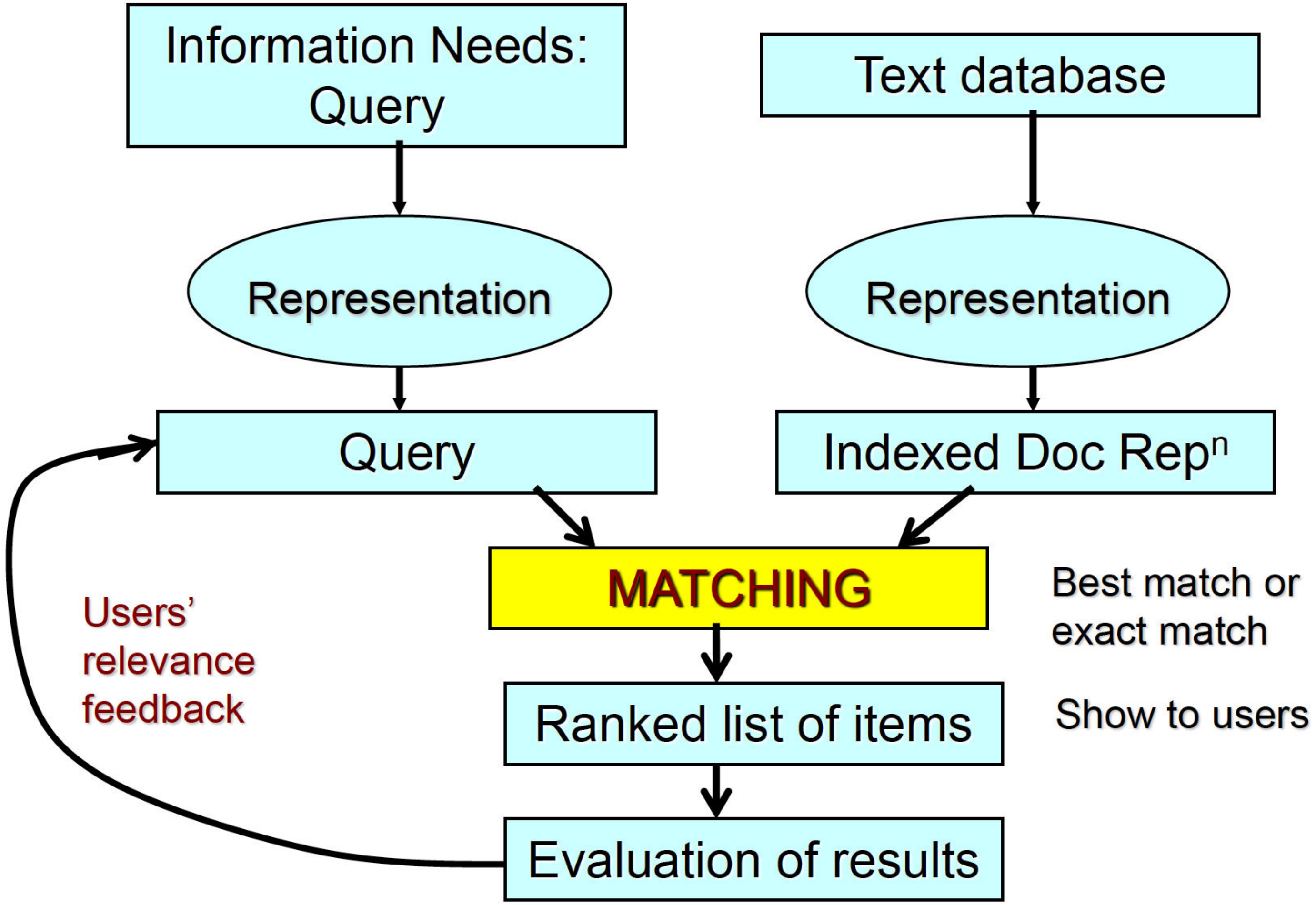
The task: Text Search

Given a textual query, rank a collection of documents according to relevancy.



The Dataset: Medline Collection

- A collection of articles from a medical journal
- 1033 documents, 30 (textual) queries
  - Query Example: “electron microscopy of lung or bronchi.”
- Three files:
  - med.all: contains the 1033 documents
  - med.que: contains the 30 queries
  - med.rel: contains the ground truth (which documents are relevant for each query)

What is provided

- Code (self-explanatory) to do the following operations:
    - Loading the dataset
    - Performing text search using TF-IDF
    - Evaluating performances with MAP
- A README.txt with the instructions to run the code

What you need to do

Implement the relevance feedback function (see lecture notes):

- Using vector adjustment only
- Using vector adjustment and query extension

- Performance (MAP) is expected to increase with the relevance feedback function
- Please, write you code in the file ‘relevance_feedback.py’

Relevance Feedback (vector adjustment only)

- Implement the **Pseudo** relevance feedback (see lecture notes):
  - User issues query Q
  - System returns top N relevant documents, $\{D_k\}, k = 1, \dots, N$
  - User provides relevance judgment, $\{R_k\}, k = 1, \dots, N_k$
  - Compute weights of terms to be added to query (vector adjustment):
    - $Q^{i+1} = Q^i + \alpha * \sum_{D_i \in R} D_i - \beta * \sum_{D_j \in NR} D_j$ (R: relevant documents, NR: non relevant)

Can also:

- perform more iterations
- try different values for weights α and β

Relevance Feedback (with query extension)

- Implement the **Pseudo** relevance feedback (see lecture notes):
  - User issues query Q
  - System returns top N relevant documents, $\{D_k\}, k = 1, \ldots, N$
  - User provides relevance judgment, $\{R_k\}, k = 1, \ldots, N_k$
  - Compute weights of terms to be added to query:
    - $Q^{i+1} = Q^i + \alpha * \sum_{D_i \in R} D_i - \beta * \sum_{D_j \in NR} D_j$ (R: relevant documents, NR: non relevant)
  - Select top n terms, $\{r_i\}, i = 2, \ldots, N$
  - $Q^{i+1} = Q^i + \{r_i\}$

A glance at the code

```
docs, queries, gt = load_data(args.docs, args.queries, args.gt)
vec_docs, vec_queries, tfidf_model = tf_idf(docs, queries, tokenize_text)
sim_matrix = cosine_similarity(vec_docs, vec_queries)
evaluate_retrieval(sim_matrix, gt, verbose=args.verbose)
```

- *sim_matrix[i,j]* contains the cosine similarity between document *i* and query *j*.
- The top k relevant document for query *q* can be obtained with: *np.argsort(-rf_sim_matrix[:, i])[:k]*
- *Evaluate_retrieval(…)* computes and prints the Mean Average Precision

A glance at the code (2)

```python
def relevance_feedback(vec_docs, vec_queries, sim, n=10):
    """
    relevance feedback
    Parameters
        ----------
        vec_docs: sparse array,
            tfidf vectors for documents. Each row corresponds to a document.
        vec_queries: sparse array,
            tfidf vectors for queries. Each row corresponds to a document.
        sim: numpy array,
            matrix of similarities scores between documents (rows) and queries (columns)
        n: integer
            number of documents to assume relevant/non relevant

    Returns
    -------
    rf_sim : numpy array
        matrix of similarities scores between documents (rows) and updated queries (columns)
    """
    rf_sim = sim # change
    return rf_sim
```