Support Vector Machines

Javier Béjar ⓒⓘⓔⓞ

LSI - FIB

Term 2012/2013

Linear separators

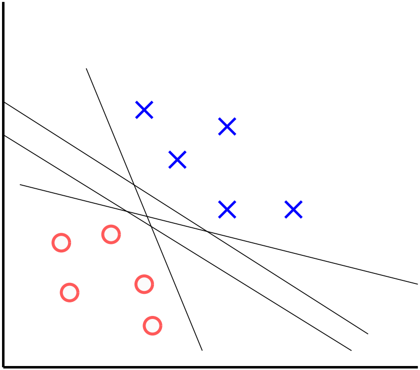
- As we saw with the perceptron, one way to perform classification is to find a linear separator (for binary classification)
- The idea is to find the equation of a line $w^Tx + b = 0$ that divides the set of examples in the target classes so:

$$w^Tx + b > 0 \Rightarrow \textit{Positive class}$$
$$w^Tx + b < 0 \Rightarrow \textit{Negative class}$$

- This is just the problem that the single layer perceptron solves (b is the bias weight)

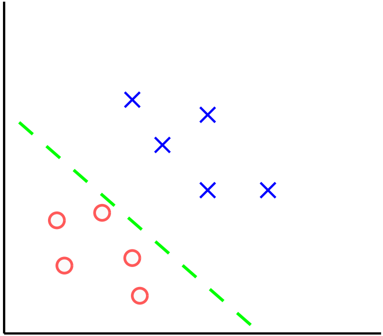
Optimal linear separator

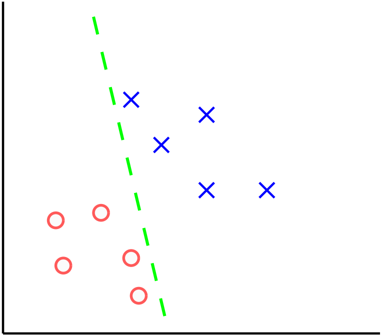
- Actually any line that divides the examples can be the answer for the problem
- The question that we can ask is, what line is the best one?



Optimal linear separator

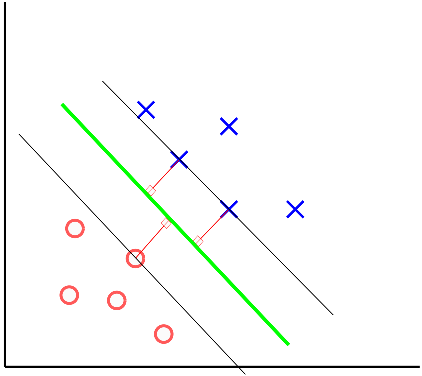
- Some boundaries seem a bad choice due to poor generalization
- We have to maximize the probability of classifying correctly unseen instances
- We want to minimize the *expected generalization loss* (instead of the expected empirical loss)





Maximum Margin Classification

- Maximizing the distance of the separator to the examples seems the right choice (actually supported by PAC learning theory)
- This means that only the nearest instances to the separator matter (the rest can be ignored)

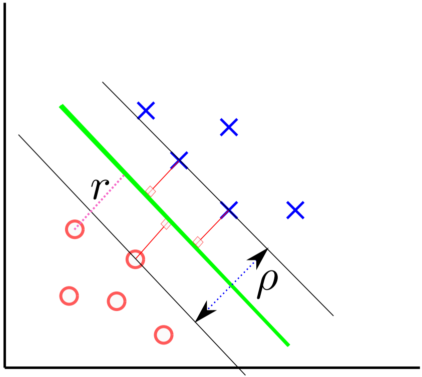


Classification Margin

- We can compute the distance from an example x_i to the separator as:

$$r = \frac{w^T x_i + b}{||w||}$$

- The examples closest to the separator are support vectors
- The margin (ρ) of the separator is the distance between support vectors



Large Margin Decision Boundary

- The separator has to be as far as possible from the examples of both classes
- This means that we have to **maximize** the margin
- We can normalize the equation of the separator so the distance in the supports are 1 or -1, by

$$r = \frac{w^T x + b}{||w||}$$

- So the length of the optimal margin is $m = \frac{2}{||w||}$
- This means that maximizing the margin is the same that minimizing the norm of the weights

Computing the decision boundary

- Given a set of examples $\{x_1, x_2, \ldots, x_n\}$ with class labels $y_i \in \{+1, -1\}$
- The decision boundary that classify the examples correctly holds

$$y_i(w^T x_i + b) \geq 1, \quad \forall i$$

- This allows to define the problem of learning the weights as an optimization problem

Computing the decision boundary

- The primal formulation of the optimization problem is:

  Minimize $\frac{1}{2}||w||^2$

  subject to $y_i(w^T x_i + b) \geq 1, \ \ \forall i$

- This problem is difficult to solve in this formulation, but we can rewrite the problem in a more convenient form

Constrained optimization (a little digression)

- Suppose we want to minimize $f(x)$ subject to $g(x) = 0$
- A necessary condition for a point x_0 to be a solution is

$$\begin{cases} \frac{\partial}{\partial x}\left(f(x) + \alpha g(x)\right)\Big|_{x=x_0} & = 0 \\ g(x) = 0 \end{cases}$$

- Being α the Lagrange multiplier
- If we have multiple constraints, we just need one multiplier per constraint

$$\begin{cases} \frac{\partial}{\partial x}\left(f(x) + \sum_{i=1}^{n} \alpha_i g_i(x)\right)\Big|_{x=x_0} & = 0 \\ g_i(x) = 0 \;\; \forall i \in 1 \dots n \end{cases}$$

Constrained optimization (a little digression)

- For inequality constraints $g_i(x) \leq 0$ we have to add the constraint that the Lagrange multipliers have to be positive $\alpha_i \geq 0$
- If x_0 is a solution to the constrained optimization problem

$$\min_x f(x) \text{ subject to } g_i(x) \leq 0 \ \ \forall i \in 1 \dots n$$

- There must exist $\alpha_i \geq 0$ such that x_0 satisfies

$$\begin{cases} \frac{\partial}{\partial x}\left(f(x) + \sum_{i=1}^{n} \alpha_i g_i(x)\right)\Big|_{x=x_0} & = 0 \\ g_i(x) = 0 \ \ \forall i \in 1 \dots n \end{cases}$$

- The function $f(x) + \sum_{i=1}^{n} \alpha_i g_i(x)$ is called the Lagrangian
- The solution is the point of gradient 0

Computing the decision boundary (we are back)

- The primal formulation of the optimization problem is:

  Minimize $\frac{1}{2}||w||^2$

  subject to $1 - y_i(w^T x_i + b) \leq 0, \;\; \forall i$

- The Lagrangian is[1]:

$$\mathcal{L} = \frac{1}{2} w^T w + \sum_{i=1}^{n} \alpha_i (1 - y_i(w^T x_i + b))$$

[1] $||w|| = w^T w$

Computing the decision boundary (we are back)

- If we compute the derivative of $\mathcal{L}$ with respect to w and b and we set them to zero (we are computing the gradient):

$$w + \sum_{i=1}^{n} \alpha_i(-y_i)x_i = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

and

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

The dual formulation

- If we substitute w in the Lagrangian $\mathcal{L}$

$$
\begin{aligned}
\mathcal{L} &= \frac{1}{2}\sum_{i=1}^{n}\alpha_i y_i x_i^T \sum_{j=1}^{n}\alpha_j y_j x_j + \sum_{i=1}^{n}\alpha_i \left(1 - y_i(\sum_{j=1}^{n}\alpha_j y_j x_j^T x_i + b)\right) \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\alpha_i y_i \sum_{j=1}^{n}\alpha_j y_j x_j^T x_i \\
&\quad -b\sum_{i=1}^{n}\alpha_i y_i \quad (and \;\; given \;\; that \;\; \sum_{i=1}^{n}\alpha_i y_i = 0) \\
&= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n}\alpha_i \quad (rearranging \;\; terms)
\end{aligned}
$$

The dual formulation

- This problem is known as the *dual problem* (the original is the primal)
- This formulation only depends on the α_i
- Both problems are linked, given the w we can compute the α_i and viceversa
- This means that we can solve one instead of the other
- Now this problem is a maximization problem ($\alpha_i \geq 0$)

The dual problem

- The formulation of the dual problem is

  Maximize $\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$

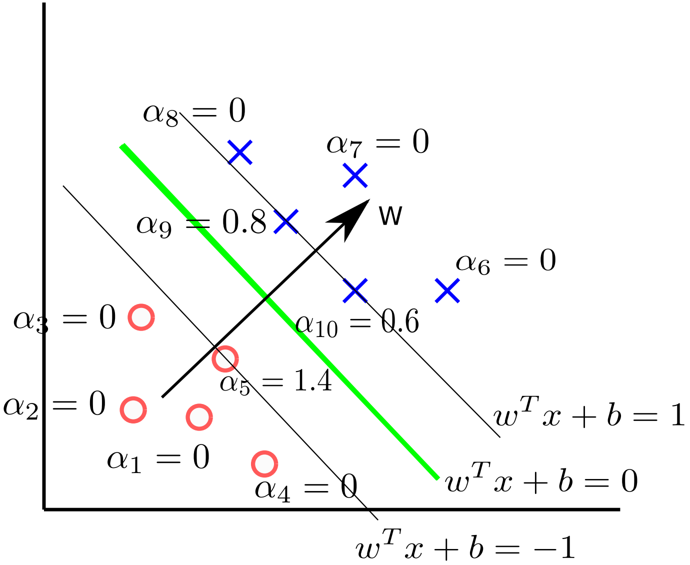
  subject to $\sum_{i=1}^{n} \alpha_i y_i = 0, \;\; \alpha_i \geq 0 \;\; \forall i$

- This a Quadratic Programming problem (QP)
- This means that the parameters form a parabolloidal surface, and an optimal can be found
- We can obtain w by

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

- b can be obtained from a positive support vector (x_{psv}) knowing that $w^T x_{psv} + b = 1$

Geometric Interpretation



Characteristics of the solution

- Many of the α_i are zero
  - w is a linear combination of a small number of examples
  - We obtain a *sparse* representation of the data (data compression)
- The examples x_i with non zero α_i are the support vectors (SV)
- The vector of parameters can be expressed as:

$$w = \sum_{\forall i \in SV} \alpha_i y_i x_i$$

Classifying new instances

- In order to classify a new instance z we just have to compute

$$sign(w^T z + b) = sign(\sum_{\forall i \in SV} \alpha_i y_i x_i^T z + b)$$

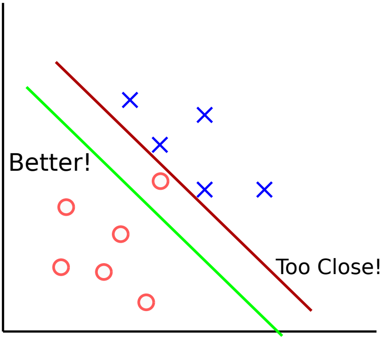
- This means that w does not need to be explicitly computed

Solving the QP problem

- Quadratic programming is a well known optimization problem
- Many algorithms have been proposed
- One of the most popular is Sequential Minimal Optimization (SMO)
  - Picks a pair of variables and solves a QP problem for two variables (trivial)
  - Repeats the procedure until convergence

Non separable problems

- This algorithm works well for separable problems
- Sometimes data has errors and we want to ignore them to obtain a better solution
- Sometimes data is just non linearly separable
- We can obtain better linear separators being less strict

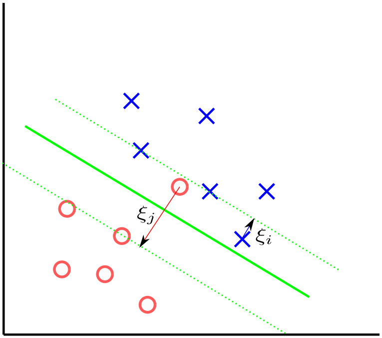


Soft Margin Classification

- We want to be permissive for certain examples, allowing that their classification by the separator diverge from the real class
- This can be obtained by adding to the problem what is called **slack variables** (ξ_i)
- This variables represent the deviation of the examples from the margin
- Doing this we are relaxing the margin, we are using a **soft** margin

Soft Margin Classification

- We are allowing an error in classification based on the separator $w^T x + b$
- The values of ξ_i approximate the number of missclassifications



Soft Margin Hyperplane

- In order to minimize the error, we can minimize $\sum_i \xi_i$ introducing the slack variables to the constraints of the problem:

$$\begin{cases} w^T x_i + b \geq 1 - \xi_i & y_i = 1 \\ w^T x_i + b \geq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 \end{cases}$$

- $\xi_i = 0$ if there are no errors (linearly separable problem)
- The number of resulting supports and slack variables give an upper bound of the leave one out error

The primal optimization problem

- We need to introduce this slack variables on the original problem, we have now:

  Minimize $\frac{1}{2}||w||^2 + C\sum_{i=1}^n \xi_i$

  subject to $y_i(w^T x_i + b) \geq 1 - \xi_i, \;\; \forall i, \xi_i \geq 0$

- Now we have an additional parameter C that is the tradeoff between the error and the margin
- We will need to adjust this parameter

The dual optimization problem

- Performing the transformation to the dual problem we obtain the following:

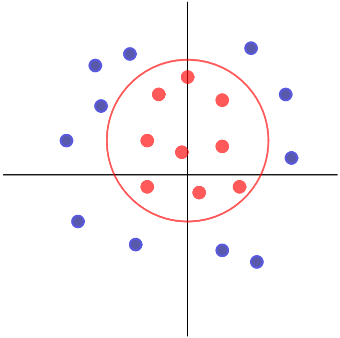
  Maximize $\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$

  subject to $\sum_{i=1}^{n} \alpha_i y_i = 0, \;\; C \geq \alpha_i \geq 0 \;\; \forall i$
- We can recover the solution as $w = \sum_{\forall i \in SV} \alpha_i, y_i x_i$
- This problem is very similar to the linearly separable case, except that there is a upper bound C on the values of α_i
- This is also a QP problem

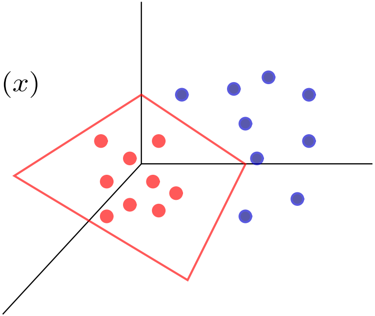
Non linear large margin classification

- So far we have only considered large margin classifiers that use a linear boundary
- In order to have better performance we have to be able to obtain non-linear boundaries
- The idea is to transform the data from the input space (the original attributes of the examples) to a higher dimensional space using a function $\phi(x)$
- This new space is called the **feature space**
- The advantage of the transformation is that linear operations in the feature space are equivalent to non-linear operations in the input space
- Remember the RBFs networks?

Feature Space transformation



$$\phi : x \Rightarrow \varphi(x)$$



The XOR problem

- The XOR problem is not linearly separable in its original definition, but we can make it linearly separable if we add a new feature $x_1 \cdot x_2$

x_1	x_2	$x_1 \cdot x_2$	x_1 XOR x_2
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	1

- The linear function $h(x) = 2x_1x_2 - x_1 - x_2 + 1$ classifies correctly all the examples

Transforming the data

- Working directly in the feature space can be costly
- We have to explicitly create the feature space and operate in it
- We may need infinite features to make a problem linearly separable
- We can use what is called the **Kernel trick**

The Kernel Trick

- In the problem that define a SVM only the inner product of the examples is needed
- This means that if we can define how to compute this product in the feature space, then it is not necessary to explicitly build it
- There are many geometric operations that can be expressed as inner products, like angles or distances
- We can define the kernel function as:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

Kernels - Example

- We can show how this kernel trick works in an example
- Lets assume a feature space defined as:

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

- A inner product in this feature space is:

$$\left\langle \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) \right\rangle = (1 + x_1 y_1 + x_2 y_2)^2$$

- So, we can define a kernel function to compute inner products in this space as:

$$K(x, y) = (1 + x_1 y_1 + x_2 y_2)^2$$

and we are using only the features from the input space

Kernel functions - Properties

- Given a set of examples $X = \{x_1, x_2, \ldots x_n\}$, and a symmetric function $k(x, z)$ we can define the kernel matrix K as:

$$K = \phi^T \phi = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \cdots & \cdots & \cdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}$$

- If K is a symmetric and positive definite matrix, then $k(x, z)$ is a kernel function
- The justification is that if K is a sdp matrix then can be decomposed in:

$$K = V \Lambda V^T$$

- V can be interpreted as a projection in the feature space (remember feature projection?)

Kernel functions

- Examples of functions that hold this condition are:
  - Polynomial kernel of degree d: $K(x, y) = (x^T y + 1)^d$
  - Gaussian function with width σ: $K(x, y) = \exp(-||x - y||^2/2\sigma^2)$
  - Sigmoid hiperbolical tangent with parameters k and θ: $K(x, y) = \tanh(kx^T y + \theta)$ (only for certain values of the parameters)
- Kernel functions can also be interpreted as similarity function
- A similarity function can be transformed in a kernel given that hold the sdp matrix condition

The dual problem

- Due to the introduction of the kernel function, the optimization problem has to be modified:

  Maximize $\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j K(x_i, x_j)$

  subject to $\sum_{i=1}^{n} \alpha_i y_i = 0, \quad C \geq \alpha_i \geq 0 \ \ \forall i$

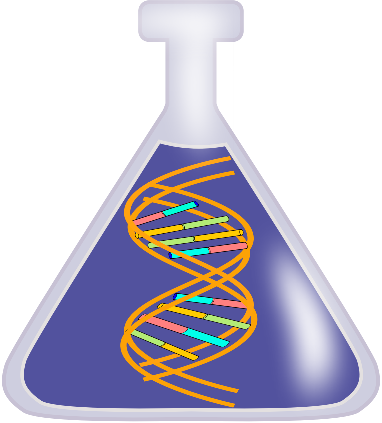
- For classifying new examples:

$$h(z) = sign\left(\sum_{\forall i \in SV} \alpha_i, y_i k(x_i, z) + b\right)$$

The advantage of using kernels

- Since the training of the SVM only needs the value of $K(x_i, x_j)$ there is no constrains about how the examples are represented
- We only have to define the kernel as a similarity among examples
- We can define similarity functions for different representations
  - Strings, sequences
  - Graphs/Trees
  - Documents
  - ...

Splice-Junction gene sequences



- Identification of gene sequence type (Bioinformatics)
- 60 Attributes (Discrete)
- Attributes: DNA base at position 1-60
- 3190 instances
- 3 classes
- Methods: Support vector machines
- Validation: 10 fold cross validation

Splice-Junction gene sequences: Models

- SVM (linear): accuracy 90.9%, 1331 Support Vectors
- SVM (linear, C=1): accuracy 91.8%, 608 Support Vectors
- SVM (linear, C=5): accuracy 91.5%, 579 Support Vectors
- SVM (quadratic): accuracy 91.4%, 1305 Support Vectors
- SVM (quadratic, C=5): accuracy 91.6%, 1054 Support Vectors
- SVM (cubic, C=5): accuracy 91.9%, 1206 Support Vectors