IST 2018/2019

Processamento e Recuperação de Informação

Lab 07: Learning to Rank

The Whoosh search engine provides three different ranking functions: *BM25*, *TF_IDF* (i.e. cosine similarity) and *Frequency*[1].

The following example code shows how to perform a query using the TF_IDF scoring function and obtain the corresponding textual similarity score:

```
from whoosh.index import open_dir
from whoosh.qparser import *
ix = open_dir("indexdir")
with ix.searcher(weighting=scoring.TF_IDF()) as searcher:
    query = QueryParser("content", ix.schema, group=OrGroup).parse(u"a query")
    results = searcher.search(query, limit=100)
    for i,r in enumerate(results):
        print r, results.score(i)
```

A similar procedure can be applied for the remaining ranking functions.

The goal of this exercise is to create a method for scoring the documents that combines the results from these three functions.

1

Using the Whoosh search engine with the document collection of the previous labs (files `pri_cfc.txt` and `pri_queries.txt`), implement a script that performs searches and returns the results ordered by a *linear combination* of the three textual similarities presented above.

The rank combination formula should be:

$$\mathrm{score}(q, d) = \alpha_1 \,\mathrm{bm25}(q, d) + \alpha_2 \cos(q, d) + \alpha_3 \,\mathrm{freq}(q, d)$$

where d is the document, q is the query, bm25 is the score obtained using the BM25 ranking function, cos is the score obtained using the TF_IDF ranking function, and freq is the score obtained using the Frequency ranking function.

Experiment with different values for weights α_1, α_2, and α_3 and try to find an improvement in Mean Average Precision (MAP) over the results achieved with each individual ranking function used in isolation.

[1] `https://whoosh.readthedocs.io/en/latest/api/scoring.html`

2

The goal now is to try a more sophisticated approach for combining the ranking functions used in the previous exercise. To this effect we will use a *pointwise Learning to Rank* (L2R) approach.

Our approach consists in training a Logistic Regression classifier[2] on the set of queries available in `pri_queries.txt`. More specifically, you should:

(a) Create a dataset for training and testing your L2R approach:

- Use 50% of the queries for training and 50% for testing (you can vary these percentages if you wish);
- With the training queries, build the *training dataset*. This dataset should contain, for each *(query q, document d)* pair, a set of classification instances with the format:

$$\mathrm{bm25}(q,d), \cos(q,d), \mathrm{freq}(q,d), r$$

  where $r = 1$ if document d is relevant for query q and $r = 0$ otherwise. You can store this data on a *numpy* array;
- Use the same number of relevant and non-relevant documents for each query.

(b) Use the training dataset to learn a Logistic Regression classifier:

- The three ranking scores will be your classification features and r will be the target class.

(c) Execute the queries on the testing set, using the Logistic Regression classifier as your ranking function. Measure Precision, Recall, and F_1 scores for the classifier, and measure the Mean Average Precision (MAP) for the produced ranking.

- To do this, first perform regular searches, using a each ranking function in isolation;
- The score of each ranking function will be the classification features and the classifier will return 1 if the document is relevant or 0 if otherwise;
- To order the resulting documents, you should use the *probability of the document being relevant*. This can be obtained through the `predict_proba` method of the `LogisticRegression` class.

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

3 Pen and Paper Exercise

Consider the problem of ranking search results with a learning-based method, leveraging the perceptron ranking algorithm introduced in the classes. Consider also a training dataset in which there are two user queries, each with three candidate documents that should be presented to the user. Each document-query pair is represented as a feature vector x, together with a relevance judgement y in a 3-point scale (i.e., $y \in 0, 1, 2, 3$):

- Query 1 and document 1 : $x =$¡$0.50, 0.00, 0.25, 0.75$¿$, y = 1$
- Query 1 and document 2 : $x =$¡$0.25, 0.00, 0.00, 0.25$¿$, y = 0$
- Query 1 and document 4 : $x =$¡$0.75, 0.25, 0.25, 1.00$¿$, y = 3$
- Query 2 and document 1 : $x =$¡$0.50, 0.00, 0.25, 1.00$¿$, y = 2$
- Query 2 and document 3 : $x =$¡$0.25, 0.00, 0.00, 0.50$¿$, y = 0$
- Query 2 and document 4 : $x =$¡$0.25, 0.00, 0.25, 0.50$¿$, y = 1$

(a) Simulate the execution of the training procedure for the perceptron ranking algorithm, considering one epoch over the training data. Consider an initial all-zeroes weight vector, and consider also an initial value of zero for each of the 3 thresholds associated to the possible values for the relevance estimates.

(b) Consider a new user query, for which there are two candidate documents. Each of the document-query pairs is represented by a feature vector x as follows:

- Query 3 and document 1 : $x =$¡$0.50, 0.00, 0.25, 0.25$¿
- Query 3 and document 2 : $x =$¡$0.50, 0.25, 0.50, 0.75$¿

Using the trained perceptron from the previous exercise, estimate which of the documents should be ranked higher.