CS281B/Stat241B: Advanced Topics in Learning & Decision Making

Support Vector Machines

Lecturer: Michael I. Jordan *Scribes: Mark Whitney*

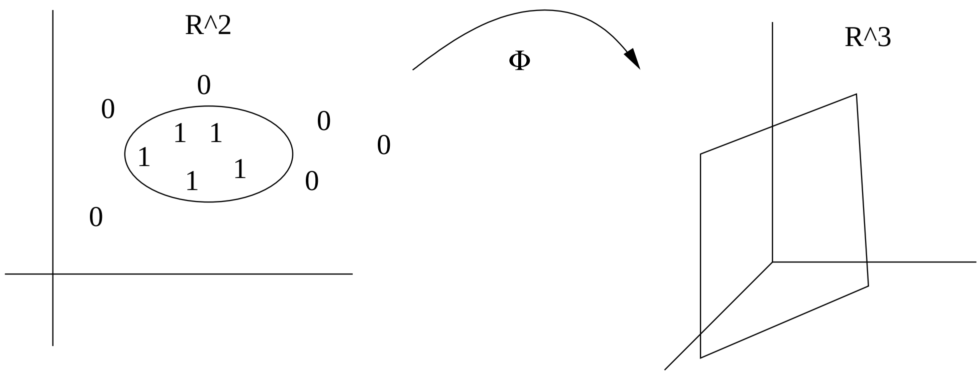
There is a new homework assignment out; here are a few corrections/clarifications:

Problem 2

$$\Re^{backwards\ \epsilon} \to \Re^3$$

$$\Re^{\epsilon} \to \Re^2$$

Everything inside the ellipse is in class 1, everything outside the ellipse is in class 0.



Problem 5 This problem requires Matlab and is an implementation of a particle filter. Which leads to the following discussion of importance sampling.

1 Importance sampling

Basic idea: try to calculate expectation by sampling from a simpler distribution.

p is the probability distribution of X to sample from and $f(X)$ is the function for which we want the expectation.

$$E(f) = \int f(x)p(x)dx \approx \frac{1}{N}\sum_i f(x_i), x_i \sim p(x)$$

If $p(x)$ is too hard to sample from, introduce a new distribution on X, $q(x)$, which is easier to sample from, and thus calculate the expectation.

$$E(f) = \int \frac{f(x)p(x)}{q(x)} q(x)dx \approx \frac{1}{N} \sum_i \frac{f(x_i)p(x_i)}{q(x_i)}, x_i \sim q(x)$$

In this case, one cannot sample from p but can calculate $p(x_i)$. In some cases one cannot even calculate $p(x_i)$ but only a unnormalized version of $p(x_i)$, $\tilde{p}(x_i)$. In this case normalize over the term $\frac{\tilde{p}(x_i)}{\tilde{q}(x_i)}$.

An improvement on importance sampling is called *SIR (sampling importance resampling)* (Rubin, 1988). Here, a tractable $q(x)$ is again used.

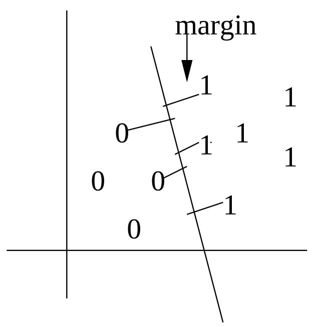
After importance sampling with $q(x)$, each sample has weight $w_i = \frac{p(x_i)}{q(x_i)}$, normalize each weight from the samples $w_i' = \frac{w_i}{\sum_i w_i}$ and resample from the samples using the w_i' weights as multinomial probalities. This method concentrates the samples to where p has more mass.

The particle filter from problem 5 in the homework applies SIR sequentially to the data points, maintaining a set of samples or "particles". This gives a posterior distribution of the data.

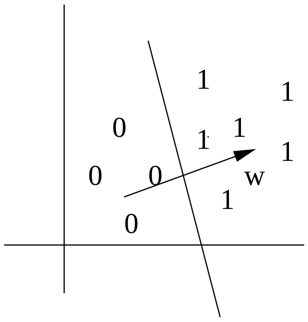
2 Support vector machines

SVMs perform classification, making use of two key ideas: maximal margin classification and a "kernel trick."

2.1 Maximal margin classification



The SVM tries to find a hyperplane that separates the differently classified data the most. It maximizes the minimal distance between the hyperplane and each data class.



w is the weight vector orthogonal to the hyperplane and we want to impose the following constraints:

$$w^T x + b \geq c \; for \; all \; x \; in \; class \; 1$$

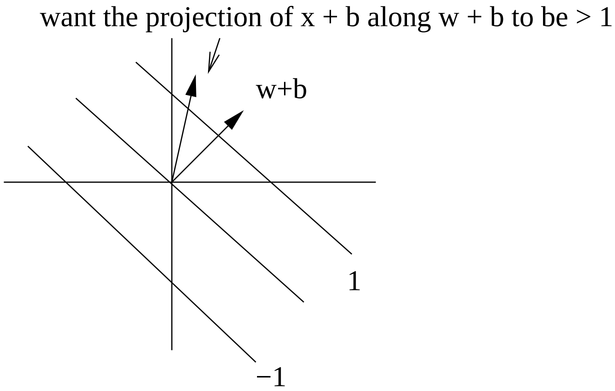
$$w^T x + b \leq -c \; for \; all \; x \; in \; class \; 0$$

or, in standard form, divide by c:

$$w^T x + b \geq 1 \; for \; all \; x \; in \; class \; 1$$

$$w^T x + b \leq -1 \; for \; all \; x \; in \; class \; 0$$

Here is a picture:



The b term in the constraints just moves the w vector to the origin, we want the data vectors to have w components that are greater than 1 or less than -1.

The optimization algorithm underlying the SVM is a quadratic programming problem—we want to *minimize* $w^T w$ under the constraints. Why is this?

Define the margin $= \|x_1^* - x_0^*\|$ where the x_0^* and x_1^* vectors are locations of hyperplanes, parallel to the classifying hyperplane, delimiting the border where the data of that class comes closest to the classifying hyperplane.

$$w^T x_1^* + b - (w^T x_0^* + b) = 1 - (-1) = 2$$

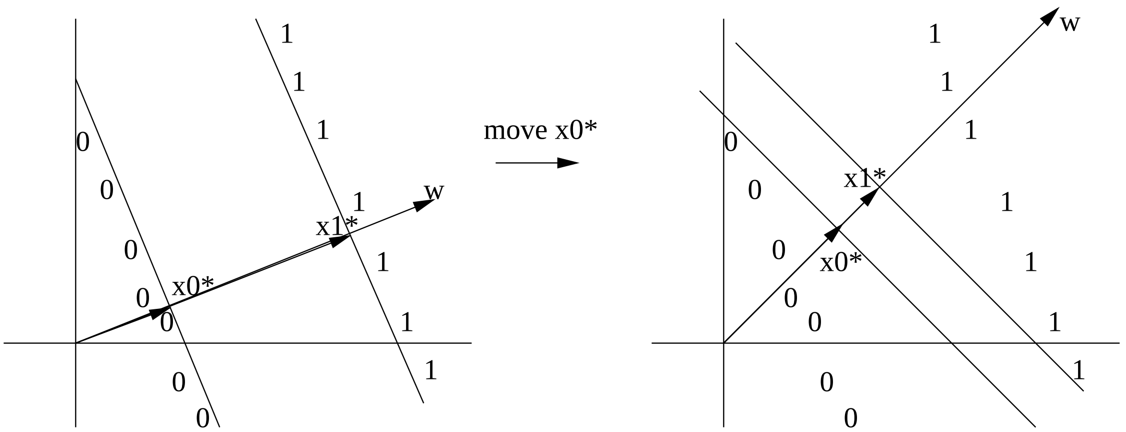
$$w^T (x_1^* - x_0^*) = 2$$

$$w^T(x_1^* - x_0^*) = \|w\|\|x_1^* - x_2^*\|$$
$$\left(\frac{w}{\|w\|}\right)^T (x_1^* - x_0^*) = \|x_1^* - x_2^*\|$$

The last two equalities are true by Cauchy-Schwartz.

All this implies that $\|x_1^* - x_0^*\| = \frac{2}{(w^T w)^{1/2}}$, so to maximize the margin, we must minimize $w^T w$.

Here is an example:



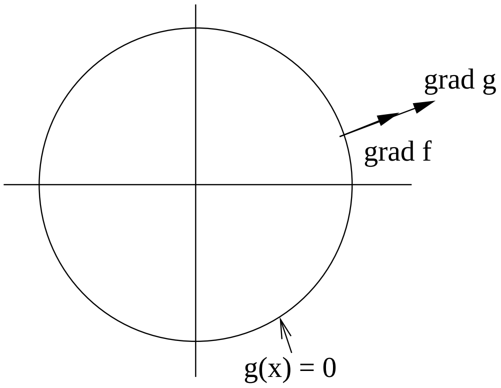
The left configuration is a good placement of the hyperplane, the norm $\|x_1^* - x_0^*\|$ is pretty large and thus the w vector does not need to be too long to satisfy the constraints.

The right configuration is not a good placement, the norm $\|x_1^* - x_0^*\|$ is quite small and thus the w vector will need to be large to satisfy the constraints.

Solving for minimum $w^T w$ under constraints is a quadratic programming problem.

3 Constrained optimization

The goal is to minimize $f(x)$ such that $g(x) = 0$.



The only points where ∇f points in the same direction as ∇g could f be at a local minima. Otherwise, we could move opposite ∇f along the circle and further reduce $f(x)$. Thus the condition for optimality is:

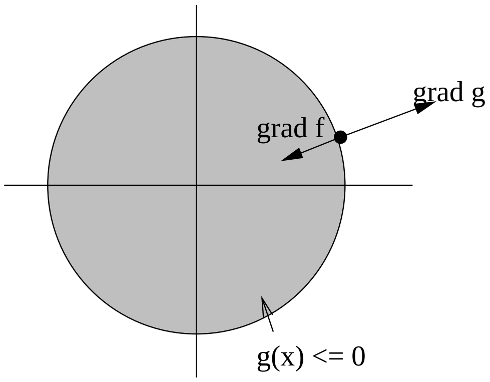
$f(x)$ is in a local optimum if ∇f and ∇g are collinear (i.e. $\nabla f = \lambda \nabla g$ for some λ)

Introduce an augmented function $\tilde{f} = f - \lambda g$, take gradient of $\tilde{f}$ and set to zero:

$$\nabla \tilde{f} = \nabla f - \lambda \nabla g = 0$$

So, $\nabla f = \lambda \nabla g$.

What if the constraint were $g(x) \leq 0$?



What is the condition for optimality for $f(x)$ now?

∇f must be collinear and opposite in direction from ∇g. Thus, no perturbation can make $f(x)$ further decrease. $\nabla f = \lambda \nabla g \ for \ \lambda < 0$

This leads to the Karush-Kuhn-Tucker (KKT) conditions:

Minimize $f(x)$ such that $g_i(x) \leq 0$ and $h_i(x) = 0$

If $\bar{x}$ is a local minimum, $\exists u_i, v_i$ such that

$$\nabla f(\bar{x}) + \sum_i u_i \nabla g_i(\bar{x}) + \sum_i v_i \nabla h_i(\bar{x}) = 0, u_i \geq 0$$

The u_i coefficients are like λ from the previous problem of minimization of $f(x)$ w.r.t. the constraint $g(x) = 0$.

4 References

Rubin, D.B. Using the SIR algorithm to simulate posterior distributions. *Bayesian Statistics 3,* 1988.