THE UNIVERSITY *of* ADELAIDE

Kernel Method

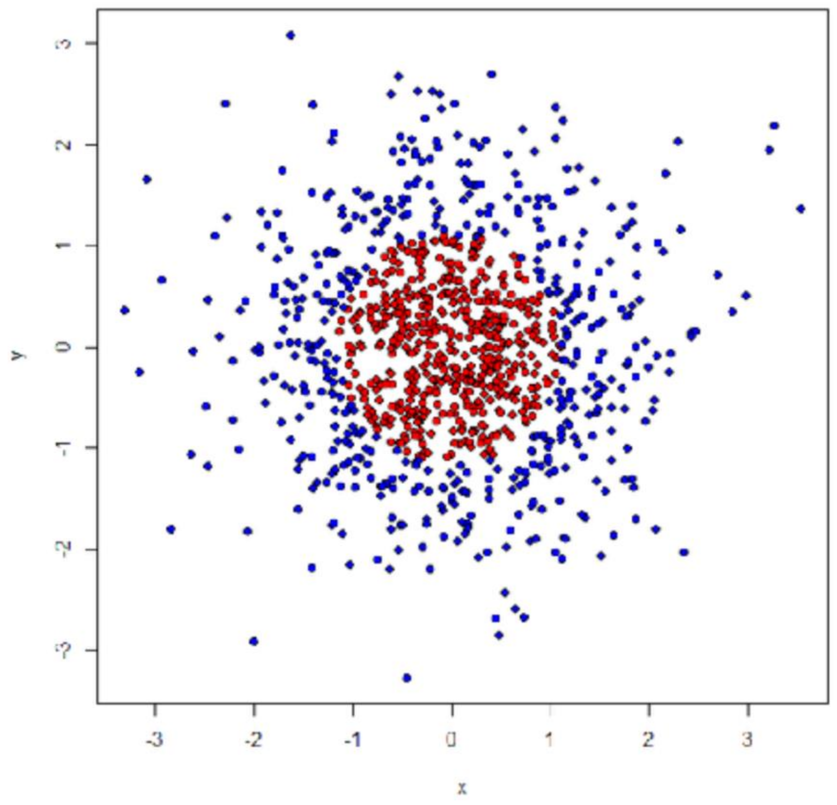
Lingqiao Liu
University of Adelaide

Outlines

- Why Kernel methods?
  - Motivation
  - Benefits
- Kernel Method
  - Criteria for kernel functions
  - Kernel SVM
  - Commonly used kernels
- Kernelizing machine learning algorithms
  - Simple practice
  - Kernel k-means
  - Kernel Regression
  - Kernel PCA

Motivation

- Limitation of Linear Classifiers

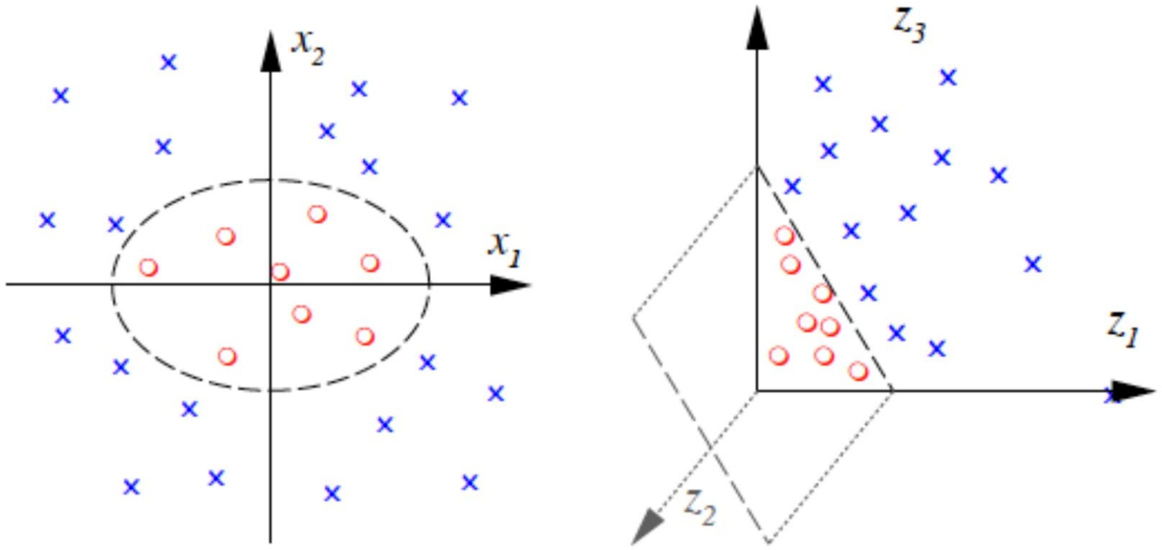


Feature transform

- Idea: transform the feature to higher dimensional space

$$\Phi : R^2 \rightarrow R^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt(2)x_1 x_2, x_2^2)$$



Demo

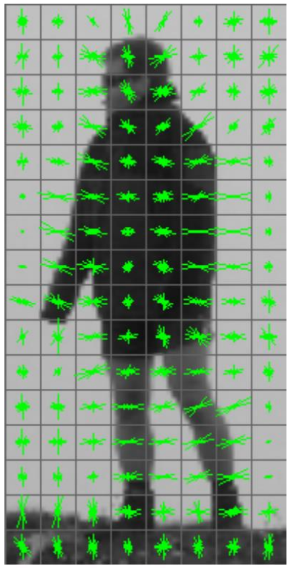
https://www.youtube.com/watch?v=3liCbRZPrZA

Feature transform

- Importance of the feature transform
  - High-dimensional feature space usually works
- The transform function
  - Can be a simple arithmetic operation
  - Can be anything!

Feature transform

e.g. Instead of using raw pixel, using histograms of oriented gradient



Issue

- The dimensionality of the mapped feature
  - can be high or even infinite
  - computational cost or infeasible
  - More convenient to define it implicitly

Kernel trick

- In many cases, we are only interested in the inner product of the mapped features

$$K(\mathbf{x}, \mathbf{x}') =< \varphi(\mathbf{x}), \varphi(\mathbf{x}') >$$

- In that case, we can define the feature map implicitly

- Kernel function
  - can be more concise than the feature map

Polynomial kernel

$O(n)$

- Definition

$$K(\mathbf{x}, \mathbf{x}') = (c + \boxed{\mathbf{x}^T\mathbf{x}'})^d$$

- Can be expanded as (when d = 2):

Recall that $< \mathbf{a}, \mathbf{b} >= \sum_i a_i b_i$

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + c)^2$$
$$= \sum_{i=1}^{n}(x_i^2)(x_i'^2) + \boxed{\sum_{i=2}^{n}\sum_{j=1}^{i-1}}(\sqrt{2}x_i x_j)(\sqrt{2}x'_i, x'_j)$$
$$+ \sum_{i=1}^{n}(\sqrt{2}cx_i)(\sqrt{2}cx'_i) + \boxed{c^2}$$

n

Polynomial kernel

$O(n^2)$

Equivalent to the following feature transform $\frac{n(n-1)}{2}$ 1

$$\varphi(\mathbf{x}) = \Big\langle \overbrace{x_n^2, \cdots, x_1^2}^{n}, \overbrace{\sqrt{2}x_n x_{n-1}, \sqrt{2}x_{n-1}x_{n-2}, \cdots, \sqrt{2}x_{n-1}x_1, \cdots, \sqrt{2}x_2 x_1}, \sqrt{2c}x_n, \cdots, \sqrt{2c}x_1, c \Big\rangle$$

$$\underline{K(\mathbf{x}, \mathbf{x}')} = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle$$

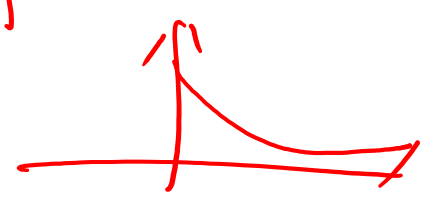
Calculating inner product

1. using kernel function
2. using feature transform

Which one is more efficient?

Gaussian RBF Kernel

exp(− z)



- Definition

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\sigma^2}\right)$$

Can be expand as

$$= \sum_{j=0}^{\infty} \sum_{\sum n_i = j} \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}\|^2\right) \frac{x_1^{n_1} \cdots x_k^{n_k}}{\sqrt{n_1! \cdots n_k!}} \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}'\|^2\right) \frac{x'^{n_1}_1 \cdots x'^{n_k}_k}{\sqrt{n_1! \cdots n_k!}}$$

Understand Kernel

- Intuitively, modelling the similarity between two feature points
  - Guideline for designing kernels
  - Not all similarity measurement can be a kernel function

Outlines

- Why Kernel methods?
  - Motivation
  - Benefits
- Kernel Method
  - Criteria for kernel functions
  - Kernel SVM
  - Commonly used kernels
- Kernelizing machine learning algorithms
  - Simple practice
  - Kernel k-means
  - Kernel Regression
  - Kernel PCA

Criterion for a kernel function

- By definition, check if a kernel function can be written in this form

$$K(\mathbf{x}, \mathbf{x}') = \underline{< \varphi(\mathbf{x}), \varphi(\mathbf{x}') >}$$

- Equivalently, with arbitrary m samples, check if the kernel matrix

$$K = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \cdots & k(x_m, x_m) \end{bmatrix}$$

can be decomposed into

$$\boxed{\overset{m\times m}{\mathbf{K}} = \overset{m\times d}{\phi(\mathbf{X})^T}\, \overset{d\times m}{\phi(\mathbf{X})}}$$

Criterion for a kernel function

- Implies Semi-positive-definite of K

$K = \phi^T(X)\phi(X)$

$1\times m \quad m\times d \quad d\times m \quad m\times 1$

- Definition

$\boxed{a^T\phi^T(X)}\boxed{\phi(X)a}$

$$\mathbf{a}^T\mathbf{K}\mathbf{a} \geq 0$$

$$= \sum_i \sum_j K(\mathbf{x}_i, \mathbf{x}_j) a_i a_j \geq 0$$

$= (\phi(x)a)^T(\phi(x)a)$

$= \|\phi(x)a\|_2^2 \geq 0$

- We can extend this to infinite number of samples (functional space)

Criterion for a kernel function

- Mercer condition

A real-valued function $K(x, y)$ is said to fulfill Mercer's condition if for all square integrable function $g(x)$ one has

$$\int\int g(x)K(x,y)g(y)dxdy \geq 0$$

$$\int_{-\infty}^{\infty} |g(x)|^2 dx < \infty$$

Summary so far

- Kernel function
  - Sometimes we are interested in the inner product of transformed features
  - We define a function to calculate the inner product between a pair of features, called the kernel function
  - Define a kernel function is equivalent to defining a feature transform
- Application?

Kernel SVM

- Linear SVM (Primal form)

$$\min_{\mathbf{w},b,\{\xi_i\}} \|\mathbf{w}\|_2^2 + \lambda \sum_i \xi_i$$

$$\begin{aligned} s.t. \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \forall i \end{aligned}$$

Kernel SVM

- Dual form

$$\max_{\{\alpha_i\}} \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j y_i \alpha_i y_j \alpha_j \boxed{\underline{\langle \mathbf{x}_i, \mathbf{x}_j \rangle}}$$

$$s.t. \quad \sum_i \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq \lambda \quad \forall i$$

Kernel SVM

SVM dual

$$\max_{\{\alpha_i\}} \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j y_i \alpha_i y_j \alpha_j < \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) >$$

$$s.t. \quad \sum_i \alpha_i y_i = 0$$

$$0 \le \alpha_i \le \lambda \quad \forall i$$

Kernel SVM

SVM dual

$$\max_{\{\alpha_i\}} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j y_i \alpha_i y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$$

$$s.t. \quad \sum_i \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq \lambda \quad \forall i$$

Define $k(\mathbf{x}_i, \mathbf{x}_j) = < \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) >$

Kernel SVM

- Decision function for kernel SVM

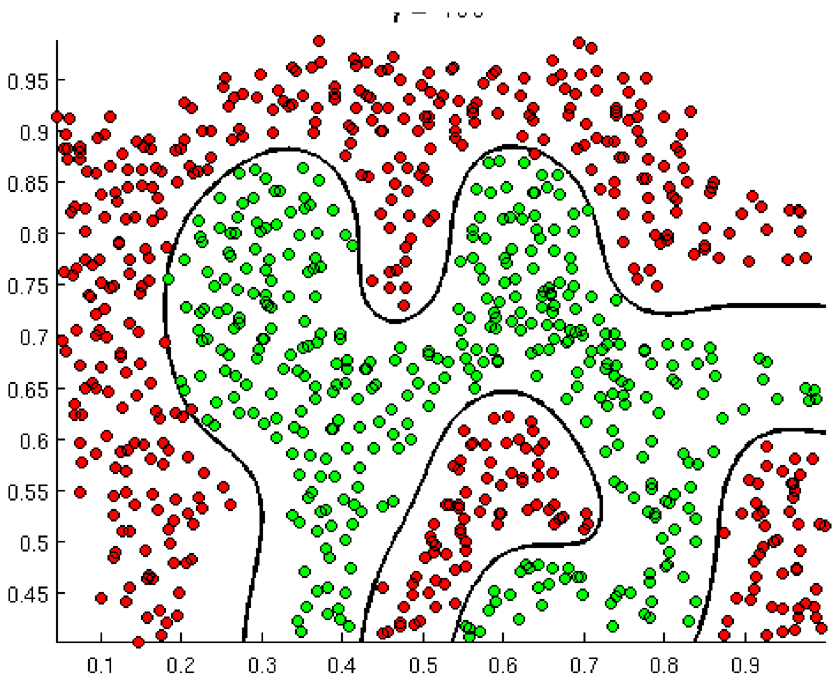
$$w = \sum_i \alpha_i y_i \varphi(x_i)$$

$$\begin{aligned}
f(\mathbf{x}_t) &= \mathbf{w}^\top \varphi(\mathbf{x}_t) + b \\
&= \boxed{\Big(\sum_{i=1}^{N_{train}} \alpha_i y_i \varphi(\mathbf{x}_i)\Big)}^\top \underline{\varphi(\mathbf{x}_t)} + b \\
&= \left(\sum_{i=1}^{N_{train}} \alpha_i y_i < \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_t) > \right) + b \\
&= \left(\sum_{i=1}^{N_{train}} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}_t)\right) + b
\end{aligned}$$

No need to access to the exact value of the transformed feature, but just their inner product

Kernel SVM

- Decision boundary of a Kernel SVM



Lesson learned

- Kernel SVM produce more flexible decision boundary and thus can lead to better classification performance
- Choosing a good kernel is the key to success

Commonly used kernels

- Linear Kernel

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

- Polynomial kernel

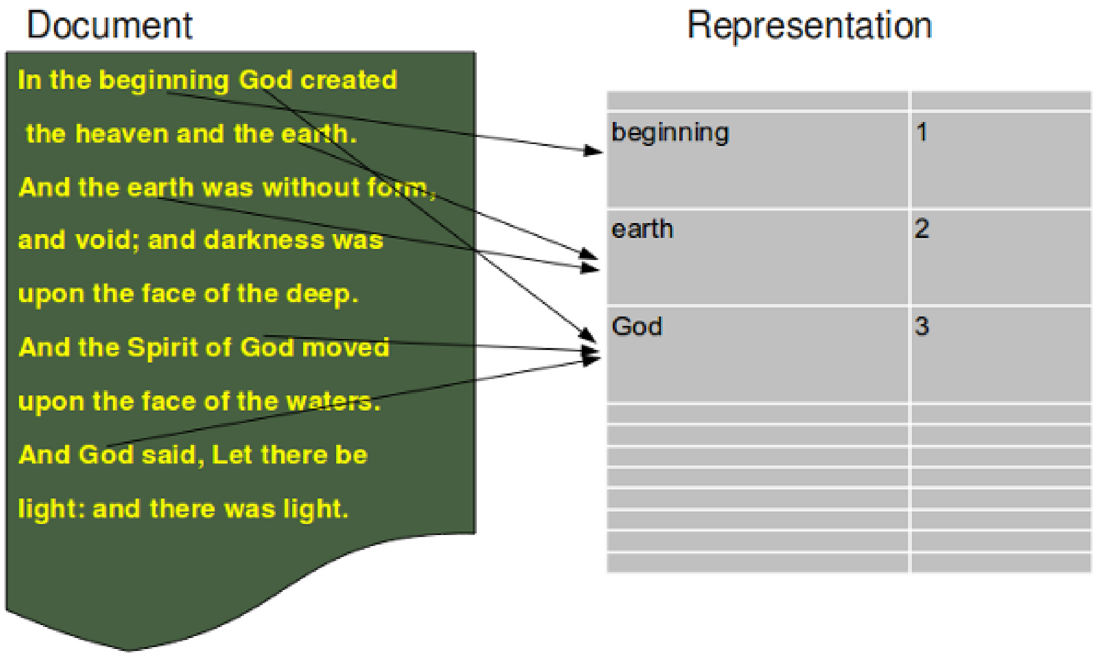
$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d$$

- Gaussian RBF kernel

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\sigma^2}\right)$$

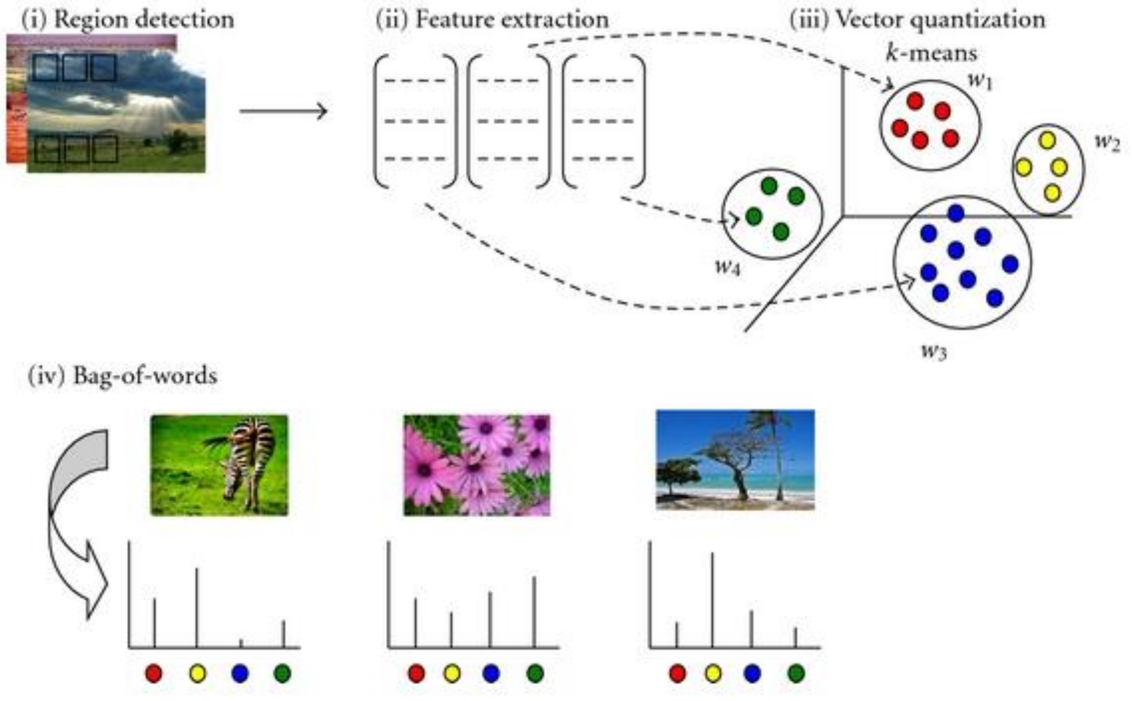
Commonly used kernels

- Histogram alike features: Bag-of-words


Document
In the beginning God created
the heaven and the earth.
And the earth was without form,
and void; and darkness was
upon the face of the deep.
And the Spirit of God moved
upon the face of the waters.
And God said, Let there be
light: and there was light.
Representation
beginning
1
earth
2
God
3

Commonly used kernels

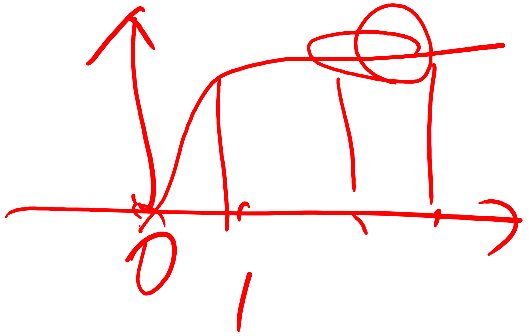
- Histogram alike feature: bag-of-features model



Commonly used kernels for histogram liked features

- Hellinger kernel

$$K(\mathbf{x}, \mathbf{x}') = \sum_i \sqrt{x_i x'_i}$$



- Histogram Intersection Kernel

$$K(\mathbf{x}, \mathbf{x}') = \sum_i \min\{x_i, x'_i\}$$

- RBF Kernel

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \sum_i \frac{(x_i - x'_i)^2}{x_i + x'_i}\right)$$

Lesson learned (from kernel SVM)

- Kernel SVM produce more flexible decision boundary and thus can lead to better classification performance
- Choosing a good kernel is the key to success
- We need to reformulate the original form to make the problem only depend on the inner product of transformed features

SVM dual

$$\max_{\{\alpha_i\}} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j y_i \alpha_i y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$$

$$s.t. \quad \sum_i \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq \lambda \quad \forall i$$

Outlines

- Why Kernel methods?
  - Motivation
  - Benefits
- Kernel Method
  - Criteria for kernel functions
  - Kernel SVM
  - Commonly used kernels
- Kernelizing machine learning algorithms
  - Simple practice
  - Kernel k-means
  - Kernel Regression
  - Kernel PCA

Kernelize Machine Learning Algorithm

- How to kernelize a ML algorithm
  - Assume we already know $\varphi(\mathbf{x})$
  - Reformulate the calculation to ensure that the algorithm only depend on

$$K(\mathbf{x}, \mathbf{x}') = < \varphi(\mathbf{x}), \varphi(\mathbf{x}') >$$

- Revisit ML algorithms
  - Kernel K-means
  - Kernel regression
  - Kernel PCA

Warm-up Practice

- Euclidean distance

$$\underline{\|\varphi(\mathbf{x}) - \varphi(\mathbf{x}')\|_2^2}$$

$$\underline{K(x, x')}$$

$$= (c + x^T x')^2$$

$$= \langle \varphi(x), \varphi(x')$$

Warm-up Practice

- Euclidean distance

$$= \underline{(\varphi(x)-\varphi(x'))^T(\varphi(x)-\varphi(x'))}$$

$$\|\varphi(\mathbf{x})-\varphi(\mathbf{x}')\|_2^2$$

$$\begin{aligned}&= \underline{\langle\varphi(\mathbf{x}),\varphi(\mathbf{x})\rangle+\langle\varphi(\mathbf{x}'),\varphi(\mathbf{x}')\rangle-2\langle\varphi(\mathbf{x}),\varphi(\mathbf{x}')\rangle}\\ &= K(\mathbf{x},\mathbf{x})+K(\mathbf{x}',\mathbf{x}')-2K(\mathbf{x},\mathbf{x}')\end{aligned}$$

K-means algorithm

- E-step: Fix μ_k, minimize J w.r.t. r_{ik}.
  - Assign each data point to its nearest prototype.
- M-step: Fix r_{ik}, minimize J w.r.t. μ_k.
  - Set each prototype to the mean of the points in that cluster, i.e., $\mu_k = \frac{\sum_i r_{ik} x_i}{\sum_i r_{ik}}$.
- This procedure is guaranteed to converge.
- Converges to a local minimum.

How to kernelize k-means

- Key step: Assigning to the nearest centre - calculating distance between feature and centre

$$\text{assign}(\mathbf{x}_i) = argmin_k \, \|\varphi(\mathbf{x}_i) - \mu_k\|_2^2$$

- Challenges
  - We cannot explicitly access $\varphi(\mathbf{x}_i)$
  - Consequently, we cannot explicitly access μ_k

- How to represent centre?

$$\underline{\mu_k = \frac{\sum_i r_{ik}^{t-1} \varphi(\mathbf{x}_i)}{\sum_i r_{ik}^{t-1}}}$$

How to kernelize k-means

$$\text{assign}(\mathbf{x}_i) = argmin_k \, \|\varphi(\mathbf{x}_i) - \mu_k\|_2^2$$

$$\text{assign}(\mathbf{x}_i) = argmin_k \, \|\varphi(\mathbf{x}_i) - \frac{\sum_j r_{jk}^{t-1}\varphi(\mathbf{x}_j)}{\sum_j r_{jk}^{t-1}}\|_2^2$$

How to calculate $\|\varphi(\mathbf{x}_i) - \frac{\sum_j r_{jk}^{t-1}\varphi(\mathbf{x}_j)}{\sum_j r_{jk}^{t-1}}\|_2^2$?

Define $\boxed{r'_j = \frac{r_{jk}^{t-1}}{\sum_j r_{jk}^{t-1}}}$

$$\|\varphi(\mathbf{x}_i) - \frac{\sum_j r_{jk}^{t-1}\varphi(\mathbf{x}_j)}{\sum_j r_{jk}^{t-1}}\|_2^2 = \|\varphi(\mathbf{x}_i) - \sum_j r'_j \varphi(\mathbf{x}_j)\|_2^2$$

How to kernelize k-means

$$\|\varphi(\mathbf{x}_i) - \textstyle\sum_j r'_j \varphi(\mathbf{x}_j)\|_2^2$$

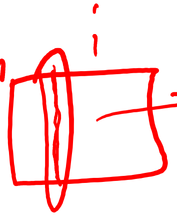
$$= < \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_i) > -2 < \varphi(\mathbf{x}_i), \sum_j r'_j \varphi(\mathbf{x}_j) > + < \sum_j r'_j \varphi(\mathbf{x}_j), \sum_j r'_j \varphi(\mathbf{x}_j) >$$

$$= < \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_i) > -2 \sum_j r'_j < \varphi(\mathbf{x}_j), \varphi(\mathbf{x}_i) > + \sum_i \sum_j r'_i r'_j < \varphi(\mathbf{x}_j), \varphi(\mathbf{x}_i) >$$

$$= k(\mathbf{x}_i, \mathbf{x}_i) - 2 \sum_j r'_j k(\mathbf{x}_j, \mathbf{x}_i) + \sum_i \sum_j r'_i r'_j k(\mathbf{x}_j, \mathbf{x}_i) \quad = r^T K r$$

Please note that $(\sum_i a_i)(\sum_i a_i) = (\sum_i a_i)(\sum_j a_j) = \sum_i \sum_j a_i a_j$

$K \in R^{m \times m}$



$\rightarrow k(x_j, x_i)$

$j: 1 \rightarrow n$

$2 r^T K(:,i)$

$K(i,:)$

$K \quad K$

Put them together: Kernel K-means

- We could combine E step and M step into a single step:
  - Calculating the assignment

$$\mathbf{r}_i^t = \text{assign}(\mathbf{x}_i) = argmin_k \left\|\varphi(\mathbf{x}_i) - \boxed{\frac{\sum_j r_{jk}^{t-1}\varphi(\mathbf{x}_j)}{\sum_j r_{jk}^{t-1}}}\right\|_2^2$$

By using the above kernelized algorithm

  - Iteratively execute above step until the assignment won't change between iterations

Kernel Regression

- Linear Regression Problem

$$\sum_i \|\mathbf{w}^\top \varphi(\mathbf{x}_i) - y_i\|_2^2$$

- However, directly expand this term cannot kernelize the expression

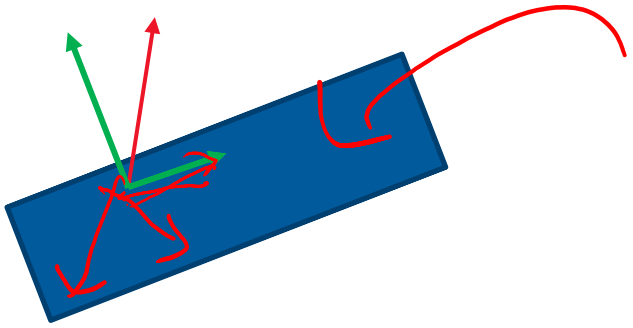
$$\overset{1\times d}{\overline{w^T}}\ \overset{d\times 1}{\varphi(x_i)}\ \overset{1\times d}{\varphi(x_i)}\ \overset{d\times 1}{\overline{w^T}} - 2\overline{w^T}\,\underline{\varphi(x_i) y_i}^{T}$$

Represent w by training data

- All the training data span a linear subspace

$$\Omega = \{\mathbf{v} | \mathbf{v} = \textstyle\sum_i^{N_{train}} \alpha_i \mathbf{x}_i \in \mathrm{R}^d \quad \forall \alpha_i\}$$

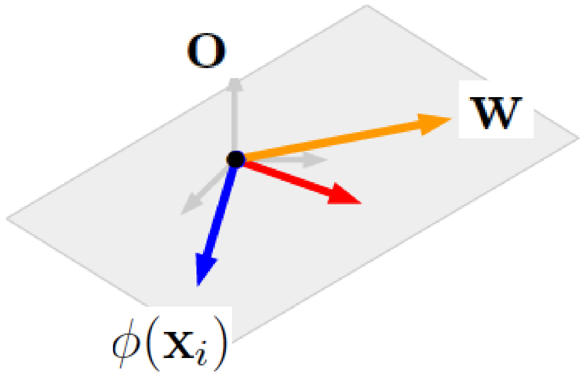
- You can imagine it as a plane in high-dimensional space
- We can project any d-dimensional vector as a component in the subspace and a component orthogonal to the subspace



Kernel Regression

- Rewrite w

$$\mathbf{w} = \boxed{\sum_i^{N_{train}} \alpha_i \mathbf{x}_i} + \boxed{\mathbf{o}}$$



Component on the subspace. Because it is on the subspace, we can find a combination weight $\{\alpha_i\}$ to reconstruct it from the training samples

Component orthogonal to the subspace. So $< \mathbf{x}_i, \mathbf{o} >= 0$

Kernel Regression

$$\sum_i \|\mathbf{w}^\top \varphi(\mathbf{x}_i) - y_i\|_2^2$$

$$= \sum_i \| \left(\sum_j \alpha_j \varphi(\mathbf{x}_j) + \mathbf{o} \right)^\top \varphi(\mathbf{x}_i) - y_i\|_2^2$$

$$= \sum_i \| \underline{\sum_j \alpha_j < \varphi(\mathbf{x}_j), \varphi(\mathbf{x}_i) >} + \underbrace{< \mathbf{o}, \varphi(\mathbf{x}_i) >}_{0} - y_i\|_2^2$$

$$= \sum_i \| \sum_j \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) - y_i\|_2^2$$

So instead of solving w, we solve $\{\alpha_i\}$ instead, then we use

$\sum_j \alpha_j k(\mathbf{x}_j, \mathbf{x}_i)$ to calculate the prediction

Lesson Learned

- Represent model parameters by linear combination of training features, learn the combination weight instead
- Representer theorem: https://en.wikipedia.org/wiki/Representer_theorem

PCA and kernel PCA

- Basic concept
  - The dimensionality reduction function is a mapping function that maps the original features to low-dimensional features
  - In general, this mapping function can be learned from training data and apply to other data
  - The parameters of the mapping function in PCA (and KPCA):
    - Mean vector
    - Project matrix P

Key steps in PCA

- Learning parameters:
  - Centralize data
  - Calculate covariance matrix
  - Calculate eigen vectors and eigen values
  - Sort eigen values and their corresponding eigen vectors, keep the top-k eigen vectors as the projection matrix P
- Apply transform
  - Centralize all data with the mean vector calculated from training data
  - Apply the learned projection matrix to the centralized data

Kernel PCA

- Main differences:
  - The inner product between two samples are measured by the kernel function
  - Each sample can be essentially represented by $\varphi(x)$
  - We do not know the actual representation of $\varphi(x)$
- Challenges
  - We cannot directly centralize data (for training and testing)
  - We cannot directly calculate the covariance matrix
  - We cannot directly calculate the projection matrix P

Kernel PCA

- Main differences:
  - The inner product between two samples are measured by the kernel function
  - Each sample can be essentially represented by $\varphi(x)$
  - We do not know the actual representation of $\varphi(x)$
- Challenges
  - We cannot directly centralize data (for training and testing)
  - We cannot directly calculate the covariance matrix
  - We cannot directly calculate the projection matrix P

Overcome Challenge 2

Notations:
$\bar{X} \in R^{d \times N}$: Centralized data matrix
d: dimensionality of features; N: number of samples

In PCA, we need to calculate the eigen-vectors of the covariance matrix, that is to solve

$$\bar{X}\bar{X}^T v = \lambda v,$$

where $\bar{X}\bar{X}^T \in R^{d \times d}$ and v is an eigen-vector of the covariance matrix.

Anther way of calculating v is to use the following relationship between the eigen-vector of the covariance matrix and kernel matrix.

Assume v' is the eigen-vector of the kernel matrix $\bar{X}^T\bar{X}$, by the definition of the eigen vector, v' should satisfy the following equation

$$\bar{X}^T\bar{X}v' = \lambda v'$$

Overcome Challenge 2

Assume v' is the eigen-vector of the kernel matrix X^TX, by the definition of the eigen vector, v' should satisfy the following equation

$$X^TXv' = \lambda v' \tag{1}$$

Multiply $\bar{X}$ on both sides of EQ(1), we will get

$$\bar{X}\bar{X}^T\bar{X}v' = \lambda\bar{X}v' \tag{2}$$

Now if we let $z = \bar{X}v'$, we will have

$$\underline{\bar{X}\bar{X}^Tz = \lambda z} \tag{3}$$

We can see that z satisfy the definition of the eigen vector of the covariance matrix $\bar{X}\bar{X}^T$. In other words, **the eigen vector of the covariance matrix can be calculated by first calculating the eigen vector of the kernel matrix, that is, v', and then multiply it by $\bar{X}$.**

Overcome Challenge 2

$1\nu \times 1\nu$

$1\nu \nu 1$

$d \times N \quad N \times 1$

In other words, $v = \boxed{\bar{X}v'}$. Recall that v is the eigen vector of the covariance matrix.

The above relationship can be used to calculate the eigen-vector of covariance matrix for mapped features $\varphi(\bar{x})$ in KPCA. In such a case, each element of the kernel matrix is $K'_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, that is,

$K = X^T X$

$$K_{ij} = [\varphi(\bar{X})^T \varphi(\bar{X})]_{i,j} = \underline{< \varphi(\bar{x}_i), \varphi(\bar{x}_j) >} = \underline{k'(\mathbf{x}_i, \mathbf{x}_j)}$$

N

$k(x_i, x_j)$

Note that, in the above equations, we assume transformed feature $\varphi(\mathbf{x})$ is not centred, not the original feature $\mathbf{x}$

Kernel PCA

- Main differences:
  - The inner product between two samples are measured by the kernel function
  - Each sample can be essentially represented by $\varphi(x)$
  - We do not know the actual representation of $\varphi(x)$
- Challenges
  - We cannot directly centralize data (for training and testing)
  - We cannot directly calculate the covariance matrix
  - We cannot directly calculate the projection matrix P

Overcome Challenge 1 (at training stage)

Note that in the above calculation, we only need to use the kernel matrix rather than the original data. So we only care about the kernel matrix of the centralized data, that is,

$$\begin{aligned}\bar{k}(x_i, x_j) &= (\varphi(x_i) - \mu)^T(\varphi(x_j) - \mu)\\ &= (\varphi(x_i) - \underline{\frac{1}{N}\sum_k} \varphi(x_k))^T(\varphi(x_j) - \underline{\frac{1}{N}\sum_k \varphi(x_k)})\\ &= \underline{k(x_i, x_j) - \frac{1}{N}\sum_k k(x_i, x_k) - \frac{1}{N}\sum_k k(x_k, x_j) + \frac{1}{N^2}\sum_{ij} k(x_i, x_j)}\end{aligned}$$

Written in a matrix form:

$$\underline{K' = K - 1_N K - K 1_N + \underline{1_N} K 1_N}$$

where 1_N denotes a N-by-N matrix for which each element takes value $1/N$

Kernel PCA

- Main differences:
  - The inner product between two samples are measured by the kernel function
  - Each sample can be essentially represented by $\varphi(x)$
  - We do not know the actual representation of $\varphi(x)$
- Challenges
  - We cannot directly centralize data (for training and testing)
  - We cannot directly calculate the covariance matrix
  - We cannot directly calculate the projection matrix P

Overcome Challenge 3 and Challenge 1 (at testing time)

- Recall that the projection matrix P in PCA obtained by stacking eigen vectors of the covariance matrix (sorting them through their corresponding eigen-values in the descending order)

For a new data sample, PCA applies the following transformation:

$$y = \underline{P}^T(\underline{x} - \mu).$$

where $y \in R^{m\times 1}$ $P \in R^{d\times m}$ and $\mu \in R^{d\times 1}$.

Each column of P is an eigen-vector of the covariance matrix. To reduce the dimensionality to m, PCA selects the top-m eigen-vectors

If we just look at one dimension of y, which corresponding to one column of P, that is,

$$y_k = \boxed{P_{\underline{k}}^T} \underline{(x - \boxed{\mu})}$$

Overcome Challenge 3 and Challenge 1 (at testing time)

Now let's move on to kernel PCA. In kernel PCA, by using the relationship

$$v = \varphi(\bar{X})v'$$

where $\phi(\bar{X})$ is the centralized training data and v is any eigen-vector of the covariance matrix. So it applies to the k-th eigen-vector v_k. Note that p_k is actually v_k. So

$$\begin{aligned}
y_k &= p_k^T(\varphi(x_t) - \mu) \\
&= v_k'^T \varphi(\bar{X})^T (\varphi(x_t) - \frac{1}{N}\sum_i \varphi(x_i)) \\
&= v_k'^T (\varphi(X) - \frac{1}{N}\sum_i \varphi(x_i))^T (\varphi(x_t) - \frac{1}{N}\sum_i \varphi(x_i))
\end{aligned}$$

Putting together: KPCA

- Training
  - Centralized kernel matrix (slide 51)
  - Calculating centeralized Kernel matrix and its eigen vector (slide 49)
  - Calculate the eigen vector, eigen value of the centralized kernel matrix (obtain v')
  - Sort eigen vectors by its corresponding eigen values (same as in PCA)
- Testing
  - Use the equation in slide 54 to calculate the dimension-reduced features .

Summary

- Kernel Method: theory
  - Motivation
  - Criterion for kernel functions.
  - Use of Kernels: Kernel SVM
  - Commonly used kernel

- Kernel Method: kernelize machine learning algorithm
  - Kernel k-means
  - Kernel Regression
  - Kernel PCA