Information Retrieval (Part VI)

[DAT640] Information Retrieval and Text Mining

Krisztian Balog
University of Stavanger

October 9, 2019

Today

- Learning-to-rank
- Elasticsearch

Learning-to-rank

Recap

- Classical retrieval models
  - Vector space model, BM25, LM
- Three main components
  - Term frequency
    - How many times query terms appear in the document
  - Document length
    - Any term is expected to occur more frequently in long document; account for differences in document length
  - Document frequency
    - How often the term appears in the entire collection

Additional factors

- So far: content-based matching
- Many additional signals, e.g.,
  - Document quality
    - PageRank
    - SPAM score
    - ...
  - Implicit (click-based) feedback
    - How many times users clicked on a document given a query?
    - How many times this particular user clicked on a document given the query?
    - ...
  - ...

Discussion

Question

How to combine all these clues for ranking?

Machine learning for IR

- We hypothesize that the probability of relevance is related to some combination of *features*
  - Each feature is a clue or signal that can help determine relevance
- We employ machine learning to learn an "optimal" combination of features, based on training data
  - There may be several hundred features; impossible to tune by hand
  - Training data is (item, query, relevance) triples
- Modern systems (especially on the Web) use a great number of features
  - In 2008, Google was using over 200 features[1]

[1] The New York Times (2008-06-03)

Some example features

- Log frequency of query word in anchor text
- Query word in color on page?
- #images on page
- #outlinks on page
- PageRank
- URL length
- URL contains "$\sim$"?
- Page length
- ...

Simple example

- We assume that the relevance of a document is related to a linear combination of all the features:

$$\text{log}\,\frac{P(R=1|q,d)}{1-P(R=1|q,d)} = \beta_0 + \sum_{i=1}^{n} \beta_i x_i$$

  - x_i is the value of the i^{th} feature
  - β_i is the weight of the i^{th} feature
- This leads to the following probability of relevance:

$$P(R=1|q,d) = \frac{1}{1+\text{exp}\{-\beta_0 - \sum_{i=1}^{n} \beta_i x_i\}}$$

- This *logistic regression* method gives us an estimate in $[0,1]$

Learning-to-Rank (LTR)

- Learn a function automatically to rank items (documents) effectively
  - Training data: (item, query, relevance) triples
  - Output: ranking function $h(q, d)$
- Three main groups of approaches
  - Pointwise
  - Pairwise
  - Listwise

Pointwise LTR

- Specifying whether a document is relevant (binary) or specifying a degree of relevance
    - Classification: Predict a categorical (unordered) output value (relevant or not)
    - Regression: Predict an ordered or continuous output value (degree of relevance) $\Leftarrow$
- All the standard classification/regression algorithms can be directly used
- Note: classical retrieval models are also point-wise: $score(q, d)$

Pairwise LTR

- The learning function is based on a pair of items
  - Given two documents, classify which of the two should be ranked at a higher position
  - I.e., learning relative preference
- E.g., Ranking SVM, LambdaMART, RankNet

Listwise LTR

- The ranking function is based on a ranked list of items
  - Given two ranked list of the same items, which is better?
- Directly optimizes a retrieval metric
  - Need a loss function on a list of documents
  - Can get fairly complex compared to pointwise or pairwise approaches
- Challenge is scale: huge number of potential lists
- E.g., AdaRank, ListNet

How to?

- Develop a feature set
  - The most important step!
  - Usually problem dependent
- Choose a good ranking algorithm
  - E.g., Random Forests work well for pairwise LTR
- Training, validation, and testing
  - Similar to standard machine learning applications

Features for document retrieval

- Query features
  - Depend only on the query
- Document features
  - Depend only on the document
- Query-document features
  - Express the degree of matching between the query and the document

Query features

- Query length (number of terms)
- Sum of IDF scores of query terms in a given field (title, content, anchors, etc.)
- Total number of matching documents
- Number of named entities in the query
- ...

Document features

- Length of each document field (title, content, anchors, etc.)
- PageRank score
- Number of inlinks
- Number of outlinks
- Number of slash in URL
- Length of URL
- ...

Query-document features

- Retrieval score of a given document field (e.g., BM25, LM, TF-IDF)
- Sum of TF scores of query terms in a given document field (title, content, anchors, URL, etc)
- Retrieval score of the entire document (e.g., BM25F, MLM)
- ...

Practical considerations

- Feature normalization
- Computation cost
- Class imbalance

Feature normalization

- Feature values are often normalized to be in the $[0, 1]$ range *for a given query*
  - Esp. matching features that may be on different scales across queries because of query length
- Min-max normalization:

$$\tilde{x}_i = \frac{x_i - \mathsf{min}(x)}{\mathsf{max}(x) - \mathsf{min}(x)}$$

  - $x_1, \ldots, x_n$: original values for a given feature
  - $\tilde{x}_i$: normalized value for the i^{th} instance

Computation cost

- Implemented as a re-ranking mechanism (two-step retrieval)
  - Step 1 (initial ranking): Retrieve top-N candidate documents using a strong baseline approach (e.g., BM25)
  - Step 2 (re-ranking): Create feature vectors and re-rank these top-N candidates to arrive at the final ranking
- Document features may be computed offline
- Query and query-document features are computed online (at query time)
  - Avoid using too many expensive features!

Class imbalance

- Many more non-relevant than relevant instances
- Classifiers usually do not handle huge imbalance well
- Need to address by over- or under-sampling

Code

- Pointwise learning-to-rank using regression
- GitHub: `code/LTR.ipynb`

Elasticsearch

Elasticsearch

- Introduction
  - GitHub: `code/elasticsearch`
- Install Elasticsearch and go through the sample Jupyter notebook
  - GitHub: `code/elasticsearch/Elasticsearch.ipynb`
- Exercise
  - GitHub: `exercises/lecture_12/exercise_1.ipynb`

Assignment 2B

Reading

- Text Data Management and Analysis (Zhai&Massung)
  - Section 10.4