

Fast prediction on a tree

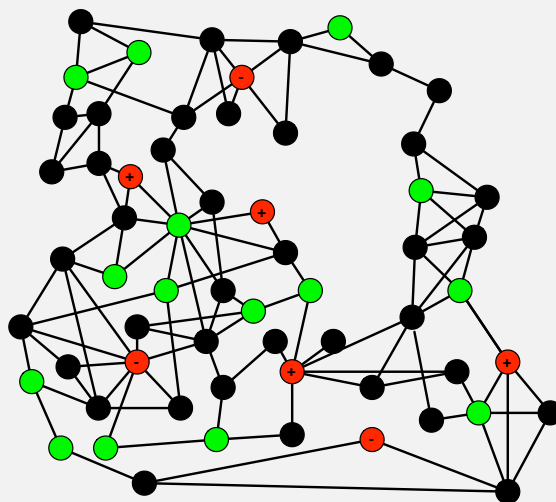
NIPS 2008

Mark Herbster (with M. Pontil and S. Rojas-Galeano)

University College London
Department of Computer Science

9 December, 2008

Learning model



Legend:

| | | |
|-----|---|--------------------|
| l | : | # labeled points |
| p | : | # test points |
| u | : | # unlabeled points |

- Predict only a small subset of points
- Expectation: $l + p \ll u$

Overview

- ▶ Aim: Speed up Laplacian-based semi-supervised learning
- ▶ Model: **large** graph with **small** label and test set
- ▶ Method:
 1. Approximate graph with a tree
 2. Compute Lap. kernel quickly via resistive network analogy
- ▶ Experiments:
 1. Approximate with MST and SPT trees (ensembles)
 2. Predict with kernel perceptron
 3. Web graph data 400,000 vertices 10 million+ edges

Preliminaries

Graph Laplacian $\mathbf{G} := \mathbf{D} - \mathbf{W}$

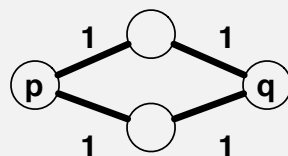
- \mathbf{W} is the weighted adjacency matrix
- $\mathbf{D} := \text{diag}(\sum_{i=1}^n W_{i1}, \dots, \sum_{i=1}^n W_{in})$ (vertex degree matrix)
- \mathbf{G}^+ is the “kernel” (pseudoinverse of \mathbf{G})

Theorem [KR93]

Effective resistance between v_p and v_q is

$$r_{\mathbf{G}}(p, q) = \mathbf{G}_{pp}^+ + \mathbf{G}_{qq}^+ - 2\mathbf{G}_{pq}^+$$

Graph is the circuit and edge (i, j) has resistance $\pi_{ij} := W_{ij}^{-1}$



- *Geodesic distance* is 2 from v_p to v_q
- **Effective resistance** is 1 from v_p to v_q

Graph connectivity

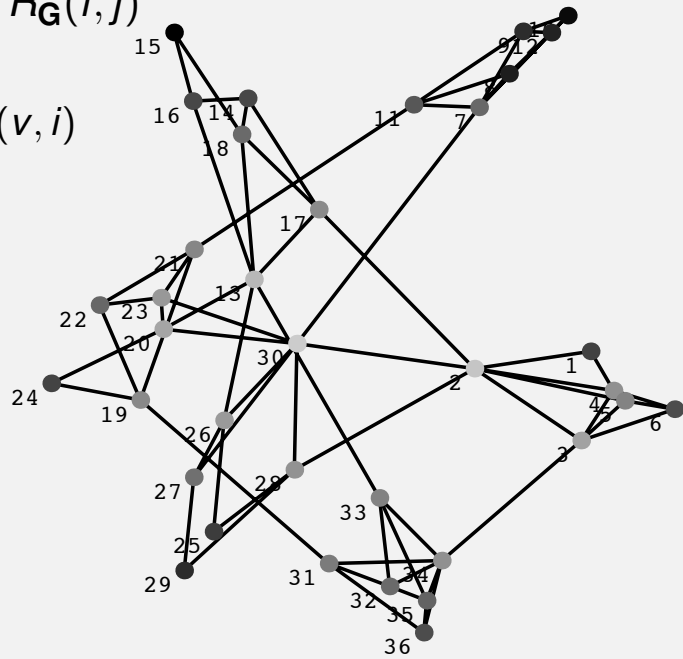
Inverse Connectivities

► Graph : $R_{\text{tot}} := \sum_{i>j} R_{\mathbf{G}}(i, j)$

► Vertex :

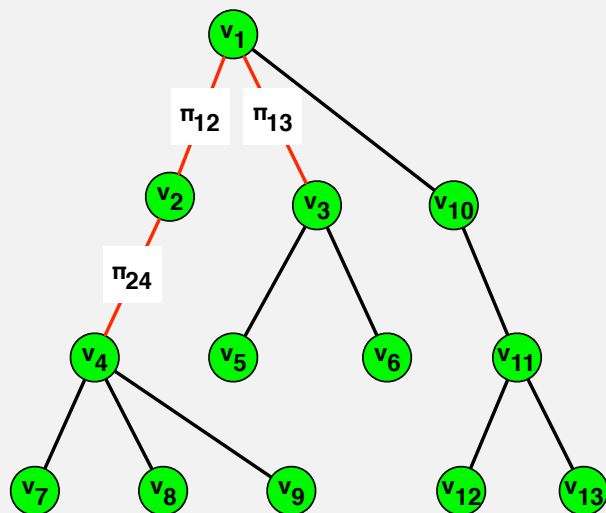
1. $R(v) := \sum_{i=1}^n R_{\mathbf{G}}(v, i)$

2. $\mathbf{G}_{vv}^+ = \frac{R(v)}{n} - \frac{R_{\text{tot}}}{n^2}$



Grey-scale \mathbf{G}_{vv}^+ : $\mathbf{G}_{30,30}^+ = .21$ (min), $\mathbf{G}_{15,15}^+ = .94$ (max)

Effective resistance on a tree



Effective resistance from v_3 to v_4

$$R_T(3, 4) = \pi_{13} + \pi_{12} + \pi_{24}$$

► “Resistors in series” : sum along unique path

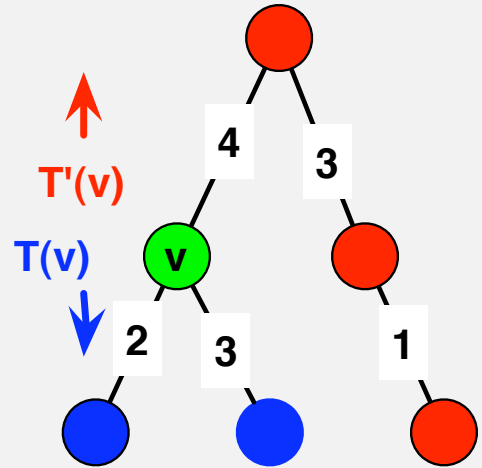
Sketch: computing $m \times m$ block of the kernel \mathbf{G}^+ (tree)

Computing the diagonal in $O(n)$ time

1. Compute $R(v)$, $v = 1, \dots, n$.
2. $R_{\text{tot}} = \frac{1}{2} \sum_{v=1}^n R(v)$
3. $\mathbf{G}_{vv}^+ = \frac{R(v)}{n} - \frac{R_{\text{tot}}}{n^2}$

Computing an $R(v)$

1. Define $T(v) = \sum_{x \in \text{descendants}(v)} r_{\mathbf{G}}(v, x)$
2. Define $T'(v) = \sum_{x \notin \text{descendants}(v)} r_{\mathbf{G}}(v, x)$
3. $R(v) = T(v) + T'(v)$



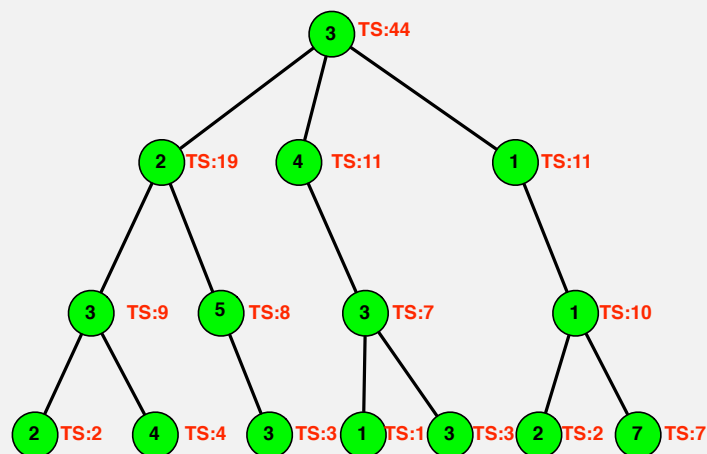
Computing tree-sums

- ▶ First consider the simpler problem of computing tree sums



$$\text{Tree-sum}(v) := \begin{cases} \text{val}(v) & v \text{ is a leaf} \\ \text{val}(v) + \sum_{x \in \text{children}(v)} \text{Tree-sum}(x) & v \text{ is not a leaf} \end{cases}$$

- ▶ Ascent from leaves gives all n tree sums in $O(n)$ time.

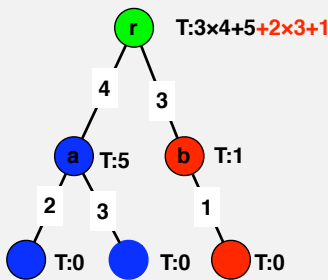


Tree Sum

Effective resistance tree sums

Recall $T(v) = \sum_{x \in \text{descendants}(v)} r_{\mathbf{G}}(v, x)$

$$T(v) := \begin{cases} \sum_{x \in \text{children}(v)} [(1 + |\text{descendants}(x)|)\pi_{vx} + T(x)] & v \text{ is not a leaf} \\ 0 & v \text{ is a leaf} \end{cases}$$



Effective Resistance Tree Sum

1. Recall $T'(v) = \sum_{x \notin \text{descendants}(v)} r_{\mathbf{G}}(v, x)$
2. Similar recursion gives T'
3. Now $R(v) = T(v) + T'(v)$;
4. Thus computing the diagonal of \mathbf{G}^+ in $O(n)$.

Computing $m \times m$ block of the kernel \mathbf{G}^+ (tree)

Computing the off-diagonal

Recall: $\mathbf{G}_{ij}^+ = \frac{1}{2}(\mathbf{G}_{ii}^+ + \mathbf{G}_{jj}^+ - r_{\mathbf{G}}(i, j))$

1. Single: \mathbf{G}_{ij}^+ in $O(S)$ (S is diameter) by two ascents to root.
2. Naive Block ($m \times m$): $O(m^2 S)$.
3. Amortized Block ($m \times m$): $O(m^2 + mS)$ saving $\min(m, S)$

Amortized algorithm for $m \times m$ block

$O(m^2 + mS)$ time

1. **Input:** $\{v_1, \dots, v_m\} \subseteq V$
2. **Initialization:** $\text{visited}(\text{all}) = \emptyset$
3. **for** $i = 1, \dots, m$ **do**
4. $p = -1$; $c = v_i$; $r_{\mathbf{T}}(c, c) = 0$
5. **Repeat**
6. **for** $w \in \text{visited}(c) \cap \overline{\{p\} \cup \downarrow^*(p)}$ **do**
7. $r_{\mathbf{T}}(v_i, w) = r_{\mathbf{T}}(w, v_i) = r_{\mathbf{T}}(v_i, c) + r_{\mathbf{T}}(c, w)$
8. **end**
9. $\text{visited}(c) = \text{visited}(c) \cup v_i$
10. $p = c$; $c = \uparrow(c)$
11. $r_{\mathbf{T}}(v_i, c) = r_{\mathbf{T}}(c, v_i) = r_{\mathbf{T}}(v_i, p) + \pi_{p,c}$
12. **until** (" p is the root")
13. **end**

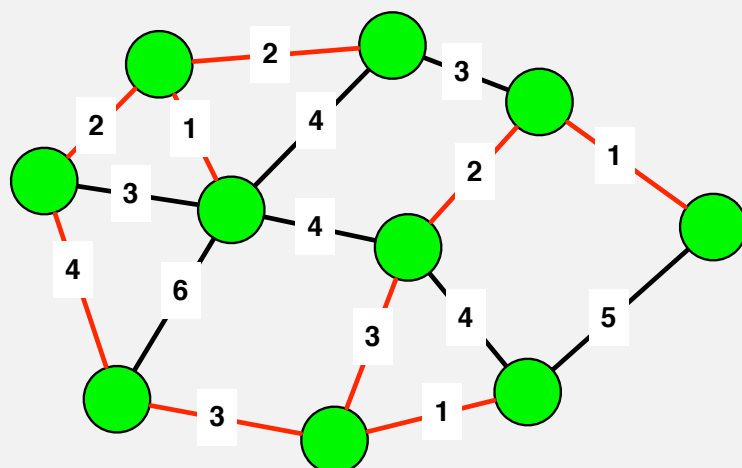
- "Initialization" + amortized algorithm: $O(n + m^2 + mS)$

Approximate the graph

- What if the graph is not a tree?
- Approximate with minimum or shortest paths spanning tree
- Use ensembles of trees

Minimum spanning tree

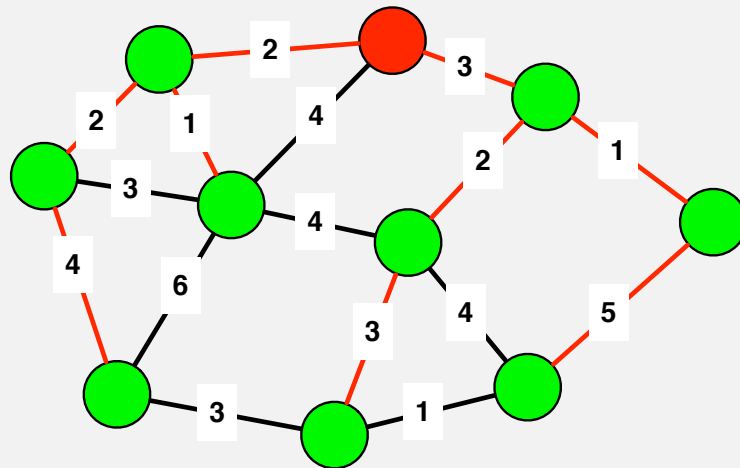
“Best approximation to \mathbf{G} ”



$$\begin{aligned} \text{MST}(\mathbf{G}) &:= \operatorname{argmin} \left\{ \sum_{(i,j) \in E(\mathbf{T})} \pi_{ij} : \mathbf{T} \in \mathcal{T}(\mathbf{G}) \right\} \\ &= \operatorname{argmin} \{ \operatorname{tr}(\mathbf{G} - \mathbf{T}) : \mathbf{T} \in \mathcal{T}(\mathbf{G}) \} \end{aligned}$$

Shortest paths tree

“Maximizes local connectivity about vertex i ”



$$\text{SPT}(\mathbf{G}, i) = \operatorname{argmin}\{R(i) : \mathbf{T} \in \mathcal{T}(\mathbf{G})\}$$

Tree construction requires

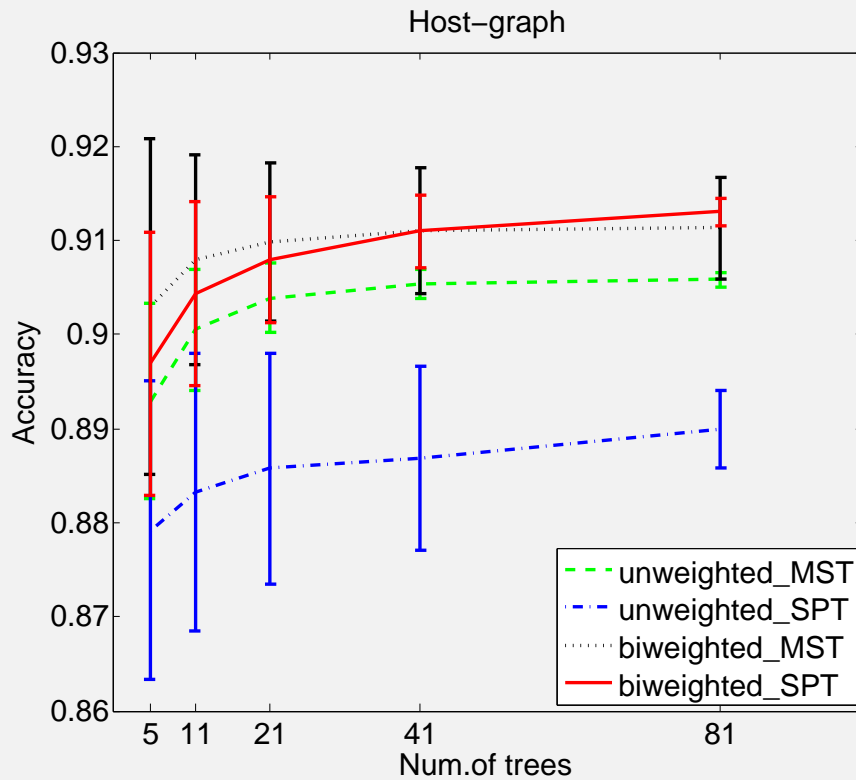
1. Unweighted : $O(E)$ time ($O(n)$ space)
2. Weighted : $O(E + n \log n)$ time ($O(n)$ space)

Results

| Method | Agg. (81/21) | AUC | Single | AUC |
|---|--------------|--------------|-------------------|-------------------|
| Host-graph (9,072 vertices, 464,959 edges; aggregates: 81) | | | | |
| MST | 0.907 | 0.950 | 0.857 ± 0.022 | 0.841 ± 0.045 |
| SPT | 0.889 | 0.952 | 0.850 ± 0.026 | 0.804 ± 0.063 |
| MST (bidir) | 0.912 | 0.944 | 0.878 ± 0.033 | 0.851 ± 0.100 |
| SPT (bidir) | 0.913 | 0.960 | 0.873 ± 0.028 | 0.846 ± 0.065 |
| Abernathy <i>et al.</i> | 0.896 | 0.952 | ... | ... |
| Tang <i>et al.</i> | 0.906 | 0.951 | ... | ... |
| Filoché <i>et al.</i> | 0.889 | 0.927 | ... | ... |
| Benczúr <i>et al.</i> | 0.829 | 0.877 | ... | ... |
| Web-graph (400,000 vertices, 10,455,544 edges; aggregates: 21) | | | | |
| MST (bidir) | 0.991 | 1.000 | 0.976 ± 0.011 | 0.993 ± 0.005 |
| SPT (bidir) | 0.994 | 0.999 | 0.985 ± 0.002 | 0.992 ± 0.003 |
| Witschel <i>et al.</i> | 0.995 | 0.998 | ... | ... |
| Filoché <i>et al.</i> | 0.973 | 0.991 | ... | ... |
| Benczúr <i>et al.</i> | 0.942 | 0.973 | ... | ... |
| Tang <i>et al.</i> | 0.296 | 0.989 | ... | ... |

- ▶ Classifier: kernel perceptron
- ▶ “bidir”: double-weight mutually linked edges

Aggregate performances



Accuracy versus number of trees

Recap

- ▶ Fast kernel method : $O(m^2 + mS + n)$ time
(m : “labeled” + “test”, S : diameter, n : total vertices)
- ▶ **Key insight : resistive network analogy**
- ▶ $m \ll n \implies$ time **linear** in unlabeled data
- ▶ If a graph is not a tree, we approximate it with MST or SPT
- ▶ Large scale experiments with Laplacian-based SSL
- ▶ Results competitive with methods which used the full graph

Thanks

