Kernels

Virgil Pavlu November 30, 2014

1 Intro to similarity functions

Similarities and distances are often the critical aspect of machine learning : knowing these, one can make the prediction/classification problem easy. Trivial examples: kNN, Clustering, Collaborative Filtering

SVM-dual formulation: in the SVM dual form problem, all datapoints x appear only as part of a dot product $< x1 * x2 >$, never alone. Also the prediction function on a test point z, $F(z)$, can be computed as a formula of dot products between z and the support vectors in the training set $< x * z >$.

Non-separable data: SVMs and other kernel machines have two ways to deal with non-separable data:

- by model/representation which is by design, in advance : deciding on a kernel that is likely to make data [more] separable
- by dealing with the actual data, after the model/kernel is decided : use slack variables to allow for "points inside the margins" that cost a regularization penalty.

1.1 data similarities and dot product

- measurement of data similarities : a fundamental problem in ML
- reflects a priori knowledge of the problem/data
- dot product : a natural measure for similarity
  $\langle \mathbf{x} \cdot \mathbf{y} \rangle = \sum_i x_i \cdot y_i$
- dot product amounts to being able to carry all geometric constructions formulated in terms of angles, lengths and distances

$\cos(\mathbf{x}, \mathbf{y}) = \frac{\langle \mathbf{x} \cdot \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|}$ $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x} \cdot \mathbf{x} \rangle}$

1.2 Data similarity and kernels

feature space, kernels

- general measure for similarity
  $k : X \times X \to \mathbf{R}$, symetric $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$

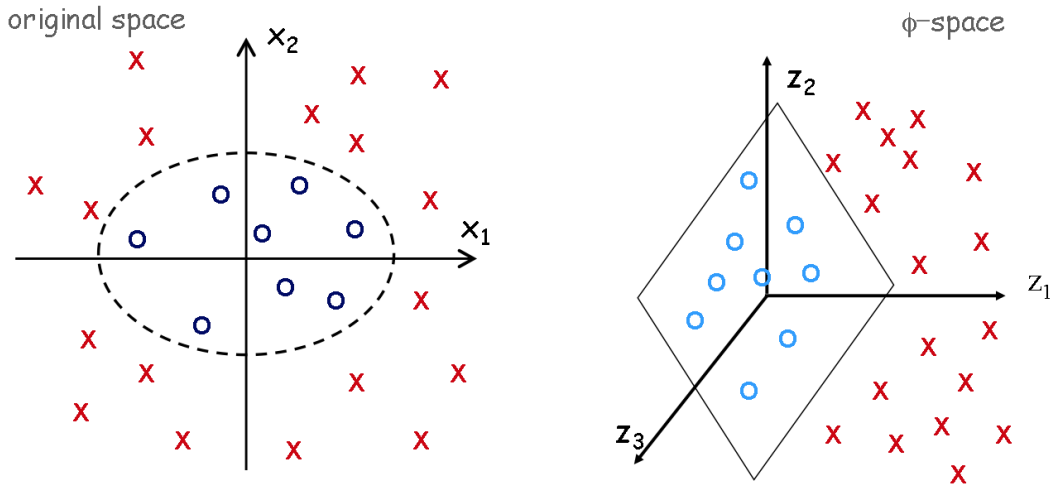
- symmetry is too general, we want something that feels like dot product
  $\exists \Phi : X \to \mathbf{H}$ mapping function
  $k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$
  where **H**=feature space (Hilbert space, supports dot product)

H= feature space, Φ= map(feature) function

- for which k there exits Φ ?
- given k, if Φ exists, it may be not unique

2 Kernels examples, polynomial: how non-separable data becomes separable through a mapping into a high-dimension space

polynomial kernel : example 1



use the map $\mathbf{x}=(x_1, x_2) \to \Phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
ellipse from 2D-input space becomes hyperplane into 3D-feature space

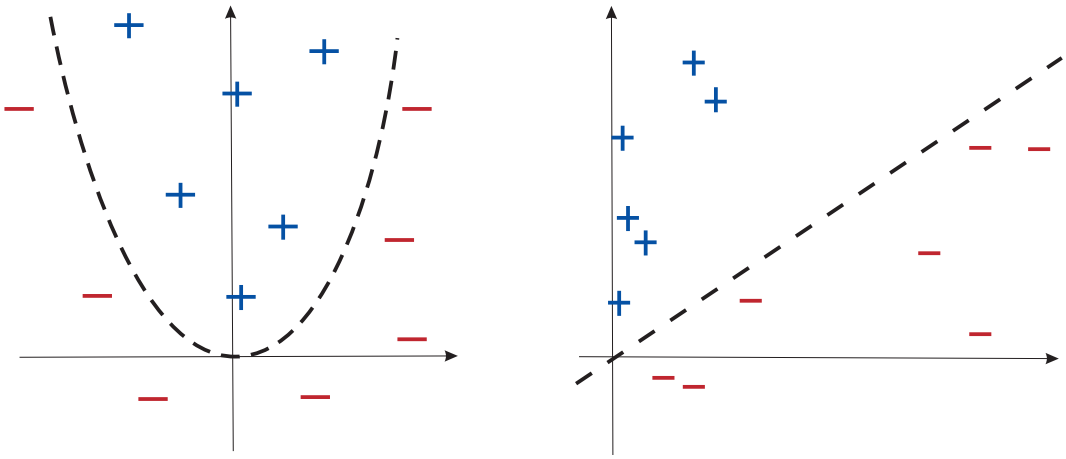
Note that a different mapping $\Phi_2(\mathbf{x}) = (x_1^2, x_2^2, x_1x_2, x_2x_1)$ maps data in a 4D-feature space, but it generates the same kernel k
$k(\mathbf{x}, \mathbf{y}) = \langle \Phi_2(\mathbf{x}) * \Phi_2(\mathbf{y}) \rangle = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2$

The kernel trick: We dont need Φ to calculate k, as we can obtain k directly from the dot product $< x * y >= x_1y_1 + x_2y_2$ by applying a polynomial (hence the name "polynomial kernel"):

$$k(x, y) = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2 = (x_1y_1 + x_2y_2)^2 =< x * y >^2$$

polynomial kernel : example 2

input space : $\mathbf{x} = (x_1, x_2)$ (2 attributes)
feature space : $\Phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1)$ (6 attributes)



$k(\mathbf{x}, \mathbf{y}) = \langle \Phi_2(\mathbf{x}) * \Phi_2(\mathbf{y}) \rangle = x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 y_1 x_2 y_2 + 1 =$
$= (x_1 y_1 + x_2 y_2 + 1)^2 = (< x * y > +1)^2$

More broadly, the kernel $K(x, y) = (x^T y + c)^d$ corresponds to a feature mapping to an (n+d choose d) feature space, corresponding of all monomials of the d form $x_i^1 x_i^2 ... x_i^l$ that are up to order d. However, despite working in this $O(n^d)$-dimensional space, computing $K(x, y)$ still takes only $O(n)$ time, and hence we never need to explicitly represent feature vectors in this very high dimensional feature space.

3 Use of kernels with SVM

plug a kernel into SVM

the primal problem
minimize $\langle \mathbf{w} \cdot \mathbf{w} \rangle$
subject to $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \forall i$

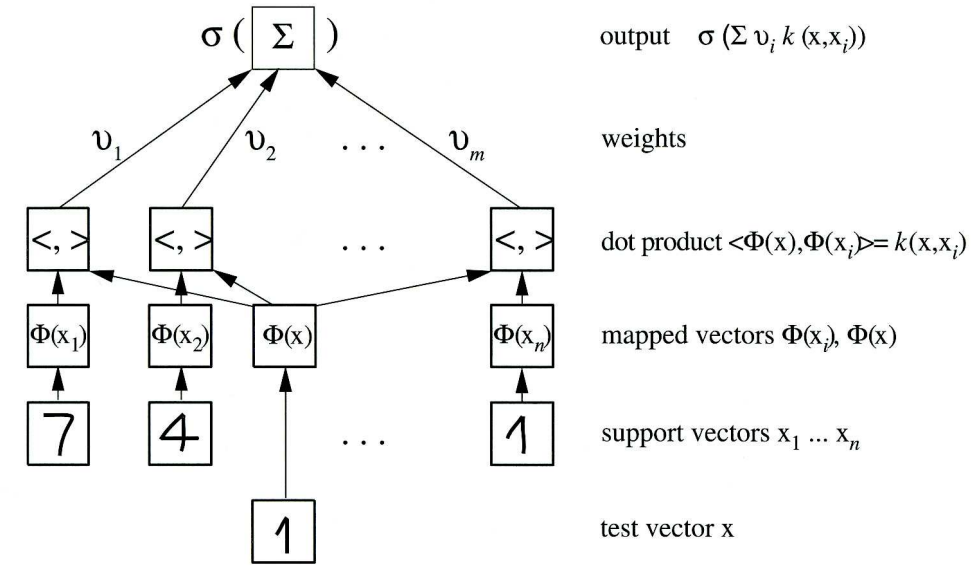
the dual problem
maximize $P(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$
subject to $\sum_{i=1}^{m} y_i \alpha_i = 0, \alpha_i \geq 0, \forall i$

kernel trick replace the dot product $\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$ with a kernel $k(\mathbf{x}_i, \mathbf{x}_j)$:
maximize

$$P(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to $\sum_{i=1}^{m} y_i \alpha_i = 0, \alpha_i \geq 0, \forall i$

SVM with kernels



the kernel trick

maximize

$$P(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to $\sum_{i=1}^{m} y_i \alpha_i = 0, \alpha_i \geq 0, \forall i$

- we need only the kernel k ,**not** Φ - thats good...
- any algorithm that only depends on dot products (rotationally invariant) can be kernelized
- any algorithm that is formulated in terms of positive definite kernel(s) supports a kernel-replace

• math was around for long time (1940s Kolgomorov, Aronszajn, Schoenberg) but the practical importance was underestimated

3.1 Kernel trick

We notice that there are many dot products $(x_i^T x_j)$ in our formula. We can keep the whole SVM-DUAL setup, and the algorithms for solving these problems, but choose **kernel function** $k(x_i^T, x_j)$ to replace the dot products $(x_i^T x_j)$. To qualify as a kernel, informally, the function $k(x_i, x_j)$ must be a dot product $k(x_i, x_j) = \Phi(x_i) * \Phi(x_j)$, where $\Phi(x)$ is a mapping vector from the original feature space $\{X\}$ into a different feature space $\{\Phi(X)\}$.

The essential "trick" is that usually Φ is not needed or known, only $k(x_i, x_j)$ is computable and used. To see this for the SVM, it is clear that the dual problem is an optimization written in terms of the dot products replaceable with a given kernel $k(x_i, x_j)$.

How about testing? The parameter $w = \sum_{i=1}^m \alpha_i y_i \Phi(x_i)$ is not directly computable if we dont know explicitly the mapping $\Phi()$, but it turns out we dot need to compute w explicitly; we only need to compute predictions for test points z:
$w\Phi(z) + b = \sum_{i=1}^m \alpha_i y_i \Phi(x_i)\Phi(z) + b = \sum_{i=1}^m \alpha_i y_i k(x_i, z) + b$

This fact has profound implications to the ability of representing data and learning from data: we can apply SVM to separate data which is not linearly separable! Thats because even if the data is not separable in the original space $\{X\}$, it might be separable in the mapped space $\{\Phi(X)\}$. The kernel trick is not specific to SVMs; it works with all algorithms that can be written in terms of dot products $x_i * x_j$.

4 valid kernels

- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_1(\mathbf{x}, \mathbf{z}) + \mathcal{K}_2(\mathbf{x}, \mathbf{z})$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = a\mathcal{K}_1(\mathbf{x}, \mathbf{z})$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_1(\mathbf{x}, \mathbf{z})\mathcal{K}_2(\mathbf{x}, \mathbf{z})$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$

$p(x)$ a polynomial with positive coefficients applied to an existing kernel
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = p(\mathcal{K}_1(\mathbf{x}, \mathbf{z}))$

$p(x)$ exponential applied to an existing kernel
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = exp(\mathcal{K}_1(\mathbf{x}, \mathbf{z}))$

The RBF/Gaussian kernel
- $\mathcal{K}(\mathbf{x}, \mathbf{x}') = exp(-||\mathbf{x} - \mathbf{x}'||^2/\sigma^2)$

5 Kernels Theory [optional material]

Characterization theorem, informal: A symmetric matrix K is a kernel if and only if K is positive semi-definite or $\alpha^T K\alpha \geq 0$ for any vector α.

Kernel characterization theorem if the Gram matrix $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive definite then k is a dot product : $\exists\Phi$ such that $k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$

proof K positive definite $\Rightarrow K = SDS^T$ (diagonalization)
where S is orthogonal and D is diagonal with non-negative entries

then $k(\mathbf{x}_i, \mathbf{x}_j) = (SDS^T)_{ij} = \langle S_i \cdot DS_j \rangle = \langle \sqrt{D}S_i \cdot \sqrt{D}S_j \rangle$
take $\Phi(\mathbf{x}_i) = \sqrt{D}S_i$

Kernel characterization theorem (converse) if the kernel k is a dot product $\exists \Phi$, $k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$ then the Gram matrix $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive definite

proof for any $\alpha \in \mathbf{R}^m$

$$\alpha^T K \alpha = \sum_{i,j=1}^{m} \alpha_i \alpha_j K_{ij} = \langle \sum_{i=1}^{m} \alpha_i \Phi(\mathbf{x}_i), \sum_{j=1}^{m} \alpha_j \Phi(\mathbf{x}_j) \rangle = \| \sum_{i=1}^{m} \alpha_i \Phi(\mathbf{x}_i) \|^2 \geq 0$$

so K is positive definite

5.1 Mercer theorem

theorem[Mercer] Let $\mathcal{X}$ be a compact subset of $\mathbf{R}^n$. Suppose $\mathcal{K}$ is a continuous symmetric function such that

$$\int_{\mathcal{X}} \int_{\mathcal{X}} \mathcal{K}(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0$$

for all $f \in \mathcal{L}_2(\mathcal{X})$. Then, $\mathcal{K}(\mathbf{x}, \mathbf{z})$ can be expanded in a uniformly convergent series

$$\mathcal{K}(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{z})$$

in terms of the eigenfunctions $\phi_j \in \mathcal{L}_2(\mathcal{X})$ of $(\mathcal{T}_{\mathcal{K}} f)(\cdot) = \int_{\mathcal{X}} \mathcal{K}(\cdot, \mathbf{x}) f(\mathbf{x}) d\mathbf{x}$ normalized so that $||\phi_j||_{\mathcal{L}_2} = 1$ and positive associated eigenvalues $\lambda_j \geq 0$.

5.2 Representer Theorem

We have seen that the dual perceptron and the support vector machine (SVM) have identical forms for the final weight vector i.e., $w^* = \sum_{i=1}^{N} \alpha_i y_i x_i$. We have also seen that both these algorithms can work with kernels, that allows us to work efficiently in high dimensional spaces enabling us to learn complex non-linear decision boundaries and use these learning methods to work with other types of data such as strings, trees, etc. But, is the use of kernels limited to only these algorithms or is it possible to kernelize other learning methods as well? The answer to this is provided by the Representer theorem, which "roughly" states that:

If a learning algorithm can posed as a minimization problem of the form:

$$\min_{w} \quad Loss(y, f(w, x)) + \lambda Penalty(w) \tag{1}$$

where, w are the model parameters, $f(w, x)$ represents the classifier output, y is the actual label and λ is a regularization parameter. Then under some "weak" conditions on the loss and penalty functions, the solution has the form $w^* = \sum_i \alpha_i y_i x_i$, i.e., a linear combination of the training instances.

This is a very powerful result that allows us to apply the kernel trick to a broader range of learning algorithms. Representer theorem can be applied to SVMs, ridge regression and Logistic regression, among other methods. The Theorem shows the dramatic effect of regularizing a problem by including a dependency in $Penalty(w)||w||^2$ or in the function to optimize. This penalization makes sense because it forces the solution to be smooth, which is usually a powerful protection against overfitting of the data. The representer theorem shows that this penalization also has substantial computational advantages: any solution to the optimization problem is known to belong to a subspace of a dimension at most n, the number of points in S, even though the optimization is carried out over a possibly infinite-dimensional space. A practical

consequence is that the optimization can now be reformulated as an n-dimensional optimization problem, by substituting w into the objective and optimizing over $(_1, , _n) \in R^n$. Most kernel methods can be seen in light of the representer theorem: indeed one can often explicitly write the functional that is minimized, which involves a norm as regularization penalty. This observation can serve as a guide to choosing a kernel for practical applications, if one has some prior knowledge about the function the algorithm should output: it is in fact possible to design a kernel such that a priori desirable functions have a small norm.

6 dot product kernels

$k(\mathbf{x}, \mathbf{y}) = k(\langle \mathbf{x}, \mathbf{y} \rangle)$

theorem
•the function k of the dot product kernel must satisfy
$k(t) \geq 0, k'(t) \geq 0$ and $k'(t) + tk"(t) \geq 0 \; \forall t \geq 0$
in order to be a positive definite kernel. That may still be insufficient

• if k s a power series expansion

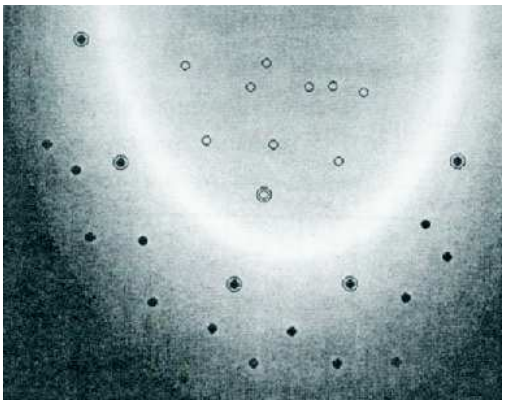
$$k(t) = \sum_{n=0}^{\infty} a_n t^n$$

then k is a positive definite kernel iff $\forall n, a_n \geq 0$

7 Kernel construction. Popular kernels

polynomial kernel

theorem define the map $\mathbf{x} \to C_d(\mathbf{x})$ where $C_d(\mathbf{x})$ the vector consisting in all possible d^{th} degree ordered products of the entries of $\mathbf{x}$=$(x_1, x_2, ..., x_N)$ then $\langle C_d(\mathbf{x}), C_d(\mathbf{y})\rangle = \langle \mathbf{x}, \mathbf{y}\rangle^d$



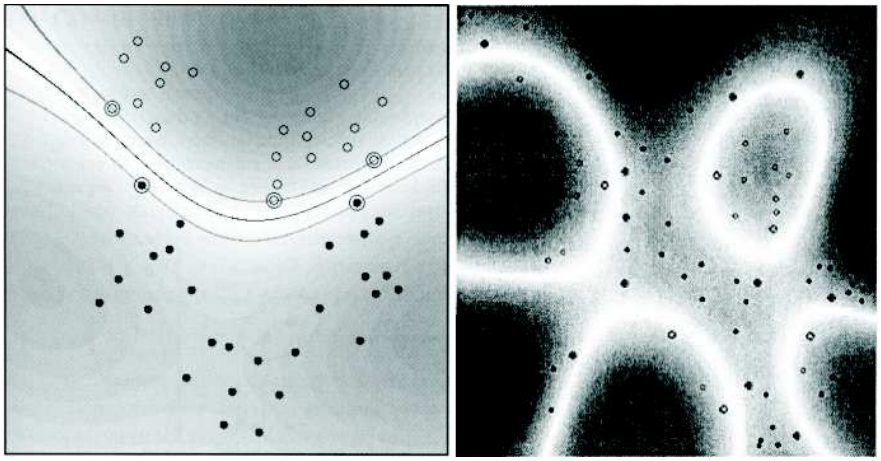
• polynomial kernel :
$k(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y}\rangle + c)^d$

• invariant to group of all orthogonal transformations(rotations, mirroring)

Gaussian ($\mathrm{R}_{\text{adial}}\,\mathrm{B}_{\text{asis}}\,\mathrm{F}_{\text{unction}}$) kernel

$k(\mathbf{x}, \mathbf{y}) = \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})$

more general $k(\mathbf{x}, \mathbf{y}) = f(d(\mathbf{x}, \mathbf{y}))$
where d is a metric on X and f is a function on $\mathbf{R}_0^+$; usually d arises from dot product $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$



• invariant on translations $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x} + \mathbf{z}, \mathbf{y} + \mathbf{z})$
• $\cos(\angle(\Phi(\mathbf{x}), \Phi(\mathbf{y}))) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y})\rangle = k(\mathbf{x}, \mathbf{y}) \geq 0 \Rightarrow$ enclosed angle between any 2 mapped points is smaller than $\pi/2$

The RBF kernel can still be written as a dot product in a new feature space $k(x, x') = \Phi(x) * \Phi(x')$, only with an infinite number of dimensions. To see this for $\sigma = 1$, consider the expansion

$$\exp\left(-\frac{1}{2}||\mathbf{x}-\mathbf{x}'||^2\right) = \sum_{j=0}^{\infty} \frac{(\mathbf{x}^\top \mathbf{x}')^j}{j!} \exp\left(-\frac{1}{2}||\mathbf{x}||^2\right) \exp\left(-\frac{1}{2}||\mathbf{x}'||^2\right)$$

theorem if $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$ all distinct and $\sigma > 0$ then the matrix $K_{ij} = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$ has full rank $\Rightarrow \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2), ..., \Phi(\mathbf{x}_m)$ are linear independent.

Fisher kernel [optional]

• knowledge about objects in form of a generative probability model
• deals with mising/incomplete data, uncertainty, variable length

family of generative models (density functions)
$p(x|\theta)$, smoothly parametrized by $\theta = (\theta^1, ..., \theta^r)$; $l(x, \theta) = \ln p(x|\theta)$

score $V_\theta(x) := (\delta_{\theta^1} l(x, \theta), ..., \delta_{\theta^r} l(x, \theta)) = \nabla_\theta l(x, \theta) = \nabla_\theta \ln p(x|\theta)$
Fisher information matrix $I := \mathbf{E}_p[V_\theta(x) V_\theta(x)^T]$
$I_{ij} = \mathbf{E}_p[\delta_{\theta^i} \ln p(x|\theta) \cdot \delta_{\theta^j} \ln p(x|\theta)]$, $\mathbf{E}_p$ is called *Fisher information metric*

Fisher kernel
$K_I(x, y) := V_\theta(x)^T I^{-1} V_\theta(y)$
natural kernel M positive definite matrix
$K_M^{nat}(x, y) := V_\theta(x)^T M^{-1} V_\theta(y)$

[information] diffusion kernel [optional]

- local relationships

the exponential of a suared matrix H is
$e^{\beta H} = \lim_{n\to\infty} (1 + \frac{\beta H}{n})^n = I + \beta H + \frac{\beta^2}{2!} H^2 + \frac{\beta^3}{3!} H^3 + ...$

exponential kernel $K_\beta = e^{\beta H}$, $\frac{\delta K_\beta}{\delta \beta} = H K_\beta$(heat eq)

diffusion kernel on graph : consider
$H_{ij} = 1$ if i j ; $-d_i$ (degree) if $i = j$; 0 otherwise
$w^T H w = -\sum_{i,j \in E} (w_i - w_j)^2$ negative semidefinite
$-H$=Laplacian of the graph

convolution kernel [optional]

kernel between composite objects building on similarities of resp. parts
$k_d : X_d \times X_d \to \mathbf{R}$, R-relation. define the R-convolution kernel

$$(k_1 \star k_2 \star ... \star k_D)(\mathbf{x}, \mathbf{y}) := \sum_R \prod_{d=1}^{D} k_d(x_d, y_d)$$

where the sum runs over all possible decompositions of $\mathbf{x} \to (x_1, x_2, ..., x_D)$ and of $\mathbf{y} \to (y_1, y_2, ..., y_D)$ s.t. $R(\mathbf{x}, x_1, x_2, ..., x_D)$ and $R(\mathbf{y}, y_1, y_2, ..., y_D)$
• proved valid if R is finite

ANOVA kernel [optional](analysis of variance)

if $X = S^N$ and $k^{(i)}$ kernel on $S \times S$ for $i = 1, 2, ..., N$,the ANOVA kernel of order D is

$$k_D(\mathbf{x}, \mathbf{y}) := \sum_{1 \le i_1 < ... < i_D \le N} \prod_{d=1}^{D} k^{i_d}(x_{i_d}, y_{i_d})$$

string kernel

- similarities between two documents

$\sum$=alphabet, $\sum^n$=set of all strings of length n

for a given index sequence $\mathbf{i} = (1 \le i_1 < i_2 < ... < i_r \le |s|)$

define $s(\mathbf{i}) := s(i_1)s(i_2)....s(i_r)$ and $l_s(\mathbf{i}) = i_r - i_1 + 1 \ge r$

example $s = \texttt{fast food}$,$\mathbf{i} = (2, 3, 9) \Rightarrow s(\mathbf{i}) = \texttt{asd}, l_s(\mathbf{i}) = 9 - 2 + 1 = 8$

$0 < \lambda \le 1$ parameter, define $[\Phi_n(s)]$ a map with $|\sum^n|$ components

$$[\Phi_n(s)]_u = \sum_{\mathbf{i}: s(\mathbf{i}) = u} \lambda^{l_s(\mathbf{i})}$$

example $[\Phi_3(\texttt{Nasdaq})]_{\texttt{asd}} = \lambda^3$, $[\Phi_3(\texttt{lass das})]_{\texttt{asd}} = 2\lambda^5$

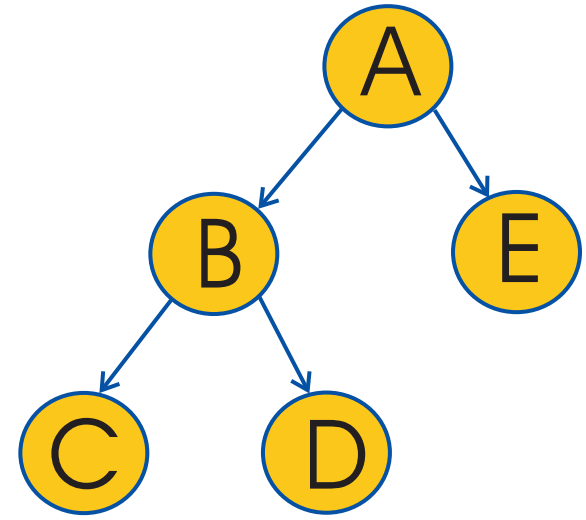
the kernel induced

$$k_n(s, t) = \sum_{u \in \sum^n} [\Phi_n(s)]_u [\Phi_n(t)]_u = \sum_{u \in \sum^n} \sum_{(\mathbf{i}, \mathbf{j}): s(\mathbf{i}) = t(\mathbf{j}) = u} \lambda^{l_s(\mathbf{i})} \lambda^{l_t(\mathbf{j})}$$

$k := \sum_n c_n k_n$ linear combination of kernels on different substring-lengths

tree kernel

• encode a tree as a string by traversing
in preorder and parenthesing



`tag(T)=(A(B(C)(D))(E))`

- substrings correspond to subset trees
- tag can be computed in loglinear time
- then use a string kernel