BITS Pilani
Pilani Campus

Support Vector Machines

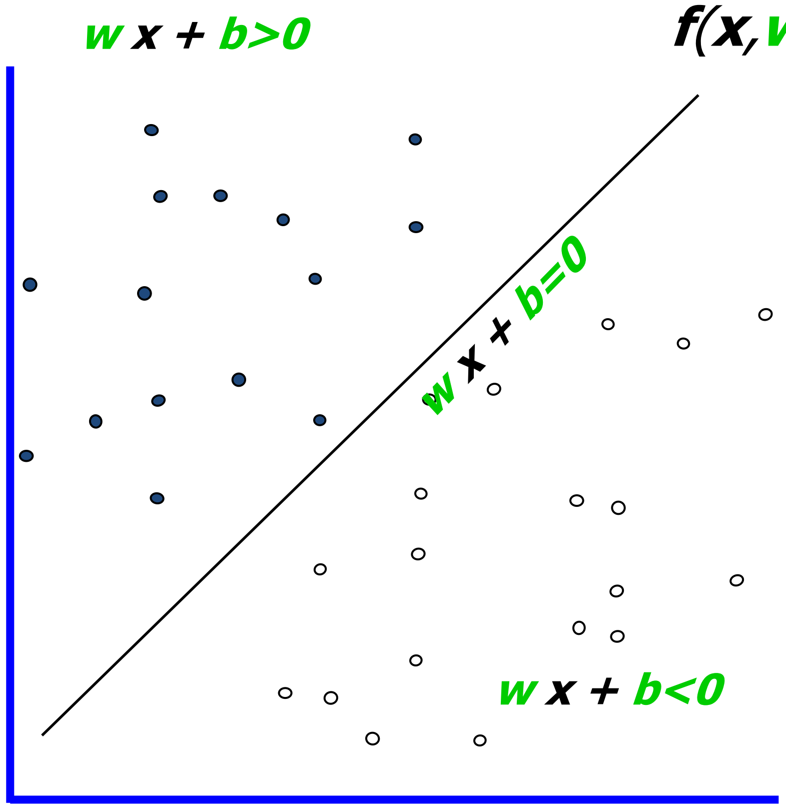
Dr. Sugata Ghosal
sugata.ghosal@pilani.bits-pilani.ac.in

SVM - I

- Linear Classifiers
- Maximum Margin Classification
- Linear SVM
- SVM optimization problem
- Soft Margin SVM

Linear Classifiers

- denotes +1
- denotes -1



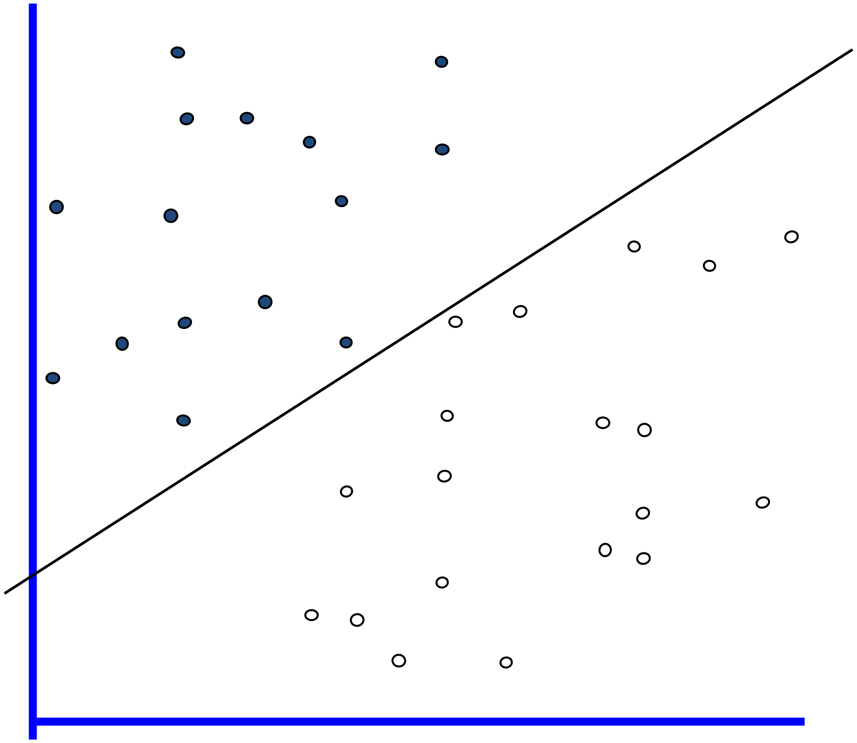
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$

How would you classify this data?

Linear Classifiers

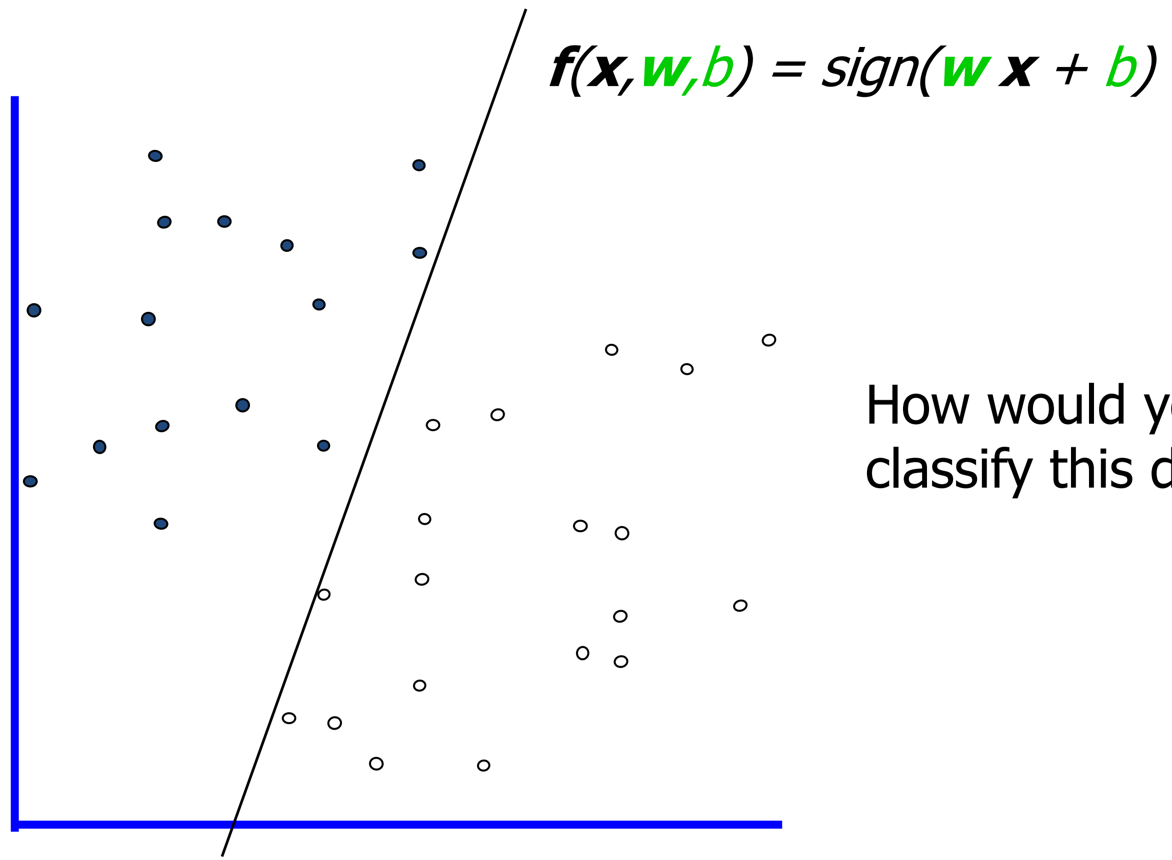
$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\ \mathbf{x} + b)$$

- denotes +1
- denotes -1



How would you classify this data?

Linear Classifiers



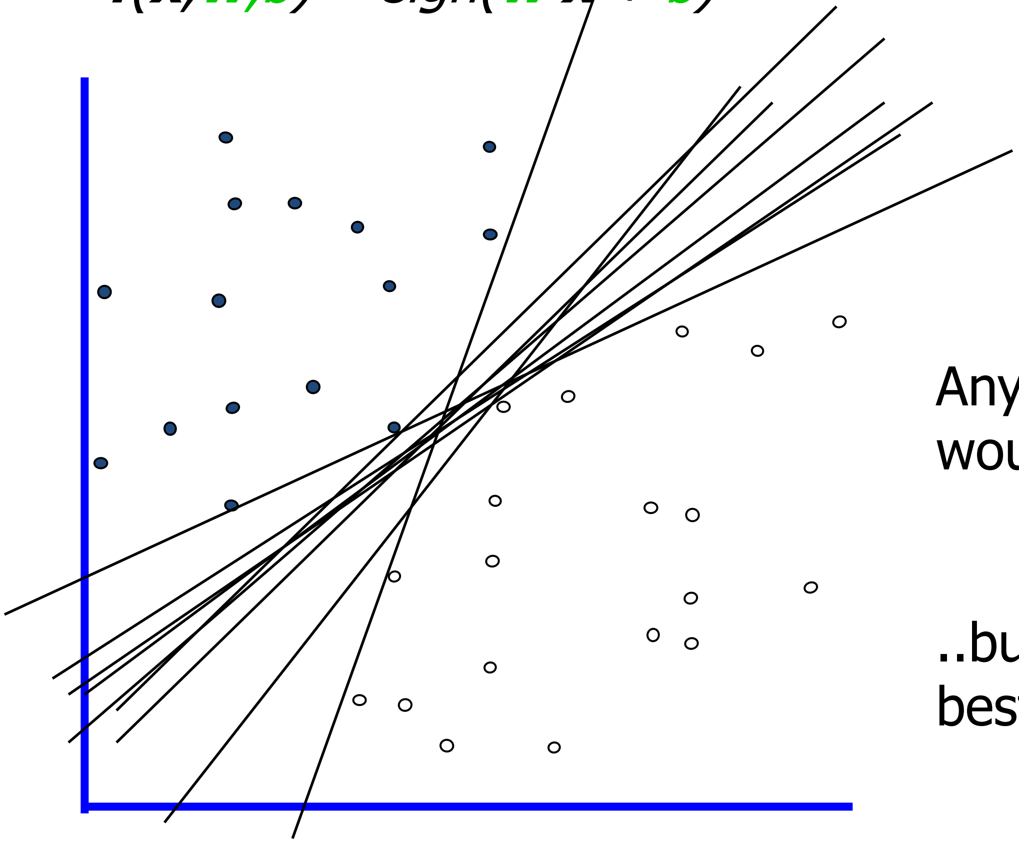
- denotes +1
- denotes -1

How would you classify this data?

Linear Classifiers

$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\,\mathbf{x} + b)$$

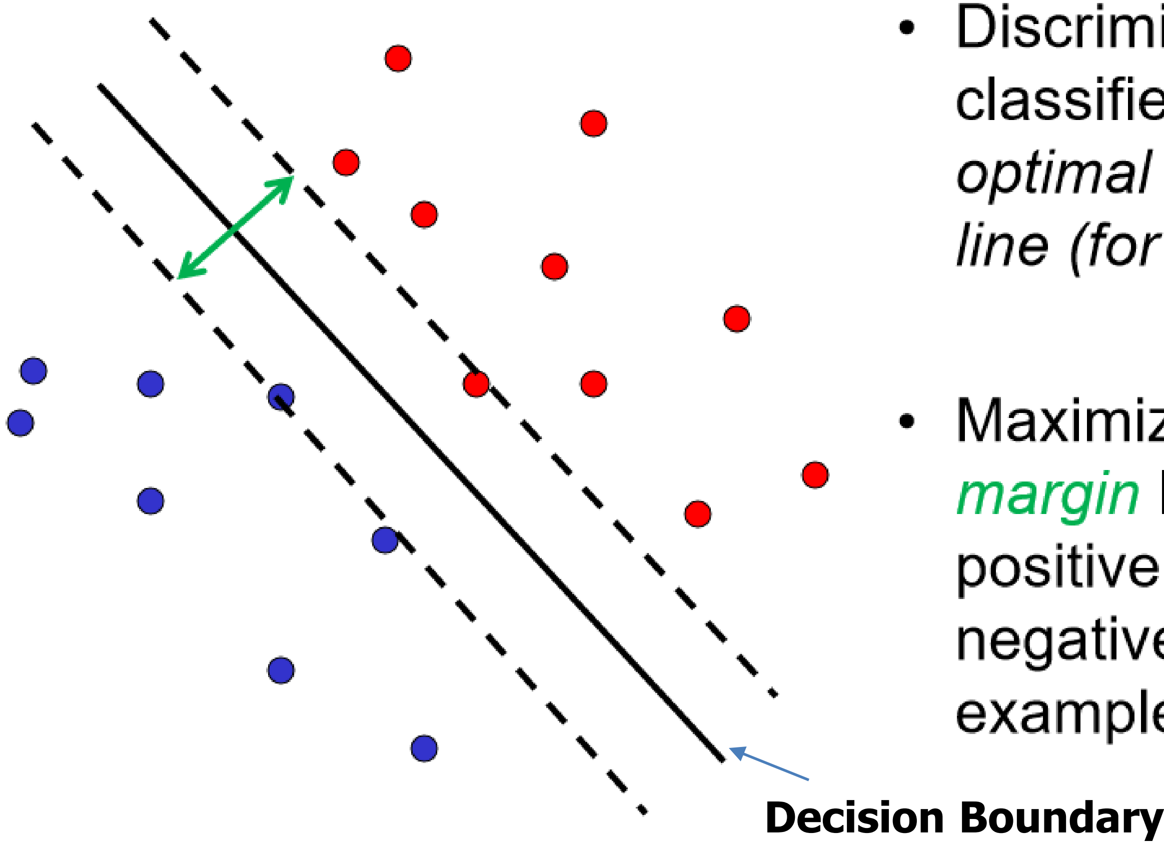
- denotes +1
- denotes -1



Any of these would be fine..

..but which is best?

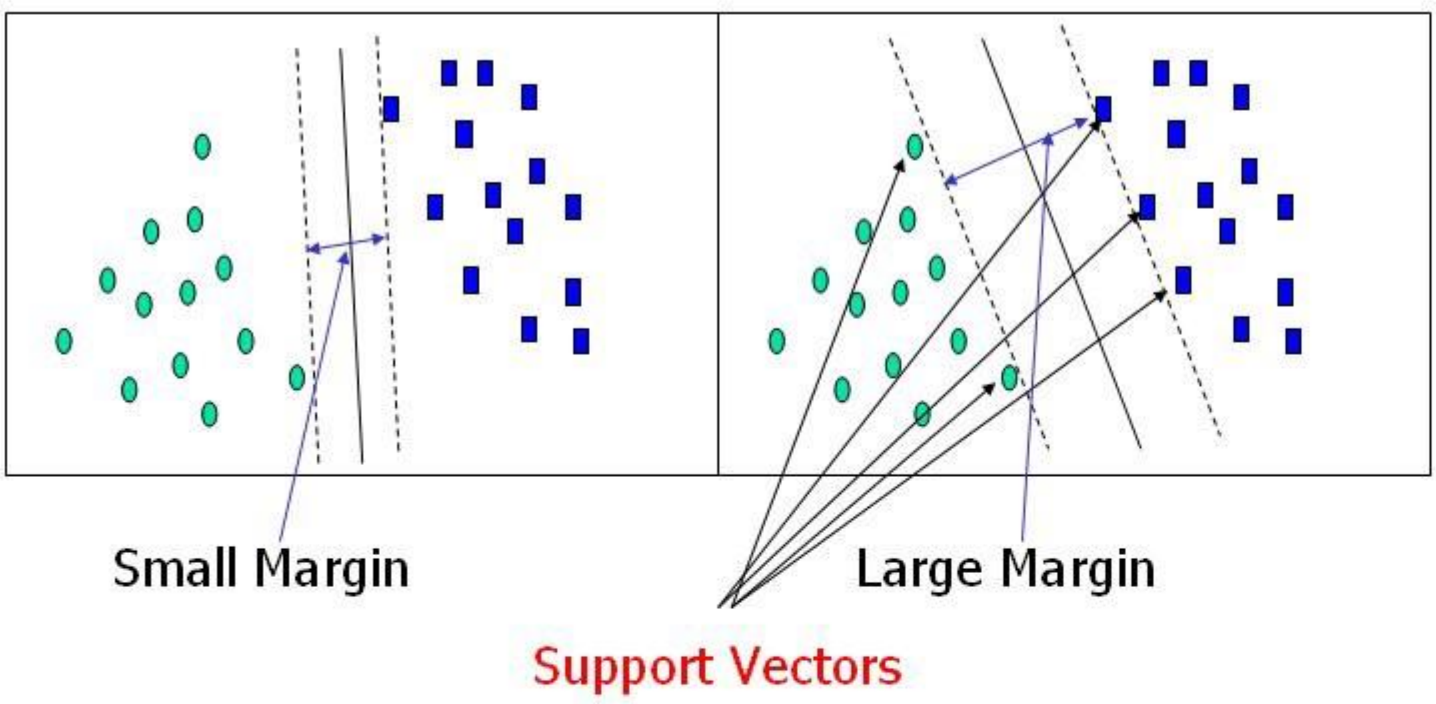
Linear Classifier



- Discriminative classifier based on *optimal separating line (for 2d case)*
- Maximize the *margin* between the positive and negative training examples

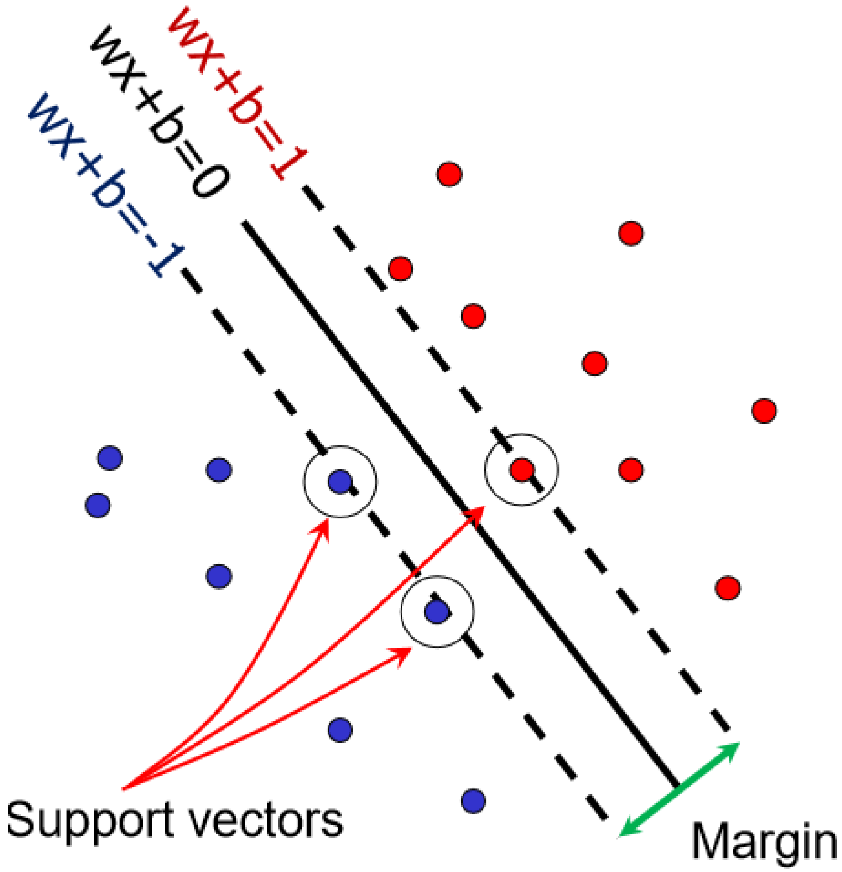
C. Burges, [A Tutorial on Support Vector Machines for Pattern Recognition](), Data Mining and Knowledge Discovery, 1998

Large margin and support vectors



Support Vector Machines

- Want line that maximizes the margin.



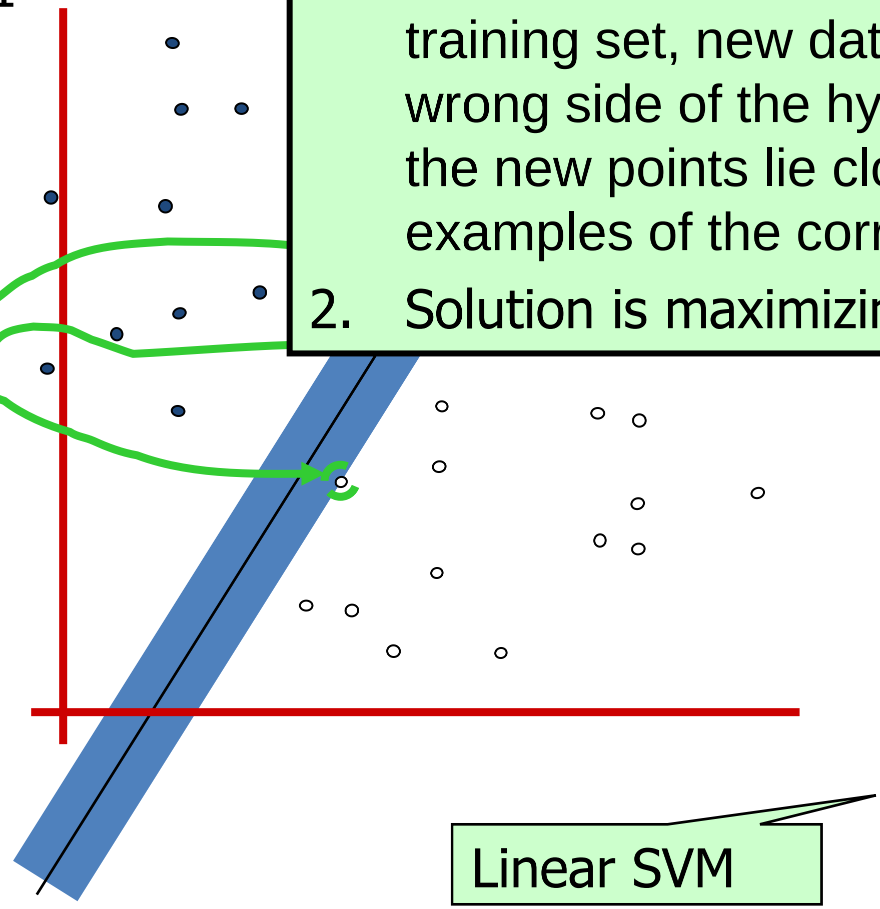
$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

For support vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Maximum Margin

- denotes +1
- ◦ denotes -1



Support Vectors are those datapoints that the margin pushes up against

1. If hyperplane is oriented such that it is close to some of the points in your training set, new data may lie on the wrong side of the hyperplane, even if the new points lie close to training examples of the correct class.
2. Solution is maximizing the margin

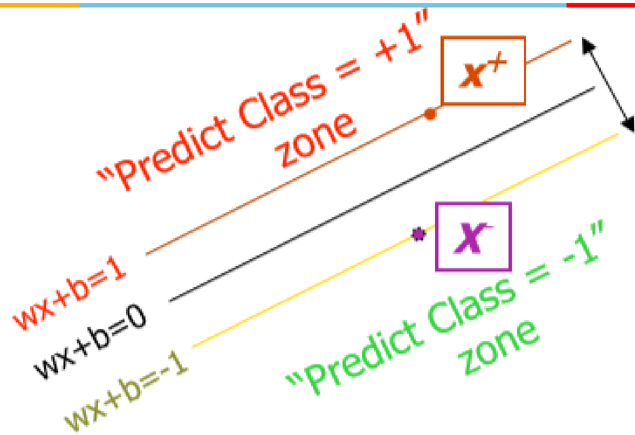
with the, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

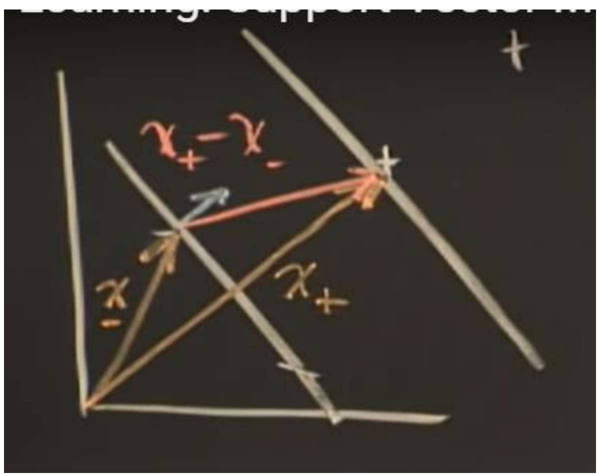
Support Vectors

- Geometric description of SVM is that the max-margin hyperplane is completely determined by those points that lie nearest to it.
- Points that lie on this margin are the support vectors.
- The points of our data set which if removed, would alter the position of the dividing hyperplane

Linear SVM Mathematically



M=Margin Width



w . x⁺ + b = +1

$$w \cdot x^+ + b = +1$$

$$w \cdot x^- + b = -1$$

Margin width

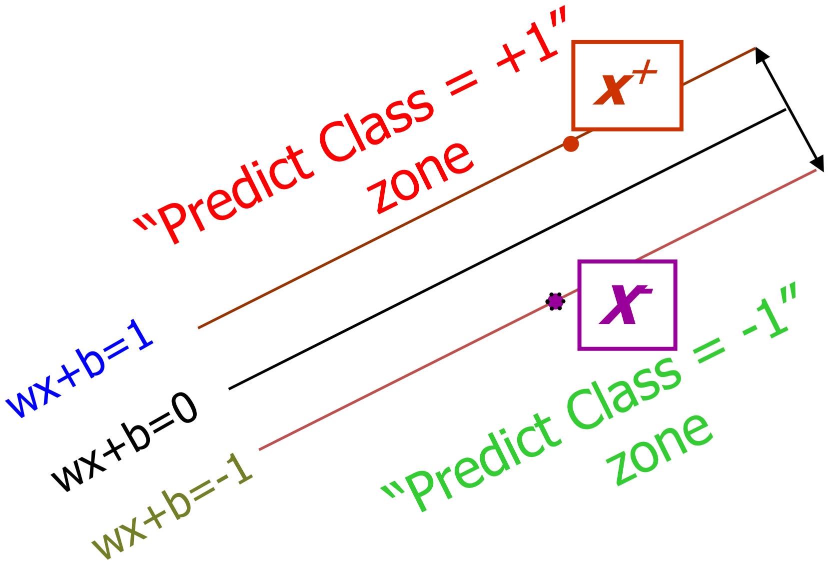
$$= (x^+ - x^-) \cdot \frac{w}{||w||}$$

$$= \frac{w \cdot x^+ - w \cdot x^-}{||w||}$$

$$= (1-b) - (-1-b) \,/\, ||w||$$

$$= \frac{2}{||w||}$$

Linear SVM Mathematically



M=Margin Width

Distance between lines given by solving linear equation:

What we know:

- $\mathbf{w} \cdot \mathbf{x}^+ + b = +1$
- $\mathbf{w} \cdot \mathbf{x}^- + b = -1$

Maximize margin: $M = \dfrac{2}{||w||}$

Equivalent to minimize: $\frac{1}{2}||w||^2$

Solving the Optimization Problem

1. Maximize margin $2/||\mathbf{w}||$
2. Correctly classify all training data points:

$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

Quadratic optimization problem:

Find **w** and b such that

$\mathbf{\Phi}(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2$ is minimized;

and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

$y_i\ (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

$+1(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

$-1(\mathbf{w}^T\mathbf{x}_i + b) \leq 1$

same as $(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Solving the Optimization Problem

Find $\mathbf{w}$ and b such that
$\mathbf{\Phi}(\mathbf{w}) = \frac{\mathbf{1}}{\mathbf{2}}||\mathbf{w}||^2$ is minimized; Type equation here.
and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i\,(\mathbf{w^T x_i} + b) \geq 1$

← Primal

- **Need to optimize a *quadratic* function subject to *linear inequality* constraints.**
- **All constraints in SVM are linear**
- **Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.**
- **The solution involves constructing a *unconstrained problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:**

Optimization Problem

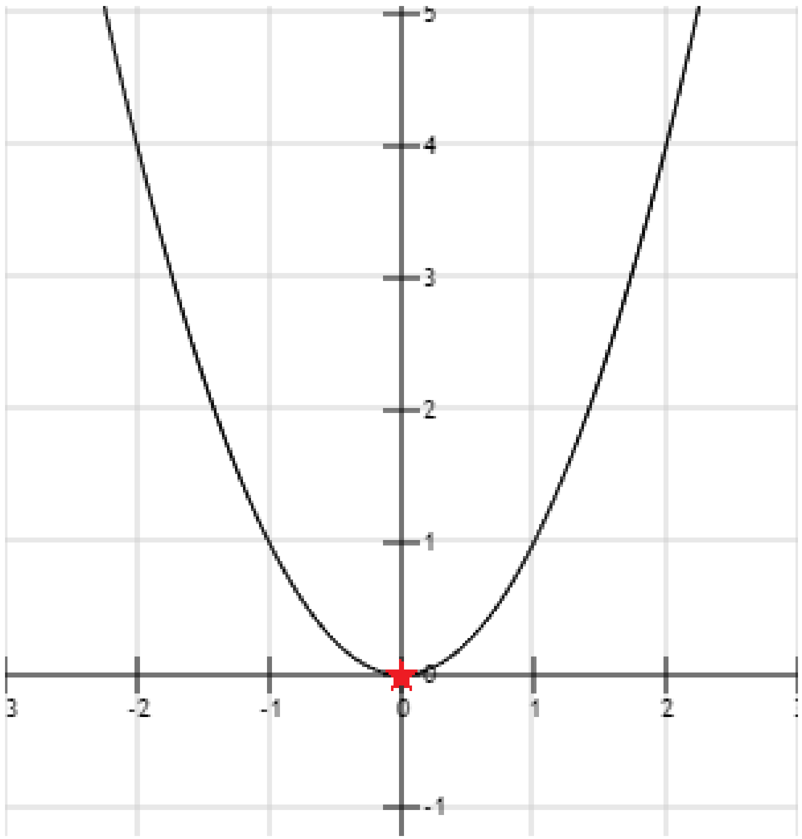
- Optimization problem is typically written:

$$\text{Minimize } f(x)$$
$$\text{subject to}$$
$$g_i(x) = 0, \quad i=1,\dots,p$$
$$h_i(x) <= 0, \quad i=1,\dots,m$$

- $f(x)$ is called the objective function
- By changing x (the optimization variable) we wish to find a value $x*$ for which $f(x)$ is at its minimum.
- p functions of g_i define equality constraints and
- m functions h_i define inequality constraints.
- The value we find MUST respect these constraints!

Unconstrained Optimization

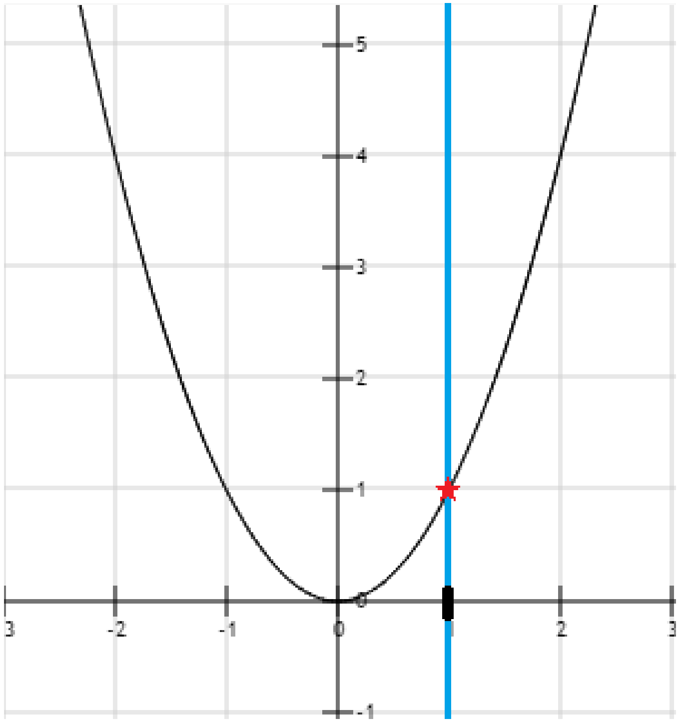
- Minimize x^2



Constrained Optimization -Equality Constraint

Minimize x^2

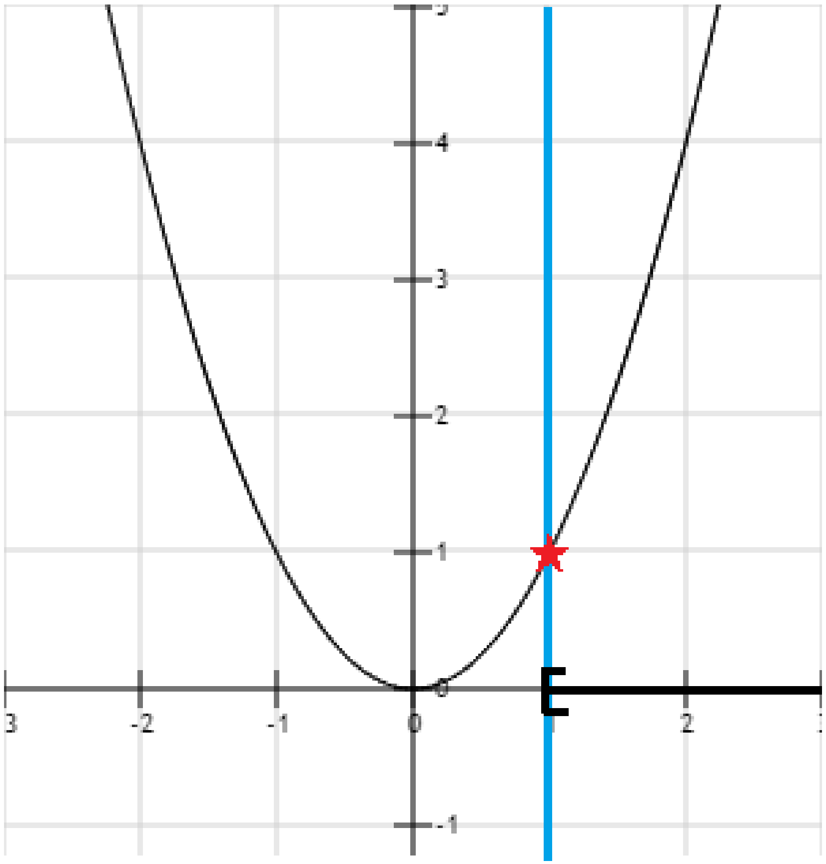
Subject to $x = 1$



Constrained Optimization -Inequality Constraint

Minimize x^2

Subject to x >= 1



Example:

$$\max_{x,y} xy \text{ subject to } x + y = 6$$

- Introduce a Lagrange multiplier λ for constraint
- Construct the Lagrangian

$$L(x, y) = xy - \lambda(x + y - 6)$$

- Stationary points

$$\frac{\partial L(x,y)}{\partial \lambda} = x + y - 6 = 0$$

$$\left.\begin{aligned} \frac{\partial L(x,y)}{\partial x} &= y - \lambda = 0 \\ \frac{\partial L(x,y)}{\partial y} &= x - \lambda = 0 \end{aligned}\right\} \Rightarrow x = y = \lambda$$

$$\Rightarrow x = y = 3$$

x and y values remain same even if you take +λ or -λ for equality constraint

2 x = 6
x = y = 3
λ = 3

Karush–Kuhn–Tucker (KKT) theorem

- KKT approach to nonlinear programming (quadratic) generalizes the method of Lagrange multipliers, which allows only equality constraints.
- KKT allows inequality constraints

Karush-Kuhn-Tucker (KKT) conditions

- Start with

  max f(x) subject to

  $$g_i(x) = 0 \text{ and } h_j(x) \geq 0 \text{ for all } i, j$$

- Make the Lagrangian function

  $$\mathcal{L} = f(x) - \sum_i \lambda_i g_i(x) - \sum_j \mu_j h_j(x)$$

- Take gradient and set to 0 – but other conditions also.

KKT conditions

- Make the Lagrangian function

$$\mathcal{L} = f(x) - \sum_i \lambda_i g_i(x) - \sum_j \mu_j h_j(x)$$

- Necessary conditions to have a minimum are

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0$$

$$g_i(x^*) = 0 \text{ for all } i$$

$$h_j(x^*) \geq 0 \text{ for all } j$$

$$\mu_j \geq 0 \text{ for all } j$$

$$\mu_j^* h_j(x^*) = 0 \text{ for all } j$$

Solving the Optimization Problem

- **The solution involves constructing a *dual problem* where a *Lagrange multiplier α_i* is associated with every constraint in the primary problem:**

$$\boxed{L(w, b, \alpha_i) = \frac{1}{2}||\mathbf{w}||^2 - \Sigma\, \alpha_i\, [y_i\, (\mathbf{w}^T\mathbf{x_i} + b) - 1]}$$

- **Taking partial derivative with respect to w , $\frac{\partial L}{\partial w}$ = 0**
  - w - $\Sigma\, \alpha_i y_i\, \mathbf{x_i} = \mathbf{0}$
  - w $= \Sigma\, \alpha_i y_i\, \mathbf{x_i}$
- **Taking partial derivative with respect to b, $\frac{\partial L}{\partial b}$ = 0**
  - $- \Sigma\, \alpha_i\, y_i = 0$
  - $\Sigma\, \alpha_i\, y_i = 0$

Support Vectors

Using KKT conditions :

$\alpha_i\,[y_i\,(\mathbf{w}^T\mathbf{x_i} + b) - 1] = 0$

For this condition to be satisfied

either $\alpha_i = 0$ and $y_i\,(\mathbf{w}^T\mathbf{x_i} + b) - 1 > 0$

OR

$y_i\,(\mathbf{w}^T\mathbf{x_i} + b) - 1 = 0$ and $\alpha_i > 0$

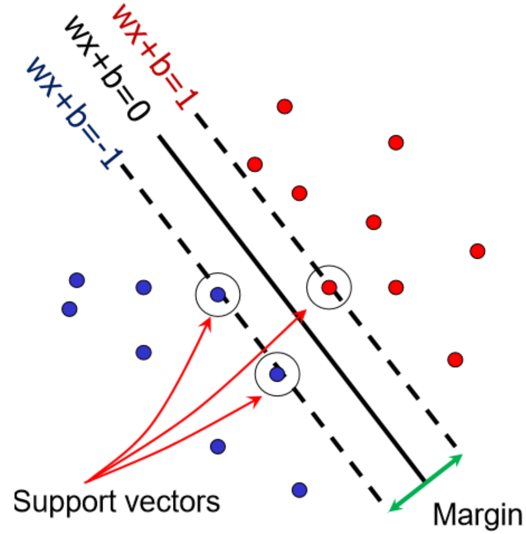
For support vectors:

$y_i\,(\mathbf{w}^T\mathbf{x_i} + b) - 1 = 0$

For all points other than support vectors:

$\alpha_i = 0$

- Want line that maximizes the margin.



$\mathbf{x}_i$ positive $(y_i = 1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support vectors, $\quad \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

$$\boxed{L(w, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \Sigma\, \alpha_i\,[y_i\,(\mathbf{w}^T\mathbf{x_i} + b) - 1]}$$

Solving the Optimization Problem

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

Learned weight

Support vector

Solving the Optimization Problem

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

  $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$ (for any support vector)

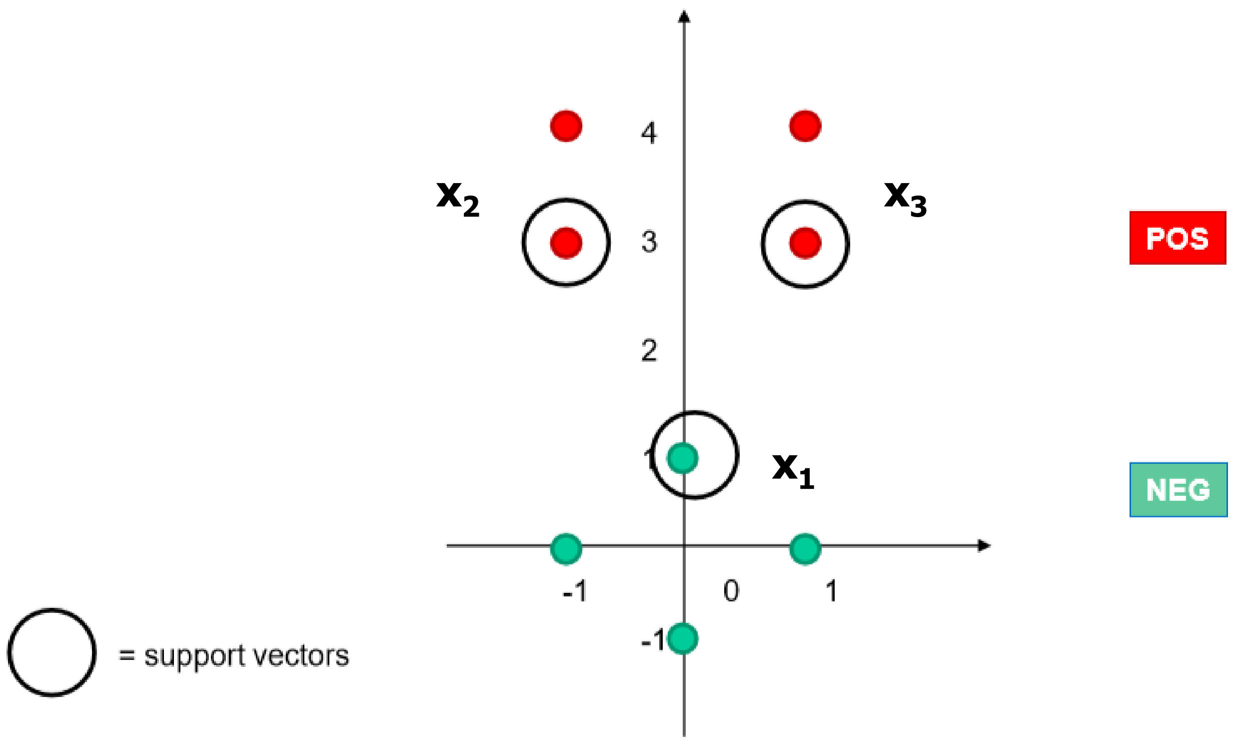
- Classification function:

$$f(x) = \text{sign}\,(\mathbf{w} \cdot \mathbf{x} + \text{b})$$
$$= \text{sign}\left(\sum_{\text{i}} \alpha_{\text{i}} y_i \boxed{\mathbf{x}_i \cdot \mathbf{x}} + b\right)$$

If f(x) < 0, classify as negative, otherwise classify as positive.

- Notice that it relies on an *inner product* between the test point $\boldsymbol{x}$ and the support vectors $\boldsymbol{x}_i$
- (Solving the optimization problem also involves computing the inner products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ between all pairs of training points)

Example



Example adapted from Dan Ventura

Solving for α

- We know that for the support vectors, f(x) = 1 or -1 exactly
- Add a 1 in the feature representation for the bias
- The support vectors have coordinates and labels:
  - x1 = [0 1 1], y1 = -1
  - x2 = [-1 3 1], y2 = +1
  - x3 = [1 3 1], y3 = +1
- Thus we can form the following system of linear equations:

Solving for α

- System of linear equations:

α1 y1 dot(x1, x1) + α2 y2 dot(x1, x2) + α3 y3 dot(x1, x3) = y1

α1 y1 dot(x2, x1) + α2 y2 dot(x2, x2) + α3 y3 dot(x2, x3) = y2

α1 y1 dot(x3, x1) + α2 y2 dot(x3, x2) + α3 y3 dot(x3, x3) = y3

-2 * α1 + 4 * α2 + 4 * α3 = -1

-4 * α1 + 11 * α2 + 9 * α3 = +1

-4 * α1 + 9 * α2 + 11 * α3 = +1

$\alpha_i\,[-1\,(\mathbf{w} \bullet \mathbf{x_i} + b)] = -1$

$\alpha_i\,[+1\,(\mathbf{w} \bullet \mathbf{x_i} + b)] = 1$

- Solution: α1 = 3.5, α2 = 0.75, α3 = 0.75

Solving for w and b

We know $w = \alpha_1 y_1 x_1 + \ldots + \alpha_N y_N x_N$ where N = # SVs

Thus w = -3.5 * [0 1 1] + 0.75 [-1 3 1] + 0.75 [1 3 1] = [0 1 -2]

Separating out weights and bias, we have: w = [0 1] and b = -2

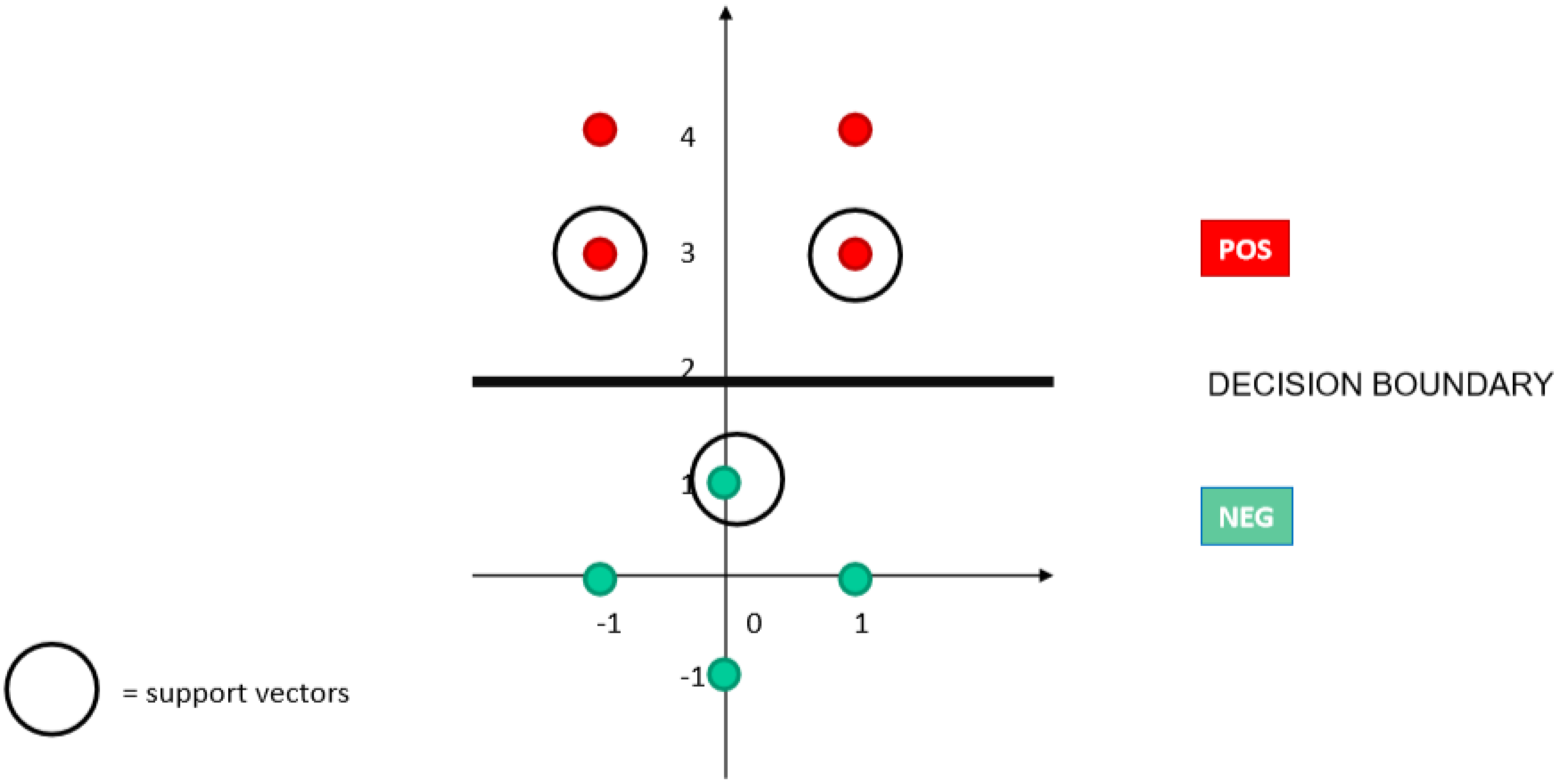
For SVMs, we used this eq for a line: ax + cy + b = 0 where w = [a c]

Thus ax + b = -cy ➔ y = (-a/c) x + (-b/c)

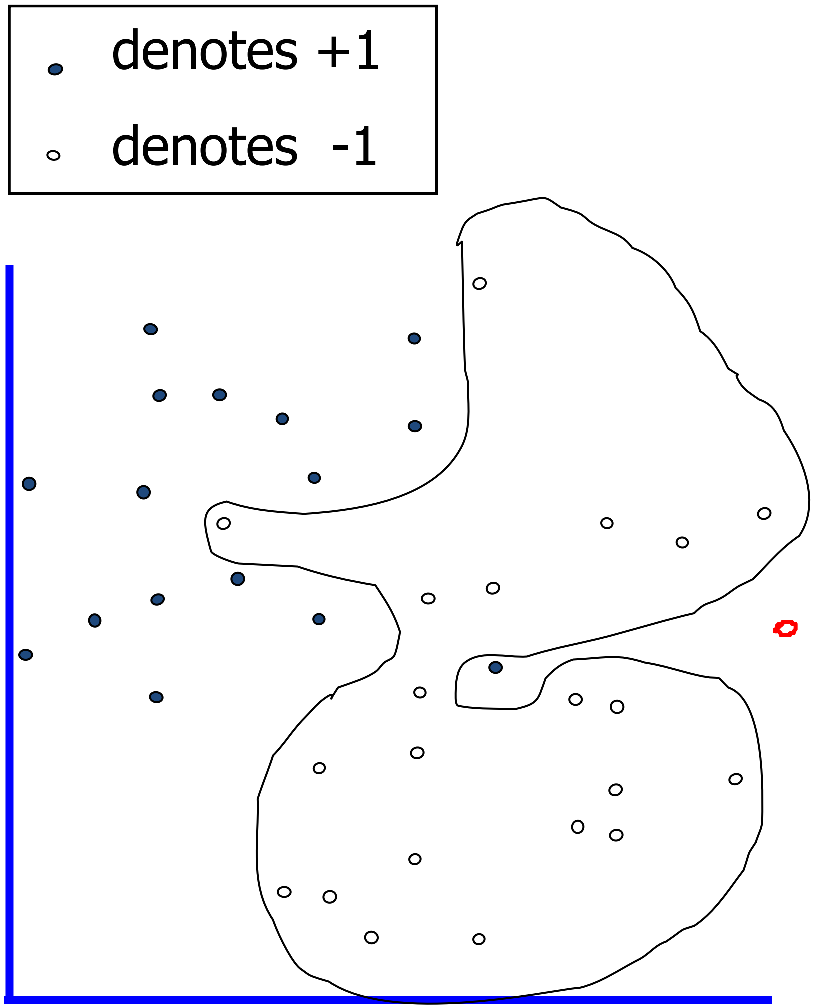
Thus y-intercept is -(-2)/1 = 2

The decision boundary is perpendicular to w and it has slope -0/1 = 0

Decision boundary



Dataset with noise



- **Hard Margin: So far we require all data points be classified correctly**

  - No training error

- **What if the training set is noisy?**

Soft Margin Classification

Slack variables ξ_i **can be added to allow misclassification of difficult or noisy examples.**

What should our quadratic optimization criterion be?

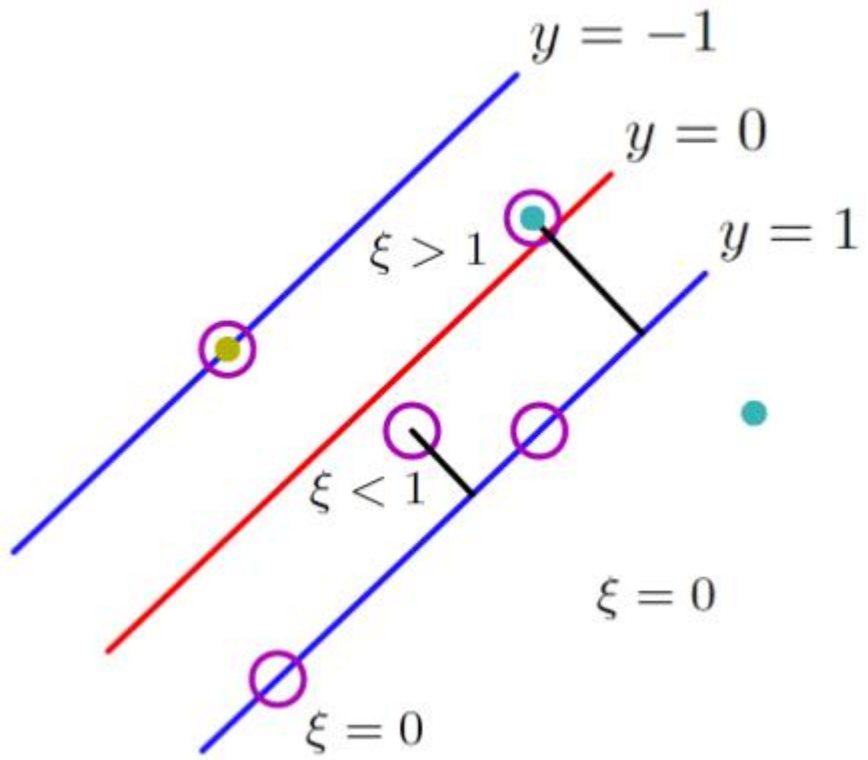
Minimize

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

Slack Variable

- **Slack variable** as giving the classifier some leniency when it comes to moving around points near the **margin**.
- When C is large, larger slacks penalize the objective function of SVM's more than when C is small.

Soft margin example



Soft Margin

Misclassification cost

data samples

Slack variable

$$\min_{\boldsymbol{w}} \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2}_{\text{Maximize margin}} + \underbrace{C\sum_{i=1}^{N}\xi_i}_{\text{Minimize misclassification}}$$

The *w* that minimizes…

$$\text{subject to} \quad y_i \boldsymbol{w}^T \boldsymbol{x}_i \geq 1 - \xi_i,$$

$$\xi_i \geq 0, \quad \forall i = 1, \ldots, N$$

Hard Margin versus Soft Margin

- **Hard Margin:**

Find $\mathbf{w}$ and b such that

$\boldsymbol{\Phi}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

$y_i (\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

- **Soft Margin incorporating slack variables:**

Find $\mathbf{w}$ and b such that

$\boldsymbol{\Phi}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum \xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$

$y_i (\mathbf{w}^T\mathbf{x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

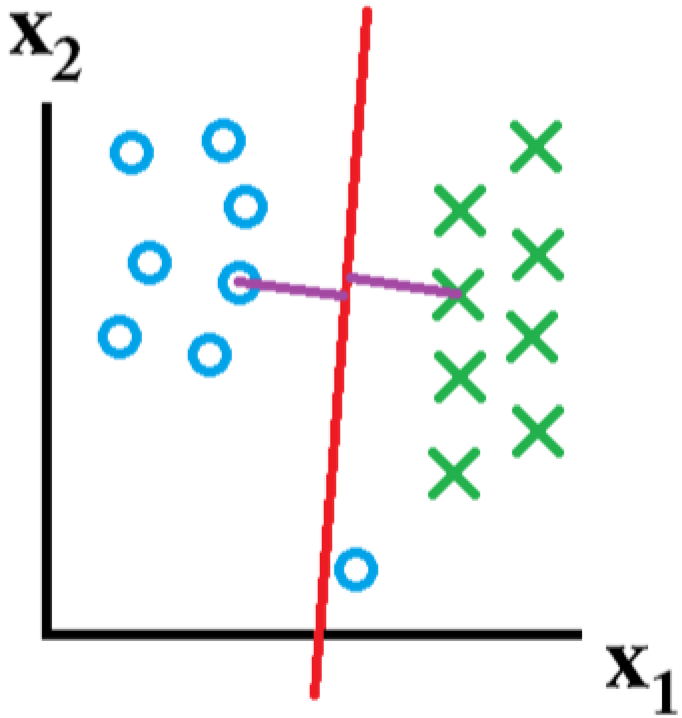
- **Parameter *C* can be viewed as a way to control overfitting.**

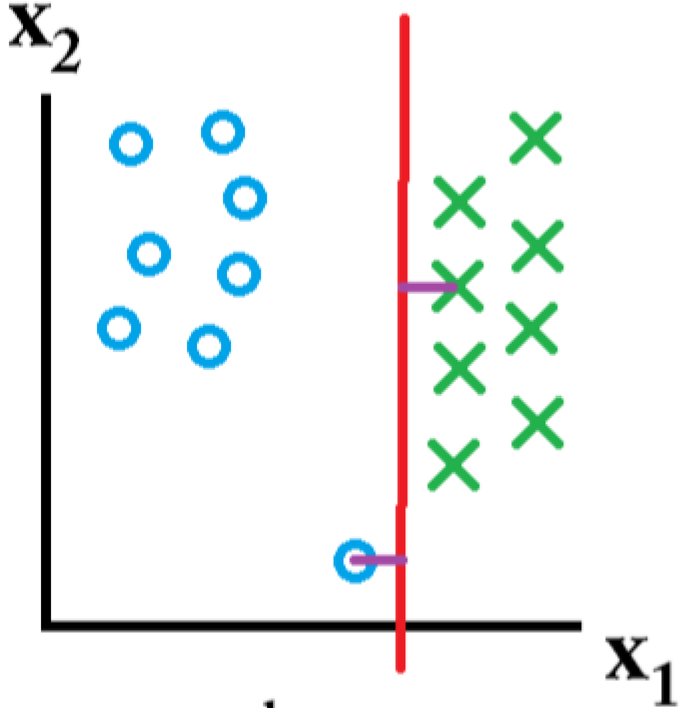
Value of C parameter

- C parameter tells the SVM optimization how much you want to avoid misclassifying each training example.
- For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.
- Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

Effect of Margin size v/s misclassification cost

Training set



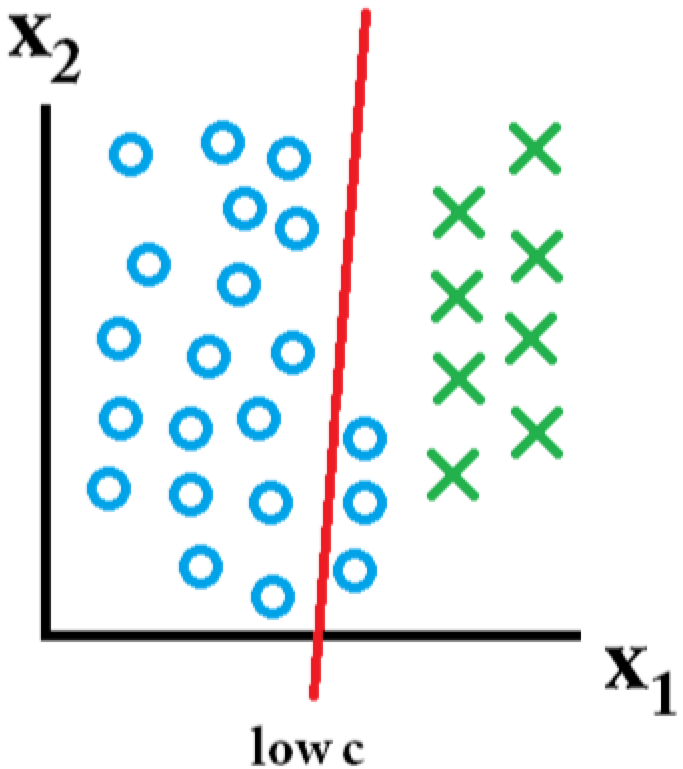


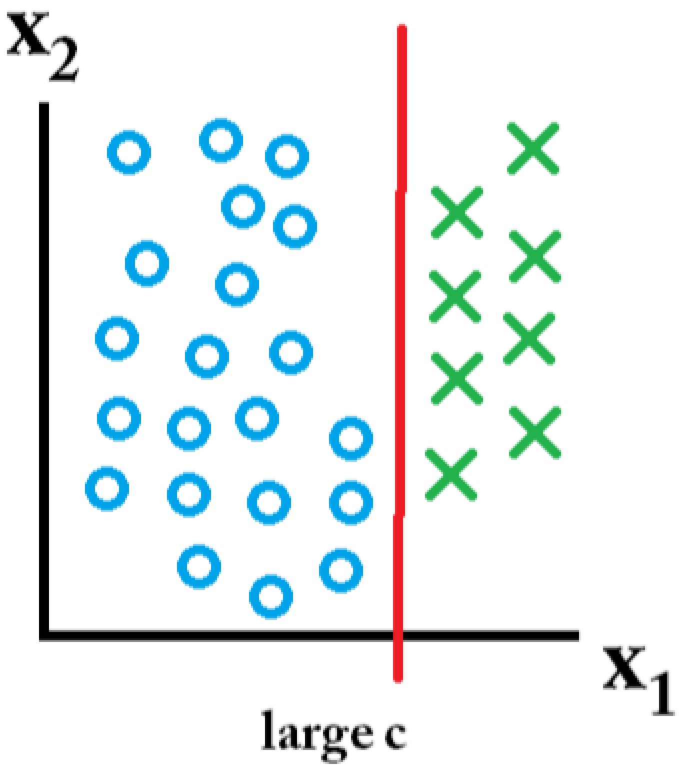
Misclassification ok, want large margin

Misclassification not ok

Effect of Margin size v/s misclassification cost

Including test set A



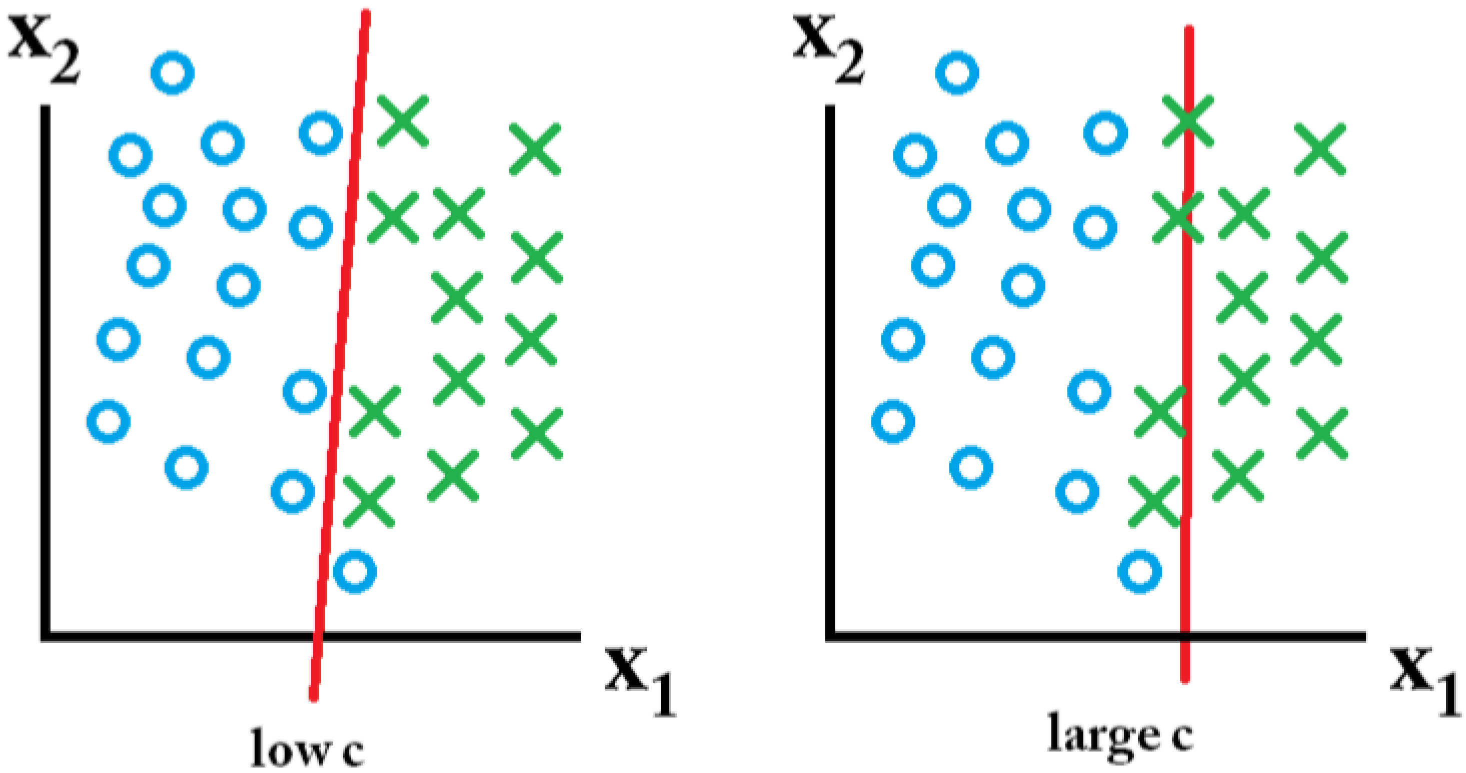


Misclassification ok, want large margin

Misclassification not ok

Effect of Margin size v/s misclassification cost

Including test set B



Misclassification ok, want large margin

Misclassification not ok

Good Web References for SVM

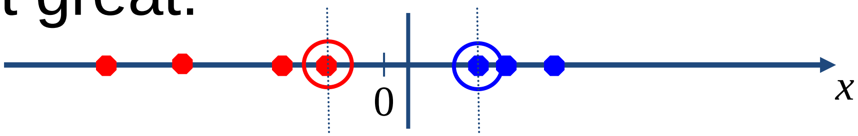
- **Text categorization with Support Vector Machines: learning with many relevant features** - T. Joachims, ECML
- **A Tutorial on Support Vector Machines for Pattern Recognition**, Kluwer Academic Publishers - Christopher J.C. Burges
- http://www.cs.utexas.edu/users/mooney/cs391L/
- https://www.coursera.org/learn/machine-learning/home/week/7
- https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
- https://data-flair.training/blogs/svm-kernel-functions/
- MIT 6.034 Artificial Intelligence, Fall 2010
- https://stats.stackexchange.com/questions/30042/neural-networks-vs-support-vector-machines-are-the-second-definitely-superior
- https://www.sciencedirect.com/science/article/abs/pii/S0893608006002796
- https://medium.com/deep-math-machine-learning-ai/chapter-3-support-vector-machine-with-math-47d6193c82be
- Radial basis kernel

SVM - II

- Nonlinear SVM
- Kernel Trick
- SVM Kernels
- Multi-Class Problem
- SVM vs Logistic Regression
- SVM Applications

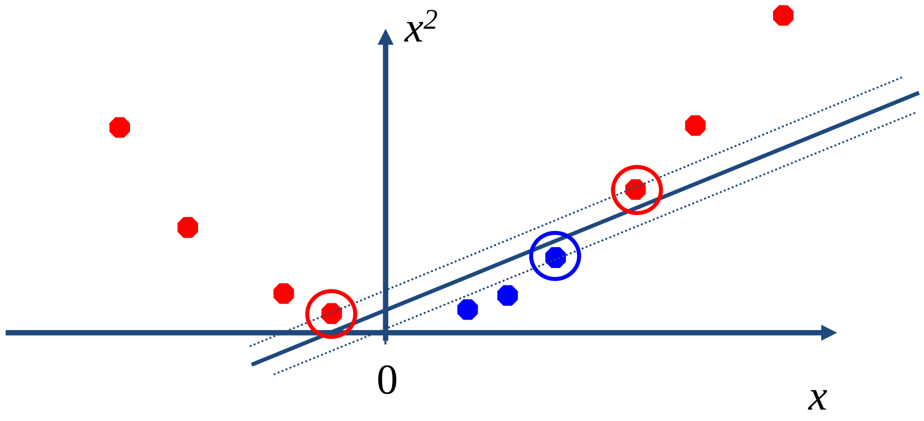
Non-linear SVMs

- Datasets that are linearly separable with some noise soft margin work out great:

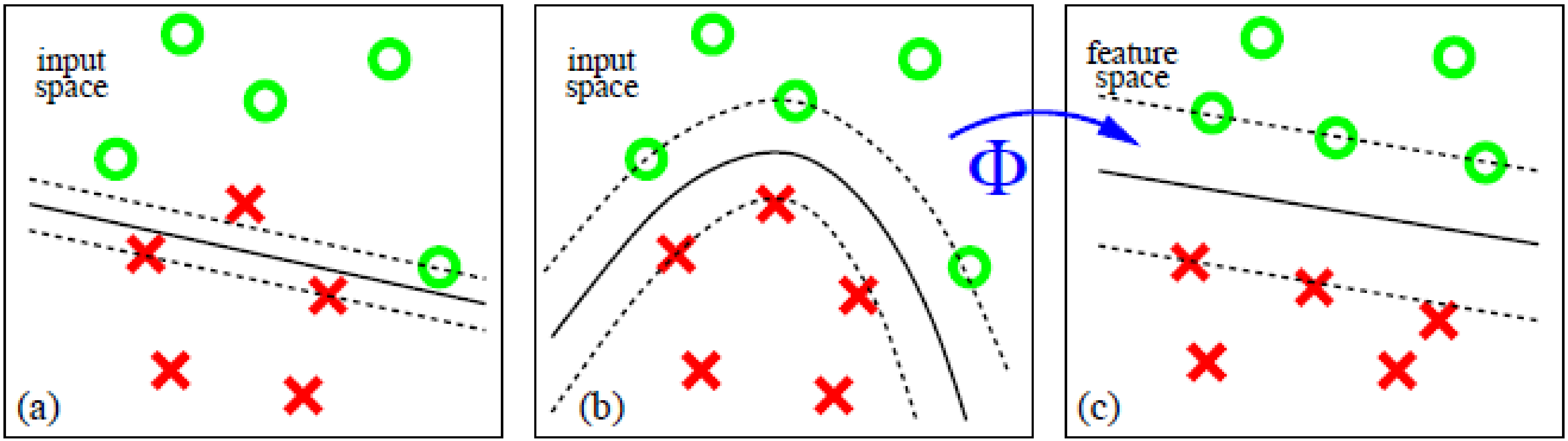


- But what are we going to do if the dataset is just too hard?

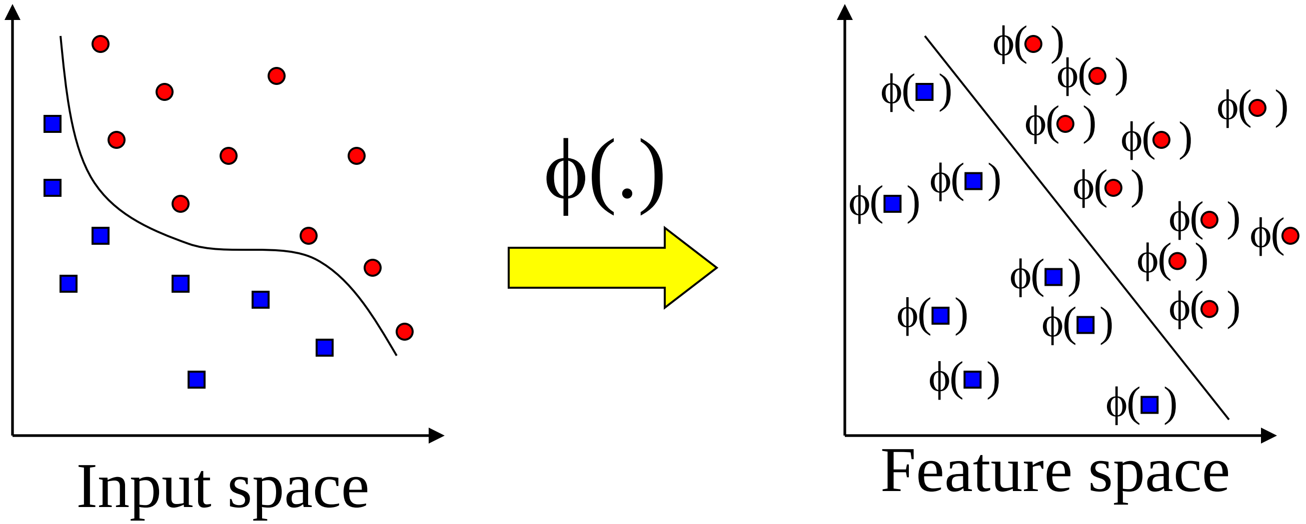
- How about… mapping data to a higher-dimensional space:



Find a feature space



Transforming the Data

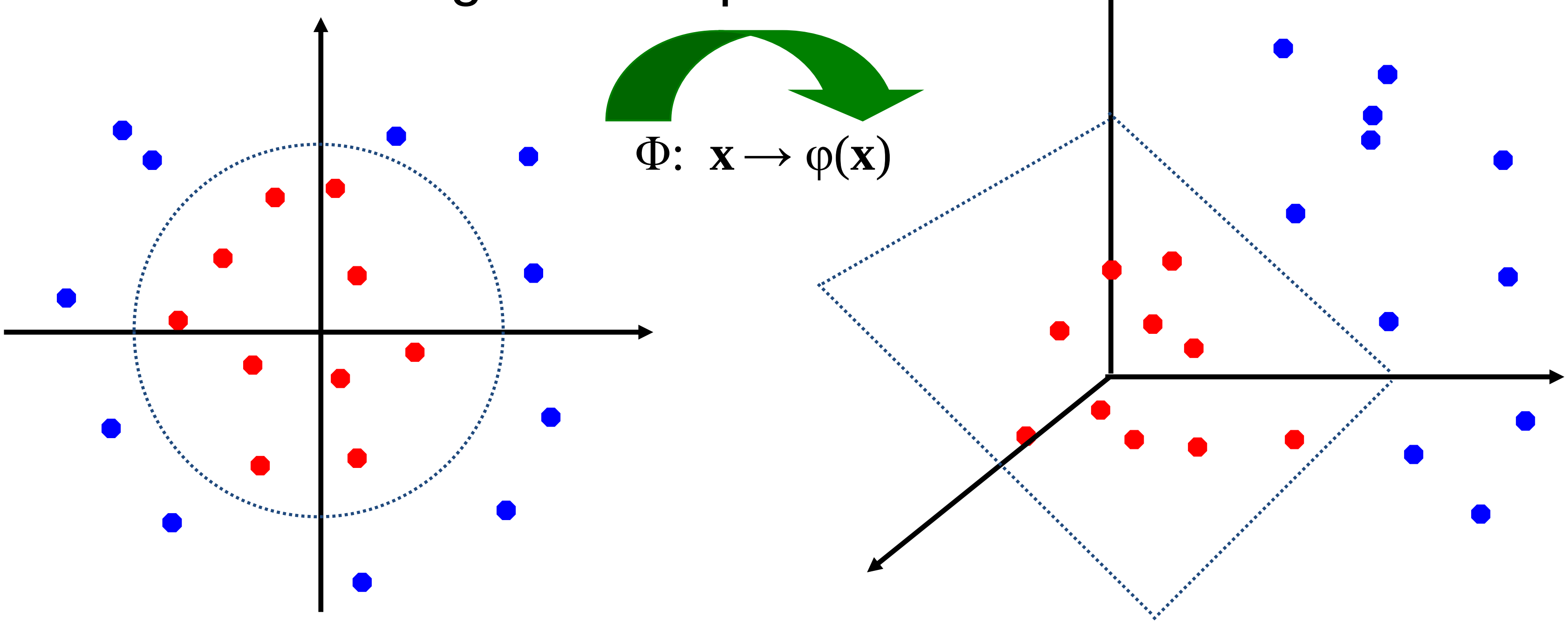


Note: feature space is of higher dimension than the input space in practice

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue

Non-linear SVMs: Feature spaces

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



SVM – Overlapping Class Scenario

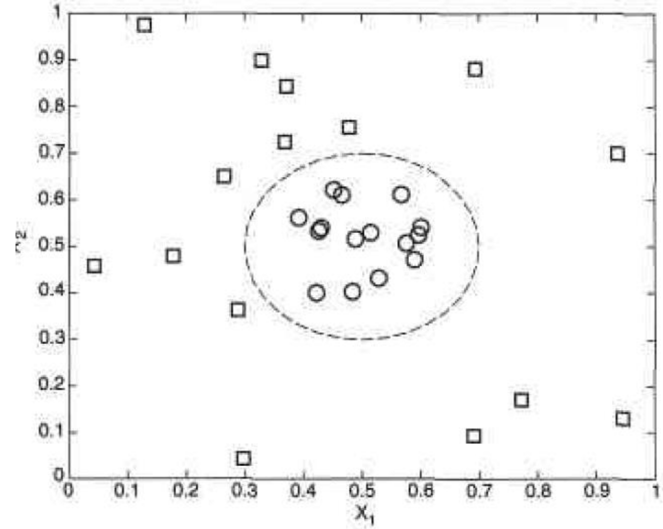
- Data is not separable linearly
- Margin will become inefficient
- Data needs to be transformed from original coordinate space **x** to a new space **Φ(x)**, so that linear decision boundary can be applied
- A non-linear transformation function is needed, like, ex:

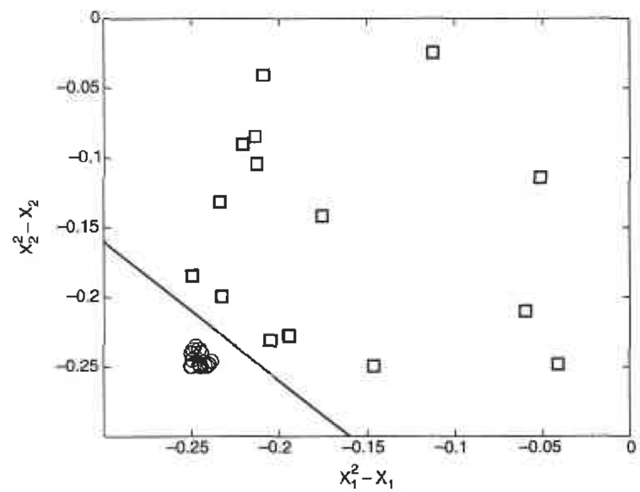
$$\Phi : (x_1, x_2) \longrightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$

- In the transformed space we can choose w = (w_0, w_1, … , w_4) such that

$$w_4 x_1^2 + w_3 x_2^2 + w_2\sqrt{2}x_1 + w_1\sqrt{2}x_2 + w_0 = 0.$$

- The linear decision boundary in the transformed space has the following form: w. **Φ**(x) + b = 0





The "Kernel Trick"

- **The linear classifier relies on dot product between vectors**
  - $\mathbf{x_i}^T \cdot \mathbf{x_j}$
- **If every data point is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the dot product becomes:**

$$K(\mathbf{x_i}, \mathbf{x_j}) = \varphi(\mathbf{x_i})^T \varphi(\underline{\mathbf{x_j}})$$

- **A *kernel function* is some function that corresponds to an inner product in some expanded feature space.**

SVM Kernels

- SVM algorithms use a set of mathematical functions that are defined as the kernel.
- Function of kernel is to take data as input and transform it into the required form.
- Different SVM algorithms use different types of kernel functions. Example *linear, nonlinear, polynomial, and sigmoid etc.*

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

All degree zero terms

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [\underbrace{1}_{\text{All degree zero terms}}, \underbrace{x_1, x_2, \cdots, x_n}_{\text{All degree one terms}}, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [\underbrace{1}_{\text{All degree zero terms}}, \underbrace{x_1, x_2, \cdots, x_n}_{\text{All degree one terms}}, \underbrace{x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1}x_n}_{\text{All degree two terms}}]^T$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

All degree zero terms   All degree one terms   All degree two terms

and compute the dot product $A = \phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

and compute the dot product $A = \phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

and compute the dot product A = $\phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

- Instead, in the original space, compute

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

and compute the dot product A = $\phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

- Instead, in the original space, compute

$$B = K(\mathbf{x}, \mathbf{z}) = \left(1 + \mathbf{x}^T \mathbf{z}\right)^2$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n]^T$$

and compute the dot product $A = \phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

- Instead, in the original space, compute

$$B = K(\mathbf{x}, \mathbf{z}) = \left(1 + \mathbf{x}^T \mathbf{z}\right)^2$$

Claim: A = B (Coefficients do not really matter)

Example: Two dimensions, quadratic kernel

$A = \phi(\mathbf{x})^{T} \phi(\mathbf{z})$ $\qquad$ $B = K(\mathbf{x}, \mathbf{z}) = \left(1 + \mathbf{x}^T \mathbf{z}\right)^2$

$$\phi(x_1, x_2) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$$

The Kernel Trick

Suppose we wish to compute $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$

Here ϕ maps **x** and **z** to a high dimensional space

The Kernel Trick: Save time/space by computing the value of $K(\mathbf{x}, \mathbf{z})$ by performing operations in the original space (without a feature transformation!)

Computing dot products efficiently

Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be operate using vectors in a space of dimensionality $n(n+1)/2$.

The kernel trick allows you to save time/space and compute dot products in an n dimensional space.

(Not just for degree 2 polynomials)

Which functions are kernels?

Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$.
Then, your dot product will be operate using vectors in a space of dimensionality $n(n+1)/2$.

The kernel trick allows you to save time/space and compute dot products in an n dimensional space.

(Not just for degree 2 polynomials)

- Can we use any function K(.,.)?

Which functions are kernels?

Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be operate using vectors in a space of dimensionality $n(n+1)/2$.

The kernel trick allows you to save time/space and compute dot products in an n dimensional space.

(Not just for degree 2 polynomials)

- Can we use any function K(.,.)?
  - No! A function K(x,z) is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.

Which functions are kernels?

Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be operate using vectors in a space of dimensionality $n(n+1)/2$.

The kernel trick allows you to save time/space and compute dot products in an n dimensional space.

(Not just for degree 2 polynomials)

- Can we use any function $K(.,.)$?
  - No! A function $K(x,z)$ is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.

- General condition: construct the Gram matrix $\{K(\mathbf{x}_i, \mathbf{z}_j)\}$; check that it's positive semi definite

The Kernel Matrix

- The Gram matrix of a set of n vectors S = $\{\mathbf{x}_1 \ldots \mathbf{x}_n\}$ is the $n \times n$ matrix $\mathbf{G}$ with $\mathbf{G}_{ij} = \mathbf{x}_i^T \mathbf{x}_j$
  - The kernel matrix is the Gram matrix of $\{\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)\}$
  - (size depends on the # of examples, not dimensionality)

Mercer's condition

Let K(**x**, **z**) be a function that maps two n dimensional vectors to a real number

K is a valid kernel if for every finite set {x_1, x_2, , , }, for any choice of real valued c_1, c_2, . . . , we have

$$\sum_i \sum_j c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

Polynomial kernels

- Linear kernel: $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$

- Polynomial kernel of degree *d*: $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z})^d$
  – only *d* th-order interactions

- Polynomial kernel up to degree *d*: $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d$ $(c>0)$
  – all interactions of order *d* or lower

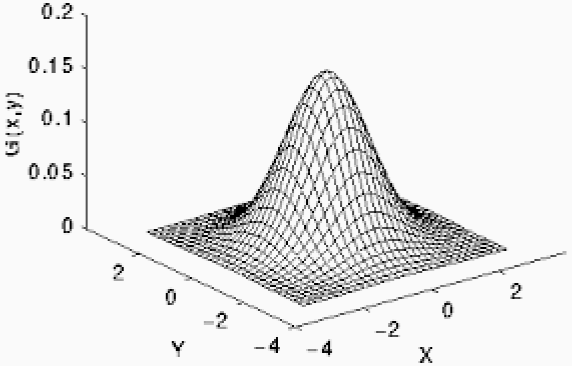
Gaussian Kernel

(or the radial basis function kernel)

$$K_{rbf}(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{z}||^2}{c}\right)$$

- $(x - z)^2$: squared Euclidean distance between **x** and **z**
- $c = \sigma^2$: a free parameter
- very small c: K ≈ identity matrix (every item is different)
- very large c: K ≈ unit matrix (all items are the same)

- k(**x**, **z**) ≈ 1 when **x**, **z** close
- k(**x**, **z**) ≈ 0 when **x**, **z** dissimilar



Constructing New Kernels

You can construct new kernels $k'(\mathbf{x}, \mathbf{x'})$ from existing ones:

- Multiplying $k(\mathbf{x}, \mathbf{x'})$ by a constant c

  $ck(\mathbf{x}, \mathbf{x'})$

- Multiplying $k(\mathbf{x}, \mathbf{x'})$ by a function f applied to **x** and **x'**

  $f(\mathbf{x})k(\mathbf{x}, \mathbf{x'})f(\mathbf{x'})$

- Applying a polynomial (with non-negative coefficients) to $k(\mathbf{x}, \mathbf{x'})$

  $P(\ k(\mathbf{x}, \mathbf{x'})\ )$ with $P(z) = \sum_i a_i z^i$ and $a_i \geq 0$

- Exponentiating $k(\mathbf{x}, \mathbf{x'})$

  $\exp(k(\mathbf{x}, \mathbf{x'}))$

Constructing New Kernels (2)

- You can construct $k'(\mathbf{x}, \mathbf{x'})$ from $k_1(\mathbf{x}, \mathbf{x'})$, $k_2(\mathbf{x}, \mathbf{x'})$ by:
  - Adding $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k_1(\mathbf{x}, \mathbf{x'}) + k_2(\mathbf{x}, \mathbf{x'})$
  - Multiplying $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k_1(\mathbf{x}, \mathbf{x'})k_2(\mathbf{x}, \mathbf{x'})$

Name	Function	Type problem
Polynomial Kernel	$(x_i^t x_j + 1)^q$ q is degree of polynomial	Best for Image processing
Sigmoid Kernel	$\tanh(a x_i^t x_j + k)$ k is offset value	Very similar to neural network
Gaussian Kernel	$\exp(\lVert x_i - x_j \rVert^2 / 2\sigma^2)$	No prior knowledge on data
Linear Kernel	$\left(1 + x_i x_j min(x_i, x_j) - \frac{(x_i + x_j)}{2} min(x_i, x_j)^2 + \frac{min(x_i, x_j)^3}{3}\right)$	Text Classification
Laplace Radial Basis Function (RBF)	$\left(e^{(-\lambda \lVert x_i - x_j \rVert)}, \lambda >= 0\right)$	No prior knowledge on data

There are many more kernel functions.

Non-linear SVM using kernel

1. Select a kernel function.
2. Compute pairwise kernel values between labeled examples.
3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.

Nonlinear SVM - Overview

- SVM locates a separating hyperplane in the feature space and classify points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the dot product in the feature space.

Multi-Class Problem

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

Multi-Class Problem

Instead of just two classes, we now have C classes

- E.g. predict which movie genre a viewer likes best
- Possible answers: action, drama, indie, thriller, etc.

Two approaches:

- One-vs-all
- One-vs-one

Multi-Class Problem

One-vs-all (a.k.a. one-vs-others)

- Train C classifiers
- In each, pos = data from class *i*, neg = data from classes other than *i*
- The class with the most confident prediction wins
- Example:
  - You have 4 classes, train 4 classifiers
  - 1 vs others: score 3.5
  - 2 vs others: score 6.2
  - 3 vs others: score 1.4
  - 4 vs other: score 5.5
  - Final prediction: class 2
- Issues?

Multi-Class Problem

One-vs-one (a.k.a. all-vs-all)

- Train C(C-1)/2 binary classifiers (all pairs of classes)
- They all vote for the label
- Example:
  - You have 4 classes, then train 6 classifiers
  - 1 vs 2, 1 vs 3, 1 vs 4, 2 vs 3, 2 vs 4, 3 vs 4
  - Votes: 1, 1, 4, 2, 4, 4
  - Final prediction is class 4

SVM versus Logistic Regression

- When viewed from the point of view of regularized empirical loss minimization, SVM and logistic regression appear quite similar:

SVM: $$\sum_{i=1}^{n} \left(1 - y_i \left[w_0 + \mathbf{x}_i^T \mathbf{w}_1\right]\right)^+ + \|\mathbf{w}_1\|^2/2$$

Logistic: $$\sum_{i=1}^{n} \overbrace{-\log \sigma\left(y_i \left[w_0 + \mathbf{x}_i^T \mathbf{w}_1\right]\right)}^{-\log P(y_i|\mathbf{x},\mathbf{w})} + \|\mathbf{w}_1\|^2/2$$

where $\sigma(z) = (1 + \exp(-z))^{-1}$ is the logistic function.

SVM versus Logistic Regression

- The difference comes from how we penalize "errors":

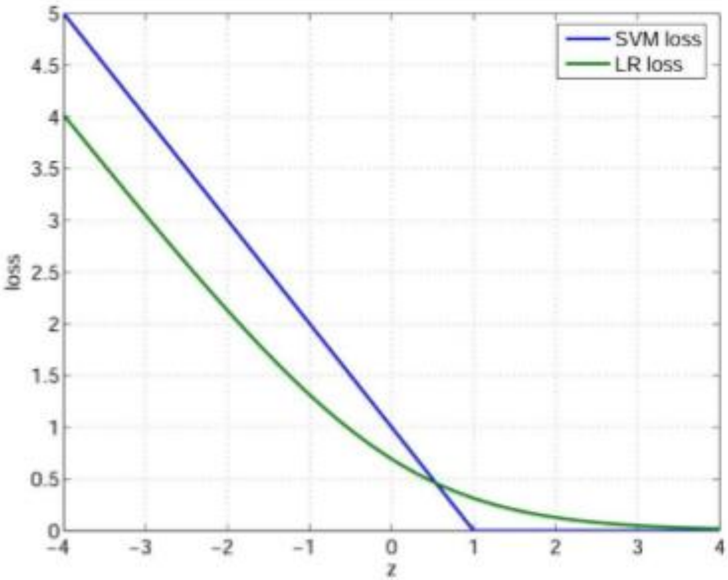
$$\text{Both:} \qquad \sum_{i=1}^{n} \text{Loss}\Big(\overbrace{y_i\left[w_0 + \mathbf{x}_i^T \mathbf{w}_1\right]}^{z}\Big) + \quad \|\mathbf{w}_1\|^2/2$$

- SVM:

$$\text{Loss}(z) = (1 - z)^+$$

- Regularized logistic reg:

$$\text{Loss}(z) = \log(1 + \exp(-z))$$

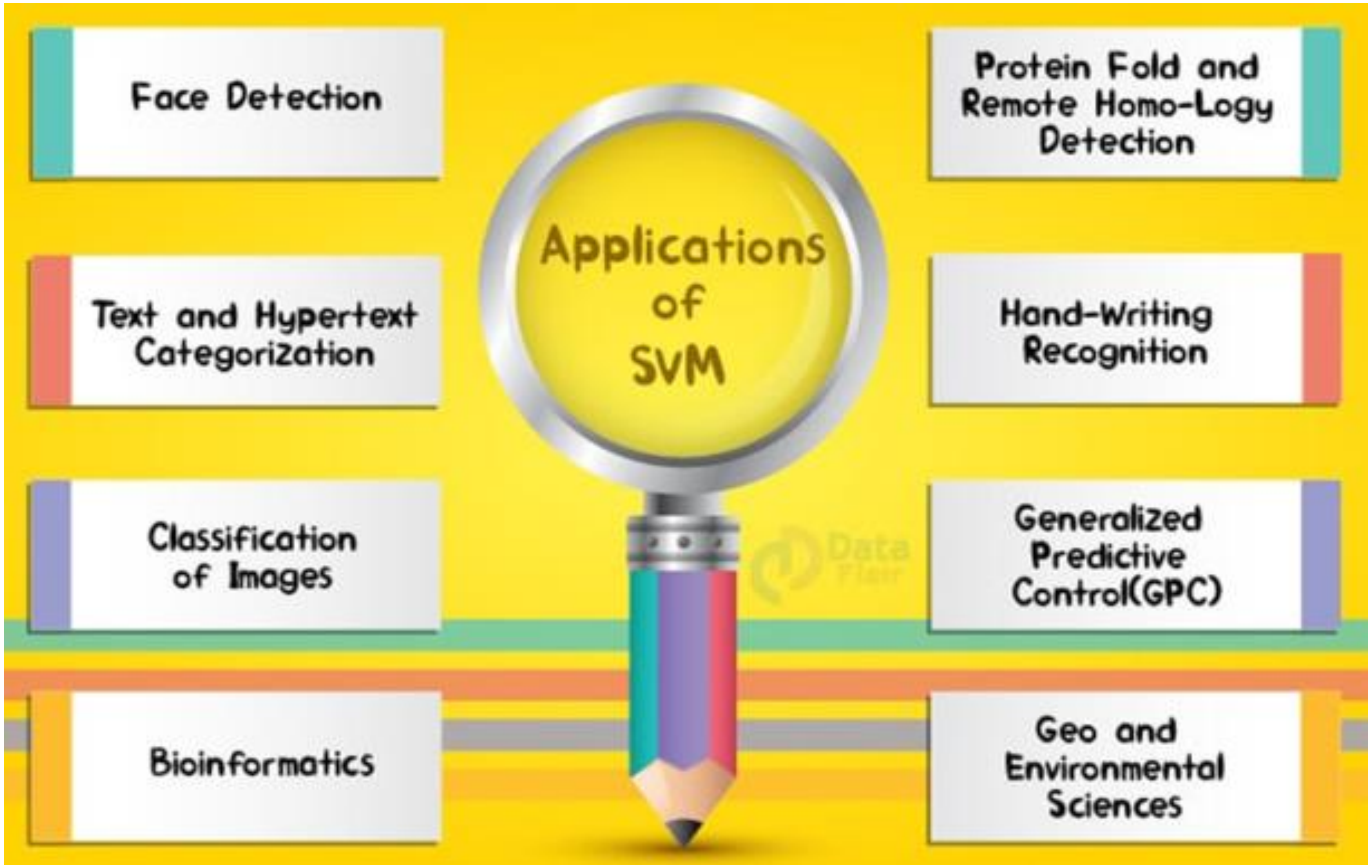


Properties of SVM

- Flexibility in choosing a similarity function
- Sparseness of solution when dealing with large data sets
  - Only support vectors are used to specify the separating hyperplane
  - Therefore SVM also called sparse kernel machine.
- Ability to handle large feature spaces
  - complexity does not depend on the dimensionality of the feature space
- Overfitting can be controlled by soft margin approach
- Nice math property: a simple convex optimization problem which is guaranteed to converge to a single global solution
- Feature Selection

SVM Applications

SVM has been used successfully in many real-world problems



Application : Text Categorization

- Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content.

  A document can be assigned to more than one category, so this can be viewed as a series of binary classification problems, one for each category

Text Categorization using SVM

- The distance between two documents is $\phi(x)\cdot\phi(z)$

- $K(x,z) = \phi(x)\cdot\phi(z)$ is a valid kernel, SVM can be used with $K(x,z)$ for discrimination.

- Why SVM?

  -High dimensional input space

  -Few irrelevant features (dense concept)

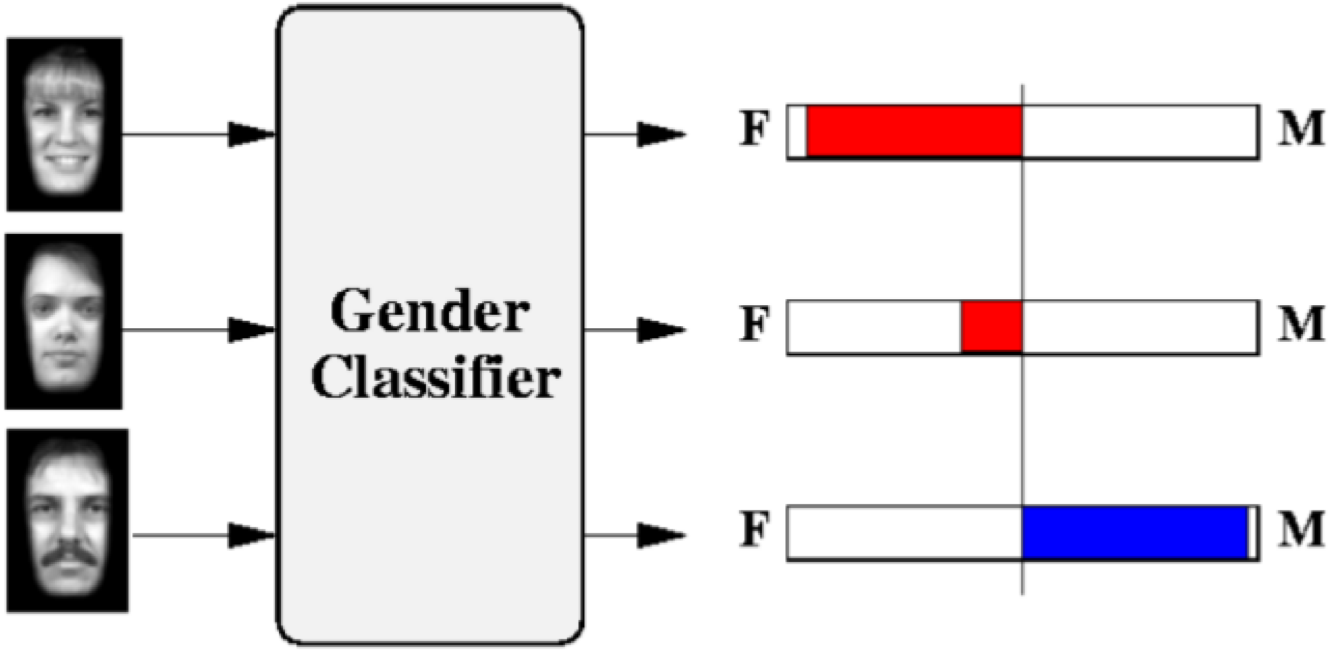
  -Sparse document vectors (sparse instances)

  -Text categorization problems are linearly separable

Using SVM

1. Select a kernel function.
2. Compute pairwise kernel values between labeled examples.
3. Use this “kernel matrix” to solve for SVM support vectors & alpha weights.
4. To classify a new example: compute kernel values between new input and support vectors, apply alpha weights, check sign of output.

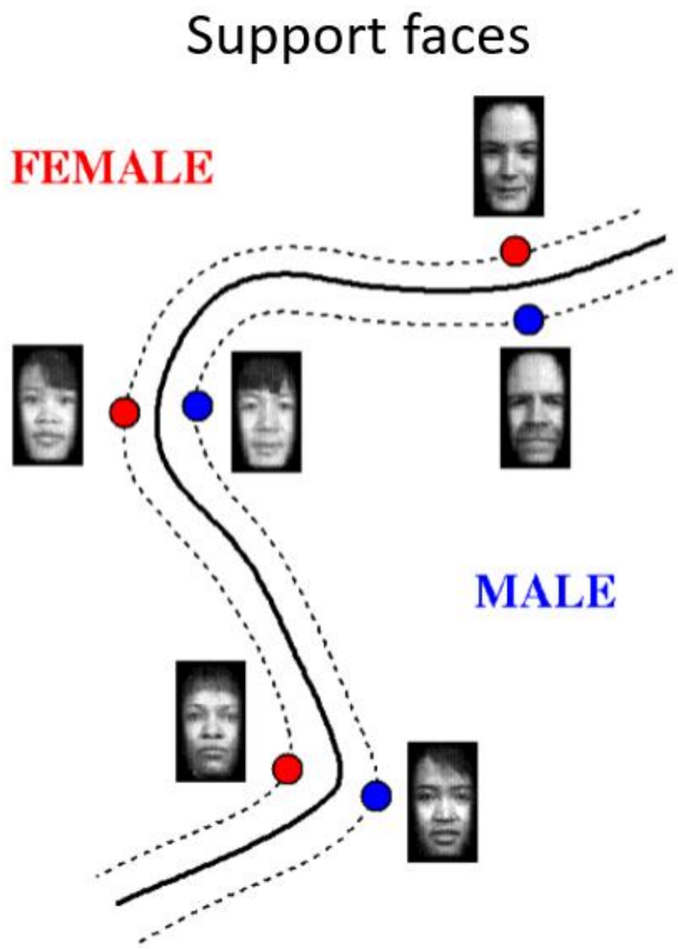
Learning Gender from image with SVM



Moghaddam and Yang, Learning Gender with Support Faces, TPAMI 2002

Moghaddam and Yang, Face & Gesture 2000

Support faces



Accuracy of SVM Classifier

Figure 6. SVM vs. Human performance

- SVMs performed better than humans, at either resolution

Some Issues

- **Sensitive to noise**

  - A relatively small number of mislabeled examples can dramatically decrease the performance

- **Choice of kernel**

  - Gaussian or polynomial kernel is default

  - if ineffective, more elaborate kernels are needed

  - domain experts can give assistance in formulating appropriate similarity measures

- **Choice of kernel parameters**

  - e.g. σ in Gaussian kernel

  - σ is the distance between closest points with different classifications

  - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

- **Optimization criterion** – Hard margin v.s. Soft margin

  - a lengthy series of experiments in which various parameters are tested

Reference

- **Support Vector Machine Classification of Microarray Gene Expression Data**, Michael P. S. Brown William Noble Grundy, David Lin, Nello Cristianini, Charles Sugnet, Manuel Ares, Jr., David Haussler
- **Text categorization with Support Vector Machines: learning with many relevant features**

  T. Joachims, ECML - 98
- Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
- **A Tutorial on Support Vector Machines for Pattern Recognition**, Kluwer Academic Publishers - Christopher J.C. Burges

Good Web References for SVM

- http://www.cs.utexas.edu/users/mooney/cs391L/
- https://www.coursera.org/learn/machine-learning/home/week/7
- https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
- https://data-flair.training/blogs/svm-kernel-functions/
- MIT 6.034 Artificial Intelligence, Fall 2010
- https://stats.stackexchange.com/questions/30042/neural-networks-vs-support-vector-machines-are-the-second-definitely-superior
- https://www.sciencedirect.com/science/article/abs/pii/S0893608006002796
- https://medium.com/deep-math-machine-learning-ai/chapter-3-support-vector-machine-with-math-47d6193c82be
- Radial basis kernel
- http://www.engr.mun.ca/~baxter/Publications/LagrangeForSVMs.pdf

Thank You