

Learning to Rank

Soumen Chakrabarti
IIT Bombay

<http://www.cse.iitb.ac.in/~soumen>

The world of IR before learning to rank

- ▶ Documents, queries represented in the **vector space model**
- ▶ Term frequency TF, (inverse) document frequency IDF
- ▶ Cosine match and BM25 match

$$\text{BM25}(q, d) = \sum_{\text{term } t} \frac{\text{IDF}(t) \text{TF}(t, d) (a + 1)}{\text{TF}(t, d) + a \left(1 - b + b \frac{\text{length}(d)}{\text{length}}\right)}$$

where $\overline{\text{length}}$ is the average doc length

- ▶ You wouldn't have guessed this form overnight!
- ▶ a, b tuned by hand
- ▶ Elegant post-facto theory

Ranking in Web search

- ▶ User query q , Web pages $\{v\}$
- ▶ (q, v) can be represented with a rich feature vector
- ▶ **Text match score** with title, anchor text, headings, bold text, body text, . . . , of v as a hypertext document
- ▶ Exact phrase match, URL match, product/service catalog match
- ▶ **Pagerank, topic-specific Pageranks, personalized Pageranks of v as a node in the Web graph**
- ▶ Estimated location of user, commercial intent, . . .
- ▶ Prior click stats for same/similar queries
- ▶ Must we guess the relative importance of these features?
- ▶ How to combine these into a single scoring function on (q, v) so as to induce a ranking on $\{v\}$?

Ranking for ad and link placement

- ▶ Here, the “query” is the surfer’s contextual information
- ▶ More noisy than queries, which are noisy enough!
- ▶ Plus page and site contents
- ▶ A response is an ad to place, or a link to insert
- ▶ Must rank and select from a large pool of available ads or links

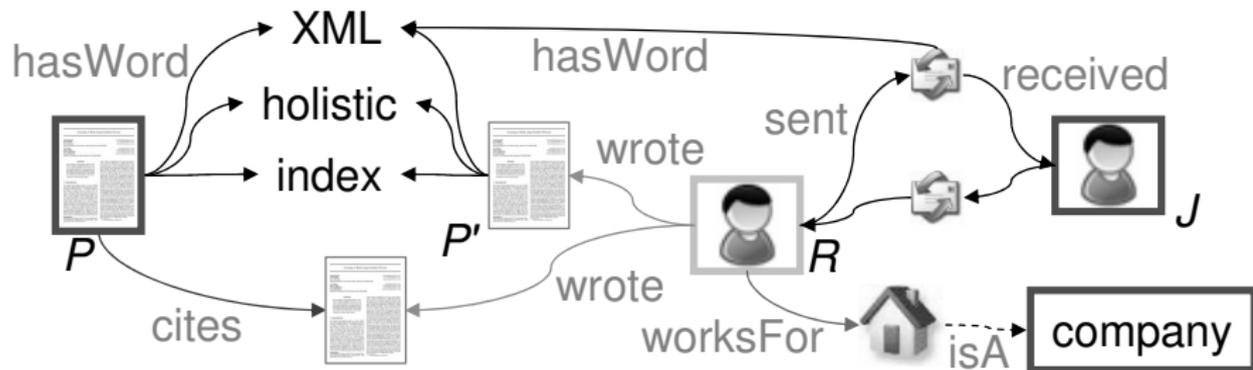
Ranking in desktop search

- ▶ The Web has only a few kinds of hyperlinks: same-host subdirectory, same-host superdirectory, same-host across-path, different-host same-domain, different-domain etc.
- ▶ Often differentiated by hardwired policy, e.g, HITS completely ignores same-host links
- ▶ Entity-relationship (ER) graphs are richer
- ▶ E.g. A personal information management (PIM) system has many node/entity types (person, organization, email, paper, conference, phone number) and edge/relation types (works-for, sent, received, authored, published-in)
- ▶ Ranking needs to exploit the richer type system
- ▶ Don't want to guess the relative importance of edge types (may be dependent on queries)

Desktop/enterprise search example

Journal editor J must find reviewer R from a company for a submitted paper P

- ▶ P shares words with papers P' written by R
- ▶ P cites papers P' written by R
- ▶ R works for organization O is-a company
- ▶ R and J have exchanged many emails



Relevance feedback for ranking in graphs

- ▶ Relevance feedback is well-explored in traditional IR
- ▶ User-assisted local modification of ranking function for vector-space models
- ▶ How to extend these to richer data representations that incorporate entities, relationship links, entity and relation types?
- ▶ Surprisingly unexplored area

Preliminaries

- ▶ Training and evaluation scenarios
- ▶ Measurements to evaluate quality of ranking
 - ▶ Label mismatch loss functions for ordinal regression
 - ▶ Preference pair violations
 - ▶ Area under (true positive, false positive) curve
 - ▶ Average precision
 - ▶ Rank-discounted reward for relevance
 - ▶ Rank correlations
- ▶ What's useful vs. what's easy to learn

Ranking in vector spaces

Instance v is represented by a feature vector $x_v \in \mathbb{R}^d$

- ▶ Itemwise, pairwise, listwise paradigms
- ▶ Regression, gradient boosting
- ▶ Discriminative max-margin ranking (RankSVM)
- ▶ Linear-time max-margin approximation
- ▶ Probabilistic ranking in vector spaces (RankNet)
- ▶ Sensitivity to absolute rank and cost of poor rankings
- ▶ Max-margin and conditional listwise approaches
- ▶ Local learning for query classes (navigational, informational)
- ▶ Diversity in ranking

Ranking in graphs

Instance v is a node in a graph $G = (V, E)$

- ▶ The graph-Laplacian approach
 - ▶ Assign scores to nodes to induce ranking
 - ▶ G imposes a smoothness constraint on node scores
 - ▶ Large difference between neighboring node scores penalized
- ▶ The Markov walk approach
 - ▶ Random surfer, Pagerank and variants; by far most popular way to use graphs for scoring nodes
 - ▶ Walks constrained by preferences
 - ▶ How to incorporate node, edge types and query words
- ▶ Connections between the two approaches

Forms of training input

Regression: For each entity x , an absolute real score y
(unrealistic to expect users to assign absolute scores)

Ordinal regression: For each entity x , a score y from a discrete, ordered domain, such as a r -point scale
(implemented in many sites like Amazon.COM)



Bipartite ranking: Ordinal regression with $r = 2$

Pairwise preferences: A (possibly inconsistent) partial order between entities, expressed as a collection of “ $u \prec v$ ” meaning “ u is less preferred than v ” (low cognitive load on users, can be captured from click-logs and eye-tracking data)

Complete rank order: A total order on the entities **but no scores** (highly impractical for large entity sets)

Prefix of rank order: A total order on the top- k entities, meaning that all the other entities are worse (iffy)

Evaluation of ranking algorithms

Error on score vectors: In case of standard regression, if \hat{f} is the score assigned by the algorithm and f is the “true score”, measure $\|\hat{f} - f\|_1$ or $\|\hat{f} - f\|_2$.

Ordinal reversals: If $y_u > y_v$ and $\hat{f}(u) < \hat{f}(v)$ then u and v have been **reversed**. Count the number of reversed pairs.

Precision at k : For a specific query q , let T_k^q and \hat{T}_k^q be the top- k sets as per f and \hat{f} scores. The precision at k for query q is defined as $|T_k^q \cap \hat{T}_k^q|/k \in [0, 1]$. Average over q .

Relative aggregated goodness (RAG): For a specific query q ,

$$\text{RAG}(k, q) = \frac{\sum_{v \in \hat{T}_k^q} f(v)}{\sum_{v \in T_k^q} f(v)} \in [0, 1]$$

Note that \hat{f} is not used! Average over q .

Evaluation of ranking algorithms (2)

Mean reciprocal rank (MRR): For each query there is one or more **correct** responses. Suppose for specified query q , the first rank at which a correct response occurs is $R(q)$. Then MRR is

$$\frac{1}{|Q|} \sum_{q \in Q} \frac{1}{R(q)}$$

Normalized discounted cumulative gain (NDCG): For a specific query q ,

$$N_q \sum_{i=1}^k \frac{2^{\text{rating}(i)} - 1}{\log(1 + i)}$$

Here N_q is a normalization factor so that a perfect ordering gets NDCG score of 1 for each query, k is the number of top responses considered, and $\text{rating}(i)$ is the evaluator rating for the item returned at position i .

Evaluation of ranking algorithms (3)

Pair preference violation: If $u \prec v$ and $\hat{f}(u) > \hat{f}(v)$ a pair has been **violated**. Count the number of pair violations.

Rank correlation: Order entities by decreasing $f(u)$ and compute a rank correlation with the ground truth ranking. Impractical if a full ground truth ranking is expected.

Prefix rank correlation: Let exact and approximate scores be denoted by $S_q^k(v)$ and $\hat{S}_q^k(v)$ respectively for items v , where the scores are forced to zero if $v \notin T_k^q$ and $v \notin \hat{T}_k^q$. A node pair $v, w \in T_k^q \cup \hat{T}_k^q$ is *concordant* if $(S_q^k(v) - S_q^k(w))(\hat{S}_q^k(v) - \hat{S}_q^k(w))$ is strictly positive, and *discordant* if it is strictly negative. It is an *exact-tie* if $S_q^k(v) = S_q^k(w)$, and is an *approximate tie* if $\hat{S}_q^k(v) = \hat{S}_q^k(w)$. If there are c, d, e and a such pairs

Evaluation of ranking algorithms (4)

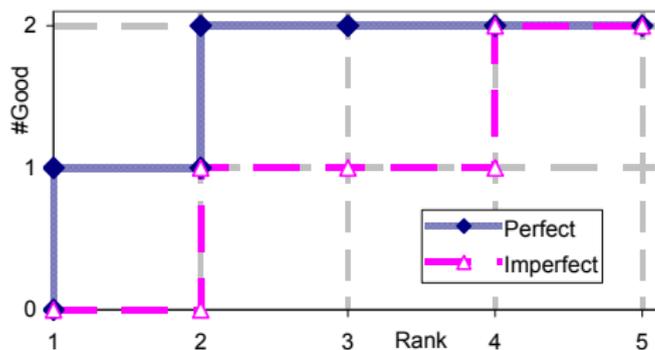
respectively, and m pairs overall in $T_k^q \cup \hat{T}_k^q$, then Kendall's τ is defined as

$$\tau(k, q) = \frac{c - d}{\sqrt{(m - e)(m - a)}} \in [-1, 1].$$

Average over q .

- ▶ Theoretically sound and scalable rank learning techniques typically address simpler evaluation objectives
- ▶ Designing learning algorithms for the more complicated, non-additive evaluation objectives is very challenging
- ▶ Sometimes, we are lucky enough to establish a connection between the two classes of objectives

Area under the curve (AUC)



- ▶ Good = 1, bad = 0
- ▶ For perfect ordering 1, 1, 0, 0, 0 plot passes through (1, 0), (1, 1), (2, 1), (2, 2), (3, 2), (4, 2), (5, 2)
- ▶ For imperfect ordering 0, 1, 0, 1, 0, plot passes through (1, 0), (2, 0), (2, 1), (3, 1), (4, 1), (4, 2), (5, 2)
- ▶ Area between two plots related to number of pair inversions

Bipartite ranking and AUC

- ▶ In bipartite ranking labeled data is of the form (x, y) where $y \in \{-1, 1\}$
- ▶ Algorithm orders instances by decreasing $f(x)$
- ▶ For $i = 0, 1, \dots, n$
 - ▶ Assign label $+1$ to the first i instances
 - ▶ Assign label -1 to the rest
 - ▶ True positive rate at i

$$\frac{\text{number of positive instances labeled positive}}{\text{number of positive instances}}$$

- ▶ False positive rate at i

$$\frac{\text{number of negative instances labeled positive}}{\text{number of negative instances}}$$

- ▶ Plot $y = \text{TruePosRate}$ vs. $x = \text{FalsePosRate}$
- ▶ Measure area under curve

AUC and pair preference violations

- ▶ m positive and n negative examples
- ▶ Area under curve (AUC) using f for ranking can also be written as

$$\hat{A}(f, T) = \frac{1}{mn} \sum_{\substack{i:y_i=+1 \\ j:y_j=-1}} \left(\llbracket f(i) > f(j) \rrbracket + \frac{1}{2} \llbracket f(i) = f(j) \rrbracket \right)$$

where T is the training set

- ▶ The important part is the **fraction of satisfied pair preferences** between positive and negative instances
- ▶ Optimizing AUC is different from optimizing 0/1 error

y_i	-1	-1	-1	-1	+1	+1	+1	+1
$f_1(x_i)$	-2	-1	3	4	1	2	5	6
$f_2(x_i)$	-2	-1	5	6	1	2	3	4

Concordant and discordant instance pairs

- ▶ Suppose there are R relevant documents in response to a query
- ▶ The search engine creates a ranking r_{engine} which lists them at ranks $p_1 < p_2 < \dots < p_R$
- ▶ An ideal system creates a ranking r_{ideal} that lists all relevant documents before any irrelevant document
- ▶ But keeps the relative ordering within the relevant and irrelevant subsets the same

$$r_{\text{engine}} = d_1^+, d_2^-, d_3^+, d_4^+, d_5^-, d_6^-, d_7^+, d_8^-$$

$$r_{\text{ideal}} = d_1^+, d_3^+, d_4^+, d_7^+; d_2^-, d_5^-, d_6^-, d_8^-$$

- ▶ Let there be Q discordant pairs in r_{engine} compared to r_{ideal}

Relating ranks and discordant pairs

- ▶ Account for Q as follows: First consider the relevant document at position p_1 in r_{engine} . Because it has been pushed out from position 1 to position p_1 , the number of inversions introduced is $p_1 - 1$.
- ▶ For the document at position p_2 in r_{engine} , the number of inversions introduced is $p_2 - 1 - 1$, the last “-1” thanks to having the first relevant document ahead of it.
- ▶ Summing up, we get

$$\sum_{i=1}^R p_i - 1 - (i - 1) = Q, \quad \text{or}$$

$$\sum_{i=1}^R p_i = Q + \sum_{i=1}^R i = Q + \frac{R(R+1)}{2} = Q + \binom{R+1}{2}.$$

Average precision

- ▶ The **average precision** of r_{engine} wrt r_{ideal} is defined as

$$\text{AvgPrec}(r_{\text{engine}}, r_{\text{ideal}}) = \frac{1}{R} \sum_{i=1}^R \frac{i}{p_i}$$

- ▶ Like NDCG, average precision rewards the search engine if all p_i are as small as possible
- ▶ Intuitively, if Q is small, $\text{AvgPrec}(r_{\text{engine}}, r_{\text{ideal}})$ should be large.
- ▶ This can be formalized by framing an optimization problem that gives a lower bound to $\text{AvgPrec}(r_{\text{engine}}, r_{\text{ideal}})$ given a fixed Q (and R)

Bounding average precision given Q

- ▶ To lower bound average precision, optimize:

$$\min_{p_1, \dots, p_R} \frac{1}{R} \sum_{i=1}^R \frac{i}{p_i} \quad \text{such that}$$

$$p_1 + \dots + p_R = Q + \binom{R+1}{2}$$

$$1 \leq p_1 < p_2 < \dots < p_R$$

p_1, \dots, p_R are positive integers

- ▶ Relaxing the last two constraints can only decrease the optimal objective, so we still get a lower bound
- ▶ The relaxed optimization is also convex because $1/p_i$ is convex in p_i , as far as p_i is concerned the numerator i is a “constant”, and sum of convex functions is convex

Solving the relaxed optimization

- ▶ Using the Lagrangian method, we get

$$\mathcal{L}(p_1, \dots, p_R; \lambda) = \frac{1}{R} \sum_{i=1}^R \frac{i}{p_i} + \lambda \left(\sum_{i=1}^R p_i - Q - \binom{R+1}{2} \right)$$
$$\therefore \frac{\partial \mathcal{L}}{\partial p_i} = -\frac{i}{R p_i^2} + \lambda \stackrel{\text{set}}{=} 0 \quad \text{to get} \quad p_i^* = \sqrt{\frac{i}{R\lambda}}.$$

- ▶ Replace back in the Lagrangian, set the derivative wrt λ to zero, and again substitute in the Lagrangian to get the optimal objective (in the relaxed optimization) as

$$\text{AvgPrec}(r_{\text{engine}}, r_{\text{ideal}}) \geq \frac{\left(\sum_{i=1}^R \sqrt{i} \right)^2}{R \left(Q + \binom{R+1}{2} \right)}$$

- ▶ Q and the lower bound on average precision are inversely related, which makes sense.

Broad learning techniques

Itemwise: For query q , each feature vector x_{qi} has an associated relevance $z_{qi} \in \mathbb{R}$.

- ▶ Try to learn a regression f from x_{qi} to z_{qi}
- ▶ Ordinal regression, MCRANK
- ▶ Loss may look like $\sum_i (f(x_{qi}) - z_{qi})^2$

Pairwise: For query q , try to learn to order document pairs i, j , i.e., whether $i \prec_q j$ or $j \prec_q i$

- ▶ RANKSVM, RANKNET, RANKBOOST
- ▶ Loss may look like $\sum_{g,b} \mathbb{I}[f(x_{qg}) < f(x_{qb})]$

Listwise: For each query q , try to make a structured prediction of the ideal permutation y_q of documents, given all their feature vectors x_q . SVMMAUC, SVMMAP, DORM, ...

Warmup: standard regression

- ▶ Inputs $X \in \mathbb{R}^{n \times d}$ and $y \in \mathbb{R}^n$
- ▶ Want $w \in \mathbb{R}^d$ to minimize $\|Xw - y\|_2$
- ▶ Equivalently minimize $(Xw - y)^\top (Xw - y)$ wrt w
- ▶ Objective simplifies to $w^\top X^\top Xw - 2w^\top X^\top y + \text{const.}$
- ▶ Setting derivative wrt w to 0, we get $\hat{w} = (X^\top X)^{-1} X^\top y$
- ▶ Linear least square fit
- ▶ What if $(X^\top X)^{-1}$ does not exist?
- ▶ Regularize the objective
- ▶ One popular option is to add $\frac{1}{2}\lambda w^\top w$
- ▶ Solution changes to $\tilde{w} = (X^\top X + \lambda \mathbb{I})^{-1} X^\top y$
- ▶ May instead add $\lambda \|w\|_1$
- ▶ Quadratic program, gives sparse w

Ordinal regression

- ▶ Items assigned *ratings* on a discrete r -point scale, e.g., items for sale at Amazon.COM
- ▶ The task is to regress instance $x \in \mathcal{X}$ to label $y \in \mathcal{Y}$ where \mathcal{Y} is typically small
- ▶ Bipartite ranking is a special case with $|\mathcal{Y}| = 2$ so we can write $\mathcal{Y} = \{-1, +1\}$

Ordinal regression is different from plain classification because

- ▶ Unlike in classification, where labels in \mathcal{Y} are *incomparable*, here they have a total order imposed on them. (In standard regression, $\mathcal{Y} = \mathbb{R}$.)
- ▶ The accuracy measures of practical interest here are different from those (0/1 error, recall, precision, F_1) used in classification.

Max-margin ordinal regression

- ▶ Apart from β , we will optimize over $r - 1$ thresholds

$$-\infty = b_0 \leq b_1 \leq b_2 \leq \dots \leq b_{r-2} \leq b_{r-1} \leq b_r = +\infty$$

- ▶ Let $j \in \{1, \dots, r\}$ index score levels, and the i th instance in the j level be denoted x_i^j

- ▶ We wish to pick β such that, for any x_i^j ,

$$b_{j-1} < \beta^\top x_i^j < b_j$$

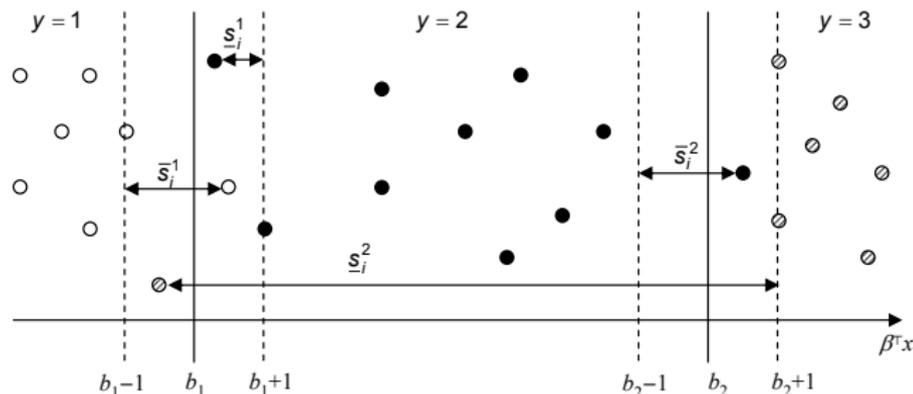
- ▶ Using the max-margin principle, we will insist that

$$b_{j-1} + 1 < \beta^\top x_i^j < b_j - 1$$

Max-margin ordinal regression (2)

- ▶ To avoid infeasibility, introduce lower slacks $\underline{s}_i^j \geq 0$ and upper slacks $\bar{s}_i^j \geq 0$, and relax the above inequalities to

$$b_{j-1} + 1 - \underline{s}_i^j \leq \beta^\top x_i^j \leq b_j - 1 + \bar{s}_i^j$$



- ▶ The objective to minimize is modified to

$$\min_{\beta, b, \underline{s} \geq \vec{0}, \bar{s} \geq \vec{0}} \frac{1}{2} \beta^\top \beta + B \sum_{j,i} (\underline{s}_i^j + \bar{s}_i^j),$$

Max-margin ordinal regression (3)

- ▶ Yet another quadratic program with linear inequalities
- ▶ Training time scales roughly as $n^{2.18-2.33}$ where n is the number of training instances
- ▶ More accurate than replacing ordinal regression with plain regression

Regression trees

- ▶ A regression tree maps $x_{qi} \in \mathbb{R}^d$ to \mathbb{R} , trying to learn a function
- ▶ \mathbb{R}^d chopped up using successive guillotine cuts into (possibly open) hyper-rectangles
- ▶ Tree node corresponds to hyper-rectangle and a cut leading to children
- ▶ Within each leaf node, function is approximated with a constant value

Encoding the ranking problem

- ▶ We will use a unary representation of the K target relevance scores $z_i \in \{0, 1, \dots, K - 1\}$, using a bit vector

$$y_{ik} = \begin{cases} 1, & z_i = k, \\ 0, & \text{otherwise} \end{cases}$$

- ▶ We will regard y_{ik} as some kind of $\Pr(Z_i = k|x_i)$
- ▶ For each $k = 0, 1, \dots, K - 1$, we will build a module which, given x_i , will output a real number $F_k(x_i)$
- ▶ We will then set empirical probabilities

$$p_{ik} = \frac{\exp(F_k(x_i))}{\sum_{\kappa} \exp(F_{\kappa}(x_i))},$$

as in logistic regression

- ▶ We will say that p s are functions of F , denoted $p(F)$
- ▶ The purpose of training is to bring p_{ik} close to y_{ik}

Loss function, training and testing

- ▶ A reasonable loss objective to minimize is

$$\text{loss}(y, F) = \sum_i \sum_k -y_{ik} \log p_{ik}. \quad (1)$$

- ▶ We will train the modules in stages numbered $m = 0, 1, \dots, M - 1$
- ▶ Specifically, in the m th stage we will train K regression trees T_{mk} , $k = 0, \dots, K - 1$
- ▶ When instance x is submitted to tree T_{mk} , we get output $f_k^{(m)}(x)$ from the tree
- ▶ For training instance $x = x_i$, shorthand $f_k^{(m)}(x_i)$ with $f_{i,k}^{(m)}$.

Greedy training

- ▶ After all trees are trained, we will regard the response for input x to be

$$F_k(x) = \sum_{0 \leq m < M} f_k^{(m)}(x);$$

equivalently, we can write the recurrence

$$\begin{aligned} F_k^{(0)}(x) &= 0 \\ F_k^{(m)}(x) &= F_k^{(m-1)}(x) + f_k^{(m)}(x). \end{aligned}$$

- ▶ As we increment m , we will greedily optimize f using the gradient of the loss (1)

Gradient boosting

- ▶ In other words, we will set

$$\begin{aligned} f_k^{(m)}(x) &= -\eta_{mk} \left[\frac{\partial \text{loss}(y, F)}{\partial F_{ik}} \right]_{F=F^{(m-1)}} \\ &= \eta_{mk} \left[\sum_i \sum_{\kappa} y_{i\kappa} \frac{\partial \log p_{i\kappa}}{\partial F_{ik}} \right]_{F=F^{(m-1)}} \end{aligned}$$

which can be verified to be $\eta_{mk}(y_{ik} - p_{ik})$ ▶ HW

- ▶ Here η_{mk} is a suitable **step size**

Step size

- ▶ Suppose regression tree T_{mk} has leaf nodes indexed by l
- ▶ Let “ $i : x_i \in T_{mkl}$ ” denote the set of training instances that belong to leaf l in tree T_{mk}
- ▶ Then we should use

$$\eta_{mkl} = \arg \min_{\eta} \sum_{i: x_i \in T_{mkl}} -y_{ik} \log p(F_k^{(m-1)}(x_i) + \eta)$$

- ▶ No closed form
- ▶ Friedman's approximation [2]:

$$\eta_{mkl} = \frac{K-1}{K} \frac{\sum_{i: x_i \in T_{mkl}} (y_{ik} - p_{ik})}{\sum_{i: x_i \in T_{mkl}} |y_{ik} - p_{ik}(F^{(m)})| (1 - |y_{ik} - p_{ik}(F^{(m)})|)} \quad (2)$$

MCRANK pseudocode

- 1: $F_{ik}^{(0)} \leftarrow 0$ for all instances i , class labels k
 - 2: **for** $m = 1, 2, \dots, M - 1$ **do**
 - 3: $p_{ik} \leftarrow \exp(F_{ik}^{(m-1)}) / \sum_{\kappa} \exp(F_{i\kappa}^{(m-1)})$ for all i, k
 - 4: **for** $k = 0, 1, \dots, K - 1$ **do**
 - 5: induce regression tree T_{mk} on $(x_i, y_{ik} - p_{ik})$ over all instances i
 - 6: **for** leaf l in T_{mk} **do**
 - 7: calculate step size η_{mkl} as in (2)
 - 8: update
- $$F_{ki}^{(m)} \leftarrow F_{ki}^{(m-1)} + \sum_{l \in T_{mk}} \eta_{mkl} \mathbb{I}[x_i \in T_{mkl}]$$

Ranking to satisfy preference pairs

- ▶ Suppose $x \in X$ are **instances** and $\phi : X \rightarrow \mathbb{R}^d$ a **feature vector generator**
- ▶ E.g., x may be a document and ϕ maps x to the “vector space model” with one axis for each word
- ▶ The **score** of instance x is $\beta^\top \phi(x)$ where $\beta \in \mathbb{R}^d$ is a **weight vector**
- ▶ For simplicity of notation assume x is already a feature vector and drop ϕ
- ▶ We wish to learn β from training data \prec : “ $i \prec j$ ” means the score of x_i should be less than the score of x_j , i.e.,

$$\beta^\top x_i \leq \beta^\top x_j$$

Soft constraints

- ▶ In practice, there may be no feasible β satisfying all preferences \prec
- ▶ For constraint $i \prec j$, introduce slack variable $s_{ij} \geq 0$

$$\beta^\top x_i \leq \beta^\top x_j + s_{ij}$$

- ▶ Charge a penalty for using $s_{ij} > 0$

$$\min_{s_{ij} \geq 0; \beta} \frac{1}{|\prec|} \sum_{i \prec j} s_{ij} \quad \text{subject to}$$

$$\beta^\top x_i \leq \beta^\top x_j + s_{ij} \quad \text{for all } i \prec j$$

A max-margin formulation

- ▶ Achieve “confident” separation of loser and winner:

$$\beta^\top x_i + 1 \leq \beta^\top x_j + s_{ij}$$

- ▶ $s_{ij} \geq 0$ and $s_{ij} \geq 1 - (\beta^\top x_j - \beta^\top x_i)$ together mean $s_{ij} \geq \max\{0, 1 - (\beta^\top x_j - \beta^\top x_i)\}$

- ▶ Because of $\sum_{i \prec j} s_{ij}$ term in objective, optimizer will pick $s_{ij} = \max\{0, 1 - (\beta^\top x_j - \beta^\top x_i)\}$

- ▶ If $\beta^\top x_i \geq \beta^\top x_j$, $s_{ij} \geq 1$

- ▶ I.e., $\sum_{i \prec j} s_{ij}$ is an upper bound on the number of violated training pair preferences

- ▶ Problem: Can achieve this by scaling β arbitrarily; must be prevented by penalizing $\|\beta\|$

$$\min_{s_{ij} \geq 0; \beta} \frac{1}{2} \beta^\top \beta + \frac{B}{|\prec|} \sum_{i \prec j} s_{ij} \quad \text{subject to}$$

$$\beta^\top x_i + 1 \leq \beta^\top x_j + s_{ij} \quad \text{for all } i \prec j$$

A max-margin formulation (2)

- ▶ B is a magic parameter that balances violations against model strength

Solving the optimization

- ▶ $\beta^\top x_i + 1 \leq \beta^\top x_j + s_{ij}$ and $s_{ij} \geq 0$ together mean $s_{ij} = \max\{0, \beta^\top x_i - \beta^\top x_j + 1\}$ (“hinge loss”)
- ▶ The optimization can be rewritten without using s_{ij}

$$\min_{\beta} \frac{1}{2} \beta^\top \beta + \frac{B}{|\mathcal{I}|} \sum_{i < j} \max\{0, \beta^\top x_i - \beta^\top x_j + 1\}$$

- ▶ $\max\{0, t\}$ can be approximated by a number of smooth functions
 - ▶ e^t – growth at $t > 0$ too severe
 - ▶ $\log(1 + e^t)$ – much better, asymptotes to $y = 0$ as $t \rightarrow -\infty$ and to $y = t$ as $t \rightarrow \infty$

Approximating with a smooth objective

- ▶ Simple unconstrained optimization, can be solved by Newton method

$$\min_{\beta \in \mathbb{R}^d} \frac{1}{2} \beta^\top \beta + \frac{B}{|\prec|} \sum_{i \prec j} \log(1 + \exp(\beta^\top x_i - \beta^\top x_j + 1))$$

- ▶ If $\beta^\top x_i - \beta^\top x_j + 1 \ll 0$, i.e., $\beta^\top x_i \ll \beta^\top x_j$, then pay little penalty
- ▶ If $\beta^\top x_i - \beta^\top x_j + 1 \gg 0$, i.e., $\beta^\top x_i \gg \beta^\top x_j$, then pay large penalty

Performance issues

- ▶ Common SVM implementations will take time almost quadratic in the number of training pairs
- ▶ Consider a TREC-style relevance judgment: for each query, we are given, say, 10 relevant and (implicitly) $1M - 10$ irrelevant documents
- ▶ Don't really need to train RankSVM with $10M$ $x_i \prec x_j$ pairs
- ▶ E.g., if $\beta^\top x_0 \leq \beta^\top x_1$ and $\beta^\top x_0 \leq \beta^\top x_2$, then $\beta^\top x_0 \leq \lambda\beta^\top x_1 + (1 - \lambda)\beta^\top x_2$ for $\lambda \in [0, 1]$
- ▶ Cannot, in general, say ahead of time which preferences will be redundant

A probabilistic interpretation of “ranking loss”

- ▶ Apart from $x_i \prec x_j$, trainer gives **target probability** \bar{p}_{ij} with which trained system should rank i worse than j
- ▶ The score of x_i is $f(x_i) \in \mathbb{R}$; $f(x_i)$ induces a ranking on $\{x_i\}$
- ▶ The **modeled posterior** p_{ij} is assumed to have a familiar log-linear form

$$p_{ij} = \frac{\exp(f(x_j) - f(x_i))}{1 + \exp(f(x_j) - f(x_i))}$$

- ▶ If $f(x_j) \gg f(x_i)$, $p_{ij} \rightarrow 1$; if $f(x_j) \ll f(x_i)$, $p_{ij} \rightarrow 0$
- ▶ Goal is to design f to minimize divergence between trainer-specified \bar{p} and modeled p , e.g.,

$$\ell(\bar{p}_{ij}, p_{ij}) = -\bar{p}_{ij} \log p_{ij} - (1 - \bar{p}_{ij}) \log(1 - p_{ij})$$

Consistency requirements on \bar{p}_{ij}

- ▶ Trainer cannot assign \bar{p}_{ij} arbitrarily
- ▶ \bar{p}_{ij} must be **consistent** with some ideal node-scoring function \bar{f} such that

$$\bar{p}_{ij} = \frac{\exp(\bar{f}(x_j) - \bar{f}(x_i))}{1 + \exp(\bar{f}(x_j) - \bar{f}(x_i))}$$

- ▶ Using above, can show that

$$\bar{p}_{ik} = \frac{\bar{p}_{ij}\bar{p}_{jk}}{1 + 2\bar{p}_{ij}\bar{p}_{jk} - \bar{p}_{ij} - \bar{p}_{jk}}$$

- ▶ Consider \bar{p}_{ik} if $\bar{p}_{ij} = \bar{p}_{kj} = p$, in particular $p = 0, .5, 1$
- ▶ Perfect uncertainty and perfect certainty propagate

Fitting f using gradient descent

- ▶ Model $f(x_i) = \beta^\top x_i$ for simplicity
- ▶ During training we are given ($i \prec j$ with) a target \bar{p}_{ij}
- ▶ We want to fit β so that

$$\bar{p}_{ij} = \frac{\exp(\beta^\top x_i - \beta^\top x_j)}{1 + \exp(\beta^\top x_i - \beta^\top x_j)}$$

- ▶ We can cast this as, say,

$$\min_{\beta} \sum_{i \prec j} \left(\bar{p}_{ij} - \frac{\exp(\beta^\top x_i - \beta^\top x_j)}{1 + \exp(\beta^\top x_i - \beta^\top x_j)} \right)^2$$

and use gradient descent

- ▶ Or we can use more complex forms of $f(x)$, like a neural network

RankBoost

- ▶ Given partial orders with preference strengths $\phi(i, j) \geq 0$: if positive, $i \succ j$, otherwise impartial
- ▶ Input pair distribution \mathcal{D} over $\mathcal{X} \times \mathcal{X}$
- ▶ **Weak learner** indexed by t gets input pairs as per a distribution \mathcal{D}_t and outputs a **weak ranking** $h_t : \mathcal{X} \rightarrow \mathbb{R}$
- ▶ Initialize $\mathcal{D}_1 = \mathcal{D}$
- ▶ For $t = 1, \dots, T$
 - ▶ Train t th weak learner using \mathcal{D}_t
 - ▶ Get weak ranking $h_t : \mathcal{X} \rightarrow \mathbb{R}$
 - ▶ Choose $\alpha_t \in \mathbb{R}$
 - ▶ Update

$$\mathcal{D}_{t+1}(x_i, x_j) \propto \mathcal{D}_t(x_i, x_j) \exp(\alpha_t(h_t(x_i) - h_t(x_j)))$$

while scaling by $Z_{t+1} = \sum_{i,j} \dots$

- ▶ Final scoring function $H(x) = \sum_{t=1}^T \alpha_t h_t(x)$

Some properties of RankBoost

- ▶ The **ranking loss** $R_{\mathcal{D}}(H)$ is defined as

$$\sum_{x_i, x_j} \mathcal{D}(x_i, x_j) \llbracket H(x_i) \leq H(x_j) \rrbracket = \Pr_{(x_i, x_j) \sim \mathcal{D}} \llbracket H(x_i) \leq H(x_j) \rrbracket$$

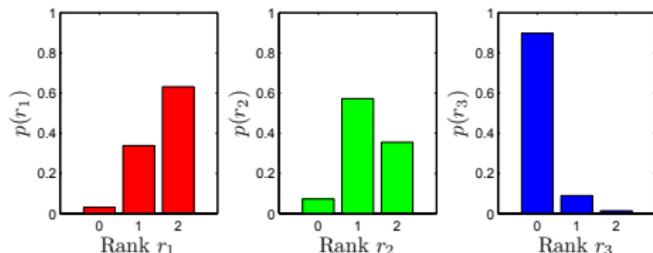
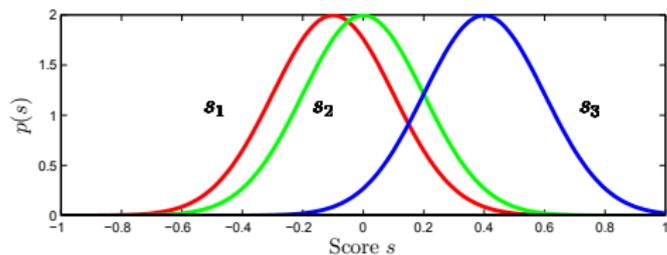
- ▶ $R_{\mathcal{D}}(H) \leq \prod_{t=1}^T Z_t$
- ▶ By suitably choosing α_t we can ensure $Z_t \leq 1$
- ▶ E.g., if $h : \mathcal{X} \rightarrow \{0, 1\}$, we can minimize Z_t analytically:
 - ▶ For $b \in \{-1, 0, +1\}$, define

$$W_b = \sum_{x_i, x_j} \mathcal{D}(x_i, x_j) \llbracket h(x_i) - h(x_j) = b \rrbracket$$

- ▶ Z_t is minimized when $\alpha = \frac{1}{2} \ln(W_{-1}/W_{+1})$ ▶ HW

Potential limitations of pairwise loss

- ▶ Loss is the number of pair violations
- ▶ Take an “ideal” ranking and swap docs at 45 and 56 — hardly anyone cares
- ▶ Swap docs at 2 and 12 — big difference! —How to model?
- ▶ Loss no longer additive over items or pairs
- ▶ Loss is a function of **permutation** induced by scoring model



A linear-time RankSVM approximation

- ▶ The primal optimization can be reformulated as

$$\min_{\beta, s \geq 0} \frac{1}{2} \beta^\top \beta + B s \quad \text{such that} \quad (\text{RankSVM2})$$

$$\forall \vec{c} \in \{0, 1\}^{\mathcal{K}} : \frac{1}{|\prec|} \beta^\top \sum_{u \prec v} c_{uv} (x_v - x_u) \geq \frac{1}{|\prec|} \sum_{u \prec v} c_{uv} - s$$

- ▶ Only one slack variable s , but $2^{|\mathcal{K}|}$ primal constraints and corresponding $2^{|\mathcal{K}|}$ dual variables
- ▶ (But if most primal constraints are redundant, most dual variables will be inactive, i.e., 0)
- ▶ Compare with

$$\min_{\beta, \{s_{uv} \geq 0 : u \prec v\}} \frac{1}{2} \beta^\top \beta + \frac{B}{|\prec|} \sum_{u \prec v} s_{uv} \quad (\text{RankSVM1})$$

$$\text{such that } \forall u \prec v : \beta^\top x_u + 1 \leq \beta^\top x_v + s_{uv}$$

Correctness

Any solution to (RankSVM2) corresponds to a solution to (RankSVM1), and vice versa

- ▶ Fix a β_0 in (RankSVM1)
- ▶ For optimality, must pick $s_{uv}^* = \max\{0, 1 + \beta_0^\top x_u - \beta_0^\top x_v\}$
- ▶ Fix the same β_0 for (RankSVM2)
- ▶ For optimality, must pick

$$s^* = \max_{\vec{c} \in \{0,1\}^{\prec}} \left\{ \frac{1}{|\prec|} \sum_{u \prec v} c_{uv} (1 + \beta_0^\top x_u - \beta_0^\top x_v) \right\}$$

- ▶ Pick \vec{c} element-wise: $c_{uv}^* = \llbracket 1 + \beta_0^\top x_u - \beta_0^\top x_v \geq 0 \rrbracket$
- ▶ In other words, if $u \prec v$ is not given adequate margin it's a violation, so set $c_{uv} = 1$
- ▶ Otherwise β_0 satisfies $u \prec v$ with a margin, so set $c_{uv} = 0$

Correctness (2)

- ▶ Can verify ▶ HW that objectives of (RankSVM1) and (RankSVM2) will be equal using $\beta_0, \{s_{uv}^*\}, s^*, \{c_{uv}^*\}$
- ▶ However, starting from $d + |\prec|$ variables and $2|\prec|$ constraints, we now have $d + 1$ variables but $2^{|\prec|} + 1$ constraints
- ▶ Next: how to avoid asserting all the constraints

Cutting plane method: General recipe

- ▶ Primal: $\min_x f(x)$ subject to $g(x) \leq \vec{0}$ (g is a vector-valued function)
- ▶ Dual:

$$\begin{aligned} & \max_{z,u} \quad z \\ & \text{subject to} \quad z \leq f(x) + u^\top g(x) \quad \forall x \\ & \quad \quad \quad u \geq 0 \end{aligned}$$

- ▶ “ $\forall x$ ” is generally infinite
- ▶ Let z_k, u_k be a solution
- ▶ Find $\min_x f(x) + u_k^\top g(x)$, let solution be x_k
- ▶ If $z_k \leq f(x_k) + u_k^\top g(x_k)$, terminate
- ▶ Otherwise add k th constraint $z \leq f(x_k) + u^\top g(x_k)$
- ▶ To approximate and terminate faster, continue only if $z_k > f(x_k) + u_k^\top g(x_k) + \epsilon$

Gradual dual variable inclusion

- ▶ Instead of all $\{0, 1\}^{|K|}$, start with $\mathcal{W} \subset \{0, 1\}^{|K|}$, typically $\mathcal{W} = \emptyset$
- ▶ Solve (RankSVM2) with \mathcal{W} instead of $\{0, 1\}^{|K|}$ to get the current β_0, s^*

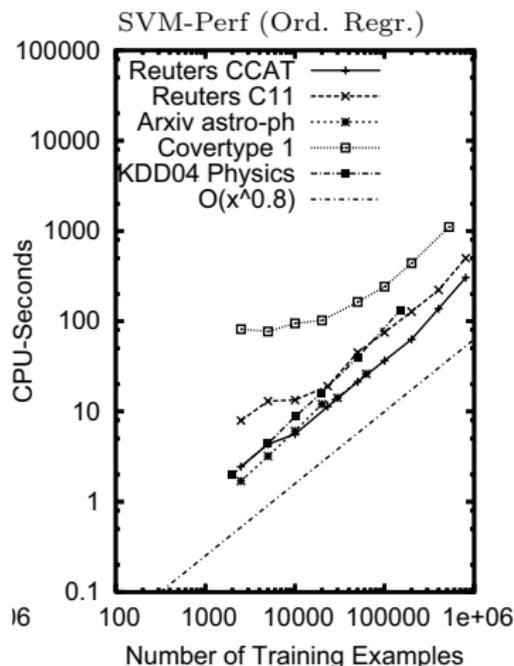
- ▶ Look for a violator c^* such that

$$\frac{1}{|\prec|} \beta_0^\top \sum_{u \prec v} c_{uv}^* (x_v - x_u) < \frac{1}{|\prec|} \sum_{u \prec v} c_{uv}^* - s^* - \epsilon$$

- ▶ If no such c^* found, exit with an objective that is at most the optimal objective plus ϵ
- ▶ Otherwise add c^* to \mathcal{W} and repeat
- ▶ For fixed (constant) ϵ, B and $\max \|x_v\|_2$, the number of inclusions into \mathcal{W} before no further c^* is found is **constant**
- ▶ Each loop above can be implemented in $O(n \log n)$ vector operations in \mathbb{R}^d where all $x_v \in \mathbb{R}^d$

Linear-time (RankSVM2) performance

- ▶ Almost linear scaling in practice too
- ▶ Dramatic improvement over (RankSVM1)
- ▶ (RankSVM1) scales roughly as $n^{3.4}$ (not shown)



General listwise training recipe

- ▶ For each query q , we know feature vectors x_q
- ▶ And ideal ranking, represented as y_q^*
- ▶ Define a **loss function** $\Delta(y_q^*, y) \geq 0$
- ▶ Define a **feature map** $\phi(x, y)$
- ▶ Solve the optimization

$$\min_{\xi \geq \vec{0}; w} w^\top w + B \sum_q \xi_q \quad \text{s.t.}$$

$$\forall q, \forall y \neq y_q^* : \quad w^\top \phi(x_q, y_q^*) - w^\top \phi(x_q, y) \geq \Delta(y_q^*, y) - \xi_q$$

using a cutting plane technique

- ▶ Key is to design a good ϕ and then design an algorithm for

$$\arg \max_y w^\top \phi(x_q, y) + \Delta(y_q^*, y)$$

- ▶ After model w is trained, for test queries, predict $\arg \max_y w^\top \phi(x_q, y)$

Listwise training for AUC

- ▶ AUC is directly related to the number of violated pair preferences
- ▶ g is a good (relevant) document; b is a bad (irrelevant) document
- ▶ Encode y and Δ_{AUC} as follows:

$$y_{gb} = \begin{cases} 1, & \text{if } g \text{ is before } b, \\ -1, & \text{if } g \text{ is after } b \end{cases}$$

$$\text{AUC}(y) = \frac{1}{n^+n^-} \sum_{g,b} \frac{1 + y_{gb}}{2},$$

$$\text{so } \Delta_{\text{AUC}}(y^*, y) = \frac{1}{n^+n^-} \sum_{g,b} \frac{1 - y_{gb}}{2}$$

- ▶ Here n^+ (n^-) is the number of good (bad) documents for the given query

Feature map for AUC

- ▶ A natural feature map is

$$\phi(x, y) = \frac{1}{n^+n^-} \sum_{g,b} y_{gb}(x_g - x_b)$$

- ▶ Note that at test time, for query q , it suffices to sort documents in decreasing order of $w^\top x_{qi}$ to maximize $w^\top \phi(x_q, y)$ over y [▶ HW](#)

Argmax for cutting plane AUC training

- ▶ Goal of argmax subroutine at training time is to

$$\arg \max_y w^\top \phi(x, y) + \Delta_{\text{AUC}}(y^*, y)$$

$$= \arg \max_y \sum_{g,b} y_{gb} (w^\top x_g - w^\top x_b) + \sum_{g,b} \frac{1 - y_{gb}}{2}$$

$$= \arg \max_y \sum_{g,b} y_{gb} \left(w^\top x_g - w^\top x_b - \frac{1}{2} \right)$$

- ▶ Can optimize each y_{gb} separately:

$$y_{g,b} = \begin{cases} 1, & \text{if } w^\top x_g - w^\top x_b - \frac{1}{2} > 0, \\ -1, & \text{otherwise} \end{cases}$$

$$\text{i.e., } y_{g,b} = \text{sign}(w^\top x_g - w^\top x_b - \frac{1}{2})$$

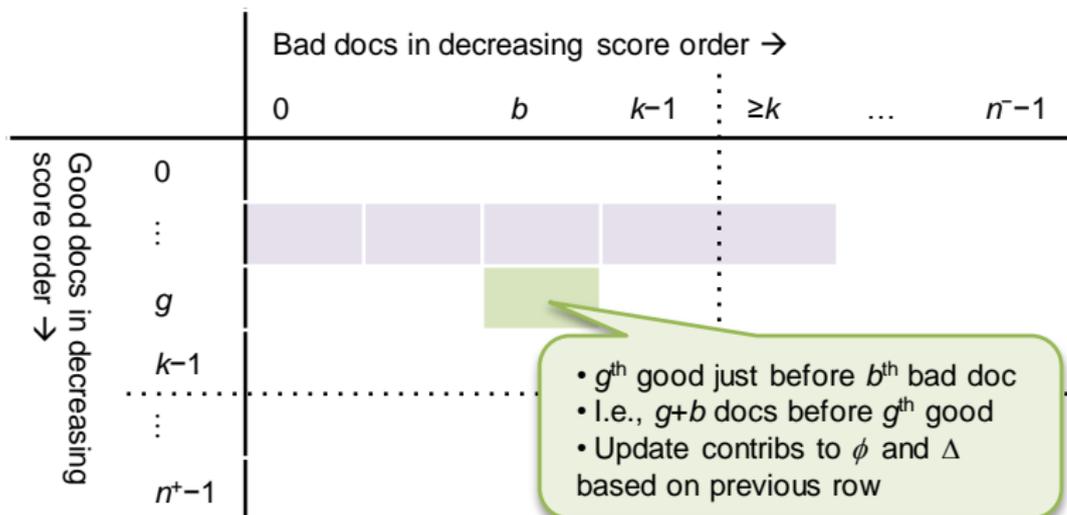
- ▶ Can find an implicit representation of the best y in $O(n \log n)$ time, where $n = n^+ + n^-$

MRR is simpler than AUC

- ▶ Recall MRR is reciprocal of rank of first good doc
- ▶ $1, 1/2, 1/3, 1/k, 0$ only possible values of Δ_{mrr}
- ▶ For a given value of MRR, say $1/r$, first good doc must be at rank r
- ▶ For a given configuration $\underbrace{b, \dots, b}_{r-1}, \underbrace{g}_r, \underbrace{?, ?, \dots}_{\text{rest}}$ need to fill good and bad slots to maximize $w^\top \phi$
- ▶ Bad docs b at $1, \dots, r - 1$ with largest $w^\top x_b$
- ▶ Good doc g with smallest $w^\top x_g$ at position r
- ▶ Add up Δ and $w^\top \phi$ for each possible Δ and take maximum
- ▶ (MRR = 0 handled separately)

Generic template to $\max w^T \phi + \Delta$

- ▶ Assume two levels of relevance $z_{qi} \in \{0, 1\}$
- ▶ Δ unchanged if two good (or bad) docs swapped
- ∴ There exists an optimal y that can be formed by merging good and bad in decreasing score order



Is training on “true” Δ always best?

	OHSUMED			TD2003			TD2004			TREC2000			TREC2001		
	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP
MRR	0.80	0.62	0.57	0.63	0.41	0.33	0.63	0.44	0.38	0.67	0.41	0.24	0.64	0.43	0.23
NDCG*	0.82	0.64	0.58	0.60	0.40	0.31	0.61	0.49	0.40	0.69	0.46	0.27	0.62	0.44	0.26
DORM	0.81	0.64	0.58	0.59	0.36	0.29	0.47	0.34	0.30	0.66	0.41	0.24	0.62	0.44	0.25
MAP	0.81	0.64	0.59	0.62	0.41	0.31	0.61	0.50	0.41	0.70	0.47	0.28	0.64	0.45	0.27

MRR: Max mean reciprocal rank of #1 good doc

NDCG: Maximize NDCG

DORM: Ditto; Hungarian docs-to-ranks assignment
(Chapelle+ 2007)

MAP: Maximize mean average precision (Yue+ 2007)

Hedging our (loss function) bets

What use is a library of perfect loss functions, if we have no idea which Δ users want?

- ▶ MRR suited for navigational queries
- ▶ MAP, NDCG suited for researching a topic
- ▶ “I’m feeling lucky” = precision at rank 1

Train for multiple Δ s: SVMCOMBO

- ▶ Can a single w do well for many Δ s?

$$\arg \min_{w; \xi \geq \bar{0}} w^\top w + \sum_{\ell} C_{\ell} \frac{1}{|Q|} \sum_q \xi_q^{\ell} \quad \text{s.t.}$$

$$\forall \ell, q, \forall y \neq y_q^* : w^\top \delta \phi_q(y) \geq \Delta_{\ell}(y_q^*, y) - \xi_q^{\ell}$$

ℓ ranges over loss types NDCG, MRR, MAP, ...

- ▶ Empirical risk (training error)

$$R(w, \Delta) = \frac{1}{|Q|} \sum_q \Delta(y_q^*, f_w(x_q))$$

- ▶ Can show

$$\sum_{\ell} C_{\ell} \frac{1}{|Q|} \sum_q \xi_q^{\ell} \geq \sum_{\ell} R(w, \Delta_{\ell}) \geq R(w, \max_{\ell} \Delta_{\ell})$$

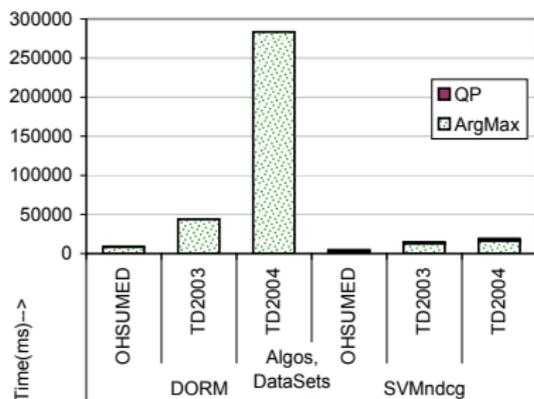
- ▶ I.e. learning minimizes upper bound on **worst** loss

Test accuracy vs. training loss function

	OHSUMED			TD2003			TD2004			TREC2000			TREC2001		
	MRR10	NDCG10	MAP												
AUC	.799	.635	.582	.510	.349	.256	.639	.501	.420	.607	.448	.267	.632	.441	.264
MAP	.808	.642	.586	.618	.411	.314	.614	.496	.412	.696	.469	.277	.636	.450	.272
NDCG	.790	.636	.581	.587	.372	.302	.631	.457	.374	.517	.323	.175	.608	.356	.171
NDCG-NC	.818	.640	.582	.595	.404	.306	.611	.486	.404	.685	.455	.265	.624	.443	.264
MRR	.795	.623	.570	.628	.405	.330	.629	.441	.383	.670	.410	.244	.643	.426	.230
COMBO	.813	.635	.578	.667	.434	.345	.647	.458	.384	.695	.465	.277	.647	.449	.272
DORM	.807	.637	.583	.587	.362	.290	.474	.340	.297	.662	.413	.243	.621	.435	.250
McRank	.701	.565	.527	.650	.403	.232	.588	.529	.453						

- ▶ Row: training Δ s, column: test criterion
- ▶ SVMCOMBO, SVMMAP good across the board
- ▶ Did not tune C_ℓ yet
- ▶ Listwise Δ s better than elementwise or pairwise

SVMNDCG speed and scalability



SVMCOMBO is

- ▶ 15× faster than DORM
- ▶ 100× faster than MCRANK

while being more accurate in over 75% of data sets

Dataset	MCRANK tree		MCRANK boost	MCRANK total		SVMNDCG	SVMRR
OHSUMED	1034	67	1102	4.8	30.6		
TD2003	9730	383	10113	14.9	125		
TD2004	8760	548	9308	19.1	148		

ListNet: Notation

- ▶ Queries $Q = \{q^{(1)}, \dots, q^{(m)}\}$
- ▶ i th query associated with documents
 $d^{(i)} = (d_1^{(i)}, \dots, d_j^{(i)}, \dots, d_{n^{(i)}}^{(i)})$
- ▶ j th candidate document for i th query
- ▶ Ground truth/gold relevance judgments
 $y^{(i)} = (y_1^{(i)}, \dots, y_j^{(i)}, \dots, y_{n^{(i)}}^{(i)})$
- ▶ Feature vector $x_j^{(i)} = \Psi(q^{(i)}, d_j^{(i)})$ for one doc;
 $x^{(i)} = (x_1^{(i)}, \dots, x_j^{(i)}, \dots, x_{n^{(i)}}^{(i)})$ over all candidates for query i
- ▶ Score $f(x_j^{(i)})$ for one doc; $z^{(i)} = (f(x_1^{(i)}), \dots, f(x_{n^{(i)}}^{(i)}))$ over all candidates
- ▶ Ranking loss is $\sum_{i=1}^m L(y^{(i)}, z^{(i)})$ where L is a listwise loss function

ListNet: Intuition

- ▶ To be listwise, $L(y, z)$ must get info from sorted order over y and z
- ▶ But sorting makes L not continuous or differentiable everywhere wrt f (and its internal model weights)
- ▶ Instead of depending on two sorted orders by y and by z , express loss as a sum over many/all orders
- ▶ Over every possible permutation π :
 - ▶ Find (smooth) compatibility between π and y
 - ▶ Ditto between π and z
 - ▶ Combine these compatibilities into a smooth loss
... while aggregating over all possible permutations
- ▶ If this is too expensive, aggregate over only top- k positions, and the sets of permutations consistent with top- k assignments

ListNet: Permutation probability

- ▶ Consider a set of n candidate docs with scores (s_1, \dots, s_n)
- ▶ π is a permutation over these; its probability is modeled as

$$\Pr_s(\pi) = \prod_{j=1}^n \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^n \phi(s_{\pi(k)})}$$

for a suitable transformation ϕ of score s

- ▶ Item score divided by sum of scores of suffix
- ▶ **HW** For any score s , $\Pr_s(\pi) > 0$ for all permutations π and $\sum_{\pi} \Pr_s(\pi) = 1$
- ▶ **HW** If $s_1 > \dots > s_n$, then the permutation with largest probability is $(1, 2, \dots, n)$ and the permutation with lowest probability is $(n, n-1, \dots, 2, 1)$

ListNet: Top- k probability

- ▶ Given docs j_1, \dots, j_k , top- k subgroup $\mathcal{G}_k(j_1, \dots, j_k)$ are all permutations in which these are in the top- k positions *in that order*
- ▶ \mathcal{G}_k is the collection of all top- k subgroups; there are $\frac{n!}{(n-k)!}$ subgroups in the collection
- ▶ Top- k probability of docs (j_1, \dots, j_k) is the probability of $\mathcal{G}_k(j_1, \dots, j_k)$, i.e., the total probability that a random π is from $\mathcal{G}_k(j_1, \dots, j_k)$:

$$\Pr_s(\mathcal{G}_k(j_1, \dots, j_k)) = \sum_{\pi \in \mathcal{G}_k(j_1, \dots, j_k)} \Pr_s(\pi)$$

▶ HW

$$\Pr_s(\mathcal{G}_k(j_1, \dots, j_k)) = \prod_{t=1}^k \frac{\phi(s_{j_t})}{\sum_{\ell=t}^n \phi(s_{j_\ell})}$$

▶ HW

$$\sum_{j_1, \dots, j_k} \Pr_s(\mathcal{G}_k(j_1, \dots, j_k)) = 1$$

ListNet: Top- k probability (2)

- ▶ Given j_u, j_v with $u \neq v, s_{j_u} > s_{j_v}$,

$$\begin{aligned} & \Pr_s(\mathcal{G}_k(j_1, \dots, j_u, \dots, j_v, \dots, j_k)) \\ & > \Pr_s(\mathcal{G}_k(j_1, \dots, j_v, \dots, j_u, \dots, j_k)) \end{aligned}$$

ListNet: Loss and training

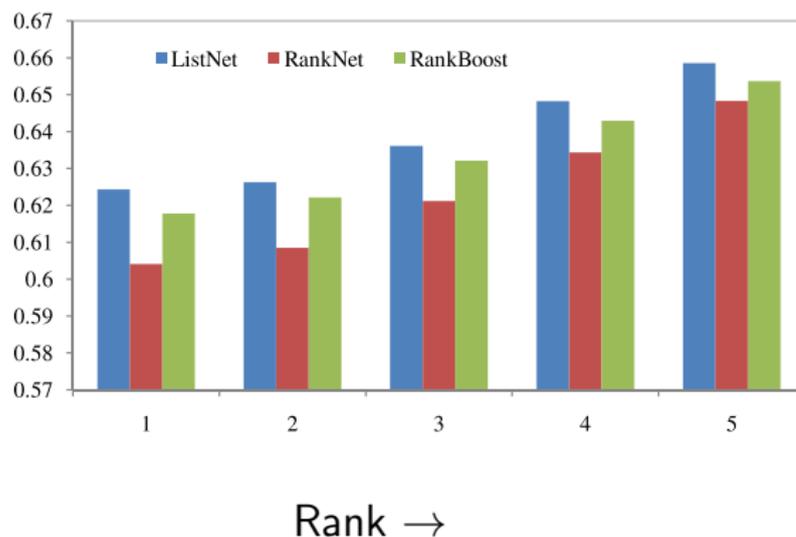
- ▶ Given two lists of scores: gold y and system z
- ▶ Let $g \in \mathcal{G}_k$
- ▶ y induces a distribution over these g s, given by $\Pr_y(g)$
- ▶ Likewise z induces $\Pr_z(g)$
- ▶ Loss is the cross entropy between these:

$$L(y, z) = - \sum_{g \in \mathcal{G}} \Pr_y(g) \log \Pr_z(g)$$

- ▶ y is fixed, z is returned by scoring function f
- ▶ f is parameterized by model weights w and can be differentiated wrt w
- ▶ Then loss is differentiable wrt w
- ▶ Gradient descent, nonconvex

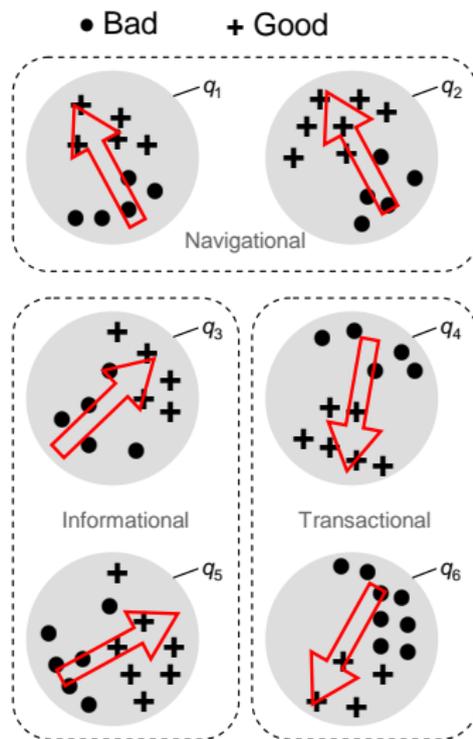
ListNet: Performance highlights

Algorithm →	ListNet	RankBoost	RankSVM
TREC	0.216	0.174	0.193
OHSUMED	0.305	0.297	0.297



Local learning

- ▶ Queries $q \in Q$, good, bad documents
 $D_q = D_q^+ \cup D_q^-$
- ▶ Query + document \rightarrow feature vector x_{qi}
- ▶ D_q interpreted as a **point cloud**
- ▶ Usually, learn a scoring model w
- ▶ Sort by decreasing score $= w^\top x_{qi}$
- ▶ Diverse queries, **navigational**, **informational**, **transactional**
- ▶ Different features of x_{qi} more important for different query clusters
- \therefore Same w not appropriate for all q
- ▶ **How to learn local models customized to query clusters?**
- ▶ Avoid error-prone query classification



Recent approaches

Lazy projection [9]:

- ▶ Compute PCA of test cloud
- ▶ Project all training clouds to principal directions
- ▶ Train model and apply to test cloud to rank
- ⊖ Lazy learning, impractical test-time computation

kNN query clustering [10]:

- ▶ Represent each training query q as a feature vector
- ▶ Train separate model for each q using D_q and clouds of nearby queries
- ⊖ Training is expensive, as many models as queries
- ▶ Given test query use model of nearest training query
- ⊕ Reasonably fast testing

Point cloud similarity

- ▶ How similar are queries q and q' ?
- ▶ How similar are point clouds D_q and $D_{q'}$?
- ▶ Order of presenting points in $D_q, D_{q'}$ irrelevant
- ▶ Shifting and scaling clouds should not matter
- ▶ Rotation **does** matter
- ▶ Natural idea
 - ▶ Assume each cloud generated from parametric distribution
 - ▶ Estimate parameters (μ, Σ , skew) for each distribution
 - ▶ Compare distributions using parameters
- ▶ Comparison methods
 - ▶ Compare parameters informally (dot product etc.)
 - ▶ Compute **divergence**: KL, Jensen-Shannon
 - ▶ Compute **kernel** between distributions: Bhattacharyya, Wolf-Shashua
- ⊖ Not shift- or scale- invariant
- ⊖ Can be expensive to compute

A different point-cloud similarity proposal

- ▶ Let $U = (u_1, \dots, u_P)$ be the sequence of P principal components of points in cloud D
- ▶ (u_1, \dots, u_P) are in order of decreasing eigen value
- ▶ Start by defining similarity between clouds D and D' as

$$\text{sim}(D_q, D_{q'}) = \frac{1}{P} \sum_{p=1}^P u_p^\top u'_p$$

- ▶ If u is a principal component of D , then so is $-u$
- ⊖ If the p th p.c. of D is u and the p th p.c. of D' is $-u$, $u_p^\top u'_p = -1$, **detracts** from similarity
- ▶ Fix: take absolute values:

$$\text{sim}(D_q, D_{q'}) = \frac{1}{P} \sum_{p=1}^P |u_p^\top u'_p|$$

Properties

- ▶ Translation Invariance

$$\text{sim}(D, D') = \text{sim}(D, D' + \eta),$$

- ▶ Scale Invariance

$$\text{sim}(D, D') = \text{sim}(D, \alpha D') \quad \forall \alpha \neq 0$$

- ▶ PCA on all query cloud takes

$$O\left(\sum_q d^2 n_q + \sum_q d^3\right) = O(d^2 \sum_q n_q + d^3 |Q|) \text{ time where } n_q = |D_q|$$

- ▶ Time for all pairs cloud comparisons is $O(d|Q|^2)$

Generic clustered training

- 1: **Input:** training queries Q , number of clusters C
- 2: **Output:** C clusters and their corresponding models
- 3: **for** each $q \in Q$ **do**
- 4: find a representation of q
- 5: cluster Q into C clusters based on the query representation and a similarity measure
- 6: **for** each cluster c **do**
- 7: train a model using queries Q_c belonging to cluster c
- 8: save model w_c obtained by training on cluster c

For our proposed representation and similarity,

- ▶ Complete link better than single link clustering
- ▶ Our $\text{sim}(D, D')$ does not readily let us summarize clusters as aggregated points
- ▶ Therefore use agglomerative, not k means

Generic clustered testing

- 1: **Input:** documents D_t for test query t , C models with training clusters
- 2: **Output:** ranked list of documents in D_t
- 3: compute proposed representation of query t
- 4: find cluster $c^* = \arg \max_c \max_{q \in Q_c} \text{sim}(D_t, D_q)$, where $\text{sim}(D_t, D_q)$ gives the similarity between t and q
- 5: use w_c to rank D_t

How to find quickly with large Q ... later

Experiments

- ▶ First: does LOCALRANK improve accuracy?
- ▶ Data sets: LETOR, Yandex
- ▶ Baseline algos: SVM MAP [11], RANKBOOST [6], KNN [10], D&K [9]

Clustering helps

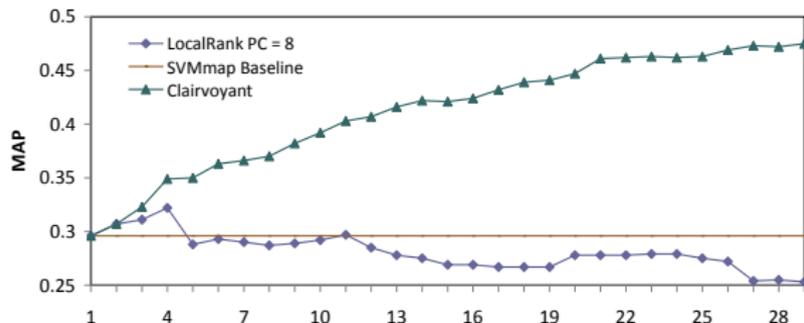


Figure: TD2003 test MAP vs. C , the number of clusters.

- ▶ $C = 1 \implies$ our algorithm same as baseline
- ▶ Peak accuracy occurs at $C > 1$, \therefore clustering is clearly helpful
- ▶ As C increases further, accuracy decreases
- ▶ Clusters become too small to build reliable models
- ▶ “Clairvoyant” = post-hoc best cluster, unactionable upper bound

Effect of C on NDCG@1, NDCG@5

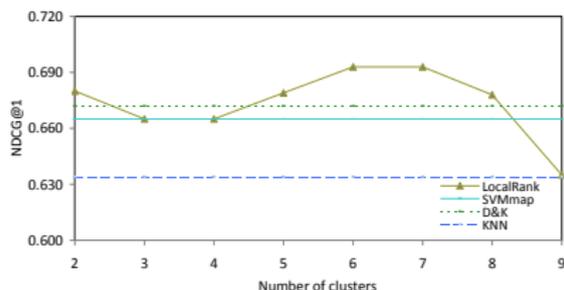
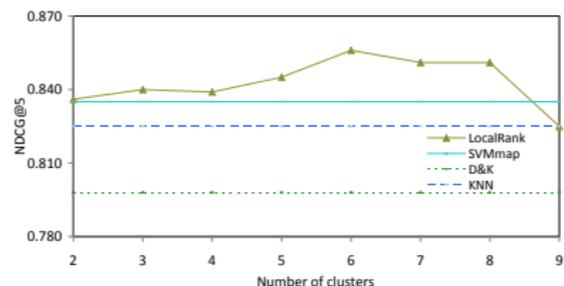


Figure: NDCG@1 vs. C for HP2004.



- ▶ Similar to MAP, best at $C \approx 5 \dots 7$
- ▶ Best C corresponds well to established ideas of query diversity
- ▶ Accuracy better than best baseline

Simple cloud aggregate based similarity

	MAP			NDCG@1			NDCG@5		
	TD2003	HP2003	HP2004	TD2003	HP2003	HP2004	TD2003	HP2003	HP2004
Mean	0.275	0.785	0.765	0.407	0.773	0.681	0.327	0.828	0.855
Variance	0.294	0.777	0.750	0.344	0.751	0.680	0.370	0.823	0.839
Skewness	0.301	0.777	0.760	0.424	0.758	0.694	0.357	0.824	0.827
LocalRank	0.333	0.798	0.769	0.447	0.787	0.707	0.395	0.843	0.859

Figure: PCA-based cloud similarity better than moment-based notions of cloud similarity

Divergence and kernels

	MAP			NDCG@1			NDCG@5		
	TD2003	HP2003	HP2004	TD2003	HP2003	HP2004	TD2003	HP2003	HP2004
KL-Divergence	0.313	0.782	0.751	0.427	0.765	0.680	0.371	0.837	0.834
Bhattacharyya	0.265	0.784	0.741	0.367	0.765	0.678	0.331	0.835	0.837
LocalRank	0.333	0.798	0.769	0.447	0.787	0.707	0.395	0.843	0.859

Figure: PCA-based cloud similarity better than symmetric KL

Overall accuracy comparison

		TD2003	TD2004	HP2003	HP2004	NP2003	NP2004	OHSUMED	YANDEX
NDCG@1	SVMmap	0.364	0.493	0.765	0.665	0.591	0.573	0.676	0.664
	Rankboost	0.360	0.453	0.714	0.653	0.636	0.530	0.617	0.576
	D&K	0.424	0.480	0.699	0.672	0.636	0.538	0.588	x
	KNN	0.313	0.324	0.762	0.578	0.584	0.499	0.638	0.661
	LocalRank	0.404	0.493	0.765	0.693	0.623	0.530	0.647	0.663
NDCG@5	SVMmap	0.368	0.363	0.829	0.835	0.801	0.830	0.621	0.763
	Rankboost	0.325	0.349	0.854	0.821	0.816	0.768	0.597	0.709
	D&K	0.366	0.378	0.848	0.798	0.836	0.805	0.577	x
	KNN	0.369	0.320	0.817	0.806	0.807	0.770	0.616	0.767
	LocalRank	0.383	0.348	0.837	0.856	0.791	0.828	0.623	0.761
NDCG@10	SVMmap	0.385	0.343	0.840	0.845	0.821	0.847	0.601	0.824
	Rankboost	0.347	0.340	0.868	0.845	0.836	0.806	0.582	0.781
	D&K	0.367	0.350	0.862	0.826	0.860	0.823	0.572	x
	KNN	0.322	0.324	0.831	0.829	0.817	0.836	0.580	0.826
	LocalRank	0.392	0.336	0.850	0.868	0.815	0.847	0.611	0.823
MAP	SVMmap	0.269	0.259	0.781	0.746	0.707	0.709	0.563	x
	Rankboost	0.277	0.263	0.782	0.739	0.740	0.664	0.545	x
	D&K	0.298	0.265	0.770	0.742	0.749	0.678	0.529	x
	KNN	0.232	0.242	0.762	0.714	0.702	0.668	0.547	x
	LocalRank	0.327	0.251	0.789	0.759	0.717	0.687	0.558	x

Figure: Summary of MAP and NDCG accuracies for all algorithms and various datasets

Comments on accuracy comparison

LOCALRANK ...

- ▶ Comparable to (± 0.001 or better) SVM_{MAP} 65% of the time
- ▶ Better than RANKBOOST 74% of the time
- ▶ Better than KNN 87% of the time
- ▶ Better than D&K 61% of the time
- ▶ (D&K has impractical test time)
- ▶ Overall best 42% of the time

Diversity in ranking

- ▶ Users do not sufficiently express information need in telegraphically short queries
- ▶ Queries become ambiguous, e.g., person name, common names as products (jaguar, apple)
- ▶ Diversify top- k responses to minimize abandonment risk
- ▶ Recall the **cluster hypothesis in IR**: If docs i and j are very similar to each other, and i is very relevant to the query, then so is j
- ▶ Even if that were the case, it does not mean the user would be interested in inspecting i **after** inspecting j
- ▶ A matter of **marginal utility**
- ▶ Diametrically opposite pov from pseudo relevance feedback (PRF) which can be solved by graph Laplacian technique

Max marginal relevance (MMR)

- ▶ Extreme lack of confidence: Report top k , assume they are all unwanted, choose $(k + 1)$ st doc
- ▶ $(k + 1)$ st doc should still be similar to query, but dissimilar to all preceding docs
- ▶ Greedily choose

$$\arg \max_{d \notin S} \lambda \text{sim}_1(d, q) - (1 - \lambda) \max_{d' \in S} \text{sim}_2(d, d')$$

as next doc

- ▶ For suitably designed similarity functions $\text{sim}_1, \text{sim}_2$ and tuned parameter λ

Graph cuts

- ▶ Represent docs as nodes V in a graph
- ▶ Edge weights w_{ij} represent similarity
- ▶ Find subset $S \subset V$ to maximize $\sum_{i \in S, j \in V \setminus S} w_{ij}$
- ▶ While minimizing the redundancy or self-similarity $\sum_{i, j \in S, i \neq j} w_{ij}$
- ▶ Overall objective is one minus the other, like MMR:

$$\arg \max_{S \subset V} \sum_{i \in S, j \in V \setminus S} w_{ij} - \lambda \sum_{i, j \in S, i \neq j} w_{ij}$$

- ▶ This is a **submodular** function (diminishing payoffs):
 $A \subseteq B \implies f(A \cup v) - f(A) \geq f(B \cup v) - f(B)$
- ▶ Can be approximated by greedy algorithms

GRASSHOPPER

- ▶ Items are nodes in a graph
- ▶ Edge weights represent similarity
- ▶ If TFIDF cosine is above a threshold then 1, else 0
- ▶ From weights to conductance as usual
- ▶ Node u_1 with largest PageRank (uniform teleport) is #1
- ▶ Now make u_1 a sink
- ▶ Walk is no longer irreducible; PageRanks of all other nodes are zero
- ▶ Absorbing Markov chain
- ▶ Measure expected number of visits before ending up in sink
- ▶ Let S be the current set of sinks
- ▶ Transition probability matrix looks like $\begin{bmatrix} \mathbb{I}_S & \mathbf{0} \\ R & Q \end{bmatrix}$

GRASSHOPPER (2)

- ▶ Consider the matrix $N = (\mathbb{I} - Q)^{-1}$
- ▶ If we start at i , the expected number of visits to j before absorption is N_{ij} ▶ HW
- ▶ u_2 is the node with the largest expected number of visits
- ▶ Now make both u_1 and u_2 sink nodes
- ▶ ... and repeat
- ▶ Easy to implement, but involves matrix inversions
- ▶ Semantics unclear compared to PageRank

DIVRANK

- ▶ Random walk with *time-variant* transition probabilities $C_T(j|i)$ at time T :

$$C_T(j|i) = (1 - \lambda)r(j) + \lambda \frac{C_0(j|i)N_T(j)}{\sum_k C_0(k|i)N_T(k)}$$

where $N_T(j)$ is the (random) number of times node j has been visited up to time T and r is a multinomial teleport distribution.

- ▶ This translates into

$$p_{T+1}(j) = (1 - \lambda)r(j) + \lambda \sum_i \frac{C_0(j|i)N_T(j)}{\sum_k C_0(k|i)N_T(k)} p_T(i)$$

- ▶ Unfortunately $N_T(i)$ must be approximated with point estimates to keep the computation practical
- ▶ Claimed to implement a “rich gets richer” effect as in preferential attachment

DIVRANK (2)

- ▶ Through random choice and/or asymmetry in the graph neighborhood, one node will emerge the “winner” in each tightly connected subgraph
- ▶ Unclear why “rich gets richer” is relevant here

Associative vs. dissociative

- ▶ Conflict between a **dissociative** goal (diversity) and **associative** graph representations
- ▶ Could instead model diversity as a graphical model with dissociative edge potentials
 - ▶ Node label is not “relevant” vs. “irrelevant” but “show” vs. “do not show”
 - ▶ Pages i, j similar and both good for query means avoid showing both
- ▶ Unfortunately, dissociative graphical models are notoriously intractable (max cut etc.)
- ▶ Is there a natural dissociative model for diversity that leverages off associative graphs?

Random vs. search-driven surfer

- ▶ PageRank is explained/motivated/rationalized using the **random surfer** model
- ▶ Today, most Web surfers are guided by search engines, not (just) the other way around
- ▶ Surfer asks a query, engine responds with 10 links, surfer explores out, perhaps returns back to engine response and starts afresh
- ▶ May use vocabulary from pages to ask new/modified queries and thus access more related pages

Learning to rank review

- ▶ In pre-Web IR, features were few in number and hand-crafted ranking was common
- ▶ Web documents exploded the feature space and its complexity
 - ▶ Query match with title, headings, anchor text, body, ...
 - ▶ PageRank, topic-specific PageRank, spamminess, ...
 - ▶ Click statistics collected from past (query, page) data
- ▶ Even more complex features from knowledge graphs, personalization, ...
- ▶ Hand-crafted ranking no longer practical
- ▶ Query q , document d together gives feature vector x
- ▶ Multiple feature vectors $\{x_i\}$, must (score and) rank them
- ▶ Rapidly decreasing user attention as they go down ranked lists

Learning to rank review (2)

- ▶ Three major loss paradigms: itemwise, pairwise, listwise
 - ▶ The first two losses give simpler decomposable objectives
 - ▶ Listwise loss sensitive to real user attention, but harder to optimize
- ▶ For itemwise loss, prediction output is a real relevance score
 - ▶ Can use complicated decision surfaces/functions, such as nonlinear SVOR and boosted regression trees
- ▶ The pairwise loss case reduces to learning w to always score $w \cdot (x_{\text{good}} - x_{\text{bad}}) > 0$ (or 1, as margin)
- ▶ This is RANKSVM, still widely used, works quite well
- ▶ Can still introduce nonlinear scoring function, if willing to tolerate nonconvex optimization
 - ▶ Instead of $w \cdot x$, design a smooth $f(x)$
 - ▶ Define $\text{Pr}(i \prec j) = \sigma(f(x_j) - f(x_i))$
 - ▶ Fit f with cross-entropy and gradient descent

Learning to rank review (3)

- ▶ None of the above capture that the loss because of swapping #1 and #12 is much more than that of swapping #12 and #13
- ▶ Main technical hurdle is that a single $f(x_i)$ (or even a pair) gives no clue of the rank of doc # i
- ▶ Even given all $f(x_i)$ together, rank of # i is not a continuous or differentiable quantity in the model parameters inside f

Learning to rank review (4)

- ▶ Two major approaches
 - ▶ Structured max-margin learning (design feature map and loss-augmented inference routine for preferred loss)
 - ▶ Turn deterministic scores into a probability of a permutation, contrast (cross-entropy) with ideal permutation
- ▶ The structured max-margin approach:
 - ▶ Represent partial or total order over docs as \mathbf{y}
 - ▶ And all feature vectors collectively as \mathbf{x}
 - ▶ Design feature map $\phi(\mathbf{x}, \mathbf{y})$ and loss $\Delta(\mathbf{y})$ (wrt ideal ranking)
 - ▶ Write routine to solve $\arg \max_{\hat{\mathbf{y}}} w \cdot \phi(\mathbf{x}, \hat{\mathbf{y}}) + \Delta(\hat{\mathbf{y}})$
 - ▶ Demonstrated for AUC, MRR, NDCG and (sort of) MAP
- ▶ Remaining topics in ranking are best covered after introducing graphs in search

References

- [1] W. Chu and S. Keerthi, “New approaches to support vector ordinal regression,” in *ICML*, 2005, pp. 145–152. <http://www.gatsby.ucl.ac.uk/~chuwei/paper/icmlsvor.pdf>
- [2] J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. <http://www-stat.stanford.edu/~jhf/ftp/trebst.pdf>
- [3] T. Joachims, “Optimizing search engines using clickthrough data,” in *SIGKDD Conference*. ACM, 2002, pp. 133–142. http://www.cs.cornell.edu/People/tj/publications/joachims_02c.pdf

References (2)

- [4] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *ICML*, 2005. http://research.microsoft.com/~cburges/papers/ICML_ranking.pdf
- [5] W. W. Cohen, R. E. Shapire, and Y. Singer, "Learning to order things," *JAIR*, vol. 10, pp. 243–270, 1999. <http://www.cs.washington.edu/research/jair/volume10/cohen99a.ps>
- [6] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *Journal of Machine Learning Research*, vol. 4, pp. 933–969, 2003. <http://jmlr.csail.mit.edu/papers/volume4/freund03a/freund03a.pdf>

References (3)

- [7] T. Joachims, “Training linear SVMs in linear time,” in *SIGKDD Conference*, 2006, pp. 217–226.
http://www.cs.cornell.edu/people/tj/publications/joachims_06a.pdf
- [8] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, “Large margin methods for structured and interdependent output variables,” *JMLR*, vol. 6, no. Sep., pp. 1453–1484, 2005. <http://ttic.uchicago.edu/~altun/pubs/TsoJoaHofAlt-JMLR.pdf>
- [9] K. Duh and K. Kirchhoff, “Learning to rank with partially-labeled data,” in *SIGIR Conference*. ACM, 2008, pp. 251–258.
<http://ssli.ee.washington.edu/people/katrin/Papers/duh-kirchhoff-sigir08.pdf>

References (4)

- [10] X. Geng, T.-Y. Liu, T. Qin, A. Arnold, H. Li, and H.-Y. Shum, “Query dependent ranking using k-nearest neighbor,” in *SIGIR Conference*. ACM, 2008, pp. 115–122. <http://research.microsoft.com/en-us/people/tyliu/fp025-geng.pdf>
- [11] Y. Yue, T. Finley, F. Radlinski, and T. Joachims, “A support vector method for optimizing average precision,” in *SIGIR Conference*, 2007, pp. 271–278. http://www.cs.cornell.edu/People/tj/publications/yue_etal_07a.pdf

References (5)

- [12] J. Carbonell and J. Goldstein, “The use of MMR, diversity-based reranking for reordering documents and producing summaries,” in *SIGIR Conference*, 1998, pp. 335–336. http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/Web/People/jgc/publication/MMR_DiversityBased_Reranking_SIGIR_1998.pdf
- [13] H. Lin, J. Bilmes, and S. Xie, “Graph-based submodular selection for extractive summarization,” in *Automatic Speech Recognition and Understanding Workshop*, 2009. <http://ssli-mail.ee.washington.edu/people/hlin/papers/lin2009-submodsum.pdf>

References (6)

- [14] H. Lin and J. Bilmes, “Multi-document summarization via budgeted maximization of submodular functions,” in *HLT Conference*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 912–920.
<http://ssli-mail.ee.washington.edu/people/hlin/papers/naaclhlt2010.pdf>
- [15] X. Zhu, A. B. Goldberg, J. Van, and G. D. Andrzejewski, “Improving diversity in ranking using absorbing random walks,” in *HLT-NAACL*, 2007, pp. 97–104. <http://pages.cs.wisc.edu/~jerryzhu/pub/grasshopper.pdf>

References (7)

- [16] Q. Mei, J. Guo, and D. Radev, “DivRank: the interplay of prestige and diversity in information networks,” in *SIGKDD Conference*, 2010, pp. 1009–1018.
http://users.cis.fiu.edu/~lzheng001/KDD_USB_key_2010/docs/p1009.pdf
- [17] S. Agarwal and P. Niyogi, “Stability and generalization of bipartite ranking algorithms,” in *COLT*, Bertinoro, Jun. 2005, pp. 32–47. <http://web.mit.edu/shivani/www/Papers/2005/colt05-stability.pdf>
- [18] A. Agarwal and S. Chakrabarti, “Learning random walks to rank nodes in graphs,” in *ICML*, 2007.
<http://www.cse.iitb.ac.in/~soumen/doc/netrank>

References (8)

- [19] D. Zhou and C. J. C. Burges, "Spectral clustering and transductive learning with multiple views," in *ICML*, 2007. http://research.microsoft.com/~denzho/papers/mv_ICML.pdf
- [20] T.-Y. Liu, T. Qin, J. Xu, W. Xiong, and H. Li, "LETOR: Benchmark dataset for research on learning to rank for information retrieval," in *LR4IR Workshop*, 2007. <http://research.microsoft.com/users/LETOR/>
- [21] T. Minka and S. Robertson, "Selection bias in the LETOR datasets," in *Learning to Rank for Information Retrieval*, 2008, workshop at SIGIR 2008.