Learning to Rank for Web Search: Some New Directions

Chris Burges

Web Learning Group
Microsoft Research

SIGIR Ranking Workshop, July 27, 2007
Joint Work With:

• Qiang Wu
• Robert Rounthwaite
• Robert Ragno
• Ping Li
• Quoc Viet Le
• Yisong Yue
• John Platt
• MSN Search Team
Goals of This Talk

• To stimulate ideas and discussion
• Five different approaches (3 new, 1 a year old, 1 older) will be covered
• Will not have time to go into all the details: but do please ask questions during the talk
Contents

• IR Metrics: Are We Learning the Wrong Thing?
• Brief Overview of Web Search
• Recent Work
  – RankNet
  – LambdaRank
• New Ideas for Ranking Algorithms
  – Simultaneous Perturbation Stochastic Approx.
  – Gradient Boosting
  – XRank
Are We Learning the Wrong Thing?
An Example: SVMs for Classification

\[
\text{Minimize: } \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \\
\text{subject to: } y_i(x_i \cdot w + b) \geq 1 - \xi_i \\
y_i \in \{\pm 1\}, \; \xi_i \geq 0
\]
Information Retrieval Costs / Gains

- Precision: $\frac{\text{number correct}}{\text{number accepted}}$
- Recall: $\frac{\text{number correct}}{\text{number positives}}$
- Average Precision: Compute precision for each positive, average over positions
- Mean Average Precision: Average AP over queries
- Mean Reciprocal Rank (TREC QA): $\frac{1}{N_Q} \sum_{i=1}^{N_Q} \frac{1}{r_i}$
- Winner Takes All: $\frac{1}{N_Q} \sum_{i=1}^{N_Q} \delta(l_{i1}, 1)$
- Pairwise error and derivatives (‘bpref’)
- Fraction pairwise correct = area under ROC curve (fraction true positives vs. fraction false positives)
We’d like a measure for several levels of relevance, that emphasizes the top ranked items: *normalized discounted cumulative gain* (Jarvelin & Kekalainen, SIGIR 2000):

\[
DCG(\text{truncation level}) = \sum_{i=1}^{T} \frac{2^r(i) - 1}{\log(1 + i)}
\]

\[r \in \{0, ..., 4\} \text{ human rating}\]

\[NDCG \equiv \frac{DCG(T)}{\max(DCG(T))}\]
Information Retrieval Costs, cont.

All of these costs have the following in common:

• They depend only on the labels and the sorted order of the documents
• Viewed as a function of the scores output by some model, the costs are everywhere either:
  • Flat (zero gradient)
    or
  • Discontinuous (gradient not defined)
One View of the Goal

Interesting but intractable problem $P$

Relax this; rewrite that.

$P'$ convex! Yaay!
Amenable to analysis.
Tractable (after some more approximations, perhaps).
But, $P' \neq P$.

XRank, SPSA

LambdaRank

RankNet, MART

Design tractable algorithms with $P$ in mind. Try hard to stay close to $P$.

$P''$ tractable, and goal:

$\|P'' - P\| < \|P' - P\|$

MUST BE BLAZINGLY FAST!
Web Search: Overview
Why is Web Ranking Interesting?

- **Economic**: Internet advertising revenues were about $4 billion in 1Q 2006 (IAB, PricewaterhouseCoopers).
- **Generality**: Key component of Information Retrieval, Collaborative Filtering, Spam Detection, Question Answering…
- **Scientific**: New practical problems are motivating us to consider new machine learning approaches.
Live Search – Some Statistics

- > 5 billion documents indexed
- 10s of millions of queries handled per day
- Thousands of machines
- Most queries served in less than 100 ms
Live Search Architecture

- Customer Query
- Front End Experience (FEX) Federator
- Web Search
- Ads
- Encarta
- Mid Level Aggregator
- Index Serving Node
Live Search Architecture, cont.
Some Recent and Not-So-Recent Approaches to Ranking
Recent Work

• “A Boosting Algorithm for Information Retrieval”, J. Xu and H. Li, SIGIR ’07 (“IR Boost”)
• “Learning to Rank Using Classification and Gradient Boosting”, P. Li, C. Burges and Q. Wu, MSR-TR-2007-74
• “Learning to Rank: From Pairwise to Listwise Approach”, Z. Cao, T. Qin, T-Y Liu, M-F Tsai, H. Li, ICML 07 (“ListRank”)
• “FRank: A Ranking Method with Fidelity Loss”, M-F. Tsai, T-Y Liu, T. Qin, H-H Chen, W-Y Ma, SIGIR 07
• “A Support Vector Method for Optimizing Average Precision”, Y. Yue, T. Finley, F. Radlinksi and T. Joachims, SIGIR 07
• “Learning to Rank with Nonsmooth Cost Functions”, C.J.C. Burges, R. Ragno and Q.V. Le, NIPS 06 (“LambdaRank”)

Less Recent Work


• “PRanking with Ranking”, K. Crammer and Y. Singer, *KDD 02* (“PRank”)

• “Online Ranking/Collaborative Filtering using the Perceptron Algorithm”, E.F. Harrington, *ICML 03*

• “Optimizing Search Engines Using Clickthrough Data”, T. Joachims, *KDD 02*

• “Support Vector Learning for Ordinal Regression”, R. Herbrich, T. Graepel and K. Obermayer, *ICANN 99*

• “Using the Future to Sort Out the Present: RankProp and Multitask Learning for Medical Risk Evaluation”, R. Caruana, S. Baluja and T. Mitchell, *NIPS 96*
Ranking as Machine Learning

• Given a set of text queries $Q_i, i = 1, ..., m$
• Each query has a large set of returned documents $D_{ij}, j = 1, ..., n$
• Use query, document, URL, anchor text, and more, to derive set of (several hundred) features
• For each query, rank returned documents in order of relevance
• Most systems map a feature vector to a single score, which is then sorted to obtain the ranking
RankNet

Ranking with Neural Nets

• Don’t need to learn ordinal regression (mapping points to actual rank values); just need to map features to reals.
• **Train system on pairs** (where first point is to be ranked higher or equal to second).
• However must **evaluate on single points**.
• Use **cross entropy cost** → probabilistic model.
• Use **gradient descent**. Would work for any differentiable function: we chose neural net.
5 human judged levels of relevance (“bad” … “perfect”).

A net with (number of features) inputs and one output

Sort documents by the score that their feature vectors (which are computed from query + doc + other data) get

Compute NDCG on a set-aside validation set, keep the net that gives the best validation NDCG
A Probabilistic Ranking Cost Fn.

• Ranking labels tend to be noisier than classification labels

Specify $P(A \triangleright B)$ for each train pair $\{A, B\}$

• The pairs of training ranks need not be complete, or consistent, but the test results are

Map to reals: $f(x_1) > f(x_2) \iff x_1 \triangleright x_2$
Probabilistic Ranking Cost Fn.

Modeled posteriors: \( P_{ij} \equiv P(x_i \succ x_j) \)

Target posteriors: \( \bar{P}_{ij} \)

Define \( o_{ij} \equiv f(x_i) - f(x_j) \)

Cross entropy cost:
\[
C_{ij} \equiv C(o_{ij}) = -\bar{P}_{ij} \log P_{ij} - (1 - \bar{P}_{ij}) \log(1 - P_{ij})
\]

Model output probabilities using logistic:
\[
P_{ij} = \frac{\exp(o_{ij})}{1 + \exp(o_{ij})}
\]

\[\Rightarrow C_{ij} = -\bar{P}_{ij} o_{ij} + \log(1 + \exp o_{ij})\]
$C(o_1-o_2)$

- $P=0.0$
- $P=0.5$
- $P=1.0$
There must exist $\bar{\alpha}_{ij}$ such that:

$$
\bar{P}_{ij} = \frac{\exp(\bar{\alpha}_{ij})}{1 + \exp(\bar{\alpha}_{ij})}
$$

$$
\bar{P}_{ik} = \frac{\bar{P}_{ij} \bar{P}_{jk}}{1 + 2\bar{P}_{ij} \bar{P}_{jk} - \bar{P}_{ij} - \bar{P}_{jk}}
$$
Complete uncertainty propagates, similarly for complete certainty:

\[ P(A \triangleright B) = 0.5, \quad P(B \triangleright C) = 0.5 \Rightarrow P(A \triangleright C) = 0.5 \]

Confidences build: for \( 0 < P < 0.5 \), \( \overline{P_{ik}} < P \);

for \( 0.5 < P < 1.0 \), \( \overline{P_{ik}} > P \)
More Formally:

How free are we to choose the $\bar{P}'s$?

**Proposition:** Specifying any set of adjacency posteriors is necessary and sufficient to uniquely determine a target posterior for every pair of samples.

**Proposition:** Let $n > 0$. Then if $\bar{P} > \frac{1}{2}$, then $\bar{P}_{i,i+n} \geq \bar{P}$ with equality when $n = 1$, and $\bar{P}_{i,i+n}$ increases strictly monotonically with $n$. If $\bar{P} < \frac{1}{2}$, then $\bar{P}_{i,i+n}$ decreases strictly monotonically with $n$. If $\bar{P} = \frac{1}{2}$, then $\bar{P}_{i,i+n} = \frac{1}{2} \forall n$. 
The Bradley-Terry Model

Bradley and Terry, *Biometrika* 1952 consider models:

\[ P(A_i | A_i \lor A_j) \text{ given, and model } P(A_i | A_i \lor A_j) = \frac{P_i}{P_i + P_j} \]

(e.g. \( P_i = N \exp(o_i) \)).
RankNet: Data

Training on pairs prohibitive? No:

- Docs are only compared to other docs for the same query, and many docs have the same rank.

<table>
<thead>
<tr>
<th></th>
<th># Queries</th>
<th># Documents</th>
<th># Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>11,336</td>
<td>384,314</td>
<td>3,464,289</td>
</tr>
<tr>
<td>Valid</td>
<td>2,384</td>
<td>2,726,714</td>
<td>-</td>
</tr>
<tr>
<td>Test</td>
<td>2,384</td>
<td>2,715,175</td>
<td>-</td>
</tr>
</tbody>
</table>

- Features from 4 ‘streams’: anchor text, URL, document title, and document body.
- 569 features: most are joint (query/doc dependent).
RankNet Conclusions

• RankNet is simple to train…
• …fast in test phase…
• and gives good results.
• For pair-based probability costs (e.g. click rates!) it’s very well suited to the problem.
• However the cost function used is not NDCG: the latter is optimized only indirectly, using a validation set.

Can we do better?
LambdaRank

Joint work with R. Ragno, Q.V. Le, NIPS 2006
RankNet Cost $\sim$ Pairwise Cost

$$C(o_1-o_2)$$

The graph shows the pairwise cost function $C(o_1-o_2)$ over the range of $o_1-o_2$ values.
Pairwise Cost Revisited

Pairwise cost fine if no errors, but:

13 errors 11 errors
LambdaRank

Instead of using a smooth approximation to the cost, and taking derivatives, write down the derivatives directly.

\[ \frac{\partial C}{\partial s_1} \gg \frac{\partial C}{\partial s_2} \]

Then use these derivatives to train a model using gradient descent, as usual.
A Simple Example

\[ D_1, D_2 : \ l_1 = 1, \ l_2 = 0 \]

Imagine some cost \( C \):

\[
\frac{\partial C}{\partial s_1} = -\lambda_1(s_1, l_1, s_2, l_2)
\]

\[
\frac{\partial C}{\partial s_2} = -\lambda_2(s_1, l_1, s_2, l_2)
\]
Letting \( x = s_1 - s_2 \):

\[
x < 0: \quad \lambda_1 = 1 = -\lambda_2 \\
0 \leq x \leq \delta: \quad \lambda_1 = \delta - x = -\lambda_2 \\
x > \delta: \quad \lambda_1 = \lambda_2 = 0
\]

Then a cost function \( C \) exists:

\[
x < 0: \quad C(s_1, l_1, s_2, l_2) = s_2 - s_1 + \frac{1}{2} \delta^2 \\
0 \leq x \leq \delta: \quad C(s_1, l_1, s_2, l_2) = \frac{1}{2} (s_2 - s_1)^2 - \delta (s_1 - s_2) + \frac{1}{2} \delta^2 \\
x > \delta: \quad C(s_1, l_1, s_2, l_2) = 0
\]

…furthermore it’s convex
LambdaRank

• Choose the $\lambda'$s to model the desired cost. *(Need not use pairs!)*
• Very general. Handles multivariate, non-smooth costs.
• But, how to choose the $\lambda'$s?
• When will there exist a cost function $C$ for your choice of $\lambda'$s?
• When will that $C$ be convex?
Some Multilinear Algebra Basics

• An ‘$n$-form’ on a manifold $M$ is a totally antisymmetric tensor that lives in the dual of the tangent space of $M$.

• You can apply the differential operator $d$ to an $n$-form to get an $(n+1)$-form.

• A closed form $f$ is one for which $df=0$.

• An exact form $g$ is one for which $g=dh$, for some form $h$.

• $dd=0$ (every exact form is closed).
Poincare’s Lemma

If $S \subset \mathbb{R}^n$ is an open set that is star-shaped with respect to the origin, then any closed form defined on $S$ is exact.

Hence on such a set, a form is exact iff it is closed.

Define the 1-form $\lambda = \sum_i \lambda_i dx_i$

Then $\lambda = dC$ for some $C$ iff $d\lambda = 0$.

Using classical notation: $\frac{\partial \lambda_i}{\partial x_j} = \frac{\partial \lambda_j}{\partial x_i}$ $\forall i, j$ : Jacobian symmetric!
The Jacobian

- Square matrix, of side $n\text{Docs}$
- Family of Jacobians, one for each label set
- Symmetric $\rightarrow$ cost function exists
- Positive semidefinite $\rightarrow$ cost function is convex
- (…like a kernel, but more general: depends on all points!)
A Physical Analogy

• Think of ranked documents as point masses, \( \lambda' \)'s as forces
• If \( \lambda = dC \), the forces are conservative – they derive from a potential
• E.g. choosing the \( \lambda' \)'s to be linear in the scores is equivalent to a spring model
LambdaRank Speedup for RankNet

• Most neural net training is stochastic (update weights after every pattern)
• Here we can compute and increment the gradients for each document (mini batch)
• Batch them, apply fprop and backprop once per doc, per query; factorize the gradient.
Speedup Results

![Graph showing speedup results for RankNet Training and LambdaRank Speedup over log number of documents and log seconds per epoch]
The Lambda Function

NDCG gain in swapping members of a pair of docs, multiplied by RankNet cost as a smoother:

$$\lambda = N \left( \frac{1}{1 + e^{s_i - s_j}} \right) \left( 2^{l(i)} - 2^{l(j)} \right) \left( \log \left( \frac{1}{1 + i} \right) - \log \left( \frac{1}{1 + j} \right) \right)$$
Accuracy Results

10K train, 5K validation, 10K test queries
LambdaRank Conclusions

• LambdaRank is simple and general (it can handle any cost function) but… how to choose $\lambda$?
• It leverages existing neural net methods (only the training changes)
• It gives a very significant speedup for RankNet
• It gives better accuracy than RankNet
• Still does not directly optimize NDCG!
• See “Ranking as Function Approximation” (Springer 2006) and “Learning to Rank with Nonsmooth Costs” (NIPS 2006)

Can we do better?
SPSA

Joint work with Yisong Yue and John Platt
http://research.microsoft.com/~cburges/pubs.htm
So you want to learn NDCG directly. *Why not just use gradient descent?*
The Finite Difference Method

Objective function $L$, parameters $w \in \mathbb{R}^d$, $c \ll 1$ and $[e_i]_j \equiv \delta_{ij}$

\[ \hat{g}(w) = \begin{bmatrix} \frac{L(w + ce_1) - L(w - ce_1)}{2c} \\ \frac{L(w + ce_2) - L(w - ce_2)}{2c} \\ \vdots \\ \frac{L(w + ce_{p-1}) - L(w - ce_{p-1})}{2c} \\ \frac{L(w + ce_p) - L(w - ce_p)}{2c} \end{bmatrix} \]

... requires $2d$ function evaluations.
Simultaneous Perturbation Stochastic Approximation (SPSA)

A general method for performing gradient descent when the gradient is too slow (or is impossible) to compute.

Simultaneous Perturbation Stochastic Approxn.

$$\frac{L(w + c\Delta) - L(w - c\Delta)}{2c\Delta_1}$$

$$\frac{L(w + c\Delta) - L(w - c\Delta)}{2c\Delta_2}$$

... only two function evaluations.

$$\Delta \in \{-1, 1\}^d$$
Spall’s Lemma (paraphrased): if the cost function’s third derivatives are bounded everywhere (independent of iteration number \( k \)), and if the \( \Delta \)'s are iid component-wise and also satisfy some simple moment conditions (which \textit{are} satisfied by the symmetric Bernoulli distribution), then:

\[
E[\hat{g}_k - g_k | \theta_k] = O(c_k^2)
\]
A Schematic View

From Spall 1998, with permission
Smoothness Tests: 100 Queries
Smoothness: 1000 Queries
Smoothness: 10,000 Queries
SPSA Results: Cross Entropy
SPSA versus FDSA
MART

Classification / Regression?

• Challenge our assumptions! Powerful, standard methods are available for classification and regression (in particular, boosted trees).
• So: let’s treat this as a classification, ordinal classification or regression problem.
• Why classification? Perfect (and some imperfect) classifications imply max DCG.

Joint work with Ping Li and Q. Wu, “Learning to Rank Using Classification and Gradient Boosting”,
http://research.microsoft.com/~cburges/pubs.htm
Three Basic Models

- Multiclass Classification: \( P(y_i = k|x_i) \)

- Ordinal Classification: \( P(y_i \leq k|x_i) \)

\[
P(y_i = k|x_i) = P(y_i \leq k|x_i) - P(y_i \leq k - 1|x_i)
\]

- Regression: model targets \( 2^{y_i} - 1 \) using least squares (cf. Cossock and Zhang, Colt ‘06)
Lemma: Given $n$ urls, originally ordered as $\{1, 2, \ldots, n\}$. Suppose a classifier assigns a relevance level $\hat{y}_i \in \{1, \ldots, k\}$ to the $i^{th}$ url, for all $n$ urls. Let a permutation mapping $\pi$ rank the urls according to $\hat{y}_i$. The corresponding DCG error is bounded by the square root of the classification error:

$$\begin{align*}
\text{DCG}_m - \text{DCG}_\pi &\leq (2^{km} - 2^1)\sqrt{2}\left(\sum_{i=1}^{n} c_i^2\right)^{\frac{1}{2}}\left(\sum_{i=1}^{n} 1_{y_i \neq \hat{y}_i}\right)^{\frac{1}{2}}
\end{align*}$$
We need a ranking score. Use the expected relevance:

$$\text{score} = \sum_{k=1}^{K} kP(y_i = k|x_i)$$

Could use any monotonic function of $k$: simplest $(k)$ gave best results.

Use cross entropy loss:

$$\Psi \equiv \text{loss} = \sum_{i=1}^{N} \sum_{k=1}^{K} -\log P(y_i = k|x_i)1_{k=k_i}$$
**Gradient Boosting: MART**

\[
(\beta_m, a_m) = \arg\min_{(\beta, a)} \sum_{i=1}^{N} \Psi(y_i, F_{m-1}(x_i) + \beta h(x_i, a))
\]

Estimate gradient:

\[
a_m = \arg\min_{(\beta, a_m)} \sum_{i=1}^{N} \left( -g_m + \beta h(x_i, a_m) \right)^2
\]

Perform line search:

\[
\rho_m = \arg\min_{(\rho, a)} \sum_{i=1}^{N} \Psi(y_i, F_{m-1}(x_i) + \rho h(x_i, a))
\]

Update:

\[
F_m(x) = F_{m-1}(x) + \rho_m h(x, a_m)
\]

MART for Ranking: Notes

- $K$ trees per boosting iteration
- Each tree fits gradient estimate using least squares
- Line search is performed for each leaf, using a Newton-Raphson step
- Tree outputs converted to probs using logistic function
- MART builds a multiclass classifier from regression trees (that fit residuals)
MART: Results

![Graph showing NDCG (%) vs Iterations for Classifications and Regressions](image)

- **Classifications**
- **Regressions**

NDCG (%)

Iterations
MART: Results, cont.
MART: Conclusions

• MART gives great results, but it’s not optimizing the cost directly (and it’s a little slow).

Building on boosting sounds like a good direction: can we build weak learners that more directly solve the problem at hand?
XRank

Joint work with Robert Rounthwaite and Qiang Wu
XRank

• An attempt to directly optimize the (non-differentiable) cost function we care about
• Build a planar, directed acyclic graph, with a single root node.
• Like a decision tree, but it’s a DAG, and has a different interpretation.
Martingale Boosting for Classification

P. Long and R. Servedio, COLT 2005
XRank

• Instead of classifying by position, encode the rank of the sample by the position of the leaf node it winds up at.

• Long and Servedio give exponential bound on the learning error rate for classification. We can extend this to a bound for the training error for pairwise ranking, for an arbitrary number of levels of relevance.
Martingale Bound for Ranking

Training samples $x_i \in \mathbb{R}^n$, $x_i \sim D$; set of training pairs
$\{x_i, x_j\}$ such that $x_i \succ x_j$

Assume: given $D$, hypotheses $h_t: \mathbb{R}^n \to \{0, 1\}$ such that $\forall t, k, l$

$$P(h_{tk}(x_i) = 1) - P(h_{tl}(x_j) = 1) \geq \gamma_t$$

Theorem: then, for $T$ levels, the final output hypotheses satisfy:

$$P\left(\text{rank}(x_i) < \text{rank}(x_j)\right) \leq \exp\left(-\frac{(\sum_{t=1}^{T} \gamma_t)^2}{4T}\right)$$
A Model: (Mini)Max NDCG

Start with some ordering in parent node.

Choose split to maximize the gain in NDCG, given that the ordering within each child node is unchanged.
Loop through thresholds; track which queries affected; compute their NDCG. (Relevance: Green/Orange/Red)

- Monotonically increases NDCG!
- … but, does not learn to completion: the Martingale bound fails: Query Fragmentation
- Too local!
• All forces equal and opposite (they sum to zero)
• Force between two samples is proportional to NDCG gain for swapping those two samples
• The sum of absolute values of the forces is a useful objective function
XRank: Three Basic Operations

SPLIT

MERGE

SWAP
Learning with XRank

• Compute gain from a split
• Compare gain from merging, then splitting, to gain from just splitting (left or right)
• At each step, choose best split, or best merge
• After every change, reorder the ‘frontier’
Why Use a Directed Acyclic Graph?

• Can grow linearly with depth instead of exponentially (less overfitting, less query fragmentation)

• Allows samples to migrate back to where they should be, if an error is made

• Can rebalance, and rearrange nodes after learning a level, to further reduce cost

• DAG advantages: they boost well, no learning rate, easy to interpret (e.g. to find most important features), fast in test phase
Physical Models Can be Tricky

Pitfall I: Oscillations

1 meter
Physical Models Can be Tricky II

Pitfall 2: Consistency

Three samples \( \{s_1, s_2, s_3\} \) with forces \( \{f_1, f_2, f_3\} \) with \( \sum_{i=1}^{3} f_i = 0 \)

Can choose splits so that net gain does not vanish (e.g. \( G = f_1 + f_3 \) )

Going to higher dimensions (e.g. \( n \)-dimensional simplex) does not help: we’d like 1-d forces!
Physical Models Can Be Tricky III

Pitfall 3: Bunching
Some Simple Theorems

Simplified XRank: only splits and swaps; also negative gain splits are allowed (i.e. if no positive gain splits exist, take the best non-positive split for the heaviest node): then we have:

Theorem: The training procedure cannot result in oscillations.

Theorem: If every pair of samples differ in at least one (binned) feature, then given sufficient iterations, XRank will learn the training data perfectly (despite not necessarily being monotonic in NDCG).

Theorem: The computational complexity of computing the best split for a node is $O(KFTN)$.
Why Not $O(KFTN^2)$?

Exercise for the reader.
More Flexible Models

Number of Queries

NDCG@10

Number of Splits

- Merge+Swap, NDCG
- No Merge, No Swap, NDCG
- Merge+Swap, # Splits
- No Merge, No Swap, # Splits
Computational Cost: Test

![Graph showing mean number of nodes visited against number of queries. The graph compares Merge+Swap and No Merge, No Swap scenarios.]
Conclusions

• Learning to Rank, with cost measures typically used in information retrieval, presents many opportunities for developing useful new machine learning solutions.
• For given features, eventually methods will likely converge to having similar performance.
• The ‘speed in test phase’ constraint is not typically the focus of current research, but it also motivates interesting new directions for work.
• We’re hiring!
• Thank You.