CS 589 Fall 2021 Lecture 5

Learning to rank

Monday 6:30-9:00
Babbio 122

All zoom links in Canvas
Slides adapted from Stanford CS276

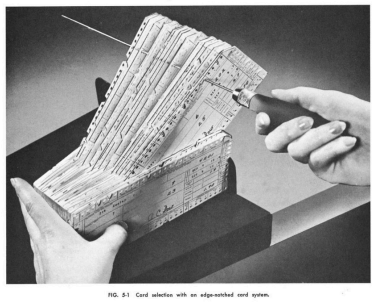
photo: https://www.scubedstudios.com/information-retrieval/

A Brief History of IR

300 BC

Callimachus: the first library catalog

1950s



Punch cards, searching at 600 cards/min

1958 Cranfield evaluation methodology; word-based indexing

1960s building IR systems on computers; relevance feedback

1970s TF-IDF; probability ranking principle

1980s TREC; **learning to rank**; latent semantic indexing

1990 - now web search; supporting natural language queries;

Review of Lecture 1-3: Parameter estimation

- Parameter estimation:
  - TF-IDF, RSJ: no parameter
  - BM25: use pre-defined values b=0.75, k1 in [1.2, 2.0]

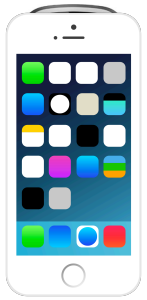
$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

  - Language model-based retrieval model
    - Dirichlet smoothing: Newton's 2nd order method, leave-one-out
    - JM smoothing: EM algorithm

How to make use of user clicks and user information, e.g., age, gender, to help improve the retrieval performance?

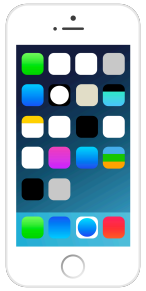
Review of Lecture 1-3: Parameter estimation

query = “best phone”



$400, 20G, Nokia

$500, 30G, Nokia



$600, 40G, iphone

session 2

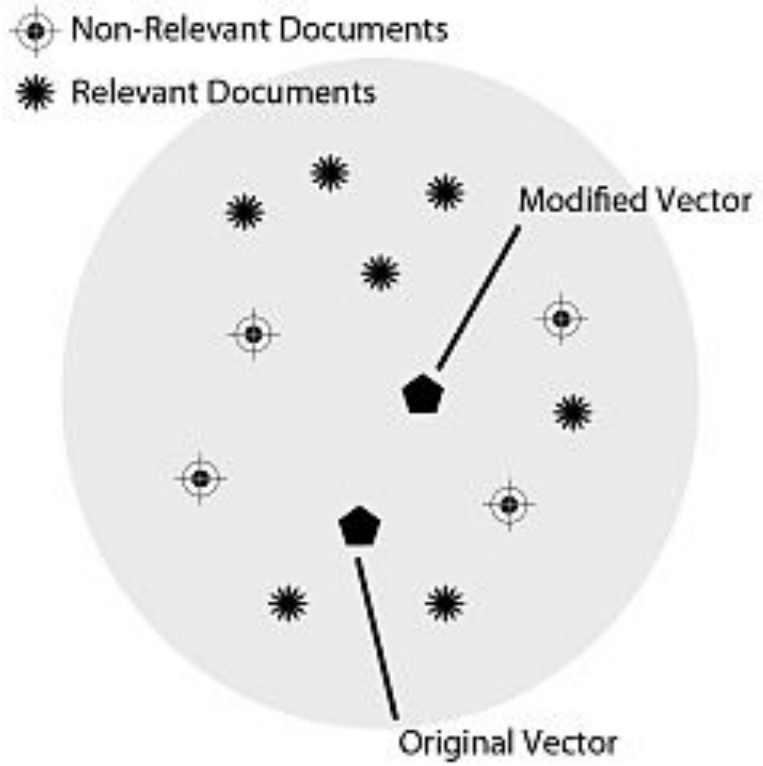
observed click

Relevance feedback with the Rocchio's model

$$q_F = \alpha q + \frac{\beta}{|D_r|} \sum_{d_r \in D_r} d_r - \frac{\gamma}{|D_n|} \sum_{d_n \in D_n} d_n$$

rel docs **non-rel docs**

beta >> gamma



Pop quiz (Inverted Index)

1. Which of the following are prefix encodings, i.e., there does NOT exists a sequence such that there are more than one ways to decode it?

A. {a: 1, b:11, c:0}

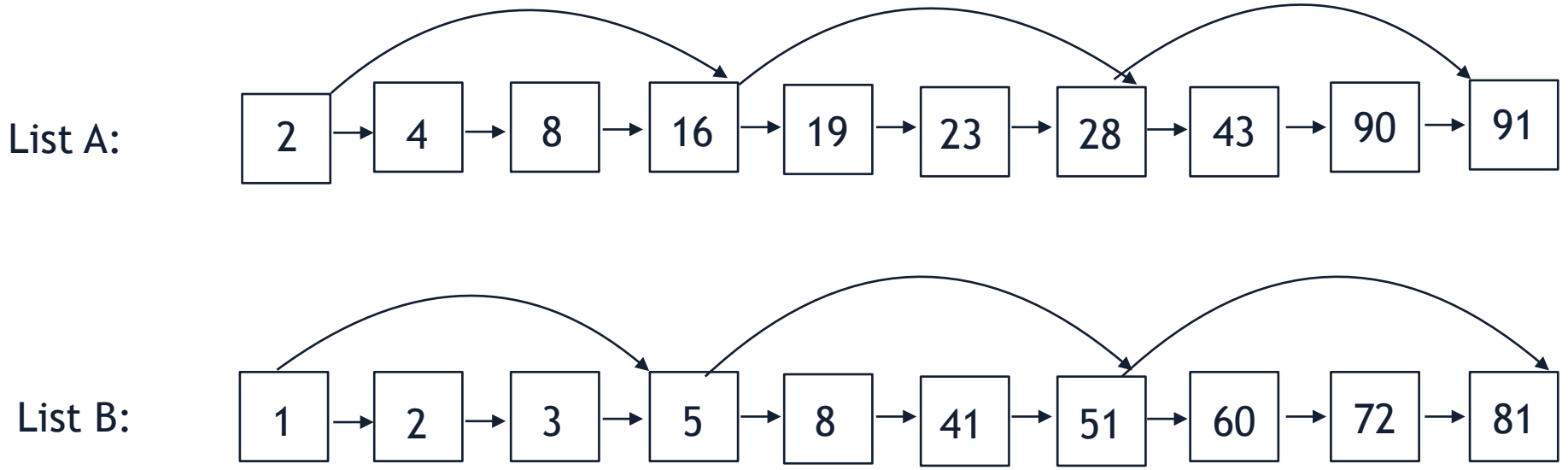
B. {a: 0, b: 10, b: 110, c:111}

C. {a: 0, b: 100, c: 101, d: 11}

D. {a: 0, b: 100, c: 10, d: 11}

Pop quiz (Inverted Index)

2. When we use the following two pointer algorithm to merge the following posting lists, select all documents that will be skipped. Can you implement the two pointer algorithm with Python?



Pop quiz (Inverted Index)

3. What is the gamma code of 14?

Background: machine learning for diabetes prediction

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
```

Output Statistics Query 768 rows returned

	PREGNANCIES	GLUCOSE	BLOODPRESSURE	INSULIN	BMI	AGE	OUTCOME
1	6	148	72	0	33.6	50	1
2	1	85	66	0	26.6	31	0
3	8	183	64	0	23.3	32	1
4	1	89	66	94	28.1	21	0
5	0	137	40	168	43.1	33	1
6	5	116	74	0	25.6	30	0
7	3	78	50	88	31	26	1
8	10	115	0	0	35.3	29	0
9	2	197	70	543	30.5	53	1
10	8	125	96	0	0	54	1
11	4	110	92	0	37.6	30	0

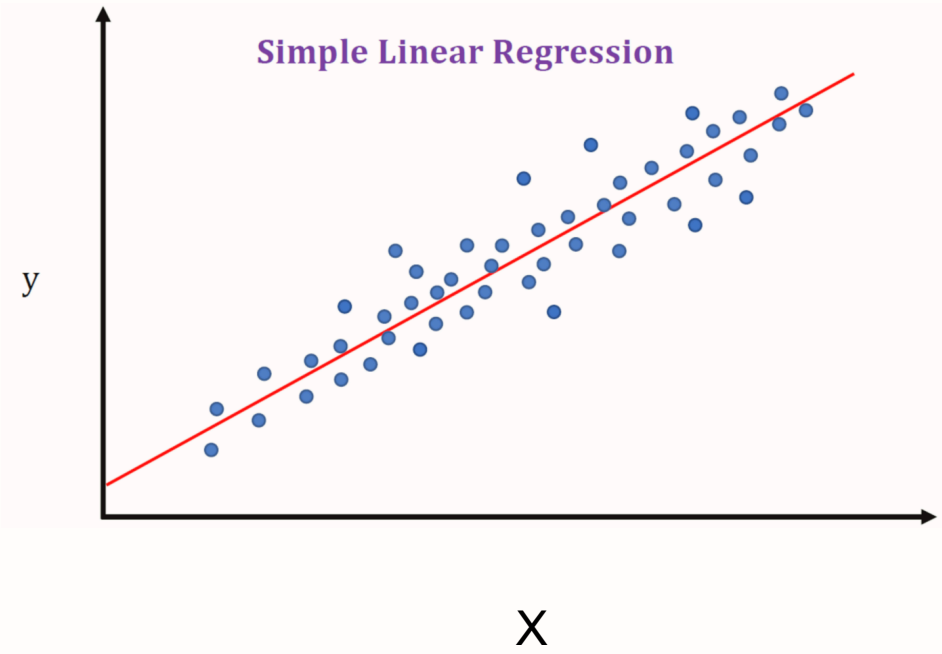
Input features: X Output label: y

Background: linear regression

- The relationship between input feature and output label is modeled using a linear predictor

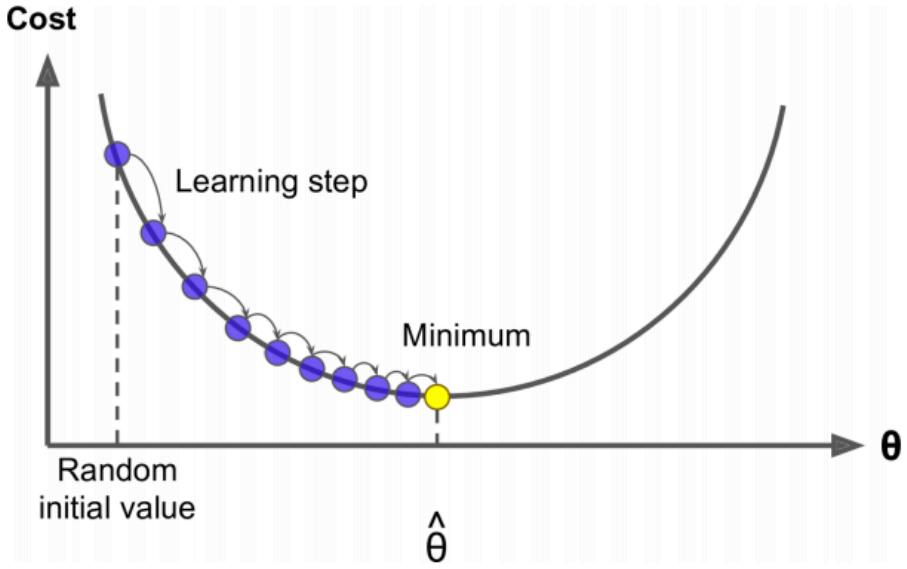
$$= \frac{1}{n}\sum_{i=0}^{n}(y_i - (w^T x_i + b))^2$$

- Disadvantage: cannot model non-linear relationships



Background: gradient descent (optimization method)

- We have a cost function $L(\theta)$ we want to minimize
- There are a list of optimization algorithm (for both convex and non-convex objectives), the most popular one is gradient descent (first-order optimization)
- At each iteration, GD calculate gradient of $L(\theta)$, then take small step in direction of negative gradient



$$\min_{\theta} L(\theta)$$

$$\theta^t = \theta^{t-1} - \lambda \frac{\partial L}{\partial \theta}$$

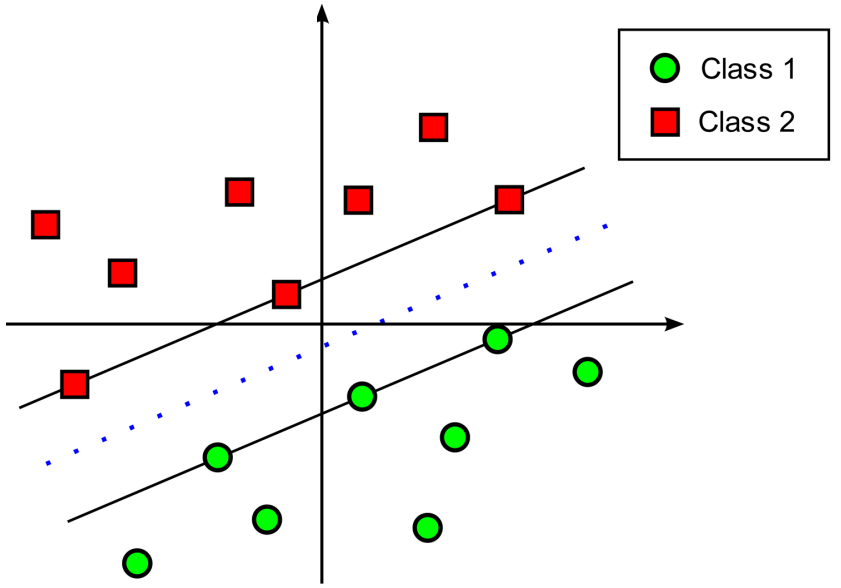
Background: support vector machine (SVM)

- Hard-margin: find the hyperplane that maximizes the margin between the *support vectors* (i.e., samples on the margin)

  Minimize $\|\mathbf{w}\|$ s.t. $y_i(\mathbf{w}^T x_i - b) \geq 1$

- Soft-margin: minimize hinge loss

$$\|\mathbf{w}\|^2 + \lambda \left[\frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i - b)\right) \right]$$

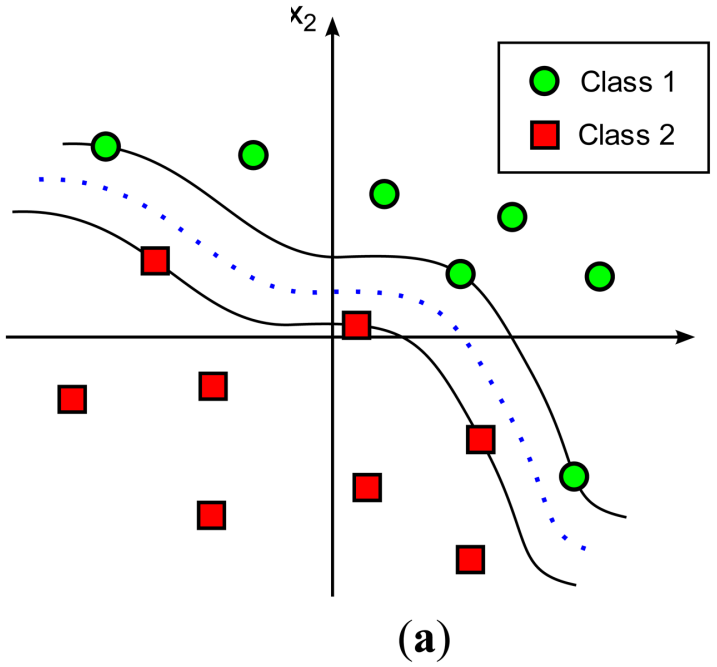


(b)

Background: support vector machine (SVM)

- Hard-margin: find the hyperplane that maximizes the margin between the *support vectors* (i.e., samples on the margin)

  Minimize $\|\mathbf{w}\|$ s.t. $y_i(\mathbf{w}^T x_i - b) \geq 1$



(a)

- Soft-margin: minimize hinge loss

$$\|\mathbf{w}\|^2 + \lambda \left[\frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i - b)\right) \right]$$

- Capture non-linearity with kernel (dual form):

Minimize $\sum_{i=1}^{n} c_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i c_i k(\mathbf{x}_i, \mathbf{x}_j) y_j c_j$ s.t. $\sum_{i=1}^{n} c_i y_i = 0$, and $0 \leq c_i \leq \frac{1}{2n\lambda}$ for all i.

Using machine learning to help improve IR

- Training features:
  - Retrieval model score: cosine similarity, IDF, BM25, proximity, pivoted document length normalization, PageRank, …
  - User demographic information: User age, gender
  - User search history

- Training labels
  - User clicks
  - Other user activities (e.g., purchase record, SAT clicks)

Machine learning for IR came in late

- This “good idea” has been actively researched – and actively deployed by major web search engines in the past 10 years, e.g., XGBoost [2016]
  - Modern supervised learning has been around for 30 years, e.g., SVM [1995], gradient boosting [2003]

- Why not earlier?
  - Limited training data
  - Poor machine learning techniques
  - Insufficient customization to IR problem
  - Not enough features for ML to show value
  - It was possible to tune parameters by hand (and people did)

Learning to rank benefits from web search

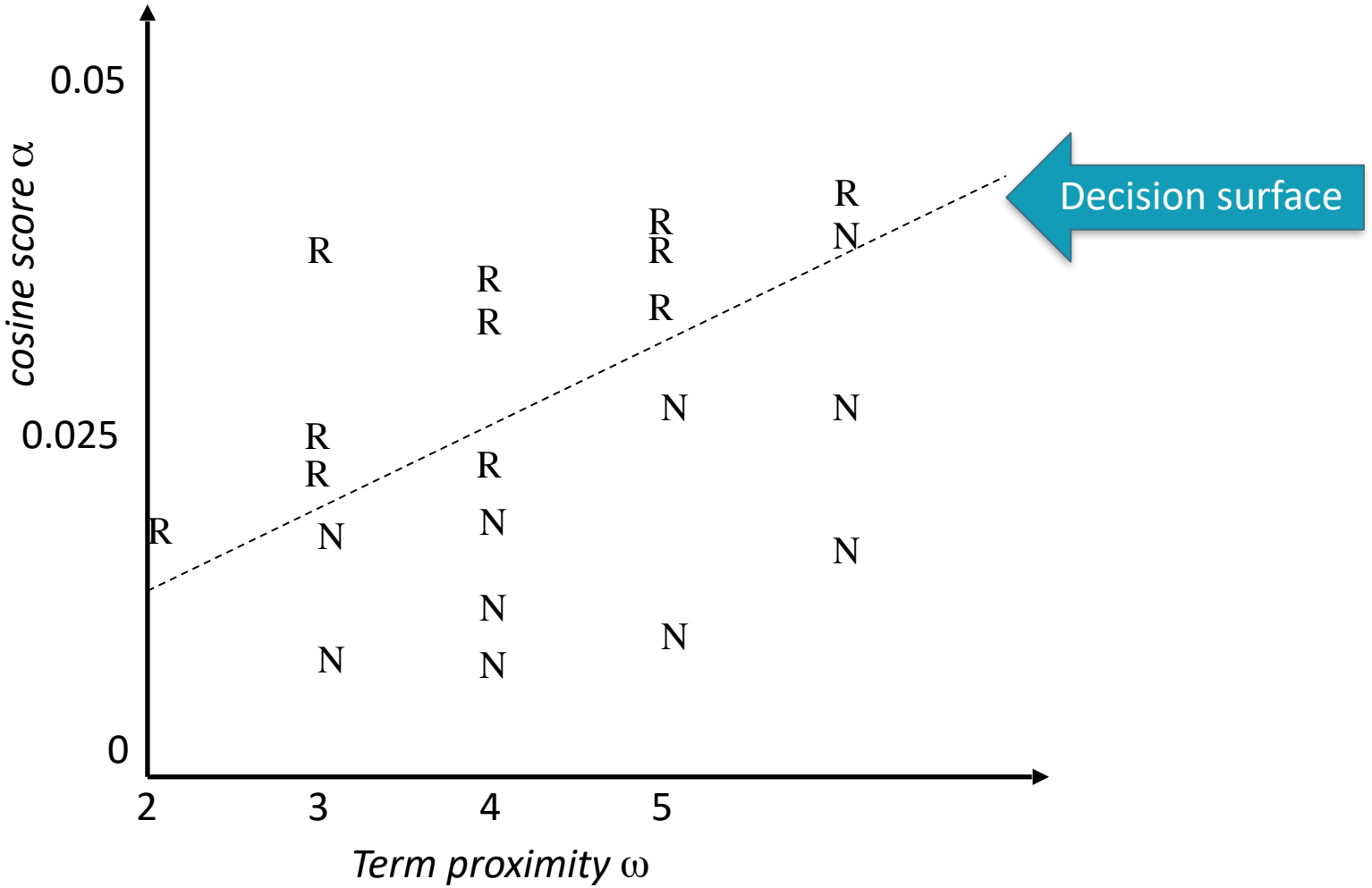
- Modern (web) systems use a great number of features:
    - Arbitrary useful features – not a single unified model
  - Log frequency of query word in anchor text?
  - Query word in color on page?
  - # of images on page?
  - # of (out) links on page?
  - PageRank of page?
  - URL length?
  - URL contains "~"?
  - Page edit recency?
  - Page loading speed
- The *New York Times* in 2008-06-03 quoted Amit Singhal as saying Google was using over 200 such features ("signals") – so it's sure to be over 500 today ☺

Learning to rank as binary classification

- For each (*q, d, r*) triples:
  - Output label y = Relevance judgment (binary)
  - Input feature **x** = (α, ω):
    - α is cosine similarity of the vector space model
    - ω is minimum query window size in the document
  - Training: apply supervised classification on **x** and y for training data
  - Prediction: use the predicted label for retrieval (1=relevant, 0=nonrelevant)

example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	*relevant*
Φ_2	37	penguin logo	0.02	4	*nonrelevant*
Φ_3	238	operating system	0.043	2	*relevant*
Φ_4	238	runtime environment	0.004	2	*nonrelevant*
Φ_5	1741	kernel layer	0.022	3	*relevant*
Φ_6	2094	device driver	0.03	2	*relevant*
Φ_7	3191	device driver	0.027	5	*nonrelevant*

Learning to rank as binary classification

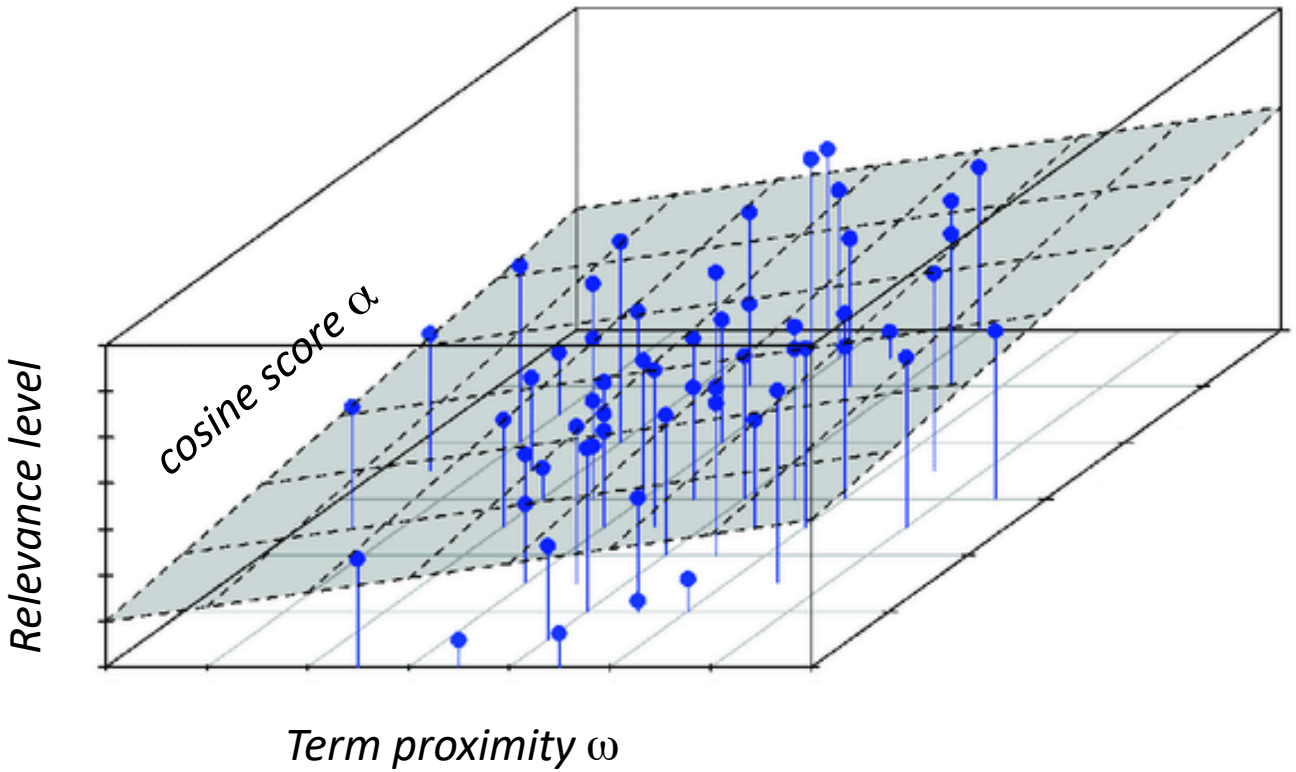


Learning to rank as regression

- For each (*q, d, r*) triples:
  - Output label y = Relevance judgment, multi-level
  - Input feature **x** = (α, ω):
    - α is the cosine similarity of the vector space model
    - ω is minimum query window size in the document
  - Training: apply linear regression on **x** and y
  - Ranking: use the predicted score for ranking

example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	1
Φ_2	37	penguin logo	0.02	4	2
Φ_3	238	operating system	0.043	2	1
Φ_4	238	runtime environment	0.004	2	3
Φ_5	1741	kernel layer	0.022	3	1
Φ_6	2094	device driver	0.03	2	3
Φ_7	3191	device driver	0.027	5	2

Learning to rank as regression



Linear model

Challenge in relevance judgment

- Exact judgment for relevance is challenging
  - Exact judgment is incomparable across sessions
  - The criterion for relevance judgment may drift over time
- Relative judgment is easier
  - i.e., for docID 2094 and 3191, only consider the “weak supervision” that 2094 is **more relevant** than 3191 (instead that 2094’s relevance is exactly 3)

example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	1
Φ_2	37	penguin logo	0.02	4	2
Φ_3	238	operating system	0.043	2	1
Φ_4	238	runtime environment	0.004	2	3
Φ_5	1741	kernel layer	0.022	3	1
Φ_6	2094	device driver	0.03	2	3
Φ_7	3191	device driver	0.027	5	2

Pairwise learning to rank

- Leveraging the relative relevance judgment, i.e., pairwise labels
- Loss function: cross entropy loss

$$\min_{\Theta} \sum_{q} \sum_{i,j \in q} -\mathbb{1}\left[r_i > r_j\right] \log P(d_i > d_j) - (1 - \mathbb{1}\left[r_i > r_j\right]) \log\left(1 - P(d_i > d_j)\right)$$

Pairwise objective

Probability that di is more relevant than dj

$$P(d_i > d_j) = \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \qquad s_i = wx_i + b$$

Ranking based on machine learning algorithms

- Support Vector Machines (Vapnik, 1995)
  - Adapted to ranking: Ranking SVM (Joachims 2002)
- Neural Nets: RankNet (Burges et al., 2006)
- Tree Ensembles
  - Random Forests (Breiman and Schapire, 2001)
  - Boosted Decision Trees
    - Multiple Additive Regression Trees (Friedman, 1999)
    - Gradient-boosted decision trees: LambdaMART (Burges, 2010)
    - Used by all search engines? AltaVista, Yahoo!, Bing, Yandex, …
- All top teams in the 2010 Yahoo! Learning to Rank Challenge used combinations with Tree Ensembles!

Yahoo! learning to rank challenge

Chapelle and Chang [2011]

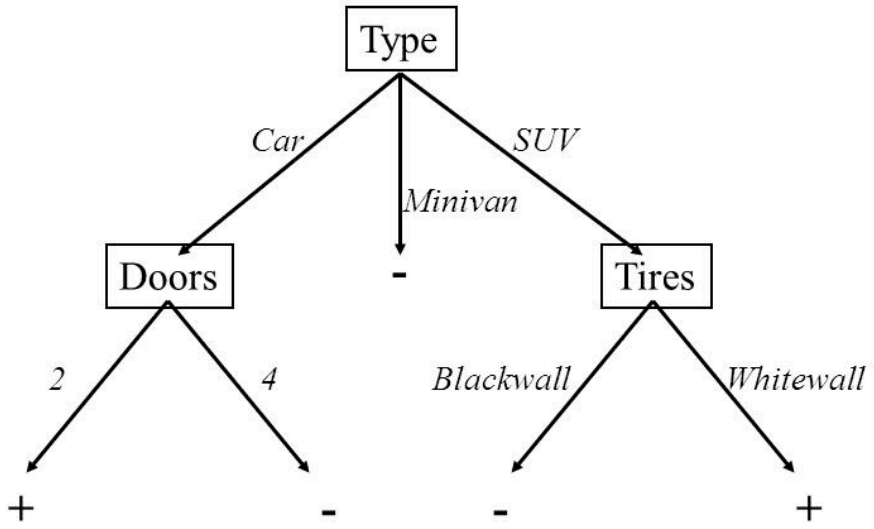
- Yahoo! Webscope dataset : 36,251 queries, 883k documents, 700 features, 5 ranking levels
  - Ratings: Perfect (navigational), Excellent, Good, Fair, Bad
  - Real web data
  - set-1: 473,134 feature vectors; 519 features; 19,944 queries
  - set-2: 34,815 feature vectors; 596 features; 1,266 queries
- Winner (Burges et al. 2011) was linear combo of 12 models:
  - 8 Tree Ensembles (LambdaMART)
  - 2 LambdaRank Neural Nets
  - 2 Logistic regression models

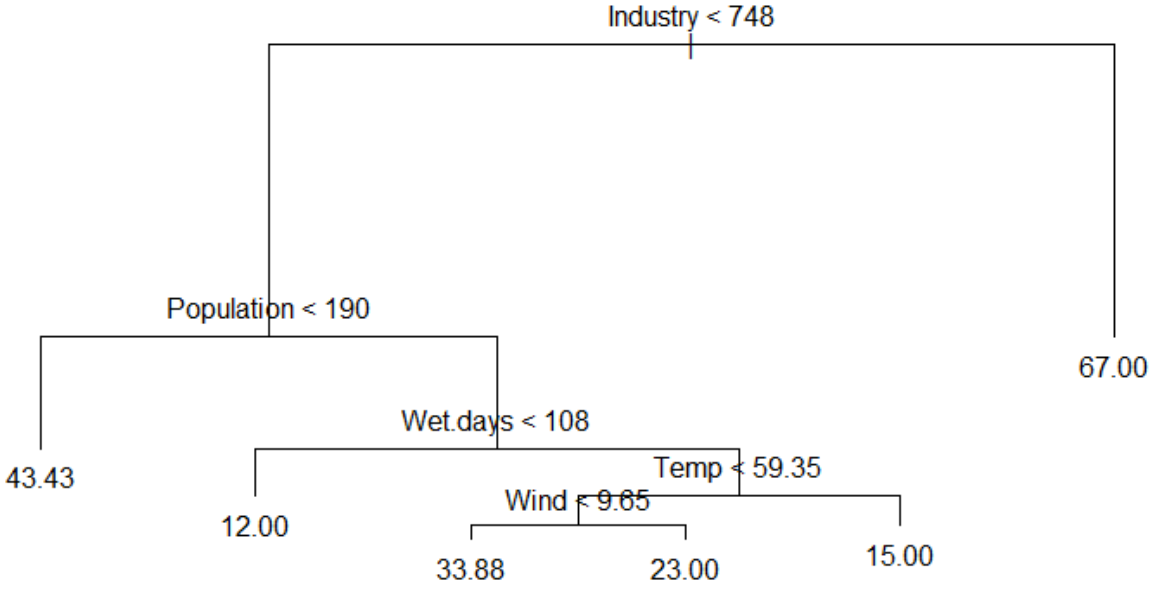
Yahoo! learning to rank challenge

- Goal was to validate learning to rank methods on a large, “real” web search problem
  - Previous work was mainly driven by LETOR datasets
    - Great as first public learning-to-rank data
    - Small: 10s of features, 100s of queries, 10k’s of docs
- Only feature vectors released
  - Not URLs, queries, nor feature descriptions
    - Wanting to keep privacy and proprietary info safe
  - But included web graph features, click features, page freshness and page classification features as well as text match features

Regression tree

- Regression tree are decision trees that can predict a value





decision tree

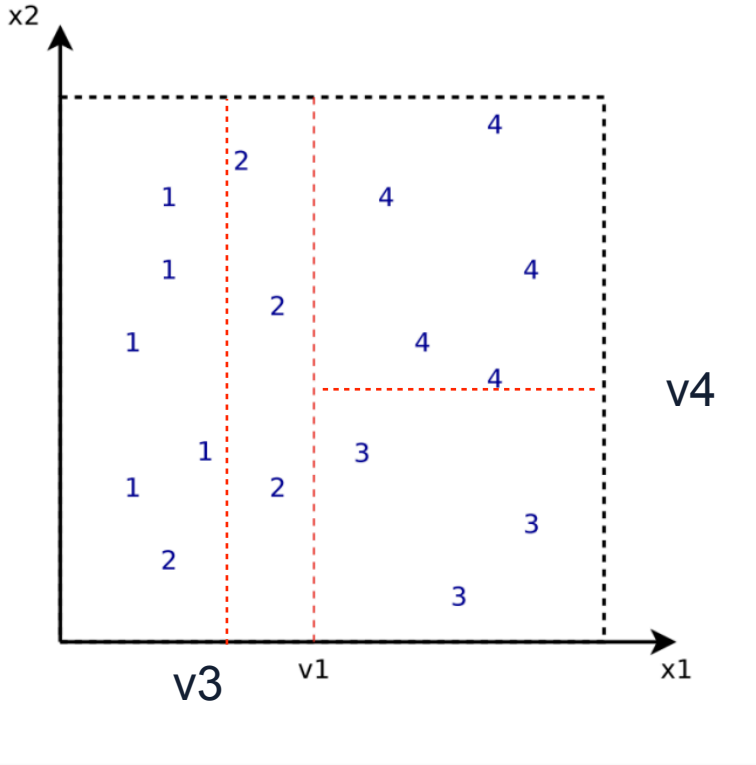
regression tree

Regression tree

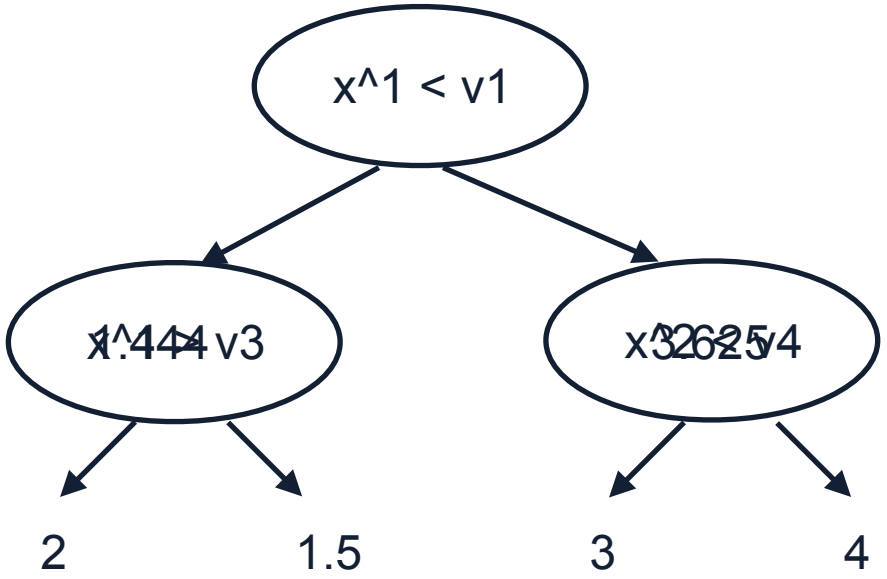
- Splitting criterion: Standard Deviation Reduction
  - Choose split value to minimize the variance (standard deviation) of the values in each child induced by split A (normally just a binary split for easy search):

- Termination: cutoff on SD or #examples or tree depth

Training a regression tree



$$\min_{\Theta} \sum_{i} (y_i - f(x_i; \Theta))^2$$



Boosting in machine learning (AdaBoost)

- **AdaBoost**: using the ensemble of multiple weak learners to build a high accuracy classifier
  - Weak learners = small decision trees (1-split decision stumps)
  - Weights for each learner and instance
  - Instances are weighed on probability it's mistaken
  - Learners are weighed on its accuracy

Input: ℓ, α, $\{(\mathbf{x}_i, y_i)\}$, $\mathbb{A}$
$H_0 = 0$
$\forall i : w_i = \frac{1}{n}$
for *t=0:T-1* **do**
  $h = \mathbb{A}(w_1, \mathbf{x}_1, y_1), \ldots, (w_n, \mathbf{x}_n, y_n)$
  $\epsilon = \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i$
  if $\epsilon < \frac{1}{2}$ **then**
    $\alpha = \frac{1}{2} \ln(\frac{1-\epsilon}{\epsilon})$
    $H_{t+1} = H_t + \alpha h$
    $\forall i : \quad w_i \leftarrow \frac{w_i e^{-\alpha h(\mathbf{x}_i) y_i}}{2\sqrt{\epsilon(1-\epsilon)}}$
  else
    return (H_t)
  end
  return (H_T)
end

Error of the weak learner

Towards gradient boostings

- Linear regression finds the **parameter** that minimizes the square loss:

$$w = argmin_w L(y, w^T \mathbf{x} + b)$$

- Gradient boosting finds the **function** $F(\mathbf{x})$ that minimizes the **expected** square loss:

$$F^*(\mathbf{x}) = argmin_{F(\mathbf{x})} \mathbb{E}_{y,\mathbf{x}} L(y, F(\mathbf{x}))$$

- Gradient boosting approximate $F^*(\mathbf{x})$ with **additive expansion**, i.e., boosting:

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

Parameter of h

Gradient boosting: estimating the parameter

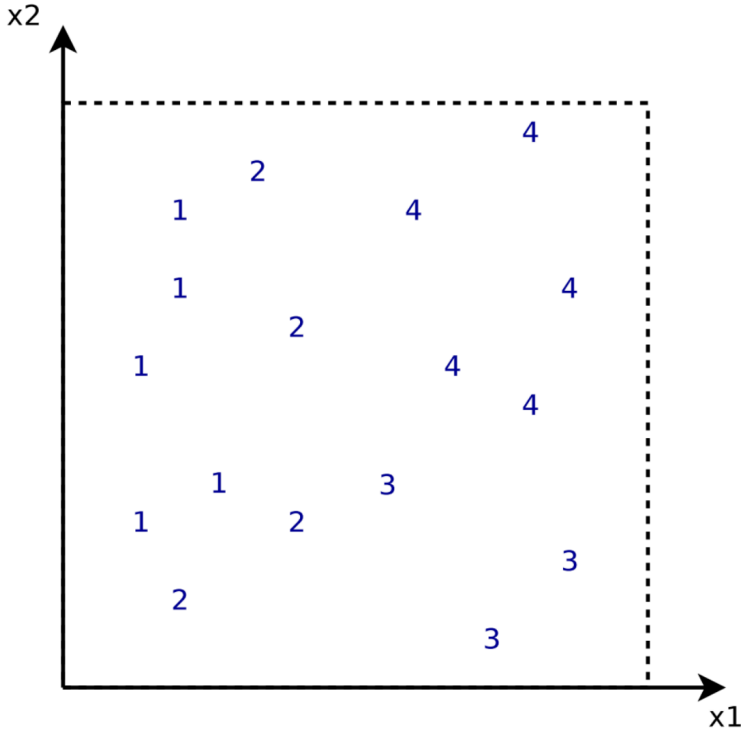
- Function parameters are iteratively fit to the training data:
  - Set $F_0(\mathbf{x})$ = initial guess (or zero)
  - For each $m = 1, 2, \ldots, M$
    - $\boldsymbol{a}_m = \text{argmin}_{\boldsymbol{a}} \sum_i L(y_i, F_{m-1}(\boldsymbol{x}_i) + \beta h(\boldsymbol{x}_i, \boldsymbol{a}))$
    - $F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}_i) + \beta h(\boldsymbol{x}_i, \boldsymbol{a})$
- You successively estimate and add a new tree to the sum
- You never go back to revisit past decisions

Gradient boosting: estimating the parameter

- Fit the function h(x; a) by least squares
  - $\boldsymbol{a}_m = \text{argmin}_a \sum_i [\tilde{y}_{im} - h(\boldsymbol{x}_i,\ \boldsymbol{a})]^2$
- to the “pseudo-residuals” (deviation from desired scores)
  - $\tilde{y}_{im} = -\left[\frac{\partial L(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\right]_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}$
- The name “gradient boosting” is because the gradient is equal to residual, i.e., we are using the gradient to improve the model
- Each h function is a small tree, e.g., a stump with depth = 1

Gradient boosting regression tree example

- In the first iteration, predict a constant value that minimizes the loss: f0(x) = mean value



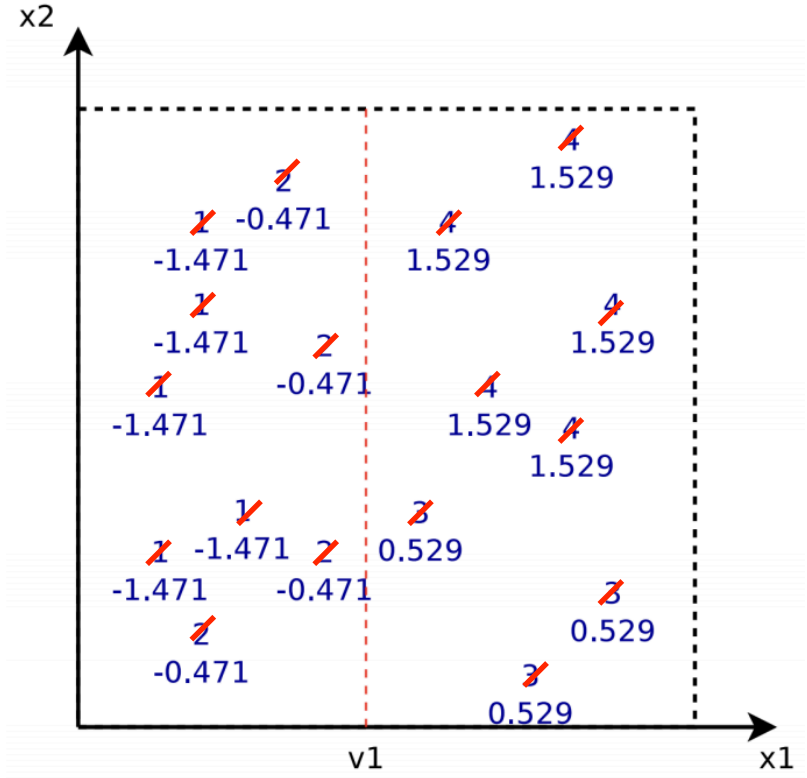
f0(x)

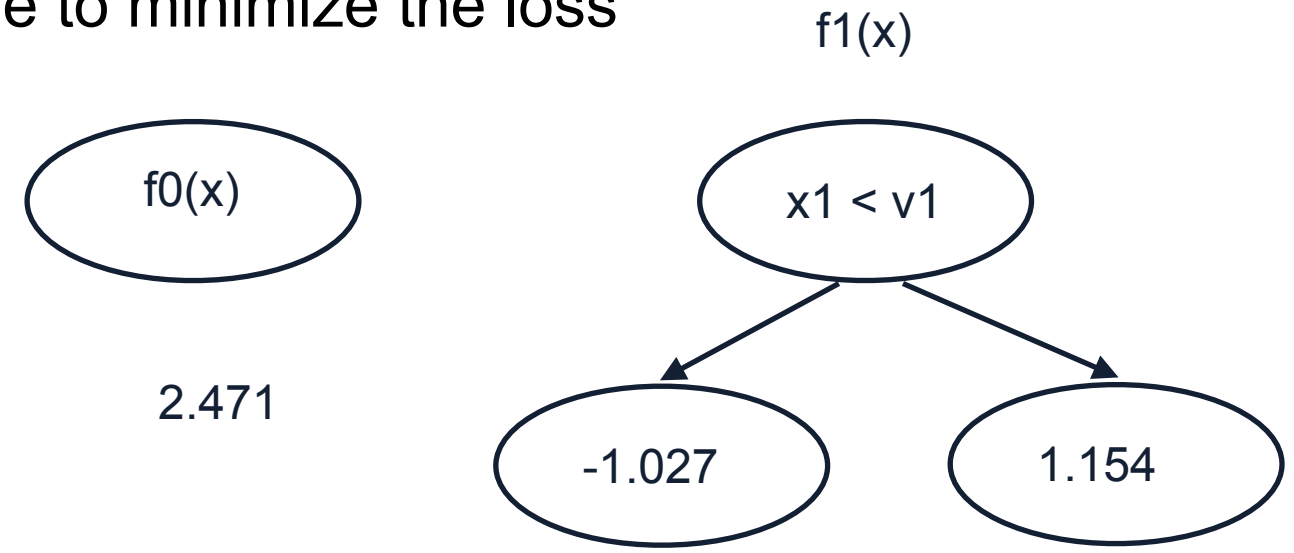
$$min_x 5(1-x)^2 + 4(2-x)^2 + 3(3-x)^2 + 5(4-x)^2$$

$$x = 2.471$$

Gradient boosting regression tree example

- After the first iteration, update each value as the new residual (subtracting 2.471), then train a regression tree to minimize the loss





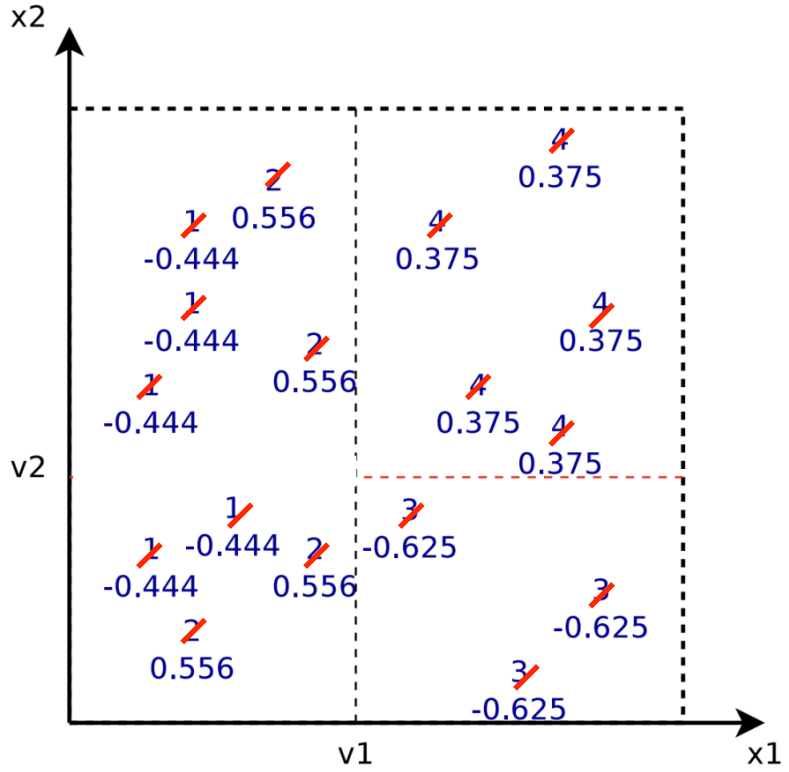
$min_x 5(1 - 2.471 - x)^2 + 4(2 - 2.471 - x)^2$

$min_x 3(3 - 2.471 - x)^2 + 5(4 - 2.471 - x)^2$

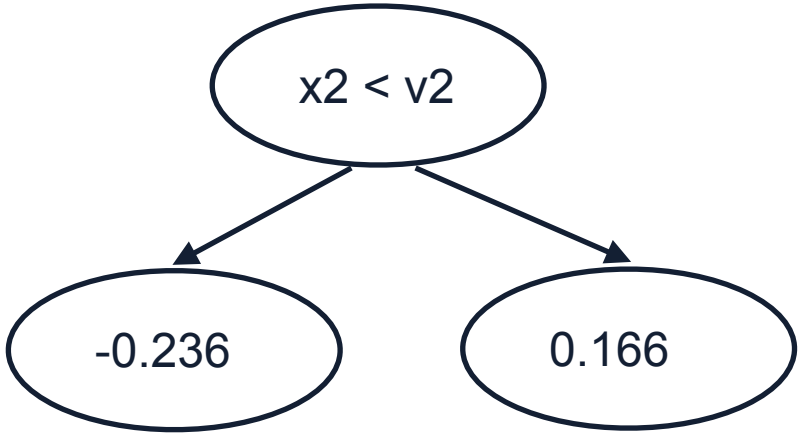
$$2.471 - 1.027 = 1.444$$
$$2.471 + 1.154 = 3.625$$

Gradient boosting regression tree example

- After the second iteration, update each value as the new residual, then train a regression tree to minimize the loss



f2(x)



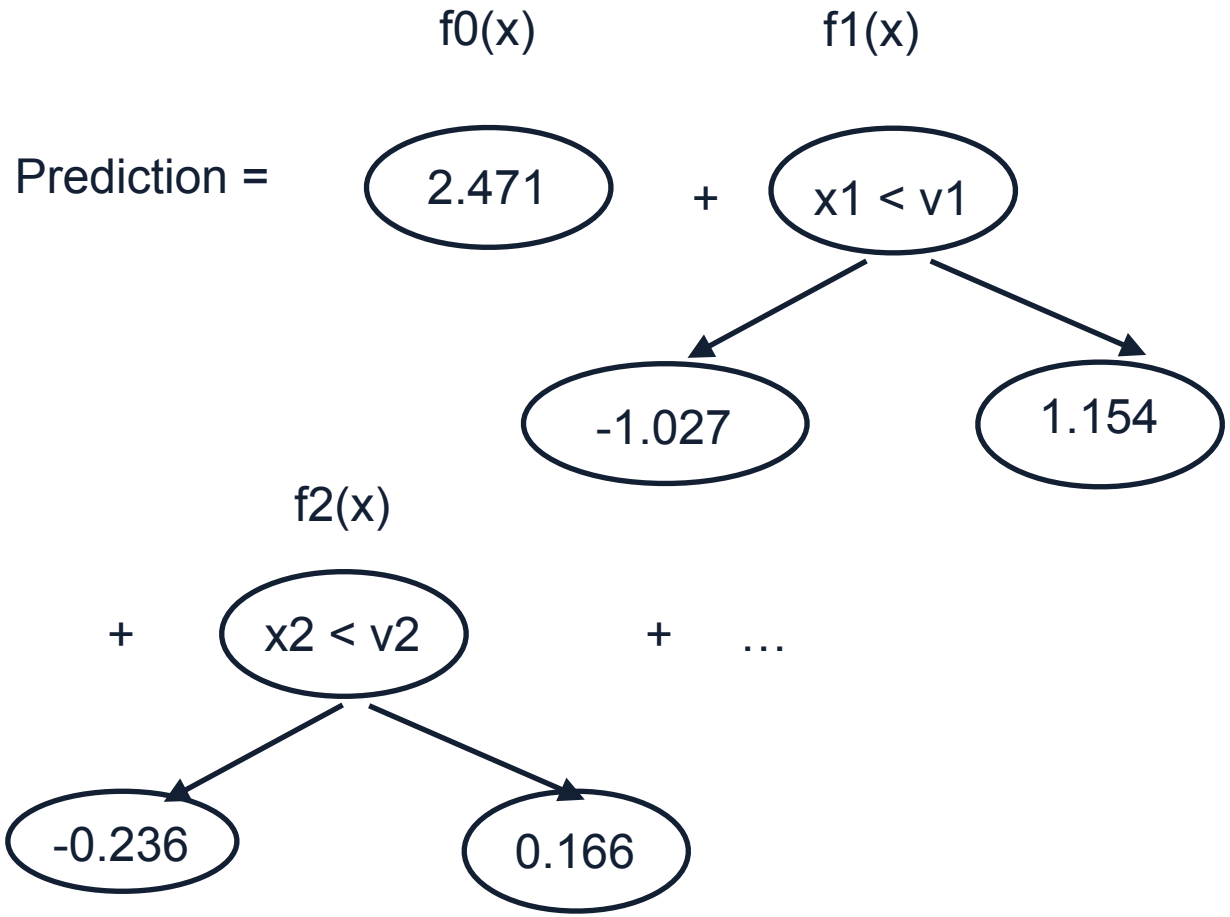
$$min_x 2(1 - 1.444 - x)^2 + 2(2 - 1.444 - x)^2 + 3(3 - 3.625 - x)^2$$

$$min_x 3(1 - 1.444 - x)^2 + 2(2 - 1.444 - x)^2 + 5(4 - 3.625 - x)^2$$

Multiple additive regression tree [Friedman 1999]

Algorithm 1 Multiple Additive Regression Trees.

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



RankNet [Burges et al. 2010]

- Use si to denote the ranking function:

$$s_i = wx_i + b \qquad P(d > d_j) = \frac{1}{1 + e^{-\sigma(s_i - s_j)}}$$

$$\min_{\Theta} \sum_{q} \sum_{i,j \in q} -\mathbb{1}\left[r_i > r_j\right] \log P(d_i > d_j) - (1 - \mathbb{1}\left[r_i > r_j\right]) \log\left(1 - P(d_i > d_j)\right)$$

- Plugging in the probability gives rise to:

$$\min_{\Theta} \sum_{i,j} C_{i,j}$$

$$C_{i,j} = \frac{1}{2}(1 - S_{i,j})\sigma(s_i - s_j) + \log\left(1 + e^{-\sigma(s_i - s_j)}\right), S_{i,j} \in \{0, +1, -1\}$$

Sij = 1 if di is more relevant than dj; –1 if the reverse, and
0 if the they have the same label

RankNet [Burges et al. 2010]

- Take the gradient of C_i,j w.r.t. w_k:

$$\frac{\partial C_{i,j}}{\partial s_i} = \sigma(\frac{1}{2}(1 - S_{i,j}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}}) = -\frac{\partial C_{i,j}}{\partial s_j}$$

$$\frac{\partial C_{i,j}}{\partial w_k} = \frac{\partial C_{i,j}}{\partial s_i}\frac{\partial s_i}{\partial w_k} + \frac{\partial C_{i,j}}{\partial s_j}\frac{\partial s_j}{\partial w_k} = \sigma(\frac{1}{2}(1 - S_{i,j}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}})(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}) = \lambda_{i,j}(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k})$$

$$\frac{\partial \Sigma}{\partial w_k} = \sum_{i,j}\frac{\partial C_{i,j}}{\partial w_k} = \sum_{i,j}\lambda_{i,j}(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}) = \sum_i (\sum_{j:\{i,j\}\in I}\lambda_{i,j} - \sum_{l:\{l,i\}\in I}\lambda_{l,i})\frac{\partial s_i}{\partial w_k}$$

λ_i

{i, j} in I: for all pairs of i, j in the data (both positive and negative)

RankNet [Burges et al. 2010]

- $\lambda_{i,j}$ describes the desired change of scores for the document pair di and dj:

$$\frac{\partial C_{i,j}}{\partial w_k} = \lambda_{i,j}\left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}\right) = \lambda_{i,j}\left(x_{i,k} - x_{j,k}\right)$$

Constant given di,dj

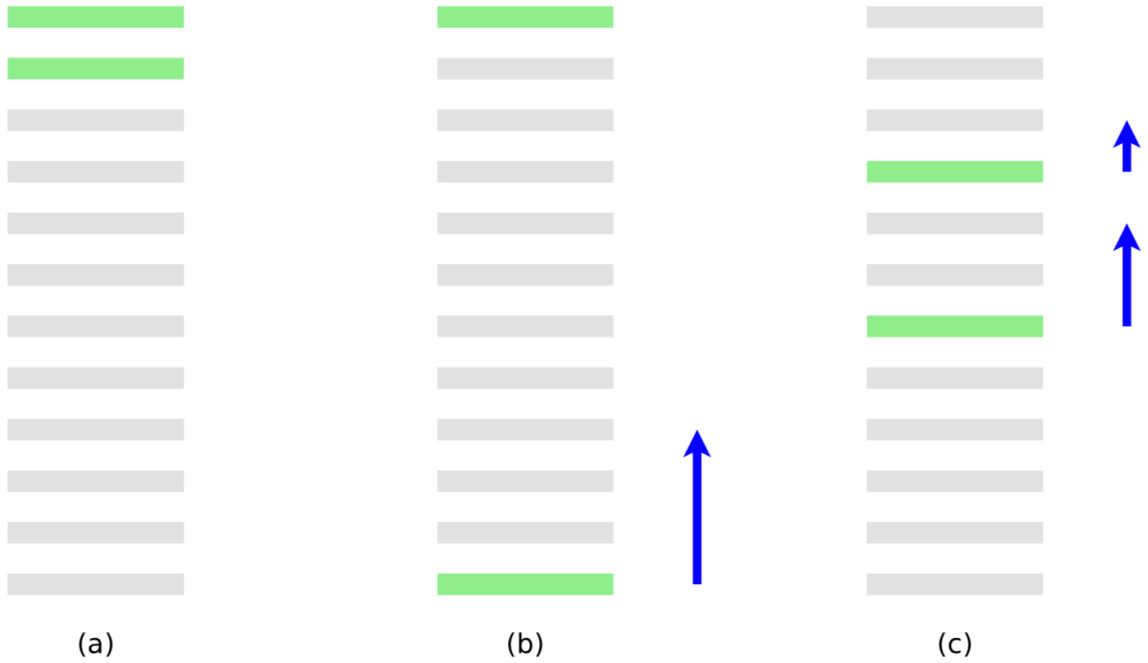
- The sum of all λ_{ij} 's and λ_{ji} 's of a query-doc vector x_i w.r.t. all other differently labelled documents for q (turned the pairwise loss into a point-wise form)

$$\frac{\partial \sum}{\partial w_k} = \sum_i \left(\sum_{j:\{i,j\}\in I} \lambda_{i,j} - \sum_{l:\{l,i\}\in I} \lambda_{l,i} \right) \frac{\partial s_i}{\partial w_k}$$

- Therefore, λ_i is sort of a "gradient" w.r.t. document di under query q

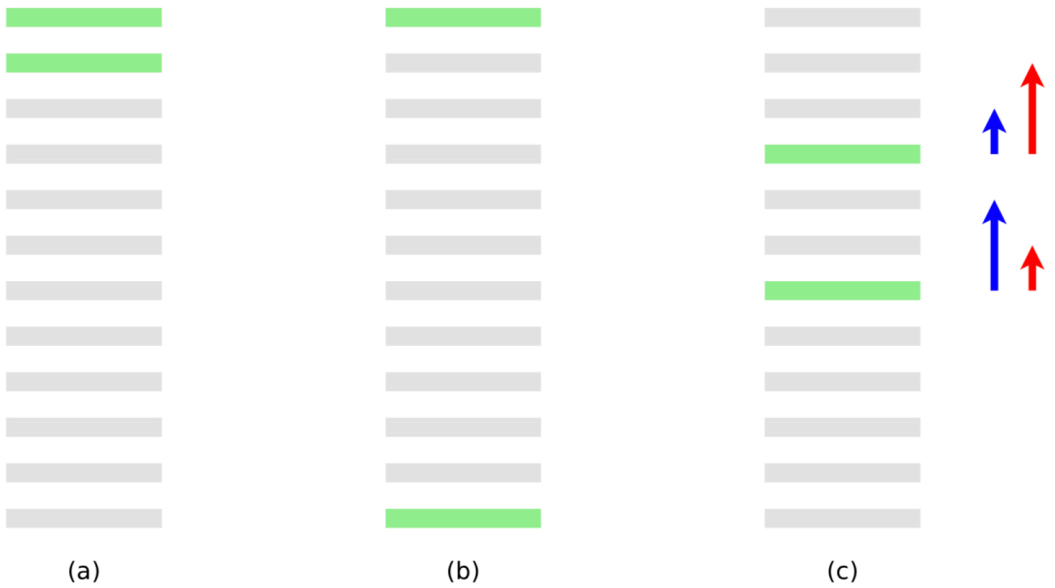
RankNet [Burges et al. 2010]

- Interpreting λ_i : (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 x_i



RankNet [Burges et al. 2010]

- RankNet minimizes the entropy loss, alternatively we can directly optimize the evaluation metrics, e.g., MAP [2007], NDCG [2010] and MRR
- Modern web search tends to emphasize better precision at a higher position



Which metrics to use for emphasizing performance at a higher position?

From RankNet to LambdaRank

- RankNet has assigned the same weight for all errors
- **LambdaRank**: scale $\lambda_{i,j}$ (desired change of scores for the document pair di and d) based on the change in the NDCG score:

$$\lambda_{i,j} = -\frac{\sigma}{1 + e^{(\sigma(s_i - s_j)}} |\Delta NDCG|$$

- Burges et al. “prove” (partly theory, partly empirical) that this change is sufficient for model to optimize NDCG

From RankNet to LambdaRank

- LambdaRank models gradients
- MART can be trained with gradients (“gradient boosting”)
- Combine both to get LambdaMART
  - MART with specified gradients and optimization step

LambdaMART [Burges et al. 2010]

- Lambdas are kind of “gradients” in RankNet
- In MART, with the specific lambda as gradients, we get:
  - LambdaMART = LambdaRank + MART (gradient boosting)

set number of trees N, number of training samples m, number of leaves per tree L
learning rate η
for $i = 0$ to m **do**
  $F_0(x_i) = \text{BaseModel}(x_i)$ //If BaseModel is empty, set $F_0(x_i) = 0$
end for
for $k = 1$ to N **do**
  for $i = 0$ to m **do**
    $y_i = \lambda_i$
    $w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$
  end for
  $\{R_{lk}\}_{l=1}^{L}$ // Create L leaf tree on $\{x_i, y_i\}_{i=1}^{m}$ R_{lk} is data items at leaf node l
  $\gamma_{lk} = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$ // Assign leaf values based on Newton step.
  $F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$ // Take step with learning rate η.
end for

Use lambda as the residual for computing the gradient

Gradient boosting

Learning to rank categorization

- Pointwise learning to rank: linear regression, SVM
- Pairwise learning to rank: RankNet, lambdaMART
- Listwise learning to rank (optimizing the metric score w.r.t. a ranked list): NDCGBoost

	Algorithm	Category	Description
1992	Linear reg	Pointwise	Staged logistic regression
1999	MART	Pairwise	Multiple additive regression tree
2005	RankNet	Pairwise	
2007	SVM^{map}	Listwise	Optimizing MAP using SVM
2010	NDCGBoost	Listwise	Optimizing NDCG using boosting
2016	XGBoost	Pairwise	Support various eval metrics

Listwise learning to rank for MAP [Yue et al. 2007]

- Listwise learning to rank approach minimizes the loss with respect to the ranked list (e.g., MAP, NDCG)

$$\sum_i L(\pi_i, y_i)$$

- SVM MAP [Yue et al. 2007]

minimize $$\sum_{i=1}^{m} \max_{\pi_i^* \in \Pi_i^*; \pi_i \in \Pi_i \setminus \Pi_i^*} \left(\left(E(\pi_i^*, \mathbf{y}_i) - E(\pi_i, \mathbf{y}_i) \right) \cdot \left[\left[S(\mathbf{x}_i, \pi_i^*) \leq S(\mathbf{x}_i, \pi_i) \right] \right] \right)$$

Taking upper bound

$$\Rightarrow \quad \sum_{i=1}^{m} \left[\max_{\pi_i^* \in \Pi_i^*, \pi_i \in \Pi_i \setminus \Pi_i^*} \left(\left(E(\pi_i^*, \mathbf{y}_i) - E(\pi_i, \mathbf{y}_i) \right) - \left(S(\mathbf{x}_i, \pi_i^*) - S(\mathbf{x}_i, \pi_i) \right) \right) \right]_+$$

+: hinge loss

Listwise learning to rank for NDCG [Valizadegan et al. 2009]

- Optimizing the NDCG measure:

$$\bar{\mathcal{L}}(Q,F) = \frac{1}{n}\sum_{k=1}^{n}\frac{1}{Z_k}\sum_{i=1}^{m_k}\left\langle \frac{2^{r_i^k}-1}{\log(1+j_i^k)}\right\rangle_F = \frac{1}{n}\sum_{k=1}^{n}\frac{1}{Z_k}\sum_{i=1}^{m_k}\sum_{\pi^k\in S_{m_k}}\Pr(\pi^k|F,q^k)\frac{2^{r_i^k}-1}{\log(1+\pi^k(i))}$$

- Directly optimizing the above function is infeasible, so they derived an approximation

$$\bar{\mathcal{M}}(Q,\tilde{F}) \le \bar{\mathcal{M}}(Q,F) + \gamma(\alpha) + \frac{\exp(3\alpha)-1}{3}\sum_{k=1}^{n}\sum_{i=1}^{m_k} f_i^k\left(\sum_{j=1}^{m_k}\frac{2^{r_i^k}-2^{r_j^k}}{Z_k}\theta_{i,j}^k\right)$$

- Then compute the ranking function by ensemble following a similar approach as boosting

Listwise learning to rank for NDCG [Valizadegan et al. 2009]

Algorithm 1 NDCG_Boost: A Boosting Algorithm for Maximizing NDCG

1: Initialize $F(d_i^k) = 0$ for all documents
2: **repeat**
3: Compute $\theta_{i,j}^k = \gamma_{i,j}^k I(j \neq i)$ for all document pairs of each query. $\gamma_{i,j}^k$ is given in Eq. (14).
3: Compute the weight for each document as

$$w_i^k = \sum_{j=1}^{m_k} \frac{2^{r_i^k} - 2^{r_j^k}}{Z_k} \theta_{i,j}^k \tag{16}$$

3: Assign each document the following class label $y_i^k = \text{sign}(w_i^k)$.
4: Train a classifier $f(\mathbf{x}) : \mathbb{R}^d \rightarrow \{0, 1\}$ that maximizes the following quantity

$$\eta = \sum_{k=1}^{n} \sum_{i=1}^{m_k} |w_i^k| f(d_i^k) y_i^k \tag{17}$$

5: Predict f_i for all documents in $\{D^k, i = 1, \ldots, n\}$
6: Compute the combination weight α as provided in Equation (15).
7: Update the ranking function as $F_i^k \leftarrow F_i^k + \alpha f_i^k$.
8: **until** reach the maximum number of iterations

Summary

- The idea of machine learning for ranking has been around for a long time
- In the last 2 decades, learning to rank has become a hot topic due to the development of machine learning, the availability of massive amount of user log and the growth in computational power
- The mainstream learning to rank algorithms focus on tree ensemble method
- Machine-learned ranking over many features now easily beats traditional hand-designed ranking functions by learning the evaluation metrics directly

Resources for learning to rank

- Ranklib: https://sourceforge.net/p/lemur/wiki/RankLib/
- XGBoost: https://github.com/dmlc/xgboost
- Yahoo! Learning to rank challenge dataset
- C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An Overview. Microsoft TR 2010.
- O. Chapelle and Y. Chang. Yahoo! Learning to Rank Challenge Overview. *JMLR Proceedings* 2011