ENTITY RESOLUTION

CS1951A INTRO TO DATA SCIENCE

Tim Kraska <tim_kraska@brown.edu>

teaching datascience .org

ANNOUNCEMENT

1) Finding Project Team-Members

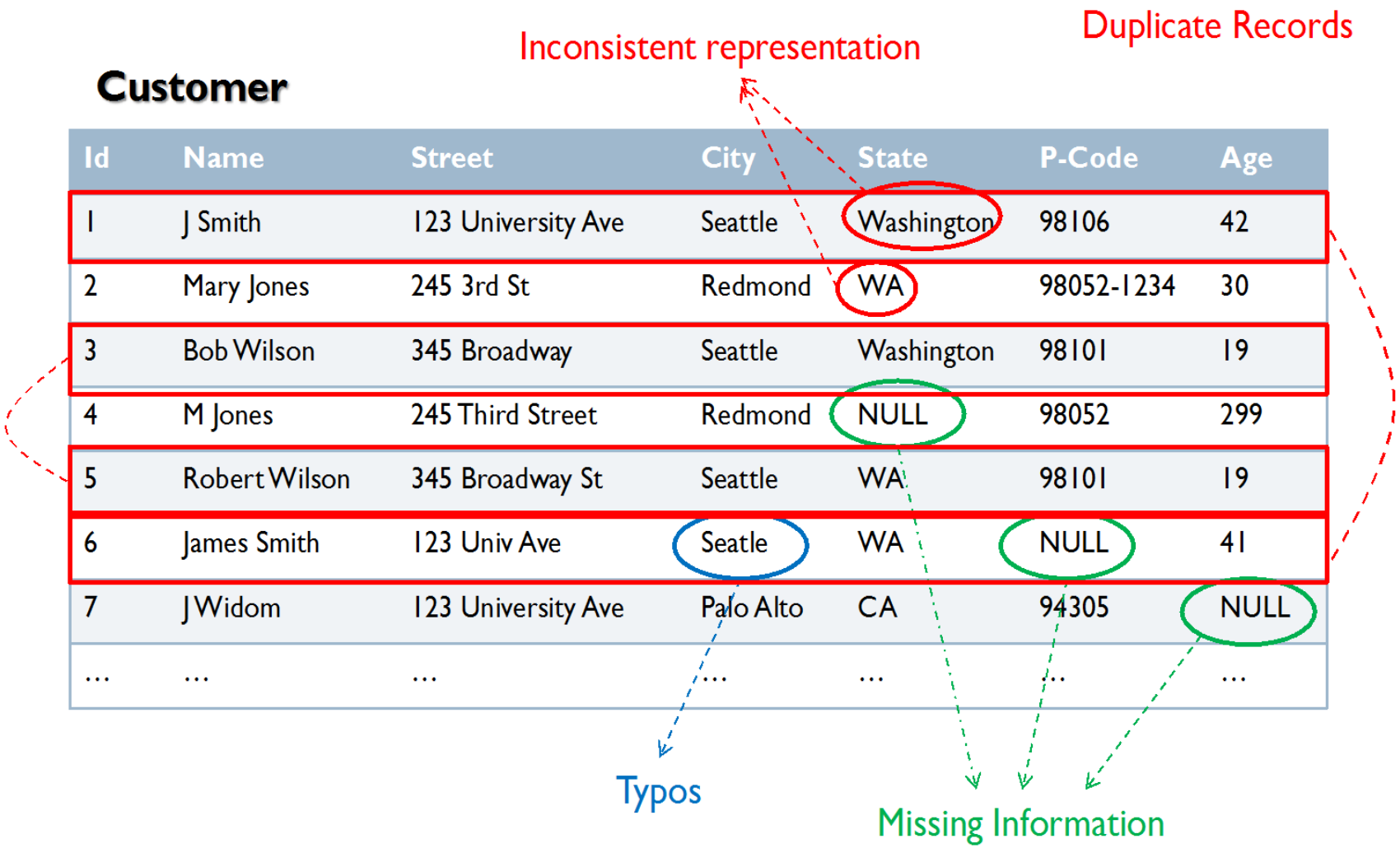
2) Bash Lab

DATA INTEGRATION

Schema Alignment

deduplication, entity clustering, merge/purge, fuzzy match, record linkage, approximate match...

REAL WORLD DATA



REAL WORLD DATA

- How many customers do I have?

```
select count(*)
from customer
```

Wrong answer because of duplicate records!

- How many customers by state?

```
select count(*)
from customer
group by state
```

State	Count
AL	60
…	…
…	…
WA	1200
Washington	50
Wasington	2

What about if you give this data to a ML algorithm?

ENTITY RESOLUTION

"[The] problem of identifying and linking/grouping different manifestations of the same real world object."

Challenges

- Fundamental ambiguity
- Diversity in representations (format, truncation, ambiguity)
- Errors
- Missing data
- Records from different times
- Relationships in addition to equality

TEXT SIMILARITY

Customer

Id	Name	Street	City	State	P-Code	Age
1	J Smith	123 University Ave	Seattle	Washington	98106	42
2	Mary Jones	245 3rd St	Redmond	WA	98052-1234	30
3	Bob Wilson	345 Broadway	Seattle	Washington	98101	19
4	M Jones	245 Third Street	Redmond	NULL	98052	299
5	Robert Wilson	345 Broadway St	Seattle	WA	98101	19
6	James Smith	123 Univ Ave	Seatle	WA	NULL	41
7	J Widom	123 University Ave	Palo Alto	CA	94305	NULL
…	…	…	…	…	…	…

TEXTUAL SIMILARITY

String Similarity function:

- $Sim(string, string) \rightarrow numeric\ value$

A "good" similarity function:

- Strings representing the same concept ⇒ high similarity
- Strings representing different concepts ⇒ low similarity

EDIT DISTANCE

EditDistance(s1, s2):

- Minimum number of edits to transform s1 to s2

Edit:

- Insert a character
- Delete a character
- Substitute a character

Note: EditDistance(s1, s2) = EditDstance(s2, s1)

"Distance" = opposite of similarity

EDIT DISTANCE

EditDistance ("Provdince", "Providence") = 2

Provdince ⟶ Providence ⟶ Providence

EditDistance("Seattle", "Redmond") = 6

Seattle ⟶ Reattle ⟶ Redttle

Redmtle ⟶ Redmole ⟶ Redmone

⟶ Redmond

EDIT DISTANCE PROBLEMS

115th Waterman St., Providence, RI

EditDistance = 1

110th Waterman St., Providence, RI

Waterman Street, Providence, RI

EditDistance = 4

Waterman St, Providence, RI

Character Level vs. Word Level Similarity?

EDIT DISTANCE PROBLEMS

148th Ave NE, Redmond, WA

EditDist = 0

148th Ave NE, Redmond, WA

148th Ave NE, Redmond, WA

EditDist = 4

NE 148th Ave, Redmond, WA

Order sensitive Similarity?

JACCARD SIMILARITY

- **Statistical measure**
- **Originally defined over sets**
- **String = set of words**

$$Jaccard(s1, s2) = \frac{|s1 \cap s2|}{|s1 \cup s2|}$$

- **Range of values = [0,1]**

JACCARD SIMILARITY

148th Ave NE, Redmond, WA

140th Ave NE, Redmond, WA

$$Jaccard = \frac{4}{4+2} \approx 0.66$$

JACCARD SIMILARITY

148th Ave NE, Redmond, WA

NE 148th Ave, Redmond, WA

$$Jaccard = \frac{5}{5} = 1.0$$

CLICKER

What is the Jaccard Similarity between:

iPad Two 16GB WiFi White

iPad 2nd generation 16GB Wifi White

(a) 3 / 8
(b) 4 / 11
(c) 4 / 7

CLICKER: WHICH JACCARD SIMILARITY IS WRONG

A)

Microsoft Corporation

↕ Jaccard = 1/3

Microsoft Corp

B)

Microsoft Corporation

↕ Jaccard = 1/3

Oracle Corporation

C)

Waterman 115 St

↕ Jaccard = 1/4

115 Waterman Street

WHAT CAN WE DO ABOUT?

Microsoft Corporation

Microsoft Corp

Microsoft Corporation

Oracle Corporation

JACCARD SIMILARITY

$$Weight\ Function = wt: Elements \rightarrow \mathbb{R}^{+}$$

$$WtJaccard(s1, s2) = \frac{wt(s1 \cap s2)}{wt(s1 \cup s2)}$$

$$wt(s) = \sum_{e \in s} wt(e)$$

wt("Microsoft") > wt("Corporation")

Wt("Oracle") > wt("Corporation")

IDF WEIGHTED

- IDF: Inverse Document Frequency

$$wt(word) = \log_e \left(\frac{size\ of\ corpus}{frequency(word)} \right)$$

- frequency(word) = defined using some “corpus”:
  - large table of records
  - Wikipedia?

IDF WEIGHTED JACCARD

Microsoft Corporation

WtJaccard = 12.21 / (12.21 + 4.21 + 4.38)

= 12.21/20.8 = 0.59

Microsoft Corp

$$\log_e\left(\frac{1{,}000{,}000}{5}\right)$$

Word	Freq	IDF
Microsoft	5	12.21
Oracle	39	10.15
Corporation	14782	4.21
Corp	12496	4.38

Microsoft Corporation

WtJaccard = 4.21/26.57 = 0.16

Oracle Corporation

Corpus size = 1M records

OTHER SIMILARITY FUNCTIONS

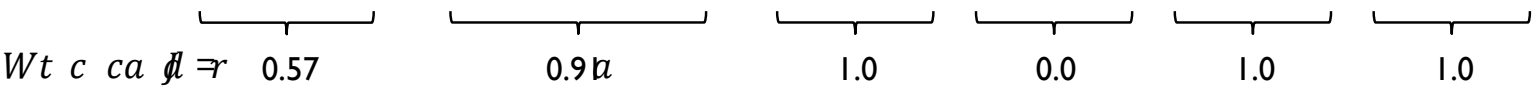
- Affine edit distance
- Cosine similarity
- Hamming distance
- Generalized edit distance
- Jaro distance
- Monge-Elkan distance
- Q-gram
- Smith-Warerman distance
- Soundex distance
- TF/IDF
- …many more

- No universally good similarity function
- Choice of similarity function depends on domains of interest, data instances, etc.

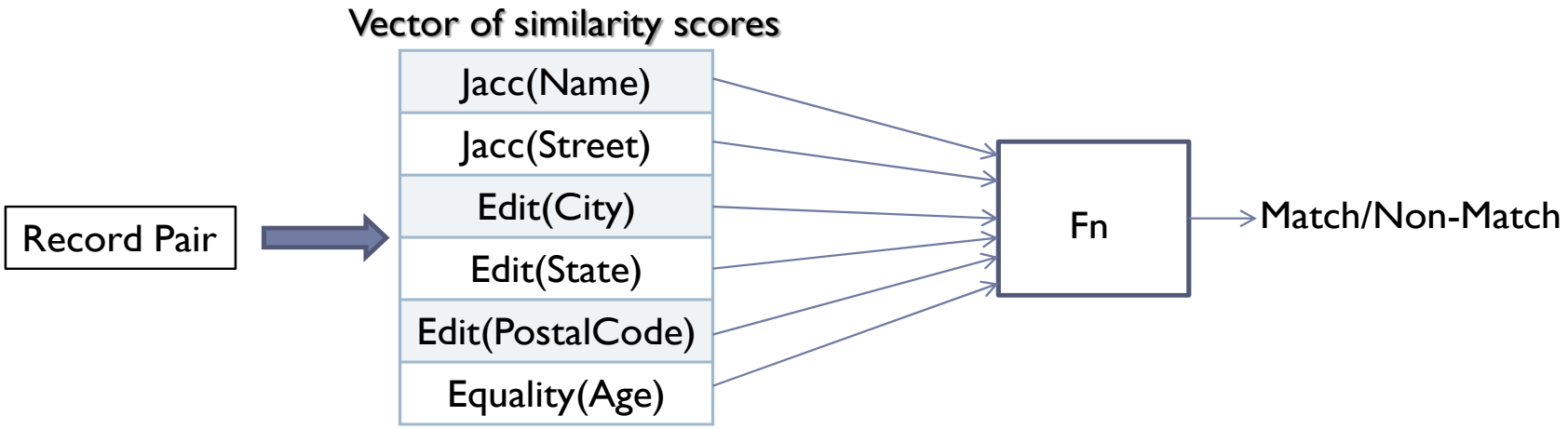
RECORD MATCHING PROBLEMS

Customer

Id	Name	Street	City	State	P-Code	Age
1	J Smith	123 University Ave	Seattle	Washington	98106	42
2	Mary Jones	245 3rd St	Redmond	WA	98052-1234	30
3	Bob Wilson	345 Broadway	Seattle	Washington	98101	19
4	M Jones	245 Third Street	Redmond	NULL	98052	299
5	Robert Wilson	345 Broadway St	Seattle	WA	98101	19
6	James Smith	123 Univ Ave	Seatle	WA	NULL	41
7	J Widom	123 University Ave	Palo Alto	CA	94305	NULL
…	…	…	…	…	…	…



COMBINING SIMILARITY FUNCTIONS



Features

Binary Classification

LEARNING-BASED APPROACH

Bob Wilson	345 Broadway	Seattle	Washington	98101	19
Robert Wilson	345 Broadway St	Seattle	WA	98101	19

Match

B Wilson	123 Broadway	Boise	Idaho	83712	19
Robert Wilson	345 Broadway St	Seattle	WA	98101	19

Non-Match

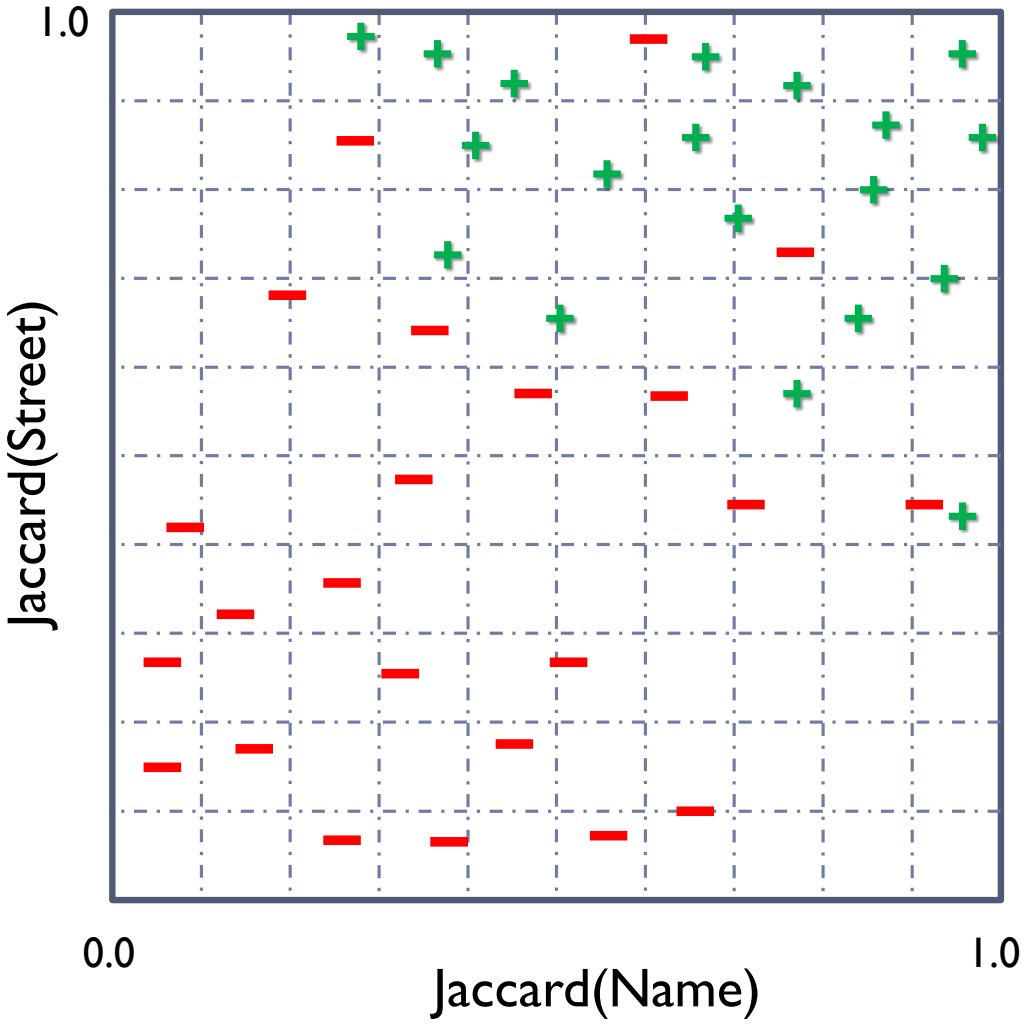
Mary Jones	245 3rd St	Redmond	WA	98052-1234	30
M Jones	245 Third Street	Redmond	NULL	98052	299

Match

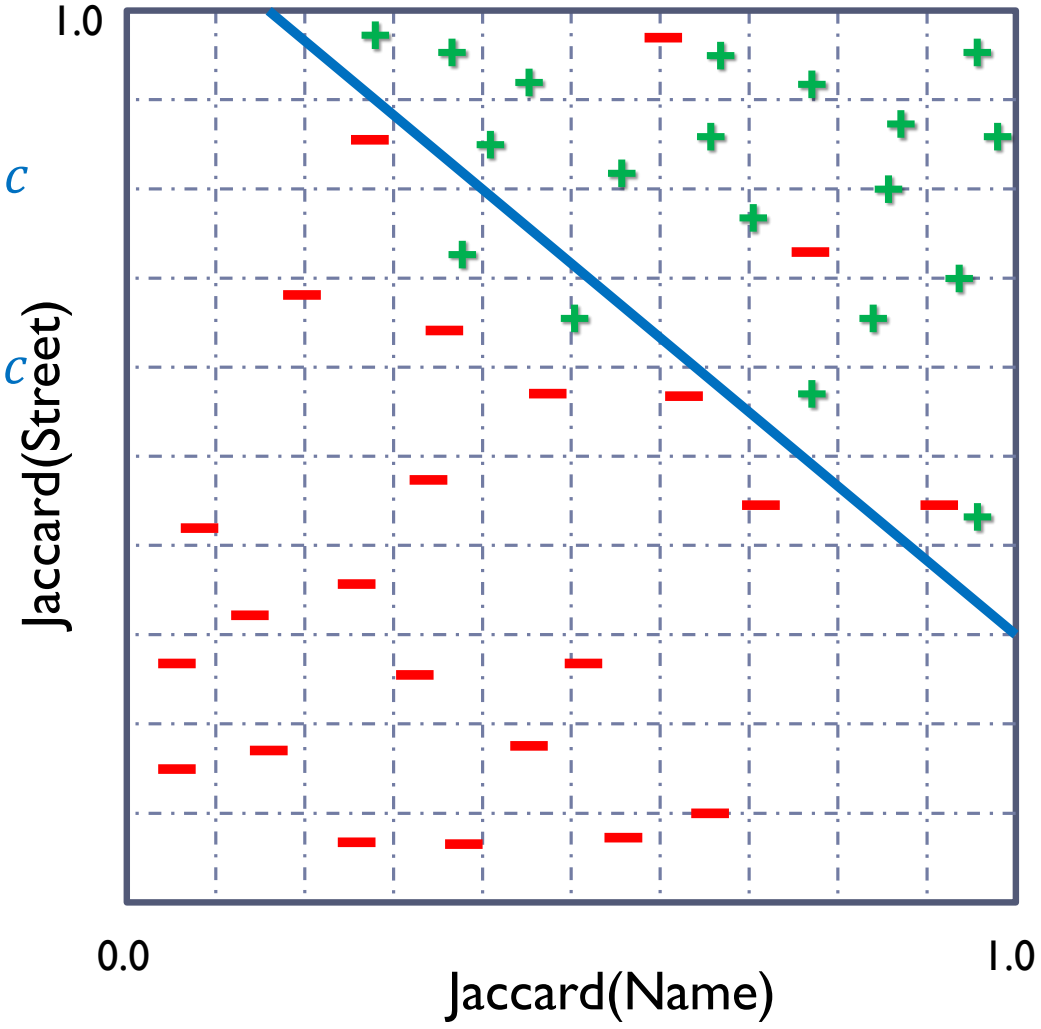
Mary Jones	245 3rd St	Redmond	WA	98052-1234	30
Robert Wilson	345 Broadway St	Seattle	WA	98101	19

Non-Match

LEARNING BASED APPROACH



LEARNING BASED APPROACH



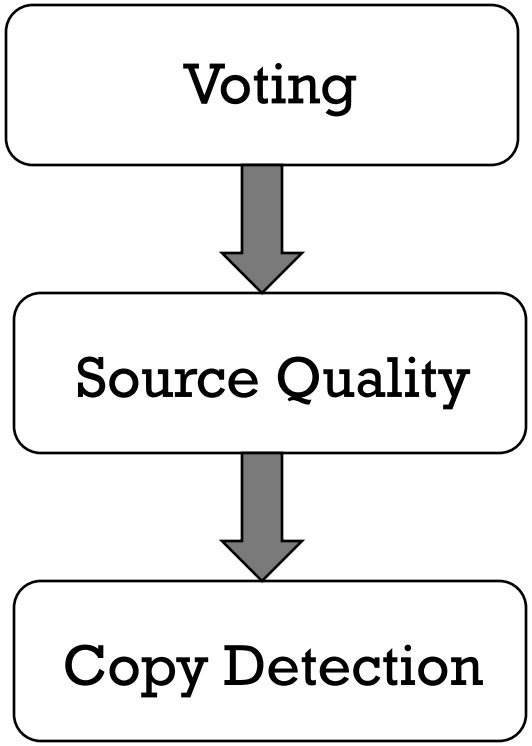
$$0.73\ Ja\ \ (Name)$$
$$+$$
$$0.89\ Ja\ \ (Street)\ \geq\ 1$$

DATA INTEGRATION

DATA FUSION'S THREE COMPONENS

Data fusion: voting + source quality + copy detection

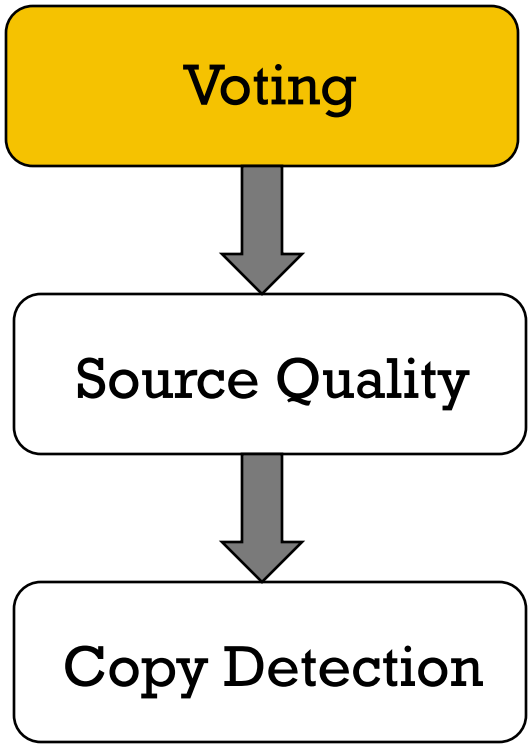
- Resolves inconsistency across diversity of sources



	S1	S2	S3	S4	S5
Jagadish	UM	ATT	UM	UM	UI
Dewitt	MSR	MSR	UW	UW	UW
Bernstein	MSR	MSR	MSR	MSR	MSR
Carey	UCI	ATT	BEA	BEA	BEA
Franklin	UCB	UCB	UMD	UMD	UMD

DATA FUSION'S THREE COMPONENS

Data fusion: voting + source quality + copy detection

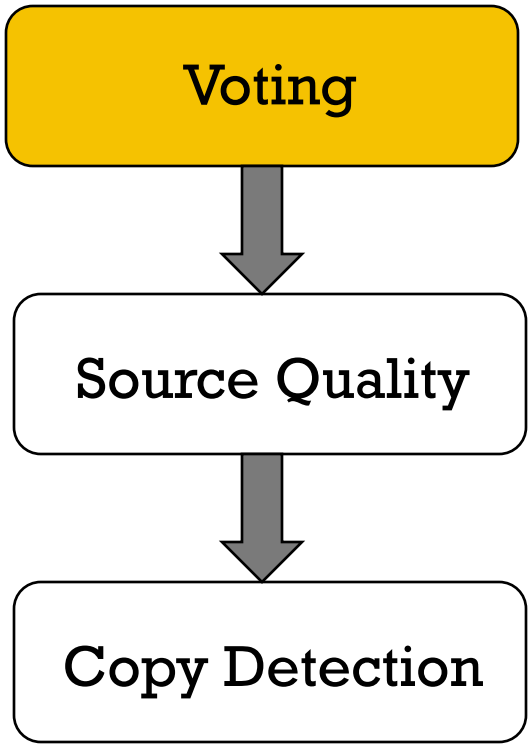


	S1	S2	S3
Jagadish	UM	ATT	UM
Dewitt	MSR	MSR	UW
Bernstein	MSR	MSR	MSR
Carey	UCI	ATT	BEA
Franklin	UCB	UCB	UMD

DATA FUSION'S THREE COMPONENS

Data fusion: voting + source quality + copy detection

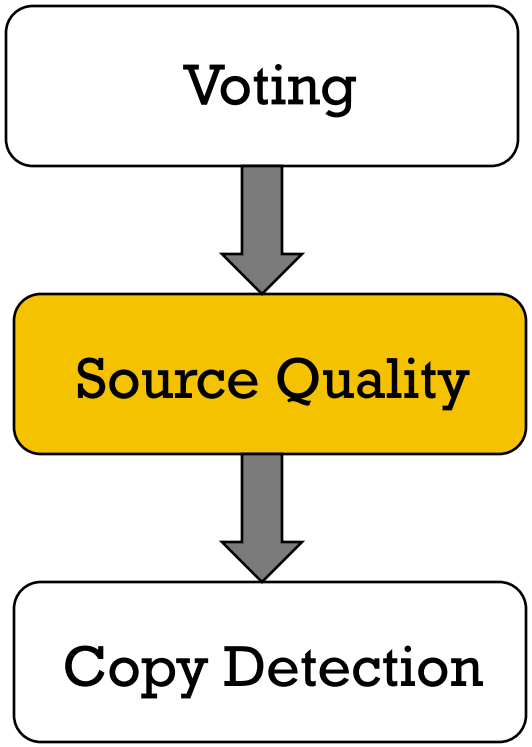
- Supports difference of opinion



	S1	S2	S3
Jagadish	UM	ATT	UM
Dewitt	MSR	MSR	UW
Bernstein	MSR	MSR	MSR
Carey	UCI	ATT	BEA
Franklin	UCB	UCB	UMD

DATA FUSION'S THREE COMPONENS

Data fusion: voting + source quality + copy detection

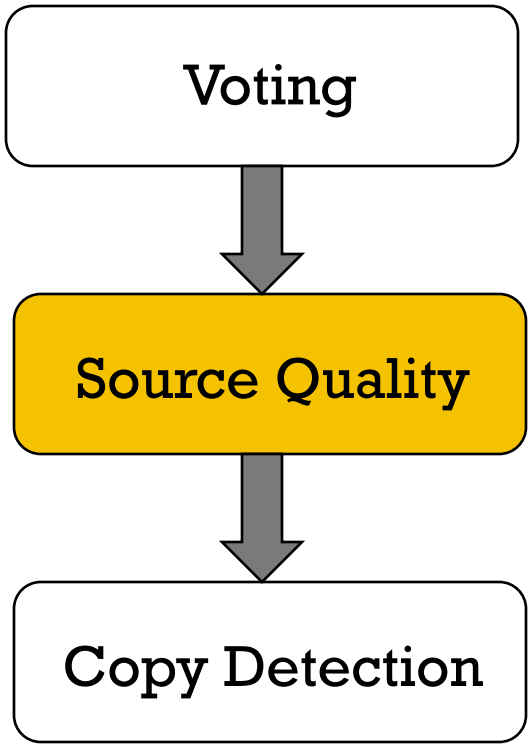


	S1	S2	S3
Jagadish	UM	ATT	UM
Dewitt	MSR	MSR	UW
Bernstein	MSR	MSR	MSR
Carey	UCI	ATT	BEA
Franklin	UCB	UCB	UMD

DATA FUSION'S THREE COMPONENS

Data fusion: voting + source quality + copy detection

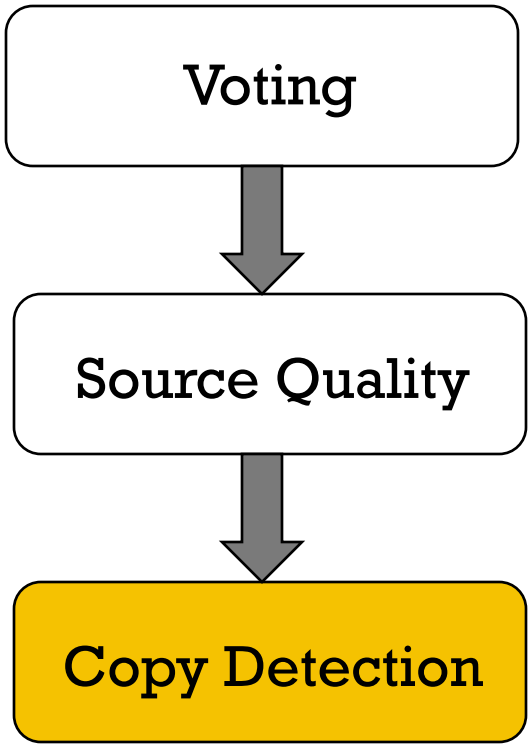
- Gives more weight to knowledgeable sources



	S1	S2	S3
Jagadish	UM	ATT	UM
Dewitt	MSR	MSR	UW
Bernstein	MSR	MSR	MSR
Carey	UCI	ATT	BEA
Franklin	UCB	UCB	UMD

DATA FUSION'S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

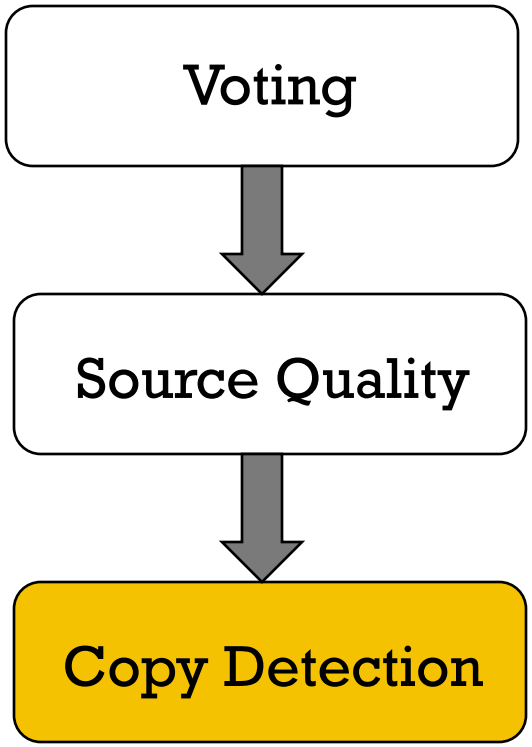


	S1	S2	S3	S4	S5
Jagadish	UM	ATT	UM	UM	UI
Dewitt	MSR	MSR	UW	UW	UW
Bernstein	MSR	MSR	MSR	MSR	MSR
Carey	UCI	ATT	BEA	BEA	BEA
Franklin	UCB	UCB	UMD	UMD	UMD

DATA FUSION'S THREE COMPONENTS

Data fusion: voting + source quality + copy detection

- Reduces weight of copier sources



	S1	S2	S3	S4	S5
Jagadish	UM	ATT	UM	UM	UI
Dewitt	MSR	MSR	UW	UW	UW
Bernstein	MSR	MSR	MSR	MSR	MSR
Carey	UCI	ATT	BEA	BEA	BEA
Franklin	UCB	UCB	UMD	UMD	UMD

DATA INTEGRATION

Schema Alignment

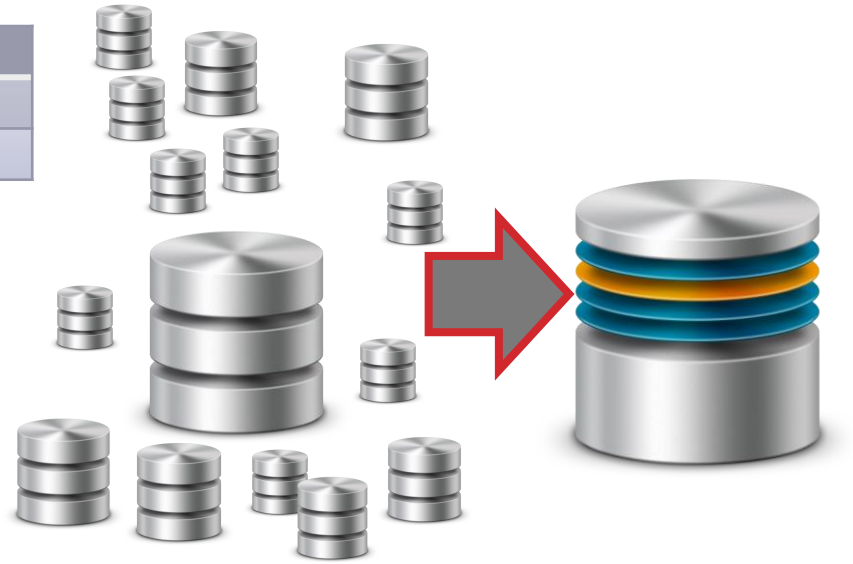
Deduplication, entity clustering, merge/purge, fuzzy match, record linkage, approximate match…

SO FAR: RELATIONAL DATA

A	B	C	D
1	b	b	b
2	s	e	f

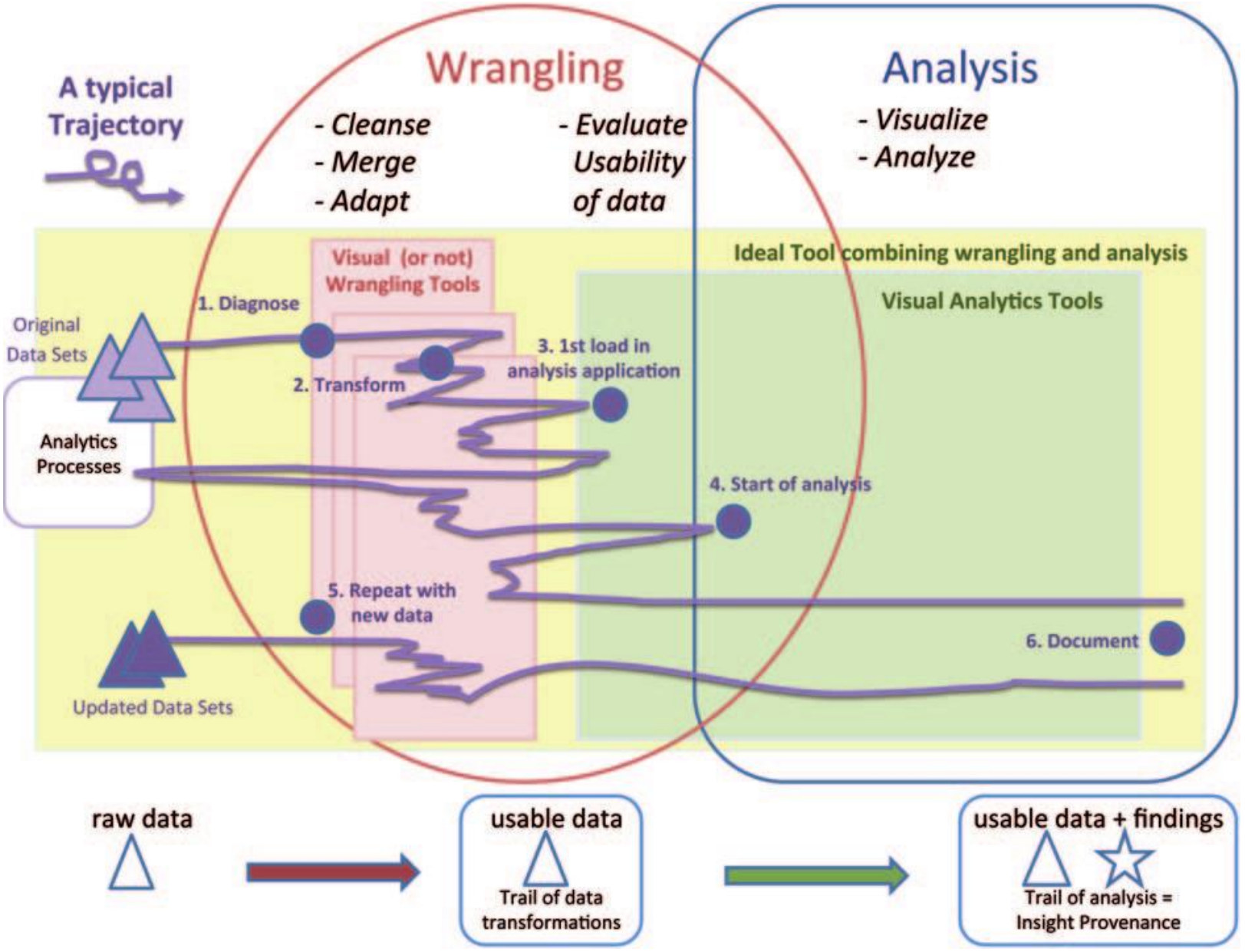
F	A	G
a	1	g
a	1	g

A	B	F
a	2	c
3	2	v



```
{|class="wikitable sortable"
|-
!Appearances
!Team
!Wins
!Losses
!Winning<br />percentage
!Season(s)
|-align=center
| {{Sort|0862|8}}||align=left style="background:#fcc;"|[[Pittsburgh Steelers]]<sup>†</sup><ref group=note name=e />
| 6|| 2|| .750
|align=left|{{Sort|1974 02|'''[[Super Bowl IX|1974]]''',<sup>†</sup> '''[[Super Bowl X|1975]]''',<sup>†</sup> '''[[Super Bowl
XIII|1978]]''',<sup>†</sup> '''[[Super Bowl XIV|1979]]''',<sup>†</sup> [[Super Bowl XXX|1995]],<sup>†</sup> '''[[Super Bowl
XL|2005]]''',<sup>†</sup><ref group=note name=c /> '''[[Super Bowl XLIII|2008]]''',<sup>†</sup> [[Super Bowl
XLV|2010]]<sup>†</sup>}}
|-align=center
|{{Sort|0853|8}}||align=left style="background:#d0e7ff;"|[[Dallas Cowboys]]<sup>*</sup>
| 5|| 3|| .625
|align=left|{{Sort|1970 02|[[Super Bowl V|1970]],<sup>*</sup> '''[[Super Bowl VI|1971]]''',<sup>*</sup> [[Super Bowl
X|1975]],<sup>*</sup><ref group=note name=c /> '''[[Super Bowl XII|1977]]''',<sup>*</sup> [[Super Bowl XIII|
....
```

DATA WRANGLING



[Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011]

THREE EXTREMELY POWERFUL TOOLS

1) grep

Basic syntax:

```
grep 'regexp' filename
```

or equivalently (using UNIX pipelining):

```
cat filename | grep 'regexp'
```

WHAT IS A REGULAR EXPRESSION?

A regular expression (*regex*) describes a set of possible input strings.

Regular expressions descend from a fundamental concept in Computer Science called *finite automata* theory

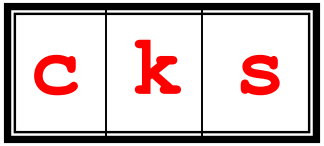
Regular expressions are endemic to Unix

- vi, ed, sed, and emacs
- awk, tcl, perl and Python
- grep, egrep, fgrep
- compilers

REGULAR EXPRESSIONS

The simplest regular expressions are a string of literal characters to match.

The string *matches* the regular expression if it contains the substring.

regular expression → 


```
UNIX Tools rocks.
```

match

```
UNIX Tools sucks.
```

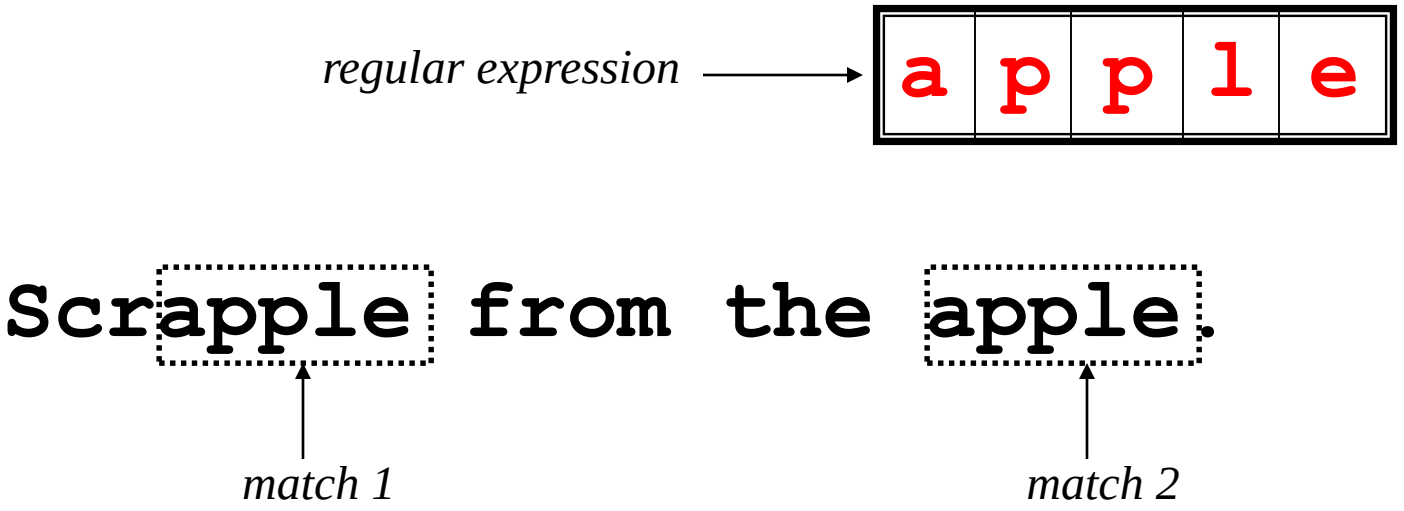
match

```
UNIX Tools is okay.
```

no match

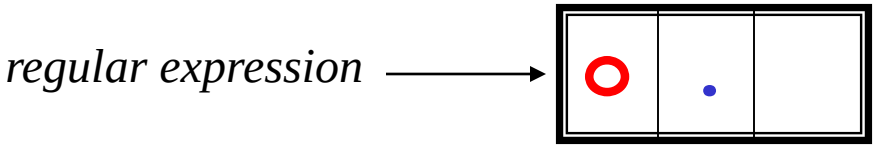
REGULAR EXPRESSIONS

A regular expression can match a string in more than one place.



REGULAR EXPRESSIONS

The . regular expression can be used to match any character.



OR

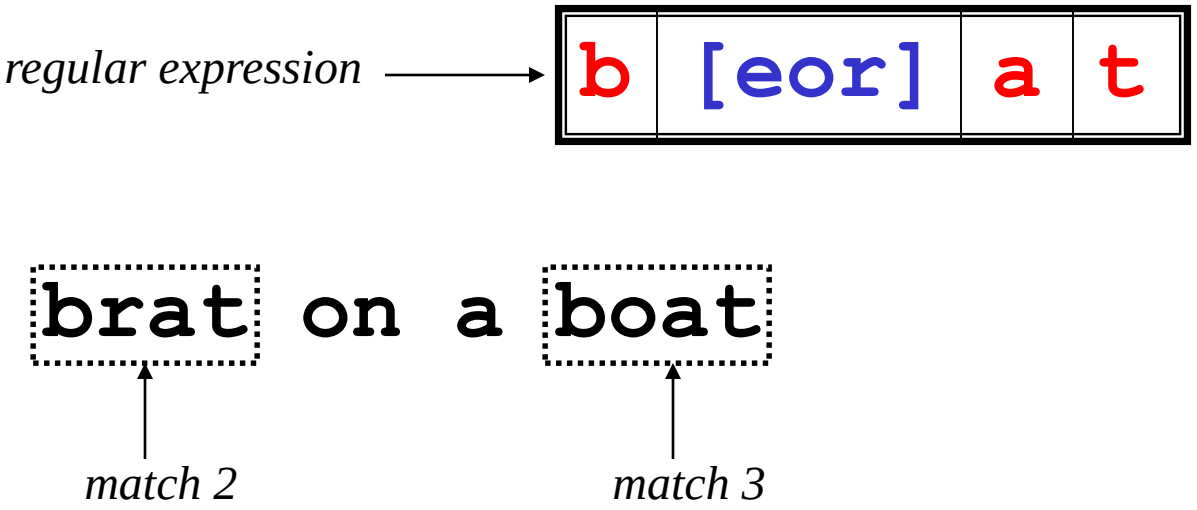
a|b* denotes {ε, "a", "b", "bb", "bbb", ...}

(a|b)* denotes the set of all strings with no symbols other than "a" and "b", including the empty string: {ε, "a", "b", "aa", "ab", "ba", "bb", "aaa", ...}

ab*(c) denotes the set of strings starting with "a", then zero or more "b"s and finally optionally a "c": {"a", "ac", "ab", "abc", "abb", "abbc", ...}

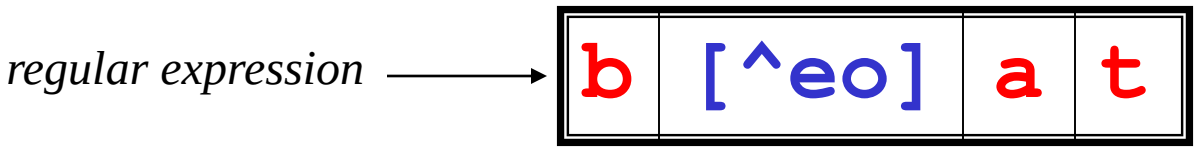
CHARACTER CLASSES

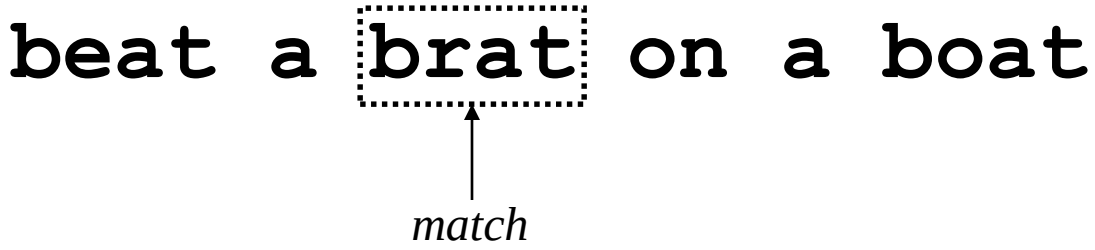
Character classes `[]` can be used to match any specific set of characters.



NEGATED CHARACTER CLASSES

Character classes can be negated with the `[^]` syntax.





MORE ABOUT CHARACTER CLASSES

- `[aeiou]` will match any of the characters `a`, `e`, `i`, `o`, or `u`
- `[kK]orn` will match `korn` or `Korn`

Ranges can also be specified in character classes

- `[1-9]` is the same as `[123456789]`
- `[abcde]` is equivalent to `[a-e]`
- You can also combine multiple ranges
  - `[abcde123456789]` is equivalent to `[a-e1-9]`
- Note that the `-` character has a special meaning in a character class ***but only*** if it is used within a range, `[-123]` would match the characters `-`, `1`, `2`, or `3`

NAMED CHARACTER CLASSES

Commonly used character classes can be referred to by name (*alpha*, *lower, upper, alnum*, *digit*, *punct*, *cntrl*)

Syntax [:*name*:]

- [a-zA-Z] [[:alpha:]]
- [a-zA-Z0-9] [[:alnum:]]
- [45a-z] [45[:lower:]]

Important for portability across languages

ANCHORS

Anchors are used to match at the beginning or end of a line (or both).

^ means beginning of the line

$ means end of the line

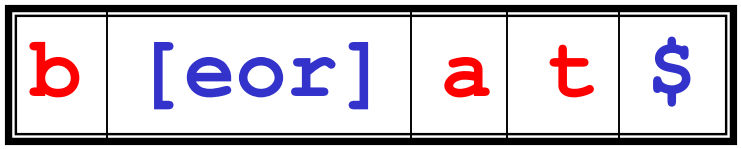
regular expression →

beat a brat on a boat

match

---

regular expression →



beat a brat on a boat

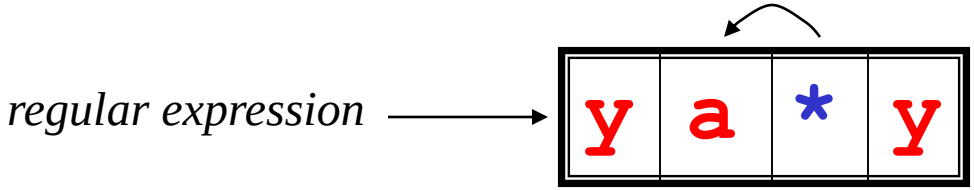
match

---

^word$** **^$

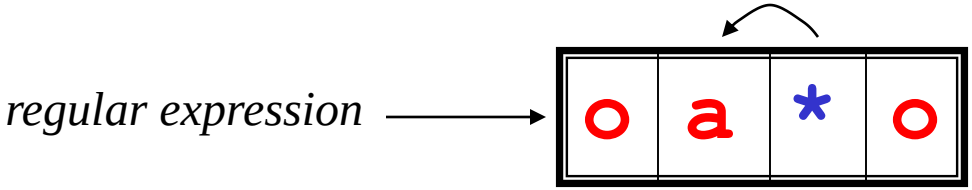
REPETITION

The * is used to define zero or more occurrences of the *single* regular expression preceding it.



I got mail, yaaaaaaaaaay!

match



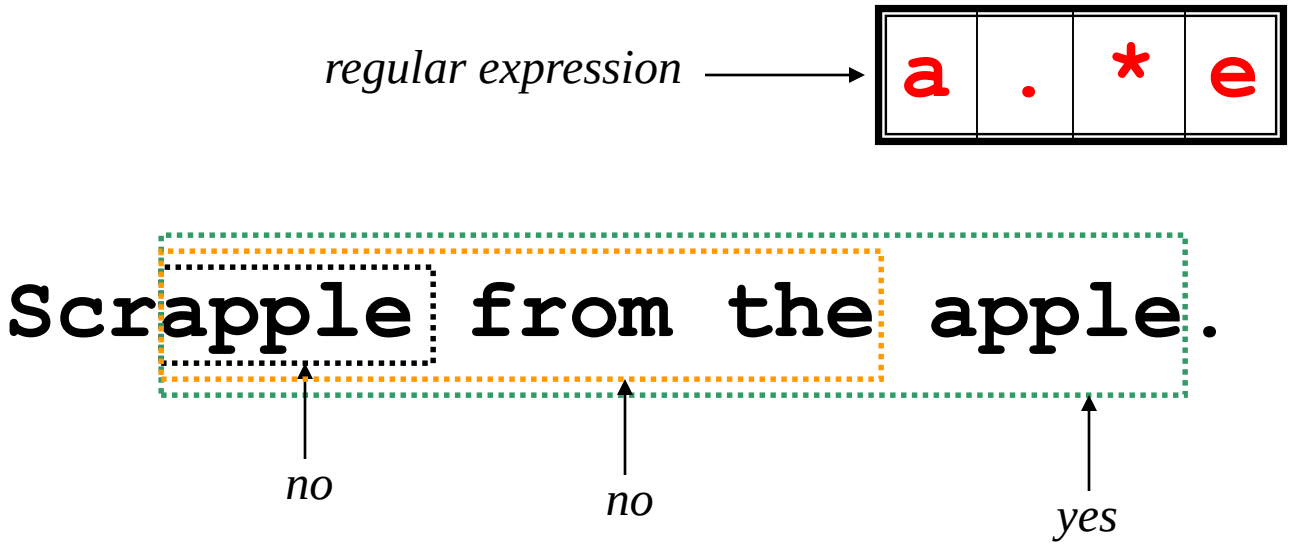
For me to poop on.

match

.*

MATCH LENGTH

A match will be the longest string that satisfies the regular expression.



REPETITION RANGES

Ranges can also be specified

- `{ }` notation can specify a range of repetitions for the immediately preceding regex
- `{n}` means exactly *n* occurrences
- `{n,}` means at least *n* occurrences
- `{n,m}` means at least *n* occurrences but no more than *m* occurrences

Example:

- `.{0,}` same as `.*`
- `a{2,}` same as `aaa*`

GREP

- grep comes from the ed (Unix text editor) search command “global regular expression print” or g/*re*/p
- This was such a useful command that it was written as a standalone utility
- There are two other variants, *egrep* and *fgrep* that comprise the *grep* family
- *grep* is the answer to the moments where you know you want the file that contains a specific phrase but you can’t remember its name

FAMILY DIFFERENCES

grep - uses regular expressions for pattern matching

fgrep - file grep, does not use regular expressions, only matches fixed strings but can get search strings from a file

egrep - extended grep, uses a more powerful set of regular expressions but does not support backreferencing, generally the fastest member of the grep family

agrep – approximate grep; not standard

GREP: BACKREFERENCES

Sometimes it is handy to be able to refer to a match that was made earlier in a regex

This is done using ***backreferences***

- \n is the backreference specifier, where *n* is a number

Looks for *n*th subexpression

For example, to find if the first word of a line is the same as the last:

- `^([[:alpha:]]{1,}) .* \1$`
- The `([[:alpha:]]{1,})` matches 1 or more letters

PRACTICAL REGEX EXAMPLES

Dollar amount with optional cents

- `\$[0-9]+(\.[0-9][0-9])?`

Time of day

- `(1[012]|[1-9]):[0-5][0-9] (am|pm)`

HTML headers <h1> <H1> <h2> …

- `<[hH][1-4]>`

CLICKER QUESTION

Select the string for which the regular expression '..\.19..' would find a match:

a) "12.1000"

b) "123.1900"

c) "12.2000"

d) the regular expression does not find a match for any of the strings above

CLICKER QUESTION

Choose the pattern that finds all filenames in which

1. the first letters of the filename are chap,
2. followed by two digits,
3. followed by some additional text,
4. and ending with a file extension of .doc

For example : chap23Production.doc

a) chap[0-9]*.doc

b) chap*[0-9]doc

c) chap[0-9][0-9].*\.doc

d) chap*doc

GREP FAMILY

Syntax

grep [-hilnv] [-e expression] [filename]

egrep [-hilnv] [-e expression] [-f filename] [expression] [filename]

fgrep [-hilnxv] [-e string] [-f filename] [string] [filename]

- **-h** Do not display filenames
- **-i** Ignore case
- **-l** List only filenames containing matching lines
- **-n** Precede each matching line with its line number
- **-v** Negate matches
- **-x** Match whole line only (*fgrep* only)
- **-e** *expression* Specify expression as option
- **-f** *filename* Take the regular expression (egrep) or a list of strings (fgrep) from *filename*

THREE EXTREMELY POWERFUL TOOLS

1) grep

Basic syntax:

```
grep 'regexp' filename
```

or equivalently (using UNIX pipelining):

```
cat filename | grep 'regexp'
```

2) sed – stream editor

Basic syntax

```
sed 's/regexp/replacement/g' filename
```

For each line in the intput, the portion of the line that matches regexp (if any) is replaced with replacement.

Sed is quite powerful within the limits of operating on single line at a time.

You can use \(\) to refer to parts of the pattern match.

THREE EXTREMELY POWERFUL TOOLS

Awk

Finally, awk is a powerful scripting language (not unlike perl). The basic syntax of awk is:

```
awk -F',' 'BEGIN{commands}
           /regexp1/ {command1} /regexp2/ {command2}
           END{commands}'
```

- For each line, the regular expressions are matched in order, and if there is a match, the corresponding command is executed (multiple commands may be executed for the same line).
- BEGIN and END are both optional.
- The -F',' specifies that the lines should be split into fields using the separator ",", and those fields are available to the regular expressions and the commands as $1, $2, etc.
- See the manual (man awk) or online resources for further details.

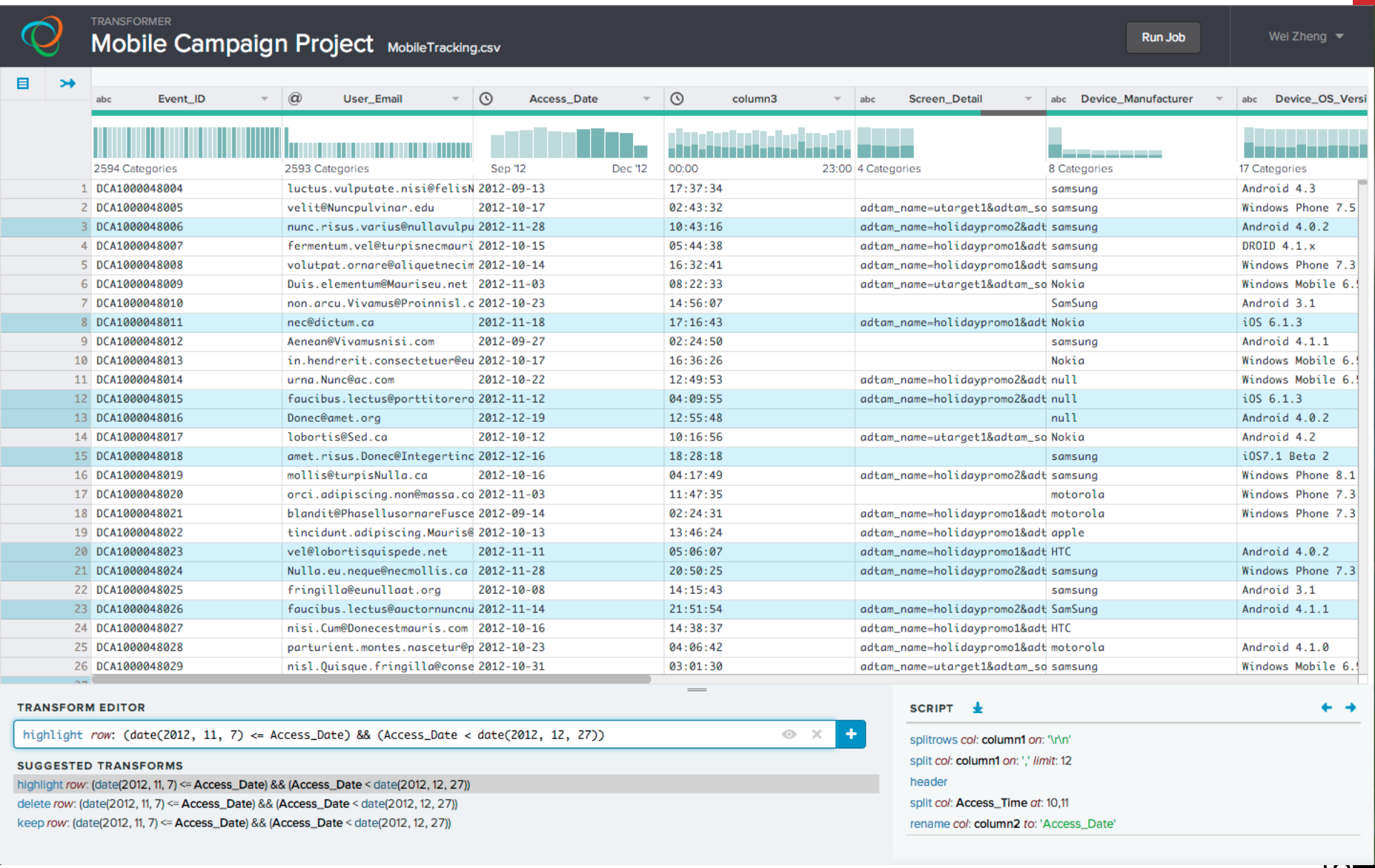
EXAMPLE

```
grep "created\_at" twitter.json
  | sed 's/.*"user":{"id":\([0-9]*\).*/\1/'
  | sort | uniq -c | sort -n | tail -5"
```

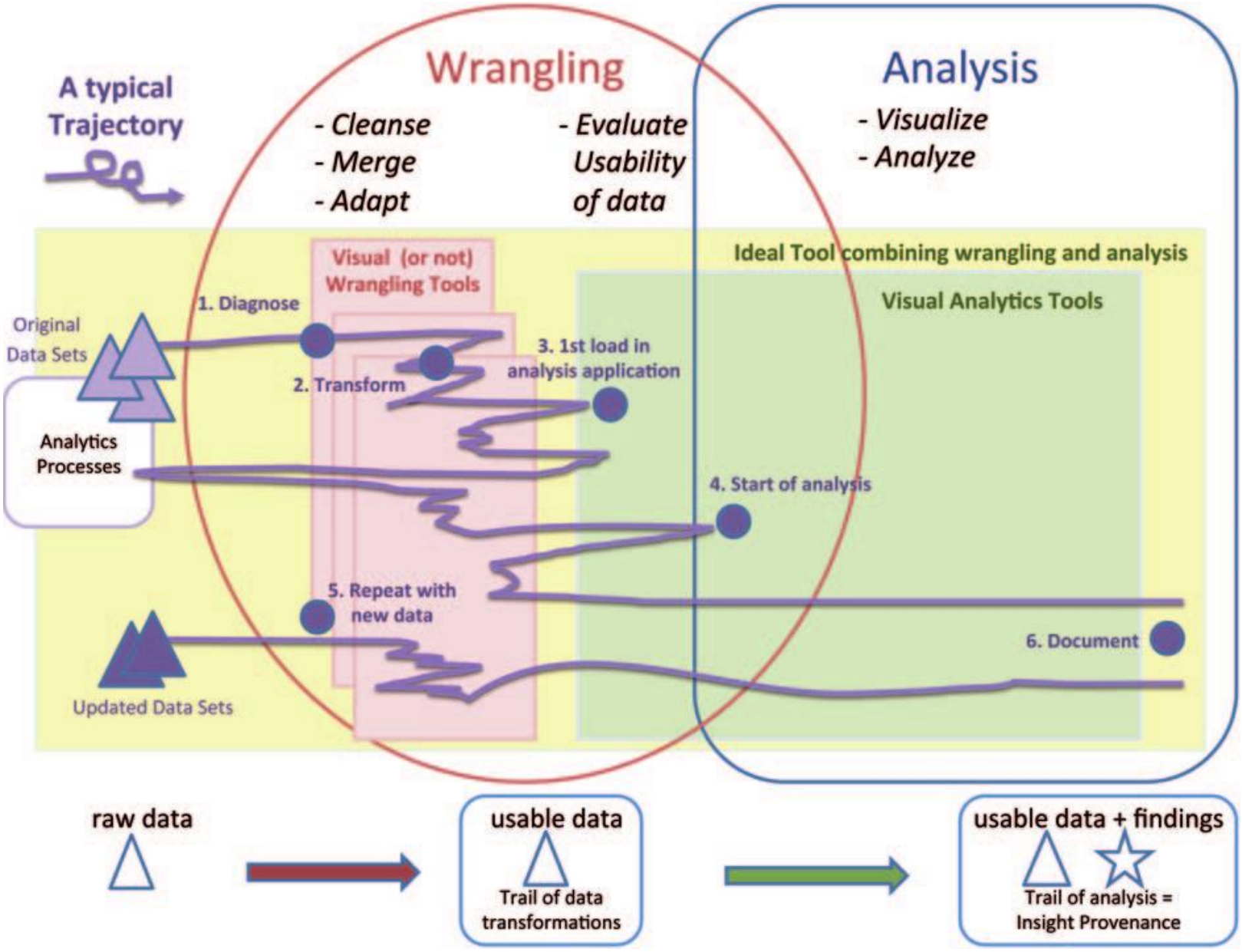

{"created_at":"Sat Aug 31 06:57:23 +0000
2013","id":373701031776882688,"id_str":"373701031776882688","text":"\u5ddd\u5d0e\u3055\u3093\u306e\u5bbf\u984c\u3084\u3063\u3066\u304f\u308c\u308b\u5fc3\u512a\u3057\u3044\u65b9\u306f\u5c45\u3089\u306c\u306e\u3067\u3059\u304b\uff1f\u2190","source":"\u003ca
href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter for
iPhone\u003c\/a\u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_to_screen_name":null,"user":{"id":1580127176,"id_str":"1580127176","name":"\u3061\u306e\u3071\u3093","screen_name":"08_chi_02","location":"","url":null,"description":"\u305f\u3060\u306e\u304a\u3070\u304b\u3067\u3059\u3002\u306f\u3044\u3002\n\u3078\u3093\u3066\u3053\u6ce8\u610f\u21af","protected":false,"followers_count":130,"friends_count":149,"listed_count":0,"created_at":
"Tue Jul 09 11:23:26 +0000
2013","favourites_count":62,"utc_offset":32400,"time_zone":"Tokyo","geo_enabled":false,"verified":false,"statuses_count":489,"lang":"ja","contributors_enabled":false,"is_translator":
false,"profile_background_color":"C0DEED","profile_background_image_url":"http:\/\/a0.twimg.com\/images\/themes\/theme1\/bg.png","profile_background_image_url_https":"https:\/\/si0.twimg.com\/images\/themes\/theme1\/bg.png","profile_background_tile":false,"profile_image_url":"http:\/\/a0.twimg.com\/profile_images\/378800000306401177\/a8912f698459a84e7343d19ac90f6fa0_normal.jpeg","profile_image_url_https":"https:\/\/si0.twimg.com\/profile_images\/378800000306401177\/a8912f698459a84e7343d19ac90f6fa0_normal.jpeg","profile_banner_url":"https:\/
\/pbs.twimg.com\/profile_banners\/1580127176\/1377526337","profile_link_color":"0084B4","pr

DATA WRANGLER / TRIFACTA

http://vis.stanford.edu/wrangler/app/

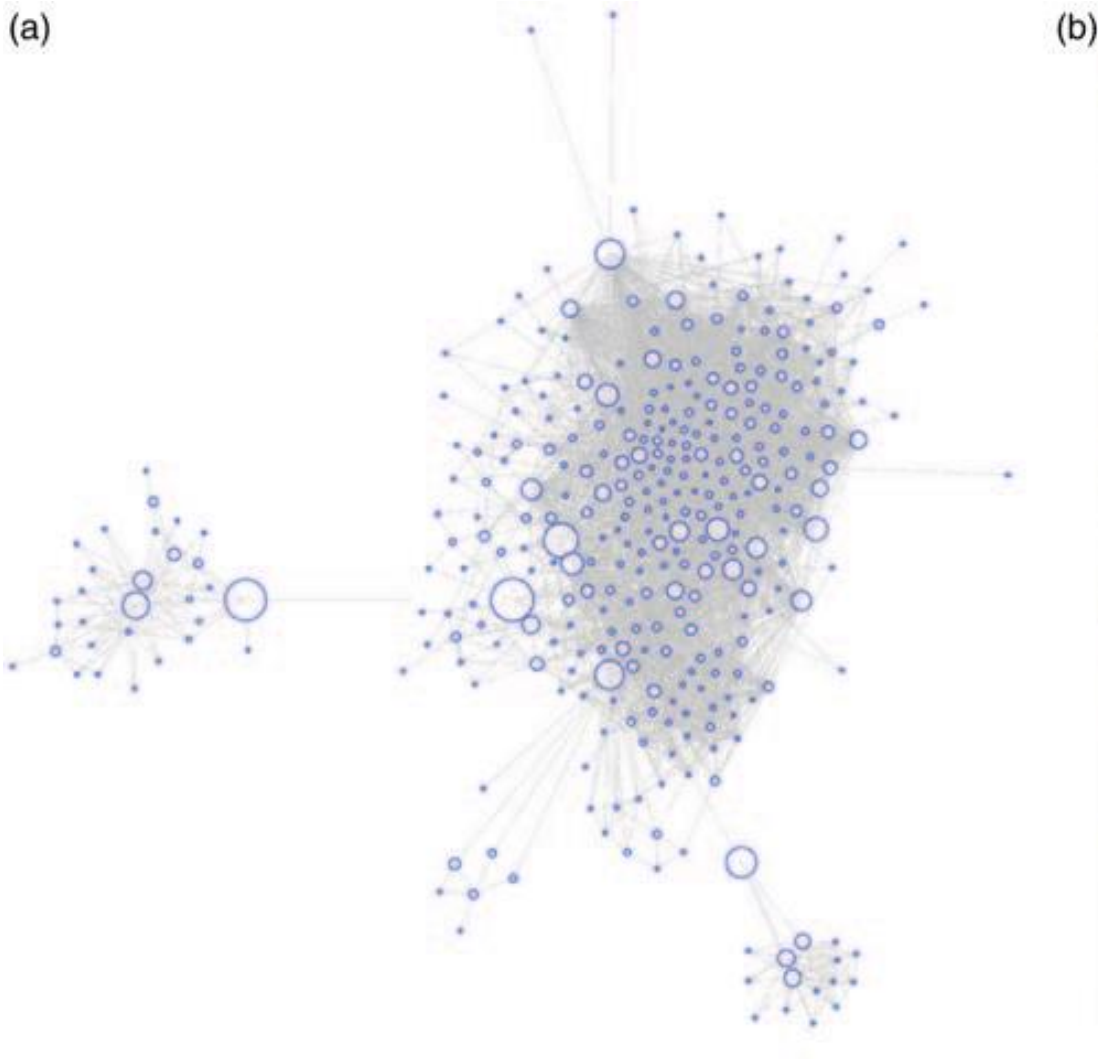


DATA WRANGLING



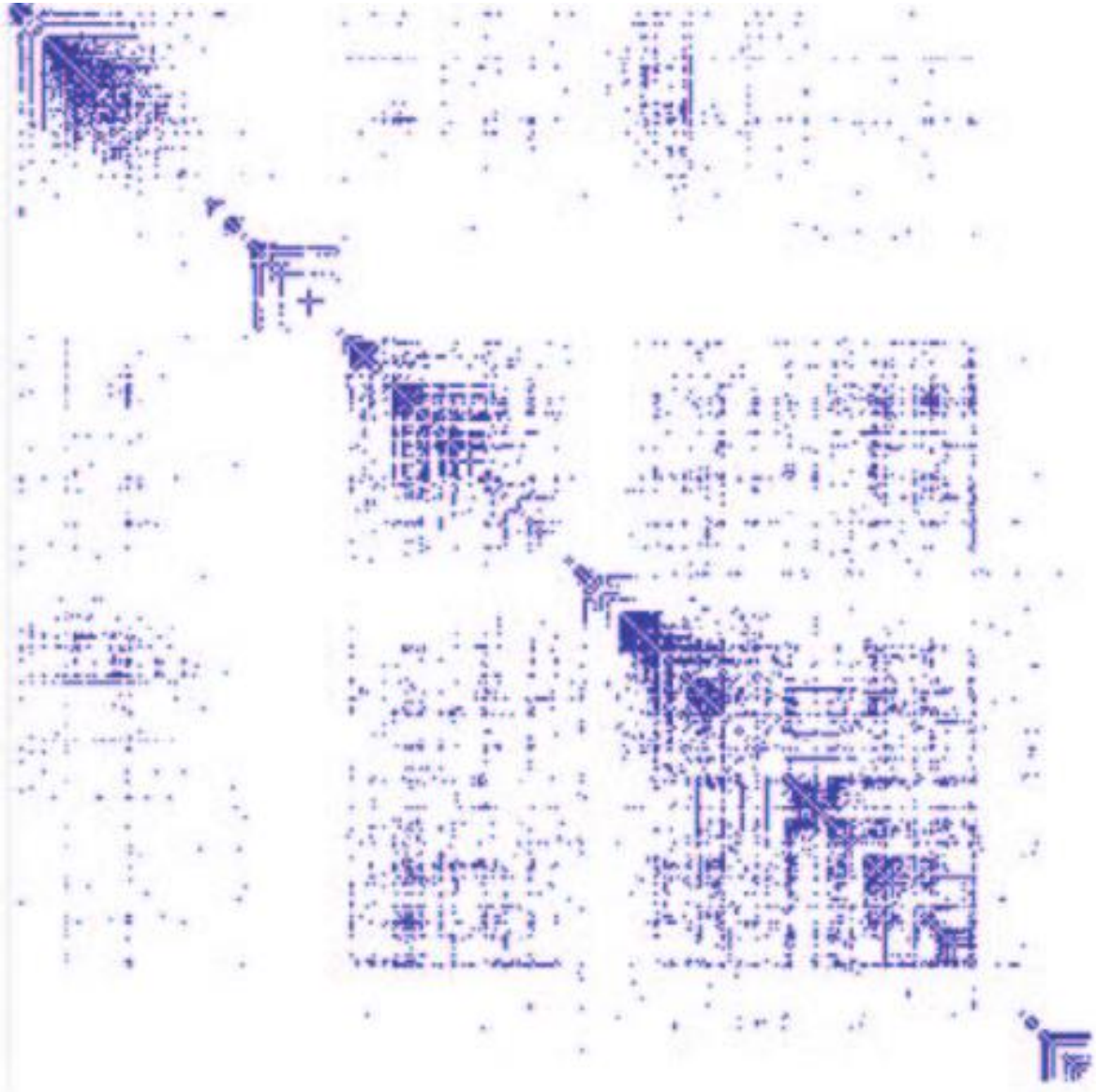
[Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011]

FACEBOOK VISUALIZATION I



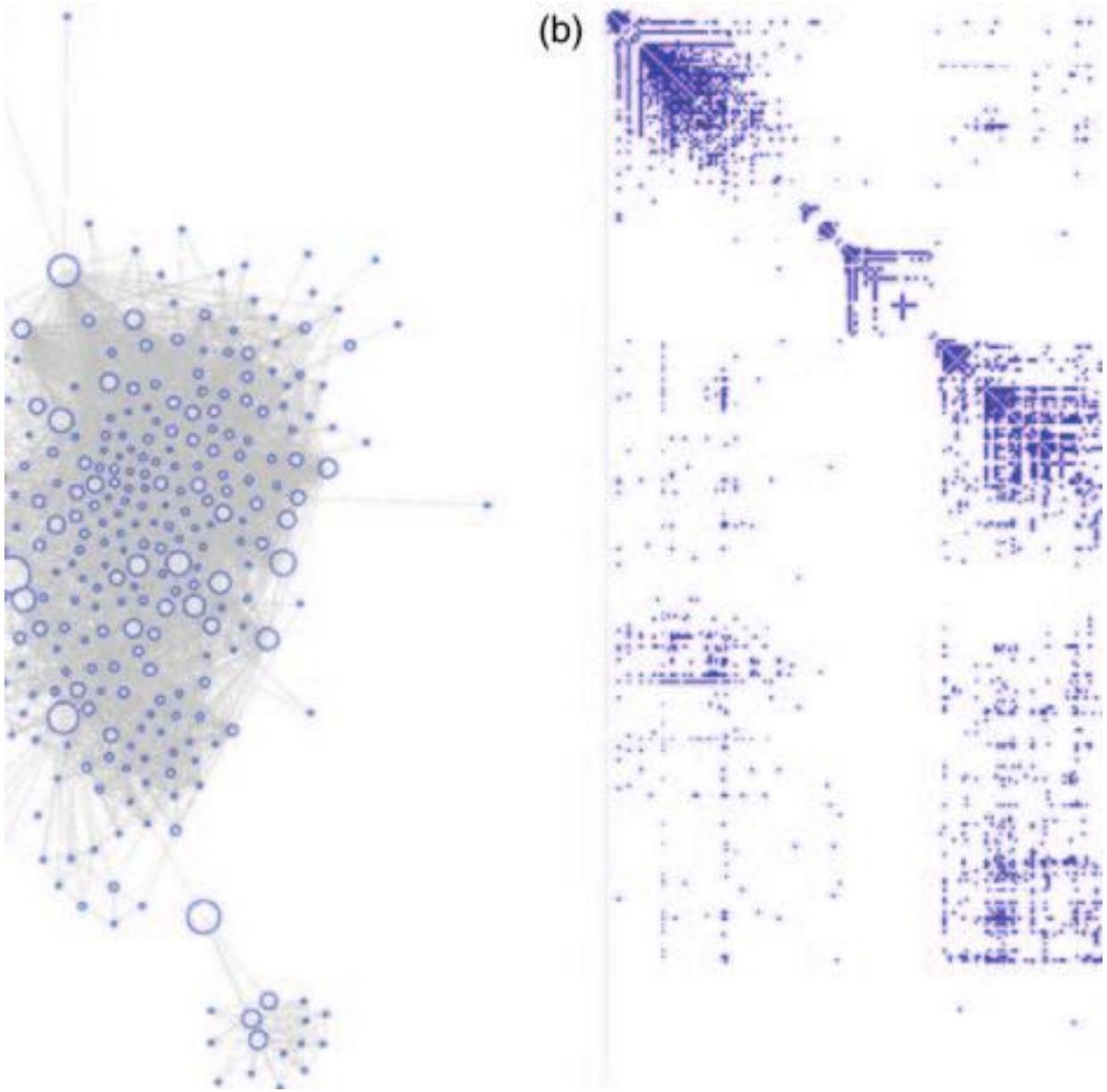
[Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011]

FACEBOOK VISUALIZATION II



[Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011]

FACEBOOK VISUALIZATION II



[Sean Kandel et al: Research directions in data wrangling: Visualizations and transformations for usable and credible data, Information Visualization, 2011]

HOMEWORK: PANDAS

Watch: 10-minute tour of pandas
http://vimeo.com/59324550

TOKENIZATION AND STEMMING

WORKING WITH TEXT

TOKENIZATION

Input: "*Friends, Romans and Countrymen*"

Output: Tokens

- *Friends*
- *Romans*
- *and*
- *Countrymen*

A token is an instance of a sequence of characters

What are valid tokens?

TOKENIZATION ISSUES

Issues in tokenization:

- Finland's capital → Finland? Finlands? Finland's?
- Hewlett-Packard → Hewlett and Packard as two tokens?
  - state-of-the-art: break up hyphenated sequence.
  - co-education
  - lowercase, lower-case, lower case ?
  - It can be effective to get the user to put in possible hyphens
- San Francisco: one token or two?
  - How do you decide it is one token?

TOKENIZATION ISSUES

3/20/91 Mar. 12, 1991 20/3/91

55 B.C.

B-52

My PGP key is 324a3df234cb23e

(800) 234-2333

- Often have embedded spaces
- Older IR systems may not index numbers
  - But often very useful: think about things like looking up error codes/stacktraces on the web
  - (One answer is using n-grams: Lecture 3)
- Will often index "meta-data" separately
  - Creation date, format, etc.

TOKENIZATION: LANGUAGE ISSUES

German noun compounds are not segmented

- ***Lebensversicherungsgesellschaftsangestellter*** → 'life insurance company employee'
- German retrieval systems benefit greatly from a **compound splitter** module (Can give a 15% performance boost for German)

French: *L'ensemble* → one token or two?

- L ? L' ? Le ?
- Want l'ensemble to match with un ensemble (Until at least 2003, it didn't on Google)

Chinese and Japanese have no spaces between words:

- 莎拉波娃现在居住在美国东南部的佛罗里达。
- Not always guaranteed a unique tokenization

Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

← → ← → ← start

STOP WORDS

With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:

- They have little semantic content: *the, a, and, to, be*
- There are a lot of them: ~30% of postings for top 30 words

For building search engines, the trend is away from doing this:

- Good compression techniques means the space for including stopwords in a system is very small
- Good query optimization techniques mean you pay little at query time for including stop words.
- You need them for:
  - Phrase queries: “King of Denmark”
  - Various song titles, etc.: “Let it be”, “To be or not to be”
  - “Relational” queries: “flights to London”

In contrast for analytics: you often remove them. Why?

NORMALIZATION TO TERMS

We need to "normalize" words in indexed text as well as query words into the same form

- We want to match ***U.S.A.*** and ***USA***

Result is terms: a term is a (normalized) word type, which is an entry in our IR system dictionary

We most commonly implicitly define equivalence classes of terms by, e.g.,

- deleting periods to form a term
  - ***U.S.A.***, ***USA*** → ***USA***
- deleting hyphens to form a term
  - ***anti-discriminatory, antidiscriminatory*** → ***antidiscriminatory***

NORMALIZATION: OTHER LANGUAGES

Accents: e.g., French *résumé* vs. *resume*.

Umlauts: e.g., German: *Tuebingen* vs. *Tübingen*

- Should be equivalent

Most important criterion:

- How are your users like to write their queries for these words?

Even in languages that standardly have accents, users often may not type them

- Often best to normalize to a de-accented term
  - ***Tuebingen, Tübingen, Tubingen* → *Tubingen***

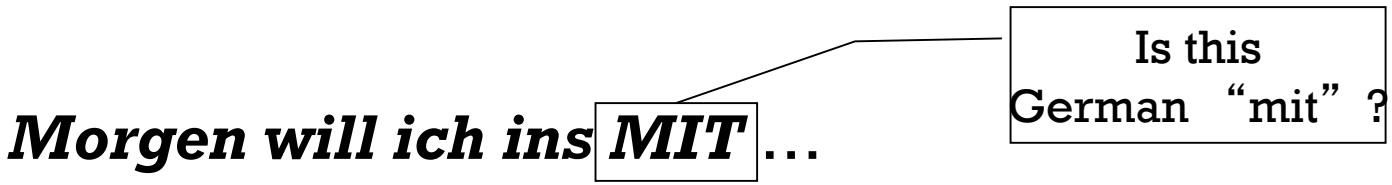
NORMALIZATION: OTHER LANGUAGES

Normalization of things like date forms

- *7月30日 vs. 7/30*
- *Japanese use of kana vs. Chinese characters*

Tokenization and normalization may depend on the language and so is intertwined with language detection

Crucial: Need to "normalize" indexed text as well as query terms into the same form



CASE FOLDING

Reduce all letters to lower case

- exception: upper case in mid-sentence?
  - e.g., ***General Motors***
  - ***Fed*** vs. ***fed***
  - ***Brown*** vs. ***brown***
- Often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…
- Again for analytics it depends

Google example:

- Query ***C.A.T.***
- #1 result is for "cat" *not* Caterpillar Inc.

NORMALIZATION TO TERMS

An alternative to equivalence classing is to do asymmetric expansion

An example of where this may be useful

- Enter: *window* Search: *window, windows*
- Enter: *windows* Search: *Windows, windows, window*
- Enter: *Windows* Search: *Windows*

Potentially more powerful, but less efficient

THESAURI AND SOUNDEX

Do we handle synonyms?

- E.g., by hand-constructed equivalence classes
  - ***car*** = ***automobile*** ***color*** = ***colour***
- We can rewrite to form equivalence-class terms
  - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
- Or we can expand a query
  - When the query contains ***automobile***, look under ***car*** as well

What about spelling mistakes?

- One approach is soundex, which forms equivalence classes of words based on phonetic heuristics

More in lectures 3 and 9

LEMMATIZATION

Reduce inflectional/variant forms to base form

E.g.

- *am, are, is → be*
- *car, cars, car's, cars' → car*

the boy's cars are different colors → the boy car be different color

Lemmatization implies doing "proper" reduction to dictionary headword form

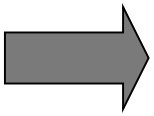
STEMMING

Reduce terms to their "roots" before indexing

"Stemming" suggest crude affix chopping

- language dependent
- e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.



for exampl compress and compress ar both accept as equival to compress

PORTER'S ALGORITHM

Commonest algorithm for stemming English

- Results suggest it's at least as good as other stemming options

Conventions + 5 phases of reductions

- phases applied sequentially
- each phase consists of a set of commands
- sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

TYPICAL RULES IN PORTER

sses → ***ss***

ies → ***i***

ational → ***ate***

tional → ***tion***

Weight of word sensitive rules

(m>1) EMENT →

- *replacement* → *replac*
- *cement* → *cement*

OTHER STEMMERS

Other stemmers exist, e.g., Lovins stemmer

- http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
- Single-pass, longest suffix removal (about 250 rules)

Full morphological analysis – at most modest benefits for retrieval

Do stemming and other normalizations help?

- English: very mixed results. Helps recall for some queries but harms precision on others
  - E.g., operative (dentistry) ⇒ oper
- Definitely useful for Spanish, German, Finnish, …
  - 30% performance gains for Finnish!

OTHER STEMMERS

Many of the above features embody transformations that are

- Language-specific and
- Often, application-specific

These are "plug-in" addenda to the indexing process

Both open source and commercial plug-ins are available for handling these