Learning to rank

Alexey Kharlamov
Vadim Kalashnikov

Яндекс

learning to rank

Найти

ПОИСК КАРТИНКИ ВИДЕО КАРТЫ МАРКЕТ НОВОСТИ ПЕРЕВОДЧИК ЕЩЁ

Нашлось 67 млн результатов

23 показа в месяц

Дать объявление

Learning to rank - Wikipedia

en.wikipedia.org › Learning to rank

Learning to rank or machine-**learned ranking** (MLR) is the application of machine **learning**, typically supervised, semi-supervised or reinforcement **learning**, in the construction of **ranking** models for information retrieval systems.

Методы обучения ранжированию (**Learning to Rank**)

machinelearning.ru › wiki/images/8/89/Voron…Ranking…

Постановка задачи и приложения Основные подходы к ранжированию. Ранжирование в Яндексе. Методы обучения ранжированию (**Learning to Rank**).

PDF

Сохранить на Яндекс.Диск | Посмотреть PDF

Learning to rank | Wiki | Everipedia

everipedia.org › wiki/Learning_to_rank/

Winning entry in the recent Yahoo **Learning to Rank** competition used an ensemble of LambdaMART models.[22]. 2008.

Learning to rank - WikiVisually

wikivisually.com › wiki/Learning_to_rank

Often a **learning-to-rank** problem is reformulated as an optimization problem with respect to one of these metrics. Examples of **ranking** quality measures

Ещё одна важная постановка задачи – **learning to rank**

logic.pdmi.ras.ru › Sergey Nikolenko › …/mlhse17/17-ranking.pdf

развитие **learning to rank**. • Обычно подходы работают на pairwise-ошибке: • RankSVM: берём SVM с ошибкой $\ell(t) = \max(0, 1 - t)$...

PDF

Relevancy

How to understand that document is relevant to some query?
What different factors can we consider?

- Analyze content (text, images, links) of web page
- Analyze query and context

Examples of content factors:

- Text representations
- Web-graph characteristics (the way to detects artificial structures)

Examples of query/context factors:

- User age/location/gender
- User search story (preferences are important)

To increase the quality of search we probably should care about:

- diversity in top-k results
- spam/fraud/irrelevant porn pics filters
- speed
-

Problem

Given:

- X: documents (objects) from some universal set

- R: set of indexes pairs (i, j)
- y: “true relevance”

Objective: model a such that:
a(X(i)) > a(X(j)) if and only if (i, j) is in R (or y(i) > y(j)).

Note that these two approaches differ from each other.

Key point:

- structure we have to optimize is not continuous at all
  (need tricks to use common optimizations methods)
- where can we get data to optimize our algorithms?
  (some precomputed experts answers, user search history(!!!))

Pointwise approach

1. look at every document
2. try to predict how relevant it is for the current query
3. sort the result list by model scores
4. profit? (not really)

Loss Function:

sum of losses L(a(x) - y(x)) for every document x

Good:

+ we already know enough methods
+ easy and fast in case we have "good" data to train

Bad:

- we completely ignored initial problem :(
- extra work: for ranking system scores (1, 2) and (1.4, 1.6) lead to the same sort result

2007	QBRank	pairwise
2007	RankCosine	listwise
2007	RankGP[19]	listwise
2007	RankRLS	pairwise
2007	SVM^{map}	listwise
2008	LambdaMART	pairwise/listwise
2008	ListMLE	listwise
2008	PermuRank	listwise
2008	SoftRank	listwise
2008	Ranking Refinement[22]	pairwise
2008	SSRankBoost[23]	pairwise
2008	SortNet[24]	pairwise
2009	MPBoost	pairwise
2009	BoltzRank	listwise
2009	BayesRank	listwise
2010	NDCG Boost[25]	listwise
2010	GBlend	pairwise
2010	IntervalRank	pairwise & listwise
2010	CRR	pointwise & pairwise
2017	ES-Rank	listwise

Pairwise approach

1. look at every pair (document, document)
2. try to predict the correct order for 2 documents
3. sort the result list by model scores
4. profit?

Loss Function:

sum of losses L(a(x) - a(y)) for every pair of documents (x, y)

$$\sum_{(i,j)\in R} [a(x_j) - a(x_i) < 0] \leqslant \sum_{(i,j)\in R} L\left(a(x_j) - a(x_i)\right)$$

Good:

+ seems reasonable & works well

Bad:

- not as fast
- don't focus on the most relevant documents

2007	QBRank	pairwise
2007	RankCosine	listwise
2007	RankGP[19]	listwise
2007	RankRLS	pairwise
2007	SVM^{map}	listwise
2008	LambdaMART	pairwise/listwise
2008	ListMLE	listwise
2008	PermuRank	listwise
2008	SoftRank	listwise
2008	Ranking Refinement[22]	pairwise
2008	SSRankBoost[23]	pairwise
2008	SortNet[24]	pairwise
2009	MPBoost	pairwise
2009	BoltzRank	listwise
2009	BayesRank	listwise
2010	NDCG Boost[25]	listwise
2010	GBlend	pairwise
2010	IntervalRank	pairwise & listwise
2010	CRR	pointwise & pairwise
2017	ES-Rank	listwise

Listwise approach

1. look at all documents
2. try to predict the correct permutation or
   some probability distribution z on all permutations

Loss Function:
y is ‘true relevancy’ z in predicted permutations probabilities.

$$Q(y,z) = -\sum_{j=1}^{n_q} P_y(j) \log P_z(j). \quad P_z(j) = \frac{\varphi(z_j)}{\sum_{k=1}^{n} \varphi(s_k)} \qquad P_z(\pi) = \prod_{j=1}^{n_q} \frac{\varphi(z_{\pi(j)})}{\sum_{k=j}^{n_q} \varphi(z_{\pi(k)})},$$

Good:
+ we consider more aspects of initial problem

Bad:
- probably not as fast
- not as easy

2007	QBRank	pairwise
2007	RankCosine	listwise
2007	RankGP[19]	listwise
2007	RankRLS	pairwise
2007	SVM^{map}	listwise
2008	LambdaMART	pairwise/listwise
2008	ListMLE	listwise
2008	PermuRank	listwise
2008	SoftRank	listwise
2008	Ranking Refinement[22]	pairwise
2008	SSRankBoost[23]	pairwise
2008	SortNet[24]	pairwise
2009	MPBoost	pairwise
2009	BoltzRank	listwise
2009	BayesRank	listwise
2010	NDCG Boost[25]	listwise
2010	GBlend	pairwise
2010	IntervalRank	pairwise & listwise
2010	CRR	pointwise & pairwise
2017	ES-Rank	listwise

Metrics

For binary classification (relevant or not) for query/document pairs
we could use simple metrics: F-score, AUC-ROC, etc.

We often focus on the quality of top-k relevant documents and don’t really care about irrelevant documents ordering.

@k is the same as top-k.

Precision metrics

In case relevancy can be represented as r^true - {0, 1} - variable:

precision at k:

$$p@K = \frac{\sum_{k=1}^{K} r^{true}(\pi^{-1}(k))}{K}$$

average precision at k:

$$ap@K = \frac{1}{K}\sum_{k=1}^{K} r^{true}(\pi^{-1}(k)) \cdot p@k.$$

Metrics

In case relevancy can be represented as y - [0, 1] - variable:

DCG (discounted cumulative gain):
g(y) represents metrics gain from including document
d(i) represents discount based on document index

$$\mathrm{DCG@k}(q) = \sum_{i=1}^{k} g(y_{(i)})d(i).$$

normalized DCG
max is taken over all possible top-k documents

$$\mathrm{nDCG@k}(q) = \frac{\mathrm{DCG@k}(q)}{\max \mathrm{DCG@k}(q)}.$$

Cascade metrics (pFound)

Consider user who consequently looks at every document in top-k and quits randomly after every irrelevant document or in case he/she found needed information.

Let y(i) is probability to find needed information in i_th document.
Let p_out is probability to quit after every failed attempt.
Let p_i is probability to look at i_th document.

Obviously p_1 = 1 and $p_{i+1} = p_i(1 - y_{(i)})(1 - p_{\text{out}}),$

Then probability to find needed information $\text{pFound@}k(q) = \sum_{i=1}^{k} p_i y_{(i)}.$

TF-IDF

Déjà vu

Variants of term frequency (TF) weight

weighting scheme	TF weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Variants of inverse document frequency (IDF) weight

weighting scheme	IDF weight ($n_t = \lvert\{d \in D : t \in d\}\rvert$)
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log \left(1 + \frac{N}{n_t}\right)$
inverse document frequency max	$\log \left(\frac{\max_{\{t' \in d\}} n_{t'}}{1 + n_t}\right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

$$\mathrm{BM25}(q, d) = \sum_{i=1}^{n} \mathrm{IDF}(q_i) \frac{\mathrm{tf}(q_i, d)(k_1 + 1)}{\mathrm{tf}(q_i, d) + k_1 \left(1 - b + b \frac{|D|}{\bar{n}_d}\right)},$$

Online assessment

We want both fast and smart system to navigate in changing Internet.

- Need recalculate our factors efficiently
- Need to store data efficiently (every extra byte is extra money to pay)
- Need to use new data from users to fit our models online

So here comes many algorithmical and engineering problems to solve.

RankNet

- RankNet – первая идея pairwise-подхода.
- Пусть у нас есть кое-какие прямые данные для обучения (т.е. про некоторые подмножества документов эксперт сказал, какие более релевантны, какие менее).
- Подход к решению: давайте обучать функцию, которая по данному вектору атрибутов $\mathbf{x} \in \mathbb{R}^n$ выдаёт $f(\mathbf{x})$ и ранжирует документы по значению $f(\mathbf{x})$.

RankNet

- Итак, для тестовых примеров $\mathbf{x}_i$ и $\mathbf{x}_j$ модель считает $s_i = f(\mathbf{x}_i)$ и $s_j = f(\mathbf{x}_j)$, а затем оценивает

$$p_{ij} = p(\mathbf{x}_i \succ \mathbf{x}_j) = \frac{1}{1 + e^{-\alpha(s_i - s_j)}}.$$

- А данные – это на самом деле $q(\mathbf{x}_i \succ \mathbf{x}_j)$, либо точные из $\{0, 1\}$, либо усреднённые по нескольким экспертам.
- Поэтому разумная функция ошибки – кросс-энтропия

$$C = -q_{ij} \log p_{ij} - (1 - q_{ij}) \log(1 - p_{ij}).$$

RankNet

- Ошибка: $C = -q_{ij} \log p_{ij} - (1 - q_{ij}) \log(1 - p_{ij})$.
- Для самого частого случая, когда оценки релевантности точные, и $q_{ij} = (1 + S_{ij})/2$ для $S_{ij} \in \{-1, 0, +1\}$, мы получаем

$$C = \frac{1}{2}(1 - S_{ij})\alpha(s_i - s_j) + \log\left(1 + e^{-\alpha(s_i - s_j)}\right), \text{ т.е.}$$

$$C = \begin{cases} \log\left(1 + e^{-\alpha(s_i - s_j)}\right), & \text{если } S_{ij} = 1, \\ \log\left(1 + e^{-\alpha(s_j - s_i)}\right), & \text{если } S_{ij} = -1. \end{cases}$$

- Т.е. ошибка симметрична, что уже добрый знак.

RankNet

- Ошибка: $C = -q_{ij} \log p_{ij} - (1 - q_{ij}) \log(1 - p_{ij})$.
- Давайте подсчитаем градиент по s_i:

$$\frac{\partial C}{\partial s_i} = \alpha \left(\frac{1 - S_{ij}}{2} - \frac{1}{1 + e^{\alpha(s_i - s_j)}} \right) = -\frac{\partial C}{\partial s_j}.$$

- И теперь осталось использовать этот подсчёт для градиента по весам:

$$\frac{\partial C}{\partial w_k} = \sum_i \frac{\partial C}{\partial s_i} \frac{\partial s_i}{\partial w_k} + \sum_j \frac{\partial C}{\partial s_j} \frac{\partial s_j}{\partial w_k}.$$

RankNet

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

где

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \alpha \left(\frac{1 - S_{ij}}{2} - \frac{1}{1 + e^{\alpha(s_i - s_j)}} \right).$$

Переупорядочив пары так, чтобы всегда было $\mathbf{x}_i \succ \mathbf{x}_j$ и $S_{ij} = 1$, получим

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = -\alpha \frac{1}{1 + e^{\alpha(s_i - s_j)}}.$$

RankNet

- $\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = -\alpha \frac{1}{1+e^{\alpha(s_i - s_j)}}$.
- Значит, если для данной выдачи есть множество пар I, в которых известно, что $\mathbf{x}_i \succ \mathbf{x}_j$, $(i, j) \in I$, то суммарный апдейт для веса w_k будет

$$\Delta w_k = -\eta \left[\sum_{(i,j)\in I} \lambda_{ij} \frac{\partial s_i}{\partial w_k} - \lambda_{ij} \frac{\partial s_j}{\partial w_k} \right] = -\eta \sum_i \lambda_i \frac{\partial s_i}{\partial w_k},$$

где $\lambda_i = \sum_{j:(i,j)\in I} \lambda_{ij} - \sum_{j:(j,i)\in I} \lambda_{ij}$.

LambdaRank

- Проблема с RankNet в том, что оптимизируется число попарных ошибок, а это не всегда то, что нужно.
- Градиенты RankNet – это не то же самое, что градиенты NDCG:

- Как оптимизировать, скажем, NDCG?

LambdaRank

- Заметим, что нам сама ошибка не нужна, а нужны только градиенты λ (стрелочки).
- Давайте просто представим себе мифическую функцию ошибки C, у которой градиент

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \frac{-\alpha}{1 + e^{\alpha(s_i - s_j)}} \left|\Delta_{\mathrm{NDCG}}\right|,$$

  где Δ_{NDCG} – это то, на сколько NDCG изменится, если поменять i и j местами.
- То есть мы считаем градиенты уже после сортировки документов по оценкам, и градиенты как будто от NDCG.

LambdaRank

- NDCG нужно максимизировать, так что берём

$$\Delta w_k = \eta \frac{\partial C}{\partial w_k}, \text{ и тогда}$$

$$\delta C = \frac{\partial C}{\partial w_k} \delta w_k = \eta \left(\frac{\partial C}{\partial w_k} \right)^2 > 0.$$

- Оказывается, что такой подход фактически напрямую оптимизирует NDCG (сглаженную версию).

LambdaMART

- MART – это бустинг, сделанный на регрессионных деревьях.
- Иначе говоря, окончательная модель будет, опять же, по $\mathbf{x} \in \mathbb{R}^d$ искать $y \in \mathbb{R}$, и искать в виде

$$F_M(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}),$$

где $f_m(\mathbf{x})$ задаётся регрессионным деревом, а $\alpha_m \in \mathbb{R}$ – веса бустинга, и в процессе обучения обучаются одновременно f_m и α_m.

LambdaMART

- Мы просто добавим в градиенты целевую метрику, например

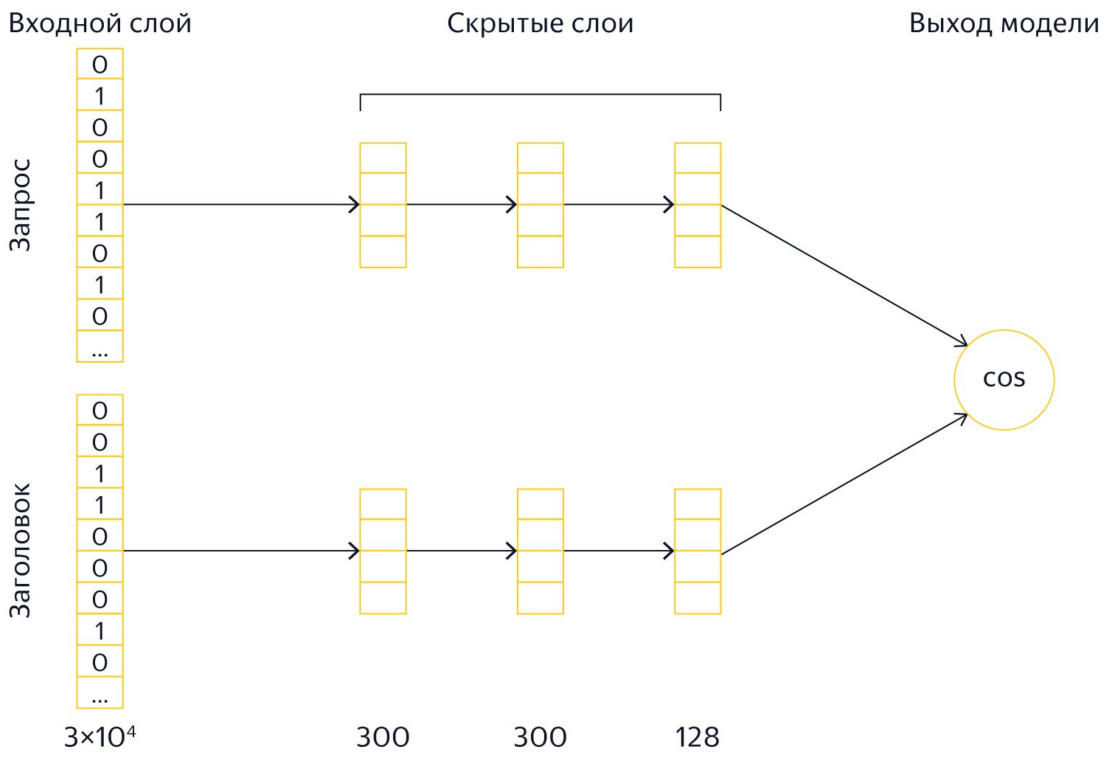
$$\lambda_{ij} = S_{ij} \left| \Delta \mathrm{NDCG} \frac{\partial C_{ij}}{\partial o_{ij}} \right|, \quad o_{ij} = F(x_i) - F(x_j).$$

- Функция ошибки нам тоже уже известна:

$$C_{ij} = C(o_{ij}) = s_j - s_i + \log\left(1 + e^{s_i - s_j}\right),$$

$$\frac{\partial C_{ij}}{\partial o_{ij}} = \frac{\partial C_{ij}}{\partial s_i} = -\frac{1}{1 + e^{o_{ij}}}.$$

DSSM



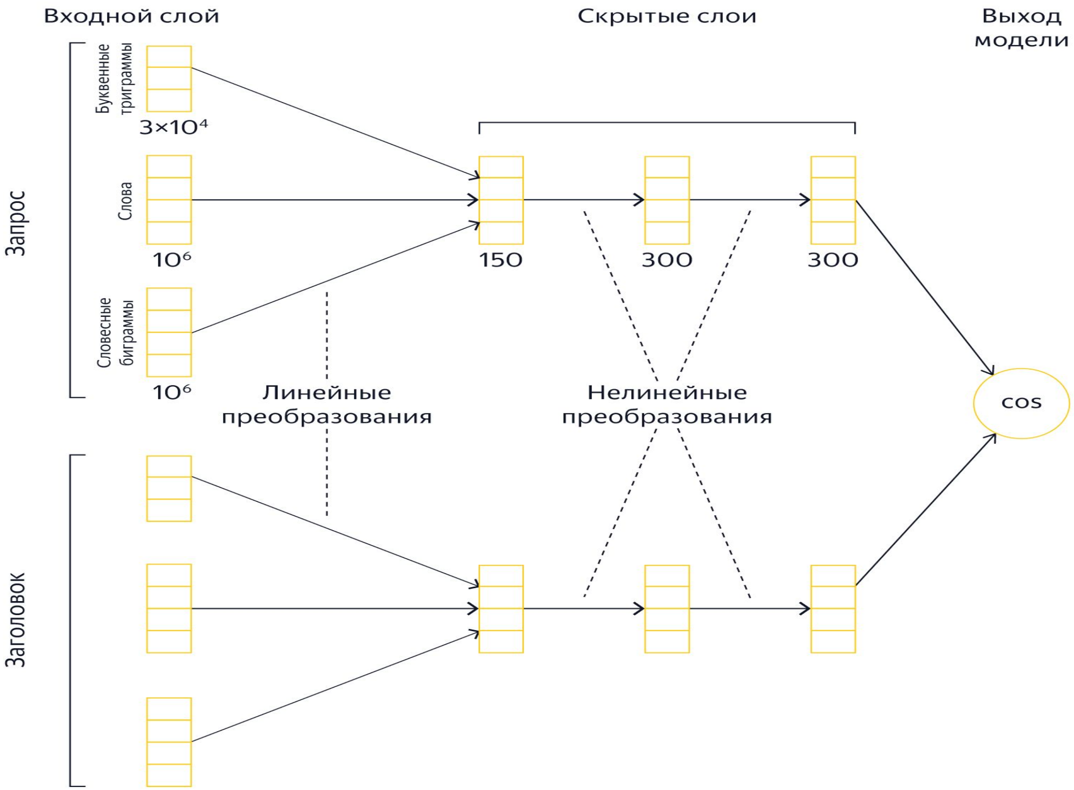
Word Hashing

$$R(Q,D) = \text{cosine}(y_Q, y_D) = \frac{{y_Q}^T y_D}{\|y_Q\|\|y_D\|}$$

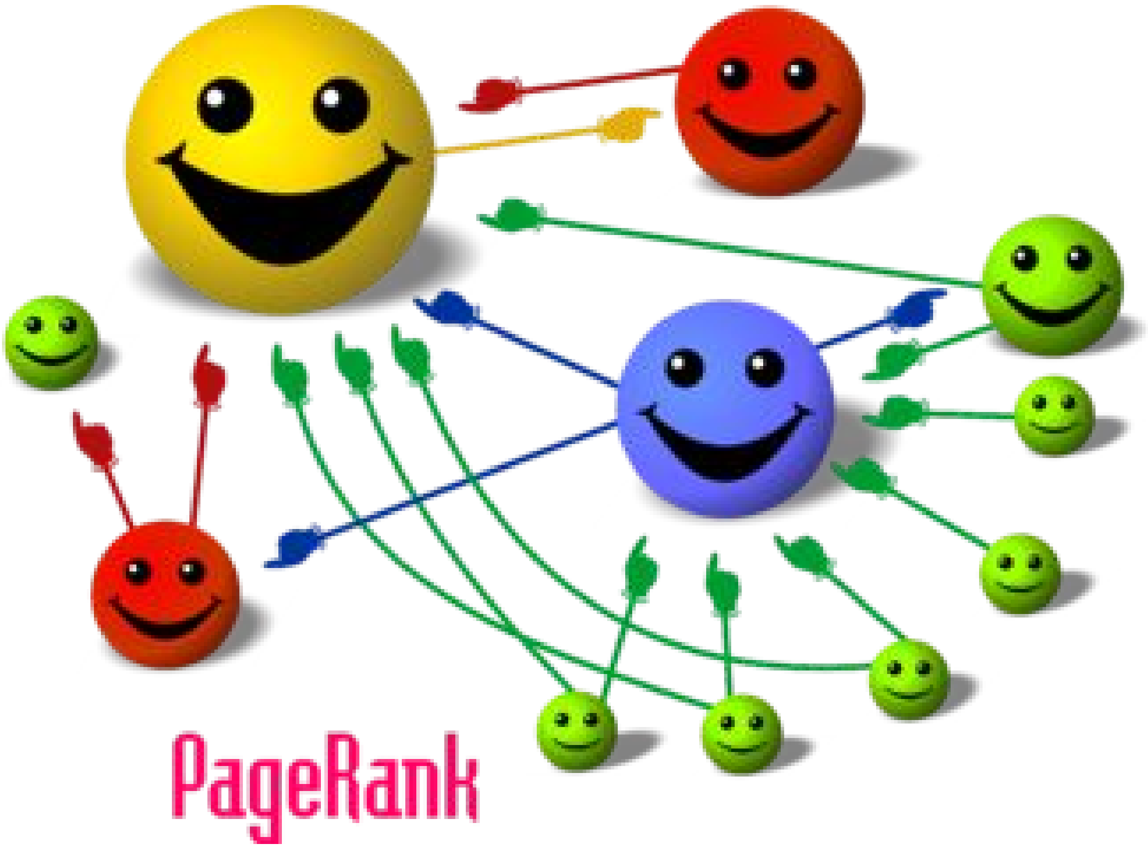
$$P(D|Q) = \frac{\exp(\gamma R(Q,D))}{\sum_{D' \in \mathbf{D}} \exp(\gamma R(Q,D'))}$$

$$L(\Lambda) = -\log \prod_{(Q,D^+)} P(D^+|Q)$$

DSSM



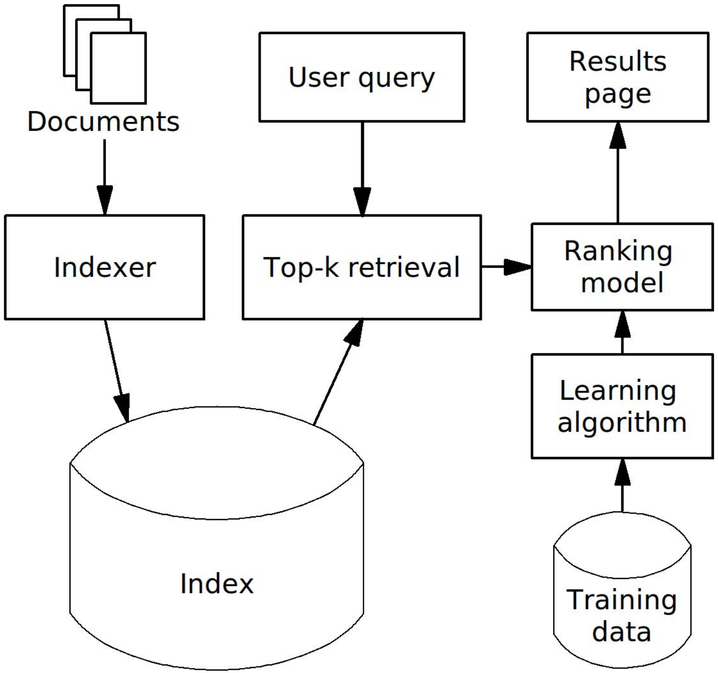
Hard negative mining



$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \longrightarrow P = \begin{bmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

$$PR(A) = \frac{1-d}{N} + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right).$$

Features



- *Query-independent* or *static* features — PageRank.
- *Query-dependent* or *dynamic* features — TF-IDF.
- *Query level features* or *query features* — the number of words in a query.

XGBoost

Evaluation metrics

- “ndcg”: Normalized Discounted Cumulative Gain
- “map”: Mean average precision
- “ndcg@n”,”map@n”: n can be assigned as an integer to cut off the top positions in the lists for evaluation

Additional info

- Objective - rank:pairwise
- Algorithm - LambdaRank

Links

- https://everipedia.org/wiki/Learning_to_rank/
- https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/cikm2013_DSSM_fullversion.pdf
- https://habrahabr.ru/company/yandex/blog/314222/
- https://logic.pdmi.ras.ru/~sergey/teaching/mlhse17/17-ranking.pdf
- https://logic.pdmi.ras.ru/~sergey/teaching/mlstc12/17-neural.pdf
- https://everipedia.org/wiki/PageRank/