Support Vector Machines and Kernels: A Tutorial

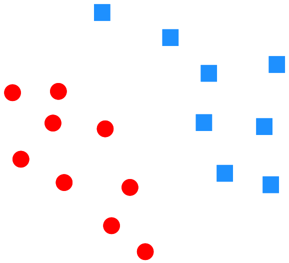
Juan José Burred

Audionamix

February 17, 2012

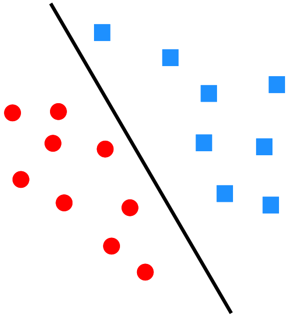
The problem

- Separate these two point classes:



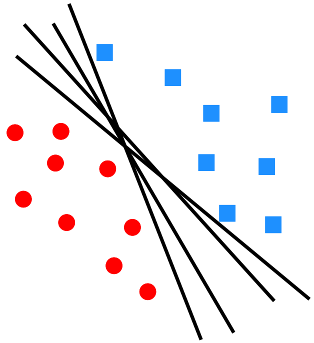
The problem

- Separate these two point classes:



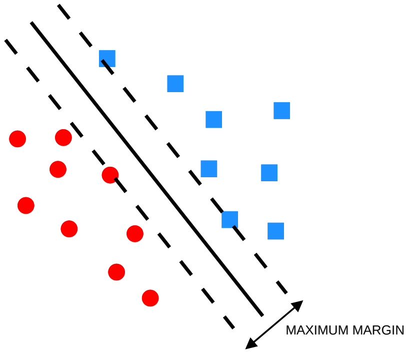
The problem

- Separate these two point classes:



The problem

- Separate these two point classes:



- Goal of SVM: find the separation line between **two linearly separable classes** that ensures the maximum separation margin.

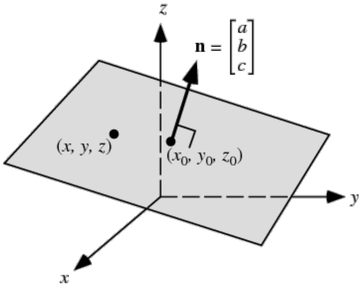
Terminology and formalization

- Machine learning slang: we want to **train a classifier** (learn a decision rule to classify a new incoming vector) from a **training database** of **feature vectors**.
- The training database is labeled (**supervised learning**):
  - 2 labels: $y_i \in \{-1, 1\}$
  - Training database: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | \boldsymbol{x}_i \in \mathbb{R}^N\}$
- Decision function: $f(\boldsymbol{x}_i) = y_i$

Expression of decision boundary

in N dimensions

- Separating hyperplane: $\mathcal{P} = \{\boldsymbol{x} | \boldsymbol{w}(\boldsymbol{x} - \boldsymbol{b}) = 0\}$
- Alternatively: $\mathcal{P} = \{\boldsymbol{x} | \boldsymbol{w}\boldsymbol{x} - b = 0\}$, where:
  - $\boldsymbol{w}$ is a vector normal to the plane
  - $\boldsymbol{b}$ is a vector pointing from the origin to the plane in the direction of $\boldsymbol{w}$
  - $b = \|\boldsymbol{b}\|\|\boldsymbol{w}\|$ is a bias parameter proportional to the distance from the plane to the origin
  - The distance from the plane to the origin is $\frac{b}{\|\boldsymbol{w}\|}$



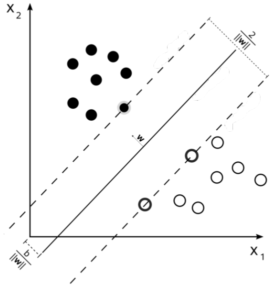
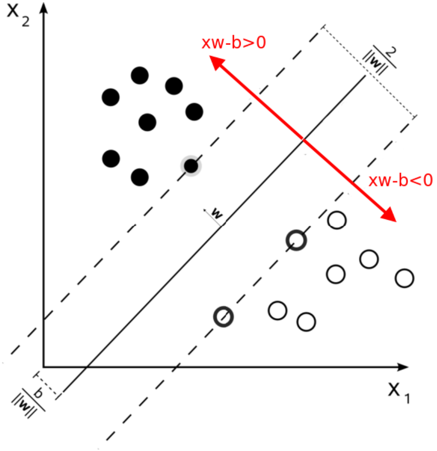


Image: http://mathworld.wolfram.com/ (left), https://en.wikipedia.org/ (right)

A closer look at the decision regions

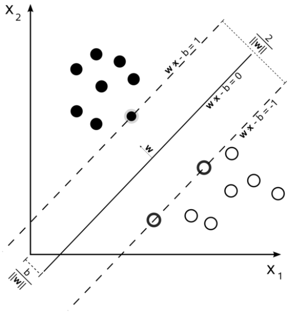


- We want to select the hyperplane parameters $\boldsymbol{w}$ and b that maximize the separation margin.
- No feature vectors inside the margin.
- The margin boundaries are given by $\boldsymbol{xw} - b = \pm k$ for some k.
- However, there are infinite solutions for different couples of $\boldsymbol{w}$ and b.

- To avoid that, we scale $\boldsymbol{w}$ and b so that the margin boundaries are $\boldsymbol{xw} - b = \pm 1$. This is called the **canonical hyperplane**.
- This is equivalent to imposing the constraint $y_i(\boldsymbol{x}_i\boldsymbol{w} - b) \geq 1$ on all training samples $\boldsymbol{x}_i$.

Image: http://en.wikipedia.org/wiki/Support_vector_machine

Optimization problem



x_2
x_1
$\vec{w}\cdot\vec{x}-b=1$
$\vec{w}\cdot\vec{x}-b=0$
$\vec{w}\cdot\vec{x}-b=-1$
$\frac{2}{\|\vec{w}\|}$
$\frac{b}{\|\vec{w}\|}$
$\vec{w}$

- With the canonical hyperplane, the width of the maximum margin is $\frac{2}{\|\boldsymbol{w}\|}$
- To maximize the margin, we need to minimize $\|\boldsymbol{w}\|$.
- However, there is a square root involved ($\|\boldsymbol{w}\| = \sqrt{w_1^2 + w_2^2 + \ldots}$), which makes the optimization difficult.
- Instead, we minimize $\frac{1}{2}\|\boldsymbol{w}\|^2$
  - This is a Quadratic Programming (QP) problem
  - Global minimum guaranteed.
  - The $\frac{1}{2}$ factor is for convenience (see later)

SVM optimization

$$\text{Minimize } J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2$$
$$\text{subject to } y_i(\boldsymbol{x}_i\boldsymbol{w} - b) \geq 1 \text{ for all } i.$$

Image: http://en.wikipedia.org/wiki/Support_vector_machine

How do we solve this?

SVM optimization

Minimize $J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2$ subject to $y_i(\boldsymbol{x}_i\boldsymbol{w} - b) \geq 1$ for all i.

- Can we use Lagrange multipliers?
- Reminder:
  - Optimization without constraints: set $\nabla_\theta J(\boldsymbol{\theta}) = 0$ and solve.
  - Optimization with i equality constraints: **Lagrange multipliers**.
    - Optimize $J(\boldsymbol{\theta})$ subject to $g(\boldsymbol{\theta}) = C$.
    - First, build Lagrangian: $\mathcal{L}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda(g(\boldsymbol{\theta}) - C)$
    - Then, set $\nabla_{\boldsymbol{\theta},\lambda}\mathcal{L}(\boldsymbol{\theta}) = 0$ and solve.
    - Multiple constraints: $\mathcal{L}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \sum_i \lambda_i(g_i(\boldsymbol{\theta}) - C_i)$
- But here, we have an **inequality constraint**! $(y_i(\boldsymbol{x}_i\boldsymbol{w} - b) \geq 1)$
- We cannot directly use Lagrange multipliers.

Generalized Lagrangian

- However, it is possible to use a Lagrange-like optimization with inequalities if some conditions are met.
- These are called the **Karush-Kuhn-Tucker (KKT)** conditions.

Generalized Lagrangian

Optimize $J(\boldsymbol{\theta})$ subject to $g_i(\boldsymbol{\theta}) = 0$ and $h_i(\boldsymbol{\theta}) \leq 0$

$$\mathcal{L}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \sum_i \lambda_i g_i(\boldsymbol{\theta}) + \sum_j \alpha_j h_j(\boldsymbol{\theta})$$

There exists an optimal solution to the above problem if the following conditions are met:

Karush-Kuhn-Tucker conditions

1. **Primal feasibility**: $g_i(\boldsymbol{\theta}) = 0 \ \forall i$, $h_i(\boldsymbol{\theta}) \leq 0 \ \forall i$
2. **Dual feasibility**: $\alpha_j \geq 0 \ \forall j$
3. **Complementary slackness**: $\alpha_j h_j(\boldsymbol{\theta}) = 0$

Primal SVM problem

Let's recast our problem:

$$\text{Minimize } J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2 \text{ subject to } y_i(\boldsymbol{x_i}\boldsymbol{w} - b) \geq 1 \text{ for all } i.$$

as a generalized Lagrangian:

$$\mathcal{L}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \sum_i \lambda_i g_i(\boldsymbol{\theta}) + \sum_j \alpha_j h_j(\boldsymbol{\theta})$$

In our case we have:

- $J(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|^2$
- $\lambda_i = 0$ (no equality constraints)
- $h_i(\boldsymbol{\theta}) = -[y_i(\boldsymbol{x_i}\boldsymbol{w} - b) - 1]$

We get the so-called Lagrangian primal problem for SVM:

Primal SVM problem

$$\text{Minimize } \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i[y_i(\boldsymbol{x_i}\boldsymbol{w} - b) - 1]$$

subject to the KKT conditions.

Dual SVM problem

Let's try to solve the primal problem

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i [y_i(\boldsymbol{x_i w} - b) - 1]$$

- Setting $\nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = 0$, we get $\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$
- Setting $\nabla_b \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = 0$, we get $\sum_i \alpha_i y_i = 0$
- Substituting back into the primal equation we get a simplified problem definition:

Dual SVM problem

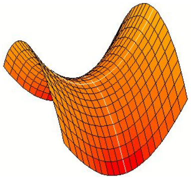
Minimize $\mathcal{L}(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \boldsymbol{x}_j$
subject to the KKT conditions.

Primal vs dual SVM problems

Primal SVM problem

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i [y_i(\boldsymbol{x_i w} - b) - 1]$$

- Depends on 3 parameters ($\boldsymbol{w}$, b, $\boldsymbol{\alpha}$)
- Difficult to optimize (saddle point)

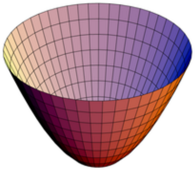


- Depends on dot products between data and a model parameter: $\boldsymbol{w x}_i$.

Dual SVM problem

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \boldsymbol{x}_j$$

- Only depends on 1 parameter ($\boldsymbol{\alpha}$)
- Easier to optimize (global minimum)



- Depends only on dot products between pairs of training data points $\boldsymbol{x}_i \boldsymbol{x}_j$ (important later!)

SVM solution

- We solve the Dual SVM problem via QP, obtaining one Lagrange multiplier α_i for each training vector $\boldsymbol{x}_i$
- Remember that $\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$. We use this to obtain parameter $\boldsymbol{w}$ from the multipliers.
- Remember also the definition of the separating hyperplane: $\boldsymbol{w}\boldsymbol{x} - b = 0$
- Thus, for a trained $\boldsymbol{w}$ we have the following decision function for an incoming unlabeled vector $\boldsymbol{x}$:

$$f(\boldsymbol{x}) = \begin{cases} 1 & \textit{if} \quad \boldsymbol{w}\boldsymbol{x} - b \geq 0 \\ -1 & \textit{if} \quad \boldsymbol{w}\boldsymbol{x} - b < 0 \end{cases} = \mathrm{sgn}(\boldsymbol{w}\boldsymbol{x} - b)$$

- We can finally substitute $\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$ to see that the solution only depends on the multipliers:

SVM decision function

$$f(\boldsymbol{x}) = \mathrm{sgn}(\textstyle\sum_i \alpha_i y_i \boldsymbol{x}\boldsymbol{x}_i - b)$$

Support vectors

- Remember the 3rd KKT condition? $\alpha_i h_i(\boldsymbol{\theta}) = 0 \ \forall i$
- In our case, this means: $-\alpha_i[y_i(\boldsymbol{x}_i\boldsymbol{w} - b) - 1] = 0$
- For this to hold, either:
  - $\alpha_i = 0$ (the training vector does not affect the decision), or
  - $y_i(\boldsymbol{x}_i\boldsymbol{w} - b) = 1$ (the training vector lies on one of the margin boundaries)

- In other words: **only the training vectors lying on the margin boundaries affect the decision!**
- They are called **support vectors**.
- After training, all the other vectors can be discarded. The model is defined only by the support vectors.

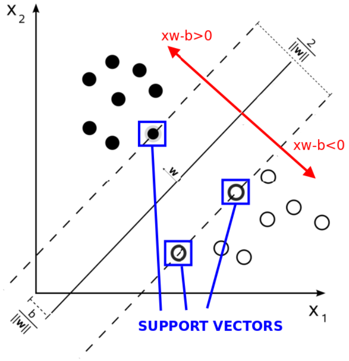


Image: http://en.wikipedia.org/wiki/Support_vector_machine

Are we done?

SVM decision function

$$f(\boldsymbol{x}) = \mathrm{sgn}(\textstyle\sum_i \alpha_i y_i \boldsymbol{x}\boldsymbol{x}_i - b)$$

- There is still one little detail missing: obtaining b!
- We can rely on the support vectors. For one of them ($\boldsymbol{x}_n$), we can write the following ($\mathcal{S}$ is the set of support vector indices):

$$y_n\left(\sum_{m\in\mathcal{S}} \alpha_m y_m \boldsymbol{x}_n \boldsymbol{x}_m - b\right) = 1$$

- Multiplying both sides by y_n ($y_n^2 = 1$):

$$b = \sum_{m\in\mathcal{S}} \alpha_m y_m \boldsymbol{x}_n \boldsymbol{x}_m - y_n$$

- Actually, you get more numerically stable results averaging across the N_s support vectors:

$$b = \frac{1}{N_s}\sum_{n\in\mathcal{S}}\left(\sum_{m\in\mathcal{S}} \alpha_m y_m \boldsymbol{x}_n \boldsymbol{x}_m - y_n\right)$$

Linear SVM: Summary

For a labeled training database of two classes $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | \boldsymbol{x}_i \in \mathbb{R}^N\}$, the SVM is trained by the following QP optimization:

SVM training

$$\text{Minimize } \mathcal{L}(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \boldsymbol{x}_j$$
$$\text{subject to } \alpha_i \geq 0 \text{ and } \sum_i \alpha_i y_i = 0.$$

We obtain one Lagrange multiplier α_i for each training vector $\boldsymbol{x}_i$. With this, classification of an unlabeled vector $\boldsymbol{x}$ is performed as follows:

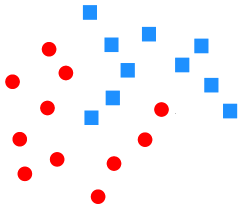
SVM classification

$$f(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{m \in \mathcal{S}} \alpha_m y_m \boldsymbol{x} \boldsymbol{x}_m - b\right)$$

where b is given by $b = \frac{1}{N_s} \sum_{n \in \mathcal{S}} \left(\sum_{m \in \mathcal{S}} \alpha_m y_m \boldsymbol{x}_n \boldsymbol{x}_m - y_n\right)$

Back to reality

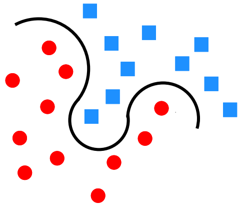
- Unfortunately, all we saw before only apply to linearly-separable cases (unrealistic).
- How do you classify this?



- In practice, classes are almost never linearly separable.

Back to reality

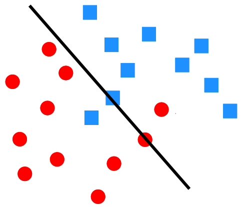
- Unfortunately, all we saw before only apply to linearly-separable cases (unrealistic).
- How do you classify this?



- In practice, classes are almost never linearly separable.
- Complex decision regions produce **overfitting**: perfect classification on training database, but poorly generalizable.

Back to reality

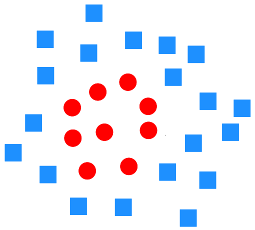
- Unfortunately, all we saw before only apply to linearly-separable cases (unrealistic).
- How do you classify this?



- In practice, classes are almost never linearly separable.
- Complex decision regions produce **overfitting**: perfect classification on training database, but poorly generalizable.
- **Occam's razor**: keep your model simple! Allow some errors on training. General performance on unknown data will be better.

Even harder...

- OK great, so how about... **this!**



- Now we are very far from linear separability, even after allowing some errors.
- How can we solve this problem while keeping a simple decision region (e.g. SVM)?

Cover's theorem

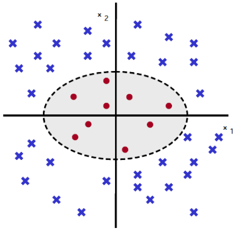
- The trick is to deform the feature space while keeping the linear boundary!

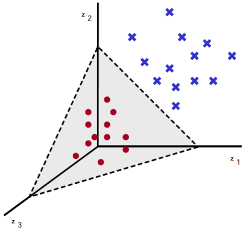
Cover's theorem

A complex pattern classification problem cast in a high-dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space.

- Example of a nonlinear mapping $\varphi : \mathbb{R}^2 \to \mathbb{R}^3$:

$$(x_1, x_2) \to (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$$





Nonlinear mappings in practice

Directly applying such a mapping to a classification problem is tempting, but there are two issues to be considered:

- **Curse of dimensionality**
  - Usually, in machine learning we perform dimensionality reduction to avoid the curse of dimensionality.
  - The curse of dimensionality means that adding new dimensions (adding new features) increases the sparsity of the learning set, and thus increases the difficulty of learning the true, underlying, general decision boundaries if the training database is not well populated.
  - So, how can adding new dimensions be beneficial?
  - In fact, SVM is virtually **immune to the curse of dimensionality** since its generalization is a function of the margin, regardless of the dimensionality!!
- **Computational complexity**
  - Performing a mapping $\varphi(\boldsymbol{x}_i)$ for each training feature can be extremely costly.
  - However, there is a way of performing the mapping without really performing the mapping.
  - ... wait.... what?

Explicit and implicit mapping

An example

- Take the previous example of mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$: $\varphi(\boldsymbol{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T$
- Let's transform a dot product in the original space $(\boldsymbol{xy})$ into a dot product in the mapped space:

$$\varphi(\boldsymbol{x})\varphi(\boldsymbol{y}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)^T(y_1^2, \sqrt{2}y_1y_2, y_2^2) =$$

$$= x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2 = (x_1y_1 + x_2y_2)^2 = (\boldsymbol{xy})^2$$

- If we define a function $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{xy})^2$, we can write:

$$\varphi(\boldsymbol{x})\varphi(\boldsymbol{y}) = k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{xy})^2$$

- In other words, we have performed an implicit mapping of dot products **without explicitly mapping each vector with $\varphi(\boldsymbol{x})$!!**
- Function $k(\boldsymbol{x}, \boldsymbol{y})$ is called the **kernel** (implicit mapping) corresponding to an explicit mapping $\varphi(\boldsymbol{x})$.

The kernel trick

- More generally: a given nonlinear mapping $\varphi(\boldsymbol{x})$ has an associated kernel if we can find a function k such that:

Kernel

$$k(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x})\varphi(\boldsymbol{y})$$

- A kernel $k(\boldsymbol{x}, \boldsymbol{y})$ defines an implicit mapping $\varphi(\boldsymbol{x})$ if it fulfills **Mercer's condition**: its Gram matrix has to be positive semi-definite.
  - **Gram matrix**: Matrix of all possible dot products: $G_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  - **Positive semi-definite matrix**: $\boldsymbol{x}^T \boldsymbol{G} \boldsymbol{x} \geq 0 \ \forall \boldsymbol{x}$
- Therefore:

The kernel trick

If a learning algorithm depends only on dot products between feature fectors, it can be easily projected into a high-dimensional nonlinear space by replacing the dot products with the kernel.

- And... remember what algorithm depended only on dot products between features???

Kernel SVMs

Exactly the same than linear SVMs, but replacing the dot products $\boldsymbol{x}_i \boldsymbol{x}_j$ with the kernels $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$:

Kernel SVM training

$$\text{Minimize } \mathcal{L}(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Kernel SVM classification

$$f(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{m \in \mathcal{S}} \alpha_m y_m k(\boldsymbol{x}, \boldsymbol{x}_m) - b\right)$$

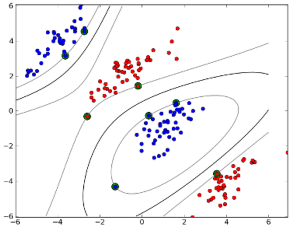


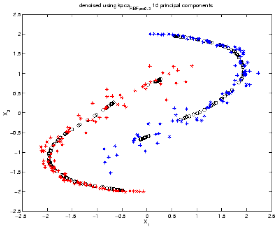
Image: http://archive.is/Aon6l

Examples of valid kernels

- **Homogeneous polynomial**: $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{xy})^d$
- **Inhomogeneous polynomial**: $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{xy} + \phi)^d$
- **Radial Basis Function (RBF)**: $k(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{y}\|^2)$
- **Sigmoidal**: $k(\boldsymbol{x}, \boldsymbol{y}) = \tanh(\beta_0 \boldsymbol{xy} + \beta_1)$
- **Inverse multiquadric**: $k(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{\sqrt{\|\boldsymbol{x}-\boldsymbol{y}\|^2 + c^2}}$
- And many others...
- There are rules to construct complex and structured kernels (pdfs, mixture models, temporal information...)
- Kernels can be interpreted as nonlinear generalizations of similarity measures.

Kernels in other algorithms

- **Kernel PCA**: instead of computing the eigenvectors of the covariance matrix, the eigenvectors of the kernel Gram matrix are computed.



- **Kernel NMF**: The factorized matrix is a kernel Gram matrix. Useful for feature extraction and classification.

$$\varphi(\boldsymbol{V}) = \boldsymbol{WH} \rightarrow \varphi(\boldsymbol{V})^T \varphi(\boldsymbol{V}) = \varphi(\boldsymbol{V})^T \boldsymbol{WH} \rightarrow \boldsymbol{G} = \boldsymbol{YH}$$

Image: https://www.esat.kuleuven.be/sista/lssvmlab/tutorial/node25.html