RAWDATA
Section 2

Information Retrieval part 3

Henrik Bulskov & Troels Andreasen

Key issues in the Portfolio Project

- What's needed to meet the requirement of the Portfolio Project?
  - Inverted indexing
    - A “binary” indexing (corresponding to the wi table)
    - A weighted indexing including a TFIDF weight
  - Querying posts
    - A best match querying procedure that returns a list of posts based on the binary index
    - A refined search procedure that returns a list of posts based on the weighted index
  - Querying word lists
    - A procedure that returns a list of words based on the binary index
    - A procedure that returns a list of words based on the weighted index
    - Simple visualization
  - Building term networks
    - From inverted indexing
    - Simple visualization

Optional challenges in the Portfolio Project

- Optional (not required) challenges
  - Similarity search (word forms),
  - Word proximity search based on positional indexing
  - Relevance feedback
  - Query expansion, term browsing, term network visualization
  - ...

  Your own ideas are obviously also welcome

  - ...
  - Concept-based querying (involving also taxonomy, ontology, …)
  - ...

Tag cloud / Word cloud

- Tag cloud / Word cloud
  - a visual representation of text data,
  - importance of each word is shown with font size and/or color
  - useful for quickly perceiving the most prominent terms
  - visualization principle commonly used on geographic maps to represent the relative size of cities

- Weighted word list
  - list of words with attached *importance weight*
  - weighted word lists is what tag clouds / word clouds build upon

Tag cloud / Word cloud

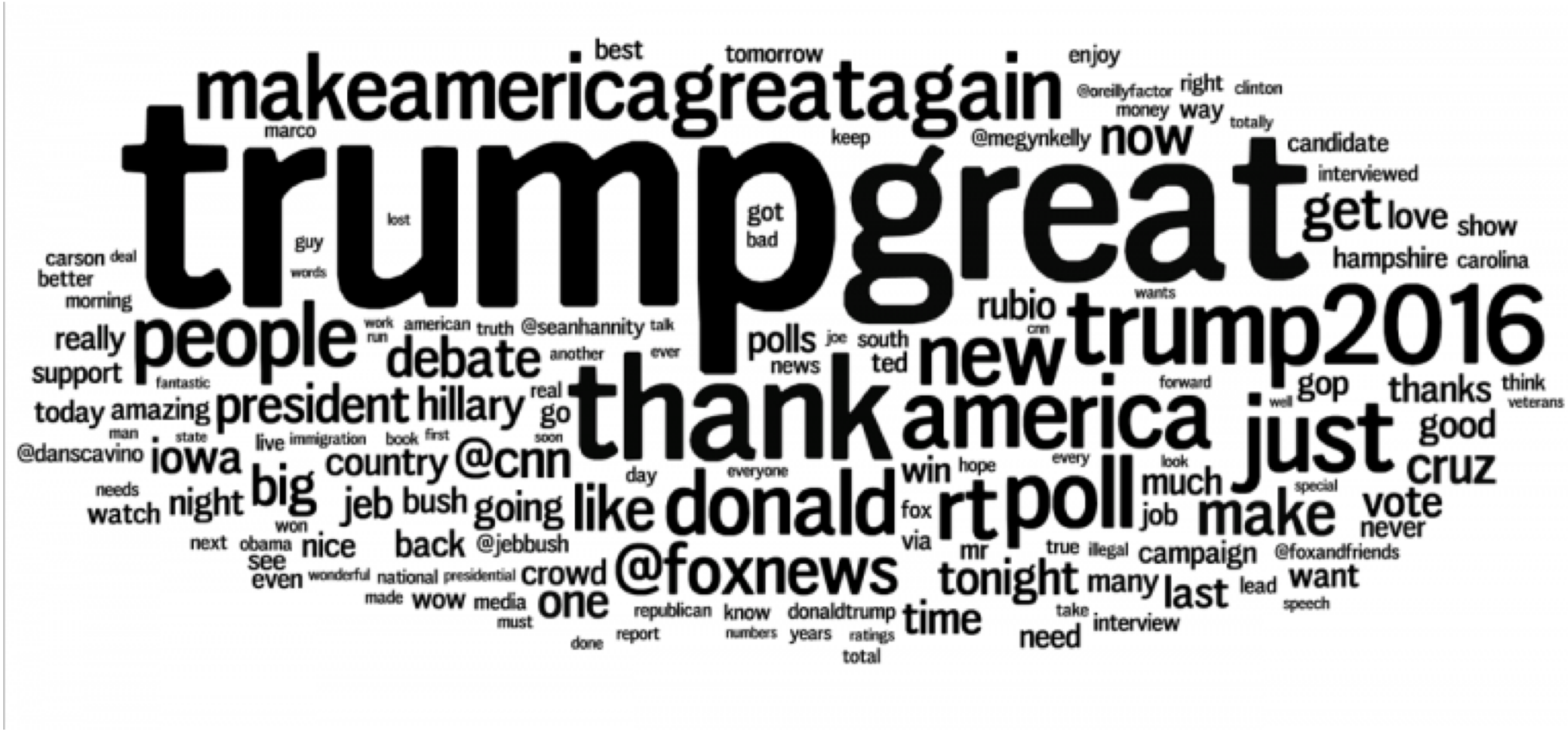
- Tag cloud / Word cloud
  - usually created with a particular purpose:
    - to present a visual overview of a collection of text
  - typically a word occurrence statistics over the text

- Also used for analytical purposes
  - examination of textual documents
  - such as
    - political speeches and
    - collections of blog posts
    - ...

- Examples
  - a speech given by Barack Obama in 2007
    as one-word and as two-word cloud
  - tweets by Trump in 2016

A one-word and a two-word cloud (Obama speech)

A word cloud based on some of Trump's tweets in 2016



Tag cloud / Word cloud problems

- Tag cloud / word cloud problems
  - long words get emphasis over short words,
  - difficult to find a single word,
  - font sizes can be difficult to compare,
  - the commonly used orderings scatter words randomly regarding their semantic relation

Word cloud

- A Mark Twain quote
  - *There are basically two types of people. People who accomplish things, and people who claim to have accomplished things. The first group is less crowded.*

Word cloud (ignoring stopwords)

- The 3-documents example:
  - Document *A*
    - *this text is an essay*
  - Document *B*
    - *here comes a fine, fine text*
  - Document *C*
    - *this text is well-written*

Importance based on frequency

- Weighted word list, frequency-based
  - weight reflects the number of occurrences of the word in the text (or set of text documents)

- Weighted tag list, frequency-based
  - weight reflects the count of documents that uses the tag

Weighted word list

- A Mark Twain quote
  - *There are basically two types of people. People who accomplish things, and people who claim to have accomplished things. The first group is less crowded.*

Weigthed word list (ignoring stopwords)

accomplish	2
basically	1
claim	1
crowded	1
group	1
people	3
things	2
types	1

- The 3-documents example:
  - Document *A*
    - *this text is an essay*
  - Document *B*
    - *here comes a fine, fine text*
  - Document *C*
    - *this text is well-written*

comes	1
essay	1
fine	2
text	3
this	2
well-written	1

Word cloud vs Weighted word list

Word cloud

Weigthed word list

accomplish	2
basically	1
claim	1
crowded	1
group	1
people	3
things	2
types	1

comes	1
essay	1
fine	2
text	3
this	2
well-written	1

Frequency-based weighted word list

- Given a set of documents *D*
  - *n(d,t)* = "number of occurrences of term t in document d" (term frequency)
  - *w(t)* = "count of occurrences of term t in the document set *D*"
  - thus
    - $w(t) = \sum_{d \in D} n(d,t)$ (sum of term frequency over all documents)

- The 3-documents example:
  - Document *A*
    - *this text is an essay*
  - Document *B*
    - *here comes a fine, fine text*
  - Document *C*
    - *this text is well-written*

Inverted index

Document	term	*n(d,t)*
A	this	1
A	text	1
A	is	1
A	an	1
A	essay	1
B	here	1
B	comes	1
B	a	1
B	fine	2
B	text	1
C	this	1
C	text	1
C	is	1
C	well-written	1

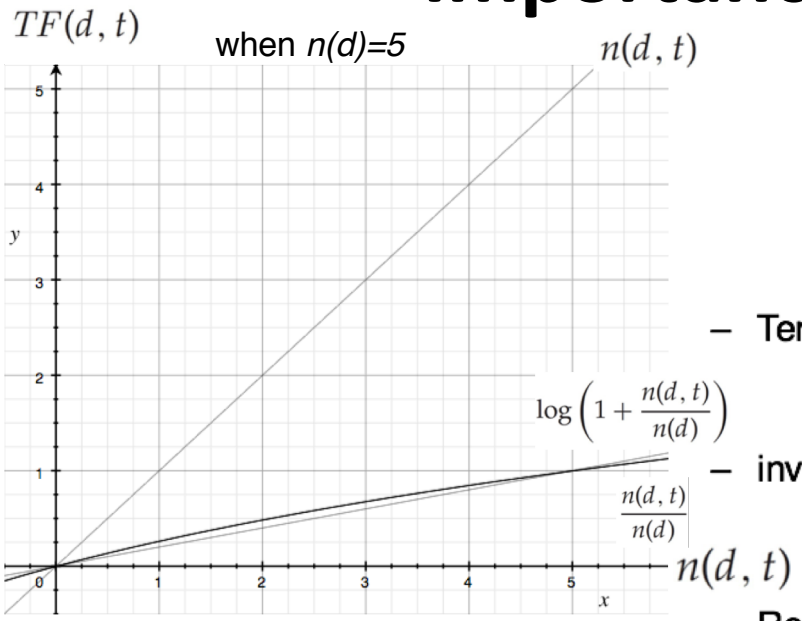
Weigthed word list

term	*w(t)*
a	1
an	1
comes	1
essay	1
fine	2
here	1
is	2
text	3
this	2
well-written	1

Importance based on relevance

- Taking relevance into account
  - relevance weighting rather than frequency

Importance based on relevance



$n(d)$: number of terms in the document *d*
$n(d,t)$: number of occurrences of term *t* in the document *d*
$n(t)$: number of documents that contain the term t

– Term frequency $$TF(d,t) = \log\left(1 + \frac{n(d,t)}{n(d)}\right)$$

– inverse document frequency $$IDF(t) = \frac{1}{n(t)}$$

– Relevance of document to *term t* $$r(d,t) = TF(d,t) * IDF(t)$$

- The 3-documents example:
  - Document *A*
    - *this text is an essay*
  - Document *B*
    - *here comes a fine, fine text*
  - Document *C*
    - *this text is well-written*

Document	term	*n(d)*	*n(d,t)*	*TF*	*n(t)*	*IDF*	*r(d,t)*
A	this	5	1	0,26	2	0,50	0,13
A	text	5	1	0,26	3	0,33	0,09
A	is	5	1	0,26	2	0,50	0,13
A	an	5	1	0,26	1	1,00	0,26
A	essay	5	1	0,26	1	1,00	0,26
B	here	6	1	0,22	1	1,00	0,22
B	comes	6	1	0,22	1	1,00	0,22
B	a	6	1	0,22	1	1,00	0,22
B	fine	6	2	0,42	2	0,50	0,21
B	text	6	1	0,22	3	0,33	0,07
C	this	4	1	0,32	2	0,50	0,16
C	text	4	1	0,32	3	0,33	0,11
C	is	4	1	0,32	2	0,50	0,16
C	well-written	4	1	0,32	1	1,00	0,32

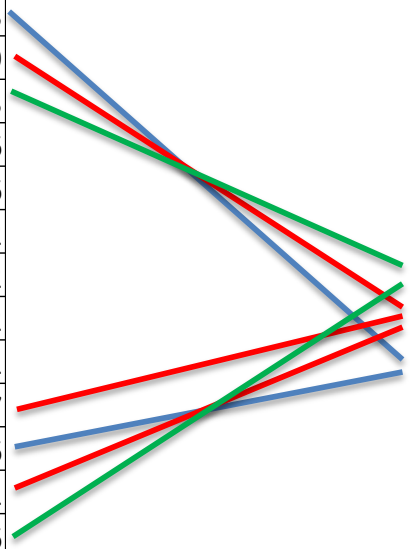
Relevance weighted word list

- Given a set of documents *D*
  - *r(d,t)* = "relevance of term t in document d"
  - *w(t)* = "sum of relevance of term t for all occurrences in the document set *D*"
  - thus
    - $w(t) = \sum_{d \in D} r(d,t)$ (sum of term relevance over all documents)

- The 3-documents example:
  - Document *A*
    - *this text is an essay*
  - Document *B*
    - *here comes a fine, fine text*
  - Document *C*
    - *this text is well-written*

Weigthed word list (with stopwords)

Document	term	*n(d)*	*n(d,t)*	*TF*	*n(t)*	*IDF*	*r(d,t)*
A	this	5	1	0,26	2	0,50	0,13
A	text	5	1	0,26	3	0,33	0,09
A	is	5	1	0,26	2	0,50	0,13
A	an	5	1	0,26	1	1,00	0,26
A	essay	5	1	0,26	1	1,00	0,26
B	here	6	1	0,22	1	1,00	0,22
B	comes	6	1	0,22	1	1,00	0,22
B	a	6	1	0,22	1	1,00	0,22
B	fine	6	2	0,42	2	0,50	0,21
B	text	6	1	0,22	3	0,33	0,07
C	this	4	1	0,32	2	0,50	0,16
C	text	4	1	0,32	3	0,33	0,11
C	is	4	1	0,32	2	0,50	0,16
C	well-written	4	1	0,32	1	1,00	0,32



term	*w(t)*
a	0,22
an	0,26
comes	0,22
essay	0,26
fine	0,21
here	0,22
is	0,29
text	0,27
this	0,29
well-written	0,32

Frequency vs relevance weighted word list

- The 3-documents example:
  - Document *A*
    - *this text is an essay*
  - Document *B*
    - *here comes a fine, fine text*
  - Document *C*
    - *this text is well-written*

Frequency weigthed word list

term	*w(t)*
text	3
fine	2
is	2
this	2
a	1
an	1
comes	1
essay	1
here	1
well-written	1

Relevance weigthed word list

term	*w(t)*
well-written	0,32
is	0,29
this	0,29
text	0,27
an	0,26
essay	0,26
a	0,22
comes	0,22
here	0,22
fine	0,21

Portfolio – Extended IR functionality

- from Portfolio 3 requirements
  - ...
  - Provide “weighted keyword lists” (ranked lists of words) as answers
  - frequency as well as relevance weighted
  - …

- so for frequency as well as for TFIDF based weighting you should
  - develop a weighted list search procedure with
    - input: a keyword list (same input as a normal query)
    - output: a weighted list of terms

- How to build a weighted list ...?

How to build a weighted list ...?

- to provide a list of terms
  - find all posts that match the query
  - find all words indexing (occurring in) these posts

- more specifically, to provide the weighted list of terms
  1) provide a list of (id, rank) for posts that match the query
  2) combine (join) this with all words indexing (occurring in) these posts
  3) group the result by word and calculate the weight by a sum(rank)

- a simple example: a specific 3-word-query

How to build a weighted list ...?

- example query keywords: ('using','regions','blocks')

1) provide a list of (id, rank) for posts that match the query

```
select id, sum(score) rank from
  (select distinct id, 1 score from wi where word = 'using'
  union all
  select distinct id, 1 score from wi where word = 'regions'
  union all
  select distinct id, 1 score from wi where word = 'blocks') t1
group by id;
```

id	rank
5338	1
5929	1
5953	1
5962	1
5971	1
6295	1
9033	1
9049	1
9063	2
12316	1

2) combine (join) this list with all words indexing (occurring in) these posts

```
select wi.id, word, rank from wi,
  (select id, sum(score) rank from
    (select distinct id, 1 score from wi where word = 'using'
    union all
    select distinct id, 1 score from wi where word = 'regions'
    union all
    select distinct id, 1 score from wi where word = 'blocks') t1
  group by id) t2
where wi.id=t2.id;
```

id	word	rank
5338	bar	1
5338	block	1
5338	blocks	1
5338	blow	1
5338	break	1
5338	building	1
5338	Business	1
5338	code	1
5338	CSS	1
5338	customers	1

How to build a weighted list ...?

- example query keywords: ('using','regions','blocks')

3) group the result by word and calculate the weight by a sum(rank)

```
//select wi.id, word, rank from wi,
select word,sum(rank) from wi,
  (select id, sum(score) rank from
    (select distinct id, 1 score from wi where word = 'using'
    union all
    select distinct id, 1 score from wi where word = 'regions'
    union all
    select distinct id, 1 score from wi where word = 'blocks') t1
  group by id) t2
where wi.id=t2.id
group by word;
```

word	sum(rank)
blocks	103
class	27
code	78
end	30
function	30
make	26
regions	39
Return	26
time	29
work	26

- further refinements
  - order the result in descending order
  - introduce a threshold as a minimum weight
  - as an alternative, set a fixed number of (highest weighted)words to be listed

Portfolio – Extended IR functionality

- The example above can be generalized
  - e.g. into 3-keyword query procedure (in line with bestmatch3 from last time)
  - and further into a dynamic procedure that can take any number of arguments

- Considerations – words and tags
  - you're supposed to use the word index (inverted file) that you have derived
  - you can also include tag-indexing based on the tagging embedded in the stackoverflow data
  - you may try with tag-only based versions, but, as it appears, this is not a requirement
    (and probably not that interesting due to very few tags compared to words)

Relevance feedback again

- Similarity can be used to refine an answer set to a keyword query
  - user selects a few relevant documents from those retrieved by a keyword query, and
  - system finds other documents similar to these
  - the above can be repeated

- Notice
  - relevance feedback can be supported by a procedure that takes
    - input: the id of a post P
    - output: result of a keyword query with e.g. the 10 highest weighted terms in the index of the post P
  - can be generalized to a procedure that takes more than one post id as input

  - the **implementation** can be **similar to** the **weighted keyword query list** procedure discussed above

Associations and co-occurrence

- Association
  - a graded relationship between a pair of two words,

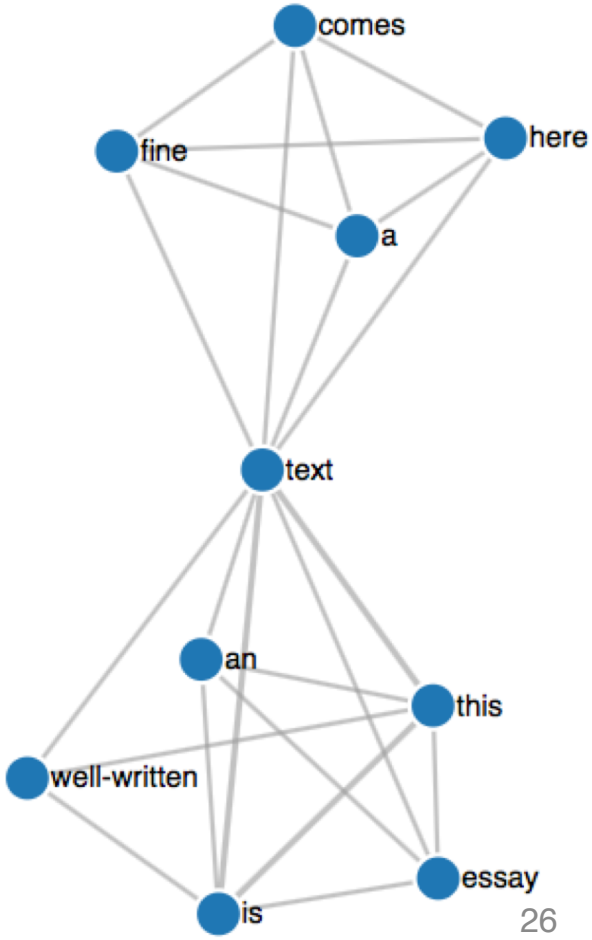
- Co-occurrence associations
  - an approach to derive associations
  - the more often two words occur together in the same document, the higher the association grade

- Co-occurrence term network
  - A graph with
    - a set of terms as nodes and
    - a set of edges each connecting two co-occurring terms

Associations and co-occurrence

- Co-occurrence term network
  - A graph with
    - a set of terms as nodes and
    - a set of edges each connecting two co-occurring terms

- Here in a simple visualization



- The 3-documents example:
  - Document *A*
    - *this text is an essay*
  - Document *B*
    - *here comes a fine, fine text*
  - Document *C*
    - *this text is well-written*

Associations and co-occurrence

- Building co-occurrence associations
  - join an inverted index with itself (self join)
  - e.g. the example **wi** on id (which is the same as postid)
  - This provides pairs of two words – group by these and count the number of occurences

```
select w1.word,w2.word,count(*) grade from wi w1,wi w2
where w1.id=w2.id and w1.word<w2.word
group by w1.word,w2.word;
```

- Notice (the number of rows in the simplified inverted index)

```
select count(*) from wi;
```

count(*)
774652

- Is this a problem?

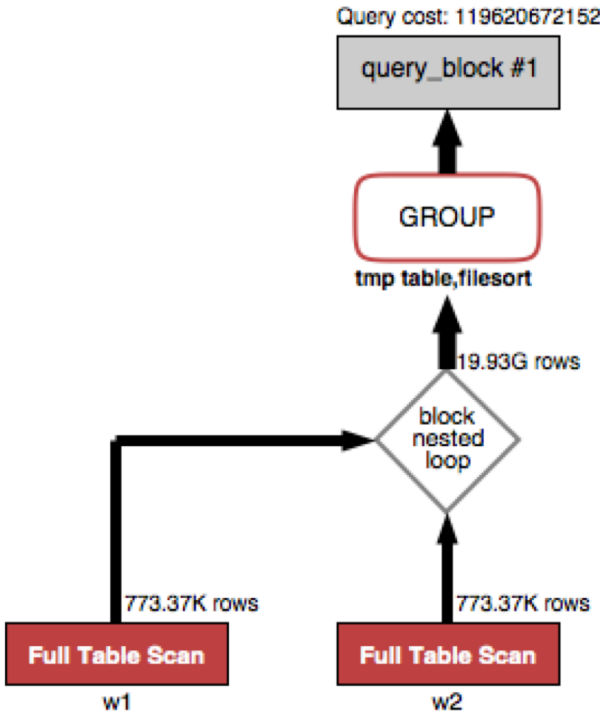
Associations and co-occurrence

- Is this a problem?
  - in principle we are considering the square of the count of combinations in the self join. 774652*774652 = 600.085.721.104

```
select w1.word,w2.word,count(*) grade from wi w1,wi w2
where w1.id=w2.id and w1.word<w2.word
group by w1.word,w2.word;
```

```
select count(*) from wi;
```

count(*)
774652



Associations and co-occurrence

- We should consider optimization by indexing here …
- In addition
  - assuming that the **wi** index include columns **tfidf** and **nt** (document frequency)
  - we could also consider to restrict:
    - words with low TFIDF are probably not good candidates for close associations
    - words that occur only in few documents are definitely not good candidates (occur in few documents => low document frequency *n(t)* (the column **nt**)
  - so add for instance limits like
    - a lower limit for TFIDF > 0.0002
    - a lower limit for nt > 20 (this is *n(t)*)

```
select w1.word,w2.word,count(*) grade from wi w1,wi w2
where w1.id=w2.id and w1.word<w2.word
and w1.tfidf>0.0002 and w2.tfidf>0.0002 and w1.nt>20 and w2.nt>20
group by w1.word,w2.word order by count(*) desc;
```

Associations and co-occurrence

- Building associations

```
select w1.word,w2.word,count(*) grade from wi w1,wi w2
where w1.id=w2.id and w1.word<w2.word
and w1.tfidf>0.0002 and w2.tfidf>0.0002 and w1.nt>20 and w2.nt>20
group by w1.word,w2.word order by count(*) desc;
```

- the highest graded associations:
- notice that these co-occurrence associations appear to make sense, semantically, at least
- the result can be stored in a separate table ...

word	word	grade
bar	foo	85
Studio	Visual	65
Height	Width	60
EventArgs	sender	58
Expressions	Regular	54
layout	tables	52
cout	endl	52
addresses	email	51
address	email	49
Chrome	FireFox	46
Explorer	Internet	45

Associations and co-occurrence

- Assume associations are stored in **assoc(word1,word2,grade)**

- Example associations

```
select word2,grade from assoc
where word1 = "address" order by grade desc;
```

word2	grade
email	49
addresses	28
validate	19
domain	15
e-mail	15
IP	14
valid	14
RFC	13
validation	13
pointer	12

```
select word2,grade from assoc
where word1 = "layout" order by grade desc;
```

word2	grade
tabular	17
layouts	15
table	12
width	12
screen	9
semantic	9
markup	8
readers	8
theme	8
rendering	7

```
select word2,grade from assoc
where word1 = "database"
order by grade desc;
```

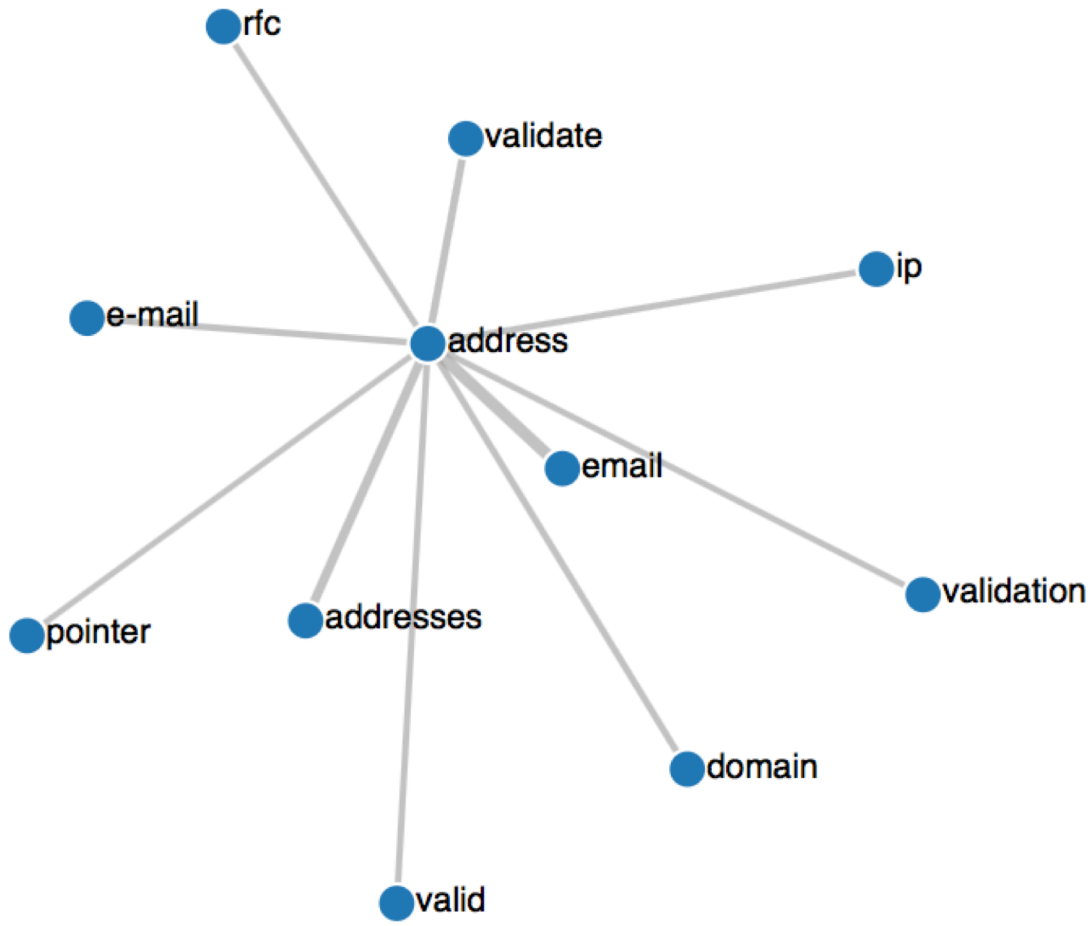
word2	grade
SQL	14
Databases	11
db	10
insert	10
query	9
folder	7
queries	7
injection	6
Password	6

Associations and co-occurrence

- Example associations + visualization:
  - subgraph over words connected to “address” to grade 12 or more

```
select word2,grade from assoc
where word1 = "address" order by grade desc;
```

word2	grade
email	49
addresses	28
validate	19
domain	15
e-mail	15
IP	14
valid	14
RFC	13
validation	13
pointer	12

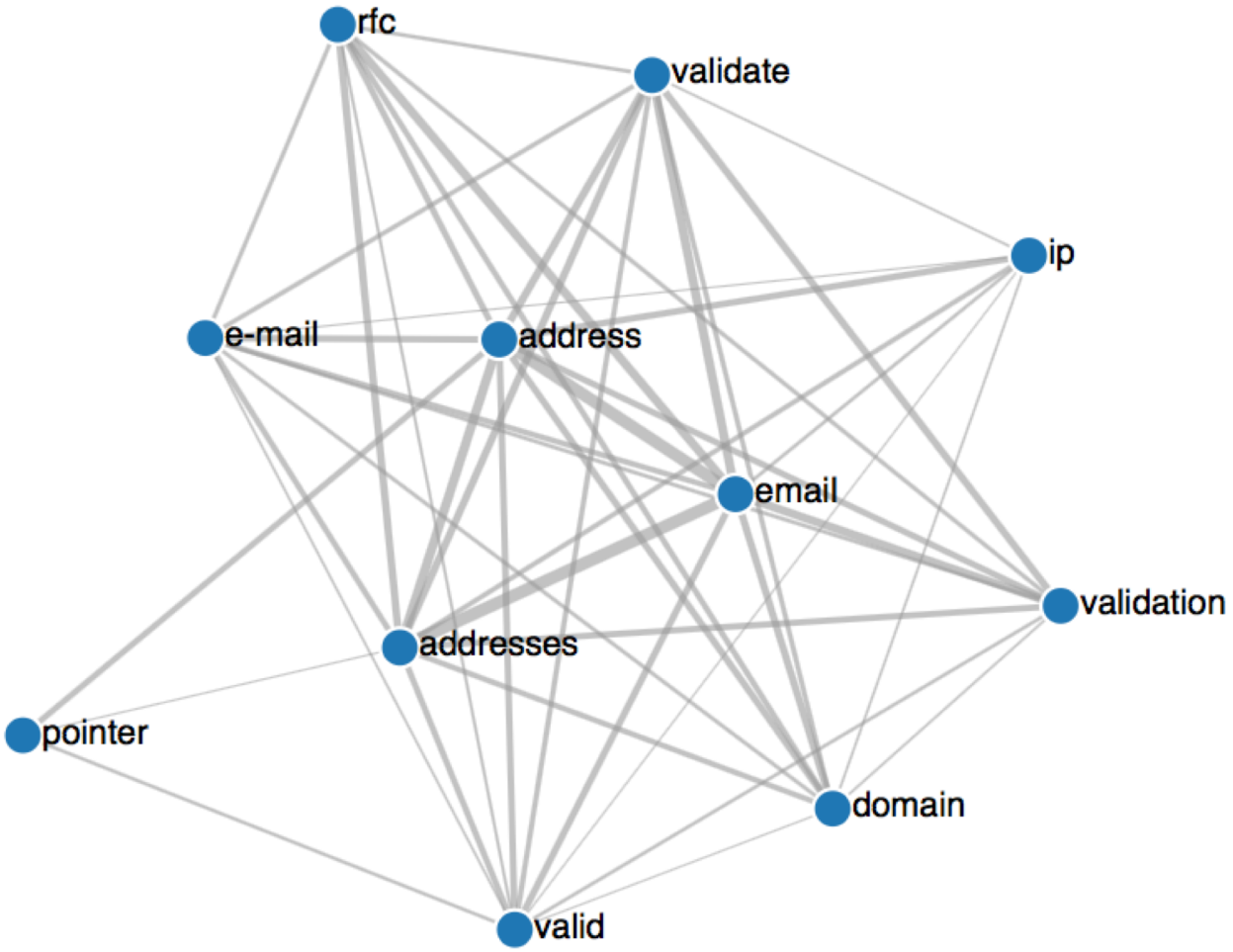


Associations and co-occurrence

- Example associations + visualization:
  - **full subgraph** over words connected to “address” to grade 12 or more

```
select word2,grade from assoc
where word1 = "address" order by grade desc;
```

word2	grade
email	49
addresses	28
validate	19
domain	15
e-mail	15
IP	14
valid	14
RFC	13
validation	13
pointer	12



Query expansion (again)

- Query expansion
  - reformulating the query to improve retrieval performance, aiming at getting
    - better query answers
  - reformulation involves
    - **expanding the query with keywords that are related** to those specified

- Example
  - Changing query
    - "motorcycle maintenance" (assume **and** as default)
  - into
    - "motorcycle ***and*** (repair ***or*** maintenance)"
  - is an example of query expansion based on synonyms

- Why not just expand into the following?
    - "motorcycle repair maintenance"

Query expansion

- Keyword query
  - a query specified as a list of keywords:
    - $k_1\ k_2\ ...\ k_n$ (such as "motorcycle maintenance")
  - has by default a conjunctive interpretation:
    - k_1 **and** k_2 **and** ... **and** k_n (such as "motorcycle **and** maintenance")

- Expansion of keyword
  - a keyword:
    - k_i (such as "maintenance")
  - can be expanded to a disjunction of **related** words:
    - k_{i1} **or** k_{i2} **or** ... **or** k_{im_i} (such as "maintenance **or** repair")

- Expansion of query
  - a query on n keywords:
    - $k_1\ k_2\ ...\ k_n$ (such as "motorcycle maintenance")
  - can be expanded into a conjunction of n disjunctions of **related** words:
    - (k_{11} **or** k_{12} **or** ... **or** k_{2m_2}) **and** (k_{21} **or**... **or** k_{2m_2}) **and** ... **and** (k_{n1} **or** k_{n2} **or** ... **or** k_{nm_n})
    - (such as "motorcycle **and** (maintenance **or** repair)")

Query expansion

- So what are these **related** words?
- Approaches to query expansion differ by definition of related words
- Common query expansion approaches use as **related** words:
  - **synonyms** of the given word
  - **word forms** (inflection forms) of the given word
  - common **spelling errors** of the given word
  - **associations**, e.g. by co-occurrence
    - the closest associated keywords by a certain threshold

Considering word forms

- Query expansion using **word forms** (inflection forms)
  - consider the excerpts taken from the words-table below
  - how can you expand this query?:
    - "programming functions"

(assume that there are no inflection forms apart from what can be seen here)

id	tablename	what	sen	idx	word	pos	lemma
189…	posts	body	8	3	programs	NNS	program
199…	posts	body	0	7	program	NN	program
200…	posts	body	2	18	programming	VBG	program

id	tablename	what	sen	idx	word	pos	lemma
4290	posts	body	0	28	function	NN	function
5923	posts	body	0	20	functions	NNS	function
5923	posts	body	0	29	function	NN	function
5990	posts	body	2	10	Functions	NNS	function

Considering word forms

- Query expansion using **word forms** (inflection forms)
  - the query:
    - "programming functions"
  - could be expanded into this
    - "(program **or** programs **or** programming) **and** (function **or** functions)"

(assume that there are no inflection forms apart from what can be seen here)

id	tablename	what	sen	idx	word	pos	lemma
189…	posts	body	8	3	programs	NNS	program
199…	posts	body	0	7	program	NN	program
200…	posts	body	2	18	programming	VBG	program

id	tablename	what	sen	idx	word	pos	lemma
4290	posts	body	0	28	function	NN	function
5923	posts	body	0	20	functions	NNS	function
5923	posts	body	0	29	function	NN	function
5990	posts	body	2	10	Functions	NNS	function

Considering word forms

- Query expansion using **word forms** (inflection forms)
  - for now my inverted index looks like (only two out of many rows shown here):

id	word	ndt	nd	nt	TFIDF
4290	function	1	12	1848	0.0000624876715238 26
5923	functions	1	20	510	0.0001380182899831 33

  - **could I create my inverted index wi in another way to obtain what I get from word form query expansion – without doing actual expansion?**

id	tablename	what	sen	idx	word	pos	lemma
189…	posts	body	8	3	programs	NNS	program
199…	posts	body	0	7	program	NN	program
200…	posts	body	2	18	programming	VBG	program

id	tablename	what	sen	idx	word	pos	lemma
4290	posts	body	0	28	function	NN	function
5923	posts	body	0	20	functions	NNS	function
5923	posts	body	0	29	function	NN	function
5990	posts	body	2	10	Functions	NNS	function