Support vector machines and kernel trick

Victor Kitov

Table of Contents

1. Support vector machines
   - Linearly separable case
   - Linearly non-separable case
2. Kernel trick
3. Kernel support vector machines
4. Kernel SVM visualization

1. Support vector machines
   - Linearly separable case
   - Linearly non-separable case

Support vector machines

(a) (b)

Support vector machines

(a) (b)

Main idea

Select hyperplane maximizing the spread between classes.

Support vector machines

Objects x_i for $i = 1, 2, ...n$ lie at distance $b/|w|$ from discriminant hyperplane if

$$\begin{cases} x_i^T w + w_0 \geq b, & y_i = +1 \\ x_i^T w + w_0 \leq -b & y_i = -1 \end{cases} \quad i = 1, 2, ...N.$$

This can be rewritten as

$$y_i(x_i^T w + w_0) \geq b, \quad i = 1, 2, ...N.$$

The margin is equal to $2b/|w|$. Since w, w_0 and b are defined up to multiplication constant, we can set $b = 1$.

Problem statement

Problem statement:

$$\begin{cases} \frac{1}{2}w^T w \to \min\limits_{w,w_0} \\ y_i(x_i^T w + w_0) \geq 1, \quad i = 1, 2, ...N. \end{cases}$$

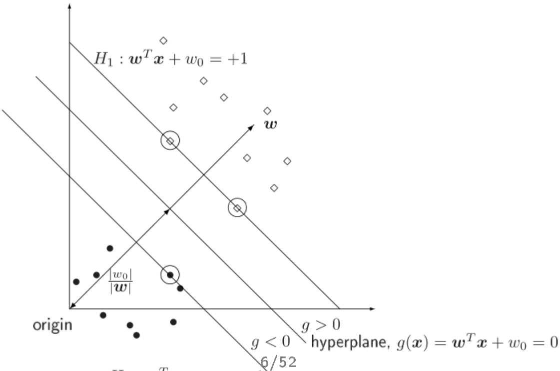
Support vectors

non-informative observations: $y_i(x_i^T w + w_0) > 1$

- do not affect the solution

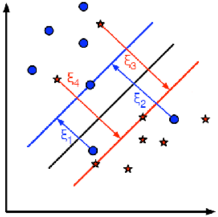
support vectors: $y_i(x_i^T w + w_0) = 1$

- lie at distance $1/|w|$ to separating hyperplane
- affect the the solution.

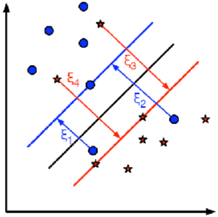


1. Support vector machines
   - Linearly separable case
   - Linearly non-separable case

Linearly non-separable case

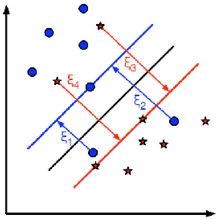


Linearly non-separable case



$$\begin{cases} \frac{1}{2}w^T w \to \min\limits_{w,w_0} \\ y_i(x_i^T w + w_0) \geq 1, \quad i = 1, 2, ...N. \end{cases}$$

Linearly non-separable case



$$\begin{cases} \frac{1}{2}w^T w \to \min_{w,w_0} \\ y_i(x_i^T w + w_0) \geq 1, \quad i = 1, 2, ...N. \end{cases}$$

Problem

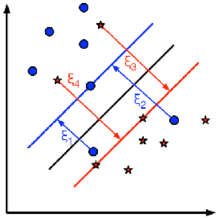
Constraints become incompatible and give empty set!

Linearly non-separable case

No separating hyperplane exists. Errors are permitted by including slack variables ξ_i:

$$\begin{cases} \frac{1}{2}w^T w + C\sum_{i=1}^N \xi_i \to \min_{w,\xi} \\ y_i(w^T x_i + w_0) \ge 1 - \xi_i,\ i = 1,2,...N \\ \xi_i \ge 0,\ i = 1,2,...N \end{cases}$$

- Parameter C is the cost for misclassification and controls the bias-variance trade-off.
- It is chosen on validation set.
- Other penalties are possible, e.g. $C\sum_i \xi_i^2$.



Classification of training objects

- **Non-informative objects**:
  - $y_i(w^T x_i + w_0) > 1$
- **Support vectors** SV:
  - $y_i(w^T x_i + w_0) \leq 1$
  - **boundary support vectors** $\widetilde{SV}$:
    - $y_i(w^T x_i + w_0) = 1$
  - **violating support vectors**:
    - $y_i(w^T x_i + w_0) > 0$: violating support vector is correctly classified.
    - $y_i(w^T x_i + w_0) < 0$: violating support vector is misclassified.

Solution

1. Solution looks like (for some $\alpha_i^* \in \mathbb{R}$, $i \in SV$, which solve *dual optimization task*)
$$w = \sum_{i \in \mathcal{SV}} \alpha_i^* y_i x_i$$
2. w_0 can be found from any edge equality for boundary support vector[1]:
$$y_i(x_i^T w + w_0) = 1,\ \forall i \in \widetilde{\mathcal{SV}} \tag{1}$$

[1] if no support vectors lie on the boundary, then select best w_0 from $\{-x_n^T w\}_{n=1}^N$ using validation set.

Robust solution for w_0

By multiplying (1) by y_i obtain

$$x_i^T w + w_0 = y_i \quad \forall i \in \widetilde{\mathcal{SV}}$$

By summing over all $i \in \widetilde{\mathcal{SV}}$ for more robust solution we obtain

$$n_{\tilde{SV}} w_0 = \sum_{j \in \tilde{SV}} \left(y_j - x_j^T w \right) = \sum_{j \in \tilde{SV}} y_j - \sum_{j \in \tilde{SV}} x_j^T \sum_{i \in \mathcal{SV}} \alpha_i^* y_i x_i$$

where $n_{\tilde{SV}}$ is the number of boundary support vectors.
Finall solution for w_0:

$$w_0 = \frac{1}{n_{\tilde{SV}}} \left(\sum_{j \in \tilde{SV}} y_j - \sum_{j \in \tilde{SV}} \sum_{i \in \mathcal{SV}} \alpha_i^* y_i x_j^T x_i \right)$$

Making predictions

1. Solve dual task to find α_i^*, $i = 1, 2, ... N$

$$\begin{cases} \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \to \max_\alpha \\ \sum_{i=1}^N \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C \quad (\alpha_i \ge 0,\ r_i \ge 0) \end{cases}$$

2. Find optimal w_0:

$$w_0 = \frac{1}{n_{\tilde{SV}}}\left(\sum_{j\in\tilde{SV}} y_j - \sum_{j\in\tilde{SV}}\sum_{i\in\mathcal{SV}} \alpha_i^* y_i \langle x_i, x_j\rangle\right)$$

3. Using $w = \sum_{i\in\mathcal{SV}} \alpha_i^* y_i x_i$, make prediction for new x:

$$\widehat{y} = \text{sign}[w^T x + w_0] = \text{sign}[\sum_{i\in\mathcal{SV}} \alpha_i^* y_i \langle x_i, x\rangle + w_0]$$

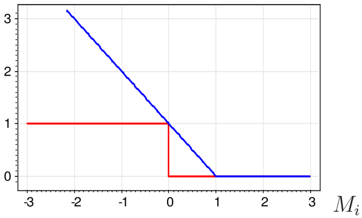
Another view on SVM

Optimization problem:

$$\begin{cases} \frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i \to \min_{w,\xi} \\ y_i(w^T x_i + w_0) = M_i(w, w_0) \geq 1 - \xi_i, \\ \xi_i \geq 0,\ i = 1, 2, ... N \end{cases}$$

can be rewritten as

$$\frac{1}{2C}|w|^2 + \sum_{i=1}^{N}[1 - M_i(w, w_0)]_+ \to \min_{w,\xi}$$



Thus SVM is linear discriminant function with cost approximated with $\mathcal{L}(M) = [1 - M]_+$ and L_2 regularization.

Table of Contents

1. Support vector machines
2. Kernel trick
3. Kernel support vector machines
4. Kernel SVM visualization

Kernel trick

Perform feature transformation: $x \to \phi(x)$. Scalar product becomes $\langle x, x' \rangle \to \langle \phi(x), \phi(x') \rangle = K(x, x')$

Kernel trick

Define not the feature representation x but only scalar product function $K(x, x')$

Kernelization of distance[2]

- Other kernelized algorithms: K-NN, K-means, K-medoids, nearest medoid, PCA, SVM, etc.
- Kernelization of distance:

[2] How can we calculate scalar product between normalized (unit norm) vectors $\phi(x)$ and $\phi(x')$?

Kernelization of distance[2]

- Other kernelized algorithms: K-NN, K-means, K-medoids, nearest medoid, PCA, SVM, etc.
- Kernelization of distance:

$$\begin{aligned}\rho(x,x')^2 &= \langle\phi(x)-\phi(x'),\phi(x)-\phi(x')\rangle\\ &= \langle\phi(x),\phi(x)\rangle+\langle\phi(x'),\phi(x')\rangle-2\langle\phi(x),\phi(x')\rangle\\ &= K(x,x)+K(x',x')-2K(x,x')\end{aligned}$$

[2] How can we calculate scalar product between normalized (unit norm) vectors $\phi(x)$ and $\phi(x')$?

Table of Contents

1. Support vector machines
2. Kernel trick
3. Kernel support vector machines
4. Kernel SVM visualization

Making predictions

1. Solve dual task to find α_i^*, $i = 1, 2, ...N$

$$\begin{cases} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \to \max_\alpha \\ \sum_{i=1}^N \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C \end{cases}$$

2. Find optimal w_0:

$$w_0 = \frac{1}{n_{\tilde{SV}}} \left(\sum_{j \in \tilde{SV}} y_j - \sum_{j \in \tilde{SV}} \sum_{i \in \mathcal{SV}} \alpha_i^* y_i \langle x_i, x_j \rangle \right)$$

3. Make prediction for new x:

$$\widehat{y} = \text{sign}[w^T x + w_0] = \text{sign}[\sum_{i \in \mathcal{SV}} \alpha_i^* y_i \langle x_i, x \rangle + w_0]$$

Making predictions

1. Solve dual task to find α_i^*, $i = 1, 2, ...N$

$$\begin{cases} L_D = \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \to \max_\alpha \\ \sum_{i=1}^N \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C \end{cases}$$

2. Find optimal w_0:

$$w_0 = \frac{1}{n_{\tilde{SV}}}\left(\sum_{j \in \tilde{SV}} y_j - \sum_{j \in \tilde{SV}} \sum_{i \in \mathcal{SV}} \alpha_i^* y_i \langle x_i, x_j \rangle\right)$$

3. Make prediction for new x:

$$\widehat{y} = \text{sign}[w^T x + w_0] = \text{sign}[\sum_{i \in \mathcal{SV}} \alpha_i^* y_i \langle x_i, x \rangle + w_0]$$

- On all steps we don't need exact feature representations, only scalar products $\langle x, x' \rangle$!

Kernel trick generalization

1. Solve dual task to find α_i^*, $i = 1, 2, ...N$

$$\begin{cases} L_D = \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i\alpha_j y_i y_j K(x_i, x_j) \to \max_\alpha \\ \sum_{i=1}^N \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C \quad \text{(using (??) and that } \alpha_i \ge 0,\ r_i \ge 0\text{)} \end{cases}$$

2. Find optimal w_0:

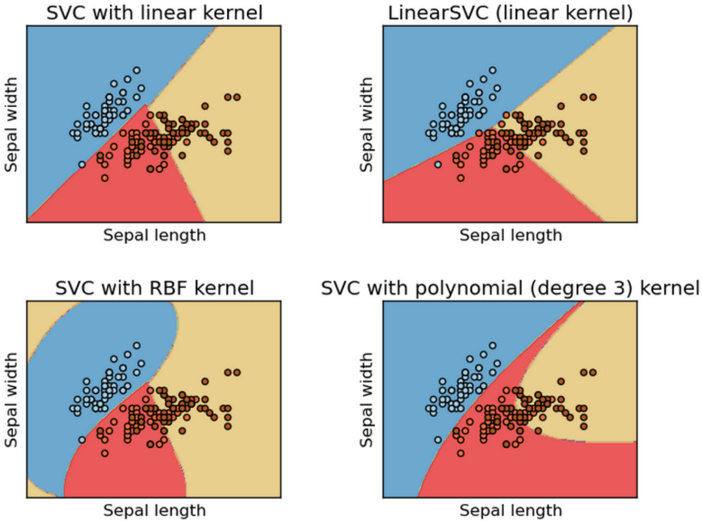
$$w_0 = \frac{1}{n_{\tilde{SV}}}\left(\sum_{j \in \tilde{SV}} y_j - \sum_{j \in \tilde{SV}} \sum_{i \in \mathcal{SV}} \alpha_i^* y_i K(x_i, x_j)\right)$$

3. Make prediction for new x:

$$\widehat{y} = \text{sign}[w^T x + w_0] = \text{sign}[\sum_{i \in \mathcal{SV}} \alpha_i^* y_i K(x_i, x_j) + w_0]$$

- We replaced $\langle x, x'\rangle \to K(x, x')$ for $K(x, x') = \langle \phi(x), \phi(x')\rangle$ for some feature transformation $\phi(\cdot)$.

Kernel results



Kernelizable algorithms

- K-NN
- SVM
- ridge regression:
- K-means
- PCA
- etc...

General motivation for kernel trick

- perform generalization of linear methods to non-linear case
  - we use efficiency of linear methods
  - local minimum is global minimum
  - no local optima=>less overfitting
- non-vectorial objects
  - hard to obtain vector representation

Kernel definition

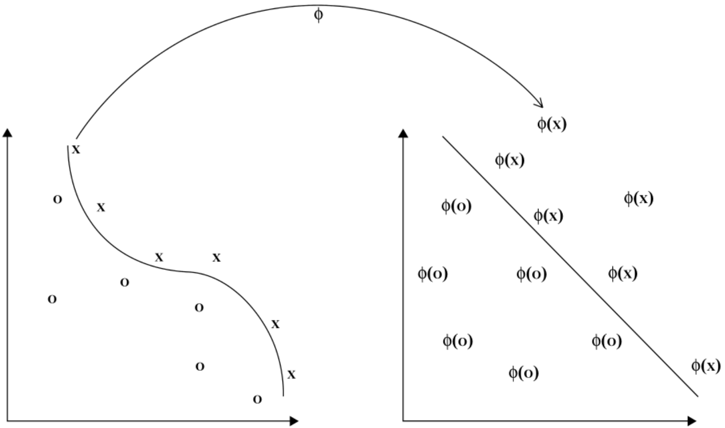
- x is replaced with $\phi(x)$
  - Example: $[x] \to [x,\, x^2,\, x^3]$

Kernel

Function $K(x,x'):\ X \times X \to \mathbb{R}$ is a kernel function if it may be represented as $K(x,x') = \langle \phi(x), \phi(x') \rangle$ for some mapping $\phi:\ X \to H$, with scalar product defined on H.

- $\langle x, x' \rangle$ is replaced by $\langle \phi(x), \phi(x') \rangle = K(x,x')$

Illustration



Polynomial kernel[3]

- Example 1: let $D=2$.

$$\begin{aligned} K(x,z) &= (x^T z)^2 = (x_1 z_1 + x_2 z_2)^2 = \\ &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 z_1 x_2 z_2 \\ &= \phi^T(x)\phi(z) \end{aligned}$$

for $\phi(x) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$

[3] What kind of feature transformation will correspond to $K(x,z) = (x^T z)^M$ for arbitrary M and D?

Polynomial kernel[4]

- Example 2: let $D=2$.

$$\begin{aligned}K(x,z) &= (1+x^Tz)^2=(1+x_1z_1+x_2z_2)^2= \\ &= 1+x_1^2z_1^2+x_2^2z_2^2+2x_1z_1+2x_2z_2+2x_1z_1x_2z_2 \\ &= \phi^T(x)\phi(z)\end{aligned}$$

for $\phi(x)=(1,\,x_1^2,\,x_2^2,\,\sqrt{2}x_1,\,\sqrt{2}x_2,\,\sqrt{2}x_1x_2)$

[4] What kind of feature transformation will correspond to $K(x,z)=(1+x^Tz)^M$ kernels for arbitrary M and D?

Kernel properties

Theorem (Mercer): Function $K(x,x')$ is a kernel is and only if

- it is symmetric: $K(x,x')=K(x',x)$
- it is non-negative definite:
  - definition 1: for every function $g: X\to\mathbb{R}$

$$\int_X\int_X K(x,x')g(x)g(x')dxdx'\geq 0$$

  - definition 2 (equivalent): for every finite set $x_1,x_2,...x_M$ Gramm matrix $\{K(x_i,x_j)\}_{i,j=1}^{M}\succeq 0$ (p.s.d.)

Kernel construction

- Kernel learning - separate field of study.
- Hard to prove non-negative definitness of kernel in general.
- Kernels can be constructed from other kernels, for example from:
  1. scalar product $\langle x, x' \rangle$
  2. constant $K(x, x') \equiv 1$
  3. $x^T Ax$ for any $A \succcurlyeq 0$[5]

[5] Under what feature transformation will case 1 transform to cases 2 and 3? You may use Choletsky decomposition.

Constructing kernels from other kernels

If $K_1(x, x')$, $K_2(x, x')$ are arbitrary kernels, $c > 0$ is a constant, $q(\cdot)$ is a polynomial with non-negative coefficients, $h(x)$ and $\varphi(x)$ are arbitrary functions $\mathcal{X} \to \mathbb{R}$ and $\mathcal{X} \to \mathbb{R}^M$ respectively, then these are valid kernels[6]:

1. $K(x, x') = cK_1(x, x')$
2. $K(x, x') = K_1(x, x')K_2(x, x')$
3. $K(x, x') = K_1(x, x') + K_2(x, x')$
4. $K(x, x') = K_1(\varphi(x), \varphi(x'))$
5. $K(x, x') = h(x)K_1(x, x')h(x')$
6. $K(x, x') = e^{K_1(x, x')}$

[6] prove some of these statements

Commonly used kernels

Let x and x' be two objects and take any $\gamma > 0, r > 0, d > 0$.

Kernel	**Mathematical form**
linear	$\langle x, x' \rangle$
polynomial	$(\gamma \langle x, x' \rangle + r)^d$
RBF	$\exp(-\gamma \|x - x'\|^2)$

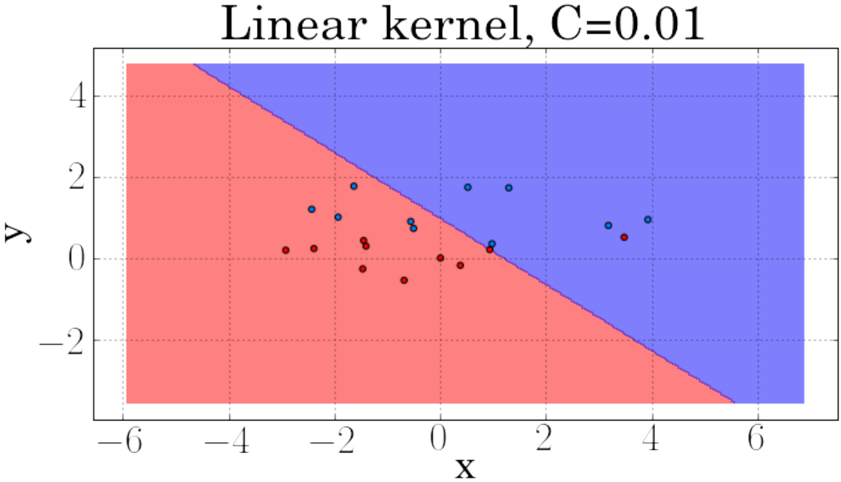
SVM prediction:

$$\widehat{y} = \text{sign}[w^T x + w_0] = \text{sign}[\sum_{i \in \mathcal{SV}} \alpha_i^* y_i K(x_i, x_j) + w_0]$$

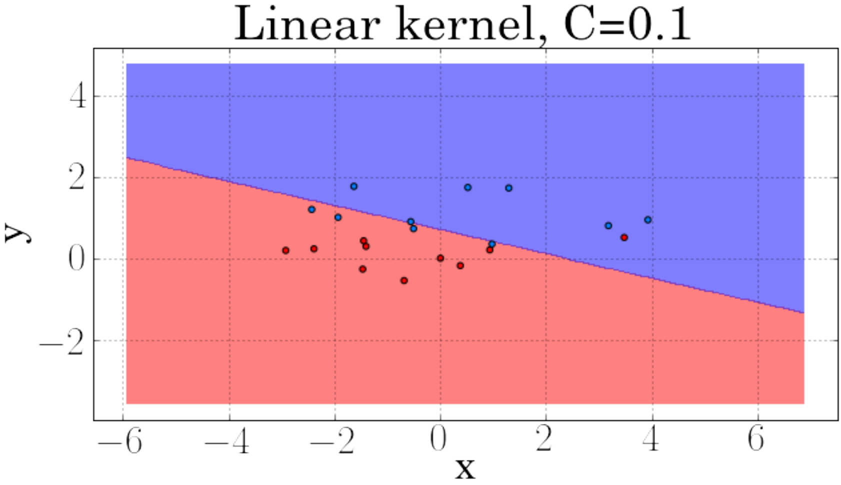
Table of Contents

1. Support vector machines
2. Kernel trick
3. Kernel support vector machines
4. Kernel SVM visualization

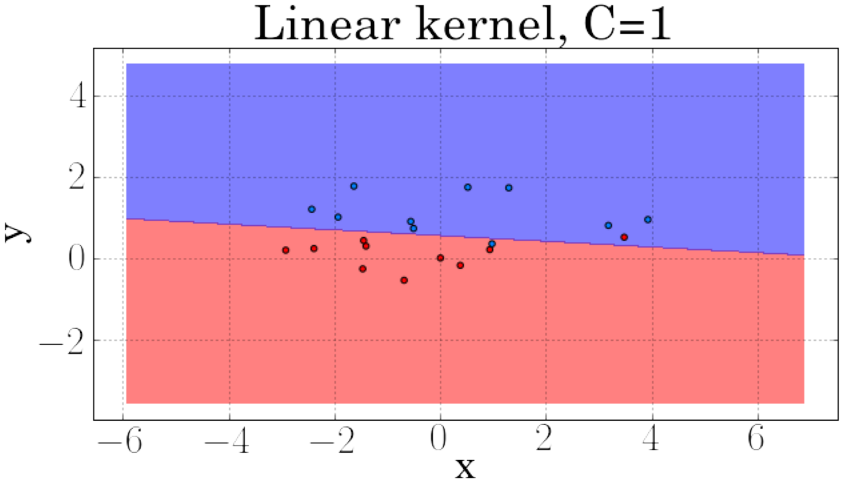
Linear kernel - variable C



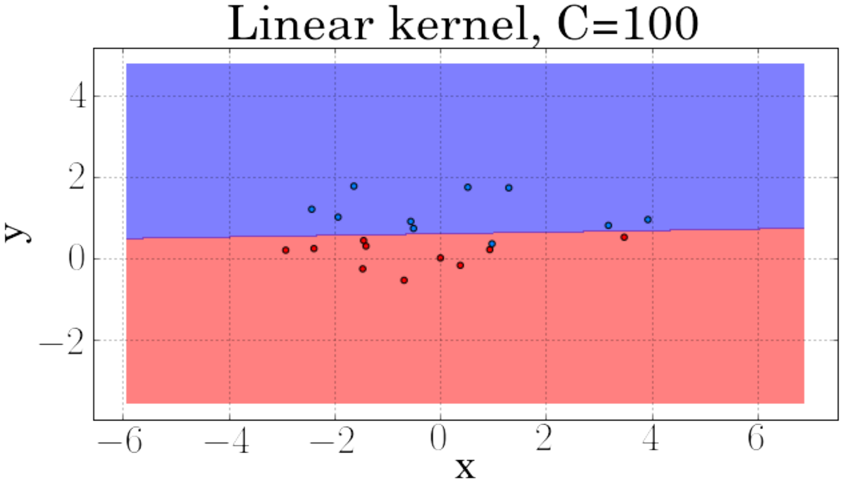
Linear kernel - variable C



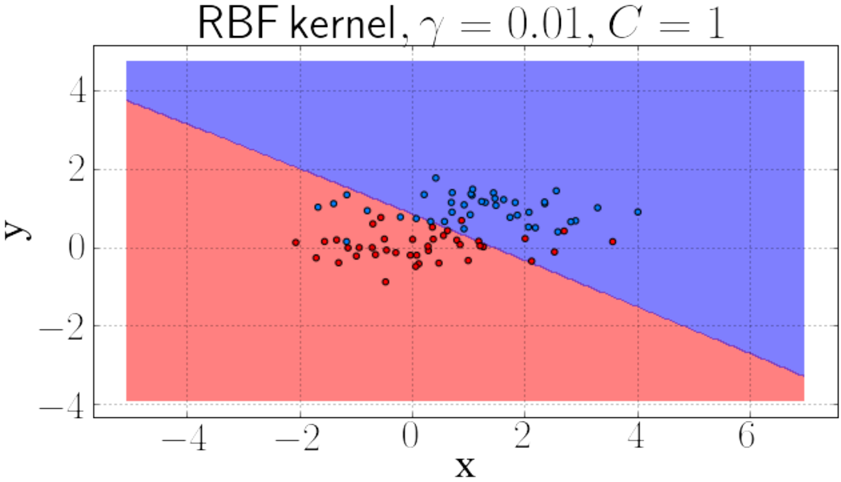
Linear kernel - variable C



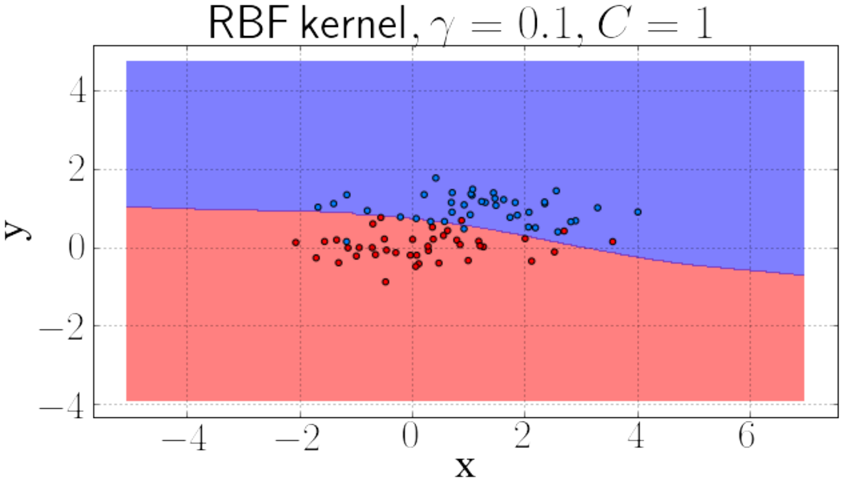
Linear kernel - variable C



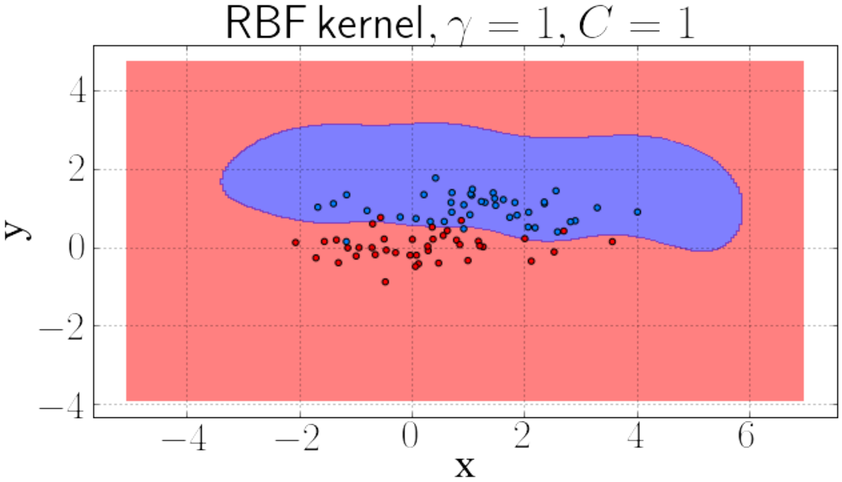
RBF kernel - variable γ



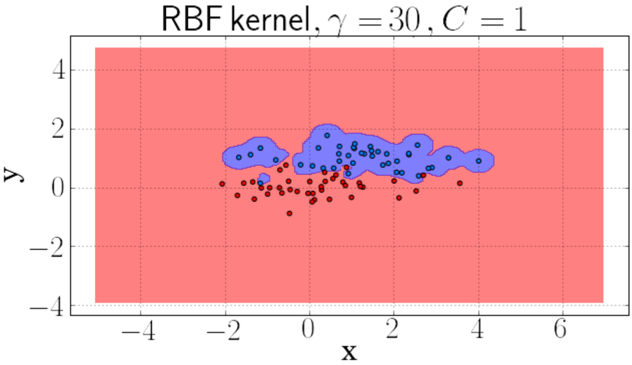
RBF kernel - variable γ



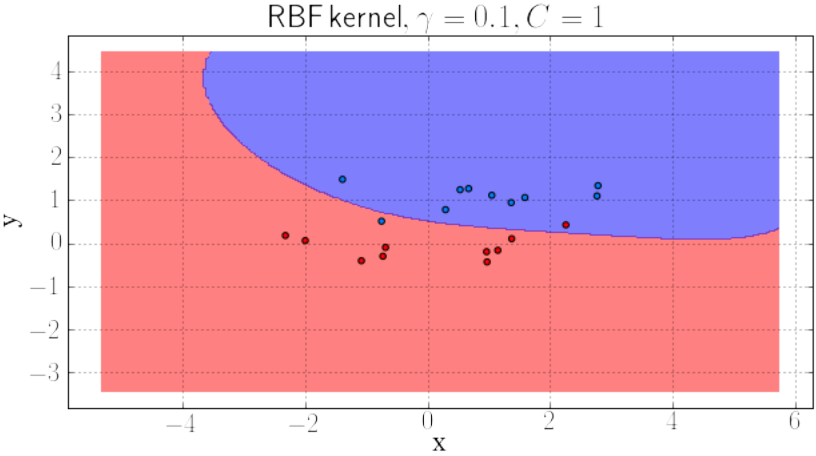
RBF kernel - variable γ



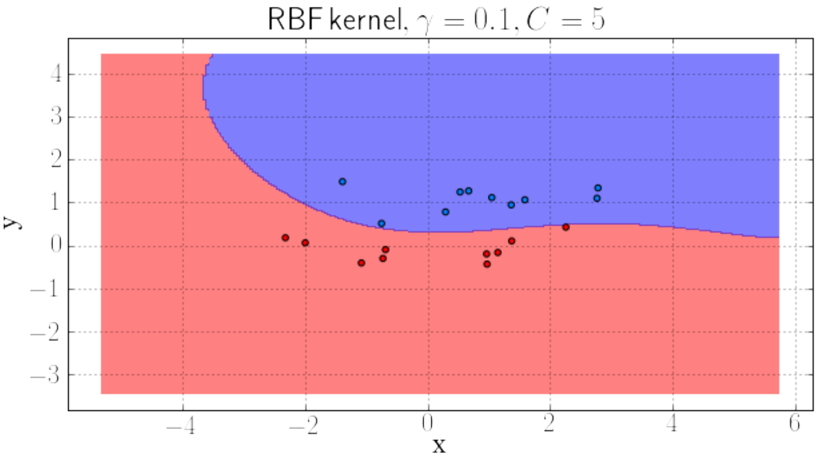
RBF kernel - variable γ



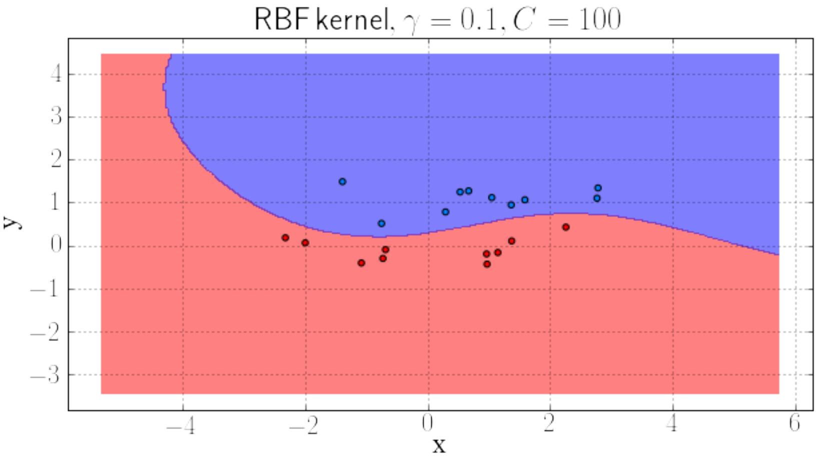
RBF kernel - variable C



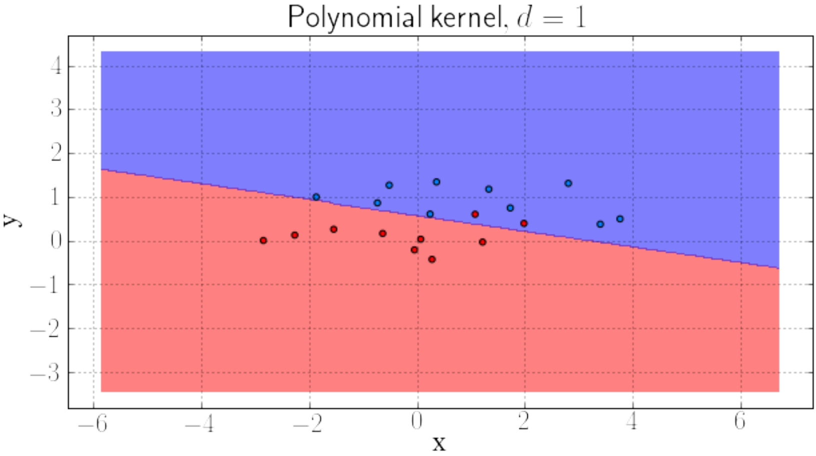
RBF kernel - variable C



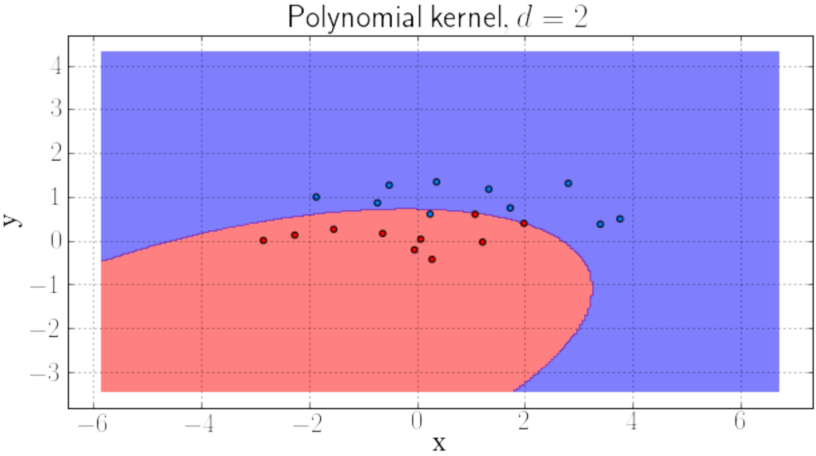
RBF kernel - variable C



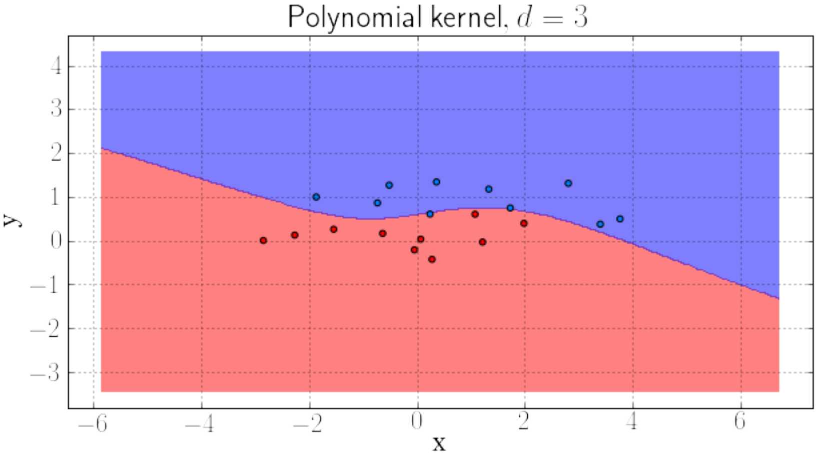
Polynomial kernel - variable d



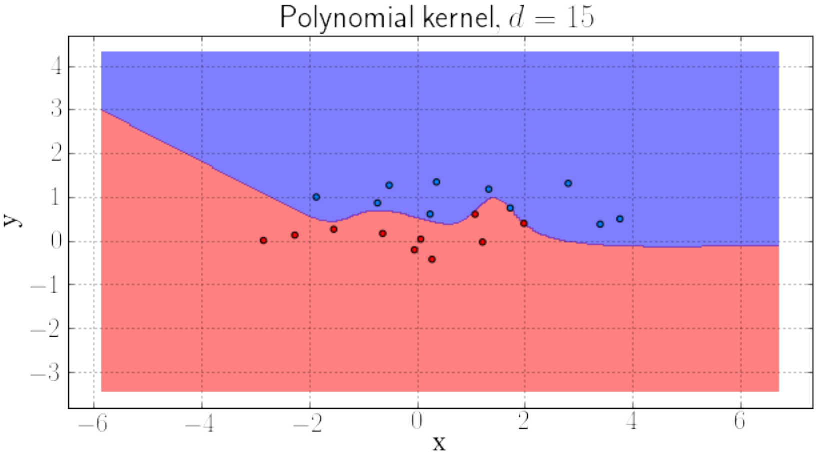
Polynomial kernel - variable d



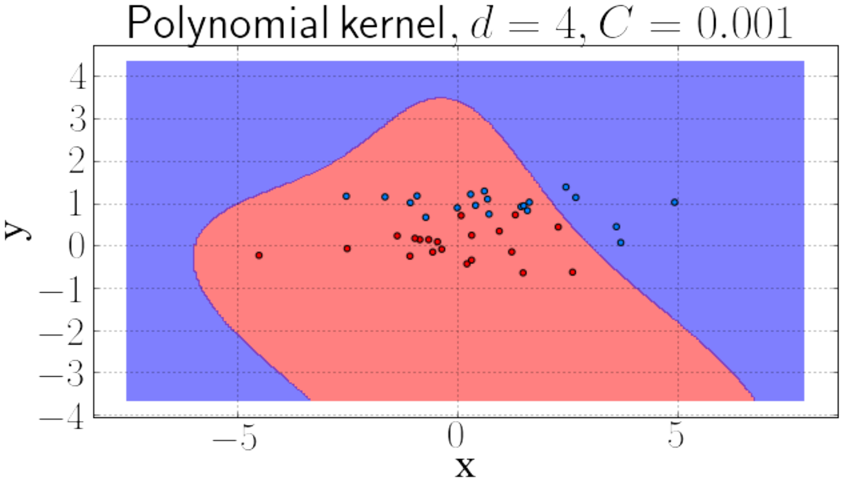
Polynomial kernel - variable d



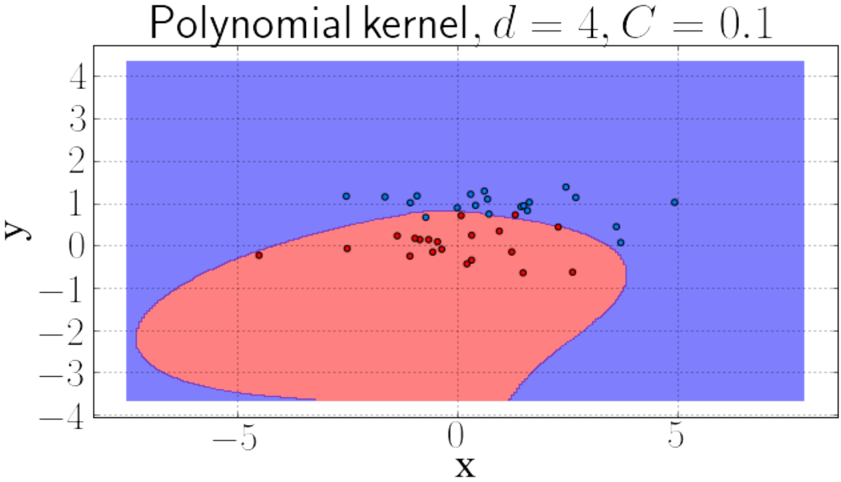
Polynomial kernel - variable d



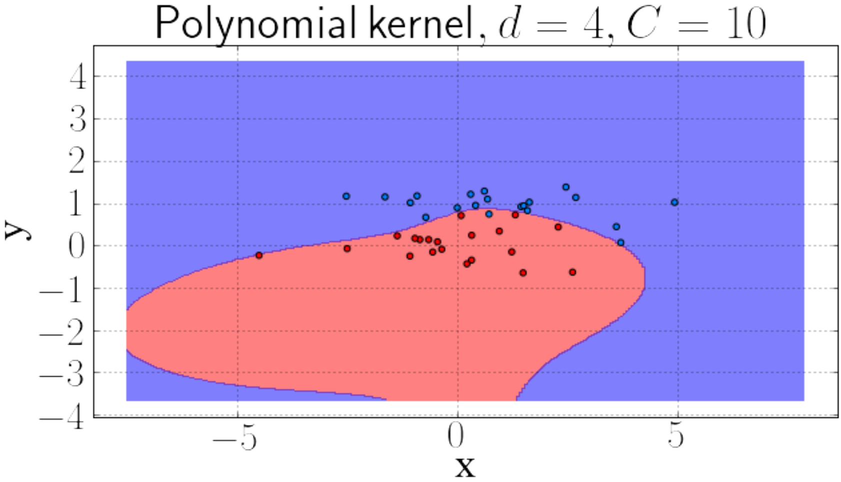
Polynomial kernel - variable C



Polynomial kernel - variable C



Polynomial kernel - variable C



Kernel trick use cases

- high-dimensional data
  - polynomial of order up to M
  - Gaussian kernel $K(x, x') = e^{-\frac{1}{2\sigma^2}\left\|x-x'\right\|^2}$ corresponds to infinite-dimensional feature space.
- hard to vectorize data
  - strings, sets, images, texts, graphs, 3D-structures, sequences, etc.
- natural scalar product exist
  - strings: number of co-occuring substrings
  - sets: size of intersection of sets
    - example: for sets S_1 and S_2: $K(S_1, S_2) = 2^{|S_1 \cap S_2|}$ is a possible kernel.
  - etc.
- scalar product can be computed efficiently