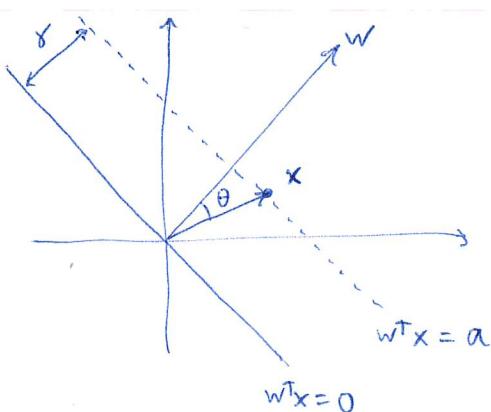
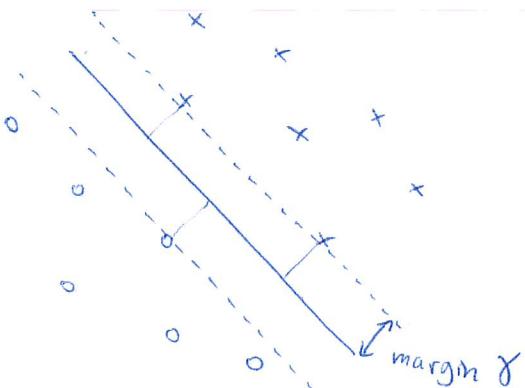


ECE 532 - lecture 23 - SVM and Kernels

①

a natural question: what is the linear classifier with "maximum margin"?



for dotted line:

$$\begin{cases} \gamma = \|x\| \cos \theta \\ w^T x = \|x\| \|w\| \cos \theta \end{cases}$$

$$\text{so } \gamma = \frac{w^T x}{\|w\|} = \frac{a}{\|w\|}$$

$$\left\{ \begin{array}{l} \text{maximize } \frac{a}{\|w\|} \quad (\text{maximize margin } \gamma) \\ \text{subject to } y_i w^T x_i \geq a \quad (\text{all points classified with margin } \gamma) \end{array} \right.$$

Note: we can scale a, w together. So we can let $a=1$ to simplify:

$$\boxed{\begin{aligned} & \max_w \frac{1}{\|w\|} \\ & \text{s.t. } y_i x_i^T w \geq 1 \quad \forall i \end{aligned}}$$

equivalent to

$$\boxed{\begin{aligned} & \underset{w}{\text{minimize}} \quad \|w\|^2 \\ & \text{s.t. } 1 - y_i x_i^T w \leq 0 \quad \forall i \end{aligned}}$$

If the points are perfectly separable, we can make w large enough so that $1 - y_i x_i^T w \leq 0$ for all i . Therefore, $(1 - y_i x_i^T w)_+ = 0$ for all i .

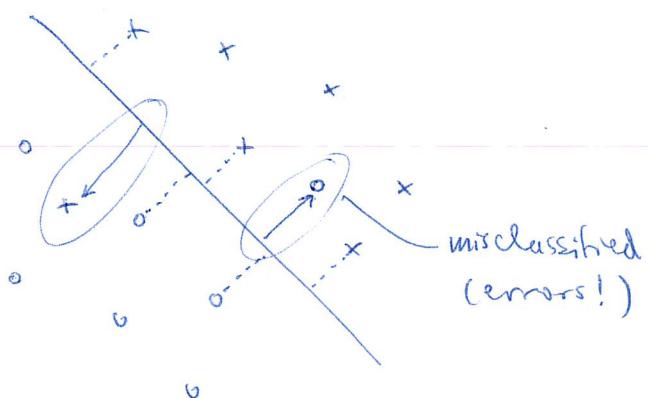
This is equivalent to solving the problem

$$\min_w \sum_{i=1}^m (1 - y_i x_i^T w)_+ + \lambda \|w\|^2 \quad \text{with very small } \lambda > 0,$$

i.e. L₂-regularized SVM! So using L₂ regularization recovers the max-margin separator when $\lambda \rightarrow 0$.

(2)

What if the points are not perfectly separable?



so instead of having $y_i x_i^T w \geq 1$ for all i , model it as $y_i x_i^T w \geq 1 - \varepsilon_i$ for all i , where $\varepsilon_i \geq 0$ is the error in point i .

The idea is to penalize $\|w\|_1$, i.e. encourage sparsity in the number of misclassified points.

$$\begin{aligned} & \text{minimize}_w \|w\|^2 \\ & \text{s.t. } 1 - y_i x_i^T w \leq 0 \quad \forall i \end{aligned}$$

(perfect classification)

$$\begin{aligned} & \text{minimize}_{w, \varepsilon} \|w\|^2 + C \sum_{i=1}^m \varepsilon_i \\ & \text{s.t. } y_i x_i^T w \geq 1 - \varepsilon_i \quad \forall i \\ & \text{and } \varepsilon_i \geq 0 \quad \forall i \end{aligned}$$

(sparse misclassifications)

re-arrange:

$$\begin{aligned} & \text{min}_{w, \varepsilon} \|w\|^2 + C \sum_{i=1}^m \varepsilon_i \\ & \text{s.t. } \varepsilon_i \geq 1 - y_i x_i^T w \\ & \varepsilon_i \geq 0 \end{aligned}$$

for any solution w , we should clearly choose

$$\varepsilon_i = \begin{cases} 1 - y_i x_i^T w & \text{if } 1 - y_i x_i^T w \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

i.e. $\varepsilon_i = (1 - y_i x_i^T w)_+$ (soft-threshold!)

$$\Rightarrow \min_w \|w\|^2 + C \sum_{i=1}^m (1 - y_i x_i^T w)_+$$

this is equivalent again to the L_2 -regularized SVM:

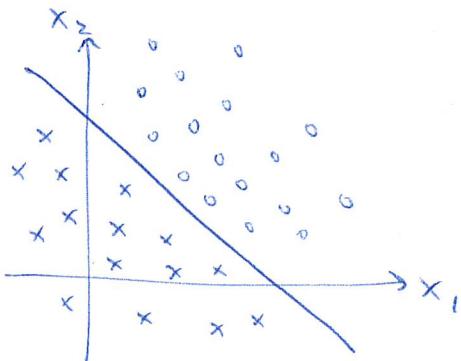
$$\min_w \sum_{i=1}^m (1 - y_i x_i^T w)_+ + \lambda \|w\|^2$$

and $\lambda \rightarrow 0$ corresponds to $C \rightarrow \infty$. (maximum possible penalty on misclassifications),

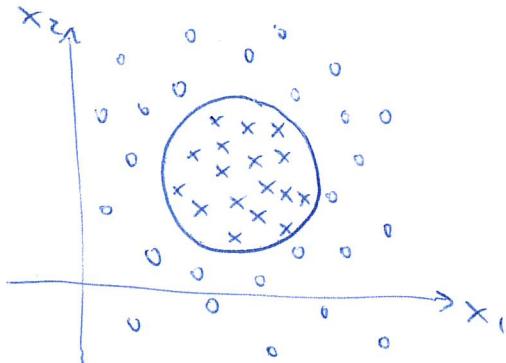
Kernel methods

Suppose two genes are involved in a disease process.

x_j is the expression level of gene $j=1, 2$.

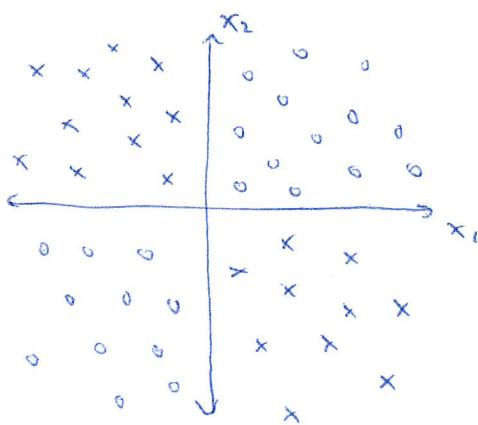


Linear separation



Nonlinear separation

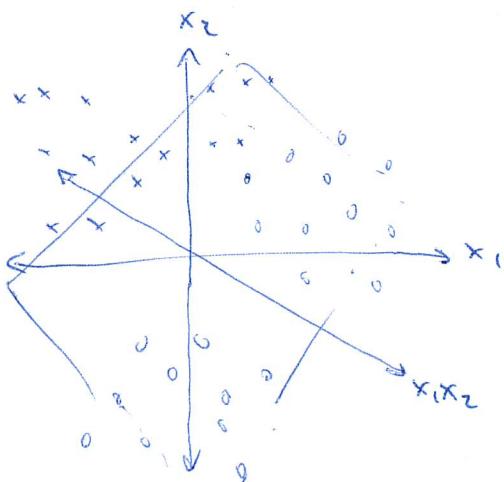
another example: disease occurs when both genes are "overexpressed" or when both genes are "underexpressed".



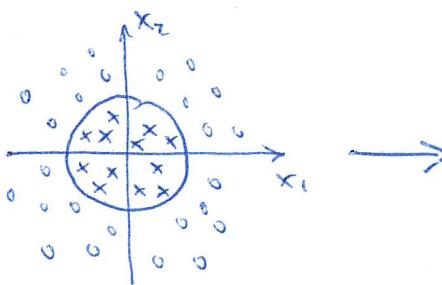
A linear separator will not work and cannot separate in the (x_1, x_2) space.

idea: add more features!

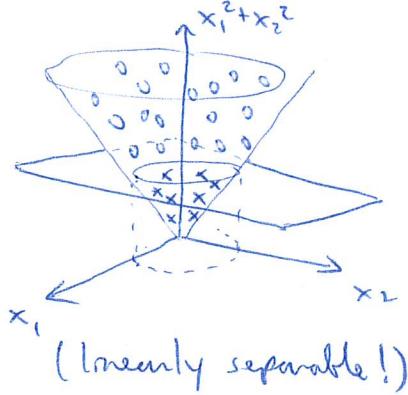
instead of using $[x_1 \ x_2]$, use $[x_1 \ x_2 \ x_1x_2]$.



now the $x_1 - x_2$ plane separates the points perfectly. Can do the same with circle:



(not separable)



(linearly separable!)

Original problem: example i has features (x_1, x_2) .

i.e. $x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix} \in \mathbb{R}^2$. Create matrix: $X = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ \vdots \\ x^{(m)\top} \end{bmatrix} \in \mathbb{R}^{m \times 2}$ (all examples), (4)

then solve $\min_{w \in \mathbb{R}^2} \|Xw - y\|^2$ (or use SVM loss).

Kernalized problem: use features to create new features:

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix} \in \mathbb{R}^2, \quad \phi(x^{(i)}) = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ x_1^{(i)}x_2^{(i)} \\ x_1^{(i)2} + x_2^{(i)2} \\ \vdots \end{bmatrix} \in \mathbb{R}^N.$$

Form an augmented matrix of data:

$$\Phi = \begin{bmatrix} \phi(x^{(1)})^\top \\ \phi(x^{(2)})^\top \\ \vdots \\ \phi(x^{(m)})^\top \end{bmatrix} \in \mathbb{R}^{m \times N}.$$

then solve $\min_{\bar{w} \in \mathbb{R}^N} \|\Phi \bar{w} - y\|^2$ (or use SVM loss).

★ Kernalized problems are often under determined ($N \geq m$) because of the combinatorial number of features introduced by the kernelization

e.g. if we have n features (x_1, x_2, \dots, x_n) and we want to create new features, say all cubics $(x_1^3, x_2^2x_5, x_1x_3x_4, \dots)$ we have to add n^3 more features!

So you almost always need to add some sort of regularization to the optimization problem,

(5)

take L_2 -regularized example:

$$\hat{w} = \underset{w}{\operatorname{arg\,min}} \|Xw - y\|^2 + \lambda \|w\|^2 \quad X \in \mathbb{R}^{m \times n}.$$

$$\left\{ \begin{array}{l} = (X^T X + \lambda I)^{-1} X^T y \quad (\text{invert an } n \times n \text{ matrix}) \\ = X^T (X X^T + \lambda I)^{-1} y \quad (\text{invert an } m \times m \text{ matrix}) \end{array} \right.$$

pick whichever is easiest.

in the kernelized method, we can choose between $m \times m$ and $N \times N$.

i.e. $\hat{w} = \Phi^T \underbrace{(\Phi \Phi^T + \lambda I)}^{-1} y$. is $m \times m$ since; does not depend on how many features we're using.

this is called the Kernel matrix $K = \Phi \Phi^T$.

the kernel matrix consists of inner products:

$$\Phi \Phi^T = \begin{bmatrix} \phi(x^{(1)})^T \\ \vdots \\ \phi(x^{(m)})^T \end{bmatrix} \begin{bmatrix} \phi(x^{(1)})^T \\ \vdots \\ \phi(x^{(m)})^T \end{bmatrix}^T. \quad \text{so } K_{ij} = \phi(x^{(i)})^T \phi(x^{(j)}).$$

★ it turns out we don't need to actually compute $\phi(x^{(i)})$ $\forall i$ in order to compute K_{ij} (this is called the "Kernel trick").

★ it turns out we don't need to actually compute \hat{w} either in order to make new predictions.

These two facts are critical when we have a very large number (∞) of features. We can actually use $N \rightarrow \infty$ in practice!

* if we have efficient way of computing $\phi(x^{(i)})^T \phi(x^{(j)})$ (6)
 then we can efficiently compute the kernel matrix K.

* to make a new prediction for some unlabeled \tilde{x} , then,

$$\hat{y} = \text{sign}(\phi(\tilde{x})^T \hat{w})$$

$$= \text{sign}(\phi(\tilde{x})^T \Phi^+ (K + \lambda I)^{-1} y)$$

$$= \text{sign}\left(\underbrace{\begin{bmatrix} \phi(\tilde{x})^T \phi(x^{(1)}) & \dots & \phi(\tilde{x})^T \phi(x^{(m)}) \end{bmatrix}}_{\text{can efficiently compute since we can presumably calc. inner products easily.}} \underbrace{(K + \lambda I)^{-1} y}_{\text{can efficiently compute since we know } K}\right)$$

can efficiently compute since we can presumably calc. inner products easily.
 can efficiently compute since we know K.

Conclusion: if we can compute inner products $\phi(x)^T \phi(y)$ efficiently, we can perform training and classification efficiently as well! so, we should look for features $\phi: \mathbb{R}^n \rightarrow \mathbb{R}^N$ that have easily computable inner products.

Can think of K_{ij} as a way of quantifying how close two data points $x^{(i)}$ and $x^{(j)}$ are in Kernel space, i.e.
 how close $\phi(x^{(i)})$ is to $\phi(x^{(j)})$.

(7)

Example quadratic kernel.

use $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \vdots \\ x_n^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_{n-1}x_n \end{bmatrix}$

this consists of all possible pairs of features (with extra coefficient to make it work),

$$\phi(x) \in \mathbb{R}^{n^2}$$

what is $\phi(x^{(i)})^T \phi(x^{(j)})$?

$$\begin{aligned} \phi(x^{(i)})^T \phi(x^{(j)}) &= \sum_{k=1}^n x_k^{(i)2} x_k^{(j)2} + \sum_{k=1}^n \sum_{l=1, l \neq k}^n 2x_k^{(i)} x_l^{(i)} x_k^{(j)} x_l^{(j)} \\ &= \left(\sum_{k=1}^n x_k^{(i)} x_k^{(j)} \right)^2 \\ &= (x^{(i)T} x^{(j)})^2 \quad (\text{wow!}), \end{aligned}$$

Therefore: $\phi(x) : \{\text{quadratics}\} \Rightarrow K_{ij} = (x^{(i)T} x^{(j)})^2$

also: $\phi(x) : \{\text{all polynomials of degree} \leq d\} \Rightarrow K_{ij} = (x^{(i)T} x^{(j)})^d$

and: "Gaussian Kernel" (∞ -dimensional) $\Rightarrow K_{ij} = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right)$