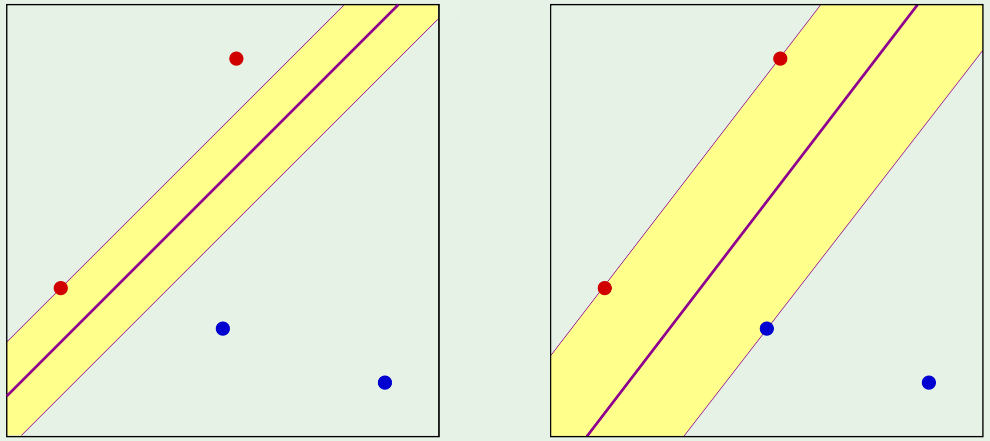
Review of Lecture 14

- The margin

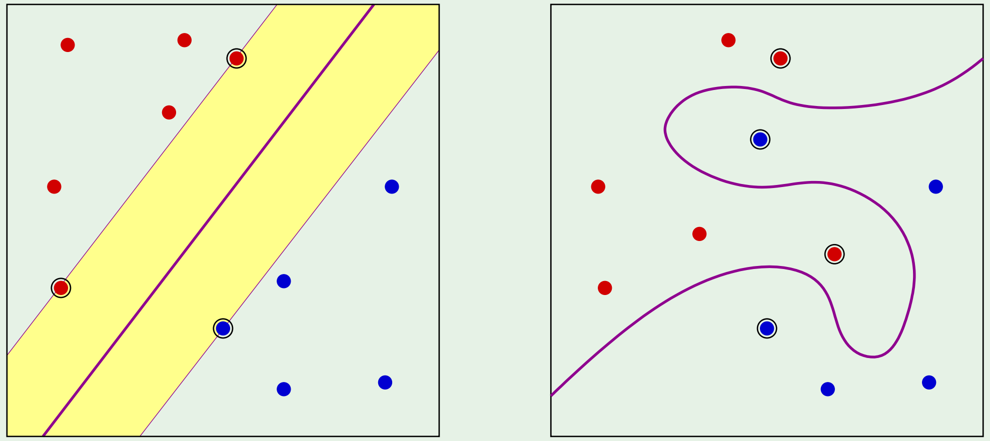


Maximizing the margin $\Longrightarrow$ dual problem:

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \; \alpha_n \alpha_m \; \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m$$

quadratic programming

- Support vectors



$\mathbf{x}_n$ (or $\mathbf{z}_n$) with Lagrange $\alpha_n > 0$

$$\mathbb{E}[E_{\text{out}}] \leq \frac{\mathbb{E}[\#\text{ of SV's}]}{N-1}$$

(in-sample check of out-of-sample error)

since the number of SV's is an in-sample quantity

- Nonlinear transform

Complex h, but simple $\mathcal{H}$ ☺

since we can use high-dimensional Z-space without fully paying the price for it in terms of generalization error

Learning From Data

Yaser S. Abu-Mostafa
California Institute of Technology

Lecture 15: Kernel Methods

Sponsored by Caltech's Provost Office, E&AS Division, and IST • Tuesday, May 22, 2012

Outline

- The kernel trick - takes care of the non-linear transformation where the Z-space can be very sophisticated (infinite dimensional) without us paying a high price for it

- Soft-margin SVM - extends SVM from linearly separable case (hard-margin) to the (slightly) non linearly separable case, allowing ourself to make errors to improve generalization

It is likely that both of these will be used in a practical problem - you go to a high dimensional space and also allow some errors so that outliers do not dictate an unduly complex nonlinear transformation

What do we need from the $\mathcal{Z}$ space?

Do we need anything from the Z-space other than the inner product? If not, instead of calculating the vector z explicitly, it will be easier to take two vectors from x space and return the corresponding Z-space inner product.

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \, \alpha_n \alpha_m \, \mathbf{z}_n^{\intercal} \mathbf{z}_m$$

Constraints: $\alpha_n \geq 0$ for $n = 1, \cdots, N$ and $\sum_{n=1}^{N} \alpha_n y_n = 0$

$$\boxed{g(\mathbf{x}) = \operatorname{sign}\left(\mathbf{w}^{\intercal}\mathbf{z} + b\right)}$$ need $\mathbf{z}_n^{\intercal}\mathbf{z}$

where $\mathbf{w} = \sum_{\mathbf{z}_n \text{ is SV}} \alpha_n y_n \mathbf{z}_n$

and b: $y_m\left(\mathbf{w}^{\intercal}\mathbf{z}_m + b\right) = 1$ need $\mathbf{z}_n^{\intercal}\mathbf{z}_m$

Hence we only deal with z as far as the inner product is concerned. Now, if we can find the inner product in the Z-space without visiting the Z-space (or even knowing what the Z-space/transformation is), we can still use the above machinery.

Generalized inner product

Given two points $\mathbf{x}$ and $\mathbf{x}' \in \mathcal{X}$, we need $\mathbf{z}^\intercal \mathbf{z}'$

Let $\mathbf{z}^\intercal \mathbf{z}' = K(\mathbf{x}, \mathbf{x}')$ (the kernel) "inner product" of $\mathbf{x}$ and $\mathbf{x}'$ (a valid kernel is an inner product in some space)

corresponds to some Z-space

Example: $\mathbf{x} = (x_1, x_2) \longrightarrow$ 2nd-order Φ

$$\mathbf{z} = \Phi(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{z}^\intercal \mathbf{z}' = 1 + x_1 x'_1 + x_2 x'_2 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + x_1 x'_1 x_2 x'_2$$

The trick

Can we compute $K(\mathbf{x}, \mathbf{x}')$ **without** transforming $\mathbf{x}$ and $\mathbf{x}'$?

Example: Consider $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^{\intercal}\mathbf{x}')^2 = (1 + x_1 x'_1 + x_2 x'_2)^2$

Give an expression for a kernel and show that it actually corresponds to a transformation to some Z-space and then takes an inner product there

$$= 1 + x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2 x_1 x'_1 + 2 x_2 x'_2 + 2 x_1 x'_1 x_2 x'_2$$

This is an inner product!

$$(\ 1 \ , \ x_1^2 \ , \ x_2^2 \ , \ \sqrt{2}\, x_1 \ , \ \sqrt{2}\, x_2 \ , \ \sqrt{2}\, x_1 x_2 \ )$$

$$(\ 1 \ , \ x'^2_1 \ , \ x'^2_2 \ , \ \sqrt{2} x'_1 \ , \ \sqrt{2} x'_2 \ , \ \sqrt{2} x'_1 x'_2 \ )$$

The polynomial kernel

$\mathcal{X} = \mathbb{R}^d$ and $\Phi : \mathcal{X} \to \mathcal{Z}$ is polynomial of order Q

The "equivalent" kernel $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^\intercal \mathbf{x}')^Q$

$$= (1 + x_1 x'_1 + x_2 x'_2 + \cdots + x_d x'_d)^Q$$

A simple number to compute regardless of Q: take the log of (...), multiply by Q and exponentiate

Compare for $d = 10$ and $Q = 100$

Note that because whenever x appears, the x' version of it appears, and multiplying any combination will result in this still being the case, so if you expanded the brackets we have terms up to order Q of different combinations of the x's. Hence it should not be a surprise that we can decompose this into (something of x) dot (something of x')

Note:

Can adjust scale: $K(\mathbf{x}, \mathbf{x}') = (a\mathbf{x}^\intercal \mathbf{x}' + b)^Q$

if worried about square-rooted coefficients. The inclusion of a and b can mitigate some of the diversity of these coefficients

We only need $\mathcal{Z}$ to exist!

If $K(\mathbf{x}, \mathbf{x}')$ is an inner product in <u>some</u> space $\mathcal{Z}$, we are good.

THE RADIAL BASIS FUNCTION

Example: $$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \left\|\mathbf{x} - \mathbf{x}'\right\|^2\right)$$

We now have to show that there is a Z-space where K produces an inner product

To see this,

Infinite-dimensional $\mathcal{Z}$: take simple case d = 1

We want to separate this into an inner product, so something coming from x and something from x' and that these are the same (so same transformation applied to both x and x'). Once they are the same, the dimensionality comes from the number of terms in the summation. It is easy to see that we get equivalent terms in the summation from x and x' - all we need to do is square root the coefficients to divide them between the transformations. Now we formally have to identical vectors, on transformed from x and one from x'.

$$K(x, x') = \exp\left(-(x - x')^2\right)$$

via Taylor series

$$= \exp\left(-x^2\right) \exp\left(-{x'}^2\right) \underbrace{\sum_{k=0}^{\infty} \frac{2^k (x)^k (x')^k}{k!}}_{\exp(2xx')}$$

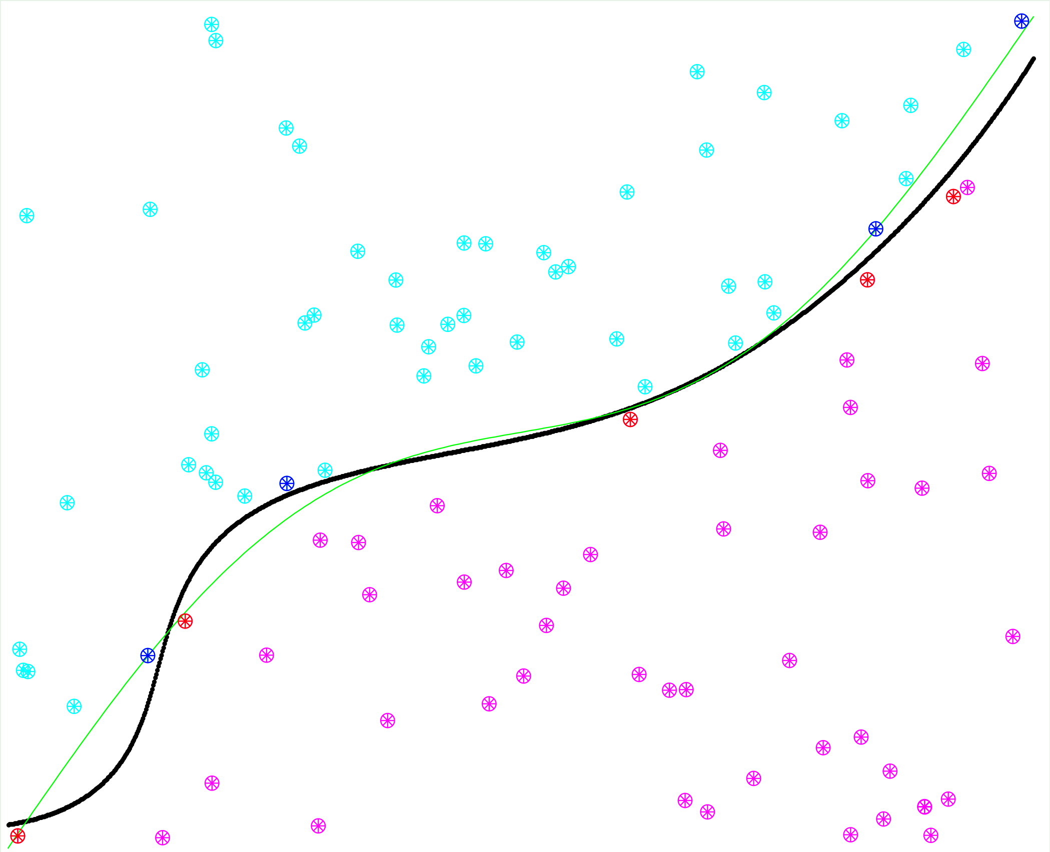
Note that as k increases, the factors decay with an exponential and 1/factorial factor, so the higher order terms decay very quickly. Since we are measuring a Euclidean distance proper in this space, if a dimension is very small it will not affect this distance very much. This means that the inner product converges (having a decaying term is a property of defining inner products in infinite spaces) and the effective number of dimensions is actually quite small (infinite dimensional in-disguise)

This kernel in action

Slightly non-separable case:

Transforming $\mathcal{X}$ into ∞-dimensional $\mathcal{Z}$

Overkill? Count the support vectors



Here we have 9 SV's for 100 data points, so Eout is well bounded
to under around 10%. A relatively small number of SV's equally indicates
that the maximized margin is of reasonable size (in Z-space).
Note that the black line, which is our final hypothesis hyperplane
separating the two classes, is found by classifying all of the points around
the boundary and finding where the model goes from returning -1 to +1.
We see that this black line is 'supported' up by the SV's, hence the name support vectors. Also,
since the SV's in x-space are simply pre-images of the 'proper' SV's in Z-space, they do not
denote the margin and so are not at equal (minimal) distance from the boundary.

Kernel formulation of SVM

Remember quadratic programming? The only difference now is:

$$\underbrace{\begin{bmatrix} y_1y_1\,K(\mathbf{x}_1,\mathbf{x}_1) & y_1y_2\,K(\mathbf{x}_1,\mathbf{x}_2) & \dots & y_1y_N K(\mathbf{x}_1,\mathbf{x}_N) \\ y_2y_1\,K(\mathbf{x}_2,\mathbf{x}_1) & y_2y_2\,K(\mathbf{x}_2,\mathbf{x}_2) & \dots & y_2y_N K(\mathbf{x}_2,\mathbf{x}_N) \\ \dots & \dots & \dots & \dots \\ y_Ny_1 K(\mathbf{x}_N,\mathbf{x}_1) & y_Ny_2 K(\mathbf{x}_N,\mathbf{x}_2) & \dots & y_Ny_N K(\mathbf{x}_N,\mathbf{x}_N) \end{bmatrix}}_{\text{quadratic coefficients}}$$

Everything else is the same.

The final hypothesis

Express $\quad g(\mathbf{x}) = \text{sign}\left(\mathbf{w}^{\intercal}\mathbf{z} + b\right) \quad$ in terms of $K(-,-)$

since we don't need anything from the Z-space other than the inner product, represented by K

$$\mathbf{w} = \sum_{\mathbf{z}_n \text{ is SV}} \alpha_n y_n \mathbf{z}_n \quad \Longrightarrow \quad g(\mathbf{x}) = \text{sign}\left(\sum_{\alpha_n > 0} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + b\right)$$

Here we see that Support Vector Machines gives different models (or hypothesis sets) depending on the choice of kernel

where $\; b = y_m - \sum_{\alpha_n > 0} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}_m)$

where m corresponds to any SV

for any support vector $(\alpha_m > 0)$

Note: the 'transformation' K(xn, x) depends on the dataset, while z does not (e.g. if you choose the RBF kernel, z can be determined before looking at the dataset and K cannot). We have seen this before with the hidden layer in NN - the hidden layer gets a non-linear transform based on the dataset, so the above situation is not foreign to us. The kernel form of g(x) allows us to compare SVMs to other approaches: say we have the RBF kernel, we have a functional form of g(x) - it is then legitimate to try solve a learning problem based on this without ever using SVM; it is just a model, so we can attempt to find a solution. You can compare the result of trying to solve it this way to the SVM route. This can also be done for neural networks and other kernels that we have.

How do we know that $\mathcal{Z}$ exists ...

... for a given $K(\mathbf{x}, \mathbf{x}')$? valid kernel

Three approaches:

1. By construction like the polynomial transformation

2. Math properties (*Mercer's condition*)

3. Who cares? ☺

Design your own kernel

$K(\mathbf{x}, \mathbf{x}')$ is a valid kernel iff

If K(xi,xj) is a genuine inner product and we had it explicitly, each term in the matrix will be the inner product in Z-space between the two data points, so we can decompose the matrix as an outer product between the column vector z and row vector z (its transpose)

1. It is symmetric and **2.** The matrix:

$$\begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \ldots & K(\mathbf{x}_1, \mathbf{x}_N) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \ldots & K(\mathbf{x}_2, \mathbf{x}_N) \\ \ldots & \ldots & \ldots & \ldots \\ K(\mathbf{x}_N, \mathbf{x}_1) & K(\mathbf{x}_N, \mathbf{x}_2) & \ldots & K(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

is **positive semi-definite**

conceptually meaning the matrix should be greater than/equal to zero

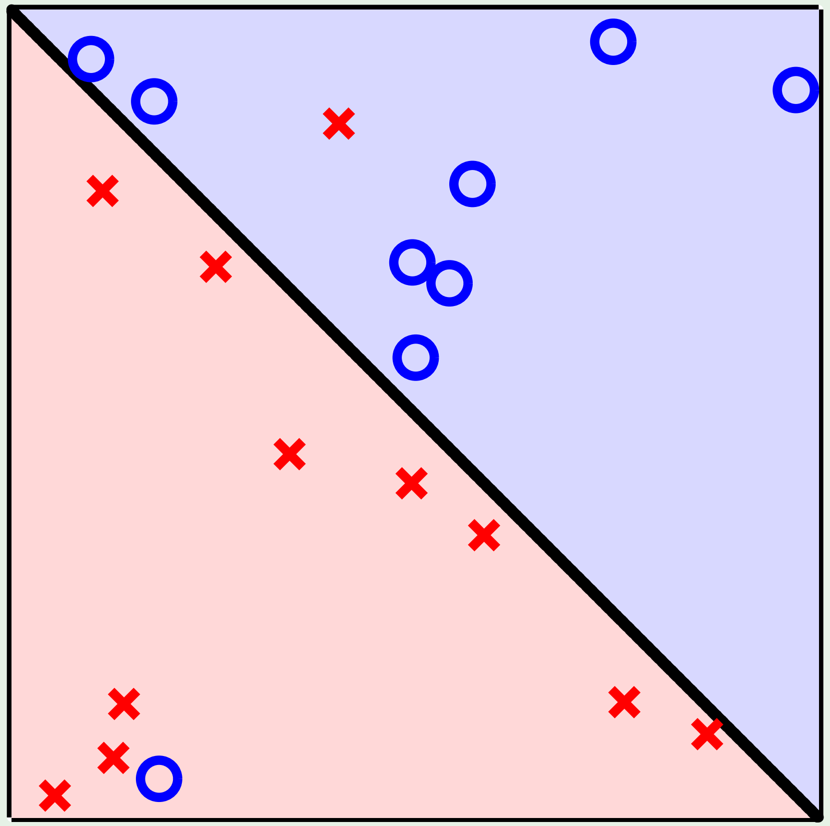
for any $\mathbf{x}_1, \cdots, \mathbf{x}_N$ (Mercer's condition)

Outline

- The kernel trick
- Soft-margin SVM

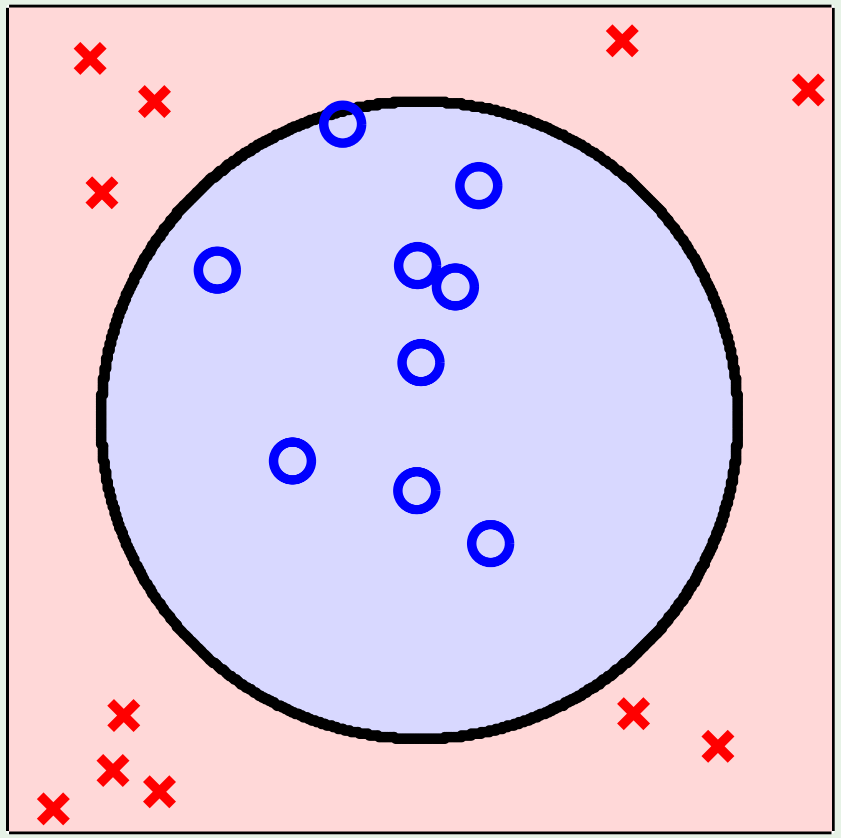
Two types of non-separable

slightly:



A few outliers - do not want to go to a high dimensional non-linear space to try classify these. It is also likely that we would have many SV's in order to do this, so poor generalization. Like the 'pocket' algorithm, we would like to make some errors on the outliers, accept a non-zero Ein but get better Eout rather than insist on Ein = 0 and end up with large Eout due to an inordinately complex transformation. Soft-margin SVM deal with this situation.

seriously:



Kernels deal with this situation, where it is not a question of outliers and we definitely have to use a non-linear transformation.

In a practical dataset, it is likely it will have aspects of both types (a built-in non-linearity and some annoying outliers), so we will combine the kernel with the soft-margin SVM.

Error measure

There are many possible ways to consider errors, e.g. we could consider the number of points we misclassify. However, dealing with the *number* of misclassified points is not a good idea as the optimization becomes completely intractable - it is a combinatorial optimization and, like with the perceptron and 'pocket' algorithm, getting the absolute optimal is NP hard.

We base the error on

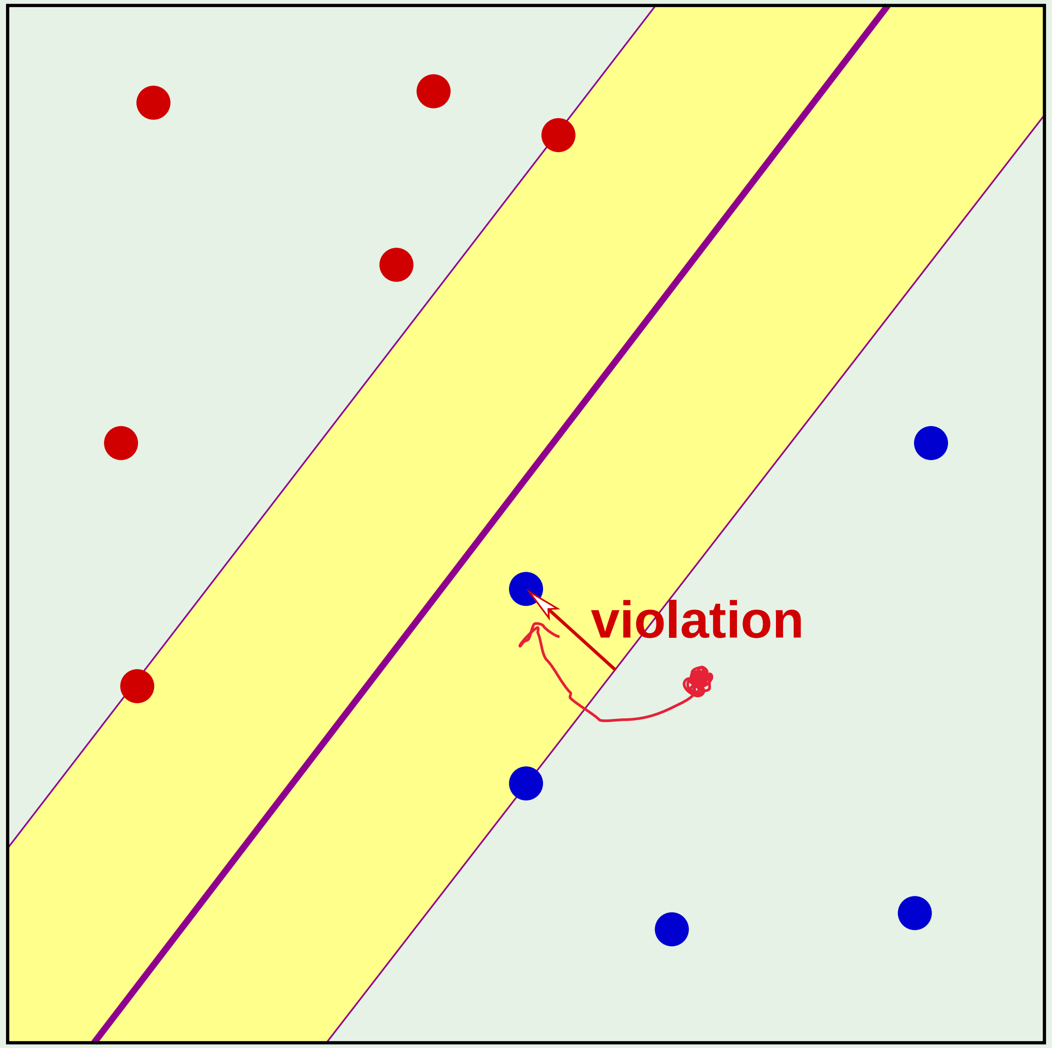
which occurs when the distance from xn to the line

Margin violation: $y_n\left(\mathbf{w}^\intercal\mathbf{x}_n + b\right) \geq 1$ fails

Introduce a slack for every point (even if most do not violate the margin):

Quantify: $y_n\left(\mathbf{w}^\intercal\mathbf{x}_n + b\right) \geq 1 - \xi_n \qquad \xi_n \geq 0$

$$\text{Total violation} = \sum_{n=1}^{N} \xi_n$$



Xi being greater than or equal to zero means that we only penalise violations of the margin, we do not reward the anti-violation of the margin (we do not give credit for having points very far from the margin)

The new optimization

To maximise the margin:

C gives the relative importance of the second (error) term compared to first term - no different to the notion of augmented error. Large C discourages any violation (very strict hard-margin), while very small C allows frequent violation

Minimize $\quad \frac{1}{2}\,\mathbf{w}^{\intercal}\mathbf{w} + C\sum_{n=1}^{N}\xi_n$

subject to $\quad y_n\left(\mathbf{w}^{\intercal}\mathbf{x}_n + b\right) \geq 1 - \xi_n \quad \text{for} \quad n = 1, \ldots, N$

and $\quad \xi_n \geq 0 \quad \text{for} \quad n = 1, \ldots, N$

$$\mathbf{w} \in \mathbb{R}^d \;, \quad b \in \mathbb{R} \;, \quad \boldsymbol{\xi} \in \mathbb{R}^N$$

Lagrange formulation

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\,\mathbf{w}^{\intercal}\mathbf{w} + C\sum_{n=1}^{N}\xi_n - \sum_{n=1}^{N}\alpha_n(y_n\,(\mathbf{w}^{\intercal}\mathbf{x}_n + b) - 1 + \xi_n) - \sum_{n=1}^{N}\beta_n\,\xi_n$$

Minimize w.r.t. $\mathbf{w}$, b, and $\boldsymbol{\xi}$ and maximize w.r.t. each $\alpha_n \geq 0$ and $\beta_n \geq 0$

L = (things to be minimized) + LagrangeMultipliers*constraints with the signs of the constraint terms determined by the type of inequality

$$\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n = \mathbf{0}$$

$$\frac{\partial\mathcal{L}}{\partial b} = -\sum_{n=1}^{N}\alpha_n y_n = 0$$

$$\frac{\partial\mathcal{L}}{\partial\xi_n} = C - \alpha_n - \beta_n = 0$$

Collecting the summations of xi in the above Lagrangian, we see that xi conveniently cancels out and we get the exact same Lagrangian as in the hard-margin case with the same solution for grad(L) and dL/db we had before. Beta did its service and bid us farewell - the ONLY ramification of beta is that because it is >= 0, from the 3rd condition we require that alpha is not only >= 0 but it also must be <= C (see next slide).

and the solution is . . .

Maximize $\mathcal{L}(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \, \alpha_n \alpha_m \, \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m$ w.r.t. to $\boldsymbol{\alpha}$

subject to $0 \le \alpha_n \le C$ for $n = 1, \cdots, N$ and $\sum_{n=1}^{N} \alpha_n y_n = 0$

So all we need to change in our routine to apply SVM is that 0 <= alpha <= C instead of 0 <= alpha <= inf.

$$\Longrightarrow \quad \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$$

minimizes $\frac{1}{2} \mathbf{w}^{\mathsf{T}} \mathbf{w} + C \sum_{n=1}^{N} \xi_n$

Remember a point
is a SV if alpha > 0

Types of support vectors

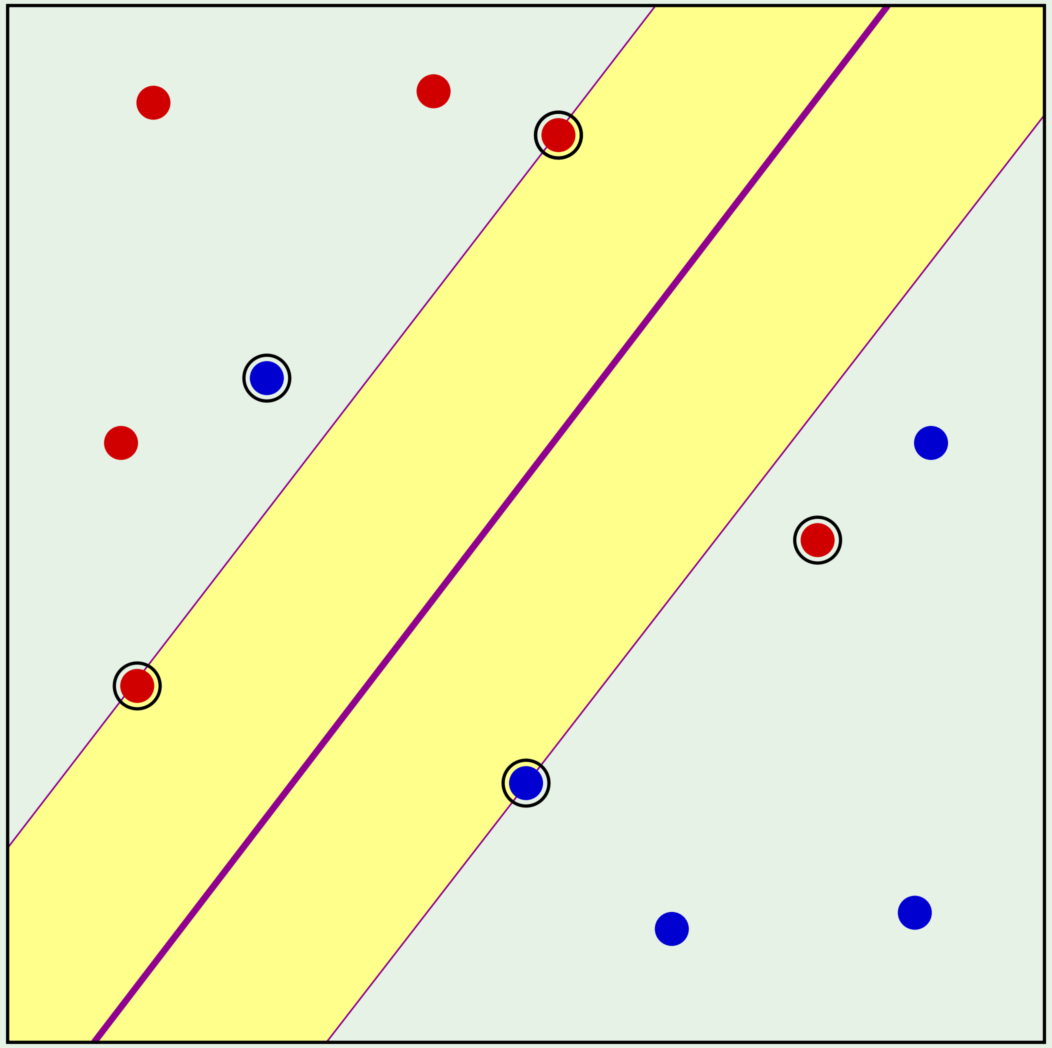
instead of the hard-margin case where we only have regular SV and interior points

margin support vectors $(0 < \alpha_n < C)$

$$y_n\left(\mathbf{w}^{\intercal}\mathbf{x}_n + b\right) = 1 \qquad (\xi_n = 0)$$

non-margin support vectors $(\alpha_n = C)$

$$y_n\left(\mathbf{w}^{\intercal}\mathbf{x}_n + b\right) < 1 \qquad (\xi_n > 0)$$



When a Lagrange multiplier is zero, the corresponding slack must become positive. e.g. in the case of alpha, if alpha=0 then the slack yn(wTxn + b) - 1 is positive (i.e. alpha=0 corresponds to an interior point which does not touch the margin). From the third condition on slide 17, when alpha=C, beta=0. When beta=0, the corresponding slack xi must be positive and so the point must violate the margin. This explains why margin SV's (which have xi=0) have alpha > 0 but not C, and non-margin SV's (which have a positive xi) have alpha=C.

Note there are two types of non-margin SV's: one type will still be classified correctly (the point violates the margin but does not cross the line, so Ein is unaffected) and the other type will be incorrectly classified (the point crosses the line and Ein is non-zero) - the second type is shown in the above diagram. The value of C is important here since it tells us how much violation we have versus the width of margin - we pick C optimally in a practical problem using cross-validation.

Two technical observations

The mathematical translation from the primal form (minimize (1/2)wTw) to the dual form (maximizing w.r.t. alpha the Lagrangian) is only valid if the data is linearly separable (i.e. if there is a feasible solution). The KKT conditions are necessary (they have to be satisfied if the point is there) - if there is no point in the domain, we have no guarantees of a reasonable solution. If we do translate to a Lagrangian, the QP will try to converge to something in infinity - i.e. it will complain. Note that there is no need to worry about this situation: we do not need to explicitly check that the data is linearly separable with a perceptron convergence or w/e . We can simply be lazy and translate to the dual, pass to QP, and receive a list of alphas. Now we can just check if the solution separates the data - evaluate the solution on each point and compare with the label. When we realize that it does not agree with the label, we realize that something is wrong. On the other hand if it is linearly separable data we will get a feasible solution.

1. Hard margin: What if data is not linearly separable?

"primal $\longrightarrow$ dual" breaks down

2. $\mathcal{Z}$: What if there is w_0?

in the Z-space weight vector

All goes to b and $w_0 \to 0$

since we are minimizing (1/2)wTw, not b, so when we get the solution all the bias is captured in b