Московский Физико-Технический Институт
(Государственный Университет)

Факультет Управления и Прикладной Математики
Кафедра «Интеллектуальные Системы»

ДИПЛОМНАЯ РАБОТА СТУДЕНТА 174 ГРУППЫ

«Порождение структурно простых ранжирующих функций для задач информационного поиска»

Выполнил:
студент 4 курса 174 группы
Кулунчаков Андрей Сергеевич

Научный руководитель:
доц. каф. Интеллектуальные системы,
к. ф.-м. н.
Стрижов Вадим Викторович

Москва, 2015

Содержание

Аннотация

В работе строится алгоритм порождения ранжирующих функций для задачи информационного поиска. Ранжирующие функции ищутся в виде суперпозиции заданных порождающих функций и переменных. Используется генетический алгоритм порождения суперпозиций. Структурная сложность получаемых моделей контролируется регуляризатором. С помощью метрики на множестве моделей определяется момент попадания в локальный минимум. Итоговые функции сравниваются относительно функционала MAP со структурно-простыми суперпозициями, полученными алгоритмом полного перебора. Вычислительный эксперимент проводится на выборке, состоящей из коллекций документов Trec5-8, запросов к каждой из них и экспертных оценок релевантности.

Ключевые слова: информационный поиск, TREC, ранжирующая функция, генетический алгоритм, символьная регрессия.

1 Введение

Во многих задачах информационного поиска необходимо, получая на вход запрос, сформировать выборку документов, возвращаемых в порядке убывания релевантности данному запросу. Для формирования такой выборки необходима процедура оценки релевантности документов. Один из методов такого оценивания — использование ранжирующей функции. Ранжирующая функция определяется на признаках пары запрос-документ и согласно этим признакам формирует оценку релевантности соответствующего документа из пары. Таким образом, присвоив каждому документу некоторую оценку его релевантности, функция формирует выборку, возвращаемую соответствующему запросу.

Многие современные модели оценивания документов были получены теоретически [1, 2, 3, 4, 5, 6]. Однако подобный подход к поиску ранжирующих моделей не является единственно возможным. Более того, модели, получаемые таким образом, являются неоптимальными, и было показано существование других ранжирующих моделей, которые в среднем на множестве коллекций были лучше [7] согласно функционалу качества MAP. В работе [7] использовался переборный алгоритм поиска таких моделей. Последние представлялись в виде суперпозиций элементов заданной грамматики — набора порождающих функций и переменных. Переменные играли роль признаков пары документ-запрос: например, нормализованная частота встречаемости слова из запроса в документе.

Из-за высокой вычислительной сложности переборный алгоритм [7] удается применить для поиска только структурно простых суперпозиций, состоящих из не более чем 8 элементов грамматики. Данный подход к порождению моделей гарантирует оптимальность решения в рассматриваемом множестве функций. Более того, получаемая функция гарантированно является структурно простой.

В данной работе предлагается сохранить плюсы данного подхода [7] и одновременно снизить вычислительную сложность. Для этого предлагается генетический алгоритм порождения ранжирующих функций.

Использование генетических алгоритмов в задаче информационного поиска не является новой идеей. Ранее было предложено использовать генетический подход к поиску оптимального описания документов при индексировании [8, 9], при решении

задачи кластеризации документов, ко-релевантных для множества запросов [10, 11]. Также генетические алгоритмы использовались для настройки параметров запросов [12, 13], для нахождения множества ключевых слов для бинарного векторного описания документа [14], для нахождения оптимальных коэффициентов для линейной суперпозиции ранжирующих функций [15, 16].

Были попытки применения генетического алгоритма и для порождения суперпозиций элементов заданной грамматики [17, 18]. Отличительной особенностью этих работ является сложность получаемых суперпозиций — в работах не использовались регуляризаторы и ничего не говорится о решении проблемы стагнации — попадания в локальные минимумы. В результате итоговые модели получались переусложненными и могли оказаться существенно неоптимальными.

Свойство застревания в локальных минимумах (*стагнация*) заключается в том, что с некоторого момента популяция преимущественно состоит из одинаковых или очень схожих по структуре моделей. При этом функции, получаемые на новых итерациях, по построению будут иметь схожую структуру с уже имеющимися. Таким образом, алгоритм перестает генерировать принципиально новые модели.

В настоящей работе предлагается выбивать популяцию из локальных минимумов добавлением к ней новых случайных моделей. Остается проблема определения момента попадания популяции моделей в локальный минимум. Для этого предлагается ввести метрику на моделях и определять степень совокупной различности моделей по получаемой матрице расстояний.

Кроме этого, предлагается использование регуляризаторов для контроля сложности получаемых функций. Приводится несколько функционалов, для каждого из которых получаемые модели сраниваются между собой.

Главная цель работы: найти структурно-простую функцию, недоступную переборному алгоритму [7], которая в среднем на множестве коллекций работает лучше эталонных функций, полученных в [7].

2 Постановка задачи.

Пусть дана коллекция C, состоящая из множества документов $\{d_i\}_{i=1}^{|C|}$. Коллекции C соответствует множество запросов $Q = \{q_i\}_{i=1}^{|Q|}$. Для каждого запроса q_i часть коллекции экспертно отранжирована: есть некоторое подмножество коллекции $C_i \subset C$, на котором экспертно заданы бинарные ранки g:

$$g : Q \times C_i \to \mathbb{Y} = \{0, 1\},$$

где 1 соответствует релевантному документу, а 0 — нерелевантному. Цель работы — найти явный вид функции g, либо аппроксимировать ее некоторой функцией.

Далее мы будем искать это приближение в множестве суперпозиций над грамматикой. Грамматика представляет собой множество порождающих функций $\mathcal{G} = \{g_i\}_{i=1}^{|G|}$ и переменных $X = \{x_i\}_{i=1}^{|X|}$. В данной работе, число переменных $|X| = 2$.

Рассматриваются множества непараметрических суперпозиций. Переменные предлагается выражать, исходя из информации о запросе и документе. А именно, полагать, что первая переменная x — нормализованная частота встречаемости слова w в документе d, а другая переменная y — нормализованная частота встречаемости слова w в коллекции:

$$x_w^d = t_d^w \log\bigl(1 + c \cdot \frac{l_{avg}}{l_d}\bigr),\ y_w = \frac{N_w}{N},$$

где N_w — количество документов в коллекции, содержащих слово w, N — общее число документов в коллекции, t_d^w — частота встречаемости слова w в документе d, l_d — длина документа в коллекции, l_{avg} — средняя длина документа в коллекции, c — некоторый параметр.

Чтобы определить суперпозицию на паре документ-запрос, используется следующее равенство:

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

Качество ранжирующей функции характеризуется функционалом MAP — $m(f, C, Q)$:

$$AveP(f, C, Q, n) = \frac{\sum_{k=1}^{n}(P(k) \times \mathrm{rel}(k))}{\#\ \mathrm{rel\ docs}},$$

$$m(f, C, Q) = \frac{\sum_{q=1}^{Q} \mathrm{AveP(q)}}{|Q|}.$$

Требуется найти функцию f, которая решает задачу максимизации:

$$f^* =_f (m(g, C, Q) - R(f)),$$

где R — регуляризатор, контролирующий структурную сложность модели, а максимизация проходит на множестве суперпозиций элементов заданной грамматики G.

В работе [7] порождаются случайные модели, причем наложено ограничение на их структурную сложность — не более 8 порождающих функций в суперпозиции. Обозначим множество отобранных в [7] лучших суперпозиций за $\mathcal{F}$.

Качество ранжирующей функции существенно зависит от коллекции документов и запросов, по которой она оценивается. Подход, описанный в данной работе, будет считаться успешным, если в итоге будет порождена модель h такая, что на рассматриваемом множестве коллекций $\{(C \times Q)_j\}$ выполнено:

$$\mathrm{avqual}(h) > \mathrm{avqual}(f_i)\ \forall f_i \in \mathcal{F},$$

где $\mathrm{avqual}(f)$ — среднее значение функционала $m(f, C, Q)$ на множестве коллекций $\{(C \times Q)_j\}$.

3 Описание данных

Вычислительный эксперимент будет проводиться на данных конференций Trec 2005-2008 годов (trec.nist.gov). На конференциях Trec командам исследователей предлагается большая коллекция документов и множество запросов. Необходимо оценить релевантность документов коллекции соответствующим запросам (на упомянутых конференциях релевантность — бинарная величина). Получаемые командами оценки релевантностей судит специальное жюри, которое составляет итоговый список

оценок релевантностей документов. Таким образом, результатом проведения конференции является триплет (коллекция-запросы-оценки). Этот триплет мы и будем называть *выборкой* Trec. Цифра после Trec указывает год проведения соответствующей конференции.

Все 4 выборки имеют примерно по 500 000 документов, по 50 запросов к ним и в среднем по 2000 оценок релевантности на каждый запрос.

4 Порождение ранжирующих суперпозиций

4.1 Обработка данных

Работа с выборками Trec проводится с помощью платформы Terrier IR Platform v3.5 (terrier.org). Terrier производит первичную обработку текстовой коллекции, индексирет её для ускорения обращения к документам, исполняет обращения по запросам к индексированной коллекции. Результатом обращения к текстовой коллекции по запросу q является набор соответствующих признаков x_w^d и y_w для каждого слова $w \in q$ и всех документов, имеющих хотя бы одно вхождение w.

Алгоритм первичной обработки данных, исполняемый платформой Terrier состоит из следующих этапов:

1. Документы разбиваются на токены, каждый из которых приводится к основной форме с помощью стемминга [2].

2. Полученное множество токенов фильтруется согласно списку стоп-слов.

3. Коллекция представляется в виде индекса документ-токен.

4. Создается лексикон, представляющий собой словарь токенов, записанных в индексе вместе с соответствующей статистикой.

5. Метаданные о коллекции собираются во время индексирования коллекции и записываются в отдельный файл.

Обработка запроса платформой Terrier выполняет следующий алгоритм:

1. Запрос разбивается на токены, которые обрабатываются с помощью стемминга и стоп-слов.

2. С помощью лексикона собирается статистика о токенах. Считается переменная y.

3. Токены с высоким значением переменной y удаляются из запроса, как малоинформативные.

4. Из индекса выделяется информация о переменной x для каждого из оставшихся токенов запроса и каждого документа коллекции.

4.2 Генетический алгоритм порождения

Введём ряд определений, необходимых для дальнейшего описания применяемого метода.

Определение 1. *Модель (ранжирующая функция, ранжирующая суперпозиция) — суперпозиция элементов заданной грамматики G. Далее понятия модели, функции и ранжирующей функции принимаются эквивалентными.*

Определение 2. *Популяция — множество моделей, анализируемых генетическим алгоритмом на конкретной итерации. От итерации к итерации популяция изменяется.*

Определение 3. *Локальный минимум — итерация, на которой соответствующая популяция состоит из одинаковых или структурно похожих моделей.*

Определение 4. *Стагнация — застревание популяции в локальном минимуме. Популяция стагнирует (алгоритм стагнирует), если при попадании в локальный минимум, она из него больше не выходит.*

Имея значения переменных, описанных в постановке задачи, можем вычислять ранки документов с помощью произвольной ранжирующей суперпозиции. Фиксируем грамматику G, состоящую из переменных двух типов и порождающих функций — унарных и бинарных. Функция аппроксимирующая заивисимость g ищется в виде суперпозиции элементов грамматики.

Используется генетический алгоритм порождения моделей из [17]. На нулевой итерации создаётся начальное множество случайных моделей. Далее итерационно генерируются новые модели по моделям из предыдущей итерации. Для этого используются методы скрещивания и мутации.

Определение 5. *Скрещивание двух моделей $f_i,\ f_j$, представленных в виде помеченных деревьев $T_i,\ T_j$, — получение двух новых моделей $\bar{f}_i,\ \bar{f}_j$ путём обмена соответствующих деревьев двумя случайно выбранными поддеревьями.*

Пример скрещивания:

$$f(x,y) = \sin(x) + \mathbf{cos(xy)},\ g(x,y) = \cos(x) + \mathbf{(x+y)}$$

$$\downarrow$$

$$f`(x,y) = \sin(x) + \mathbf{(x+y)},\ g`(x,y) = \cos(x) + \mathbf{cos(xy)},$$

где жирным шрифтом выделены случайно выбранные поддеревья. Новые функции формируются перестановкой этих поддеревьев между собой.

Определение 6. *Мутация модели f_i, представленной в виде помеченного дерева T_i, — получение новой модели $\bar{f}_i$ путём замены случайно выбранного поддерева T_i новым случайным деревом.*

Пример мутации:

$$f(x,y) = \sin(x) + \mathbf{cos(xy)} \quad \rightarrow \quad f`(x,y) = \sin(x) + \mathbf{ln(y)}.$$

Таким образом, получается новое множество моделей. После некоторого числа итераций окажется, что алгоритм стагнирует. Для определения попадания в локальный минимум используем значение метрики на всевозможных парах моделей популяции. Если соответствующее значение оказалось ниже некоторого порога, алгоритм считается стагнирующим. Из популяции удаляется часть худших моделей, добавляется множество случайных.

Выбираются лучшие модели согласно функционалу качества $m(f,C,Q) - R(f)$. Если лучшая модель удовлетворяет требуемой точности относительно $m(f,C,Q)$, завершаем алгоритм.

Объединяем сказанное в единую схему:

1. Создаётся начальное множество случайных моделей.

2. Случайные пары моделей множества скрещиваются между собой, часть моделей мутируют — получается новое множество моделей.

3. Если установлено, что алгоритм стагнирует, заменяем часть худших моделей случайными.

4. Выбираются лучшие модели согласно функционалу качества $\big(\mathcal{F} - \mathrm{R}\big)(f)$.

5. Если требуемая точность согласно функционалу $m(f, C, Q)$ достигнута, алгоритм останавливается. Иначе, возврат на 2 шаг.

5 Функции растояния между моделями

5.1 Мотивация

Добавление новых случайных моделей в популяцию может решить проблему стагнации. При этом если на каждой итерации заменять худшие модели случайными, то будет теряться важная накопленная информация. Если же опоздать с заменой, то впустую будет потеряно время, поскольку стагнирующая популяция фактически не пополняется новыми моделями. Поэтому необходимо как можно точнее определять момент начала стагнации.

Решение этой проблемы возможно при помощи метрики на множестве моделей. Данная функция даёт оценку различности функций в популяции. При этом необходима именно структурная оценка различности моделей, то есть оценка визуальной похожести функций в популяции.

Модели можно представить в виде помеченного корневого дерева. При этом метками вершин являются элементы грамматики G. Далее считаем, что модель представлена в виде именно такого дерева. Тогда для задания метрики на моделях достаточно задать метрику на помеченных деревьях.

В работе используются три метрики.

5.2 Общий подграф суперпозиций

Первая метрика μ_1 использует понятие общего подграфа двух деревьев [19].

Определение 7. *Два дерева имеют общий подграф, если у них есть два подграфа, изоморфных друг другу. Этот подграф и называется общим подграфом двух деревьев.*

Определение 8. *Мощностью дерева T назовём число вершин в нем и обозначим как $|T|$.*

Среди всех общих подграфов двух деревьев T_i и T_j выделим подграф T_{ij} с наибольшим числом вершин и назовем его *наибольшим*. Расстоянием между двумя моделями, представленными в виде покрашенных деревьев T_i и T_j назовем числовую функцию:

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

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

Стоит отметить, что в работе [19] доказаны свойства метрики для μ_1 в случае, когда мощность дерева (графа) определяется как количество рёбер в нем. Но для деревьев эти два определения метрики эквивалентны, так как число ребер на 1 меньше числа вершин. Поэтому в нашем случае μ_1 действительно является метрикой.

5.3 Редакторские расстояния

Другие метрики $\mu_2,\ \mu_3$ связаны с расстоянием Левенштейна. Изначально расстояние Левенштейна определено на строках.

Определение 9. *Расстояние Левенштейна между двумя строками — это минимальное количество операций вставки, удаления и замены одного символа на другой, необходимых для превращения одной строки в другую.*

Считаем, что функция μ_2 сопоставляет помеченным деревьям строки и возвращает расстояние между этими строками [20]. Строка является последовательностью меток вершин, возникающих при обходе дерева алгоритмом в глубину. Ясно, что μ_2 является метрикой.

Третья метрика μ_3 непосредственно переносит принцип измерения расстояния между строками на помеченные деревья [21]:

Определение 10. *Расстояние Левенштейна между двумя деревьями — это минимальное количество операций добавления, удаления и перекрашивания вершин, необходимое для того, чтобы деревья стали изоморфными.*

Функция μ_3 является метрикой, что показано в работе [21].

Все три функции μ_i ($\forall i \in \{1, 2, 3\}$) используются для оценки "сходства"между моделями. А именно, если имеется популяция $P = \{f_j\}_{j=1}^{|P|}$, то значение функции μ на всей популяции P определяется как

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

где средняя длина моделей в популяции P: $avlen(P) = \frac{1}{|P|} \sum_{j=1}^{|P|} |f_j|$. Она играет роль нормировки: все функции μ в среднем линейны по структурам своих аргументов.

Момент стагнации определяется, как $\mu_i(P) < \text{Thresh}$, где порог Thresh находится эмпирически.

6 Ругуляризаторы

Эволюционирование начальной популяции существенно зависит от наличия регуляризаторов в оценке качества моделей. В отсутствие регуляризаторов получаемые суперпозиции структурно усложняются от итерации к итерации: уже после 30-40 итераций получаемые модели громоздки. При этом переусложненная модель имеет больше степеней свободы — чем большее число элементов грамматики G могут содержать суперпозиции, тем больше различных моделей можно составить. И, несмотря на типичное для генетического алгоритма стремление к локальному минимуму, популяция оказывается богатой существенно различными моделями.

Тем не менее отсутствие регуляризаторов приводит к переобучению и получаемые суперпозиции в подавляющем своем большинстве имеют сложную структуру и неинтерпретируемы. И наоборот, если регуляризатор окажется слишком сильным, то множество моделей, достижимых из начальной популяции, окажется слишком

бедным — генетический алгоритм будет вынужден брать схожие модели и в итоге стагнировать.

Итак, генетический алгоритм использует регуляризаторы при оценке качества порождаемых функций, чтобы контролировать их структурную сложность.

Определение 11. *Структурная сложность модели — число элементов грамматики G в ней.*

В работе анализируется три регуляризатора, штрафующих суперпозиции, основываясь на их структуре. При этом все они пропорциональны значению функионала $m(f, C, Q)$, для краткости обозначаемого m.

1. $$R_1 = p \cdot m \cdot I(|f| < CT),$$

   где CT — пороговая сложность модели, p — некоторый параметр из интервала (0,1). Функционал R_1 уменьшает ранк моделей, сложность которых превосходит порог, оставляя при этом возможность получения очень точных моделей.

2. $$R_2 = p \cdot m \cdot I(|f| \geq CT) \cdot (|f| - CT),$$

   где $C > 0$ — некоторая константа. Функционал R_2 так же штрафует пропорционально значению m, но теперь размер штрафа увеличивается для структурно сложных функций.

3. $$R_3 = p \cdot m \cdot |f|^* \cdot \log(|f| + 1),$$

   Функционал R_3 тоже штрафует модели, которые достигают определенного уровня структурной сложности. Но в этом случае сложностью модели считается произведение высоты дерева суперпозиции на $|f|^*$ — количество переменных в модели. Это объясняется тем, что интерпретируемость суперпозиции зависит в основном от количества переменных в ней. Использование в регуляризаторе оценки высоты дерева исключает возможность переобучения.

Для оценки константы C будем моделировать вычисления Q на сетке различных значений. Выберем из них то, которое даёт максимум функционала Q. Пороговое значение структурной сложности CT выберем равным использованному в работе [7]. Считаем, что функции, которые проще CT, являются структурно простыми.

7 Вычислительный эксперимент

7.1 Анализ метрик и регуляризаторов

Как было отмечено ранее, алгоритм использует метрики, чтобы определить степень совокупной различности моделей в популяции. Мы имеем три метрики и три варианта регуляризаторов. В этом параграфе мы анализируем каждую из 9 возможных комбинаций выбора пары метрика-регуляризатор. На основе полученных результатов мы выбираем лучшую пару. При этом естественное требование к метрике μ: уменьшение значения, начиная с некоторой итерации. Это отражает начало стагнации алгоритма. При выборе регуляризатора отбрасываются слишком жесткие или слабые.

Оценка моделей проводится на выборке Trec7. Для генетического алгоритма с оценкой качества модели без регуляризаторов проводится 300 итераций, поскольку уже для 300-ой итерации средняя длина суперпозиции (количество порождающих функций в ней) превышает 40. Для каждого из регуляризаторов сравниваются динамики значений метрик на популяции.

Как и ожидалось, в отсутствие регуляризатора модели достаточно различны — расстояния между ними уменьшаются медленно. При этом средняя длина суперпозиции растет от итерации к итерации, что еще раз подтверждает переобучаемость моделей — ранжирующие суперпозиции длины 40 в любом случае громоздки и абсолютно неинтерпретируемы.

Приведенная ниже таблица содержит значения среднего времени вычисления значений метрик в зависимости от регуляризатора.

Среднее CPU, затраченное на вычисление μ			
# регуляризатора	μ_1	μ_2	μ_3
1	11.52	1.84	4.54
2	6.7876	0.9347	1.57
3	7.63	1.05	1.87

Итак, метрика μ_1 заметно хуже двух остальных. Не только в том, что она вычислительно сложнее, но и в том, что она не отражает момент начала стагнации — на графике нет заметных минимумов с последующей константной зависимостью. Поэтому в этом случае не представляется возможным определить значение порога Thresh.

Две другие метрики μ_2, μ_3, как видно из графиков, почти неотличимы: относительная разница не более 5 процентов для любого из регуляризаторов. Поэтому выберем метрику μ_2 как более эффективно вычислимую.

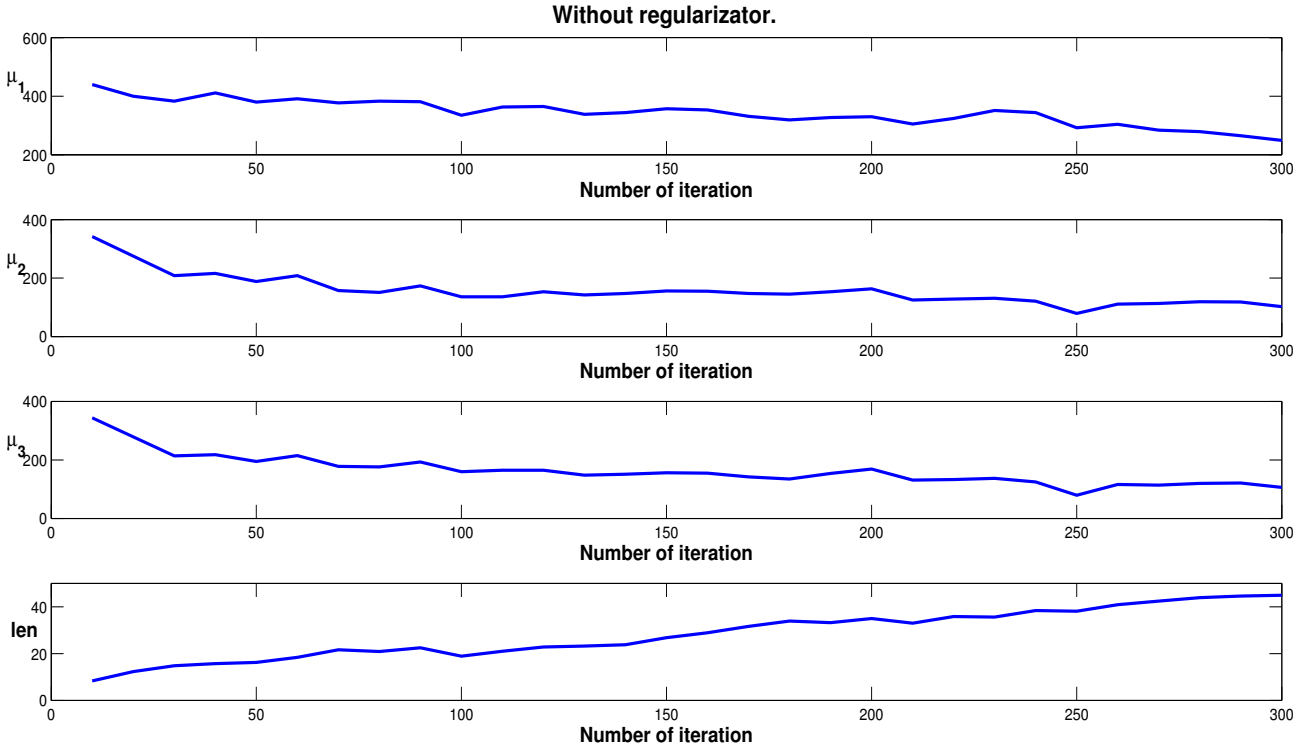

График 1. Алгоритм без регуляризаторов.

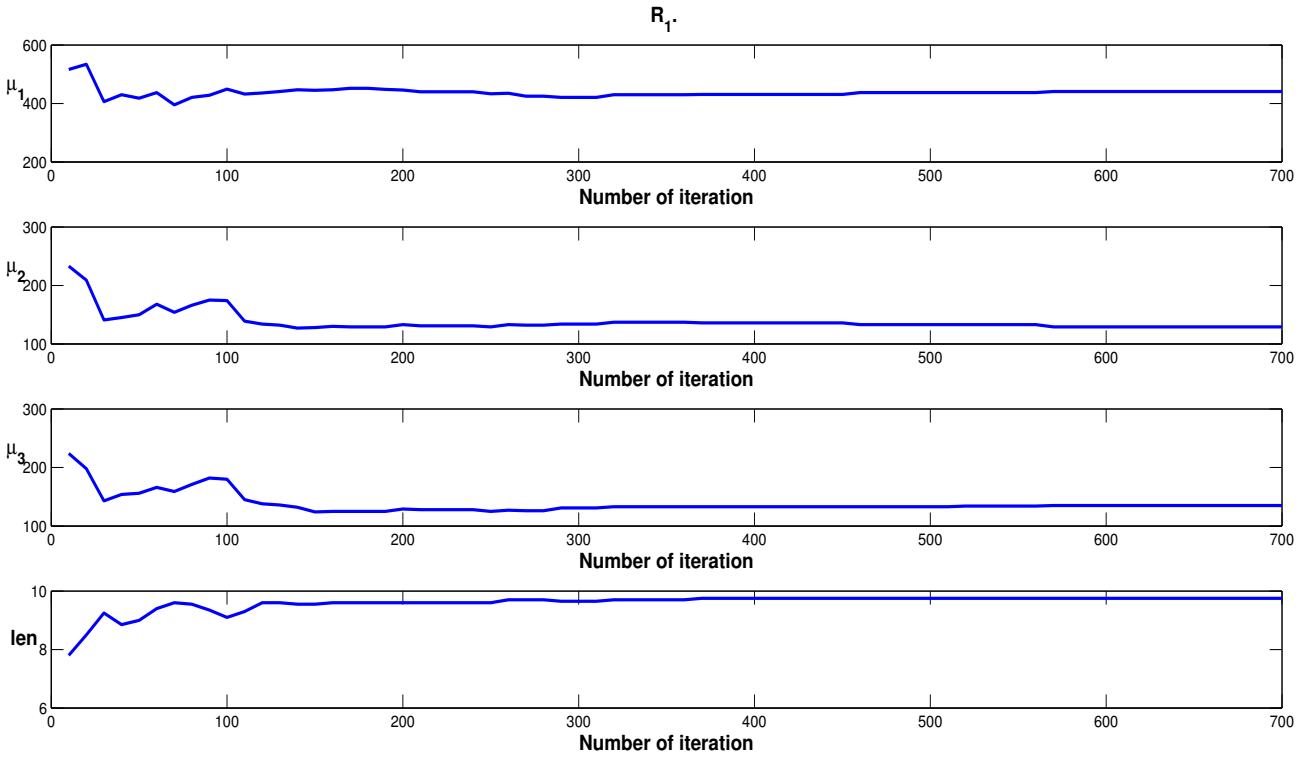

График 2. Алгоритм с $R_1 = p \cdot m \cdot I(|f| < CT)$, при $p = 0.005,\ CT = 8$.

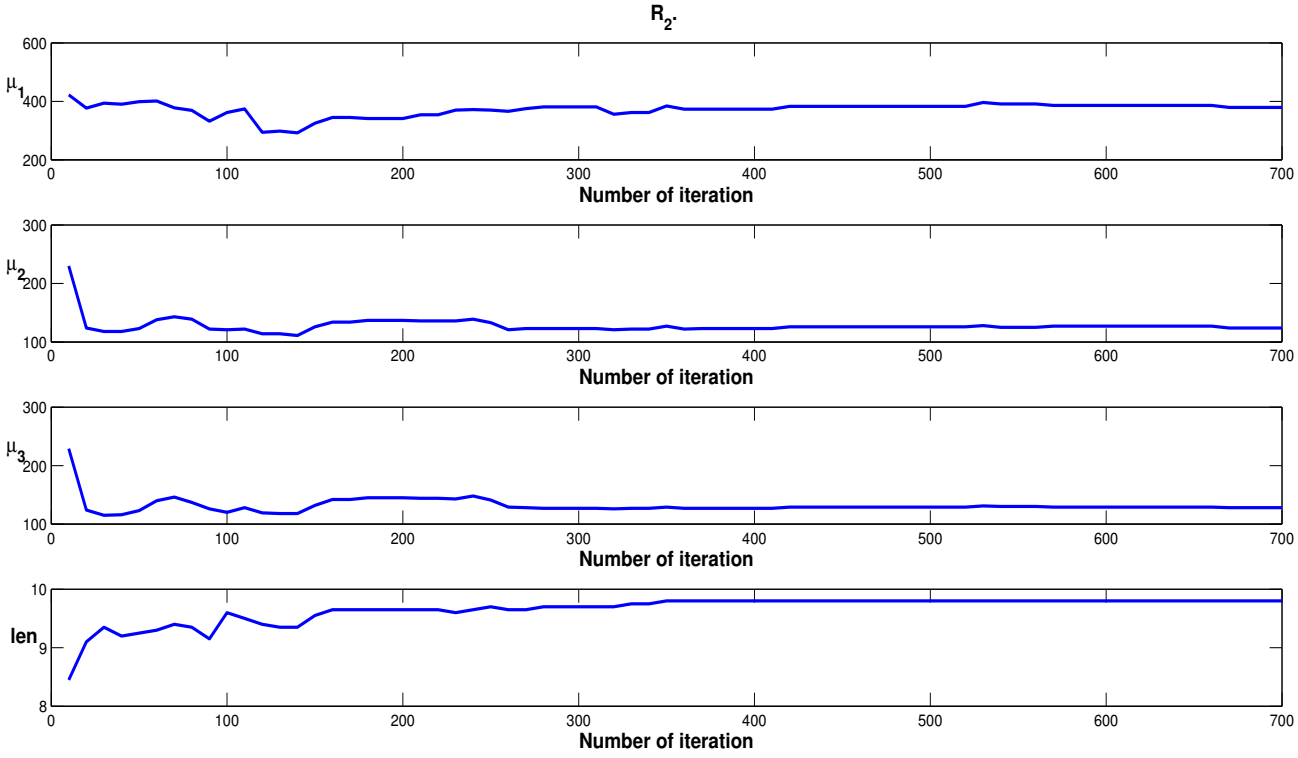

График 3. Алгоритм с $R_2 = p \cdot m \cdot I(|f| \geq CT) \cdot (|f| - CT)$, при $p = 0.005,\ CT = 8$.

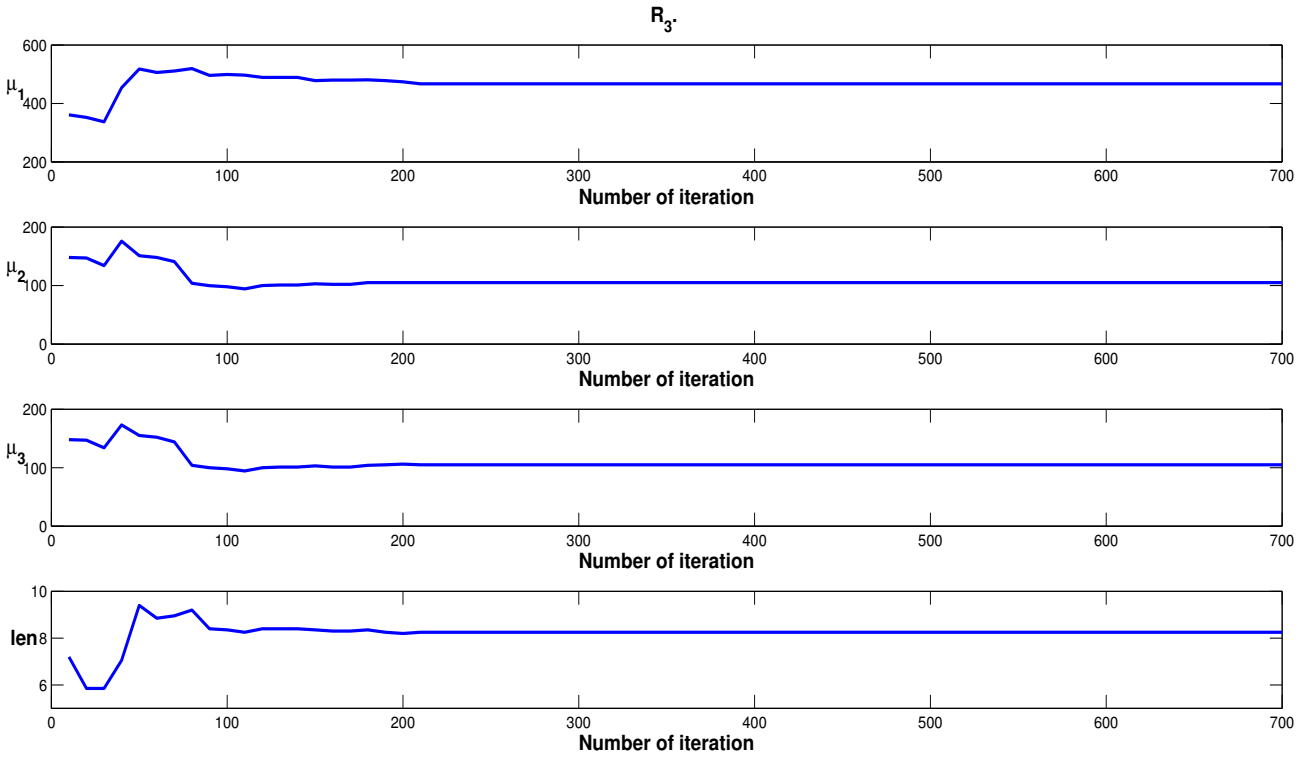

График 4. Алгоритм с $R_3 = p \cdot m \cdot |f|^* \cdot \log(|f| + 1)$, при $p = 0.005,\ CT = 8$.

Как и ожидалось, первый регуляризатор R_1 слижком жесткий — алгоритм стагнирует уже после первых итераций. Похожую ситуация наблюдается и для функции качества со вторым регуляризатором R_2 — хотя популяция не сразу попадает в локальную яму и вплоть до 300ой итерации модели обновляются, значения метрик μ существенно уменьшаются. Значит, уже на первых итерациях модели имеют существенно схожие структуры. Поэтому для дальнейшего анализа необходимо использовать третий регуляризатор.

7.2 Порождение ранжирующих моделей

Фиксируем вторую метрику μ_2 и третий регуляризатор R_3. Итоговые суперпозиции, найденные генетическим алгоритмом с заданными параметрами, будем сравнивать с лучшими моделями, отобранными в работе [7]. Они названы лучшими, поскольку в множестве структурно простых функций (структурной сложности не более 8) они в среднем лучше остальных. Кроме того, в среднем на рас-

сматриваемом множестве коллекций эти функции лучше традиционно используемых $BM25, LGD, LM_{DIR}$. Для анализа мы взяли из [7] 6 лучших:

1. $f_1 = e^{\sqrt{\ln(x/y)}}$,
2. $f_2 = \sqrt{\frac{\ln(x)}{\sqrt{y}}}$,
3. $f_3 = \sqrt[4]{\frac{x}{y}}$,
4. $f_4 = \sqrt{y + \sqrt{\frac{x}{y}}}$,
5. $f_5 = \sqrt[4]{\frac{x}{y}} \cdot e^{-y/2}$,
6. $f_6 = \sqrt{\sqrt{x} + \sqrt{\frac{x}{y}}}$.

Используем выборки Trec5-8, на которых вычисляем значение функционала MAP для итоговых моделей. Сами суперпозиции будем обучать по выборке Trec7. Остальные выборки имеют роль тестовых. После 1000 итераций генетического алгоритма получено следующее семество функций (везде далее $\ln(x)$ для удобства означает функцию $\ln(x+1)$, а $g(x) = \ln\ln(x)$):

1. $h_1 = g(\frac{g(x)}{\sqrt{\ln(x)+x}}) - \ln(y)$,
2. $h_2 = g(\frac{g(x)}{\sqrt{\frac{1}{2}\ln(x)+x}}) - \ln(y)$,
3. $h_3 = g(\ln\big(\frac{g(x)}{\sqrt{\frac{1}{2}\ln(x)+x}}\big) - \ln(y))$,
4. $h_4 = g(\frac{g(x)}{\sqrt{g(\sqrt{x})+x}}) - \ln(y)$,
5. $h_5 = g(\frac{g(x)}{\sqrt{\ln(x)+\ln(y)}}) - \ln(y)$,
6. $h_6 = g(\frac{g(\ln(x)}{\sqrt{\ln(x)+x}}) - \ln(y)$.

Приведенная ниже таблица содержит значения функционала $m(f, C, Q)$ для итоговых суперпозиций $\{h_j\}$ и моделей $\{f_i\}$ из [7].

Сравнение новых функций с имеющимися				
Функция	Trec5	Trec6	Trec7	Trec8
f_1	8.785	13.715	10.038	13.902
f_2	8.518	12.996	9.216	13.074
f_3	8.908	13.615	9.905	13.708
f_4	8.908	13.615	9.905	13.708
f_5	8.908	13.615	9.908	13.709
f_6	8.872	13.613	9.890	13.695
h_1	8.965	13.693	10.600	14.403
h_2	9.472	13.723	10.650	14.402
h_3	9.558	13.786	10.631	14.376
h_4	9.226	13.713	10.5	14.374
h_5	8.862	13.388	10.439	14.359
h_6	8.104	13.483	10.421	14.355

Функции $h_2,\ h_3,\ h_4$ равномерно лучше всех эталонных функций из [7] на всех тестовых коллекциях. Это заведомо удовлетворяет целям, поставленным перед данной работой. При этом они не переусложнены и имеют компактное описание.

8 Заключение

Описан генетический алгоритм порождения ранжирующих функций, как суперпозиций над некоторой грамматикой порождающих элементов. Проблемы, возникающие при использовании генетического алгоритма, решаются введением метрики и регуляризатора. В работе проводится анализ нескольких метрик и видов регуляризаторов, выбираются лучшие. Модифицированный алгоритм позволяет порождать ранжирующие суперпозиции компактного вида, которые при этом имеют высокую точность на тестовых данных.

Список литературы

[1] *Salton G., McGill M. J.* Introduction to Modern Information Retrieval.— New York, NY, USA: McGraw-Hill, Inc., 1986.

[2] *Porter M. F.* Readings in information retrieval / Ed. by K. Sparck Jones, P. Willett.— San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997.— Pp. 313–316.

[3] *Shamir R., Tsur D.* Faster subtree isomorphism. // *J. Algorithms.*— 1999.— Vol. 33.— Pp. 267–280.

[4] *Metzler D., Croft W. B.* A markov random field model for term dependencies // Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.— SIGIR '05.— New York, NY, USA: ACM, 2005.— Pp. 472–479.

[5] *Amati G., Van Rijsbergen C. J.* Probabilistic models of information retrieval based on measuring the divergence from randomness // *ACM Trans. Inf. Syst.*— 2002.— .— Vol. 20, no. 4.— Pp. 357–389.

[6] *Clinchant S., Gaussier E.* Information-based models for ad hoc ir // Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval.— SIGIR '10.— New York, NY, USA: ACM, 2010.— Pp. 234–241.

[7] Exploring the space of ir functions / P. Goswami, S. Moura, E. Gaussier et al.— 2014.— Pp. 372–384.

[8] *Gordon M.* Probabilistic and genetic algorithms in document retrieval // *Commun. ACM.*— 1988.— .— Vol. 31, no. 10.— Pp. 1208–1218.

[9] Learning to rank by optimizing ndcg measure / H. Valizadegan, R. Jin, R. Zhang, J. Mao // Advances in Neural Information Processing Systems 22 / Ed. by Y. Bengio, D. Schuurmans, J. Lafferty et al.— Curran Associates, Inc., 2009.— Pp. 1883–1891.

[10] *Gordon M. D.* User-based document clustering by redescribing subject descriptions with a genetic algorithm // *JASIS.*— 1991.— Vol. 42, no. 5.— Pp. 311–322.

[11] *Raghavan V., Agarwal B.* Optimal determination of user-oriented clusters: An application for the reproductive plan // Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application. — Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987. — Pp. 241–246.

[12] *Yang J., Korfhage R., Rasmussen E. M.* Query improvement in information retrieval using genetic algorithms - A report on the experiments of the TREC project // TREC. — 1992. — Pp. 31–58.

[13] The use of genetic programming to build queries for information retrieval. / F. E. Petry, B. P. Buckles, T. Sadasivan, D. H. Kraft // International Conference on Evolutionary Computation. — 1994. — Pp. 468–473.

[14] *Chen H.* Machine learning for information retrieval: Neural networks, symbolic learning, and genetic algorithms // *Journal of the American Society of Information Science.* — 1995. — Vol. 46, no. 3. — Pp. 194–216. .

[15] *Billhardt H., Borrajo D., Maojo V.* Using genetic algorithms to find suboptimal retrieval expert combinations. // SAC. — ACM, 2002. — Pp. 657–662.

[16] *Pathak P., Gordon M., Fan W.* Effective information retrieval using genetic algorithms based matching function adaptation // in Proceedings of the 33rd Hawaii International Conference on System Science (HICSS. — 2000.

[17] *Fan W., Gordon M. D., Pathak P.* A generic ranking function discovery framework by genetic programming for information retrieval // *Inf. Process. Manage.* — 2004. — . — Vol. 40, no. 4. — Pp. 587–602.

[18] *Fan W., Gordon M. D., Pathak P.* Personalization of search engine services for effective retrieval and knowledge management // Proceedings of the Twenty First International Conference on Information Systems. — ICIS '00. — Atlanta, GA, USA: Association for Information Systems, 2000. — Pp. 20–34.

[19] *Макаров.* Метрические свойства функций расстояний между молекулярными графами // *Журнал структурной химии.* — 2007. — Vol. 2. — Pp. 223–229.

[20] *Cao B., Li Y., Yin J.* Natural sciences publishing cor. measuring similarity between graphs based on the levenshtein distance. — 2013.

[21] *Zhang K., Shasha D.* D.: Simple fast algorithms for the editing distance between trees and related problems // *SIAM J. Comput.* — 1989.