WestGrid Webinar

Text Parsing and Matching with High Performance Computing Resources

Ian Percel

February 06, 2019

Outline

Outline

What problems are we discussing here?

- Turning unstructured or lightly structured data into clean, highly-structured data
- Merging distinct data sources that refer to the same entities
- Examples:
    - linking property tax records to sales records by address
    - identifying an item by matching descriptions in two different recording systems
    - associating information from two different surveys of the same group of people by matching name

A Framework for Text Modelling

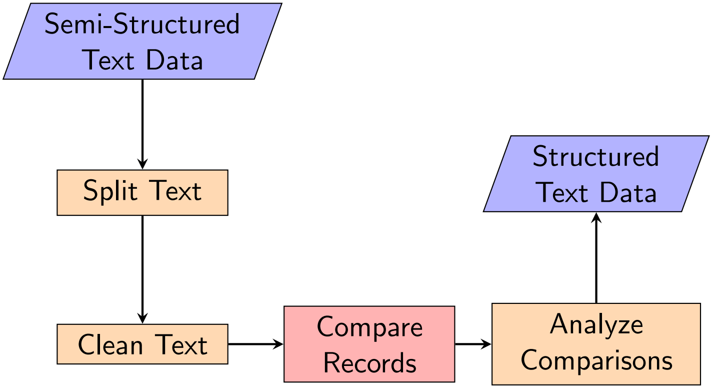


Figure 1: Four stage text analysis diagram

Defining the framework

- Split text = often a string can be broken into logical chunks
- Clean text = synonyms, abbreviations etc can be standardized
- Compare Records = most problems involve approximate relationships, these need to be measured in a consistent way
    - quantitative similarity makes automation easier
    - contextualized meaning is hard to represent [5]
- Analyze Comparisons = ML gives standard tools for turning comparison data into practical decisions

Practical Archival Data Analysis

- To illustrate these ideas we will take a classic example
- In historical records, reliable standardization is unheard of
- For more than one data source, tables must be "joined" together using criteria for imperfect matching

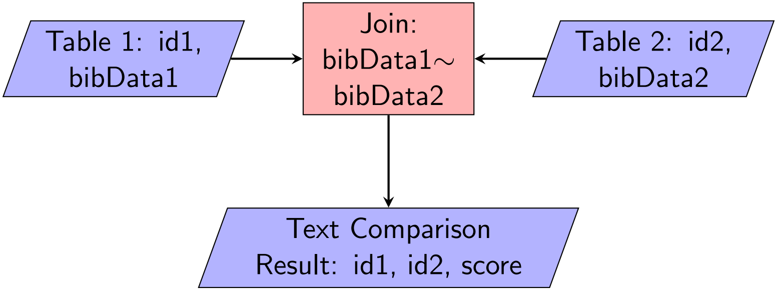


Figure 2: Table Join

Bibliographic Data

- To make our example manageable we will use a benchmark matching data set taken from the Database Group at University of Leipzig
- `https://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution`
- Sources: DBLP-Scholar
- Sample:

```
"id","title","authors","venue","year"
"aKcZKwvwbQwJ","11578 Sorrento Valley Road","QD Inc","San Diego,",
"conf/sigmod/AbadiC02","Visual COKO: a debugger for query optimizer development","D Abadi, M
Cherniack","SIGMOD Conference",2002
"ixKfiTHoaDoJ","Initiation of crazes in polystyrene","AS Argon, JG Hannoosh","Phil. Mag,",
"DMhfVNSDYD4J","The zero multiplicity of linear recurrence sequences","WM Schmidt","to",
"xSv97kdDZU8J","The Photosynthetic Reaction Center","JR Norris, J Deisenhofer","San Diego: Academic,",
"6TKMB5gO9EoJ","Multidimensional similarity structure analysis","I Borg, JC Lingoes",,1987

"f2Lea-RN8dsJ","Visual COKO: a debugger for query optimizer development","DJ Abadi","SIGMOD

Conference,",2002
```

Bibliographic Data - Text Splitting

- This data is formatted into a CSV file and so can (probably) be easily split into a structured format
- More complex partial structuring is possible and can be handled using customized tools (yacc and lexx, pyparsing)
- Taken at face value the sample data above becomes the following table

id	title	authors	venue	year
aKcZKwvwbQwJ	11578 Sorrento Valley Road	QD Inc	San Diego,	
conf/sigmod/AbadiC02	Visual COKO: a debugger for query optimizer development	D Abadi, M Cherniack	SIGMOD Conference	2002
ixKfiTHoaDoJ	Initiation of crazes in polystyrene	AS Argon, JG Hannoosh	Phil. Mag,	
DMhfVNSDYD4J	The zero multiplicity of linear recurrence sequences	WM Schmidt	to	
xSv97kdDZU8J	The Photosynthetic Reaction Center	JR Norris, J Deisenhofer	San Diego: Academic,	
6TKMB5gO9EoJ	Multidimensional similarity structure analysis	I Borg, JC Lingoes		1987
f2Lea-RN8dsJ	Visual COKO: a debugger for query optimizer development	DJ Abadi	SIGMOD Conference,	2002

Bibliographic Data - Further text Splitting

- Questions:
  - Are these columns aligned correctly?
  - Can we split them further to good effect? (for comparison)
  - Do any fields need to be merged? (have they been incorrectly split)
- The answers to questions like these drive an iterative process of segmentation analysis

Why *computational* text analysis?

- Partially structured text is ubiquitous in practical data systems
- Manually cleaning and matching can be impractical. Why?
  - tedious work is error prone
  - life is finite
- These tasks can be partly automated once you understand your data
  - You can create simple rules that cover most cases
  - Often possible to detect cases that require a human

How hard would it be to do manually?

- The bibliographic case has 2600*64200= 167 Million comparisons
- 36 million records in each table (as for a data source about the population of Canada) yield 1296 trillion comparisons
- How do we make this manageable? We need a comparison strategy that can be parallelized

What are our tools for this problem?

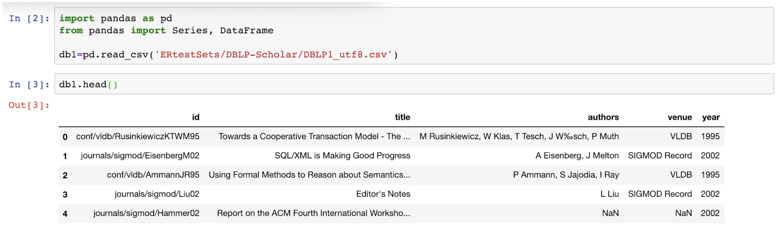
- pandas - A set of tools for data cleaning, structuring and analysis common to most Python data analysis libraries
- re - A regular expressions dialect for Python
- nltk (natural language toolkit) - A standard set of tools for performing text comparisons in Python
- scikit-learn or statsmodels - Standard tools for performing statistical analysis on clean data sets in Python

The idea behind parsing

- Unstructured data is all jumbled together
- We need to break it up into manageable chunks - Splitting or Parsing Text
- We need to standardize the form of text that means the same thing - Cleaning, Scrubbing, or Munging
- We start by introducing data analysis tools for this purpose: pandas and re

The shortest possible introduction to Pandas: Reading in a File

- Access python in Jupyter Notebooks
- `https://syzygy.ca/`
- `import`, `from`, and `as` keywords
- `read_csv` automatically handles the trivial parsing
- `db1.head()` lists top 5 entries



```
In [2]: import pandas as pd
        from pandas import Series, DataFrame

        db1=pd.read_csv('ERtestSets/DBLP-Scholar/DBLP1_utf8.csv')
```

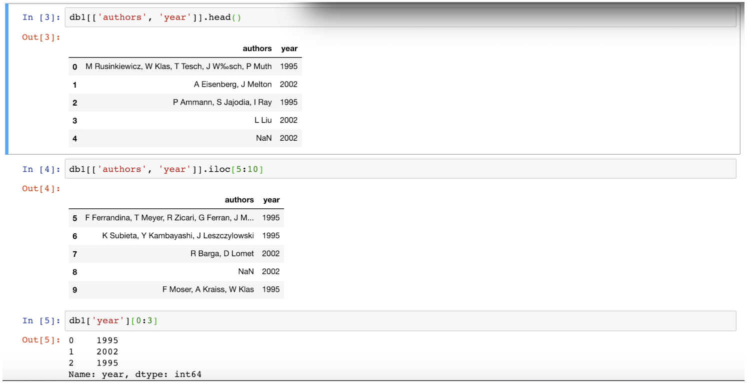
```
In [3]: db1.head()
```

Out[3]:

	id	title	authors	venue	year
0	conf/vldb/RusinkiewiczKTWM95	Towards a Cooperative Transaction Model - The ...	M Rusinkiewicz, W Klas, T Tesch, J W‰sch, P Muth	VLDB	1995
1	journals/sigmod/EisenbergM02	SQL/XML is Making Good Progress	A Eisenberg, J Melton	SIGMOD Record	2002
2	conf/vldb/AmmannJR95	Using Formal Methods to Reason about Semantics...	P Ammann, S Jajodia, I Ray	VLDB	1995
3	journals/sigmod/Liu02	Editor's Notes	L Liu	SIGMOD Record	2002
4	journals/sigmod/Hammer02	Report on the ACM Fourth International Worksho...	NaN	NaN	2002

The shortest possible introduction to Pandas: Examining a DataFrame

- indexing with `df[['col1', 'col2']]`
- examining a subset of rows with `df.iloc[m:n]`
- examining a single column (Series) with `df['col1']`



```
In [3]: dbl[['authors', 'year']].head()
Out[3]:
```

	authors	year
0	M Rusinkiewicz, W Klas, T Tesch, J W‰sch, P Muth	1995
1	A Eisenberg, J Melton	2002
2	P Ammann, S Jajodia, I Ray	1995
3	L Liu	2002
4	NaN	2002

```
In [4]: dbl[['authors', 'year']].iloc[5:10]
Out[4]:
```

	authors	year
5	F Ferrandina, T Meyer, R Zicari, G Ferran, J M...	1995
6	K Subieta, Y Kambayashi, J Leszczylowski	1995
7	R Barga, D Lomet	2002
8	NaN	2002
9	F Moser, A Kraiss, W Klas	1995

```
In [5]: dbl['year'][0:3]
Out[5]: 0    1995
        1    2002
        2    1995
        Name: year, dtype: int64
```

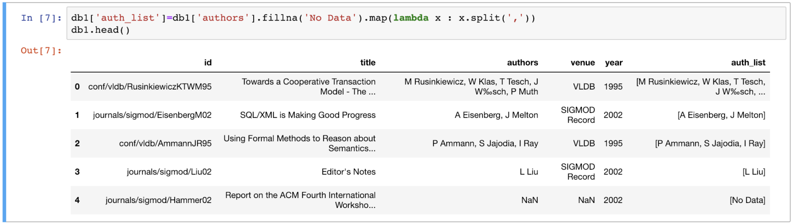
The shortest possible introduction to Pandas: Map

- Operations can be executed efficiently on every row of a Series (or a single column of a DataFrame) using Map
- `map` takes a function as an argument
- To makes a single column lowercase and free of whitespace:

```
df['description']=df['description'].map(str.lower).map(x.strip)
```

The shortest possible introduction to Pandas: Splitting further Columns

- adding a column with `df['newCol']=column`
- splitting string to a list with `string.split('delimiter')`



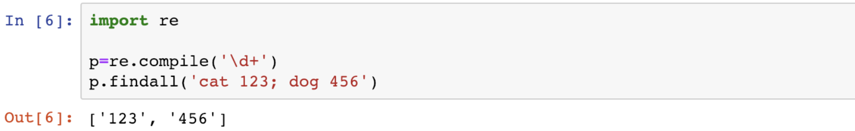
```
In [7]: db1['auth_list']=db1['authors'].fillna('No Data').map(lambda x : x.split(','))
        db1.head()
```

Out[7]:

	id	title	authors	venue	year	auth_list
0	conf/vldb/RusinkiewiczKTWM95	Towards a Cooperative Transaction Model - The ...	M Rusinkiewicz, W Klas, T Tesch, J W‰sch, P Muth	VLDB	1995	[M Rusinkiewicz, W Klas, T Tesch, J W‰sch, ...
1	journals/sigmod/EisenbergM02	SQL/XML is Making Good Progress	A Eisenberg, J Melton	SIGMOD Record	2002	[A Eisenberg, J Melton]
2	conf/vldb/AmmannJR95	Using Formal Methods to Reason about Semantics...	P Ammann, S Jajodia, I Ray	VLDB	1995	[P Ammann, S Jajodia, I Ray]
3	journals/sigmod/Liu02	Editor's Notes	L Liu	SIGMOD Record	2002	[L Liu]
4	journals/sigmod/Hammer02	Report on the ACM Fourth International Worksho...	NaN	NaN	2002	[No Data]

The shortest possible introduction to re: Pattern Matching

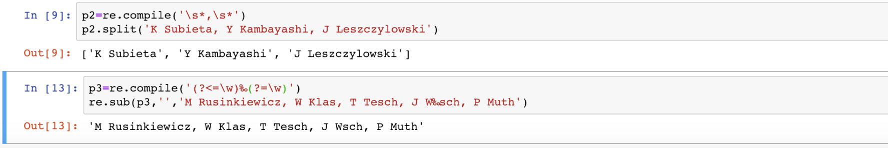
- Regular Expressions is a notation for describing text
- re uses regular expressions to systematically analyze text and find matches to complex structures
- `p=re.compile('\d+')` creates a pattern to match (in this case any whole number)
- `p.findall(dataToSearch)` returns a list of matches to the pattern that can be found in the dataToSearch



```
In [6]: import re

        p=re.compile('\d+')
        p.findall('cat 123; dog 456')

Out[6]: ['123', '456']
```

The shortest possible introduction to re: splitting and Find & Replace

- enhanced splitting `p=re.compile('\s+,\s+')`
- `p.split(dataToSplit)`
- find and replace `re.sub(p,'newString',dataToReplacePatternIn)`



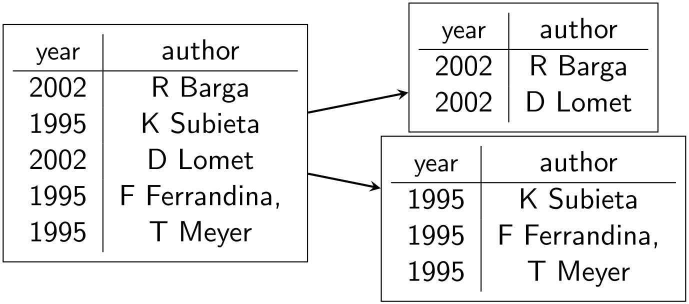
```
In [9]:  p2=re.compile('\s*,\s*')
          p2.split('K Subieta, Y Kambayashi, J Leszczylowski')

Out[9]:  ['K Subieta', 'Y Kambayashi', 'J Leszczylowski']

In [13]:  p3=re.compile('(?<=\w)‰(?=\w)')
           re.sub(p3,'','M Rusinkiewicz, W Klas, T Tesch, J W‰sch, P Muth')

Out[13]:  'M Rusinkiewicz, W Klas, T Tesch, J Wsch, P Muth'
```

The idea of parallelizing parsing and cleaning

- Everything that we have illustrates so far is good but is serial and individual
- We need a way of thinking about breaking our DataFrame into manageable chunks
- Once we have a strategy for that, we can apply it to the creation of independent jobs

Split-Apply-Combine as an overall strategy

- Similar to (but more general than) GROUP BY in SQL
- General tool for bulk changes
- The splitting step breaks data into groups using any column (including the row number) [6]
- This can be accomplished using `df.groupby('year')`



year	author
2002	R Barga
1995	K Subieta
2002	D Lomet
1995	F Ferrandina,
1995	T Meyer

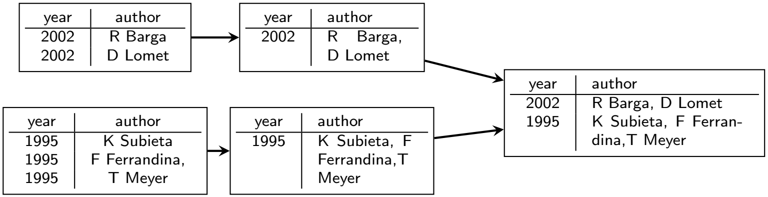
year	author
2002	R Barga
2002	D Lomet

year	author
1995	K Subieta
1995	F Ferrandina,
1995	T Meyer

Figure 3: Table Split/Fork

Split-Apply-Combine in more detail

- Groups produced by the split can be individually transformed by an arbitrary function [6]
- This is the essence of Apply (the DataFrame extension of Map)
- The result is combined back into a single DataFrame



year	author
2002	R Barga
2002	D Lomet

year	author
2002	R Barga, D Lomet

year	author
1995	K Subieta
1995	F Ferrandina,
1995	T Meyer

year	author
1995	K Subieta, F Ferrandina,T Meyer

year	author
2002	R Barga, D Lomet
1995	K Subieta, F Ferran-dina,T Meyer

Figure 4: Table Apply + Combine for concatenation

All of this is performed by a single Python interpreter on a single machine.

Split as a parallelization scheme

- The idea presented in the last slides can be generalized beyond pandas
- By splitting the table efficiently we are producing smaller data sets that can be treated separately
- In an HPC context this lends itself to parallelization
- Each data set can be handled on a separate node once the data has been segregated on the basis of a key column

Apply as a parallel job instruction set

- Although this is overkill for a few rows of concatenation, text cleaning often involves protracted parsing and synonym normalization
- These more complex operations can be used to produce derived forms of the text that are more suited to analysis
- We can write the data blocks/groups to separate files
- Then, by saving the function `f` that you would use in `groupedDF.apply(f)` to a separate python script it can be used as an instruction set to run on each job in a separate python interpreter

Combine as output preparation

- Finally, a script can be used to reassemble the separately processed files into a single new DataFrame
- This can be accomplished by:
  - creating an empty DataFrame with a fixed list of columns `df=DataFrame({'colName1':[], 'colName2':[]})`
  - iteratively loading the data from the finished processing jobs to a temporary DataFrame, say `dfTemp=pd.read_csv(file_jobNum)`
  - using `df=pd.concat([df,dfTemp])` to append the new data to the bottom of the empty dataFrame

Amdahl's Law applied to Split-Apply-Combine

- In a perfect world, parallel acceleration of a computation by using 1000 cores takes $\frac{1}{1000}$ the amount of time
- In practice, every process has serial components that weigh down the process by introducing non-parallelizable steps [2]
- In the Split-Apply-Combine inspired approach given here, the Split and Combine steps are both serial as they involve breaking a single DataFrame into pieces or assembling it back from them and they are both more expensive when used to construct jobs to run in parallel than when they are are done using the pandas on a single node.
- Since parallelization imposes a burden, we need to justify that the row by row computation is expensive enough to warrant the additional cost

Amdahl's Law applied to Split-Apply-Combine

Main Point:

- For simple calculations (like setting all letters to lowercase) that can be done quickly in the first place or small data sets, we are unlikely to see a good return on using HPC.
- A large enough data set with easy splitting and difficult calculations on each row will benefit significantly from using HPC.

The idea of Text Comparison

- Once each source is individually clean, text is ready to be compared across data sources
- Comparison consists of designing and applying a measurement model to emphasize the most important features of our text data
- Ultimately, we will need to come up with an efficient way of splitting the problem into smaller pieces that can be solved in parallel

What computational text comparison is

- Repeatable (the code used to do it is a well-defined procedure)
- Scalable (can be done quickly to large data sets)
- Literal (the negative aspect of Repeatable, it doesn't add anything that isn't already part of the metric you design) [1]
- Relies on selective similarity rather than holistic comparison [5]

What text comparison is *not*

- Model-independent (every metric relies on a theory of the semantic content of your data) [7] [4]
- Adaptive to new context cues (algorithmic processing can't adjust in the way that a human would do while reading)
- Human-feedback independent (if you want it to be any good, hard cases need to be corrected manually and incorporated)

Quantitative Comparisons rely on Selective Similarity

- Selective similarity = isolation of features of text that must agree from features that can vary without significantly changing the content
- Returning to a familiar example of two alternate representations of the same publication:

title	authors	venue	year
'Visual COKO: a debugger for query optimizer development '	['D Abadi','M Cherniack']	'SIGMOD Conference'	2002
'Visual COKO: a debugger for query optimizer development '	['DJ Abadi']	'SIGMOD Conference,'	2002

- presence or absence of comma in 'SIGMOD Conference'doesn't change the likely referent
- 'D Abadi'is probably 'DJ Abadi'given all of the other matching terms
- absence of 'M Cherniack'as second author probably shouldn't significantly impact match given that one author does match

Simple Example of Similarity: Bag Distance

- one way of looking at string similarity identifies individual words and compares them individually but discards any notion of words order
- d=max(number of words in list 2 but not in list 1, number of words in list 1 but nor in list 2)
- removes any relational character in the comparison
- bad approach if order is very important
- good approach for lists (see author words) and for capturing possible descriptive reorderings (big black cat $\sim$ big cat that is black)

$$\begin{tikzcd}
\boxed{\text{D Abadi} \sim \text{DJ Abadi}} \arrow[dr] & \\
& \boxed{d(\text{['D Abadi','M Cherniack'], ['DJ Abadi']}) = 1} \\
\boxed{\text{M Cherniack} \neq \text{DJ Abadi}} \arrow[ur] &
\end{tikzcd}$$

Simple Example of Similarity: Edit Distance

- Levenshtein distance looks at strings that differ by typographical errors
- distance is the smallest number of single letter changes (deletions, insertions, substitutions) to get from one word to the other
- d_{edit}('DJ Abadi', 'D Abadi') $= 1$
- preserves order but allows for small changes (like an extra comma or an extra middle initial J in a name)

Simple Example of Similarity: Edit Distance

- good for ignoring minor changes ('cat,'$\sim$ 'cat'),
- does nothing for synonyms and introduces real issues with short words ('cat'$\neq$'bat'but the two examples are indistinguishable by edit distance)
- computationally expensive in general and considerably harder to write code for than bag distance
- `from nltk.metrics import *`
- `edit_distance(string1,string2)` gives an efficient implementation

Simple Example of Similarity: Edit Distance+Bag Distance

- Implicitly, we suggested a two level model in our discussion of Author Names
- Compare the list of authors via a bag distance between the parsed sequences of authors in each entry d_{bag}(['D Abadi','M Cherniack'], ['DJ Abadi'])
- We require another measure of distance between individual author names, above we have suggested edit distance
  d_{edit}('DJ Abadi', 'D Abadi') $= 1$
  d_{edit}('DJ Abadi', 'M Cherniack') $= 9$

Simple Example of Similarity: Edit Distance+Bag Distance

- If we introduce a threshold for counting word level matches (say and edit distance of 2 or 3) by fiat, then we will treat the first pair as an exact match and the second pair as no match at all
- This completes the full specification of a combined measure of difference that allows individual typos while recognizing that the list of authors included may differ from record to record
- The resulting measure of difference is typical of how competing concerns are resolved in real record matching
- The result can be computationally intensive for long name lists and long names

What can go wrong?

- If we design our metric to ignore important differences, we can produce an excessive number of false positives
- If we make our metric too sensitive to noise, we can find large differences resulting from linguistically unimportant features
- If we have to compare a very large number of entries, added complexity can translate into big increases in computational time
- These competing concerns can be softened by using multiple metrics that capture different kinds of similarity and then using regression to obtain a combined classifier of matches

The idea of Blocking Keys / Indexes

- We need an efficient way to split off groups of records based on a simple matching criterion
- This strategy allows us to ignore most comparisons immediately and focus on the pairs that have a chance of matching
- Each subset that matches on key can then be sent to a different node for careful analysis
- This only works if the key is well-designed

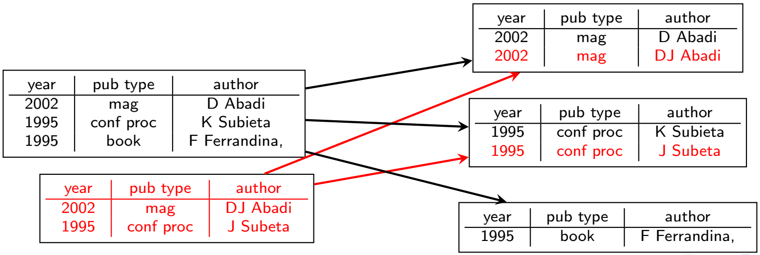
Rough Similarity

- Given the complexity of the (simple) comparisons that were discussed above, there is a strong motive to not perform needless comparisons
- The below pair hardly needs to be compared in detail to recognize the difference
- What is so different about them? No author last names are common, the year is different, the venue has no words in common
- Can we formalize this so that we only do fancy comparison for things that are roughly similar?

id	title	authors	venue	year
conf/sigmod/AbadiC02	Visual COKO: a debugger for query optimizer development	D Abadi, M Cherniack	SIGMOD Conference	2002
ixKfiTHoaDoJ	Initiation of crazes in polystyrene	AS Argon, JG Hannoosh	Phil. Mag,	1995

Indexes and Search Trees

- How much speed up does this actually buy us? In principle, a lot.
- Rough comparisons can be very fast and can be hierarchical so that groups of comparable records can be binned together using a small number of comparisons
- Strong comparisons then only need to be done within the subgroups
- 6 Strong comparisons becomes 2 strong comparisons (this effect grows substantially with table size)



year	pub type	author
2002	mag	D Abadi
1995	conf proc	K Subieta
1995	book	F Ferrandina,

year	pub type	author
2002	mag	DJ Abadi
1995	conf proc	J Subeta

year	pub type	author
2002	mag	D Abadi
2002	mag	DJ Abadi

year	pub type	author
1995	conf proc	K Subieta
1995	conf proc	J Subeta

year	pub type	author
1995	book	F Ferrandina,

Designing a key

- What makes a good key?
- Over-estimate matches
- Small relative to the overall data set
- In the bibliographic case, we could use a combination of
  - publication year
  - type of publication
  - code for a subset of author last names
- key design is a choice with real model consequences for the classification system

Risks of bad key design

- Missed matches
- Subgroups that are too big and so are slow to process
- Key construction time is long compared to direct comparison time

Splitting Data by Key

- The same idea that we applied to cleaning row by row data can be used with comparison
- The benefits are much more dramatic
- By using a node for each blocking key value, we can reap the maximum benefit of our strategy because fine computations happen in parallel
- This produces an enormous parallelization gain even for expensive key calculations and splitting

	year	pub type	author
Node 1	2002	mag	D Abadi
	2002	mag	DJ Abadi

	year	pub type	author
Node 2	1995	conf proc	K Subieta
	1995	conf proc	J Subeta

Now 1 comparison happens in parallel and the problem is done!

Job Construction by Key

- So how do we perform this splitting in practice?
- High-efficiency requires key calculation and then search tree style sorting of rows into bins by unique key value
- In general this is a pretty involved computation. Fortunately, pandas indexing and filtering already implements a version of this
- suppose we have created a function for combining columns in a given row to produce a single key, we can apply it to every row

```
df['key']=df.apply(key_computation_function, axis=1)
keyFrame=df[['key']].drop_duplicates()
for key in keyFrame['key']:
     filteredDF=db1[db1['key']==key]
     #write data subset to pickle or csv
     #with key in filename in a dedicated folder
```

What does a numeric comparison do for you?

- Numbers are simple, they have a clear notion of bigger and smaller
- Text is complicated, meaning in text is even more complicated
- No single number will be satisfactory for describing all of the possible text, even in a simple data set
- However, it is comparatively easy to define rules for making decisions with numbers

Multiplicity of similarity metrics

- We can alleviate some of the inadequacy of quantitative measures of similarity by combining several of them together
- This is like describing a picture by describing each attribute (the first pixel is red, the second pixel is blue, etc.)
- By cleverly combining metrics, we can bring about a kind of cancellation of errors
- This process is always imperfect but may be “good enough” for a given task

Linear Regression on Classification Data

- Human evaluation of examples provide a highly valuable training set for finding "decision boundary"
- Easiest decision making tool is a generalization of a cutoff (e.g. everything less than $1/2$ is close and everything greater than $1/2$ is far)
- We can find the best (linear) rule for combining a given set of metrics together by using least squares regression [3]
- let X be a matrix with every row an input vector of comparison measures and let $\bar{y}$ be a vector where every entry is an outcome corresponding to one of the inputs in X
- the best estimator for a linear model $\hat{Y} = X^T\beta$ is obtained from $\hat{\beta} = (X^TX)^{-1}X^T\bar{y}$
- This can be computed using `OLS(y,X)` from the package *statmodels* in python

Classifying Matches

- Having an optimal rule for calculating a derived metric, we can apply this to new comparison data
  - compute all basic metrics for a given comparison
  - compute the combined metric
  - compare the result to the optimal cutoff
  - if less than the cutoff by enough of a margin, conclude that the pair of records match
  - if greater than the cutoff by enough of a margin, conclude the the pair of records do not match
  - if near the cutoff, send the pair for review by a specialist

Advanced Methods

- Richer Parsing
- Advanced Comparison Measures (Term-Document Frequencies)
- Non-parametric ML models (Nearest Neighbors, Decision Trees) [3]
- Clustering Analysis
- Term-Document Matrix Factorization

[1] Peter Christen. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, 2012.

[2] Hesham El-Rewini and Mostafa Abd-El-Barr. *Advanced Computer Architecture and Parallel Processing*. Wiley, 2007.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Predication*. Springer, 2009.

[4] Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. *Data Quality and Record Linkage Techniques*. Springer, 2007.

[5] Rob Kitchin. Big data, new epistemologies and paradigm shifts. *Big Data and Society*, pages 1–12, 2014.

[6] Wes McKinney. *Python for Data Analysis*. O'Reilly, 2013.

[7] Sunita Sarawagi. *Information Extraction*. now Publishers, 2007.