

# ОБУЧЕНИЕ РАНЖИРОВАНИЮ

---

Сергей Николенко

СПбГУ – Санкт-Петербург

7 марта 2020 г.

---

## *Random facts:*

- 7 марта 321 г. по указу императора Константина Великого воскресенье было объявлено выходным днём
- 7 марта 1530 г. папа римский отказал Генриху VIII в разводе
- 7 марта 1573 г. Иван Фёдоров основал во Львове типографию
- 7 марта 1824 г. 12-летний Ференц Лист впервые выступил с концертом в Париже
- 7 марта 1876 г. Александр Белл запатентовал телефон
- 7 марта 1933 г. Чарльз Дарроу в Атлантик-Сити изобрёл игру «Монополия»
- 7 марта 1981 г. на открытии Ленинградского рок-клуба прошёл первый концерт группы «Зоопарк»

# LEARNING TO RANK

---

- Ещё одна важная постановка задачи – *learning to rank* (ранжирование).
- Задача поиска: выдать ранжированный список наиболее релевантных документов по запросу.
- Классические метрики:
  - (1) точность (precision) – количество «хороших» (релевантных запросу) документов в выдаче, делённое на общее количество документов в выдаче;
  - (2) полнота (recall) – количество «хороших» документов в выдаче, делённое на общее число релевантных документов в базе поисковой системы.
- Однако здесь те же проблемы; эти параметры не зависят от ранжирования выдачи, надо знать заранее, сколько потребуется рекомендаций.

- Метрики качества ранжирования:
  - NDCG, Normalized Discounted Commulative Gain; выберем топ- $k$  рекомендаций ( $k$  может быть заведомо больше нужного числа) и посчитаем:

$$\text{DCG}_k = \sum_{i=1}^k \frac{2^{\hat{r}_i} - 1}{\log_2(1 + i)},$$
$$\text{NDCG}_k = \frac{\text{DCG}_k}{\text{IDCG}_k},$$

где  $\hat{r}_i$  – наша оценка рейтинга продукта на позиции  $i$ , а  $\text{IDCG}_k$  – значение  $\text{DCG}_k$  при ранжировании по истинным значениям (рейтингам из валидационного набора);

- NDCG от 0 до 1, но ей трудно придумать естественную интерпретацию (как вероятность чего-нибудь, например).

- Метрики качества ранжирования:
  - AUC, Area Under (ROC) Curve; можно считать по всей выдаче сразу;
  - AUC – вероятность того, что случайно выбранная пара продуктов с разными оценками будет отранжирована правильно (релевантный будет выше в выдаче, чем нерелевантный);
  - в бинарном случае можно посчитать в замкнутом виде:

$$\hat{A} = \frac{S_0 - n_0(n_0 + 1)/2}{n_0 n_1},$$

где  $n_0, n_1$  – число релевантных и нерелевантных запросу документов,  $S_0 = \sum p_i$  – сумма номеров позиций релевантных объектов в выдаче.

- Метрики, основанные на каскадных моделях пользователей.
- ERR (Expected Reciprocal Rank) – ожидаемый обратный ранг документа, на котором остановится пользователь:

$$\begin{aligned} \text{ERR} &= \sum_{i=1}^n \frac{1}{i} p(\text{пользователь остановится на } i) = \\ &= \sum_{i=1}^n \frac{1}{i} R(y_i) \prod_{j=1}^{i-1} (1 - R(y_j)), \end{aligned}$$

где остановка происходит, если случайное число от 0 до 1 меньше  $R(y)$ ; часто используют  $R(r) = \frac{2^r - 1}{2^{r_{\max}}}$ .

- К сожалению, все метрики качества кусочно-постоянные. Надо что-то придумывать.

- Классическая функция BM25, ещё из 1970-80-х годов:

$$\text{score}(D, Q) = \sum_{i=1}^n \text{idf}(q_i) \frac{\text{tf}(q_i, D)(k_1 + 1)}{\text{tf}(q_i, D) + k_1 \left(1 - b + b \frac{|D|}{\text{Avgdoclen}}\right)},$$

где  $q_i$  – ключевые слова из запроса  $Q$ ,  $D$  – документ,  $k_i$  – параметры, которые можно обучить или выставить  
 $k_1 \in [1.2, 2.0]$ ,  $b = 0.75$ .

- Сейчас обычно пытаются переформулировать как задачу supervised learning.
- Данные вида  $(Q, D, r)$ , где  $r$  – оценка релевантности (обычно дискретная и размеченная людьми).

## ТРИ ПОДХОДА

- Выделяя признаки в паре  $(Q, D)$ , получим  $(\mathbf{x}_j^q, r_j^q)_{q,j}$ .
- Три подхода к тому, чтобы сделать непрерывную целевую функцию:

- *pointwise* (поточечный): для функции ошибки  $\ell$  (например, ошибка регрессии или классификации)

$$\sum_{q,j} \ell(f(\mathbf{x}_j^q), r_j^q) \rightarrow \min;$$

- *pairwise* (попарный): правильно упорядочиваем пары с разными оценками

$$\sum_q \sum_{i,j:r_i^q > r_j^q} \ell(f(\mathbf{x}_i^q) - f(\mathbf{x}_j^q)) \rightarrow \min;$$

- *listwise* (списочный): определим функцию потери на всём списке документов, ассоциированных с запросом,

$$\ell(\{f(\mathbf{x}_j^q)\}_{j=1}^{m_q}, \{r_j^q\}_{j=1}^{m_q}) \rightarrow \min.$$



- Обычно подходы работают на pairwise-ошибке:
  - RankSVM: берём SVM с ошибкой  $\ell(t) = \max(0, 1 - t)$  и обучаем на попарных сравнениях;
  - RankBoost, RankNet, LambdaRank (поговорим о них).
- Публичные датасеты и большие продвижения около 2010:
  - «Интернет-математика» от Яндекса (2009),
  - Microsoft Learning to Rank Datasets (2010),
  - Yahoo! Learning to Rank Challenge (2010).

# 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  $f(\mathbf{x})$ .

- 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

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

- 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

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

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

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

- 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:

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

- Основной пафос RankNet – в том, что это можно факторизовать:

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

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

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

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

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

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

- И можно просто считать  $\lambda_i$  по таким mini-batches от каждого запроса, а потом уже апдейтить.
- Иначе говоря,  $\lambda_i$  «тянет» ссылку в выдаче вверх или вниз, и мы апдейтим веса на основе этого.



## 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?

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

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

где  $\Delta_{\text{NDCG}}$  – 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.

- 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).
- Мощная идея: можно не знать функцию, а просто придумать разумные градиенты; чтобы под них существовала функция, в разумных случаях достаточно (лемма Пуанкаре), чтобы сходились вторые частные производные.

# RANKBOOST

---

- RankBoost: задачу ранжирования по массе разных фич, характеризующих документы и запросы, можно решить и бустингом.
- Формально говоря, нам надо обучить функцию  $F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$  по входу  $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  и функции частичных предпочтений  $\Phi : \mathbf{X} \times \mathbf{X} \rightarrow \mathbb{R}$  ( $\Phi(\mathbf{x}_i, \mathbf{x}_j) > 0$ , если  $\mathbf{x}_i$  лучше  $\mathbf{x}_j$ , и т.д.).
- Обычно в качестве функции  $\Phi$  подаётся просто разбиение на «хорошие» (например, релевантные) и «плохие»:  $\Phi(\mathbf{x}, \mathbf{y}) = 1$  для хорошего  $\mathbf{x}$  и плохого  $\mathbf{y}$  и 0, если они из одного множества.

- RankBoost по сути работает примерно как AdaBoost, но раньше веса давали распределение на примерах, а теперь – на парах примеров: инициализируем распределение  $D^{(1)} = D(\mathbf{x}, \mathbf{y})$  на  $\mathbf{X} \times \mathbf{X}$ , а потом для  $m = 1..M$ :
  - обучаем слабую ранжирующую функцию  $h_m(\mathbf{x}) : \mathbf{X} \rightarrow \mathbb{R}$  по распределению  $D^{(m)}$ ;
  - выбираем  $\alpha_m \in \mathbb{R}$  (потом скажу как);
  - пересчитываем новое распределение

$$D^{(m+1)}(\mathbf{x}, \mathbf{y}) = \frac{1}{Z_m} D^{(m)}(\mathbf{x}, \mathbf{y}) e^{\alpha_m (h_m(\mathbf{x}) - h_m(\mathbf{y}))}.$$

- После обучения ранжируем как  $H_M(\mathbf{x}) = \sum_{m=1}^M \alpha_m h_m(\mathbf{x})$ .

- Тогда получится такая теорема: если вернуться от  $D^{(m+1)}(\mathbf{x}, \mathbf{y})$  к  $D(\mathbf{x}, \mathbf{y})$ , будет

$$D^{(m+1)}(\mathbf{x}, \mathbf{y}) = \frac{1}{\prod_m Z_m} D(\mathbf{x}, \mathbf{y}) e^{H_M(\mathbf{x}) - H_M(\mathbf{y})}.$$

- Значит, ошибку можно оценить как

$$\begin{aligned} J_M &= \sum_{\mathbf{x}, \mathbf{y}} D(\mathbf{x}, \mathbf{y}) [H_M(\mathbf{x}) \geq H_M(\mathbf{y})] \leq \\ &\leq \sum_{\mathbf{x}, \mathbf{y}} D(\mathbf{x}, \mathbf{y}) e^{H_M(\mathbf{x}) - H_M(\mathbf{y})} = \sum_{\mathbf{x}, \mathbf{y}} D^{(m+1)}(\mathbf{x}, \mathbf{y}) \prod_m Z_m = \prod_m Z_m. \end{aligned}$$



- И выбирать  $\alpha_m$  можно (и нужно) так, чтобы минимизировать  $\prod_m Z_m$ , т.е. на шаге  $m$  минимизировать

$$Z_m = \sum_{\mathbf{x}, \mathbf{y}} D^{(m)}(\mathbf{x}, \mathbf{y}) e^{\alpha_m (h_m(\mathbf{x}) - h_m(\mathbf{y}))}.$$

- Формально для нас  $h_m$  – чёрный ящик, но на практике мы часто выбираем алгоритм обучения и для  $h_m$ , так что его тоже можно выбрать так, чтобы минимизировать  $Z_m$ .

- Теорема: для любого слабого ранжирования  $h$   $Z(\alpha)$  имеет единственный минимум, так что можно просто бинарным поиском.
- Если  $h \in \{0, 1\}$ , можно и аналитически: обозначим  $W_b = \sum_{\mathbf{x}, \mathbf{y}} D(\mathbf{x}, \mathbf{y}) [h(\mathbf{x}) - h(\mathbf{y}) = b]$ . Тогда

$$\alpha_{\text{opt}} = \frac{1}{2} \ln \left( \frac{W_{-1}}{W_{+1}} \right), \quad Z = W_0 + 2\sqrt{W_{-1}W_{+1}}.$$

- А для  $h \in [0, 1]$  можно приблизить:  $e^{\alpha x} \leq \left(\frac{1+x}{2}\right) e^{\alpha} + \left(\frac{1-x}{2}\right) e^{-\alpha}$ ,  $x \in [-1, 1]$ , так что

$$Z \leq \left(\frac{1-r}{2}\right) e^{\alpha} + \left(\frac{1+r}{2}\right) e^{-\alpha}, \quad r = \sum_{\mathbf{x}, \mathbf{y}} D(\mathbf{x}, \mathbf{y}) (h(\mathbf{x}) - h(\mathbf{y})),$$

и можно выбирать

$$\alpha_{\text{opt}} = \frac{1}{2} \ln \left( \frac{1+r}{1-r} \right),$$

а  $h$  можно обучать так, чтобы максимизировать  $|r|$ .

- Но можно и ещё лучше...

MART

---

- Теперь давайте градиентный бустинг применять к задаче ранжирования более напрямую.
- Начнём с чуть изменённых обозначений для деревьев принятия решений в случае регрессии.
- Рассмотрим датасет  $\mathbf{X} = \{\mathbf{x}_i, y_i\}$ .
- Можно определить *regression stump* (регрессионный пень) так: выбираем одну координату (атрибут)  $j$  (т.е. берём  $x_{ij}$ ) и ищем там оптимальное разбиение – такое значение порога  $t$ , что

$$S_j = \sum_{i \in \text{Left}} (y_i - \mu_{\text{Left}})^2 + \sum_{i \in \text{Right}} (y_i - \mu_{\text{Right}})^2$$

минимизируется, где Left и Right – множества точек слева и справа от  $t$  по  $j$ -й координате.

- Если повторить эту процедуру  $L$  раз (как-то выбирая для расщепления листя – например, по максимальной дисперсии), получится регрессионное дерево с  $L$  листьями.
- В каждом листе определим  $\gamma_l$  – среднее по  $y_i$  из этого листа.
- Тогда, чтобы применить регрессионное дерево, надо по нему спуститься до листа и взять  $\gamma_l$  из этого листа.

- 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  $\alpha_m$ .

- Нам нужно понять, как обучать новое дерево  $F_{m+1}$ , если мы уже обучили  $m$  деревьев.
- Зафиксируем функцию ошибки  $C$  (она дана выше).
- Идея: следующее дерево моделирует производные ошибки по текущей модели в точках из датасета:

$$\Delta C \approx \frac{\partial C(F_m)}{\partial F_m} \Delta F,$$

и  $\Delta C$  будет отрицательным, если взять для  $\eta > 0$

$$\Delta F = -\eta \frac{\partial C(F_m)}{\partial F_m}.$$

- Непараметрический метод: мы рассматриваем  $F_m$  в каждой точке из датасета как «параметр» и по ним оптимизируем.



- Пример: бинарная классификация,  $y_i \in \{\pm 1\}$ ,  $\mathbf{x} \in \mathbb{R}^n$ ,  $F(\mathbf{x})$ .
- Обозначим  $p_+ = p(y = 1 | \mathbf{x})$ ,  $p_- = p(y = -1 | \mathbf{x})$ ,  
 $I_+(\mathbf{x}_i) = [y_i = 1]$ ,  $I_-(\mathbf{x}_i) = [y_i = -1]$ .
- Будем использовать (как и в RankNet, кстати) перекрёстную энтропию

$$L(y, F) = -I_+ \log p_+ - I_- \log p_-.$$

- Логистическая регрессия моделирует log odds:

$$F_N(\mathbf{x}) = \frac{1}{2} \log \left( \frac{p_+}{p_-} \right),$$

$$p_+ = \frac{1}{1 + e^{-2\alpha F_m(\mathbf{x})}}, \quad p_- = \frac{1}{1 + e^{2\alpha F_m(\mathbf{x})}},$$

и мы получаем

$$L(y, F) = \log (1 + e^{-2y\alpha F_m(\mathbf{x})}).$$

- $L(y, F) = \log(1 + e^{-2y\alpha F_m(\mathbf{x})})$ .
- От этой функции легко взять производную по  $F(\mathbf{x})$ :

$$\bar{y}_i = \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_m(\mathbf{x})} = \frac{2y_i\alpha}{1 + e^{2y_i\alpha F_m(\mathbf{x})}}.$$

- И мы строим новое регрессионное дерево, которое пытается смоделировать  $\bar{y}_i$ .

- Осталось только выбрать вес, с которым это новое дерево войдёт в сумму.
- Мы хотим выбрать (примерно) оптимальный шаг для каждого листа, т.е. минимизировать потери:

$$\gamma_{lm} = \arg \min_{\gamma} \log (1 + e^{2y_i \alpha (F_m(\mathbf{x}) + \gamma)}) = \arg \min_{\gamma} g(\gamma).$$

- Минимизировать можно итеративно по методу Ньютона-Рапсона:  $\gamma_{n+1} = \gamma_n - \frac{g'(\gamma_n)}{g''(\gamma_n)}$ .

- И мы аппроксимируем одним шагом этого метода, начиная с нуля:

$$\gamma_{lm} = -\frac{g'}{g''} = \frac{\sum_{x_i \in R_{lm}} \bar{y}_i}{\sum_{x_i \in R_{lm}} |\bar{y}_i| (2\alpha - |\bar{y}_i|)}.$$

**Упражнение.** Проверьте эту формулу.

# LAMBDMART

---

- Теперь уже понятно, как из MART сделать LambdaMART.
- Мы просто добавим в градиенты целевую метрику, например

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

- Функция ошибки нам тоже уже известна:

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

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

- Получается, что знак  $\lambda_{ij}$  зависит только от меток  $i$  и  $j$ , и в каждой точке мы можем собрать все «действующие силы» как

$$\lambda_i = \sum_j \lambda_{ij}.$$

- Это и называется LambdaMART.
- Вариант: LambdaSMART (submodel): мы инициализируем первое дерево какой-нибудь обученной хорошей базовой моделью, а всё дальнейшее – это её уточнение.



- Остался только один вопрос: откуда взять веса  $\alpha_i$  (при  $f_i(\mathbf{x})$ )?
- Базовый LambdaMART выбирает эти веса индивидуально для каждого листа дерева.
- Мы хотим сдвинуться в сторону минимума ошибки; значит, надо идти в сторону нуля производной. Один шаг метода Ньютона-Рапсона:

$$F_m(x_i) = F_{m-1}(x_i) + v \sum_l \gamma_{lm} [x_i \in R_{lm}],$$

где  $\gamma_{lm} = \frac{F'_m(x_i)}{F''_m(x_i)}$ , а  $v$  – регуляризатор.

- Первая производная – это просто  $\lambda_i$ ; вторая производная –  $\lambda'_i$ :

$$\gamma_{lm} = \frac{\sum_{x_i \in R_{lm}} \lambda_i}{\sum_{x_i \in R_{lm}} \frac{\partial \lambda_i}{\partial F_m(x_i)}}$$

- И теперь можно собрать весь алгоритм целиком.

1.  $F_0(x_i) = \text{BaseModel}(x_i)$ ,  $i = 0..N$ .
2. Для  $m$  от 1 до  $M$  (числа деревьев в сумме):
  - 2.1  $y_i = \lambda_i = \sum_j \lambda_{ij}$ ,  $i = 0..N$  (считаем градиенты);
  - 2.2  $w_i = \frac{\partial y_i}{\partial F(x_i)}$ ,  $i = 0..N$  (вторые производные);
  - 2.3 строим новое дерево  $\{R_{lm}\}_{l=1}^L$  на вершинах  $\{y_i, x_i\}_{n=1}^N$ ;
  - 2.4  $\gamma_{lm} = \frac{\sum_{x_i \in R_{lm}} y_i}{\sum_{x_i \in R_{lm}} w_i}$ ,  $l = 1..L$  (веса узлов);
  - 2.5  $F_m(x_i) = F_{m-1}(x_i) + v \sum_l \gamma_{lm} [x_i \in R_{lm}]$ ,  $i = 0..N$ .

- В Lambda[S]MART веса бустинга подбираются как шаг метода Ньютона для каждого листа дерева.
- Но благодаря тому, что наши IR-метрики дискретные, можно просто подобрать оптимальный  $\alpha_m$ .
- Давайте рассмотрим общую задачу: предположим, что у нас есть две ранжирующие функции,  $R$  и  $R'$ , и мы хотим их оптимально линейно скомбинировать, т.е. подобрать оптимальный коэффициент  $\alpha$  в

$$s_i = (1 - \alpha)s_i^R + \alpha s_i^{R'}$$

- Идея простая: представьте себе, как  $\alpha$  меняется от 0 (в чистом виде  $R$ ) до 1 (в чистом виде  $R'$ ).
- Тогда метрики вроде NDCG реально меняются только в точках пересечения (да и то не всегда, а только если пересеклись документы с разными метками).
- Ну вот и давайте просто переберём все такие пары, найдём их точки пересечения и составим список интересных значений  $\alpha$ .
- А затем отсортируем список, подсчитаем метрику в каждом интервале и найдём оптимальный  $\alpha$ ; для ситуации бустинга просто надо это делать на каждом шаге.
- Кажется, что это очень тупо, но на самом деле это квадратичный алгоритм, что не так уж страшно.

- LambdaMART – победитель в Yahoo! Learning to Rank Challenge (2011).
- Что было нового с тех пор?
- Plackett-Luce model выражает ранжирование в виде вероятностей:
  - выбираем первый документ с вероятностью, пропорциональной релевантности;
  - выбираем второй документ из остальных тоже пропорционально релевантности...
- Это даёт непрерывную listwise-ошибку, которую можно оптимизировать.
- BayesRank оптимизирует её, MatrixNet, видимо, тоже.

Спасибо за внимание!