Kernel methods and Graph kernels

Social and Technological Networks

Rik Sarkar

University of Edinburgh, 2018.

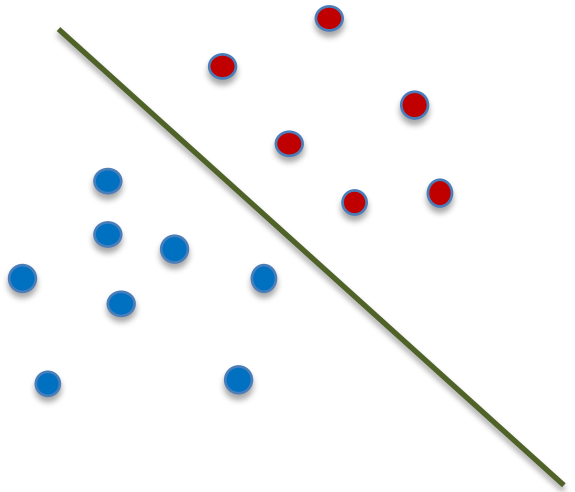
Kernels

- Kernels are a type of measures of similarity
- Important technique in Machine learning
- Used to increase power of many techniques

- Can be defined on graphs
- Used to compare, classify, cluster many small graphs
  - E.g. Molecules, neighborhoods of different people in social networks etc…

The main ML question

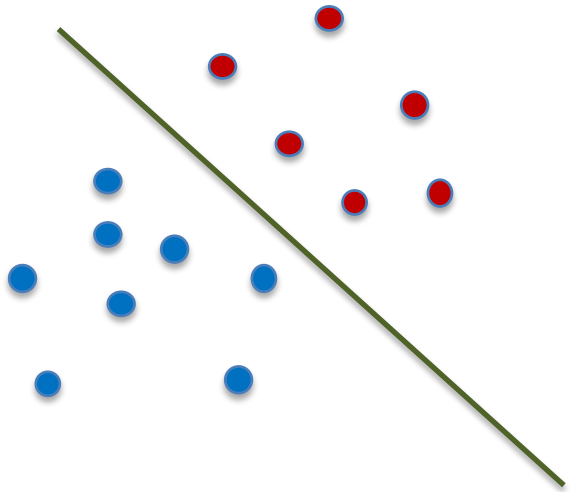
- For classes that can be separated by a line
  - ML is easy
  - E.g. Linear SVM, Single Neuron

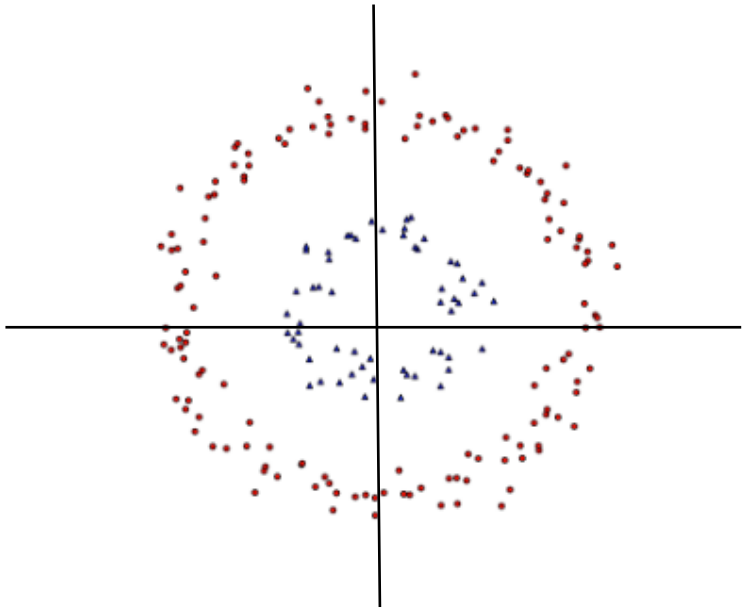


- But what if the separation is more complex?

The main ML question

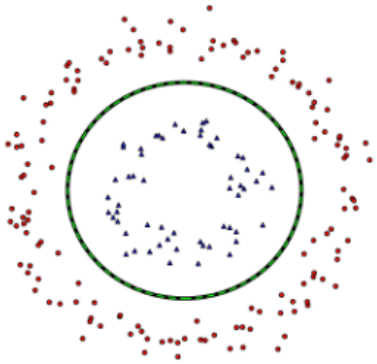
- For classes that can be separated by a line
  - ML is easy
  - E.g. Linear SVM, Single Neuron
- What if the structure is more complex?
  - Cannot separated linearly

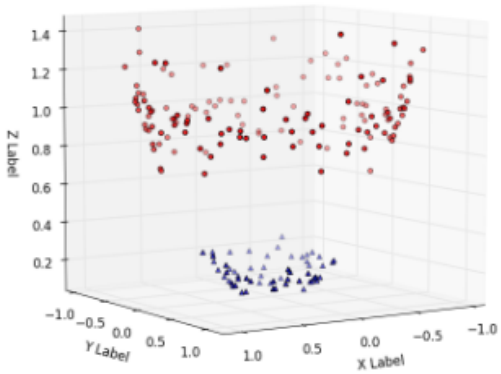




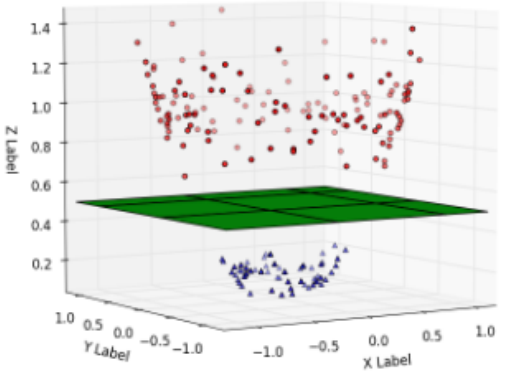
Lifting to higher dimensions

- Suppose we lift every (x,y) point to
- $(x, y) \rightarrow (x, y, \mathrm{x}^2 + \mathrm{y}^2)$:





- Now there is a linear separator!



Exercise

- Suppose we have the following data:

- How would you lift and classify?

- Assuming there is a mechanism to find linear separators if they exist

Kernels

- A similarity measure $K: X \times X \rightarrow \mathbb{R}$ is a kernel if:

- There is an embedding ψ (usually to higher dimension),
  - Such that: $\mathrm{K}(\boldsymbol{u}, \boldsymbol{v}) = \langle \psi(\boldsymbol{u}), \psi(\boldsymbol{v}) \rangle$
  - Where $\langle , \rangle$ represents inner product
  - Positive definite kernels

Example kernel

- For the examples we saw earlier, the following kernel helps:
- $K(u, v) = (u \cdot v)^2$

Example kernel

- For the examples we saw earlier, the following kernel helps:
- $K(u, v) = (u \cdot v)^2$
  - This is true with lifting map

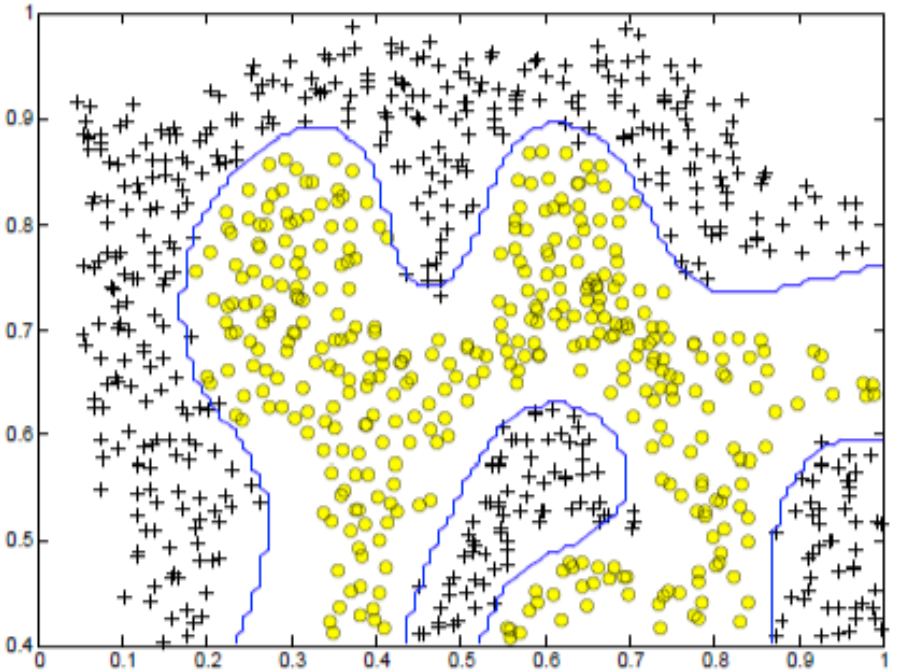
    $$\psi(u) = \left(u_x^2, \sqrt{2}\, u_x u_v, u_y^2\right)$$

  - Try it out!

More examples

- Polynnomial Kernel
- $K(u, v) = (1 + (u \cdot v))^k$

- Gaussian Kernel
- $K(u, v) = e^{-\frac{|u-v|^2}{2\sigma}}$
  - Sometimes called Radial Basis Function (RBF) kernel



Graph kernels

- To compute similarity between two attributed graphs
  - Nodes can carry labels
  - E.g. Elements (C, N, H etc) in complex molecules
- Idea: It is not obvious how to compare two graphs
  - Instead compute walks, cycles etc on the graph, and compare those

Walk counting

- Count the number of walks of length k from i to j
- Idea: i and j should be considered close if
  - They are not far in the shortest path distance
  - And there are many walks of short length between them (so they are highly connected)

- So, there would be many walks of length $\leq k$

Walk counting

- Can be computed by taking k^{th} power of adjacency matrix A
- If $A^k(i,j) = c$, that means there are c walks of length k between i and j
- Note: A^k is expensive, but manageable for small graphs

Common walk kernel

- Count how many walks are common between the two graphs

- That is, take all possible walks of length k on both graphs.
  - Count the number that are exactly the same
  - Two walks are same if the follow the same sequence of labels
    - (note that other than labels, there is no obvious correspondence between nodes)

Random walk kernel

- Perform multiple random walks of length k on both graphs
- Count the number of walks common to both graphs

Tottering

- Walks can move back and forth between adjacent vertices
  - Small structural similarities can produce a large score

- Usual technique: for a walk $v_1, v_2, \ldots$ prohibit return along an edge, ie $v_i = v_{i+2}$

Subtree kernel

- From each node, compute a neighborhood upto distance h

- From every pair of nodes in two graphs, compare the neighborhoods
  - And count the number of matches

Shortest path kernel

- Compute all pairs shortest paths in two graphs
- Compute the number of common sequences

- Tottering problem does not appear

- Problem: there can be many (exponentially many) shortest paths between two nodes
  - Computational problems
  - Can bias the similairity

Shortest distance kernel

- Instead use shortest distance between nodes
- Always unique

- Method:
  - Compute all shortest distances SD(G1) and SD(G2) in graphs G1 and G2
  - Define kernel (e.g. Gaussian kernel) over pairs of distances: $k(s_1, s_2)$, where $s_1 \in SD(G_1), s_2 \in SD(G_2)$
  - Define shortest path (SP)kernel between graphs as sum of kernel values over all pairs of distances between two graphs
    - $\mathrm{K}_{SP}(G_1, G_2) = \sum_{s_1} \sum_{s_2} k(s_1, s_2)$