Support Vector Machines

Jie Tang

Department of Computer Science and Technology
Tsinghua University

Outline

- What is a Support Vector Machine?
- Solving SVMs
- Kernel Tricks

What is a Support Vector Machine

- SVM is related to statistical learning theory [3]
- SVM was first introduced in 1992 [1]
- SVM becomes popular because of its success in handwritten digit recognition
  - 1.1% test error rate for SVM. This is the same as the error rates of a carefully constructed neural network, LeNet4.
    - See Section 5.11 in [2] or the discussion in [3] for details
- SVM is now regarded as an important example of "kernel methods", one of the key area in machine learning

[1] B.E. Boser *et al*. A Training Algorithm for Optimal Margin Classifiers. Proceedings of the Fifth Annual Workshop on Computational Learning Theory 5 144-152, Pittsburgh, 1992.
[2] L. Bottou *et al*. Comparison of classifier methods: a case study in handwritten digit recognition. Proceedings of the 12th IAPR International Conference on Pattern Recognition, vol. 2, pp. 77-82, 1994.
[3] V. Vapnik. The Nature of Statistical Learning Theory. 2nd edition, Springer, 1999.

Classification Problem

- Given a training set $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, and $x_i \in X = R^m, i = 1, \ldots, N$
- To learn a function $g(x)$, and make the decision function $f(x) = \text{sgn}(g(x))$ can classify new input x
- So this is a supervised batch learning method
- Linear classifier

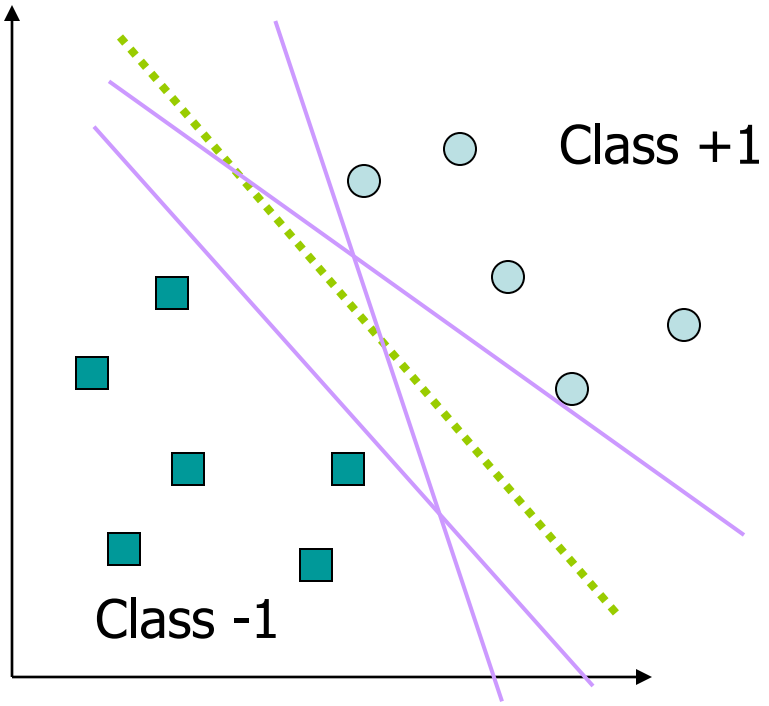
$$g(x) = (w^T x + b)$$

$$\text{sgn}(g(x)) = \begin{cases} 1, g(x) \geq 0 \\ -1, g(x) < 0 \end{cases}$$

$$f(x) = \text{sgn}(g(x))$$

What is a good Decision Boundary?

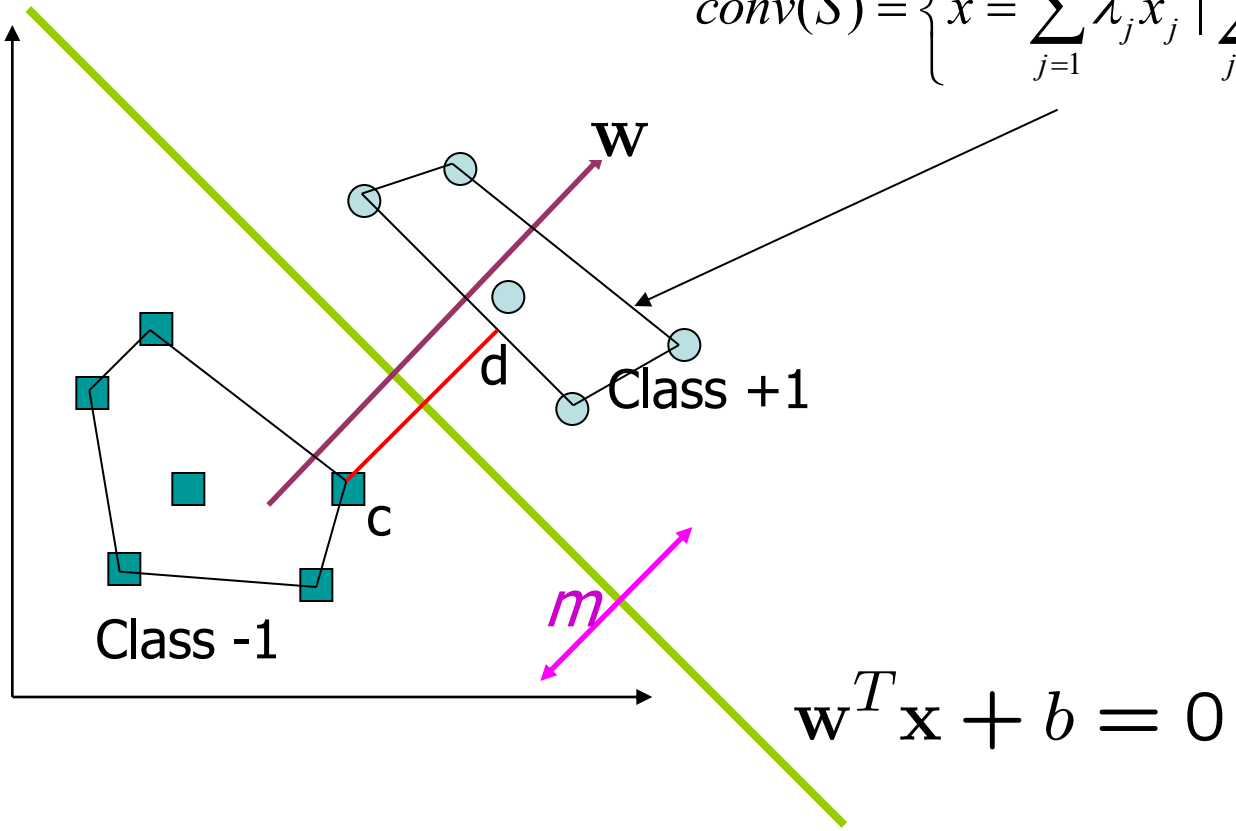
- Consider a two-class, linearly separable classification problem
- Many decision boundaries!
  - The Perceptron algorithm can be used to find such a boundary
  - Different algorithms have been proposed
- Are all decision boundaries equally good?



Geometric Interpretation

Bisector based Decision Boundary

$$conv(S) = \left\{ x = \sum_{j=1}^{k} \lambda_j x_j \mid \sum_{j=1}^{k} \lambda_j = 1, \lambda_j \geq 0, j = 1, \cdots, k \right\}$$



Formalization

$$\min_\beta \frac{1}{2}\|c-d\| \Rightarrow \min_\beta \frac{1}{2}\left\|\sum_{y_i=1}\beta_i x_i - \sum_{y_j=-1}\beta_j x_j\right\|$$

$$s.t. \qquad \sum_{y_i=1}\beta_i = 1, \sum_{y_j=-1}\beta_j = 1$$

$$0 \le \beta_i \le 1, i = [1, m]$$

The objective is to solve all the β_i.

Then we can obtain the two points having the closest distance by

$$c = \sum_{y_i=1}\beta_i x_i \qquad d = \sum_{y_j=-1}\beta_j x_j$$

Next we compute the hyperplane $w^T x + b = 0$ by

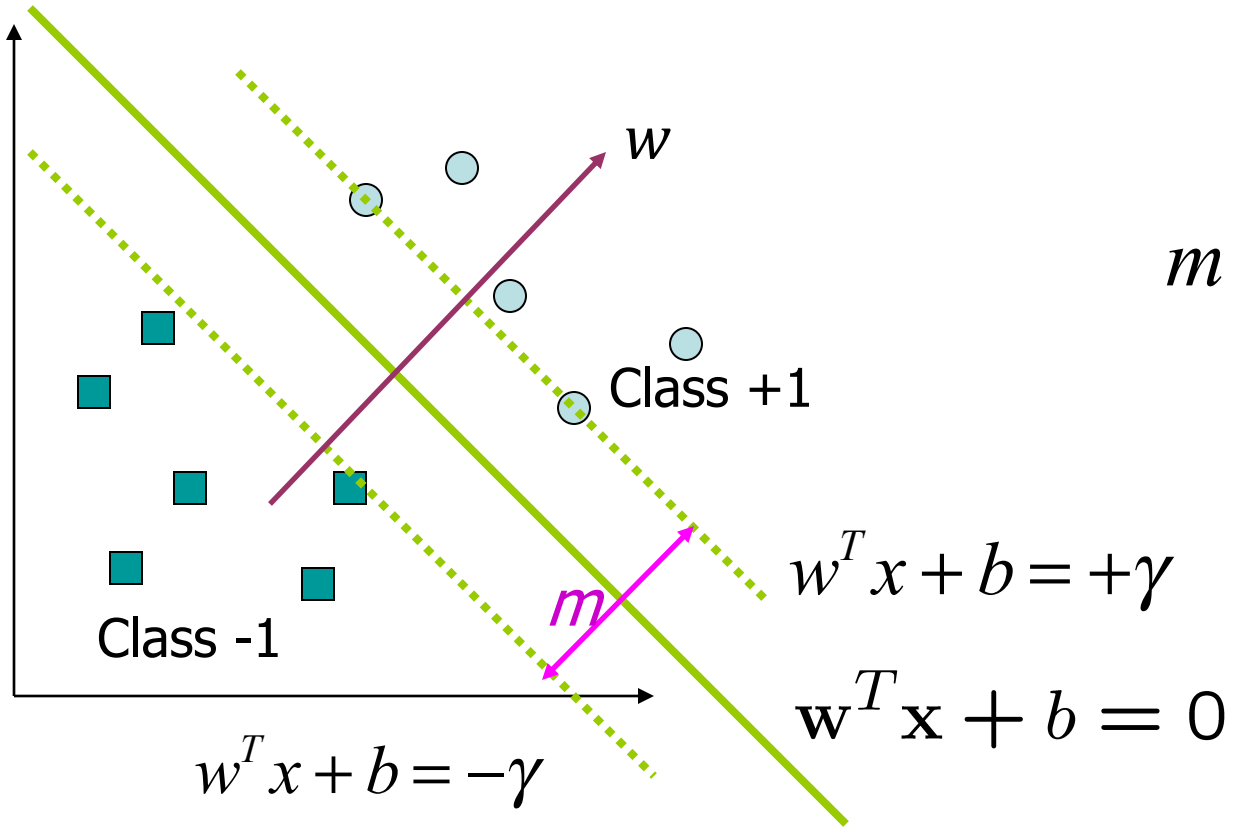
$$w = c - d = \sum_i^m \beta_i y_i x_i \qquad b = -1/2((c-d)(c+d))$$

Finally, we make prediction by $f(x) = \text{sgn}((w^T x) + b)$

Maximal Margin

Large-margin Decision Boundary

- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin, *m*
  - Distance between the origin and the line $w^T x = -b$ is $-b/||w||$



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

Formalization

$$\max_{\gamma,w,b} \frac{|2\gamma|}{\|w\|}$$

equal to

$$\min_{\gamma,w,b} \frac{\|w\|}{|2\gamma|}$$

Note: we have constraints

$$s.t. \quad w^T x^{(i)} + b \geq \gamma, 1 \leq i \leq k \quad , y_i = 1$$

$$w^T x^{(j)} + b < -\gamma, k < j \leq N \quad , y_j = -1$$

equal to

$$y^{(i)}(w^T x^{(i)} + b) \geq \gamma, 1 \leq i \leq N$$

Since we can arbitrarily scale *w* and *b* without changing anything,
We introduce the scaling constraint "$\gamma=1$"

$$\min_{w,b} \frac{\|w\|}{2}$$

Change to 2-norm space

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

loss function

$$s.t. \quad y^{(i)}(w^T x^{(i)} + b) \geq 1, 1 \leq i \leq N$$

This is a constrained optimization problem.

Loss Function

- Then we result in the Lasso loss function

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

$$s.t. \qquad y^{(i)}(b + w^T x^{(i)}) \geq 1$$

- Another popular loss function: Hinge loss + penalty

$$\min_{w,b} \sum_{i=1}^{N} [1 - y^{(i)}(b + w^T x)]_+ + \frac{\lambda}{2} \| w \|^2$$

Loss Function (cont.)

- Empirical loss function

$$\min_{w,b} \sum_{i=1}^{N} [1 - y^{(i)}(b + wx^{(i)})]_+$$

- Structural loss function

$$\min_{w,b} \sum_{i=1}^{N} [1 - y^{(i)}(b + wx^{(i)})]_+ + \frac{\lambda}{2}\|w\|^2$$

where $||w||^2$ indicates the complexity of the model, it is also named as penalty

- There are many kinds of formulation of the loss function

Optimal Margin Classifiers

For the problem:

$$\min_{w,b} \frac{\|w\|^2}{2}$$

$$s.t. \qquad y^{(i)}(w^T x^{(i)} + b) \geq 1, 1 \leq i \leq N$$

We can write lagrangian form:

$$L(w,b,\alpha) = \frac{\|w\|^2}{2} - \sum_{i=0}^{N} \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1]$$

$$s.t. \qquad \alpha_i \geq 0, 1 \leq i \leq N$$

WHY? Let us review generalized Lagrange



Review Convex Optimization and Lagrange Duality

Convex Function

- f: R^n $->$ R is convex if $dom(f)$ is a convex set and

  $$f(\theta x + (1-\theta)y) \le \theta f(x) + (1-\theta) f(y)$$

  for all $x, y \in dom(f), 0 \le \theta \le 1$



  - f is concave if $-f$ is convex
  - f is strictly convex if $dom(f)$ is convex and

    $$f(\theta x + (1-\theta)y) < \theta f(x) + (1-\theta) f(y)$$

    for all $x, y \in dom(f), x \ne y, 0 \le \theta \le 1$

First-order Condition

- f is differentiable if $dom(f)$ is open and the gradient

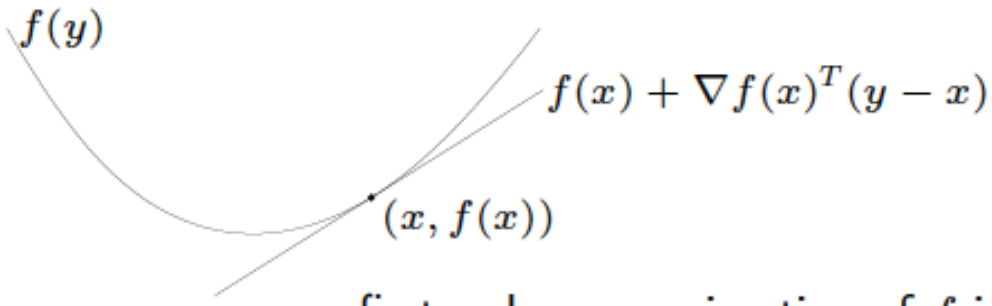
$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n} \right)$$

exists at each $x \in dom(f)$

- 1st-order condition: differentiable f with convex domain is convex iff

$$f(y) \geq f(x) + \nabla f(x)^T * (y - x)$$

for all $x, y \in dom(f)$



first-order approximation of f is global underestimator

Second-order Condition

- f is twice differentiable if $dom(f)$ is open and the Hessian

$$\nabla^2 f(x)_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}, \qquad i, j = 1, \ldots, N,$$

  exists at each $x \in dom(f)$

- 2nd-order condition: for twice differentiable *f* with convex domain
  - f is convex iff

$$\nabla^2 f(x) \geq 0 \qquad \text{for all} \quad x \in dom(f)$$

  - If $\nabla^2 f(x) > 0$ for all $x \in dom(f)$, then f is strictly convex.

Convex Optimization Problem

- Standard form convex optimization problem

$$\begin{aligned} \min \quad & f_0(x) \\ s.t. \quad & f_i(x) \geq 0, i = 1, \ldots, k \\ & a^T{}_i x - b_i = 0, i = 1, \ldots, l \end{aligned}$$

needs to satisfy

- $f_0, f_1, \ldots, f_k$ are convex;
- Equality constraints are affine
- Important property: feasible set of a convex optimization problem is convex

Example

- Convex quadratic minimization with linear constraints is a convex optimization problem:

$$\begin{aligned} \min \quad & \frac{1}{2} x^T Q x + c^T x \\ \text{subject to} \quad & Ax \leq b \end{aligned}$$

Because:

- $f_0(x) = \frac{1}{2} x^T Q x + c^T x$ is convex
- feasible set $\{x \mid Ax - b \leq 0\}$ is convex
- $f_1(x) = -(Ax - b)$ is convex

Lagrange Duality

- When solving optimization problems with constraints, lagrange duality is always used to obtain the solution of the primal problem through solving the dual problem.

- Primal optimization problem
  - If $f(x)$, $g_i(x)$, $h_j(x)$ are continuously differentiable functions defined in $\boldsymbol{R}^n$, then the following optimization problem is called primal optimization problem

$$\min_{x \in R^n} f(x)$$

$$s.t. \qquad g_i(x) \geq 0, i = 1, \ldots, k$$

$$h_j(x) = 0, j = 1, \ldots, l$$

Primal Optimization Problem

- To solve the primal optimization problem, we define the generalized Lagrangian:

$$L(x,\alpha,\beta) = f(x) - \sum_{i=1}^{k} \alpha_i g_i(x) - \sum_{j=1}^{l} \beta_j h_j(x)$$

$$s.t. \qquad \alpha_i \geq 0$$

where α_i and β_j are Lagrange multipliers. Consider the function:

$$\theta_P(x) = \max_{\alpha,\beta:\alpha_i \geq 0} L(x,\alpha,\beta)$$

Here "P" stands for "primal".

$$\min_{x \in R^n} f(x) \quad \text{Primal problem}$$

$$s.t. \qquad g_i(x) \geq 0, i = 1,\ldots,k$$

$$h_j(x) = 0, j = 1,\ldots,l$$

- Assume some x violates any of the primal constraints (i.e., if either $g_i(x) < 0$ or $h_j(x) \neq 0$ for some i), then we can verify that

$$\theta_P(x) = \max_{\alpha,\beta:\alpha_i \geq 0} [f(x) - \sum_{i=1}^{k} \alpha_i g_i(x) - \sum_{j=1}^{l} \beta_j h_j(x)] = \infty$$

  - Since if $g_i(x) < 0$ for some i , we can set α_i *as* +∞;
  - if $h_j(x) \neq 0$ for some i , we can set $\beta_j h_j(x)$ *as* +∞, and set other α_i and β_j as 0.

- Therefore:

$$\theta_P(x) = \begin{cases} f(x) & \text{If x satisfies primal constraints} \\ \infty & \text{otherwise} \end{cases}$$

- Here, if we consider the minimization problem

$$\min_x \theta_P(x) = \min_x \max_{\alpha,\beta:\alpha_i \geq 0} L(x,\alpha,\beta)$$

We see that the primal problem is represented by the min max problem of the generalized Lagrangian, i.e.,

$$p^* = \min_x \theta_P(x)$$

Dual Optimization Problem

$$\min_{x \in R^n} f(x) \quad \text{Primal problem}$$
$$s.t. \quad g_i(x) \geq 0, i = 1, \ldots, k$$
$$h_j(x) = 0, j = 1, \ldots, l$$

- Dual optimization problem:

$$\max_{\alpha, \beta : \alpha_i \geq 0} \theta_D(\alpha, \beta) = \max_{\alpha, \beta : \alpha_i \geq 0} \min_x L(x, \alpha, \beta)$$

- This is exactly the same as the primal problem, except that the order of the "max" and the "min" are now exchanged. We also define the optimal value of the dual problem's objective to be

$$d^* = \max_{\alpha, \beta : \alpha_i \geq 0} \theta_D(x)$$

- How are the primal and the dual problems related? It can easily shown that:

$$d^* = \max_{\alpha, \beta : \alpha_i \geq 0} \min_x L(x, \alpha, \beta) \leq \min_x \max_{\alpha, \beta : \alpha_i \geq 0} L(x, \alpha, \beta) = p^*$$

- Proof:

$$\theta_D(\alpha, \beta) = \min_x L(x, \alpha, \beta) \leq L(x, \alpha, \beta) \leq \max_{\alpha, \beta : \alpha_i \geq 0} L(x, \alpha, \beta) = \theta_P(x)$$

So $\theta_D(\alpha, \beta) \leq \theta_P(x)$

Because the primal and dual problem both have the optimal value, thus

$$\max_{\alpha, \beta : \alpha_i \geq 0} \theta_D(\alpha, \beta) \leq \min_x \theta_P(x)$$

i.e., $d^* = \max_{\alpha, \beta : \alpha_i \geq 0} \min_x L(x, \alpha, \beta) \leq \min_x \max_{\alpha, \beta : \alpha_i \geq 0} L(x, \alpha, \beta) = p^*$

KKT Conditions

- Under certain conditions, we will have $d^* = p^*$
- So that we can solve the dual problem in lieu of the primal problem.
- Then what's the conditions?

Suppose

(1) f and g_i are convex, and $h_i(x)$ is affine.

(2) the constraints g_i are (strictly) feasible; this means that there exists some x so that $g_i(x) > 0$ for all i.

Under the above assumptions, there must exist x^*, α^* and β^* so that x^* is the solution to the primal problem, α^*, β^* are the solution to the dual problem and $p^*=d^*=L(x^*, \alpha^*, \beta^*)$. The necessary and sufficient conditions are KKT (Karush-Kuhn-Tucher) conditions:

$$\frac{\partial L(x^*,\alpha^*,\beta^*)}{\partial x_i} = 0, i \in [1, N]$$

$$\frac{\partial L(x^*,\alpha^*,\beta^*)}{\partial \alpha_i} = 0, i \in [1, k]$$

$$\frac{\partial L(x^*,\alpha^*,\beta^*)}{\partial \beta_i} = 0, i \in [1, l]$$

$$\alpha_i^* g_i(x^*) = 0, i \in [1, k]$$

$$g_i(x^*) \geq 0, i \in [1, k]$$

$$\alpha_i^* \geq 0, i \in [1, k]$$

KKT dual complementarity condition.
If $\alpha_i^* > 0$, then $g_i(x) = 0$

Back to Our Optimal Margin Classifiers

Optimal Margin Classifiers

For the problem:

$$\min_{w,b} \frac{\|w\|^2}{2}$$

$$s.t. \qquad y^{(i)}(w^T x^{(i)} + b) \geq 1, 1 \leq i \leq N$$

We can write lagrangian form:

$$L(w,b,\alpha) = \frac{\|w\|^2}{2} - \sum_{i=0}^{N} \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1]$$

$$s.t. \qquad \alpha_i \geq 0, 1 \leq i \leq N$$

Then our problem becomes:

$$\min_{w,b} \max_{\alpha} L(w,b,\alpha)$$

If certain constraints are satisfied, then we have $\max_{\alpha} \min_{w,b} L(w, b, a)$

Solve the Dual Problem

$$\max_{\alpha} \min_{w,b} L(w,b,\alpha) = \frac{\|w\|^2}{2} - \sum_{i=1}^{N} \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1]$$

$$s.t. \qquad \alpha_i \geq 0, 1 \leq i \leq N$$

Let us first solve the inside minimal problem by setting the gradient of $L(w,b,a)$ w.r.t. *w* and b to zero, we have

$$\frac{\partial L(w,b,\alpha)}{\partial w} = w - \sum_{i=1}^{N} \alpha_i y^{(i)} x^{(i)} = 0$$

$$\Rightarrow w = \sum_{i=1}^{N} \alpha_i y^{(i)} x^{(i)}$$

$$\frac{\partial L(w,b,\alpha)}{\partial b} = \sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

Then let us substitute the two equations into $L(w,b,a)$ to solve the maximal problem

Solve the Dual Problem

Now we have: $w = \sum_{i=1}^{N} \alpha_i y^{(i)} x^{(i)}$ and $\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$

Substitute $w = \sum_{i=1}^{N} \alpha_i y^{(i)} x^{(i)}$ back to $L(w, b, \alpha)$

$$L(b,\alpha) = \frac{1}{2}\sum_{i=1}^{N} \alpha_i y^{(i)} x^{(i)} \sum_{j=1}^{N} \alpha_j y^{(j)} x^{(j)} - \sum_{i=1}^{N} \alpha_i [y^{(i)} (\sum_{j=1}^{N} \alpha_j y^{(j)} x^{(j)} x^{(i)} + b) - 1]$$

$$= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)} x^{(j)} - \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i y^{(i)} \alpha_j y^{(j)} x^{(j)} x^{(i)} - b\sum_{i=1}^{N} \alpha_i y^{(i)} + \sum_{i=0}^{N} \alpha_i$$

$$= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)} x^{(j)} - b\sum_{i=1}^{N} \alpha_i y^{(i)} + \sum_{i=1}^{N} \alpha_i$$

because $\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$, we obtain

The new objective function is a function of α_i only

$$L(\alpha) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)} x^{(j)} + \sum_{i=1}^{N} \alpha_i$$

It is known as the dual problem: if we know w, we know all a_i; vice versa

The Dual Problem (cont.)

The original problem, also known as primal problem

$$\min_{w,b} \frac{\|w\|^2}{2} \quad s.t. \quad y^{(i)}(w^T x^{(i)} + b) \geq 1, 1 \leq i \leq N$$

$$\Rightarrow \quad \min_{w,b} \max_{\alpha} L(w,b,\alpha) = \frac{\|w\|^2}{2} - \sum_{i=1}^{N} \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1] \quad s.t. \quad \alpha_i \geq 0, 1 \leq i \leq N$$

The dual problem $\max_{\alpha} \min_{w,b} L(w,b,\alpha)$

$$\max_{\alpha} -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)} x^{(j)} + \sum_{i=1}^{N} \alpha_i$$

$$s.t. \quad \alpha_i > 0, 1 \leq i \leq N; \sum_{i=1}^{N} \alpha_i y_i = 0$$

Properties of a_i when we introduce the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

Relationship between Primal and Dual Problems

$$d^* = \max_{\alpha,\beta} \min_x L(x,\alpha,\beta) \le \min_x \max_{\alpha,\beta} L(x,\alpha,\beta) = p^*$$

Note: if under some conditions, $d^*=p^*$
We can solve the dual problem in lieu of the primal problem

What is the conditions?

The famous KKT conditions
(Karush-Kuhn-Tucker conditions)

What is KKT

$$\frac{\partial L(w^*,b^*,\alpha^*)}{\partial w} = w^* - \sum_{i=1}^{N} \alpha_i^* y^{(i)} x^{(i)} = 0, i \in [1,N] \quad (1)$$

$$\frac{\partial L(w^*,b^*,\alpha^*)}{\partial b} = \sum_{i=1}^{N} \alpha^*_i y^{(i)} = 0 \quad (2)$$

In our case

$$\alpha^*_i (y^{(i)}(w^* x^{(i)} + b^*) - 1) = 0, i \in [1,N] \quad (3)$$

$$y^{(i)}(w^* x^{(i)} + b^*) - 1 \ge 0, i \in [1,N]$$

$$\alpha^*_i \ge 0, i \in [1,N]$$

In Lagrangian formula

$$L(x,\alpha,\beta) = f(x) - \sum_{i=0}^{k} \alpha_i g_i(x) - \sum_{j=0}^{l} \beta_j h_j(x)$$

$$s.t. \quad \alpha_i \ge 0$$

KKT conditions are

$$\frac{\partial L(x^*,\alpha^*,\beta^*)}{\partial x_i} = 0, i \in [1,N]$$

$$\frac{\partial L(x^*,\alpha^*,\beta^*)}{\partial \alpha_i} = 0, i \in [1,k]$$

$$\frac{\partial L(x^*,\alpha^*,\beta^*)}{\partial \beta_i} = 0, i \in [1,l]$$

$$\alpha_i^* g_i(x^*) = 0, i \in [1,k]$$

$$g_i(x^*) \le 0, i \in [1,k]$$

$$\alpha_i^* \ge 0, i \in [1,k]$$

Now We Have

Then, what we have the maximum optimum problem with respect to α:

$$\max_{\alpha} -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^{(i)}y^{(j)}x^{(i)}x^{(j)} + \sum_{i=1}^{N}\alpha_i$$

$$s.t. \qquad \alpha_i > 0, 1 \le i \le N; \sum_{i=1}^{N}\alpha_i y_i = 0$$

This is a quadratic programming (QP) problem, A global maximum of ai can always be found

Then solve w by

$$w = \sum_{i=1}^{N}\alpha_i y^{(i)}x^{(i)}$$

Finally solve b:

Since there is at least one $\alpha_j^*>0$ (if all $\alpha_j^*=0$, from equation (1) we know that $w^*=0$, however $w^*=0$ is not the optimal solution).

Then from equation (3), we know that

$$y^{(j)}(w^*x^{(j)} + b^*) - 1 = 0$$

Because $y^{(j)}y^{(j)}=1$, then

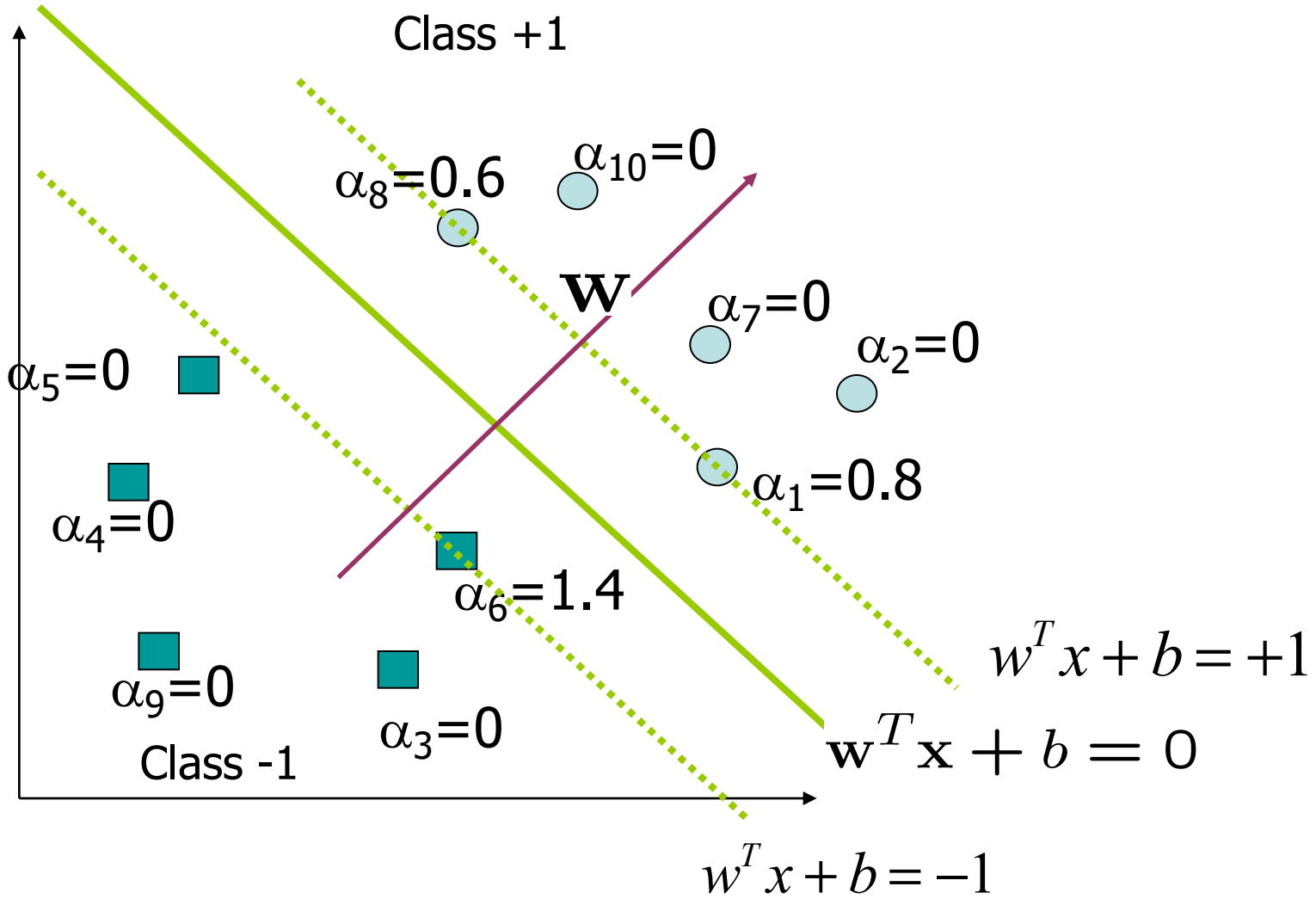
$$b = y^{(j)} - w^*x^{(j)} = y^{(j)} - \sum_{i=1}^{N}\alpha_i y^{(i)}x^{(i)}x^{(j)}$$

Characteristics of the Solution

- Many of the a_i are zero
  - w is a linear combination of a small number of data points
  - This "sparse" representation can be viewed as data compression as in the construction of knn classifier

- x_i with non-zero a_i are called **support vectors (SV)**
  - The decision boundary is determined only by the SV

A Geometrical Interpretation



How to Predict

For a new sample x, we can predict it by:

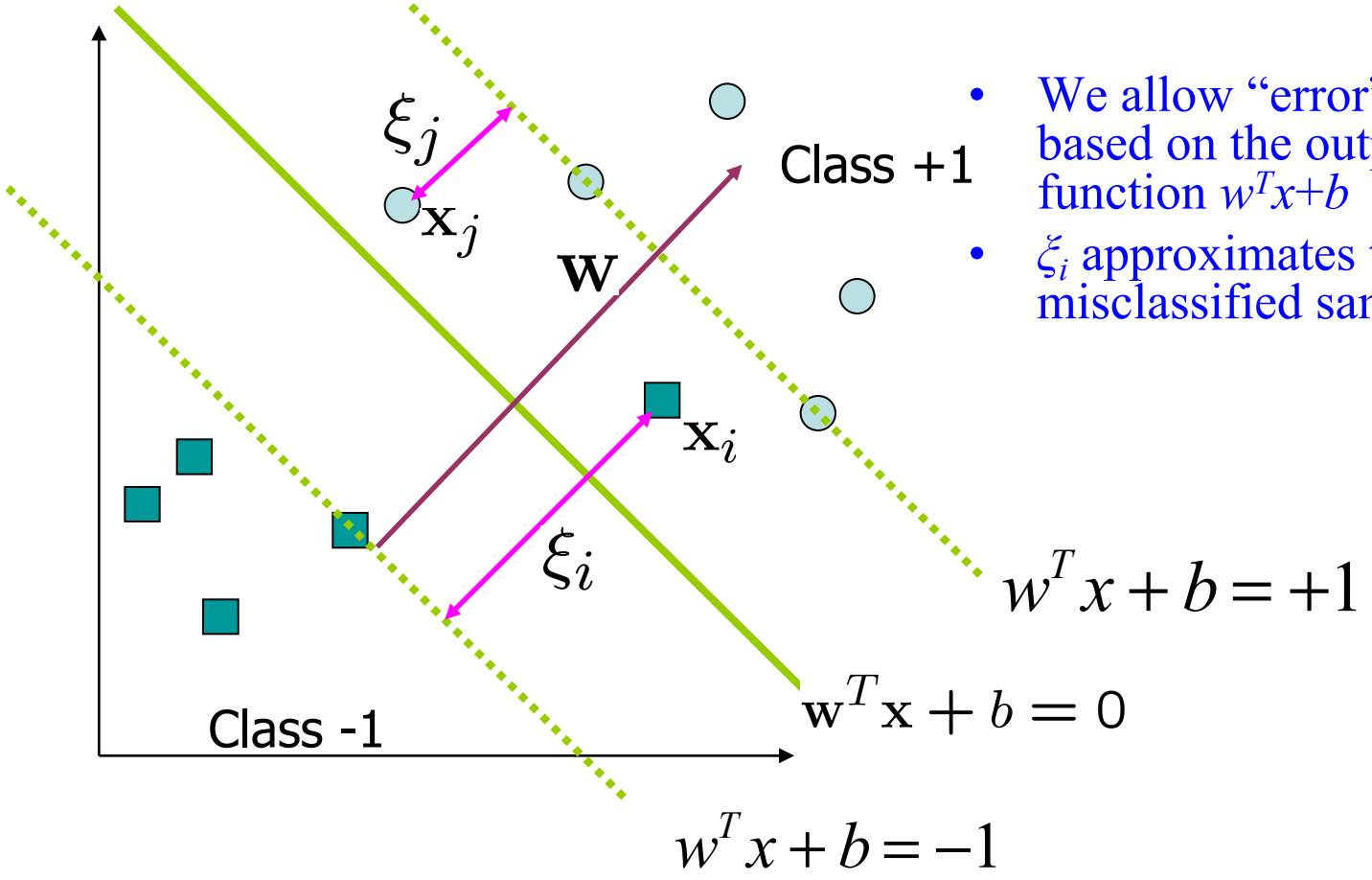
$$w^T x + b = \left(\sum_{i=1}^{N} \alpha_i y^{(i)} x^{(i)}\right)^T x + b$$

$$= \sum_{i=1}^{N} \alpha_i y^{(i)} \left\langle x^{(i)}, x \right\rangle + b$$

classify x as class +1 if the sum is positive, and class -1 otherwise
Note: w need not be formed explicitly

Non-Separable

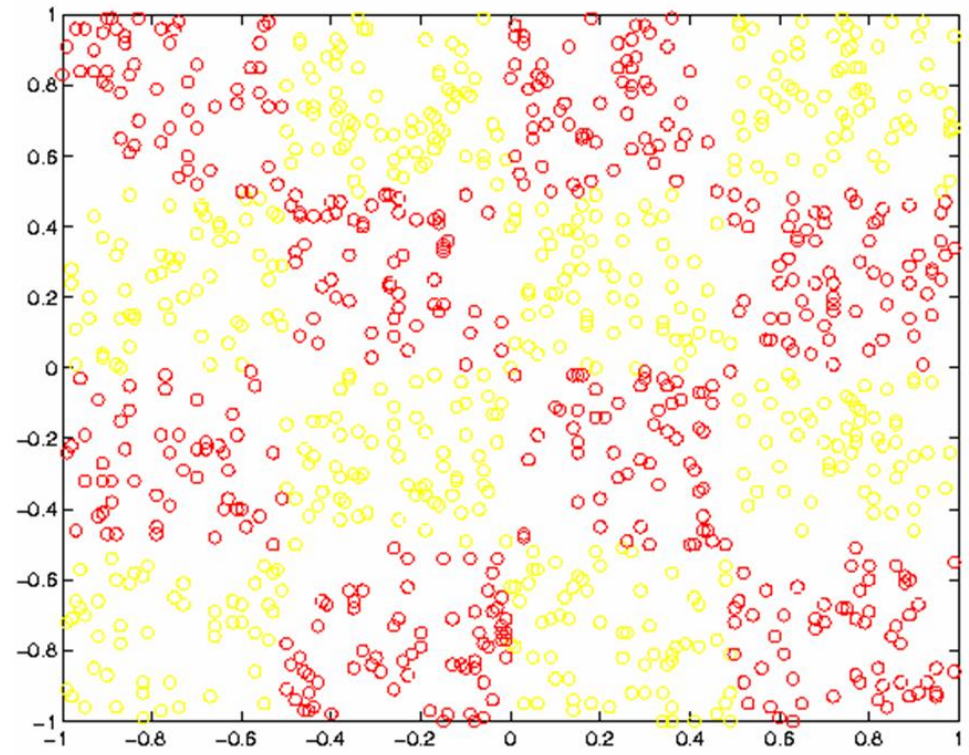
What is non-separable case?

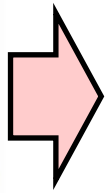


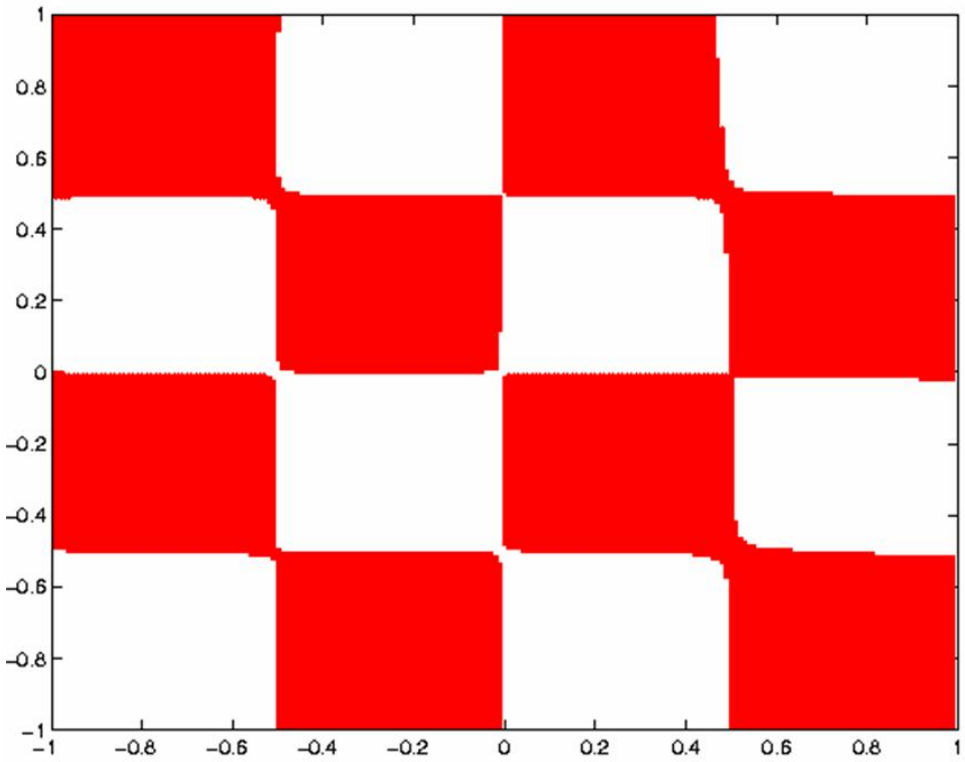
- We allow "error" ξ_i in classification; it is based on the output of the discriminant function $w^T x+b$
- ξ_i approximates the number of misclassified samples

Non-linear Cases

What is non-linear case?







Non-separable case

The formalization of the optimal problem becomes :

$$\min_{w,b,\xi} \frac{\|w\|}{2} + C\sum_{i=1}^{N} \xi_i$$

$$s.t. \qquad y(w^T x^{(i)} + b) \geq 1 - \xi_i, 1 \leq i \leq N$$

$$\xi_i \geq 0, 1 \leq i \leq N$$

Thus, examples are now permitted to have margin less than 1, and if an example has functional margin 1-ξ_i (with ξ>0), we would pay a cost of the objective function by increased by $C\xi_i$.

The parameter *C* controls the relative weighting between the twin goals of making the $||w||2$ small and of ensuring that most examples have functional margin at least 1.

Lagrangian Solution

Again, we have the lagrangian form :

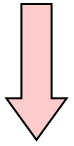
$$L(w,b,\xi,\alpha,\sigma)=\frac{\|w\|}{2}+C\sum_{i=1}^{N}\xi_i-\sum_{i=1}^{N}\alpha_i[y(w^T x^{(i)}+b)-1+\xi_i]-\sum_{i=1}^{N}\sigma_i\xi_i$$

$$s.t. \qquad \sigma_i \geq 0;\alpha_i \geq 0$$

$$\max_\alpha L(\alpha)=\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{N}y^{(i)}y^{(j)}\alpha_i\alpha_j\left\langle x^{(i)},x^{(j)}\right\rangle$$

$$s.t. \qquad C\geq\alpha_i\geq 0, i\in[1,N]$$

$$\sum_{i=1}^{N}\alpha_i y^{(i)}=0$$



What is the difference from the separable form??!!

$$\alpha_i=0\Rightarrow y^{(i)}(w^T x^{(i)}+b)\geq 1$$

$$\alpha_i=C\Rightarrow y^{(i)}(w^T x^{(i)}+b)\leq 1$$

$$0<\alpha_i<C\Rightarrow y^{(i)}(w^T x^{(i)}+b)=1$$

\+ KKT conditions:

$$\frac{\partial L(w,b,\alpha)}{\partial w_i}=0, i\in[1,N]$$

$$\frac{\partial L(w,b,\alpha)}{\partial \xi_i}=0, i\in[1,N]$$

$$\frac{\partial L(w,b,\alpha)}{\partial b}=0$$

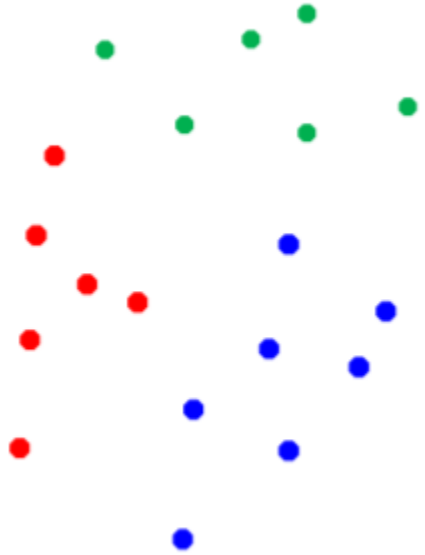
$$\alpha_i(y^{(i)}(w^T x^{(i)}+b)-1-\xi_i)=0, i\in[1,N]$$

$$y^{(i)}(w^T x^{(i)}+b)-1-\xi_i\geq 0, i\in[1,N]$$

$$\alpha_i\geq 0,\sigma_i\geq 0, i\in[1,N]$$

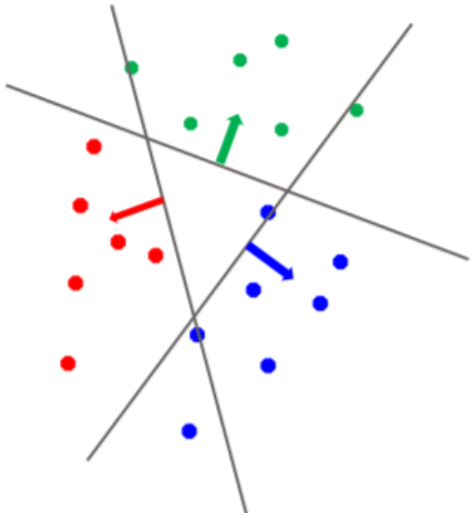
Multiclass Classification with SVMs

- How to do multiclass classification with SVMs?



One-versus-All

- Learn multiple binary classifiers separately:
  - Build |C| classifiers $(w_k, b_k)_{k=1,\ldots,|C|}$



- Prediction:
  - choose the class with greatest margin.

$$\hat{y} = argmax_k(w_k^T x + b_k)$$

Problems

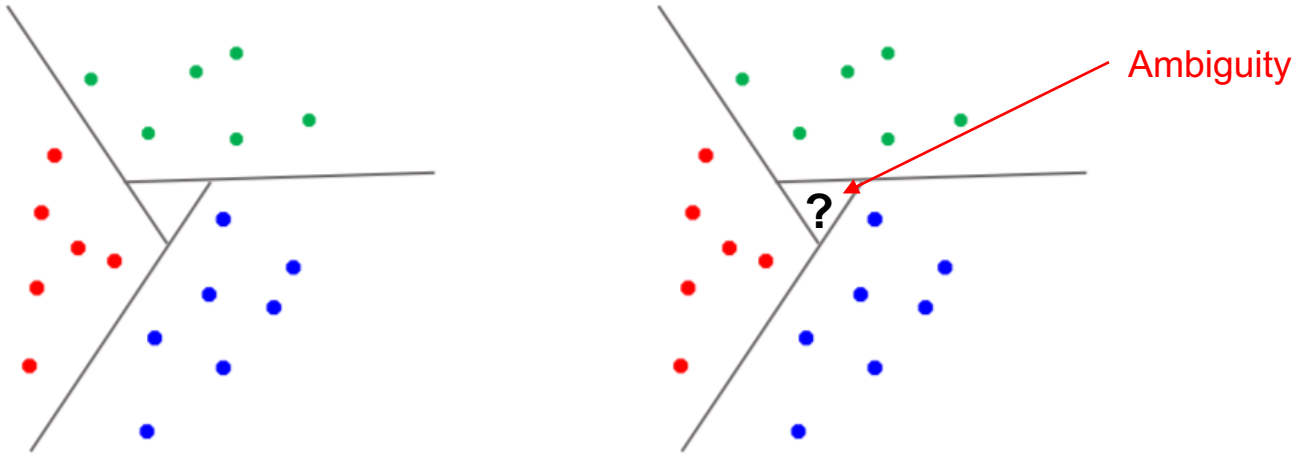
- Imbalance issue when learning each binary classifier
  - Much more negative samples than positives

$$\hat{y} = \text{argmax}_k(w_k^T x + b_k)$$

- The weights may not be based on the same scale
  - $(aw_k^T)x + (ab_k)$ is also a solution scaled by a

One-versus-One

- Build |C|(|C|-1)/2 classifiers:



- Prediction:
  - Majority voting
- Advantage:
  - the time for training classifiers may actually decrease
- Problem:
  - Ambiguity issue!

Multiclass SVMs

- Simultaneously learn |C| sets of weights
  - The **margin** is the gap between the correct class and the nearest other class
  - For all instances (x_j, y_j), for all incorrect classes y'

$$w_{y_j} x_j + b_{y_j} \geq w_{y'} x_j + b_{y'} + 1 - \xi_j, \ \forall y' \neq y_j, \forall j$$

- Target:

$$\min_w \sum_y \frac{||w^y||}{2} + C \sum_j \xi_j$$

$$s.t., w_{y_j} x_j + b_{y_j} \geq w_{y'} x_j + b_{y'} + 1 - \xi_j, \qquad \forall y' \neq y_j, \forall j$$

$$\xi_j \geq 0, \forall j$$

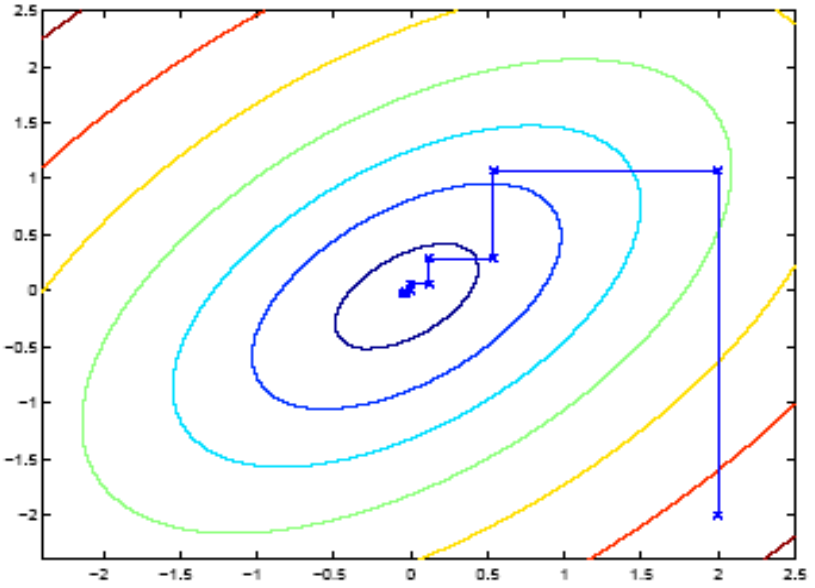
- Prediction: $\hat{y} = \text{argmax}_y (w_y^T x + b_y)$

How to train SVM

Solving the quadratic programming optimization problem directly to train the SVM is very slow when the training data grows large.

Sequential minimal optimization (SMO) algorithm, due to John Platt.

First, let us introduce coordinate ascent algorithm:
Loop until convergence:
{
  For i=1, …, m
  {
    $\alpha_i = argmax_{\widehat{\alpha_i}} L(\alpha_1, \dots, \alpha_{i-1}, \widehat{\alpha_i}, \alpha_{i+1}, \dots, \alpha_m)$
  }
}



Coordinate ascent is ok?

Recall that SVM optimization problem has added requirements that:

$$\frac{\partial L(w,b,\alpha)}{\partial b} = \sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

That is

$$\alpha_1 y^{(1)} = -\sum_{i=2}^{N} \alpha_i y^{(i)}$$

$$\alpha_1 = -y^{(1)} \sum_{i=2}^{N} \alpha_i y^{(i)}$$

Is it sufficient?

SMO Rule

- If we increase one α by an amount η, in either direction, then we have to change another α by an equal amount in the opposite direction (relative to class value).

- Let E be the error of example x, we can accomplish this by changing:

$$\alpha \leftarrow \alpha - y\eta E \qquad \text{to:}$$
$$\alpha_1 \leftarrow \alpha - y_1\eta(E_1 - E_2)$$

We then also adjust α_2 so that $y_1\alpha_1 + y_2\alpha_2$ stays same

SMO

Change the algorithm by: (this is just SMO)
Repeat until convergence
{
1. select some pair α_i and α_j to update next. (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. reoptimize $L(\alpha)$ with respect to α_i and α_j , while holding all the other α.
}

Many approaches have been proposed
-e.g., Loqo, cplex, etc. (see http://www.numerical.rl.ac.uk/qp/qp.html)

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{N} \alpha_i y^{(i)} = \varsigma$$

$$\alpha_1 = (\varsigma - \alpha_2 y^{(2)}) y^{(1)}$$

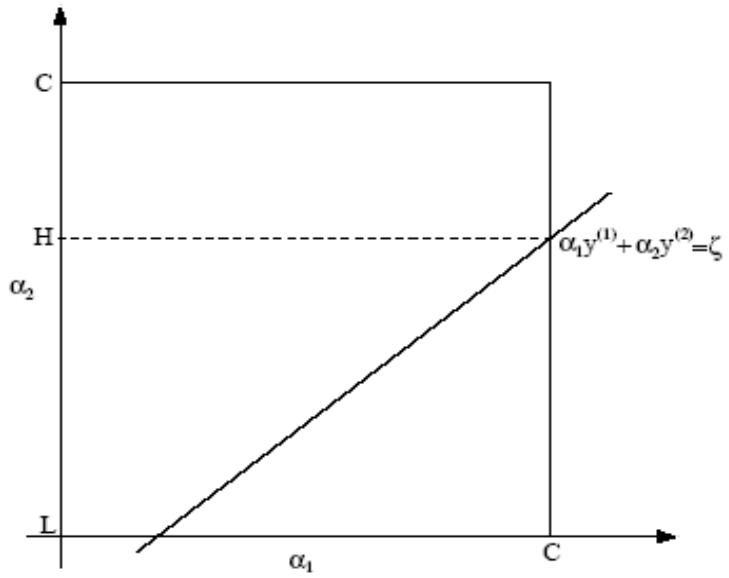
$$L(a) = L((\varsigma - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, ..., \alpha_N)$$

Updating Two $\alpha's$: One SMO Step

$$L(a) = L((\varsigma - \alpha_2 y^{(2)})y^{(1)}, \alpha_2, ..., \alpha_m)$$

This is a quadratic function in α_2, i.e., it can be written as:

$$a\alpha_2^2 + b\alpha_2 + c$$



Updating Two $\alpha's$: One SMO Step

$$a\alpha_2^2 + b\alpha_2 + c$$

For the quadratic function, we can simply solve it by setting its derivative to zero. Let us use $\alpha_2^{new,unclipped}$ as the resulting value.

$$\alpha_2^{new,unclipped} = \alpha_2 + \eta y_2(E_1 - E_2)$$

where:

$$\eta = \frac{1}{x_1^T x_1 + x_2^T x_2 - 2x_1^T x_2}$$

For a given difference in error between two examples, the step size is larger when the two examples are more similar. When x_1 and x_2 are similar, $x_1^T x_2$ is larger. Hence the step size is larger.

Updating Two $\alpha's$: One SMO Step

Recall our formulation had an additional constraint:

$$0 \leq \alpha_i \leq C, \forall\, i$$

Therefore, we "clip" any change we make to any α to respect this constraint

$$\alpha_2^{new} = \begin{cases} H & if\,(\alpha_2^{new,unclipped} > H) \\ \alpha_2^{new} & if\,(L \leq \alpha_2^{new,unclipped} \leq H) \\ L & if\,(\alpha_2^{new,unclipped} < L) \end{cases}$$

Having find α_2, we can go back to find the optimal α_1.
Set $\alpha_1^{new} = \alpha_1 + y_1 y_2 (\alpha_2 - \alpha_2^{new})$.

Please read Platt's paper if you want to read more details

SMO

- It is necessary and sufficient for a solution to our objective that all α's satisfy the following:

$$\alpha_i = 0 \Rightarrow y^{(i)}(w^T x^{(i)} + b) \geq 1$$
$$\alpha_i = C \Rightarrow y^{(i)}(w^T x^{(i)} + b) \leq 1$$
$$0 < \alpha_i < C \Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1$$

- We just check KKT to within some small epsilon, typically 10^{-3}

- Heuristics: choosing α_i and α_j to optimize so as to maximize the objective function as much as possible
  - since there are n(n−1) possible choices

SMO Algorithm

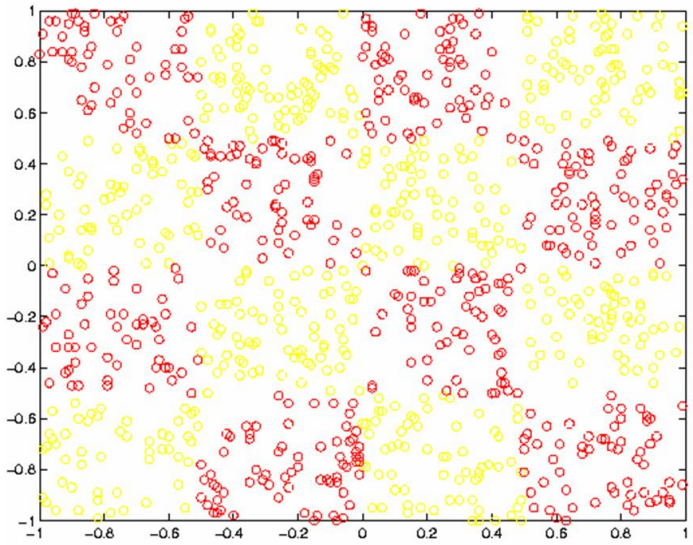
- Input: C, kernel, kernel parameters, epsilon
- Initialize b and all α's to 0
- Repeat until KKT satisfied (to within epsilon):
  - Find an example x_1 that violates KKT (prefer unbound examples here, choose randomly among those)
  - Choose a second example x_2. Prefer one to maximize step size (in practice, faster to just maximize $|E_1 - E_2|$). If that fails to result in change, randomly choose unbound example. If that fails, randomly choose example. If that fails, re-choose x_1.
  - Update α_1 and α_2 in one step
  - Compute new threshold b

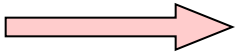


Kernel

Kernel

1. **Why kernel?**





• **Key idea:** transform x_i to a higher dimensional space to "make life easier"
  - **Input space:** the space the point x_i are located
  - **Feature space:** the space of $f(x_i)$ after transformation
• Why transform?
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  - Classification can become easier with a proper transformation. In the XOR problem, for example, adding a new feature of $x_1 x_2$ make the problem linearly separable

2. **What is feature space mapping?**

$$x \Rightarrow \Phi(x)$$

$$K(x,z) = \Phi(x)^T \Phi(z)$$ kernel function

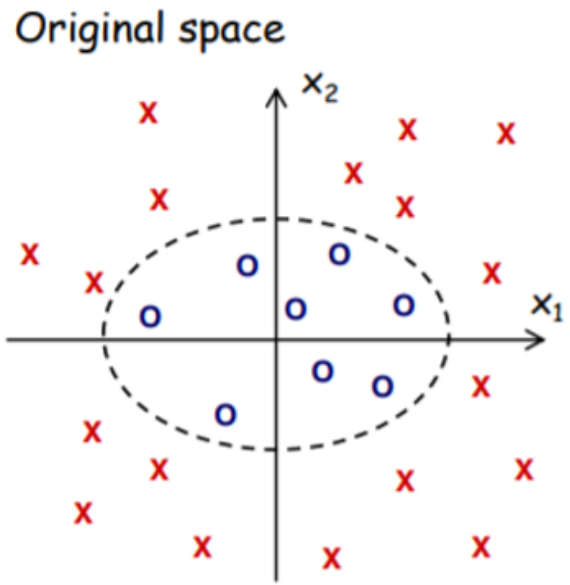
Example: Polynomial Kernel

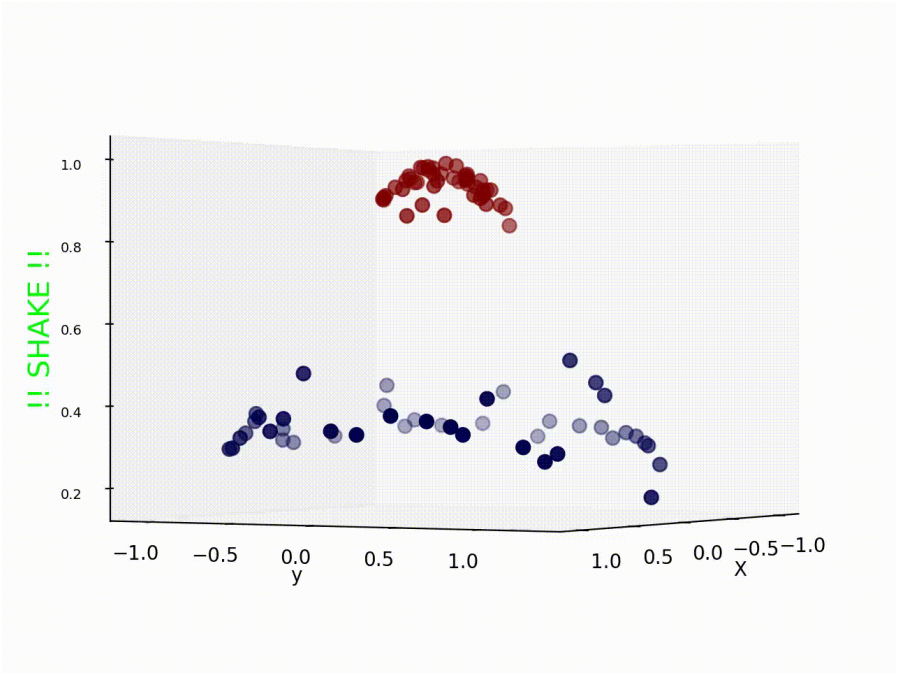
For $n = 2, d = 2$, the kernel $K(x, z) = (x \cdot z)^d$ corresponds to

$$\Phi: R^2 \to R^3, (x_1, x_2) \to$$

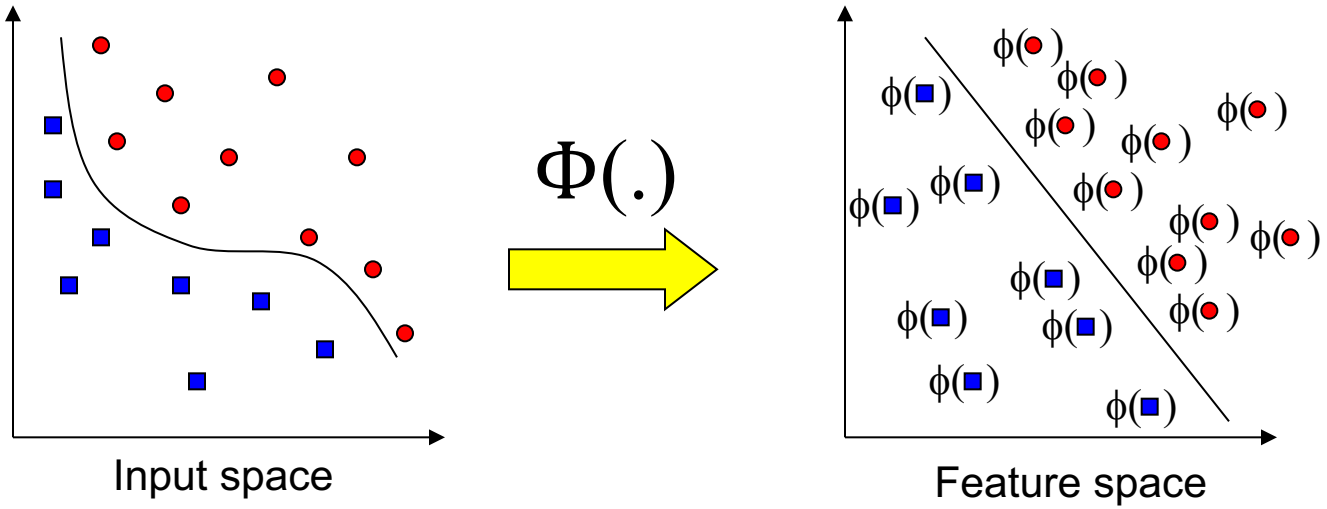
$$\Phi(x) = \left(x_1^2, x_2^2, \sqrt{2}\, x_1 x_2\right)$$

$$\Phi(x) \cdot \Phi(z) = (x_1 z_1 + x_2 z_2)^2 = K(x, z)$$





Feature Space Mapping



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

Replace all $\langle x^{(i)}, x \rangle$ by $\langle \Phi(x^{(i)}), \Phi(x) \rangle$

$$\max_{\alpha} L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle \Phi(x^{(i)}), \Phi(x^{(j)}) \rangle$$

$$s.t. \qquad \alpha_i \geq 0, i \in [1, N]$$

$$\sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

$$w^T x + b = (\sum_{i=1}^{N} \alpha_i y^{(i)} x^{(i)})^T x + b$$

$$= \sum_{i=1}^{N} \alpha_i y^{(i)} \langle \Phi(x^{(i)}), \Phi(x) \rangle + b$$

With kernel, what's more interesting to us?

We can compute kernel without calculating mapping

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is even infinite-dimensional!
- The kernel trick comes to rescue
- Both in the training and test, The data points only appear as inner product $x^T z$.
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly

$$K(x,z) = \Phi(x)^T \Phi(z)$$

For example: $$K(x,z) = \exp(-\frac{\|x-z\|^2}{2\sigma^2})$$

An Example for $\Phi(.)$ and K(.,.)

- Suppose $\Phi(.)$ is given as follows

$$\Phi([x_1\ x_2]) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An inner product in the feature space is

$$\Phi([x_1\ x_2])^T\Phi([y_1, y_2]) = (1 + x_1\ y_1 + x_2y_2)^2$$

- So, if we define the kernel function as follows, there is no need to carry out $\Phi(.)$ explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- This use of kernel function to avoid carrying out $\Phi(.)$ explicitly is known as the kernel trick

Kernel Functions

- In practical use of SVM, the user specifies the kernel function; the transformation $\Phi(.)$ is not explicitly stated
- Given a kernel function $K(x_i, x_j)$, the transformation $\Phi(.)$ is given by its eigenfunctions (a concept in functional analysis)
  - Eigenfunctions can be difficult to construct explicitly
  - This is why people only specify the kernel function without worrying about the exact transformation
- Another view: kernel function, being an inner product, is actually a similarity measure between the objects

Examples of Kernel Functions

- Polynomial kernel with degree d

$$K(x, z) = (x^T z + 1)^d$$

- Radial Basis Function (RBF) kernel with width σ

$$K(x, z) = \exp(-\gamma ||x - z||_2^2)$$

  - The feature space is infinite-dimensional
- Gaussian kernel

$$K(x, z) = \exp(-||x - z||_2^2 / 2\sigma^2)$$

  - Special Case of Radial Basis Function.

Examples of Kernel Functions

- Tangent(Sigmoid) with parameter κ and θ

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x}^T \mathbf{y} + \theta)$$

  - It does not satisfy the Mercer condition on all κ and θ
  - With SVM, this is equivalent to a 2-layer neural network

Advanced Kernels

- All problems so far: each example is a list of numbers
- What about text, relational DBs, . . . ?

- Insight: $K(x, y)$ can be defined when x and y are not fixed length

- Examples:
  - String kernels
  - Path kernels
  - Tree kernels
  - Graph kernels

String Kernels

- Pick $\lambda \in (0,1)$
- Cat $\rightarrow$ c, a, t, λ ca, λ at, λ^2 ct, λ^2 cat

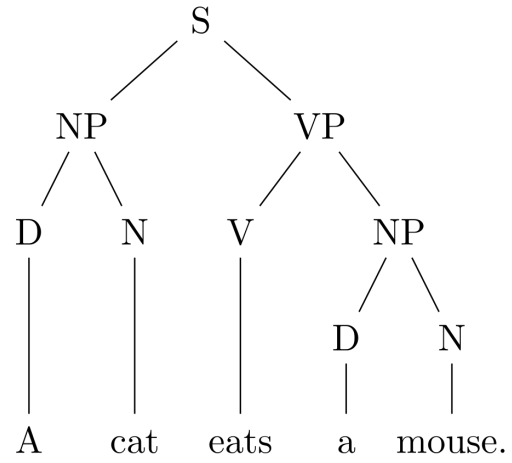
- Strings are similar if they share lots of nearly-contiguous substrings

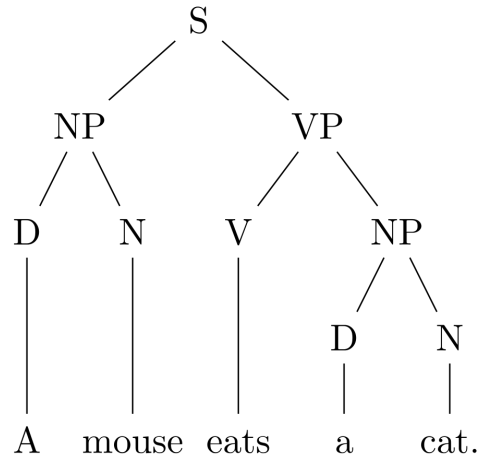
- Works for words in phrases too: “man bites dog” similar to “man bites hot dog,” less similar to “dog bites man”

- There is an efficient dynamic-programming algorithm to evaluate this kernel (Lodhi et al, 2002)

Subtree Kernel

- The subtree kernel count the number of common subtrees between two given trees.



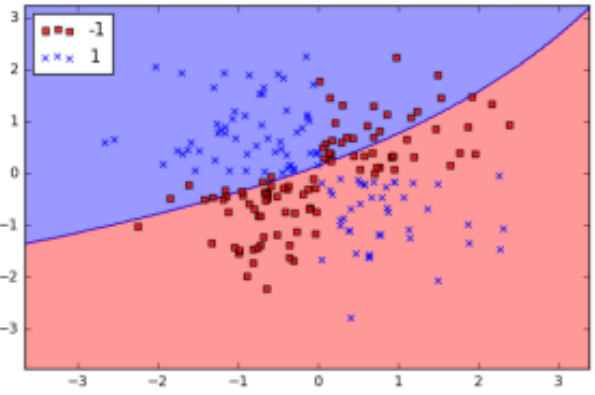


- In this example, there are seven common subtrees:

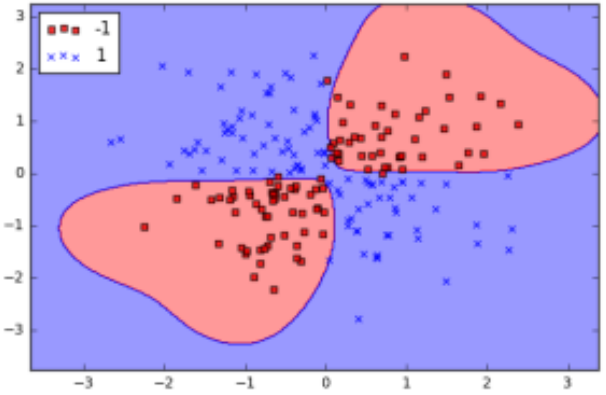
[NP [D [a]] [N [cat]]],
[NP [D [a]] [N [mouse]]],
[N [mouse]],
[N [cat]],
[V [eats]],
[D [a]] (counted twice as it appears twice).

Tuning Parameter

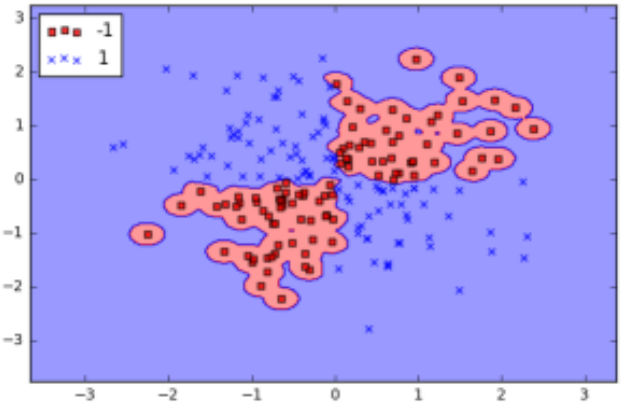
- We need to tune parameters in SVM kernels.
- For example, $K(x, z) = \exp(-\gamma ||x - z||_2^2)$
- When *gamma* parameter is high the value of kernel function will be less, even for two close by samples.
- This may cause complicated decision boundary or give rise to over-fitting.



$\gamma = 0.01$



$\gamma = 1$



$\gamma = 100$

Modification Due to Kernel Function

- Change all inner products to kernel functions
- For training,

Original

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

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

With kernel function

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

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

Modification Due to Kernel Function

- For testing, the new data z is classified as class +1 if $f \geq 0$, and as class -1 if $f < 0$

Original

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

$$f = \mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}^T \mathbf{z} + b$$

With kernel function

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \phi(\mathbf{x}_{t_j})$$

$$f = \langle \mathbf{w}, \phi(\mathbf{z}) \rangle + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

Characteristics of Kernel Functions

- Not all similarity measure can be used as kernel function, however
  - The kernel function needs to satisfy the Mercer condition, i.e., the function is "positive-definite"
  - This implies that the *n* by *n* kernel matrix, in which the (i, j)-th entry is the $K(x_i, x_j)$ is always positive definite
  - This also means that the QP is convex and can be solved in polynomial time

Mercer Theorem

- Apart from this per-defined kernels, *what conditions are there to determine which functions can be Kernels?*
- This is given by Mercer's theorem
  - Symmetric

$$K(x,z) = \Phi(x)^T \Phi(\mathrm{z}) = \Phi(z)^T \Phi(x) = K(z,x)$$

  - Kernel matrix k is *positive semi-definite*

$$k = \begin{bmatrix} K(x_1,x_1), K(x_1,x_2), \dots, \\ \dots \\ \dots \\ K(x_n,x_1), K(x_n,x_2), \dots, \end{bmatrix}$$

Mercer Theorem (cont.)

Proof*:*

If K() is a kernel and k is the Kernel matrix with $\mathrm{k_{i,j}} = \mathrm{K}(\mathrm{x_i}, \mathrm{x_j})$ then

$$c^T k c = \sum_i \sum_j c_i c_j k_{i,j} = \sum_i \sum_j c_i c_j \Phi(x_i)\Phi(x_j)$$

$$= \left(\sum_i c_i \Phi(x_i)\right)\left(\sum_j c_j \Phi(x_j)\right) = ||\left(\sum_j c_j \Phi(x_j)\right)||^2 \geq 0$$

- In a finite input space, if the Kernel matrix (also known as Gram matrix) is *positive semi-definite* then the matrix element i.e. the function K can be a kernel function.
- This allowed us to select kernels such that the underlying map could even be an infinite vector! (Example: Gaussian Kernels)

Example

- An example of finding the **mapping function from the kernel function** and here we will use Gaussian kernel function.

$$e^{-\frac{1}{2\sigma^2}(x_i-x_j)^2} = e^{\frac{-x_i^2-x_j^2}{2\sigma^2}}\left(1+\frac{2x_ix_j}{1!}+\frac{(2x_ix_j)^2}{2!}+\cdots\right)$$

$$= e^{\frac{-x_i^2-x_j^2}{2\sigma^2}}\left(1\cdot 1+\frac{\sqrt{2}x_i}{1}\frac{\sqrt{2}x_j}{1}+\frac{\sqrt{2^2}}{\sqrt{2!}}x_i^2\frac{\sqrt{2^2}}{\sqrt{2!}}x_j^2+\cdots\right)$$

$$= \Phi(x_i)^T\Phi(x_j)$$

Therefore, the feature space is infinite-dimensional

$$\Phi(x) = e^{\frac{-x^2}{2\sigma^2}}\left(1,\frac{\sqrt{2}x}{1},\frac{\sqrt{2^2}}{\sqrt{2!}}x^2,\ldots\right)$$

Characteristics of Kernel Functions

- Since the training of SVM only requires the value of $K(x_i, x_j)$, there is no restriction of the form of x_i and x_j
  - x_i can be a sequence or a tree, instead of a feature vector
- $K(xi, x_j)$ is just a similarity measure comparing x_i and x_j
- For a test object z, the function essentially is a weighted sum of the similarity between z and a pre-selected set of objects (the support vectors)

$$f(\mathbf{z}) = \sum_{\mathbf{x}_i \in \mathcal{S}} \alpha_i y_i K(\mathbf{z}, \mathbf{x}_i) + b$$

$\mathcal{S}$: the set of support vectors

Example

- Suppose we have five 1D data points
  - x_1=1, x_2=2, x_3=4, x_4=5, x_5=6, with 1, 2, 6 as class 1 and 4, 5 as class 2 $\Rightarrow$ y_1=1, y_2=1, y_3=-1, y_4=-1, y_5=1
- We use the polynomial kernel of degree 2
  - K(x,y) = $(xy+1)^2$
  - C is set to 100
- We first find α_i (i=1, ..., 5) by

$$\text{max.} \sum_{i=1}^{5} \alpha_i - \frac{1}{2} \sum_{i=1}^{5} \sum_{j=1}^{5} \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2$$

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^{5} \alpha_i y_i = 0$$

Example

- By using a QP solver, we get
  - α_1=0, α_2=2.5, α_3=0, α_4=7.333, α_5=4.833
  - Note that the constraints are indeed satisfied
  - The support vectors are {x_2=2, x_4=5, x_5=6}
- The discriminant function is

$$\begin{aligned} & f(z) \\ &= 2.5(1)(2z+1)^2 + 7.333(-1)(5z+1)^2 + 4.833(1)(6z+1)^2 + b \\ &= 0.6667z^2 - 5.333z + b \end{aligned}$$

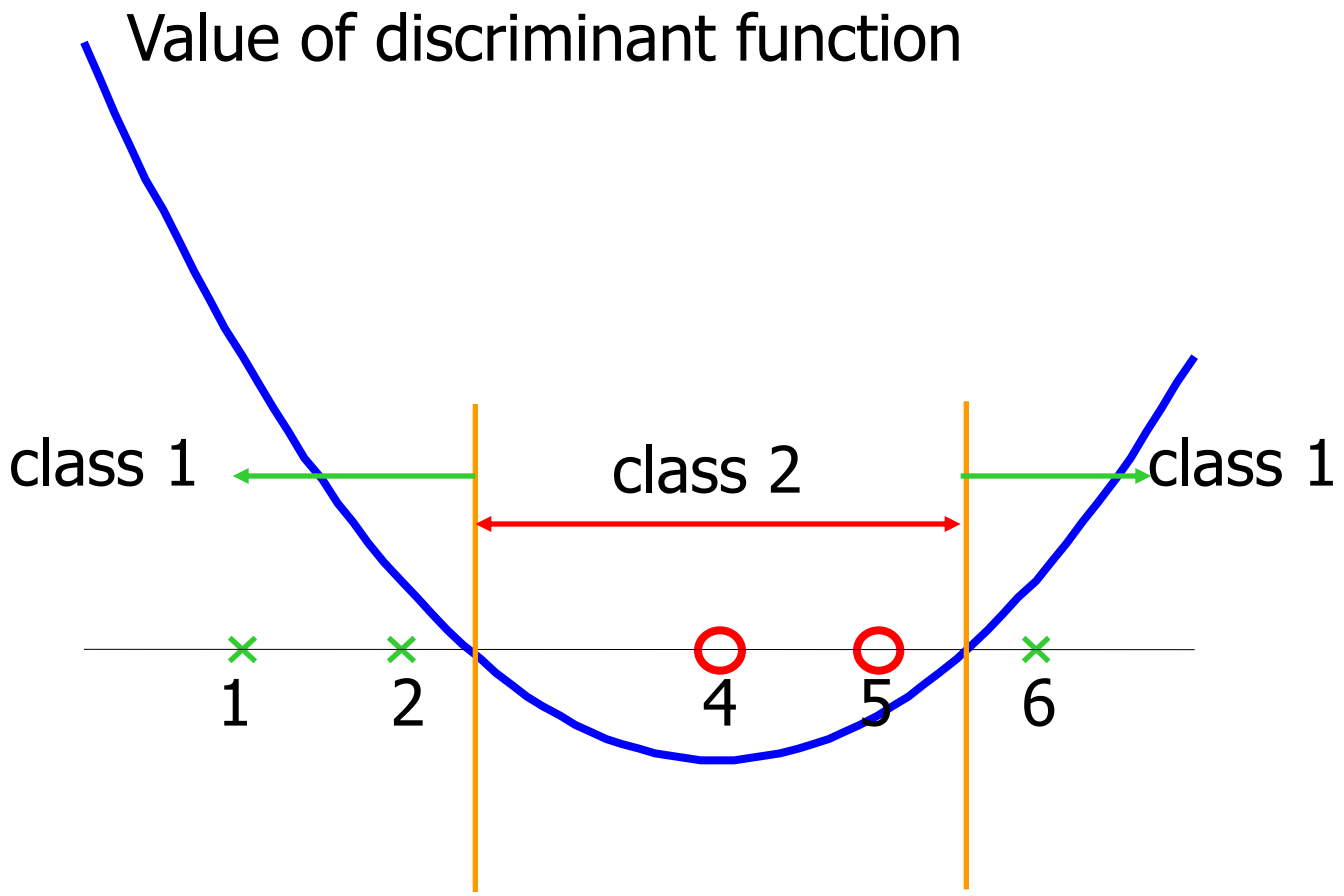
(α_5, y_5, $K(z, x_5)$)

- *b* is recovered by solving f(2)=1 or by f(5)=-1 or by f(6)=1, as x_2 and x_5 lie on the line $\phi(\mathbf{w})^T\phi(\mathbf{x}) + b = 1$ and x_4 lies on the line

$$\phi(\mathbf{w})^T\phi(\mathbf{x}) + b = -1$$

- All three give b=9 ⇒ $f(z) = 0.6667z^2 - 5.333z + 9$

Example


Value of discriminant function
class 1
class 2
class 1
1
2
4
5
6

Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarizes all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, ...)
- There is even research to estimate the kernel matrix from available information

- In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try
- Note that SVM with RBF kernel is closely related to RBF neural networks, with the centers of the radial basis functions automatically chosen for SVM

互动：Why are SVMs often more accurate than logistic regression?

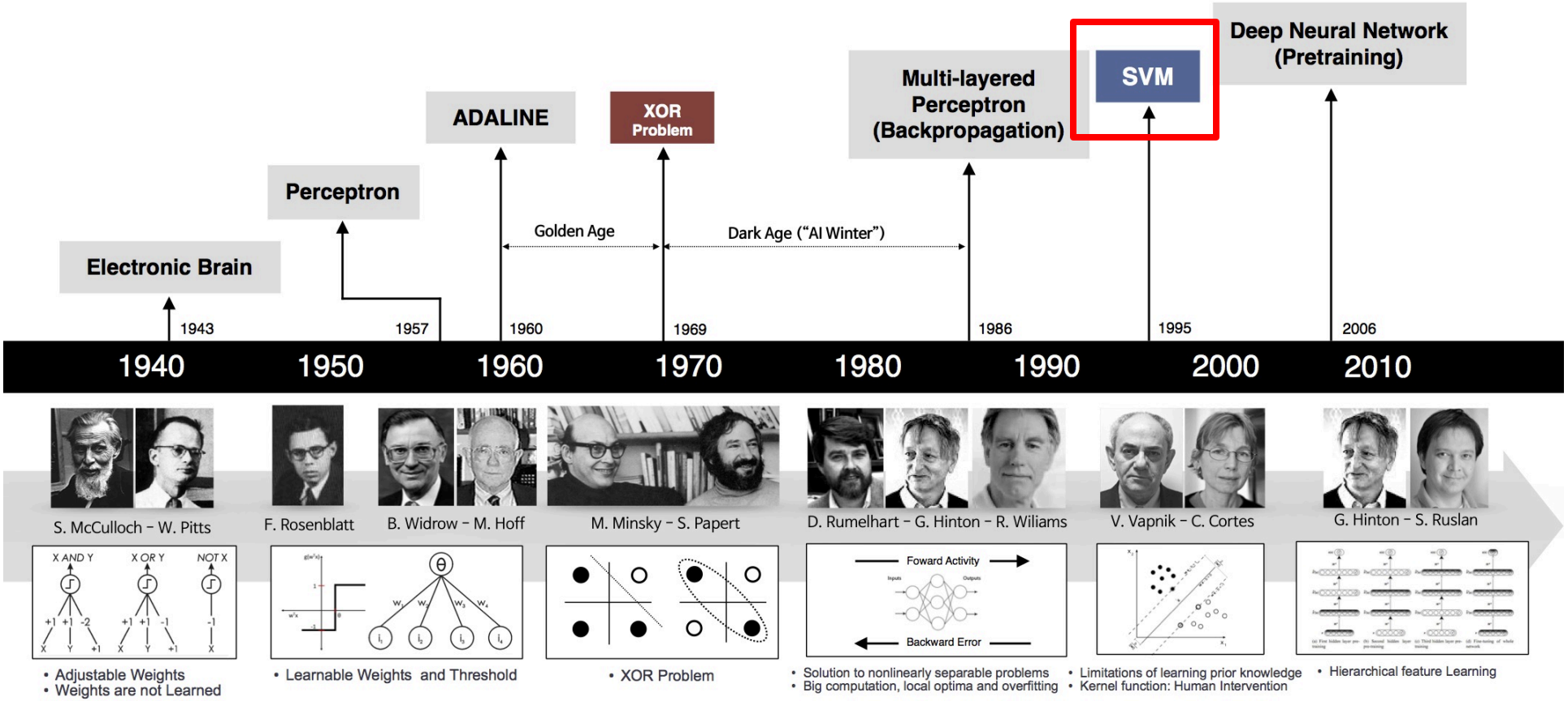
Summary

- Mapping data points from low dimensional space to a higher dimensional space can make it possible to apply SVM even for non-linear data sample.
- We don't need to know the mapping function itself as long as we know the Kernel function; ***Kernel Trick***
- Condition for a function to be a kernel function; *Positive semi-definite Gram matrix.*
- Types of kernels that are used most, especially RBF kernel.
- How the tuning parameter gamma can lead to over fitting or bias in RBF kernel.

Summary: Steps for Classification

- Prepare the feature matrix
- Select the kernel function to use
- Select the parameter of the kernel function and the value of *C*
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the α_i
- Unseen data can be classified using the α_i and the support vectors

Conclusions



- Two key concepts of SVM:
  - Maximal margin
  - Kernel trick

Software

- A list of SVM implementation can be found at
  http://www.kernel-machines.org/software.html

- Some implementation (such as LIBSVM) can handle multi-class classification

- SVM-Light is among one of the earliest implementation of SVM

- Weka

- Several Matlab toolboxes for SVM are also available

Useful Link

- http://www.kernel-machines.org/
- http://www.support-vector.net/
- http://www.support-vector.net/icml-tutorial.pdf
- http://www.kernel-machines.org/papers/tutorial-nips.ps.gz
- http://www.clopinet.com/isabelle/Projects/SVM/applist.html

References

- Vladimir N. Vapnik. The nature of statistical learning theory. Springer-Verlag New York. 1998.
- Andrew Ng. CS229 Lecture notes. Lectures from 10/19/03 to 10/26/03. Part V. Support Vector Machines
- CHRISTOPHER J.C. BURGES. A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery, 2, 121–167 (1998). 1998 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.
- Cristianini, N., Shawe-Taylor, J., An Introduction to Support Vector Machines, Cambridge University Press, (2000).

People

- Vladimir Vapnik.
- J. Platt
- J. Platt, N. Cristianini, J. Shawe-Taylor
- Shawe-Taylor, J.
- Burges, C. J. C.
- Thorsten Joachims
- Chih-Jen Lin
- Find more by
  - http://aminer.org/search?q=support vector machine

Useful Slides

- Sergios Theodoridis. Support Vector Machines: A Geometric View Point. University of Athens.
- Olvi L. Mangasarian. Mathematical Programming for Support Vector Machines. University of Wisconsin-Madison.
- Ahmed Abbasi. Pattern Recognition using Support Vector Machine and Principal Component Analysis. University of Arizona.
- Martin Law. A Simple Introduction to Support Vector Machines. Michigan State University

Thanks!

HP: http://keg.cs.tsinghua.edu.cn/jietang/

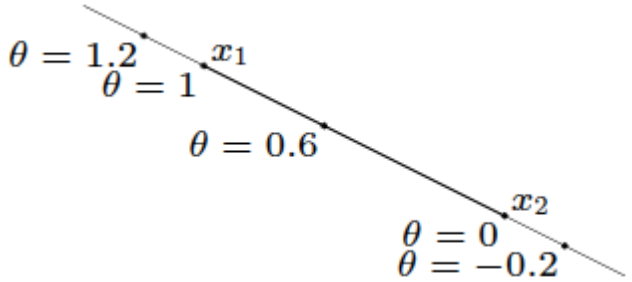
Email: jietang@tsinghua.edu.cn

Appendix

Affine Set

- Line through x_1 and x_2: all points

$$x = \theta x_1 + (1-\theta)x_2 \qquad \theta \in R$$



- Affine set: contains the line through any two distinct points in the set
- Affine function:
  - f: R^n $->$ R^m is affine if $f(x) = Ax + b$ with $A \in R^{m \times n}, b \in R^m$

A number of the following pages are from Boyd's slides

Convex Set

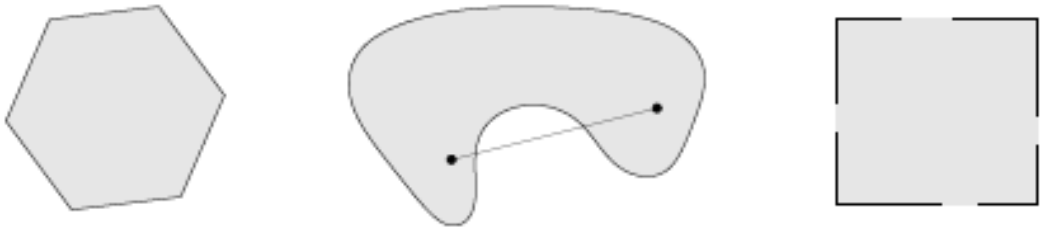
- Line segment between x_1 and x_2: all points

$$x = \theta x_1 + (1-\theta)x_2 \qquad 0 \le \theta \le 1$$

- Convex set: contains line segment between any two points in the set

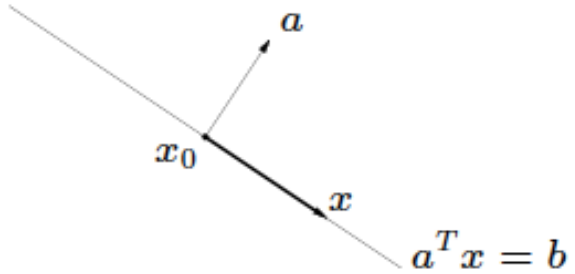
$$x_1, x_2 \in C, \qquad 0 \le \theta \le 1 \qquad => \theta x_1 + (1-\theta)x_2 \in C$$

- Examples (one convex, two nonconvex sets)

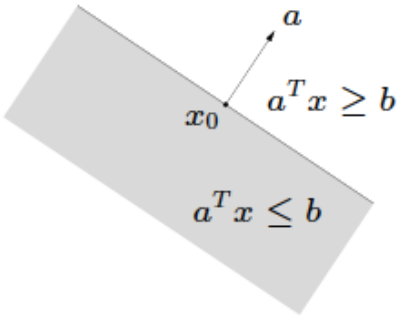


Hyperplanes and Halfspaces

- Hyperplane: set of the form $\{x \mid a^T x = b\} (a \neq 0)$



- Halfspace: set of the form $\{x \mid a^T x \leq b\} (a \neq 0)$



- a is the normal vector
- Hyperplanes are affine and convex; halfspaces are convex.

How to train large-scale data with SVM [1]

[1] Yu H F, Hsieh C J, Chang K W, et al. Large linear classification when data cannot fit in memory[J]. ACM Transactions on Knowledge Discovery from Data (TKDD), 2012, 5(4): 23.

Motivation

- Existing approaches for large linear classification:
  - Data small than memory:

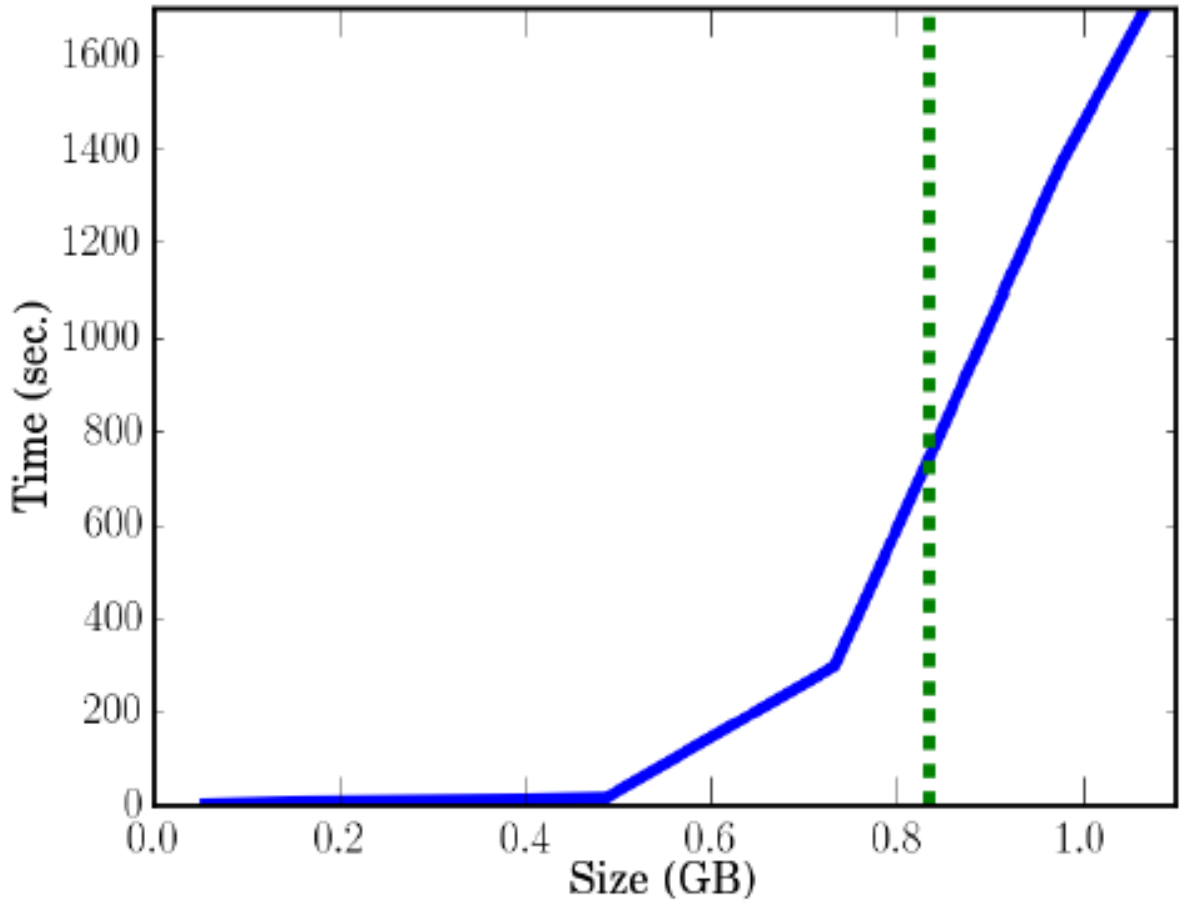
    Efficient methods are well-developed
  - Data beyond disk size:

    Usually handled in a distributed way
- Can we have something in the between?
  - A simple setting

    memory < data < disk

When Data Cannon Fit In Memory

- LIBLINEAR on a machine with 1GB memory:



Disk swap causes lengthy training time

Modeling the Training Time

- Train-time = time to train in-memory data + time to access data from disk

- Existing methods focus on the first term
- Now need to pay attention to the second part
- Loading time may dominate the training time
  - E.g., online learner consider one sample at a time

  => May require many iterations to converge

A Block Minimization Framework

1. Split {1,…,*m*} to B_1,…,B_l such that B_i fits in memory, and store data into *l* files accordingly.
2. Set initial α and w
3. For *k* = 1,2,…(outer iteration)

   For *j* = 1,…,*l* (inner iteration)

   (a) Read $x_r, \forall r \in B_j$ from disk

   (b) Conduct operations on $\{xr | r \in Bj\}$

   (c) Update α and w

Here we do not specify operations on each block

Number of Blocks and Block Size

- In the situation, data > memory
  - to avoid random access on the disk
  - $B_1,\ldots,B_l$: fixed partition of $\{1,\ldots,m\}$
- How to decide l (# of blocks)
  - Assume all blocks have similar size $|B|$
  - # blocks: $l = \frac{m}{|B|}$
- Block size
  - Cannot be too large: each B_j must fit in memory
  - Cannot be too small: reading time=initial cost $\times \frac{m}{|B|}$
  - Larger $|B|$ is better (but can't exceed memory)

Sub-problem for Dual SVM

- $\mathrm{L}(\alpha)$ is the dual function of SVM:

$$L(\alpha) = \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \left\langle x^{(i)}, x^{(j)} \right\rangle - \sum_{i=1}^{m} \alpha_i$$

$$s.t., 0 \le \alpha_i \le C, i = 1, \dots, l$$

- Each variable corresponds to a training instance. Thus we can easily update a block of variables at a time. Assume $\overline{B_j} = \{1, \dots, l\} \setminus B_j$, the following sub-problem is:

$$\min_{d_{B_j}} L(\alpha + d)$$

$$s.t., d_{\overline{B_j}} = 0 \ \ and \ \ 0 \le \alpha_i + d_i \le C, \forall\, i \in B_j$$

Sub-problem for Dual SVM

- Only α_{B_j} is changed
- At each inner iteration, only one block of data is needed (i.e., $d_{\overline{B_j}} = 0$)
- If $d^*_{B_j}$ is an optimal solution

$$\alpha_{B_j} \leftarrow \alpha_{B_j} + d^*_{B_j}$$

$$w \leftarrow w + \sum_{r \in B_j} d^*_r y_r x_r$$

$$(\text{because w} = \sum_i \alpha_i y_i x_i)$$

Solving Sub-problem By LIBLINEAR

- Any bound-constrained method can be used
  - We consider LIBLINEAR: a coordinate descent method
- Sub-problem stopping condition and overall convergence
  - A fixed number of passes to all variables in B_j

    Need to decide the number of passes
  - Gradient-based stopping condition

    Easy to know how accurate the sub-problem's solution is; we use the one in LIBLINEAR

  Can prove that convergence holds for both conditions

An Implementation for Dual SVM

1. Split {1,…,*m*} to $B_1,\ldots,B_l$ such that B_i fits in memory, and store data into *l* files accordingly.
2. Set initial α or w
3. For *k* = 1,2,…(outer iteration)

   For *j* = 1,…,*l* (inner iteration)

   (a) Read $x_r, \forall r \in B_j$ from disk

   (b) Approximately solve the sub-problem to obtain d_{B_j}

   (c) Update $\alpha_{B_j} \leftarrow \alpha_{B_j} + d_{B_j}$ and w

Block Minimization for Primal SVM

- Instead of the dual problem, we can solve the primal problem w.r.t. the primal variable w.
  - However, w does not correspond to data instances.
- Pegasos (Shalev-Shwartz et al., 2007) makes use of stochastic gradient descent.
- Pegasos considers a scaled form of the primal SVM problem:

$$f^P(w) = \frac{1}{2} w^T w - C \sum_{i=1}^{l} \max(0, 1 - y_i w^T x_i)$$

- At the each update, Pegasos chooses a block of data B and updates the primal variable w by SGD:

$$\overline{w} = w - \eta \nabla$$

where η is the sub-gradient

An Implementation for Pegasos

1. Split {1,…,m} to B_1,…,B_l such that B_i fits in memory, and store data into l files accordingly.
2. Set initial w
3. For k = 1,2,…(outer iteration)

    For j = 1,…,l (inner iteration)

        (a) Find a partition of B_j : $B^1{}_j$,…, $B^r{}_j$

        (b) For r' = 1, …, r

            (b.1) Apply Pegasos update on $B^{r'}{}_j$

Techniques To Reduce the Training Time

- Data compression for disk reading time
  - Except initial time, reading time $\propto$ data size $|B|$
  - Use zlib for compression, effectively reduces the disk reading time
- Initial Split of Data
  - If original data ordered by labels
    → a whole block with same label
    → slow convergence
  - A random split is needed
  - Sequentially read data instances and randomly decide which file it should be saved to

Data and Environment

Data set	l (data)	n (features)	Mem
yahoo-korea	460,554	3,052,939	2.5GB
webspam	350,000	16,609,143	20.8GB
epsilson	500,000	2,000	16.0GB

64-bit machine with 1 GB RAM

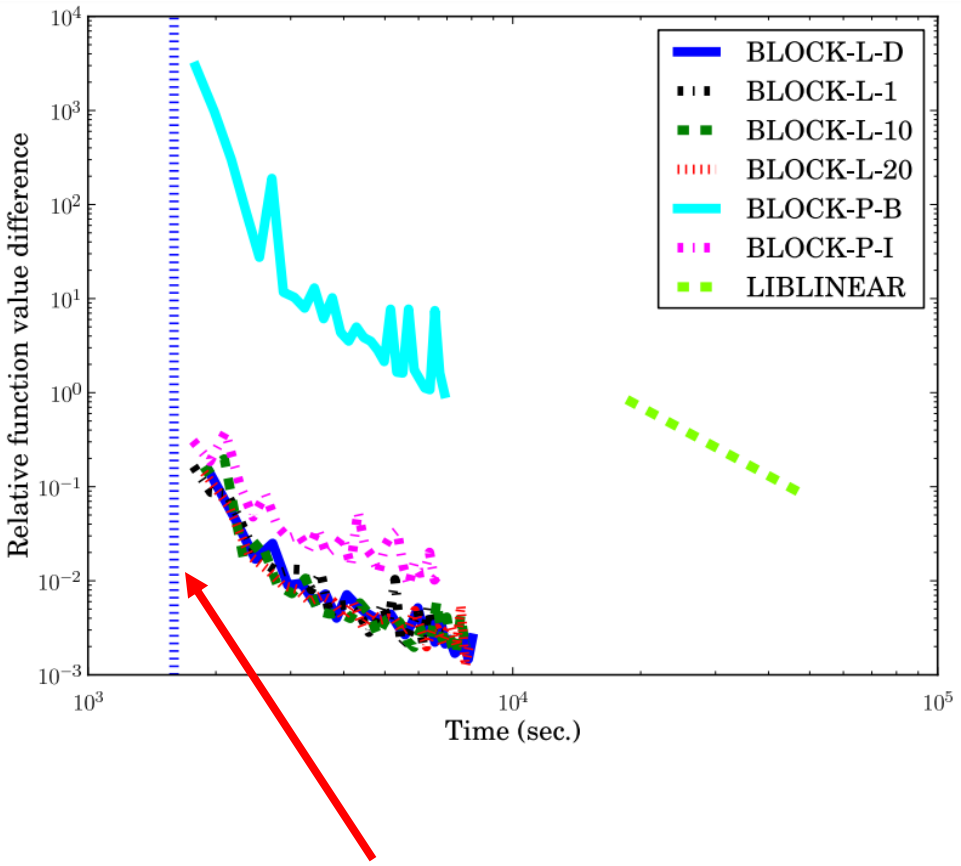
Data 20 times larger

Compared Methods

- BLOCK-L-N: Solving dual. LIBLINEAR goes through the block of data N rounds, N = 1, 10, and 20.
- BLOCK-L-D: Solving dual. LIBLINEAR's default stopping condition is adopted.
- BLOCK-P-B: Solving primal. Pegasos update on each whole block (one update).
- BLOCK-P-I: Solving primal. Pegasos updates each individual data instance of the block (|B| updates).
- LIBLINEAR: The standard LIBLINEAR without any modification to handle the situation if data cannot fit in memory.

Experiment Results

webspam



Time for initial block split

yahoo-korea

