Master Degree in Computer Science
Master Degree in Data Science and Economics

Information Retrieval

UNIVERSITÀ
DEGLI STUDI
DI MILANO

LA STATALE

Relevance feedback and query expansion

Prof. Alfio Ferrara

Department of Computer Science, Università degli Studi di Milano
Room 7012 via Celoria 18, 20133 Milano, Italia alfio.ferrara@unimi.it

sed noli modo

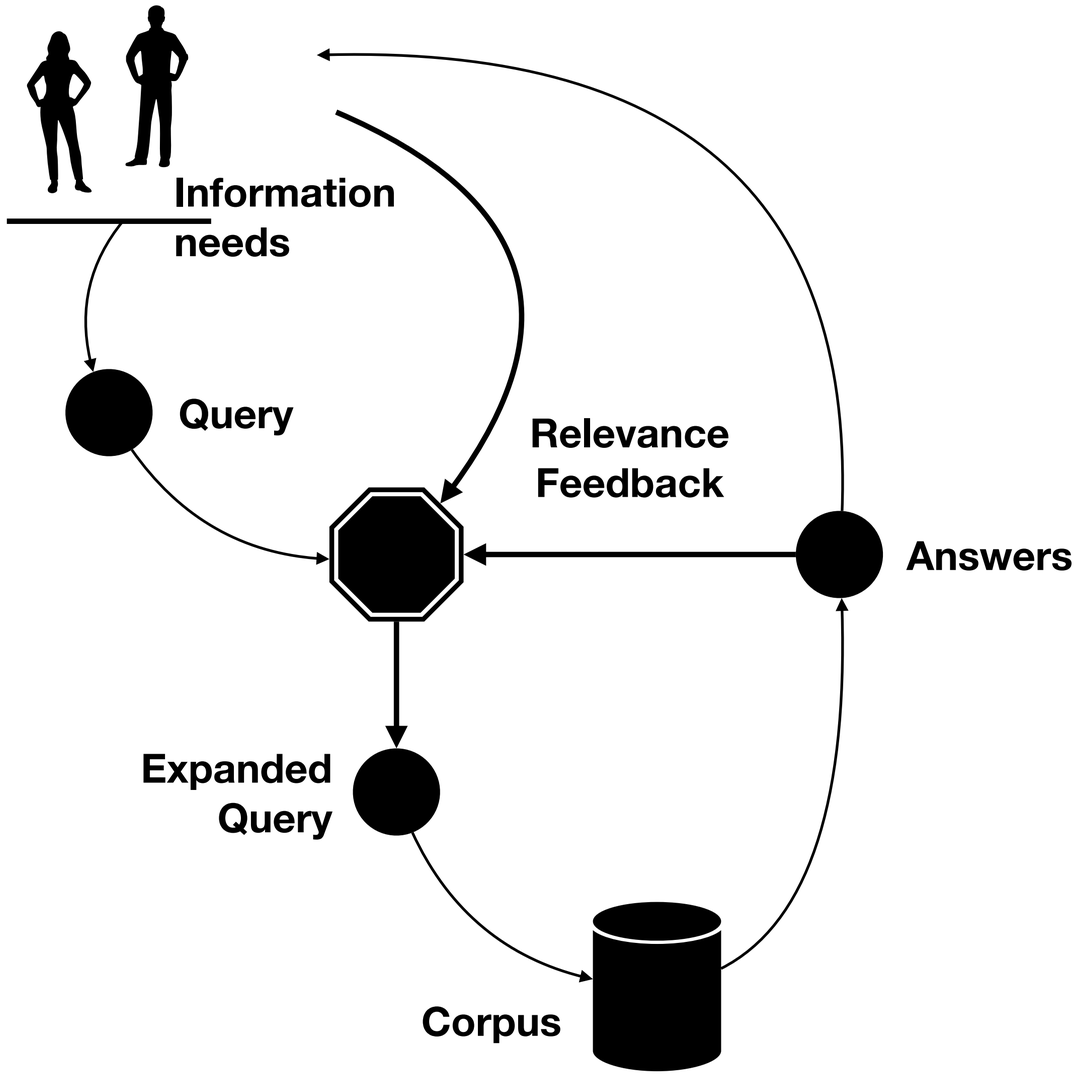
Definitions

Relevance feedback

Any feedback we can collect about the correctness of query answers (e.g., user feedback, evaluation outcomes, indirect feedback about the usefulness of answers)

Query expansion

Transform user queries by exploiting the relevance feedback and adding information to queries in order to improve the system performances



Query Expansion

Methods for query expansion are divided in two families:

LOCAL METHODS

- Relevance feedback
- Pseudo-relevance feedback
- Indirect feedback

GLOBAL METHODS

- Query expansion with a thesaurus (i.e., WordNet)
- Query expansion with automatic thesaurus generation
- Spelling correction

Local methods: relevance feedback

Given a set R of relevant results and a set N of non relevant results, we want to find an optimal query vector $\vec{q}_o$ that maximizes the similarity with relevant results and minimizes that with non relevant results

$$\vec{q}_o = argmax_{\vec{q}}[sim(\vec{q}, R) - sim(\vec{q}, N)]$$

Using cosine similarity

$$\vec{q}_o = \frac{1}{|R|} \sum_{\vec{d}_i \in R} \vec{d}_i - \frac{1}{|N|} \sum_{\vec{d}_j \in N} \vec{d}_j$$

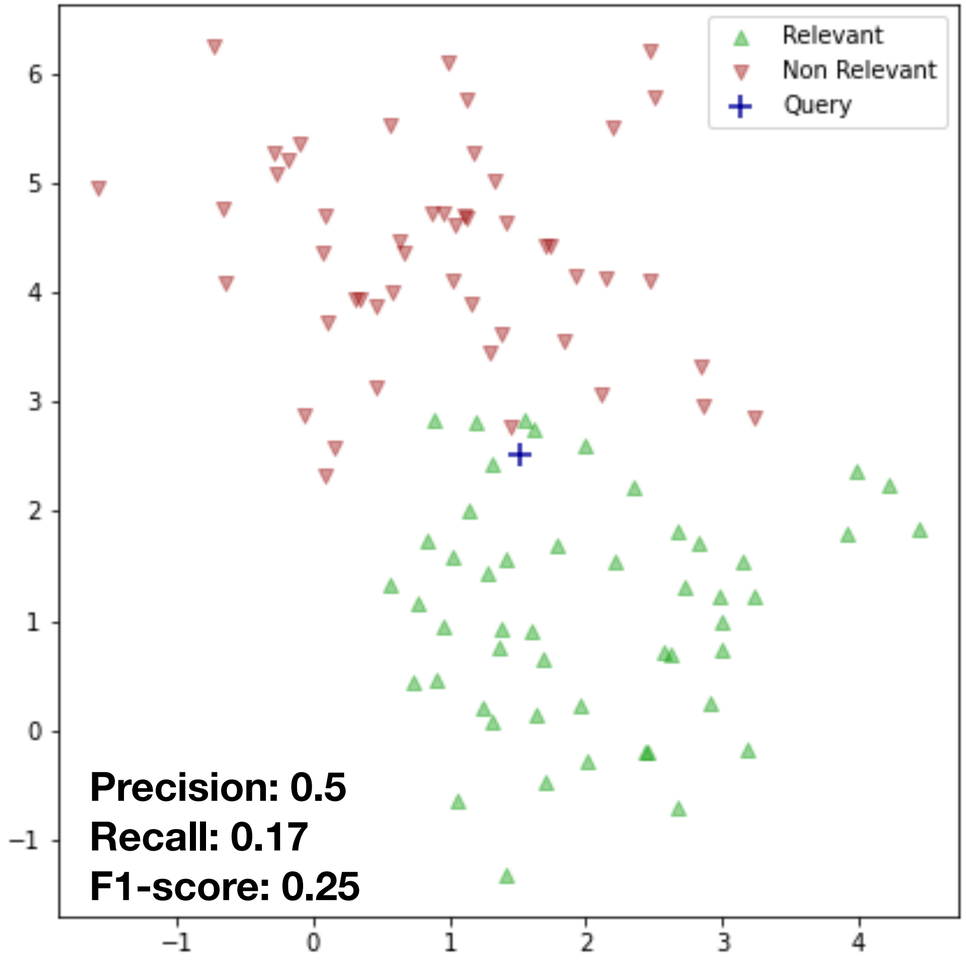
Local methods: the Rocchio algorithm

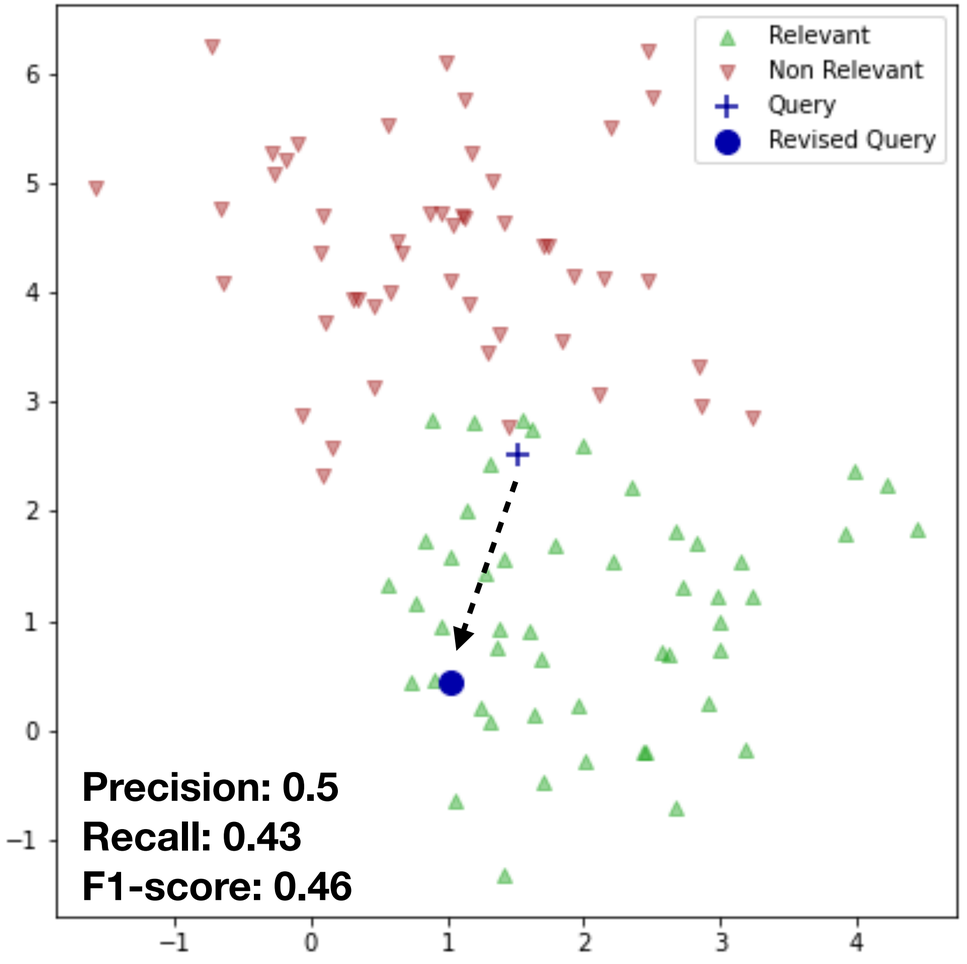
In a real IR query context, we have a user query and partial knowledge of known relevant and non relevant documents. The algorithm proposes using the modified query $\vec{q}_m$

$$\vec{q}_m = \alpha\vec{q} + \beta\frac{1}{R}\sum_{\vec{d}_i \in R}\vec{d}_i - \gamma\frac{1}{N}\sum_{\vec{d}_j \in N}\vec{d}_j$$

The idea is to push the new query towards the centroid of the true positive results, according to the relevance feedback, and far from the false positive ones. The parameters α, β, γ are used to balance the role of each component in the formula.

Local methods: the Rocchio algorithm





Local methods: other feedback

Pseudo-relevance feedback

Instead of collecting a real feedback for a query, we execute the query and we assume that the top k results are indeed relevant. Then, we refine the query.

Indirect relevance feedback

In this case, we use as feedback the analysis of the behavior of users with the query results, such as we count the number of visited documents after a query or other measures of popularity of a document given a query.

Global methods: query expansion

The idea of global methods is to **expand the query** by **adding new information** that makes the query more **accurate** and **explicit** with respect to the information need and **more similar to the target documents**

The additional information may be:

- New **terminology**
- New **query vectors** that are combined with the original query vector

We can acquire the additional information

- From a **syntactic analysis of the original query** (e.g., spelling correction, automatic translation)
- From **dictionaries, thesauri or knowledge bases**
- From a **corpus of reference documents** (e.g., pre-annotated relevant documents, query logs, feedback on previous queries)

Spelling correction

Pretty often, query mistakes are due to the fact that the query contains syntactical or morphological errors, often due to **spelling errors**

In order to automatically correct the query we need to **select the correct version of the original query** among **many possible candidates**

In particular we need I) to choose the ***nearest* candidate** (thus we need a notion of **proximity between a pair of strings**) and ii) break ties by choosing the **most common alternative** (which could depend on the previous queries or the corpus)

String similarity

Hall, P. A., & Dowling, G. R. (1980). Approximate string matching. ACM computing surveys (CSUR), 12(4), 381-402.

Edit Distance (aka Levenshtein distance)

The distance between two strings is computed by counting the minimum number of operations (i.e., insert, delete, or substitute chars) are required to transform one string into the other

```
def dist(a, b):
    if len(a) == 0:
        return len(b)
    if len(b) == 0:
        return len(a)
    if a[-1] == b[-1]:
        k = 0
    else:
        k = 1
    d = min([
         dist(a[:-1], b) + 1,
         dist(a, b[:-1]) + 1,
         dist(a[:-1], a[:-1]) + k
        ])
    return d
```

		F	L	I	N	T
	0	1	2	3	4	5
A	1	1	2	3	4	5
L	2	2	1			
I	3					
B	4					
I	5					

Min of [(x, y-1); (x-1, y-1); (x-1, y)] + 1if x != y, 0 otherwise

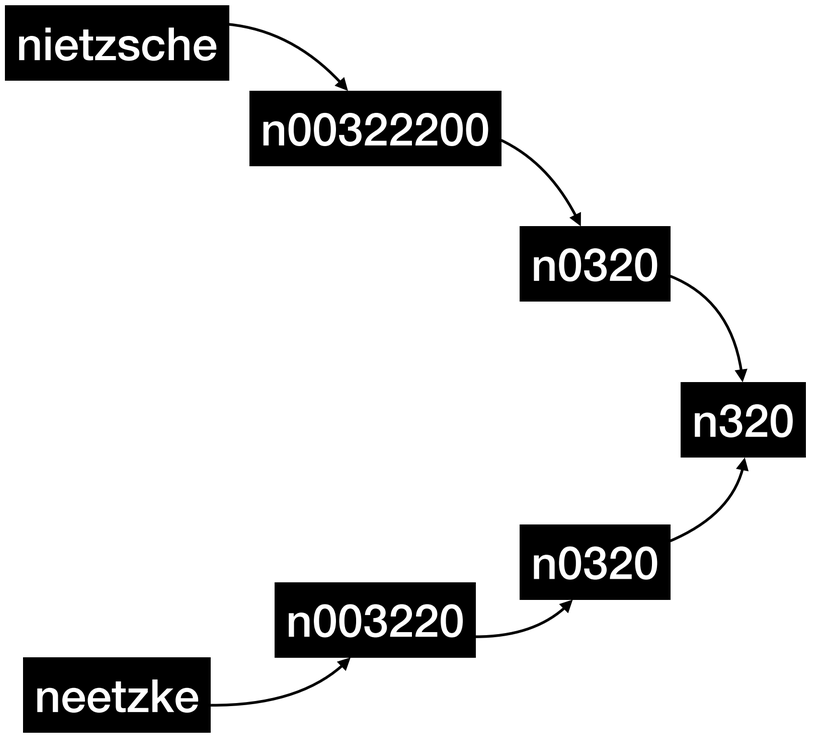
Phonetic correction

To minimize errors in queries due to phonetic misspelling, the main idea is to generate, for each term, a ***phonetic hash*** so that similar-sounding terms hash to the same value.

These algorithms are called ***Soundex*** and are, obviously, language-dependent

Example

- Retain the first letter of the term
- Change all occurrences of the following letters to ’0’ (zero): ’A’, E’, ’I’, ’O’, ’U’, ’H’, ’W’, ’Y’.
- Change letters to digits as follows:
  - B, F, P, V to 1; C, G, J, K, Q, S, X, Z to 2; D,T to 3; L to 4; M, N to 5.; R to 6
- Repeatedly remove one out of each pair of consecutive identical digits
- Remove all zeros from the resulting string. Pad the resulting string with trailing zeros and return the first four positions, which will consist of a letter followed by three digits



K-grams matching & classification

In this approach to string matching we aim at minimizing the impact of differences due to misspelling on the similarity between two strings

This can be done by **indexing k-grams of characters** in the two strings and then exploiting the index to vectorize string the the space of k-grams

Vectors are then matched by cosine similarity to choose alternatives in case of misspelling

	#sp	spl	pla	lay	ay#	y#e	ple	ley	ey#	lai	ai#	i#e	#sg	sga	gam	ame	me#	e#e
play	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
pley	1.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
plai	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
game	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0

	play	pley	plai	game
play	1.0	0.50	0.50	0.0
pley	0.5	1.00	0.33	0.0
plai	0.5	0.33	1.00	0.0
game	0.0	0.00	0.00	1.0

Sequence-to-sequence learning

As we will see later in this course, deep learning techniques can be used to learn ho to map a sequence onto another sequence

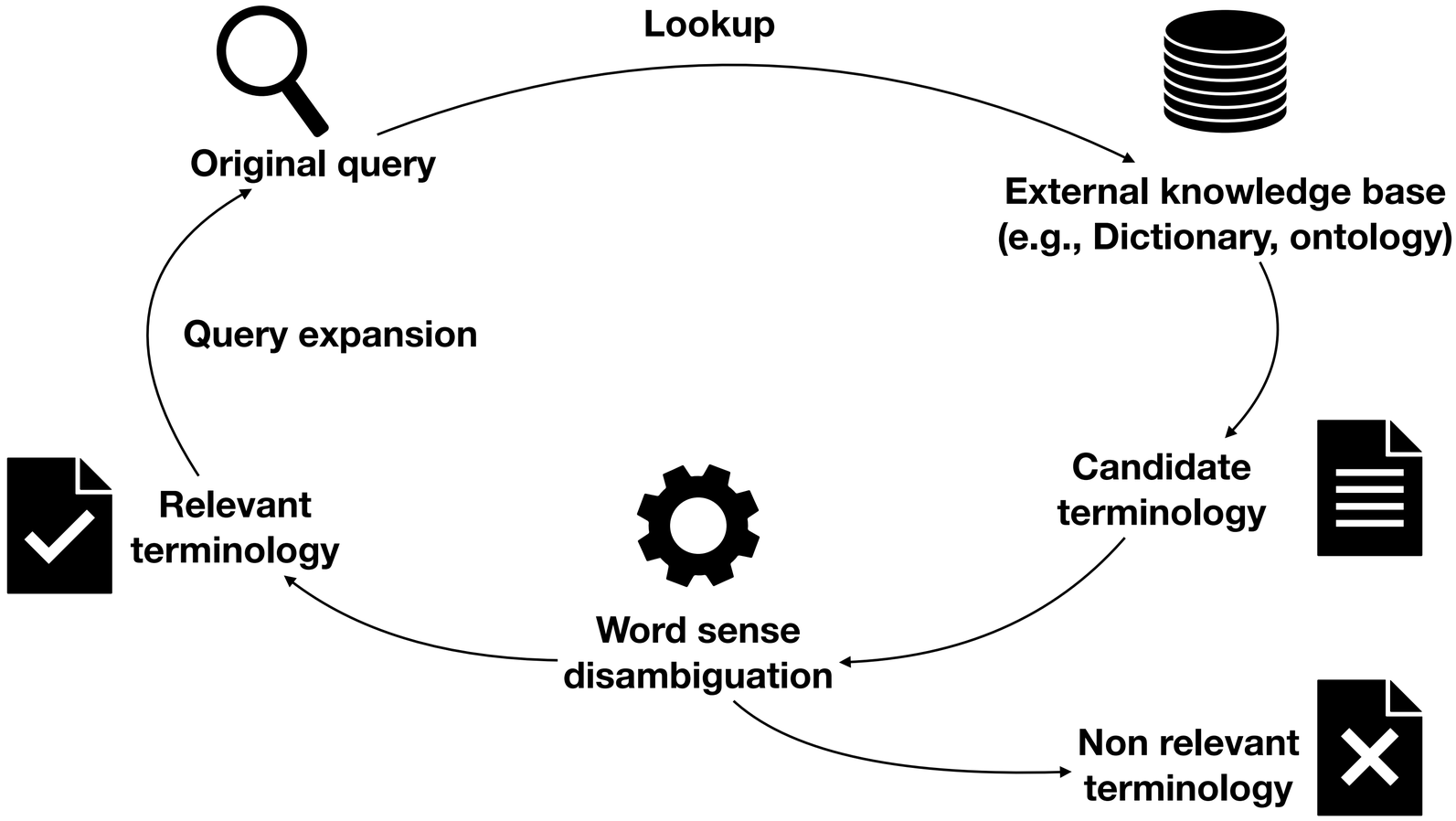
This can be used to fit a model to learn how to map incorrect and/or misspelled words into their correct counterpart

For the time being, you can just have a look at a reference and a tutorial. We will discuss the topic in the next lectures.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. arXiv preprint arXiv:1409.3215.

Tutorial: A ten-minute introduction to sequence-to-sequence learning in Keras (https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html)

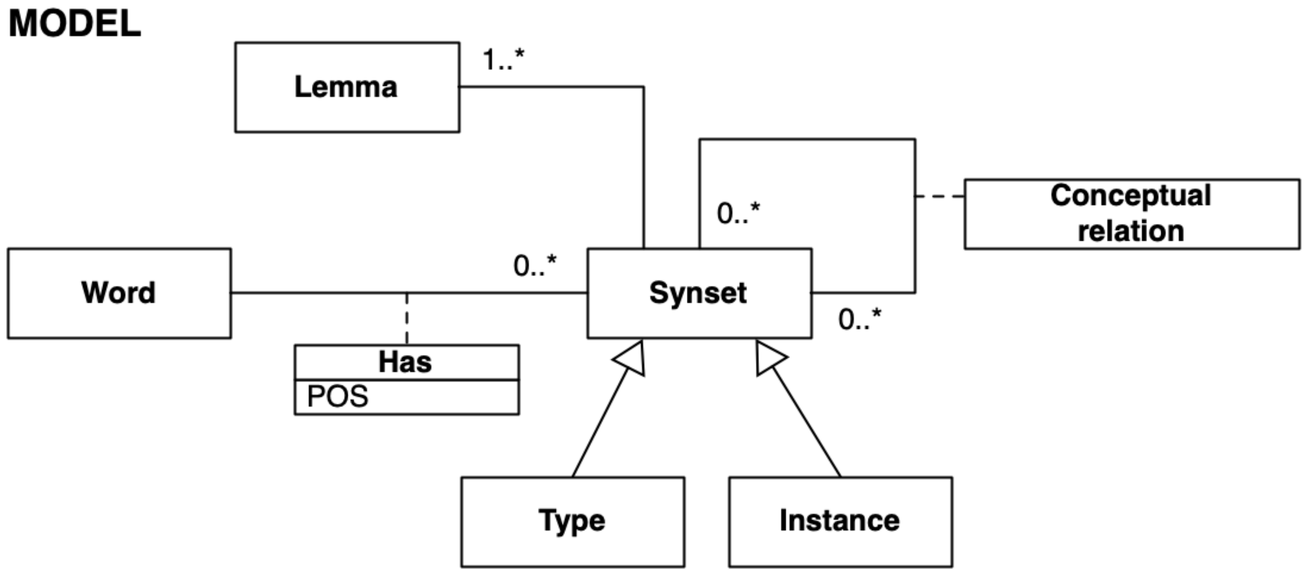
Global methods: dictionaries & knowledge bases

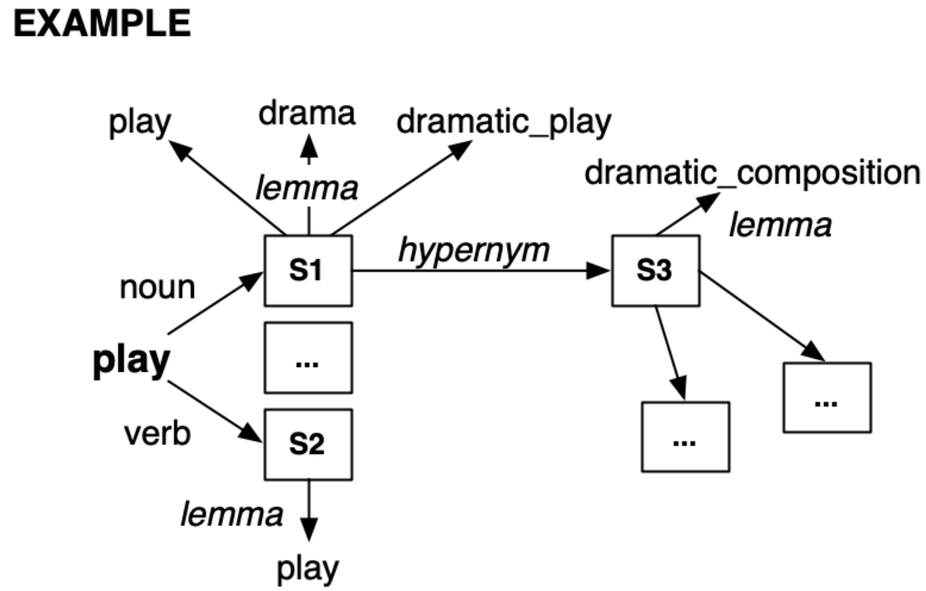


WordNet

WordNet is a lexical database for English (and other languages)

Miller, G. A. (1995). WordNet: a lexical database for English. Communications of the ACM, 38(11), 39-41.





WordNet

```
q = 'President Lincoln'
```

```
n_syns = wn.synsets('President', pos=wn.NOUN)
for n in n_syns:
    print(n.name(), n.definition())

president.n.01 an executive officer of a firm
or corporation
president_of_the_united_states.n.01 the
person who holds the office of head of state
of the United States government
president.n.03 the chief executive of a
republic
president.n.04 the officer who presides at
the meetings of an organization
president.n.05 the head administrative
officer of a college or university
president_of_the_united_states.n.02 the
office of the United States head of state
```

```
n_syns = wn.synsets('Lincoln', pos=wn.NOUN)
for n in n_syns:
    print(n.name(), n.definition())

lincoln.n.01 16th President of the United
States; saved the Union during the American
Civil War and emancipated the slaves; was
assassinated by Booth (1809-1865)
lincoln.n.02 capital of the state of
Nebraska; located in southeastern Nebraska;
site of the University of Nebraska
lincoln.n.03 long-wooled mutton sheep
originally from Lincolnshire
```

Wikidata

Wikidata is a free and open knowledge base that can be read and edited by both humans and machines. Wikidata acts as central storage for the structured data of its Wikimedia sister projects including Wikipedia, Wikivoyage, Wiktionary, Wikisource, and others.

https://www.wikidata.org/

The Wikidata Sparql endpoint supports queries over the Wikidata knowledge base either for searching properties of entities (e.g., the President Lincoln) or for searching lexemes that can be used to expand the query terminology

```
SELECT ?l ?lemma ?languageLabel  WHERE {
  ?l a ontolex:LexicalEntry;
       dct:language ?language;
       wikibase:lemma ?lemma .
  ?l wdt:P31 wd:Q376431.
  SERVICE wikibase:label { bd:serviceParam
wikibase:language "[AUTO_LANGUAGE],en". }
}
ORDER BY ?languageLabel
```

Instance of

color

Global methods: statistical terminology expansion

Given a set R of relevant documents for a query q and a set N of non relevant documents, we can select relevant terminology for q by exploiting **pointwise Kullback–Leibler divergence score**

Let $p(w) = \frac{count(w, R)}{\sum_{w_i \in R} count(w_i, R)}$ be the probability of a word in R

Let $q(w) = \frac{count(w, N)}{\sum_{w_i \in N} count(w_i, N)}$ be the probability of a word in N

$$\delta_w(p \mid\mid q) = p(w)\log \frac{p(w)}{q(w)}$$