Kernel methods and Graph kernels

Social and Technological Networks

Rik Sarkar

University of Edinburgh, 2019.
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...
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

• There are various types of kernels defined on graphs
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 $j$
  
  – Homework: Check this!

• Note: $A^k$ is expensive, but manageable for small graphs

• Kernel: compare $A^k$ for the two 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 they follow the same sequence of labels
    • (note that other than labels, there is no obvious correspondence between nodes)
Recap: dot product and cosine similarity

\[
\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{n \sum_{i=1}^{n} A_i^2} \sqrt{n \sum_{i=1}^{n} B_i^2}}
\]

Computation of A.B is the important element. Since \(|A| |B|\) is just normalization. A.B can be seen as the unnormalized similarity.
Common walk kernel as a dot product or cosine similarity

• For graphs $G_A$ and $G_B$
• Imagine vectors $A$ and $B$ representing all walks in graphs
  - Each position has a
    - Zero if that walk does not occur in the graph
    - One if the walk occurs in the graph
• Then $A.B = \text{number of common walks in the graph}$
Random walk kernel

• Perform multiple random walks of length k on both graphs
• Count the number of walks (label sequences) common to both graphs

• Check that this is analogous to a dot product

• Note that the vectors implied by the kernel do not need to be computed explicitly
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 prohibit $v_i = v_{i+2}$
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 (nodes in common)
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 similarity
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: 
    \[ k(s_1, s_2), \text{ where } s_1 \in SD(G_1), s_2 \in SD(G_2) \]
  – Define shortest path (SP) kernel between graphs as sum of kernel values over all pairs of distances between two graphs
    • \[ K_{SP}(G_1, G_2) = \sum_{s_1} \sum_{s_2} k(s_1, s_2) \]
Kernel based ML

• Kernels are powerful methods in machine learning
• We will briefly review general kernels and their use
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
Non linear separators

• Method 1:
  – Search within a class of non linear separators
  – E.g. Search over all possible circles, parabola etc.
  – higher degree polynomials allow more curved lines
Method 2: Lifting to higher dimensions

• Suppose we lift every \((x, y)\) point to

\[(x, y) \rightarrow (x, y, x^2 + y^2) :\]

• 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 (in any dimension) if they exist
Kernels

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

  • There is an embedding $\psi$ (usually to higher dimension),
    – Such that: $K(u, v) = \langle \psi(u), \psi(v) \rangle$
    – Where $\langle , \rangle$ represents inner product
      • Dot product is a type of inner product
Benefit of Kernels

• High dimensions have power to represent complex structures
  – We have seen in reference to complicated networks
• Lifting data to high dimensions can be used to separate complex structures that cannot be distinguished in low dimensions
  – But lifting to higher dimensions can be expensive (storage, computation)
  – Particularly when the data itself is already high dimensional

• Kernels define a similarity that is easy to compute
  – Equivalent to a high dimensional lift
  – Without having to compute the high-d representation

• Called the “Kernel trick”
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:

• \( K(u, v) = (u \cdot v)^2 \)
  – The implied lifting map is:
    \[ \psi(u) = (u_x^2, \sqrt{2} u_x u_v, u_y^2) \]
  – Try it out!
More examples

• General Polynomial 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
  – Extremely useful in practice when you do not have specific knowledge of data