제2차 빅데이터 분석 교육과정 강의교재

- 가완성본 -

ABRC 인공지능-빅데이터연구센터
AI-BIG DATA RESEARCH CENTER

Support Vector Machine

1. Support Vector Machine (SVM) 개요

1.1 SVM개념

- Support Vector Machine (SVM)은 분류 또는 회귀에 사용할 수 있는 기계 학습 알고리즘
- 대부분 분류 문제에 사용
- SVM 알고리즘에서는 각 데이터 항목을 n 차우너 공간 상 하나의 점으로 표시
- 이질적인 두 개 또는 그 이상의 데이터 집단을 잘 구분하는 최적의 초평면 (Optimal Hyper Plane)을 찾는 방법을 제공

SVM은 분류 또는 회귀 문제에 사용할 수 있는 기계 학습 알고리즘입니다. 보통 분류 문제에 많이 사용하는데, 회귀문제에 활용하기 위해서는 support Vector Regression이라고 하는 알고리즘을 사용합니다. SVM 알고리즘의 핵심은 n차원 공간에 있는 각 데이터를, 가장 잘 구분하는 초평면, 경계선 또는 경계면을 찾는 것입니다.

예를 들어, 다음의 그림과 같이 하늘색 동그라미와 빨강색 네모가 있는 2차원 평면을 생각해보겠습니다. 이 두 집단을 분류하는 분류선을 찾으려면 어떻게 해야 할까요? 하지만 이 두 집단을 구분할 수 있는 선은 무수히 많습니다. 우리는 이 중 최상의 분류선을 찾아내는 것이 문제라고 하겠습니다.

SVM 알고리즘은 주어진 두 집단의 데이터로부터 두 집단 사이의 중심을 구한 후, 그 중간에서 최적의 초평면을 구하도록 설계되어 있습니다. 그림 1은 SVM의 초평면 예시에 대한 내용입니다.

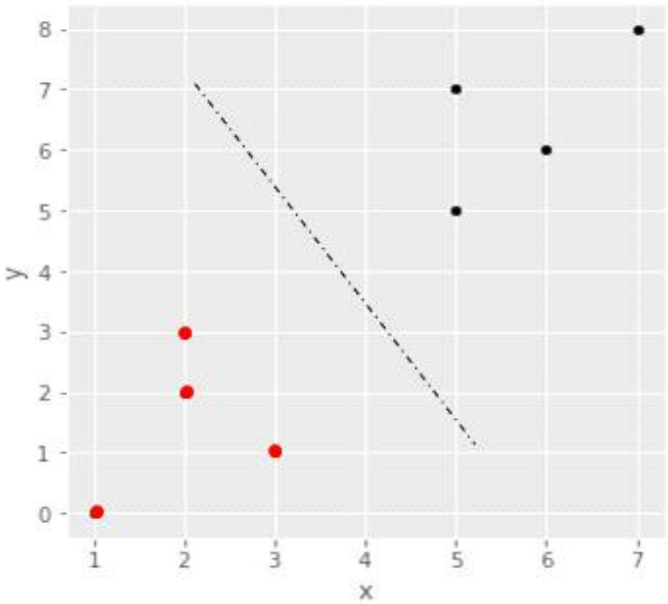


그림 1 SVM 초평면 예시

1.1.1 SVM 개념 (예시)

이번에는 몇 가지 예시를 통해 SVM의 개념에 대해 살펴보겠습니다. 그림에서와 같이 빨간 원과 하늘색 사각형을 구분 한다면 어떤 선으로 구분해야 할까요? 답은 바로 Line 2입니다. 그 이유는 바로 Margin 때문입니다. 여기서 Margin이란 초평면에서 가장 가까이에 있는 점과 초평면 사이의 거리를 뜻합니다. 그림 2는 SVM의 예 (1)에 대한 내용입니다.

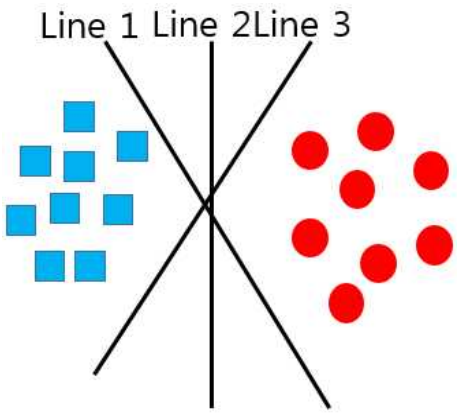


그림 2 SVM의 예 (1)

또 다른 예제를 살펴보겠습니다. 이번엔 Line1과 Line2 중 어떤 선이 데이터를 분류하는 데 적합할까요? 답은 바로 Line 1입니다. 앞에서 설명한 것처럼 Margin 관점에서 보았을 때 Line 2인데 왜 Line 1이지? 라고 의문이 드실 겁니다. Margin 관점에서 보면 Line 2가 맞습니다. 하지만 SVM은 분류를 더

잘하는 것을 Margin에 우선해 선택한다는 특징이 있습니다. 이러한 이유로 Line 1이 답인 것입니다. 그림 3은 SVM의 예 (2)에 대한 내용입니다.

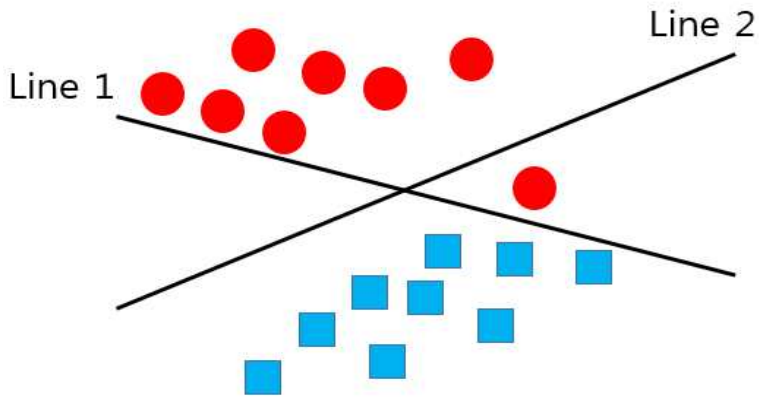


그림 3 SVM의 예 (2)

다음 예제를 살펴보도록 하겠습니다. 이번 예제에서는 Line1과 Line2 모두 정확히 두 그룹으로 나눌 수가 없습니다. 과연 어떤 선으로 분류를 해야 할까요? 답은 Line 1입니다. 이유는 선형 SVM은 Outlier를 어느 정도 무시하면서 최선의 선택을 하기 때문입니다. 그림 4는 SVM의 예 (3)에 대한 내용입니다.

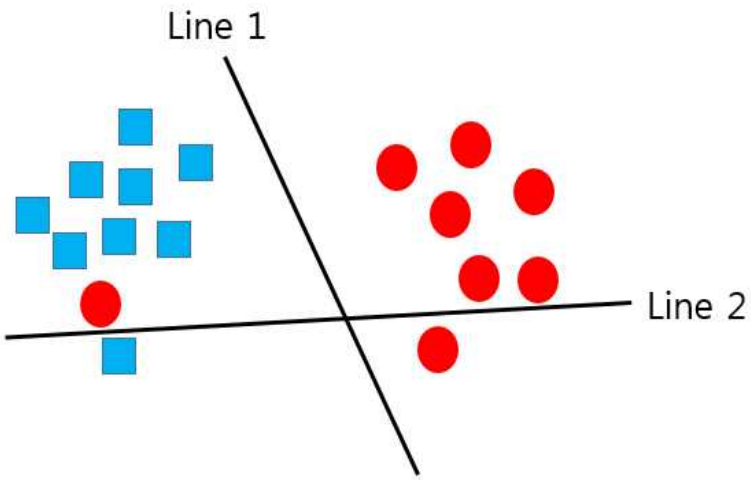


그림 4 SVM의 예 (3)

자 이번에는 SVM을 사용해 선형분류가 아닌 비선형 분류를 하는 방법에 대해 알아보도록 하겠습니다.

1.1.2 비선형 분류

- SVM 은 선형분류와 더불어 비선형 분류에서도 사용
- 비선형 분류를 하기 위해서는 주어진 데이터를 고차원 특징 공간으로 사상하는 작업이 필요
- 효율적으로 실행하기 위해 커널트릭을 사용
- 이러한 커널을 이용하여 차원을 변경하게 되면, 흩어져 있는 데이터에 대해서도 차원을 변경하여 간단하게 나눌 수 있다는 장점을 가짐
- 주요 커널은 Linear Kernel, Polynomial Kernel, Radial Basis Function (RBF)이 있음

SVM은 선형분류와 더불어 비선형 분류에서도 많이 사용됩니다. 그림과 같이 데이터가 주어졌을 때, 선형으로 분류를 할 수 없죠. SVM에서는 차원을 변경하여 이 문제를 해결합니다. 차원을 변경할 때 사용하는 데에 커널이라고 하는 것을 사용합니다. 이런 SVM에 대표적인 커널 함수는 Linear Kernel, Polynomial Kernel, RBF 등이 있습니다. 각각의 커널 함수에는 그룹을 더 잘 분류할 수 있게 최적화된 초평면을 구하는 파라미터들이 따로 존재합니다. 그림 5는 데이터의 차원 변경에 대한 내용입니다.

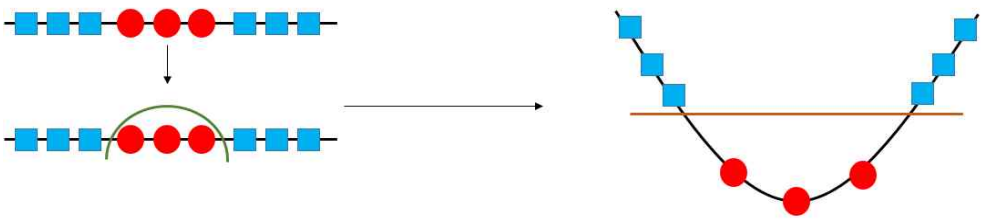


그림 5 데이터의 차원 변경

그럼 어떤 파라미터를 조정해야 분류를 더 잘할 수 있을까요?

- 각각의 커널에서는 최적화를 도와주는 파라미터들이 따로 존재
- 일반적으로 각 문제에 대해서 어떠한 커널의 파라미터를 선택하는 것이 가장 좋은지를 자동적으로 알려주는 방법은 없음
- 실험을 통해 모든 조건을 바꾸면서 SVM의 학습과 예측을 반복해서 최적의 예측률을 보여주는 조건을 찾아야 함

일반적으로 각 문제에 대해서 어떠한 커널의 파라미터를 변경해야 가장

좋은지를 자동적으로 정해주는 방법은 없습니다. 각 그룹을 잘 분류하기 위해선 여러 번의 시도를 통해 파라미터를 조정해가면서, SVM의 학습과 예측을 반복하여 최적의 분류를 해주는 조건을 찾게 됩니다. 그림 6은 비선형 분류에 대한 내용입니다.

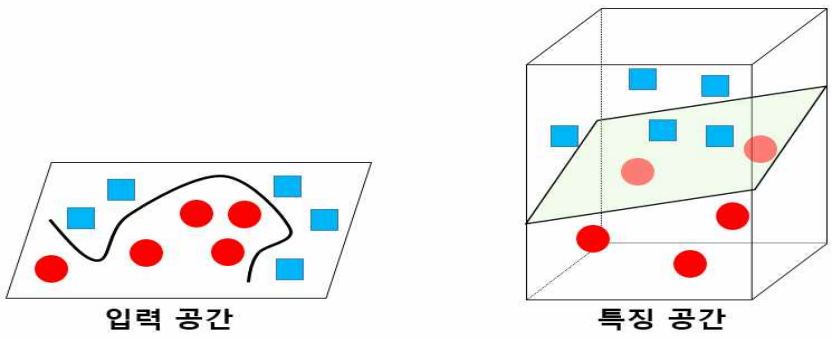


그림 6 비선형 분류의 예

다음은 n 차원 공간에서 초평면을 어떻게 구하는지에 대해 알아보겠습니다. 이를 위해 우리가 배웠던 벡터 개념에 대해 먼저 복습해보겠습니다. 예시로 어떤 사람이 나이와 혈압 두 가지 특징으로 기술된다고 가정해보겠습니다. 그림 7은 벡터 이론의 예제에 대한 내용입니다.

1.1.3 벡터 이론 복습 - 초평면을 구하기 위한 기초 이론

- n개의 특징 (특징, 변수)으로 사람이 기술함
- 어떤 사람의 특징 값은 그림과 같이 화살표로 표시할 수 있음
  - 예) 2가지 특징으로 설명되는 사람 : 혈압 = 100, 나이 = 30

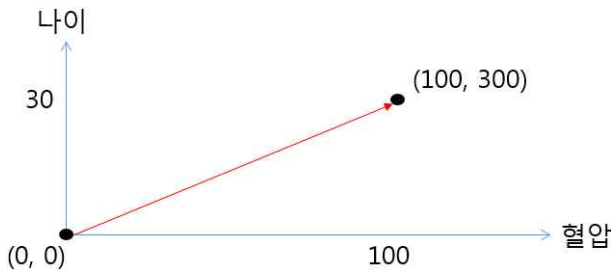


그림 7 벡터 이론의 예제

2차원 그래프에서 사람의 데이터는 다음처럼 (0,0) 좌표 값에서 특징 값의 좌표 값까지 화살표 모양으로 표현됩니다.

다음 그림에서는 3차원 공간에서 데이터를 표현하는 방법에 대해 알아보겠습니다. 표와 같이 네 명의 데이터는 콜레스테롤, 혈압, 나이 세 가지의 특징 값을 각각 다르게 갖고 있습니다. 그리고 이를 3차원 평면에 나타내기 위하여 3개의 좌표 값을 갖는 벡터 값으로 시작점과, 끝점을 나타내었습니다. 표 1은 3차원 공간에 표현할 예제 데이터입니다.

사람	콜레스테롤 (mg/dl)	혈압 (mmHg)	나이 (years)	벡터 시작점	벡터 끝점
1	140	100	30	(0, 0, 0)	(140, 100, 30)
2	230	115	25	(0, 0, 0)	(230, 115, 25)
3	120	150	60	(0, 0, 0)	(120, 150, 60)
4	280	160	40	(0, 0, 0)	(280, 160, 40)

표 1 3차원 공간에 표현할 예제 데이터

결과적으로 위의 3차원 평면처럼 네 명의 환자 데이터를 점으로 나타낼 수 있습니다. 그림 8은 예제 데이터의 3차원 공간 표현에 대한 내용입니다.

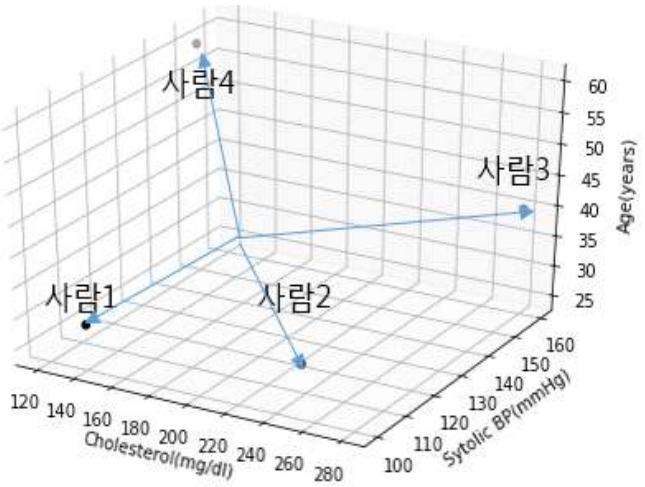


그림 8 예제 데이터의 3차원 공간 표현

1.1.4 벡터 이론 복습 - 벡터 표시의 단순화

- 모든 벡터는 시작점이 0이므로 편의상 끝점만을 표시할 수 있음
- 왼쪽의 예제 벡터는 오른쪽 그림과 같이 점만 표시하기로 함

그림 9는 예제 벡터를 점으로 표현한 내용입니다.

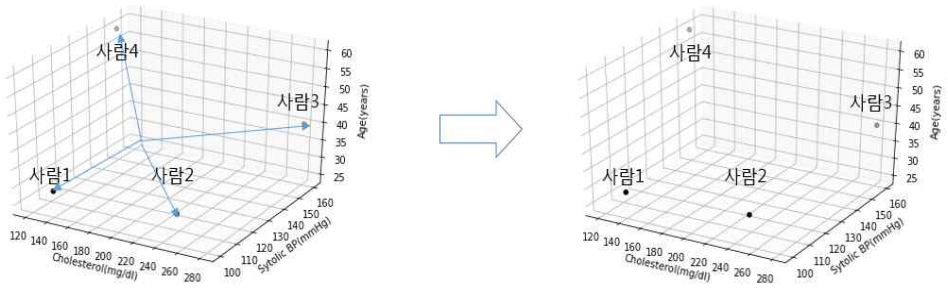


그림 9 예제 벡터를 점으로 표현

이렇게 데이터를 벡터로 표현하고 이를 두 그룹으로 분류하는 초평면을 구해봅시다.

1.1.5 초평면의 개념

2차원 공간(R^2)에서 초평면은 왼쪽 그림과 같이 나타낼 수 있고, 3차원 공간(R^3)에서 초평면은 아래 오른쪽 그림과 같이 나타낼 수 있습니다. 그림 10은 2차원과 3차원 초평면에 대한 예입니다.

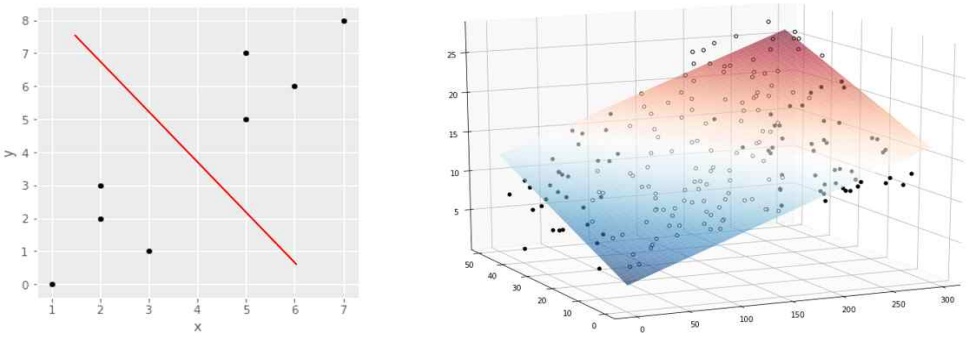


그림 10 2차원 초평면(왼쪽)과 3차원 초평면(오른쪽)에 대한 예

이를 위해 벡터의 기본 연산에 대해 알아보겠습니다. 먼저 스칼라 값과의 곱셈에 대해 살펴보겠습니다.

1.1.6 벡터에 대한 기본연산

- 스칼라와 벡터의 곱셈
- 벡터 $\vec{a}=(a_1,a_2,...,a_n)$와 스칼라 c를 생각해보자
  - $c\vec{a}=(ca_1,ca_2,...,ca_n)$

- 벡터에 스칼라를 곱하면 스칼라가 양수인지 음수인지에 따라 방향이 같은 방향이나 반대 방향으로 벡터를 늘릴 수 있음

벡터(vector)는 수식으로 나타낼 때 $\vec{a}$와 같이 방향과 크기를 함께 표현합니다. 그리고 이와 비교되는 개념으로 크기만 있고 방향 값은 가지지 않는 양을 스칼라(scalar) 값이라고 부릅니다. 이 벡터 값과 스칼라 값을 곱해보겠습니다. 예제와 같이 벡터 $\vec{a}$와 스칼라 값 c를 곱하면 벡터 값의 방향에 따라, 같은 방향이나 반대 방향으로 벡터 값이 변하게 됩니다.

예제를 통해 vector 값과 scalar 값을 곱해보겠습니다. 벡터 $\vec{a}$값 4와 2, 스칼라 c값 2 가 주어졌을 때 $c\vec{a}$값은 4×2와 2×2같이 각 벡터좌표에 스칼라 값을 곱해주어 결과 값 8과 4로 계산됩니다. 그림 11은 벡터와 스칼라의 곱셈 예제에 대한 내용입니다.

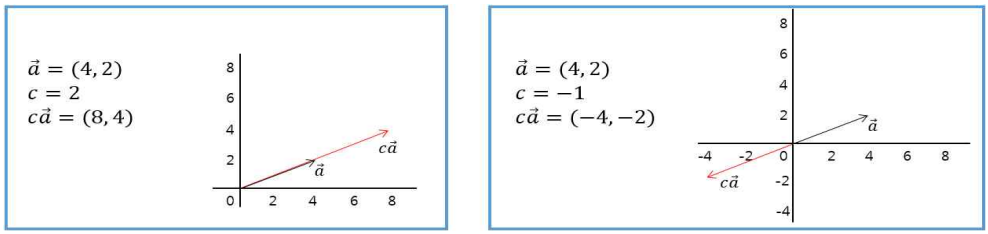


그림 11 벡터와 스칼라 곱셈의 예

이번에는 벡터간의 사칙 연산에 대해 살펴보겠습니다. 먼저 덧셈과 뺄셈입니다.

- 벡터와 벡터의 덧셈과 뺄셈
- 벡터 $\vec{a}=(a_1,a_2,...,a_n)$와 $\vec{b}=(b_1,b_2,...,b_n)$를 생각해보자
  - $\vec{a}+\vec{b}=(a_1+b_1,a_2+b_2,...,a_n+b_n)$
  - $\vec{a}-\vec{b}=(a_1-b_1,a_2-b_2,...,a_n-b_n)$

예제처럼 벡터 간의 덧셈과 뺄셈은 벡터 안의 요소 각각을 계산하여 이루어집니다. 예제를 통해 벡터 간의 덧셈과 뺄셈을 해보겠습니다. 먼저 덧셈 예제부터 살펴보겠습니다. 벡터 $\vec{a}$값 4와 2, 벡터 $\vec{b}$값 4와 4가 주어졌을 때 $\vec{a}+\vec{b}$은 같은 위치에 있는 벡터들 끼리 더해주어 4+4와 2+4와 같이 일반 덧

셈처럼 8과 6 값으로 계산됩니다.

다음으로 뺄셈 예제를 살펴보겠습니다. 벡터 $\vec{a}$값 4와 2, 벡터 $\vec{b}$값 8과 0이 주어졌을 때 $\vec{a}-\vec{b}$값은 같은 위치에 있는 벡터들 끼리 빼주어, 4-8과 2-0 같이 일반 뺄셈처럼 -4와 2값으로 계산됩니다. 그림 12는 벡터간의 덧셈과 뺄셈 예제에 대한 내용입니다.

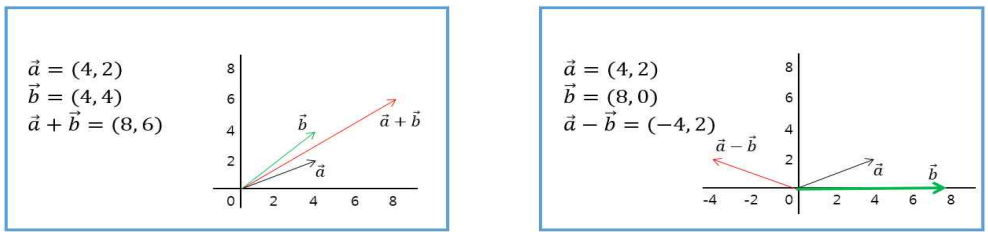


그림 12 벡터간의 덧셈과 뺄셈 예제

물론 곱셈, 나눗셈도 가능합니다. $\vec{a}$와 $\vec{b}$를 곱하면 일반 곱셈과 같은 값이 나오는 것을 확인할 수 있습니다.

- 벡터와 벡터의 곱셈
- 벡터 $\vec{a}=(a_1,a_2,...,a_n)$와 $\vec{b}=(b_1,b_2,...,b_n)$를 생각해보자
  - 원점과 해당 벡터의 가장 가까운 거리를 계산
  - $\vec{a} \cdot \vec{b}=(a_1b_1,a_2b_2,...,a_nb_n)=\sum_{i=1}^{n} a_ib_i$
  - $\vec{a} \cdot \vec{b}=\| \vec{a} \|_2 \| \vec{b} \|_2 \cos\theta$ 공식으로도 구할 수 있음
  - $\cos\theta$에서 θ는 $\vec{a}$와$\vec{b}$ 사이에 각도
  - 벡터가 수직일 때, $\vec{a} \cdot \vec{b}=0$

곱셈은 $\| \vec{a} \|_2 \| \vec{b} \|_2 \cos\theta$공식으로도 구할 수 있습니다. 이때, $\| \vec{a} \|$이 뜻하는 것은 0,0 좌표와 벡터 사이의 거리를 뜻합니다. 여기서 $cos\theta$에서 θ는 $\vec{a}$와 $\vec{b}$사이에 각도입니다. 이때 두 벡터가 수직 값 일 때, $\vec{a} \cdot \vec{b}$값은 0이 됩니다.

예제를 통해 벡터 간의 곱셈을 해보겠습니다. 먼저 곱셈 예제부터 살펴보겠습니다.

벡터 $\vec{a}$값 4와 2, 벡터 $\vec{b}$값 2와 0이 주어졌을 때 $\vec{a} \cdot \vec{b}$값은 같은 위치에 있는 벡터들 끼리 곱해주어 4×2와 2×0과 같이 일반 곱셈처럼 8과 0값으로 계산됩니다.

다음 곱셈 예제를 살펴보겠습니다. 벡터 $\vec{a}$값 0과 4, 벡터 $\vec{b}$값 8과 0이 주어졌을 때 $\vec{a} \cdot \vec{b}$값은 같은 위치에 있는 벡터들 끼리 곱해주어, 0×4와 8×0과 같이 계산되어 결과 값 0이 됩니다. 이때 앞에서 말씀 드렸듯이, 두 벡터가 수직이기 때문에 $\vec{a} \cdot \vec{b}$값은 0이 됩니다. 그림 13은 벡터간의 곱셈 예제에 대한 내용입니다.

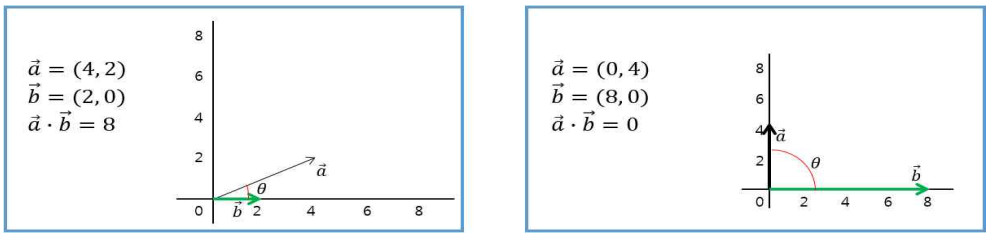


그림 13 벡터간의 곱셈 예제

다음은 Euclidean length or L2-norm에 대해 알아보도록 하겠습니다.

- 유클리드 거리(Euclidean Length) or L2-norm
- 벡터 $\vec{a} = (a_1, a_2, ..., a_n)$를 생각해보자
  - $L2-norm = \| \vec{a} \|_2 = \sqrt{a_1^2 + a_2^2 + ... + a_n^2}$
  - 각각의 요소를 제곱한 값을 합하고 루트를 취해주면 됨
  - 원점과 해당 벡터의 가장 가까운 거리를 계산함

$L2-norm$은 원점과 해당 벡터의 가장 가까운 거리를 계산합니다. $\| \vec{a} \|_2$과 같이 표현하며 $\sqrt{a_1^2 + a_2^2 + ... + a_n^2}$과 같이 수식으로 풀이 됩니다. 식과 같이 각각의 요소를 제곱한 값을 모두 합하고 루트를 취해줌으로써 원점과 해당 벡터간의 유클리드 거리를 계산합니다. 그림 14는 유클리드 거리 or L2-norm의 예제에 대한 내용입니다.

$\vec{a} = (4, 2)$
$\|\vec{a}\|_2 = \sqrt{20} \approx 4.47$

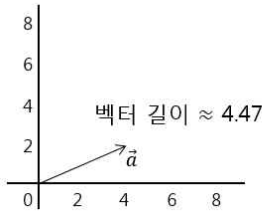


그림 14 유클리드 거리 or L2-norm의 예제

자 이제 그룹들을 분리하는 초평면을 구하는 방법을 알아보도록 하겠습니다.

1.1.7 초평면을 구하는 식

- 점 P_0과 P_0을 통과하는 벡터 $\vec{w}$를 정의
- P_0을 포함하고 벡터 $\vec{w}$에 수직인 평면을 초평면(Hyperplane)으로 정의
- $\vec{x_0} = \overrightarrow{OP_0}$, $\vec{x} = \overrightarrow{OP}$ (P = 초평면에서 임의의 점)
- 평면상의 P에 대한 조건은 벡터 $\vec{x} - \vec{x_0}$가 $\vec{w}$에 수직이어야 함
- $\vec{w} \cdot (\vec{w} - \vec{w_0}) = 0$ 또는 $\vec{w} \cdot \vec{x} - \vec{w} \cdot \vec{x_0} = 0$, $b = -\vec{w} \cdot \vec{x_0}$
- $\vec{w} \cdot \vec{x} + b = 0$
- 구하는 식은 $n > 3$인 R^n에 대해서도 적용

초평면을 구하는 식은 점 P_0와 평면에 수직인 벡터 $(\vec{w})$로 정의할 수 있습니다. 여기서 $\vec{x_0}$은 $\overrightarrow{OP_0}$와 같고 $\vec{x}$는 $\overrightarrow{OP}$와 같습니다. P는 초평면에서 임의의 점입니다. 초평면상의 P는 벡터 $\vec{x} - \vec{x_0}$가 $\vec{w}$에 수직이어야 한다는 조건을 갖습니다.

다음 식을 쉽게 이해하기 위해선 직각삼각형을 구하는 수식과 같은 방법이라고 생각하시면 됩니다. 두 변이 $\vec{x}$와 $\vec{x_0}$라고 생각하시면 나머지 변은 빼서 구할 수 있기 때문입니다. 최종적으로 초평면을 구하는 식은 $\vec{w} \cdot \vec{x} + b = 0$와 같습니다. 여기서 b는 계수(높이)를 뜻합니다. 구하는 식은 3차원이 아닌 더 높은 차원에서도 똑같이 적용 됩니다. 그림 15는 초평면 정의에 대한 내용입니다. 다음으로 예시를 통해 접근해보도록 하겠습니다.

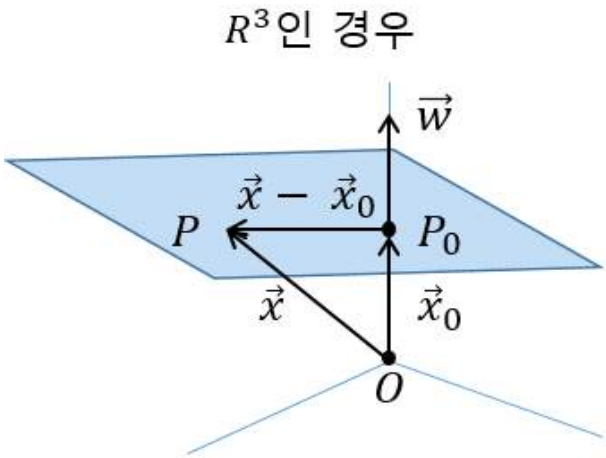


그림 15 초평면 정의

1.1.8 초평면 구하는 식 (예시)

- $\vec{w} = (2, -4, 7)$
- $P_0 = (-2, 0, -4)$
- $b = -\vec{w} \cdot \vec{x_0} = -(-4, 0, -28) = 32$
  - $\vec{w} \cdot \vec{x} + 32 = 0$
  - $(2, -4, 7) \cdot \vec{x} = 0$
  - $(2, -4, 7) \cdot \vec{x} + 32 = 0$
  - $(2, -4, 7) \cdot (x_{(1)}, x_{(2)}, x_{(3)}) + 32 = 0$
  - $2x_{(1)} - 4x_{(2)} + 7x_{(3)} + 32 = 0$

$\vec{w} = (2, -4, 7)$과 $P_0 = (-2, 0, -4)$ 값이 주어졌을 때 b를 구하는 수식 $-\vec{w} \cdot \vec{x_0}$에 대입해보겠습니다. $-(-4, 0, -28) = 32$와 같이 값이 나오는 것을 확인할 수 있습니다. 초평면을 구하는 $\vec{w} \cdot \vec{x} + 32 = 0$식에 대입하면 최종적으로 $2x_{(1)} - 4x_{(2)} + 7x_{(3)} + 32 = 0$을 얻을 수 있습니다. 계수가 바뀌면 초평면은 방향을 따라 움직이고 제시된 그림과 같이 병렬로 된 초평면들을 얻을 수 있습니다. 그림 16은 초평면을 구하는 예에 대한 내용입니다. 다음은 선형 SVM 분류에 대해 알아보도록 하겠습니다.

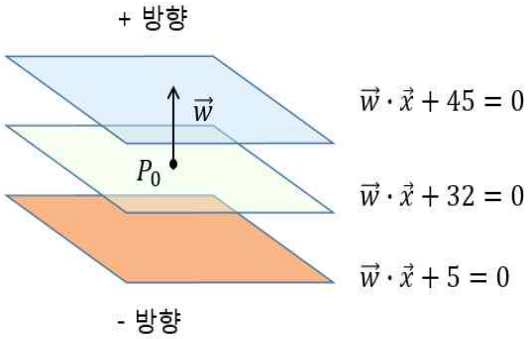


그림 16 초평면을 구하는 예

1.1.9 선형 SVM 분류란 (Linear SVM classifier)

- 데이터
  - $\overrightarrow{x_1},\overrightarrow{x_2},...,\overrightarrow{x_N}\in R^n$
  - $\overrightarrow{y_1},\overrightarrow{y_2},...,\overrightarrow{y_N}\in -1,+1$
- 부정적인 예와 긍정적인 예를 분류할 수 있는 초평면을 찾을 때 다음과 같이 초평면이 무한히 존재함
- SVM은 경계에 있는 서포트 벡터 사이의 간격을 최대화하는 초평면을 찾음
- 경계상의 서포트 벡터들이 잡음으로 인해 정확하지 않은 경우, SVM은 잘 분리하지 못함

데이터는 $\vec{x}$와 $\vec{y}$ 벡터입니다. $\vec{y}$가 나타내는 것은 음양을 나타냅니다. 해당 벡터를 점으로 표시하고 두 그룹을 나눌 수 있는 선, 즉 초평면을 찾을 때 다음의 그림과 같이 무한히 많은 초평면이 존재합니다. 그림 17은 서포트 벡터 사이의 초평면들의 대한 내용입니다.

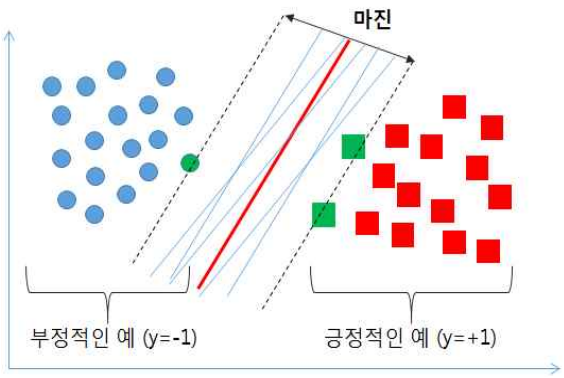


그림 17 서포트 벡터 사이의 초평면

- 마진 (Margin)은 초평면 사이의 거리를 뜻함
- 평행한 두 초평면
  - $\vec{w} \cdot \vec{x} + b = -1, \vec{w} \cdot \vec{x} + b = +1$
- 평행한 두 초평면에 마진을 구하는 식은 다음과 같음
  - $\vec{w} \cdot \vec{x} + (b+1) = 0, \vec{w} \cdot \vec{x} + (b-1) = 0$
  - 두 초평면 사이의 거리의 식은 $D = \frac{|b_1 - b_2|}{\| \vec{w} \|}$과 같음
  - 결과적으로 $D = \frac{2}{\| \vec{w} \|}$
- 우리는 마진을 극대화하기를 원하기 때문에, $\| \vec{w} \|$를 최소화하거나 $\frac{1}{2} \| \vec{w} \|^2$를 최소화해야함

이때 SVM은 경계에 있는 서포트 벡터 사이의 간격을 최대화 하는 초평면을 찾습니다. 해당 초평면은 오른쪽에 있는 빨간색 선과 같습니다. 해당 식을 간결하게 나타내면 $D = \frac{2}{\| \vec{w} \|}$이 성립됩니다.

다음은 평행한 초평면 $\vec{w} \cdot \vec{x} + b = -1, \vec{w} \cdot \vec{x} + b = +1$식이 주어졌을 때 Margin을 구해보도록 하겠습니다. 두 초평면 사이의 거리 식은 $D = \frac{|b_1 - b_2|}{\| \vec{w} \|}$과 같습니다. 여기서 b1, b2는 평행한 초평면의 b값입니다. 결과적으로 Margin을 극대화하기 위해선 $\| \vec{w} \|$ 값을 최소화하거나 $\frac{1}{2} \| \vec{w} \|^2$을 최소화해야 하는 겁니다. 그림 18은 초평면 사이의 마진에 대한 내용입니다.

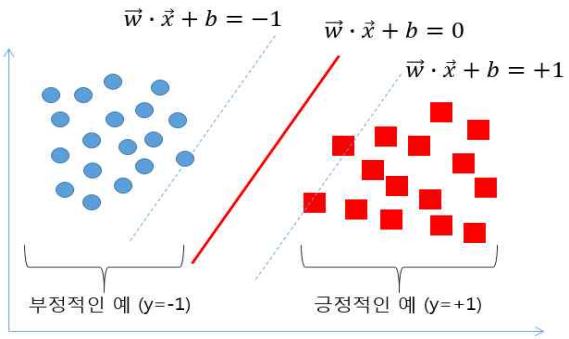


그림 18 초평면 사이의 마진

다음은 SVM 중 Soft margin Linear SVM에 대해 알아보도록 하겠습니다.

1.1.10 선형 SVM 분류란 (Soft margin Linear SVM)

- 선형으로 분리할 수 없는 데이터일 때 사용(잡음이 있는 측정값이 있거나, 데이터가 비선형)
- 잘못 분류 된 경우, 초평면에서 거리를 생각할 수 있는 여유 변수(Slack Variable)를 각 요소에 할당하고, 그렇지 않은 경우 0을 지정

Soft margin SVM이란 Hard margin 방식을 보완해서 나온 방식입니다. Hard margin이란 매우 엄격하게 두 개의 그룹으로 분리하는 초평면을 구하는 방법으로, 모든 입력 벡터들을 초평면을 사이에 두고 무조건 한 그룹에 속하게 분류해야 합니다. 이렇게 되면 몇 개의 노이즈로 인해 두 그룹을 구별하는 초평면을 잘못 구할 수도 있습니다.

이를 해결하기 위해 Soft Margin 방식이 개발되었습니다. Soft Margin은 Hard Margin 방법을 기반으로 하지만, 서포트 벡터가 위치한 경계선에 약간의 여유 변수를 두어 차이가 있습니다. 즉, 다음의 예제 그림을 살펴보면 두 그룹을 분류한 초평면의 Slack Variable 즉, 여유 변수가 입력된 노이즈들을 무시하고 초평면을 그린 것을 확인할 수 있습니다. 그림 19는 초평면에서의 여유 변수에 대한 내용입니다.

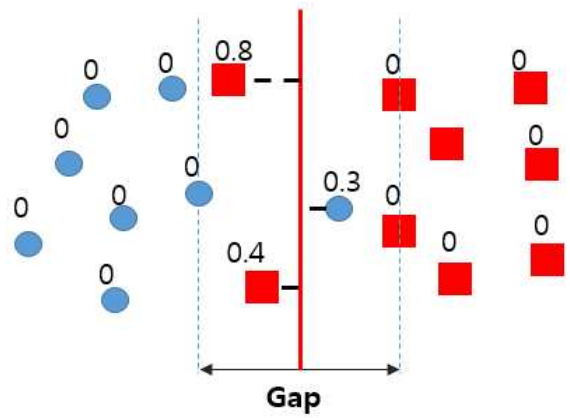


그림 19 초평면에서의 여유 변수

Soft margin Linear SVM의 두 공식

- i = 1, ... ,N
- Primal 공식

- 조건 = $y_i(\overrightarrow{w} \bullet \overrightarrow{x_i} + b \geq 1 - \xi_i)$
- 최소화하기 위한 목적 함수 = $\frac{1}{2}\sum_{i=1}^{n} w_i^2 + C\sum_{i=1}^{n}\xi_i$

• Dual 공식

- 조건 = $0 \leq a_i \leq C$와 $\sum_{i=1}^{N} a_i y_i = 0$
- 최소화하기 위한 목적 함수 = $\sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n} a_i a_j y_i y_j \overrightarrow{x_i} \bullet \overrightarrow{x_j}$

Soft margin Linear SVM을 구하는 공식은 다음과 같이 Primal 공식과 Dual 공식으로 나뉩니다. 두 공식은 오류를 최소화하기 위한 목적함수 입니다. 먼저, 첫 번째의 Primal 공식을 사용하기 위한 조건은 $\overrightarrow{w} \bullet \overrightarrow{x_i} + b \geq 1 - \xi_i$와 같습니다. 이 조건을 만족하기 위해선 ξ_i값은 음수가 아니어야 합니다. 식은 $\frac{1}{2}\sum_{i=1}^{n} w_i^2 + C\sum_{i=1}^{n}\xi_i$와 같습니다.

두 번째 Dual 공식을 사용하기 위한 조건은 $0 \leq a_i \leq C$와 $\sum_{i=1}^{N} a_i y_i = 0$입니다. 여기서 C값이 의미하는 것은 얼마만큼의 여유를 가지고 오류를 인정할 건지에 대한 값입니다. 만약 C값이 매우 커지면, Soft margin SVM은 Hard margin SVM과 같아집니다. 그리고 C값이 매우 작아지면 그림처럼 훈련 데이터에서 잘못된 분류를 하는 것을 볼 수 있습니다. 그림 20은 C값 변화에 따른 Soft margin SVM에 대한 내용입니다.

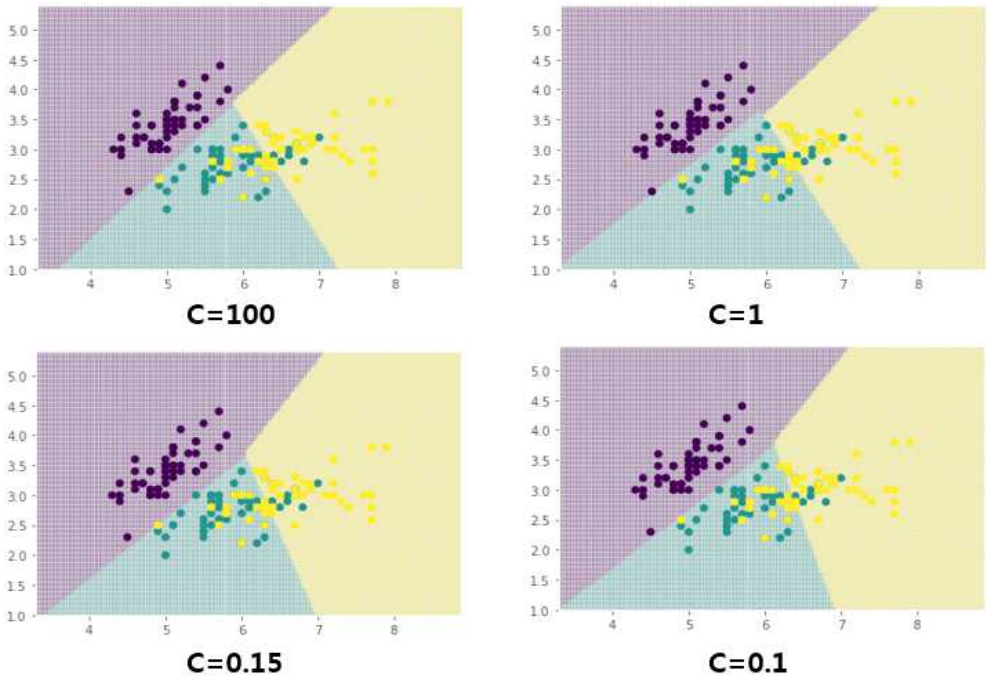


그림 20 C값 변화에 따른 Soft margin SVM

예를 들어서 C값이 100일 때 그림을 확인하면 노랑색원과 초록색 원 경계가 매우 잘 구분되어있는 것을 확인할 수 있습니다. 그렇지만 C값이 0.1값일 때 그림을 확인하면 노랑색원과 초록색 원 경계에 기존 100일 때와 비교해서 상당부분 잘못된 분류를 하는 것을 확인 할 수 있습니다. 다음으로 선형으로 분리할 수 없는 데이터일 때 사용하는 커널트릭에 대해 알아보겠습니다.

1.1.11 커널트릭

- 선형으로 분리할 수 있는 데이터가 아닐 때 사용

예를 들어 그림과 같이 데이터가 입력 공간에서 선형으로 분리가 되지 않을 때에는, 커널을 사용하여 다음의 그림과 같이 분리 가능한 공간을 생성해 선형으로 분리합니다. 그림 21은 커널트릭에 대한 내용입니다.

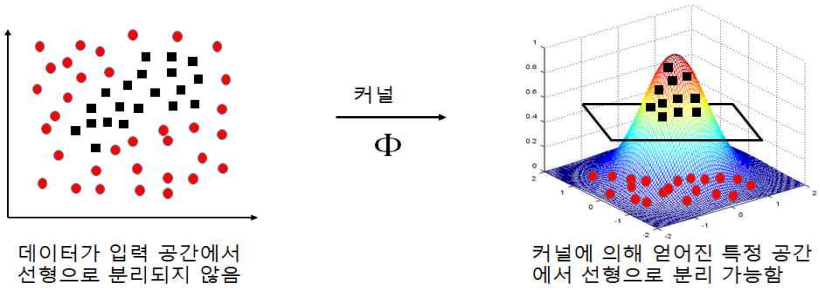


그림 21 커널트릭

- Φ에 대해서 자세히 알기보단 $K(\bullet,\bullet): R^N \times R^N \rightarrow R$함수에 대해서 알아야 함
- 모든 유효한 커널일 수 없음

커널트릭을 사용할 때 어떻게 수식이 변경되는지 알아보도록 하겠습니다. 왼쪽은 선형으로 분리되지 않는 입력 식이고 오른쪽은 고차원 공간으로 커널트릭을 사용해 구한 식입니다. 커널트릭 수식에서 $K(\overrightarrow{x_i},\overrightarrow{x})$은 커널을 의미하는데, 이를 계산하는 값은 커널 마다 다른 방식으로 구해집니다. 그림 22는 커널트릭 수식에 대한 내용입니다.

원본 데이터 $\vec{x}$(입력공간)

$f(x) = sign(\vec{w} \cdot \vec{x} + b)$

$\vec{w} = \sum_{i=1}^{N} \alpha_i y_i \vec{x}_i$

고차원 공간에서의 데이터 $\Phi(\vec{x})$

$f(x) = sign(\vec{w} \cdot \vec{x} + b)$

$\vec{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\vec{x}_i)$

$f(x) = sign(\sum_{i=1}^{N} \alpha_i y_i \Phi(\vec{x}_i) \cdot \Phi(\vec{x}) + b)$

$f(x) = sign(\sum_{i=1}^{N} \alpha_i y_i \mathrm{K}(\vec{x}_i, \vec{x}) + b)$

그림 22 커널트릭 수식

1.1.12 많이 사용하는 커널

- 커널은 일부 공간에서 점으로 표현 됨

$$K(\overrightarrow{x_i},\overrightarrow{x_j}) = \Phi(\overrightarrow{x_i}) \bullet \Phi(\overrightarrow{x_j})$$

대표적으로 사용하는 커널 함수를 살펴보겠습니다. 커널 함수에도 여러 가지 종류가 있는데, 여기서 대표적으로 가장 많이 사용되는 함수는 Linear Kernel, Polynomial kernel 입니다. 커널 함수를 선택할 때는 실험을 통해 해당 데이터에 맞는 커널을 찾아 선택할 수 있습니다. 표 2는 각 커널함수에 대한 커널함수 식에 대한 내용입니다.

Linear kernel	$K(\overrightarrow{x_i},\overrightarrow{x_j}) = \overrightarrow{x_i} \bullet \overrightarrow{x_j}$
Gaussian kernel	$K(\overrightarrow{x_i},\overrightarrow{x_j}) = \exp(-r \parallel \overrightarrow{x_i} - \overrightarrow{x_j} \parallel^2)$
Exponential kernel	$K(\overrightarrow{x_i},\overrightarrow{x_j}) = \exp(-r \parallel \overrightarrow{x_i} - \overrightarrow{x_j} \parallel)$
Polynomial kernel	$K(\overrightarrow{x_i},\overrightarrow{x_j}) = (p + \overrightarrow{x_i} \bullet \overrightarrow{x_j})^q$
Hybrid kernel	$K(\overrightarrow{x_i},\overrightarrow{x_j}) = (p + \overrightarrow{x_i} \bullet \overrightarrow{x_j})^q \exp(-r \parallel \overrightarrow{x_i} - \overrightarrow{x_j} \parallel^2)$
Sigmoidal	$K(\overrightarrow{x_i},\overrightarrow{x_j}) = \tanh(k\overrightarrow{x_i} \bullet \overrightarrow{x_j} - \delta)$

표 2 각 커널함수 별 커널함수 식

다음은 파이썬 Sklearn 라이브러리에서 SVM Parameter 세팅에 대해서 알아보도록 하겠습니다.

1.1.13 파이썬 Sklearn SVM Parameter

- SVM에서 사용되는 파라미터는 Kernel, C, Gamma가 대표적으로 많이 사용
- Kernel
  - 기본값으로 rbf를 설정 (값은 특정 중심에서 거리에 의존하는 함수값)
  - rbf (Radial Basis Function, Linear, Polynomial) 총 3가지
- C
  - 기본값으로 1.0 설정
  - C값을 낮추면 초평면이 매끄러워지고, 값을 높이면 서포트 벡터들을 더 잘 분류함 (Classifying training points correctly)
- Gamma
  - 기본값으로 auto 설정
  - Gamma 값을 낮추면 초평면에서 멀리 떨어진 서포트 벡터들의 영향이 낮고, 값을 높이면 멀리 떨어진 요소들의 값이 영향이 큼

- 값을 높일 경우 초평면에 인접한 서포트 벡터들의 영향(Weight)가 커지기 때문에 초평면이 울퉁불퉁 (Uneven)하게 됨

파이썬에서 머신러닝 학습을 도와주는 패키지 Sklearn은 SVM 모델을 이용할 수 있게 해줍니다. SVM 모델 함수에는 많은 Parameter들이 있지만 대표적으로 많이 사용하는 총 세가지 Kernel, C, gamma가 있습니다.

먼저 Kernel 파라미터 값에는 rbf, linear, polynominal 총 세 가지를 사용할 수 있는데, 디폴트 값은 rbf입니다. 다음 C 파라미터 값은 얼마 만큼의 여유를 가지고 오류를 인정할 건지에 대한 값으로, 디폴트로 1.0 값을 갖습니다. 만약, C값을 낮추면 초평면이 매끄러워지고, 값을 높이면 서포트 벡터들을 더 잘 분류합니다.

Gamma 파라미터는 값을 낮추면 초평면에서 멀리 떨어진 서포트 벡터들의 영향을 작게 만들고, 값을 높이면 멀리 떨어진 서포트 벡터들의 값들의 영향을 크게 만드는 역할을 합니다. 또한 값을 높일 경우 초평면에 인접한 서포트 벡터들의 영향이 커지기 때문에 초평면이 울퉁불퉁하게 됩니다. 앞에서는 SVM 모델의 Parameter의 개념에 대해서 익혔습니다. 하지만 Parameter를 너무 과도하게 세팅한다면 Overfitting 문제가 발생할 수 있습니다.

1.1.14 SVM Overfitting

그림 23은 Overfitting 즉, 과최적화 문제가 발생된 대표적인 그림입니다. 여기서 과최적화란 무엇이고 그림이 왜 과최적화 되었다는 것인지에 대해 알아보도록 하겠습니다.

과최적화란 문자 그대로 너무 과도하게 데이터에 대해 모델을 학습한 경우를 의미합니다. 과도하게 학습된 모델은 학습한 데이터에 대해서는 분류를 비교적 정확하게 해내지만, 이 학습 데이터의 패턴을 벗어나는 새로운 데이터에 대해선 분류를 제대로 하지 못하는 문제가 있습니다. 즉, 학습데이터에만 최적화되어 있는 것이지요. 이때 현재에 대해 잘 설명하는 것만으로 충분하지 않을까? 라고 생각할 수 있습니다. 하지만 우리가 사실 원하는 정보는 기존에 알고 있는 데이터에 대한 것들이 아닙니다.

우리가 알고자 하는 것은 새로운 데이터에 대한 것들에 대한 정보입니다.

정작 새로운 데이터에 대해서 제대로 맞추지 못하는 문제가 발생하기 때문에 과최적화 문제는 피해줘야 할 문제입니다. 자, 그럼 그림에 대해서 살펴보도록 하겠습니다. 해당 그림은 과최적화 문제가 발생된 대표적인 그림입니다. 여기서 그림이 왜 과최적화일 까요?

해당 그림은 빨간 원과 하늘색 네모가 정확히 분류되어 있습니다. 이때 새로운 데이터 X축과 Y축이 겹치는 0, 0 좌표에 하늘색 네모가 입력된다고 가정하도록 하겠습니다. 가정한 데이터를 그림에서 확인하시면 빨간 원으로 분류되는 것을 확인할 수 있습니다. 이렇게 과최적화 된 모델은 분류하고자 하는 데이터가, 학습된 데이터와 조금이라도 차이가 난다면 오분류를 하는 문제점이 있습니다.

그림과 같은 과최적화가 일어나지 않도록 하려면 영향을 주는 SVM parameter들의 값을 적절히 조정을 해주는 것이 중요합니다. 그림 23은 SVM Overfitting에 대한 내용입니다.

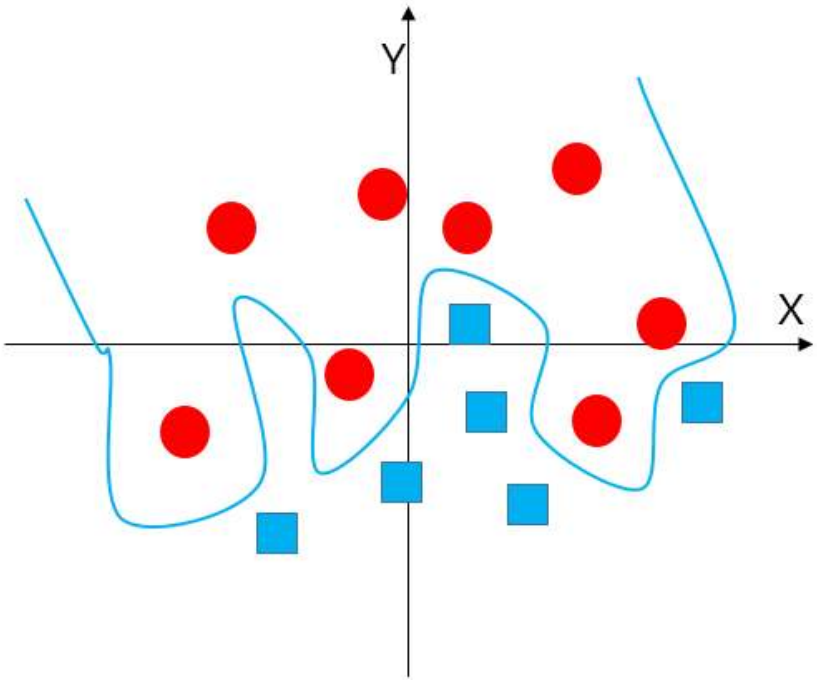


그림 23 SVM Overfitting

2. Support Vector Machine 실습

학습한 내용을 실습하는 예제 코드를 살펴보도록 하겠습니다. 실습은 먼저 실제 데이터를 SVM 모델을 이용해 분류를 해보고, 모델의 Parameter를 조정하는 순으로 진행하도록 하겠습니다.

2.1 Python package 로드

우선 사용할 패키지들을 가져오도록 하겠습니다. 첫 번째 줄에서는 numpy 패키지를 np라는 이름으로 가져옵니다. Numpy 라이브러리는 이전 강의에서 배운 것처럼 행렬, 벡터 등의 수학 계산을 위한 자료구조와 계산 함수를 제공하는 패키지입니다. 두 번째 줄에서는 Pandas 패키지를 pd라는 이름으로 가져옵니다.

Pandas패키지는 CSV파일 또는 데이터베이스에서 데이터를 읽고 쓸 수 있고 또한 데이터를 쉽게 조작할 수 있어 데이터 분석, 가공, 처리에 자주 사용되는 패키지입니다. 세 번째 줄은 Matplotlib의 서브패키지인 pyplot을 “plt” 라는 이름으로 가져오는 코드입니다. Matplotlib은 막대그래프, 히스토그램, 파이차트, 산점도 등 다양한 그림을 그리는데 사용되는 파이썬의 시각화 패키지입니다.

네 번째 줄은 Sklearn 패키지에서 제공하는 svm과 datasets 모듈을 가져옵니다. sklearn 패키지는 분류, 회귀, 군집 등의 문제에 대한 다양한 알고리즘을 제공하고 있습니다. 여기서 svm은 Support Vector Machine 모델을 생성하고 분류, 예측, 성능평가를 진행하게 도와주는 모듈입니다.

datasets은 sklearn 패키지의 open dataset을 로드할 때 사용하는 모듈입니다. 대표적인 open dataset은 Iris, tennis, titanic, boston 데이터들이 있습니다. 다섯 번째 줄은 matplotlib의 결과를 ipython notebook 안에서 출력하기 위해 사용하는 명령어입니다. 그림 그림 24 필요한 package 로드는 필요한 package 로드에 대한 내용입니다.

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from sklearn import svm, datasets
5 %matplotlib inline
```

그림 24 필요한 package 로드

2.2 Iris data set 로드

이제 사용할 데이터를 불러오도록 하겠습니다. 첫 번째 줄에서는 datasets 모듈을 통해 붓꽃 데이터를 가져와 iris 변수에 저장합니다. 이 때 붓꽃 데이터는 dictionary 자료형으로 이루어져 있습니다. 두 번째 줄에서는 dictionary 자료형을 갖는 붓꽃 데이터의 key값들을 출력합니다.

세 번째 줄은 붓꽃 데이터 중 data의 전체 행과 열의 길이를 출력합니다. 출력 결과를 보면 붓꽃 데이터는 150개의 row와 4개의 column으로 구성되어 있는 것을 알 수 있습니다. 네 번째 줄은 붓꽃 데이터 중 feature_names 값들을 출력합니다. 그림 그림 25 Iris data set 로드는 iris data set 로드하는 내용입니다.

```
1 iris = datasets.load_iris()
2 print(iris.keys())
3 print(iris.data.shape)
4 print(iris.feature_names)
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
(150, 4)
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

그림 25 Iris data set 로드

2.3 Iris data set 정보 확인

이번에는 붓꽃 데이터 셋에 대한 정보를 출력해 보도록 하겠습니다. Sklearn에서 제공하는 open dataset은 DESCR() 함수를 통해 데이터 셋에 대한 정보를 같이 제공합니다. 출력된 결과물을 확인해보면 각 feature들에 대한 설명과 길이가 적힌 것을 확인할 수 있습니다. 그림 그림 26 Iris data set 정보 확인은 iris data set 정보 확인에 대한 내용입니다.

```
1 print(iris.DESCR)
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
```

그림 26 Iris data set 정보 확인

2.4 데이터 학습

다음은 SVM모델을 만들어 데이터를 학습시켜보도록 하겠습니다. 첫 번째 줄에서는 iris 변수에 전체 데이터 중 data에 해당하는 값 중 2개 열에 전체 행 값을 x 변수에 저장합니다. 두 번째 줄은 iris 변수의 target에 해당하는 값을 y 변수에 저장합니다. 여기서 y값은 예측하고자 하는 값입니다.

세 번째 줄은 svm.SVC() 함수를 통해 kernel parameter를 linear로 지정하고 C 값을 1로 지정한 SVM모델을 만들고, fit() 함수를 통해 x와 y값을 학습한 후 SVM 변수 안에 저장합니다. 그림 그림 27 데이터 학습은 데이터 학습에 대한 내용입니다.

```
1 x = iris.data[:, :2]
2 y = iris.target
3 SVM = svm.SVC(kernel='linear', C=1).fit(x, y)
```

그림 27 데이터 학습

2.5 데이터 시각화 전처리

다음은 학습된 SVM 모델을 그래프로 출력하기 위해 전처리를 해보도록 하겠습니다. 첫 번째 줄은 x변수에 저장된 첫 번째 컬럼 값에 최솟값과 최댓

값을 구한 후 -1, +1을 해주어 plot의 처음과 끝 값을 구한 후 x_min, x_max 변수에 저장합니다. 해당 변수 값들은 X축의 범위를 설정할 때 사용합니다.

두 번째 줄은 y변수에 저장된 두 번째 컬럼 값에 최솟값과 최댓값을 구한 후, -1, +1을 해주어 plot의 처음과 끝 값을 구한 후 y_min, y_max 변수에 저장합니다. 마찬가지로 해당 변수 값들은 y축의 범위를 지정할 때 사용합니다.

그리고 세 번째 줄에서는 x_max 값을 x_min 값으로 나눈 후 100으로 다시 나누어 나온 0.025 값을 plot_unit 변수에 저장합니다. 해당 값은 축의 단위를 설정해주는 변수로, 축의 단위는 plot을 그릴 때 배경에 세밀한 정도를 나타내기 때문에 그 수치를 낮게 잡아서 세밀하게 확인합니다.

네 번째 줄에서는 np.arrange() 함수를 통해 x_min, y_min 값에서 x_max, y_max 값까지 plot_unit 값만큼 균등하게 간격을 둔 1차원 배열형태의 데이터들 만듭니다. 만들어진 1차원 배열형태의 데이터를 np.meshgrid 함수를 통해 2차원 배열형태로 교체한 후 xx, yy 변수에 저장해줍니다. xx 변수는 x가 y차원 크기의 행의 개수만큼 반복된 값을 갖고, yy 변수는 y가 x차원 크기의 열의 개수만큼 반복된 값을 가지고 있습니다. 그림 그림 28 데이터 시각화 전처리은 데이터 시각화 전처리에 대한 내용입니다.

```
1 x_min, x_max = x[:, 0].min()-1, x[:, 0].max()+1
2 y_min, y_max = x[:, 1].min()-1, x[:, 1].max()+1
3 plot_unit = 0.025
4 xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_unit), np.arange(y_min, y_max, plot_unit))
```

그림 28 데이터 시각화 전처리

2.6 데이터 시각화 및 성능 측정

이제 데이터를 시각화 하고 성능을 측정 해보도록 하겠습니다. 첫 번째 줄에서는 numpy.ravel 함수를 통해 xx와 yy 변수에 저장된 행렬의 1행부터 순차적으로 값을 불러와서 1차원 배열을 만듭니다. 이렇게 만든 배열을 np.c_[] 함수를 통해 배열에 열을 추가한 후, 학습된 SVM 모델에 데이터를 입력해 입력된 데이터가 어떤 class 값인지 분류한 값을 z 변수에 저장합니다.

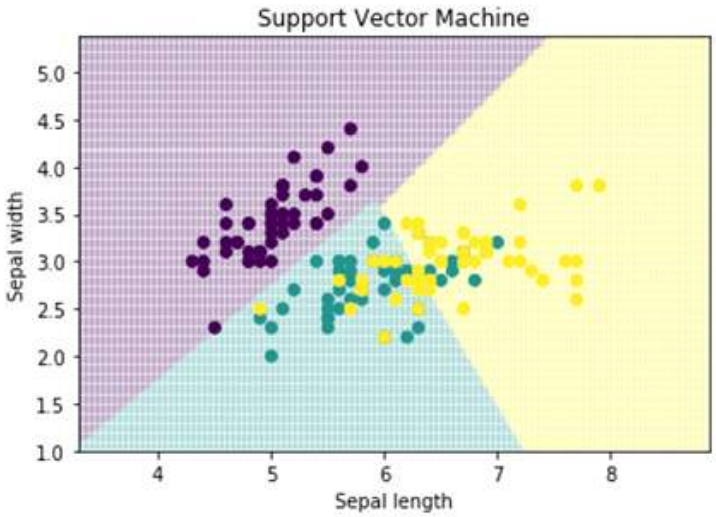
두 번째 줄에서는 Z 변수의 차원을 xx 변수의 차원과 같은 차원으로 재형

성한 후 다시 Z 변수에 저장해줍니다. 해당 차원을 같게 만들어주는 이유는 plot을 할 때 같은 차원의 데이터를 사용해야하기 때문입니다. 세 번째 줄은 pcolormesh()함수를 통해 xx, yy와 z 값을 입력한 후, 입력 받은 3개의 데이터를 그래프로 나타냅니다.

네 번째 줄은 x변수 첫 번째 컬럼에 저장된 데이터를 x축 두 번째 컬럼을 y축으로 하여 산점도를 그립니다. 다섯 번째 줄에서는 x축의 이름을 설정해주고, 여섯 번째 줄에서는 y축의 이름을 설정합니다. 일곱 번째 줄은 plot으로 보여줄 x축의 범위를 xx.min()값에서 xx.max()값을 지정해서 보여줍니다.

그리고 여덟 번째 줄은 plot에 제목을 설정하고, 아홉 번째 줄은 입력된 plot을 화면에 출력합니다. 열 번째 줄에서는 학습된 SVM 모델을 통해 입력된 데이터를 분류한 후 실제 값과 비교해 분류 정확도를 출력하였습니다. 다음의 그림처럼 결과적으로 커널 parameter를 linear로 주었기 때문에 선으로 데이터를 분류한 결과를 확인할 수 있습니다. 그림 29는 데이터 시각화 및 성능 측정 데이터 시각화 및 성능 측정에 대한 내용입니다.

```
z = SVM.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z, alpha=0.1)
plt.scatter(x[:, 0],x[:, 1],c=y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('Support Vector Machine')
plt.show()
print('정확도 : ',SVM.score(X = x, y = y))
```



정확도 : 0.82

그림 29 데이터 시각화 및 성능 측정

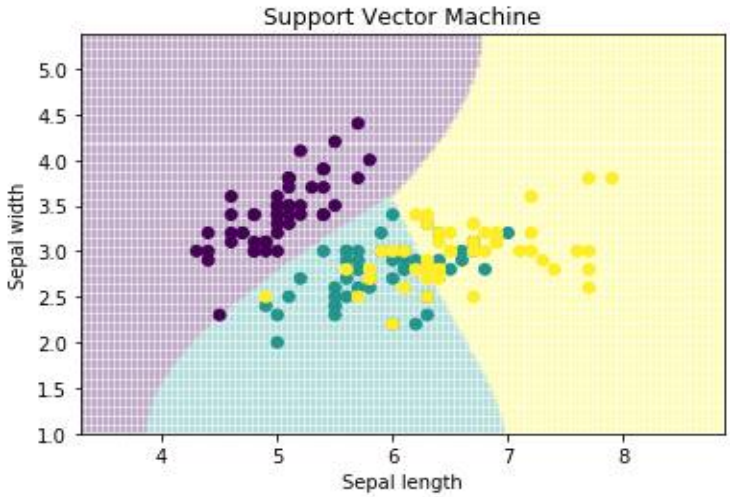
이번에는 svm.SVC()함수의 파라미터 값을 조정한 후 성능 측정 결과와 plot의 변화를 확인해 보겠습니다.

3. SVM의 Parameter 조정하는 방법 실습

3.1 데이터 파라미터 조정 및 성능 측정 (kernel = rbf, C = 1)

코드설명은 앞과 동일하기 때문에 생략하도록 하겠습니다. 자 그럼 첫 번째 매개변수 조정을 해보도록 하겠습니다. 먼저 커널 파라미터를 linear에서 rbf kernel로 교체했습니다. 결과를 확인하시면 선형이 아닌 비선형으로 분류된 것을 확인할 수 있습니다. 그림 30은 kernel 파라미터를 linear에서 rbf로 조정한 후 성능 측정에 대한 결과입니다.

```
SVM = svm.SVC(kernel='rbf', C=1).fit(x, y)
z = SVM.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z,alpha=0.1)
plt.scatter(x[:, 0],x[:, 1],c=y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('Support Vector Machine')
plt.show()
print('정확도 : ',SVM.score(X = x, y = y))
```



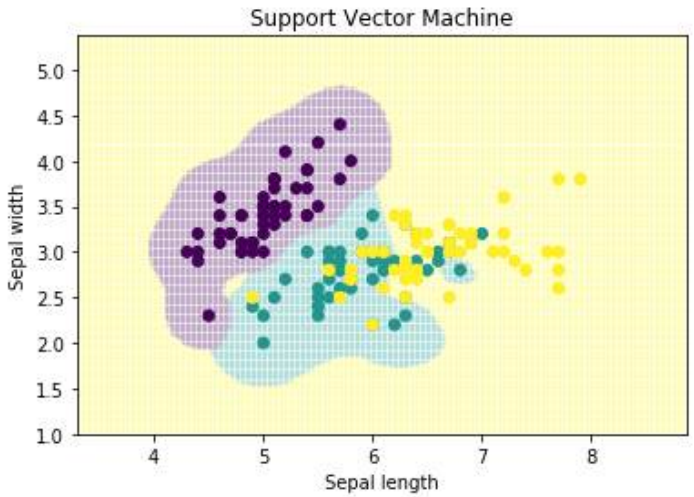

```
정확도 :  0.826666666667
```

그림 30 kernel 파라미터를 조정한 후 성능 측정 결과

3.2 데이터 파라미터 조정 및 성능 측정 (kernel = rbf, C = 1, gamma = 10)

다음은 gamma 값을 변경해보도록 하겠습니다. gamma값은 값을 낮추면 멀리 떨어진 서포트 벡터들의 영향이 낮고, 값을 높이면 멀리 떨어진 벡터들의 값이 영향을 크게 하는 역할을 하는 파라미터입니다. Gamma 값을 10을 줘보도록 하겠습니다. 결과를 확인해 보시면 margin이 작아진 것을 확인할 수 있습니다. 그림 31은 gamma 파라미터를 10으로 조정한 후 성능 측정에 대한 결과입니다.

```
SVM = svm.SVC(kernel='rbf', C=1, gamma=10).fit(x, y)
z = SVM.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z, alpha=0.1)
plt.scatter(x[:, 0],x[:, 1],c=y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('Support Vector Machine')
plt.show()
print('정확도 : ',SVM.score(X = x, y = y))
```



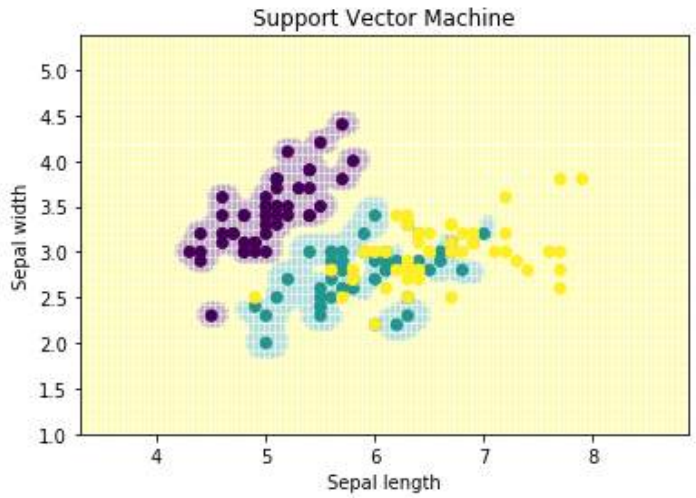
정확도 : 0.84

그림 31 gamma 파라미터를 조정한 후 성능 측정 결과 (1)

3.3 데이터 파라미터 조정 및 성능 측정 (kernel = rbf, C = 1, gamma = 100)

Gamma 값을 100을 줘보도록 하겠습니다. 결과를 확인해 보시면 margin이 매우 많이 작아진 것을 확인할 수 있습니다. 또한 앞에서 배운 Overfitting문제가 있는 것을 확인할 수 있습니다. 적당한 값 10을 줬을 때는 성능이 향상되었지만 과도하게 100이란 수치를 주면 물론 성능이 훨씬 좋아졌지만 Overfittig문제가 발생된 것을 확인했습니다. 이 문제가 중요한 이유는 새로운 값을 분류할 때 살짝 벗어나도 다른 것으로 오분류 하기 때문입니다. 그림 32는 gamma 파라미터를 100으로 조정한 후 성능 측정에 대한 결과입니다.

```
SVM = svm.SVC(kernel='rbf', C=1, gamma=100).fit(x, y)
z = SVM.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z,alpha=0.1)
plt.scatter(x[:, 0],x[:, 1],c=y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('Support Vector Machine')
plt.show()
print('정확도 : ',SVM.score(X = x, y = y))
```



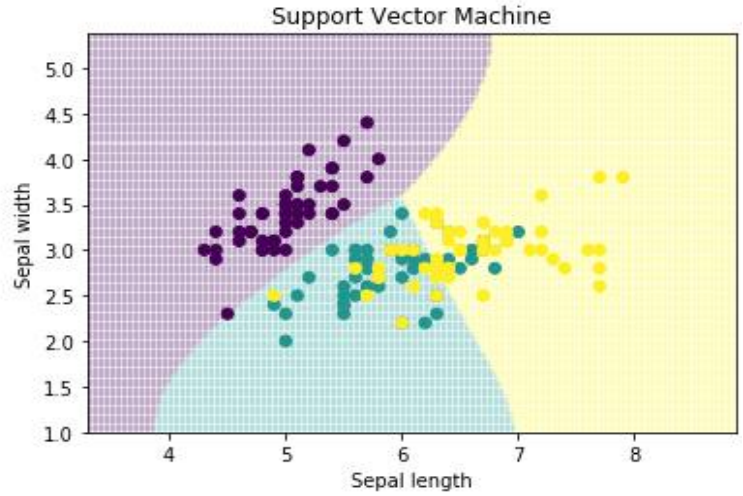
정확도 : 0.92

그림 32 gamma 파라미터를 조정한 후 성능 측정 결과 (2)

3.4 데이터 파라미터 조정 및 성능 측정 (kernel = rbf, C = 1, gamma = auto)

다음으론 C 파라미터 값을 조절해서 결과를 확인해보도록 하겠습니다. C 값은 디폴트로 1.0 값을 갖는데, 이 값을 낮추면 초평면이 매끄러워지지만 값을 높이면 서포트 벡터들을 더욱 세밀하게 분류하게 됩니다. C값을 1로 설정해 보겠습니다. 다음과 같이 초평면이 매끄럽게 분류하는 것을 확인할 수 있습니다. 그림 33은 C 파라미터를 1로 조정한 후 성능 측정에 대한 결과입니다.

```
SVM = svm.SVC(kernel='rbf', C=1).fit(x, y)
z = SVM.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z,alpha=0.1)
plt.scatter(x[:, 0],x[:, 1],c=y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('Support Vector Machine')
plt.show()
print('정확도 : ',SVM.score(X = x, y = y))
```



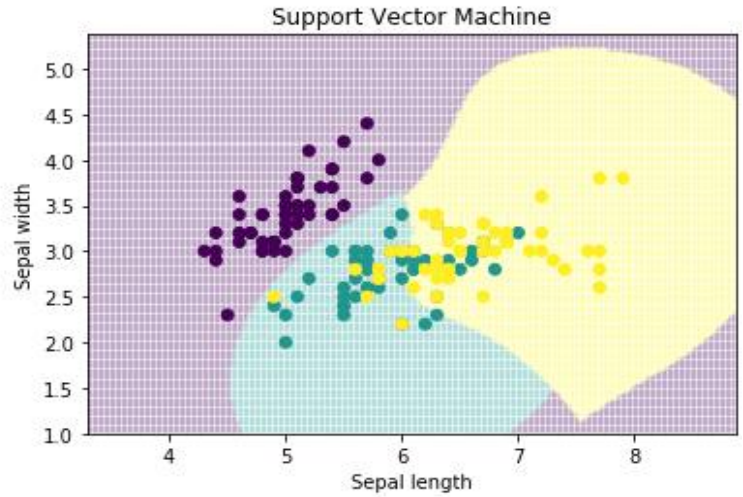

```
정확도 :  0.826666666667
```

그림 33 C 파라미터를 조정한 후 성능 측정 결과 (1)

3.5 데이터 파라미터 조정 및 성능 측정 (kernel = rbf, C = 100, gamma = auto)

C값을 100으로 설정하면 초평면이 더욱 작아지면서 마진이 작아진 것을 확인할 수 있습니다. 이유는 C값을 조절하면 멀리 있는 벡터도 서포트 벡터로 설정을 해주기 때문입니다. 그림 34는 C 파라미터를 100으로 조정한 후 성능 측정에 대한 결과입니다.

```
SVM = svm.SVC(kernel='rbf', C=100).fit(x, y)
z = SVM.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z,alpha=0.1)
plt.scatter(x[:, 0],x[:, 1],c=y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('Support Vector Machine')
plt.show()
print('정확도 : ',SVM.score(X = x, y = y))
```



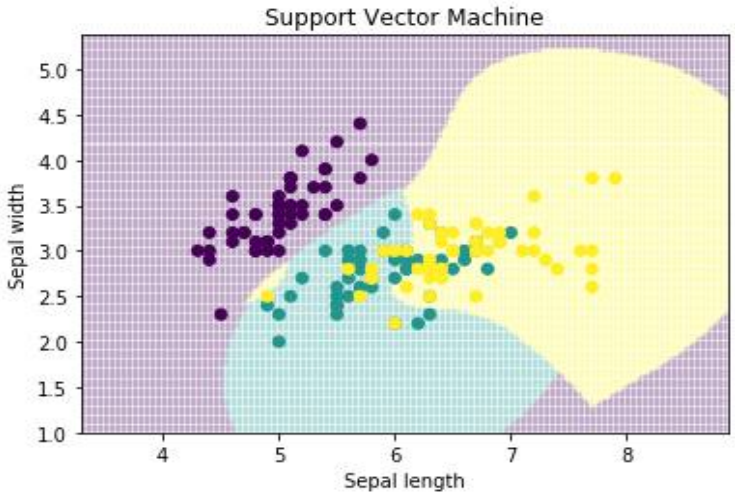
정확도 : 0.82

그림 34 C 파라미터를 조정한 후 성능 측정 결과 (2)

3.6 데이터 파라미터 조정 및 성능 측정 (kernel = rbf, C = 1000, gamma = auto)

이번에는 C값을 1000으로 설정해보도록 하겠습니다. 결과를 확인해 보시면 C값을 100으로 줬을 때와 유사한 결과를 확인할 수 있습니다. 이렇게 Parameter를 조정하는 실습을 통해 Parameter를 높게 한다면 성능을 높일 수 있지만 과도하게 높게 한다면 Overfitting 문제가 발생하는 것을 확인했습니다. 그림 35는 C 파라미터를 1000으로 조정한 후 성능 측정에 대한 결과입니다.

```
SVM = svm.SVC(kernel='rbf', C=1000).fit(x, y)
z = SVM.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z,alpha=0.1)
plt.scatter(x[:, 0],x[:, 1],c=y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('Support Vector Machine')
plt.show()
print('정확도 : ',SVM.score(X = x, y = y))
```



정확도 : 0.813333333333

그림 35 C 파라미터를 조정한 후 성능 측정 결과 (3)