Learning to Rank - Tutorial @ SEPLN 2012

Parth Gupta

NLE Lab, ELiRF, UPV

September 5, 2012
Outline

Introduction
Corpus
Evaluation Metrics
Approaches
Applications
Letor Datasets
Xapian Letor API
Summary
References
What is Ranking?

- For the given, request and collections of offerings we give offerings in particular relevance order.

In case of Document retrieval

- Input: Query and collection of documents
- Output: Ranked list of documents

Learning to Rank

- Machine Learning for Information Retrieval (IR)
- Specifically: Learning the Ranking Model
Basic IR Concepts

Crawling

▷ Crawl the necessary part of the Web and prepare a static collection of documents

Indexing

▷ preprocess to convert it into raw text format (ASCII or UTF-8)
▷ Stop-word removal [Term Pipeline]
▷ Stemming [Term Pipeline] Store relevant information of terms and documents like term frequency (TF) [doc and collection] and document length in direct and inverted index.
Query Normalisation

- Pass the query from the same pipeline

Ranking

- The simplest yet powerful model is TF-IDF

\[
Score(Q, D) = \sum_{i=1}^{n} tf(q_i, D) * idf(q_i)
\]

- \(tf(q_i, D)\) = Frequency of Term \(q_i\) in \(D\).
- \(idf(q_i) = \log(\frac{N}{\# \text{ of docs containing } q_i})\)
**tf − idf Example**

- Doc1 = Jaume I University is in Castellón
- Doc2 = Jaume I University is in Spain
- Doc3 = UPV is in Valencia
- Q = Jaume Spain

**Ranking**

- Score(Q, Doc1) = (1+0)*(0.64) = 0.64 Rank - 2
- Score(Q, Doc2) = (1+1)*(0.64) = 1.28 Rank - 1
- Score(Q, Doc3) = (0+0)*(0.64) = 0.0 Rank - 3
Other Unsupervised Ranking Models

- BM25 - Probabilistic Model
- Language Model for IR
Scope of This Tutorial

- Learning to Rank Problem Definition
- Approaches to Learning to Rank
- Applications of LTR to NLP and IR
- Research Trends with LTR
- Hands on with LTR
Conventional Unsupervised Ranking Model

Indexed Document Repository

D=\{d_i\}

Ranked List of Documents

--d1--
--d2--
--d3--
...
--dn--

Query → Ranking Model → Ranked List of Documents
Motivation

Can We Learn?

RankList A

RankList B

Irrelevant Doc
Relevant Doc

SEPLN '12 NLE Lab, ELiRF, UPV

Learning to Rank
Supervised Learning-to-Rank (LetoR) Model

[Qin et al., 2010]
Practical Ranking

Query

Un-supervised Ranker

Learning to Rank

Extract Features

Supervised Ranker

RankList

RankList
Training Process (*fig. taken from [Li, 2009]*)

1. Data Labeling (rank)

2. Feature Extraction

3. Learning $f(x)$
Testing Process (fig. taken from [Li, 2009])

1. Data Labeling (rank)
2. Feature Extraction
3. Ranking with $f(x)$
4. Evaluation
Letor 4.0 Dataset [Qin et al., 2010]

- Gov2 web page collection and two query sets of Million Query (MQ) track of TREC07 and TREC08.

- Given in form of query document pairs.

```
2 qid:10032 1:0.056537 2:0.000000 ... 45:0.000000 46:0.076923 #docid = GX029-35-5894638
0 qid:10032 1:0.279152 2:0.000000 ... 45:0.250000 46:1.000000 #docid = GX030-77-6315042
1 qid:10032 1:0.593640 2:1.000000 ... 45:0.500000 46:0.000000 #docid = GX256-43-0740276
```
Issues in Learning to Rank

- Relevance Labels - manual judgments vs. click-through data
- Feature Extraction - (Q, D, Q+D)
- Evaluation Measure - multi-level relevance labels
Why so popular?

- Easy to incorporate new features
  - age of the document
  - link information in web graph
  - click information
  - geographical information of the user etc.
- Easy to tune parameters of the ranking model automatically
Evaluation Metrics

Mean Average Precision (MAP)

$$\text{MAP} = \frac{\sum_{k=1}^{n} (P(k) \times \text{rel}(k))}{\text{Number of Relevant Documents}}$$

Normalized Discounted Cumulative Gain (nDCG)

$$\text{NDCG} @ n = Z_n * \sum_{j=1}^{n} \left[ \frac{2^c(j) - 1}{\log(1 + j)} \right]$$
Approaches

- Pointwise Approach
  Individual doc is used for learning

- Pairwise Approach
  Pairs of two documents for a query is used for learning

- Listwise Approach
  Whole ranked list for a query is used for learning
Pointwise Approach

- Use individual entity for learning even without query information
Pointwise Learning-to-Rank (LetoR) Model
Regression Approach

- Directly methods like regression or classification can be applied to solve the ranking problem.
- Consider relevance judgments as
  - Real values - in case of regression
  - Non-ordered categories - in case of classification
Regression Approach contd.

- Least Squares Retrieval Function [Fuhr, 1989]
- Loss Function can be LSE.
- Regression model is Multiple Linear Regression.

- Basic idea is to map the feature vector the the real value.
Regression Approach contd.

- Discriminative models [Nallapati, 2004]
- Here real score can be looked as probability of relevance given document and query.
- Here the weight vector can be estimated using gradient descent algorithm.
Classification Approach

- **BayesNetRank** [Gupta, 2011], **McRank** [Li et al., 2007]
- Solve the Ranking Problem using Classification
- Classes: Relevance Levels [0, 1, 2]
- Classification Model: Bayesian Networks (BayesNetRank) and Neural Networks (McRank)
- Ranking Function on top of Classification Probabilities.

Constraint

- Bayesian Networks accept only discrete data -> MDL based discretization scheme is used [Fayyad and Irani, 1993]
Bayesian Networks: Structure Learning

- \( E = \) set of Edges i.e. the structure of the Network
- So we have to Estimate
  \[
P(E|X) = \frac{P(X|E) \times P(E)}{P(X)}
  \]
- Where \( P(E) = \) Prior
- And \( P(x) \) is constant
- \( P(X|E) \) is likelihood term, Cooper Herkovitz Likelihood

\[
P(X|E) = \prod_{j=1}^{d} \prod_{i=1}^{q_j} \prod_{l=1}^{k_j} \Theta_{jil}^{n_k(x_j^{(i)}|\pi_j^{(l)})}
\]
Structure Learning contd..

$$P(X|E) = \prod_{j=1}^{d} \prod_{l=1}^{q_j} \prod_{i=1}^{k_j} \Theta_{jil}^{n_k(x_j^{(i)}|\pi_j^{(l)})}$$

- $d$ is the number of variables (No of nodes)
- $k_j$ is number of possible states that the variable $X_j$ can take
- $q_j$ is the number of possible parent configurations for variable $X_j$ and

$$\Theta_{jil} = P(X_j = x_j^{(i)}|\Pi_j = \pi_j^{(l)})$$
Structure Learning contd..

- The likelihood has to be computed for all the possible graph structure and the graph that maximizes it, is chosen.
- NP-Hard Problem: because possible number of DAGs grows super-exponentially with no. of nodes.
- e.g. for N=10, possible DAGs = $4.2 \times 10^{18}$
- So there are different ways to select subset of all the DAGs for consideration
- e.g. K2, HC etc
Bayesian Networks: Parameter Learning

- Now we know the structure and so the parent set of our Class node [Relevance in our case].
- We use the Maximum Likelihood Estimation (MLE) to give the probabilities, where Likelihood function is same [Cooper Herkovitz likelihood] as described in Structure Learning Section.
Ranking Function

- Bayesian Network will give us, $P(R=0)$, $P(R=1)$ and $P(R=2)$
- But we want a real score for the document
- Very straightforward yet effective function 'Expected Relevance' [Li et al., 2007]

$$
\text{Score}(X_i) = \sum_{i} i \times P(R = i)
$$
Issues with Regression Approach

- Here relevance is considered as either non-ordered categories or real values, which is not true in reality.
- This approach does not take any advantage of pairwise preference or total order information in learning.
Ordinal Regression

- Ordinal regression can be a better choice than Regression Approach.
- Because here categories are ordered while in classification the categories are non ordered.
Ordinal Regression contd.

- *P*ranking for Ranking [Crammer and Singer, 2001]
  - Here we project the $\vec{w}^T \cdot \vec{x}$ in the ordered categories $b_1 < b_2 < \cdots < b_n$.
  - In the training phase if the predicted category is not equal to the actual category then we modify vector $\vec{w}$ and thresholds.

Concerns:

- Still consider the absolute relevance
- does not take information of pairwise preference or total order information in learning.
Algorithms

- Regression
- Classification - McRank [Li et al., 2007], BayesNetRank [Gupta, 2011]
- Ordinal Regression - PRanking [Crammer and Singer, 2001], Large margin [Shashua and Levin, 2002]
Pairwise Approach

- Create Correct and Incorrect Ranked Pairs for each Query
- Use these pairs for Learning

Q1
Q2
Qn

- Create Correct and Incorrect Ranked Pairs for each Query
- Use these pairs for Learning

Q1
Q2
Qn
Pairwise Learning-to-Rank (LetoR) Model
Pairwise Approach

- Take the pairs of documents
- Input: $X_i, X_j \in R_t$ where $X_i > X_j$
- Output: $Y \in +1,-1$, where ' +1' denotes that $X_i > X_j$ is correct order and '+1' otherwise.
- Ex. if given data is like $(X_1, 2), (X_2, 1), (X_3, 0)$ where 0, 1 and 2 are the relevance labels
- New training data will be, $(X_1, X_2, +1), (X_2, X_1, -1), (X_1, X_3, +1), (X_3, X_1, -1), (X_2, X_3, +1)$ and $(X_3, X_2, -1)$. 
Pairwise Approach - RankSVM [Herbrich et al., 2000]

- The SVM model is as shown below,

\[
\text{minimize} \quad \frac{1}{2} ||w||^2 + C \sum_i \zeta_i \\
\text{subject to} \quad z_i \langle w, x_i^1 - x_i^2 \rangle \geq 1 - \zeta_i, \zeta_i \geq 1, i = 1, 2, \ldots, l
\]

Where,

\[
z_i = \begin{cases} 
+1 & \text{if } y_i^1 < y_i^2 \\
-1 & \text{if } y_i^2 < y_i^1 
\end{cases}
\]
Pairwise Approach contd.

- There are some concerns with Ranking SVM that
- It is biased towards queries with many relevant documents than those with fewer relevant documents. [Cao et al., 2006]
- It gives same weight to all the pairs.
- Also pairwise approach does not handle information of total order.
Algorithms

- Support Vector Machines - RankSVM [Herbrich et al., 2000], IR-SVM [Cao et al., 2006]
- Neural Networks - RankNet [Burges et al., 2005]
MICROSOFT VS. GOOGLE LAUNCHES

BY GUHMSHOO

TWITTER.COM/GUHMSHOO

WWW.BITSTRIPS.COM
horror movie
Listwise Approach

- Use whole ranked list for Learning

Q1
Q2
Qn

Q1
Q2
Qn

• Use whole ranked list for Learning
Listwise Learning-to-Rank (LetoR) Model

\[
\begin{align*}
q_1 & \quad \quad q_m \\
\vdots & \quad \quad \vdots \\
q_{m+1} & \\
\end{align*}
\]

\[
\begin{align*}
d_{1,1} & \quad \quad d_{m,1} \\
\vdots & \quad \quad \vdots \\
d_{1,n+1} & \quad \quad d_{m,n+1} \\
\end{align*}
\]

\[
f(q,d)
\]

\[
D = \{d\}
\]

\[
\begin{align*}
d_{m+1} & \quad f(q_{m+1},d_{m+1}) \\
\vdots & \quad \vdots \\
d_{m+1} & \quad f(q_{m+1},d_{m+1}) \\
\end{align*}
\]

\[
\text{Ranking Model}
\]
ListNet [Cao et al., 2007]

- Listwise Loss Function

\[ \sum_{i=1}^{N} L(y_i, z_i) \]

- \( L \) is cross entropy \((p \log p)\)
- \( z_i \) is the generated rank-list for query \( i \)
- \( y_i \) is the ground truth rank-list for query \( i \)

- Permutation Probability

\[ P_s(\pi) = \prod_{j=1}^{n} \frac{S_{\pi(j)}}{\sum_{k=j}^{n} S_{\pi(k)}} \]

Calculating for \( N \) objects is computationally intractable.
ListNet contd.

- Top k Permutation Probability

\[ P_s(j_1, j_2, \ldots, j_k) = \prod_{t=1}^{k} \frac{s_{j_t}}{\sum_{l=t}^{n} s_{j_l}} \]

- Time complexity of computation: from \( O(n!) \) to \( O\left(\frac{n!}{(n-k)!}\right) \)

- Ranking model \( s(.,.,.) \) is Neural Network

- Example, if \( s(A)=5, s(B)=4, s(C)=2 \) then \( P(ABC) \) is maximum and \( P(CBA) \) is minimum
Algorithms

▶ Neural Networks - ListNet [Cao et al., 2007]
Applications

- Search
- Collaborative Filtering [Freund et al., 2003]
- Key-Phrase Extraction [Jiang et al., 2009]
- Question Answering (why questions) [Verberne et al., 2011]
Collaborative Filtering

- **Problem formulation**
  - Input: users ratings on some items
  - Output: users ratings on other items
  - Assumption: users sharing same ratings on input items tend to agree on new items

- **Solution**
  - Classification
  - Ordinal Regression
  - Learning-to-Rank
Key Phrase Extraction

- Problem formulation
  - Input: document
  - Output: keyphrases of document
  - Two steps: phrase extraction and keyphrase identification

- Traditional approach
  - Classification: keyphrase vs non-keyphrase
Key Phrase Extraction contd.

- Ranking of Phrases
- Pairwise approach
why Question Answering

- Problem Formulations
  - Input: Question and Potential Answers
  - Output: Improved ranking of Answers (re-ranking)

- Traditional Approach
  - TF-IDF scoring
Question Answering contd.

- The data is highly imbalanced
- Pairwise approaches seem to work well
Statistics of public LTR datasets

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Queries</th>
<th>Doc.</th>
<th>Rel.</th>
<th>Feat.</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letor 3.0 - Gov</td>
<td>575</td>
<td>568 k</td>
<td>2</td>
<td>64</td>
<td>2008</td>
</tr>
<tr>
<td>Letor 3.0 - Ohsumed</td>
<td>106</td>
<td>16 k</td>
<td>3</td>
<td>45</td>
<td>2008</td>
</tr>
<tr>
<td>Letor 4.0</td>
<td>2,476</td>
<td>85 k</td>
<td>3</td>
<td>46</td>
<td>2009</td>
</tr>
<tr>
<td>Yandex</td>
<td>20,267</td>
<td>213 k</td>
<td>5</td>
<td>245</td>
<td>2009</td>
</tr>
<tr>
<td>Yahoo!</td>
<td>36,251</td>
<td>883 k</td>
<td>5</td>
<td>700</td>
<td>2010</td>
</tr>
<tr>
<td>Microsoft</td>
<td>31,531</td>
<td>3,771 k</td>
<td>5</td>
<td>136</td>
<td>2010</td>
</tr>
</tbody>
</table>
Research Trends in Learning to Rank

- Feature Extraction - *What can be useful?*
- Feature Selection - *All of them are useful?*
- Data Labeling - *Do I have to judge all of them?*
- Ranking Models - *How to Rank?*
- Dimensionality Reduction - *Can feature vector be compacted?*
- Suitability of Evaluation Metric - *Does one fit all?*
Xapian

- Xapian\(^1\) is an Open-source search engine project
- Letor API is developed under Google’s initiative to promote open-source, well known as, Google Summer of Codes (GSoC)
- Detailed description along with code of Letor API can be accessed through, wiki-page: http://trac.xapian.org/wiki/GSoC2011/LTR

Motivation

- To make an end-to-end i.e. natural language query to document ranked list type infrastructure for Learning to Rank

\(^1\)http://xapian.org/
API

Xapian::Letor ltr;
...
ltr.set_database(db);
ltr.set_query(query);
...
// prepare training file
ltr.prepare_training_file("topics.txt",
    "inex2010-article.qrels",
    100);
...

// learn the model with svm parameters
ltr.letor_learn_model(4,0);

// get the new ranked list with Letor!
map<Xapian::docid,double> letor_mset = ltr.letor_score(mset);
Summary

- We learnt the basic framework of Learning-to-Rank
- We explored the approaches to it
- We saw application of LTR to IR and NLP tasks
- Overview of Datasets and Xapian Letor API
Thank You! 😊

pgupta@dsic.upv.es
References I


References II

An efficient boosting algorithm for combining preferences.

Optimal polynomial retrieval functions based on the probability ranking principle.

Learning-to-Rank: Using Bayesian Networks.
Master's thesis, DA-IICT, India.

Large margin rank boundaries for ordinal regression.
References III

A ranking approach to keyphrase extraction.
In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR ’09, pages 756–757, New York, NY, USA. ACM.

Learning to rank.

Mcrank: Learning to rank using multiple classification and gradient boosting.
In *NIPS*.

Discriminative models for information retrieval.
In *SIGIR*, pages 64–71.

LETOR: A benchmark collection for research on learning to rank for information retrieval.
References IV

Ranking with large margin principle: Two approaches.
In *NIPS*, pages 937–944.

Learning to rank for why-question answering.