CS 533 Information Retrieval Systems
Spring 2014 Class Notes of March 17th - 19th
Prepared by Devrim Şahin

Cluster Validation

Assume that we have an IR test collection, consisting of:
- A set of documents,
- A set of queries,
- Relevant documents for each query.

Remember **cluster hypothesis**: "Documents similar to each other will be relevant to the same query"

Target cluster: A cluster that contains at least one relevant document for the query

$\mathbf{n_t}$: Average number of target clusters for the query (this should be a *small* number)

But how do we define *small*?

Observe that, for a query with 10 relevant documents, minimum number of target clusters is 1; and maximum number of target clusters is 10.

$\mathbf{n_{tr}}$: Average number of clusters for a *random* clustering structure

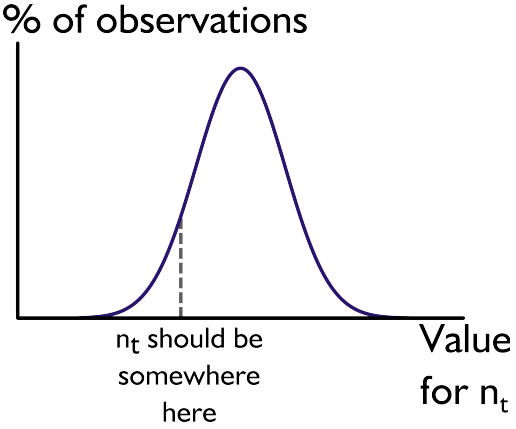
Keep the clustering structure the same, and distribute documents randomly into these clusters.

One random is not enough. Do the same thing again, keep the structure (# of docs in each cluster) the same, assign documents randomly.

Do this, say, 100 times, and take the average number of target clusters in the random case (n_{tr}). (This approach is named the **Monte Carlo experiment**.)

For a valid clustering structure, we need to have $\mathbf{n_t < n_{tr}}$.
If $n_t < n_{tr}$, then possibly we have a valid clustering structure. However, it is not enough. n_t must be *significantly* smaller than n_{tr} ($n_t << n_{tr}$).

Obtain the distribution of random observations and show them in a histogram:



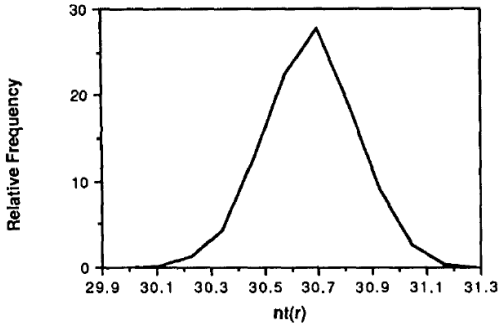


Fig. 5. Histogram of the $n_t(r)$ values for the INSPEC database (for D_{ft}) (min = 30.051; max = 31.221; average = 30.659; standard deviation = 0.172; bin length = 0.117).

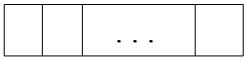
(The right-hand side plot is taken from "Concepts and the effectiveness of the cover coefficient-based clustering methodology" by F. Can and E. A. Ozkarahan)

How can we obtain the number of clusters for the random case?

Yao's formula

(see: Yao, S. B., "Approximating block accesses in database organizations." *Communications of the ACM*, Vol. 20, No. 4, 1977, pp. 260-261)

- Each block contains the same number of records (fixed record size)



Blocks

Records

- How many blocks to access?

- How should we distribute the records, so that numbers to be accessed is minimized?

- Yao suggests a formula for the solution

- **We can modify the formula to fit our needs**
(see: Edward Omiecinski, Peter Scheuermann: A Parallel Algorithm for Record Clustering. ACM Trans. Database Syst. 15(4): 599-624 (1990))

n: Number of records

m: Number of blocks (n_c)

k: Number of records to be accessed (number of relevant documents)

Block size: n/m (no. of records per block)

How many blocks to be accessed: $k \leq n - n/m$

(see: Cardenas paper)

Number of records in the j^{th} block: p = Block size = n/m

Number of records in other blocks: n - p

Number of combinations that we can have if we try to select k documents out of n documents: C^n_k

$C^n_k = n! / (k! (n-k)!)$

C^{n-p}_k: different ways of selecting k documents from n-p documents

The probability that no records are selected from the j^{th} block: C^{n-p}_k / C^n_k

To simplify the notation, let d = 1- 1/m

Then n-p = n-n/m = n(1-1/m) = nd

Probability of selecting at least a record from the j^{th} block: $E(I_j) = 1 - (C^{n-p}_k / C^n_k) = 1 - (C^{nd}_k / C^n_k)$

Expected number of blocks to be accessed: $\sum_{j=1}^{m} E(I_j) = m \times (1 - \frac{C_k^{nd}}{C_k^n})$

$$m \times (1 - \frac{C_k^{nd}}{C_k^n}) = m \times (1 - \frac{\frac{nd!}{k!(nd-k)!}}{\frac{n!}{k!(n-k)!}}) = m \times (1 - \prod_{i=1}^{k} \frac{nd-i+1}{n-i+1})$$

In an IR environment, the probability of accessing a cluster (probability of accessing a cluster as a target cluster):

$$p_j = 1 - \prod_{i=1}^{k} \frac{m_j - i + 1}{n - i + 1}$$

m_j: Number of documents in clusters other than the cluster C_j

Recall that:

n_{tr}: Average number of target clusters when documents are randomly distributed for a set of queries

n_t: The same average obtained by the clusters generated

We may tweak the formula to use it in *de*clustering applications.

Job distribution

e.g.: Signature files

EXAMPLE:

k = 3
m = 100 (no. of records)
Cluster C_1, $|C_1| = 5$ (no. of records in C_1)
Then $m_j = 100 - 5 = 95$
$p_1 = 1 – [(95 – 1 + 1)/(100-1+1)] * [(95 – 2 + 1)/(100-2+1)] * [(95 – 3 + 1)/(100-3+1)] = 1-0.86$
$p_1 = 0.14$

Do this for every cluster, and sum these values up to obtain n_{tr}.

Stemming

We use stemming with the hope of increasing recall and precision (effectiveness in general).

`computer computability compute computation ==> comput` <- stem

The version with stemming improves effectiveness by combining terms, thus detecting more documents with the same term.

Stemming vs. lemmatizing:

```
better -> [lemmatizer] -> good
better -> [Porter's stemming algorithm] -> bett
```

Stemming also helps word conflation (making words smaller).

Types of stemming:

- **1.** Manual
- **2.** Automatic
  - **2.a.** Affix Removal
    - **2.a.x.** Longest Match
    - **2.a.x.** Simple Removal
  - **2.b.** Successor Variety*
  - **2.b.** Table Lookup
  - **2.b.** N-gram*

(* We will discuss these)

Affix Removal: Both prefix & suffix

(see: Porter's Algorithm)

How to evaluate stemmers?

- Correctness: words coming from the same root
- Impact on retrieval effectiveness
- Impact on index size (compression)

Problems for stemming:

Overstemming, understemming

(see: W. B. Frates and R. Baeza-Yates, “Information Retrieval Data Structures and Algorithms”, Chapter 8)

Word truncation: First n characters

e.g.: First-5 (F5) works well for Turkish

```
bir -> [f5] -> bir
```

```
hatırlamak -> [f5] -> hatır
```

n-gram based stemming: Obtain n-letter subchains

2-gram is also named 'bi-gram'

e.g.: bi-grams for 'statistics' = {'st','ta','at','ti','is','st','ti','ic','cs'}

bi-grams for 'statistical' = {'st','ta','at','ti','is','st','ti','ic','ca','al'}

Use Dice measure to determine similarity:

- 8 common bi-grams
- dice ('statistics', 'statistical') = 2x8 / (9+10) = **16/19**

Construct clusters using this similarity measure between terms

Successor variety (SV):

SV is due to Hafer & Weiss. It tries to determine stems using a collection of words (**corpus**).

An example:

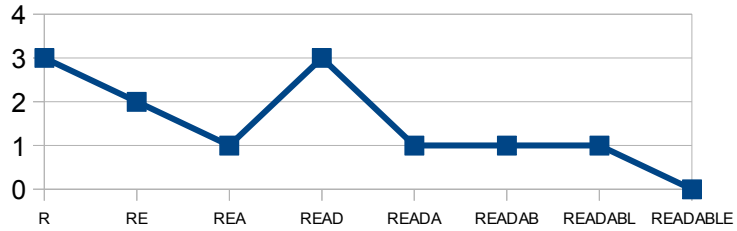
- Corpus:

```
ABLE, APE, BEATABLE, FIXABLE, READ, READABLE, READING,
READS, RED, ROPE, RIPE
```

- Test word:

```
READABLE
```

Prefix	Successor Variety	Letters
R	3	E, O, I
RE	2	A, D
REA	1	D
READ	3	A, I, S
READA	1	B
READAB	1	L
READABL	1	E
READABLE	0	



We can cut from READ.

PAT trees ('Patricia tree'):

n sistring (semi-infinite string)

For a PAT tree of n sistrings, there are n external nodes (leaves) and (n-1) internal nodes

text -> binary

Bit position: 1 2 3 4 5 6 7 8 9 10 11 12

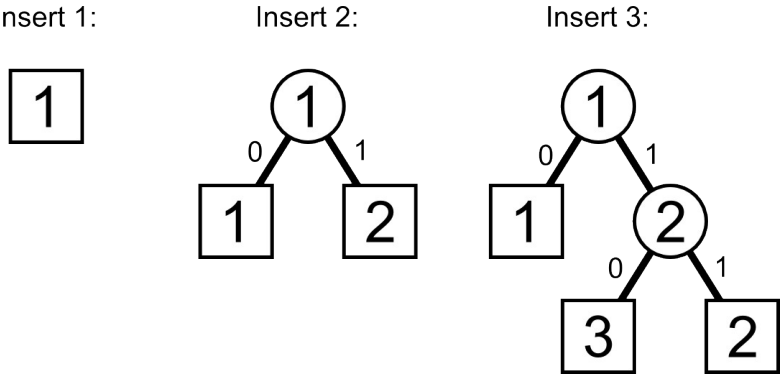
String: 0 1 1 0 0 1 0 0 0 1 0 1

Sistring1: 0 1 1 0 0 1 0 0 0 1 0 1

Sistring2: 1 1 0 0 1 0 0 0 1 0 1

Sistring3: 1 0 0 1 0 0 0 1 0 1

(...)



In fact we do not need to keep the tree structure; we can use the PAT array for tree search.

(Questions are to be added)