Ranking with Boosted Decision Trees

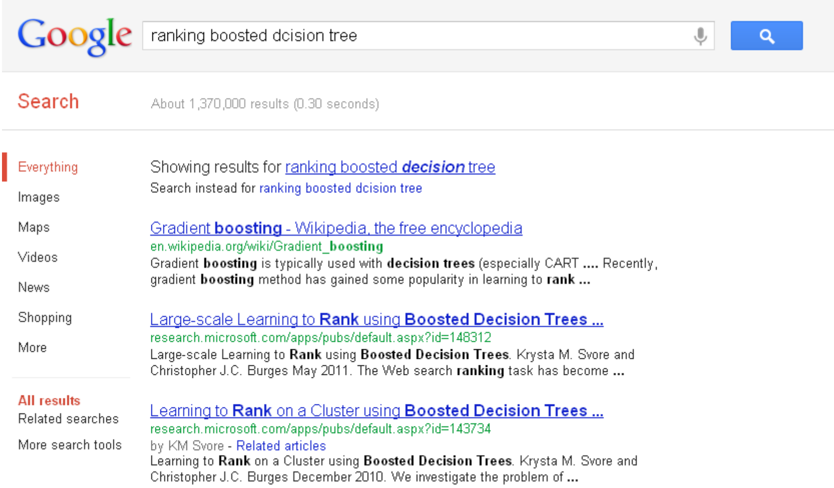
Seminar Information Retrieval
Dozentin: Dr. Karin Haenelt

Schigehiko Schamoni

Ruprecht-Karls-Universität Heidelberg
Institut für Computerlinguistik

January 16, 2012

Example: Web Search



Web Search Features

Technology features of modern web search engines:

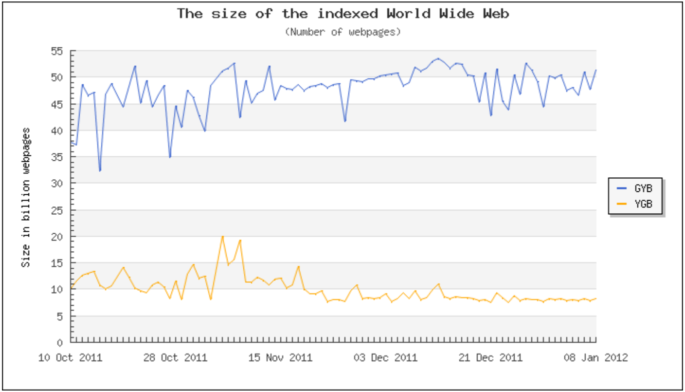
- Estimation of hit counts
- Can index many pagess
- Very fast
- Automatic spelling correction
- Preview of data
- Sophisticated ranking of results
- ...

Web Search Features

Technology features of modern web search engines:

- Estimation of hit counts
- Can index many pagess
- Very fast
- Automatic spelling correction
- Preview of data
- Sophisticated ranking of results $\leftarrow$ *Topic of this talk!*
- ...

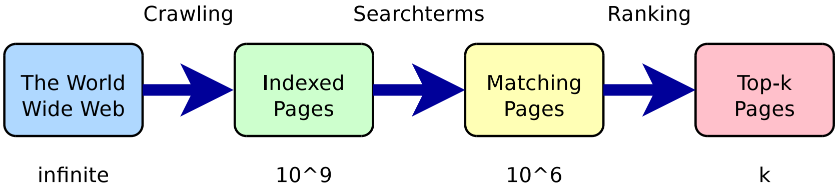
What is the Size of the Web?



From `http://www.worldwidewebsize.com/`, accessed 08.1.2012

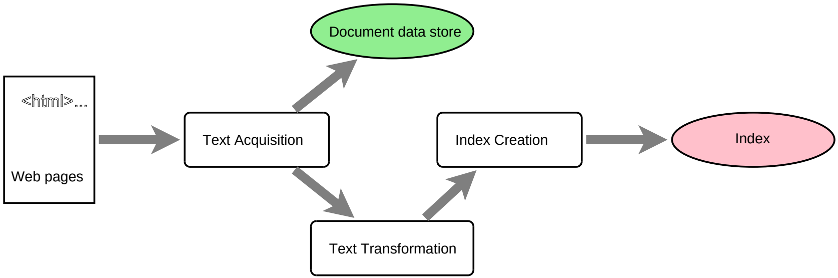
Special algorithms are needed to handle this amount of information.

Web Scale Information Retrieval



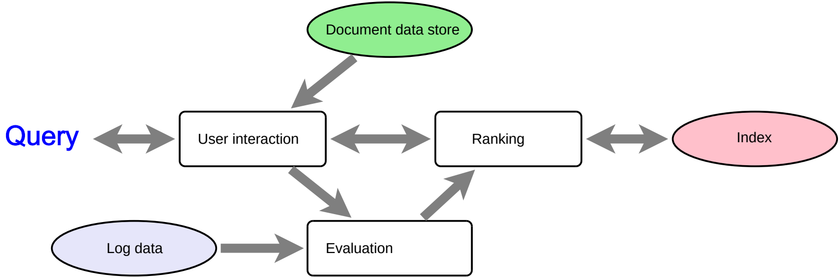
The "retrieval pipeline" must reduce the number of pages significantly!

Details of a Web Search Engine: Indexing



Components of the Indexing part of a search engine (Croft et al., 2010).

Details of a Web Search Engine: Querying



Components of the Querying part of a search engine (Croft et al., 2010).

The most important element in the whole querying-process is *ranking*.

Model Types for Information Retrieval

Classification of model types for Information Retrieval:

1. Set-theoretic models, e.g.
   - boolean models
   - extended boolean models
2. Algebraic models, e.g.
   - vector space model
   - latent semantic indexing
3. Probabilistic models, e.g.
   - probabilistic relevance (BM25)
   - language models

Relevance and Ranking

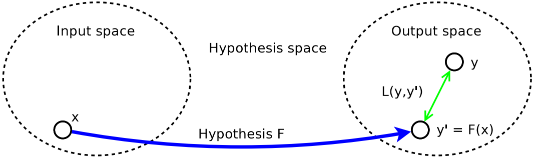
IR Models generate different values describing the relationship between a search query and the target document, e.g. "similarity".
This value expresses the relevance of a document w.r.t. to the query and induces a ranking of retrieval results.

Some important measures we heard of in this seminar[1]:

- (Normalized) term-frequency
- (Normalized) term-weight
- Inverse document frequency
- Cosine similarity (vector model)
- Retrieval status value (probabilistic model)

[1] see http://kontext.fraunhofer.de/haenelt/kurs/InfoRet/

Learning to Rank



From: LIU (2010), *Learning to Rank for Information Retrieval*.

Basic Idea of Machine Learning:

- Hypothesis F transforms input object x to output object $y' = F(x)$.
- $L(y, y')$ is the *loss*, i.e. the difference between the predicted y' and the target y.
- "Learning" process: find the hypothesis minimizing L by tuning F.

Learning a ranking function with machine learning techniques:
Learning to Rank (LTR)

Features for Learning

To learn a ranking function, each query-document pair is represented by a vector of features of three categories:

1. Features modelling web document, d (*static* features): inbound links, PAGE rank, document length, etc.
2. Features modelling query-document relationship (*dynamic* features): frequency of search terms in document, cosine similarity, etc.
3. Features modelling user query, q: number of words in query, query classification, etc.

In supervised training, the ranking function is learned using vectors of known ranking levels.

Example: Features for AltaVista (2002)

A0 - A4	anchor text score per term
W0 - W4	term weights
L0 - L4	first occurrence location (encodes hostname and title match)
SP	spam index: logistic regression of 85 spam filter variables (against relevance scores)
F0 - F4	term occurrence frequency within document
DCLN	document length (tokens)
ER	Eigenrank
HB	Extra-host unique inlink count
ERHB	ER*HB
A0W0 etc.	A0*W0
QA	Site factor - logistic regression of 5 site link and url count ratios
SPN	Proximity
FF	family friendly rating
UD	url depth

From: J. Pedersen (2008), The Machine Learned Ranking Story

Algorithms for Ranking

- Support Vector Machines (Vapnik, 1995)
  - Very good classifier
  - Can be adapted to ranking and multiclass problems
- Neural Nets
  - RankNet (Burges et al., 2006)
- Tree Ensembles
  - Random Forests (Breiman and Schapire, 2001)
  - Boosted Decision Trees
    - Multiple Additive Regression Trees (Friedman, 1999)
    - LambdaMART (Burges, 2010)
    - Used by AltaVista, Yahoo!, Bing, Yandex, ...

All top teams of the *Yahoo! Learning to Rank Challenge (2010)* used combinations of Tree Ensembles!

Yahoo! Learning to Rank Challenge

- Yahoo! Webscope dataset (Chapelle and Chang, 2011): 36,251 queries, 883 k documents, 700 features, 5 ranking levels
  - set-1:
    - 473,134 feature vectors
    - 519 features
    - 19,944 queries
  - set-2:
    - 34,815 feature vectors
    - 596 features
    - 1,266 queries
- Winner used a combination of 12 models:
  - 8 Tree Ensembles (LambdaMART)
  - 2 Tree Ensembles (Additive Regression Trees)
  - 2 Neural Nets

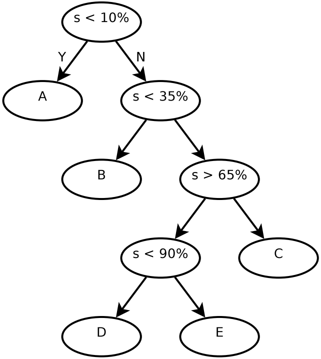
Decision Trees

Characteristics of a tree:

- Graph based model
- Consists of a root, nodes, and leaves

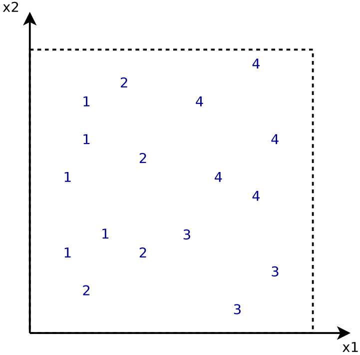
Advantages:

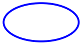
- Simple to understand and interpret
- *White box* model
- Can be combined with other techniques



Decision trees are basic learners for machine learning, e.g. *classification* or *regression trees*.

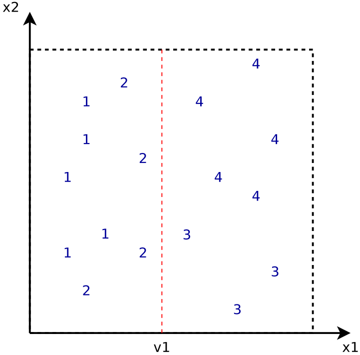
Learning a Regression Tree (I)

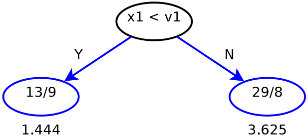




Consider a 2-dimensional space consisting of data points of the indicated values. We start with an empty root node (blue).

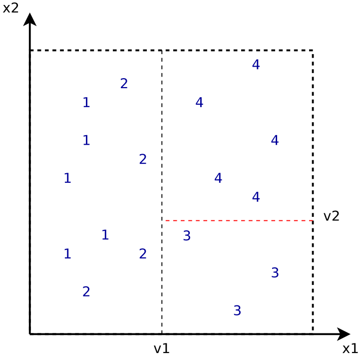
Learning a Regression Tree (II)

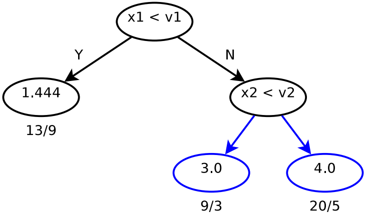




The algorithm searches for split variables and split points, x_1 and v_1, that predict values minimizing the predicted error, e.g. $\sum(y_i - f(x_i))^2$.

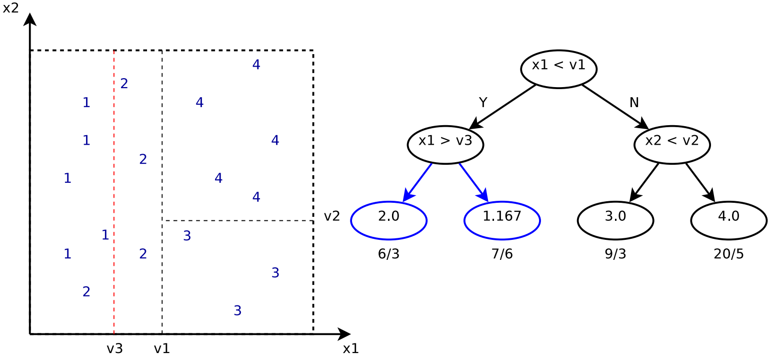
Learning a Regression Tree (III)





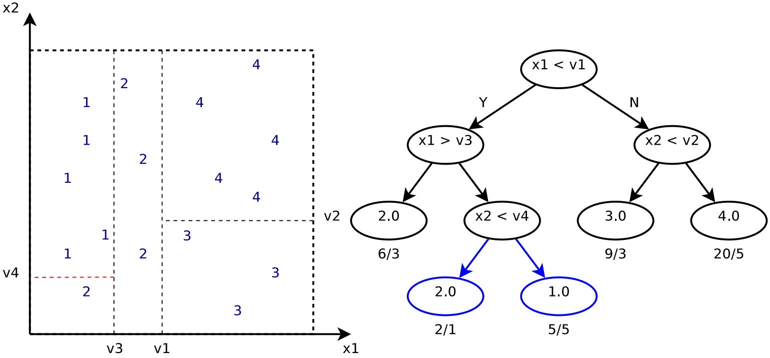
Here we examine the right side first: find a split variable and a split value that minimize the predicted error, i.e. x_2 and v_2.

Learning a Regression Tree (IV)



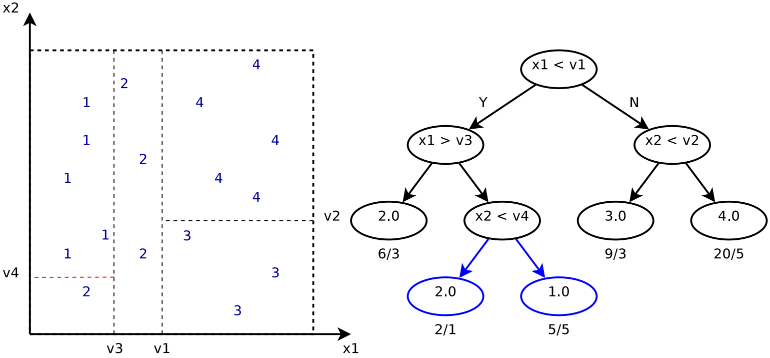
Now to the left side: Again, find a split variable and a split value that minimize the predicted error, i.e. x_1 and v_3.

Learning a Regression Tree (V)



Once again, find a split variable and a split value that minimize the predicted error, here x_2 and v_4.

Learning a Regression Tree (V)



Once again, find a split variable and a split value that minimize the predicted error, here x_2 and v_4. The tree perfectly fits the data! Problem?

Formal Definition of a Decision Tree

A decision tree partitions the parameter space into disjoint regions R_k, $k \in \{1, ..., K\}$, $K =$ number of leaves. Formally, the regression model (1) predicts a value using a constant γ_k for each region R_k:

$$T(\mathbf{x}; \Theta) = \sum_{k=1}^{K} \gamma_k 1(\mathbf{x} \in R_k) \tag{1}$$

$\Theta = \{R_k, \gamma_k\}_1^K$ describes the model parameters, $1(\cdot)$ is the *characteristic function* (1 if argument is true, 0 otherwise), and $\hat{\gamma}_k = \text{mean}(y_i | \mathbf{x}_i \in R_k)$. Optimal parameters $\hat{\Theta}$ are found minimizing the empirical risk:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in R_k} L(y_i, \gamma_k) \tag{2}$$

The combinatorial optimization problem (2) is usually split into two parts: (i) *finding* R_k and (ii) *finding* γ_k *given* R_k.

Boosting

Idea

Combine multiple weak learners to build a strong learner.
A weak learner is a learner with an error rate slightly better than random guessing. A strong learner is a learner with high accuracy.

Approach:

- Apply a weak learner to iteratively modified data
- Generate a sequence of learners
- For classification tasks: use majority vote
- For regression tasks: build weighted values

Function Estimation

Find a function $F^*(\mathbf{x})$ that maps $\mathbf{x}$ to y, s.t. the expected value of some loss function $L(y, F(\mathbf{x}))$ is minimized:

$$F^*(\mathbf{x}) = \arg\min_{F(\mathbf{x})} \mathbb{E}_{y,\mathbf{x}} \left[L(y, F(\mathbf{x})) \right]$$

Boosting approximates $F^*(\mathbf{x})$ by an additive expansion

$$F(\mathbf{x}) = \sum_{m=1}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m)$$

where $h(\mathbf{x}; \mathbf{a})$ are simple functions of $\mathbf{x}$ with parameters $\mathbf{a} = \{a_1, a_2, ..., a_n\}$ defining the function h, and β are expansion coefficients.

Finding Parameters

Expansion coefficients $\{\beta_m\}_0^M$ and the function parameters $\{\mathbf{a}_m\}_0^M$ are iteratively fit to the training data:

1. Set $F_0(\mathbf{x})$ to initial guess
2. For each $m = 1, 2..., M$

$$(\beta_m, \mathbf{a}_m) = \arg\min_{\beta,\mathbf{a}} \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i, \mathbf{a})) \tag{3}$$

and

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m) \tag{4}$$

Gradient Boosting

Gradient boosting approximately solves (3) for differentiable loss functions:

1. Fit the function $h(\mathbf{x}; \mathbf{a})$ by least squares

$$\mathbf{a}_m = \arg\min_{\mathbf{a}} \sum_{i=1}^{N} [\tilde{y}_{im} - h(\mathbf{x}_i, \mathbf{a})]^2 \tag{5}$$

   to the "pseudo"-residuals

$$\tilde{y}_{im} = - \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})} \tag{6}$$

2. Given $h(\mathbf{x}; \mathbf{a}_m)$, the β_m are

$$\beta_m = \arg\min \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(\mathbf{x}_i; \mathbf{a}_m)) \tag{7}$$

$\Rightarrow$ Gradient boosting simplifies the problem to least squares (5).

Gradient Tree Boosting

Gradient tree boosting applies this approach on functions $h(\mathbf{x}; \mathbf{a})$ representing K-terminal node regression trees.

$$h(\mathbf{x}; \{R_{km}\}_1^K) = \sum_{k=1}^{K} \bar{y}_{km} 1(\mathbf{x} \in R_{km}) \tag{8}$$

With $\bar{y}_{km} = \text{mean}_{\mathbf{x}_i \in R_{km}}(\tilde{y}_{im})$ the tree (8) predicts a constant value $\bar{y}_{km}$ in region R_{km}. Equation (7) becomes a prediction of a γ_{km} for each R_{km}:

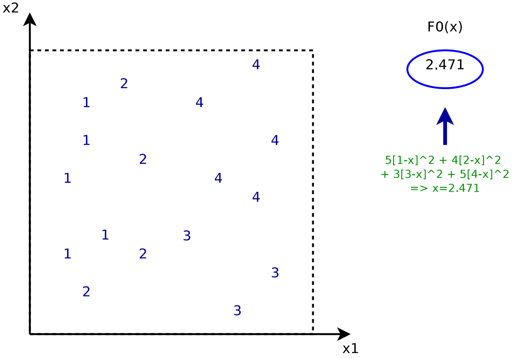
$$\gamma_{km} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{km}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma) \tag{9}$$

The approximation for F in stage m is then:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \cdot \gamma_{km} 1(\mathbf{x}_i \in R_{km}) \tag{10}$$

The parameter η controls the *learning rate* of the procedure.

Learning Boosted Regression Trees (I)



First, learn the most simple predictor that predicts a constant value minimizing the error for all training data.

Calculating Optimal Leaf Value for F_0

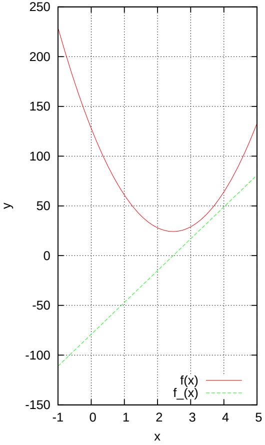
Recall the exp. coefficient: $\gamma_{km} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{km}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$

- Quadratic loss for the leaf (red):

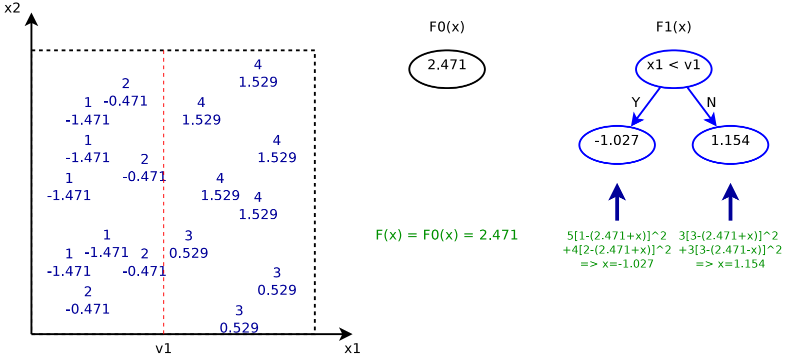
$$f(x) = 5 \cdot (1-x)^2 + 4 \cdot (2-x)^2 + 3 \cdot (3-x)^2 + 5 \cdot (4-x)^2$$

- $f(x)$ is quadratic, *convex*
  $\Rightarrow$ Optimum at $f'(x) = 0$ (green)

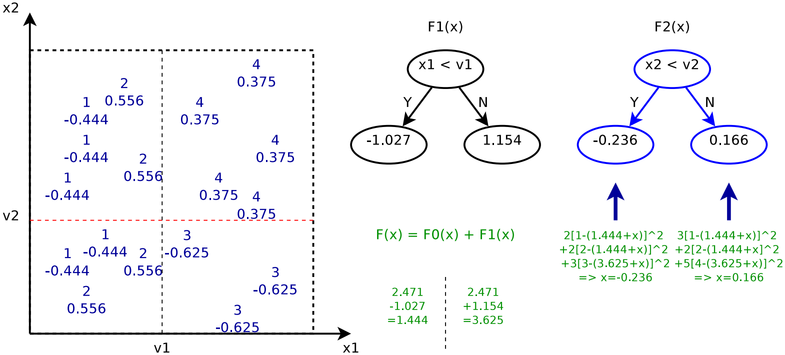
$$\frac{\partial f(x)}{\partial x} = 5 \cdot (-2+2x) + 4 \cdot (-4+2x)^2 + 3 \cdot (-6+2x)^2 + 5 \cdot (-8+2x)^2 = -84 + 34x = 32(x - 2.471)$$



Learning Boosted Regression Trees (II)

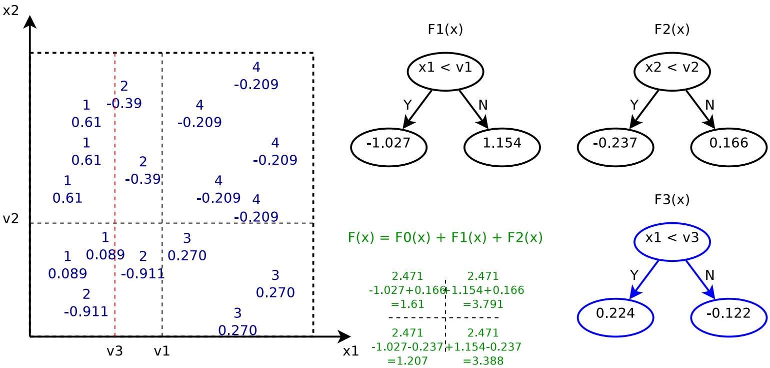


Split root node based on least squares criterion to build a tree predicting the “pseudo”-residuals.



In the next stage, another tree is created to fit the actual "pseudo"-residuals predicted by the first tree.

Learning Boosted Regression Trees (IV)



This is iteratively continued: in each stage, the algorithm builds a new tree based on the "pseudo"-residuals predicted by the previous tree ensemble.

Multiple Additive Regression Trees (MART)

Algorithm 1 Multiple Additive Regression Trees.

Initialize $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
for $m = 1, ..., M$ **do**
  for $i = 1, ..., N$ **do**
    $\tilde{y}_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x}) = F_{m-1}(\mathbf{x})}$
  end for
  $\{R_{km}\}_{k=1}^{K}$ // Fit a regression tree to targets $\tilde{y}_{im}$
  for $k = 1, ..., K_m$ **do**
    $\gamma_{km} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
  end for
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} 1(\mathbf{x}_i \in R_{km})$
end for
Return $F_M(\mathbf{x})$

RankNet Model

- Differentiable function of the model parameters, typically neural nets
- RankNet maps a feature vector $\mathbf{x}$ to a value $f(\mathbf{x};\mathbf{w})$
- Learned probabilities URL $U_i \succ U_j$ modelled via a sigmoid function

$$P_{ij} \equiv P(U_i \succ U_j) \equiv \frac{1}{1+e^{-\sigma(s_i-s_j)}}$$

  with $s_i = f(\mathbf{x}_i)$, $s_j = f(\mathbf{x}_j)$
- Cost function calculates cross entropy:

$$C = -\bar{P}_{ij}\log P_{ij} - (1-\bar{P}_{ij})\log(1-P_{ij})$$

  P_{ij} is the model probability, $\bar{P}_{ij}$ is the known probability from training.

RankNet Algorithm

Algorithm 2 RankNet Training.

```
Initialize F_0(x) = arg min_γ Σ_{i=1}^{N} L(y_i, γ)
for each query q ∈ Q do
    for each pair of URLs U_i, U_j with different label do
        s_i = f(x_i), s_j = f(x_j)
        Estimate cost C
        Update model scores w_k → w_k − η ∂C/∂w_k
    end for
end for
Return w
```

RankNet λ's

The crucial part is the update:

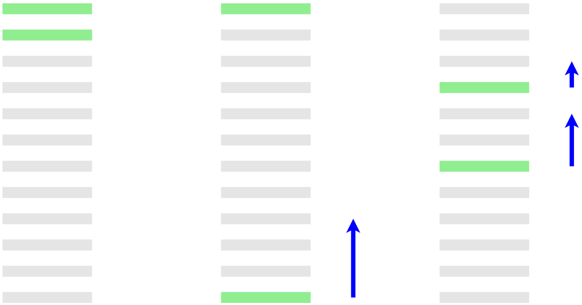
$$\frac{\partial C}{\partial w_k} = \frac{\partial C}{\partial s_i}\frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j}\frac{\partial s_j}{\partial w_k} = \lambda_{ij}\left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}\right)$$

- λ_{ij} describes the desired change of scores for the pair U_i and U_j
- The sum over all λ_{ij}'s and λ_{ji}'s of a given query-document vector x_i w.r.t. all other differently labelled documents is

$$\lambda_i = \sum_{j:\{i,j\}\in I} \lambda_{ij} - \sum_{k:\{k,i\}\in I} \lambda_{ki}$$

- λ_i is (kind of) a gradient of the pairwise loss of vector $\mathbf{x}_i$.

RankNet Example

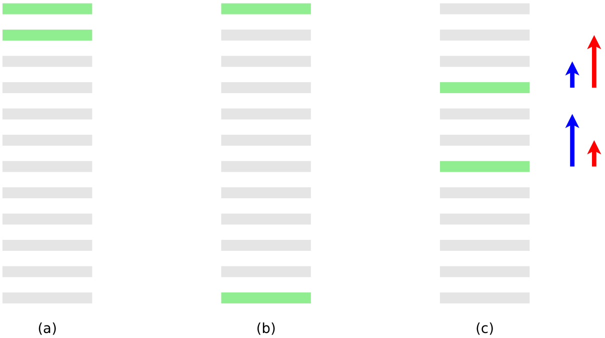


(a) (b) (c)

(a) is the perfect ranking, (b) is a ranking with 10 pairwise errors, (c) is a ranking with 8 pairwise errors. Each blue arrow represents the λ_i for each query-document vector $\mathbf{x}_i$.

From: BURGES (2010), *From RankNet to LambdaRank to LambdaMART: An Overview.*

LambdaRank Example



Problem: RankNet is based on pairwise error, while modern IR measures emphasize higher ranking positions. Red arrows show better λ's for modern IR measures.

From: Burges (2010), *From RankNet to LambdaRank to LambdaMART: An Overview.*

From RankNet to LambdaRank to LambdaMART

From RankNet to LambdaRank:

- Multiply λ's with $|\Delta Z|$, i.e. the difference of an IR measure when U_i and U_j are swapped
- E.g. $|\Delta\,\mathrm{NDCG}\,|$ is the change in NDCG when swapping U_i and U_j:

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} |\Delta\,\mathrm{NDCG}\,|$$

From LambdaRank to LambdaMART:

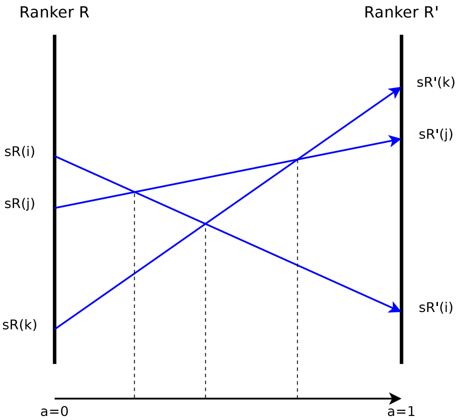
- LambdaRank models gradients
- MART works on gradients
- Combine both to get *LambdaMART*:
  $\Rightarrow$ MART with specified gradients and Newton step

LambdaMART Algorithm

Algorithm 3 LambdaMART.

for $i = 0, ..., N$ **do**
  $F_0(\mathbf{x}_i) = \text{BaseModel}(\mathbf{x}_i)$ // Set to 0 for empty BaseModel
end for
for $m = 1, ..., M$ **do**
  for $i = 0, ..., N$ **do**
    $y_i = \lambda_i$ // Calculate λ-gradient
    $w_i = \frac{\partial y_i}{\partial F_{k-1}(\mathbf{x}_i)}$ // Calculate derivative of gradient for $\mathbf{x}_i$
  end for
  $\{R_{km}\}_{k=1}^{K}$ // Create K-leaf tree on $\{\mathbf{x}_i, y_i\}$
  $\gamma_{km} = \frac{\sum_{x_i \in R_{km}} y_i}{\sum_{x_i \in R_{km}} w_i}$ // Assign leaf values
  $F_m(\mathbf{x}_i) = F_{m-1}(\mathbf{x}_i) + \eta \sum_k \gamma_{km} 1(\mathbf{x}_i \; in R_{km})$
end for

Optimally combine Rankers



From: Wu et al. (2008), *Ranking, Boosting, and Model Adaptation*.

- Linearly combine rankers: $(1-\alpha)R(\mathbf{x}_i) + \alpha R'(\mathbf{x}_i)$
- Let α go from 0 to 1:
  - Score changes only at the intersections
  - Enumerate all α for which pairs swap position
  - Calculate desired IR measure (e.g. NDCG)
- Select the α giving best scores

Solution can be found analytically, or approximated by Boosting or a LambdaRank approach.

References I

BREIMAN, LEO and E. SCHAPIRE (2001). *Random forests*. In *Machine Learning*, pp. 5–32.

BURGES, CHRISTOPHER J. C. (2010). *From RankNet to LambdaRank to LambdaMART: An Overview*.

BURGES, CHRISTOPHER J. C., R. RAGNO and Q. V. LE (2006). *Learning to Rank with Nonsmooth Cost Functions.*. In SCHÖLKOPF, BERNHARD, J. PLATT and T. HOFFMAN, eds.: *NIPS*, pp. 193–200. MIT Press.

CHAPELLE, OLIVIER and Y. CHANG (2011). *Yahoo! Learning to Rank Challenge Overview.*. Journal of Machine Learning Research - Proceedings Track, 14:1–24.

CROFT, W.B., D. METZLER and T. STROHMANN (2010). *Search Engines: Information Retrieval in Practice*. Pearson, London, England.

References II

FRIEDMAN, JEROME H. (1999). *Greedy Function Approximation: A Gradient Boosting Machine*. Annals of Statistics, 29(5):1189–1232.

GANJISAFFAR, YASSER (2011). *Tree Ensembles for Learning to Rank*. PhD thesis, University of California, Irvine.

HASTIE, TREVOR, R. TIBSHIRANI and J. FRIEDMAN (2002). *The Elements of Statistical Learning*. Springer, New York.

LIU, TIE-YAN (2010). *Learning to Rank for Information Retrieval.*. Springer-Verlag New York Inc.

MANNING, CHRISTOPHER D., P. RAGHAVAN and H. SCHÜTZE (2008). *Introduction to Information Retrieval*. Cambridge University Press.

VAPNIK, VLADIMIR N. (1995). *The Nature of Statistical Learning Theory*. Springer New York Inc., New York, NY, USA.

WU, QIANG, C. J. C. BURGES, K. M. SVORE and J. GAO (2008). *Ranking, Boosting, and Model Adaptation*.