Support Vector Machines (Formal : Version 1)

Linear Classifier

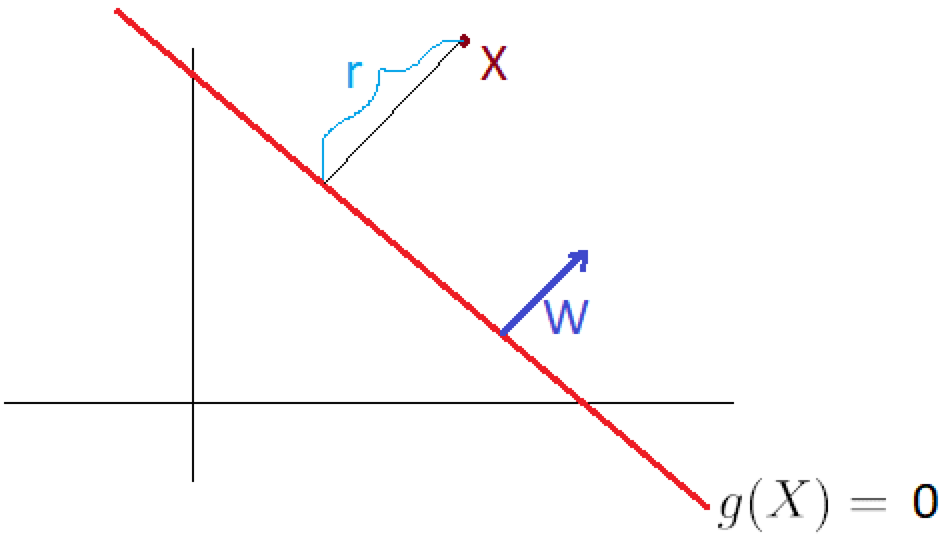
- Consider a two class problem, $\Omega = \{\omega_1, \omega_2\}$
  A pattern $X = (x_1, \ldots, x_d)^t$
- The discriminant function
  $g(X) = w_1 x_1 + \cdots + w_d x_d + w_0 \; = \; W^t X + w_0$
- $g(X) = 0$ defines a hyperplane in the feature space.
- Classification rule:

$g(X) > 0 \quad \Rightarrow \quad$ class is ω_1

$g(X) < 0 \quad \Rightarrow \quad$ class is ω_2

$g(X) = 0 \quad \Rightarrow$ class is decided arbitrarily

$$g(X) = w_1x_1 + \cdots + w_dx_d + w_0 \;=\; W^tX + w_0$$



- W is perpendicular to *g(X) = 0* hyper-plane.
- r, distance of arbitrary X from the hyper-plane is: $r = \frac{g(X)}{\|W\|}$

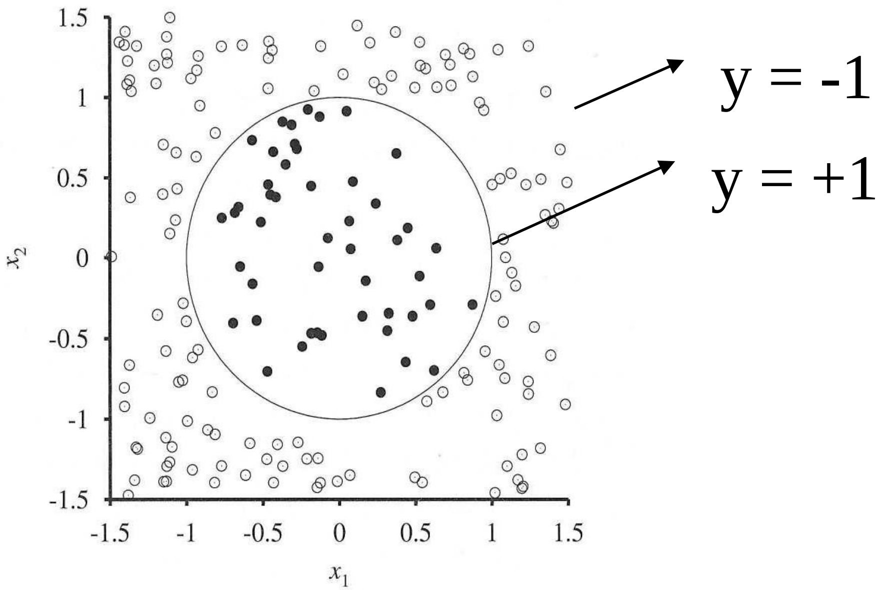
Non-linear vs linear

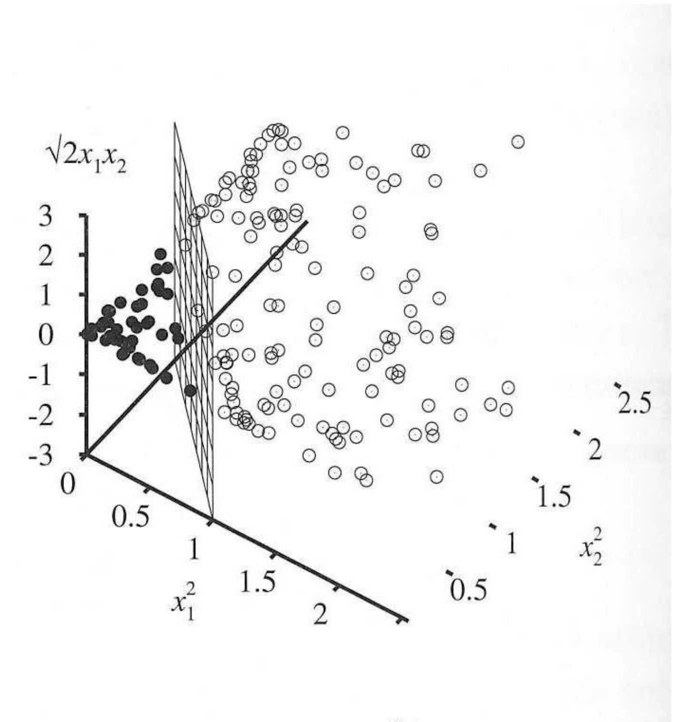
- For every function in the input space, there is an equivalent linear function in a feature space.
- Feature space is a high-dimensional one.
- A mathematically valid way to work (do the optimization) in the input space but to get the solution (linear function) in the feature space is by using the *kernel trick*.

An example

Input Space

Feature Space





The Kernel Trick !

- There is no need to do this mapping explicitly.
- For some mappings, the dot product in the feature space can be expressed as a function in the Input space.
- $\varphi(X_1) \cdot \varphi(X_2) = k(X_1, X_2)$

Eg: Consider a two dimensional problem with $X = (x_1, x_2)^t$. Let $\phi(X) = (x_1^2, x_2^2, \sqrt{2}\, x_1 x_2)^t$. Then $\phi(X_i) \cdot \phi(X_j) = \kappa(X_i, X_j) = (X_i \cdot X_j)^2$.

Support Vector Machines

- Decision surface is a hyperplane (line in 2D) in **feature** space (similar to the Perceptron)
- Arguably, one of the most important discovery in machine learning
- In a nutshell:
  - map the data to a predetermined very high-dimensional space via a kernel function
  - Find the hyperplane that maximizes the margin between the two classes
  - If data are not separable find the hyperplane that maximizes the margin and minimizes the (penalty associated with) misclassifications

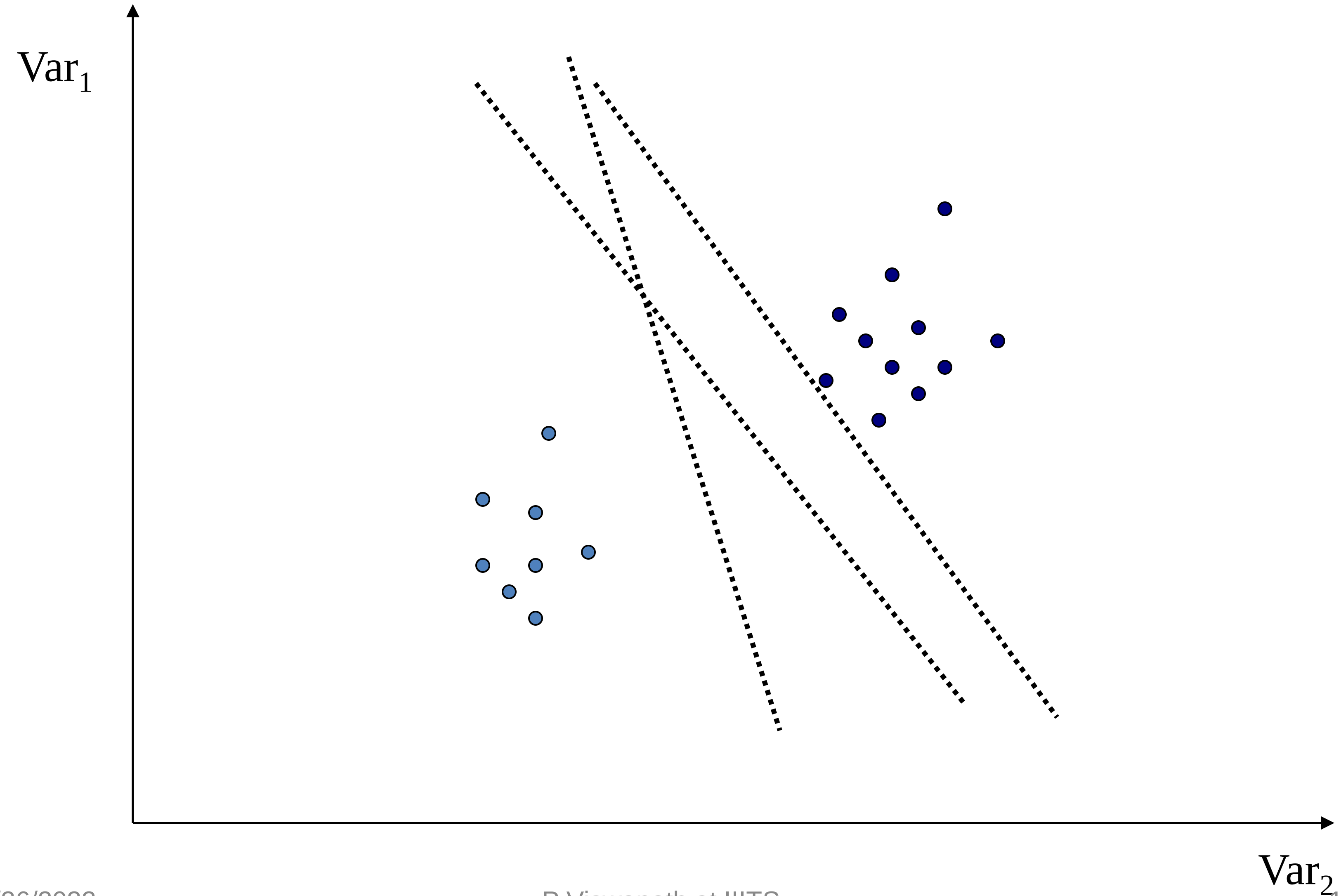
Support Vector Machines

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
  2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

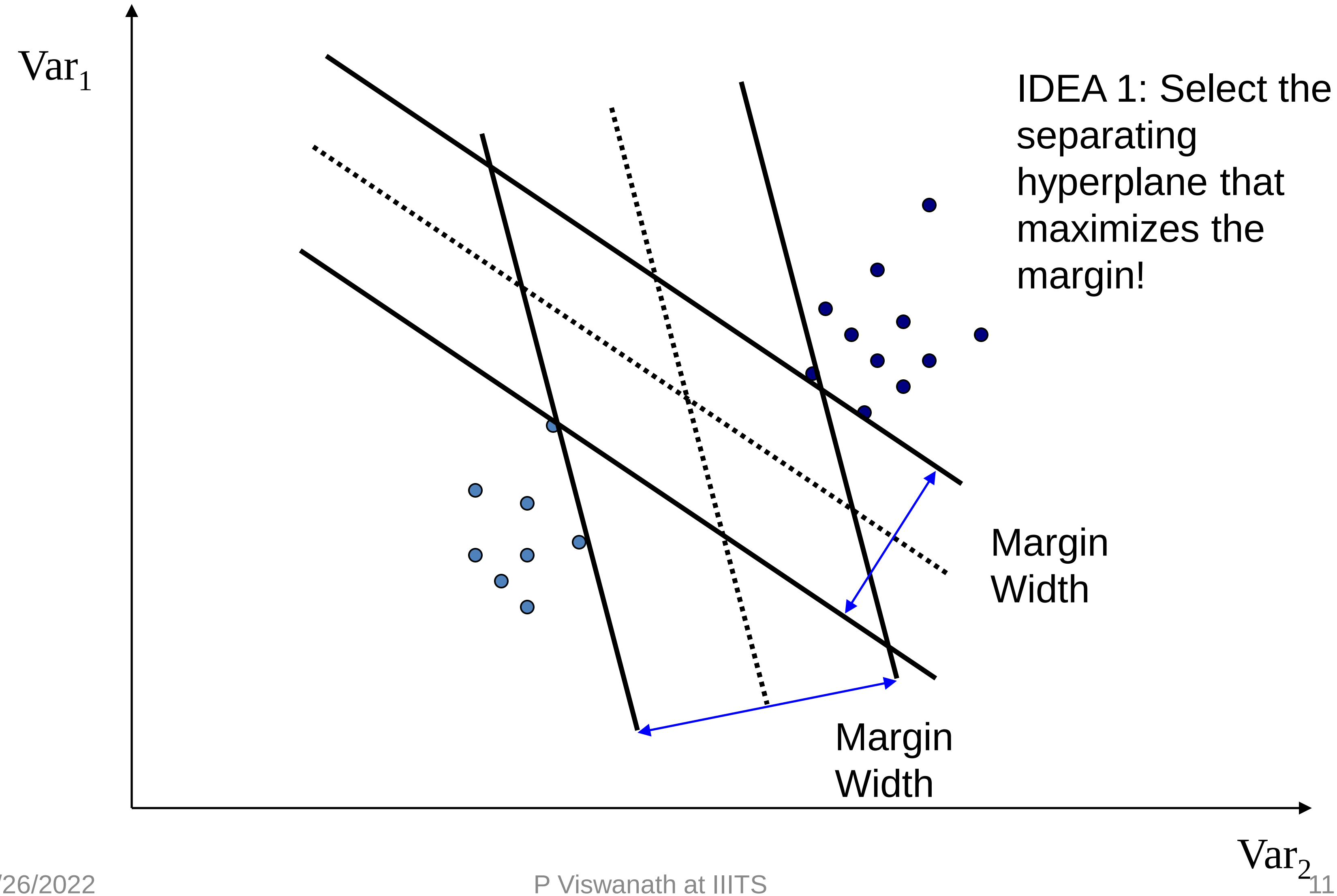
Support Vector Machines

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
  2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

Which Separating Hyperplane to Use?

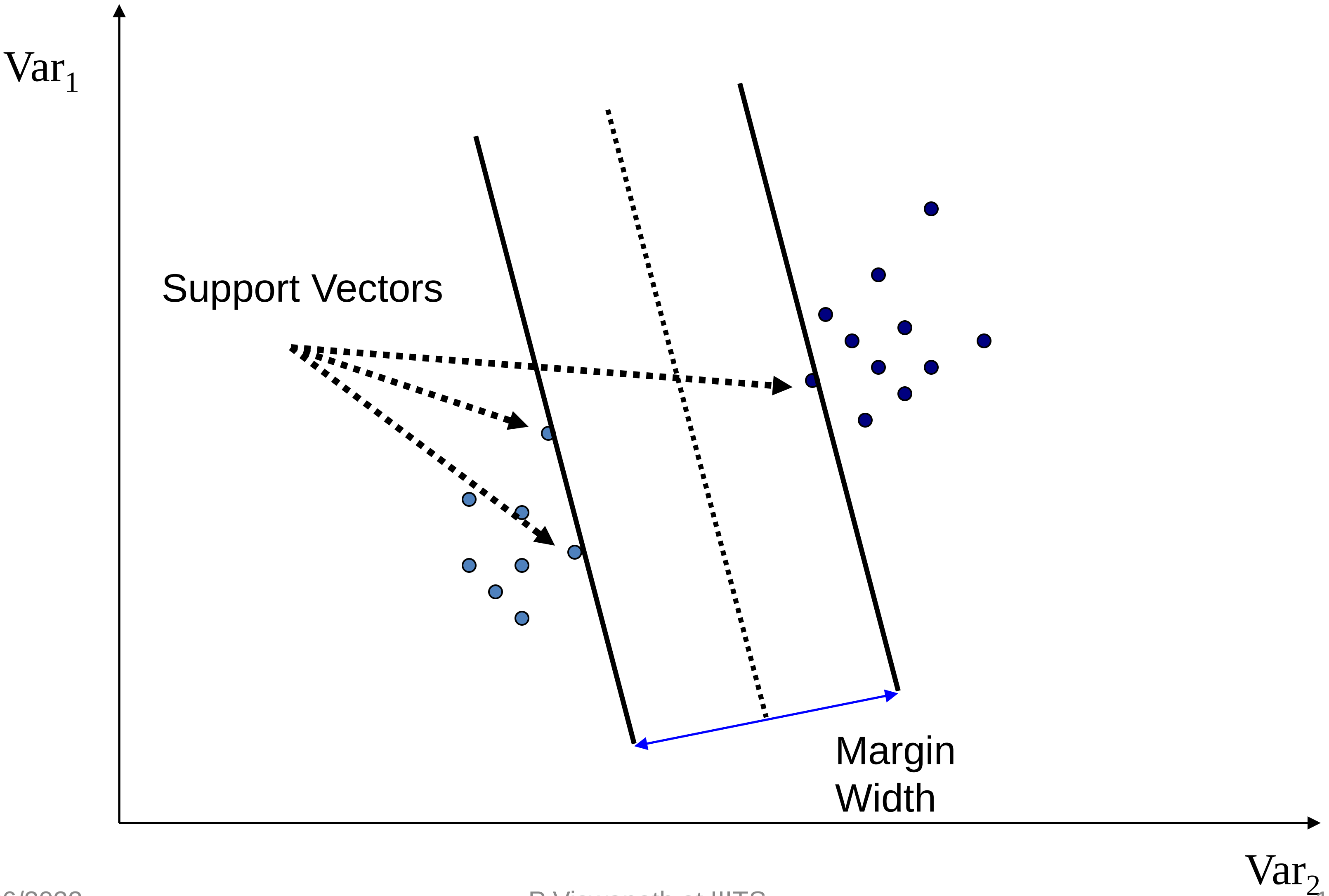


Maximizing the Margin

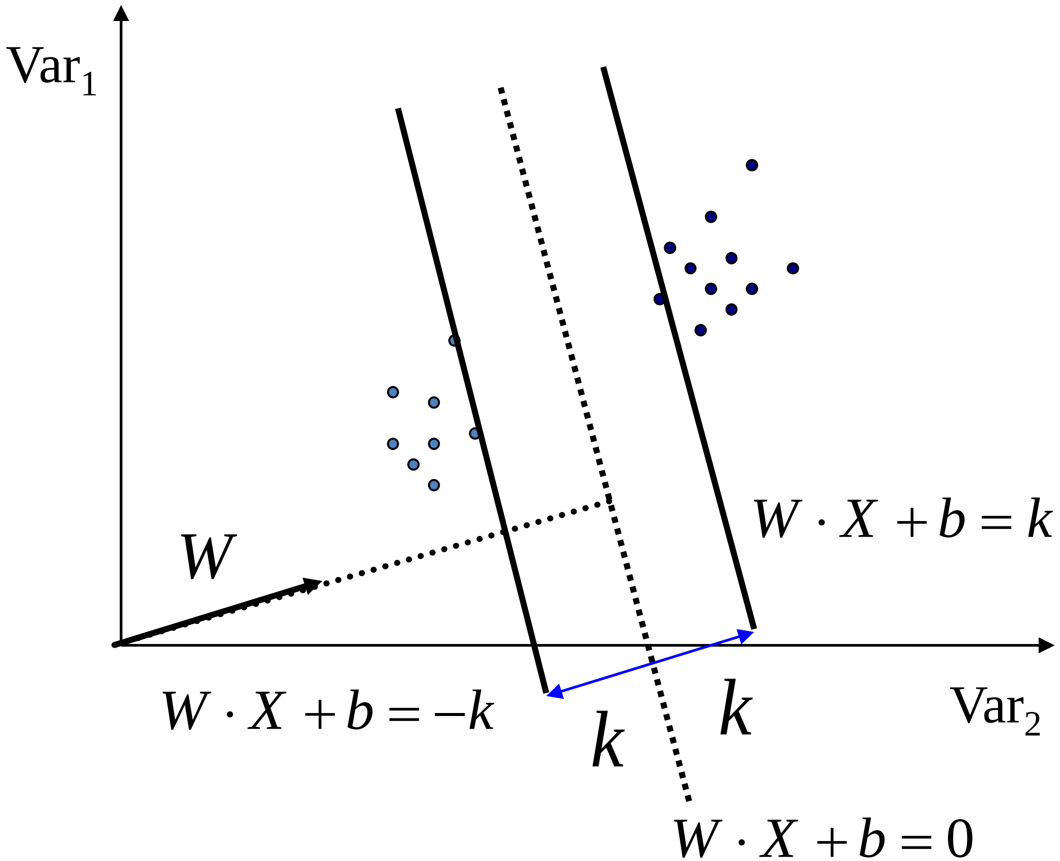


IDEA 1: Select the separating hyperplane that maximizes the margin!

Support Vectors



Setting Up the Optimization Problem



The width of the margin is:

$$\frac{2|k|}{\|W\|}$$

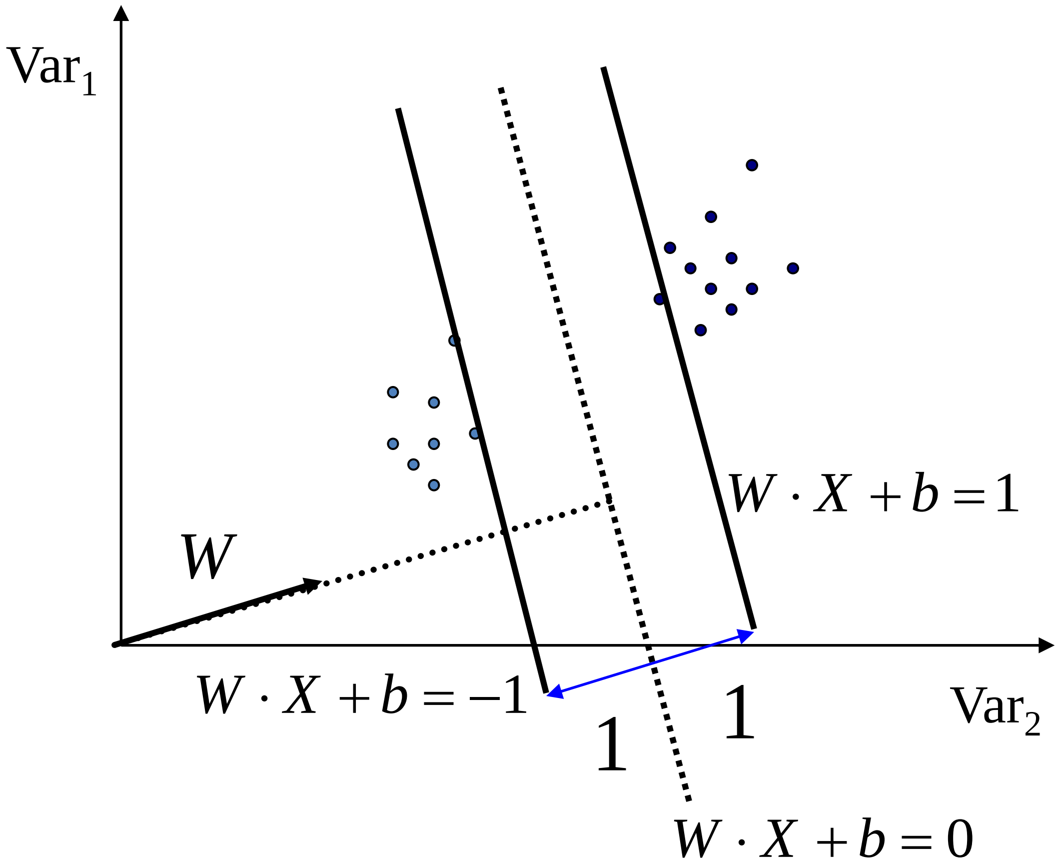
So, the problem is:

Maximize $\frac{2|k|}{\|W\|}$

s.t. $W \cdot X + b \geq k, \forall\ X\ in\ Class\ 1$

$W.X + b \leq -k, \forall\ X\ in\ Class\ 2$

Setting Up the Optimization Problem



There is a scale and unit for data so that *k=1*. Then problem becomes:

So, the problem is:

Maximize $\frac{2}{\|W\|}$

s.t. $W \cdot X + b \geq 1, \forall X \text{ in Class } 1$

$W . X + b \leq -1, \forall X \text{ in Class } 2$

Setting Up the Optimization Problem

- If class 1 corresponds to 1 and class 2 corresponds to -1, we can rewrite

$$W \cdot X_i + b \geq 1, \forall X_i \text{ with } y_i = 1$$
$$W . X_i + b \leq -1, \forall X_i \text{ with } y_i = -1$$

- as

$$\mathrm{y}_\mathrm{i}(W \cdot X_i + b) \geq 1, \forall X_i$$

- So the problem becomes:

Maximize $\frac{2}{\|W\|}$
s.t. $\mathrm{y}_\mathrm{i}(W \cdot X_i + b) \geq 1, \forall X_i$

or

Minimize $\frac{1}{2}\|W\|^2$
s.t. $\mathrm{y}_\mathrm{i}(W \cdot X_i + b) \geq 1, \forall X_i$

Linear, Hard-Margin SVM Formulation

- Find *W, b* that solves

$$\text{Minimize } \frac{1}{2}\|W\|^2$$
$$\text{s.t. } y_i(W \cdot X_i + b) \geq 1, \forall X_i$$

- Problem is convex so, there is a unique global minimum value (when feasible)
- Non-solvable if the data is not linearly separable
- Quadratic Programming
  - Very efficient computationally with modern constraint optimization engines (handles thousands of constraints).

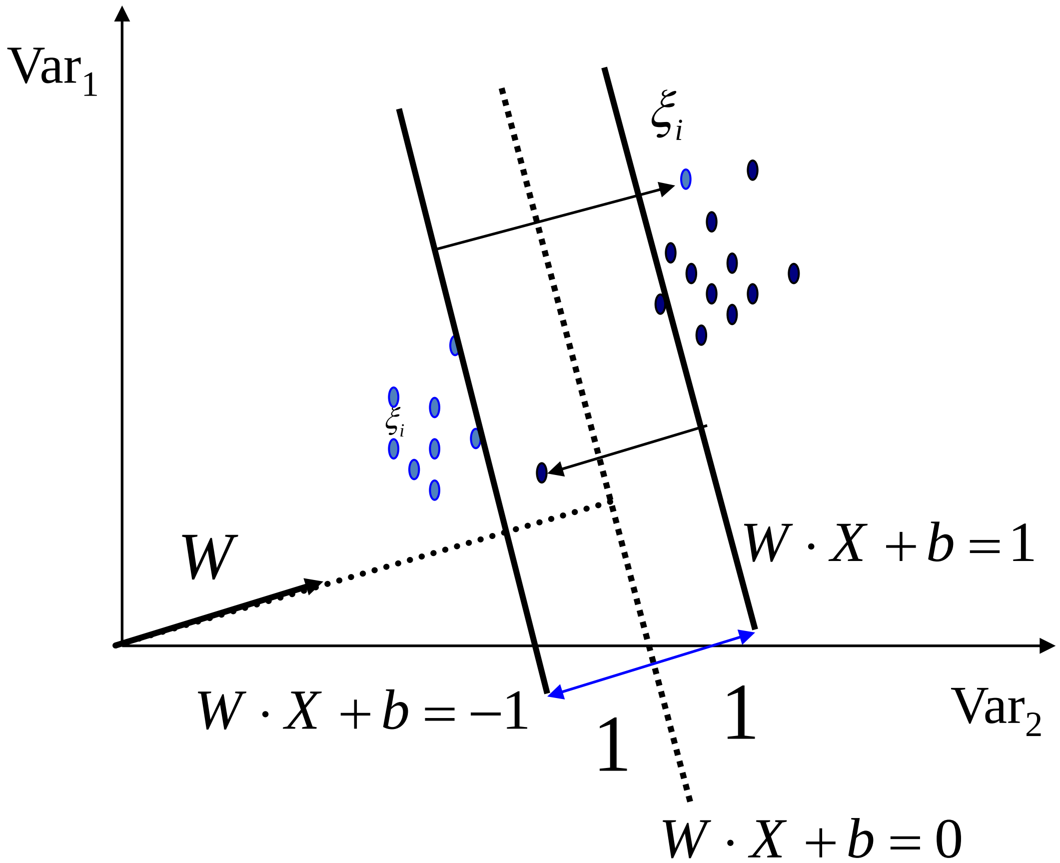
Support Vector Machines

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): <u>maximize margin</u>
  2. Extend the above definition for non-linearly separable problems: <u>have a penalty term for misclassifications</u>
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: <u>reformulate problem so that data is mapped implicitly to this space</u>

Support Vector Machines

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
  2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

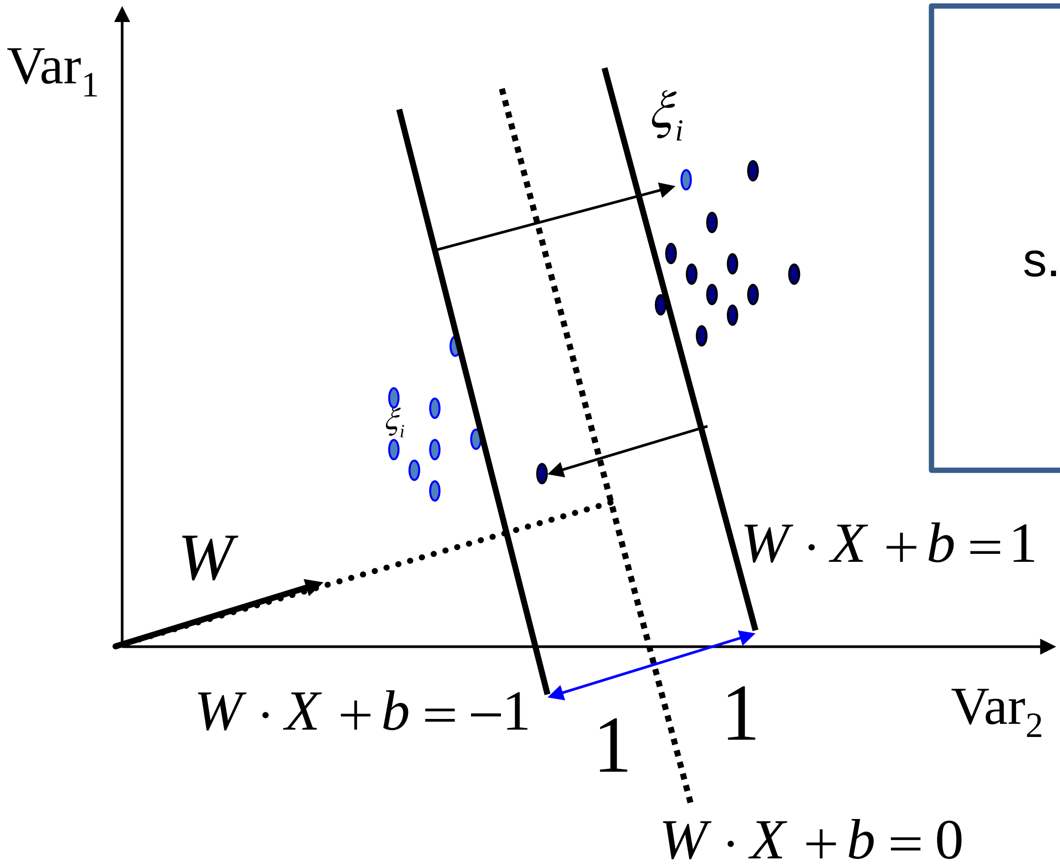
Non-Linearly Separable Data



Introduce slack variables ξ_i

Allow some instances to fall within the margin, but penalize them

Non-Linearly Separable Data



$$\text{Min } \frac{1}{2}\|W\|^2 + \mathrm{C}\sum_i \xi_i$$

$$\text{s.t. } \mathrm{y_i}(W \cdot X_i + b) \geq 1 - \xi_i, \forall X_i$$

$$\text{and} \qquad \xi_i \geq 0$$

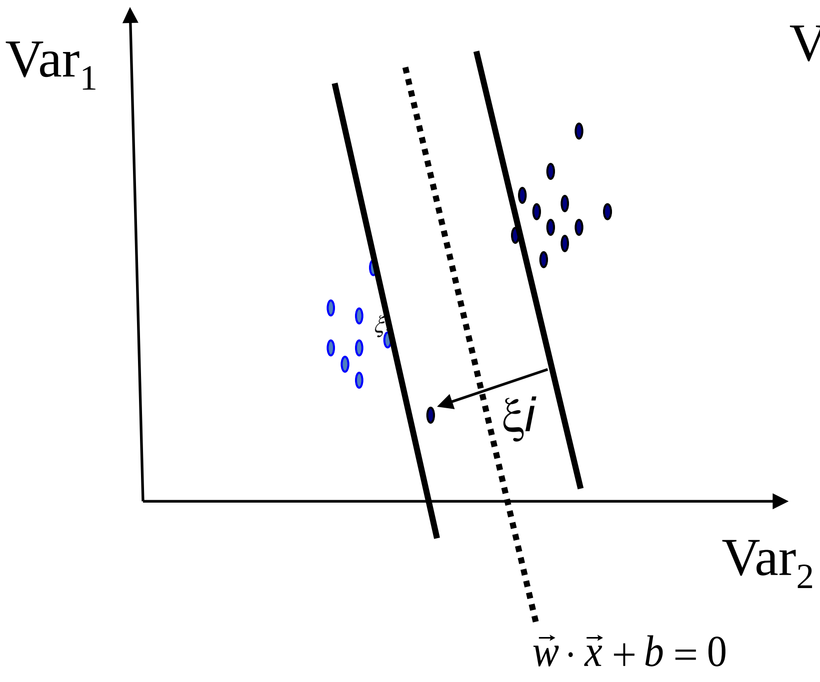
C trades-off margin width and misclassifications

Linear, Soft-Margin SVMs

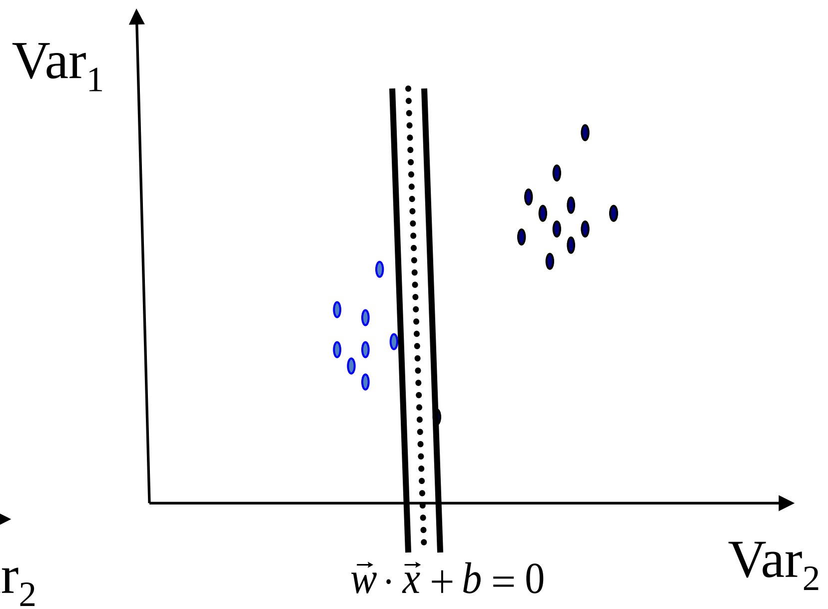
$$\min \frac{1}{2}\|w\|^2 + C\sum_i \xi_i \qquad\qquad y_i(w\cdot x_i + b) \geq 1 - \xi_i,\ \forall x_i$$
$$\xi_i \geq 0$$

- Algorithm tries to maintain ξ_i to zero while maximizing margin
- Notice: algorithm does not minimize the *number* of misclassifications (NP-complete problem) but the sum of distances from the margin hyperplanes
- Other formulations use ξ_i^2 instead
- As $C \to \infty$, we get closer to the hard-margin solution

Robustness of Soft vs Hard Margin SVMs



Soft Margin SVM



Hard Margin SVM

Soft vs Hard Margin SVM

- Soft-Margin always have a solution
- Soft-Margin is more robust to outliers
  – Smoother surfaces (in the non-linear case)
- Hard-Margin does not require to guess the cost parameter (requires no parameters at all)

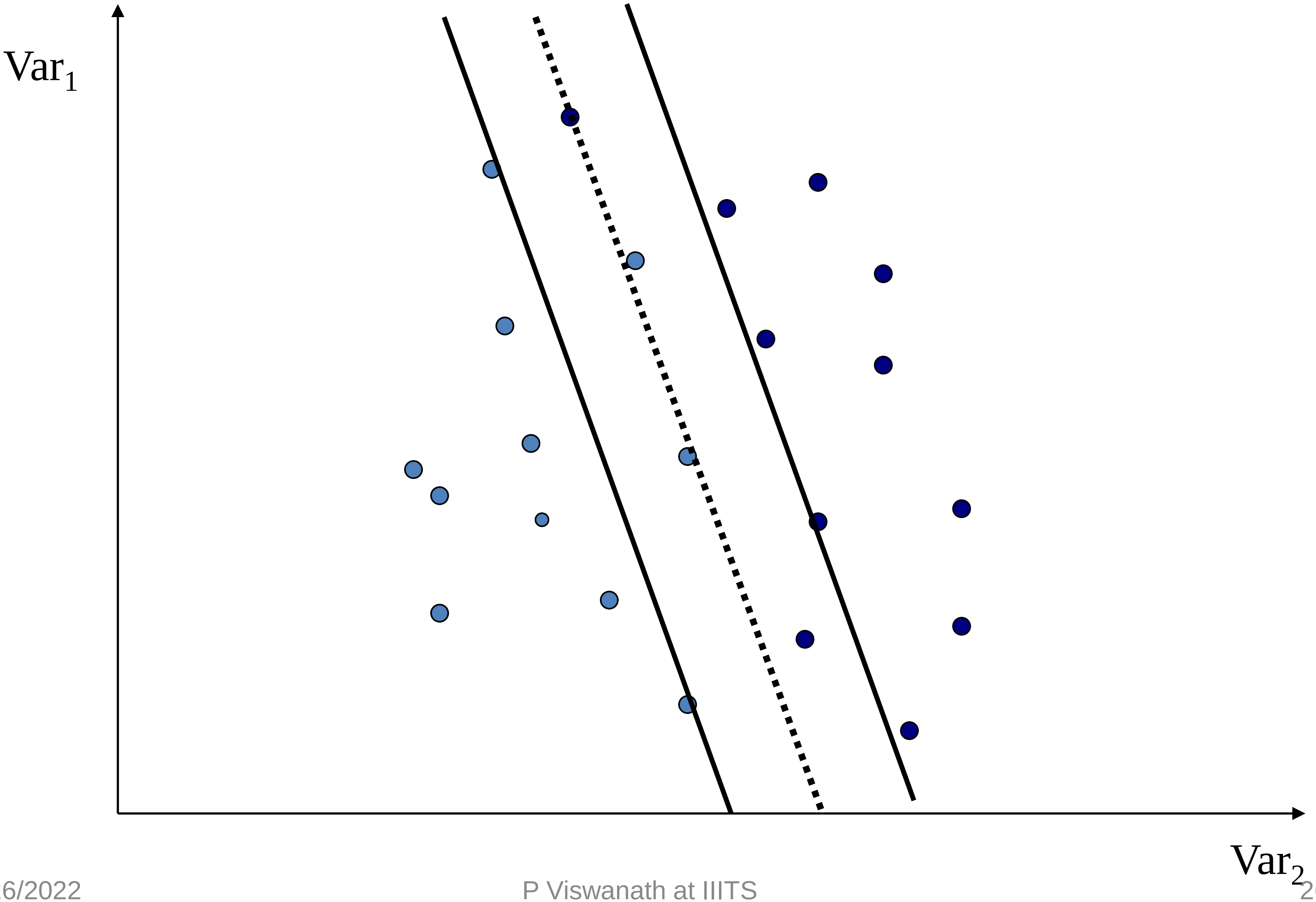
Support Vector Machines

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
  2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

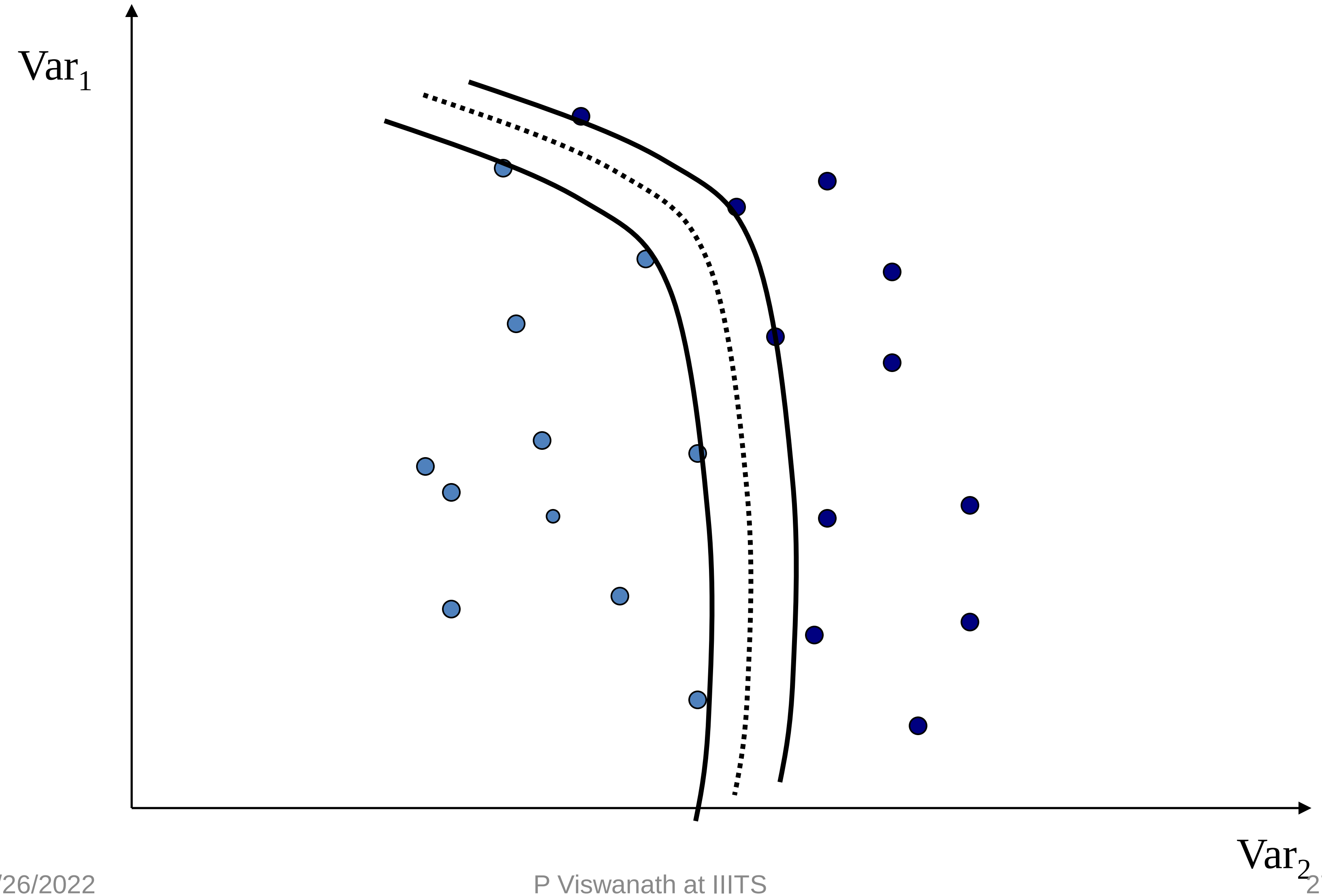
Support Vector Machines

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): <u>maximize margin</u>
  2. Extend the above definition for non-linearly separable problems: <u>have a penalty term for misclassifications</u>
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: <u>reformulate problem so that data is mapped implicitly to this space</u>

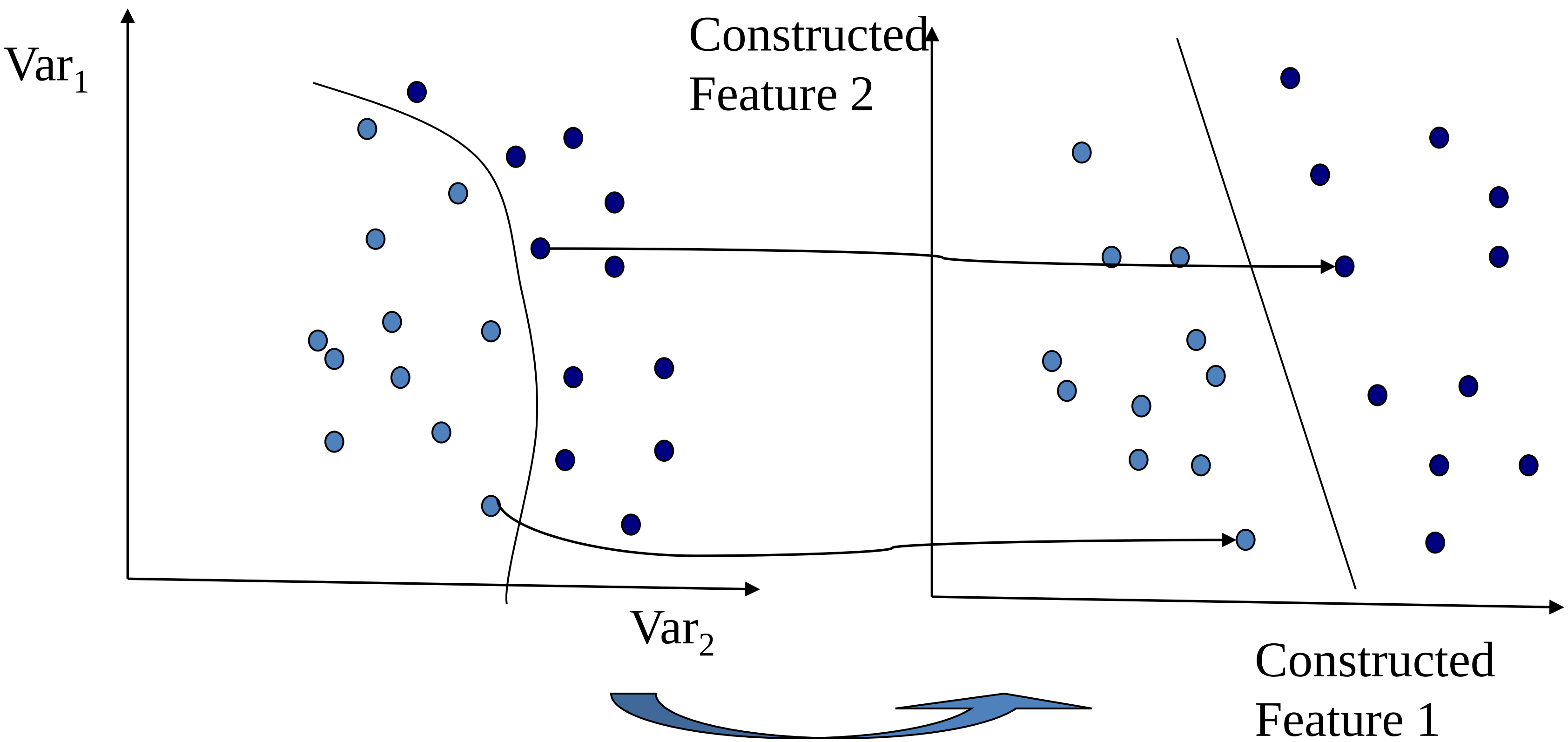
Disadvantages of Linear Decision Surfaces



Advantages of Non-Linear Surfaces



Linear Classifiers in High-Dimensional Spaces



Find function Φ(x) to map to a different space

Mapping Data to a High-Dimensional Space

- Find function Φ(x) to map to a different space, then SVM formulation becomes:
- $\min \frac{1}{2} ||W||^2 + C \sum_i \xi_i$ $\quad$ $s.t.\ y_i(W \cdot \Phi(X) + b) \geq 1 - \xi_i, \forall X_i$
  $\xi_i \geq 0$
- Data appear as Φ(X), weights *W* are now weights in the new space
- Explicit mapping expensive if Φ(X) is very high dimensional
- Solving the problem without explicitly mapping the data is desirable

The Dual of the SVM Formulation

- Original SVM formulation
  - n inequality constraints
  - *n positivity constraints*
  - n number of ξ variables

$$\min_{W,b} \frac{1}{2}\|W\|^2 + C\sum_i \xi_i$$

$$s.t.\ \ y_i(W\cdot\Phi(X)+b) \geq 1-\xi_i,\ \forall X_i$$

$$\xi_i \geq 0$$

- The (Wolfe) dual of this problem
  - one equality constraint
  - n positivity constraints
  - n number of α variables (Lagrange multipliers)
  - Objective function more complicated

$$\min_{a_i} \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j(\Phi(X_i)\cdot\Phi(X_j)) - \sum_i \alpha_i$$

$$s.t.\ \ \mathrm{C} \geq \alpha_{\mathrm{i}} \geq 0,\ \forall X_i$$

$$\sum_i \alpha_i y_i = 0$$

- NOTICE: Data only appear as $\Phi(X_i)\cdot\Phi(X_j)$

The Kernel Trick

- $\Phi(x_i) \cdot \Phi(x_j)$: means, map data into new space, then take the inner product of the new vectors
- We can find a function such that: $K(X_i, X_j) = \Phi(X_l) \cdot \Phi(X_j)$, i.e., the image of the inner product of the data is the inner product of the images of the data
- Then, we do not need to explicitly map the data into the high-dimensional space to solve the optimization problem (for training)
- How do we classify without explicitly mapping the new instances? Turns out

$$\operatorname{sgn}(W \cdot X + b) = \operatorname{sgn}\Big(\sum_i \alpha_i y_i K(X_i, X) + b\Big)$$

$$\text{where } b \text{ solves } \alpha_j \Big(y_j \sum_i \alpha_i y_i K(X_i, X_j) + b - 1\Big) = 0,$$

$$\text{for any } j \text{ with } \alpha_j \neq 0$$

Examples of Kernels

- Assume we measure two quantities

- Consider the function:

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2, x_1, x_2, 1)$$

- We can verify that:

$$K(X_1, X_2) = (X_1 \cdot X_2 + 1)^2$$

- These type of kernels are called Polynomial kernels.

Polynomial and Gaussian Kernels

$$K(X \cdot Z) = (X \cdot Z + 1)^p$$

- is called the polynomial kernel of degree *p*.
- Another commonly used Kernel is the Gaussian (maps to an infinite dimensional space):

$$K(X \cdot Z) = \exp(-\|X - Z\| / 2\sigma^2)$$

The Mercer Condition

- Is there a mapping $\Phi(x)$ for any given symmetric function $K(x,z)$? No.
- The SVM dual formulation requires calculation $K(x_i , x_j)$ for each pair of training instances. The matrix $G_{ij} = K(x_i , x_j)$ is called the Gram matrix
- There is a feature space $\Phi(x)$ when the Kernel is such that G is always semi-positive definite (Mercer condition)

Support Vector Machines

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
  2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

Other Types of Kernel Methods

- SVMs that perform regression
- SVMs that perform clustering
- ν-Support Vector Machines: maximize margin while bounding the number of margin errors
- Leave One Out Machines: minimize the bound of the leave-one-out error
- SVM formulations that take into consideration difference in cost of misclassification for the different classes
- Kernels suitable for sequences of strings, or other specialized kernels
- Kernel-PCA, Kernel-SVD

Comparison with Neural Networks

Neural Networks

- Hidden Layers map to lower dimensional spaces
- Search space has multiple local minima
- Training is expensive
- Classification extremely efficient
- Requires number of hidden units and layers
- Very good accuracy in typical domains

SVMs

- Kernel maps to a very-high dimensional space
- Search space has a unique minimum
- Training is extremely efficient
- Classification extremely efficient
- Kernel and cost the two parameters to select
- Very good accuracy in typical domains
- Extremely robust

MultiClass SVMs

- One-versus-all
  - Train *n* binary classifiers, one for each class against all other classes.
  - Predicted class is the class of the most confident classifier
- One-versus-one
  - Train *n(n-1)/2* classifiers, each discriminating between a pair of classes
  - Several strategies for selecting the final classification based on the output of the binary SVMs
- Truly MultiClass SVMs
  - Generalize the SVM formulation to multiple categories

Conclusions

- SVMs express learning as a mathematical program taking advantage of the rich theory in optimization
- SVM uses the kernel trick to map indirectly to extremely high dimensional spaces
- SVMs extremely successful, robust, efficient, and versatile while there are good theoretical indications as to why they generalize well

Suggested Further Reading

- http://www.kernel-machines.org/tutorial.html
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.
- P.H. Chen, C.-J. Lin, and B. Schölkopf. A tutorial on nu -support vector machines. 2003.
- N. Cristianini. ICML'01 tutorial, 2001.
- K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181-201, May 2001.
- B. Schölkopf. SVM and kernel methods, 2001. Tutorial given at the NIPS Conference.
- Hastie, Tibshirani, Friedman, The Elements of Statistical Learning, Springel 2001

Appendix

Linear SVM

- We like to draw two hyperplanes such that

$$\begin{aligned} W \cdot X_i + b &\geq 1 \quad \textit{if } y_i = +1 \\ W \cdot X_i + b &\leq -1 \quad \textit{if } y_i = -1 \end{aligned}$$

$$\begin{aligned} y_i(W \cdot X_i + b) &\geq 1 \quad \textit{for all } i, \qquad \text{OR} \\ 1 - y_i(W \cdot X_i + b) &\leq 0 \quad ------ \quad -----(1) \end{aligned}$$

- The two parallel hyperplanes are

$$\begin{aligned} H_1 : W \cdot X + b &= 1 \\ H_2 : W \cdot X + b &= -1 \end{aligned}$$

Linear SVM

- Distance between origin and H_1 is $\frac{b-1}{||W||}$
- Distance between origin and H_2 is $\frac{b+1}{||W||}$
- Then, the margin $= \frac{b-1}{||W||} - \frac{b+1}{||W||}$
  $= \frac{2}{||W||}$
- Then the problem is :

  Minimize $\frac{1}{2}||W||^2$ (Objective function)

  Subject to constraints: $1 - y_i(W \cdot X_i + b) \leq 0,$ for all i
- Note that the objective is *convex* and the constraints are *linear*
- Lagrangian method can be applied.

Constrained Optimization Problem

- Minimize $f(v)$
  Subject to the constraints $g_j(v) \leq 0, \;\; 1 \leq j \leq n$.
- Lagrangian,

$$\mathcal{L} = f(v) + \sum_{j=1}^{n} \alpha_j \, g_j(v)$$

  where v is called *primary* variables and α_j are the Lagrangian multipliers which are also called *dual* variables.
- $\mathcal{L}$ has to be minimized with respect to primal varibles and maximized with respect to dual variables.

Constrained Optimization Problem

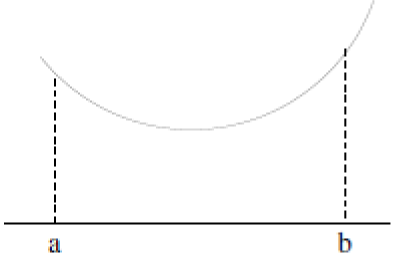
- The K.K.T (Karush-Kuhn-Tucker) conditions "necessary" at optimal v are:
  1. $\nabla_v \mathcal{L} = 0$
  2. $\alpha_j \geq 0$, for all $j = 1, \ldots, n$
  3. $\alpha_j g_j(v) = 0$, for all $j = 1, \ldots, n$
- If $f(v)$ is convex and $g_j(v)$ is linear for all j, then it turns out that K.K.T conditions are "necessary and sufficient" for the optimal v.

Convex Function

- A real valued function f defined in (a, b) is said to be convex if

$$f\left(\lambda x + (1-\lambda)y\right) \;\leq\; \lambda f(x) + (1-\lambda)f(y)$$

for $a < x, y < b,$ and $0 < \lambda < 1$



- This definition can be extended to functions in higher dimensional spaces.

Lagrangian

- Minimize $\frac{1}{2}||W||^2$ (Objective function)
- Subject to constraints: $1 - y_i(W \cdot X_i + b) \leq 0,$ for all i
- Lagrangian,

$$\mathcal{L}(W, b, \alpha) = \frac{1}{2}||W||^2 + \sum_i \alpha_i \left[1 - y_i(W \cdot X_i + b)\right]$$

- Here,

$$\alpha = \begin{pmatrix} \alpha_1 \\ \cdot \\ \cdot \\ \cdot \\ \alpha_n \end{pmatrix}$$

K.K.T. Conditions

- $\nabla_W \mathcal{L} = W - \sum_i \alpha_i y_i X_i = 0$

  $\Rightarrow \quad W = \sum_i \alpha_i y_i X_i \; - - - - - - - - - - - \rightarrow (1)$
- $\frac{\partial \mathcal{L}}{\partial b} = -\sum \alpha_i y_i = 0$

  $\Rightarrow \quad \sum \alpha_i y_i = 0 \; - - - - - - - - - - - - - - \rightarrow (2)$
- $\alpha_i \geq 0,$ for $i = 1$ to n $\; - - - - - - - - - - - \rightarrow (3)$
- $\alpha_i \left[1 - y_i (W \cdot X_i + b)\right] = 0 \; - - - - - - - - - - \rightarrow (4)$

  for $i = 1$ to n.

- Solve these equations $(1), (2), (3), (4)$ to get W and b.
- While it is possible to do this, it is tedious !

Wolfe Dual Formulation

- Other easy and advantageous ways to solve the optimization problem does exist which can be easily extended to non-linear SVMs.
- This is to get $\mathcal{L}$ where we eliminate W and b and which has $\alpha_1, \ldots, \alpha_n$.
- We know, $\mathcal{L}$ has to be maximized w.r.t. the dual variables $\alpha_1, \ldots, \alpha_n$.

Wolfe Dual Formulation

- The Lagrangian $\mathcal{L}$ is :

$$= \frac{1}{2}||W||^2 + \sum_i \alpha_i - \sum_i \alpha_i y_i X_i \cdot W - \sum_i \alpha_i y_i b$$

$$= \frac{1}{2}||W||^2 + \sum_i \alpha_i - W \cdot \underbrace{\left[\sum_i \alpha_i y_i X_i\right]}_{W} - b \underbrace{\left[\sum_i \alpha_i y_i\right]}_{0}$$

$$= -\frac{1}{2}||W||^2 + \sum_i \alpha_i$$

$$= \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j X_i \cdot X_j$$

Wolfe Dual Formulation

- Maximize $\mathcal{L}$ w.r.t α

$$\mathcal{L} = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j X_i \cdot X_j$$

  such that $\sum \alpha_i y_i = 0,$ and $\alpha_i \geq 0$ for all i.
- We need to find the Lagrangian multipliers $\alpha_i \ (1 \leq i \leq n)$ only.
- Primal variables W and b are eliminated.
- There exists various numeric iterative methods to solve this constrained convex quadratic optimization problem.
- *Sequential minimal optimization (SMO)* is one such technique which is a simple and relatively fast method.

- If $\alpha_i > 0$ (from (4)) $\Rightarrow$ X_i is on a hyperplane, i.e., X_i is a `support vector`.
  Note: X_j lies on the hyperplane $\not\Rightarrow$ $\alpha_i > 0$
- Similarly, X_i doesnot lie on hyperplane $\Rightarrow \alpha_i = 0$
  That is, for interior points $\alpha_i = 0$.

The Solution

- Once α is known, We can find $W = \sum_i \alpha_i y_i X_i$
- The classifier is

$$f(X) = Sign\{W \cdot X + b\} = Sign\left\{\sum_i \alpha_i y_i X_i \cdot X + b\right\}$$

- b can be found from (4)
  - For any $\alpha_j \neq 0$, we have $y_j(W \cdot X_j + b) = 1$
  - Multiplying with y_j on both sides we get, $W \cdot X_j + b = y_j$
  - So, $b = y_j - W \cdot X_j \ = \ y_j - \sum_i \alpha_i y_i X_i \cdot X_j$

Some observations

- In the dual problem formulation and in its solution we have only dot products between some of the training patterns.
- Once we have the matrix $\hat{K}$, the problem and its solution are independent of the dimensionality d.

Non-linear SVM

- We know that every non-linear function in X-space (input space) can be seen as a linear function in an appropriate Y-space (feature space).
- Let the mapping be $Y = \phi(X)$.
- Once the $\phi(\cdot)$ is defined, one has to replace in the problem as well as in the solution, for certain products, as explained below.
- Whenever we see $X_i \cdot X_j$, replace this by $\phi(X_i) \cdot \phi(X_j)$.
- While it is possible to explicitly define the $\phi(\cdot)$ and generate the training set in the Y-space, and then obtain the solution...
- it is tedious and amazingly unnecessary also.

Kernel Function

- For certain mappings, $\phi(X_i) \cdot \phi(X_j) = \kappa(X_i, X_j)$. That is, dot product in the Y-space can be obtained as a function in the X-space itself. There is no need to explicitly generate the patterns in the Y-space.
- Eg: Consider a two dimensional problem with $X = (x_1, x_2)^t$. Let $\phi(X) = (x_1^2, x_2^2, \sqrt{2}\, x_1 x_2)^t$. Then $\phi(X_i) \cdot \phi(X_j) = \kappa(X_i, X_j) = (X_i \cdot X_j)^2$.
- This *kernel trick* is one of the reasons for the success of SVMs.

Wolfe Dual Formulation

- Maximize $\mathcal{L}$ w.r.t α

$$\mathcal{L} = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j X_i \cdot X_j$$

such that $\sum \alpha_i y_i = 0,$ and $0 \leq \alpha_i \leq C$ for all i.

- $C = \infty \quad \Rightarrow$ Hard Margin
- $C = 0 \quad \Rightarrow$ Very Very Soft Margin

Training Methods

- Any convex quadratic programming technique can be applied.
- But with larger training sets, most of the standard techniques can become very slow and space occupying. For example, many techniques needs to store the kernel matrix whose size is n^2 where n is the number of training patterns.
- These considerations have driven the design of specific algorithms for SVMs that can exploit the sparseness of the solution, the convexity of the optimization problem, and the implicit mapping into feature space.
- One such a simple and fast method is Sequential Minimal Optimization (SMO).

- Builtin library functions, toolboxes are readily available in Python, …