Information Processing and Retrieval

Instituto Superior Técnico 2020

Lab 7: Learning to Rank

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

Let us recover the **pri_cfc** collection (files pri_cfc.txt and pri_queries.txt) introduced in earlier labs. Use Whoosh to index it and run queries using one of the provided ranking functions.

1 Guiding ranking

Let us create a method for scoring the documents that combines the results from the three scoring functions.

1.1

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:

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

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.

Assess how different values for weights α_1, α_2, and α_3 impact the Mean Average Precision (MAP) against each individual ranking function used in isolation.

1.2

The goal now is to try a more sophisticated approach for combining the ranking functions. 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.

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

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

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:

$$bm25(q,d), cos(q,d), freq(q,d), r$$

  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 the logistic regression classifier:

- the three ranking scores will be your classification features and r 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 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 Pen-and-paper exercise

Consider the problem of ranking search results with a learning-based method – the perceptron ranking algorithm.

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 ($y \in \{1, 2, 3, 4\}$):

query 1 and document 1 : $x =< 0.50, 0.00, 0.25, 0.75 >, y = 2$

query 1 and document 2 : $x =< 0.25, 0.00, 0.00, 0.25 >, y = 1$

query 1 and document 3 : $x =< 0.75, 0.25, 0.25, 1.00 >, y = 4$

query 2 and document 1 : $x =< 0.50, 0.00, 0.25, 1.00 >, y = 3$

query 2 and document 2 : $x =< 0.25, 0.00, 0.00, 0.50 >, y = 1$

query 2 and document 3 : $x =< 0.25, 0.00, 0.25, 0.50 >, y = 2$

(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 one 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.