Course 5

Boolean retrieval and *tf-idf*

Projects assignment

- P1. Lego assembling => 2A, 1B (Md, I)
- P2. Plant breeding => 1A, 2B (Md, I)
- P3. Segmentation of writings: prints and manuscripts => 5A, 4B (Ma, I)
- P4. Automatic magazines metadata filling => 6A, 5B (Ma, I)
- P5. OCR on printed texts and manuscripts => 4A, 7A (D, Ma)
- P8. The chess game => 3B, 6B (D, I)
- P9. Voice-text alignment=> 3A, E (Md, I)

- Mădălina: 1A, 2A, 3A
- Diana: 4A, 6B, 7B (?)
- Marius: 5A, 6A, 7A
- Ionuț: 1B, 2B, 3B, 4B, 5B, E

Information retrieval

- INFORMATION RETRIEVAL deals with retrieving relevant documents from a collection based on a set of keywords (terms).
  - Each document contains words.
  - For a document some words are more significant than others (for instance, have more occurrences).
  - The user addresses a query as a set of keywords.
  - She/he wants to find the most relevant documents in the collection that fit the query
  - The retrieved documents should be ranked in the descending order of relevance

Classification

- CLASSIFICATION (CLUSTERING) deals with placing objects in different classes, based on a collection of features
  - In informed (supervised) classification problems the classes are known and each contains already a number of objects
  - In uninformed (unsupervised) classification, the number of classes is not known a-priory
  - Each object has associated a vector of values, each corresponding to a certain feature
  - To each class, some features are more relevant than others (they weight more in the classification process)
  - Since the classes contain already objects, it should be possible to rank the relevance of features for each class

Reference

- This course follows closely the material in:

Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze (2009). An Introduction to Information Retrieval. Cambridge University Press

Boolean retrieval

- Let's consider that an object (document) is characterized by a set of l Boolean parameters (terms), and s_i, the parameter i ($1 <= i <= l$), has the value 1 if the object has that property and 0 otherwise.
- Let $g_1, \ldots, g_\ell \in [0, 1]$ a set of weights such that $\Sigma_{i=1}^{l} g_i = 1$.
  - then we can compute the **Boolean score** of a document: $\Sigma_{i=1}^{l} g_i s_i$,
- **Ranked boolean retrieval**: rank documents based on the Boolean score.

But how can weights be determined?

- Either assign them manually based on expert knowledge, or…
- Try to infer them from examples: this problem is known as **machine-learned relevance**
  - we are given a set of **training examples**, each of which is a tuple consisting of a query *q* and a document *d*, together with a relevance judgment for *d* on *q* (for instance, Relevant or Non-relevant);
  - weights g_i are then "learned" from these examples, in order that the learned scores approximate the relevance judgments in the training examples (can be reduced to an optimization problem);
  - this methodology is expensive because the user-generated relevance judgments from which to learn the weights are usually labor-intensive.

Computing weights

- Suppose a simple case of weighted parameter scoring, where each object (document) has 2 zones: a title (T) and a body (B). Given a query *q* and a document *d*, we compute Boolean variables $s_T(d, q)$ and $s_B(d, q)$, depending on whether the title (respectively, the body) of *d* matches query *q*.

- We will compute a score, between 0 and 1, for each pair <document, query> using $s_T(d, q)$ and $s_B(d, q)$ and a constant $g \in [0, 1]$, as follows:

  - $\text{score}(d, q) = g \cdot s_T(d, q) + (1 - g)\, s_B(d, q)$

Training examples

relavance (judged by a human expert)

- A set of training examples
  - a set of triples of the form $\phi_j = (d_j, q_j, r(d_j, q_j))$

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Non-relevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Non-relevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3191	driver	1	0	Non-relevant

  - if Relevant judgments are quantized by 1 and Non-relevant – by 0, then the error of the scoring function with weight *g* can be computer with:

    $$\varepsilon(g, \phi_j) = (r(d_i, q_i) - \text{score}(d_i, q_i))^2$$

  - minimizing the total error of a set of training examples gives the optimum weight *g*:

    $$g = \text{argmin}_g (\Sigma_j\ \varepsilon(g, \phi_j))$$

Optimal weight

s_T	s_B	Score
0	0	0
0	1	$1-g$
1	0	g
1	1	1

- Computing the error as the sum of square differences between judgment and score $g \,.\, s_T(d, q) + (1-g)\, s_B(d, q)$:
  - n_{01r} = # training examples for which $s_T(d_j, q_j) = 0$ and $s_B(d_j, q_j) = 1$ and the editorial judgment is Relevant
  - n_{01n} = # training examples for which $s_T(d_j, q_j) = 0$ and $s_B(d_j, q_j) = 1$ and the editorial judgment is Non-relevant
  - …
  - $s_T = 0, s_B = 1 \Rightarrow (1 - (1 - g))^2 \,.\, n_{01r} + (0 - (1 - g))^2 \cdot n_{01n}$
  - $s_T = 0, s_B = 0 \Rightarrow (1 - 0)^2 \,.\, n_{00r} + (0 - 0)^2 \cdot n_{00n}$
  - $s_T = 1, s_B = 0 \Rightarrow (1 - g)^2 \,.\, n_{10r} + (0 - g)^2 . \, n_{10n}$
  - $s_T = 1, s_B = 1 \Rightarrow (1 - 1)^2 \,.\, n_{11r} + (0 - 1)^2 \cdot n_{11n}$

Optimal weight

- After differentiating with respect to *g* and setting the result to zero:

$$g_{optim} = \frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}.$$

$n_{10r} = 0$, $n_{01n} = 1$, $n_{10n} = 1$, $n_{01r} = 2 \Rightarrow g = (0 + 1)/(0 + 1 + 2 + 1) = 1/4 = 0.25$

$\text{score}(d, q) = 0.25\, s_T(d, q) + 0.75\, s_B(d, q)$

But what if some terms occur more times?

- Compute a score between a query term *t* and a document *d*, based on the weight of *t* in *d*.
  - the weight can be the number of occurrences of *t* in *d*: **term frequency** => $tf_{t,d}$
- The **bag of words model**: a document is represented by frequencies of a number of terms, therefore a model in which the order of these terms is ignored
  - alternative: any weighting function that maps the number of occurrences of *t* in *d* to a positive real value
- But: are all terms equally important?

Inverse document frequency

- How to discriminate between documents in a collection?
  - If a term *t* is very common in a collection of documents, its discriminating power in the collection diminishes ➔ scale down the frequency of *t* in a document *d* by the **total # of occurrences of *t* in the whole collection**: cf_t
  - Instead and more interesting ➔ scale down a term *t* frequency by the **number of documents in the collection *t* occurs in**: df_t.
  - Which query has a higher boost: on *try* or on *insurance*?
  - If *N* = the total number of documents in a collection, the **inverse document frequency** (idf) of a term *t*:

$$idf_t = \log(N/df_t)$$

Word	cf	df
try	10422	8760
insurance	10440	3997

term	df_t	idf_t
car	18,165	1.65
auto	6723	2.08
insurance	19,241	1.62
best	25,235	1.5

tf-idf weighting

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} * \text{idf}_t$$

- *tf-idf* assigns to term *t* a weight in document *d* which is:
  - very high when *t* occurs many times within a small number of documents (thus lending high discriminating power to those documents);
  - lower when the term occurs fewer times in a document, or occurs in many documents (thus offering a less pronounced relevance signal);
  - lowest when the term occurs in virtually all documents

Using *tf-idf* in vector models

- The **overlap score measure**: the score of a document *d* with respect to a query *q* is the sum, over all query terms, of the number of times each of the query terms occurs in *d*.
  - we can refine this idea so that we add up not the number of occurrences of each query term *t* in *d*, but instead the *tf-idf* weight of each term in *d*.

$$\text{Score}(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d}.$$

- We may view each document as a vector with one component corresponding to each term in the dictionary, together with a weight for each component that is given by *tf-idf*.
  - for dictionary terms that do not occur in a document, this weight is zero.

Apply Boolean and vector models to some of our problems

- P4. Automatic magazines metadata filling
- P5. OCR on printed texts and manuscripts

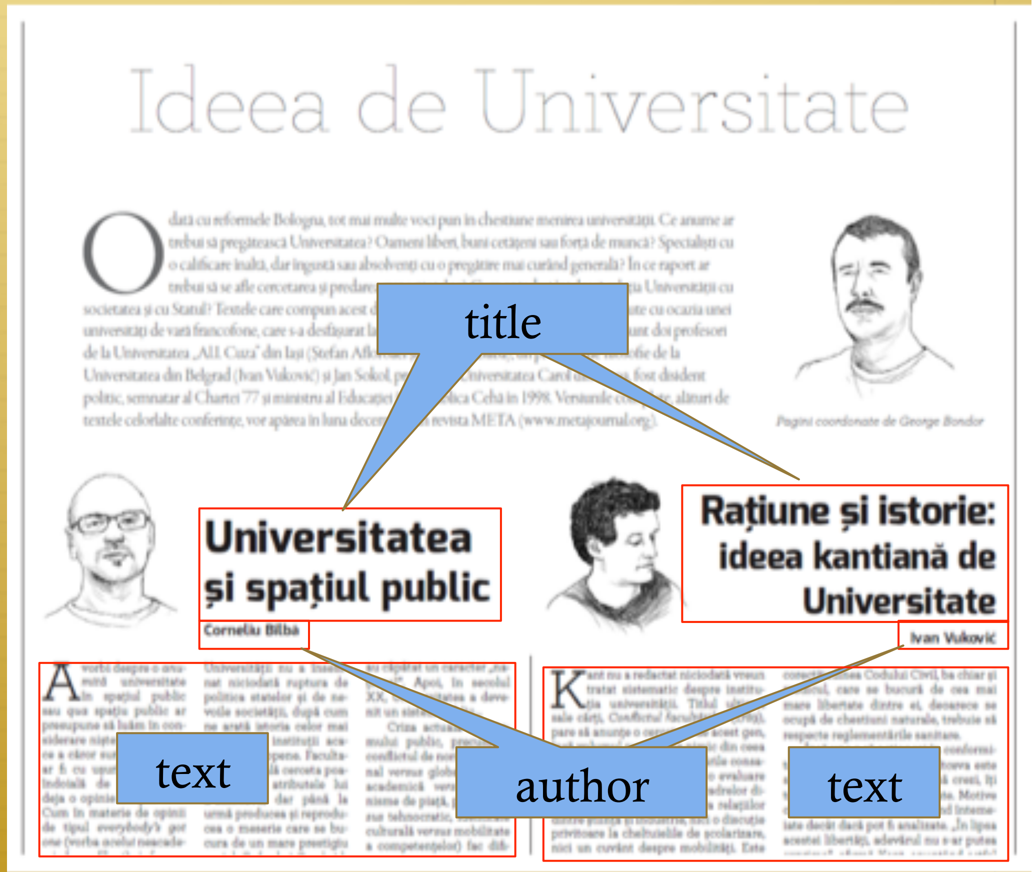
P4. Automatic magazines metadata filling

- You have a set of triplets: magazine page images + extracted text from the articles + their metadata (title, author).
- Write a program able to learn to cut out the texts and their corresponding metadata.

Features in P4: characterizing titles, authors, bodies

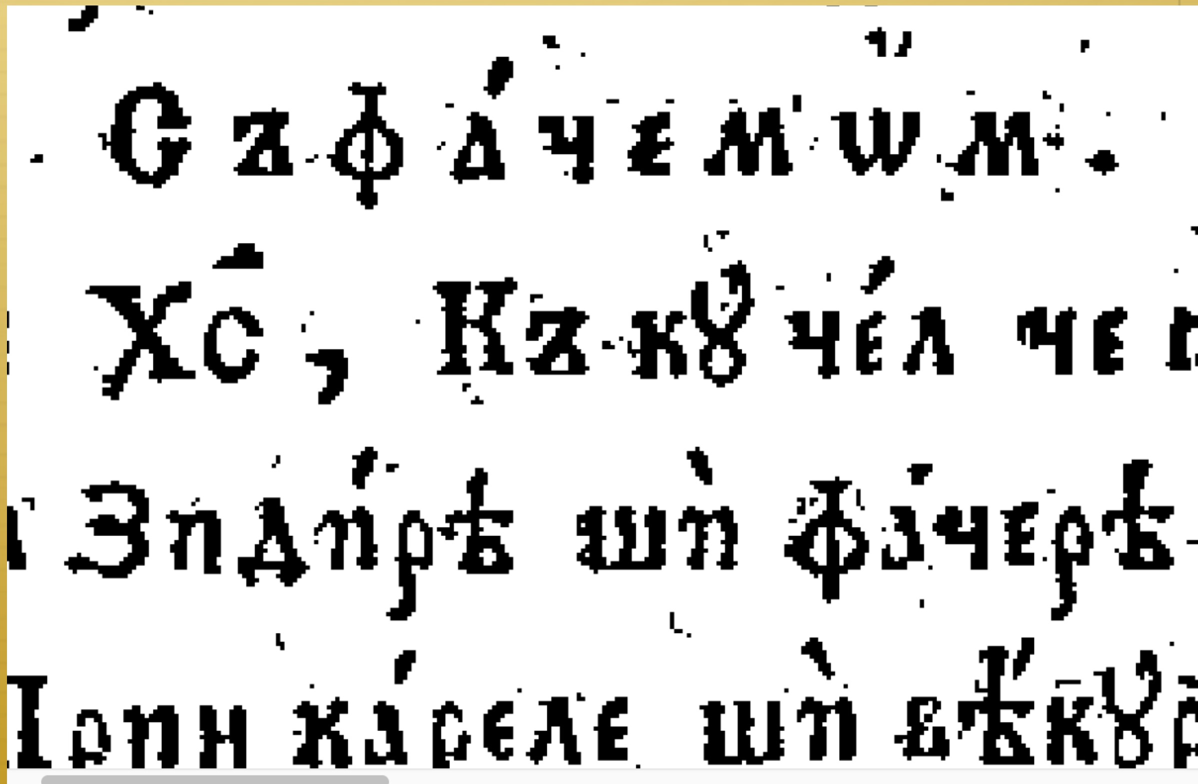
After segmentation in zones:

- font size
- length (# words)
- # lines in the segment
- # columns in the segment
- coordinates of the left-up corner
- coordinates of the right-down corner



P5. OCR on printed texts and manuscripts

- Target: decipher old Cyrillic-Romanian writings



Features in P5: recognizing Cyrillic characters

- parameters in Cartesian coordinates (after norming)

  (see https://en.wikipedia.org/wiki/Cartesian_coordinate_system)

  - rectangular area of the shape
  - ratio D_x/D_y
  - corners (upper left, upper right, lower left, lower right)
  - vertical stripes, horizontal stripes
  - occupied squared zones in the rectangular area
  - # of significant contiguous zones
  - position of weight centers of significant contiguous zones <x,y>
  - ratio L_x/L_y of significant contiguous zones
  - position of weight center of the total black area
  - # intersections of a 3-lines equidistant horizontal grid with the shape
  - # intersections of a 3-lines equidistant vertical grid with the shape
  - …

How to apply these models to our problems?

- We talked about: collections that contain documents, documents that contain terms and which can be grouped in classes
  - a common query: What are the documents of the collection in which the following terms are relevant…?
- In P5 we have: a text contains shapes which have features, shapes should be recognized as letters
  - a common query: What letter does this shape represent…?

But what are:
“the text”, “a shape”, “features”,
“a letter”?

“text”, “shapes”, “features”, “letters”

- A text contains shapes, the same as a collection of documents contains documents
- Shapes have features, the same as documents contain terms, because:
  - the same feature is common to more shapes ⇔ the same term is contained in more documents
  - some features are more relevant than others in the set of shapes ⇔ some terms are more relevant than others in the collection of documents
- More shapes represent the same letter ⇔ more documents can be grouped in classes of semantic similarity

And queries?...

- If:

  text ⇔ total collection of documents AND

  shape ⇔ document AND

  feature ⇔ term AND

  letter ⇔ classes of documents

- Then what is the equivalent of a query of the kind:

  What are the documents of the collection in which the following terms are relevant…?

  …

In our problem we put questions differently…

➔ What are the shapes of the text in which the following features are relevant…?

- In fact, our questions are more like this:

➔ To which letter does this shape resemble?

- which, translated back in the language of the model, yields:

➔ To which group of documents does this document belong?

- This is a **clustering** (**classification**) problem. We will come back to this later…

Let's see if the models presented can be applied to your problem

- Please answer the questions:
  - Can you find any resemblances between Boolean retrieval and your problem?
  - Can tf-idf be of help in your problem?