



Day48; 20221114

📅 날짜	@2022년 11월 14일
👤 유형	@2022년 11월 14일
☰ 태그	

GitHub - u8yes/AI

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<https://github.com/u8yes/ai>

u8yes/AI



1 Contributor 0 Issues 0 Stars 0 Forks

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/3fffdc0a-e83f-44ef-94c8-e7e19644acd5/01_EDA_20221111.ipynb

- 기계 학습(일반적)

7:3 (훈련데이터:평가데이터)

8:2 (훈련데이터:평가데이터)

7,8 훈련데이터 셋을 가지고 다시 검증을 한다.

다시 훈련데이터를 가지고 train:validation (8:2)로 다시 나눈다.

서로 data노출이 되면 안 된다.

이제 막 생성된 데이터를 가지고 validation을 한다. (확인, 검증)

2,3을 2번 테스트 하는 개념이라고 보면 된다.(test → validation)

validation

[U] 확인, 비준

[영어사전 결과 더보기](#)

stratify

미국·영국 [strætɪfɪ] ◌

(동사)

층을 이루게 하다, 계층화하다

a highly stratified society ◌

고도로 계층화된 사회

[영어사전 결과 더보기](#)

sklearn의 train_test_split() 사용법

➤ Parameter & Return

```
from sklearn.model_selection import train_test_split  
train_test_split(arrays, test_size, train_size, random_state, shuffle, stratify)
```

(1) Parameter

- **arrays** : 분할시킬 데이터를 입력 (*Python list, Numpy array, Pandas dataframe 등..*)
- **test_size** : 테스트 데이터셋의 비율(float)이나 갯수(int) (*default = 0.25*)
- **train_size** : 학습 데이터셋의 비율(float)이나 갯수(int) (*default = test_size의 나머지*)
- **random_state** : 데이터 분할시 셔플이 이루어지는데 이를 위한 시드값 (*int나 RandomState로 입력*)
- **shuffle** : 셔플 여부 설정 (*default = True*)
- **stratify** : 지정한 Data의 비율을 유지한다. 예를 들어, Label Set인 Y가 25%의 0과 75%의 1로 이루어진 Binary Set일 때, stratify=Y로 설정하면 나누어진 데이터셋들도 0과 1을 각각 25%, 75%로 유지한 채 분할된다.

(2) Return

- **X_train, X_test, Y_train, Y_test** : arrays에 데이터와 레이블을 둘 다 넣었을 경우의 반환이며, 데이터와 레이블의 순서쌍은 유지된다.
 - **X_train, X_test** : arrays에 레이블 없이 데이터만 넣었을 경우의 반환
-

데이터 나누기(학습 데이터, 테스트 데이터)_20221114

```
[9]: # scikit-learn library 설치  
# !conda install scikit-learn
```

```
[13]: # sklearn의 train_test_split을 사용하면 코드 한 줄로 손쉽게 데이터를 나눌 수 있다.  
from sklearn.model_selection import train_test_split  
  
# 다음에선 데이터에서 20%를 테스트 데이터 분류.  
train, test = train_test_split(df, test_size=0.2) # test_size의 default는 0.25가 셋팅돼있다.
```

```
[14]: # 학습 데이터의 갯수.  
train.shape[0]
```

```
[14]: 80
```

```
[15]: # 평가(테스트) 데이터의 갯수.  
test.shape[0]
```

```
[15]: 20
```

```
[16]: train.to_csv("data/csv/basketball_train.csv", index = None)
```

```
[17]: test.to_csv("data/csv/basketball_test.csv", index = None)
```

라이브러리 임포트_20221114

- 실습에 필요한 라이브러리를 임포트한다.

```
[2]: import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns
```

데이터 불러오기 (학습 데이터, 테스트 데이터)_20221114

- 데이터 분석 단계에서 생성한 농구 포지션 예측하기의 학습 데이터 및 테스트 데이터를 로드합니다.

```
[3]: train = pd.read_csv("data/csv/basketball_train.csv")  
test = pd.read_csv("data/csv/basketball_test.csv")
```

```
[7]: print(train.shape[0]) # 행의 값만 확인 # 훈련용 데이터 셋 갯수  
print(train.shape[1]) # feature(특징, 열)의 값만 확인 # 훈련용 데이터 셋 갯수  
  
print(test.shape[0]) # 테스트 데이터 셋 행 갯수  
print(test.shape[1]) # 테스트 데이터 셋 열 갯수
```

```
80  
5  
20  
5
```

kNN

하이퍼 파라미터 = k값의 갯수를 조절해주는 것(홀수값)

최적의 k값은 절반 안에서 가능하다.(반 안으로 다 들어오니깐)

최소는 3개 이상부터.

최적의 k값 찾기_20221114

```
] : # import kNN library
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

# 최적의 k값 찾기. 3부터 시작해서 max_k_range / 2 까지의 범위.
max_k_range = train.shape[0] # k값 반복수 제크
k_list = []

for i in range(3, max_k_range//2, 2): # 3, 5, ... 39
    k_list.append(i)

print(k_list)

[3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39]
```

- cross_validation_scores = []

R시간에 학습한 **k-fold 교차 검증**과 헷갈리지 말자.

k-fold 교차 검증은 겹으로 한번씩은 반드시 validation이 사용될 수 있게 해서

결국엔 겹을 통해 전체를 다 validation하게 됨. 전체 겹의 결과값에 대해 평균을 구하는 것.

k-fold의 의미는 validation(확인)의 갯수를 의미한다. 전체 갯수가 아니라.

어쨌든 나머지는 훈련데이터.

최적의 k값 찾기_20221114

```
[1]: # import kNN library
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

# 최적의 k값 찾기. 3부터 시작해서 max_k_range / 2 까지의 범위.
max_k_range = train.shape[0] // 2 # k값 최대값으로 반복수 제크
k_list = []

for i in range(3, max_k_range, 2): # 3, 5, ..., 11, 13, ..., 39 # 근접 3개를 찾고, 근접 5개를 찾고..., 근접 39개를 찾음.
    k_list.append(i) # SG, C 인지 계속 최근접의 것들을 담음.

cross_validation_scores = []
x_train = train[['3P', 'TRB', 'BLK']] # 3개의 feature를 사용
# DataFrame으로 접근하는 pandas - loc, iloc 중요!! DataFrame은 그냥 []로 써면 에러남. [[]]로 써야지 작동함.
y_train = train[['Pos']] # 카테고리는 2개(SG, C)

[1]: # 10-fold cross validation
for k in k_list: # 3, 5, ..., 39(19개)
    knn = KNeighborsClassifier(n_neighbors=k) # 3nn .... 39nn모델(알고리즘)이 됨. # knn은 19번 실행됨.
    scores = cross_val_score(knn, x_train, y_train.values.ravel(), cv=10, scoring='accuracy')
    # ravel() 빌드로 평균(2차원을 1차원배열로) # cv = validation으로 1/10로 사용(10등분) # 10등분의 정확도로 저장, 사용
    cross_validation_scores.append(scores.mean()) # 총 데이터 19개가 들어짐

cross_validation_scores
```

```
[1]: [0.9375,
 0.925,
 0.925,
 0.9375,
 0.9375,
 0.9375,
 0.9375,
 0.9375,
 0.9375,
 0.9375,
 0.9375,
 0.9375,
 0.9375,
 0.9375,
 0.9375,
 0.9125,
 0.9125,
 0.9125,
 0.9125,
 0.875,
 0.8875,
 0.8875,
 0.8875,
 0.875]
```

ravel

발음 미국·영국 [ˈrævl] 미국식 [ˈrævl]



동사형 3인칭 단수 현재 ravels 과거형 ravelled raveled 과거 분사 ravelled raveled

현재 분사 ravelling raveling

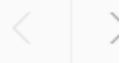
옥스퍼드

동마출판

YBM

교학사

슈프림



동사 | 동사구 | 영영사전

동사

1. 타동사 [VN] (-ㄹ-, 美 -l-)

(상황·문제를) 더 복잡하게[꼬이게] 만들다

동사구 1건

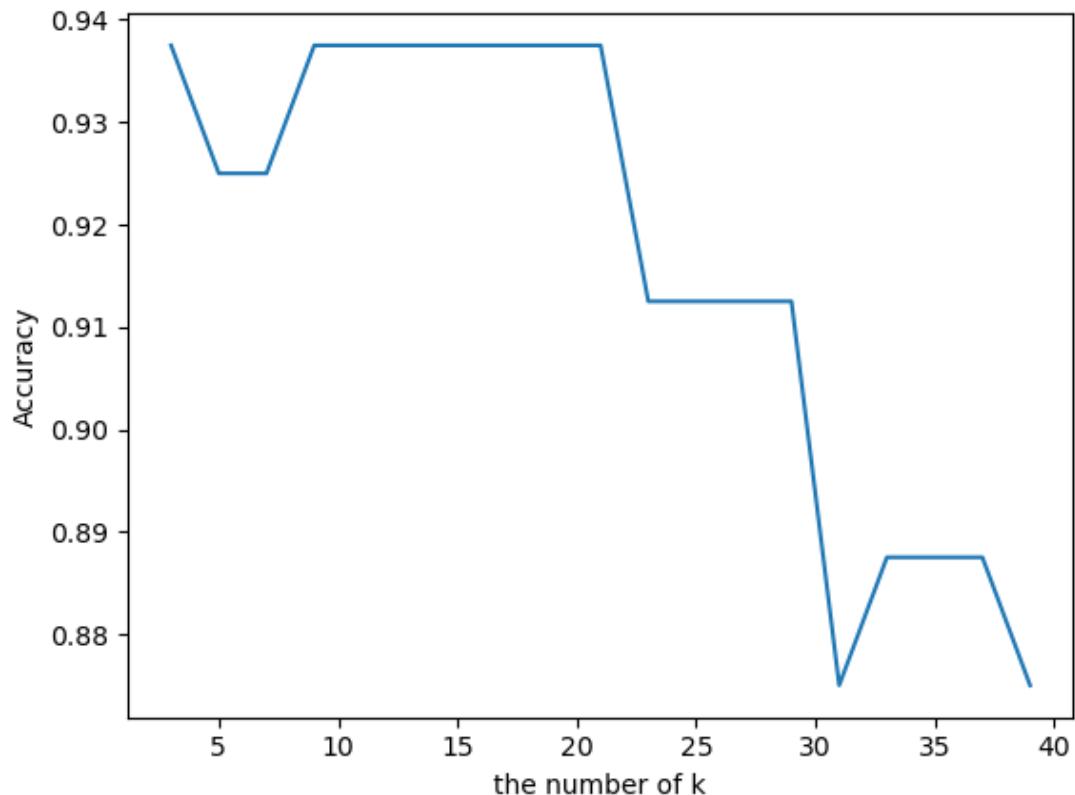
ravel something

(꼬이거나 맺힌 것을) 풀다

He was trying to ravel out the complicated series of events that had led to this situation.

그는 이런 상황에까지 이르게 된 복잡한 일련의 사건들을 풀어 보려고 매를 쓰고 있었다.

```
5] : # 시각화  
plt.plot(k_list, cross_validation_scores)  
plt.xlabel('the number of k')  
plt.ylabel('Accuracy')  
plt.show()
```



```
] : # 최적의 k 값  
cvs = cross_validation_scores  
k = k_list[cvs.index(max(cross_validation_scores))]  
print("최적의 k 값 : " + str(k))
```

최적의 k 값 : 3

2개의 특징으로 예측하기('3P', 'BLK')_20221114 ¶

```
: knn = KNeighborsClassifier(n_neighbors=k) # max 정확도를 가지고 생성함  
x_train = train[['3P', 'BLK']]  
y_train = train[['Pos']]  
  
knn.fit(x_train, y_train.values.ravel()) # fit() 학습시킨다.  
  
x_test = test[['3P', 'BLK']]  
y_test = test[['Pos']]  
  
pred = knn.predict(x_test)
```

```
: comparision = pd.DataFrame({'prediction':pred, 'target':y_test.values.ravel()}) # ravel = 1차 배열로 만들어줌.  
comparision
```

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

```
: from sklearn.metrics import accuracy_score  
  
print('accuracy : ' + str(accuracy_score(y_test.values.ravel(), pred))) # accuracy_score() 정답과 예측값 비교해서 반올해줄.  
accuracy : 0.8
```

3개의 특징으로 예측하기('3P', 'BLK','TRB')_20221114

```
|: knn = KNeighborsClassifier(n_neighbors=k)

x_train = train[['3P', 'BLK', 'TRB']]
y_train = train[['Pos']]

knn.fit(x_train, y_train.values.ravel())

x_test = test[['3P', 'BLK', 'TRB']]
y_test = test[['Pos']]

pred = knn.predict(x_test)
```

```
|: comparision = pd.DataFrame({'prediction':pred, 'target':y_test.values.ravel()})
comparision # 슈팅카드이나 셀트이나 블록 분류 분석(지도학습) - 2가지 중 하나를 예측. 데이터는 100명의 데이터.
```

```
|: prediction target
 0      SG    SG
 1      C    SG
 2      C     C
 3      SG    SG
 4      SG    SG
 5      SG    SG
 6      C     C
 7      C    SG
 8      SG    SG
 9      C     C
10     SG     C
11     SG    SG
12     SG    SG
13     SG    SG
14     SG    SG
15     C     C
16     C     C
17     C     C
18     C    SG
19     C     C
```

```
|: print('accuracy : ' + str(accuracy_score(y_test.values.ravel(), pred))) # accuracy_score() 정답과 예측값 비교해서 반환해줌.
accuracy : 0.8
```



1-05. Overfitting

- 머신러닝 진행 시 주의해야되는 가장 중요한 문제 중 하나
- Overfitting은 너무 잘 맞아 떨어진다는 의미임
- Training data에만 너무 잘 맞아 Training data 이외의 데이터들에 대해서는 잘 맞아 떨어지지 않는 경우
- 해결 방법
 - Cross Validation
 - Regularization
 - Remove Features
 - Ensembling
- underfitting : 아직 학습이 제대로 이루어지지 않은 상태
- 적절한(appropriate) 학습이 좋음!

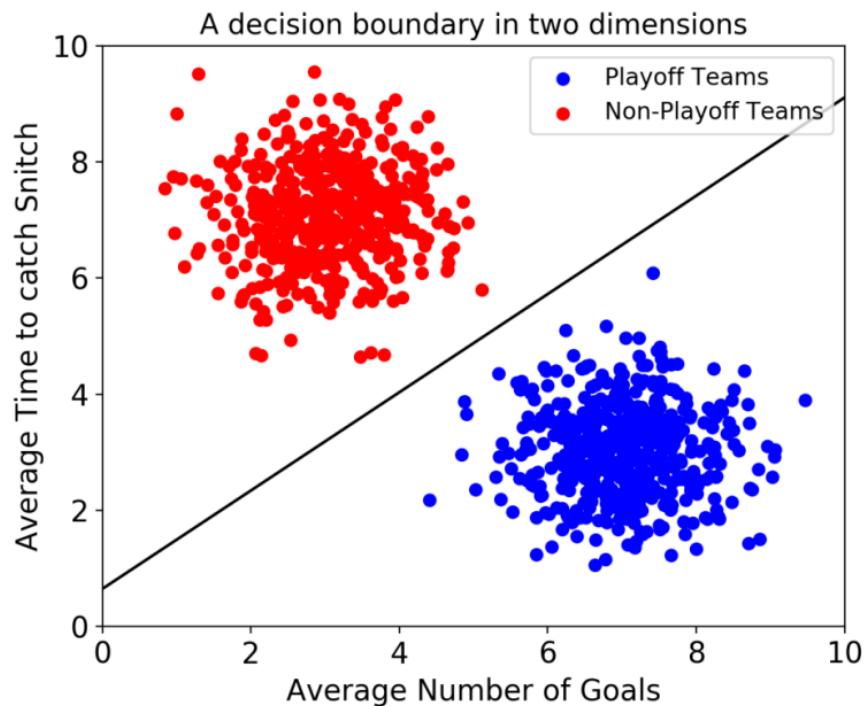
1-06. 파라미터(Parameter)와 하이퍼파라미터(Hyperparameter)

- 머신러닝 모델의 Parameter : 모델의 구성요소, 데이터로부터 학습되는 것
- 머신러닝 모델의 Hyperparameter : 모델 학습 과정에 반영되며, 학습을 시작하기 전에 미리 값을 결정하는 것
- Parameter : Linear Regression에서 $y = Wx + b$ 와 같은 직선 방정식의 W, b 를 찾는 것
- Hyperparameter : kNN에서 k 의 개수, Ridge, Lasso의 α 값, Learning Rate 등

(Support Vector Machine, SVM)

서포트 벡터머신 – 결정 경계(선)

- 데이터에 2개 속성(feature)만 있다면 결정 경계는 이렇게 간단한 선 형태



User guide: contents

User Guide: Supervised learning- Linear Models- Ordinary Least Squares, Ridge regression and classification, Lasso, Multi-task Lasso, Elastic-Net, Multi-task Elastic-Net, Least Angle

👉 https://scikit-learn.org/stable/user_guide.html



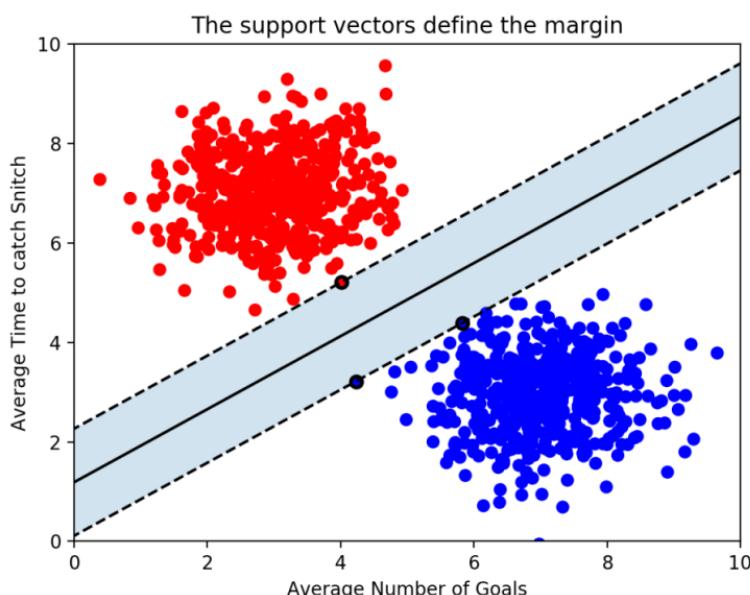
서포트 벡터머신(Support Vector Machine, SVM)

- 사용하기 편하면서도 높은 정확도를 보이는 데이터 분류를 위한 지도학습 머신러닝 알고리즘.
- 서포트 벡터를 사용해서 결정 경계를 정의하고, 분류되지 않은 점을 해당 결정 경계와 비교해서 분류.
- SVM의 중요 용어
 - 결정 경계(Decision Boundary) : 서로 다른 분류 값을 결정하는 경계.
 - 서포트 벡터(support vector) : 결정 경계선과 가장 가까이 맞닿은 데이터 포인트(클래스의 점들).
 - 마진(margin) : 서포트 벡터와 결정 경계 사이의 거리.
- SVM의 목표는 바로 이 마진을 최대로 하는 결정 경계를 찾는 것.

선은 결정 경계, 가장 가까운 점은 서포트 벡터, 결정 경계와 서포트 벡터 사이의 거리가 마진.

서포트 벡터머신 - 마진

- 마진(Margin)
 - 결정 경계와 서포트 벡터 사이의 거리를 의미한다.



- 가운데 실선이 하나 그어져 있는데, 이게 바로 '결정 경계'
- 그리고 그 실선으로부터 겹은 테두리가 있는 빨간점 1개, 파란점 2개까지 영역을 두고 점선을 그어놓았다. 이게 바로 '마진(margin)'
- 최적의 결정 경계는 마진을 최대화한다.
- 옆 그림에서는 x축과 y축 2개의 속성을 가진 데이터로 결정 경계를 그었는데, 총 3개의 데이터 포인트(서포트 벡터)가 필요했다. 즉, n개의 속성을 가진 데이터에는 최소 n+1개의 서포트 벡터가 존재한다는 걸 알 수 있다.

고차원(예: 4차원)은 알 수 없으니 초평면이라고 한다.

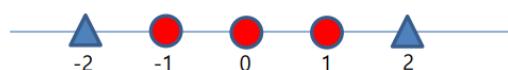
결정 경계도 단순한 평면이 아닌 고차원
이 될 텐데 이를 “초평면(hyperplane)”
이라고 부른다.

SVM에선 1차원으로는 분류 알고리즘에 적용할 수 있는가?

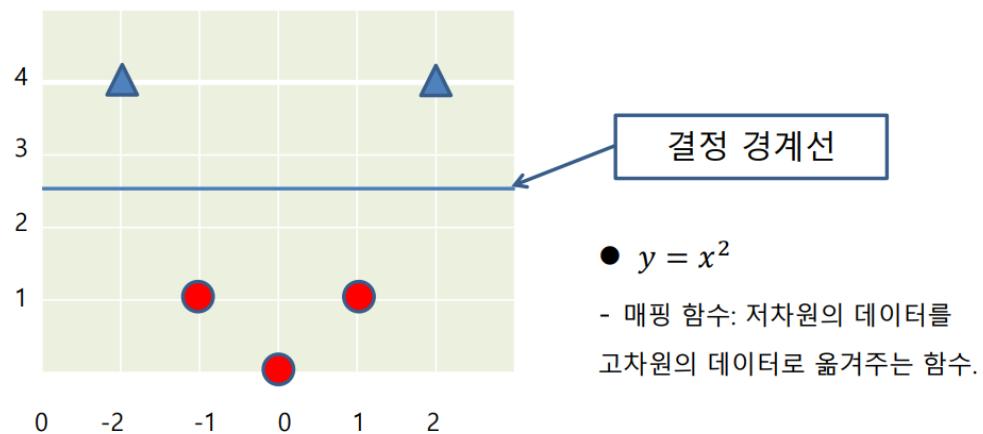
기존 데이터에 Trick을 씀.

서포트 벡터머신 – 커널 트릭

- ▶ 1차원의 데이터 결정 경계 찾기



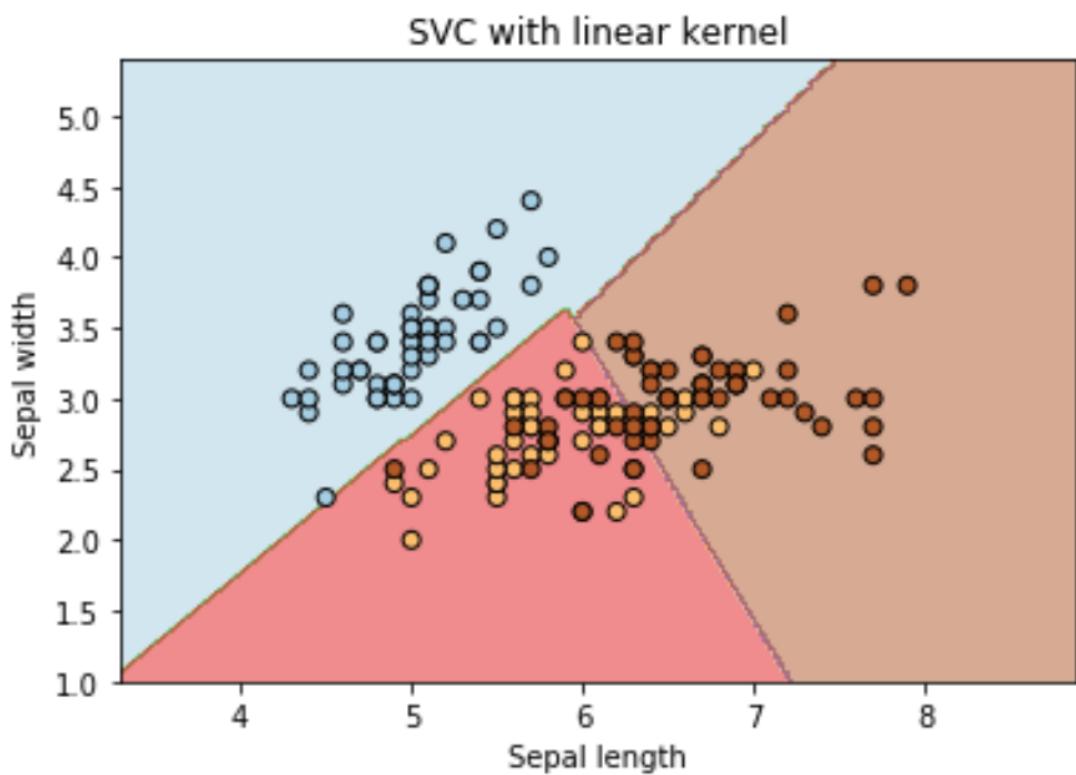
- ▶ 2차원 공간으로 옮겨진 데이터 결정 경계 찾기

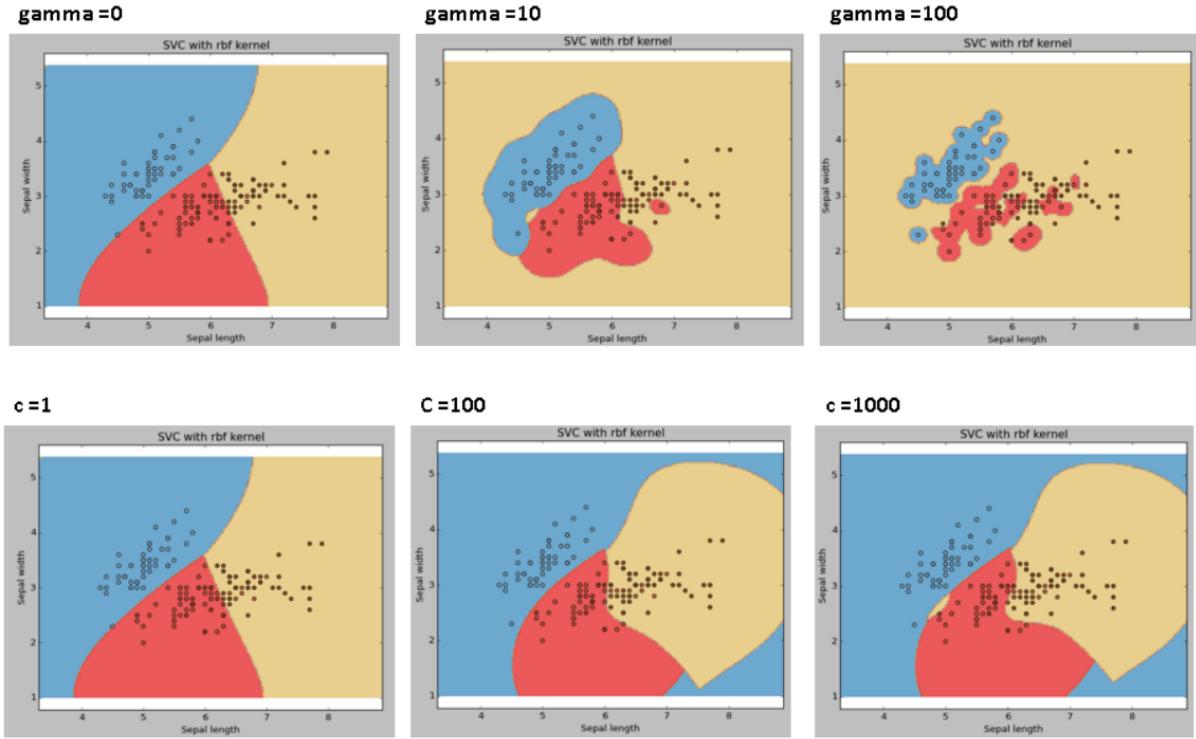




Petal : 꽃잎
Sepal : 꽃받침

서포트 벡터머신 – 파라미터 튜닝





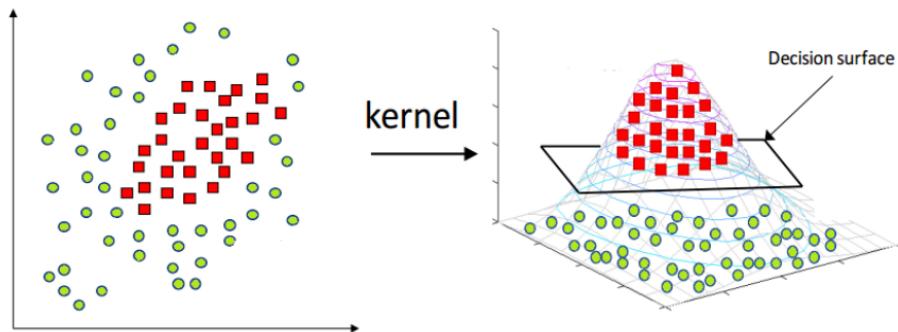
▼ 2-02. 데이터 전처리

- `sklearn.preprocessing.StandardScaler`
 - `fit(X_train)` : 전처리에 필요한 값 준비
 - `transform(X_train)` : 전처리 실행
 - `fit_transform(X_train)` : 전처리에 필요 값 준비 및 처리

서포트 벡터머신 – 커널 트릭

➤ 커널 트릭(kernel trick)

- 매핑 함수를 가지고 실제로 많은 양의 데이터를 저차원에서 고차원으로 옮기기에는 계산량이 너무 많아서 현실적으로 사용하기가 어려움.
- 데이터를 고차원으로 보내진 않지만 보낸 것과 동일한 효과를 줘서 매우 빠른 속도로 결정 경계선을 찾는 방법.
- 저차원에서 결정 경계를 찾지 못할 때 고차원으로 데이터(벡터)를 옮겨서 결정 경계를 찾는 방법.



과대적합 - 훈련용 데이터에 최적화돼서 만들어져 좋은 형태를 만들지 못한다. 차라리 오류 값을 하나 만들어서 범용적으로, 일반적으로 최적화된 결정 경계로 만들어주는 것이 더 좋다.

SVM에서

loss와 **cost**가 동일하게 사용된다.

➤ 파라미터 **C**(cost, 비용)는 허용되는 오류 양을 조절.

- 마진의 너비를 조정하는 파라미터.
- C 값이 클수록, 마진은 낮아지고, 학습 에러율은 감소하는 방향으로 결정 경계선을 만들고, 이를 **하드 마진(hard margin)**이라 부름 – 너무 높으면 과대 적합의 위험.
- C 값이 작을수록, 마진을 최대한 높이고, 학습 에러율을 증가시키는 방향으로 결정 경계선을 만들어서 **소프트 마진(soft margin)**을 만듦 – 너무 낮으면 과소 적합의 위험.

C 값이 큰 게 과대 적합.

C 값이 작은 게 과소 적합(오버 피팅).

감마가 너무 크면 학습 데이터에 너무 의존해서 오버피팅이 발생할 수 있다.

감마와 코스트값을 적절히 조절해서 최적의 값을 찾아가야한다. (경우의 수를 봐야한다.)

서포트 벡터머신 – 파라미터 튜닝

- SVM에서는 선형으로 분리할 수 없는 점들을 분류하기 위해 커널(kernel)을 사용.
- 커널(kernel)은 원래 가지고 있는 데이터를 더 높은 차원의 데이터로 변환.
 - 2차원의 점으로 나타낼 수 있는 데이터를
 - 다항식(polynomial) 커널은 3차원으로,
 - RBF 커널은 점을 무한한 차원으로 변환.
- RBF 커널에는 파라미터 감마(gamma)가 있음.
 - 커널의 데이터포인트 표준 편차를 결정하는 조절 변수.
 - 감마가 너무 크면 학습 데이터에 너무 의존해서 오버피팅이 발생할 수 있다.
- 대부분의 머신러닝 지도 학습 알고리즘은 학습 데이터 모두를 사용하여 모델을 학습.
 - SVM에서는 결정 경계를 정의하는 게 결국 서포트 벡터이기 때문에, 데이터 포인트 중에서 서포트 벡터만 잘 골라내면 나머지 슬 데 없는 수많은 데이터 포인트들을 무시할 수 있음.
 - 그래서 매우 빠르다.

feature가 많을수록 정확도가 높아진다.

SVM은 코스트와 감마를 파라미터로 조절 가능하다.



Find Open Datasets and Machine Learning Projects | Kaggle

Download Open Datasets on 1000s of Projects + Share Projects on One Platform. Explore Popular Topics Like Government, Sports, Medicine, Fintech, Food, More. Flexible Data Ingestion.

[k https://www.kaggle.com/datasets](https://www.kaggle.com/datasets)

Titanic dataset

Titanic dataset

Gender submission and test file merged

[k https://www.kaggle.com/datasets/brendan45774/test-file](https://www.kaggle.com/datasets/brendan45774/test-file)



<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/6f97a00e-3b5d-41e3-b11b-2848e6e3fa93/titanic.zip>

타이타닉 사건 생존에 대한 데이터분석.

데이터에 대한 높은 이해도.

면접 단골 질문: 타이타닉 분석 셋을 가지고 분석 해봤느냐?

라이브러리 임포트

- 실습에 필요한 라이브러리 임포트.

```
[11]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

데이터 불러오기(학습데이터, 테스트 데이터)_20221114

- 데이터 분석 단계에서 생성한 농구 선수 포지션 예측하기의 학습 데이터 및 테스트 데이터를 로드.

```
[2]: train = pd.read_csv("data/csv/basketball_train.csv")
test = pd.read_csv("data/csv/basketball_test.csv")
```

SVM 최적의 파라미터 찾기

- 1. C(Cost, 비용) = loss : 결정경계선의 마진을 결정하는 파라미터.
- 2. gamma : 커널의 데이터포인트의 표준 편차(sd)를 결정하는 조절 변수.

sklearn에서 제공하는 GridSearchCV 사용하시면, 손쉽게 최적의 cost, gamma를 구할 수 있음.

```
[8]: from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

```
[15]: 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=nfolds) # GridSearchCV은 험조자료형 # cv - cross validation # SVC - Classification
    clf.fit(X, y) # 2개 이상의 feature를 가지고 있을 때는 대문자 X # y는 정답 데이터
    print(clf.best_params_)

    return clf
```

```
[16]: X_train = train[['GP', 'BLK']]
y_train = train[['Pos']]

# 최적의 파라미터를 sklearn의 GridSearchCV()를 통해 구한다.
clf = svc_param_selection(X_train, y_train.values.ravel(), 10)
{'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
```