Tutorial on Learning to Rank

Ambuj Tewari

Department of Statistics
Department of EECS
University of Michigan

January 13, 2015 / MLSS Austin

Outline

Outline

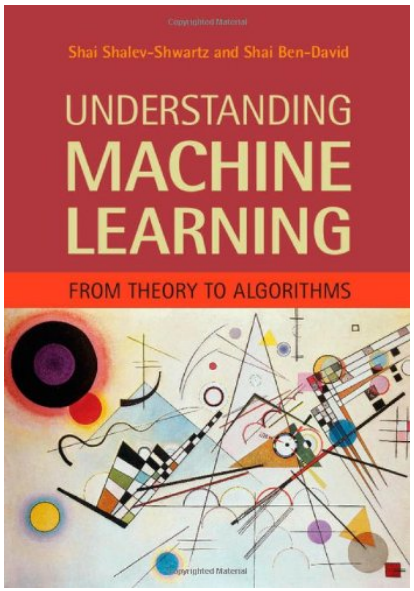
Rankings are everywhere

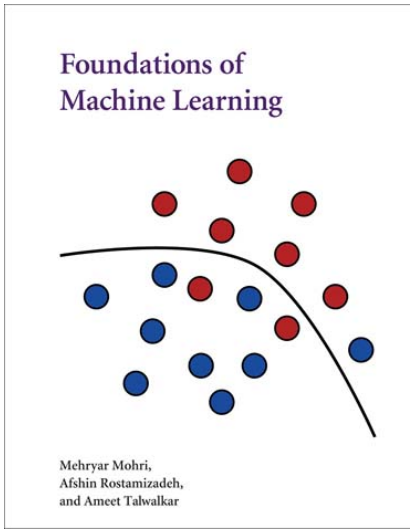
- Web search: PageRank and HITS
- Sports: FIFA rankings (int'l soccer), ICC Rankings (int'l cricket), FIDE rankings (int'l chess), BCS ratings (college football)
- Academia: US News & World Report, Times Higher Education
- Democracy and development: Voting systems, UN Human Development Index
- Recommender systems: movies (Netflix, IMDB), music (Pandora), consumer goods (Amazon)

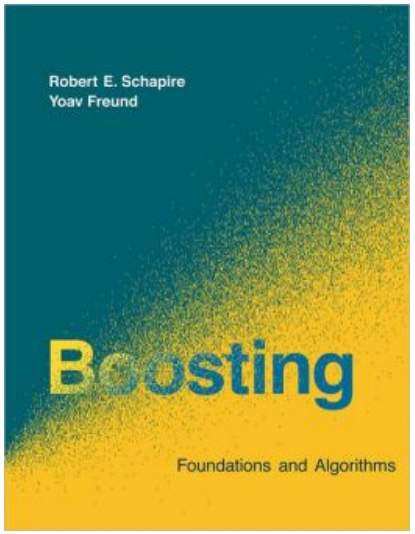
Learning to rank objects

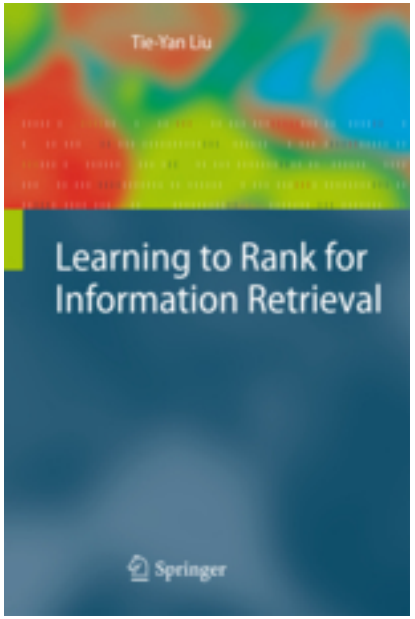
- Creating a ranking requires a lot of effort
- Can we have computers learn to rank objects?
- Will still need some human supervision
- Pose learning to rank as supervised ML problem

Ranking as a canonical learning problem









Ranking as a canonical learning problem

- NIPS Workshops: 2002, 2005, 2009, 2014
- SIGIR Workshops: 2007, 2008, 2009
- Journal of Information Retrieval Special Issue in 2010
- Yahoo! Learning to Rank Challenge: 2010
  `http://jmlr.csail.mit.edu/proceedings/papers/v14/chapelle11a/chapelle11a.pdf`
- LEarning TO Rank (LETOR) and Microsoft Learning to Rank (MSLR) datasets: 2007-2010
  `http://research.microsoft.com/en-us/um/beijing/projects/letor/`

Learning to rank as supervised ML

- Supervised Machine Learning: construct a mapping

$$f : \mathcal{I} \rightarrow \mathcal{O}$$

based on input, output pairs (training examples)

$$\underbrace{(X_1, Y_1), \ldots, (X_N, Y_N)}_{\text{training set}}$$

where $X_n \in \mathcal{I}$, $Y_n \in \mathcal{O}$

- Want $f(X)$ to be a good predictor of Y for unseen X
- In ranking problems, $\mathcal{O}$ is all possible rankings of a given set of objects (webpages, movies, etc)

A learning-to-rank training example

- Query: "MLSS Austin"
- Candidate webpages and relevance scores:
  - `https://www.cs.utexas.edu/mlss`: 2 or "good"
  - `http://mlss.cc/`: 2 or "good"
  - `https://mailman.cc.gatech.edu/...` `/000796.html`: 1 or "fair"
  - `www.councilofmls.com`: 0 or "bad"
- Think: example $X =$(query , (URL1, URL2, . . ., URL4)), and label $R = (2, 2, 1, 0)$

Typical ML cycle

1. Feature construction: Find a way to map (*query*, *webpage*) into $\mathbb{R}^p$
   - Each example (query, m webpages) gets mapped to $X_n \in \mathcal{I} = \mathbb{R}^{m \times p}$
2. Obtain labels: Get $R_n \in \{0, 1, 2, \ldots, R_{\max}\}^m$ from human judgements
3. Train ranker: Get $f : \mathcal{I} \to \mathcal{O} = S_m$ where S_m is the set of m-permutations
   - Training usually done by solving some optimization problems on the training set
4. Evaluate performance: Using some performance measure, evaluate the ranker on a "test set"

Joint query, document features

- Number of query terms that occur in document
- Document length
- Sum/min/max/mean/variance of term frequencies
- Sum/min/max/mean/variance of tf-idf
- Cosine similarity between query and document
- Any model of $P(R = 1|Q, D)$ gives a feature (e.g., BM25)
- No. of slashes in/length of URL, Pagerank, site level Pagerank
- Query-URL click count, user-URL click count

A slightly more general supervised ML setting

- For a future query and webpage-list, want to rank the webpages
- Don't necessarily want to predict the relevance scores
- Let f (ranking function) take values in the space of rankings
- So now, training examples $(X, Y) \in \mathcal{I} \times \mathcal{L}$ where $\mathcal{L}$ = label space
- Function f to be learned maps $\mathcal{I}$ to $\mathcal{O}$ (note $\mathcal{L} \neq \mathcal{O}$)
- E.g. $(1, 2, 2, 0, 0) \in \mathcal{L}$ and $(d_1 \to 3, d_2 \to 1, d_3 \to 2, d_4 \to 4, d_5 \to 5) \in \mathcal{O}$

Notation for rankings

- We'll think of m (no. of documents) as fixed but in reality it varies over examples
- We'll denote a ranking using a permutation $\sigma \in S_m$ (set of all $m!$ permutations)
- $\sigma(i)$ is the rank/position of document i
- $\sigma^{-1}(j)$ is the document at rank/position j

Notation for rankings

- Suppose we have 3 documents d_1, d_2, d_3 and we want them to be ranked thus:

$$d_2$$
$$d_3$$
$$d_1$$

- So the ranks are $d_1 : 3, d_2 : 1, d_3 : 2$
- This means $\sigma(1) = 3, \sigma(2) = 1, \sigma(3) = 2$
- Also, $\sigma^{-1}(1) = 2, \sigma^{-1}(2) = 3, \sigma^{-1}(3) = 1$.

Notation for relevance scores/labels

- Relevance labels can be binary or multi-graded
- In the binary case, documents are either relevant or irrelevant (1 or 0)
- In the multi-graded case, relevance labels are from $\{0, 1, 2, \ldots, R_{\max}\}$
- Typically, $R_{\max} \leq 4$ (at most 5 levels of relevance)
- A relevance score/label vector R as an m-vector
- $R(i)$ gives relevance of document i

Performance measures in ranking

- Performance measures could be gains (to be maximized) or losses (to be minimized)
- They take a relevance vector R and ranking σ as arguments and produce a non-negative gain/loss
- Some performance measures are only defined for binary relevance
- Others can handle more general multi-graded relevance

Performance measure: RR

- Reciprocal rank (RR) is defined for binary relevance only
- RR is the reciprocal rank of the first relevant document according to the ranking

$$\mathrm{RR}(\sigma, Y) = \frac{1}{\min\{\sigma(i) : Y(i) = 1\}} .$$

- Example: $R = (0, 1, 0, 1), \sigma = (1, 3, 4, 2)$ then RR = $\frac{1}{2}$

original	ranked	
$d_1 : 0$	$d_1 : 0$	
$d_2 : 1$	$d_4 : 1$	$\leftarrow$ RR $= \frac{1}{2}$
$d_3 : 0$	$d_2 : 1$	
$d_4 : 1$	$d_3 : 0$	

Performance measure: ERR

- Expected Reciprocal rank (ERR) generalizes RR to multi-graded relevance
- Convert relevance scores into probability (e.g., dividing by $R_{\max}$)
- Imagine a user going down the ranked list and stopping with probability given by relevance scores
- ERR is the expected reciprocal rank of the document at which the user stops
- Example: $R = (0, 2, 0, 1), \sigma = (1, 3, 4, 2)$ then ERR = $\frac{1}{2}\frac{1}{2} + \frac{1}{3}\frac{1}{2}$

original	ranked	cond. stopping prob.	stopping prob.
$d_1 : 0$	$d_1 : 0$	0	0
$d_2 : 2$	$d_4 : 1$	$\frac{1}{2}$	$1 \cdot \frac{1}{2}$
$d_3 : 0$	$d_2 : 2$	$\frac{2}{2}$	$1 \cdot \frac{1}{2} \cdot \frac{2}{2}$
$d_4 : 1$	$d_3 : 0$	0	0

Performance measure: ERR

$$ERR(\sigma, R) = \sum_{j=1}^{m} \frac{1}{j} \cdot \underbrace{\left(\prod_{k=1}^{j-1} \left(1 - g(R(\sigma^{-1}(k)))\right) \right)}_{\text{prob. of not stopping earlier}} \cdot \underbrace{g(R(\sigma^{-1}(j)))}_{\text{prob. of stopping at pos. } j}$$

- Recall: $R(\sigma^{-1}(j))$ is the relevance of document at position j
- g is any function used to convert relevance scores into stopping probabilities
- E.g., $g(r) = \frac{r}{R_{\max}}$ or $g(r) = \frac{2^r - 1}{2^{R_{\max}}}$

Performance measure: Precision@K

- Precision@K is for binary relevance only
- Precision@K is the fraction of relevant documents in the top-k

$$Prec@K(\sigma, R) = \frac{1}{K} \sum_{j=1}^{K} R(\sigma^{-1}(j))$$

original	ranked	$Prec@K$
$d_1 : 0$	$d_1 : 0$	$0/1$
$d_2 : 1$	$d_4 : 1$	$1/2$
$d_3 : 0$	$d_2 : 1$	$2/3$
$d_4 : 1$	$d_3 : 0$	$2/4$

Performance measure: AP

- Average Precision (AP) is also for binary relevance
- It is the average of Precision@K over positions of relevant documents only

$$AP(\sigma, R) = \frac{1}{\sum_{i=1}^{m} R(i)} \cdot \left(\sum_{j:R(\sigma^{-1}(j))=1} Prec@j \right)$$

original	ranked	$Prec@K$	
$d_1 : 0$	$d_1 : 0$	$0/1$	
$d_2 : 1$	$d_4 : 1$	$1/2 \leftarrow$	$AP = \frac{1/2+2/3}{2}$
$d_3 : 0$	$d_2 : 1$	$2/3 \leftarrow$	
$d_4 : 1$	$d_3 : 0$	$2/4$	

Performance measure: DCG

- Discounted Cumulative Gain (DCG) is for multi-graded relevance
- Contributions of relevant documents further down the ranking are discounted more

$$DCG(\sigma, R) = \sum_{i=1}^{m} \frac{f(R(i))}{g(\sigma(i))}$$

- f, g are increasing functions
- Standard choice: $f(r) = 2^r - 1$ and $g(j) = \log_2(1 + j)$

Performance measure: NDCG

- Normalized DCG (NDCG), like DCG, is for multi-graded relevance
- As the name suggests, NDCG normalizes DCG to keep the gain bounded by 1

$$NDCG(\sigma, R) = \frac{DCG(\sigma, R)}{Z(R)}$$

- The normalization constant

$$\max_{\sigma} DCG(\sigma, R)$$

can be computed quickly by sorting the relevance vector

Ranking functions

- A ranking function maps m query-document p-dimensional features into a permutation

$$X = \underbrace{\begin{bmatrix} \phi(q, d_1)^\top \\ \vdots \\ \phi(q, d_m)^\top \end{bmatrix}}_{m \times d \text{ matrix}} \longrightarrow \begin{matrix} d_5 & \Leftrightarrow & \sigma(5) = 1 \\ \vdots & & \\ d_{12} & \Leftrightarrow & \sigma(12) = m \end{matrix}$$

- So mathematically it is a map from $\mathbb{R}^{m \times p}$ to S_m

Ranking functions from scoring functions

- However, it is hard to learn functions that map into combinatorial spaces
- In binary classification, we often learn a real-valued function and threshold it to output a class label (positive/negative)
- In ranking, we do the same but replace thresholding by sorting
- For example, let $f : \mathbb{R}^p \to \mathbb{R}$ be a scoring function

$$\begin{bmatrix} \phi(q,d_1)^\top \\ \phi(q,d_2)^\top \\ \phi(q,d_3)^\top \end{bmatrix} \xrightarrow{f} \begin{bmatrix} -2.3 \\ 4.2 \\ 1.9 \end{bmatrix} \xrightarrow{\text{sort}} \begin{array}{lll} d_2 & \Leftrightarrow & \sigma(2) = 1 \\ d_3 & \Leftrightarrow & \sigma(3) = 2 \\ d_1 & \Leftrightarrow & \sigma(1) = 3 \end{array}$$

Recap

- Training data consists query-document features and relevance scores

$$X = \underbrace{\begin{bmatrix} \phi(q, d_1)^\top \\ \vdots \\ \phi(q, d_m)^\top \end{bmatrix}}_{m \times d \text{ matrix}}, \quad R = \begin{bmatrix} R(1) \\ \vdots \\ R(m) \end{bmatrix}$$

- Will use training data to learn a scoring function f that will be used to rank documents for a new query

$$\begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix} = \begin{bmatrix} f(\phi(q^{\text{new}}, d_1^{\text{new}})) \\ \vdots \\ f(\phi(q^{\text{new}}, d_m^{\text{new}})) \end{bmatrix}$$

- Scores can be sorted to get a ranking whose performance will evaluated using a measure such as ERR, AP, NDCG

Outline

- Many nice, innovative methods have been developed
- A partial list:

AdaRank	BoltzRank	Cranking	FRank	Infinite push
LambdaMART	LambdaRank	ListMLE	ListNET	McRank
MPRank	p-norm push	OWPC	PermuRank	PRank
RankBoost	Ranking SVM	RankNet	SmoothRank	SoftRank
SVM-Combo	SVM-MAP	SVM-MRR	SVM-NDCG	SVRank

The landscape of ideas

- Reduction: Reduce ranking to a simpler problem (e.g., classification or regression)
- Boosting: Combine weak rankers and boost them into strong ones
- Large margin approach: Extend the "large-margin" story behind SVMs to the ranking case
- Direct/Indirect optimization: Choose an appropriate optimization problem based on the performance measure one wishes to optimize

Not all ranking methods neatly fall into one of these categories!

The landscape of ideas

- Reduction: Reduce ranking to a simpler problem (e.g., classification or regression)
- Boosting: Combine weak rankers and boost them into strong ones
- Large margin approach: Extend the "large-margin" story behind SVMs to the ranking case
- Direct/Indirect optimization: Choose an appropriate optimization problem based on the performance measure one wishes to optimize

Not all ranking methods neatly fall into one of these categories!

Regression

- Perhaps the simplest approach: regress the relevance score on query-document features
- Feed examples of the form $(\phi(q, d_i), R(i))$ to your favorite regression algorithm
- Advantage: can use already built regression tools
- Disadvantage: there's no real reason for us to predict relevance scores, we're just building a ranking function

Linear regression example

- The simplest scoring functions are linear functions:

$$f(\phi(q,d)) = w^\top \phi(q,d) = \sum_{j=1}^{p} w_j \phi_j(q,d)$$

- Using linear scoring functions in a regression approach reduces ranking to linear regression:

$$\widehat{w} = \operatorname*{argmin}_w \sum_{n=1}^{N} \sum_{i=1}^{m} (R_n(i) - w^\top \phi(q_n, d_{n,i}))^2$$

- For a new query-document example, output a ranking by sorting $(\widehat{w}^\top \phi(q,d_i))_{i=1}^m$.

Classification

- If d_i has higher relevance than d_j for a query q, then feed the examples of the form $(\phi(q, d_i) - \phi(q, d_j), +1)$ to your favorite classification algorithm
- Advantages: can use already build classification tools
- Disadvantages: not clear how to use pairwise classification to create a ranking

Logistic regression example

- A logistic regression approach reduces ranking to:

$$\widehat{w} = \operatorname*{argmin}_{w} \sum_{n=1}^{N} \sum_{R_n(i)>R_n(j)} \ell_{\log}(w^\top(\phi(q,d_i) - \phi(q,d_j)), +1)$$

- $\ell_{\log}$ is the logistic loss

$$\ell_{\log}(w^\top x, y) = \log(1 + \exp(-y \cdot w^\top x))$$

- Other losses could also be used:

$$\ell_{\text{hinge}}(w^\top x, y) = \max\{0, 1 - y \cdot w^\top x\}$$

Logistic regression: producing a ranking

- On a new triple (q, d_i, d_j) the trained classifier will be able to tell you whether d_i is more relevant than d_j for the query q
- How to resolve conflicts (such as cycles) and output a ranking of $d_1, \ldots, d_m$?
- Run quick sort!
  - Choose a pivot document d_i at random
  - Use classifier to partition other documents into: more relevant $D_>$ and less relevant $D_<$ than d_i
  - Return quicksort($D_>$), d_i, quicksort($D_<$)

Boosting in classification

- Simple rules of thumb can often predict a little better than random guessing:
  "$" in subject line $\Rightarrow$ email is spam
- How to combine simple rules of thumb into a powerful classifier (say with 90% accuracy)?
- Adaboost: perhaps the most popular boosting algorithm

Adaboost

Training data $(X_1, Y_1), \ldots, (X_N, Y_N)$
Start from uniform distribution $P_1(i) = 1/N$ for all $1 \leq n \leq N$
At iteration $t = 1, 2, \ldots, T$:

- Train a weak classifier h_t on training data T using weights from P_t
- Choose step-size α_t and set

$$f_t = \sum_{s=1}^{t} \alpha_s h_s$$

- Update P_t to P_{t+1} so that P_{t+1} puts more weight on examples where h_t performs badly

Final classifier is (sign of) $\sum_{t=1}^{T} \alpha_t h_t$.

Adarank

Training data $(X_1, R_1), \ldots, (X_N, R_N)$
Start from uniform distribution $P_1(n) = 1/N$ for all $1 \leq n \leq N$
At iteration $t = 1, 2, \ldots, T$:

- Train a weak ranker h_t on training data T using weights from P_t
- Choose step-size α_t and set

$$f_t = \sum_{s=1}^{t} \alpha_s h_s$$

- Update P_t to P_{t+1} so that P_{t+1} puts more weight on examples where f_t performs badly

Final ranker is (sorted order given by) $\sum_{t=1}^{T} \alpha_t h_t$.

Adarank: weak ranker

- Assume that RL (ranking loss) is the target loss we wish to minimize (e.g., 1-NDCG)
- Weak ranker can simply be a single feature (e.g., Pagerank) that minimizes the loss, i.e.,

$$\underset{k \in \{1,\ldots,p\}}{\operatorname{argmin}} \sum_{n=1}^{N} P_t(n) RL(X_n \mathbf{e}_k, R_n)$$

$$X_n \mathbf{e}_k = \begin{bmatrix} \phi(q_n, d_{n,1})^\top \\ \vdots \\ \phi(q_n, d_{n,m})^\top \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow k\text{th position}$$

Adarank: distribution update

- The next distribution over training example is obtained using:

$$P_{t+1}(n) = \frac{\exp\left(-RL(f_t(X_n), Y_n)\right)}{Z_{t+1}}$$

- The normalization Z_{t+1} ensures that we have a probability distribution

$$Z_{t+1} = \sum_{n=1}^{N} \exp\left(-RL(f_t(X_n), Y_n)\right)$$

Support Vector Machine

SVMs solve the following problem ($\|w\|_2^2 := w^\top w$):

$$\min_w \quad \underbrace{\|w\|_2^2}_{L_2 \text{ regularization}} + \underbrace{C}_{\text{tuning param.}} \sum_{i=1}^{n} \xi_i$$

subject to

$$\xi_i \geq 0$$

$$\underbrace{Y_i w^\top X_i}_{\text{margin}} \geq 1 - \underbrace{\xi_i}_{\text{slack variable}}$$

Ranking SVM

- Consider a query q_n and documents $d_{i,n}$ and $d_{j,n}$ such that

$$R_n(i) > R_n(j)$$

- We would like to have

$$w^\top \phi(q_n, d_{n,i}) > w^\top \phi(q_n, d_{n,j})$$

- As in SVM, we can introduce a slack variable $\xi_{n,i,j}$ with the constraints

$$\xi_{n,i,j} \geq 0$$
$$w^\top \phi(q_n, d_{n,i}) \geq w^\top \phi(q_n, d_{n,j}) + 1 - \xi_{n,i,j}$$

Ranking SVM

Ranking SVM solves the following problem

$$\min_{w} \|w\|_2^2 + C \sum_{n=1}^{N} \sum_{R_n(i) > R_n(j)} \xi_{n,i,j}$$

subject to the constraints

$$\xi_{n,i,j} \geq 0$$
$$w^\top \phi(q_n, d_{n,i}) \geq w^\top \phi(q_n, d_{n,j}) + 1 - \xi_{n,i,j}$$

This is actually also a reduction approach!

Ranking SVM

Ranking SVM solves the following problem

$$\min_{w} \|w\|_2^2 + C \sum_{n=1}^{N} \sum_{R_n(i) > R_n(j)} \xi_{n,i,j}$$

subject to the constraints

$$\xi_{n,i,j} \geq 0$$
$$w^\top \phi(q_n, d_{n,i}) \geq w^\top \phi(q_n, d_{n,j}) + 1 - \xi_{n,i,j}$$

This is actually also a reduction approach!

Ranking SVM as regularized loss minimization

- Note that the minimum possible value of ξ subject to $\xi \geq 0$ and $\xi \geq t$ is

$$\max\{t, 0\} =: [t]_+$$

- This observation lets us write the ranking SVM optimization as

$$\min_w \underbrace{\|w\|_2^2}_{\text{regularizer}} + C \sum_{n=1}^{N} \underbrace{\sum_{R_n(i) > R_n(j)} [1 + w^\top \phi(q_n, d_{n,j}) - w^\top \phi(q_n, d_{n,i})]_+}_{\text{loss on example } n}$$

Regularized loss minimization

- Note that

$$X_n w = \begin{bmatrix} \phi(q_n, d_{n,1})^\top w \\ \vdots \\ \phi(q_n, d_{n,m})^\top w \end{bmatrix}$$

- Ranking SVM problem (where $\lambda = 1/C$):

$$\min_w \ \lambda \|w\|_2^2 + \sum_{n=1}^{N} \ell(X_n w, R_n)$$

- Loss in ranking SVM is

$$\ell(s, R) = \sum_{R(i)>R(j)} [1 + s_j - s_i]_+$$

Why not directly optimize performance measures?

- Let RL be some actual loss you want to minimize (e.g., 1-NDCG)
- Then why not solve the following?

$$\min_{w} \ \lambda \|w\|_2^2 + \sum_{n=1}^{N} RL(\text{argsort}(X_n w), R_n)$$

- argsort(t) is a ranking σ that puts elements of t in (decreasing) sorted order:

$$t_{\sigma^{-1}(1)} \geq t_{\sigma^{-1}(2)} \geq \cdots \geq t_{\sigma^{-1}(m)}$$

Why not directly optimize performance measures?

- The function

$$w \mapsto RL(\text{argsort}(Xw), R)$$

  is not a nice one (from optimization perspective)!
- In the neighborhood of any point, it is either flat or discontinuous
- It can only take one of at most $m!$ different values
- Minimizing a sum of such functions is computationally intractable

A smoothing approach

- Instead of deterministic score $s = Xw$, think of smoothed score distributions:

$$S \sim \mathcal{N}(s, \sigma^2 I_{m \times m})$$

- This gives us a smoothed ranking loss

$$RL_\sigma(s, R) = \mathbb{E}_S[RL(\text{argsort}(S), R)]$$

- We can then optimize (using, say, gradient descent):

$$\min_w \ \lambda \|w\|_2^2 + \sum_{n=1}^{N} RL_\sigma(X_n w, R_n)$$

Choosing the smoothing parameter

- As σ grows, the optimization problem becomes easier and we're less prone to overfitting
- As σ shrinks, the smoothed loss approximates the underlying loss better
- Annealing approach of SmoothRank:

  Initialize w_0 (e.g., using linear regression)
  Choose a large σ_1
  For $t = 1, 2, \ldots$:
  - Starting from w_{t-1}, solve regularized RL_{σ_t} minimization using (some version of) gradient descent
  - Let the new solution be w_t and set $\sigma_{t+1} = \sigma_t/2$

Convex surrogates

- The ranking SVM problem is convex hence easy to optimize
- But no direct relation to a performance measure
- Smoothing approach starts from a performance measure
- Solves smooth but non-convex problems (can get stuck in local optimum)
- Given a hard-to-optimize performance measure, can we indirectly minimize it via some convex surrogate?

Recap

- We can try to reduce ranking to another ML problem: e.g., regression or classification
- We can try to boost weak rankers into stronger ones
- The margin maximization approach of SVMs leads naturally to Ranking SVM
- Ranking SVM solves regularized loss minimization
- We can directly try to minimize target performance measure
  (hard because of flatness/discontinuity; might get stuck in local minimum even after smoothing)
- Can we solve convex problems (no non-global local minima) and yet retain a principled connection to a target performance measure?

Outline

Theory: still a long way to go

- An amazing amount of applied work exists
- Yet, a lot of theoretical work remains to be done

> *"However, sometimes benchmark experiments are not as reliable as expected due to the small scales of the training and test data. In this situation, a theory is needed to guarantee the performance of an algorithm on infinite unseen data."*
>
> *–O. Chapelle, Y. Chang, and T.-Y. Liu*
> *"Future Directions in Learning to Rank" (2011)*

Statistical Ranking Theory: Challenges

- Combinatorial nature of output space
  - Size of prediction space is $m!$ (m = no. of things being ranked)
- No canonical performance measure
  - Many choices: ERR, MAP, (N)DCG
  - Even with a fixed choice, how to come up with “good" convex surrogates
- Dealing with diverse types of feedback models
  - relevance scores, pairwise feedback, click-through rates, preference DAGs
- Large-scale challenges
  - guarantees for parallel, distributed, online/streaming approaches

Our focus

- Human feedback in form of relevance scores
- Regularized convex loss minimization based ranking methods

$$\min_{w} \ \lambda \|w\|_2^2 + \sum_{n=1}^{N} \ell(X_n w, R_n)$$

- A few commonly used performance measures: NDCG, MAP
- Generalization error bounds and consistency

Standard Setup

- Standard assumption in theory: examples

$$(X_1, R_1), \ldots, (X_N, R_N)$$

are drawn iid from an underlying (unknown) distribution

- Consider the constrained form of loss minimization

$$\widehat{w} = \underset{\|w\|_2 \leq B}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} \ell(X_n w, R_n)$$

Risk

- Risk of w: performance of w on “infinite unseen data"

$$R(w) = \mathbb{E}_{X,R}[\ell(Xw, R)]$$

Note: cannot be computed without knowledge of underlying distribution

- Empirical risk of w: performance of w on training set

$$\widehat{R}(w) = \frac{1}{N}\sum_{n=1}^{N} \ell(X_n w, R_n)$$

Note: can be computed using just the training data

Generalization error bound

- We say that $\widehat{w}$ is an empirical risk minimizer (ERM)

$$\widehat{w} = \operatorname*{argmin}_{\|w\|_2 \leq B} \widehat{R}(w)$$

- We're typically interested in bounds on the generalization error

$$R(\widehat{w})$$

- For instance, we expect the generalization error to be larger than

$$\widehat{R}(\widehat{w})$$

but by how much?

A downward bias

- Let us show that $\widehat{R}(\widehat{w})$ is, in expectation, less than $R(\widehat{w})$
- Let $w^\star$ be the scoring function with least expected loss

$$w^\star = \operatorname*{argmin}_{\|w\|_2 \leq B} R(w)$$

$$\widehat{R}(\widehat{w}) \leq \widehat{R}(w^\star) \quad \text{by definition of ERM}$$

$$\mathbb{E}[\widehat{R}(\widehat{w})] \leq \mathbb{E}[\widehat{R}(w^\star)] \quad \text{taking expectations}$$

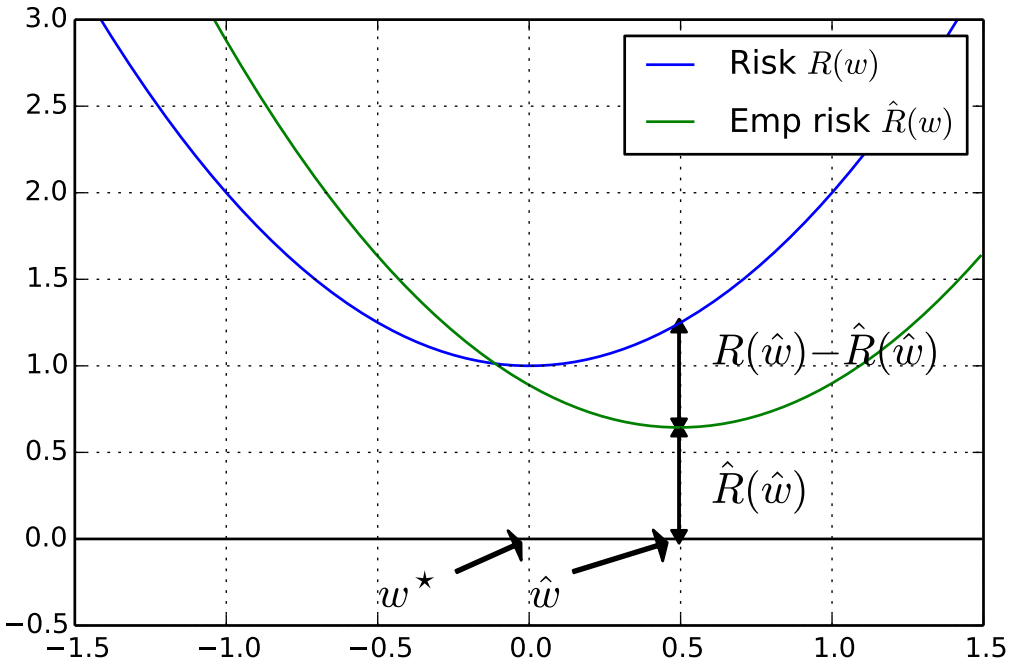
$$\mathbb{E}[\widehat{R}(\widehat{w})] \leq R(w^\star) \quad \because \forall w, \mathbb{E}[\widehat{R}(w)] = R(w)$$

$$\mathbb{E}[\widehat{R}(\widehat{w})] \leq R(\widehat{w}) \quad \text{by definition of } w^\star$$

Generalization error bound

- We might therefore want a generalization error bound of the form

$$R(\widehat{w}) \leq \widehat{R}(\widehat{w}) + \text{ stuff}$$



Generalization error bound

- A useless generalization error bound:

$$R(\widehat{w}) \leq \widehat{R}(\widehat{w}) + |R(\widehat{w}) - \widehat{R}(\widehat{w})|$$

- Better?

$$R(\widehat{w}) \leq \widehat{R}(\widehat{w}) + \max_{\|w\|_2 \leq B} |R(w) - \widehat{R}(w)|$$

Bootstrapping

- The difference $R(w) - \widehat{R}(w)$ compares the expected loss of w under

$$(X, R) \sim \text{underlying dist.} \quad \text{vs.} \quad (X, R) \sim (X_n, R_n) \text{ with wt. } \frac{1}{N}$$

- Let us replace the underlying dist. with the sample and the sample with a (weighted) sub-sample!

$$(X, R) \sim (X_n, R_n) \text{ with wt. } \frac{1}{N} \quad \text{vs.}$$

$$(X, R) \sim (X_n, R_n) \text{ with wt. } \begin{cases} \frac{2}{N} & \text{if } \epsilon_n = +1 \\ 0 & \text{if } \epsilon_n = -1 \end{cases}$$

where ϵ_i are symmetric signed Bernoulli or Rademacher random variables

Weighted subsample: an example

- Suppose original sample with weights is

$$(X_1, R_1) : \frac{1}{3}, (X_2, R_2) : \frac{1}{3}, (X_3, R_3) : \frac{1}{3}$$

- Suppose $\epsilon_1 = +1, \epsilon_2 = -1, \epsilon_3 = +1$
- Then weighted sub-sample is

$$(X_1, R_1) : \frac{2}{3}, (X_3, R_3) : \frac{2}{3}$$

Towards a generalization error bound

- Define the weighted average under on sub-sample

$$\widehat{R}'(w) = \frac{1}{N}\sum_{n=1}^{N}(1+\epsilon_n)\ell(X_n w, R_n)$$

- Replace

$$\max_{\|w\|_2 \leq B} |R(w) - \widehat{R}(w)|$$

with

$$\max_{\|w\|_2 \leq B} |\widehat{R}(w) - \widehat{R}'(w)|$$

A data-dependent generalization error bound

- With probability at least $1 - \delta$

$$R(\widehat{w}) \leq \underbrace{\widehat{R}(\widehat{w})}_{\text{shrinks as } B\uparrow} + \underbrace{\max_{\|w\|_2 \leq B} |\widehat{R}(w) - \widehat{R}'(w)|}_{\text{grows as } B\uparrow} + \sqrt{\frac{\log(1/\delta)}{n}}$$

- Can be used for data-dependent selection of B

Rademacher complexity

- Note:

$$\widehat{R}(w) - \widehat{R}'(w) = \frac{1}{N}\sum_{n=1}^{N}(1-(1+\epsilon_n))\ell(X_n w, R_n) = \frac{1}{N}\sum_{n=1}^{N} -\epsilon_n \ell(X_n w, R_n)$$

- Therefore, with high probability,

$$R(\widehat{w}) \leq \widehat{R}(\widehat{w}) + \max_{\|w\|_2 \leq B} \left| \frac{1}{N}\sum_{n=1}^{N} \epsilon_n \ell(X_n w, R_n) \right| + \sqrt{\frac{\log(1/\delta)}{n}}$$

- The quantity

$$\mathbb{E}_{\epsilon_1,\ldots,\epsilon_N}\left[\max_{\|w\|_2 \leq B} \left| \frac{1}{N}\sum_{n=1}^{N} \epsilon_n \ell(X_n w, R_n) \right| \right]$$

is called Rademacher complexity

How does Rademacher complexity scale?

Rademacher complexity depends on

- Loss function ℓ
  More "complicated" loss $\implies$ bigger complexity
- Size B of w's
  Larger B $\implies$ bigger complexity
- Size of features, say $\|\phi(q, d_i)\|_2 \leq R$
  Larger R $\implies$ bigger complexity
- Sample size N
  Larger N $\implies$ smaller complexity

Roughly speaking, scales as:

$$\mathrm{Lip}(\ell) \cdot B \cdot R \cdot \sqrt{\frac{\log(1/\delta)}{N}}$$

Lipschitz constant

- Fix arbitrary relevance vector R
- How much does $\ell(s, R)$ change as we change s?
- Lipschitz constant $\mathrm{Lip}(\ell)$ is smallest L such that

$$\forall R,\ |\ell(s, R) - \ell(s', R)| \leq L \cdot \|s - s'\|_\infty$$

where

$$\|s - s'\|_\infty = \max_{i=1}^{m} |s_i - s'_i|$$

Another type of generalization bound

- So far we have seen bounds of the form

$$R(\widehat{w}) \leq \widehat{R}(\widehat{w}) + \text{stuff}$$

- However, $\widehat{R}(\widehat{w}) \leq \widehat{R}(w^\star)$ and $\widehat{R}(w^\star)$ concentrates around $R(w^\star)$
- Therefore, it is also easy to derive bounds of the form

$$R(\widehat{w}) \leq R(w^\star) + \text{stuff}$$

Generalization bounds: recap

- Risk $R(w)$ is expected loss of w on infinite, unseen data
- Empirical risk $\widehat{R}(w)$ is average loss on training data
- A $\widehat{w}$ that minimizes loss on training data is called an empirical risk minimizer (ERM)
- Saw two types of generalization error bounds:

$$R(\widehat{w}) \leq \widehat{R}(\widehat{w}) + \text{stuff}$$

$$R(\widehat{w}) \leq \underbrace{R(w^\star)}_{\text{minimum possible risk}} + \text{stuff}$$

- Rademacher complexity can be used to give data dependent generalization bounds for ERM

Richer function classes for more data

- As we see more data, it is natural to enlarge the function class over which ERM is computed

$$\widehat{f}_n = \operatorname*{argmin}_{f \in \mathcal{F}_n} \frac{1}{N} \sum_{n=1}^{N} \ell(f(X_n), R_n) = \operatorname*{argmin}_{f \in \mathcal{F}_n} \widehat{R}(f)$$

where

$$f(X_n) = \begin{bmatrix} f(\phi(q_n, d_{n,1})) \\ \vdots \\ f(\phi(q_n, d_{n,m})) \end{bmatrix}$$

- E.g., as n grows, $\mathcal{F}_n$ = larger decision trees, deeper neural networks, SVMs with less regularization

Estimation error vs approximation error

- Let $f^\star(\mathcal{F}_n)$ be the best function in $\mathcal{F}_n$ and $f^\star$ be the best overall function:

$$f^\star(\mathcal{F}_n) = \operatorname*{argmin}_{f \in \mathcal{F}_n} R(f) \quad , \quad f^\star = \operatorname*{argmin}_{f} R(f)$$

- We can decompose the excess risk

$$R(\widehat{f}_n) - R(f^\star) = \underbrace{R(\widehat{f}_n) - R(f^\star(\mathcal{F}_n))}_{\text{estimation error}} + \underbrace{R(f^\star(\mathcal{F}_n)) - R(f^\star)}_{\text{approximation error}}$$

Consistency

- If $\mathcal{F}_n$ grows too quickly, estimation error will not go to zero
- If $\mathcal{F}_n$ grows too slowly, approximation error will not go to zero
- By judiciously growing $\mathcal{F}_n$, we can guarantee that both errors go to zero
- This results in consistency

$$R(\widehat{f}_n) \to R(f^\star) \text{ as } n \to \infty$$

What did we achieve?

- We said that we will use a nice (smooth/convex) loss ℓ as a surrogate and train scoring function $\widehat{f}_n$ on finite data using ℓ

$$\widehat{f}_n = \operatorname*{argmin}_{f \in \mathcal{F}_n} \frac{1}{N} \sum_{n=1}^{N} \ell(f(X_n), R_n)$$

- Enlarging $\mathcal{F}_n$ neither too quickly nor too slowly with n will give us

$$R(\widehat{f}_n) \overset{n \to \infty}{\longrightarrow} R(f^\star)$$

- Great, but how good is $\widehat{f}_n$ according to ranking performance measures (NDCG, ERR, AP)?

Risk under ranking loss

- Recall that $R(f)$ is the risk of f under the surrogate ℓ:

$$R(f) = \mathbb{E}_{X,R}[\ell(f(X), R)]$$

- Similarly, there is $L(f)$, the risk of f under a target ranking loss RL:

$$L(f) = \mathbb{E}_{X,R}[RL(\text{argsort}(f(X)), R)]$$

- argsort required because first argument of RL has to be a ranking, not a score vector

Calibration

- Fix a target loss RL (e.g., 1-NDCG) and a surrogate ℓ (e.g., RankingSVM loss)
- Suppose we know

$$R(\widehat{f}_n) \overset{n\to\infty}{\longrightarrow} \min_f \; R(f)$$

- Does this automatically guarantee the following?

$$L(\widehat{f}_n) \overset{n\to\infty}{\longrightarrow} \min_f \; L(f)$$

- If yes, then we say that surrogate ℓ is calibrated w.r.t. target loss RL

Calibration in binary classification

- Consider the simpler setting of binary classification where $X_i \in \mathbb{R}^p$ and $Y_i \in \{+1, -1\}$
- We learn real valued functions $f : \mathbb{R}^p \to \mathbb{R}$ and take the sign to output a class label
- Given a surrogate ℓ (hinge loss, logistic loss), we can define

$$R(f) = \mathbb{E}_{X,Y}[\ell(f(X), Y)]$$

- Target loss is often just the 0-1 loss:

$$L(f) = \mathbb{E}_{X,Y}[1_{(Yf(X) \leq 0)}]$$

Calibration in binary classification

- Suppose $\ell(Y, f(X)) = \phi(Yf(X))$ (a margin loss)
- For a margin-based convex surrogate ℓ, when does

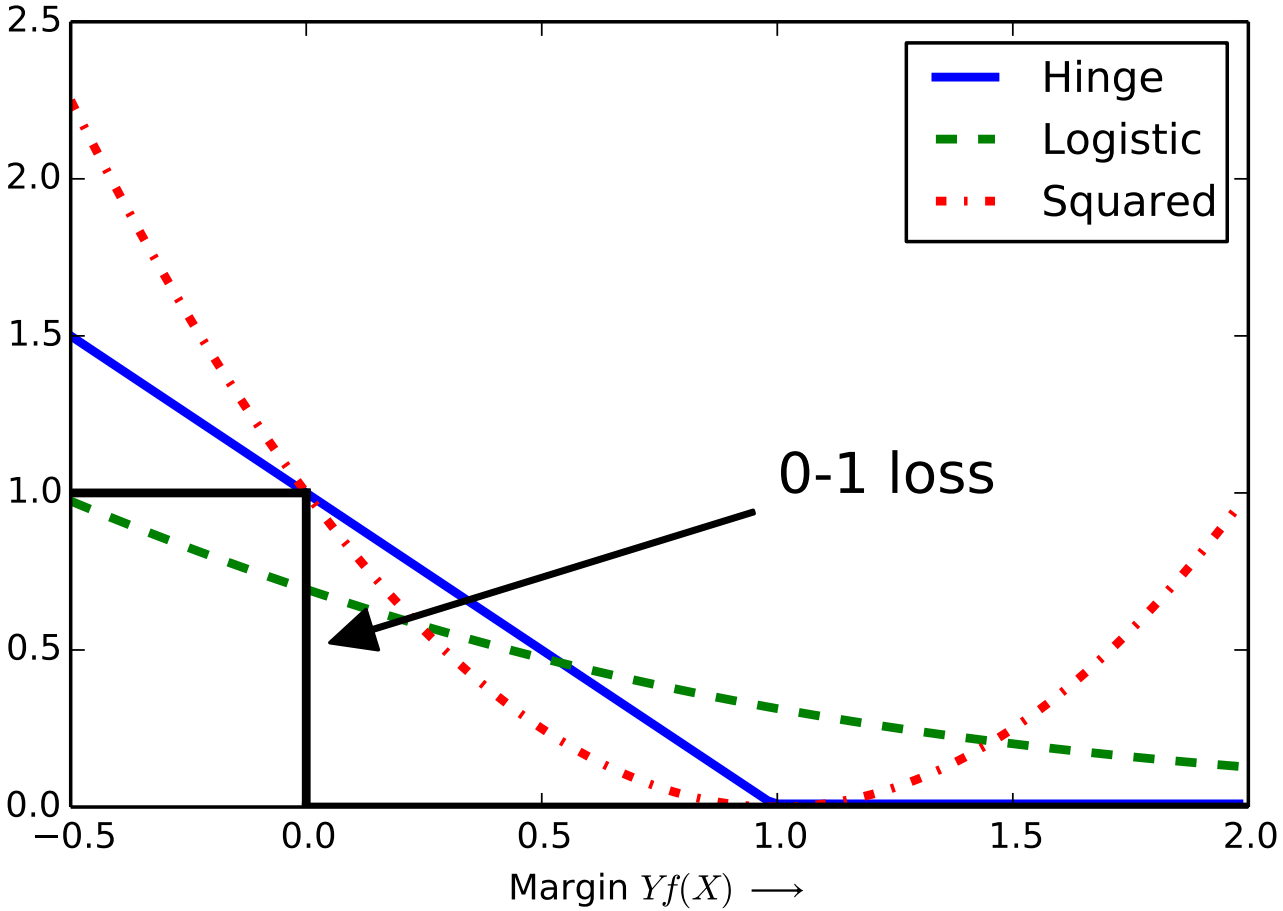
$$R(\widehat{f}_n) \xrightarrow{n\to\infty} \min_f R(f)$$

imply

$$L(\widehat{f}_n) \xrightarrow{n\to\infty} \min_f L(f)$$

- Surprisingly simple answer: $\phi'(0) < 0$

Examples of calibrated surrogates



Calibration w.r.t. DCG

- Recall that DCG is defined as:

$$DCG(\sigma, R) = \sum_{i=1}^{m} \frac{f(R(i))}{g(\sigma(i))}$$

- Let η be a distribution over relevance vectors R's
- The minimizer $\sigma(\eta)$ of

$$\sigma \mapsto \mathbb{E}_{R\sim\eta}[DCG(\sigma, R)]$$

is argsort($\mathbb{E}_{R\sim\eta}[f(R)]$)

Calibration w.r.t. DCG

- Consider the score vector $s(\eta)$ that minimizes

$$s \mapsto \mathbb{E}_{R\sim\eta}[\ell(s, R)]$$

- Necessary and sufficient condition for *DCG* calibration
- For every η, $s(\eta)$ has the same sorted order as $\mathbb{E}_{R\sim\eta}[f(R)]$
- Similar condition can be derived for NDCG by taking normalization into account

Calibration w.r.t. DCG: Example

- Consider the regression based surrogate

$$\ell(s, R) = \sum_{i=1}^{m} (s_i - f(R_i))^2$$

- Easy to see that $s(\eta)$ is exactly $\mathbb{E}_{R\sim\eta}[f(R)]$
- Thus, this regression based surrogate is DCG calibrated

Calibration w.r.t. ERR, AP

- The (N)DCG case might lead one to believe that convex calibrated surrogates always exist
- Surprisingly, this is not the case!
- It is known that there do not exist any convex calibrated surrogates for ERR and AP
- Caveat: Non-existence result assumes that we're using argsort to move from scores to rankings

Consistency: recap

- Consistency is an asymptotic (sample size $N \to \infty$) property of learning algorithms
- A consistent algorithm's performance reaches that of the "best" ranker as it sees more and more data
- A learning algorithm that minimizes a surrogate ℓ has to balance estimation error, approximation error trade-off
- If trade-off is managed well, it will have consistency in the sense of ℓ
- If this also implies consistency w.r.t. a target loss *RL*, we say ℓ is calibrated w.r.t. *RL*
- Calibration in ranking is trickier than in classification: easy in some cases (e.g. NDCG), not so easy in others (e.g., ERR, MAP)

Outline

The online protocol

Instead of training on a fixed batch dataset, ranking could occur in a more online/incremental setting

For $t = 1, \ldots, N$

- Ranker sees feature vectors $X_t \in \mathbb{R}^{m \times p}$
- Ranker outputs a scoring function $f_t : \mathbb{R} \to \mathbb{R}$
- Relevance vector R_t is revealed
- Ranker suffers loss $\ell(f_t(X_t), R_t)$

Regret

- In online settings, we often look at regret:

$$\underbrace{\sum_{t=1}^{N} \ell(f_t(X_t), R_t)}_{\text{algorithm's loss}} - \underbrace{\min_{f \in \mathcal{F}} \sum_{t=1}^{N} \ell(f(X_t), R_t)}_{\text{best loss in hindsight}}$$

- If ℓ is convex and we're using linear scoring functions $f(x) = w^\top x$ then we can use tools from online convex optimization (OCO)
- Great OCO introduction here `http://ocobook.cs.princeton.edu/`

Diversity

- Focusing solely on relevance can lead to redundancy in search results
- A query might be ambiguous: jaguar (car vs. animal)
- Simply listing highly relevant results for one facet of the query isn't good
- Need to encourage diversity in the search results
- New performance measures/methods/theoretical framework needed

Freshness

- Consider ranking news items or tweets for queries involving recent events
- Need to promote more recent webpages
- But only if the query is indeed recency sensitive
- Different time scales: for an annual event (year) vs. breaking news (days/hours)
- Possible approach: classify query as recency-sensitive or not and then route to an appropriate ranker

Large-scale learning to rank

- Gradient descent batch optimization of a regularized objective

$$\lambda \|w\|_2^2 + \sum_{n=1}^{N} \ell(X_n w, R_n)$$

  can easy be parallelized on a cluster
- Ensemble based methods like bagging can train rankers on different partitions of data in parallel
- These ideas have been used for supervised ML but not much in ranking
- See the Apache Spark project: `https://spark.apache.org/docs/latest/mllib-guide.html`

Summary

- Ranking problems arise in a variety of contexts
- Learning to rank can be, and has been, viewed as a supervised ML problem
- We surveyed a wide landscape of ranking methods
- We discussed generalization error bounds for and consistency of ranking methods
- We saw a few advanced topics in learning to rank that are being actively investigated

Suggested activities

- Get the LETOR 3.0 or 4.0 data sets and try to reproduce the reported baselines
- Download the large MSLR-WEB30k data set and see how long it takes to run your favorite learning to rank algorithm on it
- Read Chapter 9 (Ranking) of *Foundations of Machine Learning* by Mohri, Rostamizadeh and Talwalkar. Solve as many exercises as you can!
- Read Chapter 11 (Learning to rank) of *Boosting: Foundations and Algorithms* by Freund and Schapire. Solve as many exercises as you can!

Thanks

Thank you for your attention!

Questions/comments: tewaria@umich.edu