Efficiency/Effectiveness
Trade-offs in Learning to Rank

Tutorial @ ECML PKDD 2018
http://learningtorank.isti.cnr.it/

Claudio Lucchese
Ca’ Foscari University of Venice
Venice, Italy

Franco Maria Nardini
HPC Lab, ISTI-CNR
Pisa, Italy
The Ranking Problem

Ranking is at the core of several IR Tasks:

- Document Ranking in Web Search
- Ads Ranking in Web Advertising
- Query suggestion & completion
- Product Recommendation
- Song Recommendation
- ...
The Ranking Problem

Definition:
Given a query $q$ and a set of objects/documents $D$, to rank $D$ so as to maximize users’ satisfaction $Q$.

**Goal #1: Effectiveness**
- Maximize $Q$!
- but how to measure $Q$?

**Goal #2: Efficiency**
- Make sure the ranking process is feasible and not too expensive
  - In Bing ... “every 100msec improves revenue by 0.6%. Every millisecond counts.” [KDF+13]

Agenda

1. **Introduction to Learning to Rank (LtR)**
   - Background, algorithms, sources of cost in LtR, multi-stage ranking

2. **Dealing with the Efficiency/Effectiveness trade-off**
   - Feature Selection, Enhanced Learning, Approximate scoring, Fast Scoring

3. **Hands-on I**
   - Software, data and publicly available tools
   - Traversing Regression Forests, SoA tools and analysis

4. **Hands-on II**
   - Training models, Pruning strategies, Efficient scoring

At the end of the day you’ll be able to train a high quality ranking model, and to exploit SoA tools and techniques to *reduce its computational cost up to 18x*!
Document Representations and Ranking

**Document Representations**

A document is a multi-set of words

A document may have fields, it can be split into zones, it can be enriched with external text data (e.g., anchors)

Additional information may be useful, such as In-Links, Out-Links, PageRank, # clicks, social links, etc.

**Ranking Functions**

- Term-weighting [SJ72]
- Vector Space Model [SB88]
- BM25 [JWR00], BM25f [RZT04]
- Language Modeling [PC98]
- Linear Combination of features [MC07]

---

*Hundred signals in public LtR Datasets*

<table>
<thead>
<tr>
<th>Reference</th>
<th>Title and Details</th>
</tr>
</thead>
</table>
Ranking as a Supervised Learning Task

Machine Learning Algo
(NeuralNet, SVM, Decision-Tree)

Loss Function

Ranking Model
Ranking as a Supervised Learning Task

Machine Learning Algo (NeuralNet, SVM, Decision-Tree)

Loss Function

Ranking Model

Training Instance

Run-Time Instance

Top-k Results

Scored Documents

sort

Top-k Results

Lucchese C., Nardini F.M. Efficiency/Effectiveness Trade-offs in Learning to Rank
Query/Document Representation

**Useful signals**

- Link Analysis [H+00]
- Term proximity [RS03]
- Query classification [BSD10]
- Query intent mining [JLN16, LOP+13]
- Finding entities documents [MW08] and in queries [BOM15]
- Document recency [DZK+10]
- Distributed representations of words and their compositionality [MSC+13]
- Convolutional neural networks [SHG+14]
- ....

Relevance Labels Generation

**Explicit Feedback**

- Thousands of Search Quality Raters
- Absolute vs. Relative Judgments [CBCD08]

**Implicit Feedback**

- clicks/query chains [JGP+05, Joa02, RJ05]
- Unbiased learning-to-rank [JSS17]

**Minimize annotation cost**

- Active Learning [LCZ+10]
- Deep versus Shallow labelling [YR09]
Evaluation Measures for Ranking

<table>
<thead>
<tr>
<th>Rank</th>
<th>Top 10 Retrieved Documents</th>
<th>Binary Relevance Labels</th>
<th>Graded Relevance Labels</th>
<th>Account for labels:</th>
<th>Account for labels and ranks:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$d_3$</td>
<td>$y_3$ ✓</td>
<td>$y_3$ ★★★★</td>
<td>$Q_{@10} = 4 + 1 + 3$</td>
<td>$Q_{@10} = \frac{4}{1} + \frac{1}{3} + \frac{3}{7}$</td>
</tr>
<tr>
<td>2</td>
<td>$d_4$</td>
<td>$y_4$ ✗</td>
<td>$y_4$ ✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$d_7$</td>
<td>$y_7$ ✓</td>
<td>$y_7$ ★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$d_9$</td>
<td>$y_9$ ✗</td>
<td>$y_9$ ✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$d_6$</td>
<td>$y_6$ ✗</td>
<td>$y_6$ ✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$d_8$</td>
<td>$y_8$ ✗</td>
<td>$y_8$ ✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$d_2$</td>
<td>$y_2$ ✓</td>
<td>$y_2$ ★★★★</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$d_5$</td>
<td>$y_5$ ✗</td>
<td>$y_5$ ✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>$d_1$</td>
<td>$y_1$ ✗</td>
<td>$y_1$ ✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$d_{10}$</td>
<td>$y_{10}$ ✗</td>
<td>$y_{10}$ ✗</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Precision @10

$$P_{@10} = \frac{3}{10}$$
Evaluation Measures for Ranking

Many are in the form:

\[ Q@k = \sum_{r=1}^{k} \text{Gain}(d^r) \cdot \text{Discount}(r) \]

- (N)DCG [JK00]:
  \[ \text{Gain}(d) = 2^y - 1 \quad \text{Discount}(r) = \frac{1}{\log(r + 1)} \]
- RBP [MZ08]:
  \[ \text{Gain}(d) = \prod_{i=1}^{y-1} \quad \text{Discount}(r) = (1 - p)p^{r-1} \]
- ERR [CMZG09]:
  \[ \text{Gain}(d) = R_i \prod_{j=1}^{y-1} (1 - R_j) \text{ with } R_i = \frac{2^y - 1}{2^y_{\text{max}}} \quad \text{Discount}(r) = 1/r \]

Do they match User satisfaction?

- ERR correlates better with user satisfaction (clicks and editorials) [CMZG09]
- Results Interleaving to compare two rankings [CJRY12]
  - “major revisions of the web search rankers [Bing] ... The differences between these rankers involve changes of over half a percentage point, in absolute terms, of NDCG”

---


Is it an easy or difficult task?

*Gradient descent* cannot be applied directly

Rank-based measures (NDCG, ERR, MAP, ...) depend on *documents sorted order*

- gradient is either 0 (sorted order did not change)
- or undefined (discontinuity)

**Solution: we need a proxy Loss function**

- it should be *differentiable*
- and with a *similar behavior of the original cost function*
Point-Wise Algorithms

Each document is considered independently from the others

• No information about other candidates for the same query is used at training time

A different cost-function is optimized

• Several approaches: Regression, Multi-Class Classification, Ordinal regression, ... [Liu11]

Among Regression-Based:

Gradient Boosting Regression Trees [Fri01]

• Sum of Squared Errors (SSE) is minimized

Gradient Boosting Regression Trees

**Iterative algorithm:**

\[ F(d) = \sum_{i} f_i(d) \]

Each \( f_i \) is regarded as a step in the best optimization direction, i.e., a **steepest descent step**:

\[ f_i(d) = -\rho_i g_i(d) \quad - g_i(d) = - \left[ \frac{\partial L(y, f(d))}{\partial f(d)} \right]_{f=\sum_{j<i} f_j} \]

Given \( L = \text{SSE}/2 \):

\[ \frac{\partial \left[ \frac{1}{2} \text{SSE}(y, f(d)) \right]}{\partial f(d)} = - \frac{\partial \left[ \frac{1}{2} \sum (y - f(d))^2 \right]}{\partial f(d)} = y - f(d) \]

**Gradient** \( g_i \) is approximated by a Regression Tree \( t_i \)
Pair-wise Algorithms: RankNet \[\text{[BSR+05]}\]

**Documents are considered in pairs**

Estimated probability that \(d_i\) is better than \(d_j\) is:

\[
P_{ij} = \frac{e^{o_{ij}}}{1 + e^{o_{ij}}}
\]

Let \(Q_{ij}\) be the true probability, the **Cross Entropy Loss** is:

\[
C_{ij} = -Q_{ij} \log P_{ij} - (1 - Q_{ij}) \log(1 - P_{ij})
\]

We consider **only pairs where \(d_i\) is better than \(d_j\), i.e., \(y_i > y_j\):**

- If \(o_{ij} \to +\infty\) (i.e., correctly ordered) \(C_{ij} \to 0\)
- If \(o_{ij} \to -\infty\) (i.e., mis-ordered) \(C_{ij} \to +\infty\)

This is **differentiable**: used to train a **Neural Network with back-propagation**.

**Other approaches:** Ranking-SVM [Joa02], RankBoost [FISS03], ...

---


Pair-wise Algorithms

*RankNet* performs better than other pairwise algorithms

RankNet cost is not nicely correlated with NDCG quality

Figure 1. Ranking accuracies in terms of NDCG@n on TREC

Figure 4. Pairwise loss v.s. NDCG@5 in RankNet

List-wise Algorithms: LambdaMart [Bur10]

Recall: **GBRT requires a gradient** $g_i$ for every $d_i$

**First:** estimate the gradient comparing to $d_j$, with $y_i > y_j$:

$$
\lambda_{ij} = \frac{1}{1 + e^{\theta_{ij}}} \quad |\Delta NDCG| = -\lambda_{ji}
$$

\begin{align*}
\text{If } o_{ij} \to +\infty \quad (i.e., \text{correctly ordered}) & \quad \lambda_{ij} \to 0 \\
\text{If } o_{ij} \to -\infty \quad (i.e., \text{mis-ordered}) & \quad \lambda_{ij} \to |\Delta NDCG|
\end{align*}

**Then:** estimate the gradient comparing to every other $d_j$ for $q$

$$
g_i = \lambda_i = \sum_{y_i > y_j} \lambda_{ij} - \sum_{y_i < y_j} \lambda_{ij}
$$

List-wise Algorithms: some results

- NDCG@10 on public LtR Datasets

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MSN10K</th>
<th>Y!S1</th>
<th>Y!S2</th>
<th>Istella-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>RankingSVM</td>
<td>0.4012</td>
<td>0.7238</td>
<td>0.7306</td>
<td>N/A</td>
</tr>
<tr>
<td>GBRT</td>
<td>0.4602</td>
<td><strong>0.7555</strong></td>
<td><strong>0.7620</strong></td>
<td>0.7313</td>
</tr>
<tr>
<td>LambdaMART</td>
<td><strong>0.4618</strong></td>
<td>0.7529</td>
<td>0.7531</td>
<td><strong>0.7537</strong></td>
</tr>
</tbody>
</table>

Other approaches: ListNet/ListMLE\[CQL+07\], Approximate Rank\[QLL10\], SVM AP\[YFRJ07\], RankGP\[YLKY07\], others ...

Learning to Rank Algorithms

- New approaches to optimize IR measures:
  - DirectRank [XLL+08], LambdaMart [Bur10], BLMart [GCL11], SSLambdaMART [SY11], CoList [GY14], LogisticRank [YHT+16], ...
  - See [Liu11][TBH15].

- Deep Learning to improve query-document matching:
  - Conv.DNN [SM15], DSSM [HHG+13], Dual-Embedding [MNCC16], Local and Distributed repr. [MDC17], Weak Supervision [DZS+17], Neural Click Model [BMdRS16], ...

- On-line learning:
  - Multi-armed bandits [RKJ08], Dueling bandits [YJ09], K-armed dueling bandits [YBKJ12], online learning [HSWdR13][HWdR13], ...

---

In this tutorial we focus on GBRTs

Ads Click Prediction: GBDT as a feature extractor, then LogReg [HPJ+14]

Ads Click Prediction: refine/boost NN output [LDG+17]

Product Ranking: 100 GBDTs with pairwise ranking [SCP16]

Document Ranking: GBDT named LogisticRank [YHT+16]

Ranking, forecasting & recommendations: Oblivious GBRT


In this tutorial we focus on GBRTs

- **Successful in several Data Challenges:**
  - Winner of the Yahoo! LtR Challenge: combination of 12 ranking models, 8 of which were Lambda-MART models, each having up to 3,000 trees [CC11]
  - According to the 2015 statistics, GBRTs were adopted by the majority of the winning solutions among the Kaggle competitions, even more than the popular deep networks, and all the top-10 teams qualified in the KDD Cup 2015 used GBRT-based algorithms [CG16]

- New interesting open-source implementations:
  - XGBoost, LightGBM by Microsoft, CatBoost by Yandex

- **Pluggable** within Apache Lucene/Solr

---

Single-Stage Ranking

Requires to apply the learnt *model* to *every matching document*, and to generate the required *features*.

Not feasible!

We have at least *3 efficiency vs. effectiveness trade-offs*. 

Lucchese C., Nardini F.M. Efficiency/Effectiveness Trade-offs in Learning to Rank
Single-Stage Ranking

① *Feature Computation Trade-off*

- Computationally *Expensive* & *highly discriminative* features vs. computationally *Cheap* & *slightly discriminative* features
Expensive features are computed only for the top-K candidate documents passing the first stage. How to choose $K$?

2) **Number of Matching Candidates Trade-off**:

- A Large set of candidates is **Expensive** and produces **high-quality** results vs. a Small set of candidates is **Cheap** and produces **low-quality** results
  - 1000 documents [DBC13] (Gov2, ClueWeb09-B collections)
  - 1500-2000 documents [MSO13] (ClueWeb09-B)
  - “hundreds of thousands” (over “hundreds of machines”) [YHT+16a]
Multi-Stage Ranking

- 3 stages [YHT+16]: Contextual features are considered in the 3rd stage
  - Contextual => about the current result set
  - Rank based on specific features, Mean, Variance, Standardized features (see also [LNO+15a]), topic model similarity
  - First two stages are executed at each serving node

- N stages [CGBC17]: Which model in each stage? Which features? How many documents?
  - About 200 configurations tested, best results with N=3 stages, 2500 and 700 docs between stages

- Predict the best k for STAGE 1 [CCL16], and the best processing pipeline [MCB+18]

- A proper methodology/algorithm for choosing the best configuration is still missing.

---


Multi-Stage Ranking

③ Model Complexity Trade-off:

• **Complex & Slow high-quality** vs. **Simple & Fast low-quality** models:
  - **Complex** as: Random Forest, GBRT, Initialized GBRT, Lambda-MART,
  - **Simple** as: Coordinate Ascent, Ridge Regression, SVM-Rank, RankBoost
  - **In-between** as: Oblivious Lambda-Mart, ListNet
Model Complexity Trade-off

- Comparison on varying training parameters [CLN+16]:
  - #trees, #leaves, learning rate, etc.
- Complex models achieve significantly higher quality
- Best model depends on time budget

- Today is about Model Complexity Trade-off!
...some recent advances...
Learning to Rank with Deep Neural Networks

- **Issues**: *Multiple Fields / Multi-instance Fields*
- **Proposed solution**:
  - **Instance-level DNN**:
    - Layers: 3-gram hashing, embedding, 1D convolution, pooling, dense
  - **Per-field tunable**
  - Multi-instance aggregation by averaging
  - Multi-field aggregation concatenation
- **Pair-wise training**, *non-public Bing data*

<table>
<thead>
<tr>
<th>Model</th>
<th>NDCG@1</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25F</td>
<td>0.4431</td>
<td>0.6020</td>
</tr>
<tr>
<td>LTR</td>
<td>0.4888</td>
<td>0.6341</td>
</tr>
<tr>
<td>NRM-F</td>
<td>0.4906*</td>
<td>0.6380*</td>
</tr>
</tbody>
</table>

Are ANN faster than Regression Forests?

- **Rationale:**
  - Trust accuracy of LambdaMART, use a “similar” ANN at run time
- **Methodology:**
  - *Ranking Distillation* [TW18]
  - **Train a ANN that approximates the output of LambdaMART**
    - … rather than the training labels!
    - Enrich the dataset with points around discontinuities, i.e., trees’ split points
    - Networks used: Fully connected 4 layers 2000x500x500x100 and 2 layers 500x100

<table>
<thead>
<tr>
<th>Method</th>
<th># Layers</th>
<th>MSN30k MAP</th>
<th>GOV2 MAP</th>
<th>Source</th>
<th>1000 Trees</th>
<th>20,000 Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8 Leaves</td>
<td>64 Leaves</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Generated C++ for Forest</td>
<td>8.2-10.3</td>
<td>709.0-772.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If-Then-Else</td>
<td>55.9-55.1</td>
<td>4462.0-4809.0</td>
</tr>
<tr>
<td>Regression Forest</td>
<td>-</td>
<td>0.6004</td>
<td>0.2995</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N_{approx}$</td>
<td>4</td>
<td>0.5950</td>
<td>0.2995</td>
<td>TensorFlow-CPU 4-layer</td>
<td>51.1-53.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TensorFlow-CPU 2-layer</td>
<td>5.83-7.04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.5955</td>
<td>0.3007</td>
<td>PyTorch GPU 4-layer</td>
<td>0.976-1.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PyTorch GPU 2-layer</td>
<td>0.323-0.335</td>
<td></td>
</tr>
</tbody>
</table>


Dealing with large and unbalanced datasets

GBRT/LambdaMART is typically applied to (trained on) a large set of documents out of which only a few are relevant.

Can we achieve faster and more effective training?

Selective Gradient Boosting
- uses a small percentage of non-relevant documents (e.g., 1%)
- chosen, at each iteration, among the top ranked
- Achieves >3% NDCG relative improvement!

Efficiency/Effectiveness trade-offs in:

• Feature Selection
• Enhanced Learning Algorithms
• Approximate scoring
• Fast Scoring
References


References


References


References


References


References


References


References


