

# 빅데이터 분석 실무과정

- 2018년 하반기 (8.23~10.11)

Part IV : 파이썬 머신러닝

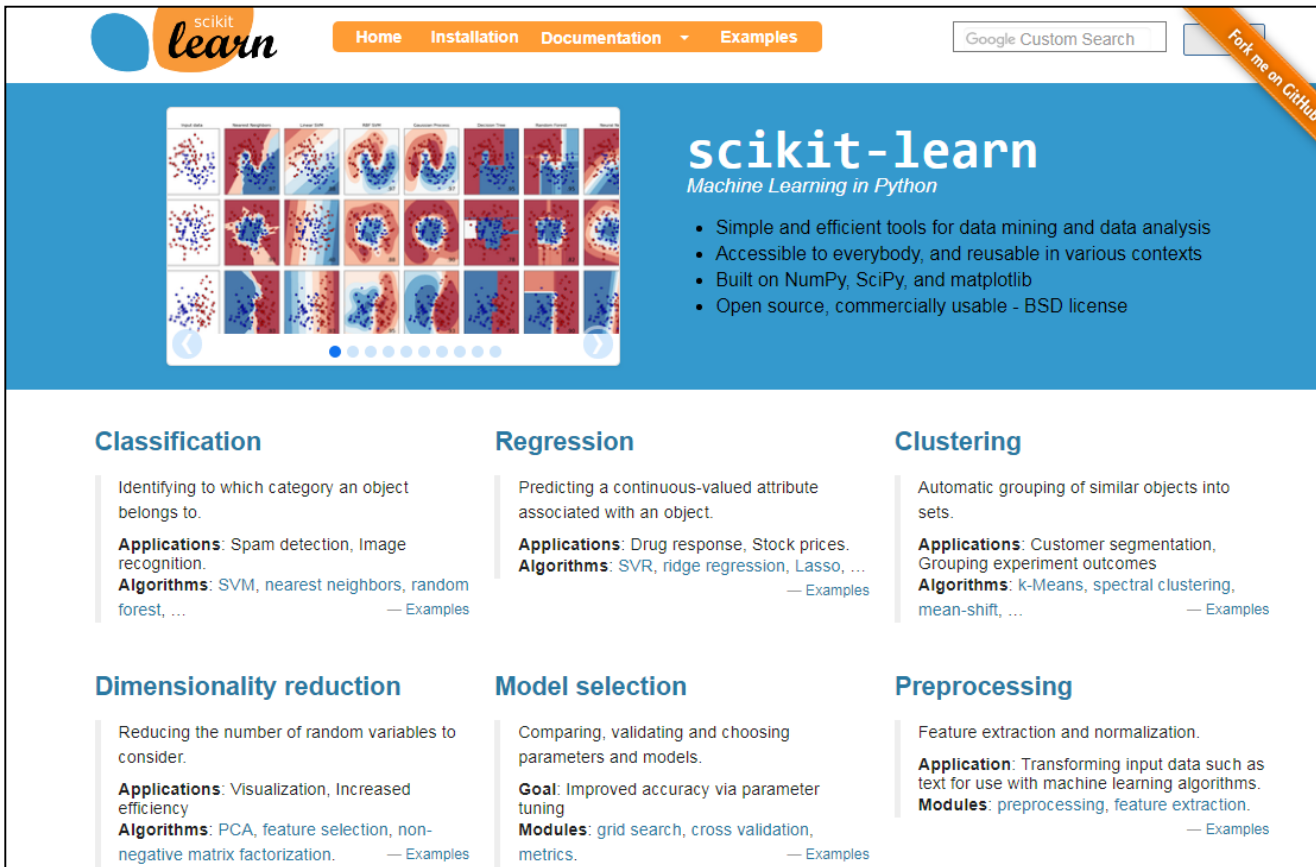
# | Part IV. 파이썬 머신러닝 - 목차

1. 데이터 불러오기/둘러보기
2. 선형 회귀
3. k-NN
4. 선형 분류 (로지스틱 회귀, 선형 SVM)
5. 결정 트리
6. SVM
7. k-means
8. 특성 분석
9. 교차검증/오차행렬/원핫인코딩
10. 파이프라인

# Scikit-learn – Python 머신러닝 모듈

## ▶▶ scikit-learn.org

- API reference → <http://scikit-learn.org>



The screenshot shows the scikit-learn.org homepage. At the top, there is a navigation bar with links for Home, Installation, Documentation, and Examples. A Google Custom Search bar is also present. The main header features the scikit-learn logo and the tagline "Machine Learning in Python". Below this, a grid of 12 small plots illustrates various machine learning concepts. To the right of the grid, a list of features is provided: Simple and efficient tools for data mining and data analysis, Accessible to everybody, and reusable in various contexts, Built on NumPy, SciPy, and matplotlib, and Open source, commercially usable - BSD license. The page is divided into six sections: Classification, Regression, Clustering, Dimensionality reduction, Model selection, and Preprocessing. Each section contains a brief description, applications, and algorithms.

**scikit-learn**  
*Machine Learning in Python*

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification	Regression	Clustering
Identifying to which category an object belongs to.	Predicting a continuous-valued attribute associated with an object.	Automatic grouping of similar objects into sets.
<b>Applications:</b> Spam detection, Image recognition.	<b>Applications:</b> Drug response, Stock prices.	<b>Applications:</b> Customer segmentation, Grouping experiment outcomes
<b>Algorithms:</b> SVM, nearest neighbors, random forest, ...	<b>Algorithms:</b> SVR, ridge regression, Lasso, ...	<b>Algorithms:</b> k-Means, spectral clustering, mean-shift, ...
— Examples	— Examples	— Examples

Dimensionality reduction	Model selection	Preprocessing
Reducing the number of random variables to consider.	Comparing, validating and choosing parameters and models.	Feature extraction and normalization.
<b>Applications:</b> Visualization, Increased efficiency	<b>Goal:</b> Improved accuracy via parameter tuning	<b>Application:</b> Transforming input data such as text for use with machine learning algorithms.
<b>Algorithms:</b> PCA, feature selection, non-negative matrix factorization.	<b>Modules:</b> grid search, cross validation, metrics.	<b>Modules:</b> preprocessing, feature extraction.
— Examples	— Examples	— Examples

# Scikit-learn – 데이터 불러오기

## ▶ sklearn 이 제공하는 기본 데이터를 사용하자

- 아래 예제는 Iris 데이터를 불러온다
- 데이터의 샘플은 총 150개, 속성은 4개이다
- 150개 샘플마다 타겟값이 지정되어 있다 (setosa:0, versicolor:1, virginica:2)

```
In []: from sklearn.datasets import load_iris
In []: iris = load_iris()

In []: iris.keys()
dict_keys(['data', 'feature_names', 'target_names', 'target', 'DESCR'])

In []: iris['feature_names'] # 속성의 이름 확인
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
In []: iris['target_names'] # 타겟의 이름 확인
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')

In []: iris_data=iris['data'] # iris.data 와 같음
In []: type(iris_data), iris_data.shape
(numpy.ndarray, (150, 4)) # 데이터 샘플의 개수는 150개, 속성의 개수는 4개이다

in []: iris_target = iris['target'] # iris.target 과 같음
In []: type(iris_target), iris_target.shape
(numpy.ndarray, (150,)) # 각 샘플당 타겟값이 지정되어 있다
```

# Scikit-learn – train/test 데이터 분리, 과적합

## ▶▶ train\_test\_split()

- 학습방법에 따라 학습결과가 학습한 데이터에 너무 딱 맞게 될 수가 있다
- 이러한 경우를 과적합이라고 하는데, 이렇게 되면 새로운 데이터를 제대로 평가하지 못하는 경우가 생긴다
- 그래서 데이터는 되도록이면 학습용과 테스트용으로 분리해 적용한다

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> from sklearn.model_selection import train_test_split

>>> X_train,X_test,y_train,y_test = train_test_split(iris['data'],iris['target'])

>>> print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
(112, 4) (38, 4) (112,) (38,)
>>> print(X_train[y_train==0].shape, X_train[y_train==1].shape, X_train[y_train==2].shape)
(40, 4) (33, 4) (39, 4)
>>> print(X_test[y_test==0].shape, X_test[y_test==1].shape, X_test[y_test==2].shape)
(10, 4) (17, 4) (11, 4)
```

[http://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.train\\_test\\_split.html](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

# Scikit-learn – 데이터 둘러보기

## ▶ X\_train 데이터를 속성별로 그래프를 그려보자

- 속성 별로 짝을 지어 6개의 플롯을 그린다
- plt.scatter() 를 주목하자 (c 는 색상값, s 는 점의 크기)

```
# ml_01.py
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

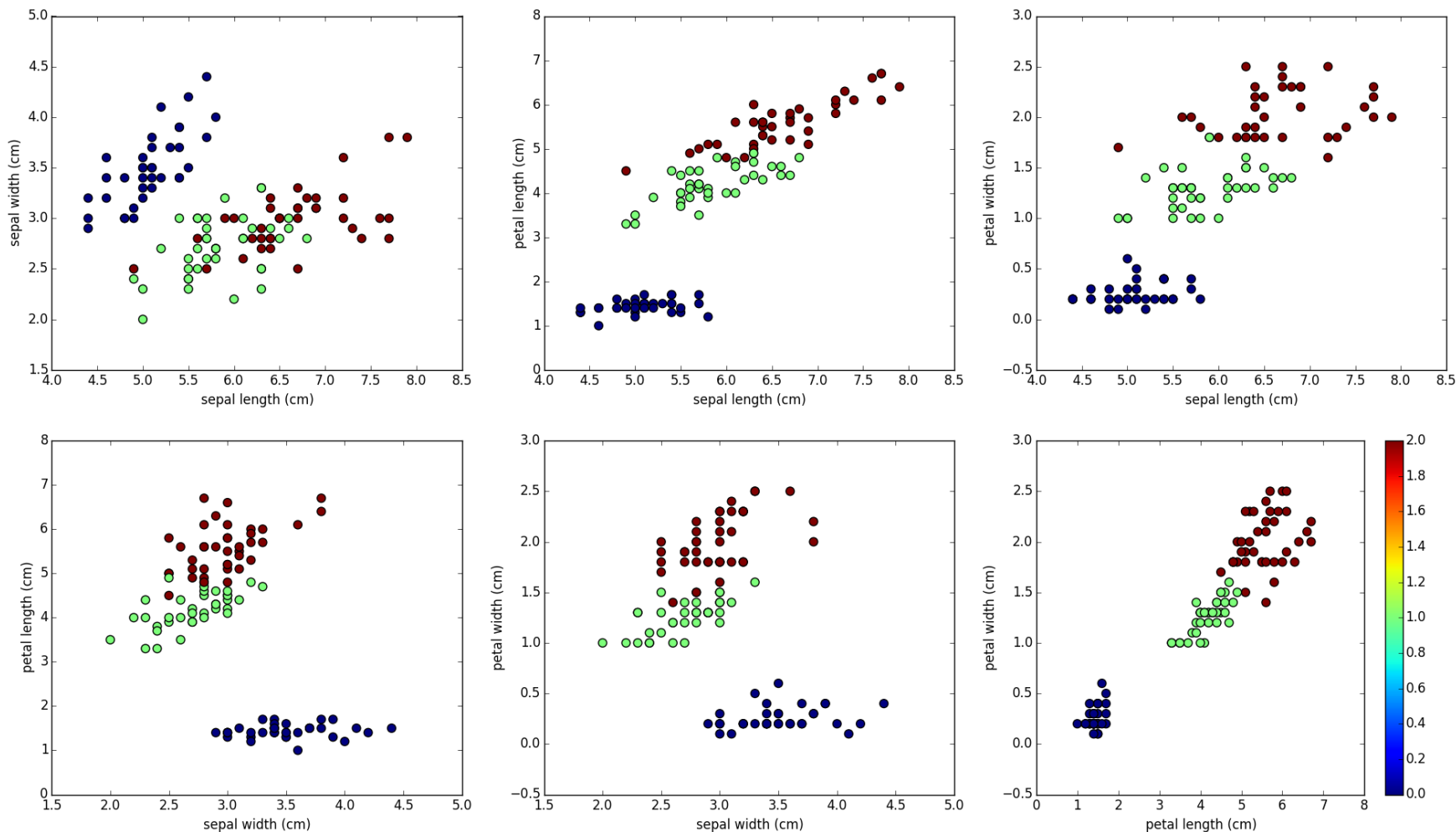
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris['data'], iris['target'])

fig=plt.figure()
fig.suptitle('Iris (setosa:0, versicolor:1, virginica:2)')
count=0
for i in range(4):
    for j in range(i+1,4):
        count+=1
        plt.subplot(2,3,count)
        plt.scatter(X_train[:,i],X_train[:,j],c=y_train,s=60)
        plt.xlabel(iris.feature_names[i])
        plt.ylabel(iris.feature_names[j])

plt.colorbar()
plt.show()
```

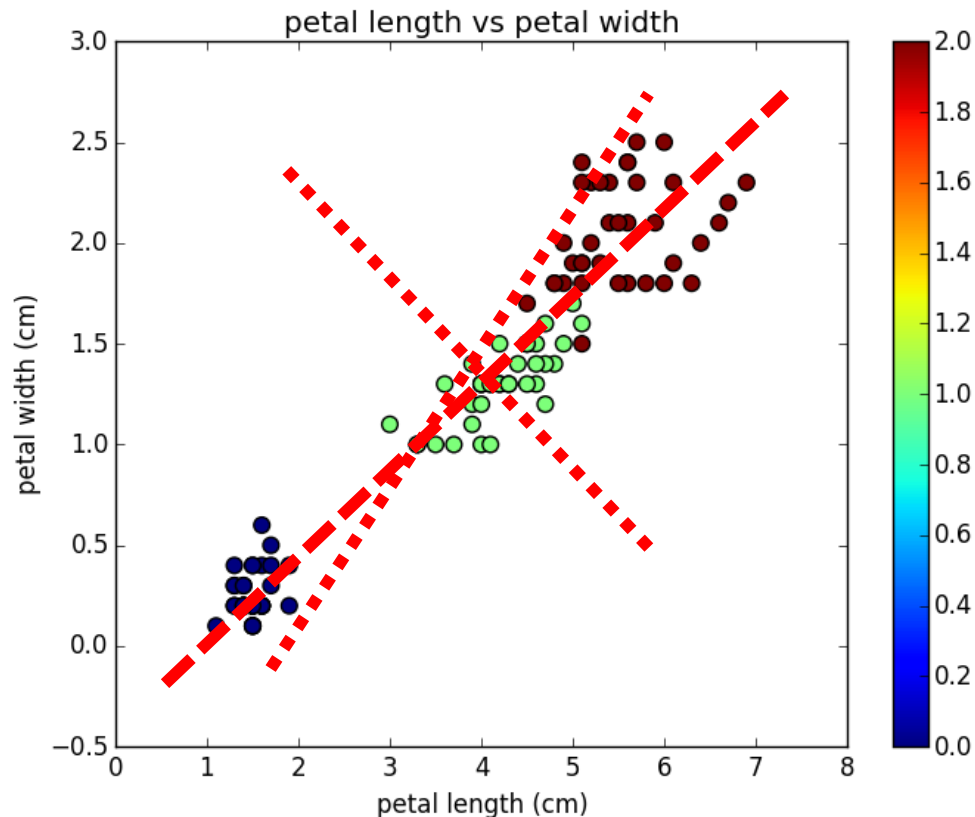
# Scikit-learn – 데이터 둘러보기 (결과)

Iris (setosa:0, versicolor:1, virginica:2)



# 선형회귀(Linear Regression)

- ▶ 선형회귀는 데이터에 근사하는 직선 또는 평면(초평면)을 그리는 것이다
  - 앞장의 Iris 그래프에서 가장 직선에 근사하는 플롯은 petal length vs petal width 이다
  - 데이터에 근사하는 직선을 그려보자





# 선형회귀(Linear Regression)

## ▶ sklearn.linear\_model.LinearRegression 을 사용한다

- fit() 함수의 첫번째 인자는 shape 가 2차원 행렬 형태여야 한다 (샘플수 \* 속성수)
- 데이터에 근사하는 직선을 그려보자

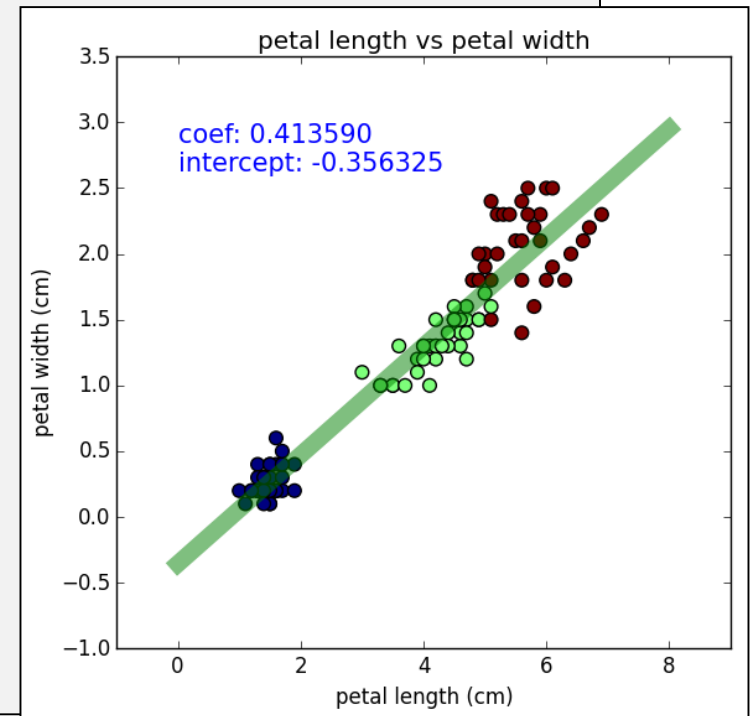
```
# ml_02.py
...
from sklearn.linear_model import LinearRegression

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

model = LinearRegression()
model.fit(X_train[:,2].reshape(-1,1),X_train[:,3])

w = model.coef_[0] # 기울기
b = model.intercept_ # 절편

plt.title('petal length vs petal width')
plt.scatter(X_train[:,2],X_train[:,3],c=y_train,s=60)
plt.xlabel(iris.feature_names[2])
plt.ylabel(iris.feature_names[3])
plt.plot([0,8],[0*w+b,8*w+b], 'g', lw=10, alpha=0.5)
plt.text(0,3,'coef: %f\nintercept: %f' % (w,b),
        va='top', fontsize=15, color='b')
plt.show()
```



# k-최근접 이웃 (k-NN) 분류

## ▶ 가장 근처에 있는 k개의 점을 가지고 타겟값을 판단한다

- k-NN 은 분류(Classification) 알고리즘이다
- k값 즉, 근처 몇 개의 점을 판단 기준으로 할 것인지가 이슈이다

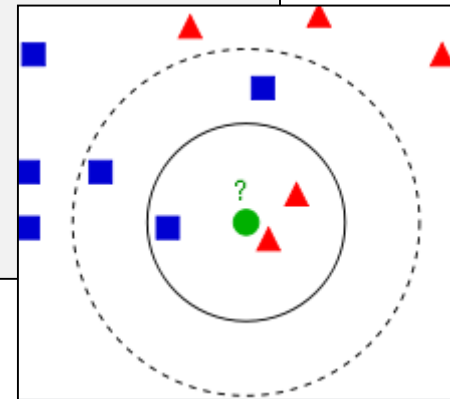
```
# ml_03.py
...
from sklearn.neighbors import KNeighborsClassifier

iris = load_iris()
X_train,X_test,y_train,y_test = train_test_split(iris.data,iris.target,random_state=3)

model=KNeighborsClassifier(3) # try various k-value (1,3,5,...), default=5
model.fit(X_train,y_train)

pred_y=model.predict(X_test)
print('Score: %f' % model.score(X_test,y_test))

print(pred_y)
print(y_test)
print('Score: %f (Error: %d)' %
      (np.mean(pred_y==y_test),np.sum(pred_y!=y_test)) )
print(np.where(pred_y!=y_test))
print(model.predict_proba(X_test[16:17]))
```

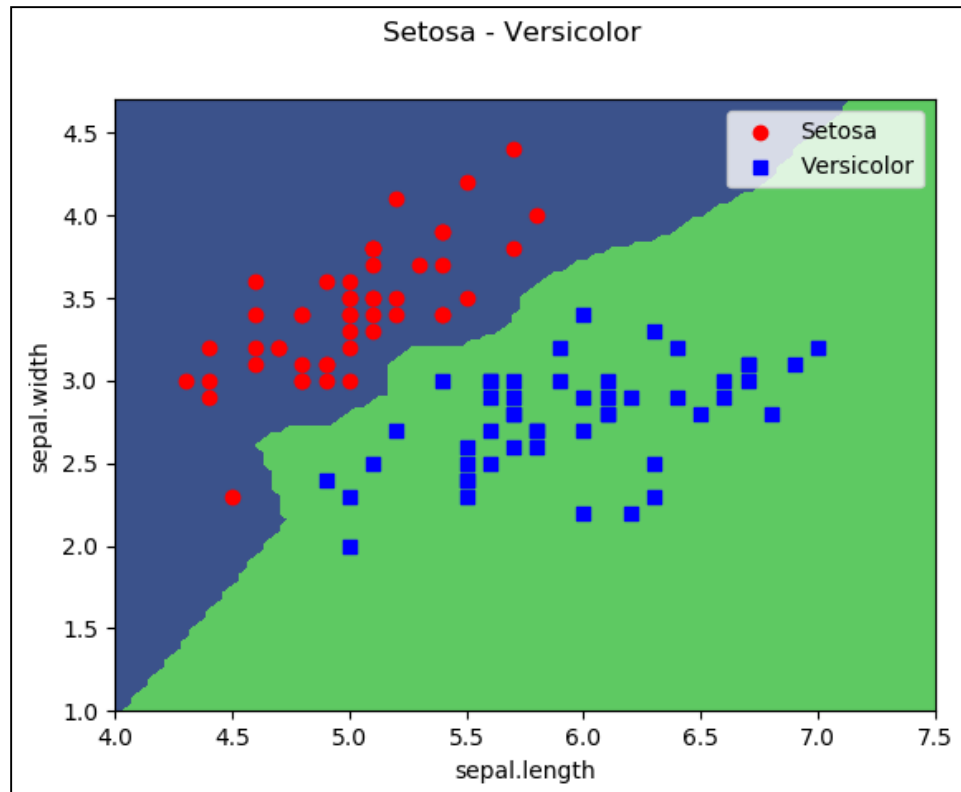


<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier>  
[https://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)

# k-최근접 이웃 (k-NN) 분류

## ▶▶ 그래프 예시

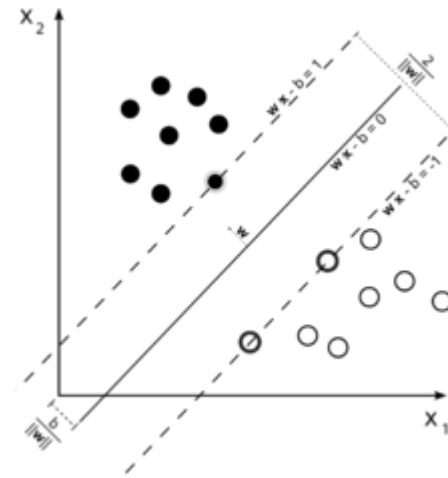
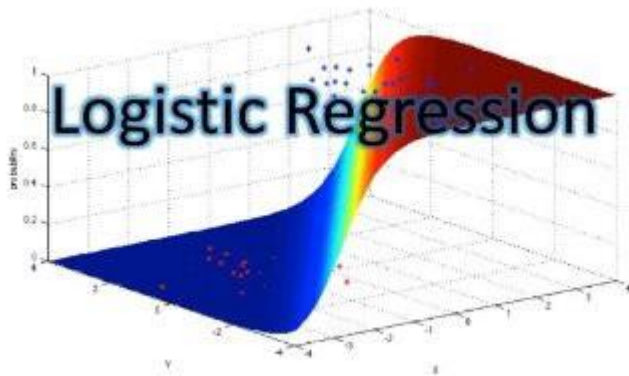
- ml\_04.py 소스 참고
- Setosa 와 Versicolor 에서 sepal width 와 sepal length 속성만 고려했을 때 ( $k=1$ )



# 선형 분류

## ▶▶ 공간 상에서 초평면으로 점들을 분류하는 알고리즘

- 로지스틱 회귀 (Logistic Regression)
  - `sklearn.linear_model.LogisticRegression`
  - sigmoid 함수 사용
- 선형 서포트 벡터 머신 (Linear Support Vector Machine)
  - `sklearn.svm.LinearSVC`
  - 최대 마진(margin)을 가지는 초평면을 찾음



## ▶ 회귀 라는 용어를 사용하지만 분류 알고리즘이다

- 타겟값을 속성처럼 취급하여 회귀를 적용하므로 회귀란 용어가 붙었다
- 중요한 옵션은 C 이다. C 값이 클수록 과적합 되는 경향이 있다.

```
# ml_05.py
...
from sklearn.linear_model import LogisticRegression

iris = load_iris()
X_train,X_test,y_train,y_test = train_test_split(iris.data,iris.target,random_state=1)

model = LogisticRegression() # default value C=1.0
model.fit(X_train,y_train)

pred_y=model.predict(X_test)
print('Score: %f' % model.score(X_test,y_test))

print(pred_y)
print(y_test)
print('Score: %f (Error: %d)' % (np.mean(pred_y==y_test),np.sum(pred_y!=y_test)) )
print(np.where(pred_y!=y_test))
print(model.predict_proba(X_test[5:6]))

print(model.coef_) # 3*4 어레이
print(model.intercept_) # 3개의 값
```

[http://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

## ▶ 결과값인 coef\_ 와 intercept\_ 로 플롯을 그려보자

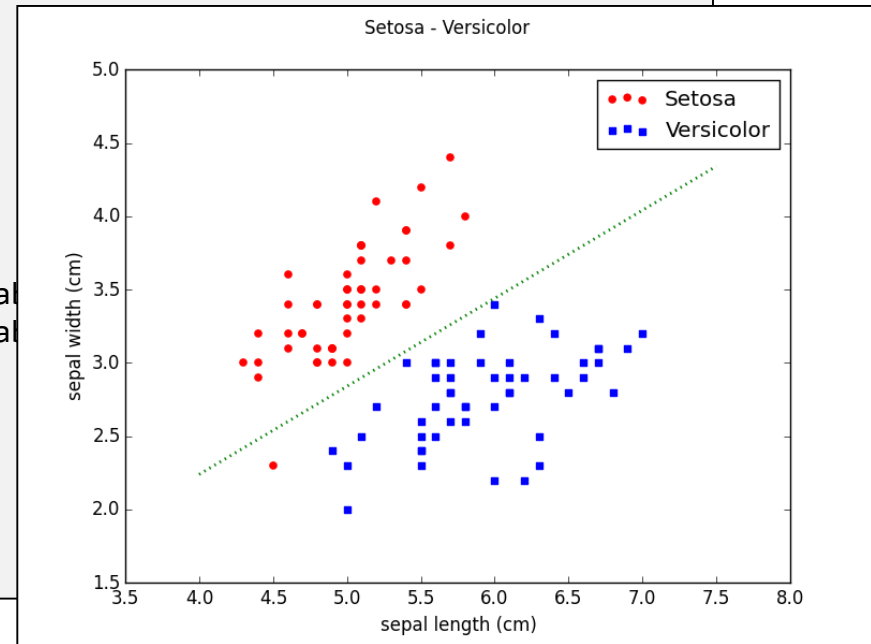
- Setosa 와 Versicolor 에서 sepal width 와 sepal length 속성만 고려했을 때
- $w_1 * x_1 + w_2 * x_2 + b = 0 \rightarrow x_2 = -(w_1 * x_1 + b) / w_2$

```
# ml_06.py
...
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X,y)
pred_y = model.predict(X)

w = model.coef_ # 1*2 어레이
b = model.intercept_

fig=plt.figure()
fig.suptitle('Setosa - Versicolor')

plt.scatter(X1[:,0],X1[:,1],marker='o',color='r',label='Setosa')
plt.scatter(X2[:,0],X2[:,1],marker='s',color='b',label='Versicolor')
dots=np.array([4,7.5])
plt.plot(dots, -(w[0,0]*dots+b)/w[0,1], 'g:', lw=2)
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.legend()
plt.show()
```



## ▶▶ 초평면으로 클래스를 분류한다

- 이론적인 내용은 뒷부분에 나오는 SVM 을 참고하자
- 옵션 값인 C 를 변경하며 테스트해 보자
- 로지스틱회귀 처럼 결과값인 coef\_ 와 intercept\_ 를 활용할 수 있다

```
# ml_07.py

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state=1)

for C in [0.1, 1, 10, 100]:
    model = LinearSVC(C=C) # default value C=1.0
    model.fit(X_train, y_train)

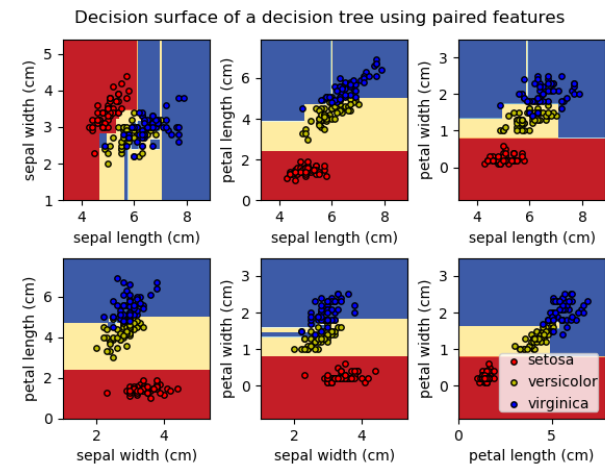
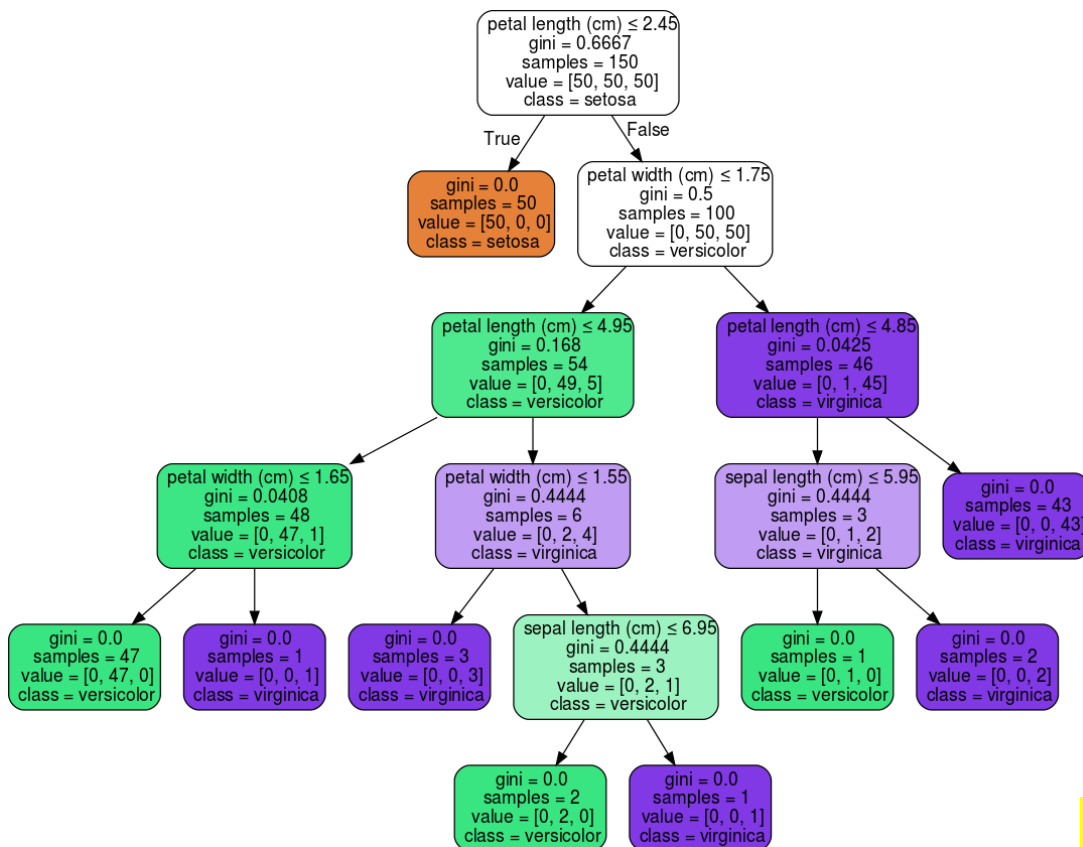
    print('== %f ==' % C)
    print('Train Score: %f' % model.score(X_train, y_train))
    print('Test Score: %f' % model.score(X_test, y_test))
```

<http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html>

# 결정 트리 (Decision Tree)

## ▶▶ 각 속성의 경계값을 기준으로 다단계로 가지를 친다

- 각 단계에서 가장 분류를 잘하는 속성과 경계값을 찾는다
- 왼쪽 또는 오른쪽으로 가지를 치며 값을 찾아간다



<http://scikit-learn.org/stable/modules/tree.html>



# 결정 트리 (Decision Tree)

## ▶ sklearn.tree.DecisionTreeClassifier

- 옵션 : max\_depth, max\_leaf\_nodes, min\_samples\_leaf
- 결과값 : feature\_importances\_, tree\_

```
# ml_08.py

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state=1)

model = DecisionTreeClassifier(max_depth=2)
model.fit(X_train, y_train)

pred_y = model.predict(X_test)
print('Train Score: %f' % model.score(X_train, y_train))
print('Test Score: %f' % model.score(X_test, y_test))

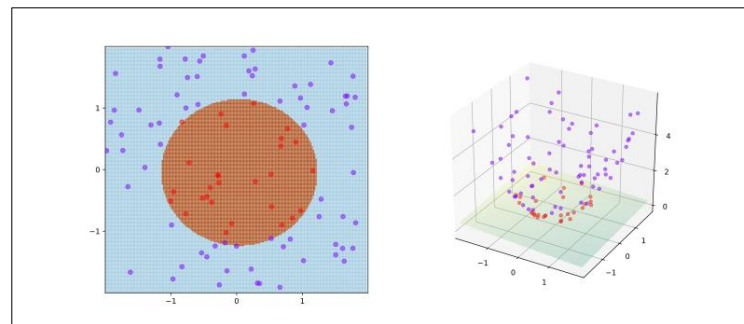
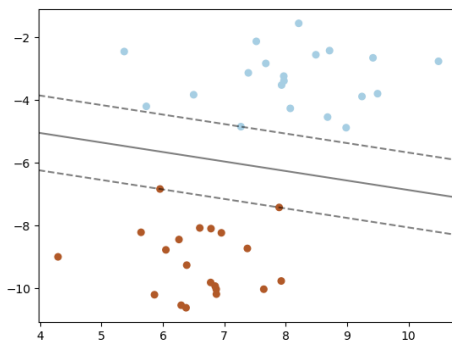
print(model.feature_importances_)
print(model.tree_) # help(model.tree_)
```

<http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

# Support Vector Machine(SVM)

## ▶ SVM 은 두 클래스를 구분하는 거리를 최대로 만드는 평면을 찾는다

- 아래 그림에서 점선에 걸린 점들을 서포트벡터 라고 하고, 실선과 점선 사이의 거리를 margin 이라고 한다.
- margin 을 최대가 되게 하는 서포트벡터를 찾는 것이 목표이다
- 두번째 그림에서는 평면으로 나눌수 없다. 이럴때,  $z=x^2+y^2$  과 같이 차원을 추가하면 고차원에서 클래스를 구분할 수 있다. → 커널 기법
- 커널 기법을 적용하면 복잡한 곡선 형태의 결과를 얻을 수 있다
- 커널 SVM 은 `sklearn.svm.SVC` 에 구현되어 있다
- 고려할 점들
  - 커널 SVM 은 가장 강력한 머신러닝 기법 중의 하나이지만, 결과를 해석하기가 쉽지 않다
  - 속도와 메모리 이슈
  - 정규화와 같은 전처리가 필요함



<http://scikit-learn.org/stable/modules/svm.html>  
[https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)

# | Support Vector Machine(SVM)

## ▶▶ sklearn.svm.SVC 적용

- 옵션 kernel : linear, poly, rbf, sigmoid 등 (기본값은 rbf)
- 옵션 C 와 gamma : 둘 다 값이 클수록 과적합됨
- 아래 소스의 결과는 60%대로 좋지 못하다. 정규화가 필요함!

```
# ml_09.py
...
from sklearn.datasets import load_iris, load_breast_cancer
from sklearn.svm import SVC

cancer = load_breast_cancer()
X_train,X_test,y_train,y_test = train_test_split(cancer.data,cancer.target)

for C in [0.1,1,1000]:
    for gamma in [0.1,1,10]:
        model=SVC(C=C,gamma=gamma)
        model.fit(X_train,y_train)

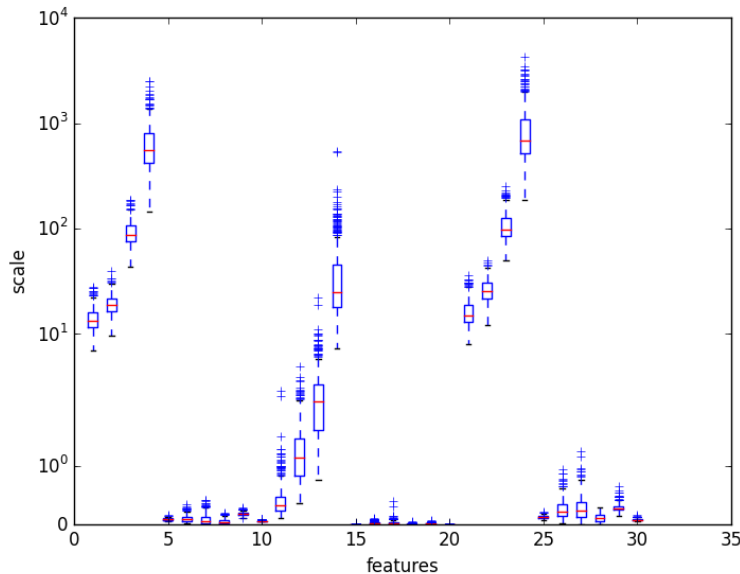
        pred_y=model.predict(X_test)
        print('\n>>> C: %.1f, gamma: %.1f' % (C,gamma))
        print('Train Score: %f' % model.score(X_train,y_train))
        print('Test Score: %f' % model.score(X_test,y_test))
```

<http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

# SVM 정규화

## ▶ SVM 은 정규화가 꼭 필요하다

- 값이 촘촘한 속성들이 제대로 분류되지 못하기 때문이다
- 아래 그림은 유방암 데이터에서 각 속성값들의 스케일을 표시한 것이다
- 속성값들의 범위가 제각각인 것을 알 수 있다
- SVM 에서는 항상 속성값을 확인하고, 반드시 정규화를 염두에 두어야 한다



```
plt.boxplot(X_train,manage_xticks=False)
plt.yscale('symlog') # 로그 스케일
plt.xlabel('features')
plt.ylabel('scale')
plt.show()
```

# SVM 정규화 적용 결과

## ▶▶ 정규화 방법

- 평균과 표준편차를 이용하는 방법
- 0~1 사이의 값으로 맞추는 방법
- 아래는 평균과 표준편차를 사용했으며 테스트셋에서 90% 이상의 결과를 얻는다

```
# ml_10.py
...
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC

cancer = load_breast_cancer()
X_train,X_test,y_train,y_test = train_test_split(cancer.data,cancer.target)

X_mean = X_train.mean(axis=0) # 평균과 표준편차는 X_train 을 이용해 얻어야 한다
X_std = X_train.std(axis=0)
X_train_scaled = (X_train-X_mean)/X_std
X_test_scaled = (X_test-X_mean)/X_std # X_test 를 X_train 기준으로 변환해야 한다

for C in [0.1,1,1000]:
    for gamma in [0.1,1,10]:
        model=SVC(C=C,gamma=gamma)
        model.fit(X_train_scaled,y_train)

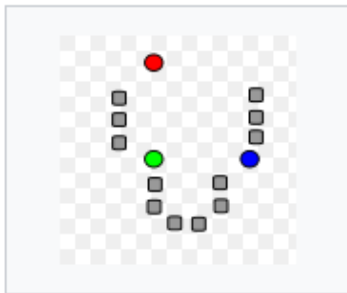
        pred_y=model.predict(X_test_scaled)
        print('\n>>> C: %.1f, gamma: %.1f' % (C,gamma))
        print('Train Score: %f' % model.score(X_train_scaled,y_train))
        print('Test Score: %f' % model.score(X_test_scaled,y_test))
```

# k-means 군집화(Clustering)

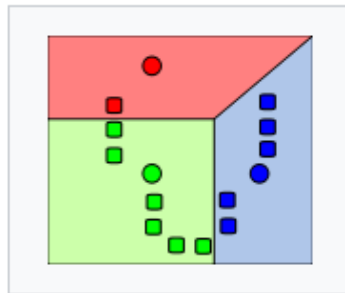
## ▶ 군집화는 데이터를 비슷한 것끼리 그룹으로 묶어주는 것이다

- 이전 분류와는 다르게 타겟값이 없는 경우이다
- k-means 는 대표적인 군집화 알고리즘으로 간단하고 빠르다
- 하지만 k-means 는 복잡한 형태의 데이터에는 적용하기 어렵다
- k-means 는 임의로 중심점을 잡고 평균위치로 이동하는 것을 단계적으로 반복한다
- 데이터의 특성에 맞게 전처리, 정규화, 차원변환 등을 검토해야 한다

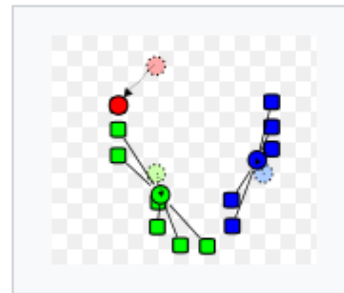
Demonstration of the standard algorithm



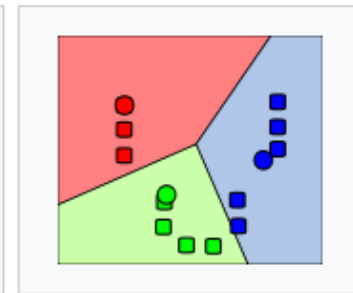
1.  $k$  initial "means" (in this case  $k=3$ ) are randomly generated within the data domain (shown in color).



2.  $k$  clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.



3. The centroid of each of the  $k$  clusters becomes the new mean.



4. Steps 2 and 3 are repeated until convergence has been reached.

[https://en.wikipedia.org/wiki/K-means\\_clustering](https://en.wikipedia.org/wiki/K-means_clustering)  
<http://scikit-learn.org/stable/modules/clustering.html>

# k-means 적용

## ▶ sklearn.cluster.KMeans

- 옵션 `n_clusters` : 분류할 클래스 개수
- 결과 : `labels_` (할당된 클래스아이디), `cluster_centers_` (중심점 좌표)

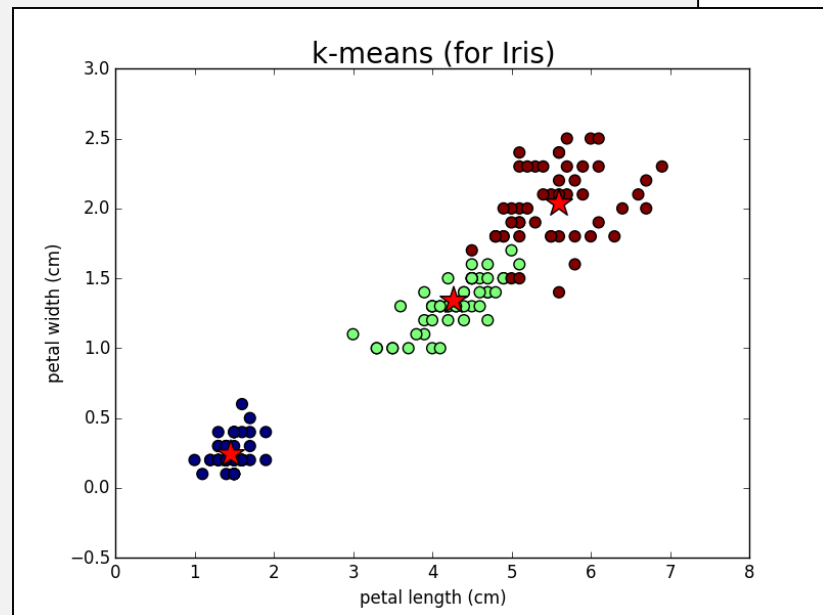
```
# ml_11.py
...
from sklearn.cluster import KMeans

iris = load_iris()
X = iris.data[:,2:] # petal length and petal width
y = iris.target

kmeans = KMeans(n_clusters=3) # 3 classes
kmeans.fit(X)

print(kmeans.labels_) # 결과값

plt.title('k-means (for Iris)', fontsize=20)
plt.scatter(X[:,0], X[:,1], c=y, s=60)
cc=kmeans.cluster_centers_ # 3개의 중심점
plt.scatter(cc[:,0], cc[:,1], s=400, marker='*', c='r')
plt.xlabel(iris.feature_names[2])
plt.ylabel(iris.feature_names[3])
plt.show()
```

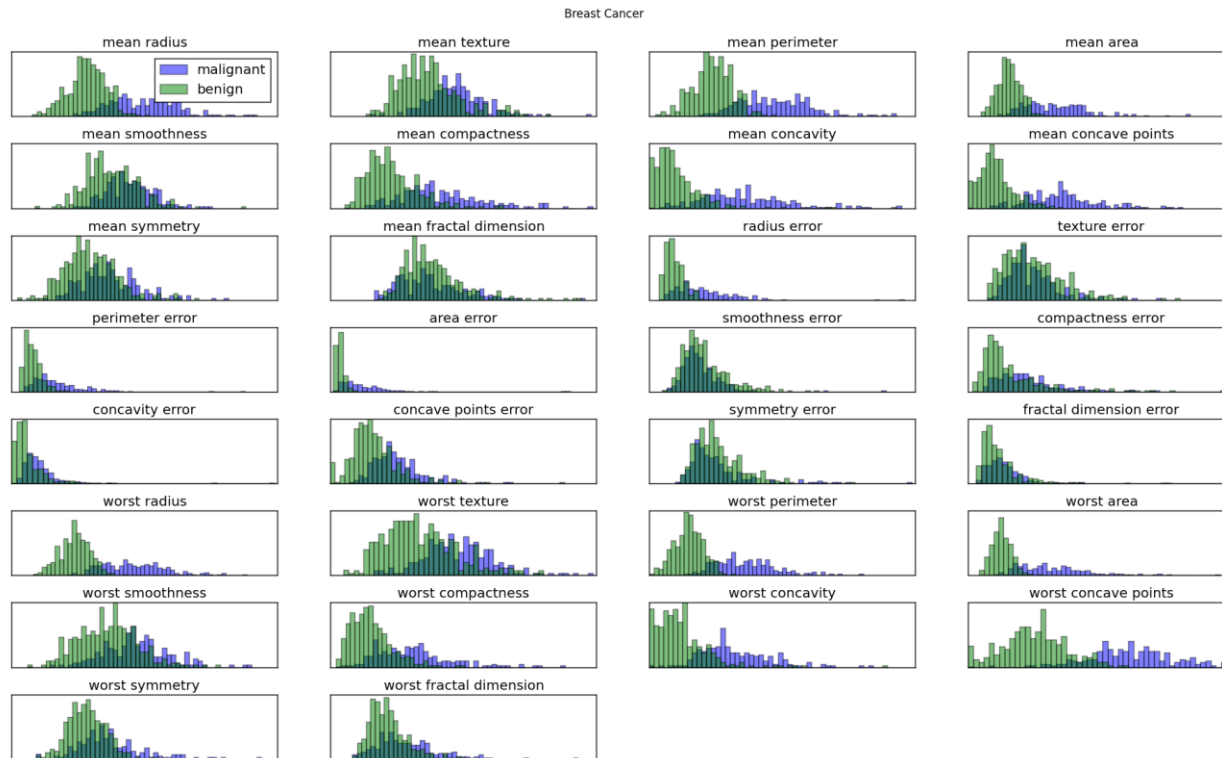


<http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

# 유방암 데이터 특성 분석

## ▶ 머신러닝의 첫단계는 데이터의 특성 분석이어야 한다

- 각 속성들의 통계적 분석과 시각적인 판단을 우선 수행하여야 한다
- 아래는 각 속성들을 히스토그램으로 표시한 것이다 (ml\_12.py 참고)
- 히스토그램을 보면서 중요한 속성이 무엇일지 생각해보자

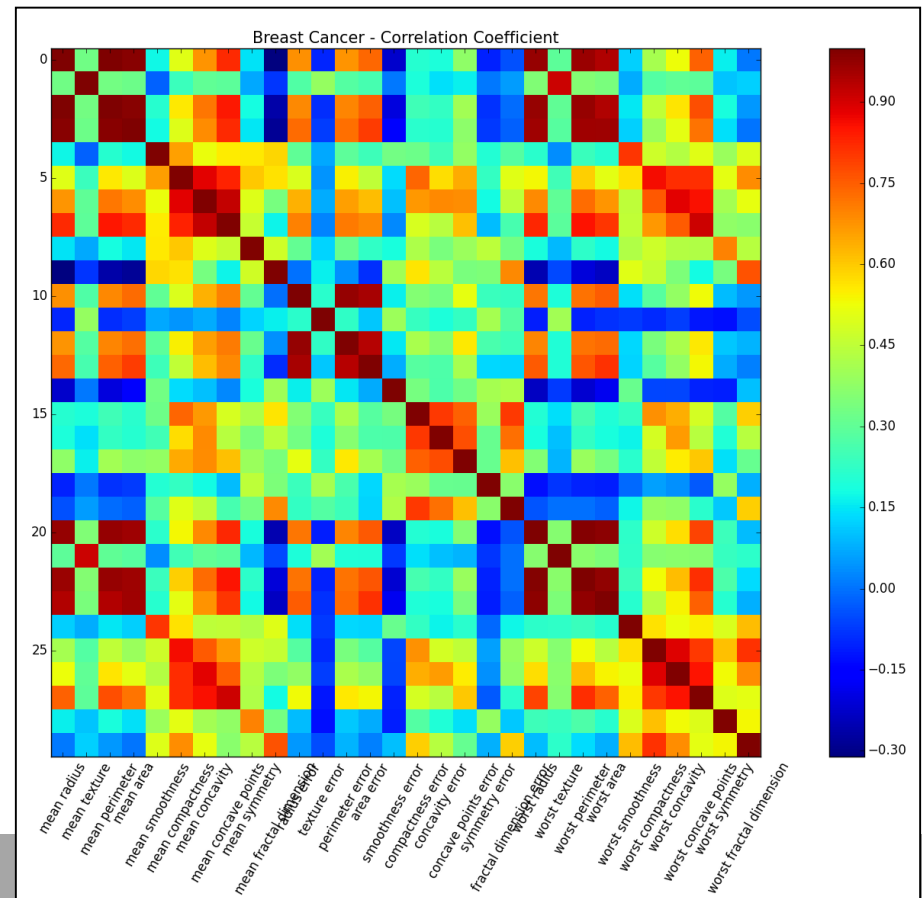
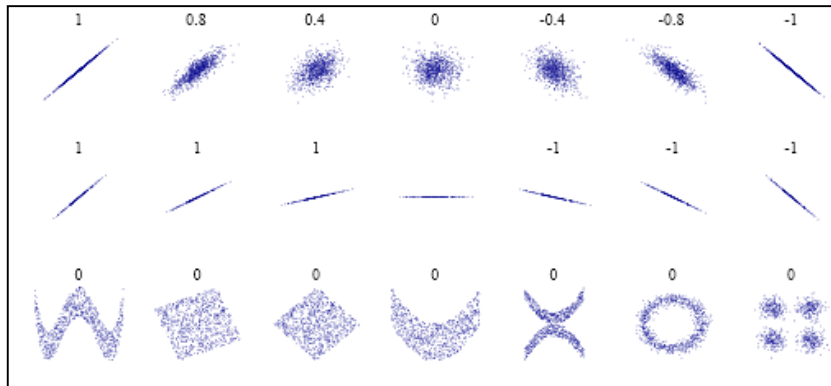




# 상관 계수 (Correlation Coefficient)

## ▶ 상관계수는 속성들 간에 연관되어 있는 정도를 -1~1 사이의 수치로 계산한 것이다

- 한속성이 증가할 때 다른 속성도 같이 증가하면 +1, 반대로 감소하면 -1
- 두 속성이 전혀 상관 없을 때 0
- 두 속성이 +1이나 -1에 가깝다면, 두 속성은 거의 같기 때문에 한 속성은 제외할 수도 있다
- 히트맵은 ml\_18.py 소스 참조



# 교차 검증 (Cross-validation)

## ▶ 데이터를 훈련세트와 테스트세트로 나누지 않고 원본데이터를 구간별로 분할

- 예를 들어 Iris 처럼 150개의 데이터가 있을 때 3개의 구간으로 나눈다면, 1구간과 2구간을 훈련세트로 3구간을 테스트세트로 잡을 수 있다. 마찬가지로 1구간과 3구간을 훈련세트, 2구간과 3구간을 훈련세트로 하는 3가지 구성을 할 수 있다.
- 이를 통해, 3번의 분석 결과를 얻을 수 있다. → 데이터의 개수가 적을 때 유용하다
- `sklearn.model_selection.cross_val_score()` 에서 cv 값에 구간의 개수를 넘긴다
- `cross_val_score()` 는 기본적으로 타겟값의 비율 대로 (stratified) 각 구간의 데이터를 뽑는다

```
# ml_13.py
...
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

iris = load_iris()
model = SVC()

for cv in [2,3,4,5]:
    scores = cross_val_score(model,iris.data,iris.target,cv=cv)
    print('cv=%d :' % cv,scores)

'''=====RESULT=====
cv=2 : [ 0.96          0.93333333]
cv=3 : [ 0.98039216  0.96078431  0.97916667]
cv=4 : [ 0.97435897  1.          0.94444444  0.97222222]
cv=5 : [ 0.96666667  1.          0.96666667  0.96666667  1.          ]
'''
```

# 교차검증 상세 옵션

## ▶ 옵션인 cv 에 교차검증기를 할당 할 수 있다

- KFold(n\_splits=3, shuffle=False, random\_state=None) : 기본적으로는 데이터 순서대로 나눔, shuffle=True 로 데이터를 섞을 수 있다
- StratifiedKFold(n\_splits=3, shuffle=False, random\_state=None) : 타겟값의 비율대로 각 구간의 데이터 비율 정함

```
# ml_14.py
...
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold, StratifiedKFold

iris = load_iris()
model = SVC()
kfold=KFold(n_splits=5,shuffle=True,random_state=55)

scores = cross_val_score(model,iris.data,iris.target,cv=kfold)
print(scores)

''' ===RESULT===
[ 0.96666667  0.96666667  0.9          0.9          1.          ]
'''
```

[http://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.KFold.html](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html)

[http://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.StratifiedKFold.htm](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.htm)

# 오차 행렬 (confusion matrix)

## ▶ 실제 타겟값과 예측값과의 관계를 보여준다

- 아래 표에서 보듯이 4가지 결과를 보여준다.
- $y_{\text{test}}=0$ ,  $\text{pred}_y=0$  인 것이 50개,  $y_{\text{test}}=0$ ,  $\text{pred}_y=1$  인 것이 5개 등등

```
# ml_15.py
...
from sklearn.metrics import confusion_matrix

cancer = load_breast_cancer()
X_train,X_test,y_train,y_test = train_test_split(cancer.data,cancer.target,random_state=1)

model = KNeighborsClassifier()
model.fit(X_train,y_train)

pred_y = model.predict(X_test)
print('Test score :',np.mean(pred_y == y_test))

print('confusion matrix :')
print(confusion_matrix(y_test,pred_y))
```

타겟값	예측 0	예측 1
실제 0	50 (TP)	5 (FN)
실제 1	4 (FP)	84 (TN)

[http://scikit-learn.org/stable/modules/model\\_evaluation.html#confusion-matrix](http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix)

[http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion\\_matrix.html](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

# | one-hot-encoding

## ▶ 수치화하기 어려운 속성을 0과 1로 이루어진 벡터로 바꾸어 준다

- 아래 표에서 '나이, 키, 지역'은 속성이고 '부자?'는 타겟값이다
- scikit-learn은 숫자만 다루므로, 지역을 서울(0), 부산(1), 대구(2)로 바꿀 수도 있다. 하지만 지역명은 순서나 크기를 가지는 값이 아니므로 이렇게 바꾸는 것은 옳지 않다.
- 서울 → (1,0,0), 부산 → (0,1,0), 대구 → (0,0,1)과 같이 바꾸어 주는 것을 원핫인코딩이라고 한다
- 변환 방법은 ml\_16.py 참조

나이	키	지역	부자?
25	172	서울	1
53	169	부산	0
...			
49	177	대구	1



나이	키	서울	부산	대구	부자?
25	172	1	0	0	1
53	169	0	1	0	0
...					
49	177	0	0	1	1

[http://scikit-learn.org/stable/modules/model\\_evaluation.html#confusion-matrix](http://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix)

[http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion\\_matrix.html](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

# | 파이프라인 (Pipeline)

## ▶ 전처리와 실제 분류 작업을 하나의 체인으로 묶어준다

- 아래 소스는 MinMaxScaler 전처리와 SVC 분류를 하나의 파이프라인으로 묶은 것이다
- 전처리 부분은 fit() 함수와 transform() 함수를 모두 가지고 있어야 한다.
- Pipeline() 에서 지정한 순서대로 작업이 차례대로 진행된다

```
# ml_17.py
...
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline

cancer = load_breast_cancer()
X_train,X_test,y_train,y_test = train_test_split(cancer.data,cancer.target)

scaler=MinMaxScaler().fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)

model=SVC()
model.fit(X_train_scaled,y_train)

print('Train Score: %f' % model.score(X_train_scaled,y_train))
print('Test Score: %f' % model.score(X_test_scaled,y_test))

pipe=Pipeline([ ('scaler',MinMaxScaler()), ('svm',SVC()) ])
pipe.fit(X_train,y_train)
print('Pipe Score : ',pipe.score(X_test,y_test))
```

<http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>  
<http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>

# | scikit-learn 작업 고려사항

## 1. 원본 데이터를 자세히 살펴본다

- 원본 데이터를 엑셀 등을 이용해 열어본다
- 수치가 아닌 문자열 등이 포함되었는지 확인한다
- 비어있거나 손상된 데이터가 없는지 확인한다
- 다양한 방법으로 시각화해 본다

## 2. 데이터 전처리

- 비어있거나 손상된 데이터를 보정한다
- 평균, 분산, 상관계수 등 통계적 분석을 시도한다
- 문자열을 숫자로 바꾸거나 원핫인코딩을 적용한다
- 필요시 데이터를 정규화하거나 구간값을 정리한다
- 다양한 방법으로 시각화해 본다

## 3. 데이터를 불러온다

- 속성값들과 타겟값을 분리하여 Numpy 어레이로 읽어온다
- `train_test_split()` 을 이용하여 훈련세트와 테스트세트를 분리한다
- 다양한 방법으로 시각화해 본다

## 4. 적용할 머신러닝 모델을 결정한다

- 타겟값이 있는지 유무, 분류/회귀에 따라 결정
- 데이터의 특성에 따른 기존 사례를 분석하여 가장 적합한 모델 선정
- 데이터가 클 경우, 작은 샘플들을 뽑아 다양한 모델에 적용해 보기

## 5. 결과 분석

- `score` 에만 연연하지 말고, 원본 데이터와 예측치가 샘플별로 어떤 점이 차이가 나는지 자세히 분석한다
- 결과를 시각화하여, 결과가 프로젝트 목표와 부합하는지 꼼꼼히 살펴본다

# 실습 문제

- ▶ 유방암 데이터의 속성들의 히스토그램과 상관계수 히트맵을 참조하여 아래 내용을 구현해 보자
  - 속성들 끼리의 산점도(scatter) 그래프를 그려보고, 가장 중요해 보이는 속성 2개를 선정
  - k-NN, 로지스틱회귀, 결정트리, SVC 중 가장 유용한 모델이 무엇인지 알아보자
  - 최종 결과를 최대한 깔끔하게 시각화해 보자

