KNN & SVM 알고리즘

4조 이한울 임현진 정기윤 황정민

목차

1

머신러닝 알고리즘 개요

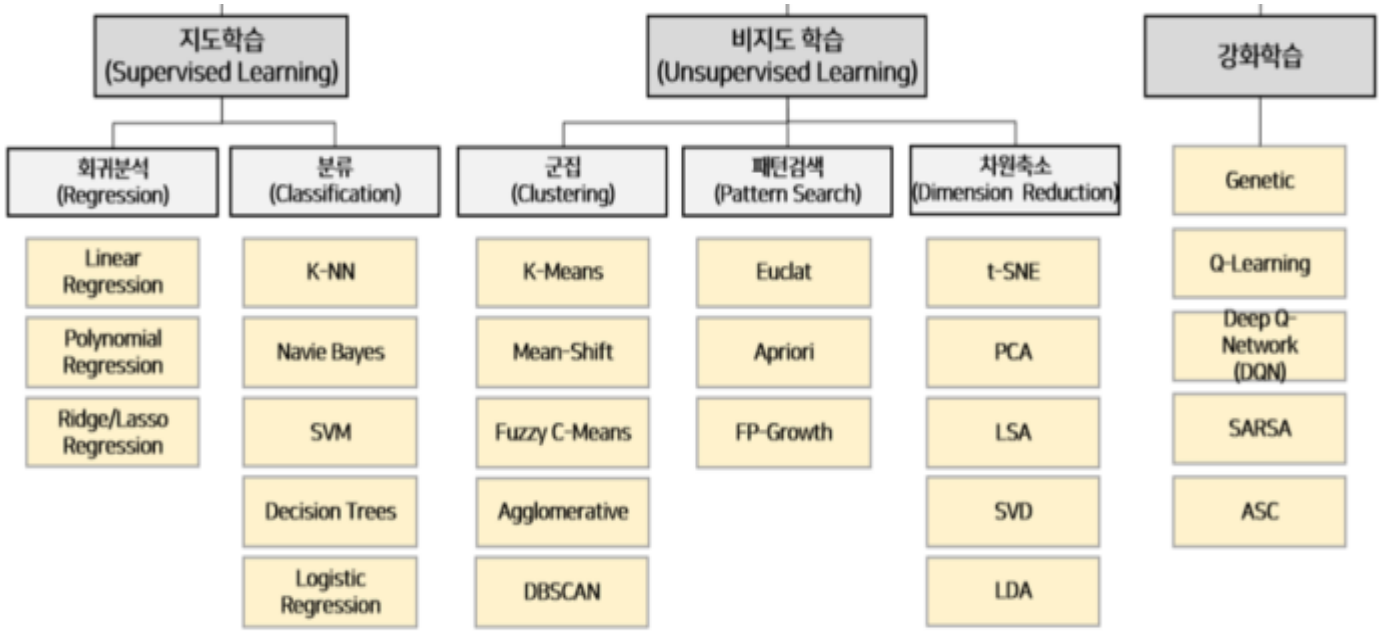
머신러닝 알고리즘 개요

머신러닝 이란?

머신러닝이란?

데이터를 수집한 후, 특정 알고리즘을 기반으로 추론을 하는 프로그램

머신러닝은 지도학습, 비지도학습, 강화학습으로 분류



머신러닝 알고리즘 개요

지도학습과 비지도학습

지도학습

정답이 있는 데이터를 활용해 학습을 진행

Ex)선형회귀, 다항회귀, KNN, SVM, 나이브베이즈

비지도학습

정답이 없는 데이터를 활용해 비슷한 특징끼리 군집화

Ex) K-means, PCA, Eclat

머신러닝 알고리즘 개요

회귀와 분류

회귀

연속적인 값을 예측

Ex) 선형 회귀, 다항회귀

분류

분리된 값을 예측

데이터가 어떤 범주에 속하는 지 예측

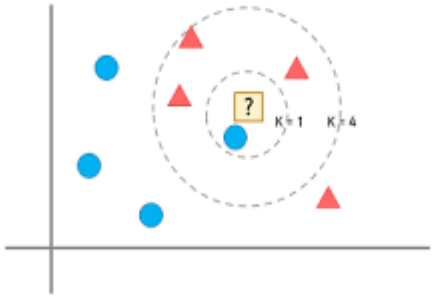
Ex) KNN, 로지스틱회귀, SVM

머신러닝 알고리즘 개요

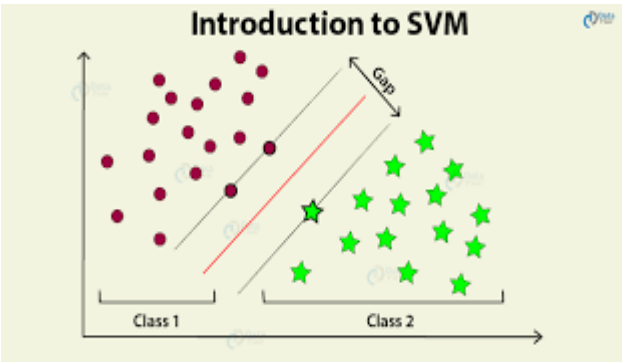
알고리즘의 선정

알고리즘의 선정

1. 데이터 분석의 목적
2. 데이터의 특성
3. 알고리즘의 성능
4. 알고리즘의 학습시간 및 예측시간 등



KNN알고리즘



SVM알고리즘

다양한 알고리즘을 사용해보면서 가장 적합한 것을 선정

이번 발표에서는 KNN과 SVM 알고리즘을 학습

2

KNN 알고리즘

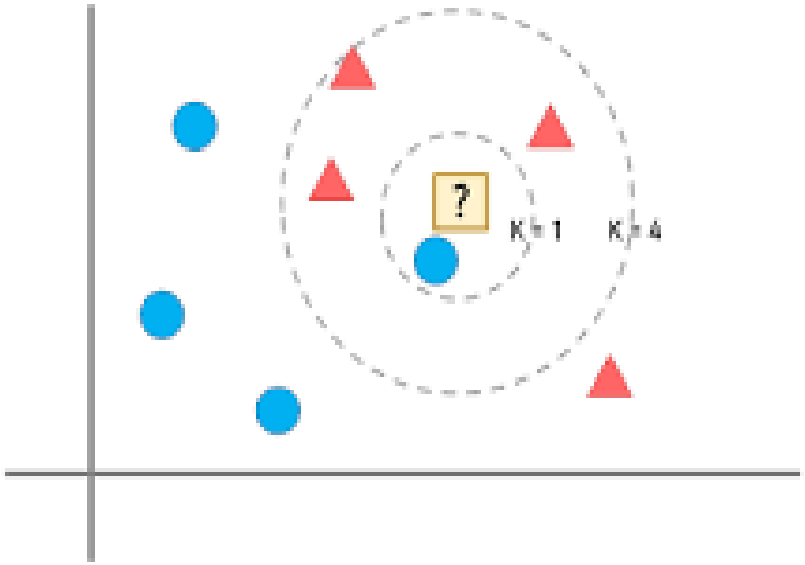
KNN 알고리즘의 이해

KNN 알고리즘이란?

KNN알고리즘이란?

현재 데이터로부터 가까운 k개의 데이터를 찾아 k개의 레이블 중 가장 많이 분류된 것으로 데이터를 분류하는 알고리즘

K=1이면 파랑색으로, k=4이면 빨간색으로 분류



KNN 알고리즘의 이해

KNN은 게으른 학습(lazy learner)

- 게으른 학습(lazy learning) :

학습 데이터(training data)를 단순하게 저장하고 테스트 데이터(test data)가 올 때까지 기다렸다가 데이터가 input되면 그 때 모델을 실행하는 방법
학습 시간보다 예측 시간이 더 걸림.
모수를 가정하지 않는 비모수 학습

- 열정적인 학습(eager learning) :

학습 데이터가 주어지면 테스트 데이터가 input되기 전부터 바로 모델을 생성하는 방법.

KNN 알고리즘의 이해

유클리드거리

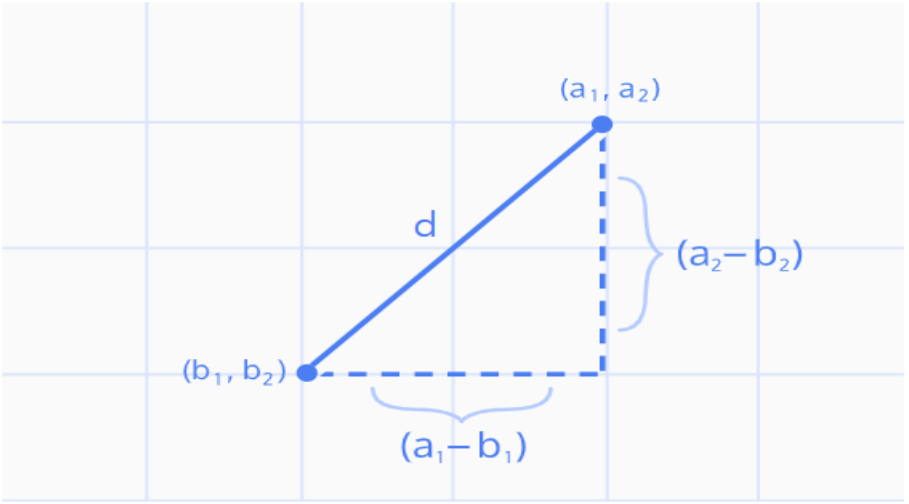
유클리드거리

KNN 알고리즘을 사용할 때 거리를 측정하는 방법으로
는 유클리드거리와 맨해튼 거리가 주로 사용된다.

$$a = \langle a_1 a_2 \cdots a_n \rangle$$

$$b = \langle b, b_2 \cdots b_n \rangle$$

$$\|a - b\| = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$



KNN 알고리즘의 이해

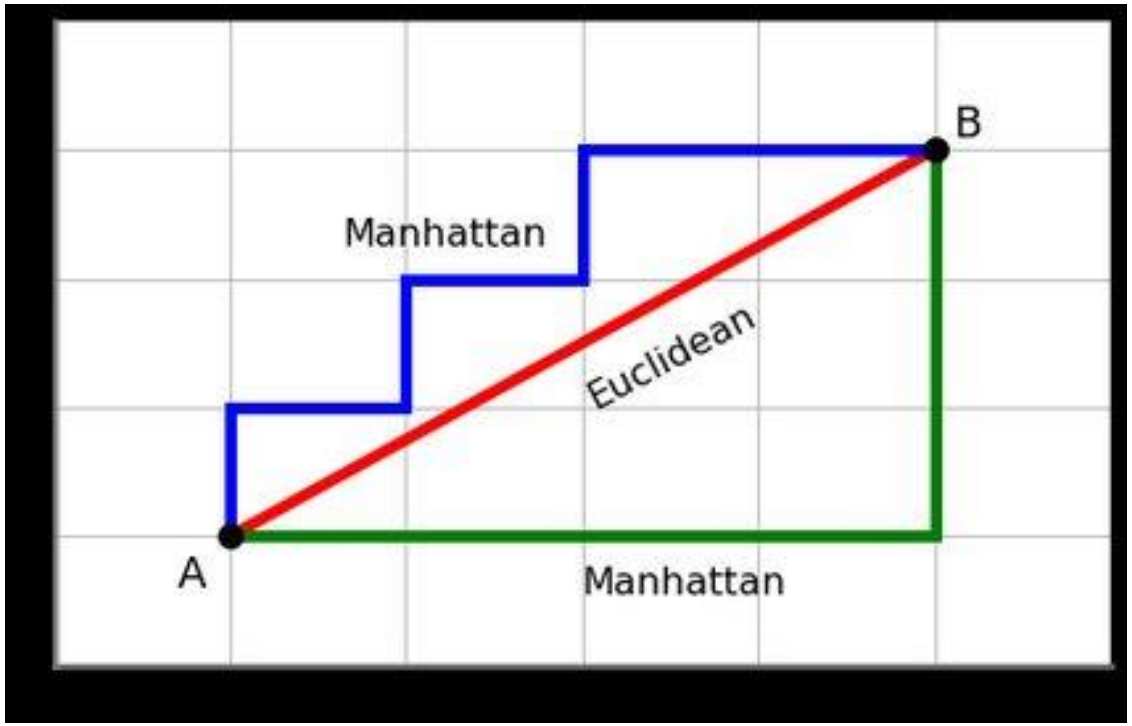
맨해튼거리

$a = \langle a_1 a_2 \cdots a_n \rangle$

$b = \langle b, b_2 \cdots b_n \rangle$

$d(a, b)$

$= |a_1 - b_1| + |a_2 - b_2| + \cdots + |a_n - b_n|$



KNN 알고리즘의 이해

해밍거리

문자열에서 같은 위치에 서로 다른 부분들이 몇 개인지

Ex)"0108921" 과 "0708179" 사이의 거리는 4

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

마할라노비스 거리

데이터가 평균에서 표준편차의 몇 배 만큼 떨어져 있는지

Ex) 평균이 5이고, 표준편차가 1인 경우,

7의 마할로노비스 거리는 2

6의 마할로노비스 거리는 1

KNN 알고리즘의 이해

KNN 알고리즘의 장점과 단점

장점

1. 이해하기가 쉽고, 직관적이다.
2. 수치 기반 데이터 분류 작업에서 성능이 우수하다.
3. 별도의 모델학습이 필요 없다.

단점

1. 예측속도가 느리다.
2. 최적의 K를 선택하기 쉽지않다.
3. 예측값이 지역정보에 많이 편향될 수 있다.

K의 선택과 정규화

K의 선택

K가 너무 작으면: Overfitting(과적합)

특정데이터에 영향을 지나치게 많이 받게 된다

구분 경계가 명확하지 않다.

K가 너무 크면: underfitting(과소적합)

미세한 경계부분의 분류 정확도가 떨어지게 된다.

이진 분류라면

동률이 나오는 상황을 방지하기 위해

홀수를 선택하는 것이 좋다.

K겹 교차검증 등의 방법을 사용하여 최적의 K를 찾는다

K의 선택과 정규화

K겹 교차검증

K겹 교차검증

데이터를 K개의 세트로 분할하고, 그 중 하나를
테스트 세트로 사용하는 과정을 반복적으로 실행하는 방법
(이 때, 테스트세트로 사용되는 데이터는 중복되면 안 됨)

K의 선택과 정규화

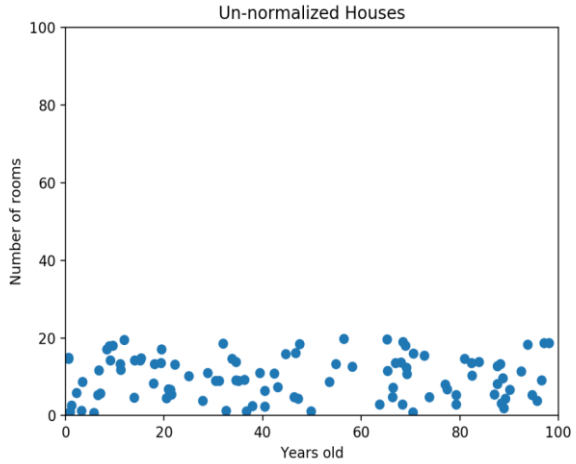
정규화(Normalization)

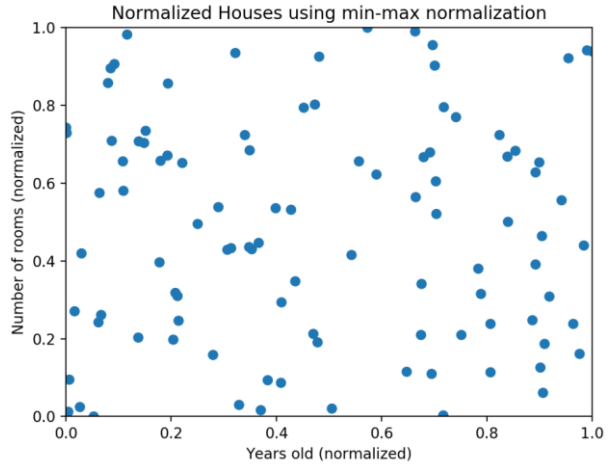
정규화(Normalization)

데이터의 스케일이 차이가 많이 나면 분류의 정확도가 떨어지게 된다.

정규화를 통해 모든 데이터가 동일한 정도로 반영되도록 해야 한다.

Ex) MinMax Scaler, MaxAbs Scaler, Robust Scaler, Standard Scaler





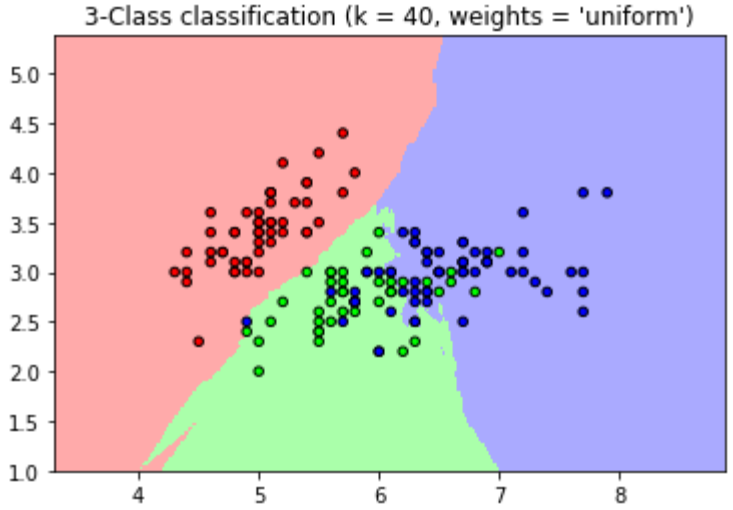
KNN 가중치

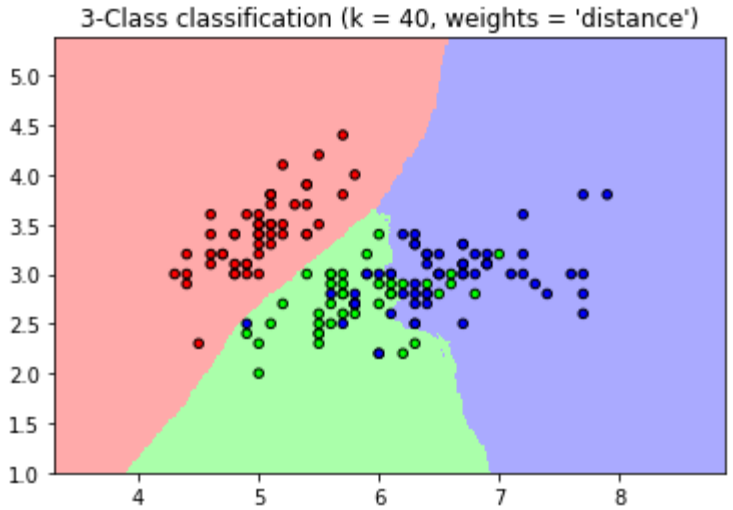
가중치

weights = ‘distance’

K개의 이웃하고 있는 점 중 가까운 점의 반영정도를 키우고
먼 점은 줄이는 방식으로 작동한다.

거리를 기준으로 가중치를 준다면
결정 경계의 구분이 명확해지고 일반화되는 장점이 있다.
하지만 예측 정확도에 부정적인 영향을 끼칠 수도 있으므로
분석가의 정확한 판단이 중요하다.





KNN 실습

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import neighbors
```

iris 데이터셋 불러오기

```
iris = sns.load_dataset('iris')
```

MinMaxScaler로 정규화하기

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

```
features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
```

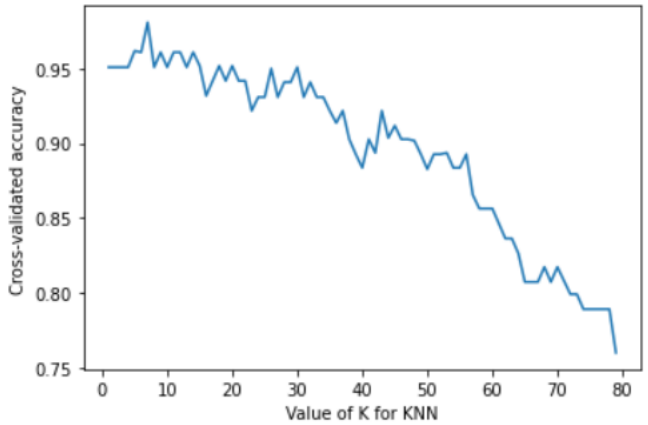
```
iris[features] = scaler.fit_transform(iris[features])
```

데이터를 훈련데이터와 테스트데이터로 나누기

```
from sklearn.model_selection import train_test_split

df_train, df_test = train_test_split(iris, test_size=0.3, random_state=1)
```

최적 k 찾기

```
from sklearn.model_selection import cross_val_score

k_range = np.arange(1,80)
k_scores = []

for k in k_range:
    knn=neighbors.KNeighborsClassifier(k)
    scores= cross_val_score(knn, df_train[features]
                            , df_train['species'], cv=10
                            , scoring="accuracy")
    k_scores.append(scores.mean())

plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-validated accuracy')
plt.show()
```



KNN 실습

```
result = pd.DataFrame()
result['k'] = k_range
result['accuracy'] = k_scores
result = result.sort_values(by='accuracy', ascending=False).reset_index(drop=True)
result.head()
```

	k	accuracy
0	7	0.980909
1	5	0.961818
2	9	0.960909
3	11	0.960909
4	15	0.951818

위 과정에서 얻은 최적 k로 테스트 데이터의 label을 예측하기

```
classifier = neighbors.KNeighborsClassifier(result['k'][0])
classifier.fit(df_train[features], df_train['species'])
pred = classifier.predict(df_test[features])
```

분류결과 및 accuracy 확인하기

```
from sklearn.metrics import confusion_matrix, accuracy_score
print(confusion_matrix(df_test['species'], pred))
print(accuracy_score(df_test['species'], pred))
```

```
[[14  0  0]
 [ 0 17  1]
 [ 0  1 12]]
0.9555555555555556
```

차원의 저주

차원의 저주

변수의 수(차원) 이 증가 함에 따라 모델의 성능이 안 좋아지는 현상

차원이 데이터의 개수 보다 클 때 발생

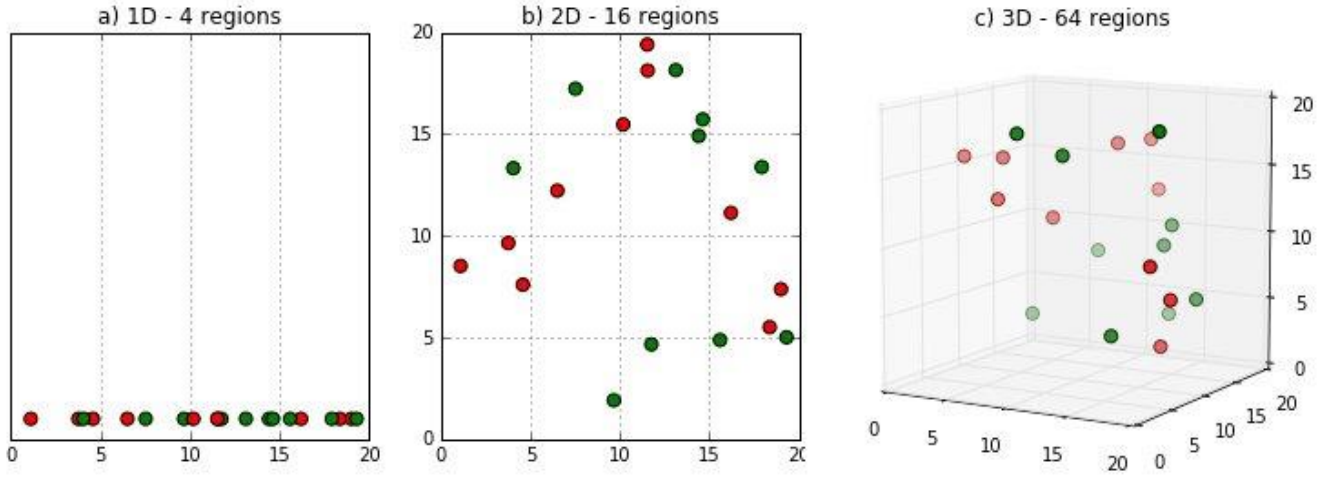
Ex) 데이터가 100개인데, 차원이 500개

차원의 저주

차원의 저주가 발생하는 이유

데이터의 차원이 증가할수록 데이터의 밀도가 낮아짐에 따라 발생

KNN에서는 차원의 저주가 발생하면
각각의 데이터의 사이의 거리가 멀어지면서
서로 비슷해져 분류하기 어려워 짐



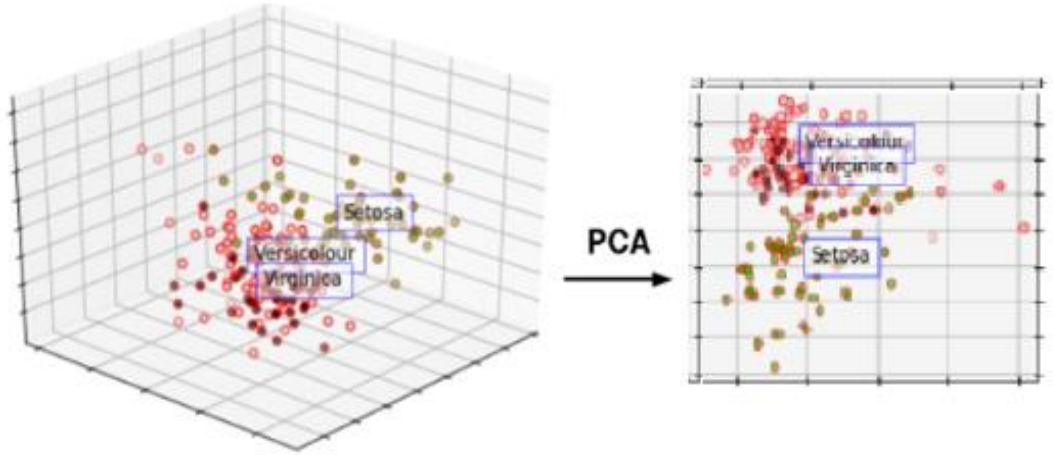
차원의 저주

해결방법

해결방법

1. 데이터의 수를 증가 시킨다.

2. 차원축소를 진행한다.



KNN 알고리즘의 활용

KNN 알고리즘의 활용

KNN 알고리즘은 현업에서 잘 사용되지 않는다

특성이나 샘플의 수가 클 경우 예측이 느려짐.

수백 개 이상의 많은 특성을 가진 데이터셋에는 잘 동작하지 않으며,
특성 값 대부분이 0인 (즉 희소한) 데이터셋과는 특히 잘 작동하지 않음

그렇지만 이해하기 쉽고, 다른 알고리즘들과 연관되어 있어
복잡한 알고리즘을 배우기에 좋은 시작점이다.

3

SVM 알고리즘

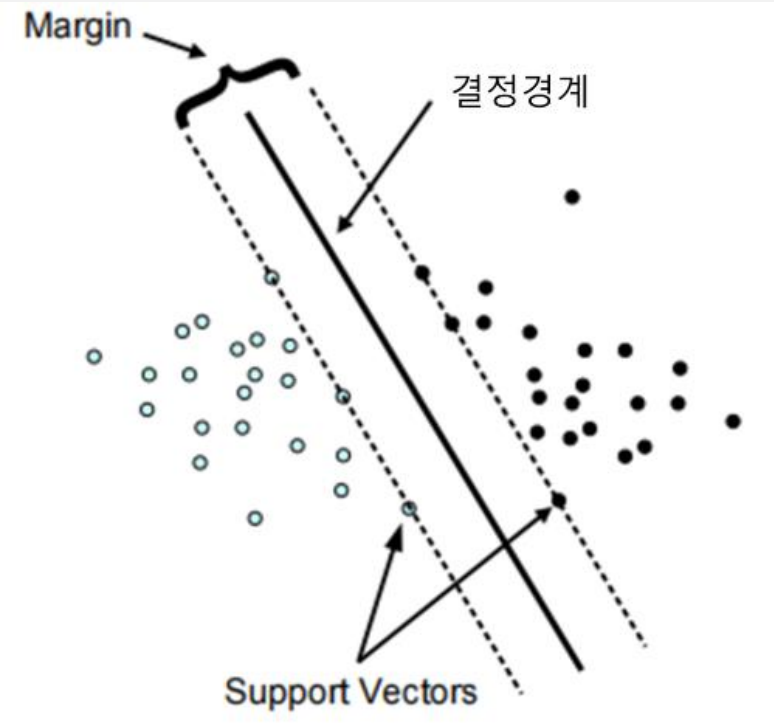
SVM 알고리즘

SVM 알고리즘이란?

SVM : support vector machine

주어진 데이터를 구분 짓는 결정 경계를 찾아

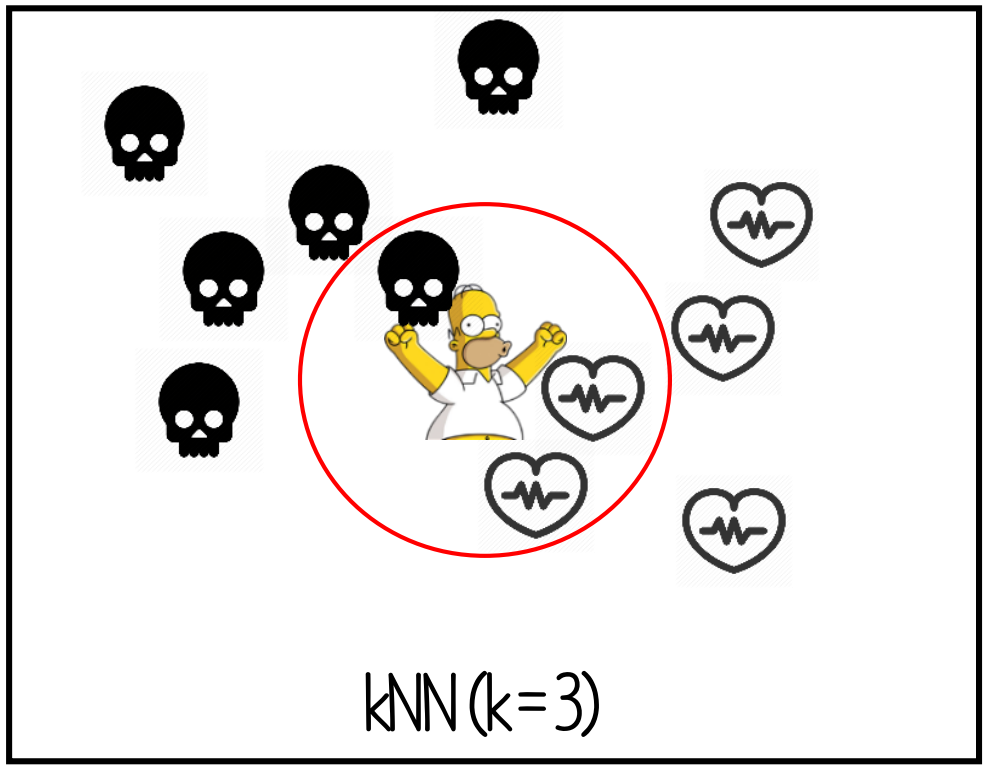
새로운 데이터를 분류 & 회귀 분석하는 지도학습 알고리즘

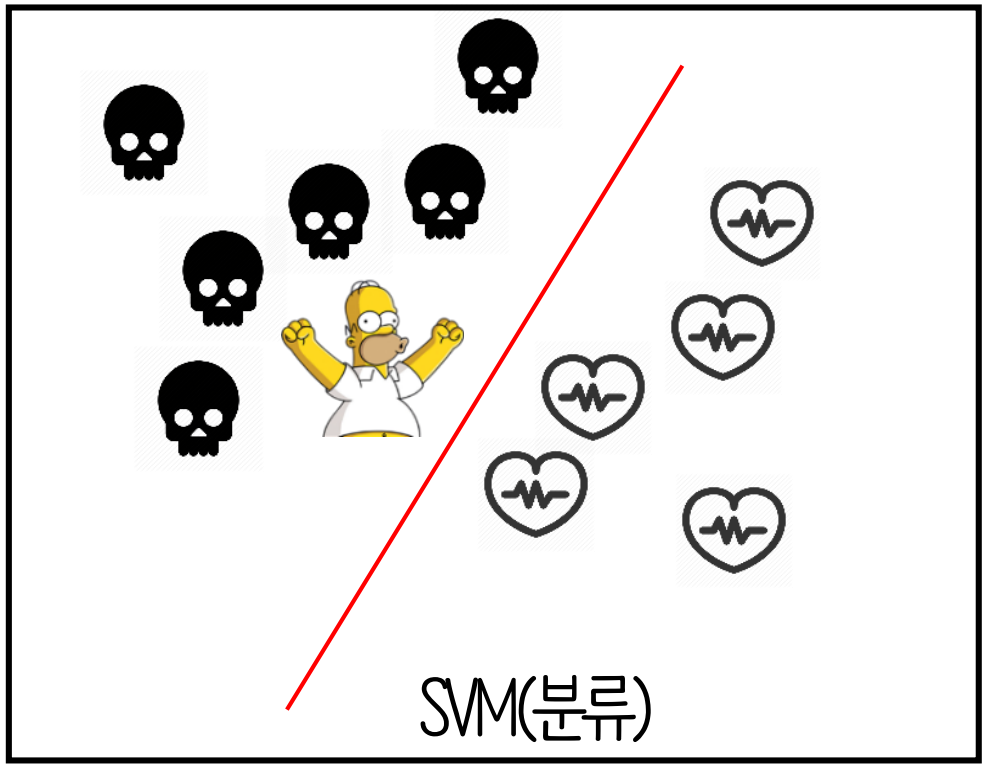


SVM 알고리즘 예시

kNN 과 SVM 알고리즘

심슨은 죽었을까 살았을까?!





SVM 알고리즘 관련 용어

SVM 알고리즘 관련 용어

- 결정 경계 : 서로 다른 분류 값을 결정하는 경계
- 서포트 벡터 : 결정 경계와 가장 가까운 데이터 포인트
- 마진 : 서포트 벡터와 결정 경계 사이의 거리
- 초모수 c: 에러를 허용하여 비용을 조절해주는 초모수
- 커널 트릭 : 저차원 데이터를 고차원으로 옮겨서 데이터를 찾는 방법
- 감마 : 결정 경계의 곡률을 조정하는 변수

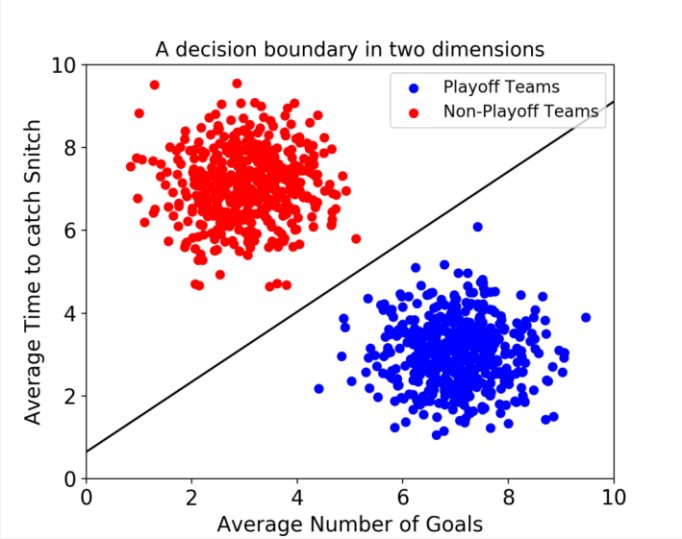
결정 경계

결정 경계

▷ 데이터를 분류하기 위한 기준 경계

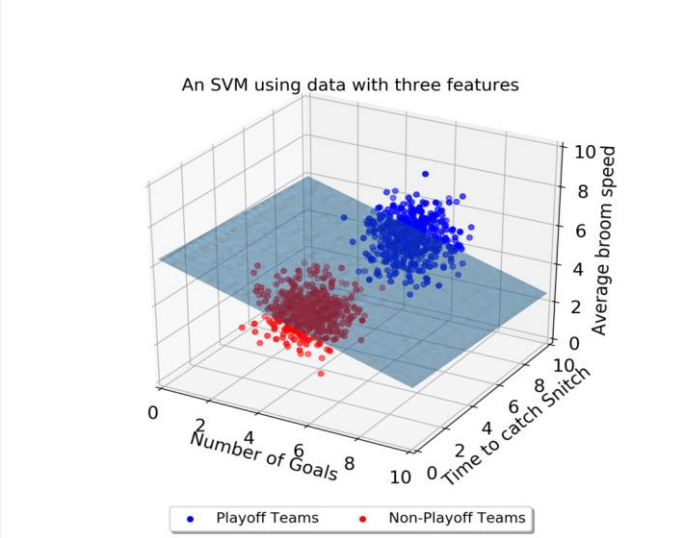
▷ 서포트 벡터를 통해 결정 경계를 찾음

▷ 결정 경계는 (데이터의 차원 - 1) 차원.



2차원 데이터의 결정 경계 -> 선의 형태

“결정 경계선 "



3차원 데이터의 결정 경계 -> 평면의 형태

“ 초평면 "

서포트 벡터

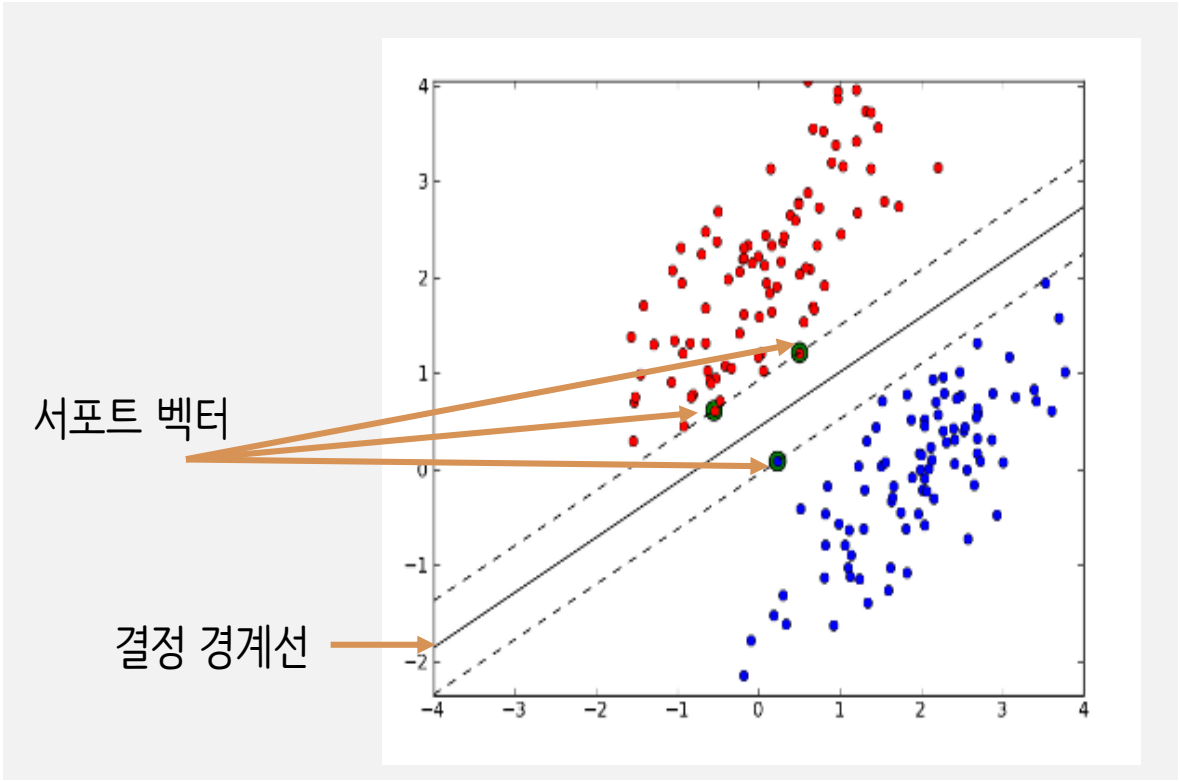
서포트 벡터

▷ 방향과 크기를 갖는 선의 벡터 개념이 아니라

(0, 0) 위치에서 특정 지점까지의 벡터를 점으로 형상한 개념

▷ 결정 경계선과 가장 맞닿아 있는 데이터

▷ 적절한 결정 경계를 찾는데 사용



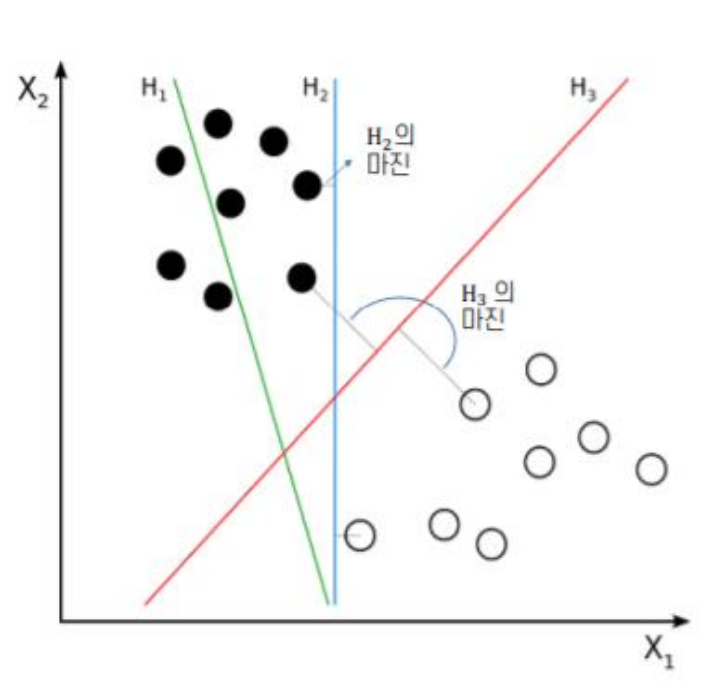
결정 경계를 정의하는데 다른 데이터들에 비해 많은 영향을 끼치므로
결정 경계를 지지(Support)한다고 하여서 "서포트 벡터" 라고함

마진

마진

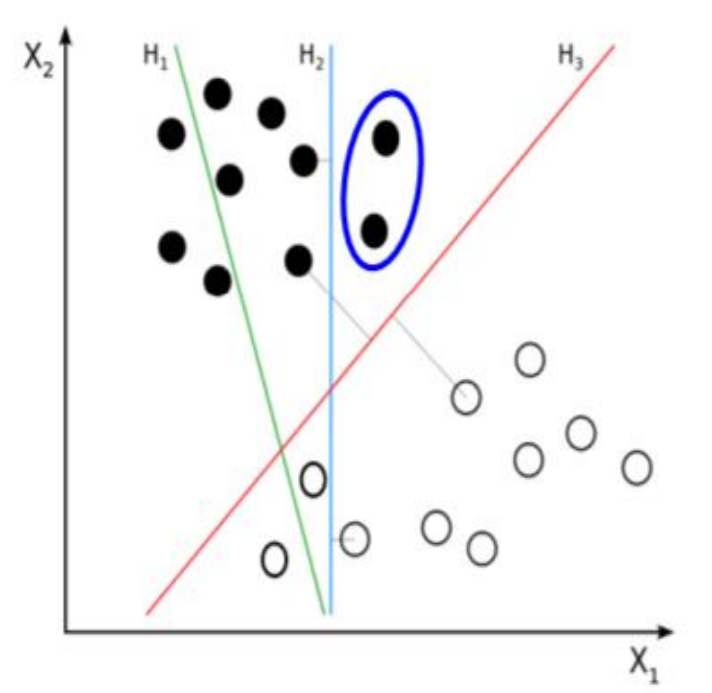
▷ 결정 경계와 서포트 벡터 사이의 거리

▷ 마진이 클수록 새로운 데이터를 안정적으로 분류할 가능성 증가



마진이 최대가 되는 H3가

적절한 결정 경계



새로운 데이터가 들어와도

H3 결정 경계가 가장 안정적

초모수 C

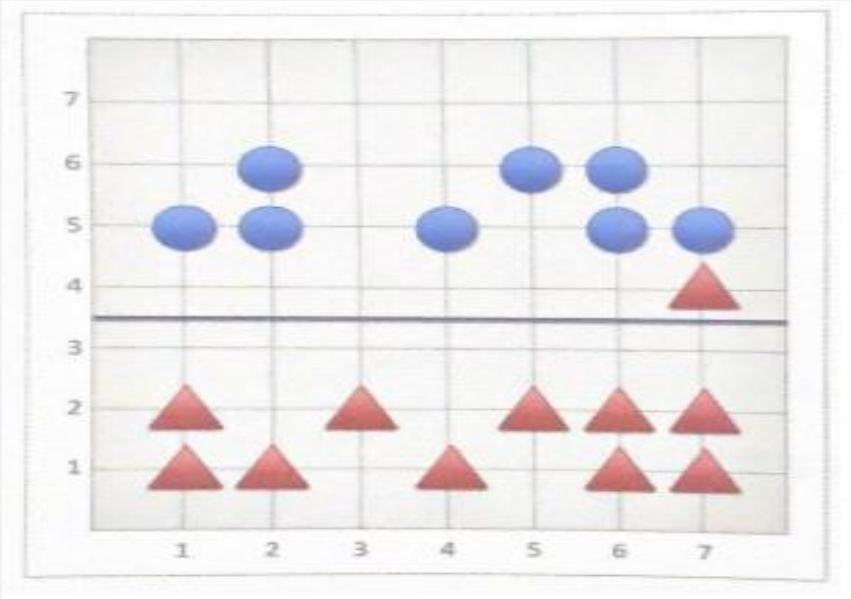
SVM을 사용하면서 우리가 마주칠 수 있는 문제

그림의 ★과 같이 분류하기 애매한 이상치(outlier)의 존재로

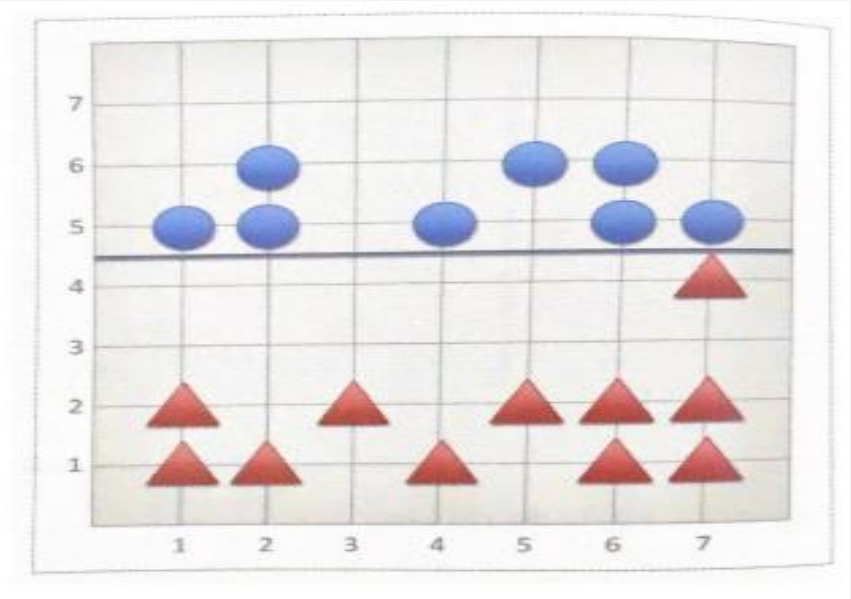
결정 경계를 정하는 것이 어려워질 수 있다.

최적의 결정 경계

SVM을 사용하면서 우리가 마주칠 수 있는 문제



마진이 큰 결정 경계를 갖지만 학습의 결과에 에러가 있는 경우



마진이 작은 결정 경계를 갖지만 학습의 결과에 에러가 없는 경우

테스트 데이터를 잘 분류하는 것이 목적이고

마진이 클수록 새로운 데이터를 안정적으로 분류할 가능성 증가하므로

학습 시 에러가 존재하더라도 마진이 큰 결정 경계를 선택하는 것이 적절함.

초모수 C

초모수 c

▷ 데이터 학습을 통해 결정 경계를 정하는 과정에서 약간의 오류를 허용하는 개념

▷ c값을 크게 줌-› 이상치 가능성 낮게 잡음-› 세심한 결정 경계 찾음

∴ 마진(거리) 낮아짐, 학습 에러률 감소

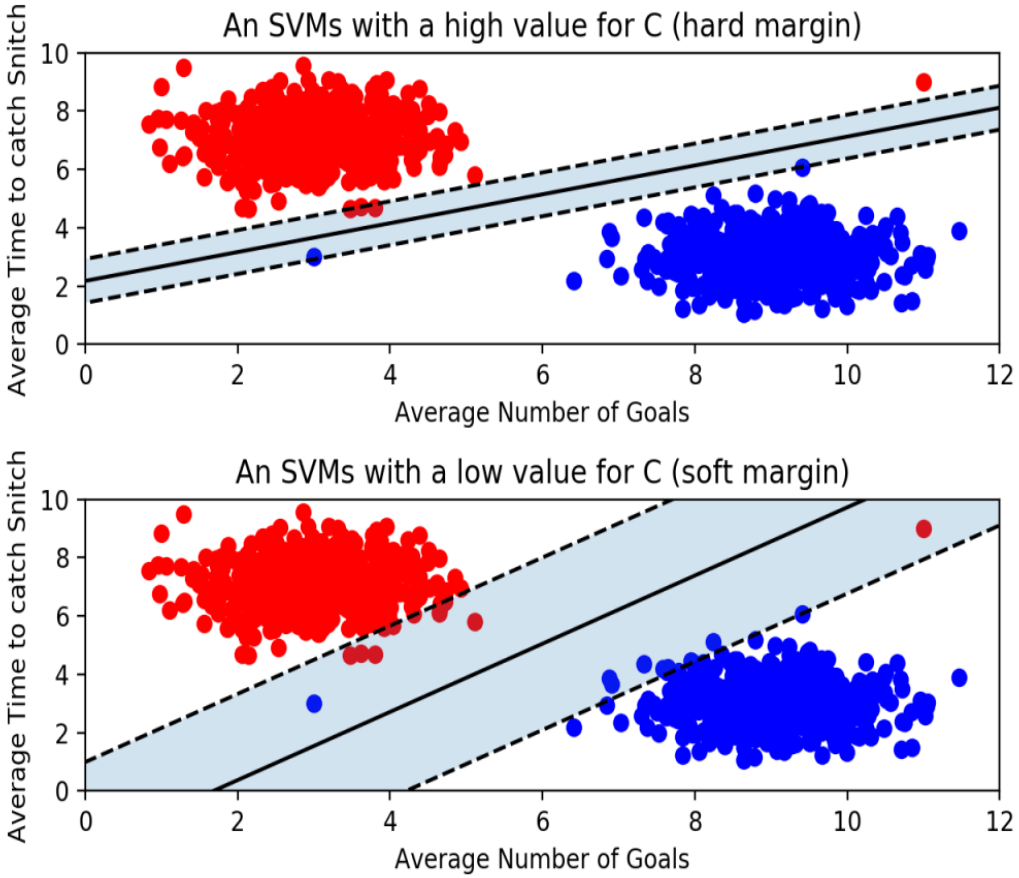
▷ c값을 작게 줌-› 이상치 가능성 높게 잡음-› 일반적인 결정 경계 찾

∴ 마진(거리) 높아짐, 학습 에러률 증가 **soft margin**

▷ SVM 기법의 정확도를 높이기 위한 파라미터 중 하나로 작용

▷ 비용이 너무 낮으면 - › 과소적합의 위험

비용이 너무 높으면 -› 과대적합의 위험

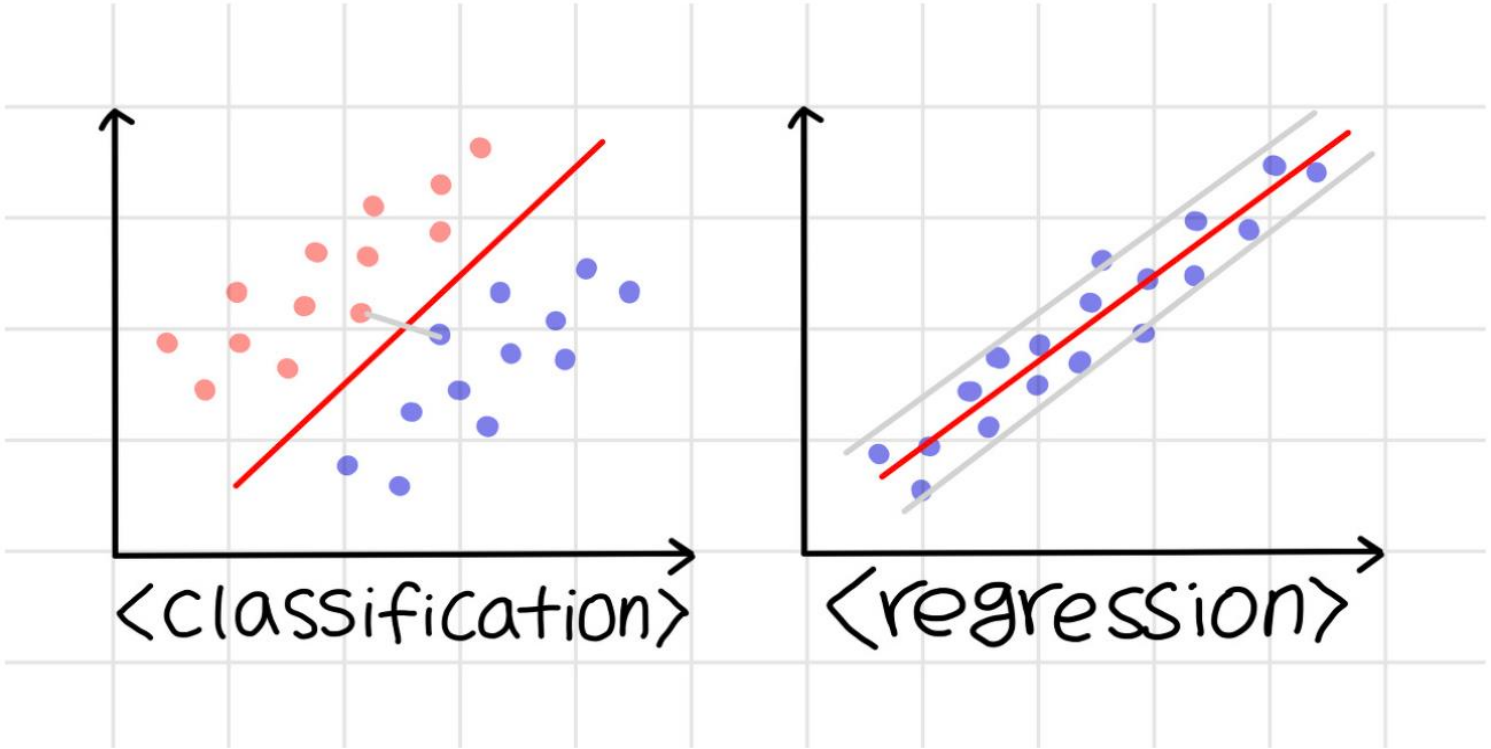


비용 변수

SVM 목적에 따른 margin 선택

▷ 분류(classification)에서는 soft margin 택함

▷ 회귀(regression)에서는 마진이 좁을수록 집단을 더욱 잘 대표하므로 hard margin 택함



커널 트릭

커널 트릭

▷ SVM의 종류

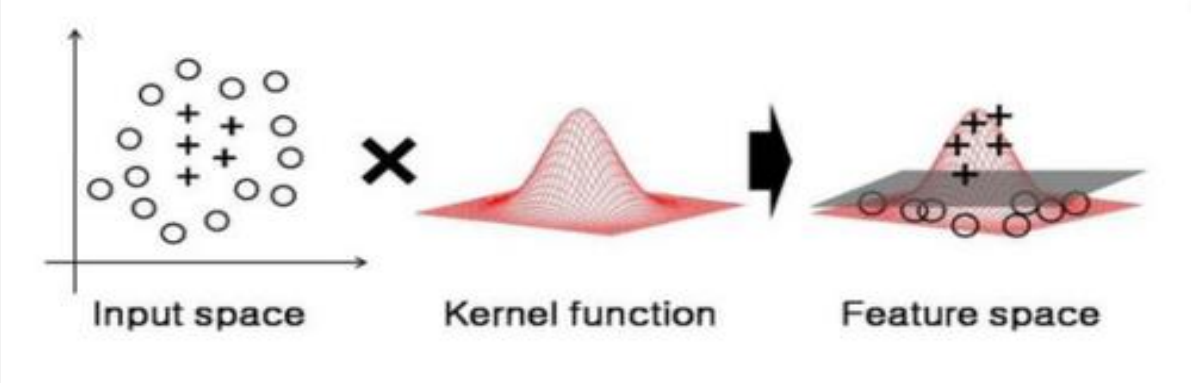
선형 SVM : 선형 결정 경계를 사용하여 데이터 분류가 가능한 경우 (별도의 커널 트릭 사용x)

비선형 SVM : 선형 결정 경계를 데이터 분류에 사용할 수 없는 경우 (커널트릭 사용)

▷ 저차원 공간(low dimensional space)을 고차원 공간(high dimensional space)으로 매핑해주는 작업

▷ 선형 경계를 사용할 수 없는 경우 커널 함수와 기법을 적용하여 결정 경계를 찾음

▷ rbf, sigmoid, Polynomial 등의 기법이 존재

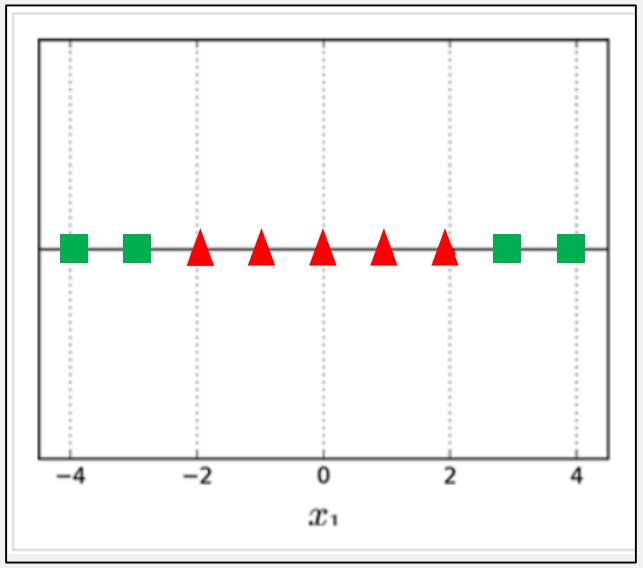


커널 함수를 적용하여

선형으로 분류할 수 있도록 함

커널 트릭 비선형SVM

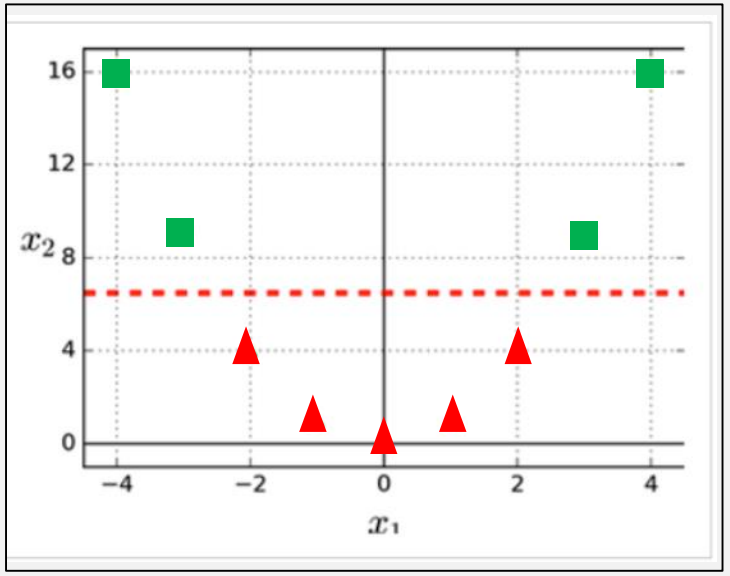
Ex 1) 1차원 데이터를 커널 트릭을 통해 2차원으로 매핑하고 선형 결정 경계를 생성



1차원 데이터

커널 함수 $x_2 = x_1^2$

커널 트릭

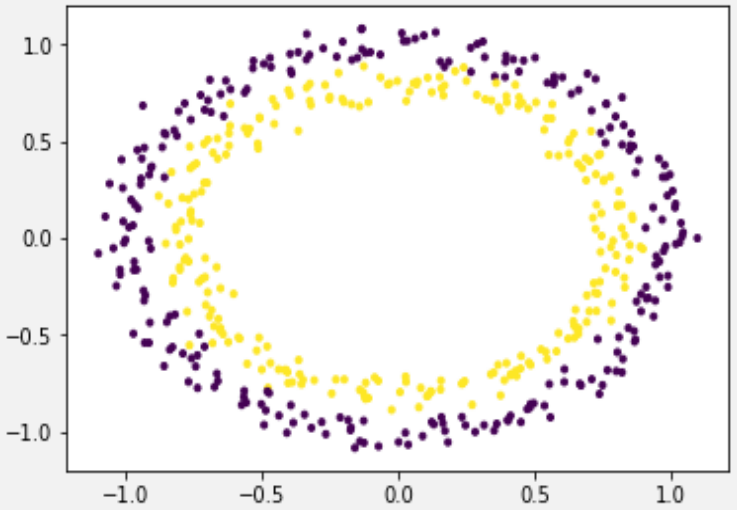


커널 함수를 통해 x2 변수를 생성한

2차원 데이터

커널 트릭 비선형SVM

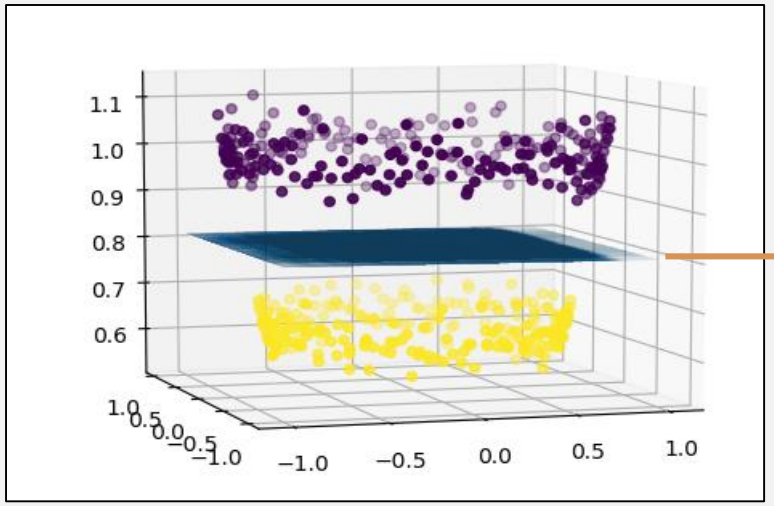
Ex2) 2차원 데이터를 3차원으로 매핑하고 결정 경계 생성



선형 결정 경계로 데이터를 분류할 수 없는
2차원 데이터

커널 함수 $Z = X^2 + Y^2$

커널 트릭



결정 경계

X, Y, Z 평면 3차원으로 확대하여
결정 경계를 생성

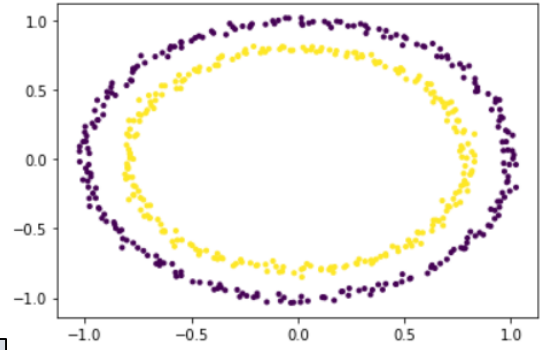
커널 트릭 비선형SVM

Ex2) 2차원 데이터를 3차원으로 매핑하고 결정 경계 생성

1. 데이터 생성

```
# importing libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from mpl_toolkits.mplot3d import Axes3D

# generating data
X, Y = make_circles(n_samples = 500, noise = 0.02)

# visualizing data
plt.scatter(X[:, 0], X[:, 1], c = Y, marker = '.')
plt.show()
```

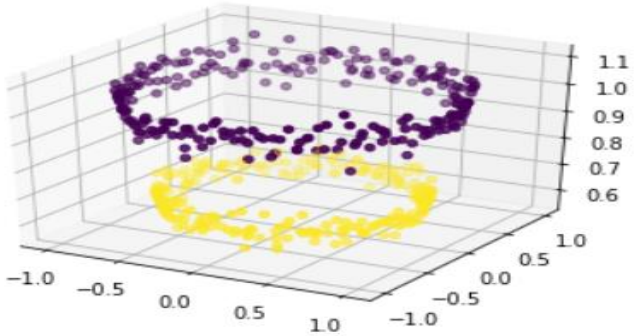
- make_circles : 두 개의 원형 데이터 생성
- n_samples : 생성할 데이터 수 (디폴트 : 100)
- noise : 데이터에 추가된 표준편차
- 생성 된 X는 좌표를 나타내고, Y는 보라색(0)/노란색(1) 클래스를 나타냄



2. 차원 변경

```
# adding a new dimension to X
X1 = X[:, 0].reshape((-1, 1))
X2 = X[:, 1].reshape((-1, 1))
X3 = (X1**2 + X2**2)
X = np.hstack((X, X3))

# visualizing data in higher dimension
fig = plt.figure()
axes = fig.add_subplot(111, projection = '3d')
axes.scatter(X1, X2, X1**2 + X2**2, c = Y, depthshade = True)
plt.show()
```

- 데이터 형태 변경 후 커널 함수를 적용하여 Z(X3) 변수를 생성
- fig_add_subplot()을 통해 3D 축을 생성하여 3차원 그래프를 출력



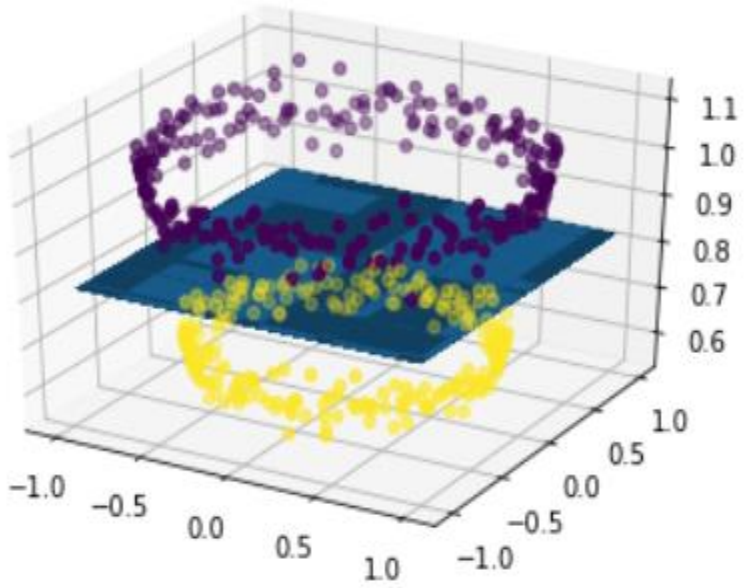
커널 트릭 비선형SVM

Ex2) 2차원 데이터를 3차원으로 매핑하고 결정 경계 생성

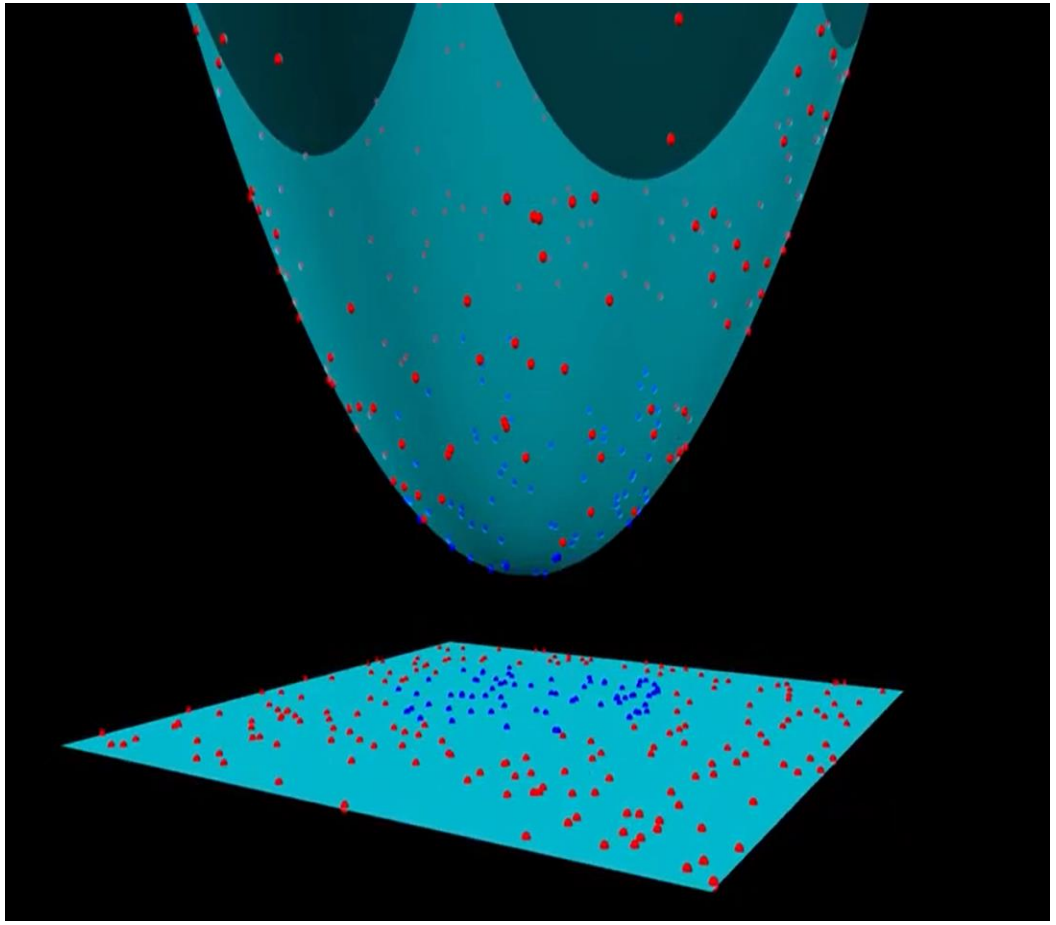
3.데이터 분류

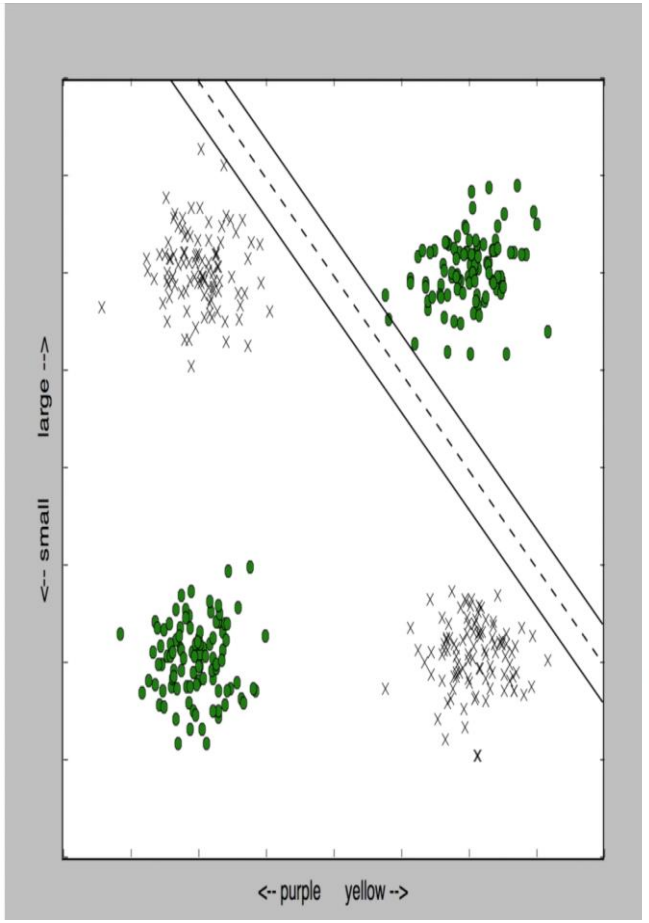
```
# create support vector classifier using a linear kernel
from sklearn import svm

svc = svm.SVC(kernel = 'linear')
svc.fit(X, Y)
w = svc.coef_
b = svc.intercept_

# plotting the separating hyperplane
x1 = X[:, 0].reshape((-1, 1))
x2 = X[:, 1].reshape((-1, 1))
x1, x2 = np.meshgrid(x1, x2)
x3 = -(w[0][0]*x1 + w[0][1]*x2 + b) / w[0][2]

fig = plt.figure()
axes2 = fig.add_subplot(111, projection = '3d')
axes2.scatter(X1, X2, X1**2 + X2**2, c = Y, depthshade = True)
axes1 = fig.gca(projection = '3d')
axes1.plot_surface(x1, x2, x3, alpha = 0.01)
plt.show()
```

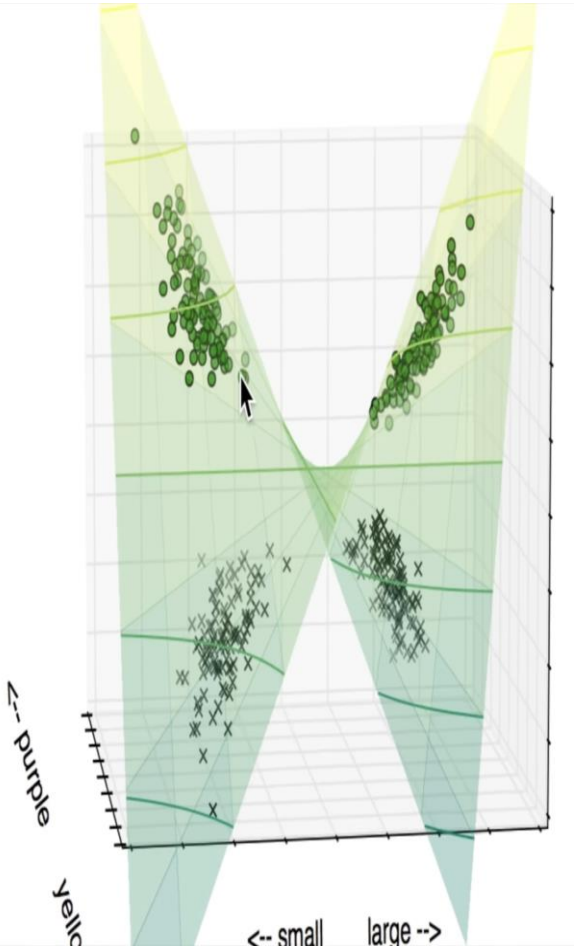
- 선형 모델을 위해 kernal을 'linear'로 지정
- plot_surface를 통해 3차원 그래프에서 평면 출력, alpha는 투명도 조절



커널 트릭 비선형SVM







https://youtu.be/0dlNM96sHio

https://youtu.be/-Z4aojJ-pdg 또 다른 예시 : 9분 25초부터

가우시안 RBF 커널

RBF 커널

▷ 가장 일반적으로 많이 사용하는 커널 (kernel = "rbf") : default값

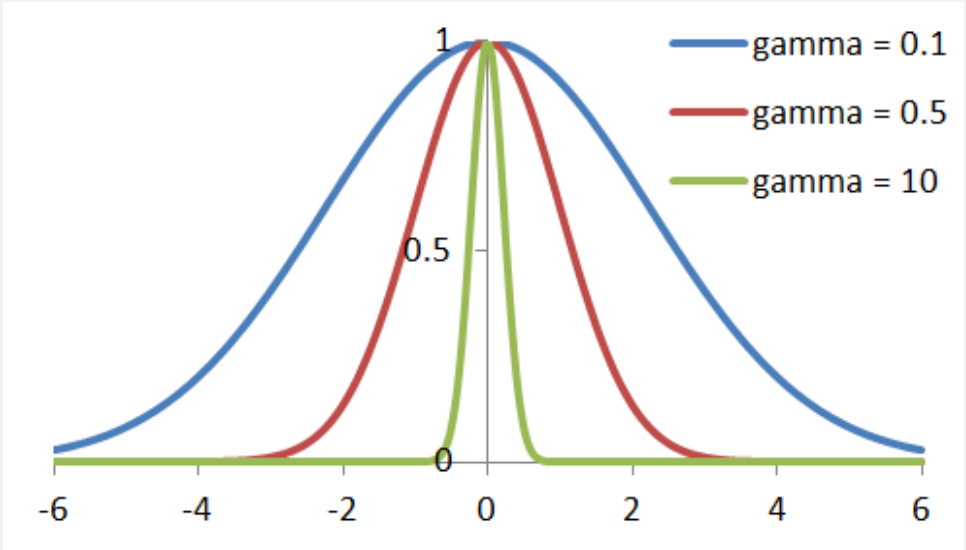
▷ 비용 변수와 감마 파라미터를 사용

- 오류 허용 정도를 조정하여 마진의 너비를 조정하는 파라미터 -> 초모수 c
- 샘플이 영향력을 행사하는 거리를 조정하는 파라미터, 가우시안 함수의 표준편차와 관련-> 감마

▷ 감마가 증가할수록 작은 표준편차

= 영향력을 행사하는 거리가 짧아짐

$$k(x_1, x_2) = \exp\left(-\gamma||x_1 - x_2||^2\right)$$



각 데이터 샘플이 영향력을 행사하는 거리 = 감마

가우시안 RBF 커널

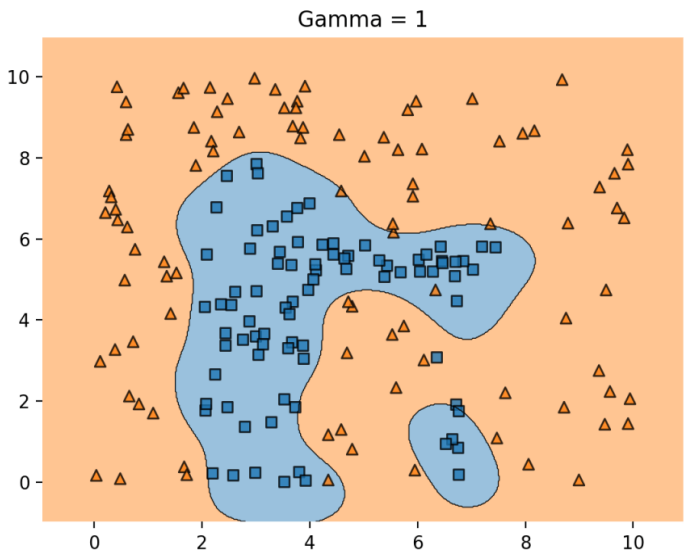
RBF 커널

▷ 감마를 사용하여 결정 경계의 곡률을 조정함

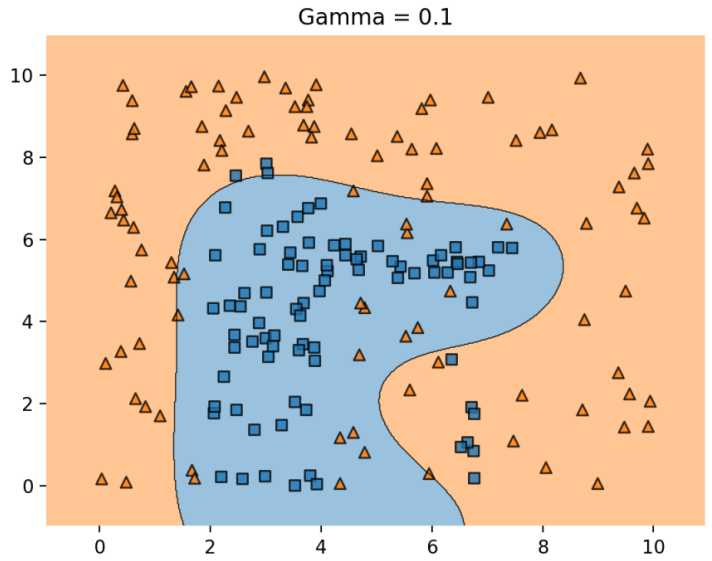
- 감마를 증가시키면 표준편차가 작아짐 -> 각 샘플의 영향 범위가 작아짐 -> 결정 경계가 불규칙해지고 각 샘플을 따라 휘어짐 .
- 감마를 감소시키면 표준편차가 커짐 -> 각 샘플의 영향 범위가 넓어짐 -> 결정 경계가 부드러워짐

▷ 감마가 너무 작으면 -> 과소적합의 위험

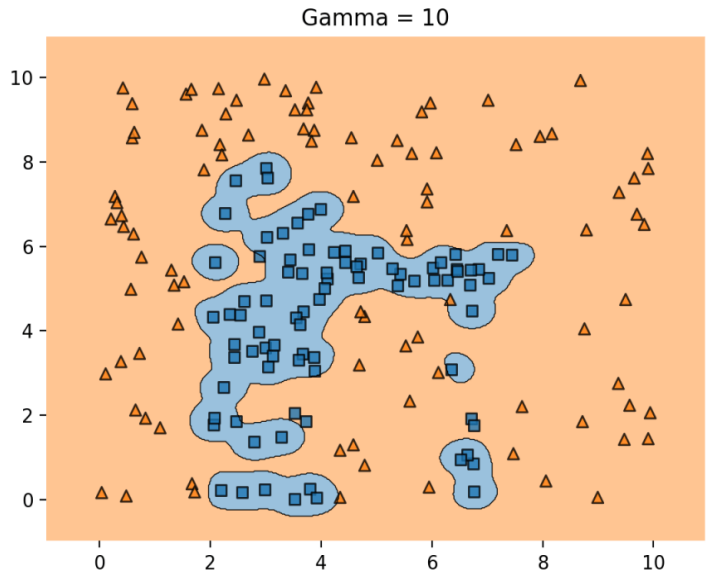
감마가 너무 크면 - > 과대적합의 위험



적절한 감마 값



더 작은 감마 값



더 큰 감마 값

SVM 실습-농구선수 포지션 예측

1. 데이터 분석

1. 데이터 수집

```
import pandas as pd

#데이터 불러오기
df = pd.read_csv("https://raw.githubusercontent.com/wikibook/machine-learning/2.0/data/csv/basketball_stat.csv")
```

```
#데이터 형태 확인
print(df.shape)
print('----------------------------------------------------')
df.head()
```

```
(100, 8)
----------------------------------------------------
```

	Player	Pos	3P	2P	TRB	AST	STL	BLK
0	Alex Abrines	SG	1.4	0.6	1.3	0.6	0.5	0.1
1	Steven Adams	C	0.0	4.7	7.7	1.1	1.1	1.0
2	Alexis Ajinca	C	0.0	2.3	4.5	0.3	0.5	0.6
3	Chris Andersen	C	0.0	0.8	2.6	0.4	0.4	0.6
4	Will Barton	SG	1.5	3.5	4.3	3.4	0.8	0.5

```
#현재 가지고 있는 데이터에서 포지션의 개수 확인
df.Pos.value_counts()
```

```
C     50
SG    50
Name: Pos, dtype: int64
```

Data 설명

포지션

- 센터(C) : 골대, 포스트 근처에서 슛을 블로킹/ 리바운드/공을 빼내서 공격찬스 만들어서 득점
- 슈팅가드(SG) : 코트내에서 3점 슛과 같은 장거리 슛을 통해 득점

변수	변수 설명	포지션 별 특성	분별력
3P	한 경기 평균 3점슛 성공 횟수	슈팅 가드의 주요 역할	0
2P	한 경기 평균 2점슛 성공 횟수	슈팅가드와 센터 모두 수행	X
TRB	한 경기 평균 리바운드 성공 횟수	센터의 주요 역할	0
AST	한 경기 평균 어시스트 성공 횟수	슈팅가드와 센터 모두 수행	X
STL	한 경기 평균 스틸 성공 횟수	슈팅가드와 센터 모두 수행	X
BLK	한 경기 평균 블로킹 성공 횟수	센터의 주요 역할	0

데이터 주소 : https://raw.githubusercontent.com/wikibook/machine-learning/2.0/data/csv/basketball_stat.csv

SVM 실습–농구선수 포지션 예측

1. 데이터 분석

2. 데이터 시각화

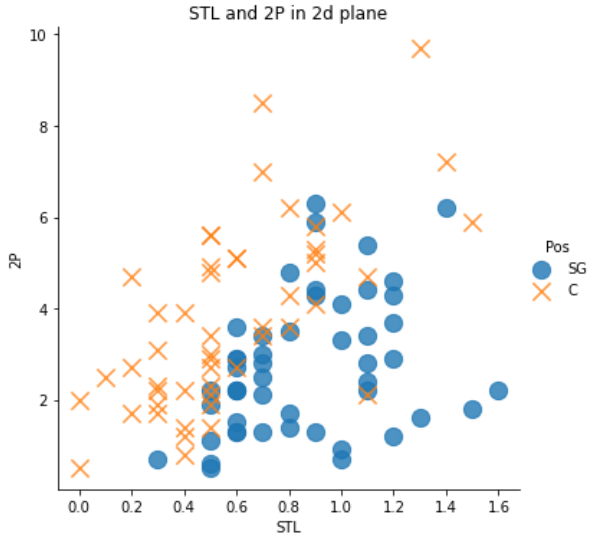
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# 스틸, 2점슛 데이터 시각화
sns.lmplot('STL', '2P', data=df, fit_reg=False,  # x 축, y 축, 데이터, 라인 없음
           scatter_kws={"s": 150}, # 좌표 상의 점의 크기
           markers=["o", "x"],
           hue="Pos") # 예측값

# title
plt.title('STL and 2P in 2d plane')
```

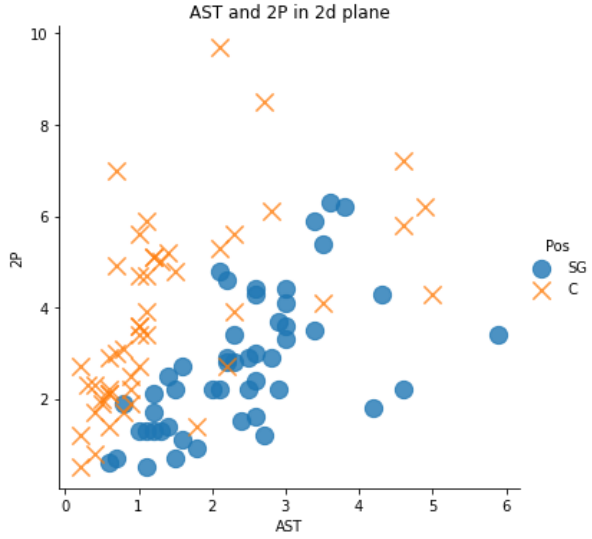
```
# 어시스트, 2점슛 데이터 시각화
sns.lmplot('AST', '2P', data=df, fit_reg=False,  # x 축, y 축, 데이터, 라인 없음
           scatter_kws={"s": 150}, # 좌표 상의 점의 크기
           markers=["o", "x"],
           hue="Pos") # 예측값

# title
plt.title('AST and 2P in 2d plane')
```

2점슛, 어시스트, 스틸 속성으로 데이터를 분류할 경우

센터와 슈팅가드 경계가 근접하여 분류에 적절하지 않다

→ 분류할 때 필요하지 않은 속성으로 데이터에서 제거





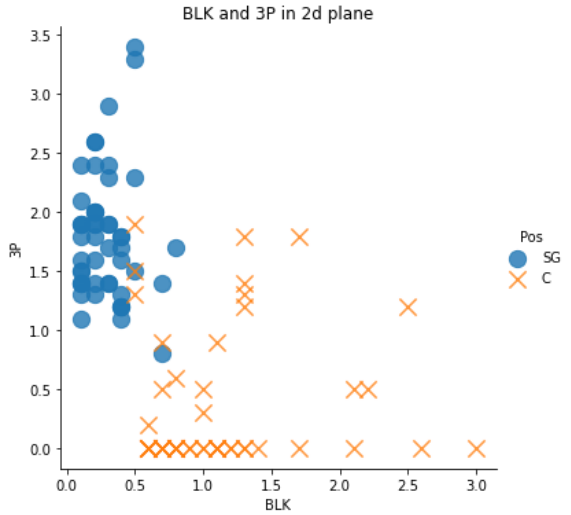
SVM 실습–농구선수 포지션 예측

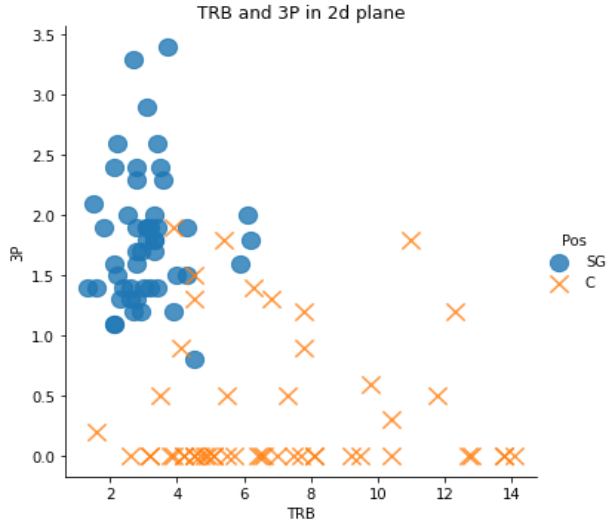
1. 데이터 분석

2. 데이터 시각화

```
# 블로킹, 3점슛 데이터 시각화
sns.lmplot('BLK', '3P', data=df, fit_reg=False,  # x 축, y 축, 데이터, 라인 없음
           scatter_kws={"s": 150}, # 좌표 상의 점의 크기
           markers=["o", "x"],
           hue="Pos") # 예측값

# title
plt.title('BLK and 3P in 2d plane')
```

```
# 리바운드, 3점슛 데이터 시각화
sns.lmplot('TRB', '3P', data=df, fit_reg=False,  # x 축, y 축, 데이터, 라인 없음
           scatter_kws={"s": 150}, # 좌표 상의 점의 크기
           markers=["o", "x"],
           hue="Pos") # 예측값

# title
plt.title('TRB and 3P in 2d plane')
```





블로킹, 리바운드, 3점슛 속성으로 데이터를 분류할 경우

센터와 슈팅가드 분류에 적절하다

→ 분류할 때 필요한 속성으로 데이터에서 유지

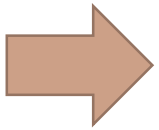
SVM 실습-농구선수 포지션 예측

1. 데이터 분석

3. 데이터 다듬기

```
# 분별력이 없는 특징(feature)을 데이터에서 제거합니다
df.drop(['2P', 'AST', 'STL'], axis=1, inplace = True)
```

	Player	Pos	3P	TRB	BLK
0	Alex Abrines	SG	1.4	1.3	0.1
1	Steven Adams	C	0.0	7.7	1.0
2	Alexis Ajinca	C	0.0	4.5	0.6
3	Chris Andersen	C	0.0	2.6	0.6
4	Will Barton	SG	1.5	4.3	0.5



```
# 2차원 데이터로 바꾸기 위해 TRB를 제거하여 다시 df로 저장
df.drop('TRB' , axis =1 , inplace = True)
df.head()
```

	Player	Pos	3P	BLK
0	Alex Abrines	SG	1.4	0.1
1	Steven Adams	C	0.0	1.0
2	Alexis Ajinca	C	0.0	0.6
3	Chris Andersen	C	0.0	0.6
4	Will Barton	SG	1.5	0.5

SVM 실습–농구선수 포지션 예측

2. SVM 결정

1. 데이터 정리

```
# 3P, BLK X로 저장
# 포지션 값은 Y로 저장

X =df[['3P',"BLK"]]
Y =df[['Pos']]
```

2. 선형 SVM

```
from sklearn.svm import SVC
import sklearn.metrics as mt
from sklearn.model_selection import cross_val_score , cross_validate

#SVM , kernel ="linear"로 선형 분리 진행
svm_clf_l = SVC(kernel= "linear")

#교차검증
scores_l = cross_val_score(svm_clf_l , X, Y , cv =5 )
scores_l

print("교차검증 평균:" , scores_l.mean())
print(pd.DataFrame(cross_validate(svm_clf_l,X,Y,cv= 5 )))
```

교차검증 평균: 0.96

	fit_time	score_time	test_score
0	0.002172	0.000932	1.00
1	0.002115	0.000936	0.95
2	0.002058	0.000888	0.95
3	0.002121	0.000917	1.00
4	0.002889	0.001023	0.90

3. 비선형 SVM

```
#SVM , kernel ="rbf"로 비선형 분리 진행
svm_clf_r = SVC(kernel= "rbf")

#교차검증
scores_r = cross_val_score(svm_clf_r , X, Y , cv =5 )
scores_r

print("교차검증 평균:" , scores_r.mean())
print(pd.DataFrame(cross_validate(svm_clf_r,X,Y,cv= 5 )))
```

교차검증 평균: 0.95

	fit_time	score_time	test_score
0	0.002523	0.001087	1.00
1	0.002396	0.000996	0.90
2	0.002235	0.001202	0.95
3	0.002281	0.000968	1.00
4	0.002573	0.000967	0.90

선형/비선형 중 하나를 선택하기 위해 전체 데이터를 대상으로 교차검증 평균을 낸다.

현재 데이터는 둘의 차이가 거의 없기 때문에 임의로 비선형으로 진행할 예정이다.

선형: kernel='linear' // 비선형: kernel='rbf'

SVM 실습–농구선수 포지션 예측

3. SVM 실습

1. 학습 / 테스트 데이터셋 분할 (데이터 분석 단계에서 생성한 df 데이터셋 사용)

```
# sklearn의 train_test_split을 사용하면 라인 한줄로 손쉽게 데이터를 나눌 수 있다
from sklearn.model_selection import train_test_split

# 다듬어진 데이터에서 20%를 테스트 데이터로 분류합니다
train, test = train_test_split(df, test_size=0.2, randomstate=100)
```

```
#학습 데이터의 형태 확인
train.shape
train
```

(80, 4)

	Player	Pos	3P	BLK
29	Channing Frye	C	1.9	0.5
83	Jason Smith	C	0.5	0.7
82	J.R. Smith	SG	2.3	0.3
43	John Henson	C	0.0	1.3

```
#테스트 데이터의 형태 확인
test.shape
```

(20, 4)

2. 최적의 SVM 파라미터 찾기

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC

import numpy as np
```

RBF 커널 사용하는 SVM 파라미터

1. C: 비용, 결정 경계선의 마진을 결정하는 파라미터
2. Gamma: 커널의 데이터포인트의 표준편차를 결정하는 파라미터

sklearn의 GridSearch를 사용하면
개발자가 부여한 비용과 감마 후보를 조합하여 최적의 C, gamma 조합을 구할 수 있다.

SVM 실습–농구선수 포지션 예측

3. SVM 실습

2. 최적의 SVM 파라미터 찾기

```
def svc_param_selection(X, y, nfolds):
    svm_parameters = [
                        {'kernel': ['rbf'],
                         'gamma': [0.00001,0.0001, 0.001, 0.01, 0.1, 1],
                         'C': [0.01, 0.1, 1, 10, 100, 1000]
                        }
                       ]

    clf = GridSearchCV(SVC(), svm_parameters, cv=10)
    clf.fit(X_train, y_train.values.ravel())
    print(clf.best_params_)

    return clf
```

```
X_train = train[['3P', 'BLK']]
y_train = train[['Pos']]
# 최적의 파라미터를 sklearn의 gridsearch를 통해 구합니다.
clf = svc_param_selection(X_train, y_train.values.ravel(), 10)
```

```
{'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
```

- 임의의 gamma, C 값을 지정하고, GridSearchCV를 적용한 clf 변수 생성 과정을 포함한 svc_param_selection 함수 정의
- 이후 X_train과 y_train 생성
- 실행 결과 최적의 비용 = 0.1, 최적의 감마 = 1 임을 알 수 있다.

3. 결정 경계 시각화

- 2번에서 찾은 파라미터가 최적의 파라미터인지 시각화를 통해 확인

```
#시각화를 위해서, 최적의 C와 이외의 C를 후보로 저장
C_candidates =[]
C_candidates.append(clf.best_params_['C']*0.01)
C_candidates.append(clf.best_params_['C'])
C_candidates.append(clf.best_params_['C']*100)

#시각화를 위해서, 최적의 gamma와 이외의 gamma를 후보로 저장
gamma_candidates =[]
gamma_candidates.append(clf.best_params_['gamma']*0.01)
gamma_candidates.append(clf.best_params_['gamma'])
gamma_candidates.append(clf.best_params_['gamma']*100)

X= train[['3P','BLK']]
Y= train['Pos'].tolist()
```

시각화할 비용/감마 후보를 저장.

→ 학습할 X는 3점슛 & 블로킹 횟수

→ 학습 모델 분류값 Y는 농구선수 포지션

SVM 실습–농구선수 포지션 예측

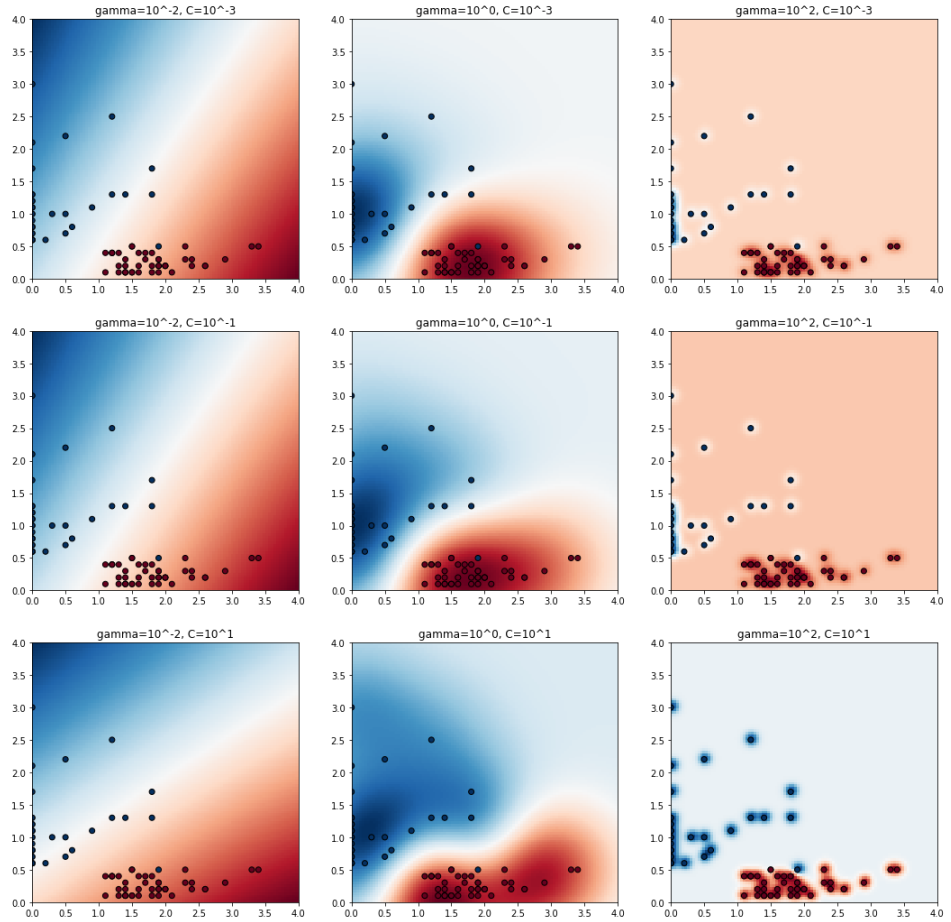
3. SVM 실습

3. 결정 경계선 시각화

```
#포지션에 해당하는 문자열 SG와 C를 벡터화
position = []
for gt in Y:
    if gt == 'C':
        position.append(0)
    else:
        position.append(1)

#각각의 파라미터에 해당하는 SVM 모델을 만들어 classifiers에 저장
classifiers = []
for C in C_candidates:
    for gamma in gamma_candidates:
        clf = SVC(C=C, gamma=gamma)
        clf.fit(X, Y)
        classifiers.append((C, gamma, clf))

# 18,18 사이즈의 챠트를 구성
plt.figure(figsize=(18, 18))
xx, yy = np.meshgrid(np.linspace(0, 4, 100), np.linspace(0, 4, 100))

# 각각의 모델들에 대한 결정 경계 함수를 적용하여 함께 시각화
for (k, (C, gamma, clf)) in enumerate(classifiers):
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # 최적의 모델을 포함한 다른 파라미터로 학습된 모델들을 함께 시각화
    plt.subplot(len(C_candidates), len(gamma_candidates), k + 1)
    plt.title("gamma=10^%d, C=10^%d" % (np.log10(gamma), np.log10(C)),
              size='medium')

    # 서포트 벡터와 결정경계선을 시각화
    plt.pcolormesh(xx, yy, -Z, cmap=plt.cm.RdBu)
    plt.scatter(X['3P'], X['BLK'], c=position, cmap=plt.cm.RdBu_r, edgecolors='k')
```

센터(C)는 0으로, 슈팅가드(SG)는 1로 벡터화

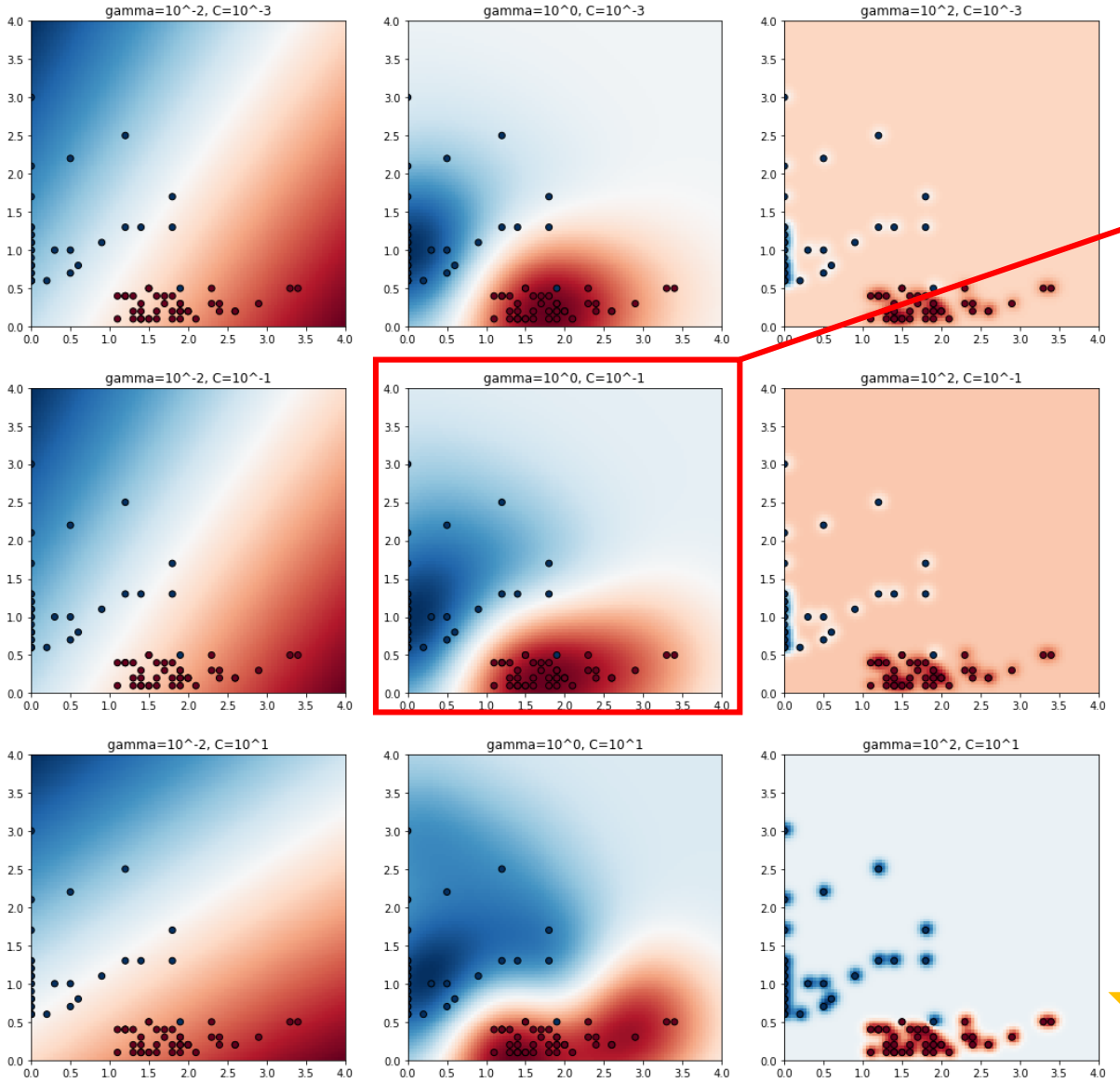
앞서 만들었던 파라미터 후보를 조합하고,
모델을 학습한 후 classifiers에 저장



SVM 실습–농구선수 포지션 예측

3. SVM 실습

3. 결정 경계선 시각화



최적의 C, gamma 사용한
SVM 모델

▷ 비용의 크기에 따라
결정 경계선 위치 변환

▷ 감마의 크기에 따라
결정 경계 곡률 변환

비용의 크기 증가

감마의 크기 증가

SVM 실습–농구선수 포지션 예측

3. SVM 실습

4. 모델 테스트

```
#테스트에 사용될 특징 변수를 X_test에 저장
X_test = test[["3P","BLK"]]

#특징으로 예측한 값과 비교할 변수를 Y_test에 저장
Y_test = test[["Pos"]]

#최적의 파라미터로 완성된 SVM에 테스트 데이터를 주입하여 실제값과 예측값을 얻음
y_true, y_pred = Y_test, clf.predict(X_test)

print(classification_report(y_true, y_pred))
print()
print("accuracy : " + str(accuracy_score(y_true, y_pred)))
```

```
              precision    recall  f1-score   support

           C       0.79      0.92      0.85        12
          SG       0.83      0.62      0.71         8

    accuracy                           0.80        20
   macro avg       0.81      0.77      0.78        20
weighted avg       0.80      0.80      0.79        20


accuracy : 0.8
```

- 테스트 결과 예측값(포지션)과 실제값의 정확도는 0.8
- 20명의 포지션 일치/ 불일치 확인

```
#예측값과 실제값의 일치 확인
comperison = pd.DataFrame({'prediction': y_pred, 'ground_truth':y_true.values.ravel()})
comperison
```

	prediction	ground_truth	
0	C	C	→ 일치
1	C	C	
2	SG	SG	
3	SG	SG	
4	C	C	
5	C	C	
6	C	SG	→ 불일치
7	C	C	
8	C	C	
9	SG	SG	
10	SG	C	
11	C	C	
12	C	C	
13	C	C	
14	C	SG	
15	C	SG	
16	SG	SG	
17	SG	SG	
18	C	C	
19	C	C	

SVM 알고리즘의 장단점

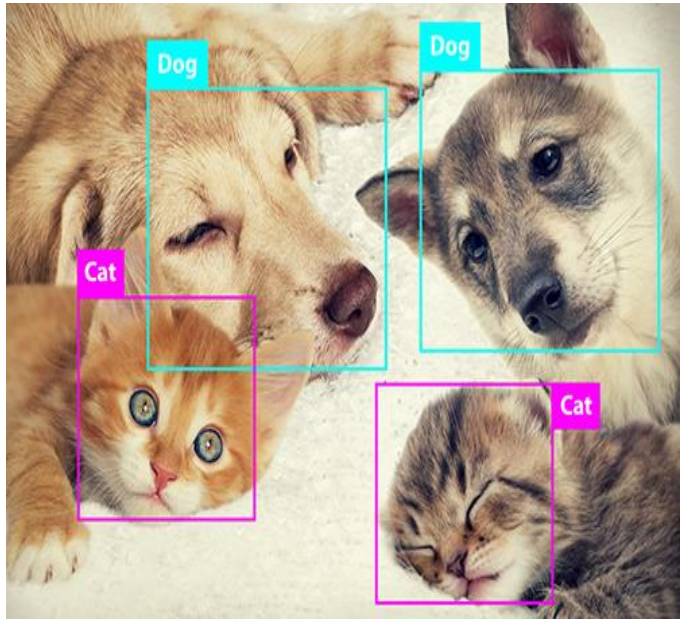
SVM 알고리즘의 단점

- 데이터의 차원이 높거나 특성이 확연히 다른 경우 데이터 전처리 과정이 더욱 중요해짐.
- 차원이 높을수록 결정 경계 및 데이터의 시각화가 어려움.

SVM 알고리즘의 장점

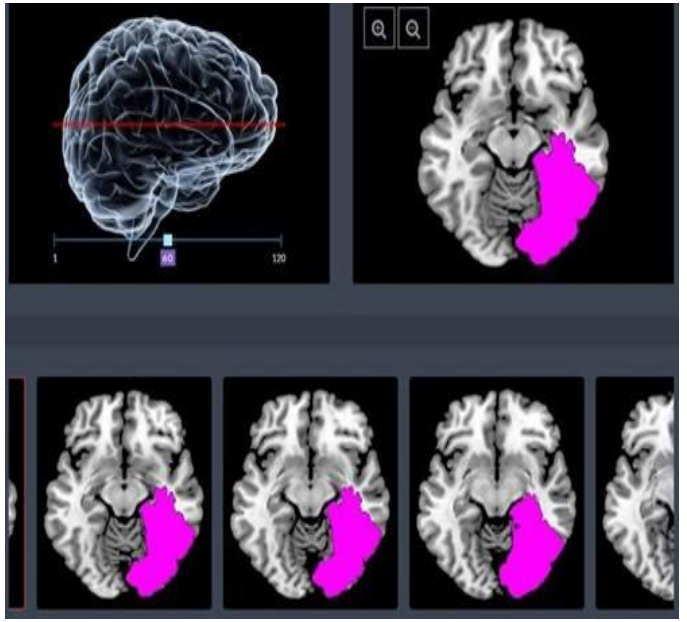
- 커널 트릭의 적용으로 특성이 다양한 데이터 분류에 강함.
- 파라미터의 조정으로 과소/ 과대 적합에 대처할 수 있음.
- 적은 학습으로도 정확도 높은 분류가 가능함.

SVM 알고리즘의 응용사례



텍스트 및 이미지 분류

유전자& 단백질 그룹핑
화합물 분류



환자 영상 데이터 &
환자 샘플 분류