SVR, KFDA, KPCA - 2021 P-SAT PaperStudy 6회차

권남택, 오정민, 유경민

2021년 3월 11일

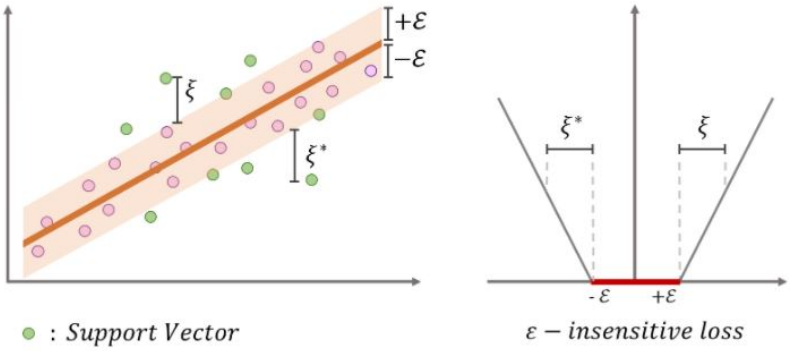
해당 내용은 'An Introduction to Kernel-Based Learning Algorithms'논문을 바탕으로 SVR, KFDA, KPCA에 대한 내용을 소개하는 자료입니다. SVM을 비롯한 기본적인 Background 내용들은 '1회차 SVM'때 다루었기에 별도로 다루지는 않았습니다.

1. SVR

1.1 Fitting function

svr에 대해 다루기에 앞서 함수를 적합시키는 것의 목표 두가지를 살펴보자. 함수 피팅은 1) loss를 최소화하도록하는 것, 2) 같은 성능이라면 단순한 함수 형태이도록 하는 목표를 갖고 있다. 예를 들어 ridge regression의 목적함수는 $min\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)+\lambda\sum_{i=1}^{d}\hat{\beta}_d^{\,2}$ 인데, 여기서 $\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)$이 loss를 최대화하는 부분이고, $\lambda\sum_{i=1}^{d}\hat{\beta}_d^{\,2}$ 은 모델이 덜 복잡한 형태를 갖도록 하는 부분이다.

1.2 SVR loss function



ξ는 회귀식 위쪽에서 튜브 밖을 벗어난 거리를, ξ^*는 회귀식 아래쪽에서 튜브 밖에 벗어난 거리를, ϵ은 허용하는 노이즈의 정도를의미한다. svr은 데이터에 노이즈가 있다고 생각하며, 이를 고려하여 노이즈가 있는 실제 값을 완벽하게 추정하려 하지 않고 적정 범위 내에서는 차이를 허용한다. 왼쪽 그림에서 회귀식 위 아래 $\pm\epsilon$ 만큼의 부분을 ϵ tube라고 한다. 이번에 다룰 가장 기본적인 ϵ-SVR에서는 hinge loss를 사용하여 $\pm\epsilon$만큼의 에러는 허용하고, 그 이상의 에러에 대해서는 C의 배율로 loss를 부과하는 방법을 택한다. squared loss를 사용하는 일반적인 선형회귀와 달리 hinge loss를 사용함으로써 이상치와 노이즈에 대해 더욱 로버스트한 모델링이 가능하다. 아래와 같이 ϵ-insensitive 외에도 다양한 loss function이 존재한다.

	loss function	density model
ε–insensitive	$c(\xi) = \lvert\xi\rvert_\varepsilon$	$p(\xi) = \frac{1}{2(1+\varepsilon)} \exp(-\lvert\xi\rvert_\varepsilon)$
Laplacian	$c(\xi) = \lvert\xi\rvert$	$p(\xi) = \frac{1}{2} \exp(-\lvert\xi\rvert)$
Gaussian	$c(\xi) = \frac{1}{2}\xi^2$	$p(\xi) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{\xi^2}{2})$
Huber's robust loss	$c(\xi) = \begin{cases} \frac{1}{2\sigma}(\xi)^2 & \text{if } \lvert\xi\rvert \le \sigma \\ \lvert\xi\rvert - \frac{\sigma}{2} & \text{otherwise} \end{cases}$	$p(\xi) \propto \begin{cases} \exp(-\frac{\xi^2}{2\sigma}) & \text{if } \lvert\xi\rvert \le \sigma \\ \exp(\frac{\sigma}{2} - \lvert\xi\rvert) & \text{otherwise} \end{cases}$
Polynomial	$c(\xi) = \frac{1}{p}\lvert\xi\rvert^p$	$p(\xi) = \frac{p}{2\Gamma(1/p)} \exp(-\lvert\xi\rvert^p)$
Piecewise polynomial	$c(\xi) = \begin{cases} \frac{1}{p\sigma^{p-1}}(\xi)^p & \text{if } \lvert\xi\rvert \le \sigma \\ \lvert\xi\rvert - \sigma\frac{p-1}{p} & \text{otherwise} \end{cases}$	$p(\xi) \propto \begin{cases} \exp(-\frac{\xi^p}{p\sigma^{p-1}}) & \text{if } \lvert\xi\rvert \le \sigma \\ \exp(\sigma\frac{p-1}{p} - \lvert\xi\rvert) & \text{otherwise} \end{cases}$

1.3 SVR formulation

1.3.1 Original Problem

아래의 목적식을 만족하는 선형회귀식 $f(\mathsf{x}) = \mathsf{w}^T\mathsf{x} + b$을 찾아야한다.

$$\min \quad \frac{{||\mathsf{w}||_2}^2}{2} + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) s.t.\ (\mathsf{x_i} \cdot \mathsf{w} + b) - y_i \le \epsilon + \xi_i y_i - (\mathsf{x_i} \cdot \mathsf{w} + b) \le \epsilon + \xi_i^* \xi_i, \xi_i^* \ge 0$$

여기서 $\frac{{||\mathsf{w}||_2}^2}{2}$ 부분이 ridge에서 $\lambda\sum_{i=1}^{d}{\hat{\beta}_d}^2$ 부분과 같은 기능으로, 회귀선의 복잡도와 관련되어 있다. 또, $C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$ 부분은 ridge에서 $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})$와 같은 기능으로, loss를 줄이는 것과 관련있는 부분이다. SVM에서는 $\frac{{||\mathsf{w}||_2}^2}{2}$가 margin을 최대화하는 부분이었다면, SVR에서는 좀 더 평평하고 단순한 회귀선을 갖도록 하는 부분이 된다.

1.3.2 Lagrange Primal Problem

목적식에 제약식과 라그랑주 승수를 곱한 항을 더하여 제약이 없는 Lagrange Primal 문제로 변환하여준다.

$$\begin{aligned} L_p =& \frac{{||\mathsf{w}||_2}^2}{2} + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) - \sum_{i=1}^{n}(\eta_i\xi_i + \eta^*\xi^*) \\ & - \sum_{i=1}^{n}\alpha_i(\epsilon + \xi_i + y_i - \mathsf{w}^T\mathsf{x_i} - b) - \sum_{i=1}^{n}\alpha_i^*(\epsilon + \xi_i^* - y_i + \mathsf{w}^T\mathsf{x_i} + b) \end{aligned}$$

$$\alpha_i^{(*)}, \eta_i^{(*)} \ge 0$$

KKT조건에 의해 목적식의 미지수에 대해 미분 값이 0일때 최적해를 갖게된다. 따라서 $b, \mathsf{w}, \xi_i^{(*)}$에 대해 미분한다.

$$\frac{\partial L_p}{\partial \mathsf{w}} = \mathsf{w} - \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\mathsf{x_i} = 0 \rightarrow \mathsf{w} = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\mathsf{x_i} \quad \frac{\partial L_p}{\partial b} = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0 \quad \frac{\partial L_p}{\partial \xi_i^{(*)}} = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0$$

1.3.3 Lagrange Dual Problem

$b, \mathsf{w}, \xi_i^{(*)}$에 대해 미분하여 정리한 최적해 조건을 L_p에 대입하여 α에 대한 dual 문제로 정리한다.

$$L_D = -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\mathrm{x_i}^T\mathrm{x_j} - \epsilon\sum_{i,j=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i,j=1}^{n} y_i(\alpha_i^* - \alpha_i)$$

$$s.t.\sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0, \quad \alpha_i, \alpha_i^* \in [0, C]$$

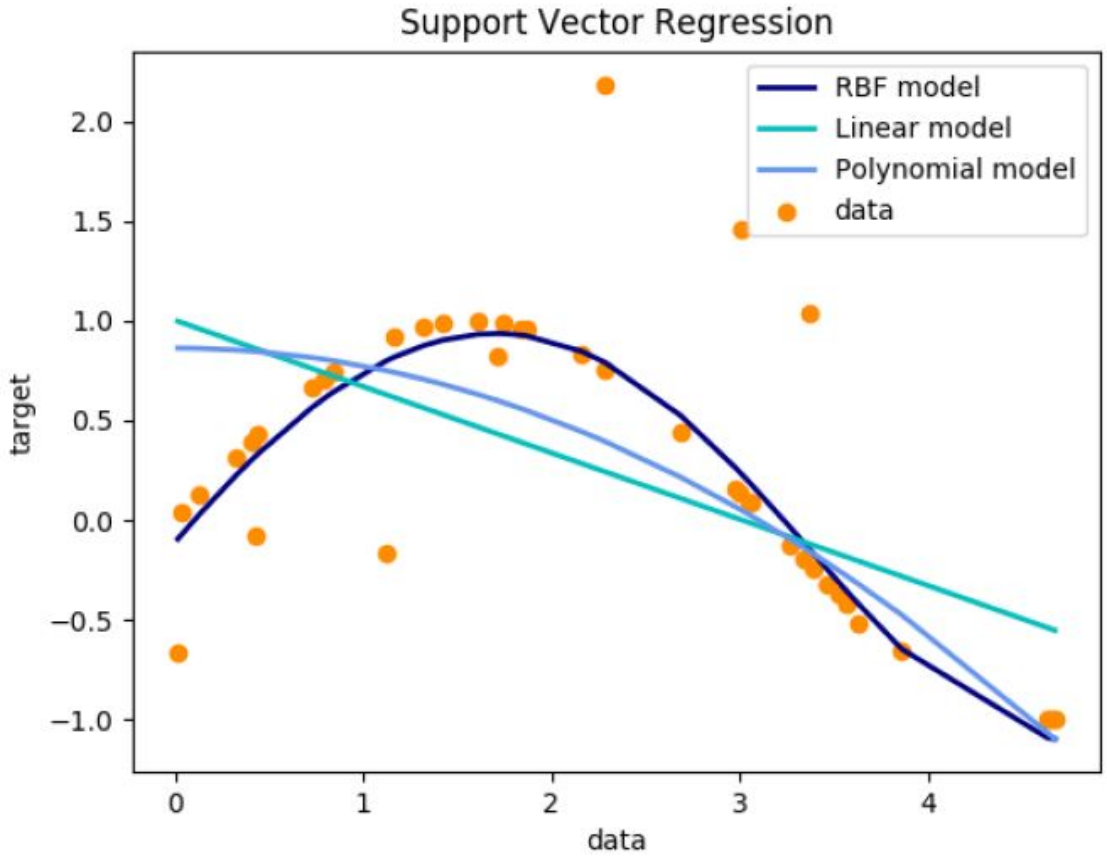
그 결과 구해진 α_i, α_i^*를 대입하여 우리가 구하고자한 w를 $\mathrm{w} = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\mathrm{x_i}$로 구할 수 있고, 회귀식은 $f(x) = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\mathrm{x_i}^T\mathrm{x} + b$이다. 여기서 $\alpha > 0$인 데이터들이 support vector가 되고, bias는 $b = f(x_{xv}) - \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\mathrm{x_i}^T\mathrm{x_{sv}}$로 계산된다.

SVM과 마찬가지로 kernel trick을 사용해 비선형회귀식을 구할수도 있다. 이 경우 dual 식은 다음과 같다.

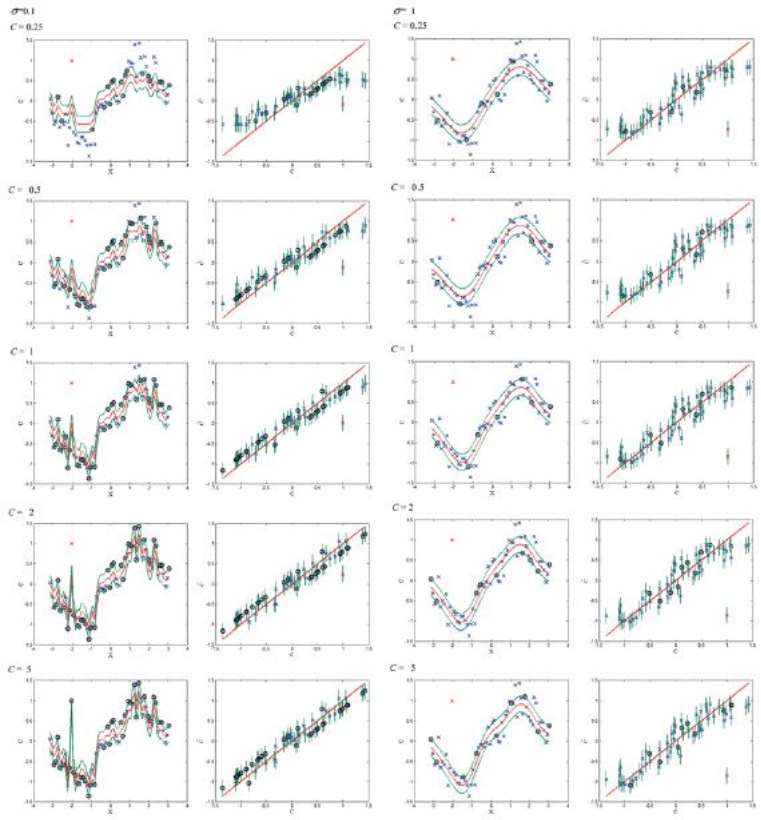
$$L_D = -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(\mathrm{x_i}, \mathrm{x_j}) - \epsilon\sum_{i,j=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i,j=1}^{n} y_i(\alpha_i^* - \alpha_i)$$

$$s.t.\sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0, \quad \alpha_i, \alpha_i^* \in [0, C]$$

그 결과 구해진 α_i, α_i^*를 대입하여 우리가 구하고자한 w를 $\mathrm{w} = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\Phi(\mathrm{x_i})$로 구할 수 있고, 회귀식은 $f(x) = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)K(\mathrm{x_i}, \mathrm{x}) + b$이다. kernel function을 이용한 svr은 아래 그림과 같이 비선형적 형태를 가질 수 있다.



1.4 SVR with different sigma and C combination



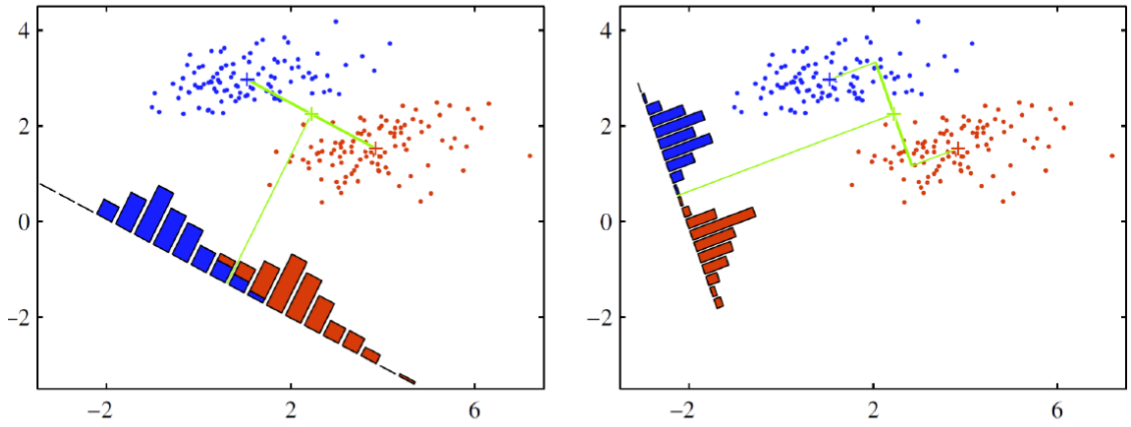
동일한 C에 대해서 σ가 더 작을수록 더 복잡한 형태를 갖게 되고, 동일한 σ에 대해서 C가 클수록 더 복잡한 형태를 갖는다.

2. KFDA

KFDA는 Kernel Fisher Disciminant Analysis의 줄임말이다. 판별분석에 커널버전이다. 따라서 판별분석에 대해 먼저 다루고, KFDA를 다루자.

2.1 Linear Discriminant Analysis (선형판별분석)

2.1.1 LDA의 아이디어

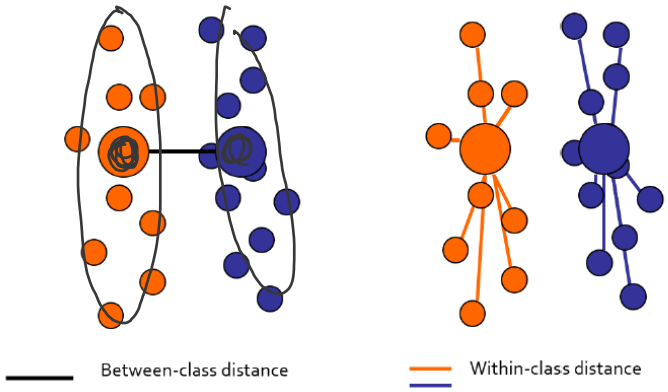


LDA는 두 클래스를 가장 잘 구분하는 하나의 축을 찾는 것이다. PCA같은 경우에는 분산을 최대로 보존하는 여러개의 축을 찾았지만, LDA는 단 하나의 축에 Projection 시킨다. 사진처럼 하나의 축에 Projection 시키는 것은 되게 다양할 수 밖에 없는데, 어떻게 해야 최적의 축(기저, Basis)를 찾을 수 있을까?

두 개의 클래스가 있다고 가정하자. 왼쪽의 사진처럼 축을 찾으면, 두 클래스가 겹치는 부분이 많다. 반면에 오른쪽 사진처럼 축을 잡으면, 두 클래스가 비교적 명확하게 나뉘는 것을 확인할 수 있다. 두 클래스는 명확하게 나뉘면서, 동시에 같은 클래스는 충분히 밀집해있어야 좋은 분류가 가능하다고 직관적으로 파악 가능하다.

이를 LDA의 언어로 말하자면,

1) 다른 클래스 사이의 거리(분산)는 최대화 하면서,
2) 같은 클래스 사이의 거리(분산)는 최소화해야 한다.



2.1.2 Notation

우리가 찾아야하는 축은 w이다. x를 새로운 기저로 Projection을 시키면, 다음과 같다.

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

각 클래스의 중심은 다음과 같이 정의하자.

$$\mathsf{m}_1 = \frac{1}{N_1}\sum_{n\in C_1}\mathsf{x}_n, \qquad \mathsf{m}_2 = \frac{1}{N_2}\sum_{n\in C_2}\mathsf{x}_n$$

1) 먼저 다른 클래스 사이의 거리를 계산하자.

$$m_2 - m_1 = \mathsf{w}^T(\mathsf{m}_2 - \mathsf{m}_1)$$

2) 그 다음 각각 클래스 내의 분산을 계산하자.

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

클래스 사이의 거리는 커져야하고, 클래스 내의 분산은 작아져야하기 때문에, 각각을 분자와 분모로 넣으면 최대화하는 문제로 바꿀수 있다.

$$J(\mathsf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{\mathsf{w}^T\mathsf{S}_B\mathsf{w}}{\mathsf{w}^T\mathsf{S}_W\mathsf{w}}$$

$$\mathsf{S}_B = (\mathsf{m}_2 - \mathsf{m}_1)(\mathsf{m}_2 - \mathsf{m}_1)^T$$

$$\mathsf{S}_W = \sum_{n\in C_1}(\mathsf{x}_n - \mathsf{m}_1)(\mathsf{x}_n - \mathsf{m}_1)^T + \sum_{n\in C_2}(\mathsf{x}_n - \mathsf{m}_2)(\mathsf{x}_n - \mathsf{m}_2)^T$$

우리는 언제나 최대화/최소화 문제에서 미분값 = 0으로 놓고 문제를 푼다. 목적함수 J가 Convex 문제인지는 우리의 생각으로는 애매할 수 있지만, Convex 문제라고한다…!! 따라서 걱정없이 미분을 때려버리자! 지금의 형태는 $\frac{f}{g}$의 상황이기 때문에, 이를 미분 때리면 $\frac{f'g - fg'}{g^2}$가 된다. 미분값=0으로 해버리면 다음과 같다.

$$(\mathsf{w}^T\mathsf{S}_B\mathsf{w})\mathsf{S}_W\mathsf{w} = (\mathsf{w}^T\mathsf{S}_W\mathsf{w})\mathsf{S}_B\mathsf{w}$$

근데 여기서 $\mathsf{w}^T\mathsf{S}_B\mathsf{w}$와 $\mathsf{w}^T\mathsf{S}_W\mathsf{w}$는 스칼라 값이다. $\mathsf{S}_B\mathsf{w}$의 경우는

$$\mathsf{S}_B\mathsf{w} = (\mathsf{m}_2 - \mathsf{m}_1)(\mathsf{m}_2 - \mathsf{m}_1)^T\mathsf{w} = \beta(\mathsf{m}_2 - \mathsf{m}_1)$$

와 같이 변경될 수 있다.

따라서 각각의 scalar 값을 걍 무시하면 다음과 같은 요약이 가능하다.

$$\mathsf{w} \propto \mathsf{S}_W^{-1}(\mathsf{m}_2 - \mathsf{m}_1)$$

우리가 찾고자하는 축 w는 위의 식에 비례하게 되고, 그냥 스칼라 값은 무시한다음 저 벡터를 유닛벡터로 만들어 주면 끝나게 된다!

2.1.3 LDA의 추가적인 설명

맨처음 LDA의 그림에서 projection 했을 때 이들의 분포가 일종의 정규분포와 같은 형태를 띄는 것을 확인할 수 있다. 이처럼 LDA는 data가 normally distributed되었다는 가정에 기반한다. 각각의 그룹 또한 동일한 공분산 구조를 가진다고 가정하기 때문에, 매우 제약이 쎈 모형이다. 클래스 별로 공분산 구조가 다르다고 가정할 경우, QDA (Quadratic Discriminant Analysis)가 된다. 이 경우에는 비선형적인 분리가 가능해진다.

그러면 QDA와 KFDA는 어쩌면 비슷하지 않을까, 특정 조건에서는 같지 않을까 생각도 잠깐 해봤는데(왜냐하면 2차원 혹은 3차원에서의 비선형 분리를 하게되면 시각적으로는 QDA나 KFDA나 큰 차이가 없지 않을까?), QDA의 Kernel 버전도 존재한다. 따라서 아니라고 생각해도 될 것 같다.

2.2 Kernel Fisher Disciminant Analysis

2.2.1 Notation

KFDA는 LDA의 커널화된 버전이다. $\mathsf{x} \rightarrow \Phi(\mathsf{x})$로, x를 더 고차원인 $\Phi(\mathsf{x})$ 상에서 projection을 진행한다. 해당 공간에 대한 공분산을 계산하면 다음과 같다.

$$\mathsf{C}^{\Phi} = \frac{1}{N}\sum_{n=1}^{N}(\Phi(\mathsf{x}_n) - \mathsf{m}^{\Phi})(\Phi(\mathsf{x}_n) - \mathsf{m}^{\Phi})^T, \qquad \mathsf{m}^{\Phi} = \frac{1}{N}\sum_{n=1}^{N}\Phi(\mathsf{x}_n)$$

더불어서 고차원으로 매핑된 공간에서의 '클래스 내의 분산'과 '클래스 간의 분산'을 계산하자.

$$\mathsf{S}_W^{\Phi} = \sum_{i=1,2}\sum_{n=1}^{N_i}(\Phi(\mathsf{x}_n^i) - \mathsf{m}_i^{\Phi})(\Phi(\mathsf{x}_n^i) - \mathsf{m}_i^{\Phi})^T$$

$$\mathsf{S}_B = (\mathsf{m}_2^{\Phi} - \mathsf{m}_1^{\Phi})(\mathsf{m}_2^{\Phi} - \mathsf{m}_1^{\Phi})^T$$

2.2.2 Objective Function

KFDA의 목적함수는 결국 LDA와 다르지 않지만, 분산 텀 부분만 달라진다.

$$J(\mathsf{w}) = \frac{\mathsf{w}^T\mathsf{S}_B^{\Phi}\mathsf{w}}{\mathsf{w}^T\mathsf{S}_W^{\Phi}\mathsf{w}}$$

더불어서 $\mathsf{w} = \sum_{n=1}^{N}\alpha_n\Phi(\mathsf{x}_n)$라고 두게 되면, mean을 다음과 같이 projection할 수 있다.

$$\begin{aligned}\mathsf{w}^T\mathsf{m}_i^{\Phi} &= \frac{1}{N_i}\sum_{n=1}^{N}\sum_{k=1}^{N_i}\alpha_n(\Phi(\mathsf{x}_n)\cdot\Phi(\mathsf{x}_k^i)) \\ &= \frac{1}{N_i}\sum_{n=1}^{N}\sum_{k=1}^{N_i}\alpha_n\mathsf{K}(\mathsf{x}_n,\mathsf{x}_k^i) = \alpha^T\mu_i\end{aligned}$$

결국 매핑된 공간에서 클래스의 중심은 다음과 같은 커널함수로 바로 나타난다.

$$(\mu_i)_n = \frac{1}{N_i}\sum_{k=1}^{N_i}\mathsf{K}(\mathsf{x}_n,\mathsf{x}_k^i)$$

그렇다면 이제 목적함수를 보자. 목적함수도 결국에는 커널함수로 바로 나타나기 때문에, 이를 미분하게 될 것이고 그 꼴이 커널 트릭의 존재 때문에 매우 간단할 것임을 예상할 수 있다.

목적함수의 분자부분을 먼저 보면,

$$\begin{aligned} \mathsf{w}^T \mathsf{S}_B^\Phi \mathsf{w} &= \mathsf{w}^T (\mathsf{m}_2^\Phi - \mathsf{m}_1^\Phi)(\mathsf{m}_2^\Phi - \mathsf{m}_1^\Phi)^T \mathsf{w} \\ &= \alpha^T (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \alpha \\ &= \alpha^T \mathsf{M} \alpha, \qquad \text{where } \mathsf{M} = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \end{aligned}$$

와 같이 정리될 수 있다.

더불어서 목적함수의 분모부분을 보면 조금 전개가 많이 귀찮다. 따라서 결과만 간단히 요약한다면

$$\begin{aligned} \mathsf{w}^T \mathsf{S}_W^\Phi \mathsf{w} &= \Big(\sum_{i=1}^{N} \alpha_i \Phi(\mathsf{x}_i)\Big) \Big(\sum_{j=1,2} \sum_{n=1}^{N_i} (\Phi(\mathsf{x}_n^j) - \mathsf{m}_j^\Phi)(\Phi(\mathsf{x}_n^j) - \mathsf{m}_j^\Phi)^T\Big) \Big(\sum_{k=1}^{N} \alpha_k \Phi(\mathsf{x}_k)\Big) \\ &= \sum_{j=1,2} \alpha^T \mathsf{K}_j \mathsf{K}_j^T \alpha - \alpha^T \mathsf{K}_j \, 1_{N_j} \, \mathsf{K}_j^T \alpha \\ &= \alpha^T \mathsf{N} \alpha, \qquad where \ \mathsf{N} = \sum_{j=1,2} \mathsf{K}_j (\mathsf{I} - 1_{N_j}) \mathsf{K}_j^T \end{aligned}$$

다음과 같다.

결국 우리의 목적함수는 다음과 같이 변경된다.

$$J(\alpha) = \frac{\alpha^T \mathsf{M} \alpha}{\alpha^T \mathsf{N} \alpha}$$

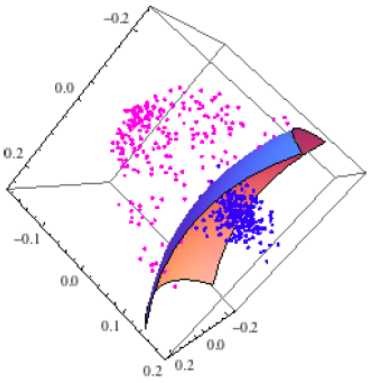
이 목적함수에 대해 미분하고 정리를 하면 이전과 결과가 비슷하다. 바로 넘어가보자.

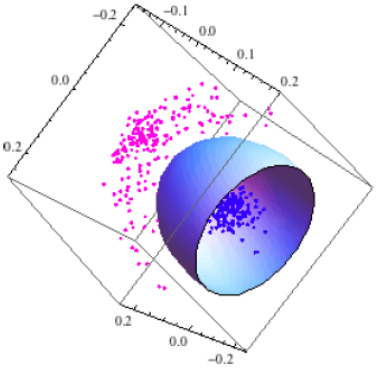
$$\alpha = \mathsf{N}^{-1} (\mathsf{M}_2 - \mathsf{M}_1)$$

왜 w가 아닌 α를 찾았는가? w는 고차원 공간에서 선형으로 사영하는 것이고, 이를 원래 공간에서 사영해주는 것이 바로 알파이다. 그래서 이를 새로운 데이터 포인트로 매핑하게 되면

$$y(\mathsf{x}) = (\mathsf{w} \cdot \Phi(\mathsf{x})) = \sum_{n=1}^{N} \alpha_n \mathsf{K}(\mathsf{x}_n, \mathsf{x})$$

이런 비선형적인 분리를 다음과 같이 시각화 할 수 있다. 왼쪽은 Polynomial Kernel을 사용한 것이고, 오른쪽은 Gaussian Kernel을 사용한 것이다.



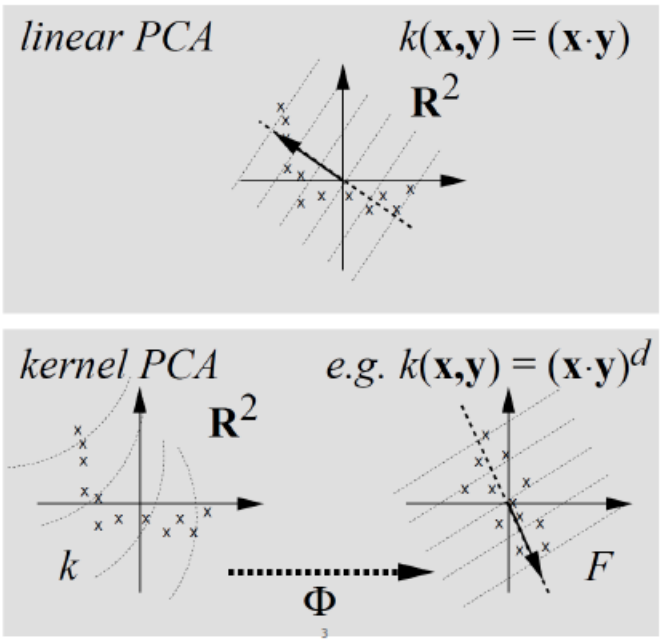


3. KPCA(Kernel Principal Component Analysis)

Kernel-based learning의 마지막 내용은 KPCA다. 이름에서도 알 수 있지만 우리가 잘 알고있는 PCA기법의 kernel 화 버전이라고 생각해주면 된다. PCA에 대해서는 여러분들이 잘 알것이라고 생각해서 PCA에 대한 자세한 설명은 생략하고 넘어가겠다!(P-SAT 선형대수학팀 클린업에도 잘 정리되어 있으니 참고!)

3.1 Basic Idea

다음의 간단한 예시를 살펴보자.



PCA의 경우 차원축소를 위한 선형변환 기법을 이용하는데, 위의 예시처럼 비선형 패턴을 가지고 있을때는 PCA를 적용하기에 적합하지 않다. 이 경우에 x를 적절한 $\Phi(x)$상으로 mapping을 한 후 해당 공간에서 PCA를 진행하면 좋은 결과를 얻을 수 있을 것이다.

3.2 Kernel PCA Procedure

가정: 계산의 편의성을 위해 mapping한 공간에서 x_i들의 평균은 0이라 가정한다.

$$\mathsf{m}^{\Phi} = \frac{1}{N}\sum_{i=1}^{N}\Phi(\mathsf{x}_i) = 0$$

해당 공간에 대한 공분산을 구하면 다음과 같다.

$$\mathsf{C}^{\Phi} = \frac{1}{N}\sum_{i=1}^{N}(\Phi(\mathsf{x}_i) - \mathsf{m}^{\Phi})(\Phi(\mathsf{x}_i) - \mathsf{m}^{\Phi})^T = \frac{1}{N}\sum_{i=1}^{N}\Phi(\mathsf{x}_i)\Phi(\mathsf{x}_i)^T$$

이때 eigenvector와 eigenvalue들은 다음과 같은 관계를 가진다.

$$\mathsf{C}^{\Phi}\mathsf{v}_k = \lambda_k\mathsf{v}_k$$

위에서 언급한 2개의 식에서 우리는 다음과 같이 쓸 수 있고,

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(\mathsf{x}_i)(\Phi(\mathsf{x}_i)^T\mathsf{v}_k) = \lambda_k\mathsf{v}_k \quad (1)$$

v_k에 대해서는 다음과 같이 적을수 있다.

$$\mathsf{v}_k = \frac{1}{N}\sum_{i=1}^{N}\alpha_{ki}\Phi(\mathsf{x}_i)$$

이렇게 구한 v_k를 다시 (1)번 식에 대입해주면, 우리는 다음과 같은 식을 얻을 수 있다.

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(\mathsf{x}_i)\Phi(\mathsf{x}_i)^T\sum_{j=1}^{N}\alpha_{kj}\Phi(\mathsf{x}_j) = \lambda_k\sum_{j=1}^{N}\alpha_{kj}\Phi(\mathsf{x}_j)$$

위의 식을 보면 kernel trick을 사용하고 싶다는 생각이 들 것이다. 그런데 이 kernel trick을 사용하기 위해서는 Φ에 대한 쌍을 만들어 주어야 한다. 해서 우리는 양변에 Φ를 곱해주는 것을 통해 kernel trick을 사용할 수 있는 환경을 조성할 수 있다. kernel function을 $\mathsf{K}(x_i, x_j) = \Phi(x_i)^T\Phi(x_j)$라 했을 때 우리는 다음과 같이 표현이 가능하다.

$$\frac{1}{N}\sum_{i=1}^{N}\Phi(\mathsf{x}_l)^T\Phi(\mathsf{x}_i)\sum_{j=1}^{N}\alpha_{kj}\Phi(\mathsf{x}_i)^T\Phi(\mathsf{x}_j) = \lambda_k\sum_{j=1}^{N}\alpha_{kj}\Phi(\mathsf{x}_l)^T\Phi(\mathsf{x}_j)$$

$$\frac{1}{N}\sum_{i=1}^{N}\mathsf{K}(\mathsf{x}_l, \mathsf{x}_i)\sum_{j=1}^{N}\mathsf{K}(\mathsf{x}_i, \mathsf{x}_j) = \lambda_k\sum_{i=1}^{N}\alpha_{kj}\mathsf{K}(\mathsf{x}_l, \mathsf{x}_i)$$

이를 matrix notation을 통해 표현하면 다음과 같다.

$$\mathsf{K}^2\alpha_k = \lambda_k N\mathsf{K}\alpha_k$$
$$\mathsf{K}\alpha_k = \lambda_k N\alpha_k$$

여기서 α_k는 kernel matrix K의 고유 벡터가 된다.

최종적으로 나오는 kernel PCA의 결과값 y의 k번재 기저에 projection한 값은 결국 kernel matrix형태가 되어 feature space로의 direct mapping이 필요없음을 알 수 있다. 즉 우리는 kernel function을 정해주기만 하면 된다는 것이다.

$$y_k(x) = \Phi(x)^T \mathrm{v}_k = \sum_{i=1}^{N} \alpha_{ki} \mathrm{K}(\mathrm{x}, \mathrm{x}_i)$$

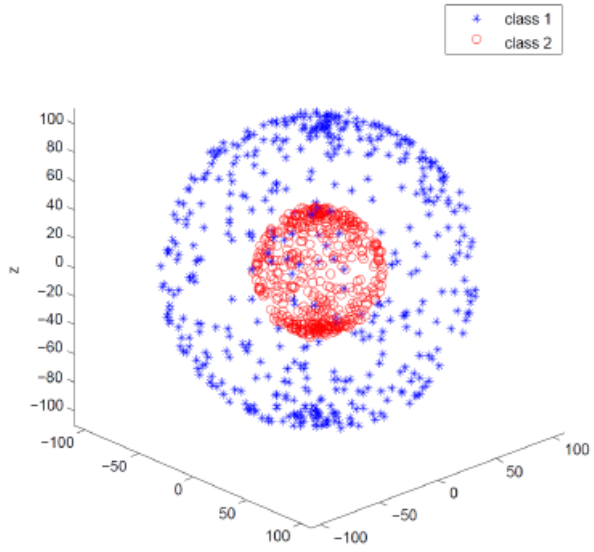
만약 projection된 dataset이 평균이 0이 아니라면 다음과 같은 식을 통해 평균을 0으로 만들어 줄 수 있다.

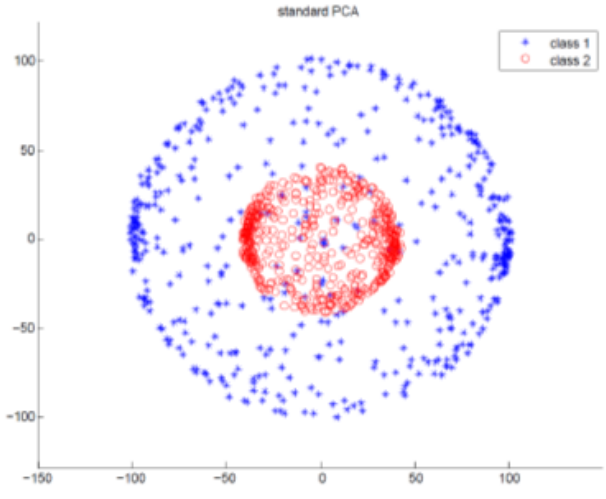
$$\begin{aligned} \hat{\mathrm{K}} &= (\mathrm{I} - 1_N)\mathrm{K}(\mathrm{I} - 1_N) \\ &= \mathrm{K} - 1_N\mathrm{K} - \mathrm{K}1_N + 1_N\mathrm{K}1_N \end{aligned}$$

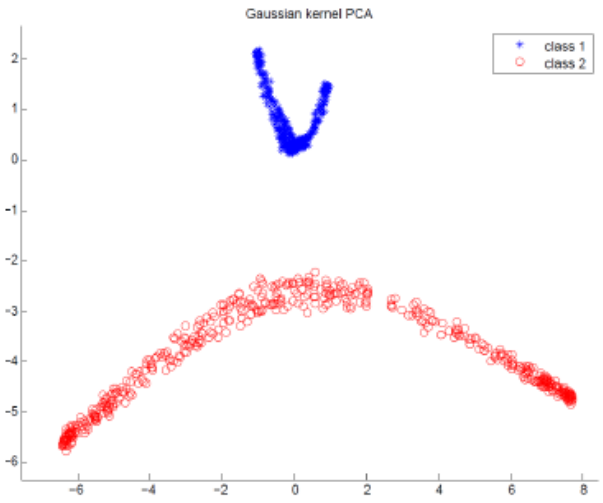
여기서 1_N은 모든 원소가 $\frac{1}{N}$인 matrix이다.

3.3 예시

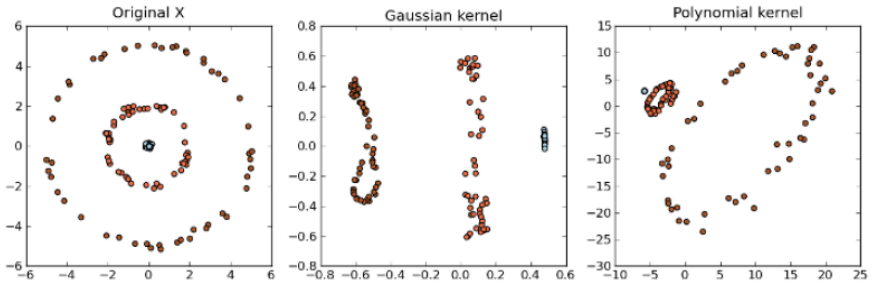
3.3.1 PCA와 KPCA비교



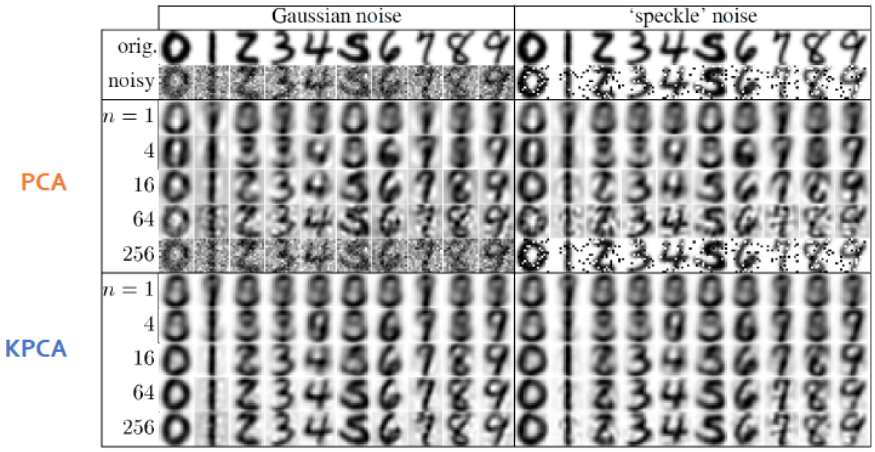




3.3.2 kernel에 따른 차이



3.3.3 Application



이미지의 De-noising과 관련해서도 KPCA가 더 좋은 결과물을 보여주고 있다.