Announcements

- Working AWS codes are out
- 605 waitlist ~= 25, slots ~= 15
- 10-805 project deadlines now posted
- William has no office hours next week

Recap

- An algorithm for testing a huge naïve Bayes classifier
  - More generally: for evaluating a linear classifier on a test set efficiently on-disk, using stream-and-sort or map-reduce ops only
- Sketch of algorithm for Rocchio training/ testing

Recap

- Abstractions for map-reduce (TFIDF example)
- map-side vs reduce-side joins

Proposed syntax:

$table2 = \text{MAP } table1 \text{ TO } \lambda\, row : f(row))$

Proposed syntax: *f(row) → {true,false}*

$table2 = \text{FILTER } table1 \text{ BY } \lambda\, row : f(row))$

Proposed syntax: *f(row) → list of rows*

$table2 = \text{FLATMAP } table1 \text{ TO } \lambda\, row : f(row))$

Proposed syntax:

$\text{GROUP } table \text{ BY } \lambda\, row : f(row)$

Could define *f* via: a function, a field of a defined *record* structure, …

Proposed syntax:

$\text{JOIN } table1 \text{ BY } \lambda\, row : f(row),$
$\quad \underline{table2 \text{ BY }} \lambda\, row : g(row)$

Today

- Less abstract abstractions

Proposed syntax:

$table2 = \text{MAP } table1 \text{ TO } \lambda\, row : f(row))$

Proposed syntax: *f(row)* → *{true,false}*

$table2 = \text{FILTER } table1 \text{ BY } \lambda\, row : f(row))$

Proposed syntax: *f(row)* → *list of rows*

$table2 = \text{FLATMAP } table1 \text{ TO } \lambda\, row : f(row))$

Proposed syntax:

$\text{GROUP } table \text{ BY } \lambda\, row : f(row)$

Could define *f* via: a function, a field of a defined *record* structure, ...

Proposed syntax:

$\text{JOIN } table1 \text{ BY } \lambda\, row : f(row),$
$\quad \underline{table2 \text{ BY }}\ \lambda\, row : g(row)$

PIG: A WORKFLOW/DATAFLOW LANGUAGE

PIG: word count

- Declarative "data flow" language

```
A = load '/tmp/bible+shakes.nopunc';
B = foreach A generate flatten(TOKENIZE((chararray)$0)) as word;
C = filter B by word matches '\w+';
D = group C by word;
E = foreach D generate COUNT(C) as count, group as word;
F = order E by count desc;
store F into '/tmp/wc';
```

PIG program is a bunch of **assignments** where every LHS is a **relation**.
No loops, conditionals, etc allowed.

More on Pig

- Pig Latin
  - atomic types + compound types like tuple, bag, map
  - execute locally/interactively or on hadoop
- can embed Pig in Java (and Python and …)
- can call out to Java from Pig

```
A = load '/tmp/bible+shakes.nopunc';
B = foreach A generate flatten(TOKENIZE((chararray)$0)) as word;
C = filter B by word matches '\w+';
D = group C by word;
E = foreach D generate COUNT(C) as count, group as word;
F = order E by count desc;
store F into '/tmp/wc';
```

Tokenize – built-in function

Flatten – special keyword, which applies to the next step in the process – ie foreach is transformed from a MAP to a FLATMAP

PIG parses and **optimizes** a sequence of commands before it executes them
It's smart enough to turn GROUP … FOREACH… SUM … into a map-reduce

- LOAD '*hdfs-path*' AS (*schema)*
  - *schemas can include int, double, bag, map, tuple, …*
- FOREACH *alias* GENERATE … AS …, …
  - *transforms each row of a relation*
- DESCRIBE *alias/* ILLUSTRATE *alias -- debugging*
- GROUP *alias* BY …
- FOREACH *alias* GENERATE *group,* SUM(….)
  - *GROUP/GENERATE … aggregate op together act like a map-reduce*
- JOIN *r* BY *field, s* BY *field, …*
  - *inner join to produce rows: r::f1, r::f2, … s::f1, s::f2, …*
- CROSS *r, s, …*
  - *use with care unless all but one of the relations are singleton*
- User defined functions as operators
  - *also for loading, aggregates, …*

```
A = load '/tmp/bible+shakes.nopunc';
B = foreach A generate flatten(TOKENIZE((chararray)$0)) as word;
C = filter B by word matches '\w+';
D = group C by word;
E = foreach D generate COUNT(C) as count, group as word;
F = order E by count desc;
store F into '/tmp/wc';
```

Example: the optimizer will compress these steps into one map-reduce operation

ANOTHER EXAMPLE: COMPUTING TFIDF IN PIG LATIN

Abstract Implementation: [TF]IDF

1/2

data = *pairs (docid, term) where term is a word appears in document with id docid*

opera

- DIST
- GRO

ep

key	value
found	~~(d123,found),(d134,found),~~… 2456
aardvark	~~(d123,aardvark),~~… 7

docId	term
d123	found
d123	aardvark

docFreq = DISTINCT data
| GROUP BY λ (docid,term):term REDUCING TO

key	value
1	12451

docIds = MAP DATA BY= λ (docid,term):docid | DISTINCT
numDocs = GROUP docIds BY λ docid:1 REDUCING TO count */* (1,numDocs) */*

dataPlusDF =
JOIN data BY λ (docid, term):term, docFreq BY λ (term, df):term
| MAP λ ((docid,term),(term,df)):(docId,term,df) */* (docId,term,document-freq) */*

unnormalizedDocVecs = JOIN dataPlusDF by λ row:1, numDocs by λ row:1
| MAP λ ((docId,term,df),(dummy,numDocs)): (docId,term,log(numDocs/df))
/ (docId, term, weight-before-normalizing) : **u** */*

Abstract Implementation: TFIDF

2/2

```
normalizers =
    GROUP unnormalizedDocVecs BY λ (docId,term,w):docid
    RETAINING λ (docId,term,w): w²
    REDUCING TO sum  /* (docid,sum-of-square-weights) */
```

key	
d1234	(d1234,found,1.542), (d1234,aardvark,13.23),… 37.234
d3214	…. 29.654

```
docVec = JOIN unnormalizedDocVecs BY λ (docId,term,w):docid,
                    normalizers BY λ (docId,norm):docid
         | MAP λ ((docId,term,w), (docId,norm)): (docId,term,w/sqrt(norm))
          /* (docId, term, weight) */
```

docId	term	w
d1234	found	1.542
d1234	aardvark	13.23

docId	w
d1234	37.234
d1234	37.234

```
DEFINE tf_idf(in_relation, id_field, text_field) RETURNS out_relation {
  token_records = foreach $in_relation generate $id_field, FLATTEN(TOKENIZE($text_field)) as tokens;

  /* Calculate the term count per document */
  doc_word_totals = foreach (group token_records by ($id_field, tokens)) generate
    FLATTEN(group) as ($id_field, token),
    COUNT_STAR(token_records) as doc_total;

  /* Calculate the document size */
  pre_term_counts = foreach (group doc_word_totals by $id_field) generate
    group AS $id_field,
    FLATTEN(doc_word_totals.(token, doc_total)) as (token, doc_total),
    SUM(doc_word_totals.doc_total) as doc_size;

  /* Calculate the TF */
  term_freqs = foreach pre_term_counts generate $id_field as $id_field,
    token as token,
    ((double)doc_total / (double)doc_size) AS term_freq;

  /* Get count of documents using each token, for idf */
  token_usages = foreach (group term_freqs by token) generate
    FLATTEN(term_freqs) as ($id_field, token, term_freq),
    COUNT_STAR(term_freqs) as num_docs_with_token;

  /* Get document count */
  just_ids = foreach $in_relation generate $id_field;
  ndocs = foreach (group just_ids all) generate COUNT_STAR(just_ids) as total_docs;

  /* Note the use of Pig Scalars to calculate idf */
  $out_relation = foreach token_usages {
    idf    = LOG((double)ndocs.total_docs/(double)num_docs_with_token);
    tf_idf = (double)term_freq * idf;
    generate $id_field as $id_field,
      token as score,
      (chararray)tf_idf as value:chararray;
  };
};
```

group outputs record with "group" as field name

(docid,token) → (docid,token,tf(token in doc))

(docid,token,tf) → (docid,token,tf,length(doc))

(docid,token,tf,n)→(…,tf/n)

(docid,token,tf,n,tf/n)→(…,df)

ndocs.total_docs

relation-to-scalar casting

(docid,token,tf,n,tf/n)→(docid,token,tf/n * id)

Debugging/visualization

```
DESCRIBE fgPhrases;
2014-04-01 16:43:06,631 [main] WARN  org.apache.pig.PigServer - Encountered
2014-04-01 16:43:06,631 [main] WARN  org.apache.pig.PigServer - Encountered
fgPhrases: {xy: (x: bytearray,y: bytearray),c: int}
grunt> ILLUSTRATE fgPhrases;

--------------------------------------------------------------
| fgPhrases1          | xy:bytearray        | c:int         |
--------------------------------------------------------------
|                     | patachon mon        | 1             |
--------------------------------------------------------------
-----------------------------------------------------------------------------------------------
| fgPhrases          | xy:tuple(x:bytearray,y:bytearray)                          | c:int         |
-----------------------------------------------------------------------------------------------
|                    | (patachon, mon)                                            | 1             |
-----------------------------------------------------------------------------------------------
```

```
DEFINE tokenize_docs `ruby tokenize_documents.rb --id_field=0 --text_field=1 --map` SHIP('tokenize_documents.rb');

raw_documents = LOAD '$DOCS' AS (doc_id:chararray, text:chararray);
tokenized     = STREAM raw_documents THROUGH tokenize_docs AS (doc_id:chararray, token:chararray);
```

TF-IDF in PIG - another version


```
DEFINE tokenize_docs `ruby tokenize_documents.rb --id_field=0 --text_field=1 --map` SHIP('tokenize_documents.rb');

raw_documents = LOAD '$DOCS' AS (doc_id:chararray, text:chararray);
tokenized     = STREAM raw_documents THROUGH tokenize_docs AS (doc_id:chararray, token:chararray);
```


```
doc_tokens       = GROUP tokenized BY (doc_id, token);
doc_token_counts = FOREACH doc_tokens GENERATE FLATTEN(group) AS (doc_id, token), COUNT(tokenized) AS num_doc_tok_usages;

doc_usage_bag    = GROUP doc_token_counts BY doc_id;
doc_usage_bag_fg = FOREACH doc_usage_bag GENERATE
                     group                                                AS doc_id,
                     FLATTEN(doc_token_counts.(token, num_doc_tok_usages)) AS (token, num_doc_tok_usages),
                     SUM(doc_token_counts.num_doc_tok_usages)              AS doc_size
                   ;

term_freqs = FOREACH doc_usage_bag_fg GENERATE
               doc_id                                             AS doc_id,
               token                                              AS token,
               ((double)num_doc_tok_usages / (double)doc_size) AS term_freq;
             ;

term_usage_bag  = GROUP term_freqs BY token;
token_usages    = FOREACH term_usage_bag GENERATE
                    FLATTEN(term_freqs) AS (doc_id, token, term_freq),
                    COUNT(term_freqs)   AS num_docs_with_token
                   ;

tfidf_all = FOREACH token_usages {
              idf    = LOG((double)$NDOCS/(double)num_docs_with_token);
              tf_idf = (double)term_freq*idf;
                GENERATE
                  doc_id AS doc_id,
                  token  AS token,
                  tf_idf AS tf_idf
                ;
             };

STORE tfidf_all INTO '$OUT';
```

GUINEA PIG

GuineaPig: PIG in Python

- Pure Python (< 1500 lines)
- Streams Python data structures
  - strings, numbers, tuples (a,b), lists [a,b,c]
  - No records: operations defined functionally
- Compiles to Hadoop streaming pipeline
  - Optimizes sequences of MAPs
- Runs locally without Hadoop
  - compiles to stream-and-sort pipeline
  - intermediate results can be viewed
- Can easily run parts of a pipeline
- http://curtis.ml.cmu.edu/w/courses/index.php/Guinea_Pig

GuineaPig: PIG in Python

- Pure Python, streams Python data structures
  - not too much new to learn (eg field/record notation, special string operations, UDFs, …)
  - codebase is small and readable
- Compiles to Hadoop or stream-and-sort, can easily run parts of a pipeline
  - intermediate results often are (and always can be) stored and inspected
  - plan is fairly visible
- Syntax includes high-level operations but also fairly detailed description of an optimized map-reduce step
  - Flatten | Group(by=…, retaining=…, reducingTo=…)

A wordcount example

```
# always start like this
from guineapig import *
import sys

# supporting rou
def tokens(line)
    for tok in l
        yield to

#always subclass
class WordCount(

    wc = ReadLines('corpus.txt') | Flatten(by=tokens) | Group(by=lambda x:x, reducing  h=ReduceToCount())

# always end like this
if __name__ == "__main__":
    WordCount().main(sys.argv)
```

```
ReduceTo(int,by=lambda accum,val:accum+1)
```

```
class WordCount(Planner):
    lines = ReadLines('corpus.txt')
    words = Flatten(lines,by=tokens)
    wordCount = Group(words, by=lambda x:x, reducingTo=ReduceToCount())
```

class variables in the planner are data structures

```
wordCount = Group(words,by=<function <lambda> at
| words = Flatten(lines, by=<function tokens at 0
| | lines = ReadLines("corpus.txt")
```

Wordcount example ….

- Data structure can be converted to a series of "abstract map-reduce tasks"

```
===================================================================
map-reduce task 1: corpus.txt => wordCount
 - +-------------------- explanation --------------------
 - |   read corpus.txt with lines
 - |   flatten to words
 - |   group to wordCount
 - +-------------------- commands --------------------
 - |   python longer-wordcount.py --view=wordCount --do=doGroupMap < corpus.txt | LC_COLLATE=C sort -k1 |
```

```
python longer-wordcount.py --view=wordCount --do=doGroupMap < corpus.txt \
      | LC_COLLATE=C sort -k1 \
      | python longer-wordcount.py --view=wordCount --do=doStoreRows \
     > gpig_views/wordCount.gp
```

More examples of GuineaPig

Join syntax, macros, Format command

```
class WordCmp(Planner):

    def wcPipe(fileName):
        return ReadLines(fileName) | Flatten(by=tokens) | Group(by=lambda x:x, reducingTo

    wc1 = wcPipe('bluecorpus.txt')
    wc2 = wcPipe('redcorpus.txt')

    cmp = Join( Jin(wc1, by=lambda(word,n):word), Jin(wc2, by=lambda(word,n):word) ) \
          | ReplaceEach(by=lambda((word1,n1),(word2,n2)):(word1, score(n1,n2)))

    result = Format(cmp, by=lambda(word,blueScore):'%6.4f %s' % (blueScore,word))
```

Incremental debugging, when intermediate views are stored:

% python wrdcmp.py –store result

…

% python wrdcmp.py –store result –reuse cmp

More examples of GuineaPig

Full Syntax for Group

```
Group(wc,  by=lambda (word,count):word[:k],
           retaining=lambda (word,count):count,
           reducingTo=ReduceToSum())
```

equiv to:

```
Group(wc,  by=lambda (word,count):word[:k],
           reducingTo=
             ReduceTo(int,
                      lambda accum,(word,count)): accum+count))
```

ANOTHER EXAMPLE: COMPUTING TFIDF IN GUINEA PIG

Actual Implementation

```
D = GPig.getArgvParams()
idDoc = ReadLines(D.get('corpus','idcorpus.txt')) | Map(by=lambda line:line.strip().split("\t"))
idWords = Map(idDoc, by=lambda (docid,doc): (docid,doc.lower().split()))
data = FlatMap(idWords, by=lambda (docid,words): map(lambda w:(docid,w),words))

#compute document frequency
docFreq = Distinct(data) \
    | Group(by=lambda (docid,term):term, retaining=lambda(docid,term):docid, reducingTo=ReduceToCount())

docIds = Map(data, by=lambda (docid,term):docid) | Distinct()
ndoc = Group(docIds, by=lambda row:'ndoc', reducingTo=ReduceToCount())

#unweighted document vectors

udocvec1 = Join( Jin(data,by=lambda(docid,term):term), Jin(docFreq,by=lambda(term,df):term) )
udocvec2 = Map(udocvec1, by=lambda((docid,term1),(term2,df)):(docid,term1,df))
udocvec3 = Augment(udocvec2, sideview=ndoc, loadedBy=lambda v:GPig.onlyRowOf(v))
udocvec = Map(udocvec3, by=lambda((docid,term,df),(dummy,ndoc)):(docid,term,math.log(ndoc/df)))

norm = Group( udocvec, by=lambda(docid,term,weight):docid,
                       retaining=lambda(docid,term,weight):weight*weight,
                       reducingTo=ReduceToSum() )

docvec = Join( Jin(norm,by=lambda(docid,z):docid), Jin(udocvec,by=lambda(docid,term,weight):docid) ) \
         | Map( by=lambda((docid1,z),(docid2,term,weight)): (docid1,term,weight/math.sqrt(z)) )
```

Actual Implementation

```
D = GPig.getArgvParams()
idDoc = ReadLines(D.get('corpus','idcorpus.txt')) | Map(by=lambda line:l
idWords = Map(idDoc, by=lambda (docid,doc): (docid,doc.lower().split()))
data = FlatMap(idWords, by=lambda (docid,words): map(lambda w:(docid,w),
```

docId	w
d123	found
d123	aardvark

Actual Implementation

```
D = GPig.getArgvParams()
idDoc = ReadLines(D.get('corpus','idcorpus.txt')) | Map(by=lambda line:l
idWords = Map(idDoc, by=lambda (docid,doc): (docid,doc.lower().split()))
data = FlatMap(idWords, by=lambda (docid,words): map(lambda w:(docid,w),
```

docId	w
d123	found
d123	aardvark

```
docFreq = Distinct(data) \
    | Group(by=lambda (docid,term):term, retaining=lambda(docid,term):docid,

                                         , reducingTo=ReduceToCount()
```

key	value
found	~~(d123,found),(d134,found),~~… 2456
aardvark	~~(d123,aardvark),~~… 7

Actual Implementation

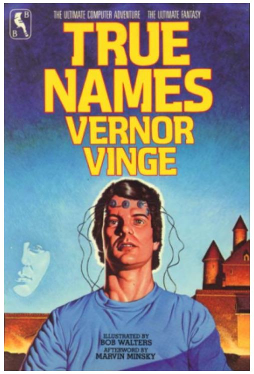
```
udocvec1 = Join( Jin(data,by=lambda(docid,term):term), Jin(docFreq,by=lambda(term,df):term) )
udocvec2 = Map(udocvec1, by=lambda((docid,term1),(term2,df)):(docid,term1,df))
udocvec3 = Augment(udocvec2, sideview=ndoc, loadedBy=lambda v:GPig.onlyRowOf(v))
udocvec = Map(udocvec3, by=lambda((docid,term,df),(dummy,ndoc)):(docid,term,math.log(ndoc/df))
```

Augment: loads a preloaded object b at mapper initialization time, cycles thru the input, and generates pairs (a,b)

```
from guineapig import *

# compute TFIDF in Guineapig

import sys
import math

class TFIDF(Planner):

    D = GPig.getArgvParams()
    idDoc = ReadLines(D.get('corpus','idcorpus.txt')) | Map(by=lambda line:line.strip().split("\t"))
    idWords = Map(idDoc, by=lambda (docid,doc): (docid,doc.lower().split()))
    data = FlatMap(idWords, by=lambda (docid,words): map(lambda w:(docid,w),words))

    #compute document frequency
    docFreq = Distinct(data) \
        | Group(by=lambda (docid,term):term, retaining=lambda(docid,term):docid, reducingTo=ReduceToCount(

    docIds = Map(data, by=lambda (docid,term):docid) | Distinct()
    ndoc = Group(docIds, by=lambda row:'ndoc', reducingTo=ReduceToCount())

    #unweighted document vectors

    udocvec1 = Join( Jin(data,by=lambda(docid,term):term), Jin(docFreq,by=lambda(term,df):term) )
    udocvec2 = Map(udocvec1, by=lambda((docid,term1),(term2,df)):(docid,term1,df))
    udocvec3 = Augment(udocvec2, sideview=ndoc, loadedBy=lambda v:GPig.onlyRowOf(v))
    udocvec = Map(udocvec3, by=lambda((docid,term,df),(dummy,ndoc)):(docid,term,math.log(ndoc/df)))

    norm = Group( udocvec, by=lambda(docid,term,weight):docid,
                           retaining=lambda(docid,term,weight):weight*weight,
                           reducingTo=ReduceToSum() )

    docvec = Join( Jin(norm,by=lambda(docid,z):docid), Jin(udocvec,by=lambda(docid,term,weight):docid) ) \
             | Map( by=lambda((docid1,z),(docid2,term,weight)): (docid1,term,weight/math.sqrt(z)) )

# always end like this
if __name__ == "__main__":
    p = TFIDF()
    p.main(sys.argv)
```

Outline: Soft Joins with TFIDF

- Why similarity joins are important
- Useful similarity metrics for sets and strings
- Fast methods for K-NN and similarity joins
  - Blocking
  - Indexing
  - Short-cut algorithms
  - Parallel implementation



In the once upon a time days of the First Age of Magic, the prudent sorcerer regarded his own true name as his most valued possession but also the greatest threat to his continued good health, for--the stories go--once an enemy, even a weak unskilled enemy, learned the sorcerer's true name, then routine and widely known spells could destroy or enslave even the most powerful. As times passed, and we graduated to the Age of Reason and thence to the first and second industrial revolutions, such notions were discredited. Now it seems that the Wheel has turned full circle (even if there never really was a First Age) and we are back to worrying about true names again:

The first hint Mr. Slippery had that his own True Name might be known--and, for that matter, known to the Great Enemy--came with the appearance of two black Lincolns humming up the long dirt driveway ... Roger Pollack was in his garden weeding, had been there nearly the whole morning.... Four heavy-set men and a hard-looking female piled out, started purposefully across his well-tended cabbage patch.…

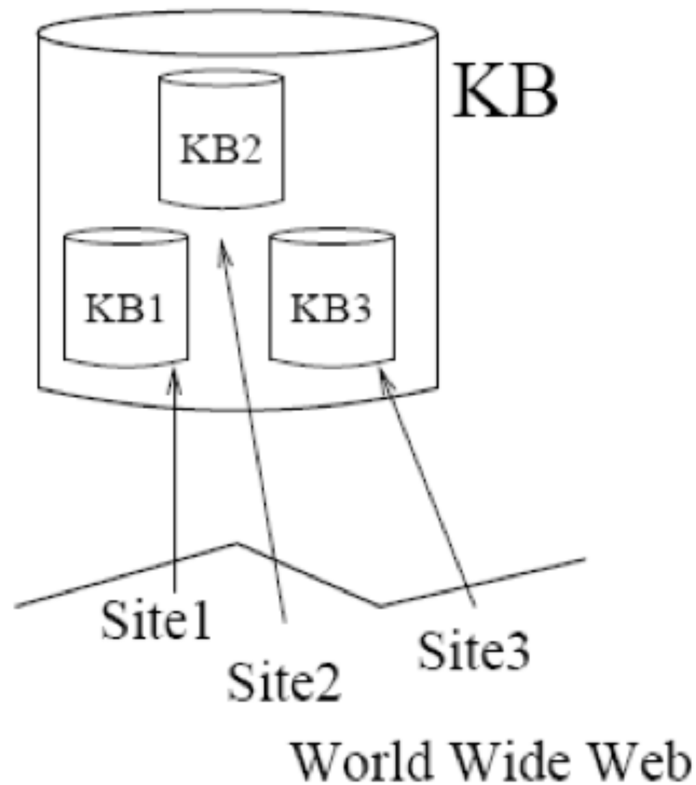
This had been, of course, Roger Pollack's great fear. They had discovered Mr. Slippery's True Name and it was Roger Andrew Pollack TIN/SSAN 0959-34-2861.

SOFT JOINS WITH TFIDF:
WHY AND WHAT

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….



Humongous	Humongous Entertainment
Headbone	Headbone Interactive
The Lion King: Storybook	Lion King Animated StoryBook
Disney's Activity Center, The Lion King	The Lion King Activity Center

Microsoft	Microsoft Kids Microsoft/Scholastic
Kestrel	American Kestrel Eurasian Kestrel
Canada Goose	Goose, Aleutian Canada
Mallard	Mallard, Mariana

Sim Joins on Product Descriptions

- Similarity can be **high** for descriptions of **distinct** items:
  - 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 ...
  - 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 ..
- Similarity can be **low** for descriptions of **identical** items:
  - 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 ...
  - 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.

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….

Softjoin Example - 1

FROM top500,hiTech SELECT * WHERE top500.name~hiTech.name

top500:	*hiTech:*
Abbott Laboratories	ACC CORP
Able Telcom Holding Corp.	ADC TELECOMMUNICATION INC
Access Health, Inc.	ADELPHIA COMMUNICATIONS CORP
Acclaim Entertainment, Inc.	ADT LTD
Ace Hardware Corporation	ADTRAN INC
ACS Communications, Inc.	AIRTOUCH COMMUNICATIONS
ACT Manufacturing, Inc.	AMATI COMMUNICATIONS CORP
Active Voice Corporation	AMERITECH CORP
Adams Media Corporation	APERTUS TECHNOLOGIES INC
Adolph Coors Company	APPLIED DIGITAL ACCESS INC
...	APPLIED INNOVATION INC

A useful scalable similarity metric: IDF weighting plus cosine distance!

How well does TFIDF work?

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

1	$\checkmark$	a_1	b_1
2	$\checkmark$	a_2	b_2
3	$\times$	a_3	b_3
4	$\checkmark$	a_4	b_4
5	$\checkmark$	a_5	b_5
6	$\checkmark$	a_6	b_6
7	$\times$	a_7	b_7
8	$\checkmark$	a_8	b_8
9	$\checkmark$	a_9	b_9
10	$\times$	a_{10}	b_{10}
11	$\times$	a_{11}	b_{11}
12	$\checkmark$	a_{12}	b_{12}

Precision at K: G_K/K

Recall at K: G_K/G

G: # good pairings

G_K: # good pairings in first K

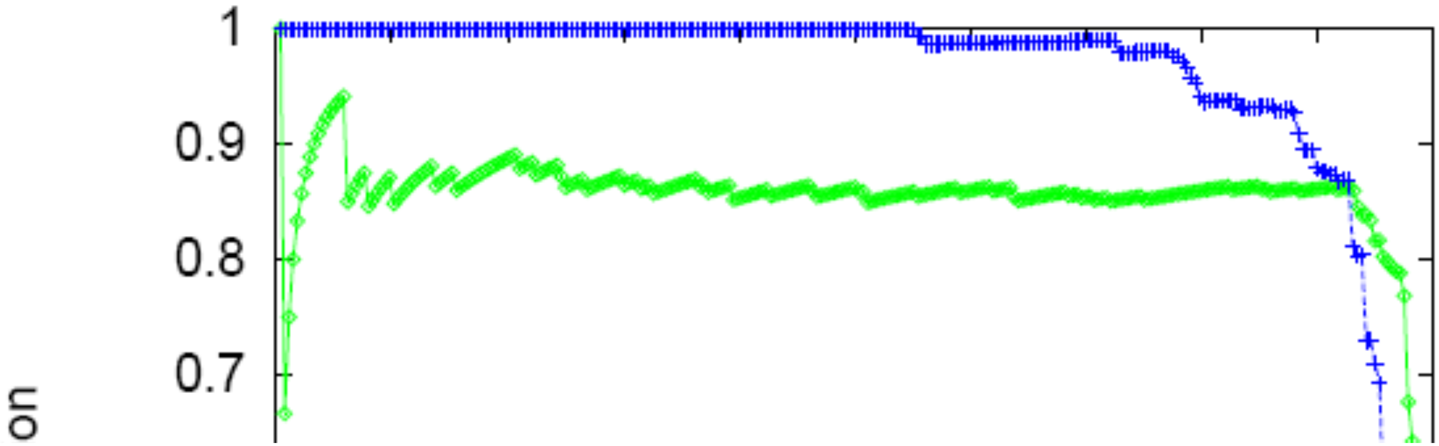


Table VI. Pairs of Names from the Hoovers and Iontech Relations

√	Texas Instruments Incorporated	TEXAS INSTRUMENTS INC
√	The New York Times Company	NEW YORK TIMES CO
√	Campo Electronics, Appliances and Computers, Inc.	CAMPO ELECTRONICS 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 I

Table V. Average Precision for Similarity Joins

Domain	Relations Joined	Average Precision
Movies	MovieLink/Review	100.0%
Animals	IntFact1/SWFact	100.0%
	IntFact2/FWSFact	99.6%
	IntFact3/NMFSFact	97.1%
	Endanger/ParkAnim	95.2%
Birds	IntBirdPic1/DonBirdPic	100.0%
	IntBirdPic2/MBRBirdPic	99.1%
	IntBirdMap/BirdMap	91.4%
	BirdCall/BirdList	95.8%
Businesses	Fodor/Zagrat	99.5%
	HooverWeb/Iontech	84.9%
National Parks	IntPark/Park	95.7%
Computer Games	Demo/AgeList	86.1%

There are refinements to TFIDF distance – eg ones that extend with soft matching at the token level (e.g., softTFIDF)

```
distance is '[JaroWinklerTFIDF:threshold=0.9]'
Pairs: 6806 Correct: 250
Matching time: 0.278
+   1     1.00 |            Agate Fossil Beds NM |           Agate Fossil Beds NM
+   2     1.00 |                     Big Bend NP |                    Big Bend NP
...
+ 194     1.00 |                     Gateway NRA |                    Gateway NRA
+ 195     0.99 |                 Gulf Islands NS |                 Gulf Island NS
+ 196     0.99 |               Rainbow Bridge NM |             Rainbow Bridges NM
+ 197     0.98 | Whiskeytown Shasta Trinity NRA |     Whiskey-Shasta-Trinity NRA
+ 198     0.97 |                 Capitol Reef NP |                Capital Reef NP
+ 199     0.95 |              Timpanogos Cave NM |           Timpanogas Caves NM
+ 200     0.94 |          War in the Pacific NHP |             War in Pacific NHP
+ 201     0.94 |     Chesapeake & Ohio Canal NHP | Chesapeake and Ohio Canal NHP
+ 203     0.92 |                      Saguaro NP |                     Saguaro NM
..
+ 210     0.88 |          Aniakchak NM & NPRES   |                   Aniakchak NM
+ 211     0.86 | National Park Of American Samoa|           NP of American Samoa
..
+ 224     0.76 |       Pu'uhonua a Honaunau NHP |      Pu'uohonua O Honaunau NHP
+ 225     0.75 |      Bering Land Bridge NPRES   | Bering Land Bridge N. Preserve
+ 226     0.75 |      Yukon Charley Rivers NPRES | Yukon-Charley Rivers N. Preserve
...
+ 241     0.69 | Wolf Trap Farm Park for the Performing Arts
                                               |              Wolf Trap Farm Park
+ 242     0.69 | Fredericksburg and Spotsylvania County Battlefields Memorial NMP
                                               | Fredericksburg & Spotsylvania NMP
+ 243     0.69 |            Great Smoky Mtn. NP |       Great Smoky Mountains NP
+ 245     0.67 |               Mount Rushmore NM |        Mount Rushmore N. Mem.
+ 246     0.67 |               Chattahoochee NSR |        Chattahoochee River NRA
...
```

William W. Cohen

Edit | Follow

Carnegie Mellon University

machine learning, information integration, information extraction, intelligent tutoring, natural language processing

Verified email at cs.cmu.edu - Homepage

My profile is public

Change photo

Title	Cited by	Year
Fast Effective Rule Induction WW Cohen Proceedings of the Twelfth International Conference on Machine Learning ...	3367	1995
A comparison of string metrics for matching names and records W Cohen, P Ravikumar, S Fienberg Kdd workshop on data cleaning and object consolidation 3, 73-78	1488	2003
Recommendation as classification: Using social and content-based information in recommendation C Basu, H Hirsh, W Cohen AAAI/IAAI, 714-720	1064	1998

Add | More | 1–20

SOFT JOINS WITH TFIDF: HOW?

Rocchio's algorithm

Many variants of these formulae

$$DF(w) = \#\text{ different docs } w \text{ occurs in}$$

$$TF(w,d) = \#\text{ different times } w \text{ occurs in doc } d$$

…as long as *u(w,d)=0* for words not in *d!*

$$IDF(w) = \frac{|D|}{DF(w)}$$

$$u(w,d) = \log(TF(w,d)+1)\cdot\log(IDF(w))$$

$$\mathbf{u}(d) = \langle u(w_1,d),\ldots,u(w_{|V|},d)\rangle$$

Store only non-zeros in **u**(*d*), so size is O(|*d*|)

$$\mathbf{u}(y) = \alpha\frac{1}{|C_y|}\sum_{d\in C_y}\frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2} - \beta\frac{1}{|D-C_y|}\sum_{d'\in D-C_y}\frac{\mathbf{u}(d')}{\|\mathbf{u}(d')\|_2}$$

But size of **u**(*y*) is O(|n_V|)

$$f(d) = \arg\max_y \frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2}\cdot\frac{\mathbf{u}(y)}{\|\mathbf{u}(y)\|_2}$$

$$\|\mathbf{u}\|_2 = \sqrt{\sum_i u_i^2}$$

TFIDF similarity

$$DF(w) = \#\,\text{different docs } w \text{ occurs in}$$

$$TF(w,d) = \#\,\text{different times } w \text{ occurs in doc } d$$

$$IDF(w) = \frac{|D|}{DF(w)}$$

$$u(w,d) = \log(TF(w,d)+1)\cdot\log(IDF(w))$$

$$\mathbf{u}(d) = \left\langle u(w_1,d),\ldots.,u(w_{|V|},d)\right\rangle$$

$$\mathbf{v}(d) = \frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2}$$

$$sim(\mathbf{v}(d_1),\mathbf{v}(d_2)) = \mathbf{v}(d_1)\cdot\mathbf{v}(d_2) = \sum_w \frac{u(w,d_1)}{\|\mathbf{u}(d_1)\|_2}\frac{u(w,d_2)}{\|\mathbf{u}(d_2)\|_2}$$

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….

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

TFIDF similarity: variant for joins

$$DF(A,w) = \#\text{different docs } w \text{ occurs in from A}$$

$$DF(B,w) = \#\text{different docs } w \text{ occurs in from B}$$

$$TF(w,d) = \#\text{different times } w \text{ occurs in doc } d$$

$$IDF(w,d) = \frac{|C_d|}{DF(C_d,w)}, \text{ where } C_d \in \{A,B\}$$

$$u(w,d) = \log(TF(w,d)+1)\cdot\log(IDF(w,d))$$

$$\mathbf{u}(d) = \langle u(w_1,d),\ldots.,u(w_{|V|},d)\rangle$$

$$\mathbf{v}(d) = \frac{\mathbf{u}(d)}{\|\mathbf{u}(d)\|_2}$$

$$sim(\mathbf{v}(d_1),\mathbf{v}(d_2)) = \mathbf{v}(d_1)\cdot\mathbf{v}(d_2) = \sum_w \frac{u(w,d_1)}{\|\mathbf{u}(d_1)\|_2}\frac{u(w,d_2)}{\|\mathbf{u}(d_2)\|_2}$$

Parallel Inverted Index Softjoin - 1

```
#compute document frequency
docFreq = Group(data, by=lambda(rel,docid,term):(rel,term), reducingTo=ReduceToCount()) \
 | ReplaceEach(by=lambda((rel,term),df):(rel,term,df))

#find total number of docs per relation
ndoc = ReplaceEach(data, by=lambda(rel,docid,term):(rel,docid)) \
      | Distinct() | Group(by=lambda(rel,docid):rel, reducingTo=ReduceToCoun

#unweighted document vectors
udocvec = Join( Jin(data,by=lambda(rel,docid,term):(rel,term)),
                Jin(docFreq,by=lambda(rel,term,df):(rel,term)) ) \
    | ReplaceEach(by=lambda((rel,doc,term),(rel_,term_,df)):(rel,doc,term,df)
    | JoinTo( Jin(ndoc,by=lambda(rel,relCount):rel), by=lambda(rel,doc,term,d
    | ReplaceEach(by=lambda((rel,doc,term,df),(rel_,relCount)):(rel,doc,term,
    | ReplaceEach(by=lambda(rel,doc,term,df,relCount):(rel,doc,term,termWeight(relCount,df)))

#normalizers
sumSquareWeights = ReduceTo(float, lambda accum,(rel,doc,term,weight): accum+weight*weight)
norm = Group( udocvec,
              by=lambda(rel,doc,term,weight):(rel,doc),
              retaining = lambda (rel,doc,term,weight): weight,
              reducingTo=ReduceToSum() ) \
      | ReplaceEach( by=lambda((rel,doc),z):(rel,doc,z))

#normalized document vector
docvec = Join( Jin(norm,by=lambda(rel,doc,z):(rel,doc)),
               Jin(udocvec,by=lambda(rel,doc,term,weight):(rel,doc)) ) \
    | ReplaceEach( by=lambda((rel,doc,z),(rel_,doc_,term,weight)): (rel,doc,term,weight/math.sqrt(z)) )
```

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

Parallel Inverted Index Softjoin - 2

```
# 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( 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,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)
```

What's the algorithm?

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

An alternative TFIDF pipeline

```
def loadDictView(view):
    result = {}
    for (key,val) in GPig.rowsOf(view):
        result[key] = val
    return result
```

```
class TFIDF(Planner):

    D = GPig.getArgvParams()
    data = ReadLines(D.get('corpus','idcorpus.txt')) \
        | Map(by=lambda line:line.strip().split("\t")) \
        | Map(by=lambda (docid,doc): (docid,doc.lower().split())) \
        | FlatMap(by=lambda (docid,words): map(lambda w:(docid,w),words))

    #compute document frequency and inverse doc freq
    docFreq = Distinct(data) \
        | Group(by=lambda (docid,term):term, retaining=lambda(docid,term):docid, reducingTo=ReduceToCount())

    ndoc = Map(data, by=lambda (docid,term):docid) \
        | Distinct() \
        | Group(by=lambda row:'ndoc', reducingTo=ReduceToCount())

    inverseDocFreq = Augment(docFreq, sideview=ndoc, loadedBy=lambda v:GPig.onlyRowOf(v)) \
        | Map(by=lambda((term,df),(dummy,ndoc)):(term,math.log(ndoc/df)))

    #compute unweighted document vectors
    udocvec = Augment(data, sideview=inverseDocFreq, loadedBy=loadDictView) \
        | Map(by=lambda ((docid,term),idfDict):(docid,term,idfDict[term]))

    #normalize
    norm = Group( udocvec, by=lambda(docid,term,weight):docid,
                           retaining=lambda(docid,term,weight):weight*weight,
                           reducingTo=ReduceToSum() )

    docvec = Augment(udocvec, sideview=norm, loadedBy=loadDictView) \
        | Map( by=lambda ((docid,term,weight),normDict): (docid,term,weight/math.sqrt(normDict[docid])))
```

Inverted Index Softjoin – PIG 1/3

```
-- invoke as: pig --param input=id-park --param rel=icepark ... phirl.pig

%default output sim
%default rel a
%default def_par 10

SET default_parallel $def_par;

-- load and tokenize the data as data:{rel,id,str,term}

raw = LOAD 'phirl/$input' AS (rel,docid,keyid,str);
data = FOREACH raw GENERATE rel,docid,FLATTEN(TOKENIZE(LOWER(str))) AS term;

-- compute relation-dependent document frequencies as docfreq:{rel,term,df:int}

docfreq =
  FOREACH (GROUP data by (rel,term))
  GENERATE group.rel AS rel, group.term as term, COUNT(data) as df;

-- find the total number of documents in each relation as ndoc:{rel,c:long}

ndoc1 = DISTINCT(FOREACH data GENERATE rel,docid);
ndoc = FOREACH (GROUP ndoc1 by rel) GENERATE group AS rel, COUNT(ndoc1) AS c;
```

Inverted Index Softjoin – 2/3

```
-- find the un-normalized document vectors as udocvec:{rel.docid,term,weight}
udocvec1 = JOIN data BY (rel,term), docfreq BY (rel,term);
udocvec2 = JOIN udocvec1 BY data::rel, ndoc BY rel;
udocvec =
   FOREACH udocvec2
   GENERATE data::rel, data::docid, data::term,
     LOG(2.0)*LOG(ndoc::c/(double)docfreq::df) AS weight;

-- find the square of the normalizer for each document: norm:{rel,docid,z2:double}

norm1 = FOREACH udocvec GENERATE rel,docid,term,weight*weight as w2;
norm =
   FOREACH (GROUP norm1 BY (rel,docid))
   GENERATE group.rel AS rel, group.docid AS docid, SUM(norm1.w2) AS z2;

-- compute the TFIDF weighted document vectors as: docvec:{rel,docid,term,weight:double}
docvec =
   FOREACH (JOIN udocvec BY (rel,docid), norm BY (rel,docid))
   GENERATE data::rel AS rel, data::docid AS docid, data::term AS term,
      weight/SQRT(z2) as weight;
```

Inverted Index Softjoin – 3/3

```
-- naive algorithm: use all terms for finding potentil matches

docsA = FILTER docvec BY rel=='$rel';
docsB = FILTER docvec BY rel!='$rel';
softjoin1 = JOIN docsA BY term, docsB BY term;
softjoin2 =
   FOREACH softjoin1
   GENERATE docsA::docid AS idA, docsB::docid AS idB, docsA::weight*docsB::weight AS p;
softjoin =
   FOREACH (GROUP softjoin2 BY (idA,idB))
   GENERATE group.idA, group.idB, SUM(softjoin2.p) AS sim;
```

```
-- diagnostic output: look: {sim,[01],idA,idB,str1,str2}

look1 = JOIN topSimPairs BY idA, raw BY docid;
look2 = JOIN look1 BY idB, raw BY docid;
look =
   FOREACH look2
   GENERATE sim, (look1::raw::keyid==raw::keyid ? 1 : 0),
     idA,idB, look1::raw::str AS str1,raw::str AS str2;

STORE look INTO 'phirl/$output';
```

Results…..

0.99436717611623	1	d00059	d00436	Carl Sandburg Home NHS	Carl Sandburg Home NHS
0.9937688379278058	1	d00354	d00611	Theodore Roosevelt NP	Theodore Roosevelt NP
0.9920648281782544	1	d00286	d00573	Oregon Caves NM	Oregon Caves NM
0.9914077975044103	1	d00274	d00566	New River Gorge NR	New River Gorge NR
0.9881961852455996	1	d00009	d00399	American Memorial Park	American Memorial Park
0.9878514547862078	1	d00154	d00500	George Washington Memorial Parkway	George Washington Me
0.9422676645498852	1	d00376	d00623	War in the Pacific NHP	War in Pacific NHP
0.92307133361005	1	d00323	d00594	Saguaro NP	Saguaro NM
0.8914304226443976	1	d00292	d00577	Pea Ridge NHS	Pea Ridge NMP
0.890829830425262	1	d00200	d00532	Jean Lafitte NHP & NPRES	Jean Lafitte NHP & Preserve
0.8873463623037525	0	d00283	d00570	Obed Wild and Scenic River	Obed Wild & Scenic River
0.8838421147370781	1	d00342	d00606	Sitka NHS	Sitka NHP
0.8838421147370781	1	d00011	d00401	Andersonville NHS	Andersonville NHP
0.8700042867436217	1	d00026	d00413	Bering Land Bridge NPRES	Bering Land Bridge N. Preser
0.8684330615122184	1	d00157	d00643	Glacier Bay NP & NPRES	Glacier Bay NP & Preserve
0.8680495192463105	1	d00339	d00603	Sequoia and Kings Canyon NP	Sequoia & Kings Canyon NP
0.8660286476353838	1	d00267	d00561	National Park Of American Samoa	NP of American Samoa
0.8593112749780314	1	d00210	d00538	Kalaupapa NHP	Kalaupapa NHS
0.8500226387429363	1	d00208	d00536	Johnstown Flood NM	Johnstown Flood N. Mem.
0.8424859579540737	1	d00222	d00646	Lake Clark NP & NPRES	Lake Clark NP & Preserve
0.8398407018438242	1	d00187	d00523	Homestead National Monument of America	Homestead NM of Amer
0.8395526626941698	1	d00230	d00548	Lincoln Boyhood NM	Lincoln Boyhood N. Mem.
0.8390553468895996	1	d00349	d00610	Sunset Crater NM	Sunset Crater Volcano NM
0.8344604123961857	1	d00259	d00559	Mount Rushmore NM	Mount Rushmore N. Mem.
0.8313853772986841	0	d00353	d00611	Theodore Roosevelt Island	Theodore Roosevelt NP
0.8301435671019225	1	d00071	d00444	Chesapeake & Ohio Canal NHP	Chesapeake and Ohio Canal NH
0.82492593280652	1	d00019	d00407	Arkansas Post NM	Arkansas Post N. Mem.
0.8202902347497227	1	d00212	d00644	Katmai NP & NPRES	Katmai NP & Preserve
0.8202902347497227	1	d00098	d00464	Denali NP & NPRES	Denali NP & Preserve
0.7965479702996782	1	d00013	d00402	Aniakchak NM & NPRES	Aniakchak NM
0.7835432589199314	1	d00031	d00417	Big Thicket NPRES	Big Thicket N. Preserve
0.7835432589199314	1	d00028	d00415	Big Cypress NPRES	Big Cypress N. Preserve

```
raw = LOAD 'phirl/$input' AS (rel,docid,keyid,str);
data = FOREACH raw GENERATE rel,docid,FLATTEN(TOKENIZE(LOWER(str))) AS term;

-- compute relation-dependent document frequencies as docfreq:{rel,term,df:int}

docfreq =
  FOREACH (GROUP data by (rel,term))
  GENERATE group.rel AS rel, group.term as term, COUNT(data) as df;

-- find the total number of documents in each relation as ndoc:{rel,c:long}

ndoc1 = DISTINCT(FOREACH data GENERATE rel,docid);
ndoc = FOREACH (GROUP ndoc1 by rel) GENERATE group AS rel, COUNT(ndoc1) AS c;

-- find the un-normalized document vectors as udocvec:{rel.docid,term,weight}
udocvec1 = JOIN data BY (rel,term), docfreq BY (rel,term);
udocvec2 = JOIN udocvec1 BY data::rel, ndoc BY rel;
udocvec =
   FOREACH udocvec2
   GENERATE data::rel, data::docid, data::term,
     LOG(2.0)*LOG(ndoc::c/(double)docfreq::df) AS weight;

-- find the square of the normalizer for each document: norm:{rel,docid,z2:double}

norm1 = FOREACH udocvec GENERATE rel,docid,term,weight*weight as w2;
norm =
   FOREACH (GROUP norm1 BY (rel,docid))
   GENERATE group.rel AS rel, group.docid AS docid, SUM(norm1.w2) AS z2;

-- compute the TFIDF weighted document vectors as: docvec:{rel,docid,term,weight:double}
docvec =
   FOREACH (JOIN udocvec BY (rel,docid), norm BY (rel,docid))
   GENERATE data::rel AS rel, data::docid AS docid, data::term AS term,
      weight/SQRT(z2) as weight;

fs -rmr phirl/docvec
STORE docvec INTO 'phirl/docvec';

-- naive algorithm: use all terms for finding potentil matches

docsA = FILTER docvec BY rel=='$rel';
docsB = FILTER docvec BY rel!='$rel';
softjoin1 = JOIN docsA BY term, docsB BY term;
softjoin2 =
   FOREACH softjoin1
   GENERATE docsA::docid AS idA, docsB::docid AS idB, docsA::weight*docsB::weight AS p;
softjoin =
   FOREACH (GROUP softjoin2 BY (idA,idB))
   GENERATE group.idA, group.idB, SUM(softjoin2.p) AS sim;
```

Making the algorithm smarter….

Inverted Index Softjoin - 2

```
-- naive algorithm: use all terms for finding potentil matches

docsA = FILTER docvec BY rel=='$rel';
docsB = FILTER docvec BY rel!='$rel';
softjoin1 = JOIN docsA BY term, docsB BY term;
softjoin2 =
   FOREACH softjoin1
   GENERATE docsA::docid AS idA, docsB::docid AS idB, docsA::weight*docsB::weight AS p;
softjoin =
   FOREACH (GROUP softjoin2 BY (idA,idB))
   GENERATE group.idA, group.idB, SUM(softjoin2.p) AS sim;
```

we should make a smart choice about which terms to use

Adding heuristics to the soft join - 1

```
-- compute maximum weight for rel2docs as: maxweight2:{term,weight}

maxweightB =
   FOREACH (GROUP docsB BY (rel,term))
   GENERATE group.term AS term, MAX(docsB.weight) AS weight;

-- augment the docvecs for rel1 with maxweight2 and docfreq information to get
-- augdocsA: {rel,docid,term, w,df,maxw,score}

docfreqB = FILTER docfreq BY rel!='$rel';
augdocsA1 = JOIN docsA BY term, docfreqB BY term, maxweightB BY term;
augdocsA =
   FOREACH augdocsA1
   GENERATE docsA::rel, docsA::docid, docsA::term, docsA::weight AS w,
     docfreqB::df AS df, maxweightB::weight AS maxw,
     docsA::weight*maxweightB::weight AS score;

-- filter out useful terms to join on, using the info in augdocsA.
-- the heuristics used here are:
--- (1) only use top K by maxscore w/in each document;
--- (2) filter by df<=maxDF
--- (3) filter by score>=minscore

usefulTerms1 =
   FOREACH (GROUP augdocsA BY (rel,docid))
   GENERATE group, TOP($top_k,6,augdocsA) AS top;
usefulTerms2 =
   FOREACH usefulTerms1 {
      filteredTop = FILTER top BY (df<=$max_df) AND score>$min_sim;
      topTerms = FOREACH filteredTop GENERATE term;
      GENERATE flatten(topTerms);
   };
usefulTerms = DISTINCT usefulTerms2;
```

Adding heuristics to the soft join - 2

```
-- use the restricted sets of terms to get candidate pairs

pairs1 = JOIN usefulTerms BY term, docsA BY term, docsB BY term;
pairs2 = FOREACH pairs1 GENERATE docsA::docid AS idA, docsB::docid AS idB;
pairs = DISTINCT pairs2;
-- STORE pairs INTO 'phirl/pairs';

softjoin1 = JOIN pairs BY idA, docsA by docid;
softjoin2 = JOIN softjoin1 BY (idB,term), docsB by (docid,term);
softjoin3 =
   FOREACH softjoin2
   GENERATE idA, idB, docsA::term AS term, docsA::weight*docsB::weight AS p;
softjoin =
   FOREACH (GROUP softjoin3 BY (idA,idB))
   GENERATE group.idA, group.idB, SUM(softjoin3.p) AS sim;
```

```
docsA = FILTER docvec BY rel=='$rel';
docsB = FILTER docvec BY rel!='$rel';

-- compute maximum weight for rel2docs as: maxweight2:{term,weight}

maxweightB =
   FOREACH (GROUP docsB BY (rel,term))
   GENERATE group.term AS term, MAX(docsB.weight) AS weight;

-- augment the docvecs for rel1 with maxweight2 and docfreq information to get
-- augdocsA: {rel,docid,term, w,df,maxw,score}

docfreqB = FILTER docfreq BY rel!='$rel';
augdocsA1 = JOIN docsA BY term, docfreqB BY term, maxweightB BY term;
augdocsA =
   FOREACH augdocsA1
   GENERATE docsA::rel, docsA::docid, docsA::term, docsA::weight AS w,
     docfreqB::df AS df, maxweightB::weight AS maxw,
     docsA::weight*maxweightB::weight AS score;
usefulTerms1 =
   FOREACH (GROUP augdocsA BY (rel,docid))
   GENERATE group, TOP($top_k,6,augdocsA) AS top;
usefulTerms2 =
   FOREACH usefulTerms1 {
      filteredTop = FILTER top BY (df<=$max_df) AND score>$min_sim;
      topTerms = FOREACH filteredTop GENERATE term;
      GENERATE flatten(topTerms);
   };
usefulTerms = DISTINCT usefulTerms2;

pairs1 = JOIN usefulTerms BY term, docsA BY term, docsB BY term;
pairs2 = FOREACH pairs1 GENERATE docsA::docid AS idA, docsB::docid AS idB;
pairs = DISTINCT pairs2;
-- STORE pairs INTO 'phirl/pairs';

softjoin1 = JOIN pairs BY idA, docsA by docid;
softjoin2 = JOIN softjoin1 BY (idB,term), docsB by (docid,term);
softjoin3 =
   FOREACH softjoin2
   GENERATE idA, idB, docsA::term AS term, docsA::weight*docsB::weight AS p;
softjoin =
   FOREACH (GROUP softjoin3 BY (idA,idB))
   GENERATE group.idA, group.idB, SUM(softjoin3.p) AS sim;
```