dual_coef_.ravel() > 0
mglearn.discrete_scatter(sv[:, 0], sv[:, 1], sv_labels, s=15,
                          markededgewidth=3)
plt.xlabel("특성 0")
plt.ylabel("특성 1")
```

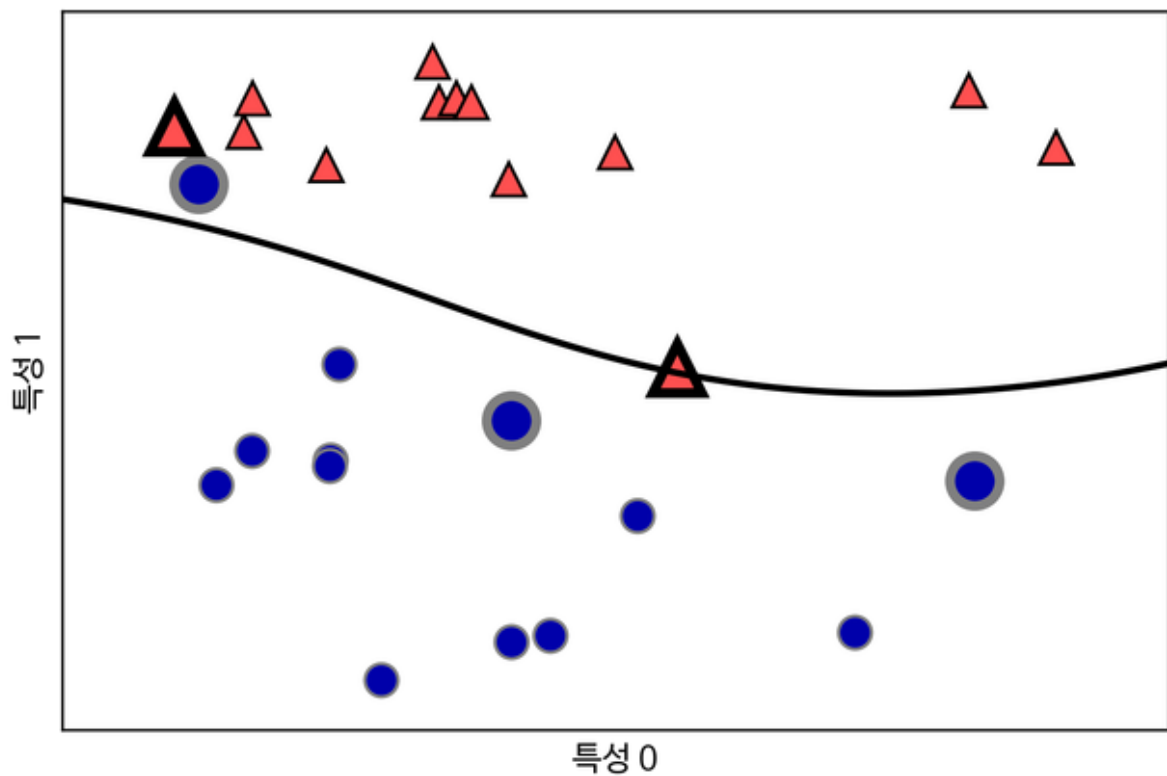


그림 2-41 RBF 커널을 사용한 SVM 으로 만든 결정 경계와 서포트 벡터

이 그림에서 SVM 은 매우 부드럽고 비선형(직선이 아닌) 경계를 만들었습니다. 여기서 사용한 두 매개변수 C 와 gamma 에 대해 자세히 살펴보겠습니다.

SVM 매개변수 튜닝 매개변수 C와 gamma

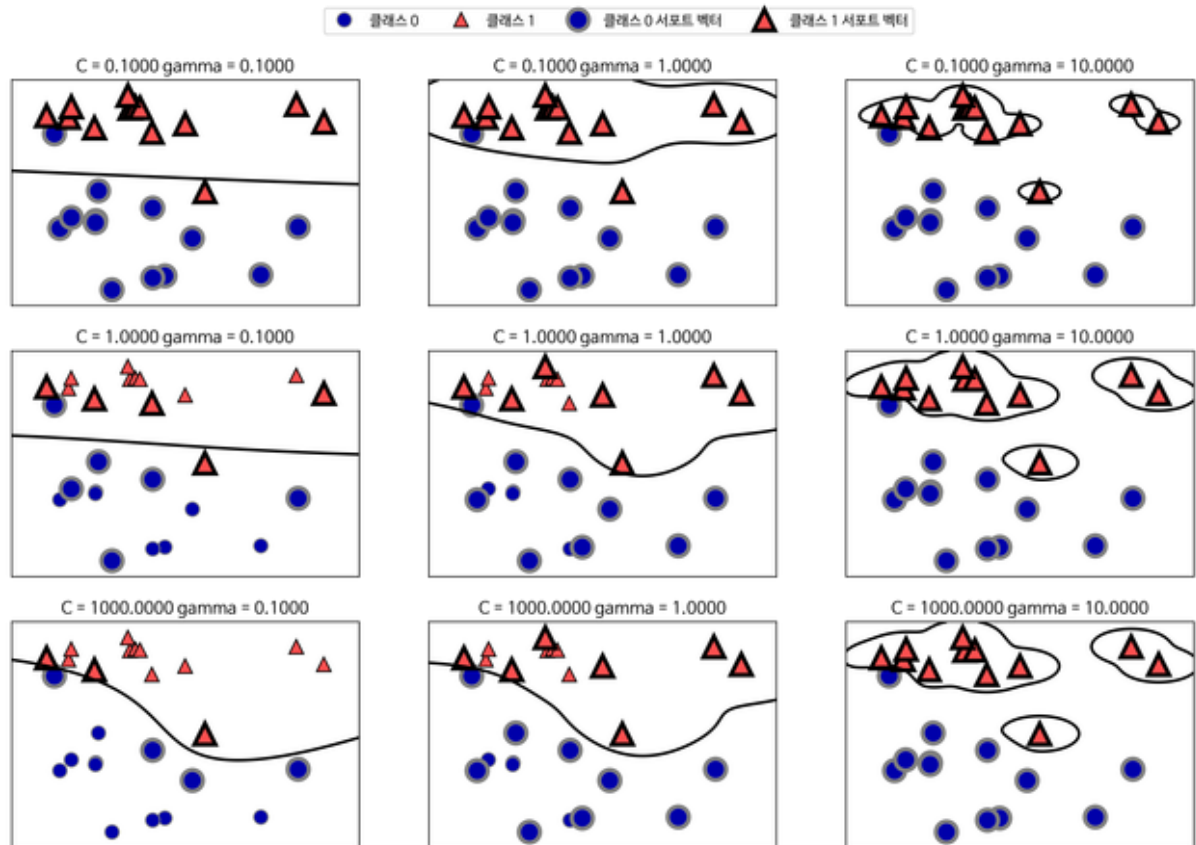
C : 선형, 비선형 // gamma : 차원변경

gamma 매개변수는 앞 절의 공식에 나와 있는 γ 로 가우시안 커널 폭의 역수에 해당합니다. gamma 매개변수가 하나의 훈련 샘플이 미치는 영향의 범위를 결정합니다. 작은 값은 넓은 영역을 의미하며 큰 값일 경우 영향이 미치는 범위가 제한적입니다. 다른 말로 하면, 가우시안 커널의 반경이 클수록 훈련 샘플의 영향 범위도 커집니다. 3 C 매개변수는 선형 모델에서 사용한 것과 비슷한 규제 매개변수입니다. 이 매개변수는 각 포인트의 중요도(정확히는 `dual_coef_` 값)를 제한합니다. - 개별 관측값 bias 부여

cf. C, 감마 0~1 의 값

In[83]:

```
fig, axes = plt.subplots(3, 3, figsize=(15, 10))
for ax, C in zip(axes, [-1, 0, 3]):
    for a, gamma in zip(ax, range(-1, 2)):
        mglearn.plots.plot_svm(log_C=C, log_gamma=gamma, ax=ax)
axes[0, 0].legend(["클래스 0", "클래스 1", "클래스 0 서포트 벡터",
                  "클래스 1 서포트 벡터"],
                  ncol=4, loc=(.9, 1.2))
```

왼쪽에서 오른쪽으로 가면서 γ 매개변수를 0.1 에서 10 으로 증가시켰습니다. 작은 γ 값은 가우시안 커널의 반경을 크게 하여 많은 포인트들이 가까이 있는 것으로 고려됩니다. 그래서 왼쪽 그림의 결정 경계는 매우 부드럽고 오른쪽으로 갈수록 결정 경계는 하나의 포인트에 더 민감해집니다. 작은 γ 값이 결정 경계를 천천히 바꿔게 하므로 모델의 복잡도를 낮춥니다. 반면에 큰 γ 값은 더 복잡한 모델을 만듭니다.

위에서 아래로는 C 매개변수를 0.1 에서 1000 으로 증가시켰습니다. 선형 모델에서처럼 작은 C 는 매우 제약이 큰 모델을 만들고 각 데이터 포인트의 영향력이 작습니다. 왼쪽 위의 결정 경계는 거의 선형에 가까우며 잘못 분류된 데이터 포인트가 경계에 거의 영향을 주지 않습니다. 왼쪽 아래에서 볼 수 있듯이 C 를 증가시키면 이 포인트들이 모델에 큰 영향을 주며 결정 경계를 휘어서 정확하게 분류하게 합니다.

커널 서포트 벡터 머신 - 유방암 데이터 분석

1. 아무 매개변수 조정 없이 SVC 적용

```
In [104]: X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=0)
```

```
In [106]: from sklearn.svm import SVC    #not linear SVC! 비선형 살펴보기
```

```
In [107]: svc=SVC().fit(X_train, y_train)
```

```
In [109]: svc.score(X_train,y_train)
```

```
Out[109]: 1.0
```

```
In [110]: svc.score(X_test,y_test)'
```

```
Out[110]: 0.6293706293706294
```

**SVC-> 차원을 확장시켜가며 복잡성이 높아짐 ~ 예측력 높음 / 하지만 overfit 현상
=> 잘 제어하면 그래디언트 부스팅 방식, 랜덤포레스트보다 낫다. (고차원에 훨씬 적합함)**

훈련 세트에는 완벽한 점수를 냈지만 테스트 세트에는 63% 정확도라서 이 모델은 상당히 과대적합되었습니다. SVM은 잘 작동하는 편이지만 매개변수 설정과 데이터 스케일에 매우 민감합니다. 특히 입력 특성의 범위가 비슷해야 합니다. 각 특성의 최솟값과 최댓값을 로그 스케일로 나타내보겠습니다

2. 각 매개변수의 스케일 확인 - 설명변수 분포도

In [23]: plt.boxplot(x_train, manage_xticks=False)

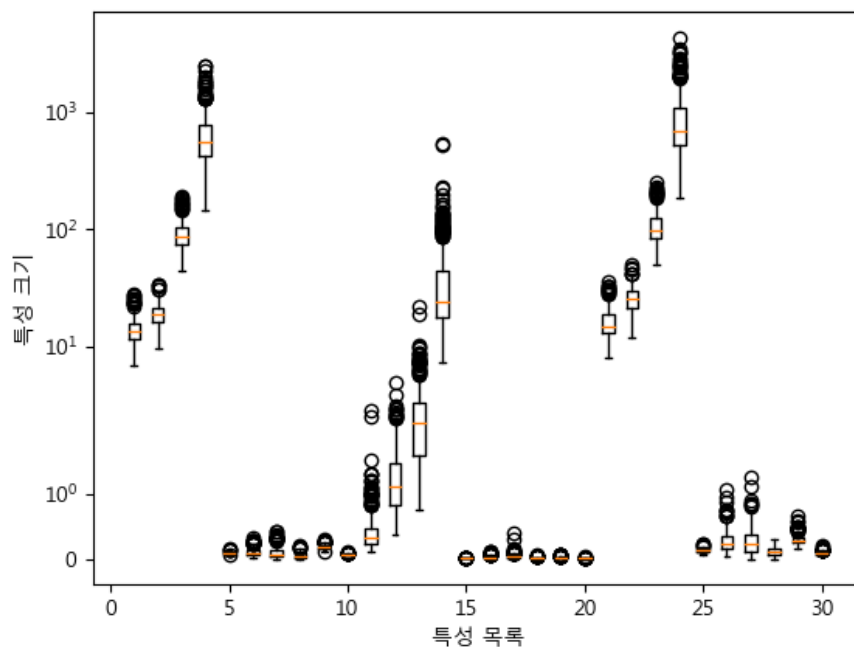
In [20]: plt.yscale("symlog")

In [21]: plt.xlabel("특성 목록")

Out[21]: Text(0.5,23.4122,'특성 목록')

In [22]: plt.ylabel("특성 크기")

Out[22]: Text(39.4094,0.5,'특성 크기')



13, 14, 15, 25 ~ 분산이 커보임

그래프를 보니 유방암 데이터셋의 특성은 자릿수 자체가 완전히 다릅니다. 이것이 일부 모델(선형 모델 등)에서도 어느 정도 문제가 될 수 있지만, 커널 SVM에서는 영향이 아주 큼니다. 이 문제를 해결하는 방법을 알아보겠습니다.

[[회귀 관련 : 표준화, 변수선택에 민감하다.

그런데,

SVC from 회귀분석 --> 항상 scale 먼저 확인해야 한다.

스케일 조정 필요 확인 후 표준화 필요 + 매개변수 잘 조절]]

cf. Random Forest -> 스케일 민감 X

3. SVM 을 위한 데이터 전처리

(이상치 제거 + scale 조절)

훈련 세트에서 특성별 최솟값 계산

In [34]: min_on_training=x_train.min(axis=0)

훈련 세트에서 특성별(최댓값-최솟값) 범위 계산

In [36]: range_on_training=(x_train-min_on_training).max(axis=0)

#훈련 데이터에 최솟값을 빼고 범위로 나누면, 각 특성의 최솟값은 0, 최댓값은 1

In [37]: x_train_scaled=(x_train-min_on_training)/range_on_training

In [41]: x_test_scaled=(x_test-min_on_training)/range_on_training

test 세트에도 같은 작업을 적용하지만 훈련 세트에서 계산한 최솟값과 범위를 사용함

In [38]: svc=SVC()

In [39]: svc.fit(x_train_scaled, y_train)

Out[39]:

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False)

In [40]: svc.score(x_train_scaled,y_train)

Out[40]: 0.9483568075117371

In [43]: svc.score(x_test_scaled, y_test)

Out[43]: 0.951048951048951

스케일만 조정했을 뿐인데 test 정확도가 높아졌다. - overfit 현상을 앞보다 해결한 편.
그러나 오히려 훈련 set 정확도가 더 떨어짐 . 이에 따른 처리가 필요해보임.

4. 매개변수의 조정 및 분석 결과(+scale 조정)

In [44]: svc=SVC(C=1000) # C 매개변수 조정

In [45]: svc.fit(x_train_scaled, y_train)

Out[45]:

SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False)

In [46]: svc.score(x_train_scaled, y_train)

Out[46]: 0.9882629107981221

In [47]: svc.score(x_test_scaled, y_test)

Out[47]: 0.972027972027972

3 번에 비해 train, test 성능 모두 오름.

#gamma 조절

In [120]: svc = SVC(gamma=0.1).fit(X_train_scaled, y_train)^M

In [121]: svc.score(X_train_scaled,y_train)

Out[121]: 0.9577464788732394

In [122]: svc.score(X_test_scaled,y_test)

Out[122]: 0.951048951048951

In [123]: svc = SVC(gamma=1).fit(X_train_scaled, y_train)^M

In [124]: svc.score(X_train_scaled,y_train)

Out[124]: 0.9835680751173709

In [125]: svc.score(X_test_scaled,y_test)

Out[125]: 0.972027972027972

#역시 3 번에 비해서 train, test 성능 모두 올랐다.

장단점과 매개변수

장점

- 데이터의 특성이 몇 개 안되더라도 복잡한 경계를 만들 수 있는 분석방법
- 저차원, 고차원의 데이터 모두 잘 작동

단점

- 100,000 개 이상의 데이터 셋에서는 분석이 매우 느림
- 데이터 전처리와 매개변수 설정 신경써야 함
(랜덤포레스트와 그래디언트 부스팅 모델은 거의 필요 X).
- 분석과 해석 및 예측이 어려움(모델의 복잡성)

/하지만 모든 특성이 비슷한 단위이고 (ex. 모든 값이 픽셀의 컬러 강도), 스케일이 비슷 -> SVM 시도 할만 함

중요 매개변수

- C : 모델의 복잡도를 규제하는 매개변수로, 각 포인트의 중요도를 제한.
클수록 데이터 포인트의 영향력을 최대화
- gamma : 매개변수가 하나의 훈련 샘플이 미치는 영향의 범위를 결정, 클수록 복잡한 경계
-> 설명변수 자체를 통제. like 라쏘, 릿지 in 회귀분석

C 가 클수록 비선형으로 바뀐 + 예측력 높아짐 (overfit 생각해야 함)

gamma 가 커질수록 각 계수의 크기가 커짐

cf.

C 조절

```
In [138]: train_acc, test_acc=[],[]
```

```
In [142]: k=np.arange(0.1,1.1,0.1)
```

```
In [143]: for i in k :
```

```
...:     svc=SVC(C=i).fit(X_train_scaled,y_train)
```

```
...:     train_acc.append(svc.score(X_train_scaled,y_train))
```

```
...:     test_acc.append(svc.score(X_test_scaled,y_test))
```

```
...:
```

```
In [148]: plt.plot(k,train_acc,label="훈련정확도")
```

```
Out[148]: [<matplotlib.lines.Line2D at 0xdcfea20>]
```

```
In [149]: plt.plot(k,test_acc,label="테스트정확도")
```

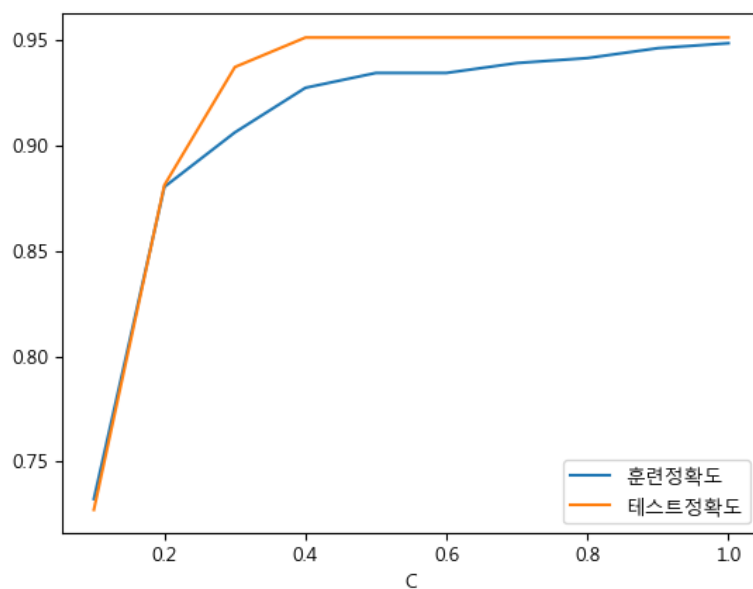
```
Out[149]: [<matplotlib.lines.Line2D at 0xdd100b8>]
```

```
In [150]: plt.xlabel("C")
```

```
Out[150]: Text(0.5,24.4122,'C')
```

```
In [151]: plt.legend()
```

```
Out[151]: <matplotlib.legend.Legend at 0xcdcd2f28>
```



gamma 조절

```
In [154]: train_acc, test_acc=[],[]
```

```
In [142]: k=np.arange(0.1,1.1,0.1)
```

```
In [155]: for i in k :
```

```
...:     svc=SVC(gamma=i).fit(X_train_scaled,y_train)
```

```
...:     train_acc.append(svc.score(X_train_scaled,y_train))
```

```
...:     test_acc.append(svc.score(X_test_scaled,y_test))
```

```
In [156]: plt.plot(k,train_acc,label="훈련정확도")
```

```
Out[156]: [<matplotlib.lines.Line2D at 0xe022748>]
```

```
In [157]: plt.plot(k,test_acc,label="테스트정확도")
```

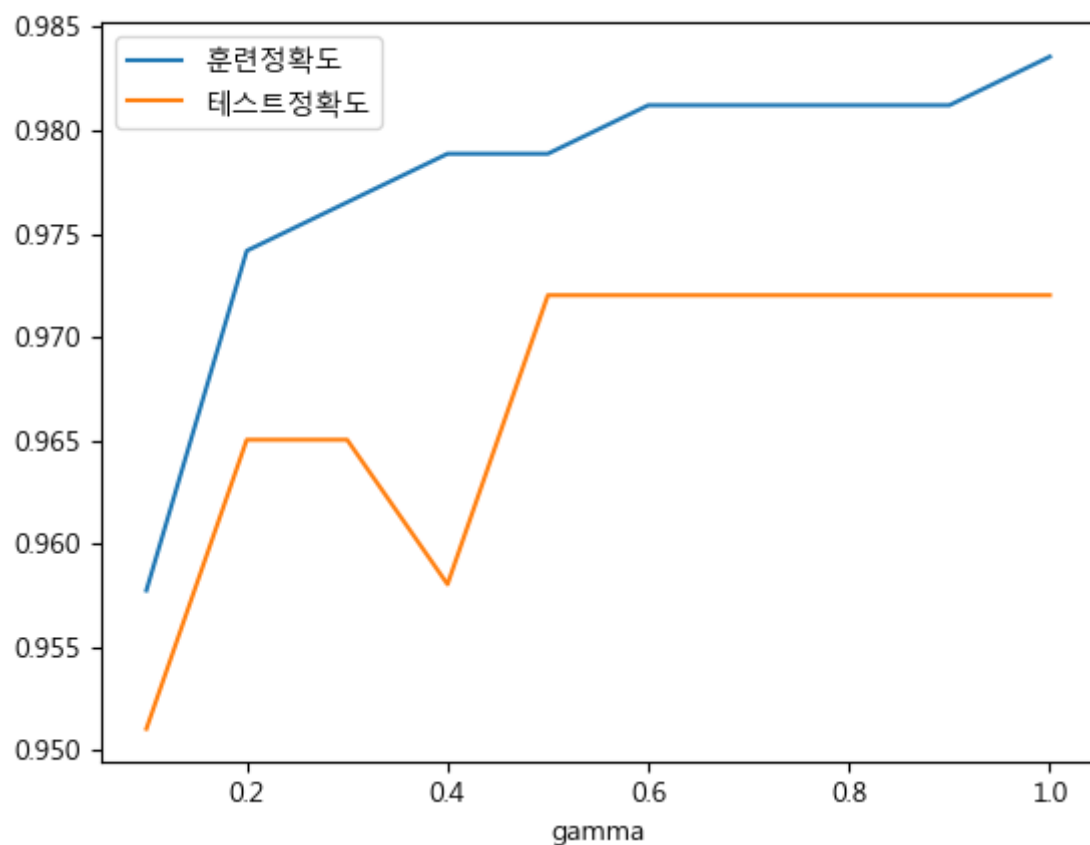
```
Out[157]: [<matplotlib.lines.Line2D at 0xe022e10>]
```

```
In [158]: plt.xlabel("gamma")
```

```
Out[158]: Text(0.5,24.4122,'gamma')
```

```
In [159]: plt.legend()
```

```
Out[159]: <matplotlib.legend.Legend at 0xe0333c8>
```



데이터 전처리와 스케일 조정

~회귀, 거리 기반 꼭 필요! [kNN, 군집분석, SVM, ...등] (분류 tree 제외)

분석 전@@ data handling 필요 !

train / test 나누기 전에 원 데이터 한꺼번에 있을 때 하기!!

주의 - y값은 원본 그대로! 절대 조정 X

1st

sklearn.StandardScaler

- 각 특성의 평균을 0, 분산을 1로 변경하여 모든 특성이 같은 크기를 가지게 하는 함수
- 최대값과 최소값을 제한하는 방법은 아니다.

cf 자유도 - np, pd 다르지만, sckitlearn에서 알아서 제공해줌 !

2nd

sklearn.MinMaxScaler

- 모든 특성이 정확하게 0과 1 사이에 위치하도록 데이터를 변경

$(x - \min(x)) / (\max(x) - \min(x))$

3rd

sklearn.RobustScaler

- StandardScaler와 비슷하게 같은 스케일을 갖게 하는 함수
- 평균과 분산 대신 중간값과 사분위 값을 사용
- 전체 데이터와 아주 동떨어진 데이터 포인트에 영향을 받지 않음

$(x - q_2) / (q_3 - q_1)$

데이터 변환 적용하기

cancer 데이터의 SVM분석 시 스케일 조정

```
In [160]: X_train,X_test,y_train,y_test=train_test_split(cancer.data, cancer.target, random_state=1)
```

```
In [161]: X_train.shape
```

```
Out[161]: (426, 30)
```

```
In [162]: X_test.shape
```

```
Out[162]: (143, 30)
```

주의 - y값은 원본 그대로! 절대 조정 X

```
In [164]: scaler=MinMaxScaler()
```

```
In[7]:
# 데이터 변환
X_train_scaled = scaler.transform(X_train)
# 스케일이 조정된 후 데이터셋의 속성을 출력합니다.
print("변환된 후 크기: {}".format(X_train_scaled.shape))
print("스케일 조정 전 특성별 최솟값: \n {}".format(X_train.min(axis=0)))
print("스케일 조정 전 특성별 최댓값: \n {}".format(X_train.max(axis=0)))
print("스케일 조정 후 특성별 최솟값: \n {}".format(X_train_scaled.min(axis=0)))
print("스케일 조정 후 특성별 최댓값: \n {}".format(X_train_scaled.max(axis=0)))
```

```
Out[7]:
변환된 후 크기: (426, 30)
스케일 조정 전 특성별 최솟값:
[ 6.98  9.71 43.79 143.50  0.05  0.02  0.  0.  0.11
  0.05  0.12  0.36  0.76  6.80  0.  0.  0.  0.
  0.01  0.  7.93 12.02 50.41 185.20 0.07 0.03  0.
  0.  0.16  0.06]
스케일 조정 전 특성별 최댓값:
[ 28.11  39.28 188.5 2501.0  0.16  0.29  0.43  0.2
  0.300  0.100  2.87  4.88 21.98 542.20  0.03  0.14
  0.400  0.050  0.06  0.03 36.04 49.54 251.20 4254.00
  0.220  0.940  1.17  0.29  0.58  0.15]
스케일 조정 후 특성별 최솟값:
[ 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.]
스케일 조정 후 특성별 최댓값:
[ 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.]
```

In[8]:

```
# 테스트 데이터 변환
X_test_scaled = scaler.transform(X_test)
# 스케일이 조정된 후 테스트 데이터의 속성을 출력합니다.
print("스케일 조정 후 특성별 최솟값:\n{}".format(X_test_scaled.min(axis=0)))
print("스케일 조정 후 특성별 최댓값:\n{}".format(X_test_scaled.max(axis=0)))
```

Out[8]:

```
스케일 조정 후 특성별 최솟값:
[ 0.034  0.023  0.031  0.011  0.141  0.044  0.    0.    0.154 -0.006
 -0.001  0.006  0.004  0.001  0.039  0.011  0.    0.   -0.032  0.007
  0.027  0.058  0.02  0.009  0.109  0.026  0.    0.   -0.    -0.002]
스케일 조정 후 특성별 최댓값:
[ 0.958  0.815  0.956  0.894  0.811  1.22  0.88  0.933  0.932  1.037
  0.427  0.498  0.441  0.284  0.487  0.739  0.767  0.629  1.337  0.391
  0.896  0.793  0.849  0.745  0.915  1.132  1.07  0.924  1.205  1.631]
```

놀랍게도 스케일을 조정한 테스트 세트의 최솟값과 최댓값이 0과 1이 아닙니다. 일부 특성은 0~1 범위를 벗어났습니다! **MinMaxScaler**는 (그리고 다른 모든 스케일 모델도) 항상 훈련 세트와 테스트 세트에 같은 변환을 적용해야 합니다. transform 메서드는 테스트 세트의 최솟값과 범위를 사용하지 않고, 항상 훈련 세트의 최솟값을 빼고 훈련 세트의 범위로 나눕니다.⁶

#test scaled는 최소,최대가 0과 1이 아닌 이유는 train 기준 test에 적용했기 때문!
--.fit(train) / .transform(test)

make_blobs 데이터 셋 스케일 조정 예제

In [3]: from sklearn.datasets import make_blobs

In [4]: X, _ = make_blobs(n_samples=50,centers=5,random_state=4,cluster_std=2)

In [9]: from sklearn.model_selection import train_test_split

In [17]: X_train, X_test, y_train, y_test=train_test_split(X,_random_state=5, test_size=.1)

In [18]: fig, axes = plt.subplots(1,3,figsize=(13,4))

In [19]: plt.rc("font",family="Malgun Gothic")

#1 원본데이터 산점도 그리기

In [21]: axes[0].scatter(X_train[:,0],X_train[:,1],c=mglearn.cm2(0),label="훈련 세트", s=60)

Out[21]: <matplotlib.collections.PathCollection at 0x94bd7f0>

In [30]: axes[0].scatter(X_test[:,0],X_test[:,1],c=mglearn.cm2(1),marker='^',label="테스트 세트", s=60)

Out[30]: <matplotlib.collections.PathCollection at 0x97f9f60>

In [31]: axes[0].set_title("원본 데이터")

Out[31]: Text(0.5,1,'원본 데이터')

In [32]: axes[0].legend(loc='upper left')

Out[32]: <matplotlib.legend.Legend at 0x9cc5be0>

#2 스케일 기준 같게 조정하여 스케일 표준화

#MinMaxScaler 사용하기

```
In [34]: from sklearn.preprocessing import MinMaxScaler
```

```
In [35]: scaler=MinMaxScaler()
```

```
In [36]: scaler.fit(X_train)
```

```
Out[36]: MinMaxScaler(copy=True, feature_range=(0, 1))
```

```
In [37]: X_train_scaled=scaler.transform(X_train)
```

```
In [38]: X_test_scaled=scaler.transform(X_test)
```

```
In [40]: axes[1].scatter(X_train_scaled[:,0],X_train_scaled[:,1],c=mglern.cm2(0),label="훈련 세트",  
s=60)
```

```
Out[40]: <matplotlib.collections.PathCollection at 0x9818128>
```

```
In [41]: axes[1].scatter(X_test_scaled[:,0],X_test_scaled[:,1],c=mglern.cm2(1),marker='^',label="테스  
트 세트", s=60)
```

```
Out[41]: <matplotlib.collections.PathCollection at 0x97f90b8>
```

```
In [42]: axes[1].set_title("스케일 조정된 데이터")
```

```
Out[42]: Text(0.5,1,'스케일 조정된 데이터')
```

#3 각각 다른 스케일 함수로 스케일 표준화

```
In [44]: test_scaler=MinMaxScaler()
```

```
In [45]: test_scaler.fit(X_test)
```

```
Out[45]: MinMaxScaler(copy=True, feature_range=(0, 1))
```

```
In [48]: X_test_scaled_badly=test_scaler.transform(X_test)
```

```
In [49]: axes[2].set_title("잘못 조정된 데이터")
```

```
Out[49]: Text(0.5,1,'잘못 조정된 데이터')
```

```
In [50]: axes[2].scatter(X_train_scaled[:,0],X_train_scaled[:,1],c=mglern.cm2(0),label="훈련 세트",
```

s=60)

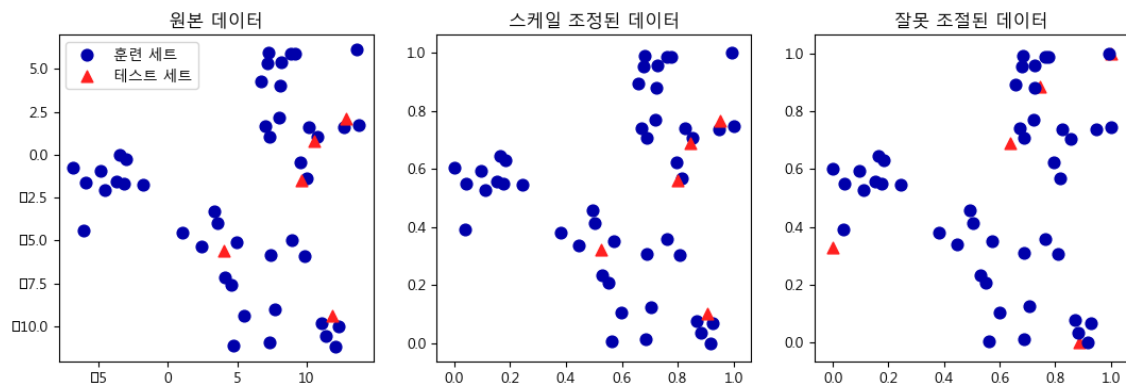
Out[50]: <matplotlib.collections.PathCollection at 0x986d908>

In

[51]:

```
axes[2].scatter(X_test_scaled_badly[:,0],X_test_scaled_badly[:,1],c=mglearn.cm2(1),marker='^',label="테스트 세트", s=60)
```

Out[51]: <matplotlib.collections.PathCollection at 0x986deb8>



그래프 1, 2 분포 일치 / 그래프 3 : 원본과 분포가 달라짐 -> 잘못 조정된 dataset

결국 test/set 분리(train_test_split) 이전! 동일 데이터에서 scale 조정 후 70:30

나누기(train_test_split)

cancer 데이터 셋에 SVM 분석 시 데이터 전처리 과정 예제

```
In [64]: from sklearn.svm import SVC
```

```
In [65]: from sklearn.datasets import load_breast_cancer
```

```
In [66]: cancer=load_breast_cancer()
```

```
In [67]: X_train,X_test,y_train,y_test=train_test_split(cancer.data, cancer.target, random_state=0)
```

```
In [68]: svm=SVC(C=1000)
```

```
In [69]: svm.fit(X_train,y_train)
```

```
In [70]: svm.score(X_test,y_test)
```

```
Out[70]: 0.6293706293706294
```

@전처리 안 했을 때 test score 점수 in SVM

#MinMaxScaler 사용

```
In [71]: scaler=MinMaxScaler().fit(X_train)
```

```
In [73]: X_train_scaled=scaler.transform(X_train)
```

```
In [74]: X_test_scaled=scaler.transform(X_test)
```

#조정된 데이터로 SVM학습

```
In [75]: svm.fit(X_train_scaled,y_train)
```

```
In [76]: svm.score(X_test_scaled,y_test)
```

```
Out[76]: 0.972027972027972
```

#StandardScaler 사용

```
In [77]: from sklearn.preprocessing import StandardScaler
```

```
In [78]: scaler=StandardScaler()
```

```
In [80]: scaler.fit(X_train)
```

```
Out[80]: StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
In [81]: X_train_scaled=scaler.transform(X_train)
```

```
In [82]: X_test_scaled=scaler.transform(X_test)
```

#조정된 데이터로 SVM학습

```
In [83]: svm.fit(X_train_scaled,y_train)
```

```
In [84]: svm.score(X_test_scaled,y_test)
```

```
Out[84]: 0.958041958041958
```


0716

주성분분석(Principal Component Analysis : PCA) for 차원 축소

특성

- 상관관계가 있는 변수들 사이의 복잡한 구조를 좀 더 간편하고 이해하기 쉽게 설명하기 위한 기법
- 설명변수를 많이 포함할수록 설명력은 높아질 수 있으나 동시에 변수들 간의 다중공선성 발생
- 변수들의 선형결합을 통하여 변수들이 가지고 있는 전체 정보를 최대한 설명하는 새로운 인공변수를 유도하여 해석하는 다변량 분석방법
- 제거되는 설명변수는 없음, 가중치에 따른 결합으로 새로운 인공변수 생성
- 단점 : 결합된 변수의 해석이 어렵다. 오로지 예측에만 사용..

목적

- : 서로 '독립적인' 주성분 유도 및 해석
- : 차원축소에 따른 다변량 자료의 요약
- : 주성분을 통한 분석 모델 유도

주성분 추출 방법

- 변수들이 전체 분산 대부분을 소수의 주성분을 통하여 설명하는 것이 주 목적
- 첫 번째 주성분은 변수들의 전체 분산 중 가장 큰 부분을 설명할 수 있도록 유도
- 두 번째 주성분은 첫 번째 주성분과 독립적이면서 첫 번째 주성분에 의해 설명되지 않은 잔여분산을 최대한 설명할 수 있도록 유도

주 목적 - 다른 알고리즘 적용 "전" 차원축소로 선택할 수 있는 방법 중 하나

cf. 이미지 목적 - 인공변수를 증가시켜 설명력을 높일 목적

cf. PCA : 직접적으로 예측/분류하려는 목적이 아니기 때문에 비지도학습에 포함
(+군집분석)

#여기도 scale 조정을 해야 하는군!!

PCA 를 활용한 고차원 데이터의 시각화

```
In [2]: fig, axes=plt.subplots(15,2,figsize=(10,20))
```

```
In [3]: from sklearn.datasets import load_breast_cancer
```

```
In [4]: cancer=load_breast_cancer()
```

```
In [5]: malignant=cancer.data[cancer.target==0]
```

```
In [7]: benign=cancer.data[cancer.target==1]
```

```
In [8]: ax=axes.ravel()
```

```
In [9]: for i in range(30) :
```

```
...:     _ , bins=np.histogram(cancer.data[:,i],bins=50)
...:     ax[i].hist(malignant[:,i],bins=bins,color=mglearn.cm3(0),alpha=.5)
...:     ax[i].hist(benign[:,i],bins=bins,color=mglearn.cm3(2),alpha=.5)
...:     ax[i].set_title(cancer.feature_names[i])
...:     ax[i].set_yticks(())
...:
```

```
In [15]: ax[0].set_xlabel("특성 크기")
```

```
Out[15]: Text(0.5,652.83,'특성 크기')
```

```
In [16]: ax[0].set_ylabel("빈도")
```

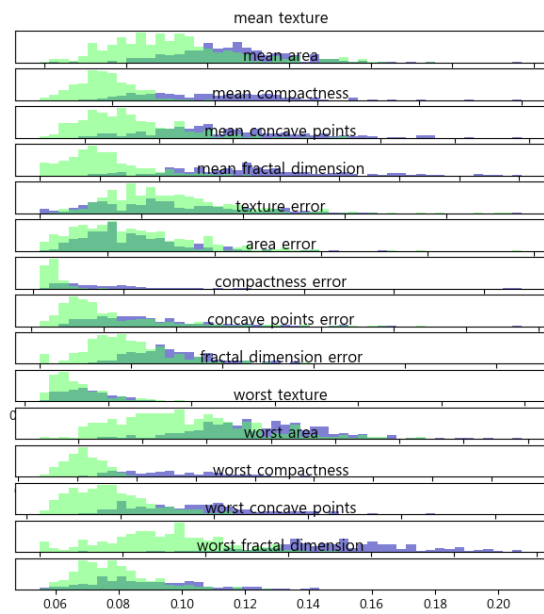
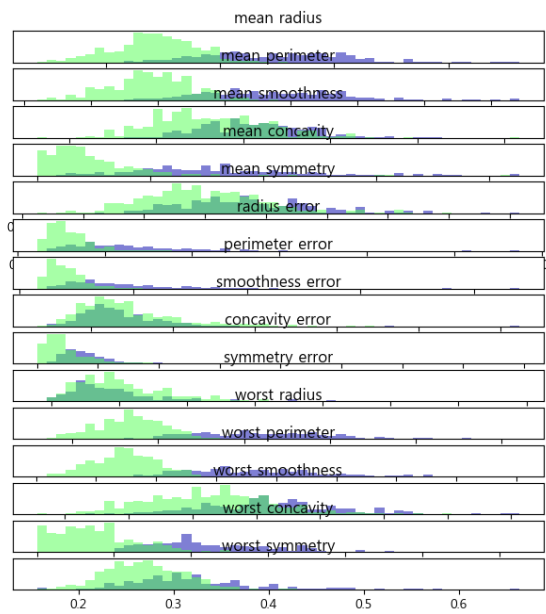
```
Out[16]: Text(119.444,0.5,'빈도')
```

```
In [17]: ax[0].legend(['악성','양성'],loc='best')
```

```
Out[17]: <matplotlib.legend.Legend at 0x1421d5f8>
```

```
In [18]: fig.tight_layout()
```

#cf. 원래는 axes[0,0] 등 2 차원 형식으로 지정해서 2 중 for 문을 돌려야 하는데 이를 단순화시킴



#StandardScaler 로 스케일 조정

```
In [30]: scaler=StandardScaler()
```

```
In [31]: scaler.fit(cancer.data)
```

```
Out[31]: StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
In [32]: X_scaled=scaler.transform(cancer.data)
```

#PCA 를 적용하여 두 개의 주성분 추출

#기본값일 경우 PCA 는 데이터를 회전만 시키고 모든 주성분 유지

#데이터의 차원을 줄이려면 PCA 를 만들 때 얼마나 많은 성분을 유지할 지 지정

```
In [33]: from sklearn.decomposition import PCA
```

```
In [34]: pca=PCA(n_components=2)  #
```

```
In [35]: pca.fit(X_scaled)
```

```
Out[35]:
```

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,  
      svd_solver='auto', tol=0.0, whiten=False)
```

```
In [36]: X_pca=pca.transform(X_scaled)
```

```
In [38]: X_scaled.shape  #원본 데이터 형태
```

```
Out[38]: (569, 30)
```

```
In [39]: X_pca.shape  #CPA 를 통해 축소된 데이터 형태
```

```
Out[39]: (569, 2)
```

```
# 두 개의 주성분으로 설명되는 각 데이터 셋의 산점도 출력
```

```
#클래스를 색으로 구분하여 처음 두 개의 주성분을 그래프로 나타냄
```

```
In [40]: plt.figure(figsize=(8,8))
```

```
Out[40]: <Figure size 800x800 with 0 Axes>
```

```
In [41]: mglearn.discrete_scatter(X_pca[:,0],X_pca[:,1],cancer.target)
```

```
# 각 col 끼리의 산점도 -- cancer.target 으로 class 별 color 지정
```

```
Out[41]:
```

```
[<matplotlib.lines.Line2D at 0xa755f28>,
```

```
<matplotlib.lines.Line2D at 0xa7610f0>]
```

```
In [42]: plt.legend(["악성","양성"],loc='best')
```

```
Out[42]: <matplotlib.legend.Legend at 0xa798518>
```

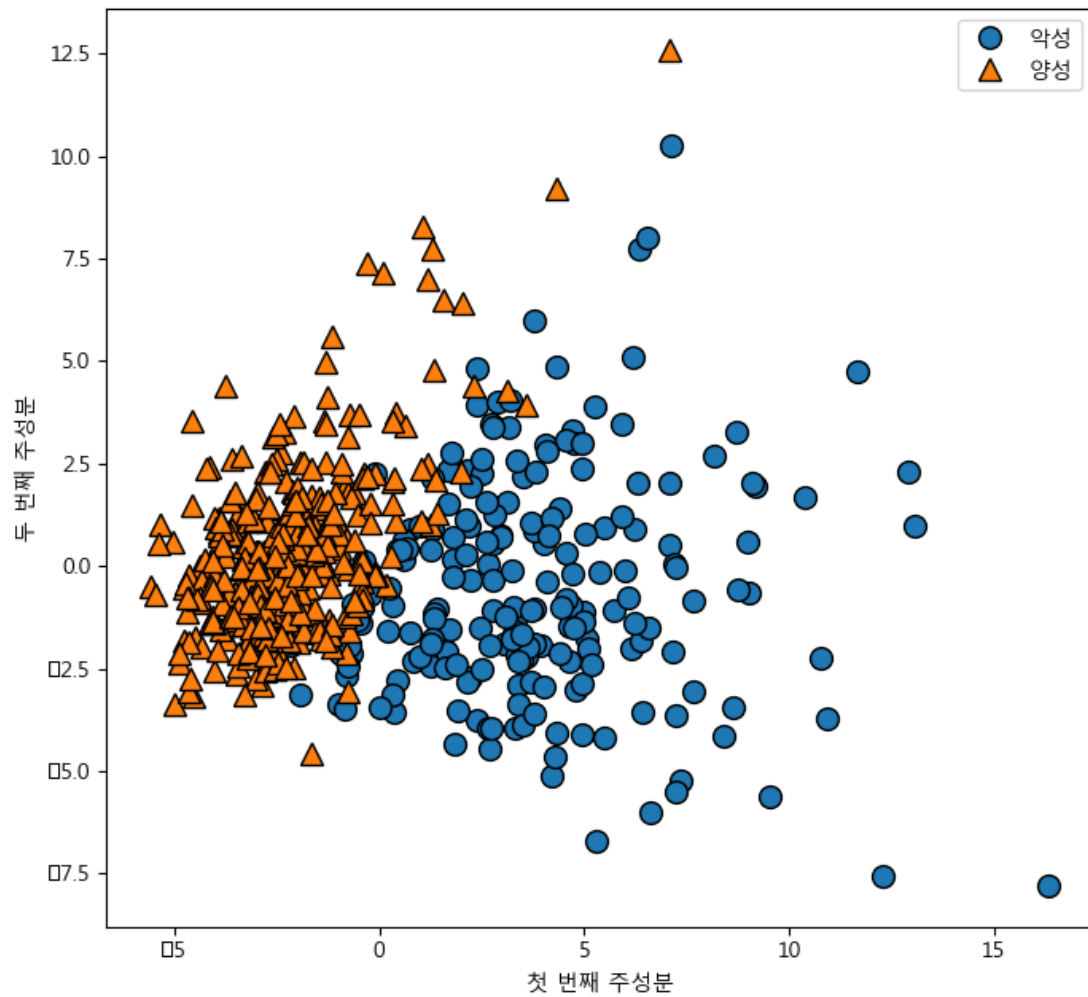
```
In [43]: plt.gca().set_aspect('equal')
```

```
In [44]: plt.xlabel('첫 번째 주성분')
```

```
Out[44]: Text(0.5,78.9835,'첫 번째 주성분')
```

```
In [45]: plt.ylabel('두 번째 주성분')
```

```
Out[45]: Text(57.2222,0.5,'두 번째 주성분')
```



c1, c2 에 따라 독립적으로 생성됨 => 선형관계 없게끔.

분리할 수 있는 선형식 찾기

#주성분의 형태 확인

#PCA 가 학습될 때 components_ 속성에 주성분을 구성하는 각 계수들이 저장

```
In [46]: print("PCA 주성분형태: {}".format(pca.components_.shape))
```

PCA 주성분형태: (2, 30)

```
In [47]: print("PCA 주성분:\n{}".format(pca.components_))
```

PCA 주성분:

```
[[ 0.21890244  0.10372458  0.22753729  0.22099499  0.14258969  0.23928535
   0.25840048  0.26085376  0.13816696  0.06436335  0.20597878  0.01742803
   0.21132592  0.20286964  0.01453145  0.17039345  0.15358979  0.1834174
   0.04249842  0.10256832  0.22799663  0.10446933  0.23663968  0.22487053
   0.12795256  0.21009588  0.22876753  0.25088597  0.12290456  0.13178394]
 [-0.23385713 -0.05970609 -0.21518136 -0.23107671  0.18611302  0.15189161
   0.06016536 -0.0347675   0.19034877  0.36657547 -0.10555215  0.08997968
  -0.08945723 -0.15229263  0.20443045  0.2327159   0.19720728  0.13032156
   0.183848    0.28009203 -0.21986638 -0.0454673  -0.19987843 -0.21935186
   0.17230435  0.14359317  0.09796411 -0.00825724  0.14188335  0.27533947]]
```

#c1, c2 로 유도된 식 설명 for 모델

ex. $c1 = 0.219x_1 + 0.104x_2 + 0.228x_3 + \dots$

#주성분을 이용한 선형 SVC 모델 적용

```
In [48]: from sklearn.svm import LinearSVC
```

```
In [51]: linear_svm=LinearSVC().fit(X_pca,cancer.target)
```

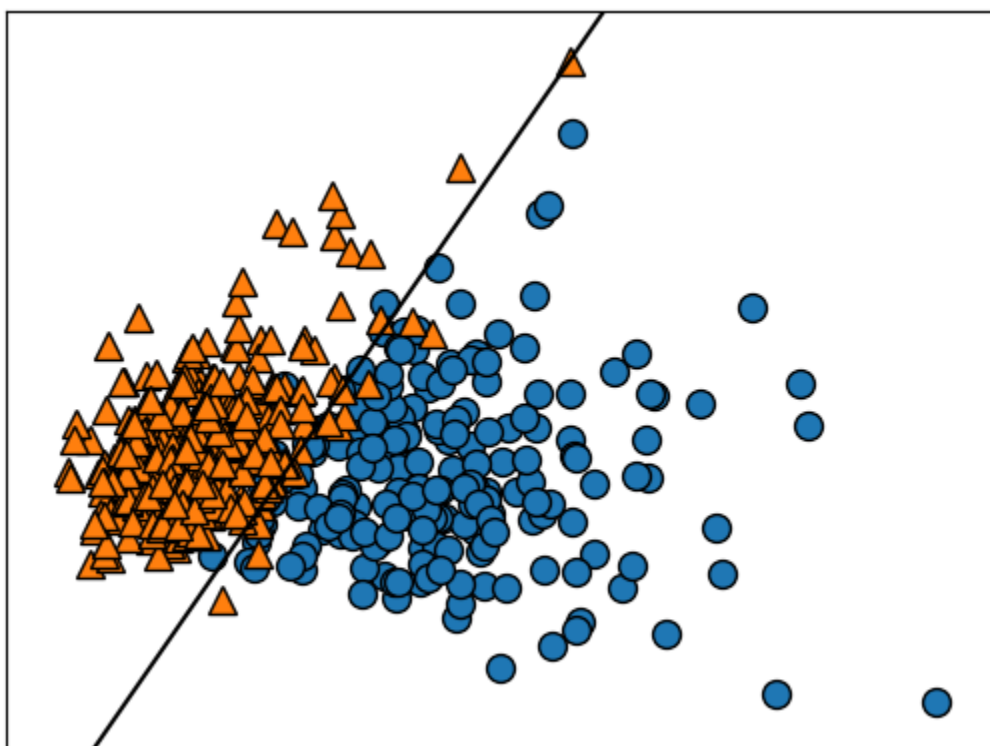
#pca 로 transform 된 c1,c2 를 인공변수로 가지고 있는 X_pca

```
In [52]: mglearn.plots.plot_2d_separator(linear_svm, X_pca)
```

```
In [57]: print("훈련 세트 정확도: {:.2f}".format(linear_svm.score(X_pca,cancer.target)))
```

훈련 세트 정확도: 0.95

#2 차원인데도 꽤 높게 유도됨 -> 2 차원이라도 PCA 를 통해 설명력이 높은 인공변수를 만들어서



SVM 말고도 다른 모델 적용 가능! KNN 이나 등등
random forest 는 고차원 일 때!

PCA 를 활용한 고유 얼굴 특성 추출 - 얼굴인식

cf. 이미지인식은 차원의 축소 X -> 의미있는 설명치에 가중치 부여로 재해석됨

PCA 를 이용한 특성 추출

- 이미지를 다루는 애플리케이션은 특성 추출이 가능
- 이미지는 적색,녹색,청색(RGB)의 강도가 기록된 픽셀로 구성(raw file 로 변환)

LFW 데이터 셋

- 2000 년 초반 이후의 정치인, 가수, 배우, 운동선수들의 얼굴 데이터
- 처리속도를 높이기 위해 흑백 이미지 사용, 스케일을 줄인 상태로 분석 시도
- 총 62 명의 얼굴을 찍은 이미지가 총 3,023 개 존재
- 각 이미지의 크기는 87*65 픽셀 (규격화)

시험

PCA ~ 학습시켜 핵심 설명변수에 (일부) 더 높은 가중치를 부여하는 방법으로 해석
in 이미지 --> 설명변수가 많을수록 유리하다.

(의미있는 픽셀에 가중치를 부여하며 예측력을 높임. 모델 복잡도를 신경쓰지 않음)

#LFW 데이터 셋 확인

```
In [58]: from sklearn.datasets import fetch_lfw_people
```

```
In [59]: people=fetch_lfw_people(min_faces_per_person=20,resize=0.7)
```

#people data set 도 딕셔너리 형태로 target ~ 등등 분리되어 있음

```
In [60]: image_shape=people.images[0].shape
```

#첫 번째 RGB 강도의 차원 출력

```
In [63]: fig,axes=plt.subplots(2,5,figsize=(15,8), subplot_kw={'xticks':(),'yticks':()})
```

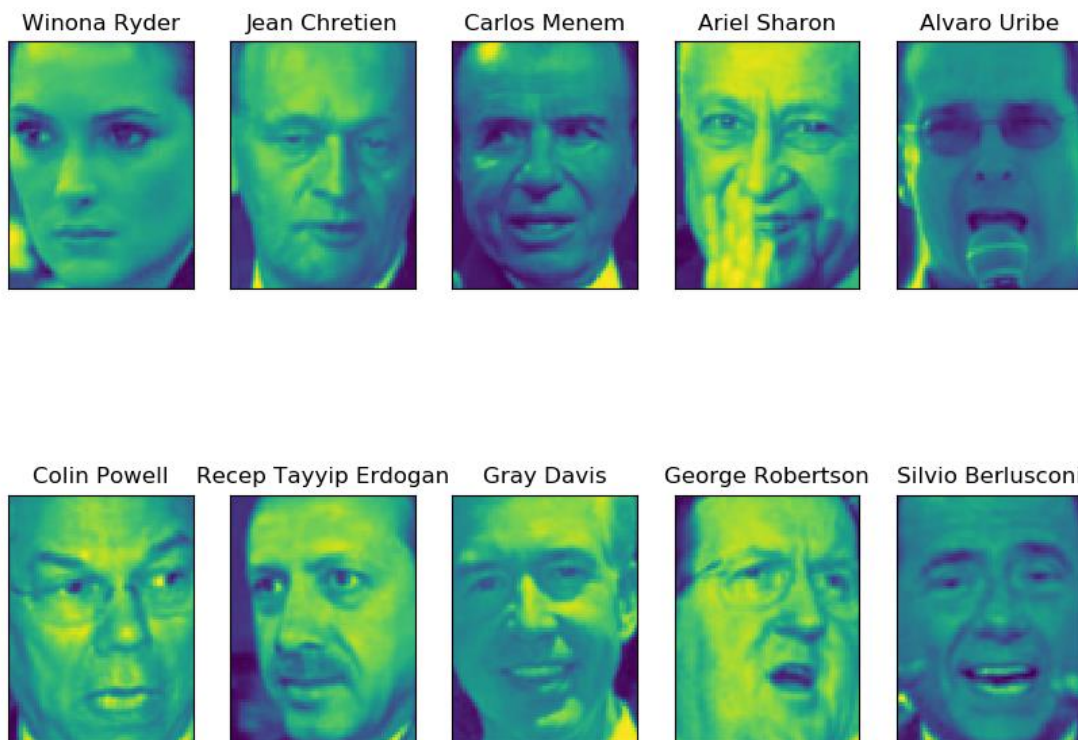
```
In [67]: for target, image, ax in zip(people.target, people.images, axes.ravel()):
```

```
....:     ax.imshow(image)
```

```
....:     ax.set_title(people.target_names[target])
```

이중 for 문 불편 --> 각 축(axes.ravel()) 활용하여 zip 형태의 튜플로 표현

imshow : 로우파일형태로 픽셀의 RGB 강도를 가지고 있다면 이미지 출력을 가능하게 하는 함수



이미지 데이터에 대한 shape 확인

```
In [70]: people.images.shape
```

```
Out[70]: (3023, 87, 65)    # 총 3023 데이터 (87*65 픽셀형태)
```

```
In [71]: len(peoples.target_names)
```

```
Out[71]: 62    #클래스 개수
```

#데이터 : 1*1 픽셀에서의 RGB 강도

이미지 데이터의 각 클래스별 빈도 확인

In [72]: counts=np.bincount(people.target)

#np.bincount : 빈도수 확인

In [74]: for i, (count,name) in enumerate(zip(counts,people.target_names)) :

...: print("{0:25}{1:3}".format(name,count),end=' ')

...: if (i+1)%3==0 :

...: print()

#enumerate ~ zip 에 번호 부여해서 i 숫자 생성하기 위해서 만듦

#1st print

-- {0:25}{1:3} -- name 의 크기 / count 의 크기 ! 25byte, 3byte 규격된 size 로 정렬 목적.

-- end=' ' -- 연달아 출력하기 위해서

#2nd print

if(i+1)%3==0 : ---> i+1 : i 는 0 부터 시작

print()

-- > 세 개씩 정렬하기 위해서! 다음칸으로 넘기기

Alejandro Toledo	39	Alvaro Uribe	35	Amelie Mauresmo	21
Andre Agassi	36	Angelina Jolie	20	Ariel Sharon	77
Arnold Schwarzenegger	42	Atal Bihari Vajpayee	24	Bill Clinton	29
Carlos Menem	21	Colin Powell	236	David Beckham	31
Donald Rumsfeld	121	George Robertson	22	George W Bush	530
Gerhard Schroeder	109	Gloria Macapagal Arroyo	44	Gray Davis	26
Guillermo Coria	30	Hamid Karzai	22	Hans Blix	39
Hugo Chavez	71	Igor Ivanov	20	Jack Straw	28
Jacques Chirac	52	Jean Chretien	55	Jennifer Aniston	21
Jennifer Capriati	42	Jennifer Lopez	21	Jeremy Greenstock	24
Jiang Zemin	20	John Ashcroft	53	John Negroponte	31
Jose Maria Aznar	23	Juan Carlos Ferrero	28	Junichiro Koizumi	60
Kofi Annan	32	Laura Bush	41	Lindsay Davenport	22
Lleyton Hewitt	41	Luiz Inacio Lula da Silva	48	Mahmoud Abbas	29
Megawati Sukarnoputri	33	Michael Bloomberg	20	Naomi Watts	22
Nestor Kirchner	37	Paul Bremer	20	Pete Sampras	22
Recep Tayyip Erdogan	30	Ricardo Lagos	27	Roh Moo-hyun	32
Rudolph Giuliani	26	Saddam Hussein	23	Serena Williams	52
Silvio Berlusconi	33	Tiger Woods	23	Tom Daschle	25
Tom Ridge	33	Tony Blair	144	Vicente Fox	32
Vladimir Putin	49	Winona Ryder	24		

결과 : 조지부시와 콜린파월에 편중된 데이터

#데이터의 편증을 없애기 위해 사람마다 50 개의 이미지 선택

```
In [96]: mask=np.zeros(people.target.shape,dtype=np.bool)
```

```
In [97]: for target in np.unique(people.target) :  
...:     mask[np.where(people.target==target)[0][:50]]=1  
...:
```

```
In [98]: X_people=people.data[mask]
```

```
In [99]: y_people=people.target[mask]
```

```
In [100]: X_people=X_people/255
```

#0~255 사이의 흑백 이미지의 픽셀 값을 0~1 스케일로 조정한다.

#MinMaxScaler 를 적용하는 것과 거의 같음

k-NN 을 이용한 이미지 클래스 분류

```
In [101]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [102]: from sklearn.model_selection import train_test_split
```

```
In [103]:
```

```
X_train,X_test,y_train,y_test=train_test_split(X_people,y_people,stratify=y_people,random_state=0)
```

```
In [104]: knn=KNeighborsClassifier(n_neighbors=1) #이웃 개수 1 개
```

```
In [106]: knn.fit(X_train,y_train)
```

```
Out[106]:
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                    metric_params=None, n_jobs=1, n_neighbors=1, p=2,  
                    weights='uniform')
```

```
In [107]: knn.score(X_test,y_test)
```

```
Out[107]: 0.23255813953488372
```

#1-최근접 이웃의 테스트 세트 점수 : 0.23

#PCA 를 활용한 얼굴 인식

- 얼굴의 유사도를 측정하기 위한 방법으로 각 픽셀과의 공간의 거리를 계산하는 방식은 신뢰도가 떨어짐
- 주성분으로 변환하여 거리를 계산하면 정확도가 향상될 수 있음
- PCA 모델 적용 시 화이트닝 옵션은 주성분의 스케일이 StandardScaler 를 적용한 것과 같은 효과

- PCA 객체를 훈련데이터로 학습시켜 처음 100 개의 주성분 추출

```
In [14]: pca=PCA(n_components=100, whiten=True, random_state=0).fit(X_train)
```

```
In [15]: X_train_pca=pca.transform(X_train)
```

```
In [16]: X_test_pca=pca.transform(X_test)
```

```
In [17]: X_train_pca.shape
```

```
Out[17]: (1547, 100)
```

#pca 한 데이터 ~ k-nn 을 활용한 분류 분석 시도

```
In [18]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [19]: knn=KNeighborsClassifier(n_neighbors=1)
```

```
In [20]: knn.fit(X_train_pca,y_train)
```

```
Out[20]:
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                    metric_params=None, n_jobs=1, n_neighbors=1, p=2,  
                    weights='uniform')
```

```
In [21]: knn.score(X_test_pca,y_test)
```

```
Out[21]: 0.312015503875969
```

#주성분의 시각화

```
In [22]: pca.components_.shape
```

```
Out[22]: (100, 5655)
```

#총 5655 개의 설명변수(원래)를 PCA 를 통해 가중치를 주고 100 개의 새로운 변수 생성

```
In [23]: fig, axes=plt.subplots(3,5,figsize=(15,12),subplot_kw={'xticks':(),'yticks':()})
```

```
In [27]: for i, (component, ax) in enumerate(zip(pca.components_,axes.ravel())) :
```

```
...:     ax.imshow(component.reshape(image_shape),cmap='viridis')
```

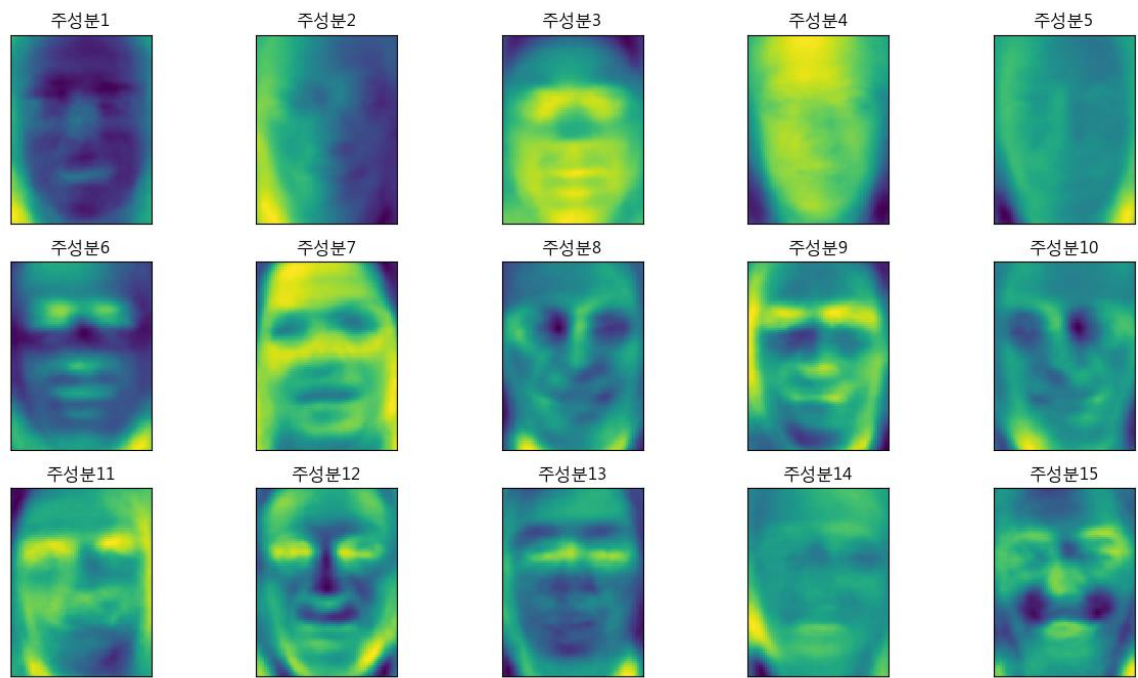
```
...:     ax.set_title("주성분{}".format((i+1)))
```

```
...:
```

#image_shape : 첫 번째(image_shape=people.images[0].shape)

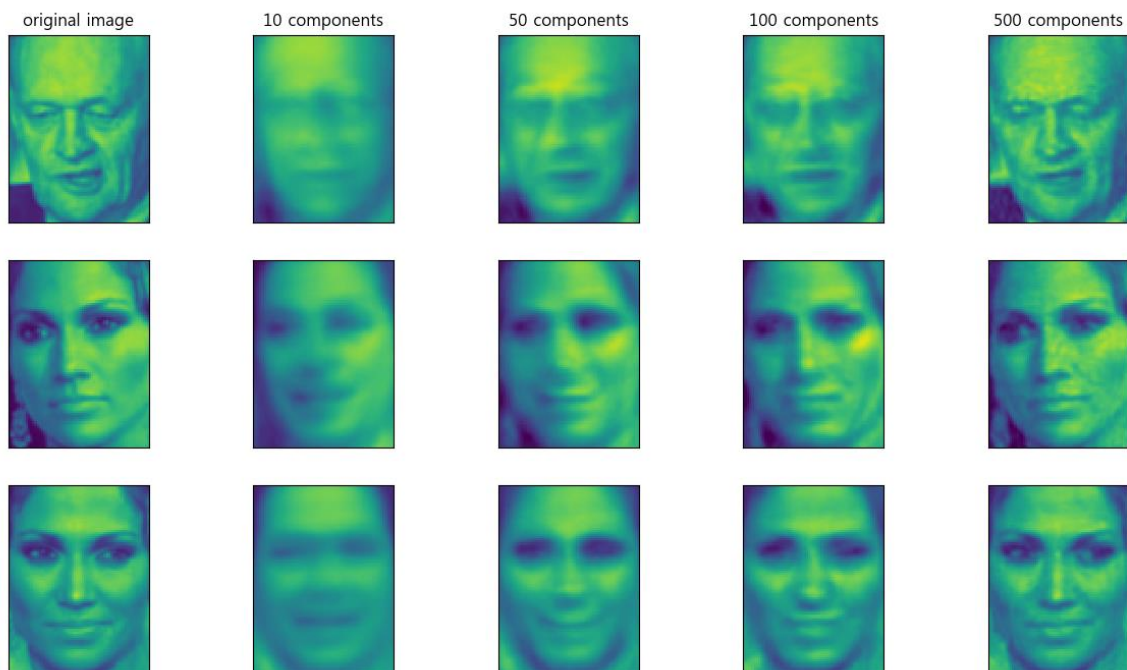
#추출된 주성분으로 데이터 해석하는 방법

- 테스트 포인트를 주성분의 가중치 합으로 나타내는 데 필요한 수치를 찾는 것으로 해석
- 각 x_0 와 x_1 들은 데이터 포인트에 대한 주성분 계수를 의미



얼굴 데이터 셋에 적용하여 이미지를 재구성한 과정

In [38]: mglearn.plots.plot_pca_faces(X_train,X_test,image_shape)



#주성분개수가 많아질 수록 설명력이 높아보임. 많은 주성분 개수 예측력 재확인 필요

(하둡 ~)리눅스~파이썬~알

<https://www.ubuntu.com/>

우분투 데스크톱 iso 파일 다운로드

디렉토리

cd : change directory

cd 하위폴더이름 : 하위폴더로 이동

cd .. : 상위폴더로 이동

ls : 현재 디렉토리 내용 확인

리눅스

VMware

설치 위치 키보드 드라이버 그냥 두고

업데이트 ~ 세부조항 체크 해제

cmd 창에서 ipconfig 시 두 개 네트워크 더 생김

설치 후 리눅스 설치 중

New Virtual Machine Wizard

25

Specify Disk Capacity
How large do you want this disk to be?

The virtual machine's hard disk is stored as one or more files on the host computer's physical disk. These file(s) start small and become larger as you add applications, files, and data to your virtual machine.

Maximum disk size (GB):

Recommended size for Ubuntu: 20 GB

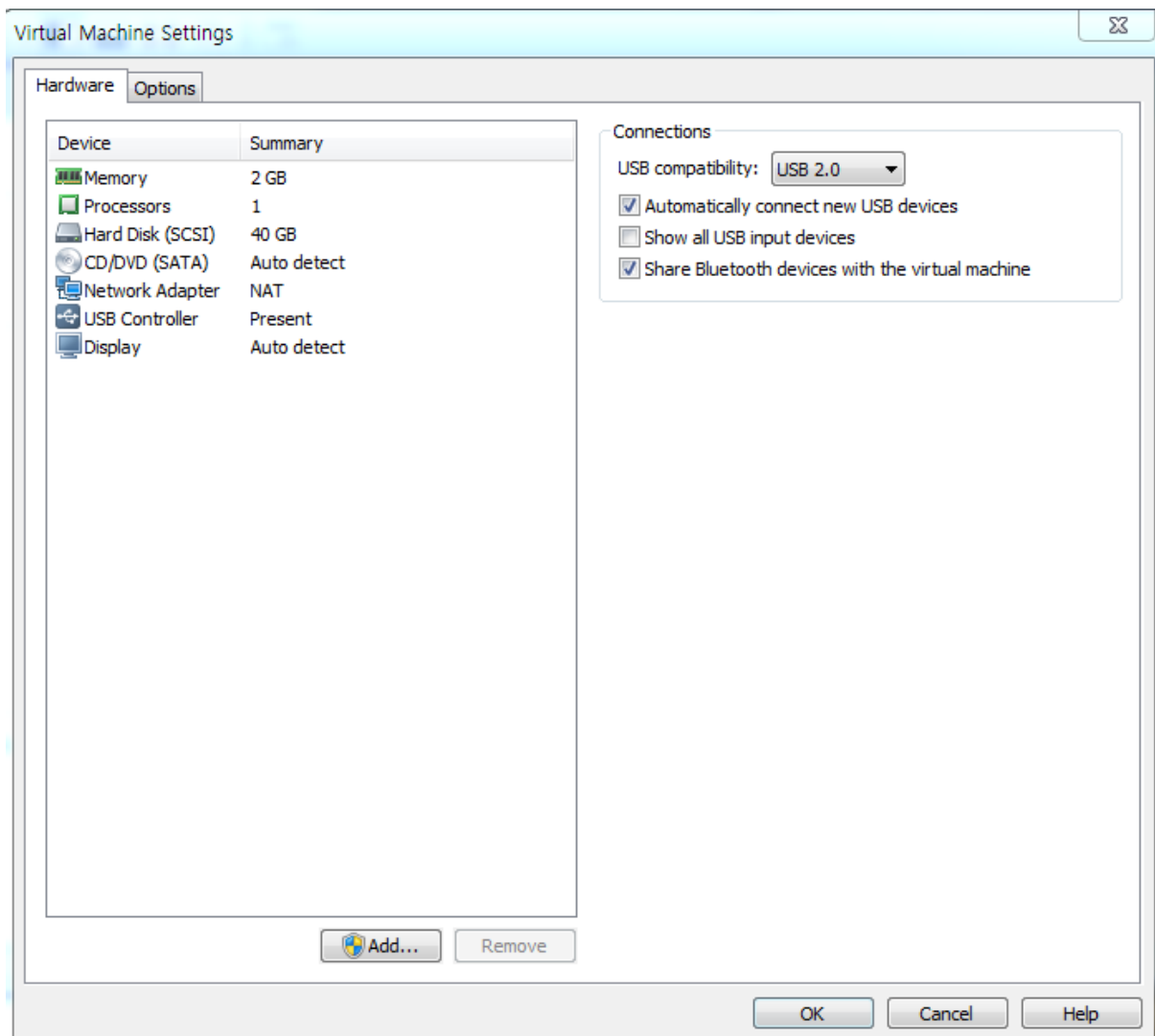
☒ Store virtual disk as a single file

☐ Split virtual disk into multiple files

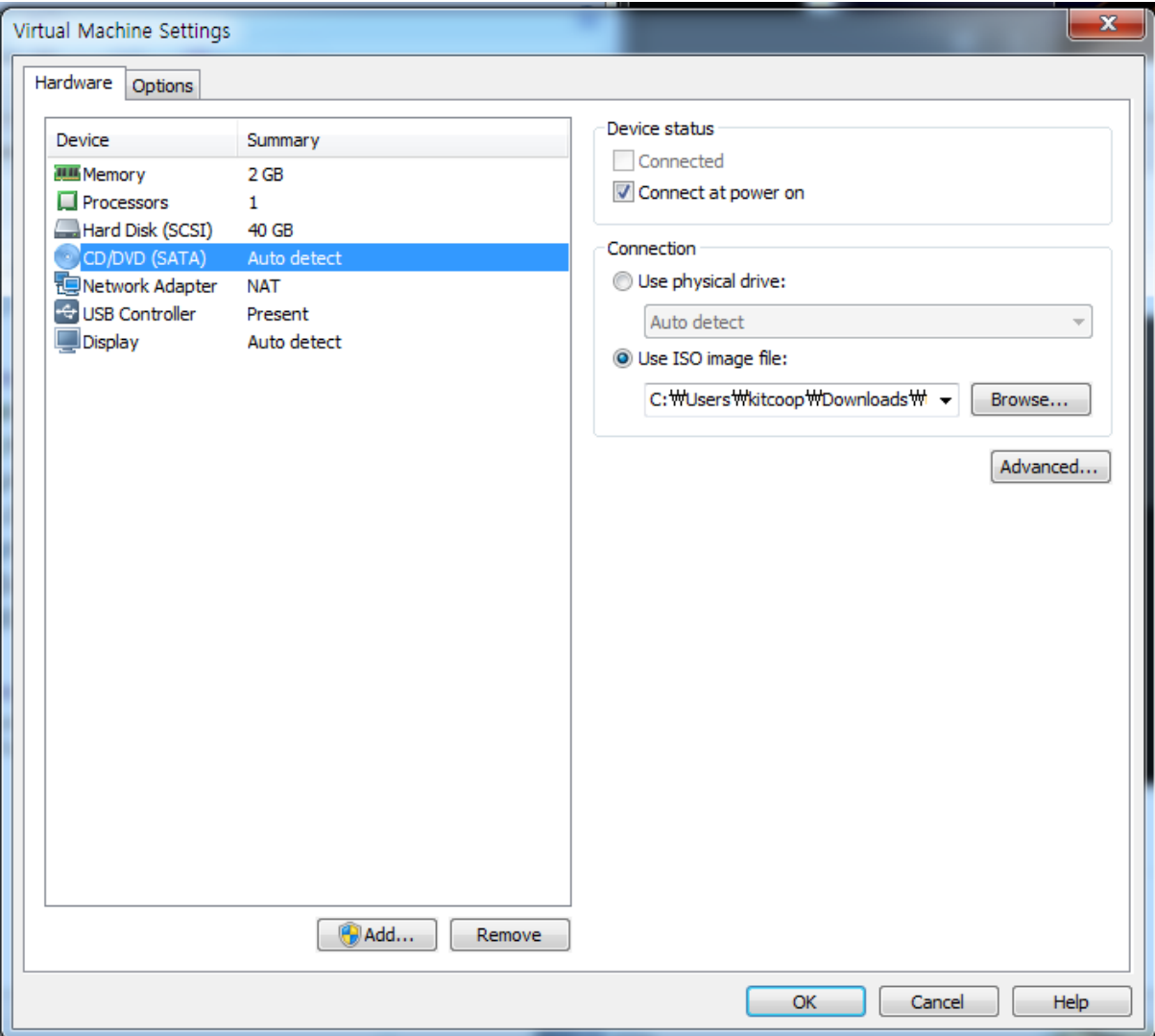
Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.

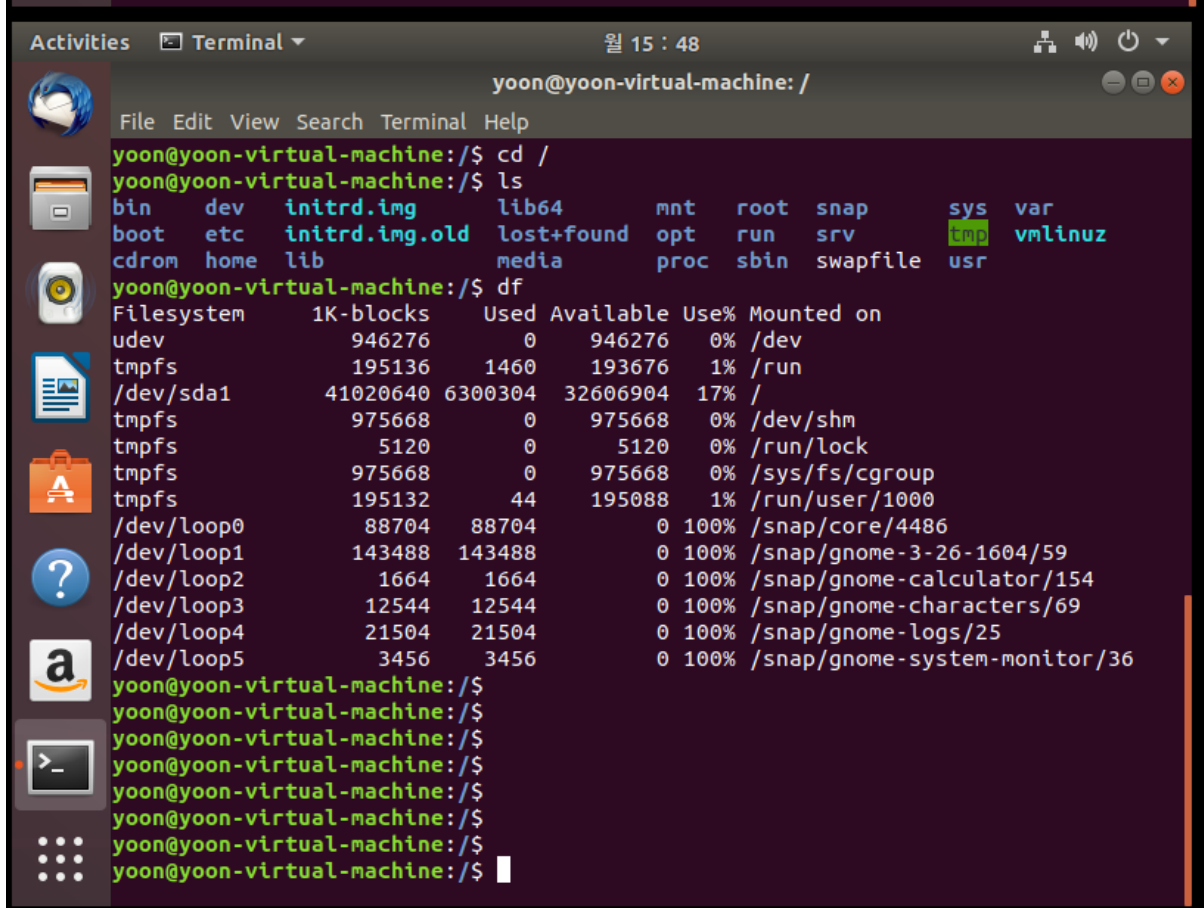
Help < Back Next > Cancel

저장공간 할당을 하나로 관리하는 게 더 편함 @0@

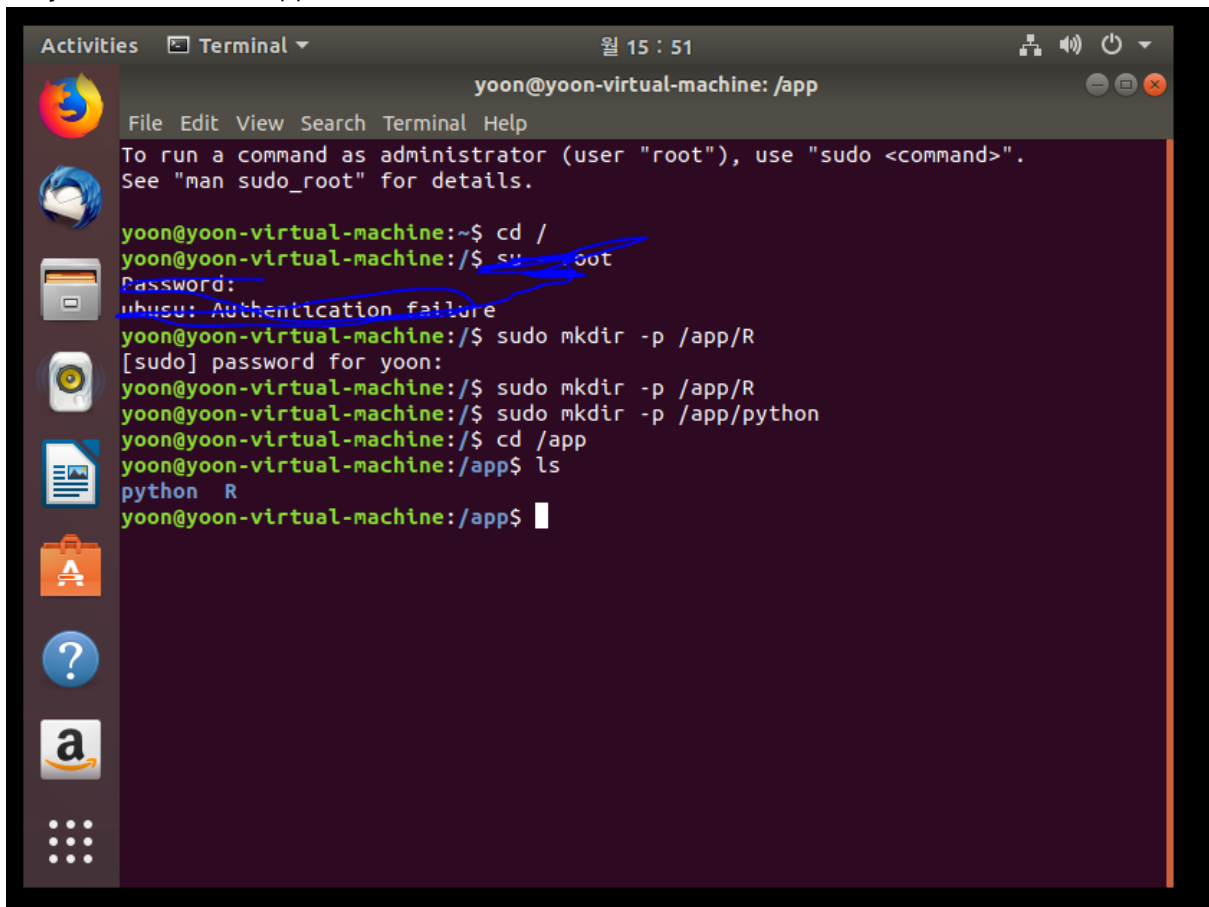


-프린터, 사운드카드 제거 램 2G 상승





R, Python 설치 위한 app 만들기 in 우분투



The image shows a terminal window in Ubuntu. The title bar indicates the user is 'yoon' on a 'yoon-virtual-machine' at the '/app' directory, with the time '월 15 : 51'. The terminal content shows the user navigating to the root directory, attempting to switch to root with 'su' (which fails with 'Authentication failure'), and then using 'sudo' to create directories. The commands and their outputs are as follows:

```
yoon@yoon-virtual-machine:~$ cd /
yoon@yoon-virtual-machine:/$ su - root
Password:
ubusu: Authentication failure
yoon@yoon-virtual-machine:/$ sudo mkdir -p /app/R
[sudo] password for yoon:
yoon@yoon-virtual-machine:/$ sudo mkdir -p /app/R
yoon@yoon-virtual-machine:/$ sudo mkdir -p /app/python
yoon@yoon-virtual-machine:/$ cd /app
yoon@yoon-virtual-machine:/app$ ls
python  R
yoon@yoon-virtual-machine:/app$
```

cd /

sudo mkdir -p /app/R -- 관리자 권한으로 디렉토리 만들기

sudo mkdir -p /app/Python

0717

vmware

디스플레이 설정

오른쪽 위 설정->디바이스 -> 1280*960 등 4:3 조절

vmware- > 윈도우 ctrl+c ctrl+v 할 수 있게끔 설치

VM->VMwareTools -> open with files 클릭

tar.gz 파일 ctrl+c-> Home 에 ctrl+v Extract here

Open Terminal

cd VMwareTools-10.01 (VM 만 치고 Tab 눌러도 알아서 써줌)

cd vmware + tab

ifconfig **#이거 치면 나오는 거 따라 치면 설치!!**

sudo apt install net-tools **#ifconfig**

sudo ./ vmware-install.real.pl

처음 y

계속 엔터

리눅스 IP 확인

yoona@yoona-virtual-machine:~\$ ifconfig

```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.204.128  netmask 255.255.255.0  broadcast 192.168.204.255
    inet6 fe80::5d5:f71a:ff4a:1a02  prefixlen 64  scopeid 0x20<link>
    ether 00:0c:29:19:3e:93  txqueuelen 1000  (Ethernet)
    RX packets 195  bytes 193639 (193.6 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 189  bytes 22966 (22.9 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
    device interrupt 19  base 0x2000
```

```
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
```

loop txqueuelen 1000 (Local Loopback)

RX packets 142 bytes 9950 (9.9 KB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 142 bytes 9950 (9.9 KB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

R 설치

```
sudo apt-get update  
sudo apt-get install r-base
```

R 입력

Rstudio 설치

<https://www.rstudio.com/products/rstudio/download-server/>

64bit

Size: 60.6 MB MD5: 3c546fa9067f48ed1a342f810fca8be6

Version: 1.1.453 Released: 2018-05-16

```
$ sudo apt-get install gdebi-core  
$ wget https://download2.rstudio.org/rstudio-server-1.1.453-amd64.deb  
$ sudo gdebi rstudio-server-1.1.453-amd64.deb
```

ifconfig 로 ip 그때그때 확인

yoony@yoony-virtual-machine:~\$ ifconfig

```
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500  
    inet 192.168.204.128  netmask 255.255.255.0  broadcast 192.168.204.255  
    inet6 fe80::5d5:f71a:ff4a:1a02  prefixlen 64  scopeid 0x20<link>  
    ether 00:0c:29:19:3e:93  txqueuelen 1000  (Ethernet)  
    RX packets 112133  bytes 169079855 (169.0 MB)  
    RX errors 1  dropped 1  overruns 0  frame 0  
    TX packets 56106  bytes 3076364 (3.0 MB)  
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0  
    device interrupt 19  base 0x2000
```

```
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536  
    inet 127.0.0.1  netmask 255.0.0.0  
    inet6 ::1  prefixlen 128  scopeid 0x10<host>  
    loop  txqueuelen 1000  (Local Loopback)  
    RX packets 208  bytes 15526 (15.5 KB)  
    RX errors 0  dropped 0  overruns 0  frame 0  
    TX packets 208  bytes 15526 (15.5 KB)
```

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

리눅스 아이피를 윈도우 인터넷창에서 검색 원격접속 (8787 은 기본포트) + 우분투 유저네임,비번
<http://192.168.204.128:8787/>

~ 가벼운 작업은 할 수 있지만 큰 작업에 권장하지 않음(윈도우에서 하는 것보단 빠르지만
리눅스로 하는 것보다는 느림)

웬만하면 콘솔로 연습하는 법 권장 (속도 차이)

python 설치

리눅스는 대개 설치되어있음

yoon@yoon-virtual-machine:~\$ **python**

Command 'python' not found, but can be installed with:

```
sudo apt install python3          #최근 파이썬 -> 깔려있음
sudo apt install python          #2.x 버전 파이썬
sudo apt install python-minimal
```

You also have python3 installed, you can run 'python3' instead.

yoon@yoon-virtual-machine:~\$ **python3**

Python 3.6.5 (default, Apr 1 2018, 05:46:30)

[GCC 7.3.0] on linux

Type "help", "copyright", "credits" or "license" for more information.

빠져나오기 ctrl+D / exit()입력

아나콘다 설치

wget https://repo.anaconda.com/archive/Anaconda3-5.2.0-Linux-x86_64.sh

bash Anaconda3-5.2.0-Linux-x86_64.sh

(중간에 license 관련 yes 작성) + default directory 그냥 넘어가기

아마

home/yoona/anaconda3 기본값 그냥 엔터

vscode 안해도 됨

anaconda 패스 잡아주기 필요

터미널창

cd

ls -a

(.으로 시작하는 숨김파일도 모두 표현해줌)

.profile : 환경설정 파일 -- 항상 유지됨

cf. export PATH 로 패스를 잡으면 창에서만 정해짐. 매번 다시 해야 함

vi: 리눅스에서 텍스트 파일 편집기 (불편함)

vi .profile 선택 후 아래로 쪽 내려간 다음

i 누른 다음에(입력모드)

```
# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi

export PATH=/home/voon/anaconda3/bin:$PATH
```

엔터 두 칸만 띄구 밑줄 입력 cf. dd 연속 누름 : 지우기

esc 누른 후(입력모드 종료)

:wq 입력

. ./profile 로 실행

ipython 실행

ipython --pylab 실행

jupyter notebook 실행

mglearn 설치는 !conda install mglearn 으로 설치

cf. pip in windows

6. 원격접속 환경 구성

winscp

xme--유료어플

cf. 우분투 : 외부에서 접속하도록 하는 패키지 깔아야 함

-->

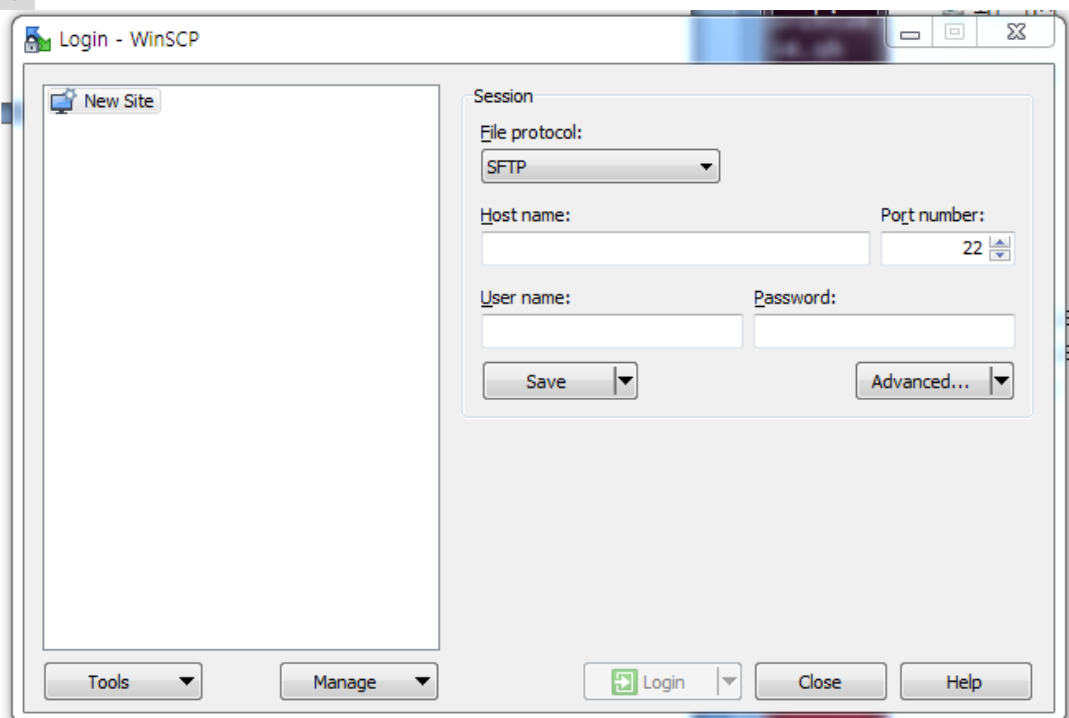
sudo apt-get install telnetd

#원격 접속을 가능하게 함

sudo apt-get install ssh

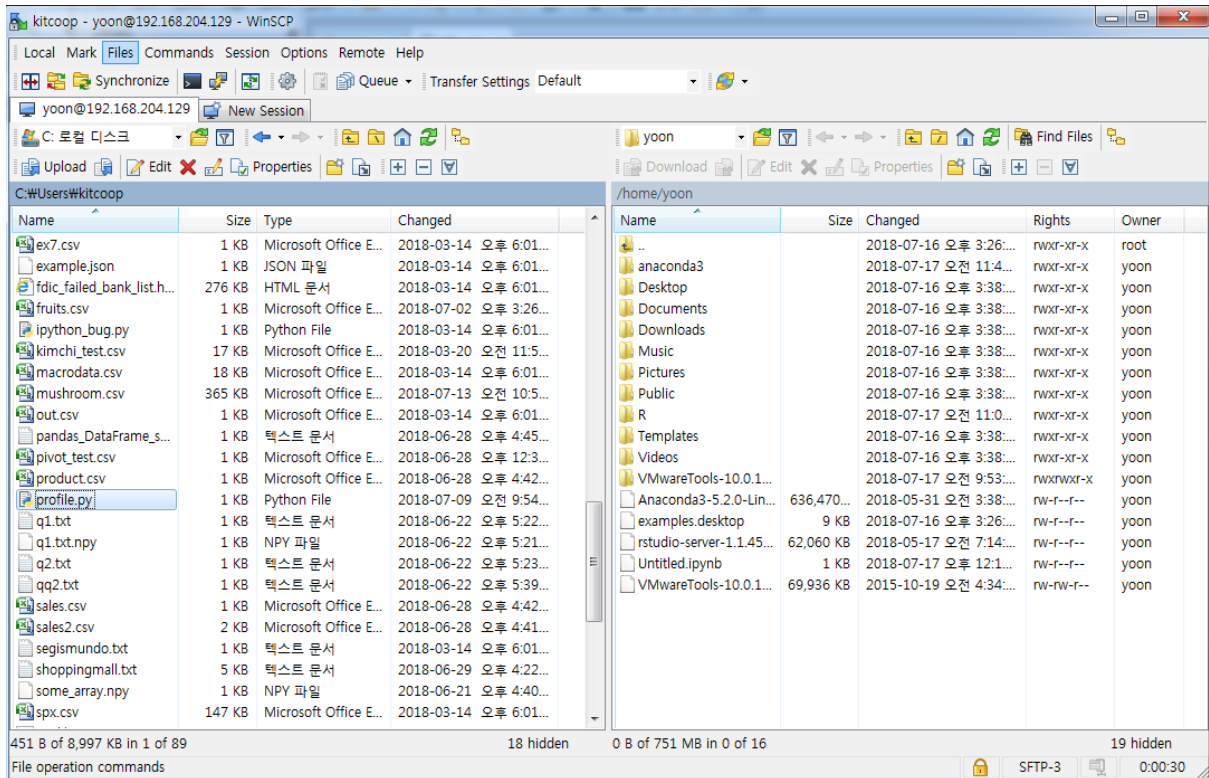
/레드햇기반은 필요 없음.

winscp 접속



접속하고자 하는 ip 확인 (ifconfig in ubuntu) + ubuntu 이름 비번

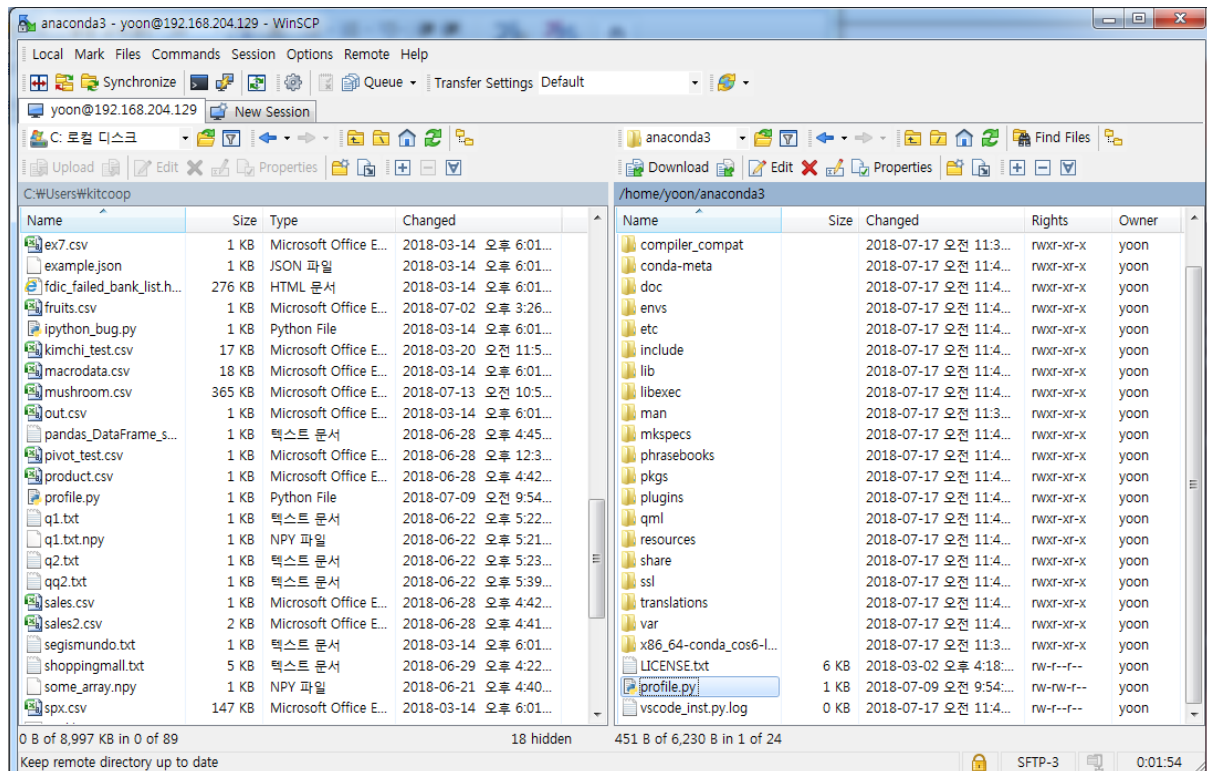
.



왼쪽 : 윈도우 상의 정보 / 오른쪽 : 서버 상의 정보

윈도우에서 profile.py 를 서버로 옮기기 위해서 사용
(서버에 있는 걸 내려받을 때도 사용)

오른쪽 폴더에 yoona/anaconda3 이 default
이동 후 파일을 드래그해서 옮기기



`cd /home/yoon/anaconda3` 들어간 후에 `ipython..!`

cf. 추가 설치한 패키지들 진행 똑같이 하면 됨

그런데 등록된 패키지가 다 `conda install` 로 실행되지는 않음. ex. `mglearn`

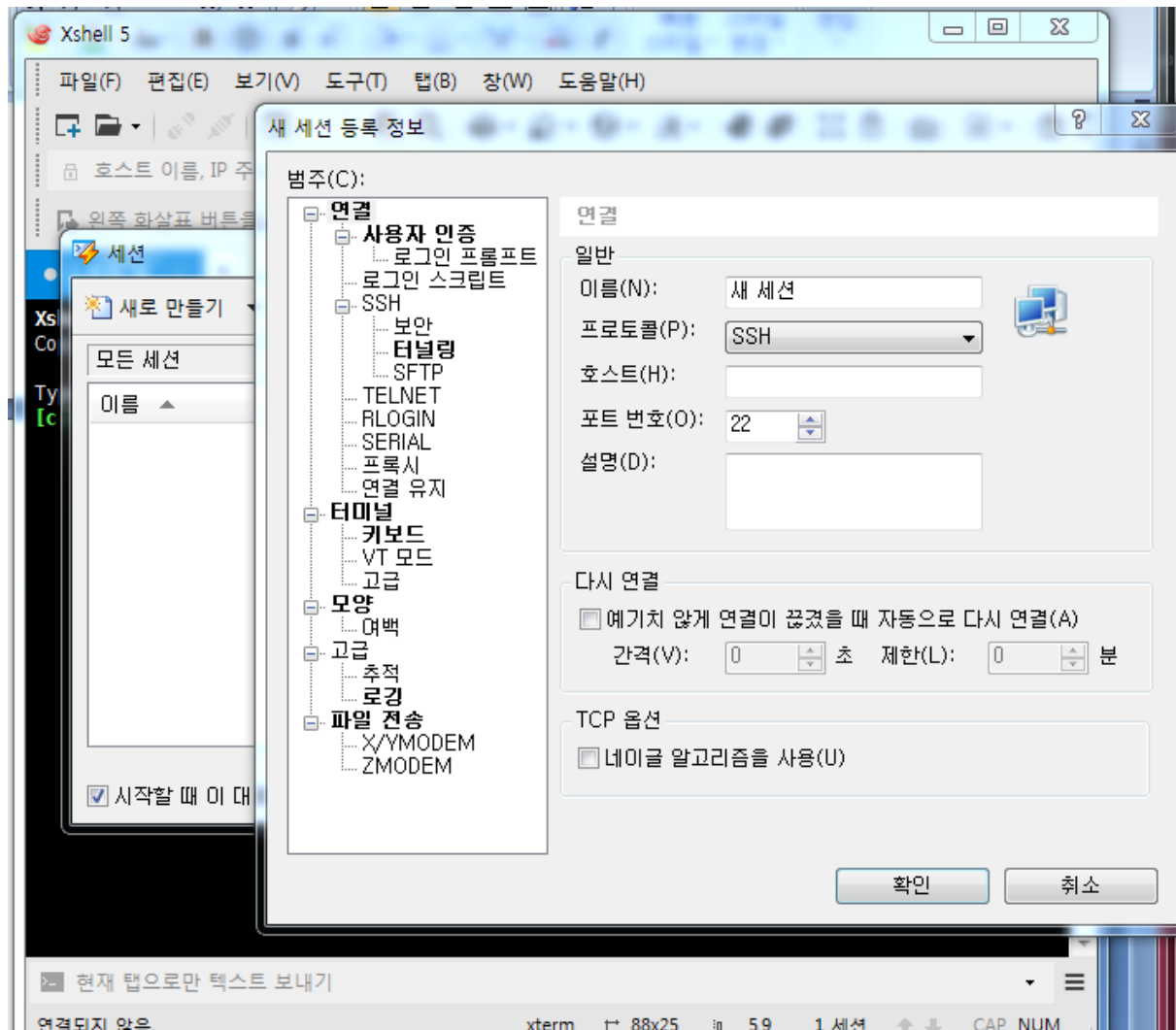
그럼 이런 명령어로 설치 불가, 외부에서 설치파일을 받아와서 진행해야 함

xmanager

cf. putty 파일

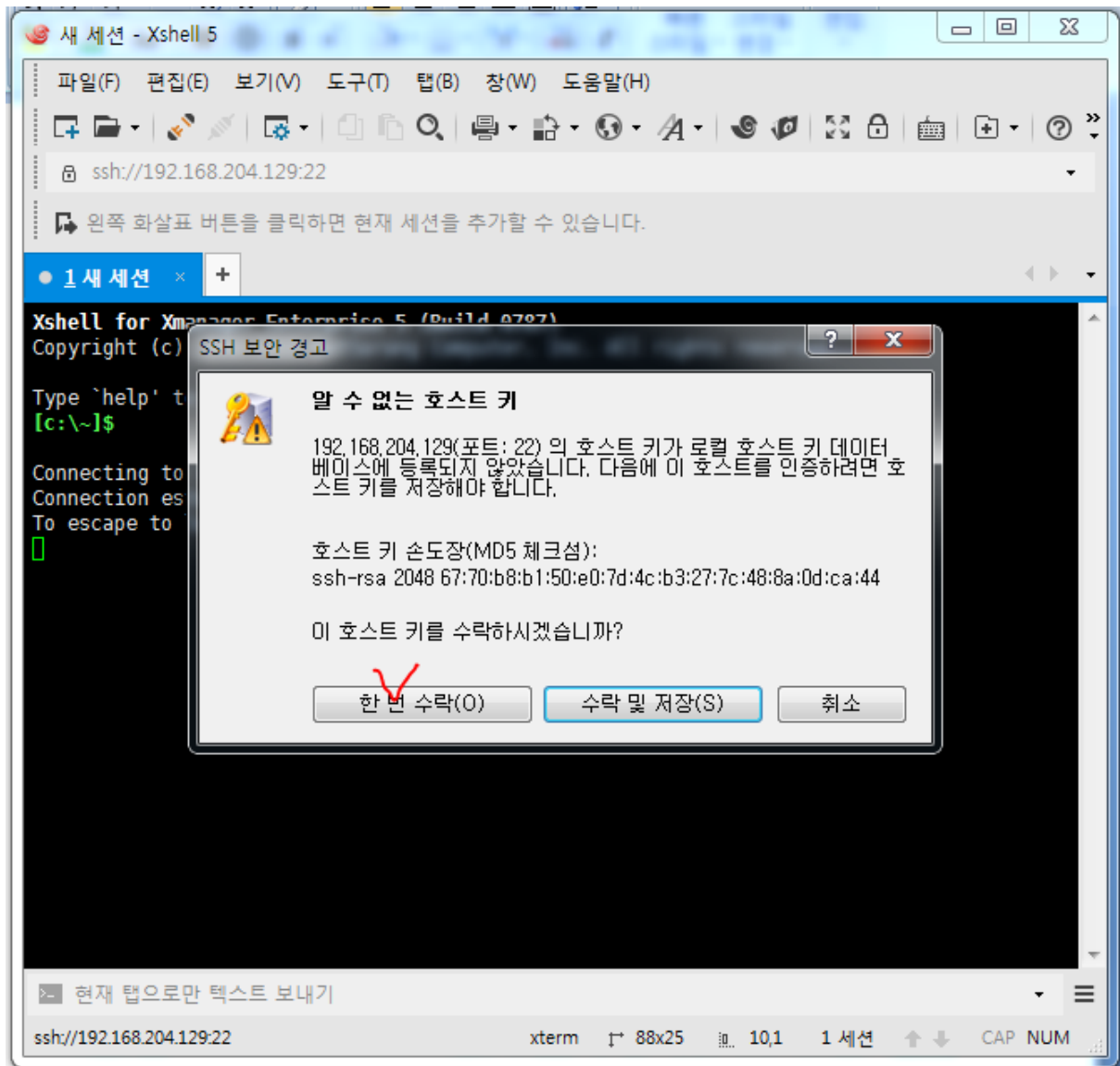
윈도우에서 원격 접속 (서버 접속) 프로그램

시작-설치폴더-Xshell : 원격 접속 프로그램



이름 : 그냥 뒤도 됨

호스트 : ip 적기(우분투 ifconfig)



+ 우분투 사용자 이름, 비밀번호 적기