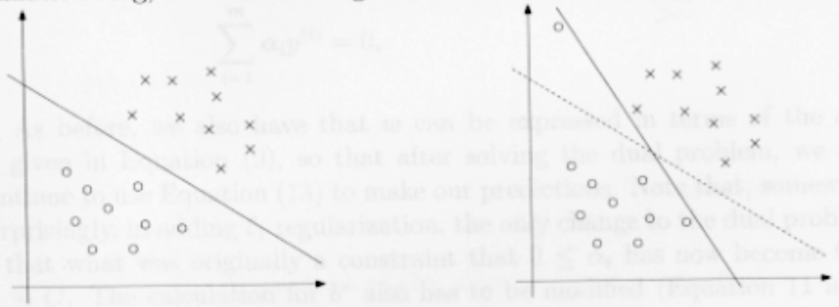
algorithms that we'll see later in this class will also be amenable to this method, which has come to be known as the "kernel trick."

8 Regularization and the non-separable case

The derivation of the SVM as presented so far assumed that the data is linearly separable. While mapping data to a high dimensional feature space via ϕ does generally increase the likelihood that the data is separable, we can't guarantee that it always will be so. Also, in some cases it is not clear that finding a separating hyperplane is exactly what we'd want to do, since that might be susceptible to outliers. For instance, the left figure below shows an optimal margin classifier, and when a single outlier is added in the upper-left region (right figure), it causes the decision boundary to make a dramatic swing, and the resulting classifier has a much smaller margin.



To make the algorithm work for non-linearly separable datasets as well as be less sensitive to outliers, we reformulate our optimization (using ℓ_1 **regularization**) as follows:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \ \ i = 1, \ldots, m \\ & \xi_i \geq 0, \ \ i = 1, \ldots, m. \end{aligned}$$

Thus, examples are now permitted to have (functional) margin less than 1, and if an example whose functional margin is $1 - \xi_i$, we would pay a cost of the objective function being increased by $C\xi_i$. The parameter C controls the relative weighting between the twin goals of making the $||w||^2$ large (which we saw earlier makes the margin small) and of ensuring that most examples have functional margin at least 1.

As before, we can form the Lagrangian:

$$\mathcal{L}(w,b,\xi,\alpha,r) = \frac{1}{2}w^T w + C\sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i\left[y^{(i)}(x^T w + b) - 1 + \xi_i\right] - \sum_{i=1}^m r_i\xi_i.$$

Here, the α_i's and r_i's are our Lagrange multipliers (constrained to be ≥ 0). We won't go through the derivation of the dual again in detail, but after setting the derivatives with respect to w and b to zero as before, substituting them back in, and simplifying, we obtain the following dual form of the problem:

$$\begin{aligned}\max_\alpha \quad & W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \ \ i = 1,\ldots,m \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0,\end{aligned}$$

As before, we also have that w can be expressed in terms of the α_i's as given in Equation (9), so that after solving the dual problem, we can continue to use Equation (13) to make our predictions. Note that, somewhat surprisingly, in adding ℓ_1 regularization, the only change to the dual problem is that what was originally a constraint that $0 \leq \alpha_i$ has now become $0 \leq \alpha_i \leq C$. The calculation for b^* also has to be modified (Equation 11 is no longer valid); see the comments in the next section/Platt's paper.

Also, the KKT dual-complementarity conditions (which in the next section will be useful for testing for the convergence of the SMO algorithm) are:

$$\alpha_i = 0 \ \Rightarrow \ y^{(i)}(w^T x^{(i)} + b) \geq 1 \tag{14}$$

$$\alpha_i = C \ \Rightarrow \ y^{(i)}(w^T x^{(i)} + b) \leq 1 \tag{15}$$

$$0 < \alpha_i < C \ \Rightarrow \ y^{(i)}(w^T x^{(i)} + b) = 1. \tag{16}$$

Now, all that remains is to give an algorithm for actually solving the dual problem, which we will do in the next section.

9 The SMO algorithm

The SMO (sequential minimal optimization) algorithm, due to John Platt, gives an efficient way of solving the dual problem arising from the derivation of the SVM. Partly to motivate the SMO algorithm, and partly because it's interesting in its own right, lets first take another digression to talk about the coordinate ascent algorithm.

9.1 Coordinate ascent

Consider trying to solve the unconstrained optimization problem

$$\max_{\alpha} W(\alpha_1, \alpha_2, \ldots, \alpha_m).$$

Here, we think of W as just some function of the parameters α_i's, and for now ignore any relationship between this problem and SVMs. We've already seen two optimization algorithms, gradient ascent and Newton's method. The new algorithm we're going to consider here is called **coordinate ascent**:

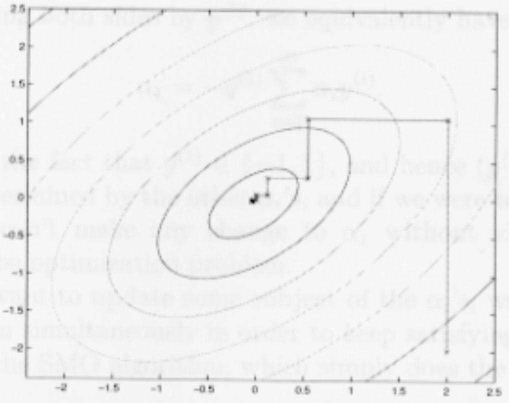
```
Loop until convergence: {

    For $i = 1, \ldots, m$, {
        $\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$.
    }

}
```

Thus, in the innermost loop of this algorithm, we will hold all the variables except for some α_i fixed, and reoptimize W with respect to just the parameter α_i. In the version of this method presented here, the inner-loop reoptimizes the variables in order $\alpha_1, \alpha_2, \ldots, \alpha_m, \alpha_1, \alpha_2, \ldots$. (A more sophisticated version might choose other orderings; for instance, we may choose the next variable to update according to which one we expect to allow us to make the largest increase in $W(\alpha)$.)

When the function W happens to be of such a form that the "arg max" in the inner loop can be performed efficiently, then coordinate ascent can be a fairly efficient algorithm. Here's a picture of coordinate ascent in action:



The ellipses in the figure are the contours of a quadratic function that we want to optimize. Coordinate ascent was initialized at $(2, -2)$, and also plotted in the figure is the path that it took on its way to the global maximum. Notice that on each step, coordinate ascent takes a step that's parallel to one of the axes, since only one variable is being optimized at a time.

9.2 SMO

We close off the discussion of SVMs by sketching the derivation of the SMO algorithm. Some details will be left to the homework, and for others you may refer to the paper excerpt handed out in class.

Here's the (dual) optimization problem that we want to solve:

$$\max_\alpha \quad W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle. \tag{17}$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \quad i = 1, \ldots, m \tag{18}$$

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0. \tag{19}$$

Lets say we have set of α_i's that satisfy the constraints (18-19). Now, suppose we want to hold $\alpha_2, \ldots, \alpha_m$ fixed, and take a coordinate ascent step and reoptimize the objective with respect to α_1. Can we make any progress? The answer is no, because the constraint (19) ensures that

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

Or, by multiplying both sides by $y^{(1)}$, we equivalently have

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

(This step used the fact that $y^{(1)} \in \{-1, 1\}$, and hence $(y^{(1)})^2 = 1$.) Hence, α_1 is exactly determined by the other α_i's, and if we were to hold $\alpha_2, \ldots, \alpha_m$ fixed, then we can't make any change to α_1 without violating the constraint (19) in the optimization problem.

Thus, if we want to update some subject of the α_i's, we must update at least two of them simultaneously in order to keep satisfying the constraints. This motivates the SMO algorithm, which simply does the following:

```
Repeat till convergence {
    1. Select some pair $\alpha_i$ and $\alpha_j$ 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 $W(\alpha)$ with respect to $\alpha_i$ and $\alpha_j$, while holding all the
       other $\alpha_k$'s ($k \neq i, j$) fixed.
}
```

To test for convergence of this algorithm, we can check whether the KKT conditions (Equations 14-16) are satisfied to within some *tol*. Here, *tol* is the convergence tolerance parameter, and is typically set to around 0.01 to 0.001. (See the paper and pseudocode for details.)

The key reason that SMO is an efficient algorithm is that the update to α_i, α_j can be computed very efficiently. Lets now briefly sketch the main ideas for deriving the efficient update.

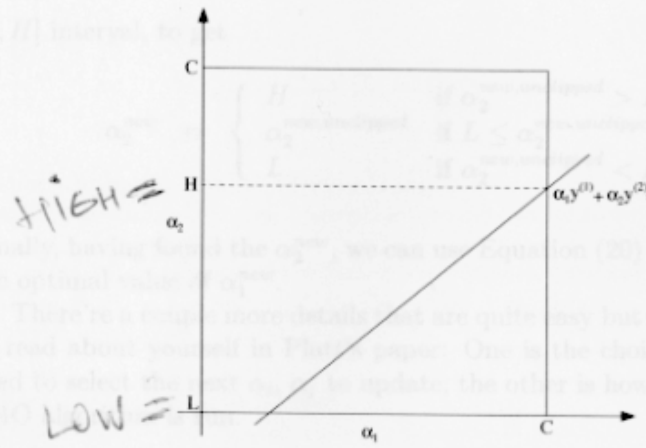
Lets say we currently have some setting of the α_i's that satisfy the constraints (18-19), and suppose we've decided to hold $\alpha_3, \ldots, \alpha_m$ fixed, and want to reoptimize $W(\alpha_1, \alpha_2, \ldots, \alpha_m)$ with respect to α_1 and α_2 (subject to the constraints). From (19), we require that

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

Since the right hand side is fixed (as we've fixed $\alpha_3, \ldots \alpha_m$), we can just let it be denoted by some constant ζ:

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta. \qquad (20)$$

We can thus picture the constraints on α_1 and α_2 as follows:



From the constraints (18), we know that α_1 and α_2 must lie within the box $[0, C] \times [0, C]$ shown. Also plotted is the line $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$, on which we know α_1 and α_2 must lie. Note also that, from these constraints, we know $L \leq \alpha_2 \leq H$; otherwise, (α_1, α_2) can't simultaneously satisfy both the box and the straight line constraint. In this example, $L = 0$. But depending on what the line $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$ looks like, this won't always necessarily be the case; but more generally, there will be some lower-bound L and some upper-bound H on the permissable values for α_2 that will ensure that α_1, α_2 lie within the box $[0, C] \times [0, C]$.

Using Equation (20), we can also write α_1 as a function of α_2:

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

(Check this derivation yourself; we again used the fact that $y^{(1)} \in \{-1, 1\}$ so that $(y^{(1)})^2 = 1$.) Hence, the objective $W(\alpha)$ can be written

$$W(\alpha_1, \alpha_2, \ldots, \alpha_m) = W((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, \ldots, \alpha_m).$$

Treating $\alpha_3, \ldots, \alpha_m$ as constants, you should be able to verify that this is just some quadratic function in α_2. I.e., this can also be expressed in the form $a\alpha_2^2 + b\alpha_2 + c$ for some appropriate a, b, and c. If we ignore the "box" constraints (18) (or, equivalently, that $L \leq \alpha_2 \leq H$), then we can easily maximize this quadratic function by setting its derivative to zero and solving. We'll let $\alpha_2^{new,unclipped}$ denote the resulting value of α_2. You should also be able to convince yourself that if we had instead wanted to maximize W with respect to α_2 but subject to the box constraint, then we can find the resulting value optimal simply by taking $\alpha_2^{new,unclipped}$ and "clipping" it to lie in the

$[L, H]$ interval, to get

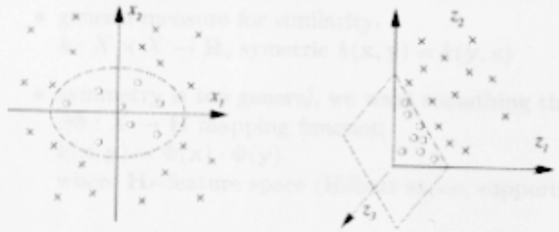
$$\alpha_2^{new} = \begin{cases} H & \text{if } \alpha_2^{new,unclipped} > H \\ \alpha_2^{new,unclipped} & \text{if } L \le \alpha_2^{new,unclipped} \le H \\ L & \text{if } \alpha_2^{new,unclipped} < L \end{cases}$$

Finally, having found the α_2^{new}, we can use Equation (20) to go back and find the optimal value of α_1^{new}.

There're a couple more details that are quite easy but that we'll leave you to read about yourself in Platt's paper: One is the choice of the heuristics used to select the next α_i, α_j to update; the other is how to update b as the SMO algorithm is run.

3 The need for kernels example

polynomial kernel : toy example



$ax^2+by^2 \leq -c$

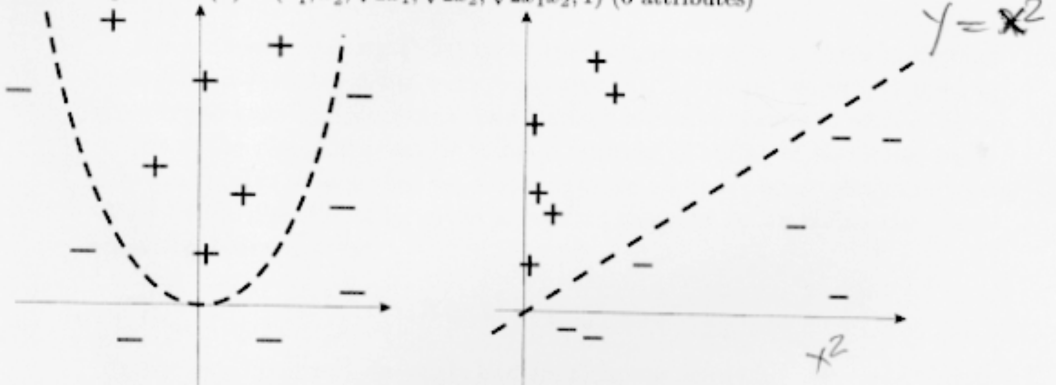
ellipse $ax^2+by^2+c=0$

use the map $\mathbf{x}=(x_1,x_2) \rightarrow \Phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
ellipse from 2D-input space becomes hyperplane into 3D-feature space

note $C_2(\mathbf{x}) = (x_1^2, x_2^2, x_1x_2, x_2x_1)$ maps data in a 4D-feature space but it generates the same kernel
$k(\mathbf{x},\mathbf{y}) = \langle\Phi(\mathbf{x}),\Phi(\mathbf{y})\rangle = \langle C_2(\mathbf{x}), C_2(\mathbf{y})\rangle = x_1^2y_1^2 + x_2^2y_2^2 + 2x_1y_1x_2y_2$

feature space : example

input space : $\mathbf{x} = (x_1, x_2)$ (2 attributes)
feature space : $\Phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1)$ (6 attributes)



data similarities & dot product

- measurement of data similarities : a fundamental problem in ML
- reflects a priori knowledge of the problem/data
- dot product : a natural measure for similarity
  $\langle \mathbf{x} \cdot \mathbf{y} \rangle = \sum_i x_i \cdot y_i$
- dot product amounts to being able to carry all geometric constructions formulated in terms of angles, lengths and distances

(usually a good similarity can solve the problem)

$\cos(\mathbf{x},\mathbf{y}) = \frac{\langle\mathbf{x}\cdot\mathbf{y}\rangle}{\|\mathbf{x}\|\|\mathbf{y}\|}$ $\qquad$ $\|\mathbf{x}\| = \sqrt{\langle\mathbf{x}\cdot\mathbf{x}\rangle}$

4 Data similarity

feature space

- general measure for similarity
  $k : X \times X \to \mathbf{R}$, symetric $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$
- symmetry is too general, we want something that feels like dot product
  $\exists \Phi : X \to \mathbf{H}$ mapping function
  $k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$
  where $\mathbf{H}$=feature space (Hilbert space, supports dot product)

sion using the features x, x^2 and x^3 (say) to obtain a cubic function. To distinguish between these two sets of variables, we'll call the "original" input value the input **attributes** of a problem (in this case, x, the living area). When that is mapped to some new set of quantities that are then passed to the learning algorithm, we'll call those new quantities the input **features**. (Unfortunately, different authors use different terms to describe these two things, but we'll try to use this terminology consistently in these notes.) We will also let ϕ denote the **feature mapping**, which maps from the attributes to the features. For instance, in our example, we had

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}.$$

Rather than applying SVMs using the original input attributes x, we may instead want to learn using some features $\phi(x)$. To do so, we simply need to go over our previous algorithm, and replace x everywhere in it with $\phi(x)$.

Since the algorithm can be written entirely in terms of the inner products $\langle x, z\rangle$, this means that we would replace all those inner products with $\langle \phi(x), \phi(z)\rangle$. Specificically, given a feature mapping ϕ, we define the corresponding **Kernel** to be

$$K(x, z) = \phi(x)^T\phi(z).$$

Then, everywhere we previously had $\langle x, z\rangle$ in our algorithm, we could simply replace it with $K(x, z)$, and our algorithm would now be learning using the features ϕ.

Now, given ϕ, we could easily compute $K(x, z)$ by finding $\phi(x)$ and $\phi(z)$ and taking their inner product. But what's more interesting is that often, $K(x, z)$ may be very inexpensive to calculate, even though $\phi(x)$ itself may be very expensive to calculate (perhaps because it is an extremely high dimensional vector). In such settings, by using in our algorithm an efficient way to calculate $K(x, z)$, we can get SVMs to learn in the high dimensional feature space space given by ϕ, but without ever having to explicitly find or represent vectors $\phi(x)$.

Lets see an example. Suppose $x, z \in \mathbb{R}^n$, and consider

$$K(x, z) = (x^T z)^2.$$

We can also write this as

$$\begin{aligned}
K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\
&= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)
\end{aligned}$$

Thus, we see that $K(x,z) = \phi(x)^T\phi(z)$, where the feature mapping ϕ is given (shown here for the case of $n = 3$) by

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}.$$

Note that whereas calculating the high-dimensional $\phi(x)$ requires $O(n^2)$ time, finding $K(x,z)$ takes only $O(n)$ time—linear in the dimension of the input attributes.

For a related kernel, also consider

$$\begin{aligned}
K(x,z) &= (x^Tz + c)^2 \\
&= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) + \sum_{i=1}^{n} (\sqrt{2c}x_i)(\sqrt{2c}z_i) + c^2.
\end{aligned}$$

(Check this yourself.) This corresponds to the feature mapping (again shown for $n = 3$)

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ \sqrt{2c}x_3 \\ c \end{bmatrix},$$

and the parameter c controls the relative weighting between the x_i (first order) and the x_ix_j (second order) terms.

More broadly, the kernel $K(x, z) = (x^Tz + c)^d$ corresponds to a feature mapping to an $\binom{n+d}{d}$ feature space, corresponding of all monomials of the form $x_{i_1}x_{i_2}\dots x_{i_k}$ that are up to order d. However, despite working in this $O(n^d)$-dimensional space, computing $K(x, z)$ still takes only $O(n)$ time, and hence we never need to explicitly represent feature vectors in this very high dimensional feature space.

Now, lets talk about a slightly different view of kernels. Intuitively, (and there are things wrong with this intuition, but nevermind), if $\phi(x)$ and $\phi(z)$ are close together, then we might expect $K(x, z) = \phi(x)^T\phi(z)$ to be large. Conversely, if $\phi(x)$ and $\phi(z)$ are far apart—say nearly orthogonal to each other—then $K(x, z) = \phi(x)^T\phi(z)$ will be small. So, we can think of $K(x, z)$ as some measurement of how similar are $\phi(x)$ and $\phi(z)$, or of how similar are x and z.

Given this intuition, suppose that for some learning problem that you're working on, you've come up with some function $K(x, z)$ that you think might be a reasonable measure of how similar x and z are. For instance, perhaps you chose

$$K(x, z) = \exp\left(-\frac{||x - z||^2}{2\sigma^2}\right).$$

This is a resonable measure of x and z's similarity, and is close to 1 when x and z are close, and near 0 when x and z are far apart. Can we use this definition of K as the kernel in an SVM? In this particular example, the answer is yes. (This kernel is called the **Gaussian kernel**, and corresponds

5 Kernels. Properties. Mercer Theorem

kernels

$\exists\Phi : X \to \mathbf{H}$, $k : X \times X \to \mathbf{R}$

$k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$
$k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$

$\mathbf{H}$= feature space, Φ= map(feature) function

- for which k there exits Φ ?
- given k, if Φ exists, it may be not unique

kernel characterization (mercer 1)

data dependent - X finite

theorem if the Gram matrix $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive definite then k is a dot product : $\exists\Phi$ such that $k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$

proof K positive definite $\Rightarrow K = SDS^T$ (diagonalization)
where S is orthogonal and D is diagonal with non-negative entries
then $k(\mathbf{x}_i, \mathbf{x}_j) = (SDS^T)_{ij} = \langle S_i \cdot DS_j\rangle = \langle\sqrt{D}S_i \cdot \sqrt{D}S_j\rangle$
take $\Phi(\mathbf{x}_i) = \sqrt{D}S_i$

kernel characterization (converse) (mercer 2)

data dependent - X finite

theorem if the kernel k is a dot product $\exists\Phi$, $k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$
then the Gram matrix $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive definite
proof for any $\alpha \in \mathbf{R}^m$

$$\sum_{i,j=1}^{m} \alpha_i\alpha_j K_{ij} = \langle\sum_{i=1}^{m}\alpha_i\Phi(\mathbf{x}_i), \sum_{j=1}^{m}\alpha_j\Phi(\mathbf{x}_j)\rangle = \|\sum_{i=1}^{m}\alpha_i\Phi(\mathbf{x}_i)\|^2 \geq 0$$

so K is positive definite

mercer theorem (mercer 3)

theorem[Mercer] Let $\mathcal{X}$ be a compact subset of $\mathbf{R}^n$. Suppose $\mathcal{K}$ is a continuous symmetric function such that

$$\int_{\mathcal{X}}\int_{\mathcal{X}} \mathcal{K}(\mathbf{x},\mathbf{z})f(\mathbf{x})f(\mathbf{z})d\mathbf{x}d\mathbf{z} \geq 0$$

for all $f \in \mathcal{L}_2(\mathcal{X})$. Then, $\mathcal{K}(\mathbf{x},\mathbf{z})$ can be expanded in a uniformly convergent series

$$\mathcal{K}(\mathbf{x},\mathbf{z}) = \sum_{j=1}^{\infty}\lambda_j\phi_j(\mathbf{x})\phi_j(\mathbf{z})$$

to an infinite dimensional feature mapping ϕ.) But more broadly, given some function K, how can we tell if it's a valid kernel; i.e., can we tell if there is some feature mapping ϕ so that $K(x,z) = \phi(x)^T\phi(z)$ for all x, z?

Suppose for now that K is indeed a valid kernel corresponding to some feature mapping ϕ. Now, consider some finite set of m points (not necessarily the training set) $\{x^{(1)}, \ldots, x^{(m)}\}$, and let a square, m-by-m matrix K be defined so that its (i,j)-entry is given by $K_{ij} = K(x^{(i)}, x^{(j)})$. This matrix is called the **Kernel matrix**. Note that we've overloaded the notation and used K to denote both the kernel function $K(x,z)$ and the kernel matrix K, due to their obvious close relationship.

Now, if K is a valid Kernel, then $K_{ij} = K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T\phi(x^{(j)}) = \phi(x^{(j)})^T\phi(x^{(i)}) = K(x^{(j)}, x^{(i)}) = K_{ji}$, and hence K must be symmetric. Moreover, letting $\phi_k(x)$ denote the k-th coordinate of the vector $\phi(x)$, we find that for any vector z, we have

$$\begin{aligned}
z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\
&= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\
&= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\
&\geq 0.
\end{aligned}$$

The second-to-last step above used the same trick as you saw in Problem set 1 Q1. Since z was arbitrary, this shows that K is positive semi-definite ($K \geq 0$).

Hence, we've shown that if K is a valid kernel (i.e., if it corresponds to some feature mapping ϕ), then the corresponding Kernel matrix $K \in \mathbb{R}^{m\times m}$ is symmetric positive semidefinite. More generally, this turns out to be not only a necessary, but also a sufficient, condition for K to be a valid kernel (also called a Mercer kernel). The following result is due to Mercer.[5]

[5]Many texts present Mercer's theorem in a slightly more complicated form involving L^2 functions, but when the input attributes take values in $\mathbb{R}^n$, the version given here is equivalent.

Theorem (Mercer). Let $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x^{(1)}, \ldots, x^{(m)}\}$, $(m < \infty)$, the corresponding kernel matrix is symmetric positive semi-definite.

Given a function K, apart from trying to find a feature mapping ϕ that corresponds to it, this theorem therefore gives another way of testing if it is a valid kernel. You'll also have a chance to play with these ideas more in problem set 2.

In class, we also briefly talked about a couple of other examples of kernels. For instance, consider the digit recognition problem, in which given an image (16x16 pixels) of a handwritten digit (0-9), we have to figure out which digit it was. Using either a simple polynomial kernel $K(x, z) = (x^T z)^d$ or the Gaussian kernel, SVMs were able to obtain extremely good performance on this problem. This was particularly surprising since the input attributes x were just a 256-dimensional vector of the image pixel intensity values, and the system had no prior knowledge about vision, or even about which pixels are adjacent to which other ones. Another example that we briefly talked about in lecture was that if the objects x that we are trying to classify are strings (say, x is a list of amino acids, which strung together form a protein), then it seems hard to construct a reasonable, "small" set of features for most learning algorithms, especially if different strings have different lengths. However, consider letting $\phi(x)$ be a feature vector that counts the number of occurrences of each length-k substring in x. If we're considering strings of english alphabets, then there're 26^k such strings. Hence, $\phi(x)$ is a 26^k dimensional vector; even for moderate values of k, this is probably too big for us to efficiently work with. (e.g., $26^4 \approx 460000$.) However, using (dynamic programming-ish) string matching algorithms, it is possible to efficiently compute $K(x, z) = \phi(x)^T \phi(z)$, so that we can now implicitly work in this 26^k-dimensional feature space, but without ever explicitly computing feature vectors in this space.

digit recognit Kernel

The application of kernels to support vector machines should already be clear and so we won't dwell too much longer on it here. Keep in mind however that the idea of kernels has significantly broader applicability than SVMs. Specifically, if you have any learning algorithm that you can write in terms of only inner products $\langle x, z \rangle$ between input attribute vectors, then by replacing this with $K(x, z)$ where K is a kernel, you can "magically" allow your algorithm to work efficiently in the high dimensional feature space corresponding to K. For instance, this kernel trick can be applied with the perceptron to to derive a kernel perceptron algorithm. Many of the in terms of the eigenfunctions $\phi_j \in \mathcal{L}_2(\mathcal{X})$ of $(T_\mathcal{K} f)(\cdot) = \int_\mathcal{X} \mathcal{K}(\cdot, \mathbf{x}) f(\mathbf{x}) d\mathbf{x}$ normalized so that $||\phi_j||_{\mathcal{L}_2} = 1$ and positive associated eigenvalues $\lambda_j \geq 0$.

valid kernels

- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_1(\mathbf{x}, \mathbf{z}) + \mathcal{K}_2(\mathbf{x}, \mathbf{z})$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = a\mathcal{K}_1(\mathbf{x}, \mathbf{z})$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_1(\mathbf{x}, \mathbf{z})\mathcal{K}_2(\mathbf{x}, \mathbf{z})$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = \mathcal{K}_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$

$p(x)$ a polynomial with positive coefficients

- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = p(\mathcal{K}_1(\mathbf{x}, \mathbf{z}))$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = exp(\mathcal{K}_1(\mathbf{x}, \mathbf{z}))$
- $\mathcal{K}(\mathbf{x}, \mathbf{z}) = exp(-||\mathbf{x} - \mathbf{z}||^2/\sigma^2)$

dot product kernels

$k(\mathbf{x}, \mathbf{y}) = k(\langle \mathbf{x}, \mathbf{y} \rangle)$

theorem — Mercer

•the function k of the dot product kernel must satisfy
$k(t) \geq 0, k'(t) \geq 0$ and $k'(t) + tk''(t) \geq 0 \ \forall t \geq 0$
in order to be a positive definite kernel. that may still be insufficient

• if k s a power series expansion

$$k(t) = \sum_{n=0}^{\infty} a_n t^n$$

then k is a positive definite kernel iff $\forall n, a_n \geq 0$

$\Phi(x) = x_1^2, x_2^2, x_1x_2, x_2x_1$

$\langle \Phi(x) \cdot \Phi(y) \rangle = x_1^2y_1^2 + x_2^2y_2^2 + x_1x_2y_1y_2 + x_2x_1y_2y_1 = (x_1y_1 + x_2y_2)^2$

Th if $\Phi(x)$ = vector of all possibles d dim terms
generalization
(d = 2 in example)

then $K(x, y) = \langle \Phi(x) \cdot \Phi(y) \rangle = \langle x \cdot y \rangle^d$

6 Use of kernels with SVM

plug a kernel into SVM

after a long discussion on optimization theory...

the primal problem
minimize $\langle \mathbf{w} \cdot \mathbf{w} \rangle$
subject to $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \forall i$

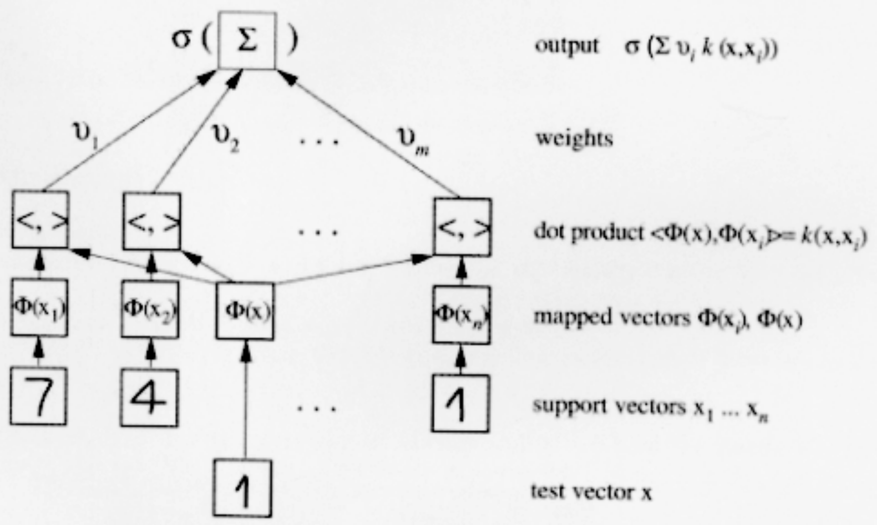
the dual problem
maximize $P(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$
subject to $\sum_{i=1}^{m} y_i \alpha_i = 0, \alpha_i \geq 0, \forall i$

kernel trick replace the dot product $\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$ with a kernel $k(\mathbf{x}_i, \mathbf{x}_j)$:
maximize

$$P(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$$

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

SVM with kernels


σ (Σ)
output σ (Σ υ_i k (x,x_i))
υ_1
υ_2
υ_m
weights
<, >
dot product <Φ(x),Φ(x_i)>= k(x,x_i)
Φ(x_1)
Φ(x_2)
Φ(x)
Φ(x_n)
mapped vectors Φ(x_i), Φ(x)
7
4
1
support vectors x_1 ... x_n
1
test vector x

the kernel trick

maximize

$$P(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$$

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

- we need only the kernel k ,**not** Φ - thats good...
- any algorithm that only depends on dot products (rotationally invariant) can be kernelized

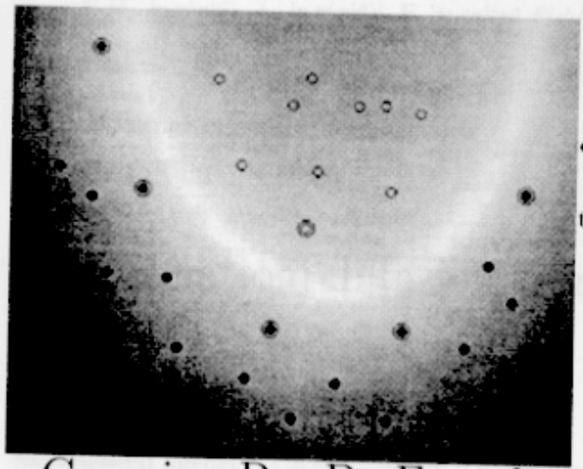
• any algorithm that is formulated in terms of positive definite kernel(s) supports a kernel-replace
• math was around for long time (1940s Kolgomorov, Aronszajn, Schoenberg) but the practical importance was underestimated

7 Kernel construction. Popular kernels

polynomial kernel

theorem define the map $\mathbf{x} \to C_d(\mathbf{x})$ where $C_d(\mathbf{x})$ the vector consisting in all possible d^{th} degree ordered products of the entries of $\mathbf{x}=(x_1, x_2, ..., x_N)$ then $\langle C_d(\mathbf{x}), C_d(\mathbf{y})\rangle = \langle \mathbf{x}, \mathbf{y}\rangle^d$

$$k(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y}\rangle + c)^d$$



polynomial kernel

• invariant to group of all orthogonal transformations(rotations, mirroring)

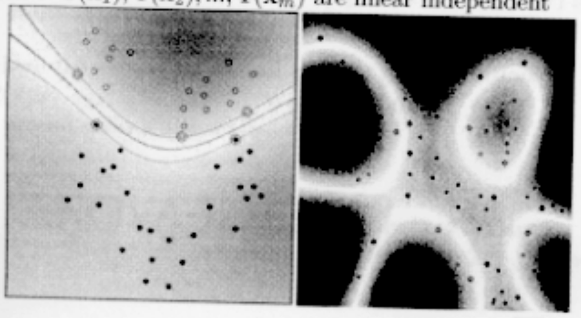
Gaussian $\text{R}_{\text{adial}}\text{B}_{\text{asis}}\text{F}_{\text{unction}}$ kernel

$k(\mathbf{x}, \mathbf{y}) = \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})$

more general $k(\mathbf{x}, \mathbf{y}) = f(d(\mathbf{x}, \mathbf{y}))$
where d is a metric on X and f is a function on $\mathbf{R}_0^+$; usually d arises from dot product $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$

• invariant on translations $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x} + \mathbf{z}, \mathbf{y} + \mathbf{z})$
• $\cos(\angle(\Phi(\mathbf{x}), \Phi(\mathbf{y}))) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y})\rangle = k(\mathbf{x}, \mathbf{y}) \geq 0 \Rightarrow$ enclosed angle between any 2 mapped points is smaller than $\pi/2$

theorem if $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$ all distinct and $\sigma > 0$ then the matrix $K_{ij} = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2})$ has full rank $\Rightarrow \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2), ..., \Phi(\mathbf{x}_m)$ are linear independent



Fisher kernel

- knowledge about objects in form of a generative probability model
- deals with mising/incomplete data, uncertainty, variable length

family of generative models (density functions)
$p(x|\theta)$, smoothly parametrized by $\theta = (\theta^1, ..., \theta^r)$; $l(x,\theta) = \ln p(x|\theta)$ log likelihood

score $V_\theta(x) := (\delta_{\theta^1} l(x,\theta), ..., \delta_{\theta^r} l(x,\theta)) = \nabla_\theta l(x,\theta) = \nabla_\theta \ln p(x|\theta)$
Fisher information matrix $I := \mathbf{E}_p[V_\theta(x) V_\theta(x)^T]$
$I_{ij} = \mathbf{E}_p[\delta_{\theta^i} \ln p(x|\theta) \cdot \delta_{\theta^j} \ln p(x|\theta)]$, $\mathbf{E}_p$ is called *Fisher information metric*

Fisher kernel
$K_I(x,y) := V_\theta(x)^T I^{-1} V_\theta(y)$
natural kernel M positive definite matrix
$K_M^{nat}(x,y) := V_\theta(x)^T M^{-1} V_\theta(y)$

[information] diffusion kernel

- local relationships

the exponential of a suared matrix H is
$e^{\beta H} = \lim_{n\to\infty} (1 + \frac{\beta H}{n})^n = I + \beta H + \frac{\beta^2}{2!} H^2 + \frac{\beta^3}{3!} H^3 + ...$

exponential kernel $K_\beta = e^{\beta H}$, $\frac{\delta K_\beta}{\delta \beta} = H K_\beta$(heat eq)

diffusion kernel on graph : consider
$H_{ij} = 1$ if $i\ j$; $-d_i$ (degree) if $i = j$; 0 otherwise
$w^T H w = -\sum_{i,j\in E} (w_i - w_j)^2$ negative semidefinite
$-H$=Laplacian of the graph

convolution kernel

kernel between composite objects building on similarities of resp. parts
$k_d : X_d \times X_d \to \mathbf{R}$, R-relation. define the R-convolution kernel

$$(k_1 \star k_2 \star ... \star k_D)(\mathbf{x}, \mathbf{y}) := \sum_R \prod_{d=1}^{D} k_d(x_d, y_d)$$

where the sum runs over all possible decompositions of $\mathbf{x} \to (x_1, x_2, ..., x_D)$ and of $\mathbf{y} \to (y_1, y_2, ..., y_D)$ s.t. $R(\mathbf{x}, x_1, x_2, ..., x_D)$ and $R(\mathbf{y}, y_1, y_2, ..., y_D)$
- proved valid if R is finite

ANOVA kernel (analysis of variance)

if $X = S^N$ and $k^{(i)}$ kernel on $S \times S$ for $i = 1, 2, ..., N$,the ANOVA kernel of order D is

$$k_D(\mathbf{x}, \mathbf{y}) := \sum_{1 \le i_1 < ... < i_D \le N} \prod_{d=1}^{D} k^{i_d}(x_{i_d}, y_{i_d})$$

string kernel

- similarities between two documents

$\sum$=alphabet, $\sum^n$=set of all strings of length n

for a given index sequence $\mathbf{i} = (1 \le i_1 < i_2 < ... < i_r \le |s|)$

define $s(\mathbf{i}) := s(i_1)s(i_2)....s(i_r)$ and $l_s(\mathbf{i}) = i_r - i_1 + 1 \ge r$

example $s = \texttt{fast food}$,$\mathbf{i} = (2,3,9) \Rightarrow s(\mathbf{i}) = \texttt{asd}, l_s(\mathbf{i}) = 9 - 2 + 1 = 8$

$0 < \lambda \le 1$ parameter, define $[\Phi_n(s)]$ a map with $|\sum^n|$ components

$$[\Phi_n(s)]_u = \sum_{\mathbf{i}:s(\mathbf{i})=u} \lambda^{l_s(\mathbf{i})}$$

example $[\Phi_3(\texttt{Nasdaq})]_{\texttt{asd}} = \lambda^3$, $[\Phi_3(\texttt{lass das})]_{\texttt{asd}} = 2\lambda^5$

the kernel induced

$$k_n(s,t) = \sum_{u \in \sum^n} [\Phi_n(s)]_u [\Phi_n(t)]_u = \sum_{u \in \sum^n} \sum_{(\mathbf{i},\mathbf{j}):s(\mathbf{i})=t(\mathbf{j})=u} \lambda^{l_s(\mathbf{i})} \lambda^{l_t(\mathbf{j})}$$

$k := \sum_n c_n k_n$ linear combination of kernels on different substring-lengths

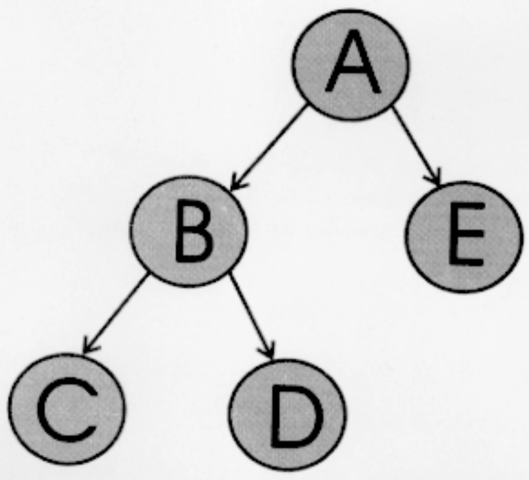
tree kernel

- encode a tree as a string by traversing in preorder and parenthesing

- substrings correspond to subset trees

- tag can be computed in loglinear time

- then use a string kernel



tag(T)=(A(B(C)(D))(E))