Non Linear SVM

Kernel Trick

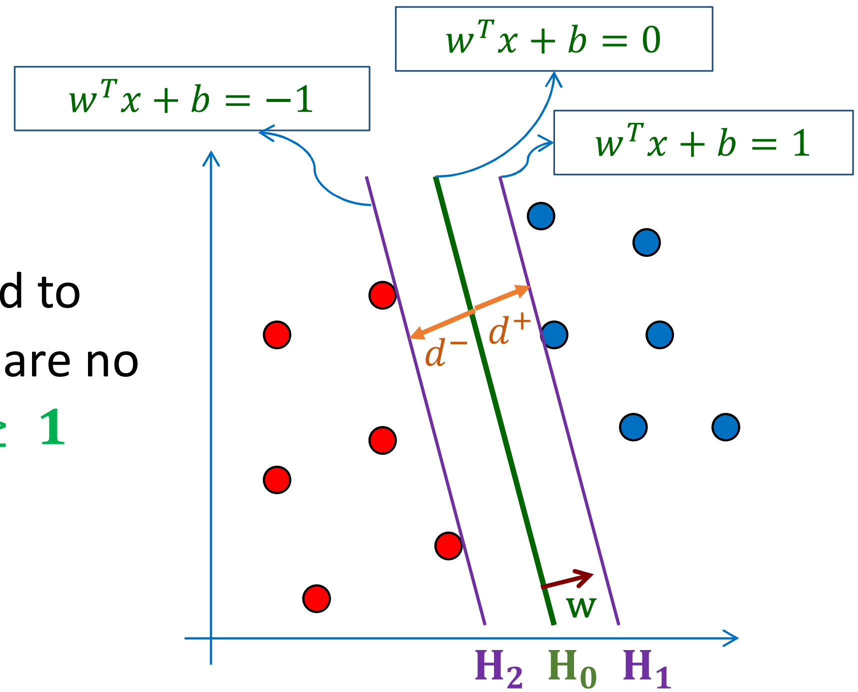
Maximizing the Margin

- We want a classifier (linear separator) with as big a margin as possible.

- In order to maximize the margin, we thus need to minimize $||w||$. With the condition that there are no datapoints between H_1 and H_2 : $\boldsymbol{y_i}\ (\mathbf{w}^{\boldsymbol{T}}\boldsymbol{x_i})\ \geq\ \mathbf{1}$

- Minimize $\mathbf{J(w)} = \frac{1}{2}\boldsymbol{w^T w}$, subject to:
  $\forall_i,\ \boldsymbol{y_i}(\mathbf{w}^{\boldsymbol{T}}\boldsymbol{x_i} + \boldsymbol{b}) \geq \mathbf{1}$

- Constrained quadratic optimization problem solved by the **Lagrangian multipler method**

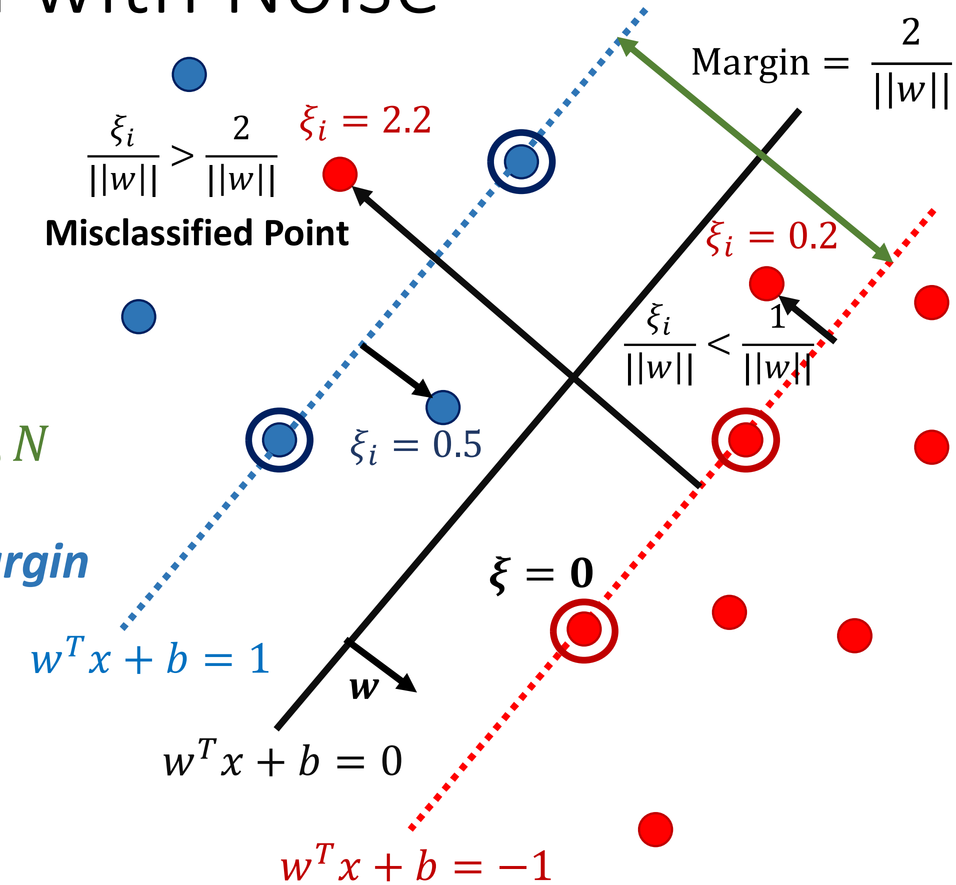


Learning Maximum Margin with Noise

- The error terms $\xi_N^{\prime s}$ are incorporated into our optimization problem by:

$$\begin{cases} \min\limits_{w,b} ||w||^2 + C \sum_{i=1}^{N} \xi_i \\ \text{such that } y_i(w^T x_i + b) \geq 1 - \xi_i,\ \ i = 1, \dots, N \end{cases}$$

- The solution to this problem is called ***soft margin support vector classification***



SVM Solution

Involves computing the ***inner products*** $x_i^T x_j$ between all training points

1. Maximize: $Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boxed{\mathbf{x}_i{}^{\mathrm{T}} \mathbf{x}_j}$, subject to $\mathbf{0} \le \boldsymbol{\alpha}_i \le \boldsymbol{C}\ \forall \boldsymbol{i}$, $\sum_{i=1}^{N} \boldsymbol{\alpha}_i \boldsymbol{y}_i = \mathbf{0}$

- $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$;   (very few α_i s are non-zero: support vectors. The sum is therefore only to be over the support vectors)

- $\mathbf{b} = \boldsymbol{y}_K(\mathbf{1} - \boldsymbol{\xi}_K) - \boldsymbol{w}_K^T \boldsymbol{x}_K = \boldsymbol{y}_K(\mathbf{1} - \boldsymbol{\xi}_K) - \sum_{i=1}^{N} \alpha_i y_i \boxed{x_i^T \mathbf{x}_K}$ with $K = \arg\max_i \alpha_i$

Note: Classification:

$$\mathbf{f}(\mathbf{x}_t) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_t + \mathbf{b} = \sum_{i=1}^{N} \alpha_i y_i \boxed{\mathbf{x}_i{}^{\mathrm{T}} \mathbf{x}_t} + b$$

Relies on an ***inner product*** between the test point $\boldsymbol{x}_t$ and the support vectors $\boldsymbol{x}_i$

Theoretical Justification for Maximum Margin

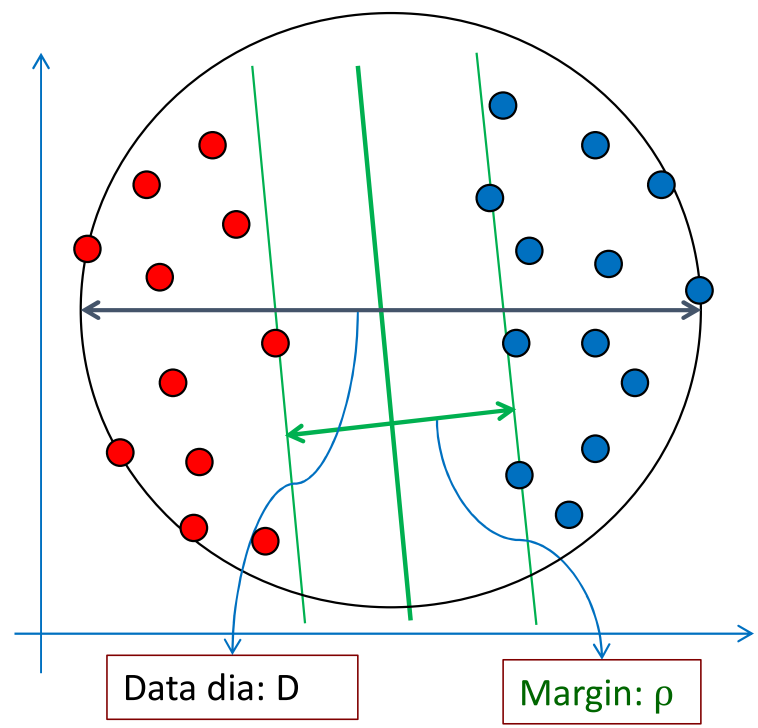
- V.N. Vapnik and A.Ya. Červonenkis quantified complexity
  - Higher the VC-dimension (h), more complex the classifier
  - Bound on Expected Loss:

$$R_{tst}(\alpha) = R_{trn}(\alpha) + f(h, N)$$

$$h \leq min\left\{d, \left\lceil \frac{1}{m^2} \right\rceil\right\} + 1,$$

where d is the dimensionality. $m = {}^{\rho}/_{D}$ is the relative margin, with ρ as the margin, and D as the diameter of the smallest sphere that encloses all of the training examples.

  - Implication: If ρ /D is high, VC dimension is low and expected error is low, regardless of the dimensionality $\boldsymbol{d}$



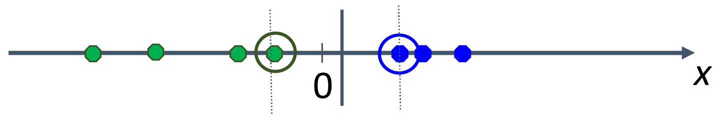
- Complexity of the classifier is kept small regardless of dimensionality.

Linear SVMs: Overview

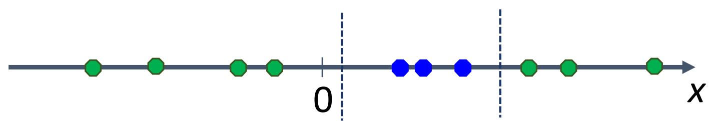
- Convex Optimization guarantees the global optimum.
- Support vectors are automatically identified.
- Use of max margin optimizes test accuracy
  - One of the best classifiers, given a feature representation
  - SVMs works well even with fewer training samples.
    - Note: Minimizes overfitting
  - Does not scale to huge datasets
    - GD based approaches work better in such cases.

Non-linear SVMs

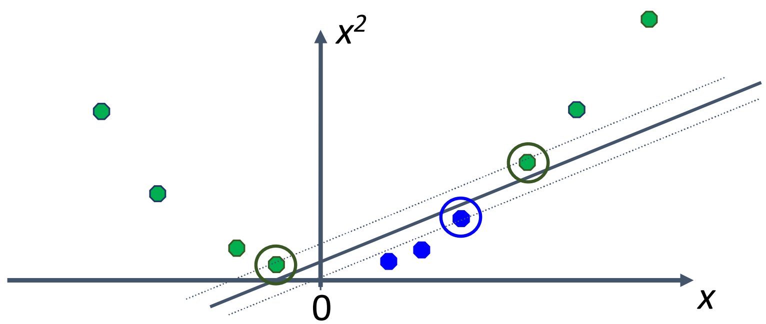
- Datasets that are linearly separable with some noise work out great with Linear SVMs:



- But what should be done if the dataset is just not linearly separable.

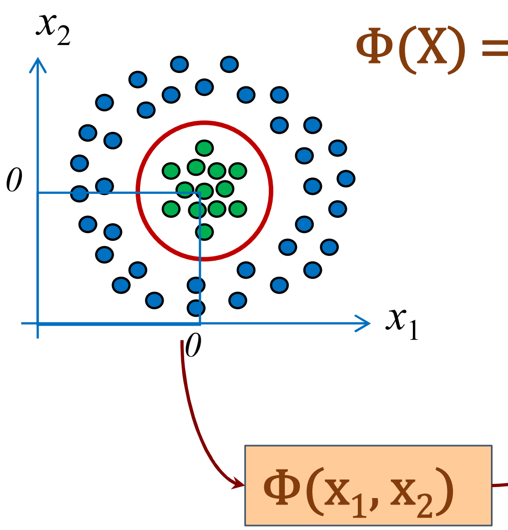


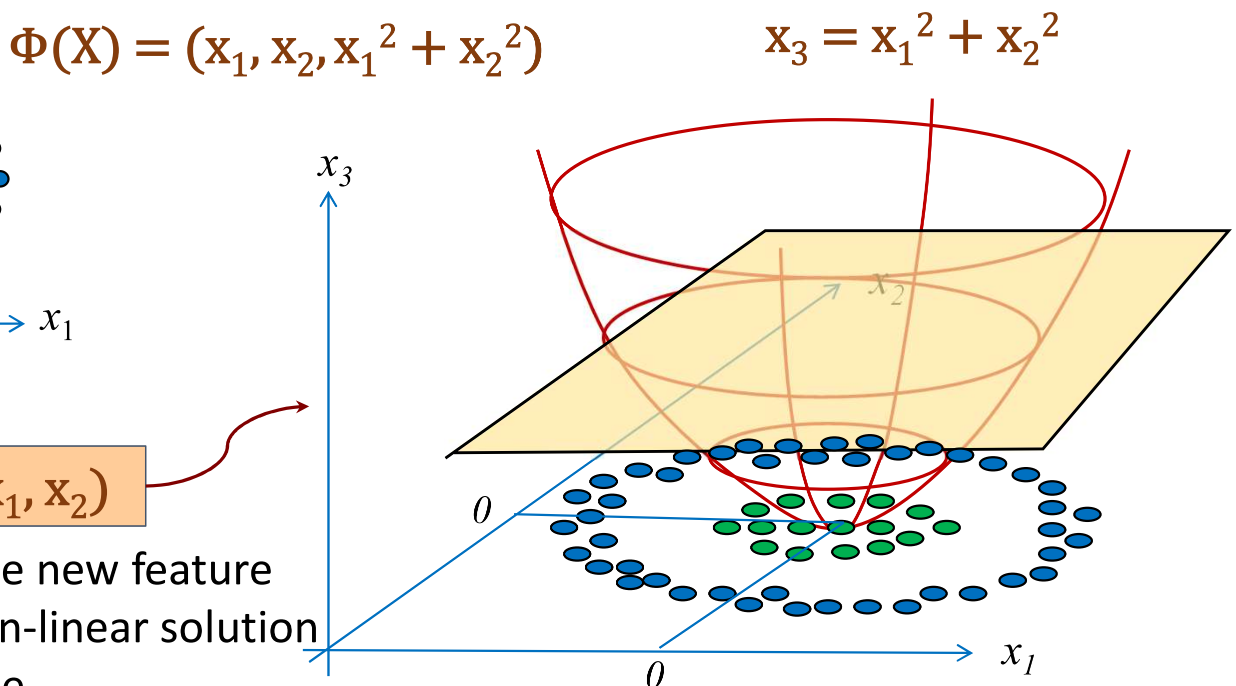
- Apply a non-linear transformation, to the feature space such that the samples become linearly separable



$$\boldsymbol{\Phi}_k = (\boldsymbol{x}_k, \boldsymbol{x}_k^2)$$

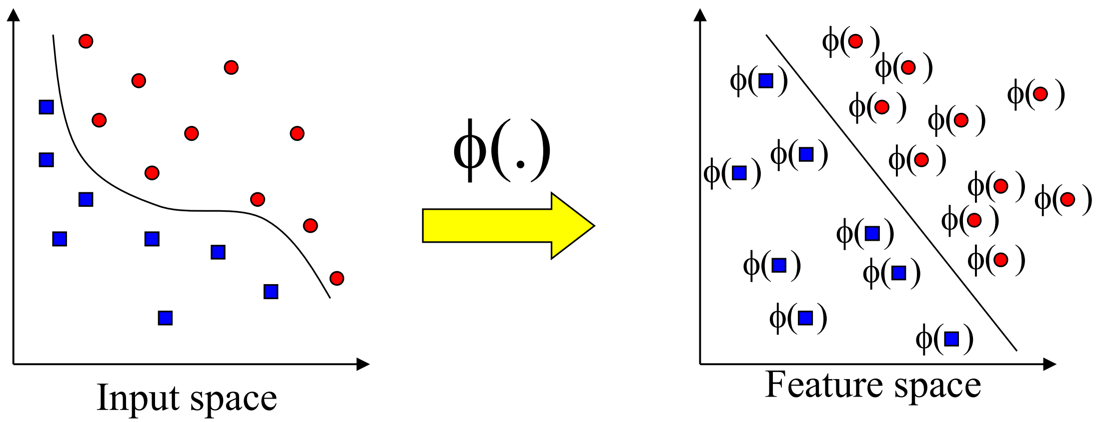
Circular Boundary





- A linear SVM solution in the new feature space is equivalent to a non-linear solution in the original feature space.
- The mapping, $\Phi(\mathbf{X})$, is however unknown. Depends on the distribution of points in the feature space. It can be a complex mapping in general

Transforming the Data



Note: feature space is of higher dimension than the input space in practice

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- Question: Can we find the SVM solution without knowing $\Phi(\mathbf{X})$?
- The kernel trick comes to rescue

Quadratic Basis Function

$$\mathbf{x} \rightarrow \boldsymbol{\Phi}(\mathbf{x}) \rightarrow \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

- Let there be a mapping from $\mathbf{x} \rightarrow \boldsymbol{\Phi}(\mathbf{x})$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \boldsymbol{\Phi}(\mathbf{x}) \rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \\ \dots \end{bmatrix}$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Choose $\boldsymbol{\Phi}(\mathbf{x})$ to be pairwise monomial terms,
$\boldsymbol{\Phi}(\mathbf{x}) \in \mathbb{R}^{m^2}$, with $\boldsymbol{x} \in \mathbb{R}^{m}$

$$\boldsymbol{\Phi}(\mathbf{x}) \rightarrow \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix} \boldsymbol{\Phi}(\mathbf{z}) \rightarrow \begin{bmatrix} z_1z_1 \\ z_1z_2 \\ z_1z_3 \\ z_2z_1 \\ z_2z_2 \\ z_2z_3 \\ z_3z_1 \\ z_3z_2 \\ z_3z_3 \end{bmatrix}$$

$$\Phi^T(x)\Phi(z) = [x_1x_1 \ x_1x_2 \ x_1x_3 \ \dots x_3x_3] \begin{bmatrix} z_1z_1 \\ z_1z_2 \\ z_1z_3 \\ z_2z_1 \\ z_2z_2 \\ z_2z_3 \\ z_3z_1 \\ z_3z_2 \\ z_3z_3 \end{bmatrix} = \sum_{i=1}^{3}\sum_{j=1}^{3}(x_ix_j)(z_iz_j)$$

SVM Solution

1. Maximize: $Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j \boxed{\mathbf{x}_i{}^{\mathrm{T}}\mathbf{x}_j},$ $\boldsymbol{\Phi}^{T}(x_i)\boldsymbol{\Phi}(x_j)$

subject to $\mathbf{0} \leq \boldsymbol{\alpha_i} \leq \boldsymbol{C}\ \forall \boldsymbol{i},\quad \sum_{i=1}^{N} \boldsymbol{\alpha_i y_i} = \mathbf{0}$

- $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{\Phi}(x_i)$

Note: Classification:

$$\mathbf{f}(\mathbf{x}_t) = \mathbf{w}^{\mathrm{T}}\mathbf{x}_t + \mathbf{b} = \sum_{i=1}^{N} \alpha_i y_i \boxed{\mathbf{x}_i{}^{\mathrm{T}}\mathbf{x}_t} + b$$

$\boldsymbol{\Phi}^{T}(x_i)\boldsymbol{\Phi}(x_j)$

- We need to do $\frac{N(N+1)}{2} \approx \frac{N^2}{2}$ dot products to calculate $\boldsymbol{\Phi}^{T}(x_i)\boldsymbol{\Phi}(x_j)$
- Each dot product further requires m^2 calculations
- The whole calculation will cost $\frac{N^2 m^2}{2}$

Quadratic Basis Function

Choose $\mathbf{\Phi(x)}$ to be pairwise monomial terms, $\mathbf{\Phi(x)} \in \mathbb{R}^{m^2}$, with $\boldsymbol{x} \in \mathbb{R}^{m}$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{\Phi(x)} \rightarrow \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix} \quad \mathbf{\Phi(z)} \rightarrow \begin{bmatrix} z_1z_1 \\ z_1z_2 \\ z_1z_3 \\ z_2z_1 \\ z_2z_2 \\ z_2z_3 \\ z_3z_1 \\ z_3z_2 \\ z_3z_3 \end{bmatrix}$$

$$\Phi^T(x)\Phi(z) = [x_1x_1 \; x_1x_2 \; x_1x_3 \; \dots x_3x_3] \begin{bmatrix} z_1z_1 \\ z_1z_2 \\ z_1z_3 \\ z_2z_1 \\ z_2z_2 \\ z_2z_3 \\ z_3z_1 \\ z_3z_2 \\ z_3z_3 \end{bmatrix}$$

$$= \sum_{i=1}^{3}\sum_{j=1}^{3}(x_ix_j)(z_iz_j)$$

$\mathrm{O(m^2)}$

- Just out of casual, innocent, interest, let's look at another function of $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$(\boldsymbol{x}^T\boldsymbol{z})^2 = \left(\sum_{i=1}^{3} x_iz_i\right)\left(\sum_{j=1}^{3} x_jz_j\right) = \sum_{i=1}^{3}\sum_{j=1}^{3} x_iz_ix_jz_j = \sum_{i=1}^{3}\sum_{j=1}^{3}(x_ix_j)(z_iz_j)$$

Both are same, but $(\boldsymbol{x}^T\boldsymbol{z})^2$ is only $\mathrm{O(m)}$ to compute

Quadratic Form

$\mathbf{x} \to \mathbf{\Phi}(\mathbf{x})$

$m = 3$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \to \mathbf{\Phi}(\mathbf{x}) \to \begin{bmatrix} \mathbf{1} \\ \sqrt{\mathbf{2}}\boldsymbol{x}_1 \\ \sqrt{\mathbf{2}}\boldsymbol{x}_2 \\ \sqrt{\mathbf{2}}\boldsymbol{x}_3 \\ \boldsymbol{x}_1^2 \\ \boldsymbol{x}_2^2 \\ \boldsymbol{x}_3^2 \\ \sqrt{\mathbf{2}}\boldsymbol{x}_1\boldsymbol{x}_2 \\ \sqrt{\mathbf{2}}\boldsymbol{x}_1\boldsymbol{x}_3 \\ \sqrt{\mathbf{2}}\boldsymbol{x}_2\boldsymbol{x}_3 \end{bmatrix} = O(10) = O(2\mathrm{m} + \frac{m(m-1)}{2} + 1) \approx O\left(\frac{m^2}{2}\right)$$

$$\mathbf{\Phi}(\mathbf{x}).\,\mathbf{\Phi}(\mathbf{z}) = \mathbf{1} + \mathbf{2}\sum_{i=1}^{m} \boldsymbol{x}_i\boldsymbol{z}_i + \sum_{i=1}^{m} \boldsymbol{x}_i^2\boldsymbol{z}_i^2 + \sum_{i=1}^{m}\sum_{j=i+1}^{m} \mathbf{2}\boldsymbol{x}_i\boldsymbol{x}_j\boldsymbol{z}_i\boldsymbol{z}_j$$

$$(x.z+1)^2 = (x.z)^2 + 2\,x.z + 1 = \left(\sum_{i=1}^{m} x_i z_i\right)^2 + 2\sum_{i=1}^{m} x_i z_i + 1 = \sum_{i=1}^{m}\sum_{j=1}^{m} x_i z_i x_j z_j + 2\sum_{i=1}^{m} x_i z_i + 1$$

$$= \mathbf{1} + \mathbf{2}\sum_{i=1}^{m} \boldsymbol{x}_i\boldsymbol{z}_i + \sum_{i=1}^{m} \boldsymbol{x}_i^2\boldsymbol{z}_i^2 + \sum_{i=1}^{m}\sum_{j=i+1}^{m} \mathbf{2}\boldsymbol{x}_i\boldsymbol{x}_j\boldsymbol{z}_i\boldsymbol{z}_j \qquad \boldsymbol{O}(\boldsymbol{m})$$

For a Cubic Kernel

- If the original Space is 3-dimensional:

$$\mathbf{K}(\mathbf{X}, \boldsymbol{Z}) = (\mathbf{X} \cdot \boldsymbol{Z})^3 = (x_1 z_1 + x_2 z_2 + x_3 z_3)^3$$

- Equivalent to working in a 10-dimensional space
- Kernel: 5(3+2):× and 2:+
- $\Phi(X) = 13{:}\times,\ \ \Phi(Z) = 13{:}\times,$
- $\Phi(X)\Phi(Z) = 10{:}\times, 9{:}\ +$
- Total 36(13+13+10):× and 9:+

$$\Phi(\mathbf{X}) = \Phi\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} x_1^3 \\ x_2^3 \\ x_3^3 \\ x_1^2 x_2 \\ x_1 x_2^2 \\ x_1^2 x_3 \\ x_1 x_3^2 \\ x_2^2 x_3 \\ x_2 x_3^2 \\ x_1 x_2 x_3 \end{bmatrix}$$

Higher Order Polynomials

Kernel Function

Polynomial	$\mathbf{\Phi(x)}$	Cost to build Q_{kl} matrix traditionally	Cost if 100 features	$\mathbf{\Phi(x).\,\Phi(z)}$ $K(x,z)$	Cost to build Q_{kl} matrix sneakily	Cost if 100 features
Quadratic	All $^{m^2}/_2$ terms up to degree 2	$m^2\, N^2 /4$	$2{,}500\, N^2$	$(\mathbf{x}.\mathbf{z}+1)^2$	$m\, N^2 / 2$	$50\, N^2$
Cubic	All $m^3/6$ terms up to degree 3	$m^3\, N^2 /12$	$83{,}000\, N^2$	$(\mathbf{x}.\mathbf{z}+1)^3$	$m\, N^2 / 2$	$50\, N^2$
Quartic	All $m^4/24$ terms up to degree 4	$m^4\, N^2 /48$	$1{,}960{,}000\, N^2$	$(\mathbf{x}.\mathbf{z}+1)^4$	$m\, N^2 / 2$	$50\, N^2$

Slide inspired from, Andrew W. Moore

SVM Solution: Kernel function

1. Maximize:

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

2. $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i;$

3. $\mathbf{b} = \mathbf{1} - \mathbf{w}^{\mathrm{T}} \mathbf{x}_{s+} = \mathbf{1} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{K}(\mathbf{x}_i, \mathbf{x}_{s+})$

Do we know the $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)$.?

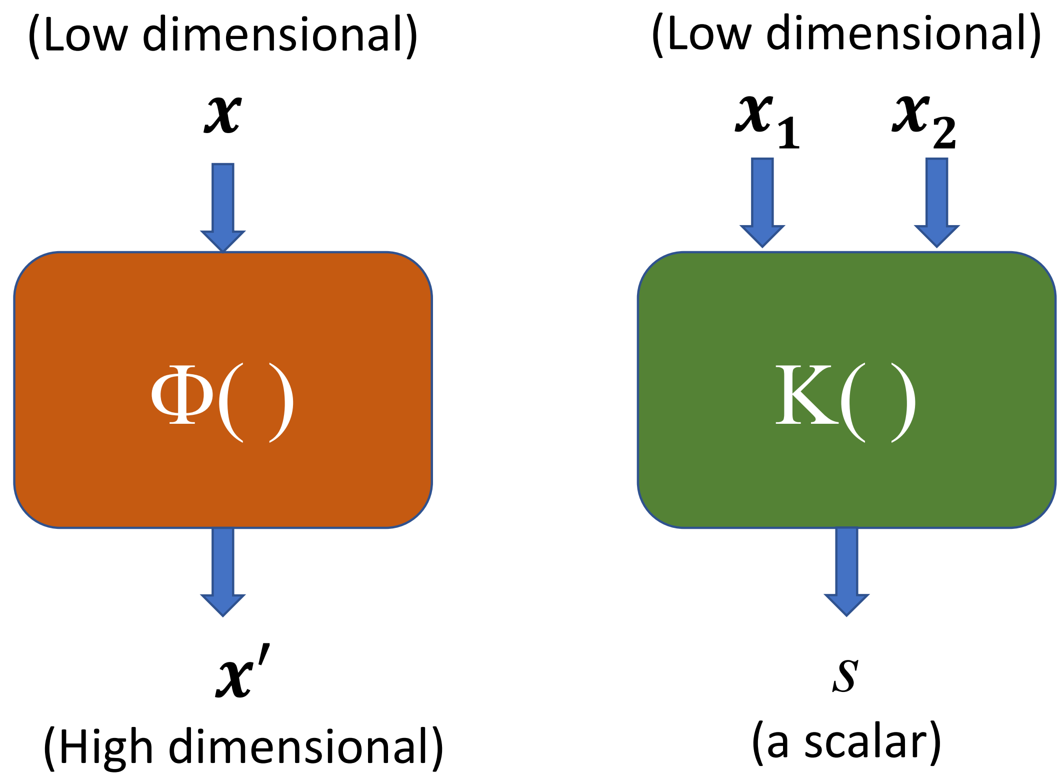
Let us compare Φ and $\mathbf{K}()$.

Note: Classification:

$$\mathbf{g}(\mathbf{x}_t) = \mathbf{w}^{\mathrm{T}} \mathbf{x}_t + \mathbf{b} = \sum_{i=1}^{N} \alpha_i y_i \ \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) + b$$

Comparing compare $\Phi()$ and $\mathbf{K}()$

- $\Phi(\mathbf{x})$ is a complex non-linear mapping of $\mathbf{x}$ into a high-dimensional space.
- $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j)$ is a simple function that measures the similarity between two vectors.
- If we know the K function, we do not need Φ.
- A kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\boldsymbol{\phi}(\mathbf{x})$ explicitly).



What Functions are Kernels?

- For some functions $\mathbf{K(x_i,x_j)}$ checking that $\mathbf{K(x_i, xj)} = \mathbf{\Phi^T(x_i)\Phi(x_j)}$ can be cumbersome.
- Mercer's theorem determines which functions can be used as a kernel function:

Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

K =

$K(x_1,x_1)$	$K(x_1,x_2)$	$K(x_1,x_3)$	…	$K(x_1,x_n)$
$K(x_2,x_1)$	$K(x_2,x_2)$	$K(x_2,x_3)$		$K(x_2,x_n)$
…	…	…	…	…
$K(x_n,x_1)$	$K(x_n,x_2)$	$K(x_n,x_3)$	…	$K(x_n,x_n)$

✓ $K(x_i, x_j)$ measures the similarity or proximity between two data points x_i and x_j in the input space

Examples of Kernel Functions

- Linear: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x}_j$
  - Mapping Φ: $\boldsymbol{x} \rightarrow \boldsymbol{\Phi}(\boldsymbol{x})$, where $\boldsymbol{\Phi}(\boldsymbol{x})$ is $\boldsymbol{x}$ itself
- Polynomial of power p: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (1 + \boldsymbol{x}_i^{\boldsymbol{T}} \boldsymbol{x}_j)^p$
  - Mapping Φ: $\boldsymbol{x} \rightarrow \boldsymbol{\Phi}(\boldsymbol{x})$, where $\boldsymbol{\Phi}(\boldsymbol{x})$ has ${}^{\mathbf{d+p}}_{\mathbf{p}}\mathbf{C}$ dimensions
- Gaussian (radial-basis function): $\mathrm{K}(\mathbf{x}_{\mathrm{i}}, \mathbf{x}_{\mathrm{j}}) = \mathrm{e}^{-\frac{(\|\mathrm{x_i - x_j}\|)^2}{2\sigma^2}}$
  - Mapping Φ: $\boldsymbol{x} \rightarrow \boldsymbol{\Phi}(\boldsymbol{x})$, where $\boldsymbol{\Phi}(\boldsymbol{x})$ is *infinite-dimensional*: every point is mapped to *a function* (a Gaussian); combination of functions for support vectors is the separator.

Example

- Suppose we have 5 one-dimensional data points
  - $\boldsymbol{x_1 = 1, x_2 = 2, x_3 = 4, x_4 = 5, x_5 = 6}$, with 1, 2, 6 as class 1 and 4, 5 as class 2 $\Rightarrow \boldsymbol{y_1 = 1, y_2 = 1, y_3 = -1, y_4 = -1, y_5 = 1}$
- We use the polynomial kernel of degree 2
  - $\boldsymbol{K(x, y) \;=\; (xy + 1)^2}$
  - C is set to 100
- We first find $\boldsymbol{\alpha_i}$ $(\boldsymbol{i = 1, \ldots, 5})$ by

$$\mathbf{max.} \sum_{i=1}^{5} \alpha_i - \frac{1}{2} \sum_{i=1}^{5} \sum_{j=1}^{5} \alpha_i \alpha_j y_i y_j \, (\mathbf{x}_i \mathbf{x}_j + 1)^2$$

$$\textbf{subject to } 0 \le \alpha_i \le 100, \sum_{i=1}^{5} \alpha_i y_i = 0$$

Slide content borrowed from Andrew Moore

Example

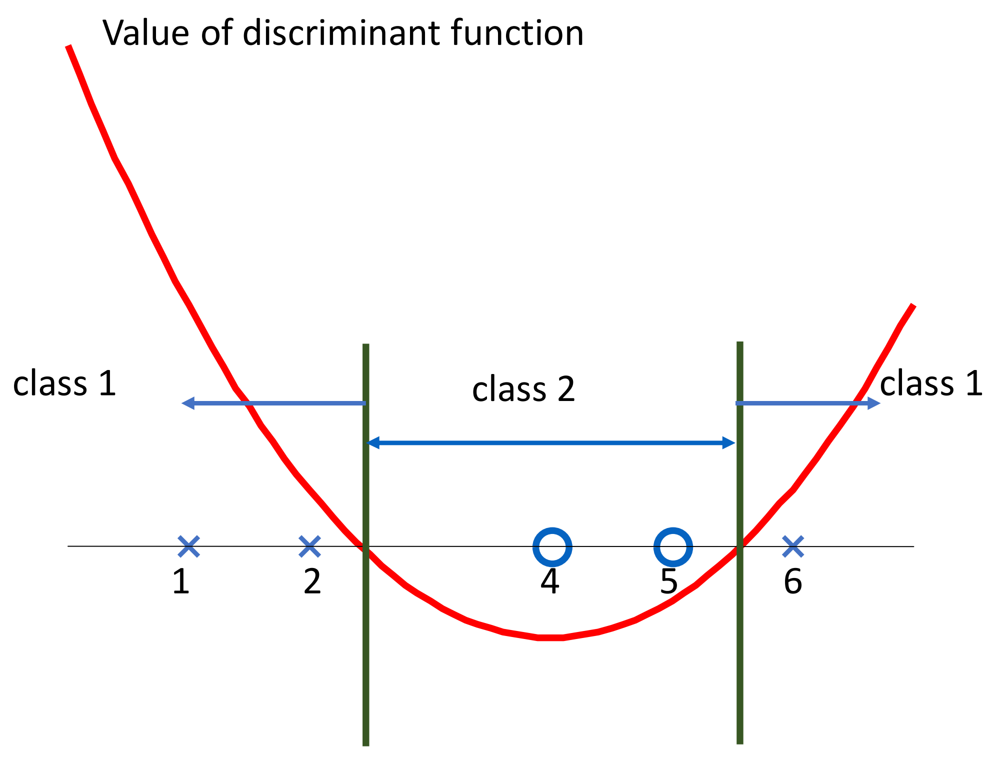
- By using a QP solver, we get
  - $\boldsymbol{\alpha_1 = 0, \alpha_2 = 2.5, \alpha_3 = 0, \alpha_4 = 7.333, \alpha_5 = 4.833}$
  - Note that the constraints are indeed satisfied
  - The support vectors are $\{\boldsymbol{x_2 = 2, x_4 = 5, x_5 = 6}\}$
- The discriminant function is

$$\mathbf{f}(\boldsymbol{z}) = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i{}^{\mathrm{T}} z + b = \mathbf{2.5}\ \underbrace{(\mathbf{1})(\mathbf{2}\boldsymbol{z} + \mathbf{1})^{\mathbf{2}}}_{} + \mathbf{7.333}\ (-\mathbf{1})(\mathbf{5}\boldsymbol{z} + \mathbf{1})^{\mathbf{2}} + \overset{\alpha_5}{\mathbf{4.833}}\ \overset{y_5}{(\mathbf{1})}\overset{K(z, x_5)}{(\mathbf{6}\boldsymbol{z} + \mathbf{1})^{\mathbf{2}}} + \boldsymbol{b}$$

$$= \mathbf{0.6667}\ \boldsymbol{z}^{\mathbf{2}} - \mathbf{5.333}\ \boldsymbol{z} + \boldsymbol{b}$$

- *b* is recovered by solving f(2)=1 or by f(5)=-1 or by f(6)=1, as x_2 and x_5 lie on the line $\boldsymbol{w^T x + b = 1}$ and x_4 lies on the line $\boldsymbol{w^T x + b = -1}$
- All three give b=9 $\Longrightarrow \boldsymbol{f}(\boldsymbol{z}) = \mathbf{0.6667}\ \boldsymbol{z}^{\mathbf{2}} - \mathbf{5.333}\ \boldsymbol{z} + \mathbf{9}$

Example

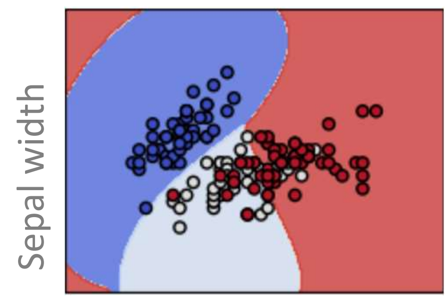

Value of discriminant function
class 1
class 2
class 1
1
2
4
5
6

Scikit Learn Implementation

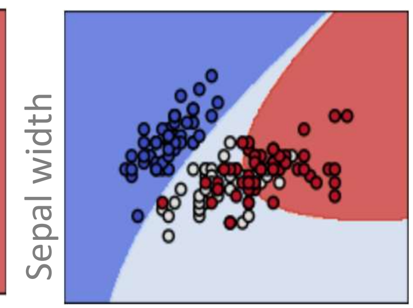
- libsvm based: sklearn.svm.SVC
- Several Kernels: linear, polynomial, rbf, sigmoid, custom.
- Other HyperParameters: C, kernel params
- Usage:

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC(kernel='rbf')
>>> clf.fit(X, y)
>>> clf.predict([[2., 2.]])
```

SVC: RBF Kernel



SVC: Poly (3) Kernel




```
>>> # get support vectors
>>> clf.support_vectors_
>>> # get support vector indices
>>> clf.support_
```

https://scikit-learn.org/stable/modules/svm.html

Summary

- Linear SVMs generalize well, but cannot separate non-linear data
- If features can be transformed appropriately, SVMs can learn non-linear boundaries.
- How do we find the feature transformation?
  - Use some popular Kernel functions.
- Kernels (nonlinear) SVMs are also good at generalization and can deal with non-linear data.