LambdaMART Demystified

Tomáš Tunys

Czech Technical University

tunystom@fel.cvut.cz

January 23, 2015

Overview

Learning to Rank: Problem Statement (Informal)

The goal of learning to rank models (so-called rankers) in Information Retrieval is to sort a collection of documents according to the degree of their relevance to a given query.

This statement begs the following questions:

- How is a **document** represented?
- How is a **query** represented?
- How is the relationship between the two represented?
- What does *degree of* **relevance** mean?
- What is the **measure** of quality of ranking?

Learning to Rank: Problem Statement (Informal) Cont'd

The goal of learning to rank models (so-called rankers) in Information Retrieval is to sort a collection of documents according to the degree of their relevance to a given query.

From the set of all possible answers we will use:

- Queries and documents are jointly represented as vectors in $\mathbb{R}^n$
- Relationship between query and document – bunch of additional (important) features.
- Relevance – binary (relevant/non-relevant), multi-labeled $(0, 1, 2, \ldots)$
- Ranking quality measures: NDCG, MAP, ERR, ... just name it, **but?!**

Learning to Rank as Risk Minimization Problem

Given an annotated dataset set $S = \{(D_q, y_q)\}_{q=1}^{Q}$, where

- Q is the total number of queries in your set.
- $D_q = \{d_1^q, \ldots, d_{n(q)}^q\}$ is set of documents for query q.
- $y_q = \{y_1^q, \ldots, y_{n(q)}^q\}$ is a corresponding set of relevance judgements.

The goal is to find a ranking function $f : \mathbb{R}^n \to \mathbb{R}$, which minimizes

$$R_{emp}[f] = \frac{1}{Q} \sum_{q=1}^{Q} \Delta(\pi(f, D_q), y_q)$$

where $\pi(f, D_q)$ is the ranking of documents for query q and Δ measures the discrepancy between $\pi(f, D_q)$ and y_q.

Learning to Rank as Risk Minimization Problem Cont'd

The ranking function f produces $\pi(f, D_q)$ such that

- $f(d_i^q) > f(d_j^q) \iff \pi(f, d_i^q) < \pi(f, d_j^q)$

Ultimate goal (overfitting rings a bell?!):

- *ranking scores* produced by f mimics the order imparted by relevance judgements y^q.
- $y_i^q > y_j^q \iff f(d_i^q) > f(d_j^q)$

Reminder: Learning to Rank models are categorized according to the loss functions (Δ) they are trained to minimize.

Learning to Rank Approaches

Pointwise approach

- Δ is defined on the basis of single documents
- reduces the problem to simple classification/regression
- Example: $\Delta(\pi(f, D_q), y_q) = \frac{1}{n(q)} \sum_{i=1}^{n(q)} (f(d_i^q) - y_i^q)^2$

Pairwise approach

- Δ is defined on the basis of pairs of documents with different relevance judgements.
- reduces the problem to classification
- Example: $\Delta(\pi(f, D_q), y_q) = \sum_{(i,j):y_i^q < y_j^q} \log(1 + \exp(f(d_i^q) - f(d_j^q)))$
- Ranking SVM, RankNET, RankBoost, ...

Learning to Rank Approaches Cont'd

Listwise approach

- Δ is defined on the basis of the whole document lists
- Example: see [Xia, F. et al, 2008]
- ListMLE, $\mathrm{SVM}^{\mathrm{map}}$, LambdaRank, LambdaMART, ...

Moral from the previous lecture:

pointwise $<$ *pairwise* $<$ *listwise*

Optimizing Information Retrieval Quality Measures

Most learning to rank models are not trained to optimize the IR measures (directly), not even the listwise methods. But that is what we care about! Why is that?

- IR measures are wild and not well-behaved beasts (non-smooth, non-differentiable, ...)
- Indirect optimization is also hard: designing a good surrogate measure is hard due to sorting.

Regardless of their accuracy, pointwise and pairwise approaches still can work pretty well. The loss functions they optimize has been shown to upper-bound (1 - NDCG) *loss*, see [Chen, W. et al. 2009].

- The inferior performance of these models is actually due to spending too much capacity on doing more than is required.

RankMART Model Definition

In order to understand how LambdaMART (current state of the art learning to rank model) works let's make our own.

RankMART will be *pairwise learning to rank model* of $P_f(d_i^q > d_j^q)$, i.e. probability that document i should be ranked higher than document j (both of which are associated with same query q).

- Note: random variables are usually denoted with capital letters, but keep in mind d_i^q, d_j^q in P_f on the left-hand side are such.
- Ignore for the moment what the model actually is (linear function, decision tree, ...).

How are we going to model the probability, $P_f(d_i^q > d_j^q)$ given a ranker f?

RankMART Model Definition Cont'd

We will model the probability of an event $d_i^q > d_j^q$ via **logistic function**:

$$P_f(d_i^q > d_j^q) = \frac{1}{1 + \exp(-\alpha(f(x_i^q) - f(x_j^q)))} \quad (\alpha > 0)$$

- Bigger the $f(x_i^q) - f(x_j^q)$, sometimes referred to as **margin**, bigger the probability.

What kind of (statistical) method can we use to learn the "parameters" f?

RankMART Model Training

We will train the model using **maximum likelihood estimation**. For that we need to preprocess our data into *preference judgements*:

$$I^q = \{(i,j) : y_i^q > y_j^q\}$$

- Ignore documents of the same relevance because their relative order does not matter, or does it?

A maximum likelihod estimator would be than

$$f^* = \underset{f}{\operatorname{argmax}}\, L(f) = ?$$

RankMART Model Training Cont'd

$$
\begin{aligned}
f^* &= \operatorname*{argmax}_f L(f) = \operatorname*{argmax}_f \prod_q \prod_{(i,j)\in I^q} P_f(d_i^q > d_j^q) \\
&= \operatorname*{argmax}_f \sum_q \sum_{(i,j)\in I^q} \log(P_f(d_i^q > d_j^q)) \\
&= \operatorname*{argmin}_f \sum_q \sum_{(i,j)\in I^q} \log(1 + \exp(\alpha(f(x_j^q) - f(x_i^q)))) \\
&= \operatorname*{argmin}_f \sum_q \sum_{(i,j)\in I^q} C(f(x_j^q) - f(x_i^q))
\end{aligned}
$$

This is lot more general learning method than you might think. You can have more than one judgement for the same pair of documents and they do not need to agree.

- The loss C above is so-called **cross-entropy** – by minimizing it we make the probability distribution learnt by f match as closely as possible to the empirical probability distribution induced by pairwise judgements.

Stochatic Gradient Descent

The plan is to use ensemble of regression trees for our model f, but how about using a linear function $f(x) = w^T x$, **just for now**.

- "Ranking Logistic Regression"?

Update rule for a randomly selected pair of documents (d_i^q, d_j^q):

$$
\begin{aligned}
w \longleftarrow w - \eta \frac{\partial C}{\partial w} &= w - \eta \left(\frac{\partial C}{\partial f(x_i^q)} \frac{\partial f(x_i^q)}{\partial w} + \frac{\partial C}{\partial f(x_j^q)} \frac{\partial f(x_j^q)}{\partial w} \right) \\
&= w - \eta \frac{\partial C}{\partial f(x_i^q)} \left(\frac{\partial f(x_i^q)}{\partial w} - \frac{\partial f(x_j^q)}{\partial w} \right) \\
&= w + \eta \frac{\partial C}{\partial f(x_i^q)} (x_i^q - x_j^q) \\
&= w + \eta (1 - P_f(d_i^q > d_j^q))(x_i^q - x_j^q)
\end{aligned}
$$

Mini-Batch Stochatic Gradient Descent

Single update step may be very costly (for example, one pass of backpropagation in RankNET).

Mini-batch update rule for a randomly selected query q:

$$
\begin{aligned}
w \leftarrow w - \eta \sum_{(i,j)\in I^q} \frac{\partial C}{\partial w} &= w - \eta \sum_{(i,j)\in I^q} \left(\frac{\partial C}{\partial f(x_i^q)} \frac{\partial f(x_i^q)}{\partial w} + \frac{\partial C}{\partial f(x_j^q)} \frac{\partial f(x_j^q)}{\partial w} \right) \\
&= w - \eta \sum_{(i,j)\in I^q} \left(\lambda_{ij}^q \frac{\partial f(x_i^q)}{\partial w} - \lambda_{ij}^q \frac{\partial f(x_j^q)}{\partial w} \right) \\
&= w - \eta \sum_{d_i^q \in D_q} \left(\sum_{j:(i,j)\in I^q} \lambda_{ij}^q - \sum_{j:(j,i)\in I^q} \lambda_{ji}^q \right) \frac{\partial f(x_i^q)}{\partial w} \\
&= w - \eta \sum_{d_i^q \in D_q} \lambda_i^q x_i^q
\end{aligned}
$$

Mini-Batch Stochatic Gradient Descent Cont'd

Some identities that pop out from the previous slides.

For $(\mathbf{i}, \mathbf{j}) \in \mathbf{I^q}$ (see appendix for generalization):

$$\lambda_{ij}^q = \frac{\partial C(f(x_j^q) - f(x_i^q))}{\partial f(x_i^q)} = -\frac{\partial C(f(x_j^q) - f(x_i^q))}{\partial f(x_j^q)} = -\lambda_{ij}^q$$

For any document d_i^q in dataset S:

$$\lambda_i^q = \sum_{j:(i,j)\in I^q} \lambda_{ij}^q - \sum_{j:(j,i)\in I^q} \lambda_{ji}^q = \sum_{j:(i,j)\in I^q} P_f(d_i^q < d_j^q) - \sum_{j:(j,i)\in I^q} P_f(d_i^q > d_j^q)$$

Using the mini-batch update rule we are not messing things up within the chosen query or at least not as much as in case of the previous rule.

- [Burges, C. 2010] shows that the training time of RankNET dropped from close to quadratic in the number of documents per query, to close to linear.

Lambdas as Forces

The expression for λ_i^q

$$\lambda_i^q = \sum_{j:(i,j)\in I^q} \lambda_{ij}^q - \sum_{j:(j,i)\in I^q} \lambda_{ji}^q$$

has also a very nice physical interpretation. You may think of the documents as point masses. λ_i^q is then the (resultant) force on the point mass d_i^q.

- First sum accounts for all the forces coming from *less* relevant documents – pushes d_i^q up in the ranking.
- Second sum accounts for all the forces coming from *more* relevant documents – pushes d_i^q down in the ranking.
- Try to figure out how the magnitude of the forces change during training.
- You can find out more about this in [Burges, C. et al. 2007].

Gradient Tree Boosting

We are willing to use an **ensemble of regression trees** as our ranker f:

$$f_M(x) = \sum_{i=1}^{M} \Psi(x; \Theta_i)$$

where M is the number of trees and Θ_i are the parameters of the i-th tree.

This model is also called **MART**, which stads for **M**ultiple **A**dditive **R**egression **T**rees.

How can we possibly use a bunch of regression trees and optimize our cross-entropy loss when there are no differentiable parameters?

- Sure we can via (general) optimization method – gradient tree boosting.
- We will just cover the algorithm, all the gory details can be found, for example in [Hastie, T. et al. 2001]

Gradient Tree Boosting Algorithm

RankMART Gradient Tree Boosting Algorithm

Input: preference judgements I, loss function C, and number of trees M

1. Initialize: $f_0(\cdot) \leftarrow 0$
2. For $m = 1$ to M:
   1. Compute lambdas for each document (the gradients):

$$\lambda_{im}^q = \sum_{j:(i,j)\in I^q} \frac{\partial C}{\partial f(x_i^q)} - \sum_{j:(j,i)\in I^q} \frac{\partial C}{\partial f(x_j^q)} \Bigg|_{f=f_{m-1}}$$

   2. Fit a next regression tree to the lambdas:

$$\Theta_m^* \leftarrow \underset{\Theta_m}{\operatorname{argmin}} \sum_{q}^{Q} \sum_{i=1}^{n(q)} (-\lambda_{mi}^q - \Psi(x_i^q; \Theta_m))^2$$

   3. Find the appropriate gradient step for each leaf node $\{\Psi_{mt}\}_{t=1}^J$ of the new tree $\Psi(x; \Theta_m^*)$ and apply "shrinkage" η:

$$\{\gamma_{mt}^*\}_{t=1}^J = \eta \cdot \underset{\{\gamma_{mt}\}_{t=1}^J}{\operatorname{argmin}} \sum_{\mathbf{q=1}}^{\mathbf{Q}} \sum_{\substack{(i,j)\in I^q \\ x_i^q \in \Psi_{mr}, x_j^q \in \Psi_{ms} \\ \Psi_{mr} \neq \Psi_{ms}}} \mathbf{C}(\mathbf{f_{m-1}}(\mathbf{x_j^q}) - \mathbf{f_{m-1}}(\mathbf{x_i^q})) - \gamma_{\mathbf{mr}} + \gamma_{\mathbf{ms}})$$

Gradient Tree Boosting Algorithm Cont'd

4. Update the tree:

$$\Psi(x; \Theta_m^*) = \sum_{t=1}^{J_m} \gamma_{mt}^* [\![x \in \Psi_{mt}]\!]$$

5. Update the model:

$$f_m(\cdot) \leftarrow f_{m-1}(\cdot) + \Psi(\cdot; \Theta_m^*)$$

3. Return $f_M(\cdot)$.

One way to optimize the gradient step in a leaf is using **Newton's method** (just one step, starting with $\gamma_{mt} = 0$):

$$\gamma_{\mathbf{mt}}^* = -\eta \frac{\sum_{\mathbf{x_i^q} \in \boldsymbol{\Psi}_{\mathbf{mt}}} \lambda_{\mathbf{i}}^{\mathbf{q}}}{\sum_{\mathbf{x_i^q} \in \boldsymbol{\Psi}_{\mathbf{mt}}} \omega_{\mathbf{i}}^{\mathbf{q}}}$$

Gradient Tree Boosting Algorithm Cont'd

Where ω_i^q is (not correctly!) defined as

$$\omega_{\mathbf{i}}^{\mathbf{q}} = \frac{\mathbf{d}\lambda_{\mathbf{i}}^{\mathbf{q}}}{\mathbf{df_{m-1}}(\mathbf{x_i^q})}$$

Lot of things are hidden behind the formulas above, what you can actually read from scientific papers can be pretty "hazy", see [Burges, C. 2010], for example.

See the appendix for the exact derivation of the γ_{mt}^* and for what I mean by (not correctly!).

RankMART Summary

This is the summary what we did so far:

1. We created a pairwise learning to rank model of $P(d_i > d_j)$.
2. We derived a SGD learning algorithm for a logistic regression model, and prepared data for it.
3. We saw a gradient tree boosting method and applied it to train an ensemble model under the (fictitious) name RankMART.

All of this just to find out that LambdaMART is just RankMART with an additional twist that will make it work better.

LambdaMART Demystified

To get the notorious LambdaMART, just take our model RankMART and do the following:

1. Before training a new regression tree, sort the documents according to the current model f_{m-1}.
2. Compute lambdas in following way (see appendix):

$$\lambda_{ij}^{q} = \frac{-\alpha|\Delta Z_{ij}^{q}|}{1 + exp(\alpha(f(x_i^q) - f(x_j^q)))}$$

3. Do a single step of Newton's method to optimize gradient (lambda) predictions in terminal nodes:

$$\gamma_{mt}^{*} = -\eta \frac{\sum_{x_i^q \in \Psi_{mt}} \lambda_i^q}{\sum_{x_i^q \in \Psi_{mt}} \omega_i^q}$$

4. Voilà! RankMART's LambdaMART :).

LambdaMART Demystified Cont'd

The only difference (regardless for the maximization) is the $|\Delta Z_{ij}^q|$ term in the definition of lambdas.

1. This term can be computed from any IR performance measure, such as NDCG, MAP, ERR, ...
2. It is an absolute value of the change in the performance metric given by swapping the rank positions of d_i^q and d_j^q, while leaving the other documents untouched.
3. It has been empirically demonstrated that pluging in NDCG, LambdaRank (uses NN instead of MART) can directly optimize it.

LambdaMART Demystified Cont'd

To understand how LambdaMART works, consider the following figure demonstrating the problems with target/training performance measure mismatch (think of WTA vs pair-wise errors):

Figure was adopted from [Burges, C. 2010].

LambdaMART Demystified Cont'd

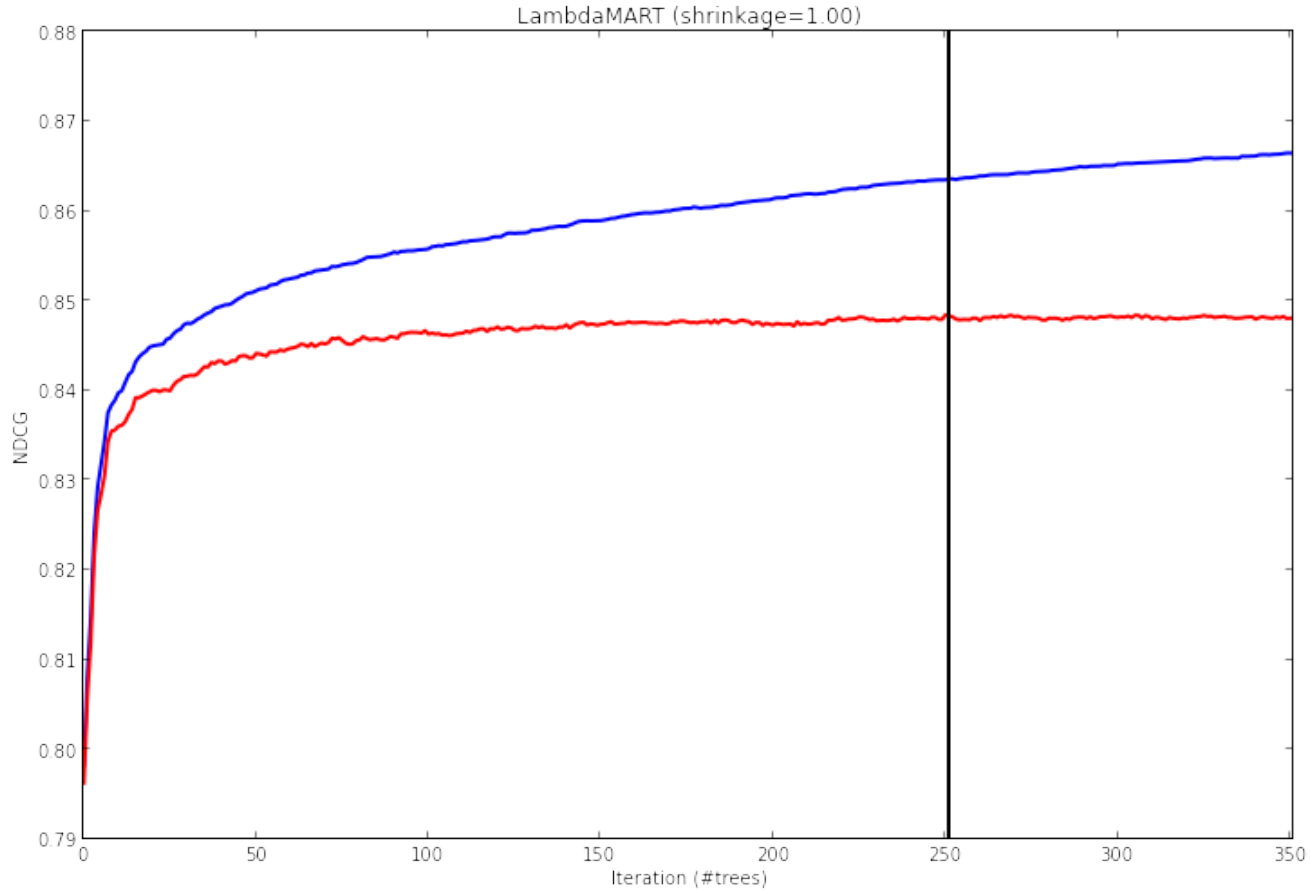
LambdaMART basically treats the pairwise errors differently. It weighs them according to

1. how badly the model orders the corresponding pairs in terms of the margin.
2. how important the correct order is from the performance measure's perspective.

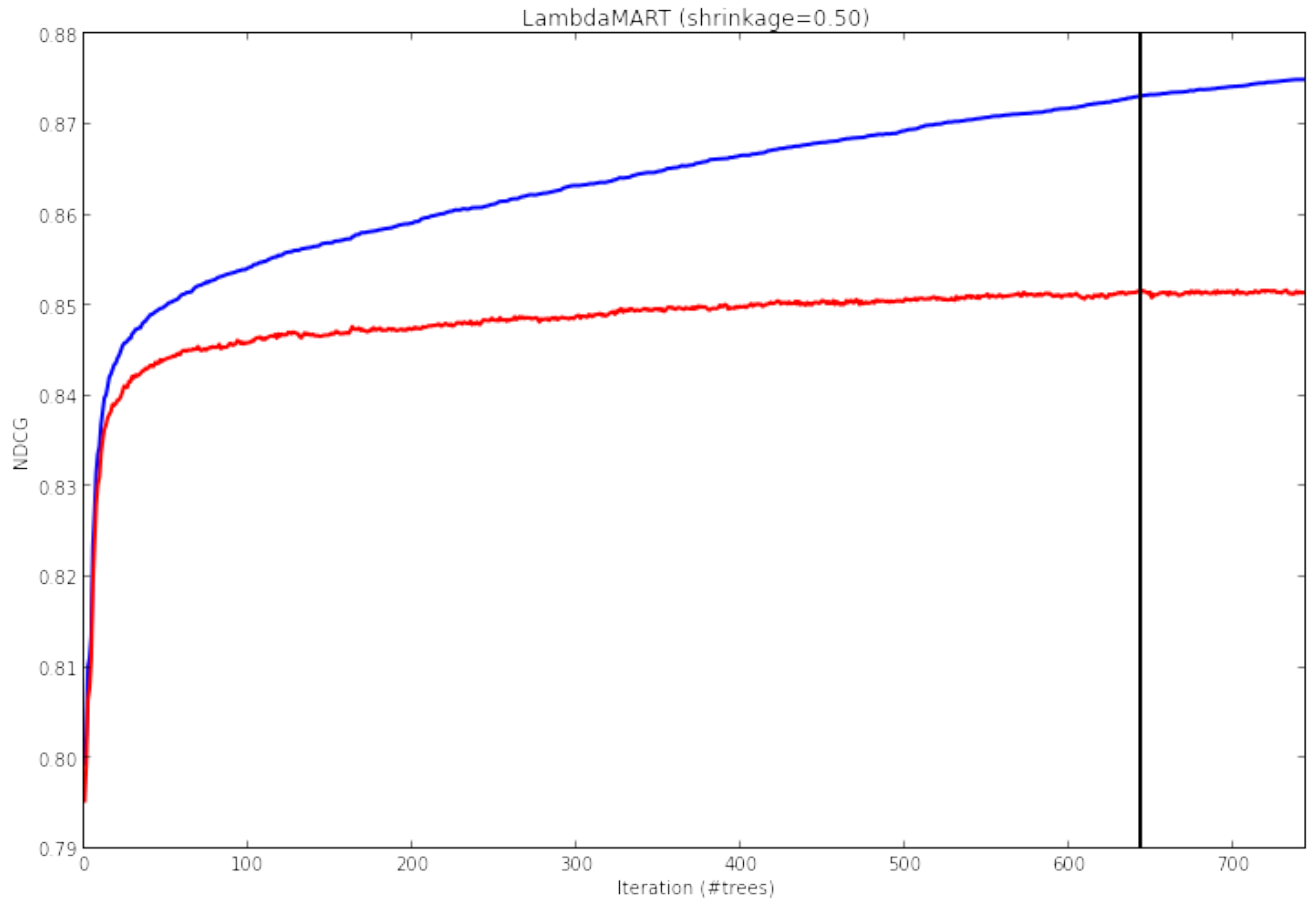
Still, the model has its own flaws, see [Svore, K. et al. 2011], for example. Trying to fix them might as well become your future project.

Appendix

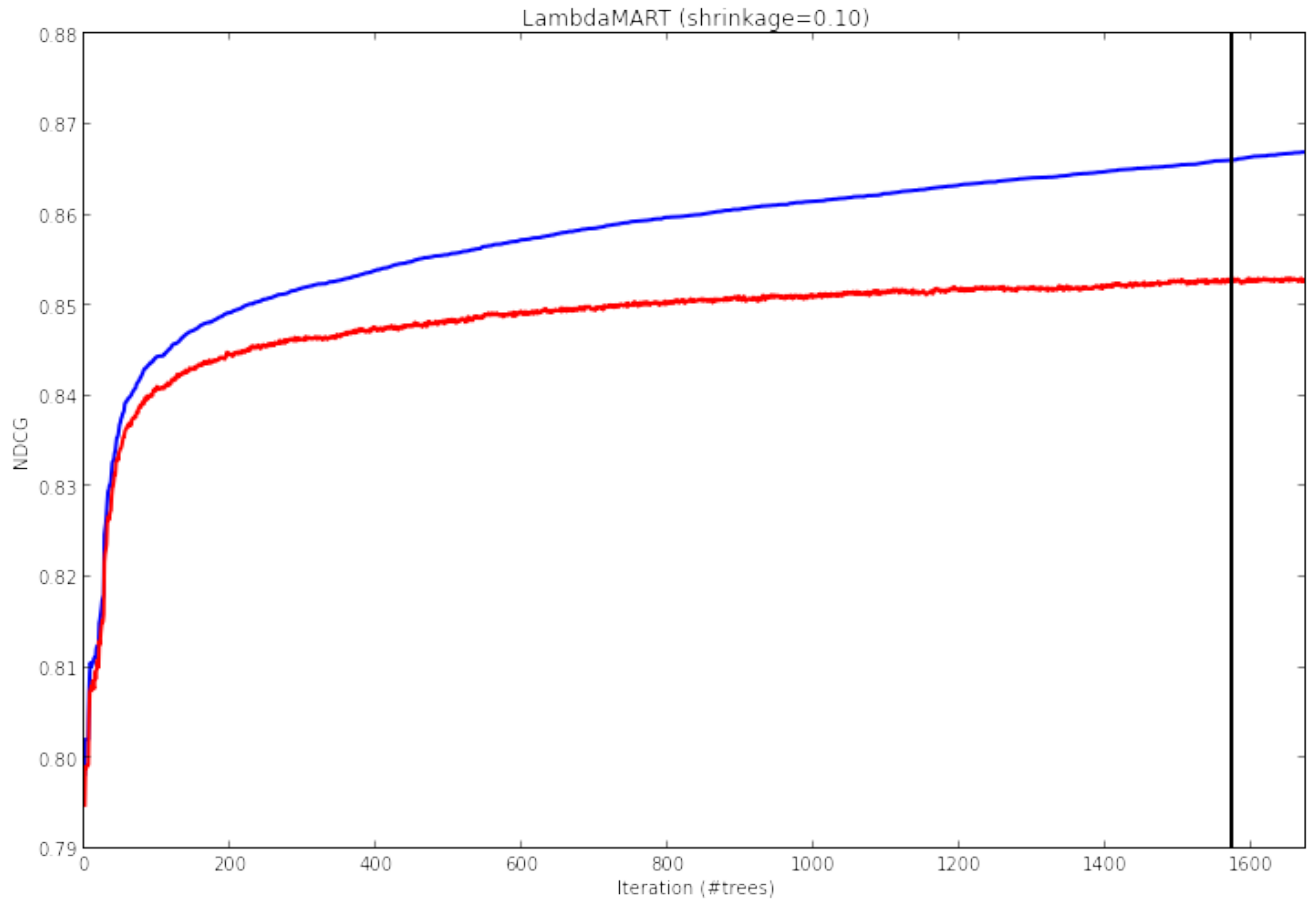
LambdaMART Training: Shrinkage 1.0



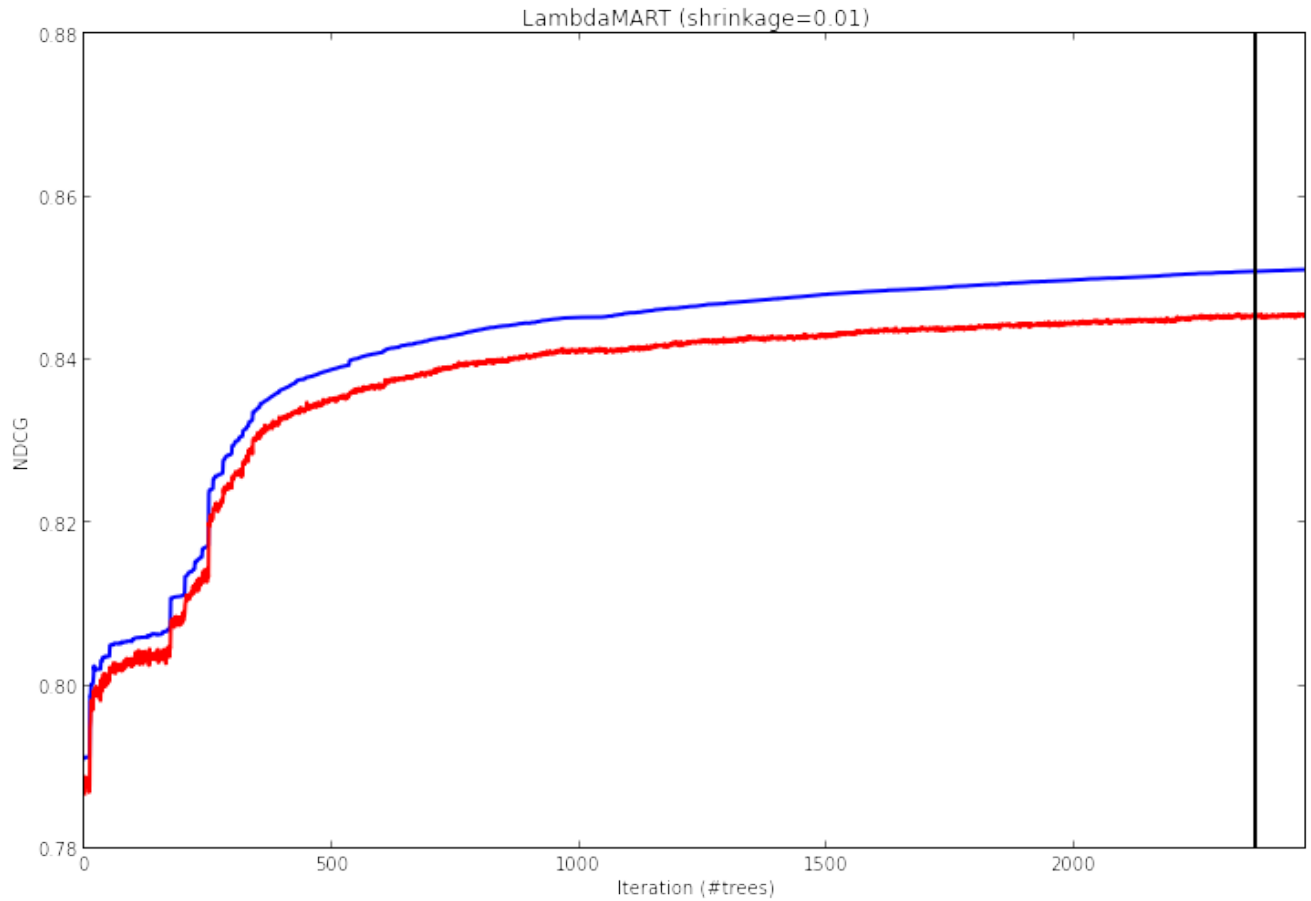
LambdaMART Training: Shrinkage 0.5



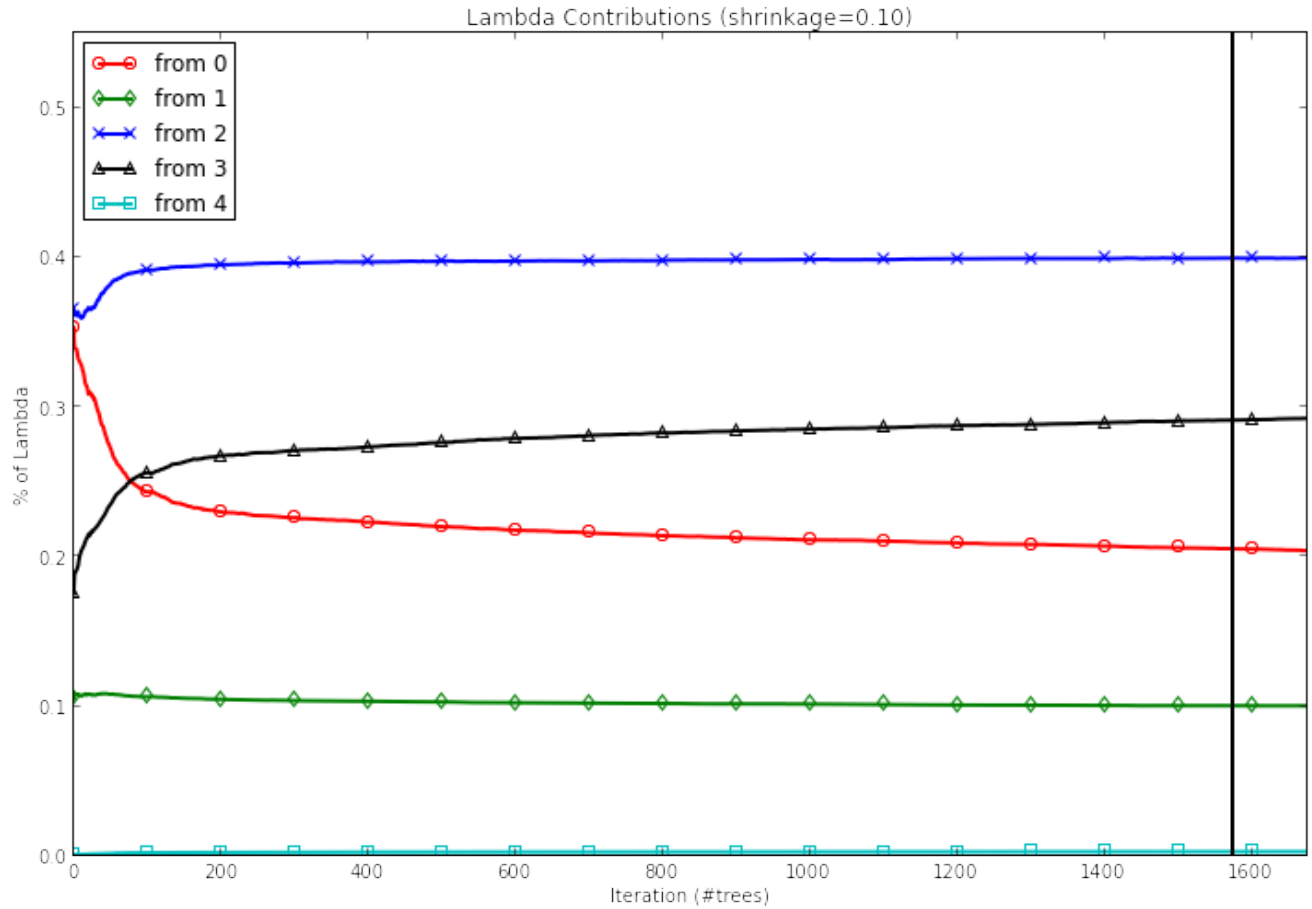
LambdaMART Training: Shrinkage 0.1



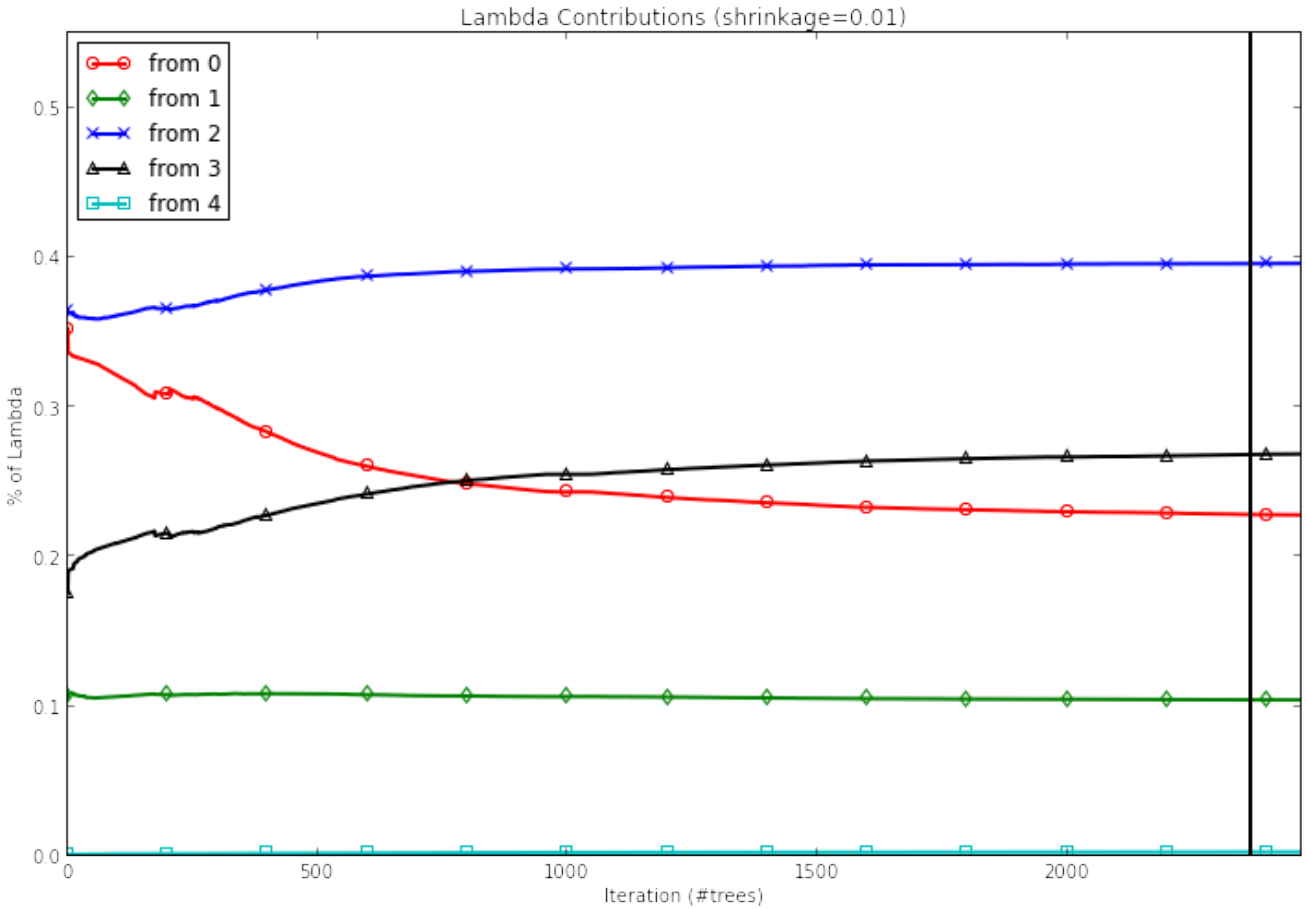
LambdaMART Training: Shrinkage 0.01



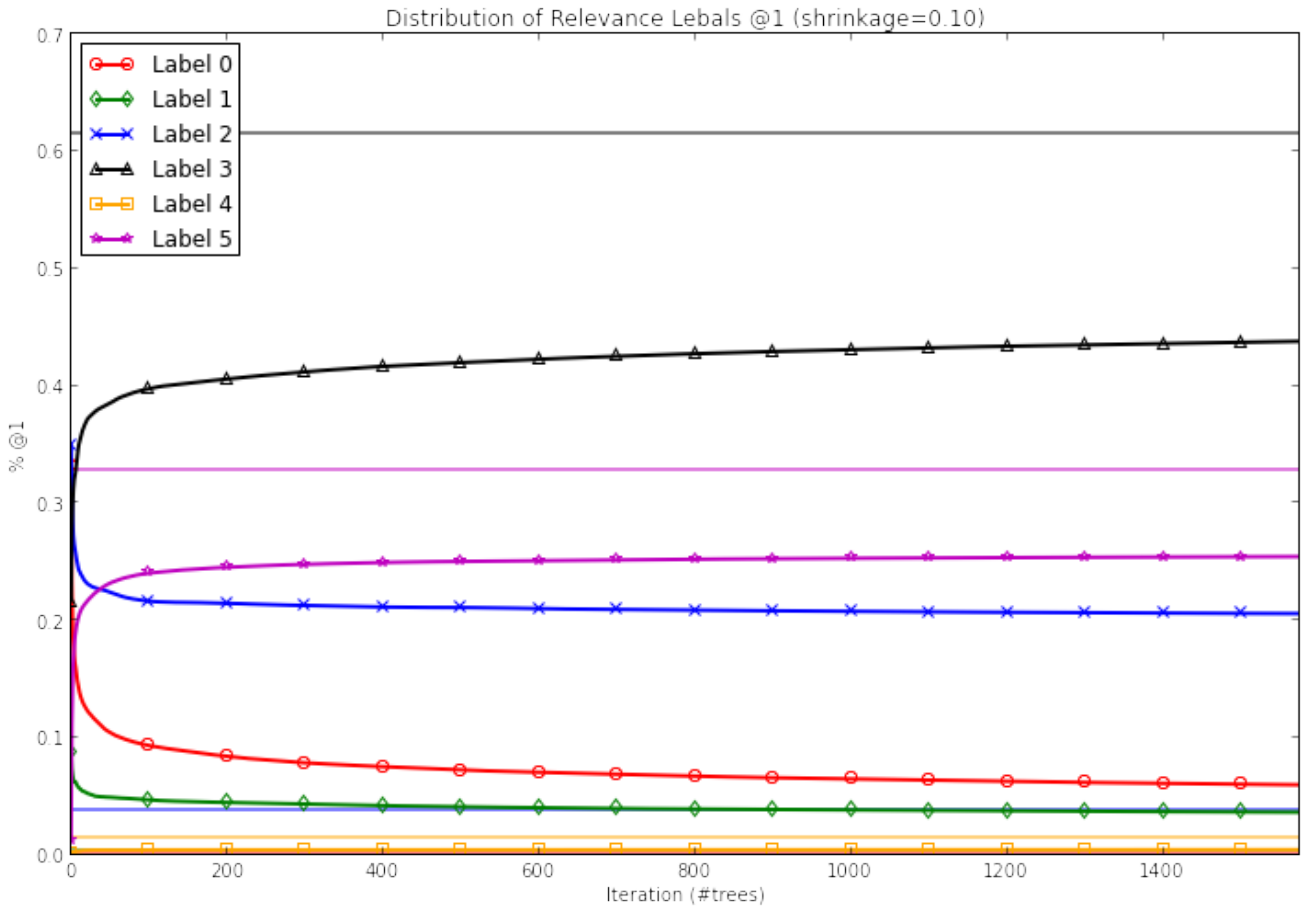
LambdaMART Training: Lambda Contributions (0.1)



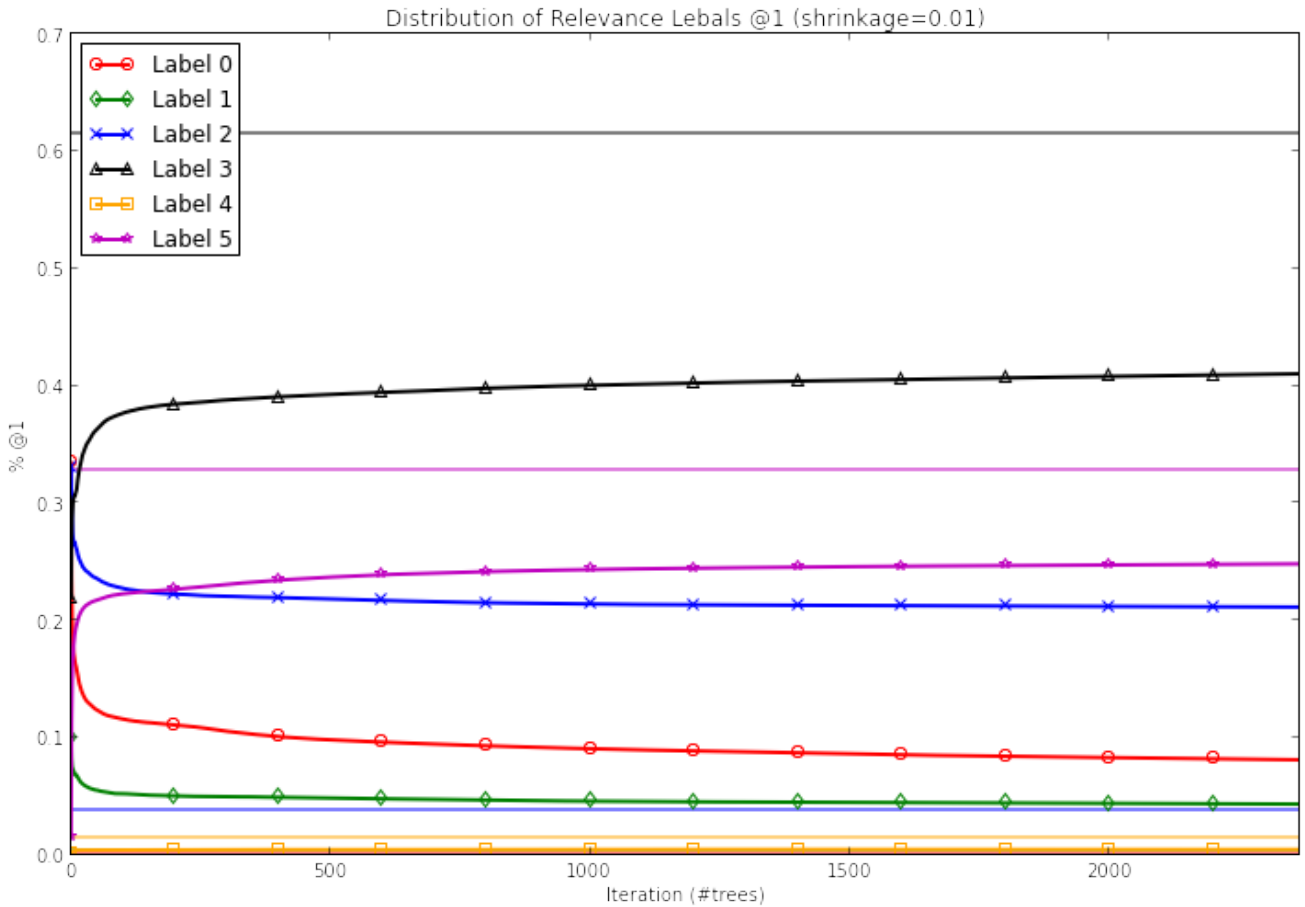
LambdaMART Training: Lambda Contributions (0.01)



LambdaMART Training: Rel. Label Distribution (0.1)



LambdaMART Training: Rel. Label Distribution (0.01)



Generalization of Lambda formula

The order, in which you plug x_i^q and x_j^q into the formula for computation of λ_{ij}^q is not arbitrary! The formula (silently) assumes that $(i,j) \in I^q$ holds!

To generalize the formula a bit, consider S_{ij}^q defined as follows

$$S_{ij}^q = \begin{cases} +1 & (i,j) \in I^q \\ -1 & (j,i) \in I^q \end{cases}$$

then λ_{ij}^q (for d_i^q and d_j^q of arbitrary relevance) is

$$\lambda_{ij}^q = \frac{-\alpha S_{ij}^q |\Delta Z_{ij}^q|}{1 + exp(\alpha S_{ij}^q (f(x_i^q) - f(x_j^q)))}$$

and conveniently for every d_i^q

$$\lambda_i^q = \sum_{j:S_{ij}^q} \lambda_{ij}^q$$

Deriving Optimal Gradient Step for γ^*_{mt}

Optimization of

$$\{\gamma^*_{mt}\}_{t=1}^{J} = \eta \cdot \underset{\{\gamma_{mt}\}_{t=1}^{J}}{\operatorname{argmin}} \sum_{\mathbf{q=1}}^{\mathbf{Q}} \sum_{\substack{(i,j)\in I^q \\ x_i^q \in \Psi_{mr}, x_j^q \in \Psi_{ms} \\ \Psi_{mr} \neq \Psi_{ms}}} \mathbf{C}(\mathbf{f_{m-1}}(\mathbf{x_j^q}) - \mathbf{f_{m-1}}(\mathbf{x_i^q})) + \gamma_{\mathbf{ms}} - \gamma_{\mathbf{mr}})$$

will be demonstrated for a single x_i^q, where f will be substituted for f_{m-1} (for convenience), also whenever Ψ_{ms} appears it is **never** equal Ψ_{mr}:

$$C_{x_i^q} = \sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} C(f(x_j^q) - f(x_i^q) - \gamma_{mr} + \gamma_{ms}) + \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} C(f(x_i^q) - f(x_j^q) + \gamma_{mr} - \gamma_{ms})$$

This is one-dimensional problem, taking a derivative with respect to γ_{mr} gives us:

$$\frac{dC_{x_i^q}}{d\gamma_{mr}} = \sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{dC(f(x_j^q) - f(x_i^q) - \gamma_{mr} + \gamma_{ms})}{d\gamma_{mr}} + \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{dC(f(x_i^q) - f(x_j^q) + \gamma_{mr} - \gamma_{ms})}{d\gamma_{mr}}$$

$$= \sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{dC(f(x_j^q) - f(x_i^q) - \gamma_{mr} + \gamma_{ms})}{df(x_i^q)} - \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{dC(f(x_i^q) - f(x_j^q) + \gamma_{mr} - \gamma_{ms})}{df(x_j^q)}$$

Deriving Optimal Gradient Step for γ^*_{mt} Cont'd

Continuing from previous slide:

$$\frac{dC_{x_i^q}}{d\gamma_{mr}} = \sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{dC(f(x_j^q) - f(x_i^q) - \gamma_{mr} + \gamma_{ms})}{df(x_i^q)} - \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{dC(f(x_i^q) - f(x_j^q) + \gamma_{mr} - \gamma_{ms})}{df(x_j^q)}$$

$$= \sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \lambda_{ij}^q - \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \lambda_{ji}^q$$

$$= \sum_{x_i^q \in \Psi_{mr}} \sum_{j:(i,j)\in I^q} \lambda_{ij}^q - \sum_{j:(j,i)\in I^q} \lambda_{ji}^q$$

$$= \sum_{x_i^q \in \Psi_{mr}} \sum_{j:(i,j)\in I^q} \lambda_i^q$$

The derivatives of λ_{ij}^q above are correct only when we plugin 0 for the γ values in the gradients. But still, the correct nominator in Newton's method popped out.

Deriving Optimal Gradient Step for γ^*_{mt} Cont'd

To finally get the Newton's step, we need to compute $\frac{dC_{x_i^q}}{d\gamma_{mr}^2}$. For that it is good to realize that λ_{ij}^q is defined for our cross-entropy loss C as follows:

$$\lambda_{ij}^q = \frac{\partial C(f(x_j^q) - f(x_i^q) - \gamma_{mr} + \gamma_{ms})}{df(x_i^q)} = \frac{-\alpha}{1 + \exp(\alpha(f(x_i^q) - f(x_j^q) + \gamma_{mr} - \gamma_{ms})} = -\alpha \cdot \sigma(f(x_j^q) - f(x_i^q) - \gamma_{mr} + \gamma_{ms})$$

where γ values need to be evaluated at 0 to match our earlier definition of λ_{ij}^q (and here $y_i^q > y_j^q$!), but nothing is preventing us from taking the 2nd derivative with respect to γ_{mt}:

$$\begin{aligned}
\frac{dC_{x_i^q}}{d\gamma_{mr}^2} &= -\alpha \sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{d\sigma(f(x_j^q) - f(x_i^q) - \gamma_{mr} + \gamma_{ms})}{d\gamma_{mr}} - \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{d\sigma(f(x_i^q) - f(x_j^q) + \gamma_{mr} - \gamma_{ms})}{d\gamma_{mr}} \\
&= -\alpha \sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{d\sigma(f(x_j^q) - f(x_i^q) - \gamma_{mr} + \gamma_{ms})}{df(x_i^q)} + \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{d\sigma(f(x_i^q) - f(x_j^q) + \gamma_{mr} - \gamma_{ms})}{df(x_j^q)} \\
&= -\sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \lambda_{ij}^q (1 + \frac{\lambda_{ij}^q}{\alpha}) + \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \lambda_{ji}^q (1 + \frac{\lambda_{ji}^q}{\alpha})
\end{aligned}$$

Deriving Optimal Gradient Step for γ^*_{mt} Cont'd

Continuing from previous slide:

$$\frac{dC_{x_i^q}}{d\gamma^2_{mr}} = -\sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \lambda_{ij}^q \left(1 + \frac{\lambda_{ij}^q}{\alpha}\right) + \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \lambda_{ji}^q \left(1 + \frac{\lambda_{ji}^q}{\alpha}\right)$$

$$= \sum_{x_i^q \in \Psi_{mr}} \sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{d\lambda_{ij}^q}{df(x_i^q)} - \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \frac{d\lambda_{ji}^q}{df(x_i^q)}$$

$$= \sum_{x_i^q \in \Psi_{mr}} \frac{d\left(\sum_{\substack{j:(i,j)\in I^q \\ x_j^q \in \Psi_{ms}}} \lambda_{ij}^q - \sum_{\substack{j:(j,i)\in I^q \\ x_j^q \in \Psi_{ms}}} \lambda_{ji}^q\right)}{df(x_i^q)}$$

$$\neq \sum_{\mathbf{x_i^q \in \Psi_{mr}}} \frac{\mathbf{d\lambda_i^q}}{\mathbf{df(x_i^q)}} = \sum_{\mathbf{x_i^q \in \Psi_{ms}}} \omega_{\mathbf{i}}^{\mathbf{q}}$$

Equality holds if no two documents from the same query end up in the same leaf of the regression tree!

Deriving Optimal Gradient Step for γ^*_{mt} Cont'd

The significance of using an incorrect terms in the denominator of Newton's step is unknown to me.

- Given the fact that the trees in LambdaMART have usually very few leaves (< 10) and queries usually have more documents ($>> 10$), the deviation from correct terms can be substantial (hypothesis).
- On the other hand, given all the approximations... and since ω_i's are always positive, they only reduce the magnitude of the predicted gradients, which on the one hand slows down convergence in the "correct" directions, but on the other makes smaller steps in "wrong" direction.
- From my experience, LambdaMART is pretty robust to different modification of lambdas (in some case even to wrong computation of them :)).

References I

Xia, F., Liu, T.Y., Wang, J., Zhang, W., Li, H. (2008)

Listwise Approach to Learning to Rank: Theory and Algorithm

Proceedings of the 25th International Conference on Machine Learning, ICML '08, 1192 – 1199.

W. Chen, T.-Y. Liu, Y. Lan, Z. Ma, and H. Li. (2009)

Ranking measures and loss functions in learning to rank.

In Advances in Neural Information Processing Systems 22, NIPS '09, 315 – 323.

Christopher J. C. Burges (2010)

From RankNet to LambdaRank to LambdaMART: An Overview

Microsoft Research Technical Report

C.J.C. Burges and R. Ragno and Q.V. Le (2007)

Learning to Rank with Non-Smooth Cost Functions

Advances in Neural Information Processing Systems 19

References II

Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome (2001)

The Elements of Statistical Learning

Springer Series in Statistics, Springer New York Inc.

Svore, Krysta M. and Volkovs, Maksims N. and Burges, Christopher J.C.

Learning to Rank with Multiple Objective Functions

Proceedings of the 20th International Conference on World Wide Web, WWW '11, 367–376.