

Порождение структурно простых ранжирующих функций для задач информационного поиска

Кулунчаков Андрей

Московский физико-технический институт
Факультет управления и прикладной математики
Кафедра "Интеллектуальные системы"

Научный руководитель д.ф.-м.н. В. В. Стрижов

Москва,
2015 г.

Цель

Предложить способ порождения ранжирующих функций произвольной структуры и высокое значение функционала MAP.

Проблема

Переборный алгоритм порождает только структурно простые функции. Известные алгоритмы генетического поиска дают переусложненные модели и быстро стагнируют в локальные минимумы.

Предлагаемое решение

Предлагается использовать регуляризатор при оценке качества модели, порождаемой генетическим алгоритмом. При этом с помощью метрики на множестве моделей определяется момент начала стагнации.

Goswami et al. Exploring the space of ir functions, 2014

Переборный алгоритм вычислительно сложен. Анализируются только функции с очень низкой структурной сложностью.

Fan et al. A generic ranking function discovery framework by genetic programming for information retrieval, 2004.

Генетический алгоритм порождения застревает в локальных минимумах, итоговые функции переусложнены.

Billhardt P. et al. Using genetic algorithms to find suboptimal retrieval expert combinations, 2002.

Генетический алгоритм поиска линейной комбинации ранжирующих функций возвращает параметры, существенно зависящие от коллекции.

Задача

Коллекция документов C экспертно отранжирована по релевантности запросам Q — $g : \bar{C} \times Q \rightarrow \{0, 1\}$. Необходимо аппроксимировать зависимость g некоторой явной функцией.

Функционал качества F

$$\text{AveP}(n) = \frac{\sum_{k=1}^n (P(k) \times \text{rel}(k))}{\text{rel docs}}, \quad \mathcal{F} = \frac{\sum_{q=1}^Q \text{AveP}(q)}{|Q|}.$$

Постановка задачи

Оптимизационная задача:

$$f^* = \operatorname{argmax}_f (\mathcal{F}(f) - R(f)),$$

где R — регуляризатор, а максимизация проводится по всем суперпозициям над порождающими G .

Аргументы ранжирующей функции

Переменные в G — признаки пары документа и слова (d, w) :

$$x_w^d = t_d^w \log\left(1 + \frac{l_{avg}}{l_d}\right), \quad y_w = \frac{N_w}{N},$$

где N_w — # документов, содержащих w ; t_d^w — число слов w в d , l_d — длина d , l_{avg} — средняя длина документа в коллекции.

При этом значение функции f на паре (q, d) определяется:

$$f(q, d) = \sum_{w \in q} f(w, d) = \sum_{w \in q} f(x_w^d, y_w)$$

Критерии качества

Идеальная ранжирующая модель h такова, что для лучших моделей $\{f_i\}$ из (Goswami 2014) выполняется: $\mathcal{F}(h) > \mathcal{F}(f_i)$ — модель h равномерно лучше, чем все эталонные функции $\{f_i\}$.

Структурная сложность суперпозиции

Суперпозиция (модель, функция) — функция, составленная из порождающих элементов G .

Структурная сложность суперпозиции (модели, функции) — число порождающих элементов.

Предлагаемый подход

- С помощью генетического алгоритма, предложенного в работе (Fan et al., 2004), исследуется множество суперпозиций структурной сложности большей, чем 8.
- Используются регуляризаторы для контроля структурной сложности получаемых моделей.
- Используются метрики на моделях, чтобы определять момент стагнации в локальный минимум и выбивать алгоритм из него.

Алгоритм порождения ранжирующих функций

1 Создаётся начальное множество случайных моделей.

2 Часть моделей скрещиваются — обмениваются случайными подмоделями:

$$f = \sin(x) + \cos(xy), \quad g = \cos(x) + (x+y) \rightarrow \\ \rightarrow f' = \sin(x) + (x+y), \quad g' = \cos(x) + \cos(xy).$$

3 Некоторые модели мутируют — часть модели заменяется на случайную функцию:

$$f(x, y) = \sin(x) + \cos(xy) \rightarrow f' = \sin(x) + \ln(y).$$

4 Определяется, попал ли алгоритм в локальный минимум. В случае попадания удаляем часть моделей и добавляем множество случайных.

5 Выбирается некоторое число лучших моделей согласно функционалу качества $(\mathcal{F} - R)(f)$. Если требуемая точность достигнута, алгоритм останавливается. Иначе, возврат на 2 шаг.

Имеется множество моделей $P = \{f_j\}_{j=1}^{|P|}$ и метрики $\mu(f_i, f_j)$.
Значение μ на всем множестве P определяется как:

$$\mu(P) = \frac{\sum_{k < j} \mu_i(f_k, f_j)}{avlen(P)},$$

где средняя длина моделей в множестве P :

$$avlen(P) = \frac{1}{|P|} \sum_{j=1}^{|P|} |f_j|.$$

Попадание в локальный минимум определяется, как опускание значения $\mu(P)$ ниже порога `Thresh`, который определяется эмпирически.

Три метрики μ_1, μ_2, μ_3

Суперпозиции f_i, f_j представляются в виде помеченных деревьев T_i и T_j .

μ_1

$$\mu_1(f_i, f_j) = |T_i| + |T_j| - 2|T_{ij}|,$$

где T_{ij} — наибольший общий подграф деревьев T_i и T_j .

μ_2

Вторая метрика $\mu_2(f_i, f_j)$ — редакторское расстояние между строковыми представлениями деревьев T_i и T_j .

μ_3

Третья метрика $\mu_3(f_i, f_j)$ — редакторское расстояние между самими деревьями T_i и T_j .

Для контроля структурной сложности модели в функционале качества присутствует регуляризатор. Исследуются три варианта (m — значение $\text{MAP}(f)$):

- 1 $R_1(f) = p \cdot m \cdot I(|f| < CT)$,
где CT — пороговая сложность модели, $p \in (0, 1)$.
- 2 $R_2(f) = p \cdot m \cdot I(|f| \geq CT) \cdot (|f| - CT)$,
где $C > 0$ — некоторая константа.
- 3 $R_3(f) = p \cdot m \cdot |f|^* \cdot \log(|f| + 1)$,
где $|f|^*$ — количество переменных x, y в модели.

Выборки

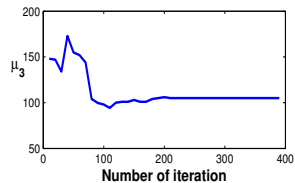
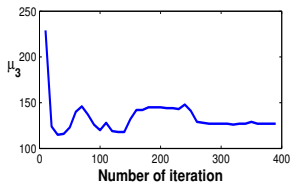
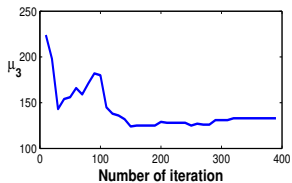
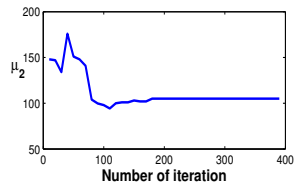
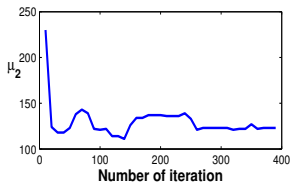
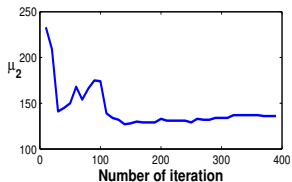
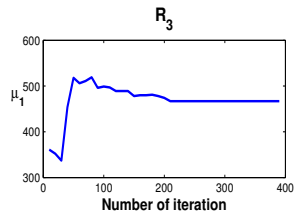
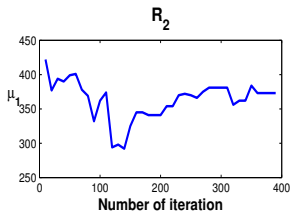
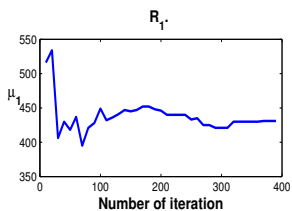
Используются выборки Trec5-8 для анализа моделей, генерируемых генетическим алгоритмом. При этом Trec7 — обучающая, а остальные — контрольные. Все выборки имеют примерно по 500 000 документов и 50 запросов к ним.

Вычислительный эксперимент

Запускается алгоритм при разных регуляризаторах на выборке Trec7. Выбирается регуляризатор, который наиболее отдаляет точку стагнации и при этом не позволяет моделям значительно переусложняться. Выбирается метрика, которая наиболее точно определяет начало стагнации.

После этого запускается алгоритм снова и итоговые модели сравниваются с теми, что были отобраны в Goswami, 2014.

Анализ метрик и регуляризаторов



- 1 Метрика μ_1 не отражает момент стагнации — на графике нет заметных минимумов и последующих прямолинейных участков.
- 2 Метрики μ_2 и μ_3 фактически неотличимы по динамике. Выбирается μ_2 , как более эффективно вычисляемое.
- 3 Регуляризатор R_1 слишком жесткий — алгоритм стагнирует уже после первых итераций.
- 4 Похожая ситуация наблюдается и для R_2 — метрики существенно уменьшаются уже на первых итерациях.
- 5 Напротив, с регуляризатором R_3 уменьшение более плавное. Далее используем **третий регуляризатор**.

Итоговые функции на Тrec7(μ_2, R_3)

После 600 итераций отобраны функции $\{h_j\}$ ($\ln(x) \rightarrow \ln(x+1)$, а $g(x) = \ln \ln(x)$). Модели $\{h_j\}$ сравниваются с функциями $\{f_i\}$, признанными лучшими в работе Goswami, 2014. Функции $\{f_i\}$ в среднем на множестве коллекций лучше, чем все модели сложностью не более 8 и известные ранжирующие модели *BM25*, *LGD*, *LM_{DIR}*:

#	Эталонные функции	#	Итоговые функции
f_1	$e^{\sqrt{\ln(x/y)}}$	h_1	$g\left(\frac{g(x)}{\sqrt{\ln(x)+x}}\right) - \ln(y)$
f_2	$\sqrt{\frac{\ln(x)}{\sqrt{y}}}$	h_2	$g\left(\frac{g(x)}{\sqrt{\frac{1}{2}\ln(x)+x}}\right) - \ln(y)$
f_3	$\sqrt[4]{\frac{x}{y}}$	h_3	$g\left(\ln\left(\frac{g(x)}{\sqrt{\frac{1}{2}\ln(x)+x}}\right) - \ln(y)\right)$
f_4	$\sqrt{y + \sqrt{\frac{x}{y}}}$	h_4	$g\left(\frac{g(x)}{\sqrt{g(\sqrt{x})+x}}\right) - \ln(y)$
f_5	$\sqrt{\sqrt{\frac{x}{y}} \cdot e^{-y}}$	h_5	$g\left(\frac{g(x)}{\sqrt{\ln(x)+\ln(y)}}\right) - \ln(y)$
f_6	$\sqrt{\sqrt{x} + \sqrt{\frac{x}{y}}}$	h_6	$g\left(\frac{g(\ln(x))}{\sqrt{\ln(x)+x}}\right) - \ln(y)$

Сравнение функций на разных выборках

Значение MAP для функций f_i и h_j				
Функция	Trec5	Trec6	Trec7	Trec8
f_1	8.79	13.715	10.038	13.902
f_2	8.52	13.00	9.22	13.07
f_3	8.91	13.62	9.91	13.71
f_4	8.91	13.62	9.91	13.71
f_5	8.91	13.62	9.91	13.71
f_6	8.87	13.61	9.89	13.70
h_1	8.97	13.69	10.60	14.40
h_2	9.47	13.72	10.65	14.40
h_3	9.56	13.79	10.63	14.38
h_4	9.23	13.71	10.50	14.37
h_5	8.86	13.39	10.44	14.36
h_6	8.10	13.48	10.42	14.36

Полученные функции заметно улучшают известные модели из Goswami, 2014. Модели $h_{1,2,3,4}$ лучше их даже на всех тестовых коллекциях.

Хотя структурная сложность моделей h_j примерно в два раза выше, чем у функций f_i , h_j имеют компактный вид.

Структурная сложность f_i и h_j			
Функция	Сложность	Функция	Сложность
f_1	6	h_1	14
f_2	6	h_2	15
f_3	5	h_3	16
f_4	7	h_4	16
f_5	8	h_5	15
f_6	8	h_6	15

- 1 Разработан генетический алгоритм поиска ранжирующих функций, который не склонен к попаданию в локальные минимумы.
- 2 Порождаемые модели не переусложнены и имеют компактное описание.
- 3 Получаемые ранжирующие функции равномерно лучше структурно простых эталонных моделей на рассматриваемом множестве тестовых коллекций.