AI 보안 기술개발 교육

머신러닝(Machine Learning) 모델 2

KISA 한국인터넷진흥원

AI 보안 기술개발 교육

머신러닝 모델 2

1. 로지스틱회귀 모델
2. K-근접이웃 모델
3. 나이브베이즈 모델
4. 서포트 벡터 머신
5. K-Means

KISA 한국인터넷진흥원

로지스틱 회귀(Logistic Regression)

- 로지스틱 회귀(Logistic Regression)

  회귀(Regression)를 사용하여 데이터가 어떤 범주에 속할 확률을 0에서 1 사이의 값으로 예측

  확률에 따라 가능성이 더 높은 범주에 속하는 것으로 분류해 주는 지도 학습 알고리즘

  메일에 대해 스팸일 확률이 0.5 이상이면 spam으로 분류하고, 확률이 0.5보다 작은 경우 ham으로 분류

  데이터가 2개의 범주 중 하나에 속하도록 결정하는 것을 2진 분류(binary classification)라고 함

  로지스틱 회귀에서 데이터가 특정 범주에 속할 확률을 예측하기 위한 과정

  ① 모든 속성(feature)들의 계수(coefficient)와 절편(intercept)을 0으로 초기화

  ② 각 속성들의 값(value)에 계수(coefficient)를 곱해서 log-odds를 계산

  ③ log-odds를 sigmoid 함수에 넣어서 [0,1] 범위의 확률 계산

로지스틱 회귀(Logistic Regression)

- Odds 란?

  특정 사건이 발생할 확률을 발생하지 않을 확률로 나누어 준 값

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

- log- Odds 란?

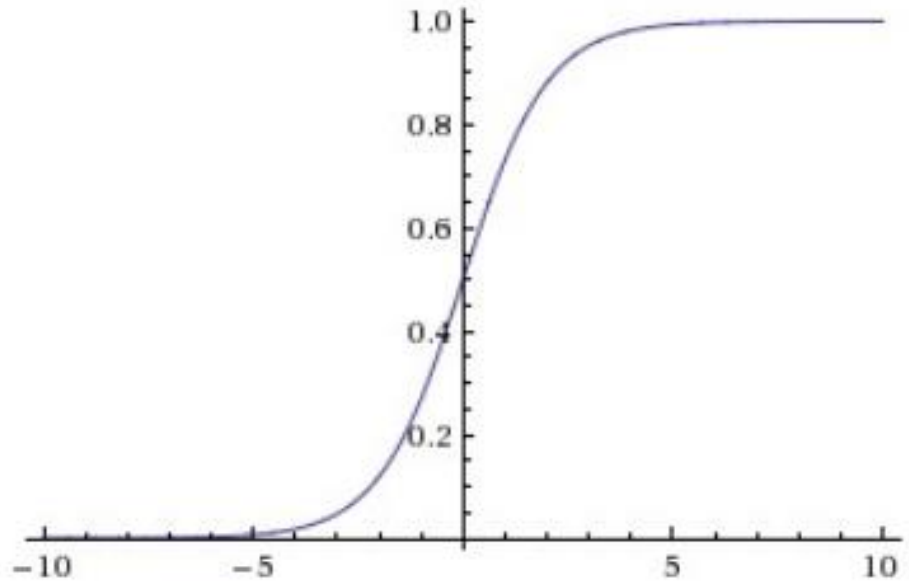
  Odds에 로그를 취한 값

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

로지스틱 회귀(Logistic Regression)

- 시그모이드(Sigmoid) 함수

  어떠한 값이 입력되더라도 0 ~ 1 사이의 값 만을 출력하는 함수



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

로지스틱 회귀(Logistic Regression)

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

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

로지스틱 회귀(Logistic Regression)

- 로지스틱 회귀 모델 fitting

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

```
clf = LogisticRegression()
```

```
from sklearn.model_selection import GridSearchCV

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

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

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

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

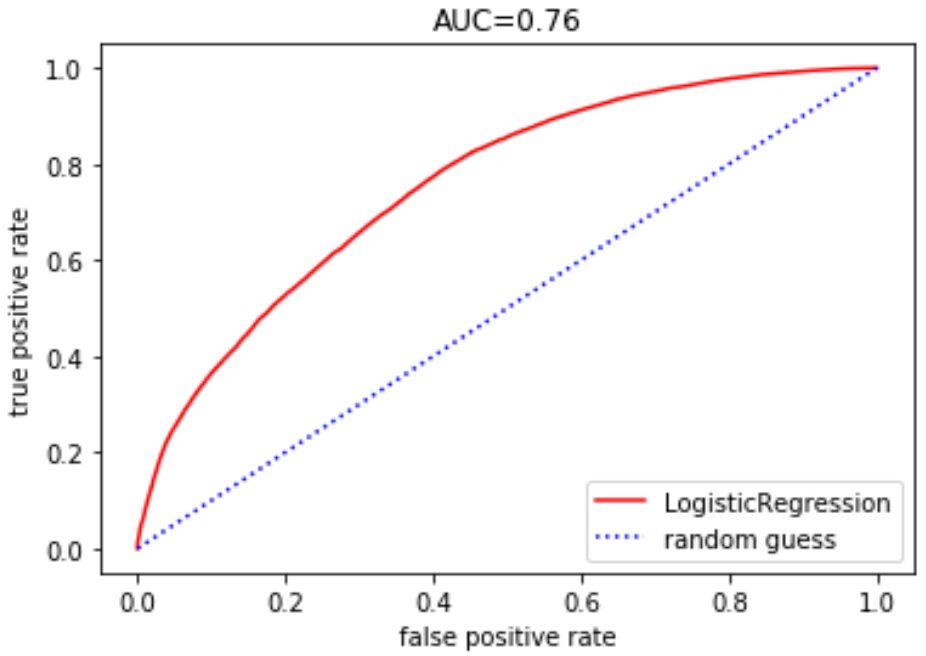
로지스틱 회귀(Logistic Regression)

- 모델 성능 평가

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

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

```
import matplotlib.pyplot as plt

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



K-최근접 이웃(Nearest Neighbors)

- KNN(K Nearest Neighbors) : 최근접 이웃 알고리즘

  새로운 데이터의 분류를 알기 위해 사용

  분류나 회귀에 사용할 수 있는 알고리즘으로 단순해 보이지만 강력하고 유용한 기법

  훈련 단계에서 학습을 하지 않기 때문에 '게으른 학습'이라 부름

  테스트/검증 단계에서 테스트 관측값과 가장 근접한 훈련 관측값을 비교

  거리에만 의존하므로 차원의 저주에 따라 예측에 필요한 특징의 개수가 늘어나면 성능이 크게 저하

K값에 의해 결정된 분류를 새로운 데이터의 분류로 확정

K-최근접 이웃(Nearest Neighbors)

- KNN(K Nearest Neighbors) : 최근접 이웃 알고리즘

  유클리드 공간의 점과 점 사이의 직선거리를 사용

  2차원 공간의 유클리드 거리의 계산방법 예)

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

K개의 가장 가까운 훈련 인스턴스를 골라 가장 많은 레이블을 분류로 선택

특징의 표준화된 스케일링 필요

K-최근접 이웃(Nearest Neighbors)

- 데이터 로드

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
%matplotlib inline
```

```
breast_cancer = pd.read_csv("data/breast-cancer-wisconsin.data", header=None)
```

```
breast_cancer.head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

K-최근접 이웃(Nearest Neighbors)

- 데이터 셋 컬럼명 지정

```
breast_cancer.columns = ["id_number", "clump_thickness", "unif_cell_size", "unif_cell_shape",
                         "marg_adhesion", "single_epith_cell_size","bare_nuclei", "bland_chromatin",
                         "normal_nucleoli", "mitoses", "class"]
```

```
breast_cancer.head()
```

	id_number	clump_thickness	unif_cell_size	unif_cell_shape	marg_adhesion	single_epith_cell_size	bare_nuclei	bland_chromatin	normal_nucleoli	mitoses	class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

K-최근접 이웃(Nearest Neighbors)

- 이상 데이터 탐색

```
breast_cancer[breast_cancer.bare_nuclei=='?']
```

	id_number	clump_thickness	unif_cell_size	unif_cell_shape	marg_adhesion	single_epith_cell_size	bare_nuclei	bland_chromatin	normal_nucleoli	mitoses	class
23	1057013	8	4	5	1	2	?	7	3	1	4
40	1096800	6	6	6	9	6	?	7	8	1	2
139	1183246	1	1	1	1	1	?	2	1	1	2
145	1184840	1	1	3	1	2	?	2	1	1	2
158	1193683	1	1	2	1	3	?	1	1	1	2
164	1197510	5	1	1	1	2	?	3	1	1	2
235	1241232	3	1	4	1	2	?	3	1	1	2
249	169356	3	1	1	1	2	?	3	1	1	2
275	432809	3	1	3	1	2	?	2	1	1	2
292	563649	8	8	8	1	2	?	6	10	1	4
294	606140	1	1	1	1	2	?	2	1	1	2
297	61634	5	4	3	1	2	?	2	3	1	2
315	704168	4	6	5	6	7	?	4	9	1	2
321	733639	3	1	1	1	2	?	3	1	1	2
411	1238464	1	1	1	1	1	?	2	1	1	2
617	1057067	1	1	1	1	1	?	1	1	1	2

K-최근접 이웃(Nearest Neighbors)

- 이상 데이터 NaN 처리 및 Data Imputation

```
breast_cancer.bare_nuclei = breast_cancer.bare_nuclei.replace("?", np.NaN)
```

```
breast_cancer.bare_nuclei.value_counts()
```

```
1     402
10    132
2      30
5      30
3      28
8      21
4      19
9       9
7       8
6       4
Name: bare_nuclei, dtype: int64
```

```
breast_cancer.bare_nuclei.mode()[0]
```

```
'1'
```

```
breast_cancer.bare_nuclei = breast_cancer.bare_nuclei.fillna(breast_cancer.bare_nuclei.mode()[0])
```

K-최근접 이웃(Nearest Neighbors)

- 이상 데이터 NaN 처리 및 Data Imputation

```
breast_cancer["class"].value_counts()
```

```
2    458
4    241
Name: class, dtype: int64
```

```
breast_cancer["cancer_ind"] = 0
breast_cancer.loc[breast_cancer["class"] == 4, "cancer_ind"] = 1
```

```
breast_cancer.cancer_ind.value_counts()
```

```
0    458
1    241
Name: cancer_ind, dtype: int64
```

```
X = breast_cancer.drop(["id_number", "class", "cancer_ind"], axis=1)
y = breast_cancer.cancer_ind
```

K-최근접 이웃(Nearest Neighbors)

- 데이터 분할, scaling 및 KNN 모델 fitting

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train)
```

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

K-최근접 이웃(Nearest Neighbors)

- 모델 성능 평가

```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score,
```

```
y_pred = knn.predict(X_test_scaled)
print(accuracy_score(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print(roc_auc_score(y_test, y_pred))
```

```
0.9761904761904762
[[141   2]
 [  3  64]]
0.9706189333055005
```

나이브 베이즈(Naive Bayes)

- 나이브 베이즈 분류기(Naive Bayes Classifier)
  - 확률 기반 분류기(Classifier)
  - 데이터가 각 클래스에 속할 예측 특징 확률을 계산
  - 나이브(Naive) : 예측한 특징이 상호 독립적이라는 가정 하에 확률 계산을 단순화
  - 베이즈(Bayes) : 입력 특징이 클래스 전체의 확률 분포 대비 특정 클래스에 속할 확률을 베이즈 정리를 기반으로 계산

나이브 베이즈(Naive Bayes)

- 베이즈 정리와 나이브 베이즈
  - 베이즈 정리에서 $P(A|B)$ 는 사건 B가 주어졌을 때 사건 A가 일어날 확률

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

  - 나이브 베이즈의 목표는 n개의 특징을 가진 샘플 데이터 x가 주어졌을 때 k개의 클래스 중 하나에 속할 확률을 결정
  - 샘플 데이터는 $x_1, x_2, x_3, \cdots, x_n$ 의 값을 가진 특징으로 구성되고, y_k는 샘플 데이터가 k에 속하는 사건을 나타냄

$$x = (x_1, x_2, x_3, \cdots, x_n)$$

$$P(y_k|(x_1, x_2, x_3, \cdots, x_n) = P(y_k|x) = \frac{P(x|y_k)P(y_k)}{P(x)}$$

나이브 베이즈(Naive Bayes)

- 나이브 베이즈 분류 절차

① 사후확률(Posterior Probability) : B가 발생했을때, A일 확률

② 사전확률(Prior Probability) : A일 확률

③ 우도(Likelihood) : B가 이전A에서 사용되었을 확률

④ 주변우도(Marginal Likelihood) : 모든 곳에서 B가 나타날 확률

나이브 베이즈(Naive Bayes)

- 나이브 베이즈 분류 절차

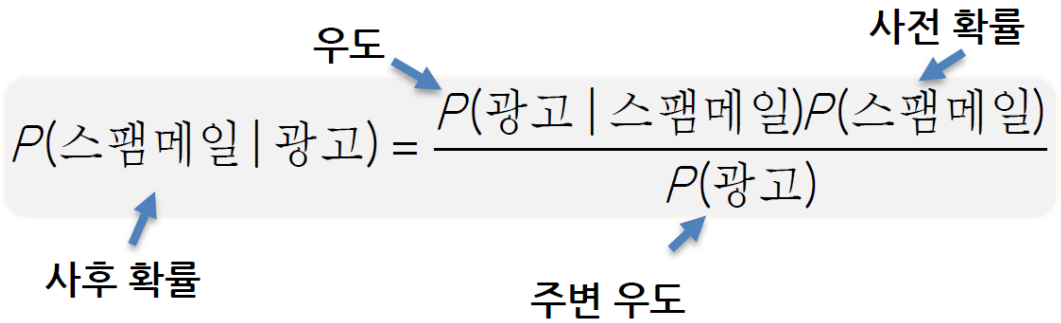
'A : 스팸메일, B : 광고'

① 사후확률(Posterior Probability) : '광고'가 스팸메일일 확률

② 사전확률(Prior Probability) : 이전 메일이 스팸메일일 확률

③ 우도(Likelihood) : '광고'가 이전 스팸메일에서 사용되었을 확률

④ 주변우도(Marginal Likelihood) : 모든 곳에서 '광고'가 나타날 확률



나이브 베이즈(Naive Bayes)

- 나이브 베이즈 분류 절차

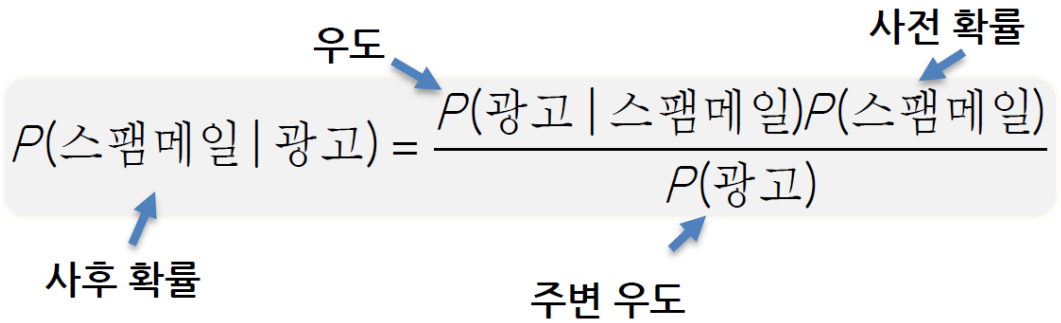
'A : 스팸메일, B : 광고'

① 사후확률(Posterior Probability) : '광고'가 스팸메일일 확률

② 사전확률(Prior Probability) : 이전 메일이 스팸메일일 확률

③ 우도(Likelihood) : '광고'가 이전 스팸메일에서 사용되었을 확률

④ 주변우도(Marginal Likelihood) : 모든 곳에서 '광고'가 나타날 확률



나이브 베이즈(Naive Bayes)

- 라플라스 스무딩(Laplace Smoothing)

  특징의 출현 횟수 초기값을 1부터 시작해 0을 곱해 발생하는 문제를 해결

  (발견되지 않은 특징의 출현 빈도 초기값을 1로 설정)

- 언더플로우(underflow) 방지

  $P(B|A)$가 너무 작아지면서 0 에 거의 가까워질 수 있음

  이러한 문제를 해결하기 위해 확률에 Log 를 취하여 언더플로우를 방지함

$$Log(P(A|B)) = Log(P(B|A)P(A))$$

나이브 베이즈(Naive Bayes)

- Twenty Newsgroups 데이터 셋
  - 20개의 다른 주제를 가진 18,846개의 뉴스그룹 데이터
  - 훈련 데이터(11,314개), 테스트 데이터(7,532개)
  - Feature Matrix : 문서 텍스트
  - Target Vector : 문서가 속한 뉴스 그룹(20 개 그룹)

['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'talk.religion.misc']

나이브 베이즈(Naive Bayes)

- 데이터 로드

```
from sklearn.datasets import fetch_20newsgroups
newsdata = fetch_20newsgroups(subset='train')
newsdata_test = fetch_20newsgroups(subset='test', shuffle=True)
```

```
newsdata.data[:2]
```

```
["From: lerxst@wam.umd.edu (where's my thing)\nSubject: WHAT car
yland, College Park\nLines: 15\n\n I was wondering if anyone out
ts car, looked to be from the late 60s/\nearly 70s. It was calle
parate from the rest of the body. This is \nall I know. If anyon
made, history, or whatever info you\nhave on this funky looking
d Lerxst ----\n\n\n\n\n",
 "From: guykuo@carson.u.washington.edu (Guy Kuo)\nSubject: SI Cl
celeration,clock,upgrade\nArticle-I.D.: shelley.1qvfo9INNc3s\nOr
hington.edu\n\nA fair number of brave souls who upgraded their s
rief message detailing\nyour experiences with the procedure. Top
f usage per day, floppy disk\nfunctionality with 800 and 1.4 m t
so please add to the network\nknowledge base if you have done th
shington.edu>\n"]
```

```
print(newsdata.target_names)
```

```
['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'com
e', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.s
ion.christian', 'talk.politics.guns', 'talk.politics.mideast',
```

나이브 베이즈(Naive Bayes)

- 문자열 Vectorize

```
from sklearn.feature_extraction.text import  TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
```

```
vectorizer = TfidfVectorizer()
# vectorizer = TfidfVectorizer(stop_words="english",
#                              token_pattern=r"\b[a-z0-9_\-\.]+[a-z][a-z0-9_\-\.]+\b")
```

```
X_train = vectorizer.fit_transform(X_train)
print(X_train.shape)
```

```
(11314, 130107)
```

```
X_test = vectorizer.transform(X_test)
print(X_test.shape)
```

```
(7532, 130107)
```

나이브 베이즈(Naive Bayes)

- 나이브 베이즈 모델 fitting 및 성능 평가

```
model = MultinomialNB()
model.fit(X_train, y_train)
```

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.7738980350504514
```

서포트 벡터 머신(SVM)

- Support Vector Machine(SVM) 개요

– 주어진 데이터 집합을 바탕으로 하여 새로운 데이터가 어느 카테고리에 포함될지 판단하는 비확률적 이진 선형 분류 모델 생성

– 생성된 분류 모델은 데이터가 사상된 공간에서 경계(Hyperplane)로 표현

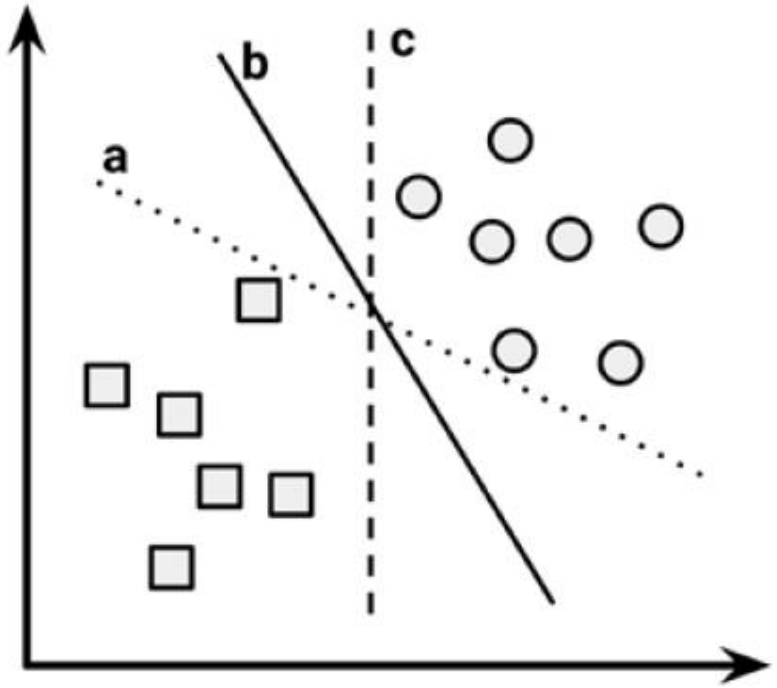
– SVM 알고리즘은 그 중 가장 큰 폭을 가진 경계를 찾는 알고리즘

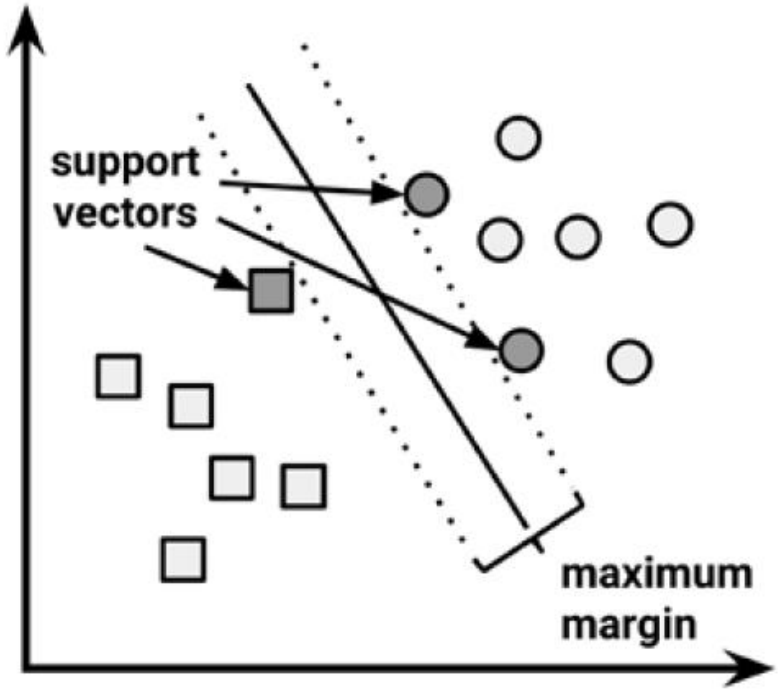
– SVM은 선형 분류, 비선형 분류 및 회귀에서도 사용

– 비선형 분류를 하기 위해서 주어진 데이터를 고차원 특징 공간으로 사상(커널 트릭)하는 작업이 필요한데, 이를 효율적으로 하기 위해 커널 함수를 사용

서포트 벡터 머신(SVM)

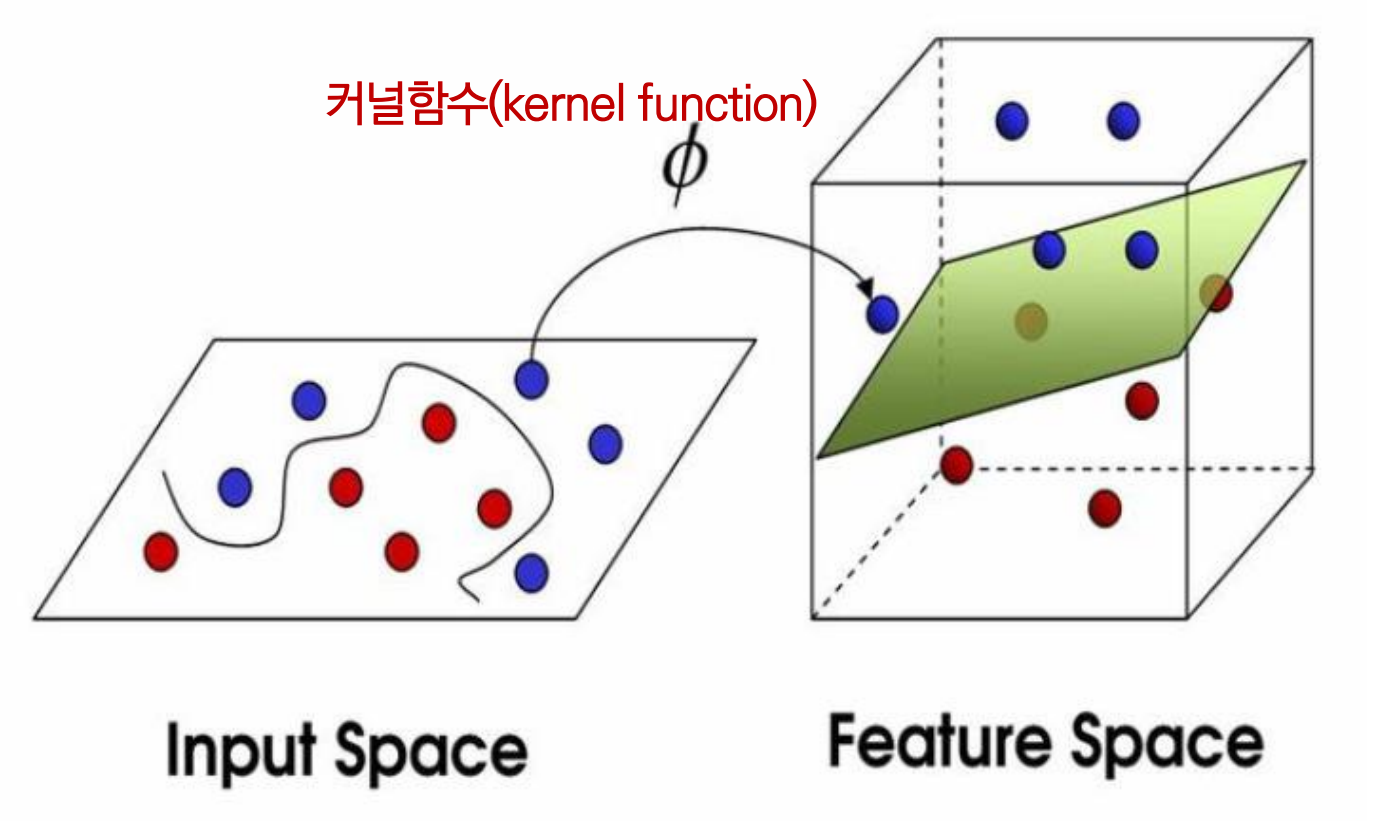
- 선형 분류





서포트 벡터 머신(SVM)

- 비선형 분류



서포트 벡터 머신(SVM)

- SVM의 특징
  - 노이즈 데이터에 크게 영향 받지 않고, overfitting이 잘 발생하지 않음
  - 높은 정확도, 사용하기 용이
  - 커널과 모델에서 매개변수의 여러가지 조합테스트 필요
  - example과 feature 수가 많으면 훈련 속도 저하
  - 해석이 불가능한 것은 아니지만, 블랙박스 솔루션

서포트 벡터 머신(SVM)

- 데이터 로드 및 Scaling

```
import pandas as pd
```

```
X_train, y_train, X_test, y_test = pd.read_pickle('data/cancer.pkl')
```

```
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(489, 9) (489,) (210, 9) (210,)
```

```
X = pd.concat([X_train, X_test])
y = pd.concat([y_train, y_test])
```

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

```
(699, 9) (699,)
```

```
from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(X)
```

서포트 벡터 머신(SVM)

- Cross Validation & SVM 모델 fitting

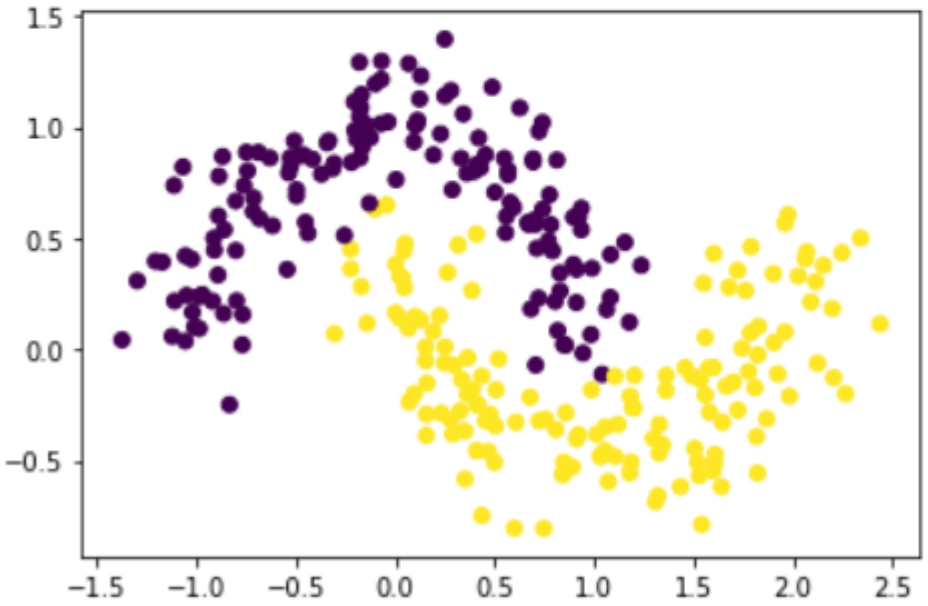
```
def svm_linear(X, y):
    svm_linear =svm.SVC(kernel = 'linear')
    result = pd.DataFrame(cross_validate(svm_linear, X, y, cv = 5))
    print(result, '\n')
    print('평균: ', result.test_score.mean(), '\n')
```

```
def svm_rbf(X, y):
#    svm_rbf =svm.SVC(kernel = 'rbf', gamma='auto')
    svm_rbf =svm.SVC(kernel = 'rbf', C=50, gamma=10)
    result = pd.DataFrame(cross_validate(svm_rbf, X, y, cv = 5))
    print(result, '\n')
    print('평균: ', result.test_score.mean(), '\n')
```

서포트 벡터 머신(SVM)

- 비선형 데이터 생성

```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
X2, y2 = make_moons(n_samples = 300, noise = 0.16, random_state = 42)

plt.scatter(X2[:,0],X2[:,1],c=y)
plt.show()
```



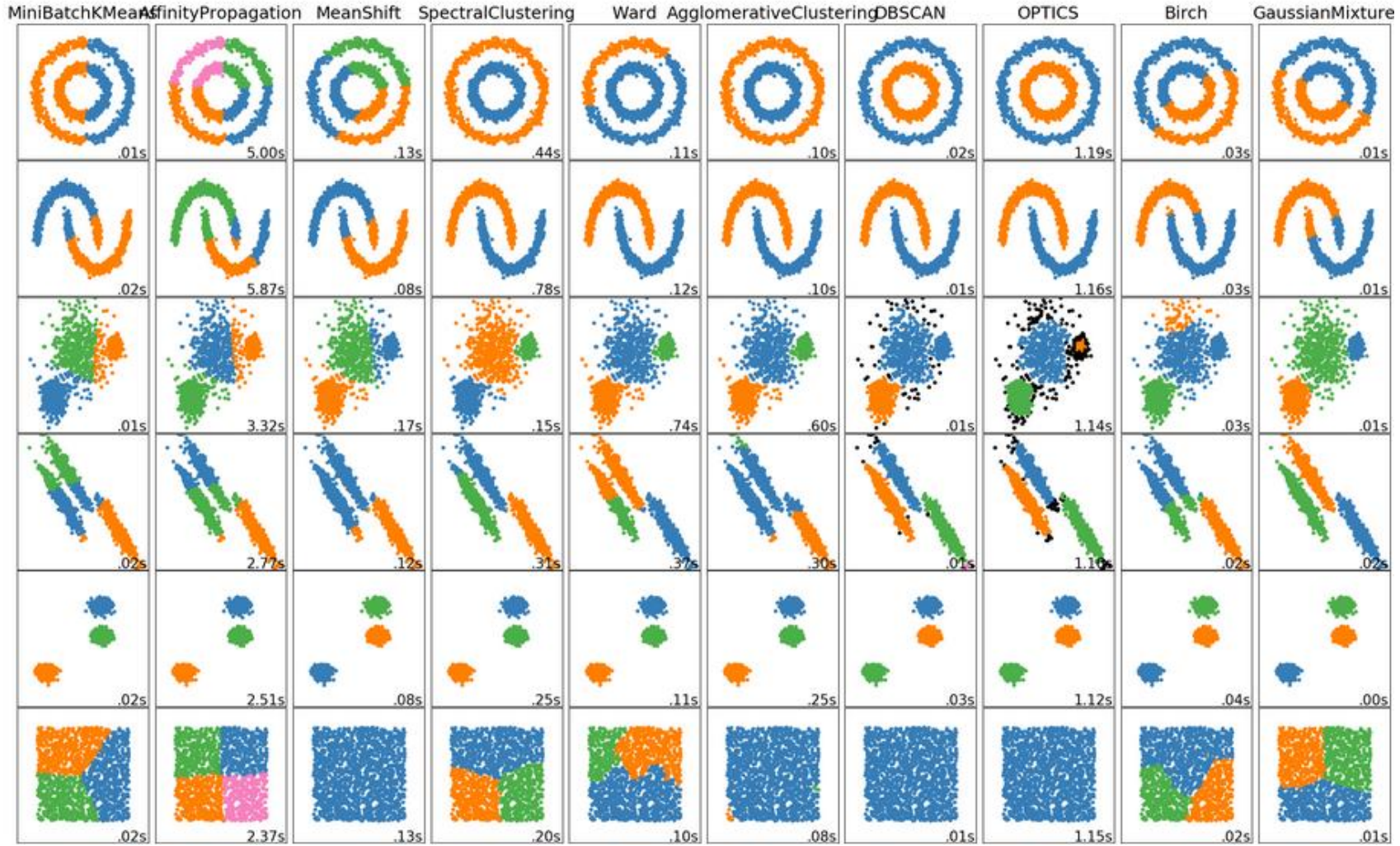
서포트 벡터 머신(SVM)

- Grid Search 를 이용한 하이퍼 파라미터 최적화

```
from sklearn.model_selection import GridSearchCV

svm_rbf = svm.SVC(kernel = 'rbf',random_state=100)
params = {'C': [0.001, 0.01, 0.1, 1, 10, 25, 50, 100],
             'gamma':[0.001, 0.01, 0.1, 1, 10, 25, 50, 100]}

grid_svm = GridSearchCV(svm_rbf, param_grid = params, cv = 5)
grid_svm.fit(X, y)

result = pd.DataFrame(grid_svm.cv_results_['params'])
result['mean_test_score'] = grid_svm.cv_results_['mean_test_score']
result.sort_values(by='mean_test_score', ascending=False)
```

군집화(Clustering)

- 군집화(Clustering)



출처 : https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html#sphx-glr-auto-examples-cluster-plot-cluster-comparison-py

K평균(K-Means)

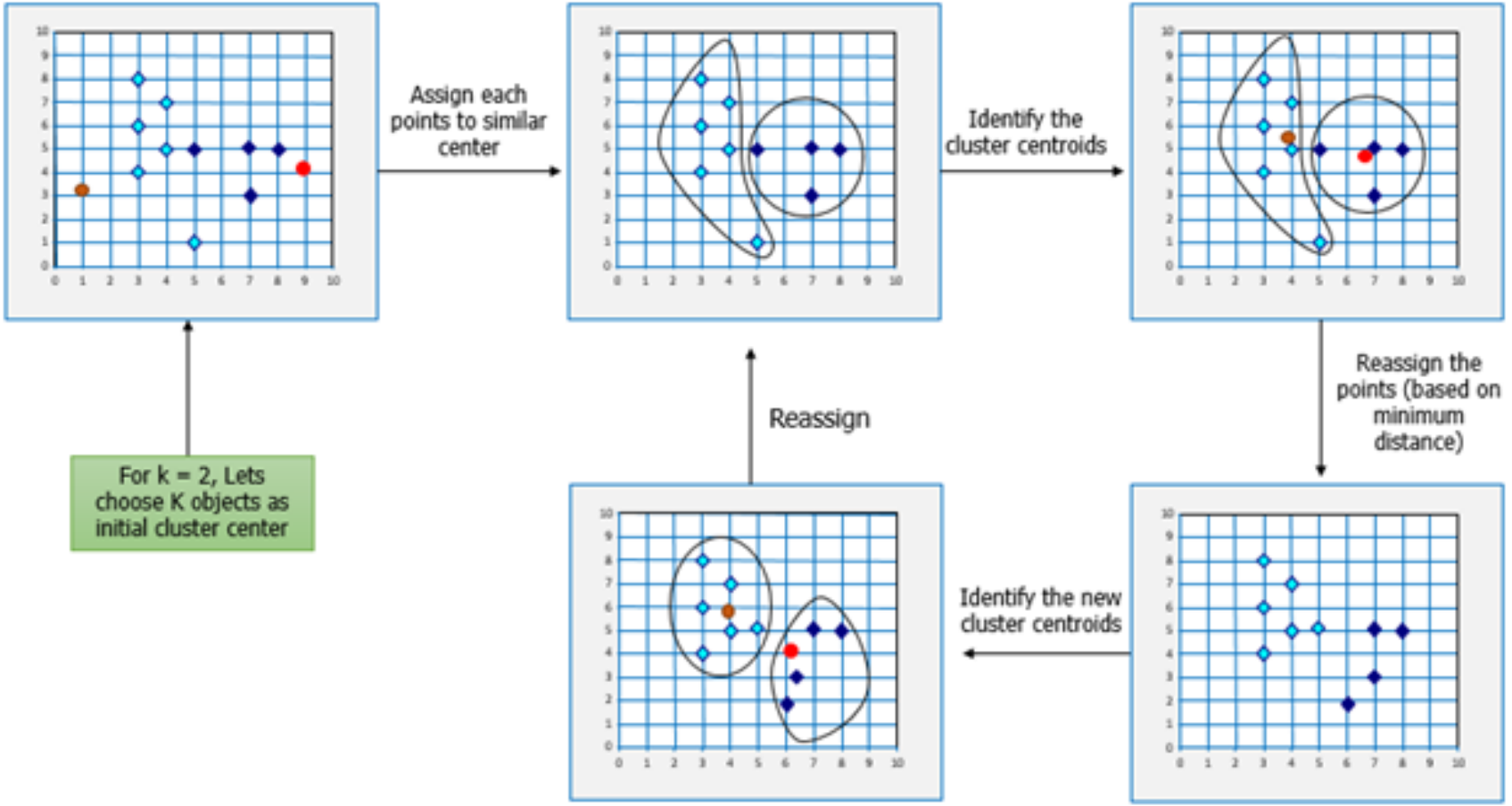
- K-평균(K-Means)

  K-Means 알고리즘은 간단하면서 매우 효과적인 통계적 클러스터링 기법

  - 1 단계 : K값, 만들어질 클러스터의 개수 정함.
  - 2 단계 : 데이터 집합에서 무작위로 k개의 인스턴스 골라 클러스터의 초기값으로 정함
  - 3 단계 : 나머지 인스턴스들은 Euclidean distance를 사용하여 가장 가까운 센터를 가진 클러스터에 배정
  - 4 단계 : 각 클러스터마다 그 안에 배정된 모든 인스턴스의 평균 값을 구하여 그것을 그 클러스터에 대한 새로운 센터로 지정
  - 5 단계 : 새로운 센터값들이 이전 센터 값들과 같다면 알고르즘은 종료되고 그렇지 않으면, 그 새로운 센터 값으로 단계 3-5를 반복함.

K평균(K-Means)

- K-평균(K-Means)



38