



"Facial Recognition Technology using Linear Algebra"

박윤준
김주찬
백승엽

1. Title

2. Abstract

3. Introduction

4. Main contents of the project

4.0 Facial Recognition Technology

4.1 얼굴영역 추출

4.2 특징 추출

4.3 매칭 단계

5. Discussion or Conclusion

6. References

7. The role and contribution(%) of team members to the project

8. Python Program

2. Abstract



연구의 개요

- 얼굴인식에는 다양한 알고리즘이 적용되며, 그 중에서도 선형대수는 핵심적인 역할을 한다.
- 본 연구에서는 얼굴인식 기술동향을 살펴보고 얼굴 영역 추출, 특징 추출, 매칭을 포함한 얼굴인식의 전체적인 프로세스에 관해 논한다.
- 특징 추출에서는 PCA와 LDA 등의 방법을, 그리고 매칭을 통한 인식 단계에서는 SVM, DL, KNN 등을 소개한다.

핵심어

Face Recognition

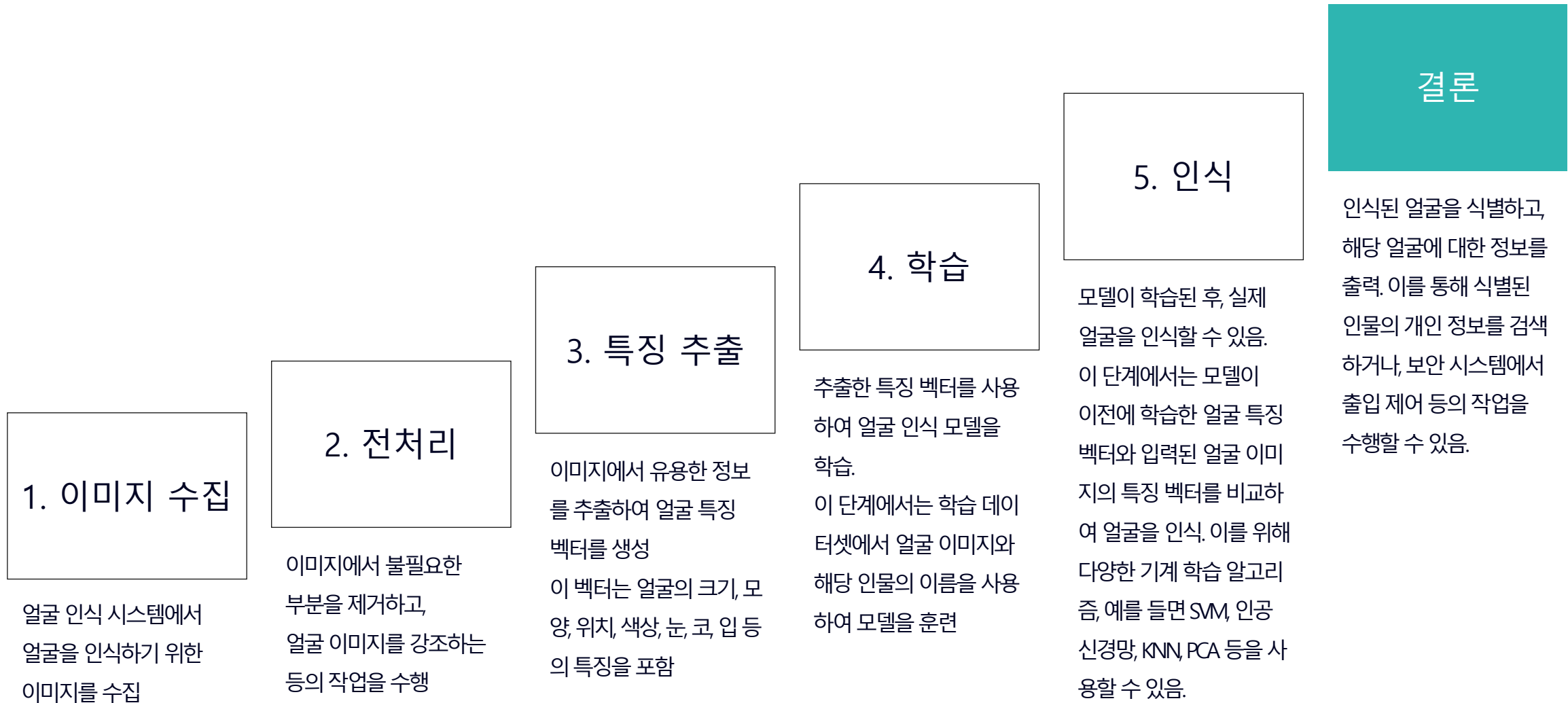
PCA # LDA

SVM # DL # KNN

3. Introduction



Facial recognition process



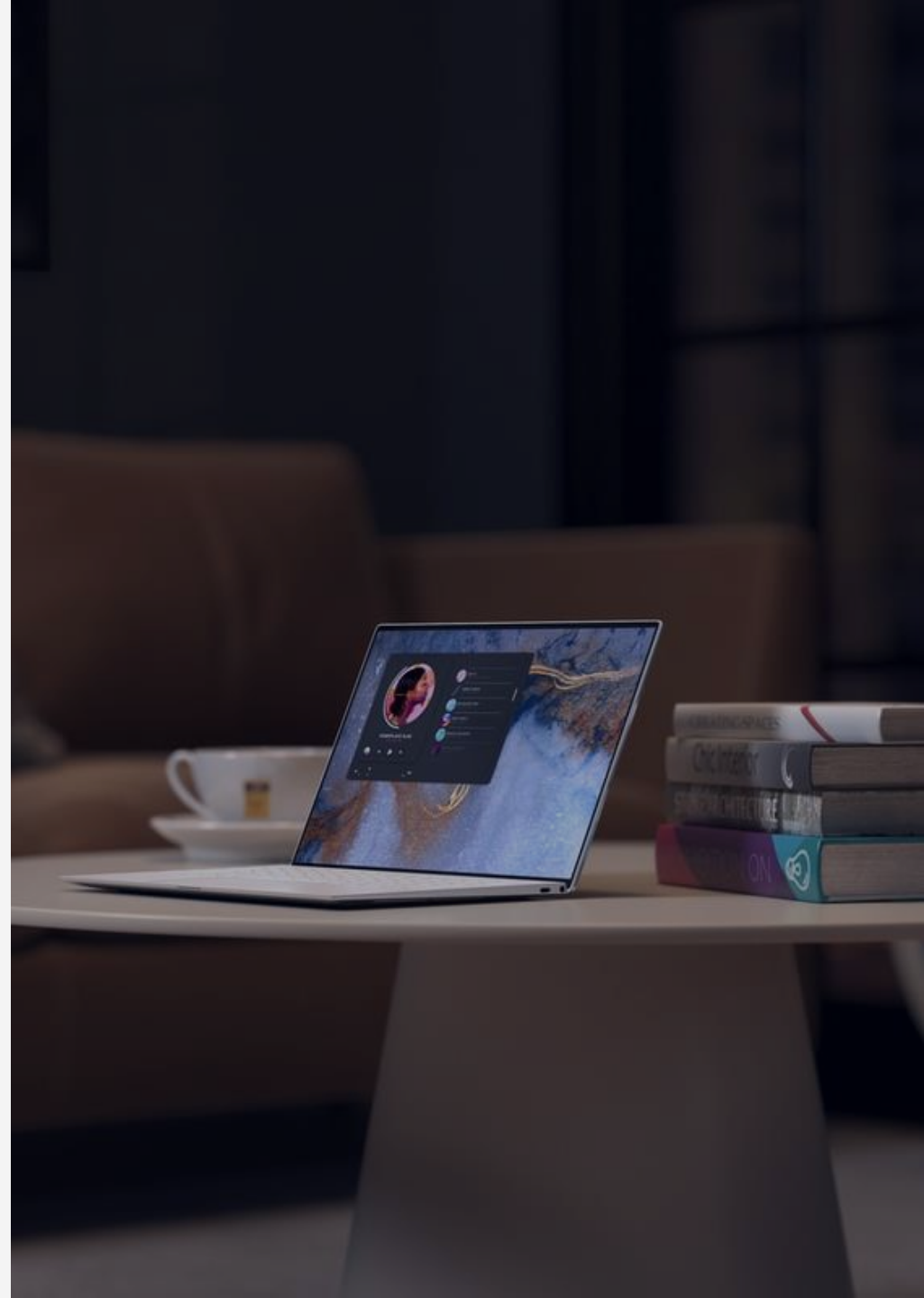
4. Main contents of the project

4.0 Facial Recognition Technology

4.1 얼굴영역 추출

4.2 특징 추출

4.3 매칭 단계



특징 추출 알고리즘

PCA

LDA

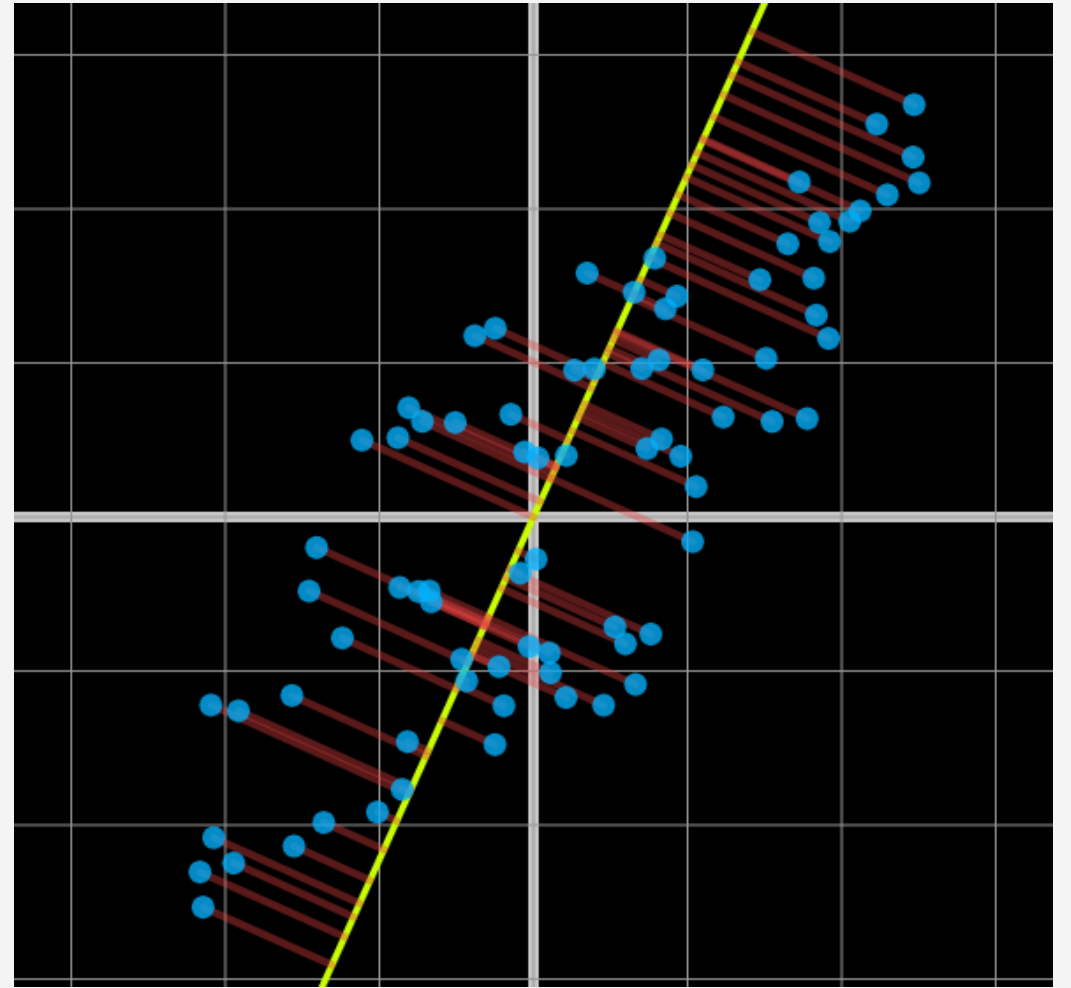
이미지 인식 알고리즘

KNN

SVM

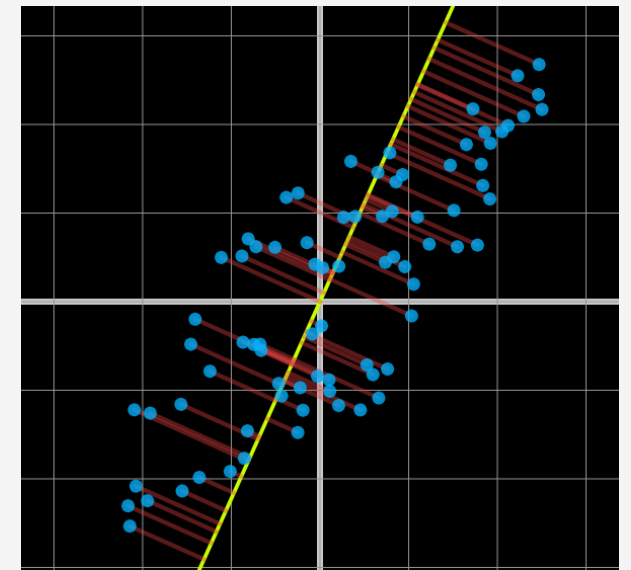
CNN

PCA(Principal Component Analysis)



PCA란 무엇인가?

- 고유 얼굴(Eigenfaces)을 생성
- 데이터들을 정사영 시켜 차원을 낮춘다면, 어떤 벡터에 데이터들을 정사영 시켜야 원래의 데이터 구조를 제일 잘 유지할 수 있을까?
- 얼굴 인식에서는 주로 고유한 얼굴 특징(예: 눈, 코, 입 등)을 추출하고 이를 사용하여 얼굴을 식별. 이를 위해 PCA(Principal Component Analysis)라는 선형대수학적 기법이 사용
- 다차원 데이터의 분산을 가장 잘 설명하는 방식으로 데이터를 변환하는 기법



P C A

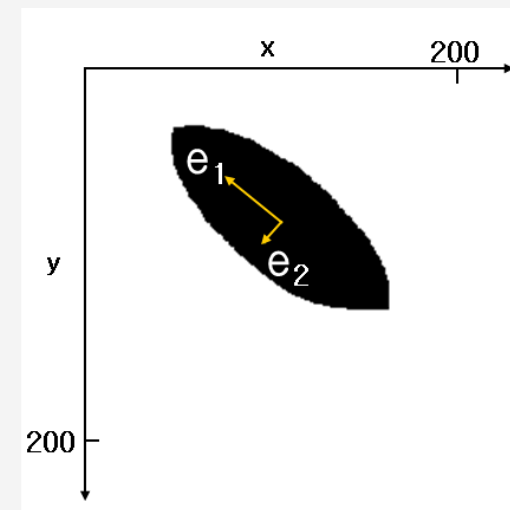
분포된 데이터들의 **주성분(Principal Component)**를 찾아주는 방법.

여기서 주성분이라 함은 그 방향으로 데이터들의 분산이 가장 큰 방향벡터를 의미.

2차원 좌표평면에 n 개의 점 데이터 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ 들이 타원형으로 분포되어 있을 때..

이 데이터들의 분포 특성을 2개의 벡터로 가장 잘 설명할 수 있는 방법은 무엇일까?

그건 바로, 그림에서와 같이 e_1, e_2 두 개의 벡터로 데이터 분포를 설명하는 것이다. e_1 의 방향과 크기, 그리고 e_2 의 방향과 크기를 알면 이 데이터 분포가 어떤 형태인지를 가장 단순하면서도 효과적으로 파악할 수 있음.



PCA 에 대한 수학적 접근

- PCA를 알기 위해서는 먼저 공분산 행렬 (covariance matrix)에 대해 알아야 한다. 공분산 행렬은 다음과 같이 정의된다. (m_x 는 x 의 평균, m_y 는 y 의 평균, $E[\cdot]$ 는 기대값이다)

$$\begin{aligned} cov(x,y) &= E[(x-m_x)(y-m_y)] \\ &= E[xy] - m_x m_y \end{aligned}$$

- 공분산 행렬 (covariance matrix)이란 데이터의 좌표 성분들 사이의 공분산 값을 원소로 하는 행렬로서 데이터의 i 번째 좌표 성분과 j 번째 좌표 성분의 공분산 값을 행렬의 i 행 j 열 원소값으로 하는 행렬이다. x 와 y 의 공분산은 x, y 의 흩어진 정도가 얼마나 서로 상관관계를 가지고 흩어졌는지를 나타낸다

- covariance

$$- cov(x,y) = E[(x-m_x)(y-m_y)]$$

- covariance matrix

$$- x = [x_1, \dots, x_n]^T : \text{sample data, } n\text{-차원 열벡터}$$

$$- C = E[(x-m_x)(x-m_x)^T] : n \times n \text{ 행렬}$$

$$- \langle C \rangle_{ij} = E[(x_i - m_{xi})(x_j - m_{xj})^T] : i\text{-번째 성분과 } j\text{-번째 성분의 공분산}$$

$$- C \text{ is real and symmetric}$$

$$C = \begin{pmatrix} C_{11} & \dots & C_{1n} \\ \vdots & \ddots & \vdots \\ C_{n1} & \dots & C_{nn} \end{pmatrix}$$

PCA 에 대한 수학적 접근

$$C = \begin{pmatrix} \text{cov}(x,x) & \text{cov}(x,y) \\ \text{cov}(x,y) & \text{cov}(y,y) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{n} \sum (x_i - m_x)^2 & \frac{1}{n} \sum (x_i - m_x)(y_i - m_y) \\ \frac{1}{n} \sum (x_i - m_x)(y_i - m_y) & \frac{1}{n} \sum (y_i - m_y)^2 \end{pmatrix}$$

PCA

- C : covariance matrix of x
- C = PΣP^T (P: orthogonal, Σ: diagonal)

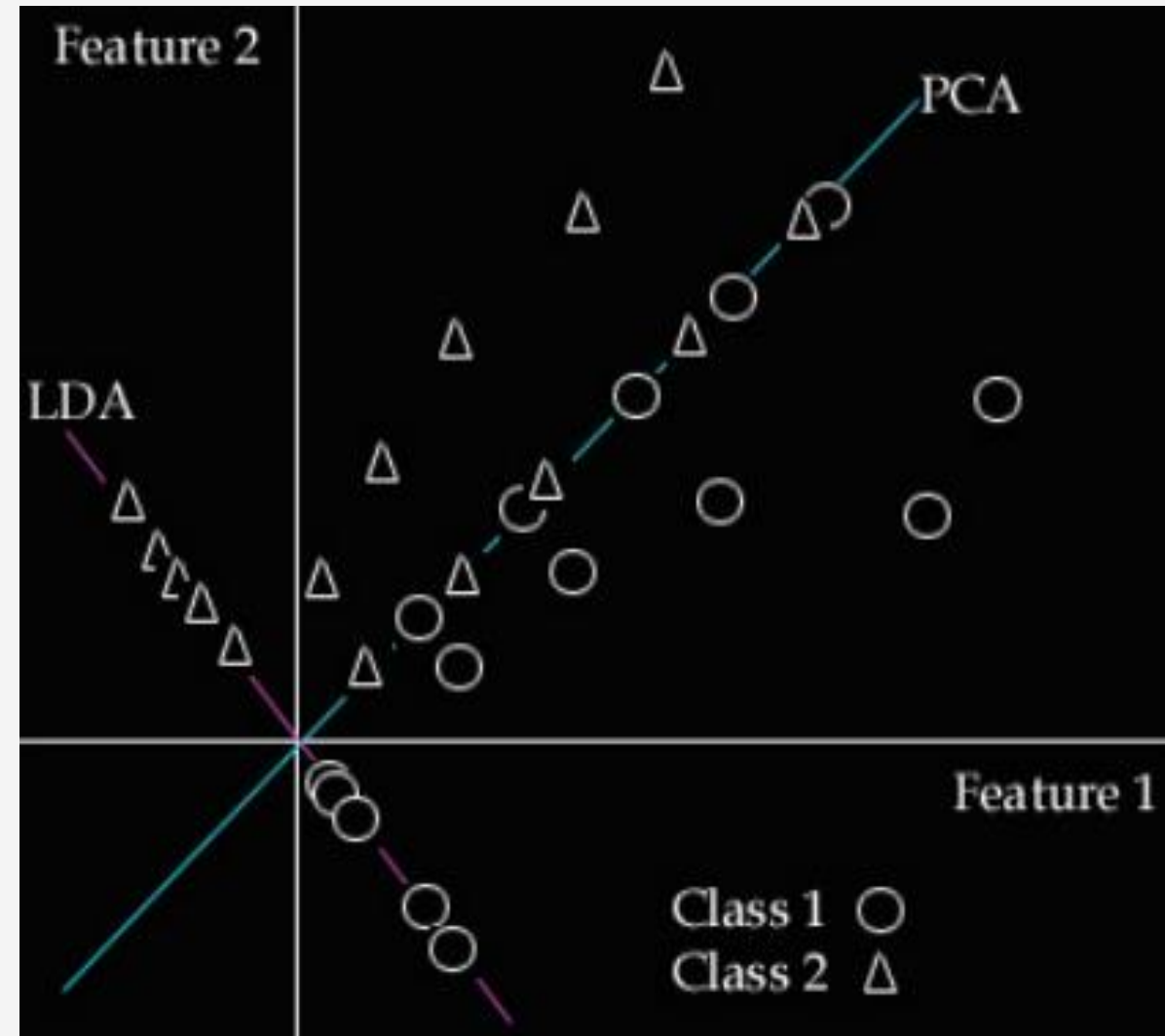
$$C = \begin{pmatrix} | & & | \\ e_1 & \dots & e_n \\ | & & | \end{pmatrix} \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \begin{pmatrix} \boxed{e_1^T} \\ \vdots \\ \boxed{e_n^T} \end{pmatrix}$$

- P : n×n orthogonal matrix
- Σ : n×n diagonal matrix
- Ce_i = λ_ie_i
 - e_i : eigenvector of C, direction of variance
 - λ_i : eigenvalue, e_i 방향으로의 분산
 - λ₁ ≥ ... ≥ λ_n ≥ 0
- e₁: 가장 분산이 큰 방향
- e₂: e₁에 수직이면서 다음으로 가장 분산이 큰 방향
- e_k: e₁, ..., e_{k-1}에 모두 수직이면서 가장 분산이 큰 방향

- 예를 들어, 2차원 데이터 n개가 (x1, y1), (x2, y2), ..., (xn, yn)와 같이 있다면 이 데이터들의 공분산 행렬은 다음과 같이 계산된다.

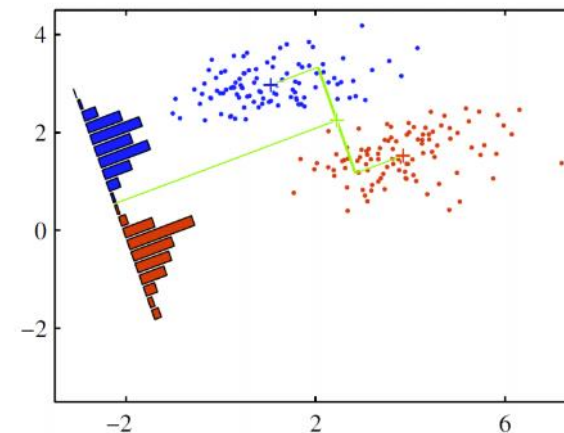
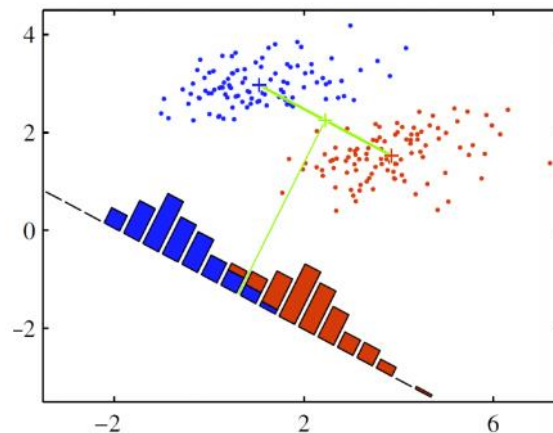
- PCA 는 입력 데이터들의 공분산 행렬(covariance matrix)에 대한 고유값분해(eigendecomposition)로 볼 수 있다. 이 때 나오는 고유벡터가 주성분 벡터로서 데이터의 분포에서 분산이 큰 방향을 나타내고, 대응되는 고유값(eigenvalue)이 그 분산의 크기를 나타낸다.

LDA (Linear Discriminant Analysis)



LDA란 무엇인가?

- LDA는 다차원 데이터를 차원 축소하고 분류 문제를 해결하는 데 사용되는 선형대수학 기반의 기계 학습 알고리즘이다.
- LDA는 입력 데이터 세트를 저차원 공간으로 투영(project)해 차원을 축소하는 기법이고, 지도학습의 분류(Classification)에서 사용된다.
- 사영(projection)한 후에 두 범주를 잘 구분할 수 있는 직선을 찾는 걸 목표로 합니다. 모델 이름에 linear라는 이름이 붙은 이유이기도 합니다.



L D A

LDA는 PCA와 유사하게 입력 데이터 세트를 저차원 공간으로 투영(project)해 차원을 축소하는 기법이고, 사영(projection)한 후에 두 범주를 잘 구분할 수 있는 직선을 찾는 걸 목표로 한다.

얼굴 인식 알고리즘에서 PCA기법과 LDA기법을 함께 사용하는 이유

- PCA 방법은 어떤 집단을 잘 축약해서 표현하는 데는 유용하나 집단 간을 잘 분리해서 나타내지는 못하는 단점이 있다.
- 얼굴인식을 위해서는 잘 축약해서 표현하는 것도 중요하지만 더 중요한 것은 identity가 다른 얼굴을 잘 분리해서 표현하는 방법일 것이다.

LDA 방법은 scatter matrix를 구성하는 과정에서 클래스 내부의 분산은 최소화하고, 클래스 간의 분산을 최대화 시키는 방향으로 선형 변환을 시킨 후 scatter matrix를 학습하는 방법이다. LDA란 바로 그런 identity가 다른 집단을 집단간의 분리가 잘 되게 표현하도록 만들어진 방법이다.

- 인식에서 쓰일 때 PCA 방법의 가장 큰 단점은 영상의 변화가 identity의 변화 때문인지 아니면 identity 이외에 조명이나 표정의 변화 때문인지 구분하지 못하는 경우가 있다. LDA 방법은 바로 이런 identity의 변화와 그 밖의 다른 요인에 의한 변화를 구분할 수 있도록 하는 방법이다.

LDA 에 대한 수학적 접근

$$y = \vec{w}^T \vec{x}$$

$$m_1 = \frac{1}{N_1} \sum_{n \in C_1} x_n$$

$$m_2 = \frac{1}{N_2} \sum_{n \in C_2} x_n$$

- p차원의 입력벡터 x (변수 p개)를 w 라는 벡터(축)에 사영시킨 후 생성되는 1차원 상의 좌표값(스칼라)를 y 라고 정의한다. 각각 N_1 개와 N_2 개의 관측치를 갖는 C_1 과 C_2 두 범주에 대해 원래 입력공간 (2차원)에서 각 범주의 중심(평균) 벡터도 m_1, m_2 라고 정의한다.

$$m_2 - m_1 = w^T (m_2 - m_1)$$

$$m_k = w^T m_k$$

- 사영후 두 범주의 중심이 멀리 떨어져 위치하는 벡터 w 를 찾아야 한다. 옆 식은 각 중심과 w 와의 관계식이다.

$$s_k^2 = \sum_{n \in C_k} (y_n - m_k)^2$$

- 사영 후 각 범주에 속한 관측치들은 해당 범주 중심에 가까이 있을 수록 좋다. 다시 말해 분산이 작아야 한다는 것이다. 옆 식은 사영 후 분산에 관한 식이다.

LDA 기법에 대한 수학적 접근

$$J(w) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{w^T S_B w}{w^T S_W w}$$

$$S_B = (m_1 - m_2)(m_1 - m_2)^T$$

$$S_W = \sum_{n \in C_1} (x_n - m_1)(x_n - m_1)^T + \sum_{n \in C_2} (x_n - m_2)(x_n - m_2)^T$$

$$(w^T S_B w) S_W w = (w^T S_W w) S_B w$$

$$S_W w = \lambda S_B w$$

$$S_B^{-1} S_W w = \lambda w$$

• 두 범주의 중심은 최대화하고, 분산은 최소화하는 것이 LDA의 목표이다. 동시에 중심을 최대화 하고 분산을 최소화 하는 방법은 두 범주 중심을 분자, 두 범주의 분산을 분모에 넣고 이 식을 최대화하는 것이다. 이를 행렬 형태의 목적함수로 나타내면 $J(w)$ 와 같다.

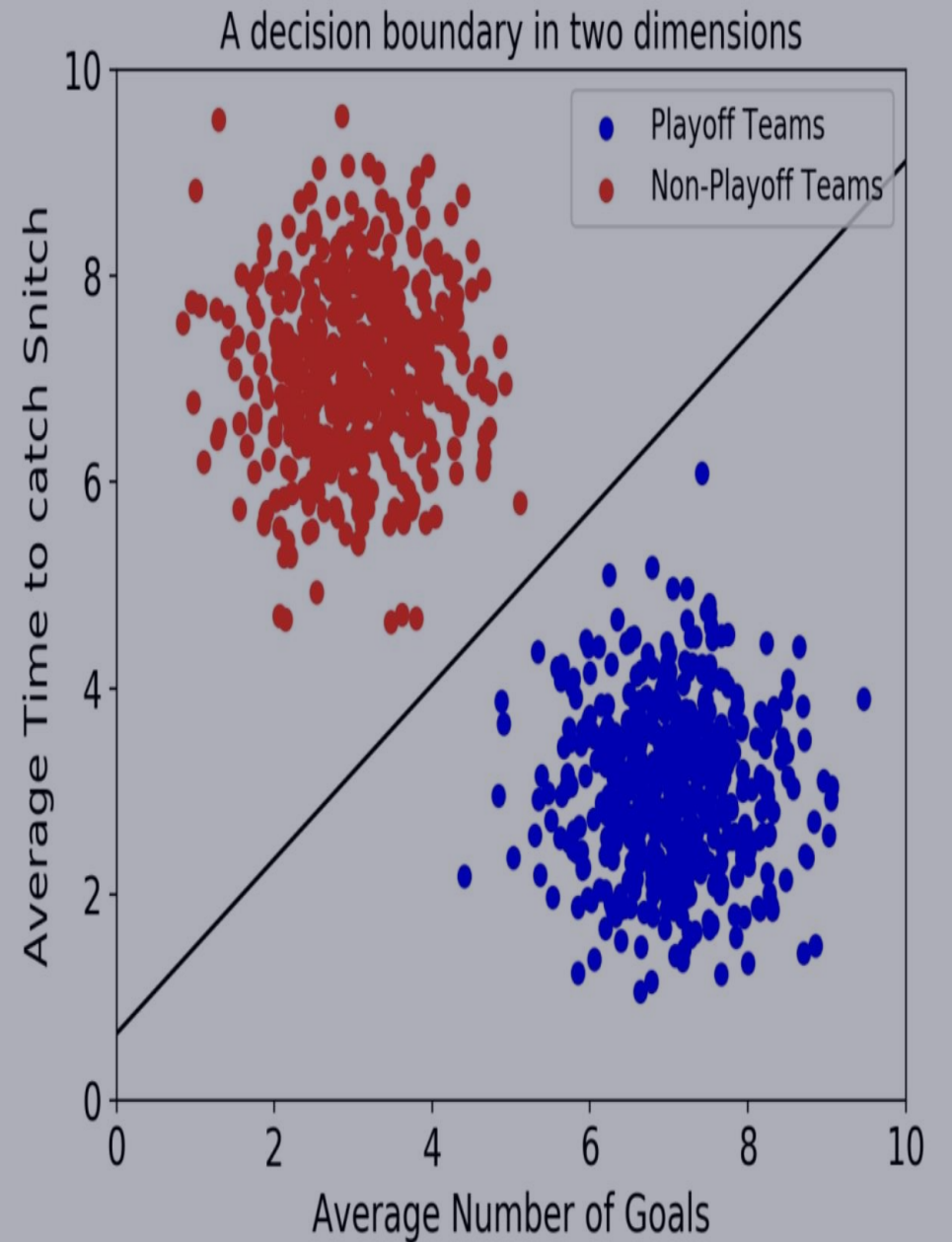
• 목적함수 $J(w)$ 은 w 에 대해 미분한 값이 0이 되는 지점에서 최댓값을 가진다.

• 위 식에서 괄호 안을 계산해보면 스칼라가 된다. 왜냐하면 각각의 차원수를 고려하면 $(1 \times d) (d \times d) (d \times 1)$ 이 되기 때문이다. 위 식 양변을 좌변 괄호안의 스칼라값으로 나누어 약간 정리를 하면 다음과 같은 형태가 된다. 이 형태는 고유값, 고유벡터 구할 때 쓰이는 $AX = \lambda X$ 꼴이다. 즉 새로운 축 w 는 S_B 의 역행렬과 S_W 를 내적한 행렬의 고유벡터라는 뜻이다. 이 행렬을 고유값분해하여 우리가 가진 평균과 분산을 이용하여 새로운 축 w 를 구할 수 있다.

⇒ 새로운 데이터 (x') 가 주어지면 이를 w 와 내적해 각각의 스코어를 낼 수 있다. 그 스코어가 일정값보다 크면 C_1 범주, 작으면 C_2 범주로 분류를 하게 된다.

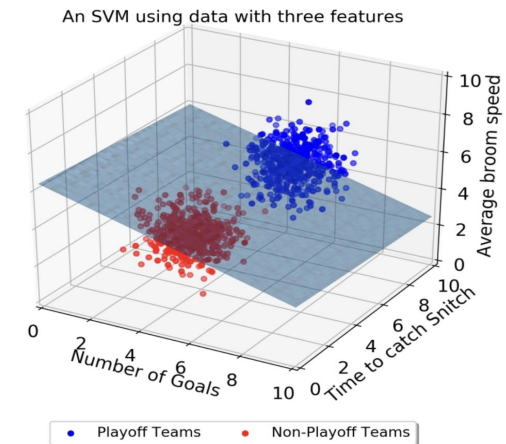
4.3.1 SVM

SVM(Support Vector Machine)



SVM이란 무엇인가?

- 결정 경계(Decision Boundary)
- 최적의 결정 경계(Decision Boundary)
- 마진(Margin)
- 이상치(Outlier)



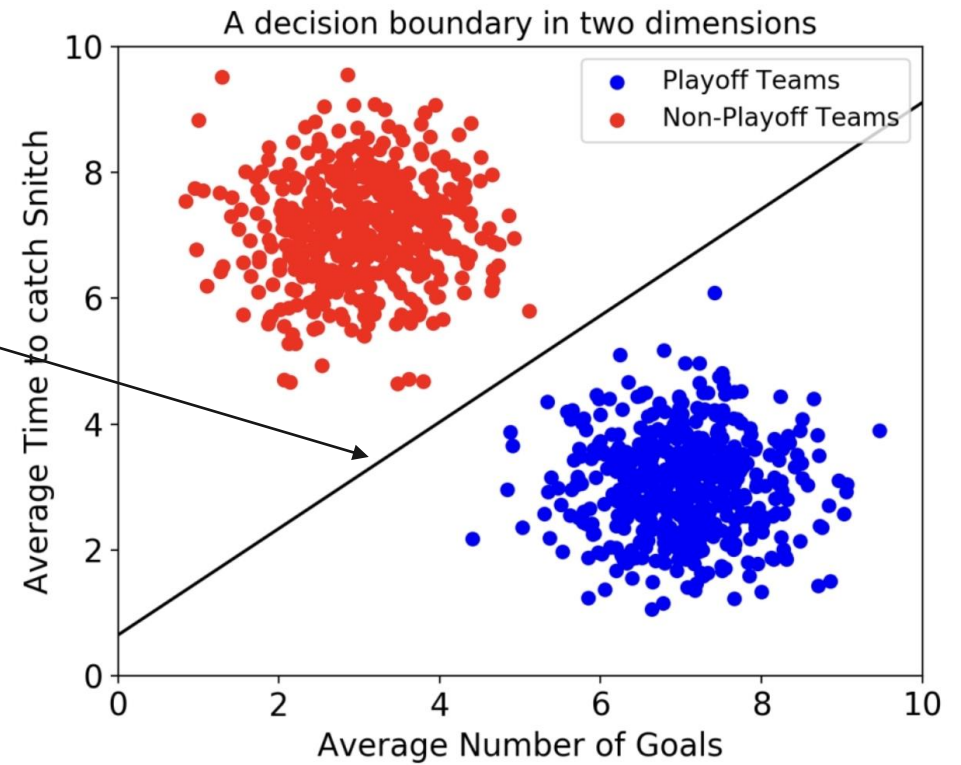
S V M

두 클래스로부터 최대한 멀리 떨어져 있는 **결정 경계**를 찾는 분류기로 특정 조건을 만족하는 동시에 클래스를 분류, 결정 경계를 통해 어느 쪽에 속하는지 판단하는 것으로, 선형이나 비선형 분류, 회귀, 이상치 탐색에도 사용할 수 있는 강력한 성능을 갖는 지도 학습 모델

결정 경계(Decision Boundary)

결정 경계?

데이터에 2개 속성(feature)만 있다면 **결정 경계**는
이렇게 간단한 선 형태가 될 것 입니다.

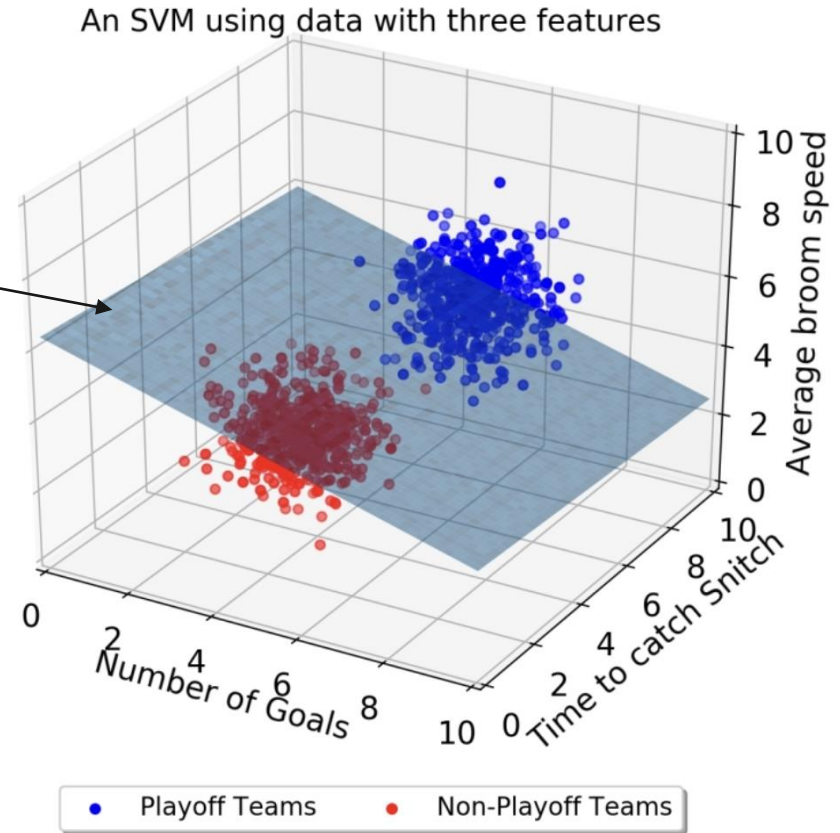


2개 속성

결정 경계(Decision Boundary)

속성 3개일 경우

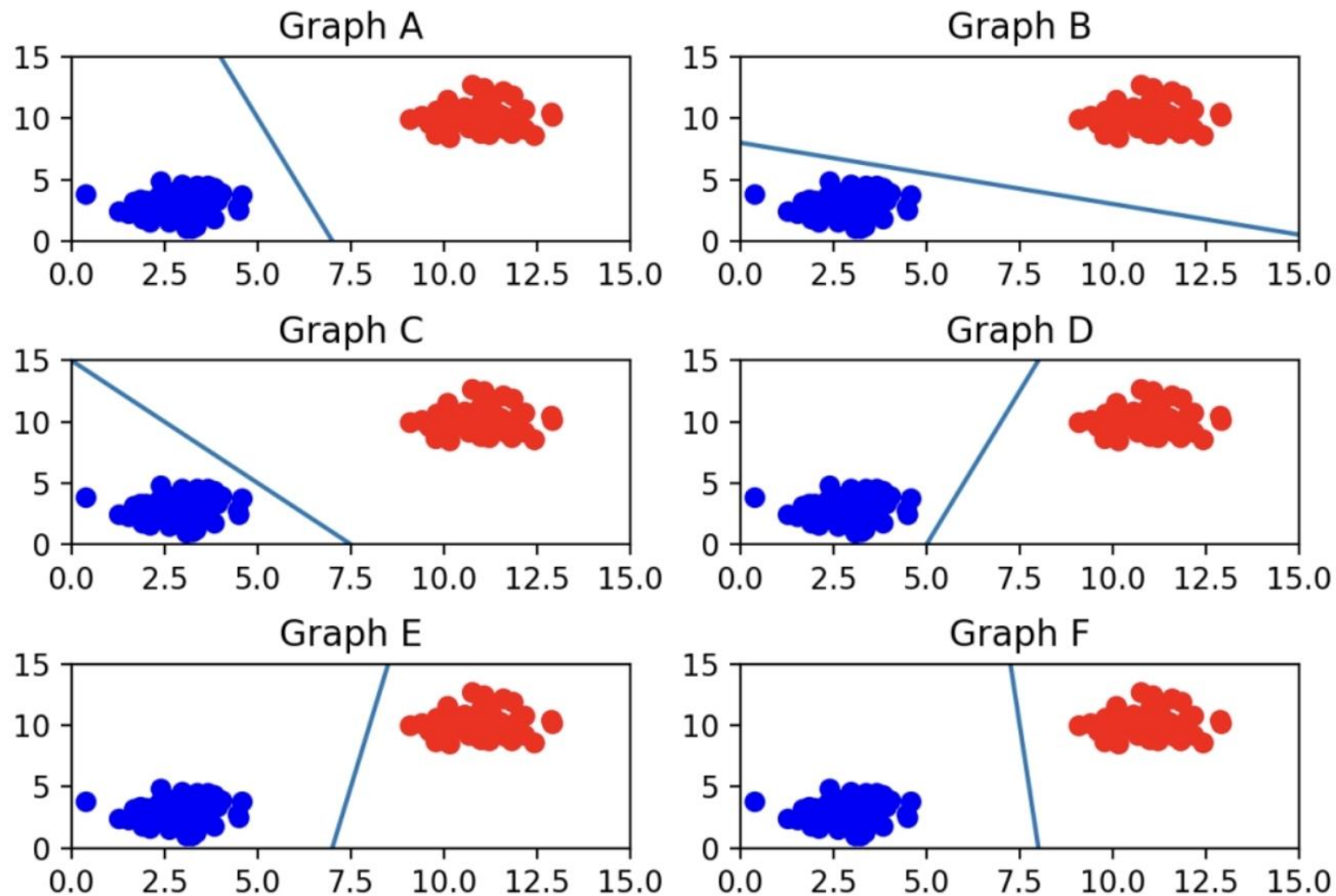
속성이 3개로 늘어난다면 이렇게 3차원으로 그려야 합니다.
그리고 이때의 결정 경계는 '선' 이 아닌 '평면'이 됩니다.
우리가 이렇게 시각적으로 인지할 수 있는 범위는
3차원 까지 입니다. 차원, 즉 속성의 개수가 늘어날수록
당연히 복잡해지고, 결정 경계도 단순한 평면이 아닌
고차원이 되는데 이를 초평면(hyperplane)이라고 합니다.



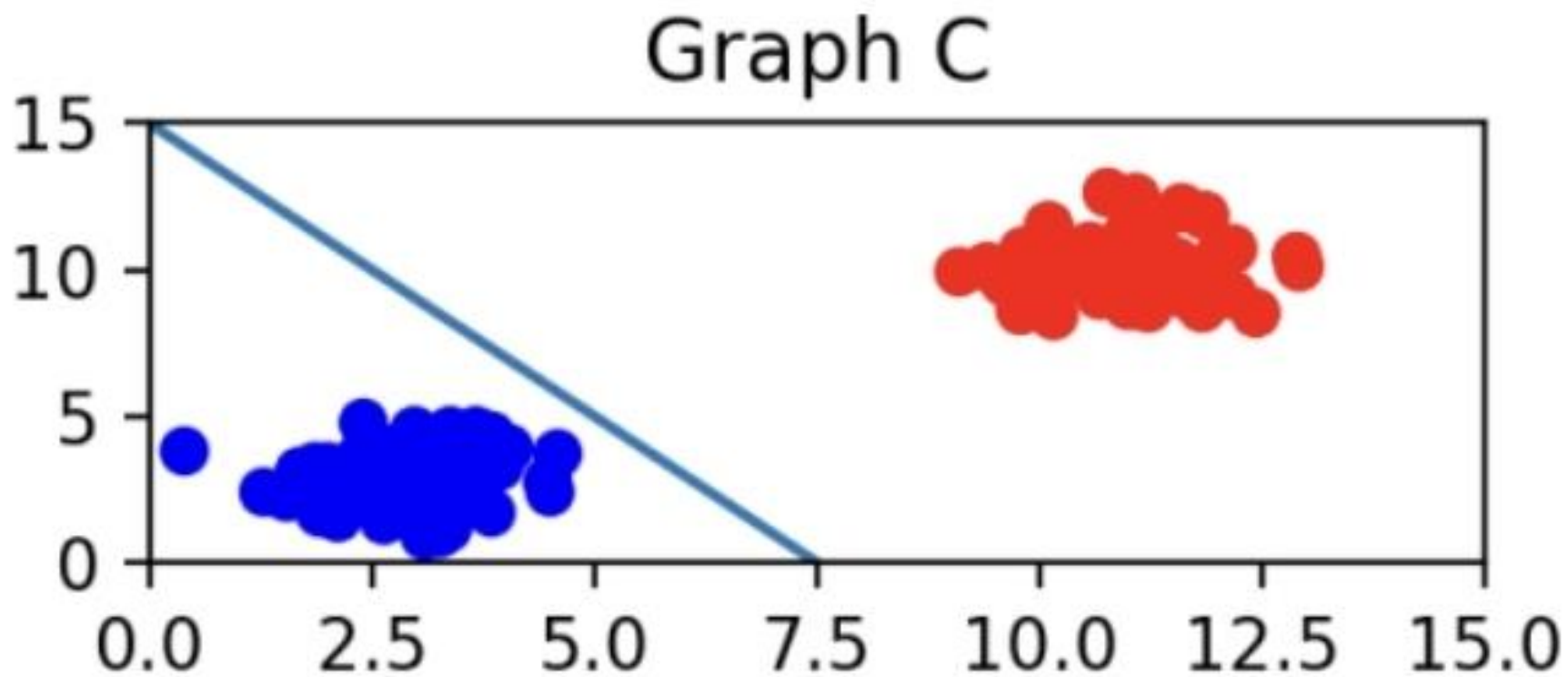
3개 속성

어떤 게 좋은 경계일까??

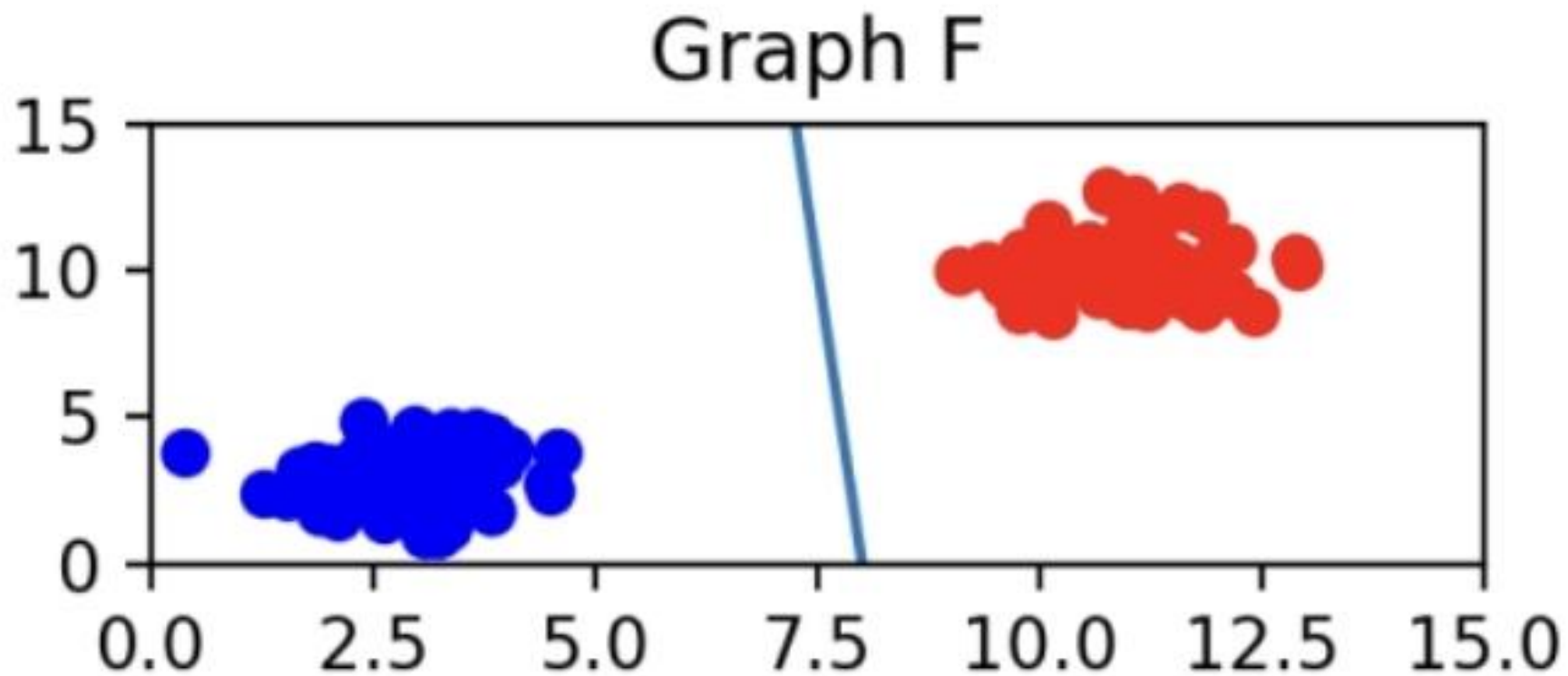
Different Decision Boundaries



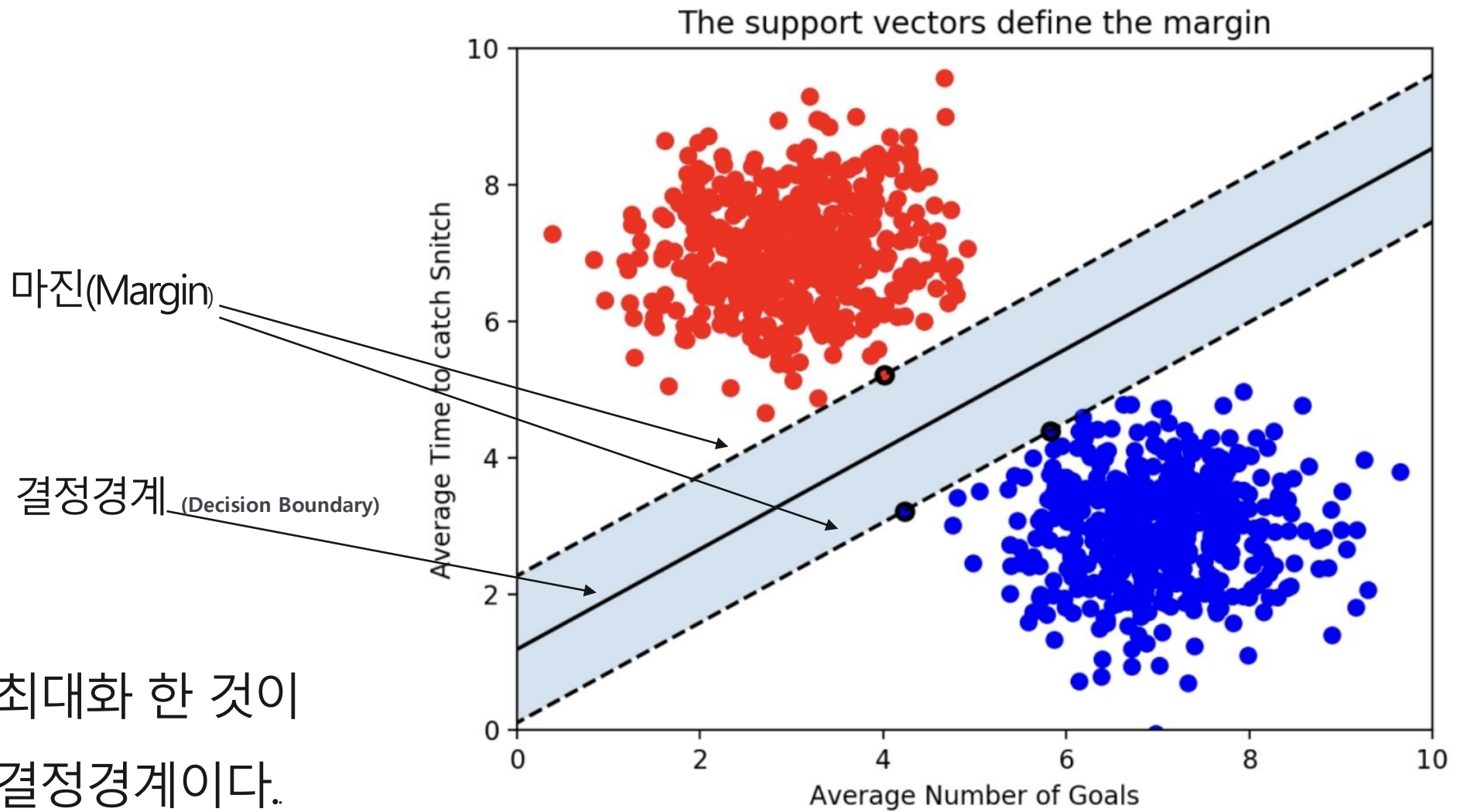
제일 부적절한 결정 경계



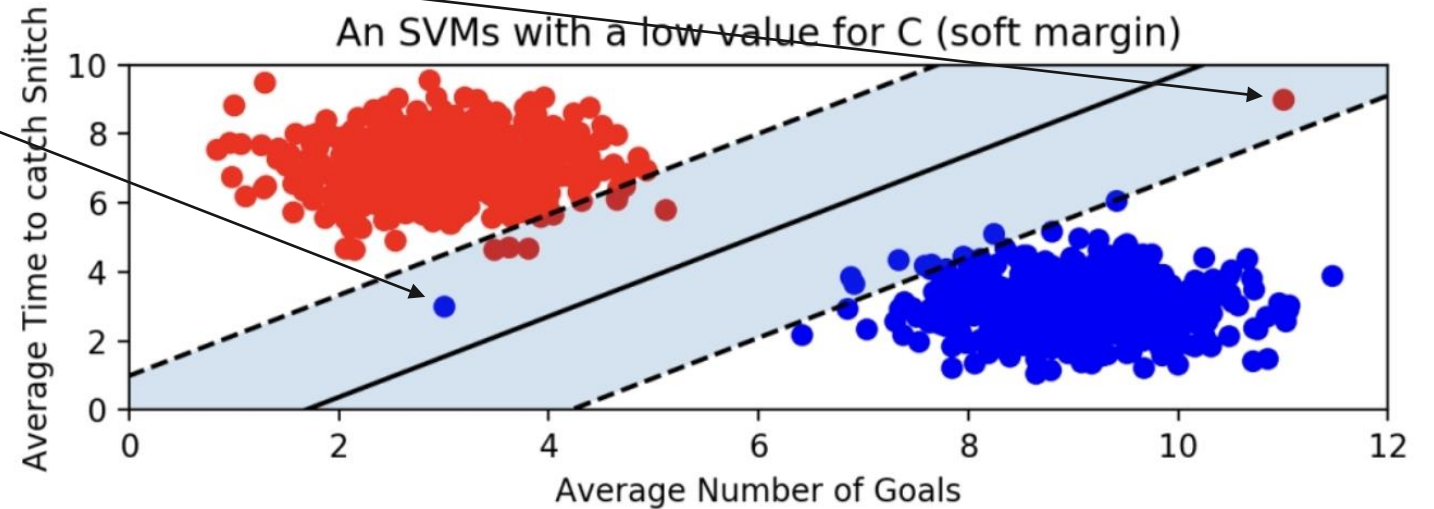
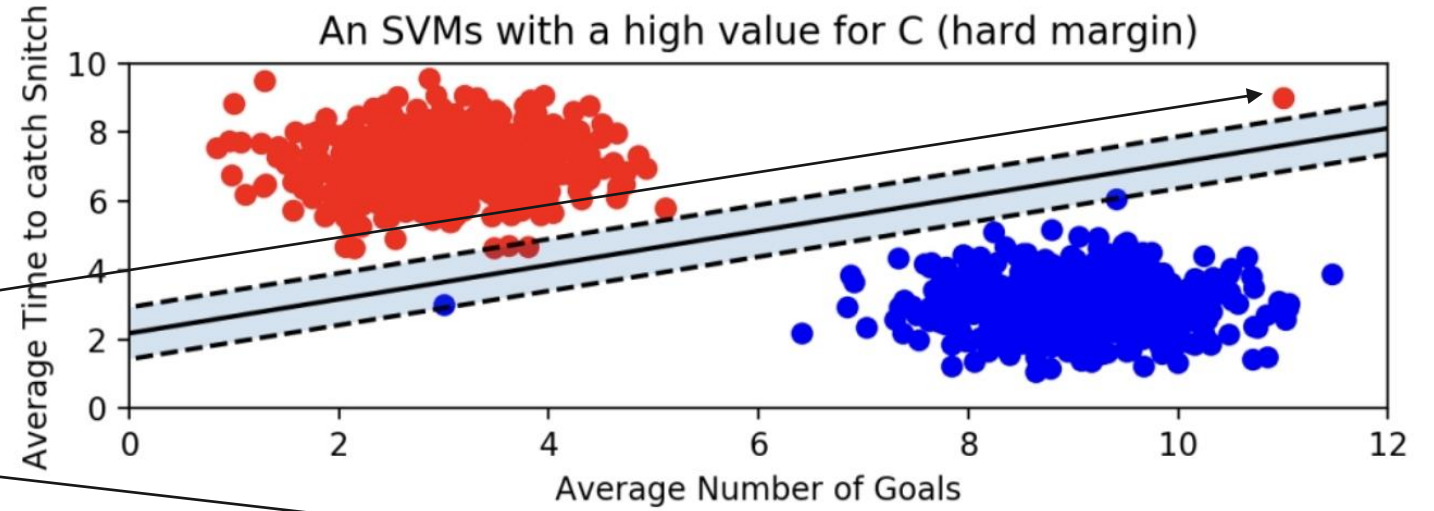
제일 적절한 결정 경계



마진(Margin)

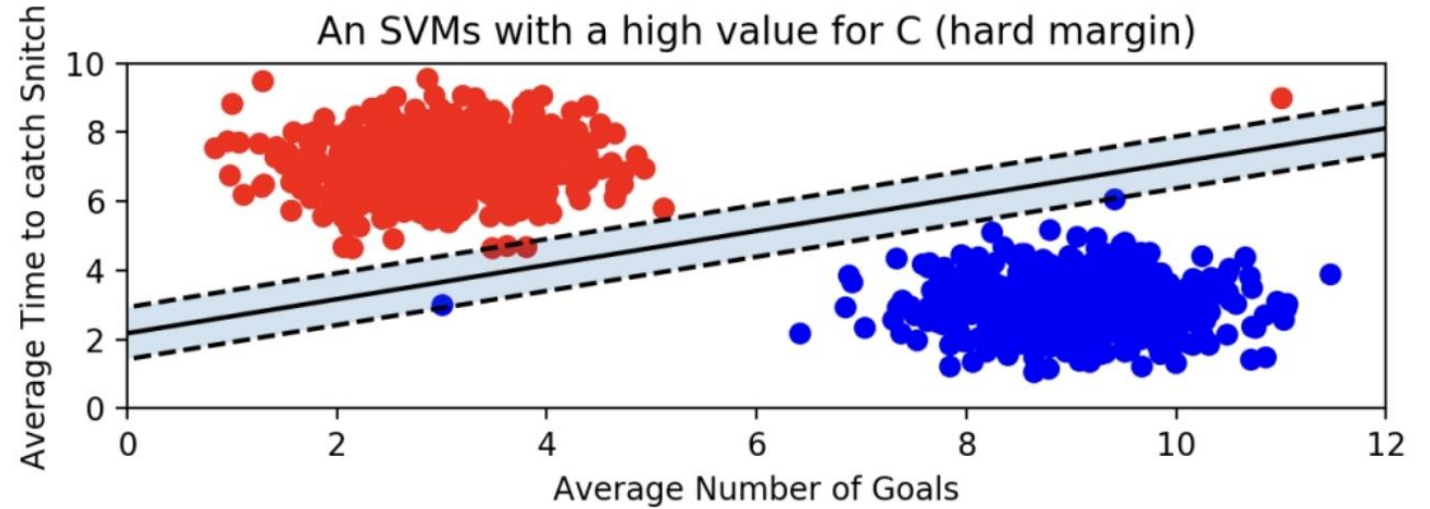


이상치(Outlier)



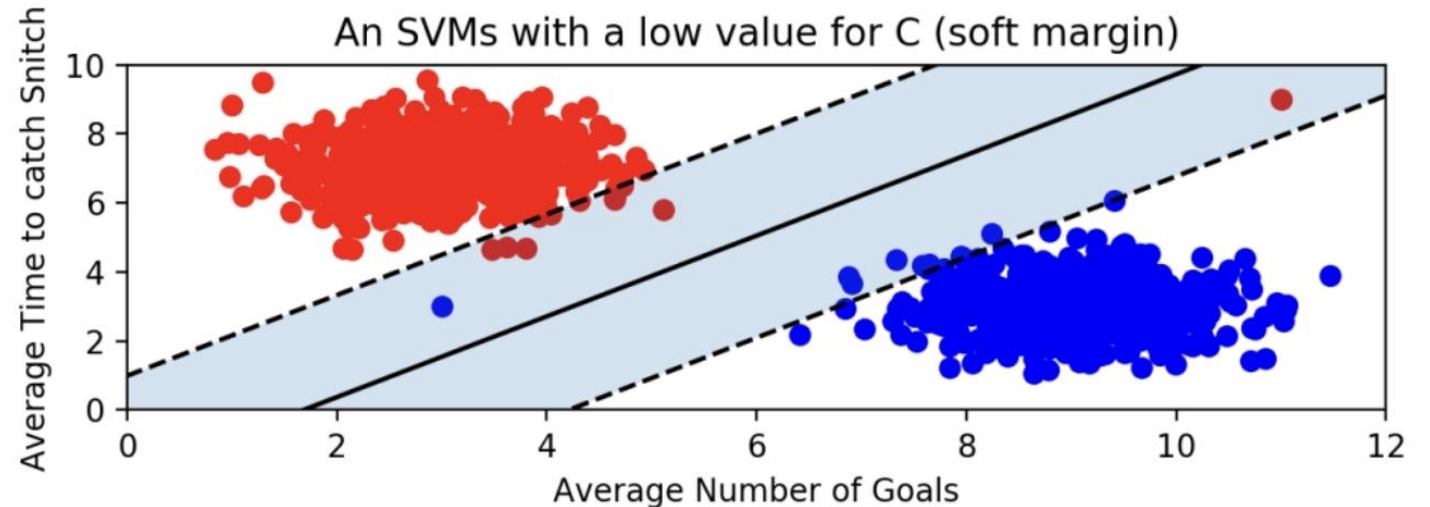
하드 마진(hard margin)

이상치를 허용하지 않고 기준을
까다롭게 한 마진



소프트 마진(soft margin)

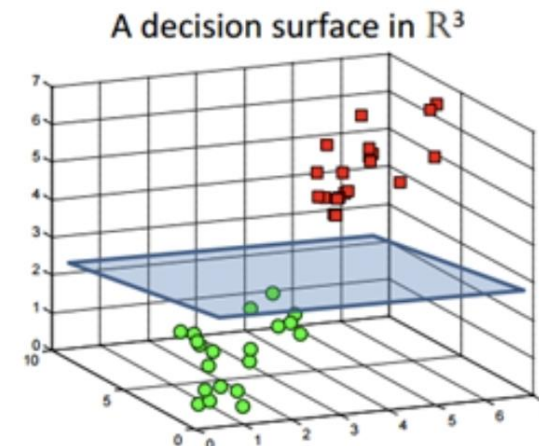
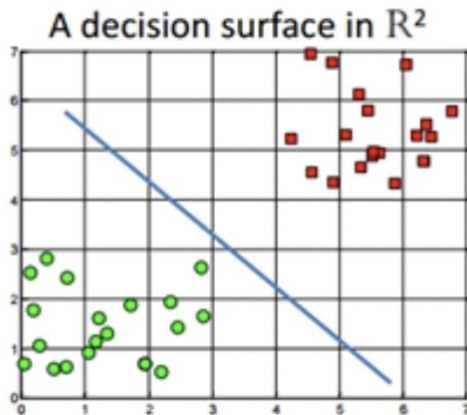
이상치가 어느정도 마진 안에 포함
되도록 기준을 너그럽게 한 마진



n 차원 공간에서의 초평면(hyperplane)은 $n-1$ 차원의 부분공간(subspace)를 의미합니다.

위에서 봤듯이, 3차원 공간에서는 2차원인 평면이 되고, 2차원 공간에서는 1차원인 선이 되는 걸 봤습니다.

n 차원에 대해 일반화 시키기 위해 3차원과 2차원의 초평면(hyperplane)의 방정식을 생각해봅시다.



법선 벡터 $\vec{v} = (a, b, c)$ 에 대해

원점과 거리가 d인 평면의 방정식은
 $ax+by+cz+d=0$

2차원의 경우는 $\vec{w} = (a, b)$ 에 대해 직선의 방정식은
 $ax+by+d=0$ 이고

식을 변형해 보면 $y = -\frac{a}{b}x - \frac{d}{b}$ 벡터 w와
수직임을 알 수 있고 법선 벡터와 임의의 실수를
이용하면 n차원 공간에서
초평면(hyperplane)을 쉽게 생각할 수 있습니다.

위와 같은 사실을 이용하면 n 차원의 실수 공간에서 초평면(hyperplane)을 밑에 그림처럼 정의 할 수 있습니다.

Definition 1. Hyperplane

Let a_1, a_2, \dots, a_n be scalars not all equal to 0. Then set consisting of all vectors

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$$

such that

$$a_1 x_1 + a_2 x_2 + \dots + a_n x_n = c$$

for c a constant is a subspace of \mathbb{R}^n called a hyperplane.

Decision rule는 새로운 입력 \vec{u} 에 대해 class가 +인지 -인지 결정하는 방법입니다.

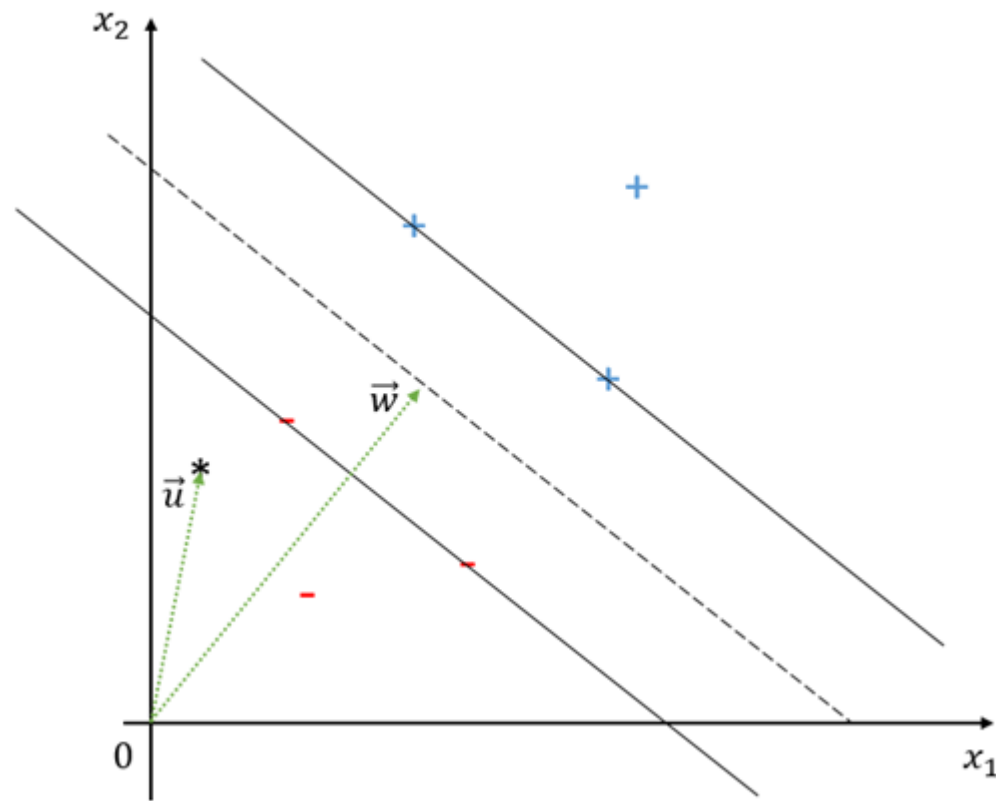
다음 그림과 같이 2차원 공간에 hyperplane과
법선 벡터 \vec{w} 가 있다고 하면

선 위에 있으면 + 아래 있으면 -로 분류하려고 하는데
decision rule을 다음과 같이 설정 할 수 있습니다.

$$\vec{w} \cdot \vec{u} \geq c \quad (c \text{는 임의의 상수})$$

약간 변형 시키면 다음과 같습니다.

$$\vec{w} \cdot \vec{u} + b \geq 0$$



positive samples x_+ negative samples x_-

최대한 가까운 positive samples와 negative samples로부터 동일하게 δ 만큼 떨어져야 하고,
그렇지 않으면 δ 보다 크게 떨어져야 한다. 따라서 다음식을 생각 할 수 있습니다.

$$\vec{w} \cdot \vec{x}_+ + b \geq \delta$$

\vec{w} 도 b 도 모르기 때문에

$$\vec{w} \cdot \vec{x}_+ + b \geq 1$$

$$\vec{w} \cdot \vec{x}_- + b \leq -\delta$$

다음과 같습니다

$$\vec{w} \cdot \vec{x}_- + b \leq -1$$

따라서 실선 위에 있게 되는

따라서 이렇게 표현할 수 있고

두 식을 합치면 이렇게 표현할 수 있습니다.

samples의 조건은 이렇게 됩니다.

$$y_i = \begin{cases} +1 & \text{for } x_+ \\ -1 & \text{for } x_- \end{cases}$$

$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1$$

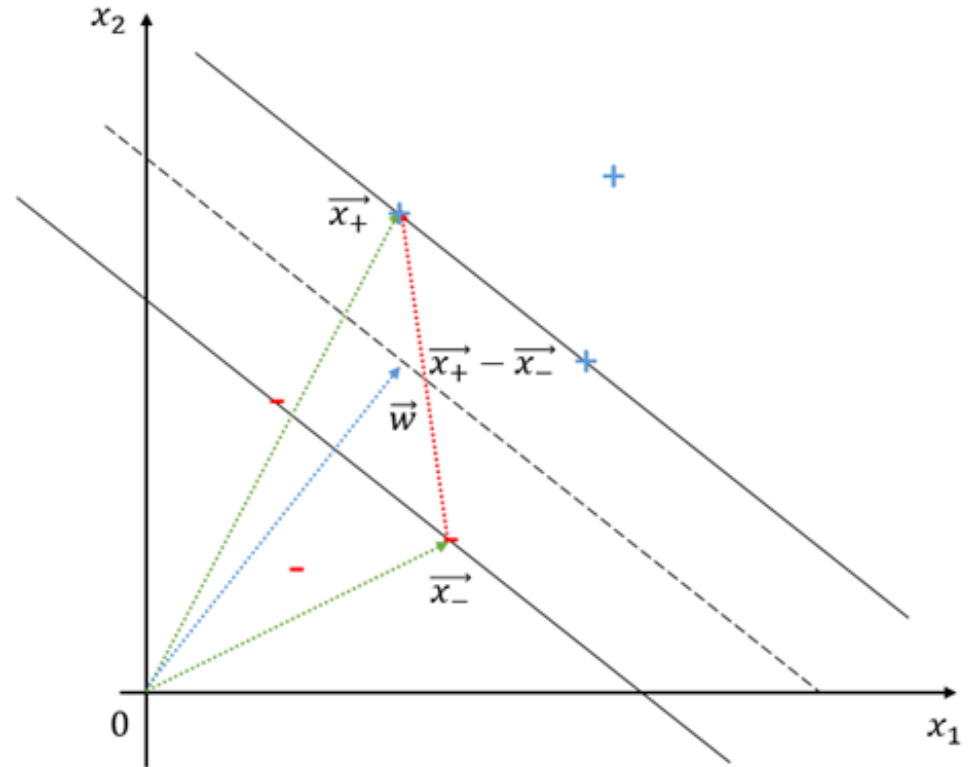
$$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 = 0$$

마진(Margin) 너비

원점으로부터 x_+ 와 x_- 에 대해서 \vec{x}_+ \vec{x}_- 라 하자

\vec{w} 는 선에 수직하기 때문에 \vec{w} 와 방향은 같고 크기는 1인 벡터를 이용하면 두 실선 사이의 거리를 구할 수 있다.
따라서 구하면 다음과 같습니다.

$$\frac{\vec{w}}{|\vec{w}|} \cdot (\vec{x}_+ - \vec{x}_-)$$



마진(Margin) 너비

$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 = 0$ 이식으로부터

x_i 가 positive sample이면 $y_i = 1$ 이므로

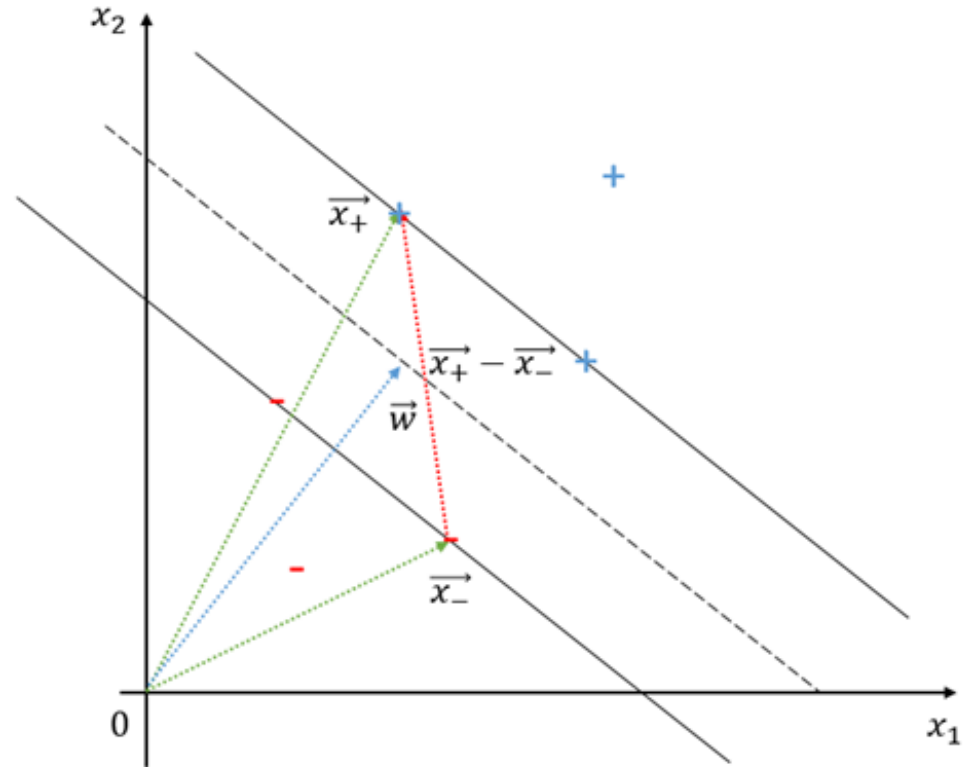
$$\vec{w} \cdot \vec{x} + b - 1 = 0$$

$$\Rightarrow \vec{w} \cdot \vec{x}_i = 1 - b$$

x_i 가 negative sample이면 $y_i = -1$ 이므로

$$\vec{w} \cdot \vec{x} + b + 1 = 0$$

$$\Rightarrow \vec{w} \cdot \vec{x}_i = -1 - b$$

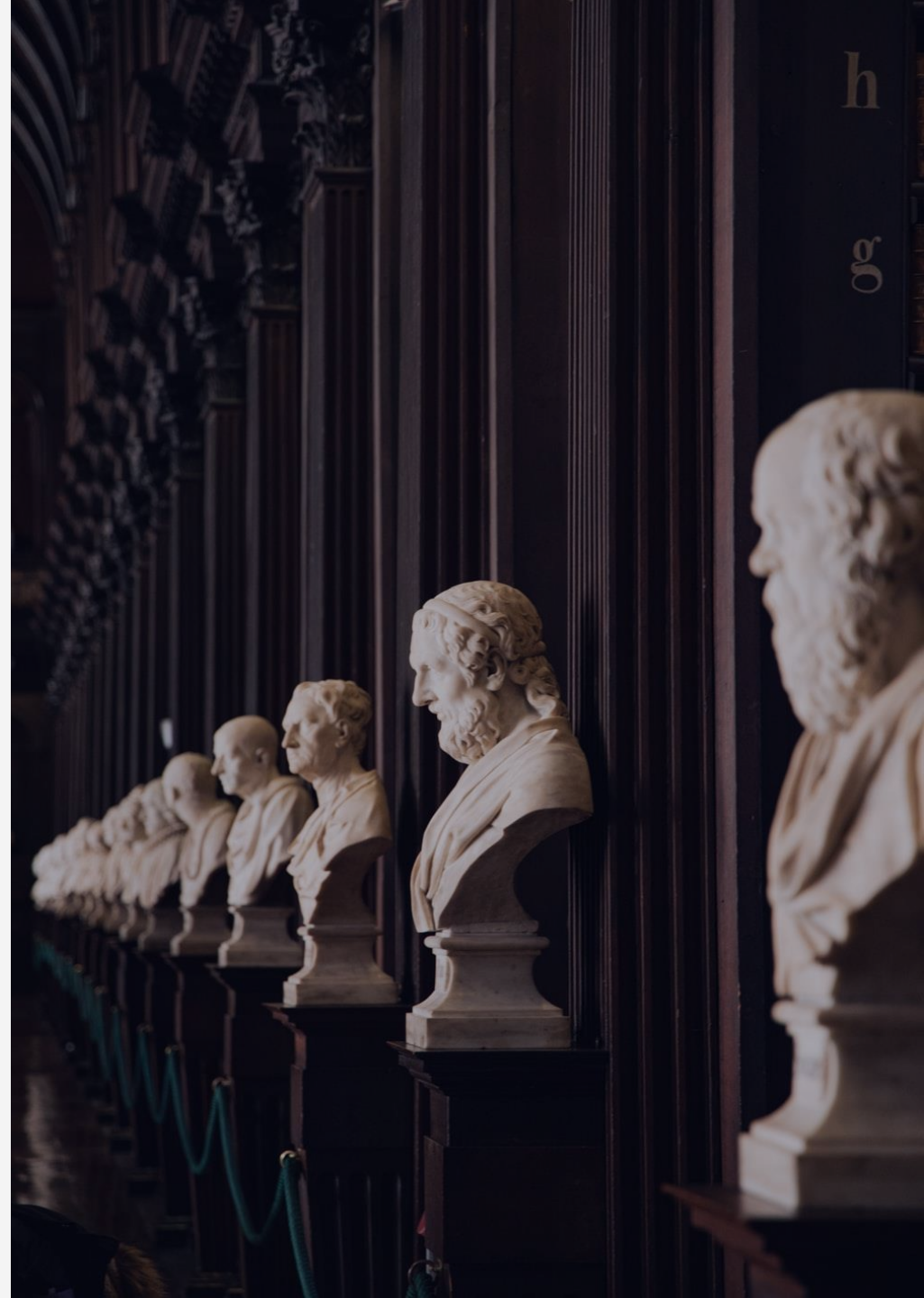


다음식에 구한 값을 대입하면

$$\begin{aligned}\frac{\vec{w}}{|\vec{w}|} \cdot (\vec{x}_+ - \vec{x}_-) &\Rightarrow \frac{1}{|\vec{w}|} (\vec{w} \cdot \vec{x}_+ - \vec{w} \cdot \vec{x}_-) \\ &= \frac{1}{|\vec{w}|} (1 - b + 1 + b) \\ &= \frac{2}{|w|}\end{aligned}$$

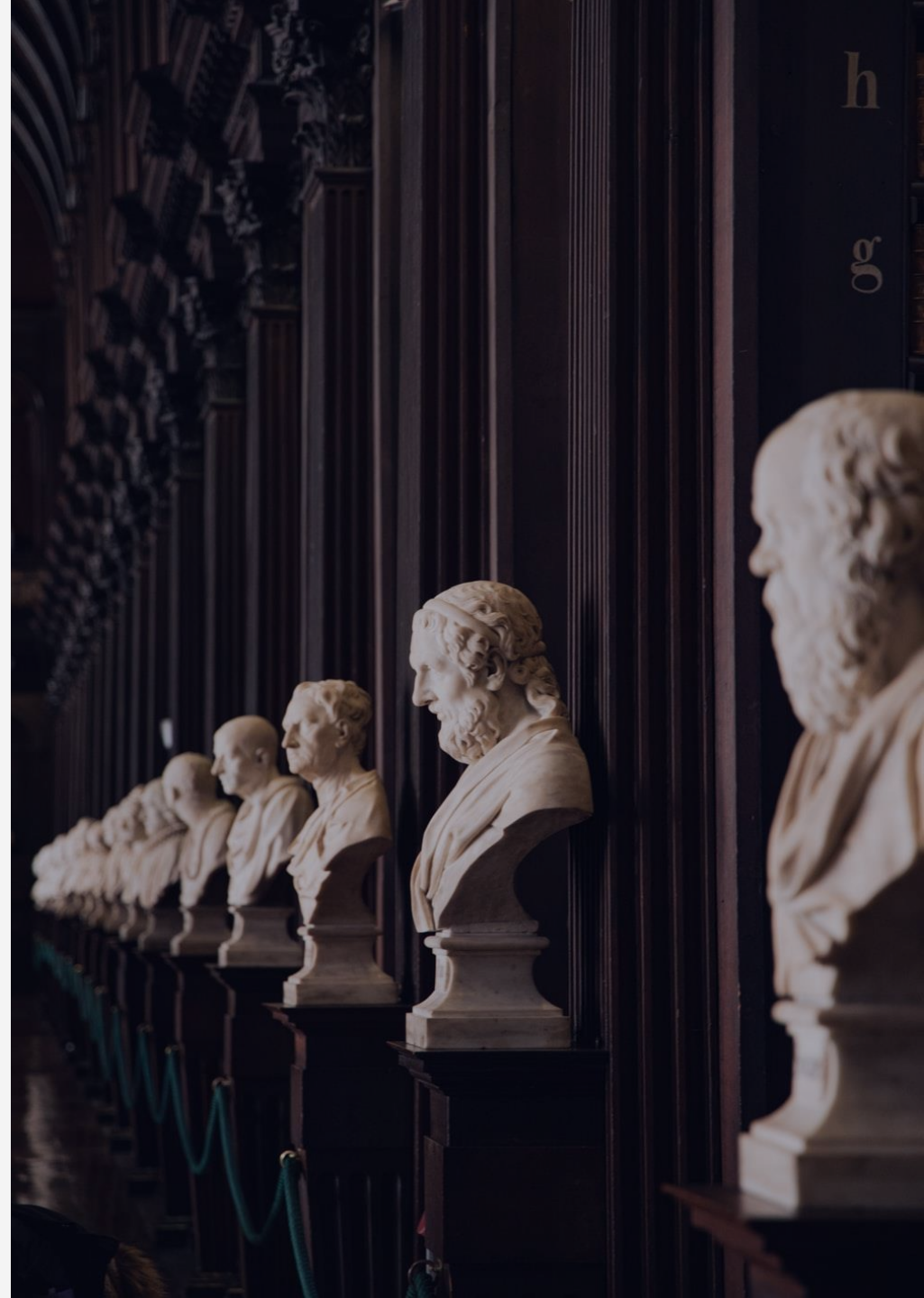
따라서 두 실선 사이의 거리는 $\frac{2}{|w|}$ SVM에서의 목표는 $\frac{2}{|w|}$ 을 최대화 하는 것이라고 할 수 있습니다.

5. Conclusion

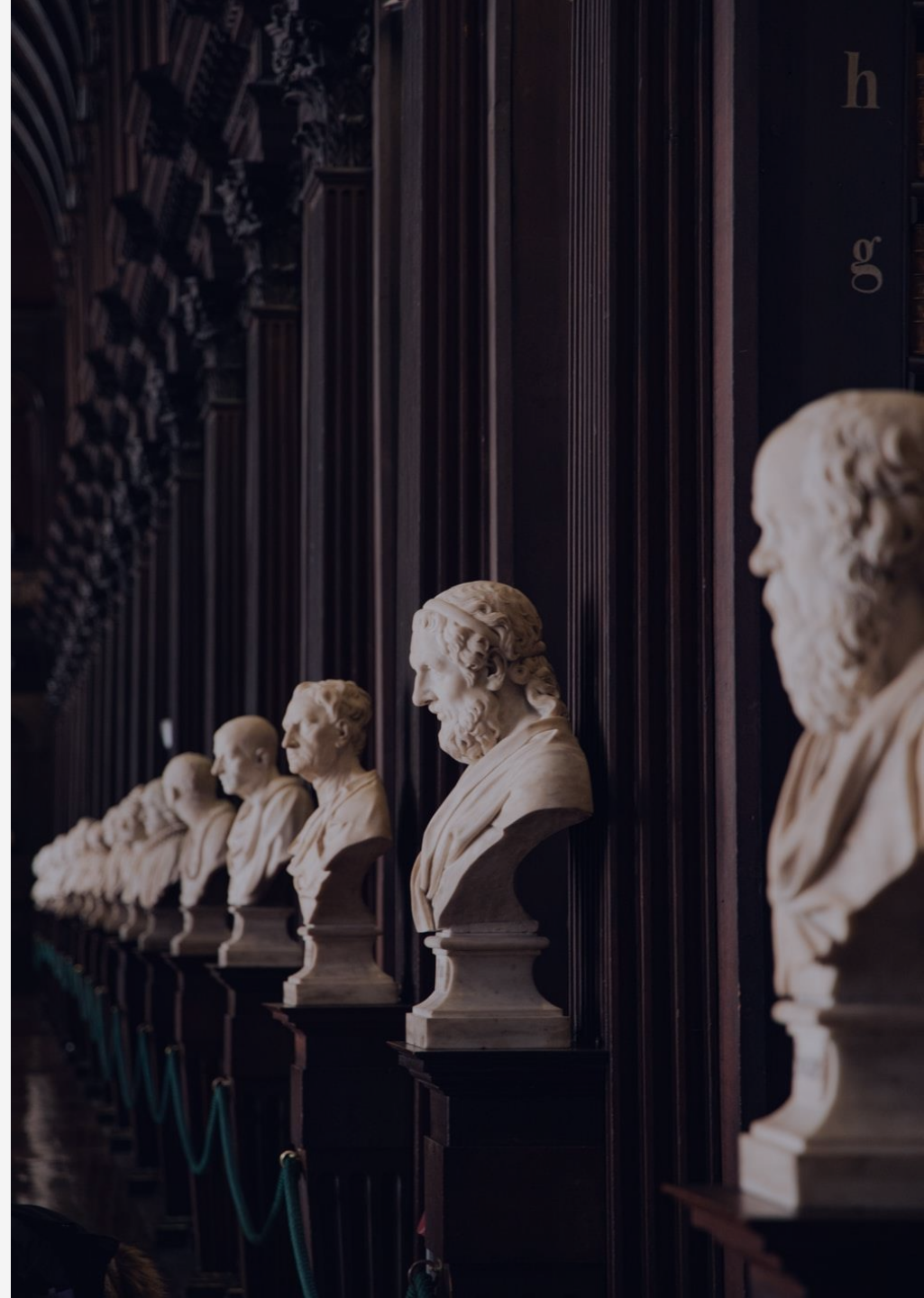


1. 특징 추출 알고리즘인 PCA와 LDA 그리고 이미지 인식 알고리즘인 SVM, CNN의 원리를 연구
2. PCA, LDA, SVM에 사용된 선형대수를 연구
3. LFW 데이터셋을 이용하여 얼굴 인식에 대한 (PCA-SVM), (LDA-SVM), CNN, (PCA-KNN), (LDA-KNN)의 정확도를 비교 분석하여 최적의 특징 추출 알고리즘과 이미지 인식 알고리즘을 연구

6. References



7. The role and contribution(%) of team members to the project



7. The role and contribution(%) of team members to the project

Members Role and Contribution	박윤준	백승엽	김주찬
Role	<ul style="list-style-type: none"> - PCA 연구 및 ppt 제작 - PCA, LDA, CNN, KNN, SVM을 이용하여 얼굴 인식 프로그램을 코드로 구현 	<ul style="list-style-type: none"> - LDA 연구 및 ppt 제작 - 논문 자료 조사 	<ul style="list-style-type: none"> - SVM 연구 및 ppt 제작 - 논문 자료 조사
Contribution	33%	33%	33%

8. Python Program



Prompt

-

얼굴 인식의 예시 코드를 만들어 주는데 특징 추출은 PCA를 이용해주고 인식 알고리즘은 SVM을 이용해주고 그리고 데이터셋은 LFW를 활용해서 만들어줘. 그리고 Colab에서 실행시킬거라서 참고해서 코드를 만들어줘



물론이죠! 얼굴 인식을 위한 예시 코드를 만들어드리겠습니다. 요구사항에 따라 PCA를 사용하여 특징 추출을 수행하고, SVM을 사용하여 얼굴 인식 알고리즘을 구현할 것입니다. 데이터셋으로는 LFW(Labeled Faces in the Wild) 데이터셋을 활용하겠습니다. 또한, Colab에서 실행할 수 있도록 코드를 작성하겠습니다.

아래의 코드를 참고하여 진행해주세요. 코드는 Python으로 작성되었습니다.

Answer

```
# 필요한 라이브러리 import
import numpy as np
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_lfw_people

# LFW 데이터셋 로드
lfw_dataset = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
X = lfw_dataset.data
y = lfw_dataset.target
target_names = lfw_dataset.target_names

# 데이터 분할: 훈련 세트와 테스트 세트
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# PCA를 사용하여 특징 추출
n_components = 150
pca = PCA(n_components=n_components, whiten=True).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

```
# SVM 분류기 생성 및 훈련
svm = SVC(kernel='rbf', class_weight='balanced')
svm.fit(X_train_pca, y_train)

# 테스트 세트에 대한 예측 수행
y_pred = svm.predict(X_test_pca)

# 정확도 평가
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

- 1.필요한 라이브러리를 가져옵니다.
- 2.LFW 데이터셋을 로드하고 데이터와 레이블을 변수에 할당합니다.
- 3.데이터를 훈련 세트와 테스트 세트로 분할합니다.
- 4.PCA를 사용하여 주어진 차원 수로 데이터를 변환합니다.
- 5.SVM 분류기를 생성하고 훈련 데이터를 사용하여 모델을 훈련시킵니다.
- 6.테스트 데이터에 대한 예측을 수행합니다.
- 7.예측 결과를 평가하여 정확도를 계산하고 출력합니다.

-

얼굴 인식의 예시 코드를 만들어 주는데 특징 추출은 PCA를 이용해주고 인식 알고리즘은 SVM을 이용해줘 그리고 데이터셋은 LFW을 활용해서 만들어줘. 그리고 Colab에서 실행시킬거라서 참고해서 코드를 만들어줘

위의 Prompt와 형식은 비슷하게 하고 **조건만 바꾸어서** Prompt를 다시 작성하여 (LDA-SVM), CNN, (PCA-KNN), (LDA-KNN)을 코드로 구현

PCA-SVM

PCA / SVM

```
# 필요한 라이브러리 import
import numpy as np
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_lfw_people

# LFW 데이터셋 로드
lfw_dataset = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
X = lfw_dataset.data
y = lfw_dataset.target
target_names = lfw_dataset.target_names

# 데이터 분할: 훈련 세트와 테스트 세트
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# PCA를 사용하여 특징 추출
n_components = 150
pca = PCA(n_components=n_components, whiten=True).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

# SVM 분류기 생성 및 훈련
svm = SVC(kernel='rbf', class_weight='balanced')
svm.fit(X_train_pca, y_train)

# 테스트 세트에 대한 예측 수행
y_pred = svm.predict(X_test_pca)
```

```
# 정확도 평가
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

Accuracy: 84.11%

Accuracy : 84.11%

LDA-SVM

LDA / SVM

```
# 필요한 라이브러리 import
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_lfw_people

# LFW 데이터셋 로드
lfw_dataset = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
X = lfw_dataset.data
y = lfw_dataset.target
target_names = lfw_dataset.target_names

# 데이터 분할: 훈련 세트와 테스트 세트
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 클래스 수 계산
n_classes = len(target_names)

# LDA를 사용하여 특징 추출
n_components = min(X.shape[1], n_classes - 1)
lda = LDA(n_components=n_components)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)

# SVM 분류기 생성 및 훈련
svm = SVC(kernel='rbf', class_weight='balanced')
svm.fit(X_train_lda, y_train)
```



```
# 테스트 세트에 대한 예측 수행
y_pred = svm.predict(X_test_lda)

# 정확도 평가
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

Accuracy: 61.24%

Accuracy : 61.24%

CNN

CNN

```
[ ] # 필요한 라이브러리 import
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_lfw_people
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# LFW 데이터셋 로드
lfw_dataset = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
X = lfw_dataset.images
y = lfw_dataset.target
target_names = lfw_dataset.target_names

# 데이터 분할: 훈련 세트와 테스트 세트
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 데이터 전처리: 정규화 및 크기 조정
X_train = X_train / 255.0
X_test = X_test / 255.0

# 레이블을 one-hot 벡터로 변환
num_classes = len(target_names)
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```



```
# CNN 모델 생성
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(50, 37, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# 모델 컴파일
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# CNN 모델 훈련
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# 테스트 세트에 대한 예측 수행
y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)

# 정확도 평가
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```


8. Python Program

CNN

```
Epoch 1/10
33/33 [=====] - 6s 128ms/step - loss: 1.7923 - accuracy: 0.3796 - val_loss: 1.6094 - val_accuracy: 0.4612
Epoch 2/10
33/33 [=====] - 4s 114ms/step - loss: 1.7174 - accuracy: 0.3990 - val_loss: 1.5948 - val_accuracy: 0.4612
Epoch 3/10
33/33 [=====] - 3s 106ms/step - loss: 1.7217 - accuracy: 0.3990 - val_loss: 1.5960 - val_accuracy: 0.4612
Epoch 4/10
33/33 [=====] - 6s 177ms/step - loss: 1.7057 - accuracy: 0.3990 - val_loss: 1.6333 - val_accuracy: 0.4612
Epoch 5/10
33/33 [=====] - 4s 122ms/step - loss: 1.7127 - accuracy: 0.3990 - val_loss: 1.6076 - val_accuracy: 0.4612
Epoch 6/10
33/33 [=====] - 3s 94ms/step - loss: 1.7059 - accuracy: 0.3990 - val_loss: 1.6195 - val_accuracy: 0.4612
Epoch 7/10
33/33 [=====] - 3s 97ms/step - loss: 1.7116 - accuracy: 0.3990 - val_loss: 1.5943 - val_accuracy: 0.4612
Epoch 8/10
33/33 [=====] - 4s 123ms/step - loss: 1.7091 - accuracy: 0.3990 - val_loss: 1.6082 - val_accuracy: 0.4612
Epoch 9/10
33/33 [=====] - 4s 113ms/step - loss: 1.7086 - accuracy: 0.3990 - val_loss: 1.6056 - val_accuracy: 0.4612
Epoch 10/10
33/33 [=====] - 3s 97ms/step - loss: 1.7098 - accuracy: 0.3990 - val_loss: 1.6113 - val_accuracy: 0.4612
9/9 [=====] - 0s 23ms/step
Accuracy: 46.12%
```

Accuracy : 46.12%

PCA-KNN

PCA / KNN

```
# 필요한 라이브러리 import
import numpy as np
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_lfw_people

# LFW 데이터셋 로드
lfw_dataset = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
X = lfw_dataset.data
y = lfw_dataset.target
target_names = lfw_dataset.target_names

# 데이터 분할: 훈련 세트와 테스트 세트
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# PCA를 사용하여 특징 추출
n_components = 150
pca = PCA(n_components=n_components, whiten=True)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# KNN 분류기 생성 및 훈련
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_pca, y_train)

# 테스트 세트에 대한 예측 수행
y_pred = knn.predict(X_test_pca)
```

```
# 정확도 평가
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

➤ Accuracy: 71.71%

Accuracy : 71.71%

LDA-KNN

LDA / KNN

```
# 필요한 라이브러리 import
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import fetch_lfw_people

# LFW 데이터셋 로드
lfw_dataset = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
X = lfw_dataset.data
y = lfw_dataset.target
target_names = lfw_dataset.target_names

# 데이터 분할: 훈련 세트와 테스트 세트
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# LDA를 사용하여 특징 추출
n_components = min(X.shape[1], len(np.unique(y)) - 1)
lda = LDA(n_components=n_components)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)

# KNN 분류기 생성 및 훈련
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_lda, y_train)

# 테스트 세트에 대한 예측 수행
y_pred = knn.predict(X_test_lda)
```

```
# 정확도 평가
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

Accuracy: 65.50%

Accuracy : 65.50%

■ 왜 PCA-SVM을 사용했을 때 Accuracy가 가장 높았을까?

차원 감소 효과 : PCA는 차원 축소를 통해 원본 데이터의 정보를 보존하면서 불필요한 차원을 제거한다. 이로 인해 SVM 분류기에 입력되는 **특징 벡터의 차원이 줄어들어**, 학습 및 분류 과정에서 **효율적인 연산**을 수행할 수 있다. 작은 차원에서 SVM 모델이 데이터를 더 잘 분류할 수 있는 경우가 있을 수 있다.

데이터셋의 특성 : LFW(Labeled Faces in the Wild) 데이터셋은 얼굴 이미지의 다양한 인물과 각 인물의 다양한 표정, 조명 조건, 각도 등을 포함한다. 이러한 다양성은 데이터셋의 복잡성을 증가시키고 특징 추출 및 분류의 어려움을 초래할 수 있다. PCA는 데이터의 주성분을 추출하여 주요한 정보를 유지하면서 차원을 줄이는 효과를 가지므로, 이러한 데이터의 복잡성을 다루는 데 도움이 될 수 있다.

SVM의 분류 능력 : SVM은 비선형 결정 경계를 찾을 수 있는 커널 기법을 활용하여 **복잡한 분류 문제에 대해 높은 성능**을 발휘할 수 있다. SVM은 데이터의 분포를 고려하여 최적의 결정 경계를 찾는 특징이 있다. 따라서 LFW 데이터셋과 같은 **얼굴 인식 문제**에서 SVM은 주어진 특징 벡터들을 잘 분류할 수 있는 경계를 학습할 수 있다.

데이터셋의 크기 : LFW 데이터셋은 상대적으로 작은 크기를 가지며, PCA는 차원 감소를 통해 잡음이나 불필요한 정보를 줄이는 효과를 가지므로 작은 데이터셋에서도 좋은 성능을 발휘할 수 있다.



감사합니다.

박윤준
김주찬
백승엽