CS 410 Technology Review
Darius Nguyen (huy2)

Topic: Review of Ranking Models Implemented in Apache Solr

Introduction to Apache Solr

In information retrieval, one of the most common applications is building a search engine for users to retrieve information based on their queries. There are a few mainstream open-source platforms for building search engines, and among them, Apache Solr is one of the most popular and established.

Apache Solr, based on the framework of Apache Lucene, is a standalone enterprise search server with a REST-like API. Solr was originally developed using Java and built for REST-compliant. Developers can import documents into Solr ("indexing") via JSON, XML, CSV or binary over HTTP. Then, the imported documents can be queried via HTTP GET and receive JSON, XML, CSV or binary results (“Solr Features”).

Solr Retrieval Algorithms

To effectively retrieve documents based on a user’s query, Solr employs a ranking algorithm chosen by the developer. Solr leverages ranking algorithms from classes built in Lucene, which are defined in the *org.apache.lucene.search.similarities* class. Some of the popular models are as follows:

1. Vector Space Model/TF-IDF Weighting: This scoring model in Lucene involves several scoring factors:
   - tf - Term Frequency. The frequency with which a term appears in a document. Given a search query, the higher the term frequency, the higher the document score.
   - idf - Inverse Document Frequency. The rarer a term is across all documents in the index, the higher its contribution to the score.
   - coord - Coordination Factor. The more query terms that are found in a document, the higher its score.
   - fieldNorm - Field length. The more words that a field contains, the lower its score. This factor penalizes documents with longer field values.

   (“Solr Search Relevancy”)

2. Okapi BM25: Solr/Lucene allows applying the Okapi BM25 model using class BM25Similarity. The BM25Similarity class in Solr can be configured using 2 parameters: k1, b to fit the data set:
   - k1 - Controls non-linear term frequency normalization (saturation).
   - b - Controls to what degree document length normalizes tf values.

   (“Class BM25Similarity”)

The evolution of Machine Learning: Learning to Rank

In recent years, the growth of massive data volume, especially click-through data, has given rise to using Machine Learning in the search application, and the most popular ML technique in this area is Learning to Rank (LTR).

While traditional retrieval models produce a similarity score, then return documents based on the highest scores, LTR aims to come up with optimal ordering of a list of documents. LTR doesn't care much about the exact score that each document gets, but cares more about the relative ordering among all documents.

LTR first runs a simple query to get matching documents, then re-rank the top N documents using the scores from a different, complex query. Re-ranking the top N documents is done using trained machine learning models. Such sophisticated models aim to make more nuanced ranking decisions than standard ranking functions like TF-IDF or BM25 ("Learning to Rank-Solr Ref Guide"). It in fact employs these ranking functions as "features", as described below.

An LTR model can combine many features and learn what weights should be applied to provide the best ranking, based on the training data. Some common features are:

- BM25 score
- PageRank
- Term Frequency in document title/anchor text
- Time since last updated

Learning to Rank Models: RankSVM and LambdaMART

To combine the features into a final ranking model, there are two prominent methods: RankSVM & LambdaMART. RankSVM is a pairwise linear model, while LambdaMART is a listwise non-linear model.

RankSVM

The ranking SVM algorithm was published by Thorsten Joachims in 2002. This method uses a mapping function to describe the match between a search query and the features of each of the possible results. Each data pair, such as a search query and clicked webpage, for example, is projected onto a feature space. These features are combined with the corresponding click-through data, which can act as a proxy for how relevant a page is for a specific query, and can then be used as the training data for the Ranking SVM algorithm ("Ranking SVM").

In general, the steps of the Ranking SVM are:

1. Map the similarities between queries and the clicked pages onto a certain feature space.
2. Calculate the distances between any two of the vectors obtained above.
3. Optimize the problem using regular SVM solver.

In Solr, RankSVM is implemented in class *org.apache.solr.ltr.model.LinearModel*. The class can be instantiated following this example:

```
{
"class":"org.apache.solr.ltr.ranking.RankS
VMModel",
   "name":"myModelName",
   "features":[
      { "name": "userTextTitleMatch"},
      { "name": "originalScore"},
      { "name": "isBook"}
   ],
   "params":{
      "weights": {
         "userTextTitleMatch": 1.0,
         "originalScore": 0.5,
         "isBook": 0.1
      }
}
                                (Benedetti)
```

Given a list of features, the 'score' method in this class calculates and returns a ranking score. For instance, given the documents D1=[1.0, 100, 1], D2=[0.0, 80, 1], the 'score' method returns
*score(D1) = 1.0 * userTextTitleMatch(1.0) + 0.5 * originalScore(100) + 0.1 * isBook(1.0) = 51.1*
*score(D2) = 1.0 * userTextTitleMatch(0.0) + 0.5 * originalScore(80) + 0.1 * isBook(1.0) = 40.1*

This shows D1 is more relevant (Benedetti).

LambdaMART

LambdaMART is the boosted tree version of LambdaRank, which is based on RankNet. RankNet, LambdaRank and LambdaMART were developed by Chris Burges et al. at Microsoft Research. RankNet, LambdaRank, and LambdaMART have proven to be very successful in solving real world ranking problems. Ranking is transformed into a pairwise classification or regression problem. That means you look at pairs of items at a time, come up with the optimal ordering for that pair of items, and then use it to come up with the final ranking for all the results.

MART is a boosted tree model in which the output of the model is a linear combination of the outputs of a set of regression trees. LambdaMART uses gradient boosted decision trees using a cost function derived from LambdaRank for solving a ranking task (Burges 2010).

In Solr, LambdaMART can be invoked by calling class *org.apache.solr.ltr.model.MultipleAdditiveTreesModel*, such as in the following example:

```
{
"class":"org.apache.solr.ltr.ranking.Lambd
aMARTModel",
   "name":"lambdamartmodel",
   "features":[
      { "name": "userTextTitleMatch"},
      { "name": "originalScore"}
   ],
   "params":{
      "trees": [
         {
            "weight" : 1,
            "root": {
               "feature":
"userTextTitleMatch",
               "threshold": 0.5,
               "left" : {
                  "value" : -100
               },
               "right": {
                  "feature" : "originalScore",
                  "threshold": 10.0,
                  "left" : {
                     "value" : 50
                  },
                  "right" : {
                     "value" : 75
                  }
               }
            }
         },
         {
            "weight" : 2,
            "root": {
               "value" : -10
            }
         }
      ]
   }
}
```

(Benedetti)

Given the documents D1= [1, 9] and D2 = [0, 10], the LambdaMART model return the scores:

Score(D1) = 1 (userTextTitleMatch (1) > 0.5 go right , originalScore (9) < 10 = 50) +*
*2 * -10 = 30*
*Score(D2) = 1 *(userTextTitleMatch(0) <= 0.5 = -100) +*
*2 * -10 = -120*

So D1 is more relevant, according this model.

Conclusion

Apache Solr and Lucene have been around for 21 years, but it is still a leading open-source platform for enterprise search. Solr is used at many leading internet companies that heavily rely on search, such as Amazon, Bloomberg, Apple, etc.

Historically, retrieval models such as TF-IDF Weighting (Vector Space Model) and Okapi BM25 have been implemented in Lucene libraries and provided robust ranking capabilities for simple search use cases. However, with the meteoric rise of web click-through data, new Machine Learning ranking models, especially Learning to Rank, have become superior in complex scenarios where a combination of multiple features in ranking is needed.

Learning to Rank has grown manifold since its inception, with the current generation embodying various algorithms, including RankSVM and LambdaMART as the most popular. While RankSVM provides a straightforward way to linearly combine features with a set of weights, LambdaMART allows using decision trees to solve for an optimal rank among documents. Empirically, LambdaMART has proven to be robust by showing better results than its predecessors, LambdaRank and RankNet.

References

Benedetti , Alessandro, “Solr Is Learning To Rank Better – Part 2 – Model Training”, https://sease.io/2016/08/apache-solr-learning-to-rank-part-2-model-training.html

Burges, Christopher, “From RankNet to LambdaRank to LambdaMART: An Overview” https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/

“Class BM25Similarity”, https://lucene.apache.org/core/7_0_1/core/org/apache/lucene/search/similarities/BM25Similarity.html

“Learning To Rank-Solr Ref Guide”, https://lucene.apache.org/solr/guide/6_6/learning-to-rank.html

“Solr Features”, https://lucene.apache.org/solr/features.html

“Solr Search Relevancy”, http://www.solrtutorial.com/solr-search-relevancy.html