SOFT JOINS WITH TFIDF: WHY AND WHAT
Sim Joins on Product Descriptions

• Surface similarity can be **high** for descriptions of **distinct** items:

  o AERO TGX-Series Work Table -42" x 96" Model 1TGX-4296 All tables shipped KD AEROSPEC- 1TGX Tables are Aerospec Designed. In addition to above specifications; - All four sides have a V countertop edge ...

  o AERO TGX-Series Work Table -42" x 48" Model 1TGX-4248 All tables shipped KD AEROSPEC- 1TGX Tables are Aerospec Designed. In addition to above specifications; - All four sides have a V countertop ...

• Surface similarity can be **low** for descriptions of **identical** items:

  o Canon Angle Finder C 2882A002 Film Camera Angle Finders Right Angle Finder C (Includes ED-C & ED-D Adapters for All SLR Cameras) Film Camera Angle Finders & Magnifiers The Angle Finder C lets you adjust ...

  o CANON 2882A002 ANGLE FINDER C FOR EOS REBEL® SERIES PROVIDES A FULL SCREEN IMAGE SHOWS EXPOSURE DATA BUILT-IN DIOPTRIC ADJUSTMENT COMPATIBLE WITH THE CANON® REBEL, EOS & REBEL EOS SERIES.
Motivation

• Integrating data is important
• Data from different sources may not have consistent *object identifiers*
  – Especially automatically-constructed ones
• But databases will have human-readable names for the objects

• But names are tricky....
One solution: Soft (Similarity) joins

- A similarity join of two sets A and B is
  - an ordered list of triples \((s_{ij}, a_i, b_j)\) such that
    - \(a_i\) is from A
    - \(b_j\) is from B
    - \(s_{ij}\) is the similarity of \(a_i\) and \(b_j\)
    - the triples are in descending order
  
  - the list is either the top K triples by \(s_{ij}\) or ALL triples with \(s_{ij} > L\) ... or sometimes some approximation of these....
Example: soft joins/similarity joins

**Input: Two Different Lists of Entity Names**

- Abraham Lincoln Birthplace NHS
- Acadia NP
- Adams NHS
- Agate Fossil Beds NM
- Alagnak Wild River
- Alaska Public Lands Inf. Center
- Alibates Flint Quarries NM
- Allegheny Portage Railroad NHS
- American Memorial Park
- Amistad NRA
- Andersonville NHS
- Andersonville NHP
- Aniakchak NM
- Antietam NB
- Apostle Islands NL
- Appomattox Court House NHP
- Arches NP
- Arkansas Post N. Mem.
- Assateague Island NS
- Aztec Ruins NM
- Badlands NP
- Bandelier NM
- Bent's Old Fort NHS
- Bering Land Bridge N. Preserve
- Big Bend NP
- Big Cypress N. Preserve

...
TFIDF similarity works well for names

- Low weight for mismatches on common abbreviations (NPRES)
- High weight for mismatches on unusual terms (Appomattox, Aniakchak, ...)

<table>
<thead>
<tr>
<th>Adams NHS</th>
<th>Amistad NRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agate Fossil Beds NM</td>
<td>Andersonville NHP</td>
</tr>
<tr>
<td>Alagnak Wild River</td>
<td>Aniakchak NM</td>
</tr>
<tr>
<td>Alaska Public Lands Inf. Center</td>
<td>Antietam NB</td>
</tr>
<tr>
<td>Alibates Flint Quarries NM</td>
<td>Apostle Islands NL</td>
</tr>
<tr>
<td>Allegheny Portage Railroad NHS</td>
<td>Appomattox Court House NHP</td>
</tr>
<tr>
<td>American Memorial Park</td>
<td>Arches NP</td>
</tr>
<tr>
<td>Amistad NRA</td>
<td>Arkansas Post N. Mem.</td>
</tr>
<tr>
<td>Andersonville NHS</td>
<td>Assateague Island NS</td>
</tr>
<tr>
<td>Andrew Johnson NHS</td>
<td>Aztec Ruins NM</td>
</tr>
<tr>
<td>Aniakchak NM &amp; NPRES</td>
<td>Badlands NP</td>
</tr>
<tr>
<td>Antietam NB</td>
<td>Bandelier NM</td>
</tr>
<tr>
<td>Apostle Islands NL</td>
<td>Bent's Old Fort NHS</td>
</tr>
<tr>
<td>Appalachian National Scenic Trail</td>
<td>Bering Land Bridge N. Preserve</td>
</tr>
<tr>
<td>Appomattox Courthouse NHP</td>
<td>Big Bend NP</td>
</tr>
<tr>
<td>Arches NP</td>
<td>Big Cypress N. Preserve</td>
</tr>
<tr>
<td>Arkansas Post NM</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example: soft joins/similarity joins

Output: Pairs of Names Ranked by Similarity

- Chickamauga & Chattanooga NMP:d445
  George Washington Carver NM:d499
  Salinas Pueblo Missions NM:d597
  Florissant Fossil Beds NM:d473
  Hagerman Fossil Beds NM:d517
  Gila Cliff Dwellings NM:d502
  Booker T. Washington NM:d423
  Chickamauga & Chattanooga NMP:d72
  George Washington Carver NM:d153
  Salinas Pueblo Missions NM:d329
  Florissant Fossil Beds NM:d116
  Hagerman Fossil Beds NM:d177
  Gila Cliff Dwellings NM:d156
  Booker T. Washington NM:d38

- Obed Wild & Scenic River:d570
  Andersonville NHP:d401
  Sitka NHP:d606
  Bering Land Bridge N. Preserve:d413
  Sequoia & Kings Canyon NP:d603
  Glacier Bay NP & Preserve:d643
  NP of American Samoa:d561
  Kalaupapa NHS:d538
  Obed Wild and Scenic River:d283
  Andersonville NHS:d11
  Sitka NHS:d342
  Bering Land Bridge NPRE:d26
  Sequoia and Kings Canyon NP:d339
  Glacier Bay NP & NPRE:d157
  National Park Of American Samoa:d267
  Kalaupapa NHS:d210

- Lake Mead NRA:d545
  Upper Delaware Scenic & Rec. River:d617
  Lake Mead NRA (Nevada):d224
  Upper Delaware Scenic & Recreational River:d368
How well does TFIDF work?

- **Input:** query
- **Output:** ordered list of documents

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>√</td>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>2</td>
<td>√</td>
<td>$a_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>3</td>
<td>×</td>
<td>$a_3$</td>
<td>$b_3$</td>
</tr>
<tr>
<td>4</td>
<td>√</td>
<td>$a_4$</td>
<td>$b_4$</td>
</tr>
<tr>
<td>5</td>
<td>√</td>
<td>$a_5$</td>
<td>$b_5$</td>
</tr>
<tr>
<td>6</td>
<td>√</td>
<td>$a_6$</td>
<td>$b_6$</td>
</tr>
<tr>
<td>7</td>
<td>×</td>
<td>$a_7$</td>
<td>$b_7$</td>
</tr>
<tr>
<td>8</td>
<td>√</td>
<td>$a_8$</td>
<td>$b_8$</td>
</tr>
<tr>
<td>9</td>
<td>√</td>
<td>$a_9$</td>
<td>$b_9$</td>
</tr>
<tr>
<td>10</td>
<td>×</td>
<td>$a_{10}$</td>
<td>$b_{10}$</td>
</tr>
<tr>
<td>11</td>
<td>×</td>
<td>$a_{11}$</td>
<td>$b_{11}$</td>
</tr>
<tr>
<td>12</td>
<td>√</td>
<td>$a_{12}$</td>
<td>$b_{12}$</td>
</tr>
</tbody>
</table>

Precision at $K$: $G_K/K$

Recall at $K$: $G_K/G$

$G$: # good pairings

$G_K$: # good pairings in first $K$
Table VI. Pairs of Names from the Hoovers and IonTech Relations

| ✓   | Texas Instruments Incorporated | TEXAS INSTRUMENTS INC |
| ✓   | The New York Times Company      | NEW YORK TIMES CO     |
| ✓   | Campo Electronics, Appliances   | CAMPO ELECTRONICS     |
|     | and Computers, Inc.             | APPLIANCES            |
| ✓   | Cascade Communications Corp.    | CASCADE COMMUNICATION |
| ✓   | The McGraw-Hill Companies, Inc. | MCGRAW-HILL CO        |
| ✓   | U S WEST Communications Group   | U S WEST INC          |
| ✗   | Silicon Valley Group, Inc.      | SILICON VALLEY RESEARCH INC |
| ✗   | The Reynolds and Reynolds Company | REYNOLDS & REYNOLDS CO |
| ✓   | InTime Systems International, Inc. | INTIME SYSTEMS INTERNATIONAL INC |
There are refinements to TFIDF distance – eg ones that extend with soft matching at the token level (e.g., softTFIDF)
SOFT JOINS WITH TFIDF: HOW?
**TFIDF similarity**

\[ DF(w) = \# \text{different docs } w \text{ occurs in} \]
\[ TF(w, d) = \# \text{different times } w \text{ occurs in doc } d \]
\[ IDF(w) = \frac{|D|}{DF(w)} \]
\[ u(w, d) = \log(TF(w, d) + 1) \cdot \log(IDF(w)) \]
\[ u(d) = \langle u(w_1, d), \ldots, u(w_{|V|}, d) \rangle \]
\[ v(d) = \frac{u(d)}{\|u(d)\|_2} \]
\[ sim(v(d_1), v(d_2)) = v(d_1) \cdot v(d_2) = \sum_w \frac{u(w, d_1)}{\|u(d_1)\|_2} \cdot \frac{u(w, d_2)}{\|u(d_2)\|_2} \]
Soft TFIDF joins

• A similarity join of two sets of TFIDF-weighted vectors A and B is
  – an ordered list of triples \((s_{ij}, a_i, b_j)\) such that
    • \(a_i\) is from A
    • \(b_j\) is from B
    • \(s_{ij}\) is the dot product of \(a_i\) and \(b_j\)
    • the triples are in descending order

• the list is either the top K triples by \(s_{ij}\) or ALL triples with \(s_{ij} > L\) … or sometimes some approximation of these….  
• do not compare similarity of all pairs!
PARALLEL SOFT JOINS
Efficient Parallel Set-Similarity Joins Using MapReduce

Rares Vernica  
Department of Computer Science  
University of California, Irvine  
rares@ics.uci.edu

Michael J. Carey  
Department of Computer Science  
University of California, Irvine  
mjcarey@ics.uci.edu

Chen Li  
Department of Computer Science  
University of California, Irvine  
chenli@ics.uci.edu

SIGMOD 2010
Adapted to Guinea Pig....
want this to work for long documents or short ones... and keep the relations simple

Statistics for computing TFIDF with IDFs local to each relation
What’s the algorithm?

- **Step 1**: create document vectors as \((C_d, d, \text{term}, \text{weight})\) tuples
- **Step 2**: *join* the tuples from A and B: one sort and reduce
  - Gives you tuples \((a, b, \text{term}, w(a,\text{term}) \times w(b,\text{term}))\)
- **Step 3**: *group* the common terms by \((a,b)\) and reduce to aggregate the components of the sum

```python
# naive algorithm: use all pairs for finding matches
rel1Docs = Filter(docvec, by=lambda(rel,doc,term,weight): rel=='icepark')
rel2Docs = Filter(docvec, by=lambda(rel,doc,term,weight): rel=='npspark')
softjoin = Join(Join(rel1Docs,by=lambda(rel,doc,term,weight):term),
                Join(rel2Docs,by=lambda(rel,doc,term,weight):term)) \
    | ReplaceEach(by=lambda((rel1,doc1,term,weight1),(rel2,doc2,term2,weight2)): (doc1,doc2,weight1*weight2)) \
    | Group(by=lambda(doc1,doc2,p):(doc1,doc2), \retaining=lambda(doc1,doc2,p):p, 
        reducingTo=ReduceToSum()) \
    | ReplaceEach(by=lambda((doc1,doc2),sim):(doc1,doc2,sim))
simpairs = Filter(softjoin, by=lambda(doc1,doc,sim):sim>0.75)
```
Making the algorithm smarter....
# naive algorithm for the soft joint will use all pairs for finding matches
rel1Docs = Filter(docvec, by=lambda(rel,doc,term,weight): rel=='icepark')
rel2Docs = Filter(docvec, by=lambda(rel,doc,term,weight): rel=='nspspark')
softjoin = Join(Jin(rel1Docs, by=lambda(rel,doc,term,weight): term), Jin(rel2Docs, by=lambda(rel,doc,term,weight): term),
  | ReplaceEach(by=lambda((rel1,doc1,term,weight1),(rel2,doc2,term_,weight2)): (doc1,doc2,weight1+weight2))
  | Group(by=lambda(doc1,doc2,p):(doc1,doc2), reducingTo=sum0fP)
  | ReplaceEach(by=lambda((doc1,doc2),sim):(doc1,doc2,sim))

we should make a **smart** choice about which terms to use
Adding heuristics to the soft join - 1

# 1) pick only top terms in each document

```python
topTermsInEachDocForRel1 = Group(rel1Docs,
    by=lambda(rel,doc,term,weight):doc,
    retaining=lambda(rel,doc,term,weight):(weight,term)) 
    | ReplaceEach(by=lambda(doc,termList):sorted(termList,reverse=True)[0:NUM_TOP_TERMS]) 
    | Flatten(by=lambda x:x) | ReplaceEach(by=lambda(weight,term):term)
```

# 2) pick terms that have some minimal weight in their documents

```python
highWeightTermsForRel1 = Filter(rel1Docs, by=lambda(rel,doc,term,weight):weight>=MIN_TERM_WEIGHT) 
    | ReplaceEach(by=lambda(rel,doc,term,weight):term)
```

# 3) pick terms with some maximal DF

```python
lowDocFreqTerms = Filter(docFreq,by=lambda(rel,term,df):df<=MAX_TERM_DF) 
    | ReplaceEach(by=lambda(rel,term,df):term)
```

# terms we will join on should pass all of the tests above

```python
usefulTerms = Join(Jin(topTermsInEachDocForRel1), Jin(highWeightTermsForRel1)) 
    | ReplaceEach(by=lambda(term1,term2):term1) 
    | JoinTo(Jin(lowDocFreqTerms)) 
    | ReplaceEach(by=lambda(term1,term2):term1) | Distinct()
```
Adding heuristics to the soft join - 2

```
softjoin = Join( Jin(rel1Docs, by=lambda(rel,doc,term,weight):term),
                Jin(usefulTerms)) \ 
  | ReplaceEach(by=lambda(rel1doc,term):rel1doc) \ 
  | JoinTo( Jin(rel2Docs, by=lambda(rel,doc,term,weight):term),
        by=lambda(rel,doc,term,weight):term)
  | ReplaceEach( \ 
        by=lambda((rel1,doc1,term,weight1),(rel2,doc2,term_,weight2)): \ 
        (doc1,doc2,weight1*weight2)) \ 
  | Group(by=lambda(doc1,doc2,p):(doc1,doc2),
        retaining=lambda (doc1,doc2,p):p, 
        reducingTo=ReduceToSum()) \ 
  | ReplaceEach(by=lambda(((doc1,doc2),sim):(doc1,doc2,sim))
```
Adding heuristics

• Parks:
  – input 40k
  – data 60k
  – docvec 102k
  – softjoin
    • 539k tokens
    • 508k documents
    • 0 errors in top 50

• w/ heuristics:
  – input 40k
  – data 60k
  – docvec 102k
  – softjoin
    • 32k tokens
    • 24k documents
    • 3 errors in top 50
    • < 400 useful terms
Adding heuristics

• SO vs Wikipedia:
  – input 612M
  – docvec 1050M
  – softjoin 3.7G
  – join time: 54.30m

• with heuristics
  – max DF=50
  – softjoin 2.5G
  – join time 27:30m

• with heuristics
  – max DF=20
  – softjoin 1.2G
  – join time 11:50m

  – includes 70% of the pairs with score $\geq 0.95$

Single-threaded Java baseline: 2-3 days