Min-Hashing for Probabilistic Frequent Subtree Feature Spaces

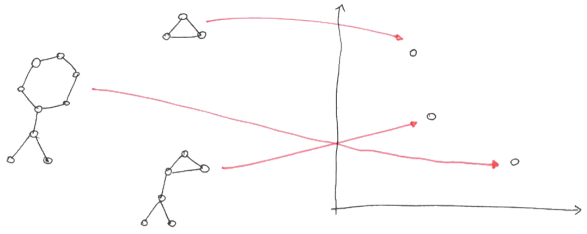
Pascal Welke, Tamás Horváth, and Stefan Wrobel

LWDA 2016

universitätbonn

Graph Kernels

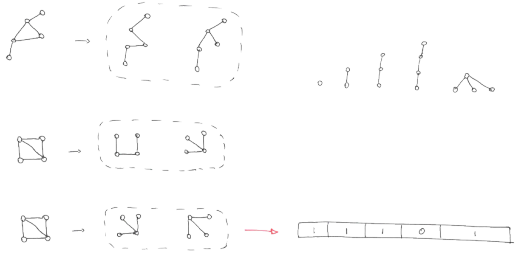
- Measure the similarity between graphs
- Enable us to learn models on graphs with generic learners
  - e.g. support vector machines, kernel PCA, etc.
- Expressive graph kernels usually suffer from severe computational complexity
  - most are NP-hard to compute



- We can learn a representation from a graph dataset
  - mine all frequent connected subgraphs
  - computationally intractable

- We can learn a representation from a graph dataset
  - mine all frequent connected subgraphs
  - computationally intractable
- And use this as the embedding function of our kernel
  - every graph can be represented as a binary vector
  - computing the embedding is NP-hard

- We simplify our problem by mining only frequent *subtrees*
  - thus far, mining and embedding are still intractable
- we reduce the graph to a set of some sampled spanning trees
  - introduces one sided error
  - if a tree is found, it is frequent
  - some frequent trees might not be found



Computing the Probabilistic Subtree Kernel Embedding

- We solved the mining part
- How do we embed (new) graphs into the corresponding feature space?

Computing the Probabilistic Subtree Kernel Embedding

- We solved the mining part
- How do we embed (new) graphs into the corresponding feature space?
- *Given* a graph, G, a parameter k, and a set of tree patterns $\mathcal{F}$:
  - Initialize an all-zero vector x of length $|\mathcal{F}|$
  - Sample k spanning trees of G
  - For each $T \in \mathcal{F}$
    - If T is a subgraph of one of these spanning trees of G
    - Then set the corresponding entry of x to one
  - Return x

Computing the Probabilistic Subtree Kernel Embedding

- We solved the mining part
- How do we embed (new) graphs into the corresponding feature space?
- *Given* a graph,G, a parameter k, and a set of tree patterns $\mathcal{F}$:
  - Initialize an all-zero vector x of length $|\mathcal{F}|$
  - Sample k spanning trees of G
  - For each $T \in \mathcal{F}$
    - If T is a subgraph of one of these spanning trees of G
    - Then set the corresponding entry of x to one
  - Return x
- Wait! Do we need to do this explicitly?

Computing the Probabilistic Subtree Kernel Embedding

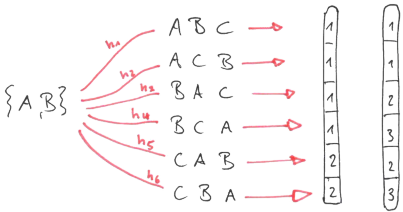
- We solved the mining part
- How do we embed (new) graphs into the corresponding feature space?
- *Given* a graph, G, a parameter k, and a set of tree patterns $\mathcal{F}$:
  - Initialize an all-zero vector x of length $|\mathcal{F}|$
  - Sample k spanning trees of G
  - For each $T \in \mathcal{F}$
    - If T is a subgraph of one of these spanning trees of G
    - Then set the corresponding entry of x to one
  - Return x
- Wait! Do we need to do this explicitly?

Jaccard Similarity and Min-Hashing

- The Jaccard similarity can be approximated via Min-Hashing

$$Jacc(A,B) = \frac{A \cap B}{A \cup B} = Prob_{h \in H}(h(A) = h(B))$$

  - Each $h \in H$ corresponds to a permutation of the feature set $\mathcal{F}$
  - It returns the smallest element in the set w.r.t. the permutation



universitätbonn

Properties of Min-Hashing

- Advantages:
  - saves space by using relatively small sketch vectors
  - similarities can be computed fast once sketches are available
  - is a kernel

Properties of Min-Hashing

- Advantages:
  - saves space by using relatively small sketch vectors
  - similarities can be computed fast once sketches are available
  - is a kernel
- Open Questions:
  - How can we compute the sketch vectors intelligently?

Computing Min-Hash Sketches Fast

- If a subgraph of a pattern does not appear in a graph, then the pattern itself cannot appear

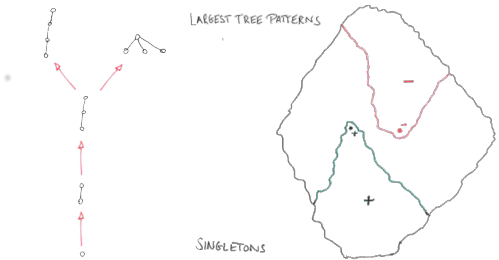
$\Leftrightarrow$

- If a pattern appears in a graph, all its subgraphs must appear

- If a subgraph of a pattern does not appear in a graph, then the pattern itself cannot appear

$\Leftrightarrow$

- If a pattern appears in a graph, all its subgraphs must appear

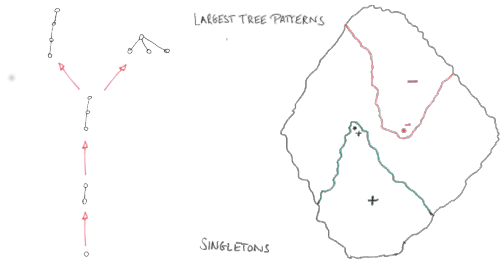


Computing Min-Hash Sketches Fast

- If a subgraph of a pattern does not appear in a graph, then the pattern itself cannot appear

$\Leftrightarrow$

- If a pattern appears in a graph, all its subgraphs must appear



$\Rightarrow$ When computing the embedding of a graph, we do not need to test all patterns for subgraph isomorphism

Implementation

Input: graph G, directed graph $F = (\mathcal{F}, E)$ representing a poset $(\mathcal{F}, \preccurlyeq)$ and K permutations $\sigma_1, \ldots, \sigma_K$ of $\mathcal{F}$
Output: *sketch*(G)

```
init sketch := [⊥, ..., ⊥]
init state(T) := 0 for all T ∈ F
for i = 1 to |F| do
    for j = 1 to K do
        if |σ_j| ≥ i ∧ sketch[j] = ⊥ then
            if state[σ_j[i]] ≠ 0 then
                if state[σ_j[i]] = 1 then sketch[j] = σ_j[i]
            else if σ_j[i] ≼ G then
                sketch[j] = σ_j[i]
                for all T' ∈ F (including T) that can reach T in F do
                    set state(T') := 1
            else
                for all T' ∈ F (including T) that are reachable from T in F do
                    set state(T') := −1
return sketch
```

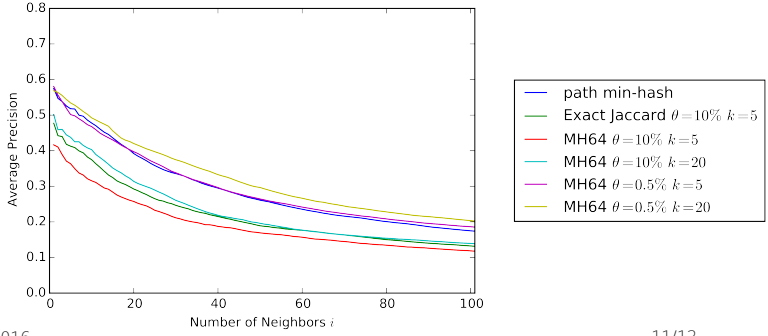
Does it work?

Dataset	k	θ	$size(\mathcal{F})$	naive	MH32	MH64	MH128	MH256
MUTAG	5	10%	452	206.38	49.93	68.24	96.12	127.42
MUTAG	10	10%	543	244.11	42.77	63.77	90.57	125.39
MUTAG	15	10%	562	254.86	45.39	65.96	94.87	133.91
MUTAG	20	10%	573	260.18	55.34	76.32	105.15	135.11
PTC	5	10%	1,430	321.04	70.07	102.62	121.12	156.12
PTC	5	1%	9,619	734.79	236.31	327.27	475.35	611.92
PTC	10	10%	1,566	354.20	79.63	108.59	109.44	147.91
PTC	20	10%	1,712	376.65	17.60	25.81	31.49	39.62
DD	5	10%	8,111	3,547.22	260.47	486.09	846.09	1,374.76
DD	10	10%	18,137	6,670.93	317.82	568.23	1,072.58	1,936.42
DD	20	10%	33,100	11,005.49	344.59	653.66	1,242.03	2,190.15
NCI1	5	10%	1,819	431.19	89.12	137.75	185.22	221.21
NCI1	5	1%	21,306	900.68	615.62	920.17	1,227.52	1,378.18
NCI1	20	10%	2,441	557.70	115.07	183.54	220.14	255.58
NCI109	5	10%	2,182	462.62	115.62	170.43	206.23	254.70
NCI109	5	1%	19,099	886.06	532.38	727.15	1057.18	1,348.27
NCI109	20	10%	2,907	598.36	110.42	175.76	226.07	284.92

Table: Average number of subtree isomorphism test per graph for several datasets with varying number k of sampled spanning trees and frequency thresholds θ. The table reports $size(\mathcal{F})$ and the average number of subtree isomorphism tests evaluated by the naive method and by our algorithm for K =32,64,128,256 (last four columns).

Active Molecule Retrieval on NCI-HIV

- On a highly imbalanced dataset, we want to retrieve examples of the smaller class
- We are given a positive example as query



Conclusion

- We presented an approximation of an approximation of a graph kernel
- It’s tractable for arbitrary graphs
- It’s faster to compute than the approximation
- The computed sketched embeddings use much less memory than the approximated embeddings
- Works well for some problems in chemoinformatics