Support Vector Machines

COMP5318/COMP4318 Machine Learning and Data Mining

semester 1, 2023, week 6a

Irena Koprinska

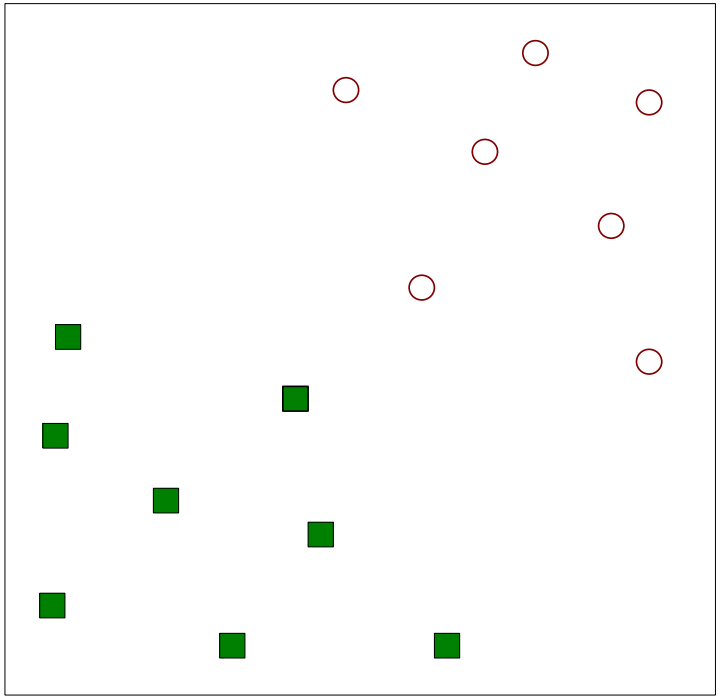
Reference: Tan ch.4.9, Witten ch.7.2: 252-256

Outline

- Maximum margin hyperplane
- Linear SVM
- Non-linear SVM

Separation by a hyperplane

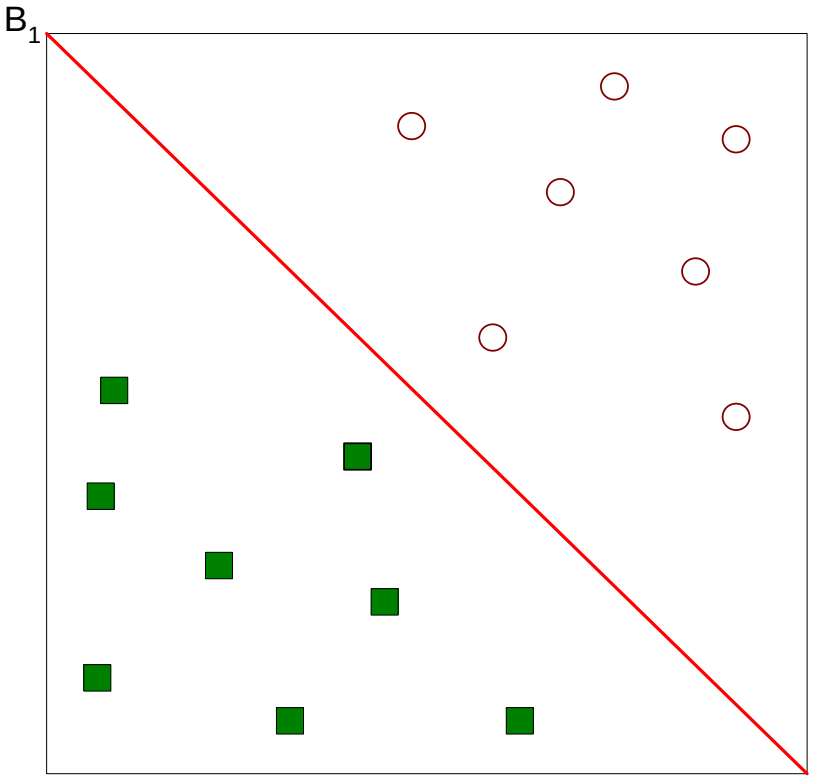
- Given is the following training data (2 class problem: squares and circles):



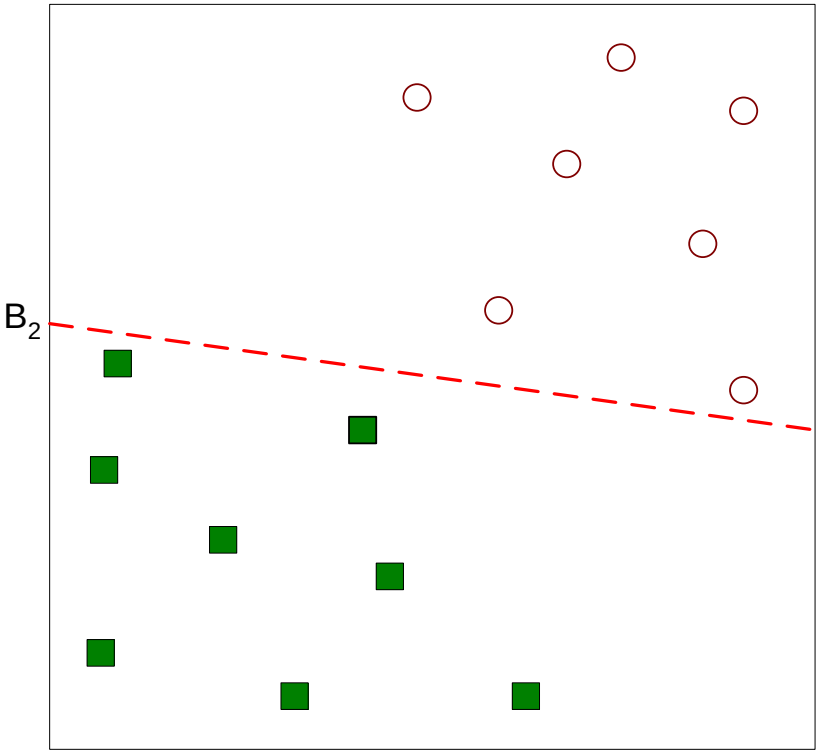
- Find a linear boundary (line, hyperplane) that separates the data
- Is the data linearly separable?

Separation by a hyperplane (2)

- One possible decision boundary

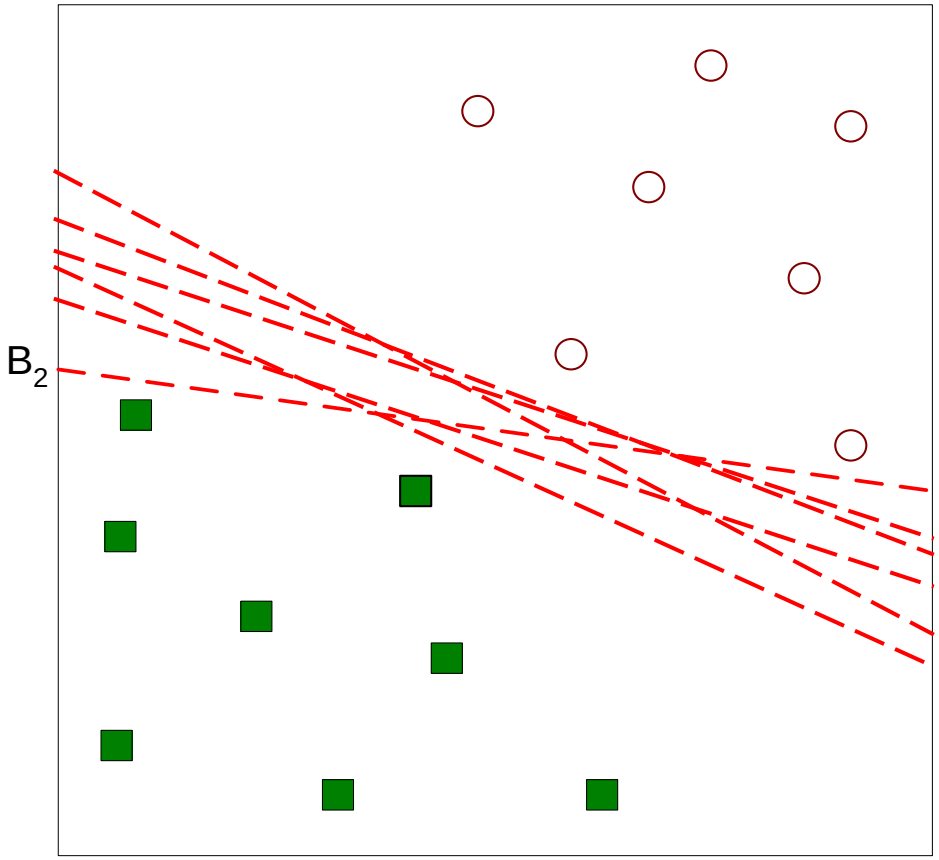


Another possible decision boundary



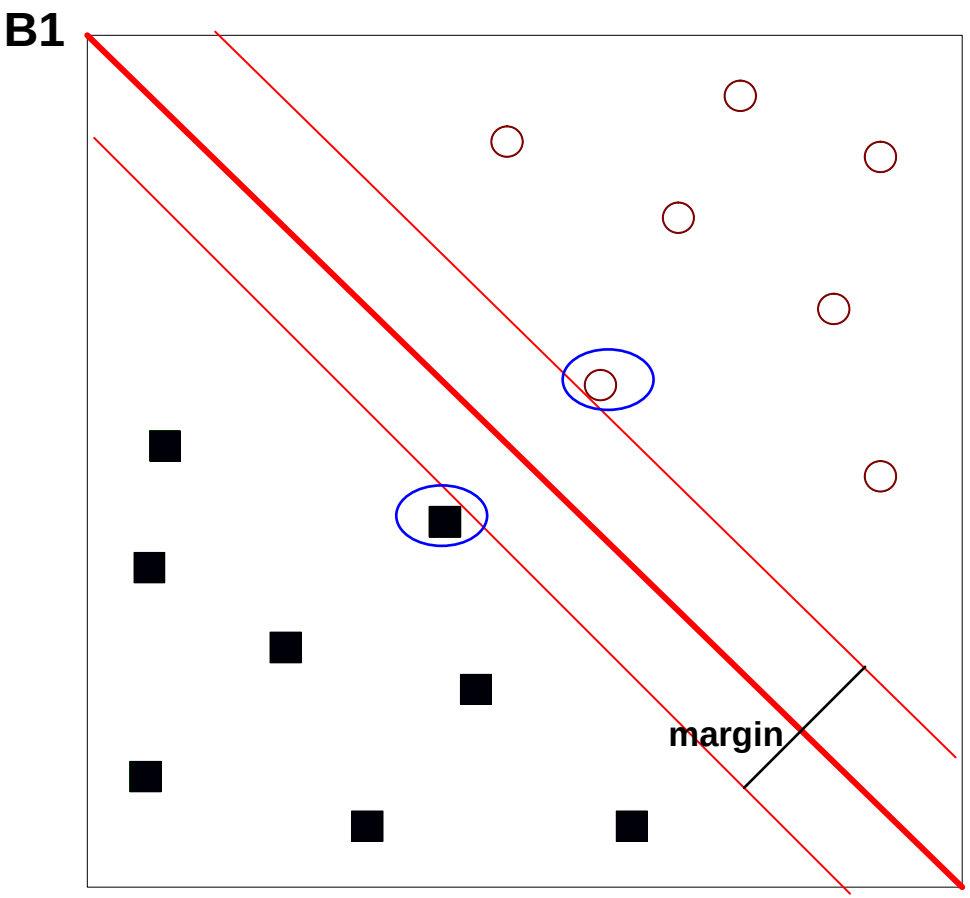
Separation by a hyperplane (3)

- Other possible decision boundaries – there are many options



Margin of a hyperplane and support vectors

- A decision boundary B1:

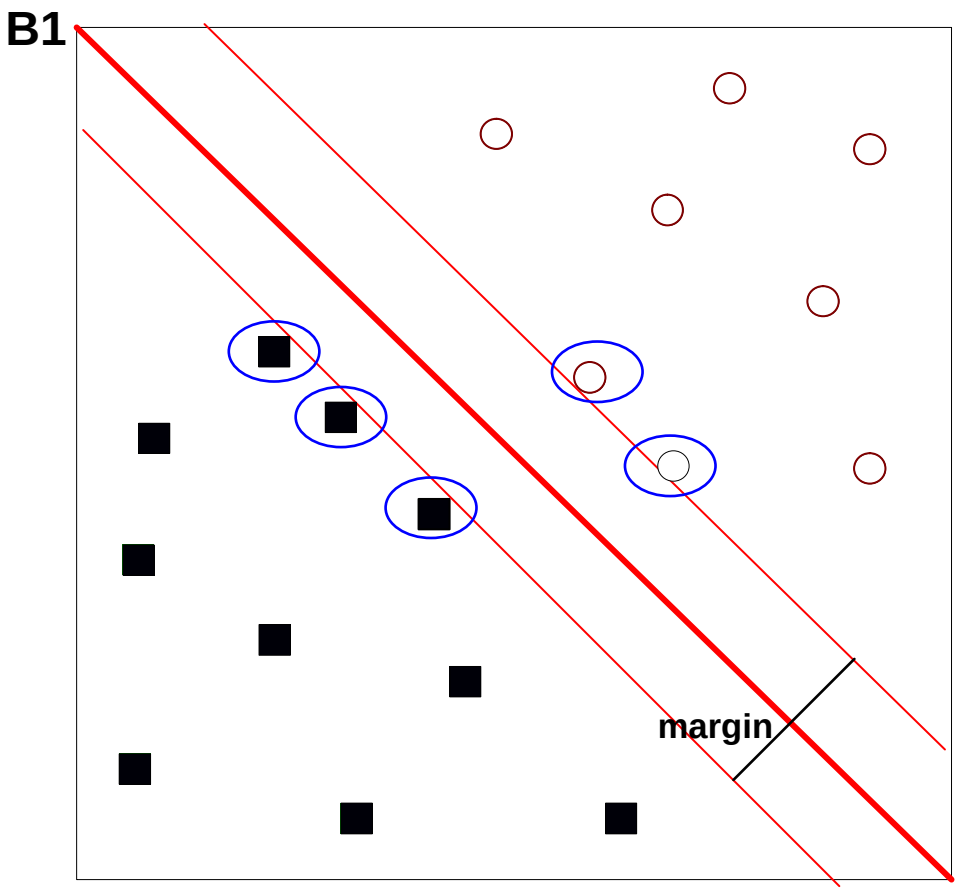


Support vectors are the examples (data points) that lie closest to the decision boundary; they are circled

Margin – the separation between the boundary and the closest examples

The boundary is in the middle of the margin

Support vectors

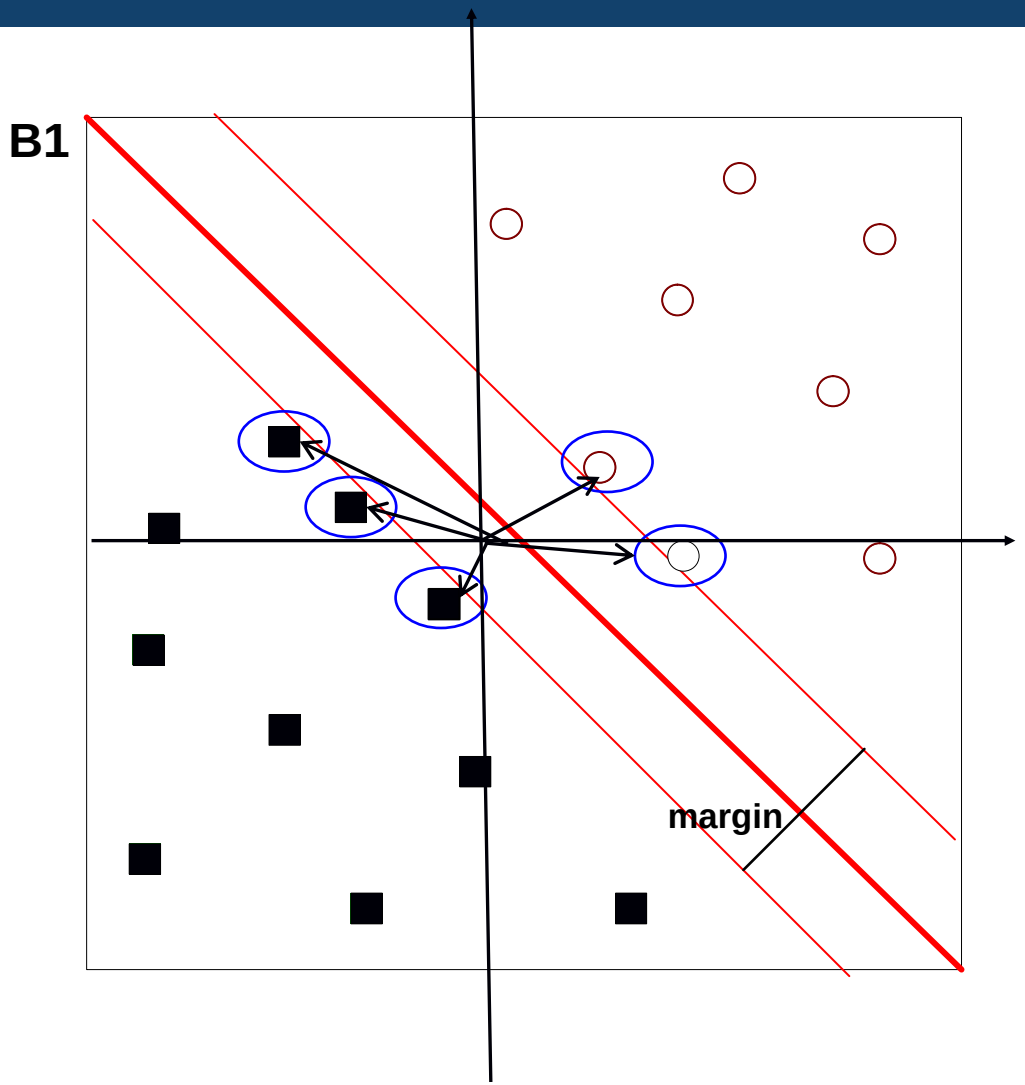


The support vectors just touch the margin of the decision boundary

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”?

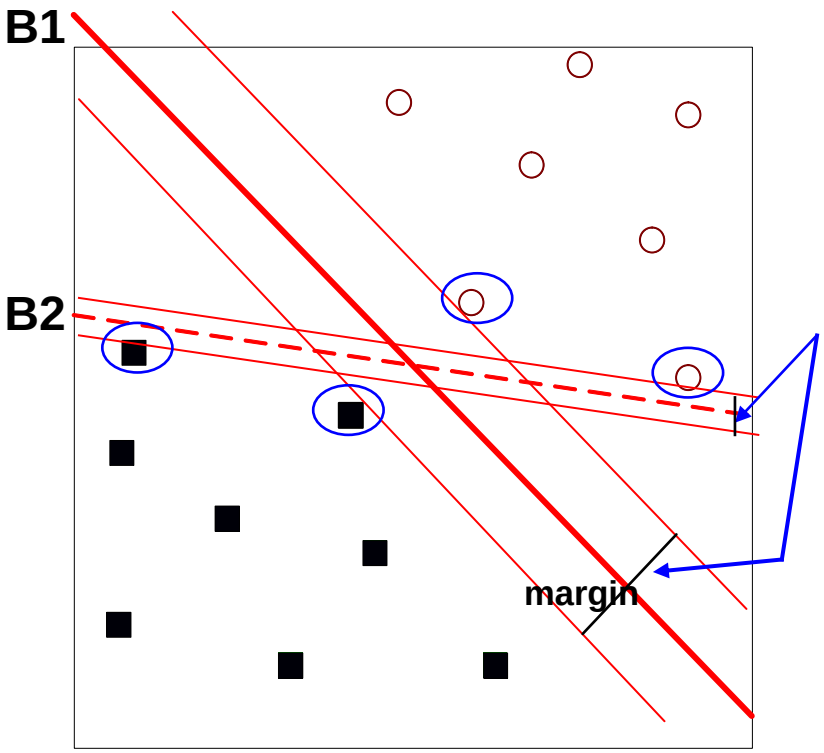
On the previous slides we saw only the tips of the support vectors; here we can see the actual vectors

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

Which hyperplane is better?

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



- The hyperplane with the bigger margin, B1

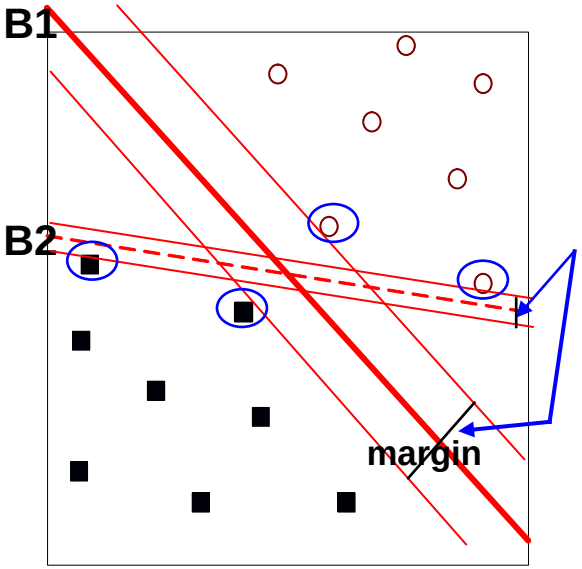
Maximum margin hyperplane

- The hyperplane with the biggest margin is called the *maximum margin hyperplane*
- 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

Maximum margin hyperplane - rationale

- If the margin is small, then any slight change in the hyperplane or the training examples at the boundary is more likely to affect the classification as there is very narrow room to allow data perturbations
- => small margin is more susceptible to overfitting
- On the other hand, if the margin is big, there is more leeway to be robust to minor changes in the data
- => likely to have better generalization performance



- There is also justification from statistical learning theory, the so called structural risk minimization principle

Linear decision boundary

- 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:

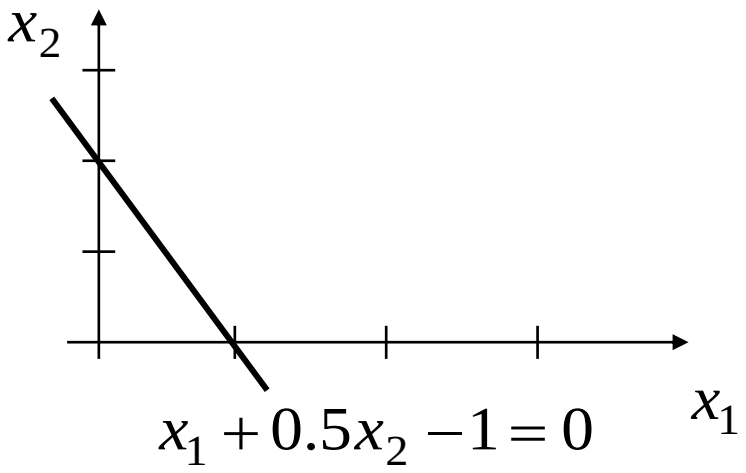
$$\mathbf{x}_1 = (2,2), y_1 = 1$$

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

- A decision boundary of a linear classifier is

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

parameters



Learning a linear model is equivalent to finding **w** and **b**

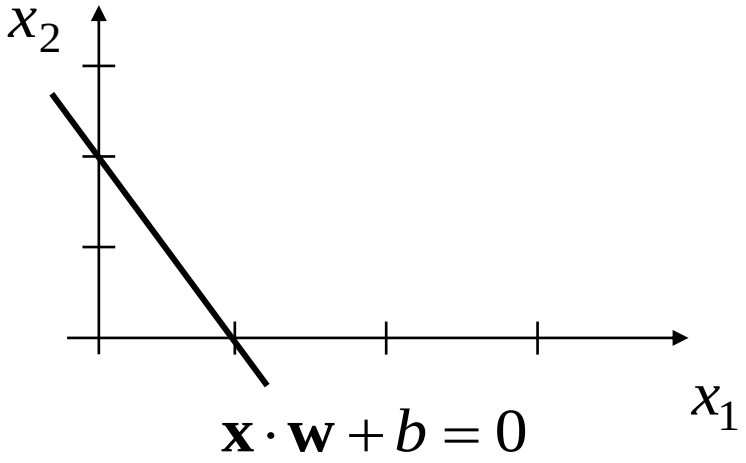
Linear model – classifying new examples

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

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

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

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



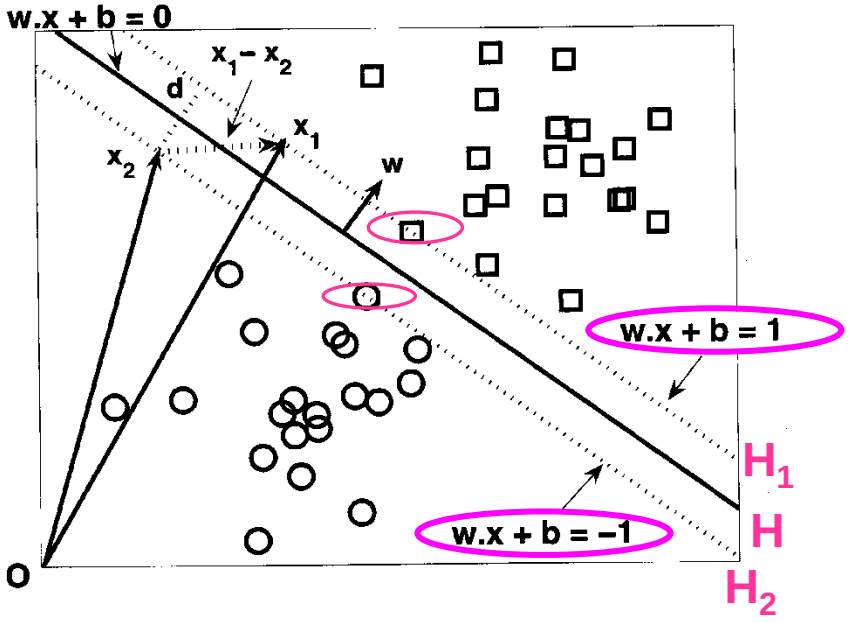
SVM - problem statement

- 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: $d = \frac{2}{\|\mathbf{w}\|}$



- To maximize the margin d, we need to minimize ||w||
- This is equivalent to minimizing 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

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

Another way of expressing correct classification of all training examples i=1..N

The optimization problem

- This is an optimization problem that can be solved using Quadratic Programming (QP) and the Lagrange multiplier method
- Firstly, the problem is transformed into an equivalent form using Lagrange multipliers λ :

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

Dot product of pairs of training vectors

Class value of the training vectors

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

- The values of λs are found using QP
- The solution (i.e. the optimal decision boundary) is given by:

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

Properties of the solution

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

- The optimal decision boundary (the maximum margin hyperplane) w is a linear combination (coefficient λ * target value*training vector) of the training examples
- But many of the λs are 0 -> linear combination 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

Classifying new examples

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

- 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

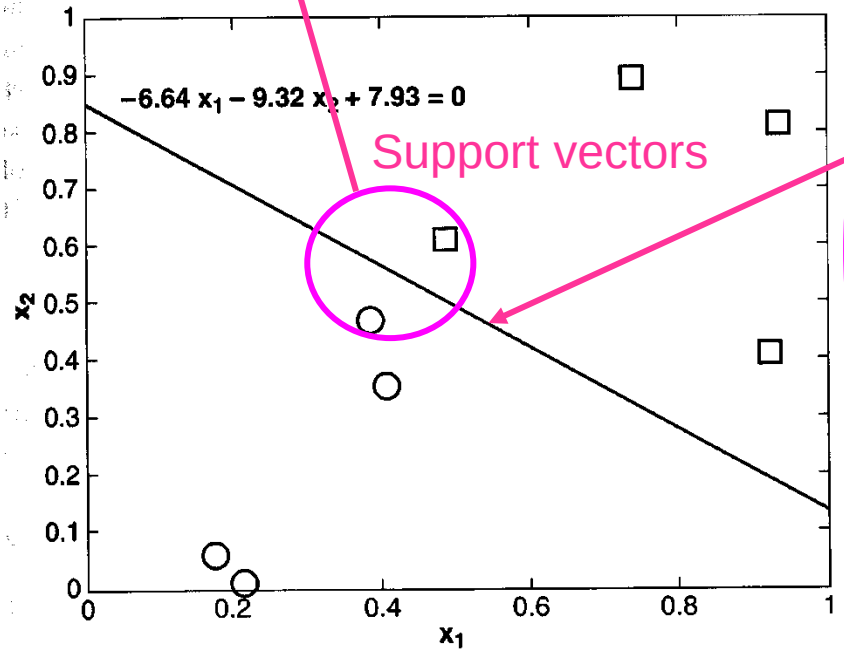
$$sign(f)$$

i.e. the new example belongs to class 1, if f>0 or class -1 if f<0

Example

features x_1	features x_2	class y	Lagrange Multiplier
0.3858	0.4687	1	65.5261
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

x1
x2



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

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

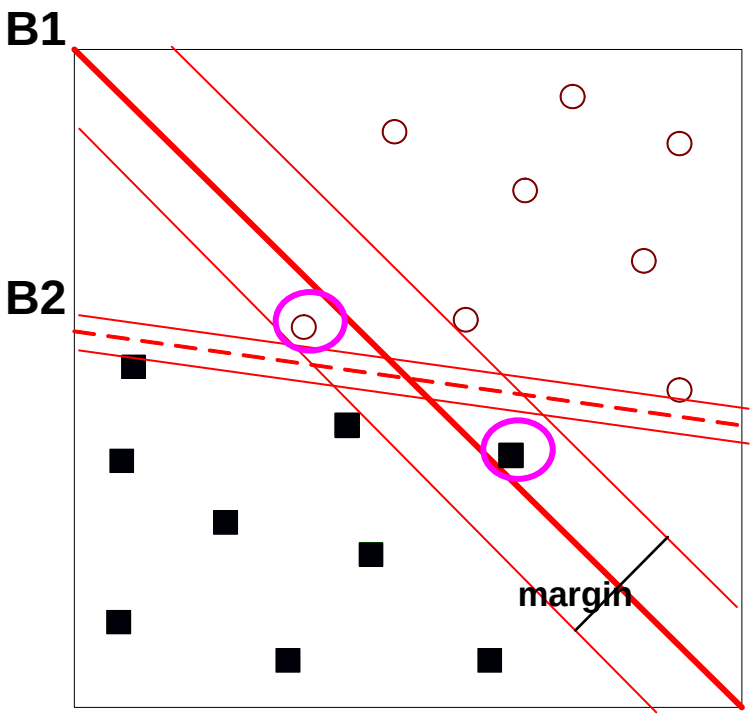
$$w_2 = \sum_{i=1}^{2} \lambda_i y_i \mathbf{x}_{i2} = 65.5261(1*0.4687 - 1*0.611) = -9.32$$

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

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

SVM with 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



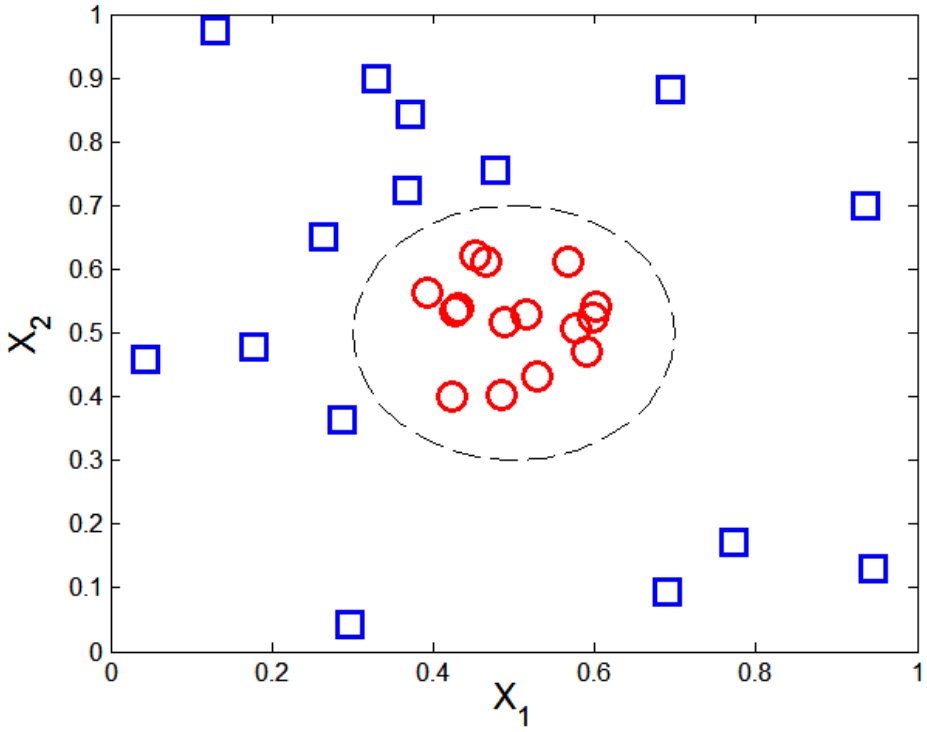
- This does not mean that B2 is better as the new examples may be noise
- B1 should still be preferred as it has a bigger margin and is less sensitive to overfitting and noise
- We can modify our method to allow some misclassifications, i.e. by considering the trade-off between the margin width and the 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 definition of the optimization function
- C is a hyper-parameter that allows for a trade-off between maximizing the margin and minimizing the training error
  - Large C: more emphasis on minimizing the training error than maximizing the margin

Non-linear SVM

- In practice most problems are linearly non-separable
- SVM can further be extended to find a non-linear boundary



The decision boundary is non-linear

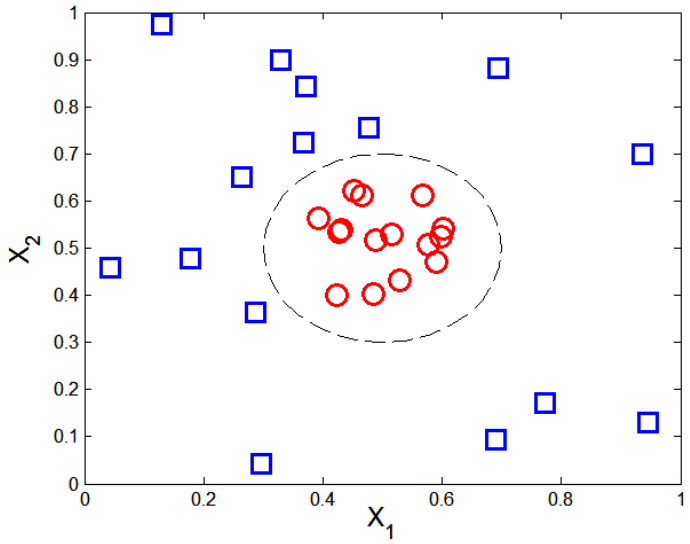
Non-linear SVM - Idea

- Transform the data from its original feature space to a new space where a linear boundary can be used to separate the data
- If the transformation is non-linear and to a higher dimensional space, it is more likely than a linear decision boundary can be found in it

- The learned linear decision boundary in the new feature space is mapped back to the original feature space, resulting in a non-linear decision boundary in the original space

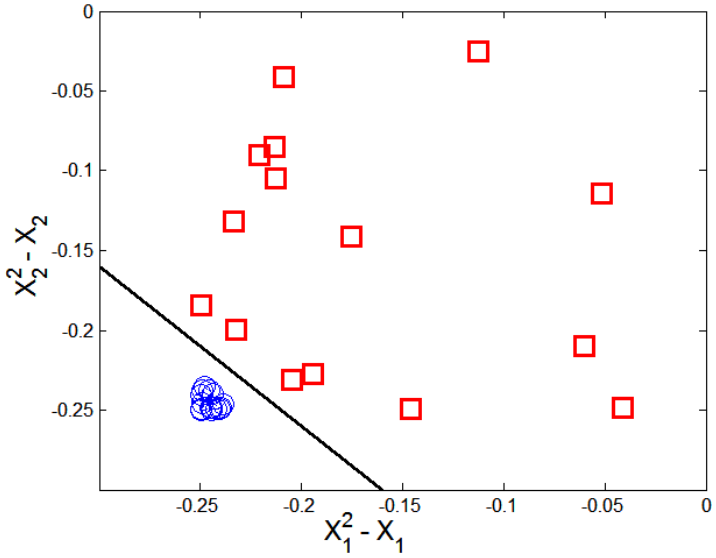
Example

- Non-linearly separable data in the original space



ϕ

- Becomes linearly separable in the new space



transformation from old to new space: $\phi = (x_1, x_2) \rightarrow ({x_1}^2 - x_1, {x_2}^2 - x_2)$

Learning non-linear SVM

- What type of mapping function Φ should we use?
- Even if we know Φ , how can we do the computation in the new space efficiently?
  - The new space is typically with higher dimensionality
  - Computing dot products of vectors in high dimensional space is computationally expensive
  - Recall that to find the decision boundary we compute dot products of input vectors

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

in the original space

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

$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ in the new space – higher dim vectors

- How to effectively compute dot products in high dimensional space?
- 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: $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$

- We will not firstly transform 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 dot product of the transformed features
- The kernel function specifies the relationship between the dot products in the original and transformed space

Kernel trick - example

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

- u, v – vectors in the original space (2-dim)
- Φ(u), Φ(v) – transformed vectors u and v in the new space (3-dim)

$$\mathbf{u} \xrightarrow{\Phi} \Phi(\mathbf{u}), \mathbf{v} \xrightarrow{\Phi} \Phi(\mathbf{v})$$

- 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!
- This relationship is specified by the kernel function: $\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^2$

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

Kernel function

- Kernel function for our example:

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

- Kernel functions allow the dot product in the new space to be computed without first computing Φ for each input vector
- As they specify the relationship between the dot products in the original and new feature space, we can compute a function of dot product in the original space and use it to evaluate the dot product in the new spaces
- Hence, we learn in higher dimensional space by computing kernel functions in lower dimensional space

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

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

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

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

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

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

Kernel trick again

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

- 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

SVM training and classification using kernel functions

- Training:

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

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

- Optimal hyperplane in the new space: $\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \Phi(\mathbf{x}_i)$

- Classifying new example z:

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

SVM – summary

- Very popular classification method
- Can form arbitrary decision boundaries (both linear and non-linear)
- Three key concepts:
  - The decision boundary is the maximum margin hyperplane – the task is formulated as an optimization problem
  - Transform data into a new (typically higher dimensional space) where it is more likely to be linearly separable
  - Kernel trick – do the calculations in the original, not in the new higher dimensional space
- Can be applied to multi-class classification problems too – they are transformed into 2-class problems
- There is an extension for regression tasks – Support Vector Regression

More on kernels

- 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

References

- 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