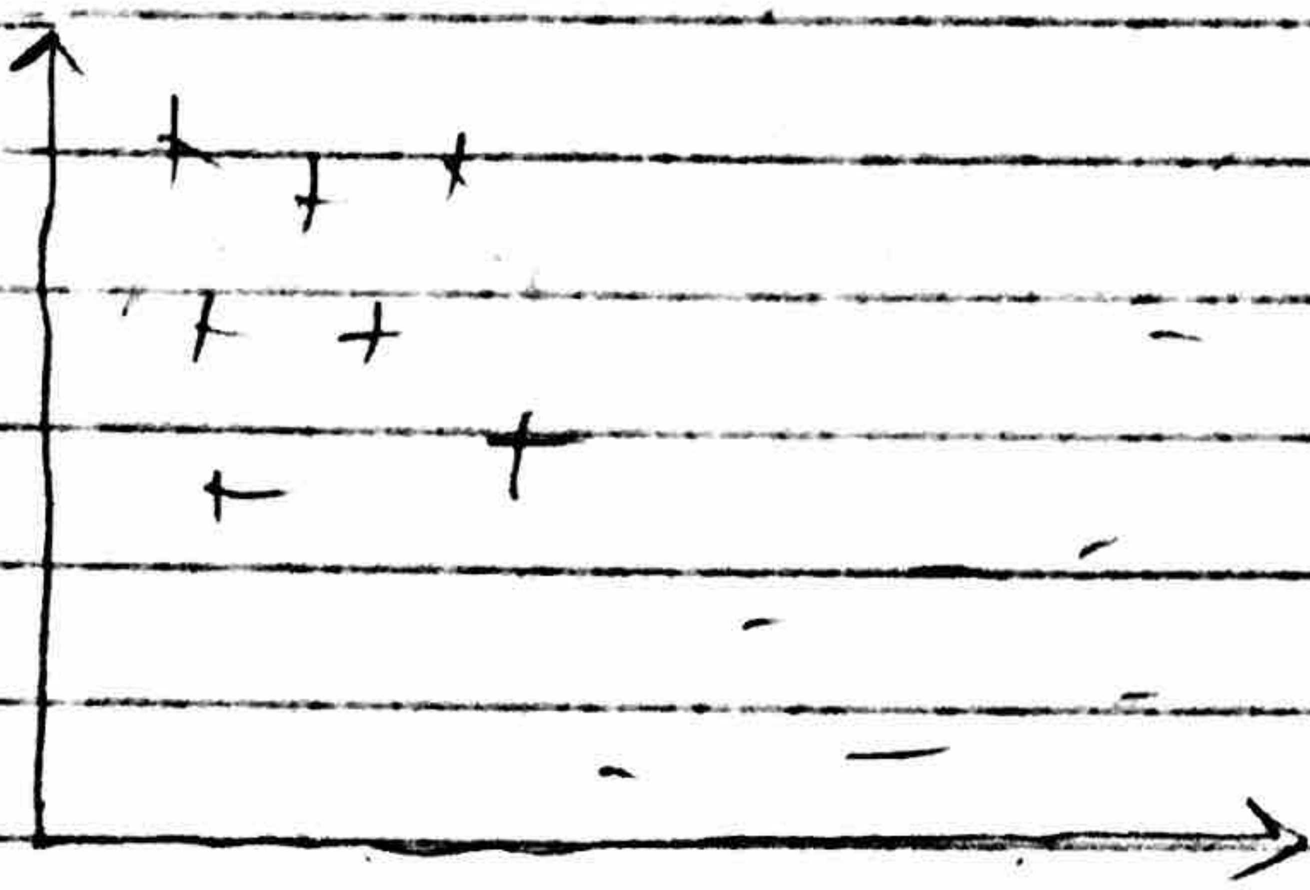
Lecture 4 – Non-Linear Classifiers and Kernels

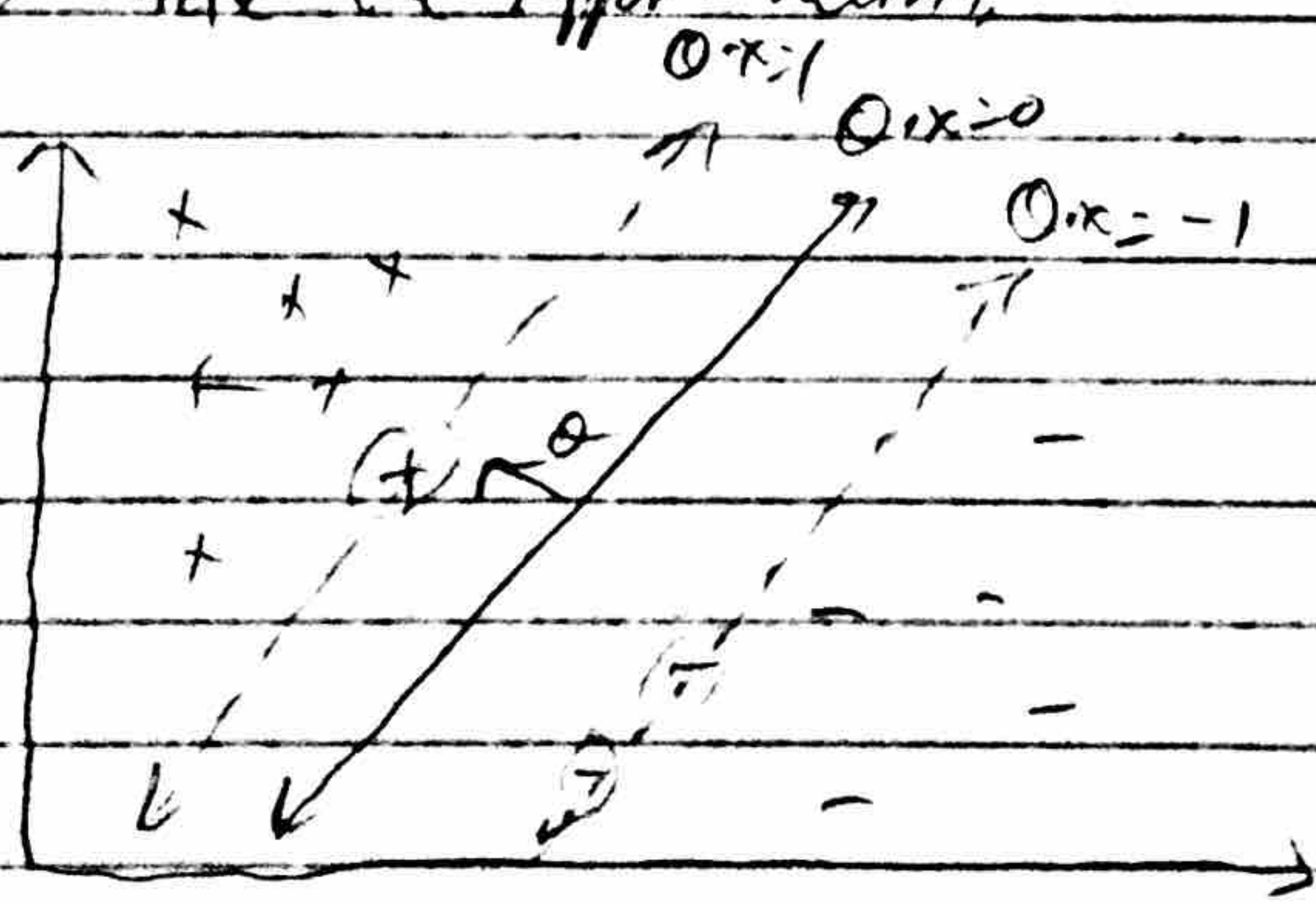
Review!

1)



a) Draw the decision boundary and margins for a maximum margin separator on the above data.

b) Circle the support vectors.



2) Let $\theta = \begin{pmatrix} 3 \\ 5 \\ -2 \end{pmatrix}$, $x = \begin{pmatrix} -2 \\ -3 \\ -1 \end{pmatrix}$, $y = +1$.

a) Compute the magnitude of θ, $\|\theta\|$.

$\|\theta\| = \sqrt{3^2 + 5^2 + (-2)^2} = \sqrt{9+25+4} = \sqrt{38}$

b) How far are the margins from the decision boundary?

$D = \frac{1}{\|\theta\|} = \frac{1}{\sqrt{38}}$

c) What is the hinge loss for x?

$Loss_h(y\theta \cdot x) = \max\left(0, 1 - 1 \cdot \begin{pmatrix} 3 \\ 5 \\ -2 \end{pmatrix} \cdot \begin{pmatrix} -2 \\ -3 \\ -1 \end{pmatrix}\right) = \max(0, 1 - (-6-15+2)) = 20$

(3) Let the objective function be $J(\theta) = \cos(-\theta) + \theta^7$

a) Compute the gradient of $J(\theta)$ with respect to θ, $\nabla_\theta J(\theta)$.

$$\nabla_\theta J(\theta) = \sin(-\theta) + 7\theta^6$$

b) What is the gradient descent update step for θ when $\eta = 7$

$$\theta = \theta - 7(\sin(-\theta) + 7\theta^6)$$

Today

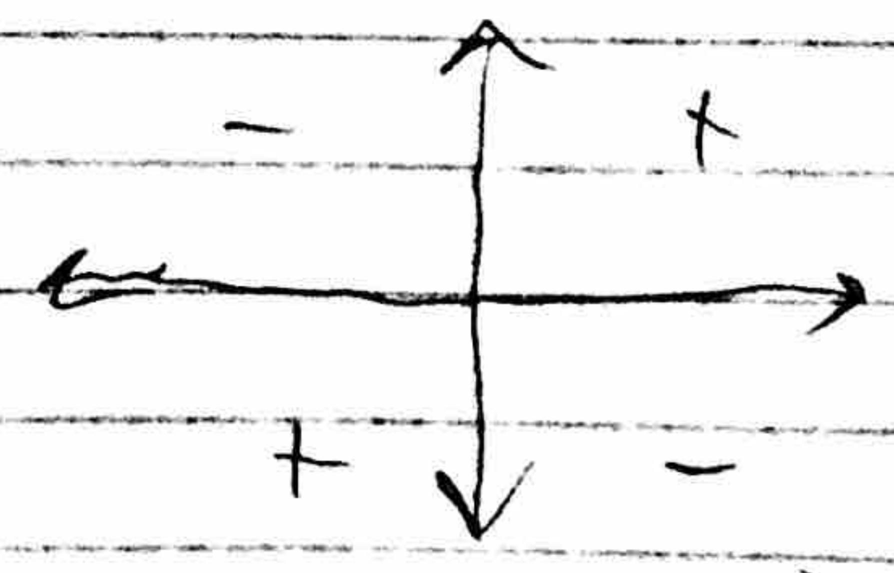
- why non-linearity?
- k nearest neighbors (KNN)
- Non-linear transformations
  - Examples - Drawbacks
- kernels
  - Motivation
  - Definition
  - Example - Properties - Radial basis kernel
- Kernel perceptron

Why non-linearity?

There are a lot of things that linear classifiers can't classify.

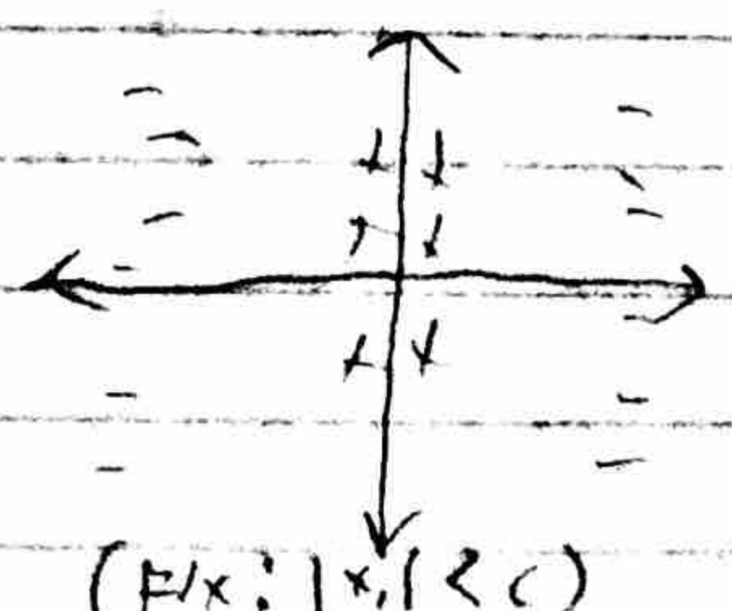
(?) Try to draw other examples of datasets which are not linearly separable.

Classic example: XOR



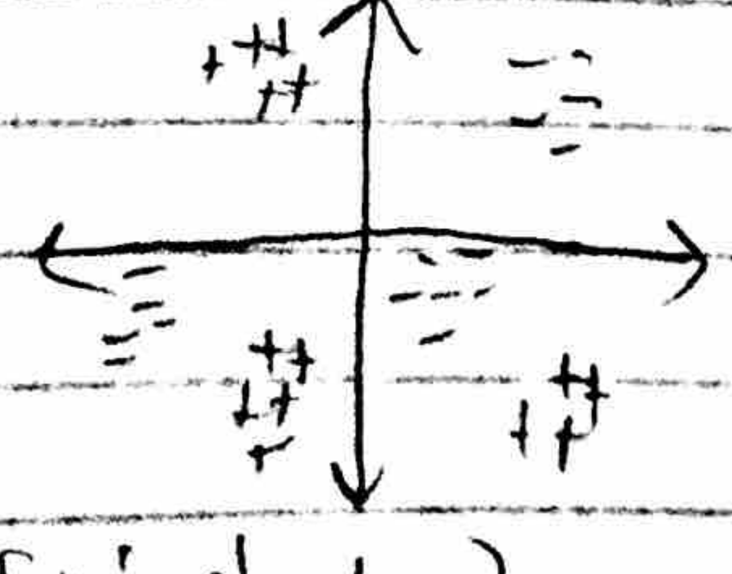
(Fix: $\text{sgn}(x_1 x_2) = 1$)

Other examples



(Fix: $|x_1| < c$)

(?) How could we classify these datasets?



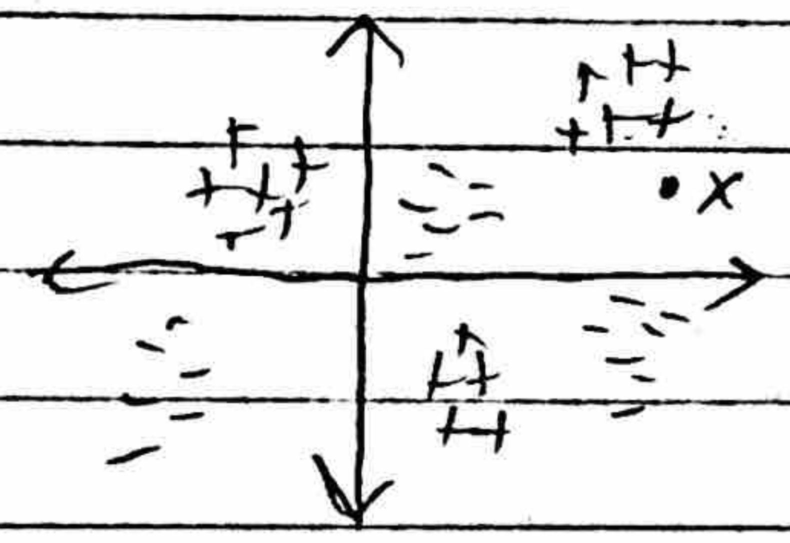
(Fix: clusters)

(Fix: $x_2 > c \cdot x_1^2$)

Two solutions:

1) Use a non-linear classifier with the given features.
   Example: KNN
2) Use a linear classifier with non-linear features.
   Example: kernel perceptron

k nearest neighbors (KNN)



? How should we predict whether x is + or −?

Idea: • Points with the same label usually have similar features and are therefore nearby in feature space.
• Assign x the same label as the points it is closest to.

Formally: Let $S_n = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$ = training set and let x be the point we're trying to classify.

minimum distance between vectors

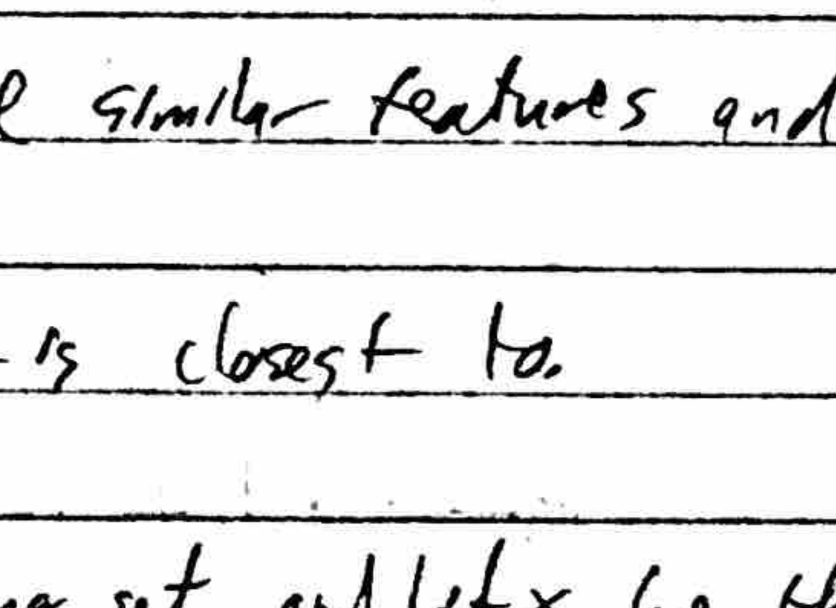
Define: $N_k(x; S_n) = \{k$ training examples in S_n which are closest to $x\}$

? Reminder: distance between d-dimensional vectors v and u = $d(v,u) = \sqrt{(v_1-u_1)^2 + \dots + (v_d-u_d)^2}$

KNN: $h_k(x; S_n) = \text{majority}(N_k(x; S_n))$

Essentially the KNN classifier chooses the majority label from the k nearest training examples.

? What does the decision boundary look like?
Actually multiple non-linear decision boundaries



Non-linear transformations

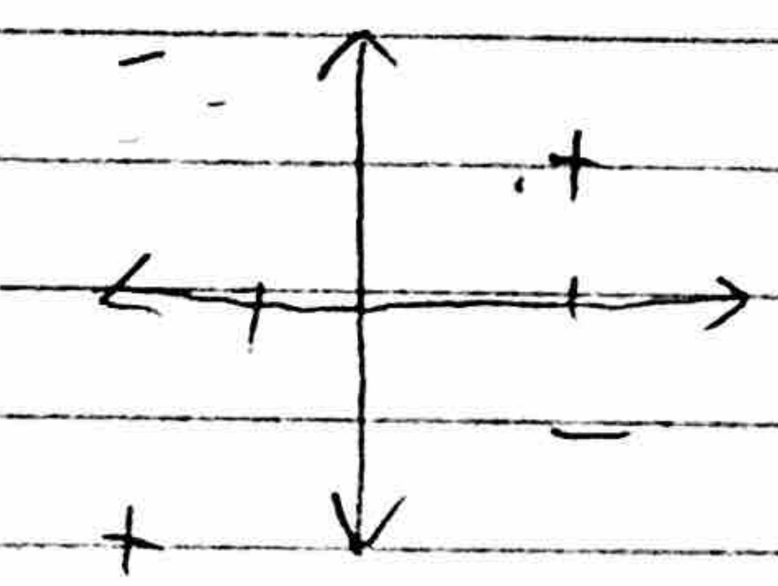
Rather than using a non-linear classifier, we can use a linear classifier with nonlinear features. This can be accomplished by performing a non-linear transformation on the given features.

Formally: Instead of classifying $x \in \mathbb{R}^d$, we classify $\phi(x) \in \mathbb{R}^{d'}$ where $\phi: \mathbb{R}^d \to \mathbb{R}^{d'}$

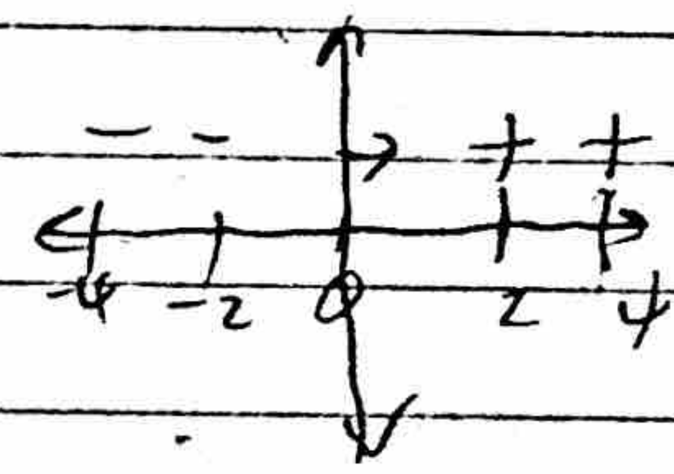
Note: d' is often larger than d but it doesn't have to be.

Examples

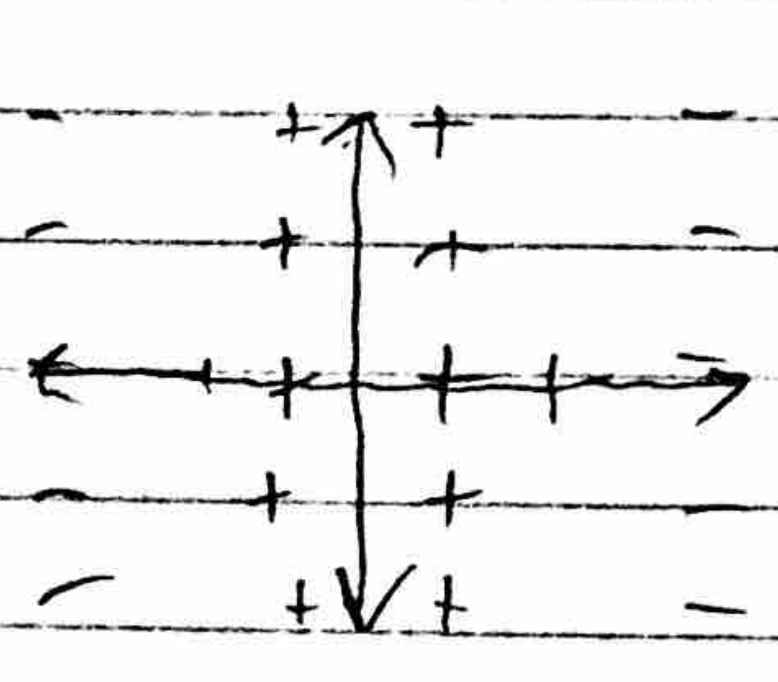
? (1)



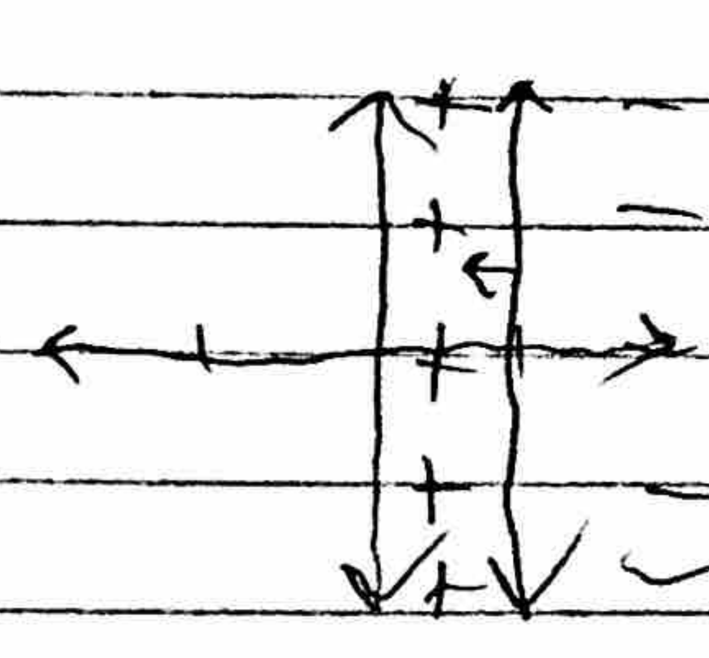
$$\phi: \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to \begin{bmatrix} x_1 \cdot x_2 \end{bmatrix}$$



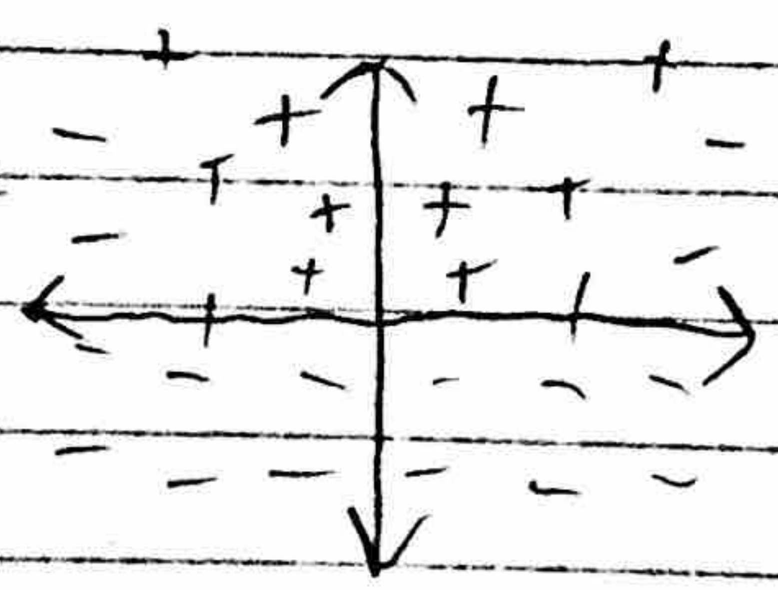
? (2)



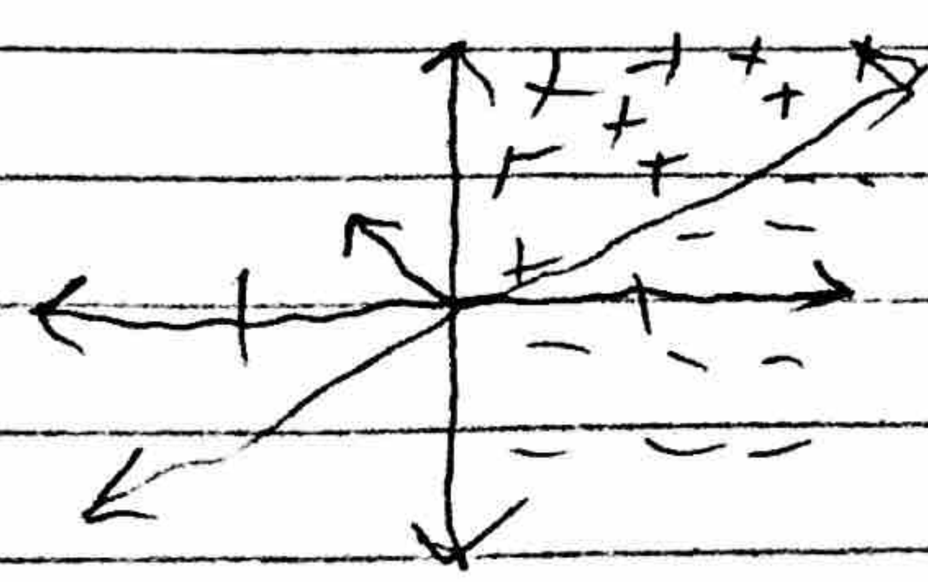
$$\phi: \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to \begin{bmatrix} |x_1| \\ x_2 \end{bmatrix}$$



? (3)



$$\phi: \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to \begin{bmatrix} x_1^2 \\ x_2 \end{bmatrix}$$



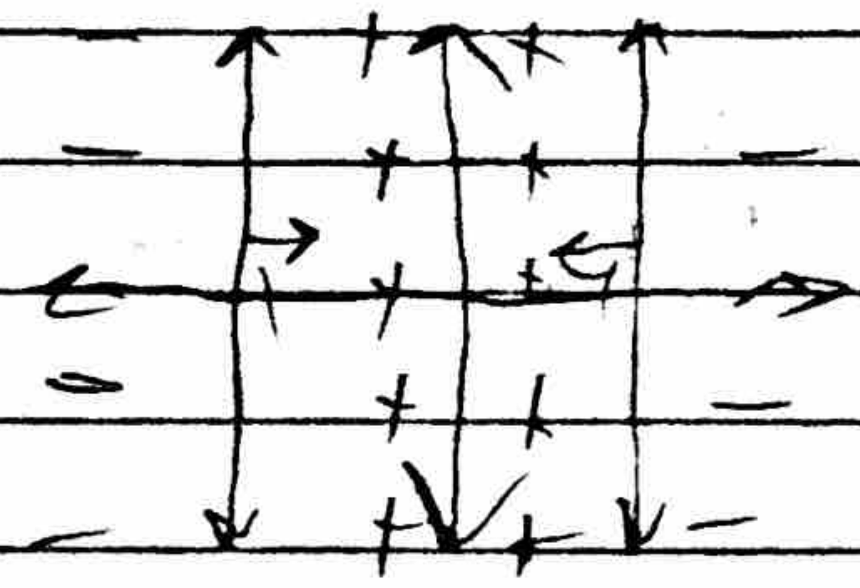
? These classifiers are linear in non-linear feature space. If we were to draw these classifiers in the original feature space, what would they look like?

$\longrightarrow$

1

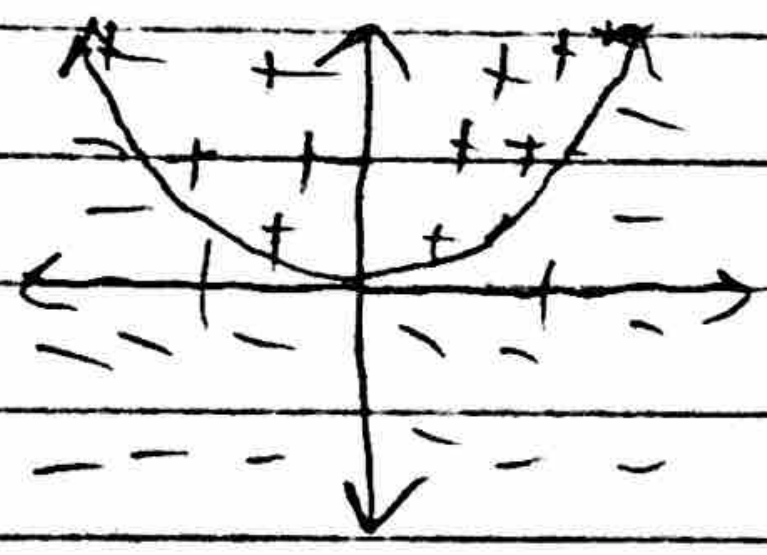
(The classifier is along the axes and classifies quadrants I and III as + and classifies II and IV as negative.)

2



(Note: A single linear classifier in non-linear feature space acts as two linear classifiers in the original feature space)

3



(Note: A linear classifier in non-linear feature space acts as a non-linear classifier in the original feature space.)

Conclusion: Non-linear data transformations allow us to use linear classifiers to make non-linear classifications.

Good news: Now we can classify a much larger variety of datasets.
Bad news: Non-linear data transformations can be computationally expensive, especially since the transformed data $\phi(x)$ often has many more dimensions than x.

Example: Perceptron computes $\theta \cdot x$ on every step. This dot product takes d multiplications and additions, where $x, \theta \in \mathbb{R}^d$. If we use $\phi(x) \in \mathbb{R}^{d'}$ where $d' \gg d$, then this will be significantly slower.

? Is there any way we can use nonlinear features in an efficient way?
Answer: kernel trick

Kernels

Problem: We want to compute $\theta \cdot \phi(x)$ but working with $\phi(x)$ is slow
Idea: Compute $\theta \cdot \phi(x)$ without ever computing $\phi(x)$.

(?) What does the idea even mean?
Our goal is to compute a function, $\theta \cdot \phi(x)$, in the transformed feature space. But maybe there exists a different function in the original feature space which always produces the same output as $\theta \cdot \phi(x)$ but uses fewer computations. This is exactly what a kernel is.

(?) Math notation: Definition: A kernel function is a function $k: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that $\exists \phi: \mathbb{R}^d \to \mathbb{R}^{d'}$ such that $\forall x, y \in \mathbb{R}^d$ we have $k(x,y) = \langle \phi(x), \phi(y) \rangle$

(Check understanding of math notation, esp. < >)

Essentially a kernel function is able to compute the dot product of transformed data without actually performing the transformation

Example

$$\phi: \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to \begin{bmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{bmatrix}$$

(?) $$\langle \phi(x), \phi(y) \rangle = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ \sqrt{2}\, y_1 y_2 \\ y_2^2 \end{pmatrix} = x_1^2 y_1^2 + 2 x_1 x_2 y_1 y_2 + x_2^2 y_2^2$$

(This takes 9 multiplications and 2 additions)

Trick: $= (x_1 y_1 + x_2 y_2)^2 = \langle x, y \rangle^2$
(This takes 3 multiplications and 1 addition)

Therefore, $k(x,y) = \langle x, y \rangle^2 = \langle \phi(x), \phi(y) \rangle$
(Note: By the above math we can see that k always computes the same function as $\langle \phi(x), \phi(y) \rangle$ but is more efficient about it by using the original data. Advantage of kernels.)

(It's not always easy to find a kernel for a φ but if you can then it significantly speeds up the computation. Therefore in practice it's often easier to choose a kernel function and then determine the φ for which it is a kernel.)

Properties

Many of the properties of kernel functions derive from the fact that kernels are computing dot products and dot products have certain properties.

(?) Properties of dot product

- $k(x,x) \geq 0$
- $k(x,y) = k(y,x)$
- $\alpha k(x,y)$ is a valid kernel function if $\alpha \geq 0$
- $k_1(x,y) + k_2(x,y)$ is a valid kernel function
- $k_1(x,y) \times k_2(x,y)$ is a valid kernel function

(Note: Above we assume k, k_1, k_2 are valid kernel functions.)

- $\langle x,y \rangle$ is a valid kernel function (by definition)
- $\langle x,y \rangle \cdot \langle x,y \rangle$ is a valid kernel function
- $\langle x,y \rangle^n$ is a valid kernel function for $n \geq 0$

The above properties tell us that we can construct a kernel representing any polynomial function of dot products.

poly kernel $k(x,x') = (x \cdot x' + 1)^d$

Ex. $k(x,y) = 5\langle x,y \rangle^2 - 3\langle x,y \rangle^5 + \langle x,y \rangle$

This allows us to find kernels for a wide variety of polynomial transformations.

(?) So far we've only been considering transformations φ which output a finite dimensional vector. Is it possible to use a φ which transforms the data to infinite dimensions? If so, is there a kernel for such a φ that wouldn't require infinite operations to compute?

Radial basis kernel: $k_r(x,y) = \langle \varphi(x), \varphi(y) \rangle$ where φ is infinite dimensional

4-8

$e^{-\frac{\|x-y\|^2}{2\sigma^2}}$ but σ is a free parameter so we can choose it to be 1.

$$\|x-y\|^2 = \sqrt{(x_1-y_1)^2+\dots}^{\,2} = (x_1-y_1)^2+\dots$$
$$= x_1^2 - 2x_1y_1 + y_1^2 + \dots$$
$$= \|x\|^2 - 2\langle x,y\rangle + \|y\|^2$$

Specifically, $k_r(x,y) = e^{-\frac{1}{2}\|x-y\|^2} = e^{-\frac{1}{2}(\|x\|^2 - 2\|x\|\|y\| + \|y\|^2)}$
$= e^{-\frac{1}{2}\|x\|^2} e^{\langle x,y\rangle} e^{-\frac{1}{2}\|y\|^2}$

(?) How is this infinite dimensional?
Taylor expansion for e^x
(Reminder: $f(x) = f(0) + f'(0)x + \frac{f^{(2)}(0)x^2}{2!} + \frac{f^{(3)}(0)x^3}{3!} + \dots$)

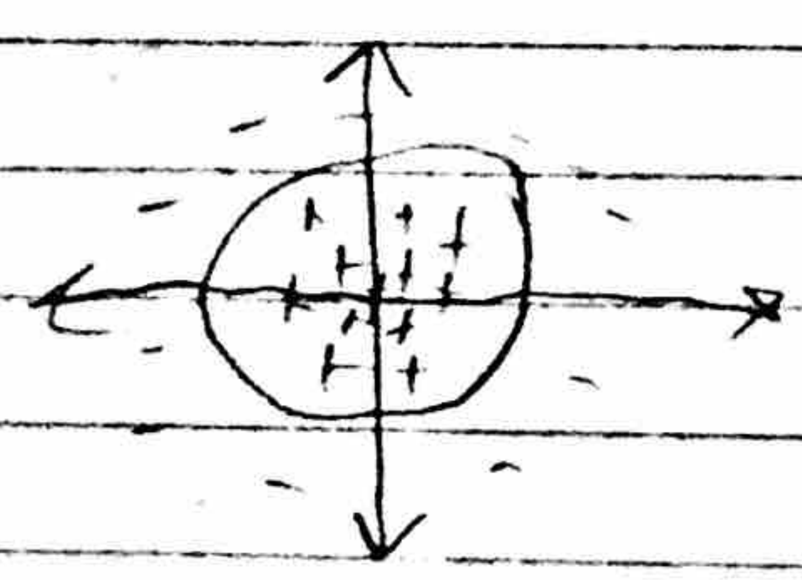
$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$

$e^{\langle x,y\rangle} = 1 + \langle x,y\rangle + \frac{\langle x,y\rangle^2}{2!} + \frac{\langle x,y\rangle^3}{3!} + \dots$

This is a valid kernel because it is a sum and product of valid kernels ($\langle x,y\rangle$ is valid as is 1).
It is infinite dimensional because it's an infinite sum of kernels.

(?) What's the benefit of an infinite dimensional kernel?
Ability to classify what no polynomial can classify.

Example



The radial basis kernel is named as such because it can classify data separated by radius, as in the above example.

Pause: Now that we understand kernels and how they enable efficient computation of dot products in nonlinear space, how do we actually use them to learn a classifier?

Furthermore the RBF takes on values between 0 and 1 and thus acts as a similarity measure. The RBF has a preference for smooth solutions and forms circular contours about points.

Kernel Perceptron

A variant of the perceptron algorithm which uses kernels to learn a linear classifier in nonlinear feature space.

Goal: We need to compute $\theta \cdot \phi(x) = \langle \theta, \phi(x) \rangle$

Kernel: Tells us $k(x,y) = \langle \phi(x), \phi(y) \rangle$

(?) How do we compute $\theta \cdot \phi(x)$?

Remember: when we make a mistake, our update step is $\theta = \theta + y\phi(x)$ (note that we are now working with nonlinear features $\phi(x)$).

Since we add each data point each time we make a mistake on that point we can see that θ is just a sum of data points multiplied by the number of mistakes

$$\theta = \alpha^{(1)} y^{(1)} \phi(x^{(1)}) + \alpha^{(2)} y^{(2)} \phi(x^{(2)}) + \dots + \alpha^{(n)} y^{(n)} \phi(x^{(n)})$$

where $\alpha^{(i)}$ = the number of mistakes made on the i^{th} data point

$$\begin{aligned} \text{Therefore, } \theta \cdot \phi(x) &= (\alpha^{(1)} y^{(1)} \phi(x^{(1)}) + \dots + \alpha^{(n)} y^{(n)} \phi(x^{(n)})) \cdot \phi(x) \\ &= \alpha^{(1)} y^{(1)} \phi(x^{(1)}) \cdot \phi(x) + \dots + \alpha^{(n)} y^{(n)} \phi(x^{(n)}) \cdot \phi(x) \\ &= \alpha^{(1)} y^{(1)} k(x^{(1)}, x) + \dots + \alpha^{(n)} y^{(n)} k(x^{(n)}, x) \end{aligned}$$

So now we can compute $\theta \cdot \phi(x)$ without ever explicitly computing θ or $\phi(x)$! This saves us a lot of time.

Now that we have all the tools we need, we can formalize the algorithm.

Kernel perceptron algorithm

Alternatively: $\text{sign}\left(\sum_{j=1}^{n} \alpha^{(j)} y^{(j)} k(x^{(j)}, x^{(i)})\right) \neq y^{(i)}$

$\alpha^{(1)}, \alpha^{(2)}, \dots, \alpha^{(n)} = 0$

repeat T times:

(Remember: agreement ≤ 0 is a mistake)

  for $i = 1, 2, \dots, n$:

    if $y^{(i)}(\theta \cdot \phi(x^{(i)})) = y^{(i)} (\alpha^{(1)} y^{(1)} k(x^{(1)}, x^{(i)}) + \dots + \alpha^{(n)} y^{(n)} k(x^{(n)}, x^{(i)})) \leq 0$:

      $\alpha^{(i)} = \alpha^{(i)} + 1$

return $\alpha^{(1)}, \alpha^{(2)}, \dots, \alpha^{(n)}$

(Note: we can use $\alpha^{(i)}$ to classify without computing θ)