Introduction to Spark & Locality Sensitive Hashing

Octavian Suciu
osuciu@umiacs.umd.edu
Spark

- Framework for processing large amounts of data
- Driver & Workers
- Resilient Distributed Datasets - RDDs
- All workers execute the same task
Spark architecture

• Driver assigns tasks to workers
RDD

- Distributed array, evenly split across workers
Broadcast Variables

- Immutable values cached by workers
RDDs Operations

```
// transformed RDDs
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split("\t").map(r => r(1)))
messages.cache()

// action 1
messages.filter(_.contains("mysql")).count()
```
Map

- Each element of the RDD is transformed using \( f() \)
Reduce

- Merges elements using associative function \( f() \)

```
Worker 1
[a]

Worker 2
[b]

Worker 3
[c]

[f(a,b)]

Worker 2

[f(c)]

Worker 3

f(f(a,b),c)

Master
```
Filter

- Keeps only elements matching a condition

Worker 1 [a] Worker 1 [a]

Worker 2 [b] Worker 2 []

Worker 3 [c] Worker 3 [c]
Document Similarity

- Textual similarity between large sets of documents
- All-pairs similarity is not feasible

Example applications:
- Plagiarism detection
- Exploit reuse
Approach

Stream of data

Matching approximation (fast)

Similarity Computation (slow)

Document Matches

unstructured documents

candidate pairs

similar pairs
Jaccard Similarity

• Documents are sets

\[ \text{SIM}(S,T) = \frac{|S \cap T|}{|S \cup T|} \]

SIM(S,T) = 3/8
Big Picture

The set of strings of length $k$ that appear in the document

**Signatures:** short integer vectors that represent the sets, and reflect their similarity

**Candidate pairs:** those pairs of signatures that we need to test for similarity
Document Shingling

• Split document in sequences of tokens

• Tokens are words/characters etc

• Sequence of k tokens = k-shingle (k-gram)

• Example:
  ◦ D = abcab
  ◦ k = 2 chars
  ◦ S(D) = \{ ab, bc, ca\}
Big Picture

Candidate pairs: those pairs of signatures that we need to test for similarity

Document → Shingling → Min Hashing → Locality-Sensitive Hashing

The set of strings of length \( k \) that appear in the document

Signatures: short integer vectors that represent the sets, and reflect their similarity
MinHashing

• Replace large sets by smaller signatures

• Must preserve the original Jaccard similarity when compared
MinHashing Computation 1

- Characteristic matrix:

<table>
<thead>
<tr>
<th>Element</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$e$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
MinHashing Computation 2

- Random permutation of rows
- MinHash = first row in which a column has ‘1’

<table>
<thead>
<tr>
<th>Element</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$e$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td><strong>1</strong></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$h(S1)=a$
MinHashing Example

random permutations of indices

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Input matrix

documents

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Signature matrix $M$

MinHash

similarity comparison

<table>
<thead>
<tr>
<th></th>
<th>1-3</th>
<th>2-4</th>
<th>1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col/Col</td>
<td>0.75</td>
<td>0.75</td>
<td>0</td>
</tr>
<tr>
<td>Sig/Sig</td>
<td>0.67</td>
<td>1.00</td>
<td>0</td>
</tr>
</tbody>
</table>
MinHash Property

• Probability that $h(D_1) = h(D_2)$ is $\sim$ $\text{SIM}(D_1, D_2)$
Big Picture

The set of strings of length $k$ that appear in the document

Signatures: short integer vectors that represent the sets, and reflect their similarity

Candidate pairs: those pairs of signatures that we need to test for similarity
Locality Sensitive Hashing

- Generate small list of candidate pairs from collection of signatures

- Idea:
  - Hash signatures to many buckets
  - Elements in the same bucket are candidate pairs

- Documents are split in bands (chunks) then hashed independently
LSH Example
LSH Property

- Probability that two similar signatures will agree on at least one of the bands is high
  - that is when they become candidates
References


ENEE757/CMSC818V
BootCamp

Graph Analytics with NetworkX

BumJun Kwon
Graph?

$G = (V, E)$

$V = \{1, 2, 3, 4, 5, 6\}$

$(u, v) \in E$ where $u, v \in V$
Graph?

\[ G = (V, E) \]

\[ V = \{1,2,3,4,5,6\} \]

\((u,v) \in E\) where \(u,v \in V\)
Examples

Control Flow Graph (CFG)  

Downloader Graph
How to build graphs?

NetworkX

- Python software package for complex networks.
- Python 2.7
Installation

easy_install networkx
pip install networkx
https://pypi.python.org/pypi/networkx
How to create a graph?

```python
>>> import networkx as nx
>>> G = nx.Graph()
```

**Graph()**: Undirected graph  
**DiGraph()**: Directed graph
Nodes & Edges

```python
>>> G.add_node(1)
>>> G.add_nodes_from([2,3,4,5])
>>> G.nodes()
[1, 2, 3, 4, 5]
```

```python
>>> G.add_edge(1,2)
>>> G.add_edge(1,5)
>>> G.add_edge(2,3)
>>> G.edges()
[(1, 2), (1, 5), (2, 3)]
```
Attributes

```python
>>> G.add_node(1, type = 'car', time = '2015-01-01')
>>> G.nodes(data='True')
[(1, {'type': 'car', 'time': '2015-01-01'})]

>>> G.node[1]
{'type': 'car', 'time': '2015-01-01'}
>>> G.node[1]['type']
'car'
>>> G.node[1]['type'] = 'boat'
>>> G.node[1]['type']
'boat'

>>> G.add_edge(1, 2, weight = 0.5)
>>> G.edge[1][2]['weight']
0.5
```
Algorithms: Traversal

Breath First Search (BFS)

Depth First Search (DFS)
Algorithms: Traversal

Depth First Search (DFS)

```python
>>> nx.dfs_edges(H)
<generator object dfs_edges at 0x10c2860f0>

>>> list(nx.dfs_edges(H))
[(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]

>>> H = nx.dfs_tree(H, 1)
>>> H.nodes()
[1, 2, 3, 4, 5, 6]

>>> H.edges()
[(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]
```
Algorithms: Tree

Minimum Spanning Tree

```python
>>> T = nx.minimum_spanning_tree(G, weight='weight')
>>> T.edges()
[(0, 1), (0, 3), (1, 2)]
```
Algorithms: Connectivity

Strongly Connected Component

```python
>>> nx.number_strongly_connected_components(G)
3

>>> for Gsub in nx.strongly_connected_component_subgraphs(G):
    ...     print Gsub.nodes()
    ...
[6, 7]
[8, 3, 4]
[1, 2, 5]
```
Algorithms: Connectivity

Connected Component

```python
>>> nx.number_connected_components(G)
2
>>> for Gsub in nx.connected_component_subgraphs(G):
...     print Gsub.nodes()
...
[1, 2, 5]
[8, 3, 4, 6, 7]
```
>>> nx.cycle_basis(G)
[[2, 4, 3, 1], [8, 7, 5], [6, 7, 5], [8, 6, 5]]
Algorithms: Structure

Clique

```python
>>> list(nx.find_cliques(G))
[[1, 2], [1, 3], [2, 4], [3, 4], [5, 8, 6, 7], [5, 4]]
```
Algorithms: Structure

Bridge

```python
>>> list(nx.articulation_points(G))
[5, 4]
```
Other Graph Metrics

in_degree: len(DiGraph.predecessors(n))
out-degree: len(DiGraph.successors(n))

Distance Metrics:
diameter(G), shortest_path(G)
Other Graph Metrics

Neighborhood Properties
- Clustering Coefficient: `average_clustering(G)`
- Pagerank(G)
- Betweenness centrality(G,n)

Dynamic algorithms?
Reference: NetworkX
For more algorithms and details


All the codes/urls in this presentation are documented: slides_text.txt