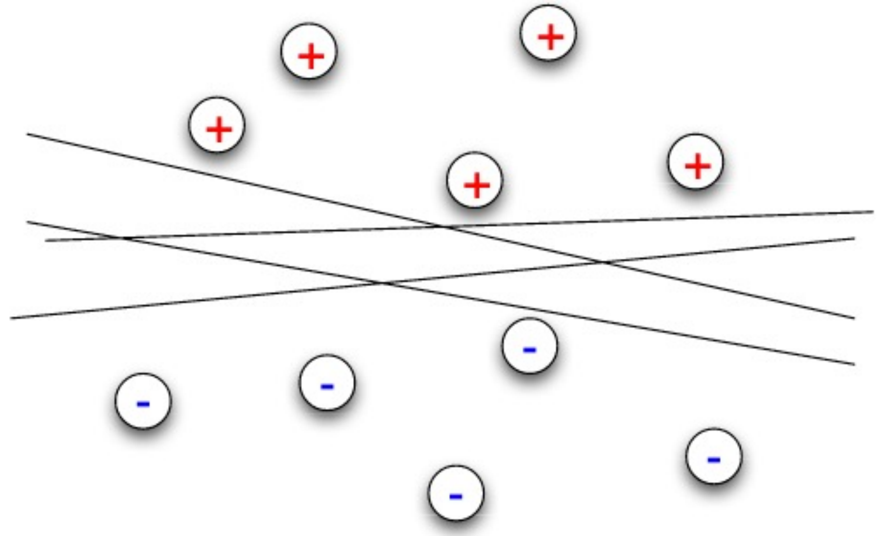
Support Vector Machines (SVMs)

Based on slides by Daniel Lowd, Doina Precup and others

Binary Classification Revisited

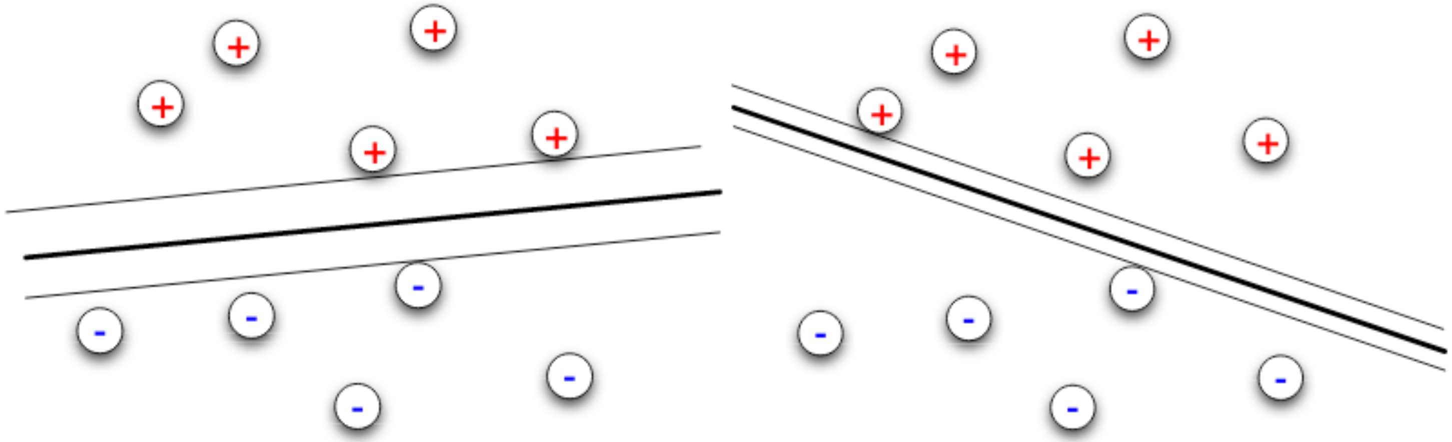
- Consider a linearly separable binary classification data set $\{\mathbf{x}_i, y_i\}_{i=1}^m$.
- There is an infinite number of hyperplanes that separate the classes:



- Which plane is best?
- Relatedly, for a given plane, for which points should we be most confident in the classification?

The margin, and linear SVMs

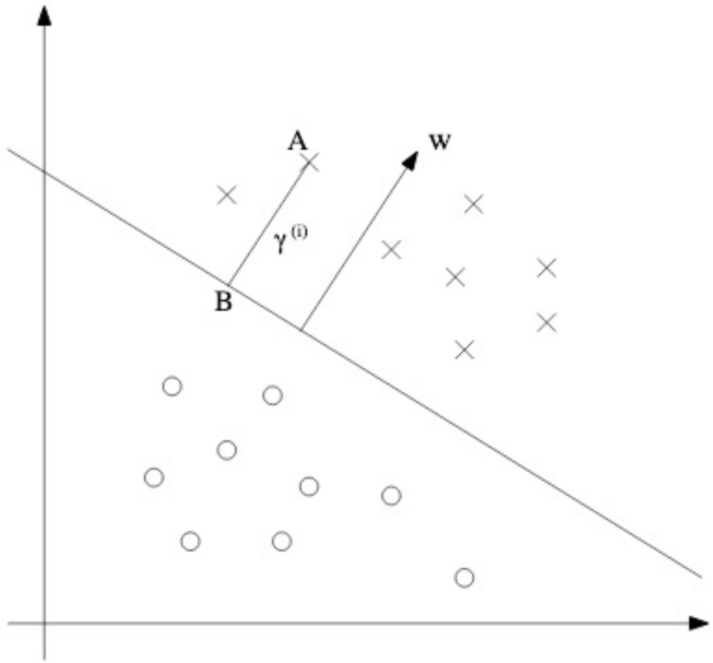
- For a given separating hyperplane, the *margin* is two times the (Euclidean) distance from the hyperplane to the nearest training example.



- It is the width of the "strip" around the decision boundary containing no training examples.
- A linear SVM is a perceptron for which we choose $\mathbf{w}, w_0$ so that margin is maximized

Distance to the decision boundary

- Suppose we have a decision boundary that separates the data.



- Let γ_i be the distance from instance $\mathbf{x}_i$ to the decision boundary.
- How can we write γ_i in term of $\mathbf{x}_i, y_i, \mathbf{w}, w_0$?

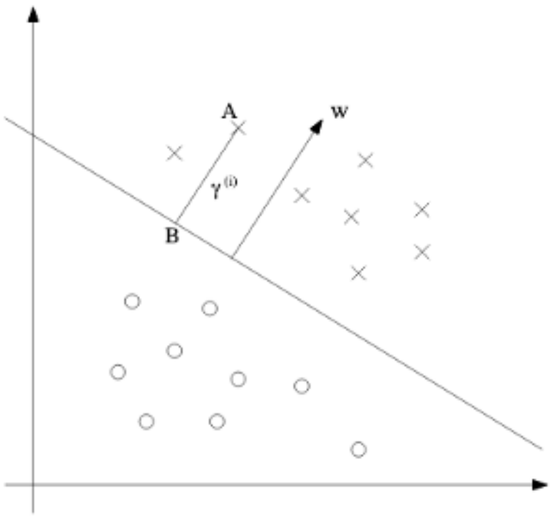
Distance to the decision boundary

- The vector $\mathbf{w}$ is normal to the decision boundary. Thus, $\frac{\mathbf{w}}{||\mathbf{w}||}$ is the unit normal.
- The vector from the B to A is $\gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$.
- B, the point on the decision boundary nearest $\mathbf{x}_i$, is $\mathbf{x}_i - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$.
- As B is on the decision boundary,

$$\mathbf{w} \cdot \left(\mathbf{x}_i - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||} \right) + w_0 = 0$$

- Solving for γ_i yields, for a positive example:

$$\gamma_i = \frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x}_i + \frac{w_0}{||\mathbf{w}||}$$



The margin

- The *margin of the hyperplane* is $2M$, where $M = \min_i \gamma_i$
- The most direct statement of the problem of finding a maximum margin separating hyperplane is thus

$$\max_{\mathbf{w},w_0} \min_i \gamma_i$$
$$\equiv \max_{\mathbf{w},w_0} \min_i y_i \left(\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x}_i + \frac{w_0}{||\mathbf{w}||} \right)$$

- This turns out to be inconvenient for optimization, however. . .

Treating γ_i as the constraints

- From the definition of margin, we have:

$$M \leq \gamma_i = y_i \left(\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x_i} + \frac{w_0}{||\mathbf{w}||} \right) \qquad \forall i$$

- This suggests:

$$\begin{array}{rl} \text{maximize} & M \\ \text{with respect to} & \mathbf{w}, w_0 \\ \text{subject to} & \mathbf{y}_i \left(\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x}_i + \frac{w_0}{||\mathbf{w}||} \right) \geq M \text{ for all } i \end{array}$$

- Problems:
  - $\mathbf{w}$ appears nonlinearly in the constraints.
  - This problem is underconstrained. If $(\mathbf{w}, w_0, M)$ is an optimal solution, then so is $(\beta\mathbf{w}, \beta w_0, M)$ for any $\beta > 0$.

Adding a constraint

- Let's try adding the constraint that $\|\mathbf{w}\| M = 1$.
- This allows us to rewrite the objective function and constraints as:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\| \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- This is really nice because the constraints are linear.
- The objective function $\|\mathbf{w}\|$ is still a bit awkward.

Final formulation

- Let's maximize $\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|$.
  (Taking the square is a monotone transformation, as $\|\mathbf{w}\|$ is postive, so this doesn't change the optimal solution.)
- This gets us to:

$$\begin{array}{rl} \text{min} & \|\mathbf{w}\|^2 \\ \text{w.r.t.} & \mathbf{w}, w_0 \\ \text{s.t.} & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{array}$$

- This we can solve! How?
  - It is a *quadratic programming* (QP) problem—a standard type of optimization problem for which many efficient packages are available.
  - Better yet, it's a convex (positive semidefinite) QP

https://en.wikipedia.org/wiki/Quadratic_programming

Quadratic programming

The quadratic programming problem with n variables and m constraints can be formulated as follows.[1] Given:

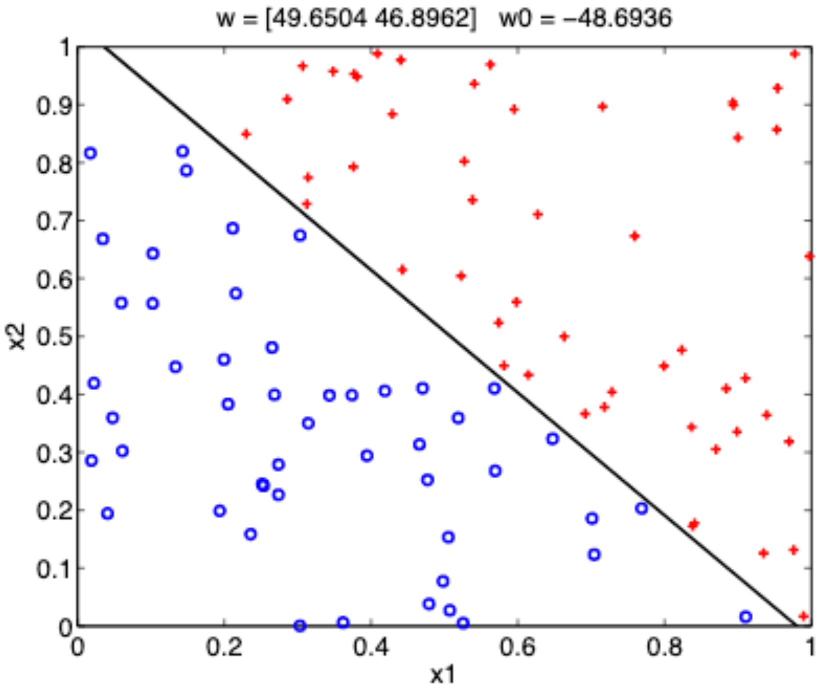
- a real-valued, n-dimensional vector $\mathbf{c}$,
- an $n \times n$-dimensional real symmetric matrix Q,
- an $m \times n$-dimensional real matrix A, and
- an m-dimensional real vector $\mathbf{b}$,

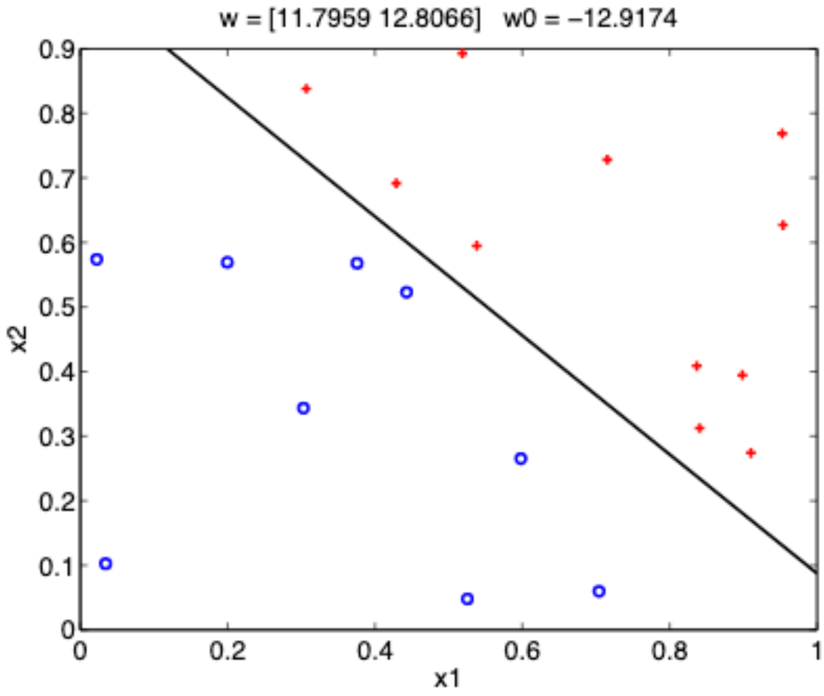
the objective of quadratic programming is to find an n-dimensional vector $\mathbf{x}$, that will

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}\mathbf{x}^{\mathrm{T}} Q\mathbf{x} + \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ &\text{subject to} && A\mathbf{x} \preceq \mathbf{b}, \end{aligned}$$

[1]: https://en.wikipedia.org/wiki/Quadratic_programming

Example


w = [49.6504 46.8962]   w0 = −48.6936
x1
x2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


w = [11.7959 12.8066]   w0 = −12.9174
x1
x2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Lagrange multipliers for inequality constraints

- Suppose we have the following optimization problem, called *primal*:

$$\min_{\mathbf{w}} f(\mathbf{w})$$
$$\text{such that } g_i(\mathbf{w}) \leq 0, \; i = 1 \dots k$$

- We define the *generalized Lagrangian*:

$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}), \tag{1}$$

where $\alpha_i, \; i = 1 \dots k$ are the Lagrange multipliers.

A different formalization

- Consider $\mathcal{P}(\mathbf{w}) = \max_{\alpha:\alpha_i \geq 0} L(\mathbf{w}, \alpha)$
- Observe that the follow is true

$$\mathcal{P}(\mathbf{w}) = \begin{cases} f(\mathbf{w}) & \text{if all constraints are satisfied} \\ +\infty & \text{otherwise} \end{cases}$$

- Hence, instead of computing $\min_{\mathbf{w}} f(\mathbf{w})$ subject to the original constraints, we can compute:

$$p^* = \min_{\mathbf{w}} \mathcal{P}(\mathbf{w}) = \min_{\mathbf{w}} \max_{\alpha:\alpha_i \geq 0} L(\mathbf{w}, \alpha)$$

Dual optimization problem

- Let $d^* = \max_{\alpha : \alpha_i \geq 0} \min_{\mathbf{w}} L(\mathbf{w}, \alpha)$ (max and min are reversed)
- We can show that $d^* \leq p^*$.
  - Let $p^* = L(w^p, \alpha^p)$
  - Let $d^* = L(w^d, \alpha^d)$
  - Then $d^* = L(w^d, \alpha^d) \leq L(w^p, \alpha^d) \leq L(w^p, \alpha^p) = p^*$.

Dual optimization problem

- If f, g_i are convex and the g_i can all be satisfied simultaneously for some $\mathbf{w}$, then we have equality: $d^* = p^* = L(\mathbf{w}^*, \alpha^*)$
- Moreover $\mathbf{w}^*, \alpha^*$ solve the primal and dual if and only if they satisfy the following conditions (called Karush-Kunh-Tucker):

$$\begin{aligned}
\frac{\partial}{\partial w_i} L(\mathbf{w}^*, \alpha^*) &= 0,\ i = 1 \dots n && (2)\\
\alpha_i^* g_i(\mathbf{w}^*) &= 0,\ i = 1 \dots k && (3)\\
g_i(\mathbf{w}^*) &\leq 0,\ i = 1 \dots k && (4)\\
\alpha_i^* &\geq 0,\ i = 1 \dots k && (5)
\end{aligned}$$

Back to maximum margin perceptron

- We wanted to solve (rewritten slightly):

$$\begin{aligned} \text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \leq 0 \end{aligned}$$

- The Lagrangian is:

$$L(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_i \alpha_i (1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0))$$

- The primal problem is: $\min_{\mathbf{w}, w_0} \max_{\alpha : \alpha_i \geq 0} L(\mathbf{w}, w_0, \alpha)$
- We will solve the dual problem: $\max_{\alpha : \alpha_i \geq 0} \min_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \alpha)$
- In this case, the optimal solutions coincide, because we have a quadratic objective and linear constraints (both of which are convex).

Solving the dual

- From KKT (2), the derivatives of $L(\mathbf{w}, w_0, \alpha)$ wrt $\mathbf{w}, w_0$ should be 0
- The condition on the derivative wrt w_0 gives $\sum_i \alpha_i y_i = 0$
- The condition on the derivative wrt $\mathbf{w}$ gives:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x_i}$$

⇒ Just like for the perceptron with zero initial weights, the optimal solution for $\mathbf{w}$ is a linear combination of the $\mathbf{x}_i$, and likewise for w_0.

- The output is

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + w_0\right)$$

⇒ Output depends on weighted dot product of input vector with training examples

Solving the dual

- By plugging these back into the expression for L, we get:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

with constraints: $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

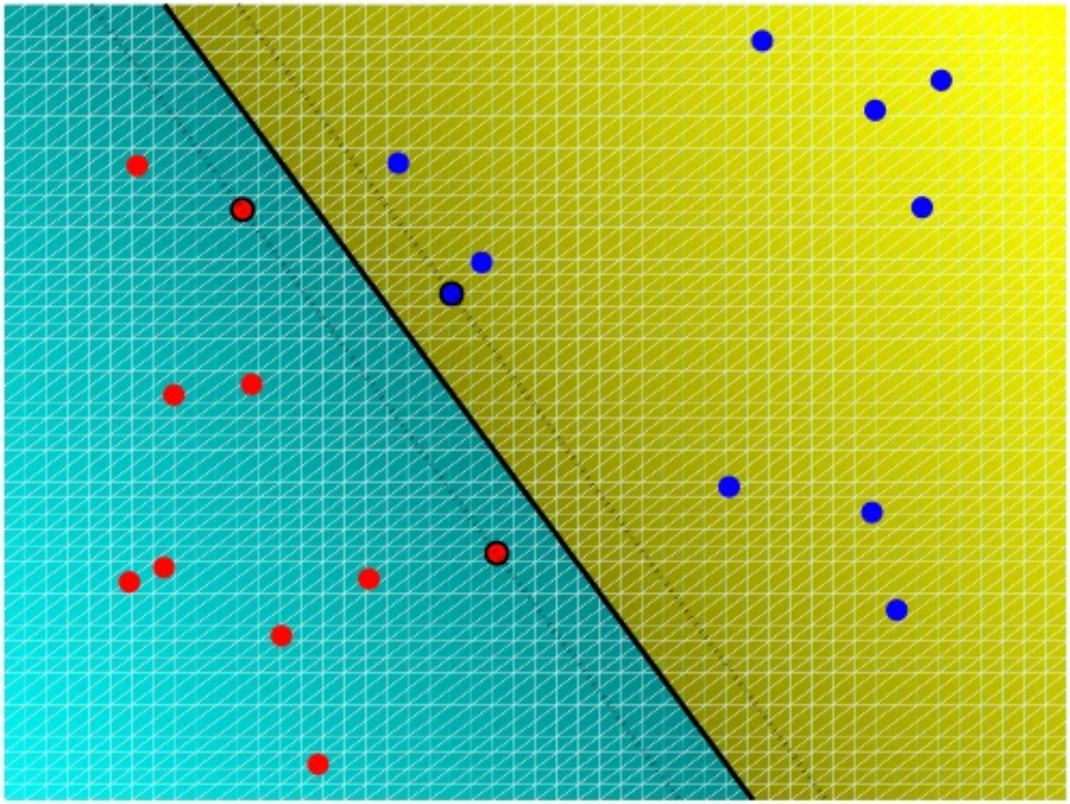
The support vectors

- Suppose we find optimal αs (e.g., using a standard QP package)
- The α_i will be > 0 only for the points for which $1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) = 0$
- These are the points lying on the edge of the margin, and they are called *support vectors*, because they define the decision boundary
- The output of the classifier for query point $\mathbf{x}$ is computed as:

$$\text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + w_0\right)$$

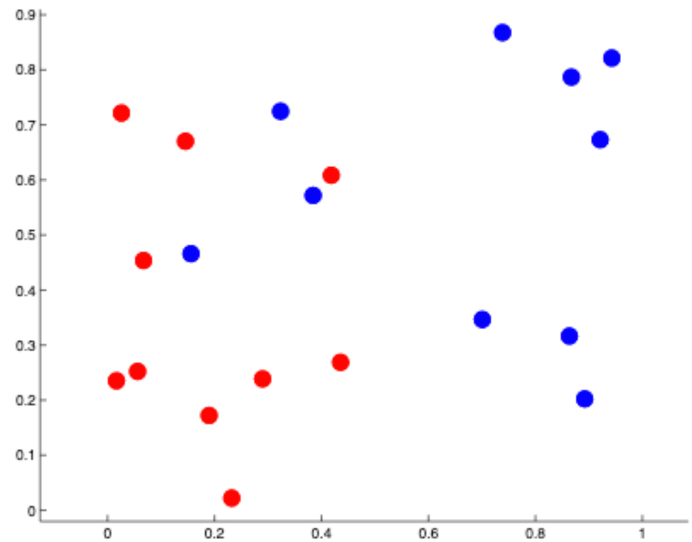
Hence, the output is determined by computing the *dot product of the point with the support vectors*!

Example



Support vectors are in bold

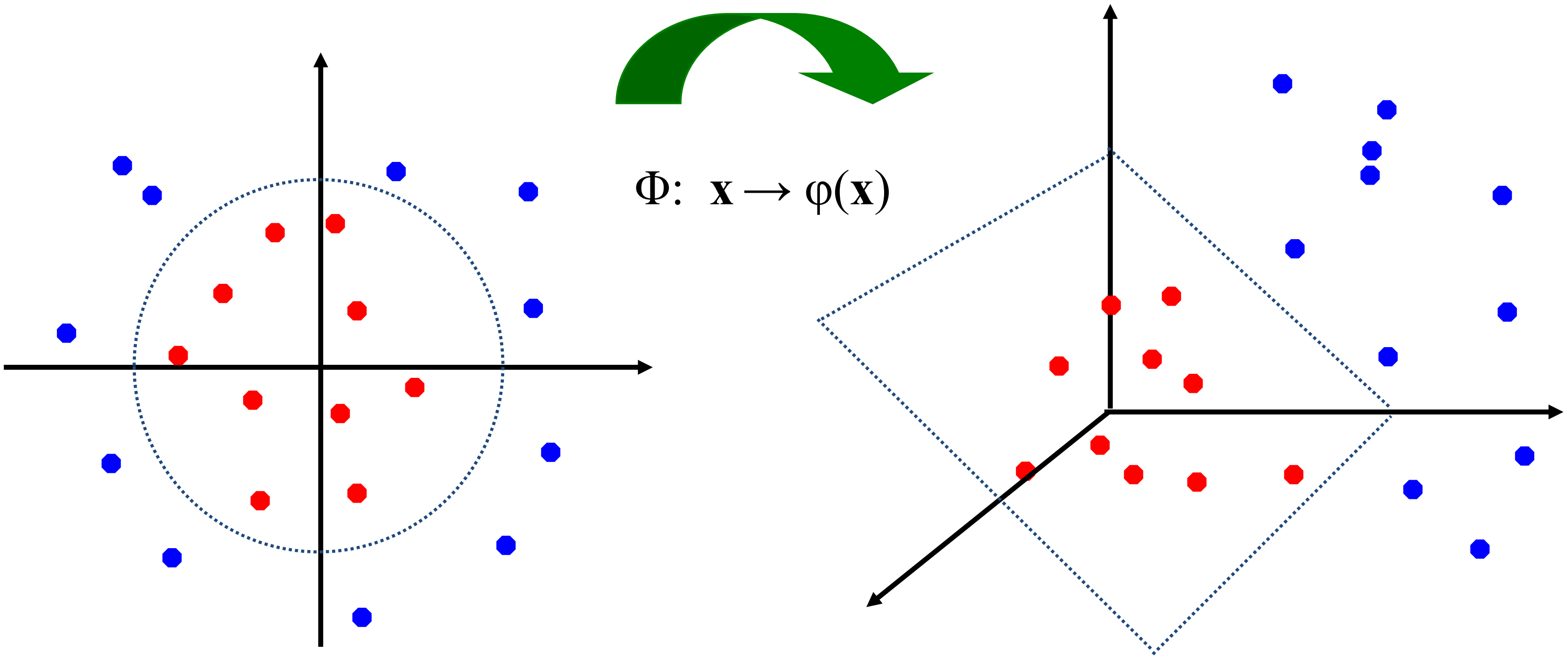
Non-linearly separable data



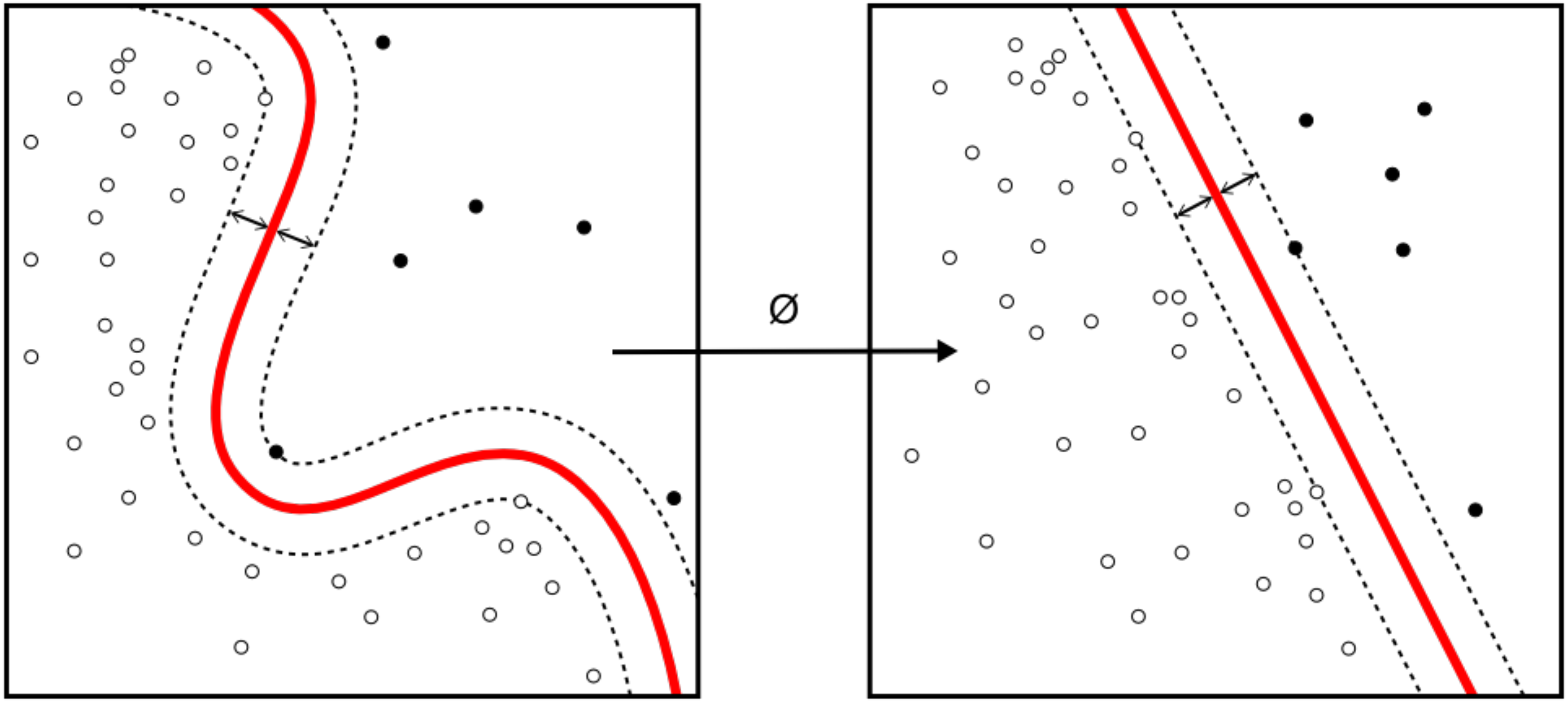
- A linear boundary might be too simple to capture the class structure.
- One way of getting a nonlinear decision boundary in the input space is to find a linear decision boundary in an expanded space

- Thus, $\mathbf{x}_i$ is replaced by $\phi(\mathbf{x}_i)$, where ϕ is called a *feature mapping*

Non-linear SVMs: Feature Space



Non-linear SVMs: Feature Space



Margin optimization in feature space

- Replacing $\mathbf{x}_i$ with $\phi(\mathbf{x}_i)$, the optimization problem to find $\mathbf{w}$ and w_0 becomes:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & \mathbf{y}_i(\mathbf{w} \cdot \phi(\mathbf{x}_i) + w_0) \geq 1 \end{aligned}$$

- Dual form:

$$\begin{aligned} \text{max} \quad & \textstyle\sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \mathbf{y}_i \mathbf{y}_j \alpha_i \alpha_j \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \\ \text{w.r.t.} \quad & \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \\ & \textstyle\sum_{i=1}^m \alpha_i \mathbf{y}_i = 0 \end{aligned}$$

Feature space solution

- The optimal weights, in the expanded feature space, are $\mathbf{w} = \sum_{i=1}^{m} \alpha_i \mathbf{y}_i \phi(\mathbf{x}_i)$.
- Classification of an input $\mathbf{x}$ is given by:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i \mathbf{y}_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + w_0\right)$$

$\Rightarrow$ Note that to solve the SVM optimization problem in dual form and to make a prediction, we only ever need to compute *dot-products of feature vectors*.

Kernel functions

- Whenever a learning algorithm (such as SVMs) can be written in terms of dot-products, it can be generalized to kernels.
- A *kernel* is any function $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ which corresponds to a dot product for some feature mapping ϕ:

$$K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) \text{ for some } \phi.$$

- Conversely, by choosing feature mapping ϕ, we implicitly choose a kernel function
- Recall that $\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = \cos \angle(\mathbf{x}_1, \mathbf{x}_2)$ where $\angle$ denotes the angle between the vectors, so a kernel function can be thought of as a notion of *similarity*.

The "kernel trick"

- If we work with the dual, we do not actually have to ever compute the feature mapping ϕ. We just have to compute the similarity K.
- That is, we can solve the dual for the α_i:

$$\begin{array}{rl} \text{max} & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \mathbf{y}_i \mathbf{y}_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{w.r.t.} & \alpha_i \\ \text{s.t.} & 0 \leq \alpha_i \\ & \sum_{i=1}^{m} \alpha_i \mathbf{y}_i = 0 \end{array}$$

- The class of a new input $\mathbf{x}$ is computed as:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sign}\left((\sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x}_i)) \cdot \phi(\mathbf{x}) + w_0 \right) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + w_0 \right)$$

- Often, $K(\cdot, \cdot)$ can be evaluated in $O(n)$ time—a big savings!

Nonlinear SVMs: The Kernel Trick

- An example:

2-dimensional vectors $\mathbf{x}=[x_1\ \ x_2]$;

let $K(\mathbf{u},\mathbf{v})=(1 + \mathbf{u}^T\mathbf{v})^2$,

Need to show that $K(\mathbf{u},\mathbf{v}) = \varphi(\mathbf{u})^T\varphi(\mathbf{v})$:

$$K(\mathbf{u},\mathbf{v})=(1 + \mathbf{u}^T\mathbf{v})^2,$$

$$= 1+ u_1^2v_1^2 + 2\, u_1v_1\, u_2v_2+ u_2^2v_2^2 + 2u_1v_1 + 2u_2v_2$$

$$= [1\ \ u_1^2\ \ \sqrt{2}\, u_1u_2\ \ u_2^2\ \ \sqrt{2}u_1\ \ \sqrt{2}u_2]^T\, [1\ \ v_1^2\ \ \sqrt{2}\, v_1v_2\ \ v_2^2\ \ \sqrt{2}v_1\ \ \sqrt{2}v_2]$$

$$= \varphi(\mathbf{u})^T\varphi(\mathbf{v}), \quad \text{where } \varphi(\mathbf{x}) = [1\ \ x_1^2\ \ \sqrt{2}\, x_1x_2\ \ x_2^2\ \ \sqrt{2}x_1\ \ \sqrt{2}x_2]$$

This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Nonlinear SVMs: The Kernel Trick

- Examples of commonly-used kernel functions:
  - Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
  - Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
  - Gaussian (Radial-Basis Function (RBF)) kernel:

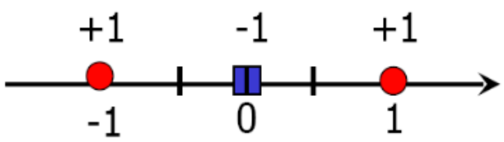
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2}{2\sigma^2})$$

  - Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

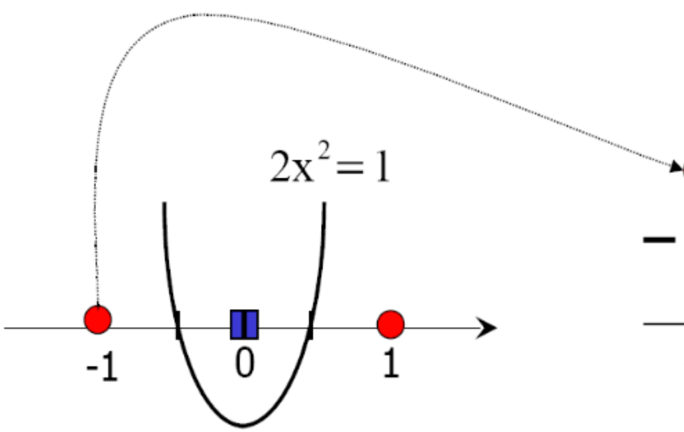
- In general, functions that satisfy *Mercer's condition* can be kernel functions: Kernel matrix should be positive semidefinite.

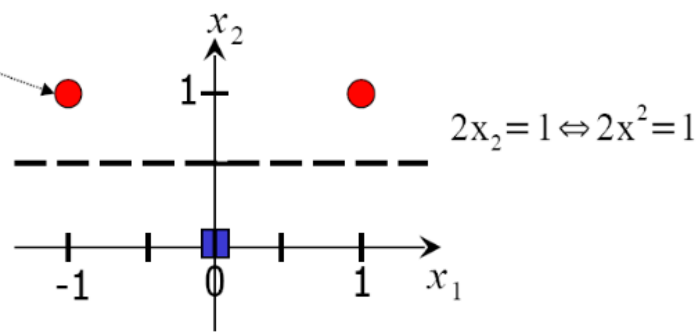
Example



- **Solutions:**
  1) Nonlinear classifiers
  2) **Increase dimensionality** of dataset and add a **non-linear mapping Φ**

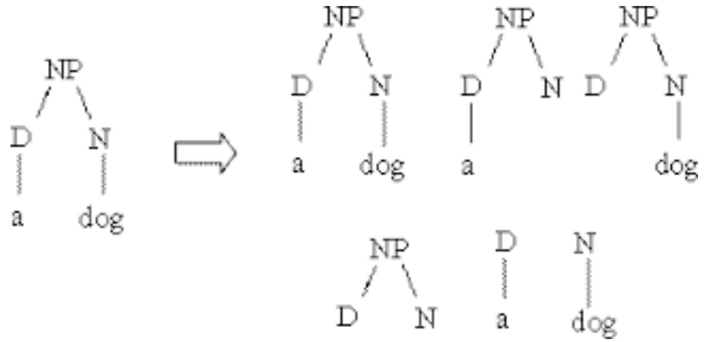
$$[x] \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x \\ x^2 \end{bmatrix}$$





Example: String kernel

- Very important for DNA matching, text classification, ...
- Example: in DNA matching, we use a sliding window of length k over the two strings that we want to compare
- The window is of a given size, and inside we can do various things:
  - Count exact matches
  - Weigh mismatches based on how bad they are
  - Count certain markers, e.g. AGT
- The kernel is the sum of these similarities over the two sequences
- How do we prove this is a kernel?

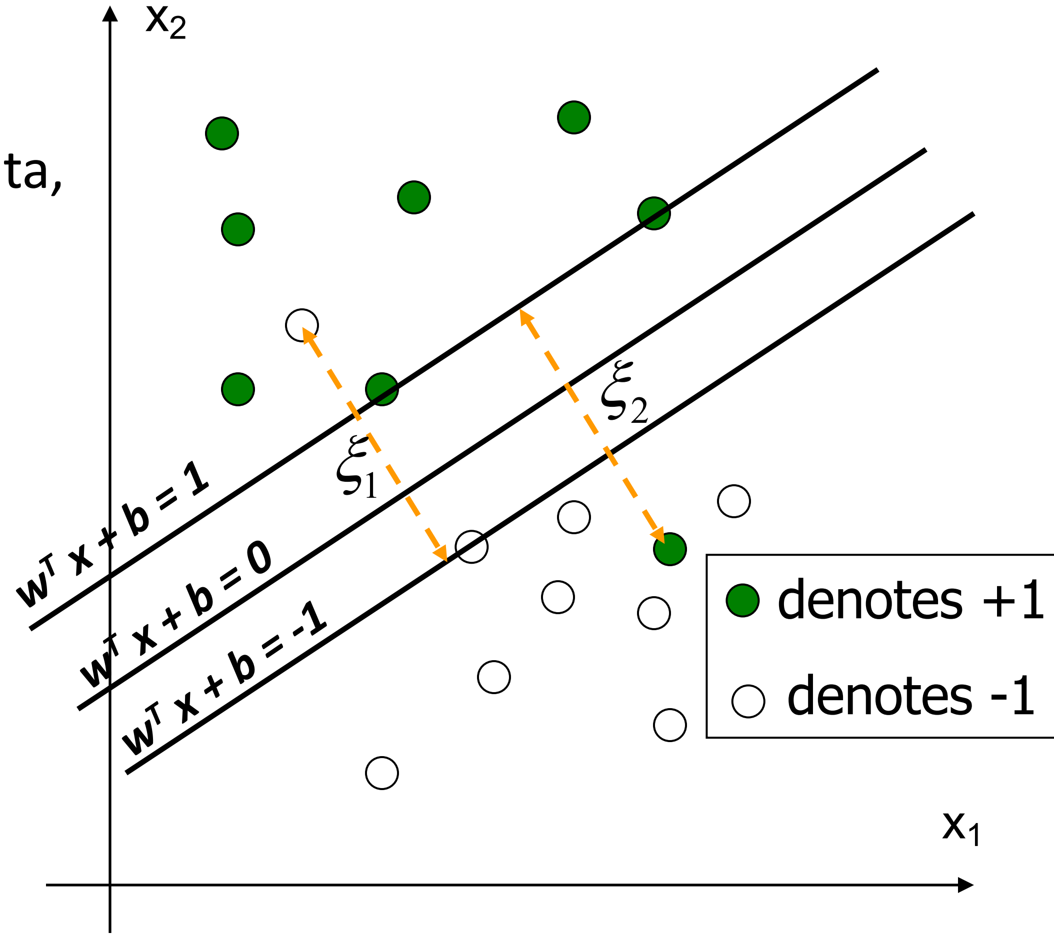


Regularization with SVMs

- Kernels are a powerful tool for allowing non-linear, complex functions
- But now the number of parameters can be as high as the number of instances!
- With a very specific, non-linear kernel, each data point may be in its own partition
- With linear and logistic regression, we used regularization to avoid overfitting
- We need a method for allowing regularization with SVMs as well.

Soft margin linear classifier

- For the data that is not linearly separable (noisy data, outliers, etc.)
- Slack variables ξ_i can be added to allow mis-classification of difficult or noisy data points



Soft margin classifiers

- Recall that in the linearly separable case, we compute the solution to the following optimization problem:

$$\begin{aligned} \text{min} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- If we want to allow misclassifications, we can relax the constraints to:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \xi_i$$

- If $\xi_i \in (0, 1)$, the data point is within the margin
- If $\xi_i \geq 1$, then the data point is misclassified
- We define the *soft error* as $\sum_i \xi_i$
- We will have to change the criterion to reflect the soft errors

New problem formulation with soft errors

- Instead of:

$$\begin{aligned} \text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

  we want to solve:

$$\begin{aligned} \text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C \textstyle\sum_i \xi_i \\ \text{w.r.t.} \quad & \mathbf{w}, w_0, \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \xi_i,\ \xi_i \geq 0 \end{aligned}$$

- Note that soft errors include points that are misclassified, as well as points within the margin
- There is a linear penalty for both categories
- The choice of the *constant C controls overfitting*

A built-in overfitting framework

$$
\begin{aligned}
\text{min} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C\textstyle\sum_i \xi_i \\
\text{w.r.t.} \quad & \mathbf{w}, w_0, \xi_i \\
\text{s.t.} \quad & y_i(\mathbf{w}\cdot\mathbf{x}_i + w_0) \geq 1 - \xi_i \\
& \xi_i \geq 0
\end{aligned}
$$

- If C is 0, there is no penalty for soft errors, so the focus is on maximizing the margin, even if this means more mistakes
- If C is very large, the emphasis on the soft errors will cause decreasing the margin, if this helps to classify more examples correctly.
- Internal cross-validation is a good way to choose C appropriately

Lagrangian for the new problem

- Like before, we can write a Lagrangian for the problem and then use the dual formulation to find the optimal parameters:

$$\begin{aligned} L(\mathbf{w}, w_0, \alpha, \xi, \mu) &= \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i \\ &+ \sum_i \alpha_i \left(1 - \xi_i - y_i(\mathbf{w}_i \cdot \mathbf{x}_i + w_0)\right) + \sum_i \mu_i \xi_i \end{aligned}$$

- All the previously described machinery can be used to solve this problem
- Note that in addition to α_i we have coefficients μ_i, which ensure that the errors are positive, but do not participate in the decision boundary

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j))$$

with constraints: $0 \le \alpha_i \le C$ and $\sum_i \alpha_i y_i = 0$

Soft margin optimization with kernels

- Replacing $\mathbf{x}_i$ with $\phi(\mathbf{x}_i)$, the optimization problem to find $\mathbf{w}$ and w_0 becomes:

$$\begin{array}{rl} \text{min} & \|\mathbf{w}\|^2 + C\sum_i \zeta_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \zeta_i \\ \text{s.t.} & \mathbf{y}_i(\mathbf{w}\cdot\phi(\mathbf{x}_i) + w_0) \geq (1-\zeta_i) \\ & \zeta_i \geq 0 \end{array}$$

- Dual form and solution have similar forms to what we described last time, but in terms of kernels

Getting SVMs to work in practice

- Two important choices:
  - Kernel (and kernel parameters)
  - Regularization parameter C
- The parameters may interact!

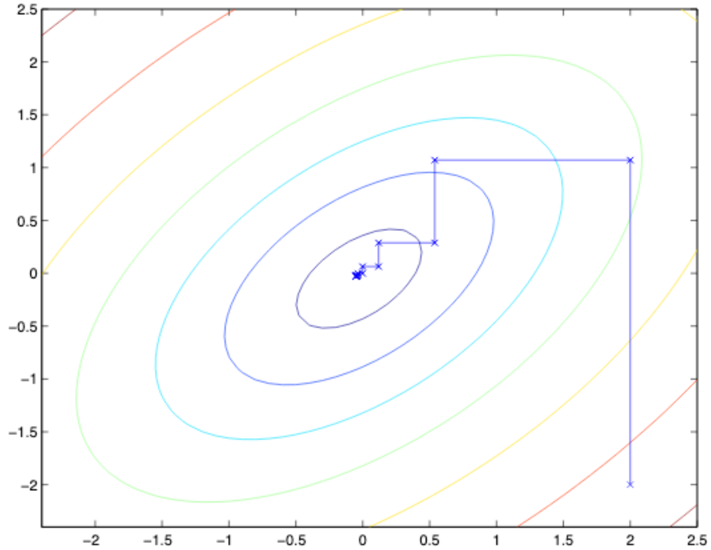
  E.g. for Gaussian kernel, the larger the width of the kernel, the more biased the classifier, so low C is better
- Together, these control overfitting: always do an internal parameter search, using a validation set!
- Overfitting symptoms:
  - Low margin
  - Large fraction of instances are support vectors

Solving the quadratic optimization problem

- Many approaches exist
- Because we have constraints, gradient descent does not apply directly (the optimum might be outside of the feasible region)
- Platt's algorithm is the fastest current approach, based on *coordinate ascent*

Coordinate ascent

- Suppose you want to find the maximum of some function $F(\alpha_1, \ldots \alpha_n)$
- Coordinate ascent optimizes the function by repeatedly picking an α_i and optimizing it, while all other parameters are fixed
- There are different ways of looping through the parameters:
  - Round-robin
  - Repeatedly pick a parameter at random
  - Choose next the variable expected to make the largest improvement
  - ...



Our optimization problem (dual form)

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j))$$

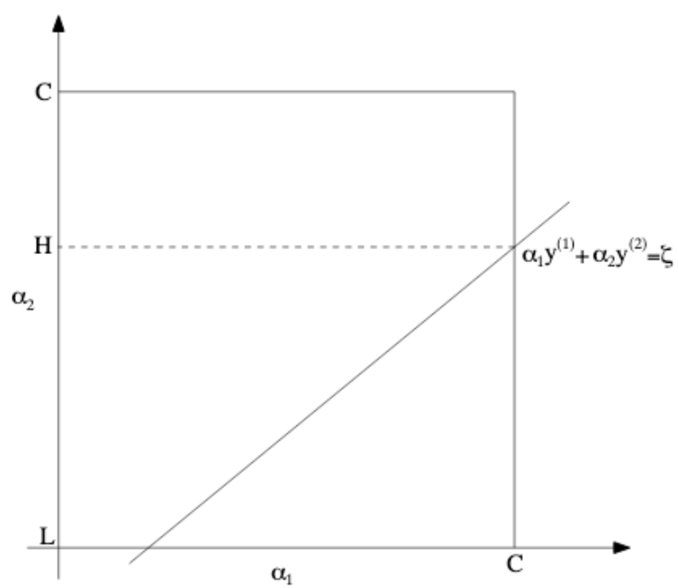
with constraints: $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i y_i = 0$

- Suppose we want to optimize for α_1 while $\alpha_2, \ldots \alpha_n$ are fixed
- We cannot do it because α_1 will be completely determined by the last constraint: $\alpha_1 = -y_1 \sum_{i=2}^{m} \alpha_i y_i$
- Instead, we have to optimize *pairs of parameters* α_i, α_j together

Sequential minimal optimization (SMO)

- Suppose that we want to optimize α_1 and α_2 together, while all other parameters are fixed.
- We know that $y_1\alpha_1 + y_2\alpha_2 = -\sum_{i=1}^{m} y_i\alpha_i = \xi$, where ξ is a constant
- So $\alpha_1 = y_1(\xi - y_2\alpha_2)$ (because y_1 is either $+1$ or -1 so $y_1^2 = 1$)
- This defines a line, and any pair α_1, α_2 which is a solution has to be on the line

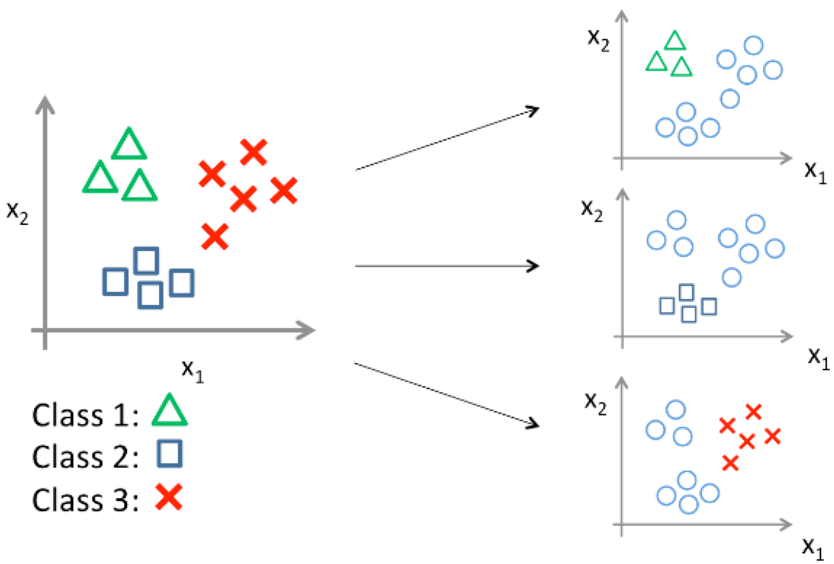
- We also know that $0 \leq \alpha_1 \leq C$ and $0 \leq \alpha_2 \leq C$, so the solution has to be on the line segment inside the rectangle below

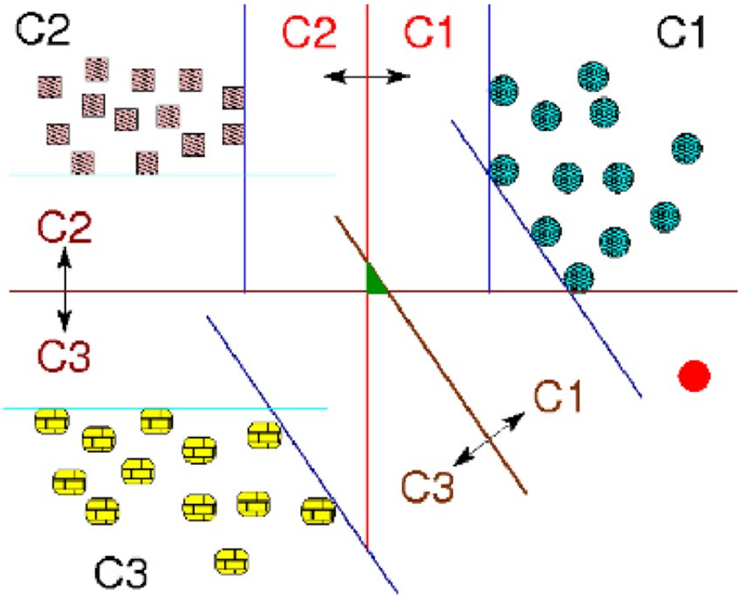


Sequential minimal optimization (SMO)

- By plugging α_1 back in the optimization criterion, we obtain a quadratic function of α_2, whose optimum we can find exactly
- If the optimum is inside the rectangle, we take it.
- If not, we pick the closest intersection point of the line and the rectangle
- This procedure is very fast because all these are simple computations.

Multi-class classification





- one-vs-all
- n classifiers
- choose the class with the largest margin

- one-vs-one
- $\frac{n(n-1)}{2}$ classifiers
- choose the class chosen by most classifiers

Complexity

- Quadratic programming is expensive in the number of training examples
- Platt's SMO algorithm is quite fast though, and other fancy optimization approaches are available
- Best packages can handle $50,000+$ instances, but not more than $100,000$
- On the other hand, number of attributes can be very high (strength compared to neural nets)
- Evaluating a SVM is *slow if there are a lot of support vectors*.
- Dictionary methods attempt to select a subset of the data on which to train.