COMP3308/3608, Lecture 10a

ARTIFICIAL INTELLIGENCE

Support Vector Machines

Reference: Russell and Norvig, pp. 744-748

Witten, Frank, Hall and Pal, ch.7.2, pp. 224-227

Outline

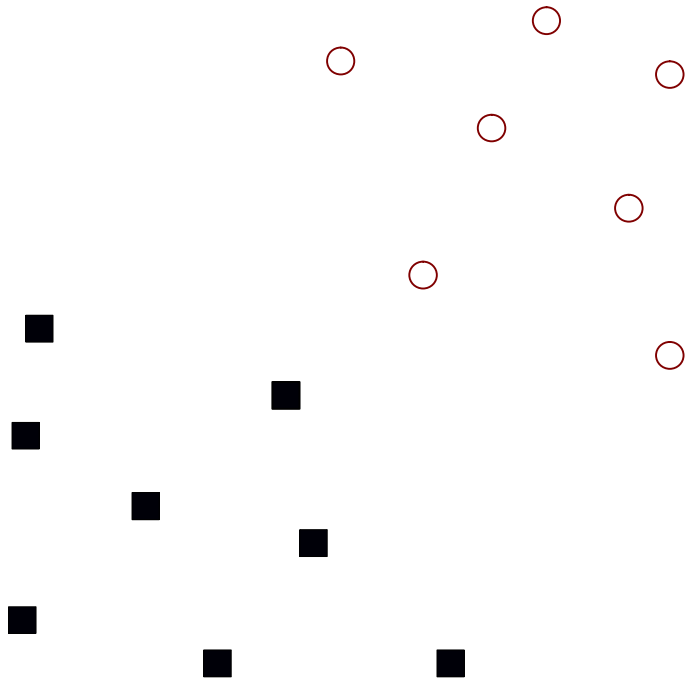
- **Maximum margin hyperplane**
- **Linear SVM with hard margin**
- **Linear SVM with soft margin**
- **Nonlinear SVM**

Support Vector Machines (SVM)

- **Very popular classification method**
- **Rooted in statistical learning theory**
- **Advantages**
  - **Can form arbitrary decision boundaries (both linear and non-linear)**
  - **The decision boundary is a maximum margin hyperplane (i.e. has the highest possible distance to the training examples from the 2 classes) and this helps to generalize well on new examples**
  - **The decision boundary is defined by a sub-set of the training vectors called support vectors => resistant to overfitting**

Separation by a Hyperplane

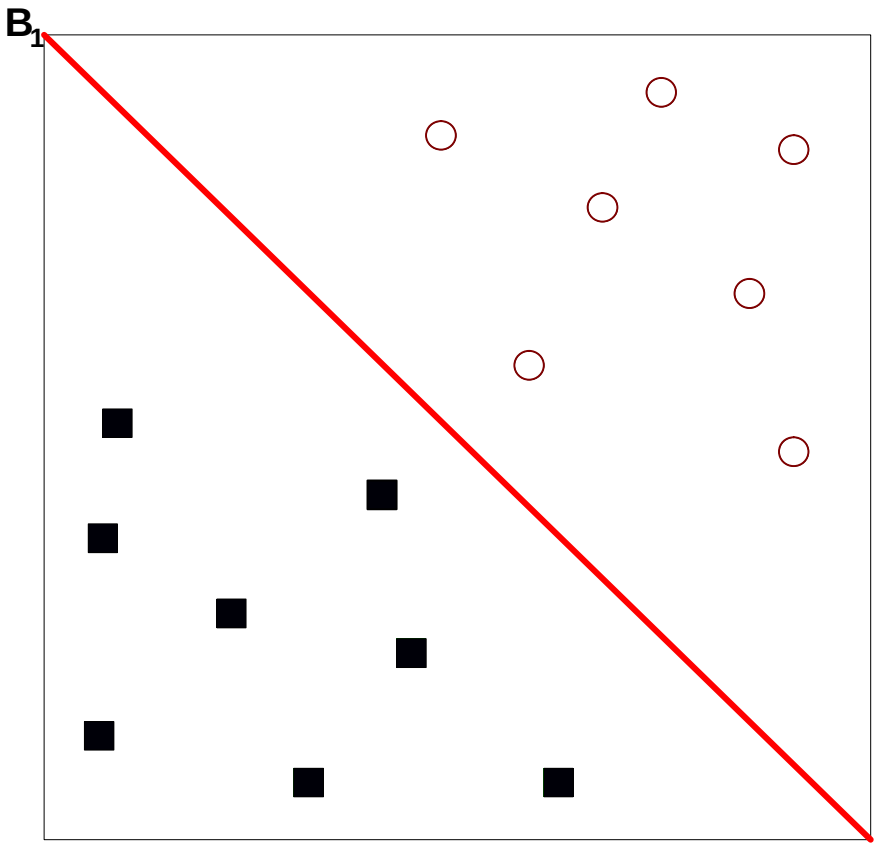
- **Given is the following training data (2 class problem: squares and circles):**

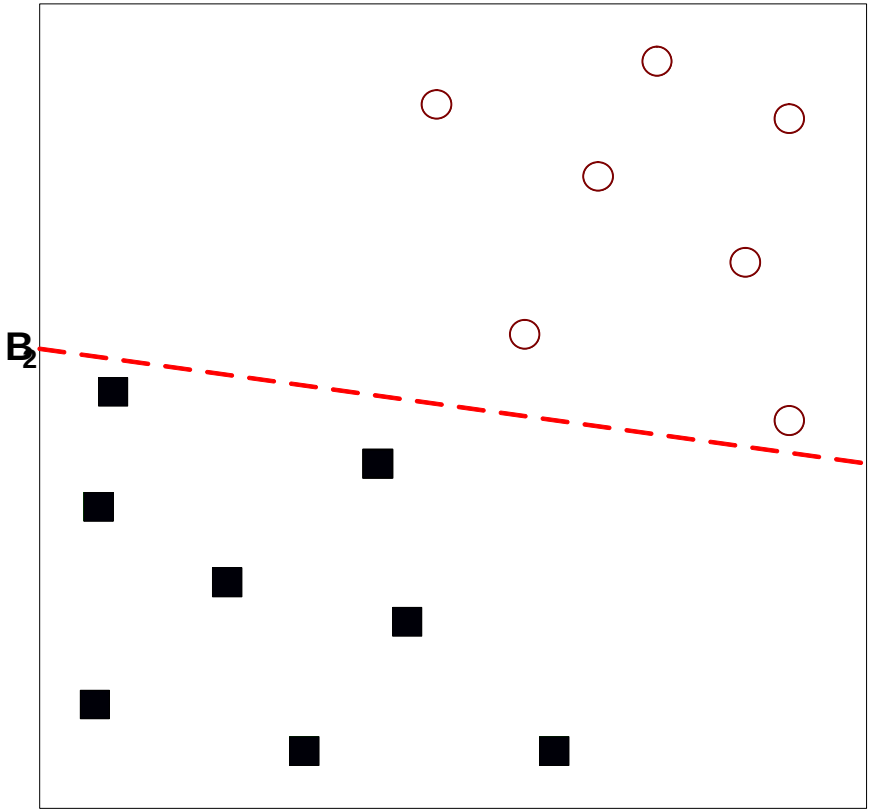


- **Find a linear boundary (line, hyperplane) that will separate the data**
- **Is the data linearly separable?**
  - **Recall: Linearly separable = there exist a line (hyperplane) such that all squares are on one side and all circles are on the other side**

Separation by a Hyperplane (2)

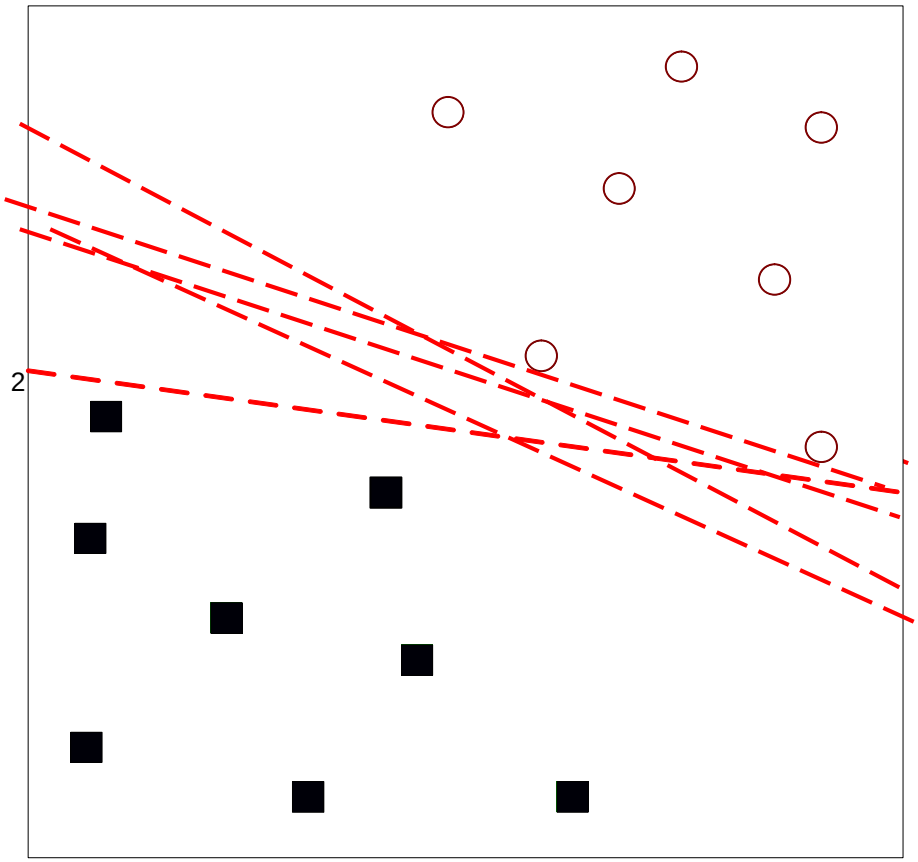
- **1 possible decision boundary** **another one**





Separation by a Hyperplane (3)

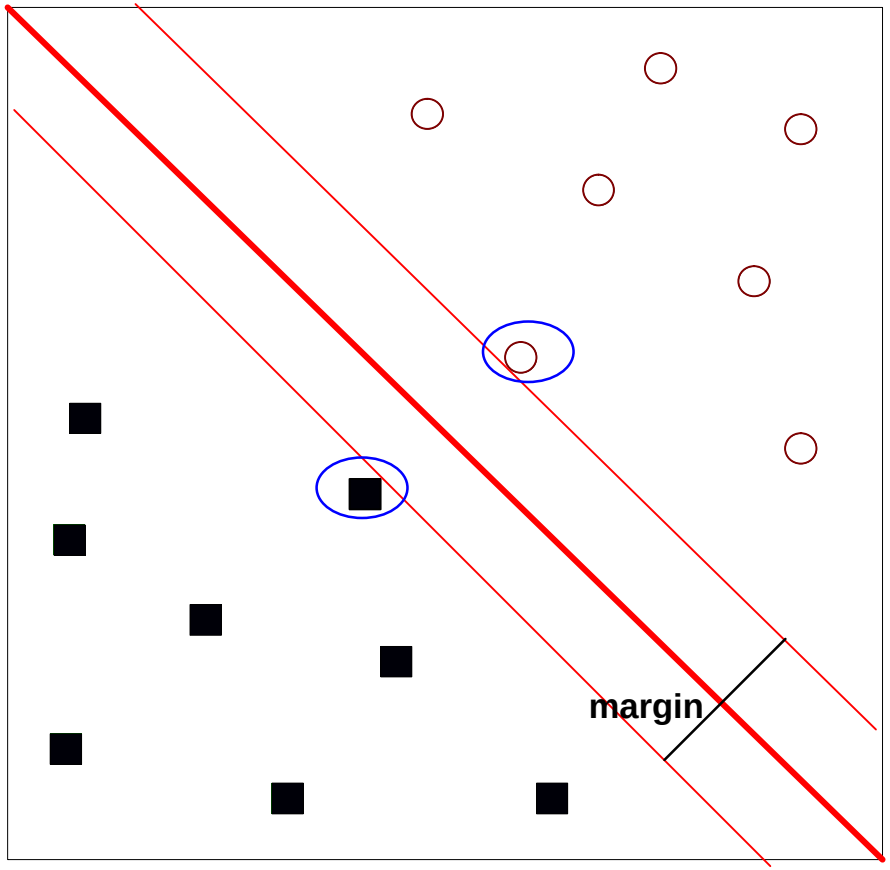
- **Other possible decision boundaries – there are many lines**



Margin of a Hyperplane and Support Vectors

- **A decision boundary B1:**

B1

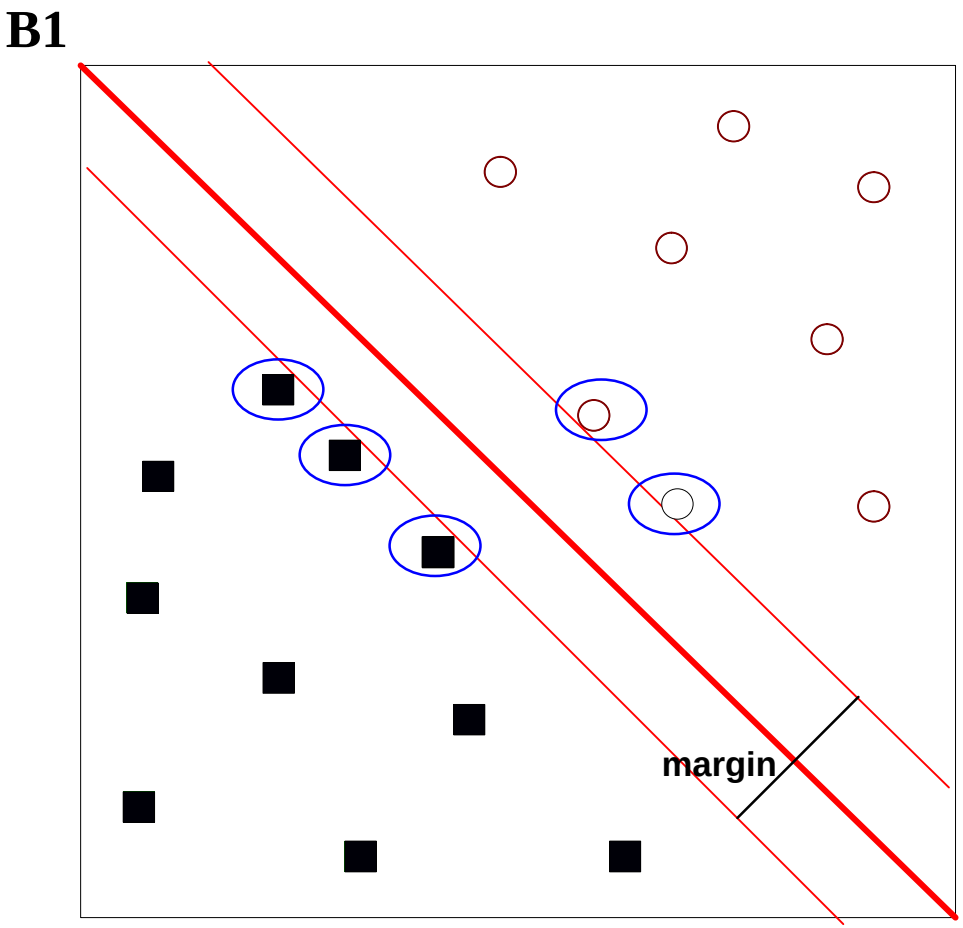


Support vectors **are the examples (data points) that lie closest to the decision boundary; they are circled**

Margin **– the separation between the boundary and the closest examples (or the width the boundary can be increased before touching an example)**

The boundary is in the middle of the margin

Support Vectors

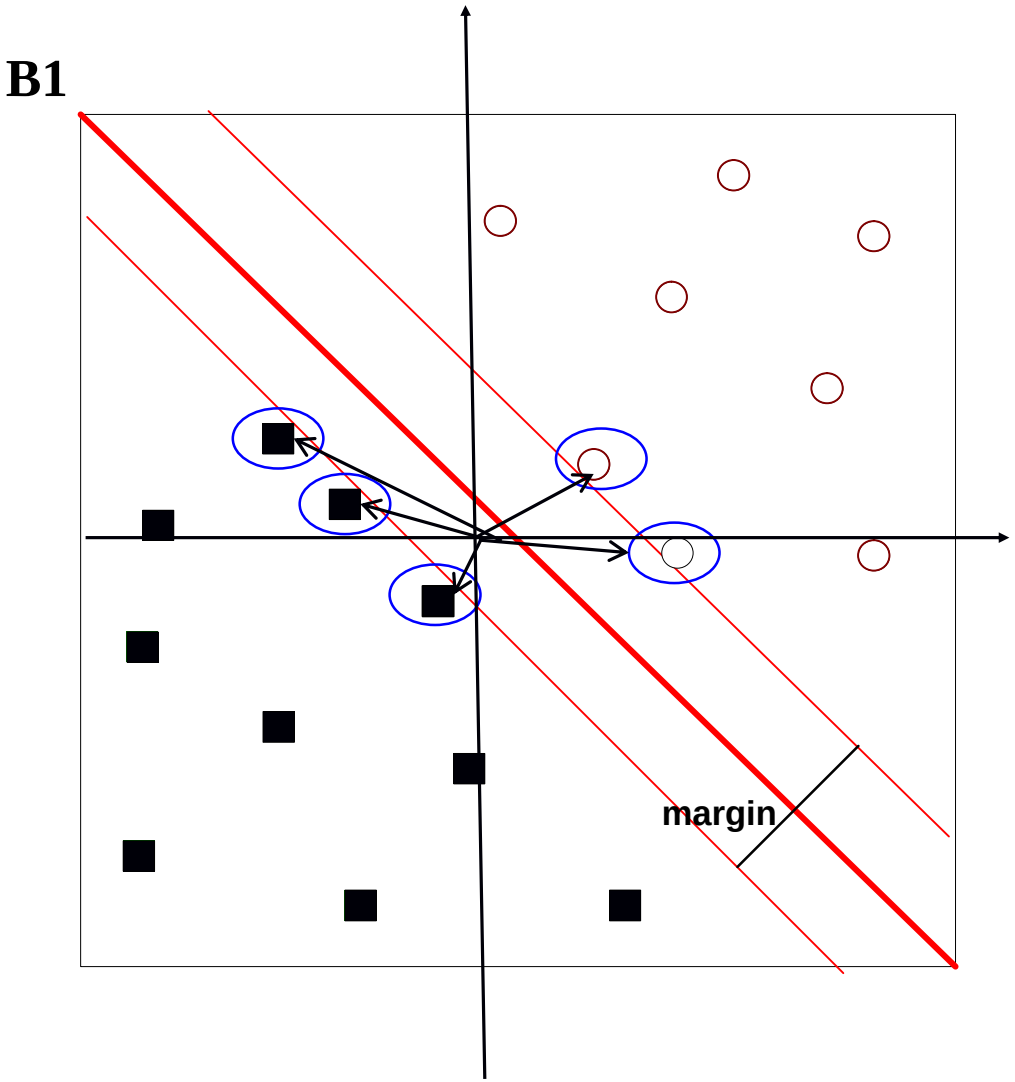


The support vectors just touch the margin of the decision boundary

It is possible to have more than 1 support vector for each class

For our example: 5 support vectors, 3 for class square and 2 for class circle

Support Vectors (2)



Why "vectors"?

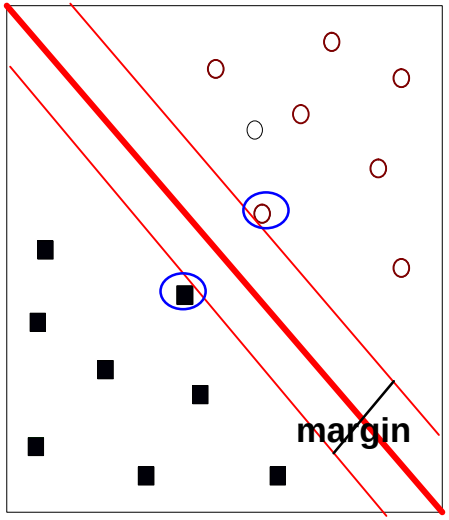
On the previous slides we saw only the tips of the support vectors; here we can see the actual vectors

Remember that the given training examples are vectors (input vectors) where each dimension corresponds to 1 feature

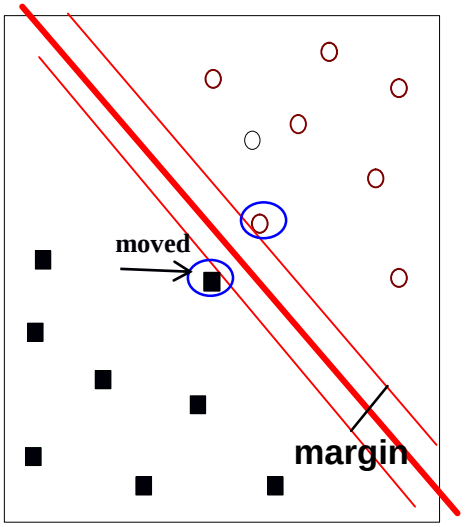
The support vectors are a subset of the input vectors => they are also vectors

Support Vectors (3)

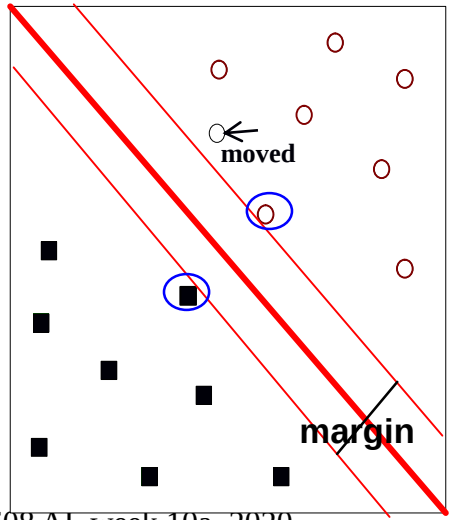
- **The support vectors define the decision boundary (they are the closest examples to the decision boundary by definition)**



If we move a support vector, the decision boundary will change

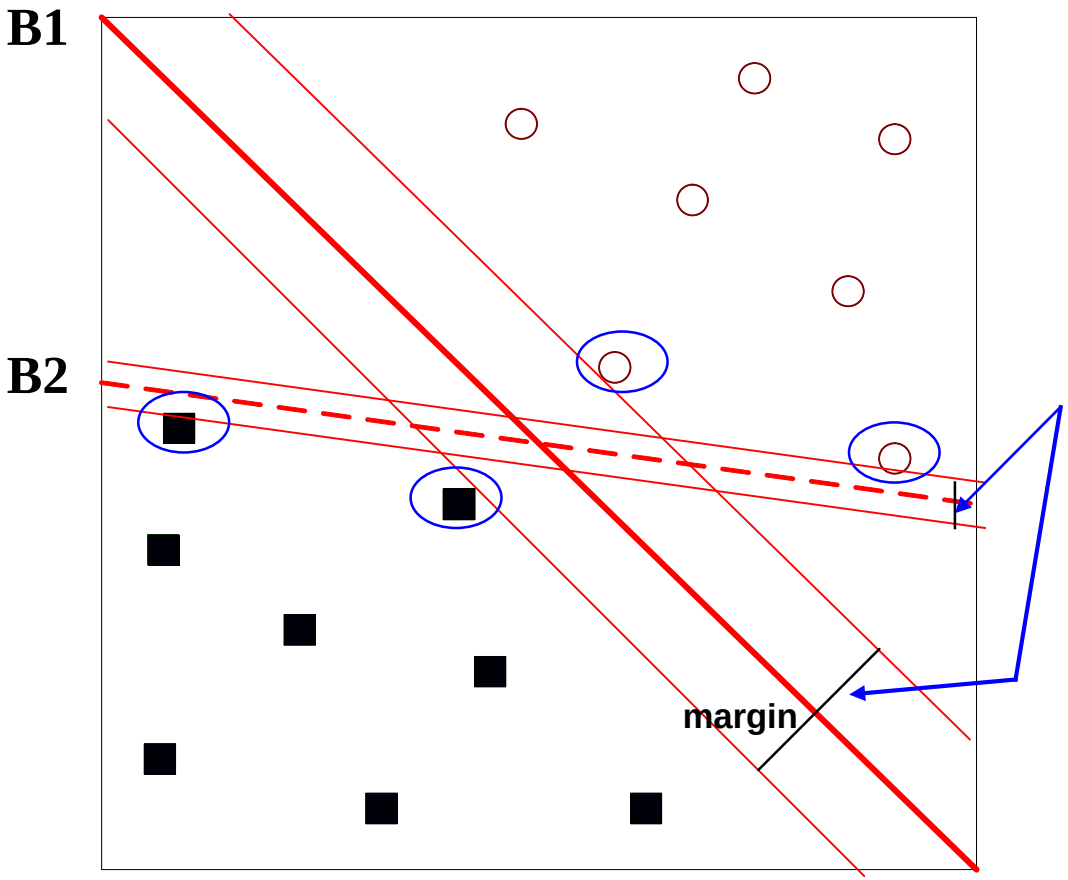


If we move another input example, that is <u>not</u> a support vector, the decision boundary will <u>not</u> change



Which Hyperplane is Better?

- **Which hyperplane (B1 or B2) should we select? Which one is likely to classify more accurately new data?**



- **The hyperplane with the bigger margin, B1**

Maximum Margin Hyperplane

- **The hyperplane (separator) with the biggest margin is called the *maximum margin hyperplane (separator)***
  - **It is the hyperplane with the highest possible distance to the training examples**
  - **SVM selects the maximum margin hyperplane**
- **Decision boundaries with large margins (i.e. farthest away from the training data) are better than those with small margins as they typically are more accurate on new examples**

- **Intuitive justification – feels safer**
  - **We don't know where the new examples will be but we assume that they are drawn from the same distribution as the training examples**
  - **If we have made a small error in the location of the boundary, the chances of causing misclassifications will be smaller if the margin is big**
  - **=> big margin is less sensitive to noise and overfitting**

Maximum Margin – Formal Justification

- **Formal justification from computational learning theory: structural risk minimization principle**
  - **The generalization error (the error on new examples) depends on the training error and model complexity (also called model capacity)**
  - **If the training error is the same, the classifier with the lower complexity will generalize better**

  - **small margin=> high complexity=>higher generalization error**
  - **big margin => low complexity => lower generalization error**

Linear Decision Boundary - Revision

- **Binary classification problem, N training examples (input vectors)**
- **x is the input vector, y is the class value**

$$(\mathbf{x}_i, y_i), i = 1,.., N$$

$$\mathbf{x}_i = (x_{i1},..., x_{im})^T, y_i = \{-1,1\}$$

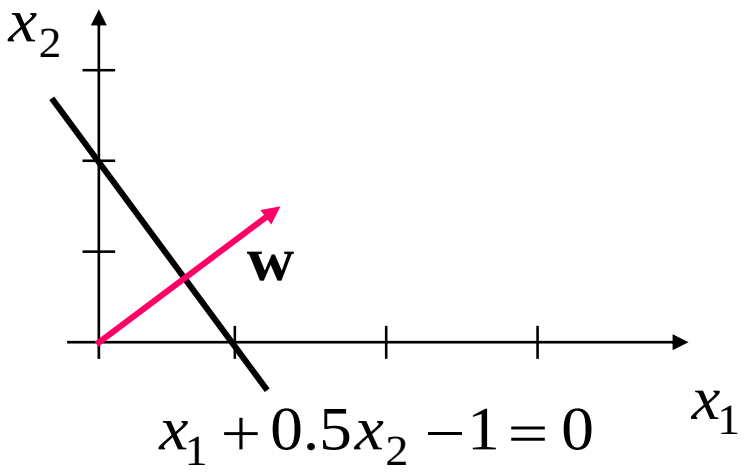
Training data example:

$$\mathbf{x}_1 = (2,2), y_1 = 1$$

$$\mathbf{x}_2 = (0.5,1), y_2 = -1, etc.$$

- **A decision boundary of a linear classifier is** $\mathbf{w} \cdot \mathbf{x} + b = 0$

parameters

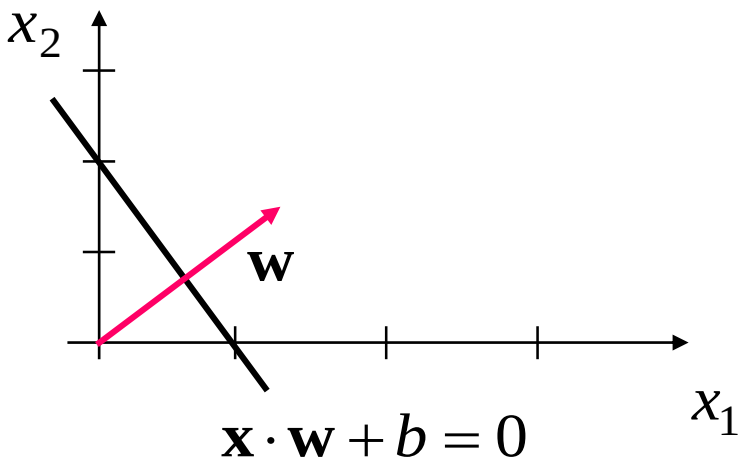


How do we Classify New Examples?

$$if\ \mathbf{x}\ is\ above\ the\ decision\ boundary : \mathbf{w}\cdot\mathbf{x}+b>0$$

$$if\ \mathbf{x}\ is\ below\ the\ decision\ boundary : \mathbf{w}\cdot\mathbf{x}+b<0$$

$$\textbf{i.e.} = sign(\mathbf{w}\cdot\mathbf{x}+b)$$



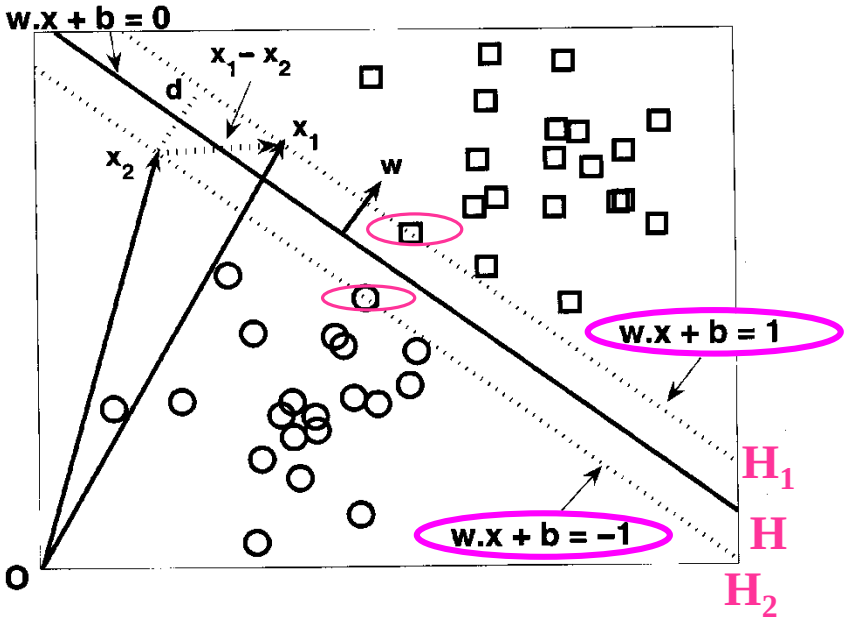
- **If we know the decision boundary, we can easily classify a new example x by calculating f=wx+b and determining the sign**

Problem Statement for SVM - Definitions

- **Our separating hyperplane is H**
- **H is in the middle of 2 other hyperplanes, H1 and H2, defined as:**

$$H_1 : \mathbf{w} \cdot \mathbf{x} + b = 1$$

$$H2 : \mathbf{w} \cdot \mathbf{x} + b = -1$$



- **The points laying on H1 and H2 are the support vectors**
- **d is the margin of H**
- **It can be shown that:** $d = \frac{2}{\|\mathbf{w}\|}$
- **(How? By calculating the distance between a point from H and the hyperplane H1 (1/(||w||), then *2)**

- **To <u>maximize</u> the margin d, we need to <u>minimize</u> ||w||**
- **This is equivalent to <u>minimizing</u> the quadratic function:** $\frac{1}{2}\|\mathbf{w}\|^2$

Learning a Maximum Margin Hyperplane in Linear SVM

- **Given: a set of labelled training examples**
- **Learn: the maximum margin hyperplane such as all training examples are classified correctly**
- **This could be formulated as a constraint optimization problem:**
  - **Given N training examples**

$$(\mathbf{x}_i, y_i), i = 1,.., N$$

$$\mathbf{x}_i = (x_{i1},..., x_{im})^T, y_i = \{-1,1\}$$

training vector **class**

  - **Minimize** $\frac{1}{2}\|\mathbf{w}\|^2$ **Maximazing the margin**
  - **Subject to the linear constraint**

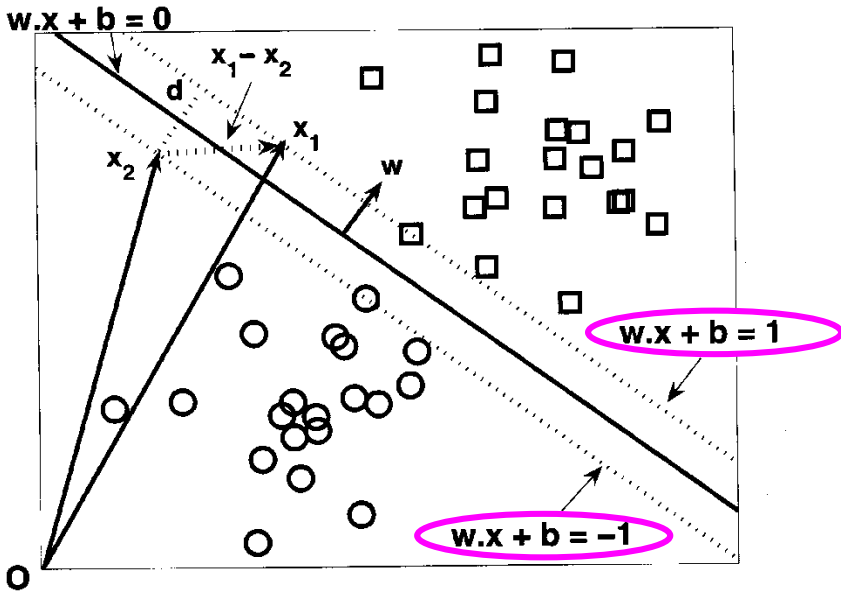
i.e. i=1..N

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i$$

Another way of expressing correct classification of all training examples

More on the Linear Constraint

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1, \forall i \quad (1)$$



- **It is a combination of 2 conditions:**
  - **All training examples from class y=1 (squares) must be above the hyperplane** $H_1 : \mathbf{w}\cdot\mathbf{x} + b = 1$

    $\mathbf{w}\cdot\mathbf{x}_i + b \geq 1 \quad if\ y_i = 1 \quad (2)$
  - **All training examples from class y=-1 (circles) must be below the hyperplane** $H_2 : \mathbf{w}\cdot\mathbf{x} + b = -1$

    $\mathbf{w}\cdot\mathbf{x}_i + b \leq -1 \quad if\ y_i = -1 \quad (3)$

(2) and (3) together can be combined in (1) (a more compact form)

The Optimization Problem

- **The optimization problem can be transformed into its dual**

$$\max \mathbf{w}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

Dot product of pairs of training vectors

Target value (class value) of the training vectors

$$subject\ to\ \lambda_i \geq 0, \sum_{i=1}^{N} \lambda_i y_i = 0$$

- **This is a Quadratic Programming (QP) problem and can be solved (i.e. the λs can be estimated) using a standard numerical procedure**
  - **Global maximum of the λs always exist**
  - **λs are called Lagrange multipliers**
  - **w (the solution, i.e. the optimal decision boundary) is given by**

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

Characteristics of the Solution

- **Many of the λs are 0**
  - **The optimal decision boundary w is a linear combination (coefficient λ * target value*training vector) of a small number of training examples**
  - **The training examples xi with non-zero λi are the *support vectors* and they are the examples closest to the decision boundary w**
  - **=> the optimal decision boundary w is a linear combination of support vectors**

- **To classify a new example z:**

$$f = \mathbf{w} \cdot \mathbf{z} + b = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{z} + b$$

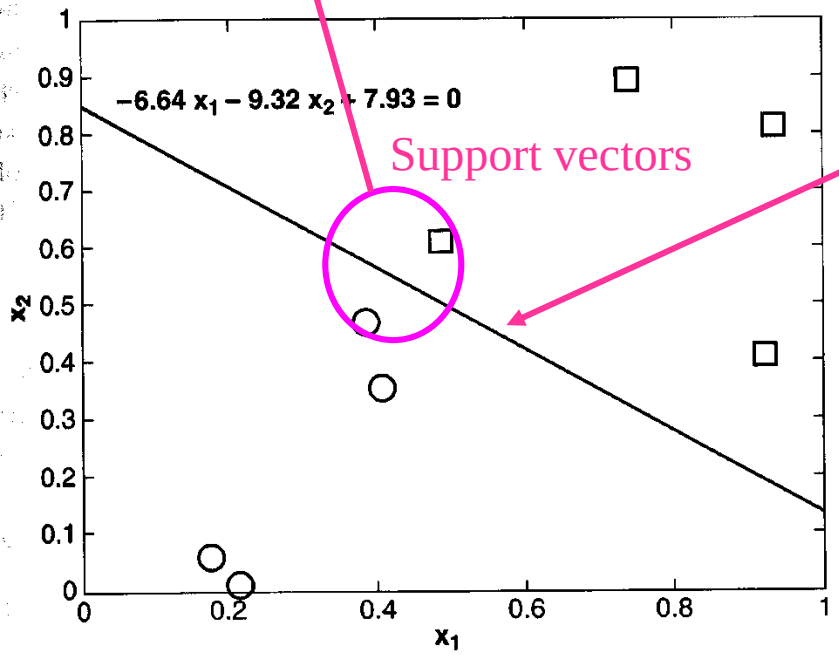
Dot product of the new vector and the support vectors

$sign(f)$ **i.e. the new example belongs to class 1, if f>0 or class -1 if f<0**

Example

features class

	x_1	x_2	y	Lagrange Multiplier
x1	0.3858	0.4687	1	65.5261
x2	0.4871	0.611	–1	65.5261
	0.9218	0.4103	–1	0
	0.7382	0.8936	–1	0
	0.1763	0.0579	1	0
	0.4057	0.3529	1	0
	0.9355	0.8132	–1	0
	0.2146	0.0099	1	0



- **8 2-dim. training examples; 2 classes: -1,1**
- **After solving the problem with QP we find the λs and only 2 of them are non-zero and they correspond to the support vectors for the data (x1 & x2)**
- **Using the λs , the weights (defining the decision boundary are):**

$$w_1 = \sum_{i=1}^{2} \lambda_i y_i \mathbf{x}_{i1} = 65.5261(1*0.3858 - 1*0.4871) = -6.64$$

$$w_2 = \sum_{i=1}^{2} \lambda_i y_i \mathbf{x}_{i2} = 65.5261(1*0.4687 - 1*0.611) = -9.32$$

$b = 7.93$ //there is a formula for b (not shown)

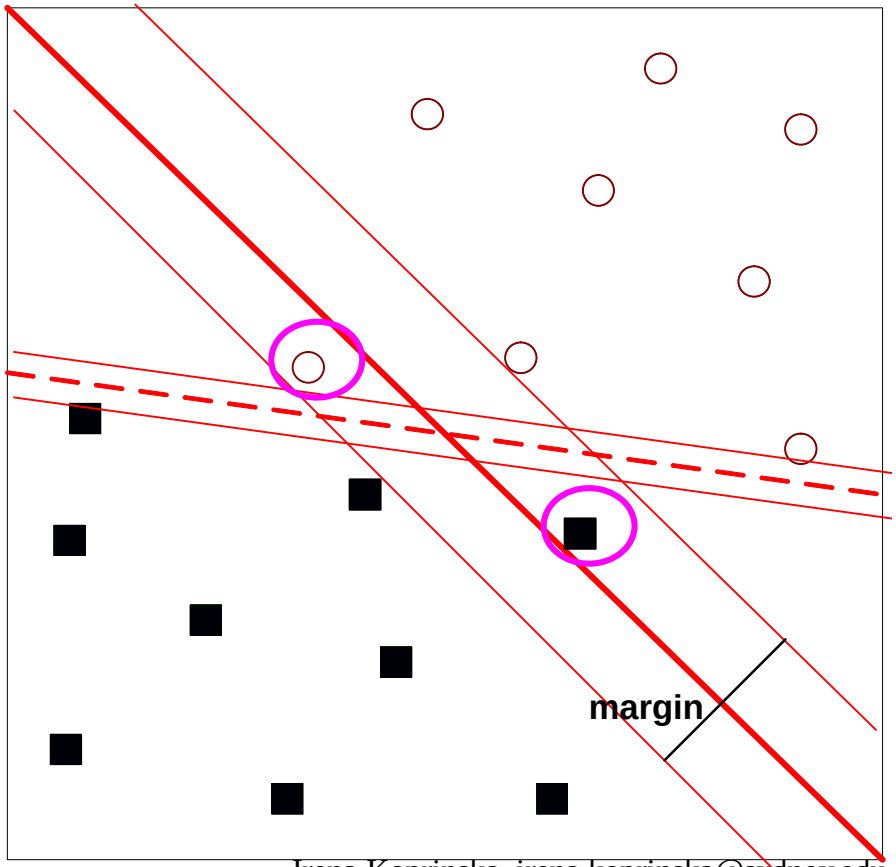
- **Classifying new examples:**
  - **above the decision boundary: class 1**
  - **below: class -1**

Soft Margin

- The method we discussed so far constructs decision boundaries that are free of misclassifications. We can allow some misclassifications.
- Ex.: B1 is better than B2 as its margin is bigger but now 2 new examples are added; B1 misclassifies them but B2 still classifies them correctly

B1

B2



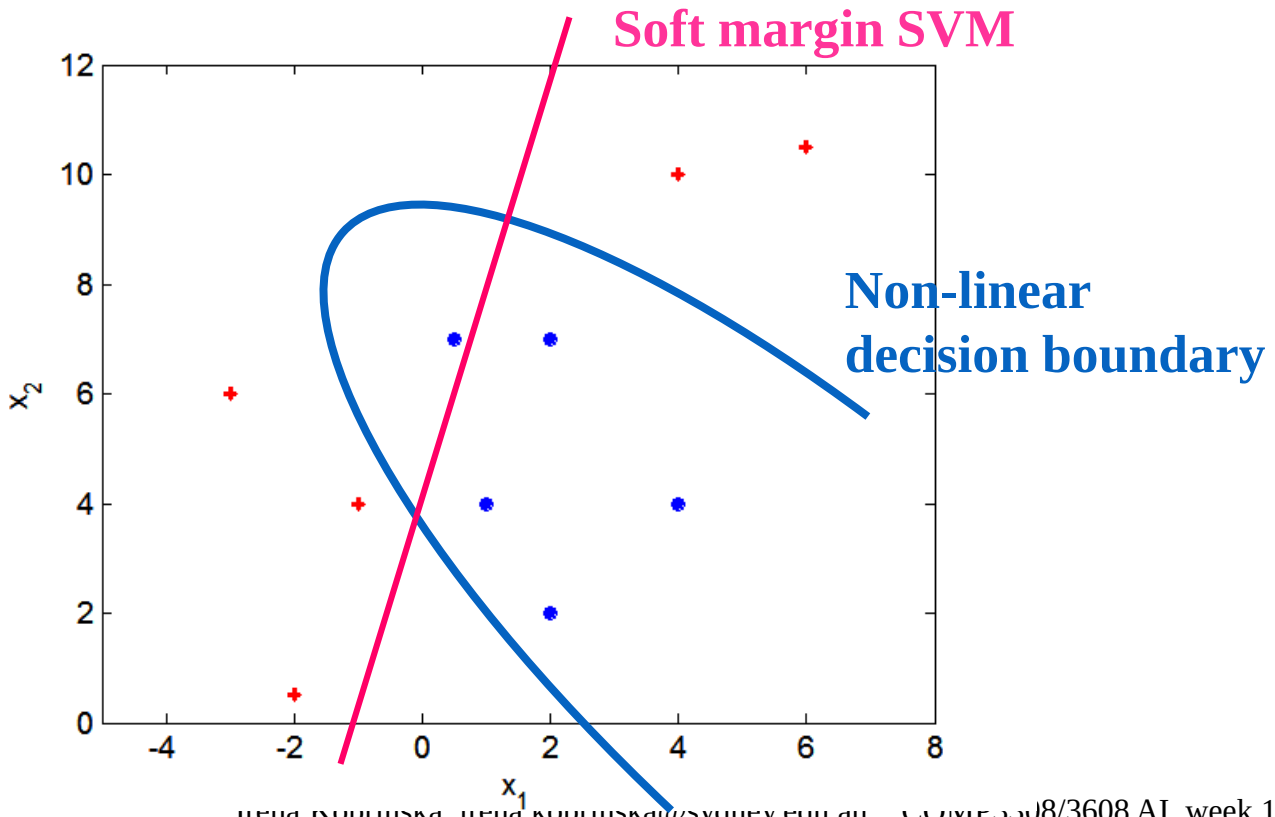
- This does not mean that B2 is better as the new examples may be noise
- B1 should still be preferred as it has wider margin and is less sensitive to overfitting and noise
- We can modify our method to allow some misclassifications, i.e. by considering the trade-off between the margin width and the number of misclassifications
- As a result, the modified method will construct linear boundary even if the data is not linearly separable

Soft Margin Solution

- **The optimisation problem formulation is similar but there is an additional parameter C in the function we want to maximize corresponding to the trade-off between error and margin**
- **The solution is the same as in the hard margin case but there is an upper bound C on the values of the λ_s**

Non-linear SVM

- **In practice most problems are linearly non-separable**
- **SVM with soft margin will find a linear boundary that optimizes the trade-off between the errors on training data and the margin**
- **SVM can further be extended to find non-linear boundary**



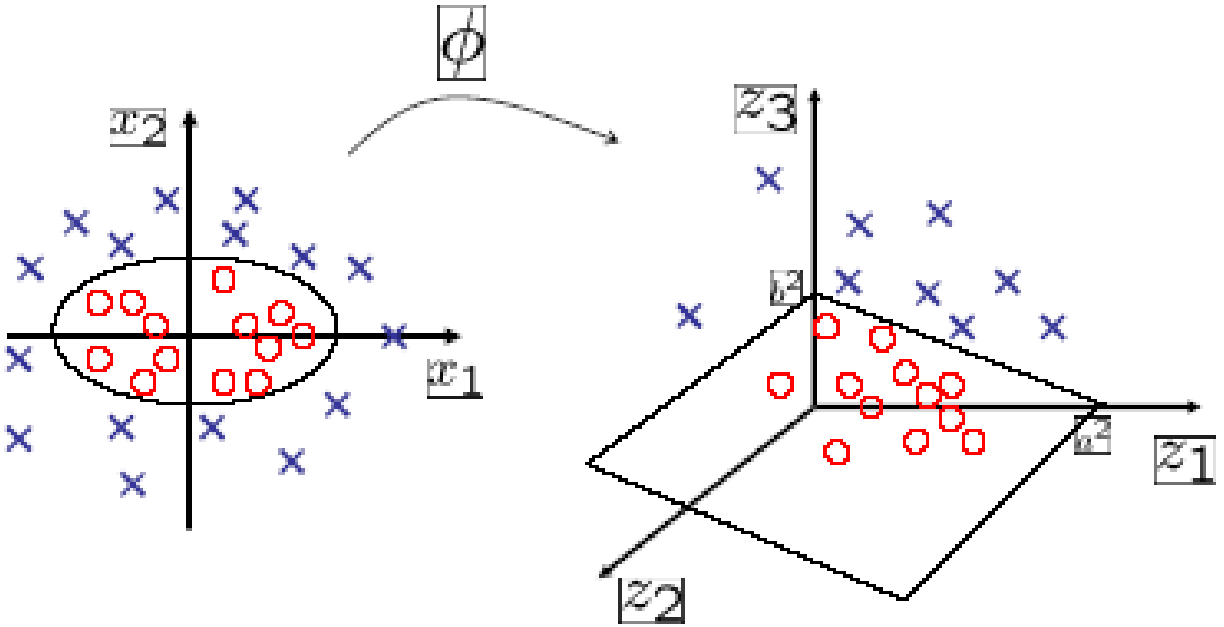
Non-linear SVM - Idea

- **Transform the data from its original feature space to a new space where a linear boundary can be used to separate the data**
- **Motivation for this - Cover's theorem:**
  - **A complex classification problem cast in high dimensional space nonlinearly is more likely to be linearly separable than in a low dimensional space (Cover, 1965)**

  - **=> 2 important properties of the transformation:**
    - **1) It is nonlinear**
    - **2) it is to a higher dimensional space**

Example

- **Non-linearly separable data $\mathbf{x} = (x_1, x_2)$ in the original space and linearly separable**



$$\phi : (x_1, x_2) \longrightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

transformation from old to new space

$$\left(\frac{x_1}{a}\right)^2 + \left(\frac{x_2}{b}\right)^2 = 1 \longrightarrow \frac{z_1}{a^2} + \frac{z_3}{b^2} = 1$$

decision boundaries in old and new space

original space (2-dim):

(x_1, x_2)

new space (3-dim):

$(x_1^2, \sqrt{2}x_1x_2, x_2^2)$

Issue 1

- **How do we choose the dimensionality of the new space, so that the data is linearly separable in the new space?**
  - **Theorem: If a data set is mapped to a space of *sufficiently high dimension*, it will always be linearly separable.**
  - **OK, then, let's use a space with infinite dimensionality**
  - **Danger: Overfitting!**
  - **Recall that a line in a d-dim space is defined by an equation with d parameters => if d ≈ N (number of training examples), there is a serious danger of overfitting**
  - **=> there are restrictions on the dim of the new space defined by the data**

  - **We will also deal with overfitting by constructing the maximum margin hyperplane in the new space (recall the structural risk minimization principle)**

Issue 2

- **Even if we know what the transformation Φ should be, solving a QP task in the new, higher dimensional space, is computationally very expensive, e.g. for our example:**

  1) Transform the 2-dim training data into 3-dim using Φ, i.e. compute the new features:

$$\mathbf{x}_i \xrightarrow{\Phi} \Phi(\mathbf{x}_i), \mathbf{x}_j \xrightarrow{\Phi} \Phi(\mathbf{x}_j)$$

  $\mathbf{x}_1, \mathbf{x}_2$ – a pair of training vectors in the original 2-dim feature space

  2) Feed the 3-dim vectors to the QP solver

  Recall that QP computes dot product of pairs of training vectors:

$$\max W(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

Before (in the original 2-dim space)

$$=$$

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

Now (in the new 3-dim space)

  However, what about if 3 is 10 or more? The computation increases dramatically! First we need to compute the new higher dim features, then their dot products.

Solution: *kernel trick*

Kernel Trick

- **Method for computing the dot product of a pair of vectors in the new space without first computing the transformation of each vector from the original to the new space**
  - **We need the dot product of the features in the new space (i.e. transformed features)** $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$
  - **We will <u>not</u> compute the features in the new space and then compute their dot product**

$$1)\ \mathbf{x}_i \xrightarrow{\Phi} \Phi(\mathbf{x}_i), \mathbf{x}_j \xrightarrow{\Phi} \Phi(\mathbf{x}_j)$$

$$2)\ \Phi(\mathbf{x}_i).\Phi(\mathbf{x}_j)$$

  - **We will compute the dot product of the original features and use it in a function (called kernel function) to determine the value the dot product of the transformed features** $\mathbf{x}_i.\mathbf{x}_j -> \Phi(\mathbf{x}_i).\Phi(\mathbf{x}_j)$

Kernel Trick for Our Example

- **2 dim original and 3 dim new space,** $\Phi : (x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

- **Consider a pair of vectors in original space: u, v (they are 2-dim). They were transformed into the (3 dim.) vectors Φ(u), Φ(v) in the new space, i.e.**

$$\mathbf{u} \overset{\Phi}{\rightarrow} \Phi(\mathbf{u}), \mathbf{v} \overset{\Phi}{\rightarrow} \Phi(\mathbf{v})$$

- **Let's calculate the dot product of Φ(u) and Φ(v)**

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (u_1^2, \sqrt{2}u_1u_2, u_2^2) \cdot (v_1^2, \sqrt{2}v_1v_2, v_2^2) =$$

$$= u_1^2v_1^2 + 2u_1u_2v_1v_2 + u_2^2v_2^2 = (u_1v_1)^2 + (u_2v_2)^2 + 2u_1u_2v_1v_2 =$$

$$= (u_1v_1 + u_2v_2)^2 = (\mathbf{u} \cdot \mathbf{v})^2$$

- **The dot product in the new space can be expressed via the dot product in the original space!**

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^2$$

Nice property!

- **This means that the dot product in the new space can be computed without first computing Φ for each input vector! Instead we will compute a function in the original space to evaluate the dot product in the new space!**

Kernel Trick

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^2$$

- **The right-hand side is called a *kernel function* and is written as** $K(\mathbf{u}, \mathbf{v})$

=> $K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^2$

- **So, we will compute a kernel function in the original space for each pair of training vectors to evaluate the pair's dot product in the new space**
- **Thus, to find a linear boundary in the new (higher dimensional) space we will feed the QP solver not with** $\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$ **but with** $K(\mathbf{u}, \mathbf{v})$

- **Thus, we learn in higher dimensional space by computing kernel functions in lower dimensional space instead of first transforming each vector in the higher dimensional space and then computing dot products between vectors**

Mercer's Theorem

- **Recall the special property of the kernel function for our example**

$$K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^2$$

- **Functions K for which this is true (i.e. such Φ exist) need to satisfy the Mercer's Theorem, i.e. this restricts the class of functions K we can use**

Frequently Used Kernel Functions

- **Some kernels satisfying the Mercer's Theorem are**

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+1)^{p} - \textit{polynomial kernel}$$

$$K(\mathbf{x},\mathbf{y}) = e^{-\frac{\|x-y\|^2}{2\sigma^2}} - RBF$$

$$K(\mathbf{x},\mathbf{y}) = \tanh(k\mathbf{x}\cdot\mathbf{y}-\theta) - \textit{tanget hyperbolic}$$

$$(\textit{satisfies Mercer's Th. only for some } k \textit{ and } \theta)$$

Kernel Trick Again

$$K(\mathbf{u},\mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) - \textit{kernel trick}$$

- **We specify K (should satisfy Mercer's Theorem), i.e. we specify Φ indirectly, instead of choosing Φ**
- **We perform the dot product computation in the original space which has smaller dimensionality**

The Optimization Problem Using Kernel Functions - Training

- **Original, without kernel function**

$$\max \mathbf{w}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$subject\ to\ \lambda_i \geq 0, \sum_{i=1}^{N} \lambda_i y_i = 0$$

solution

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

Optimal hyperplane in the original space

- **With kernel function**

$$\max \mathbf{w}(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$subject\ to\ \lambda_i \geq 0, \sum_{i=1}^{N} \lambda_i y_i = 0$$

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \Phi(\mathbf{x}_i)$$

Optimal hyperplane in the new space

The Optimization Problem Using Kernel Functions – Classification of New Examples

- **Classify new example z as sign(f), i.e. class 1 if f>0 and class -1 if f<0:**
- **Original, without kernel function**

$$f = \mathbf{w} \cdot \mathbf{z} + b = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{z} + b$$

Dot product of the new vector and the support vectors

- **With kernel function**

$$f = \mathbf{w} \cdot \mathbf{z} + b = \sum_{i=1}^{N} \lambda_i y_i K(\mathbf{x}_i, \mathbf{z}) + b$$

Kernel function of the new vector and the support vectors

- **In both cases the summation is over the support vectors, i.e. a subset of all training examples N as many λs are 0**

SVM – Summary

- **Very popular and useful method for classification**
- **Most popular "off-the-shelf" classification method as it has only a few parameters that are easy to tune**
- **Three key concepts:**
  - **1) Maximize the margin of the decision boundary**
  - **2) Transform data into a higher dimensional space where it is more likely to be linearly separable**
  - **3) Kernel trick – do the calculations in the original, not in the new higher dimensional space**

SVM – Summary (2)

- **Linear SVM – No kernel function; finds the optimal linear decision boundary between the positive and negative examples in the original feature space**
- **Non-linear SVR – Uses kernel function; finds the optimal linear boundary in the new space. This boundary (hyperplane) when mapped back to the original feature space, corresponds to a non-linear decision boundary between the positive and negative examples**
- **Scales well in high dimensions**
- **Multi-class problems need to be transformed into 2-class problems**

SVM, Perceptrons and Backpropagation NNs – Comparison

- **Perceptrons: simple and efficient training algorithm but can learn only linear decision boundaries**
- **Backpopagation NNs: hard to train (too many parameters to set, local minima) but can learn non-linear boundaries**
- **SVM: relatively efficient training algorithm that can learn non-linear decision boundaries**

More on Kernels

- **Kernels have been designed for strings, trees and non-numeric data**
- **Kernelisation can be applied to all algorithms that can be re-formulated to work only with dot products (once this is done the dot product is replaced with a kernel function)**
  - **e.g. k-nearest neighbor and others**

SVM Links

- **SVMs were introduced by Cortes and Vapnik (1995)**
  https://link.springer.com/article/10.1007/BF00994018

- **Tutorial by Christopher Burges**
  https://drive.google.com/open?id=0B6x5IP0EWft1RFZGYWpodDlGZXc

- **Software - SVM Light**
  http://svmlight.joachims.org/

- **Links**
  http://www.svms.org/
  http://www.svms.org/links.html

Acknowledgements

- **Some slides based on Tan, Steinbach, Kumar, *Introduction to Data Mining*, Addison-Wesley**
- **Some slides adapted from Martin Law's slides**

  http://download.informatik.uni-freiburg.de/lectures/ML/2004-2005WS/Misc/Slides/15_1_SVMachines_4up.pdf