Сравнительный анализ методов определения нечетких дубликатов для Web-документов

© Зеленков Ю.Г, Сегалович И.В

Яндекс
{iseg, yuryz }@yandex-team.ru

Аннотация

В работе дается сравнительное экспериментальное исследование наиболее популярных современных методов обнаружения нечетких дубликатов для текстовых документов. Приводится количественная оценка показателей *полноты*, *точности* и *F-меры*. Тестовый набор, использованный в экспериментах, представляет собой веб-коллекцию *РОМИП*. Предложены *два новых алгоритма*, имеющих высокие показатели качества.

1 Введение

Проблема обнаружения нечетких дубликатов является одной из наиболее важных и трудных задач анализа веб-данных и поиска информации в интернете. Актуальность этой проблемы определяется разнообразием приложений, в которых необходимо учитывать «похожесть», например, текстовых документов — это и улучшение качества индекса и архивов поисковых систем за счет удаления избыточной информации, и объединение новостных сообщений в сюжеты на основе сходства этих сообщений по содержанию, и фильтрация спама (как почтового, так и поискового), и установление нарушений авторских прав при незаконном копировании информации (проблема плагиата или копирайта), и ряд других.

Основным препятствием для успешного решения данной задачи является гигантский объем данных, хранимых в базах современных поисковых машин. Такой объем делает практически невозможным (в разумное время) ее «прямое» решение путем попарного сравнения текстов документов. Поэтому в последнее время большое внимание уделяется разработке методов снижения вычислительной сложности создаваемых алгоритмов за счет выбора различных эвристик (например, хеширования определенного фиксированного набора «значимых» слов или предложений документа, сэмплирования набора подстрок текста, использование дактилограмм и др.).

При применении приближенных подходов наблюдается уменьшение (иногда весьма значительное) показателя *полноты* обнаружения дублей.

Важным фактором, влияющим на *точность* и полноту определения дубликатов в задачах веб-поиска, является выделение содержательной части веб-страниц с помощью надежного распознавания элементов оформления документов и их последующего удаления. В данной работе эти вопросы не рассматриваются.

И, наконец, еще одним ключевым требованием, предъявляемым к качеству алгоритмов детектирования нечетких дубликатов, является их устойчивость к «небольшим» изменениям исходных документов и возможность уверенно обрабатывать короткие документы..

2 Обзор ключевых работ по исследуемой тематике

Одними из первых исследований в области нахождения нечетких дубликатов являются работы *U. Manber* [2] и *N. Heintze* [3]. В этих работах для построения выборки используются последовательности соседних букв. Дактилограмма файла ([2]) или документа ([3]) включает все текстовые подстроки фиксированной длины. Численное значение дактилограмм вычисляется с помощью алгоритма случайных полиномов *Карпа-Рабина* [4]. В качестве меры сходства двух документов используется отношение числа общих подстрок к размеру файла или документа. Предлагается ряд методов, направленных на снижение вычислительной сложности алгоритма.

U. Manber использовал этот подход для нахождения похожих файлов (утилита *sif*), а *N. Heintze* — для обнаружения нечетких дубликатов документов (система *Koala*).

В 1997 году *A. Broder et al*. [1, 5, 6] предложили новый, «синтаксический» метод оценки сходства между документами, основанный на представлении документа в виде множества всевозможных последовательностей фиксированной длины *k*,

Труды 9ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» — RCDL'2007, Переславль-Залесский, Россия, 2007.

состоящих из соседних слов. Такие последовательности были названы «шинглами». Два документа считались похожими, если их множества шинглов существенно пересекались. Поскольку число шинглов примерно равно длине документа в словах, т.е. является достаточно большим, авторами были предложены два метода сэмплирования для получения репрезентативных подмножеств.

Первый метод оставлял только те шинглы, чьи «дактилограммы», вычисляемые по алгоритму *Карпа-Рабина* [4], делились без остатка на некоторое число *m*. Основной недостаток — зависимость выборки от длины документа и поэтому документы небольшого размера (в словах) представлялись или очень короткими выборками, или вообще не имели таковых. Второй метод отбирал только фиксированное число *s* шинглов с наименьшими значениями дактилограмм или оставлял все шинглы, если их общее число не превышало *s*.

Дальнейшим развитием концепций *A. Broder et al.* являются исследования *D. Fetterly et al.* [9], *A. Broder et al.* [16]

Для каждой цепочки вычисляются 84 дактилограммы по алгоритму *Карпа-Рабина* [4] с помощью взаимно-однозначных и независимых функций, использующих случайные наборы («min-wise independent») простых полиномов. В результате каждый документ представлялся 84 шинглами, минимизирующими значение соответствующей функции.

Затем 84 шингла разбиваются на 6 групп по 14 (независимых) шинглов в каждой. Эти группы называются «супершинглами». Если два документа имеют сходство, например, $p \sim 0.95$ (95%), то 2 соответствующих супершингла в них совпадают с вероятностью $p^{14} \sim 0.95^{14} \sim 0.49$ (49%).

Поскольку каждый документ представляется 6 супершинглами, то вероятность того, что у двух документов совпадут не менее 2-х супершинглов, равна: $1 - (1-0.49)^6 - 6 \cdot 0.49 \cdot (1-0.49)^5 \sim 0.90$ (90%). Для сравнения: если два документа имеют сходство только $p \sim 0.80$ (80%), то вероятность совпадения не менее двух супершинглов составляет всего 0.026 (2.6%).

Таким образом, для эффективной проверки совпадения не менее 2-х супершинглов (и, следовательно, подтверждения гипотезы о сходстве содержания) каждый документ представляется всевозможными попарными сочетаниями из 6 супершинглов, которые называются «мегашинглами». Число таких мегашинглов равно 15 (число сочетаний из 6 по 2). Два документа сходны по содержанию, если у них совпадает хотя бы один мегашингл.

Ключевое преимущество данного алгоритма по сравнению с исследованиями *A. Broder et al*. состоит в том, что, во-первых, любой документ (в том числе и очень маленький) всегда представляется вектором фиксированной длины, и, во-вторых, сходство определяется простым сравнением координат вектора и не требует (как у *A. Broder*) выполнения теоретико-множественных операций.

Другой сигнатурный подход, основанный уже не на синтаксических, а на лексических принципах, был предложен *A. Chowdhury et al.* в 2002 и усовершенствован в 2004 гг. [10, 11, 12].

Основная идея такого подхода состоит в вычислении дактилограммы *I-Match* для представления содержания документов. С этой целью сначала для исходной коллекции документов строится словарь *L*, который включает слова со *средними* значениями *IDF*, поскольку такие слова обеспечивают, как правило, более точные результаты при обнаружении нечетких дубликатов. Слова с большими и маленькими значениями *IDF* отбрасываются.

Затем для каждого документа формируется множество *U* различных слов, входящих в него, и определяется пересечение *U* и словаря *L*. Если размер этого пересечения больше некоторого минимального порога (определяемого экспериментально), то список слов, входящих в пересечение упорядочивается, и для него вычисляется *I-Match* сигнатура (хеш-функция *SHA1*).

Два документа считаются похожими, если у них совпадают *I-Match* сигнатуры (имеет место коллизия хеш-кодов). Алгоритм имеет высокую вычислительную эффективность, превосходящую показатели алгоритма *A. Broder*. Другим преимуществом алгоритма (также, по сравнению с *A. Broder*) является его высокая эффективность при сравнении небольших по размеру документов. Основной недостаток — неустойчивость к небольшим изменениям содержания документа.

Для преодоления указанного недостатка исходный алгоритм был модифицирован авторами, и в него была введена возможность многократного случайного перемешивания основного словаря. Суть новых усовершенствований состоит в следующем.

Дополнительно к основному словарю *L* создаются *K* различных словарей L_1-L_K, получаемых путем случайного удаления из исходного словаря некоторого небольшой фиксированной части *p* слов, составляющей порядка 30%-35% от исходного объема L.

Для каждого документа вместо одной вычисляется *(K+1) I-Match* сигнатура по алгоритму, описанному выше, т.е. документ представляется в виде вектора размерности *(K+1)*, и два документа считаются дубликатами, если у них совпадает хотя бы одна из координат.

Если документ подвергается небольшим изменениям (порядка *n* слов), то вероятность того, что по крайней мере одна из *K* дополнительных сигнатур останется неизменной, будет равна:

$$1 - (1-p^n)^K \quad (*)$$

Действительно, вероятность того, что изменения не затронут какого-либо одного словаря, равна p^n — вероятности, что все изменения попадут в

удаленную часть исходного словаря. Тогда $(1-p^n)$ — вероятность, что сигнатура изменится, а $(1-p^n)^K$ — вероятность, что *все* сигнатуры изменятся (поскольку дополнительные словари формируются независимо), и, следовательно, *(*)* — и есть искомая вероятность.

Рекомендуемыми значениями параметров, хорошо проявившими себя на практике, являются $p = 0.33$ и $K = 10$.

Алгоритм показал высокие результаты при использовании в различных приложениях веб-поиска и фильтрации спама.

Схожий подход описан в патенте *US Patent 6,658,423 W. Pugh* [8] из *Google*. Автор предлагает полный словарь документа разбить на фиксированное число списков слов с помощью какой-либо функции хеширования (например, с помощью остатка от деления хеш-кода на число списков). Затем для каждого списка вычисляется дактилограмма и два документа считаются подобными, если они имеют хотя бы одну общую дактилограмму. Автор приводит оценки устойчивости алгоритма к небольшим изменениям содержания исходного документа. В работе отсутствуют какие-либо сведения об эффективности практического применения по соображениям конфиденциальности.

Еще одним сигнатурным подходом, также основанным на лексических принципах (т.е. использовании словаря), является метод «опорных» слов, предложенный *С. Ильинским* и др. [13]. Суть алгоритма заключается в следующем.

Сначала из индекса по некоторому правилу (см. ниже) выбирается множество из N слов (N — определяется экспериментально), называемых «опорными». Затем каждый документ представляется N-мерным двоичным вектором, где i-я координата равна 1, если i-е «опорное» слово имеет в документе относительную частоту выше определенного порога (устанавливаемого отдельно для каждого «опорного» слова), и равна 0 в противном случае. Этот двоичный вектор называется сигнатурой документа. Два документа «похожи», если у них совпадают сигнатуры.

Общие соображения для построения множества «опорных» слов таковы:

1. Множество слов должно охватывать максимально возможное число документов
2. «Качество» слова (см. ниже) должно быть наивысшим
3. Число слов в наборе должно быть минимальным

Опишем алгоритм построения множества и выбора пороговых частот.

Пусть "частота" это нормированная внутридокументная частота слова в документа *TF*, лежащая в диапазоне 0..1, где 1 частота самого частого слова в документе.

Для каждого слова (однократно) строится распределение документов по такой внутридокументной "частоте".

Проводим несколько итераций, каждая из которых состоит из двух фаз (1) и (2). В (1) максимизируется покрытие документов в индексе при фиксированной (ограниченной снизу) точности; в (2) максимизируется точность при фиксированном покрытии.

Определим "точность" слова следующим образом: точность тем выше, чем меньше встречаемость слова в дельте-окрестности данного значения относительной частоты (то есть чем меньше документов с *TF* равным $TF_{threshold} \pm \Delta$). Частоту с наилучшей точностью мы называем пороговой и запоминаем для дальнейшего использования в алгоритме (см выше).

После каждой итерации отбрасываем самые "плохие" слова. После последней итерации оставляем достаточно слов для хорошего покрытия.

Этот метод, позволяет, начав с выборки в сотни тысяч слов, оставить набор в 3-5 тысяч, расчет сигнатур по которому с применением полнотекстового индекса осуществляется на миллиардном индексе несколько минут на поисковом кластере *Яндекса* [17].

3 Идея исследования

Основная идея исследования состояла в том, чтобы, используя различные наборы текстовых документов (общедоступные коллекции, почтовые сообщения, отдельные веб-страницы и пр.), оценить качество наиболее известных, разнообразных и эффективных с вычислительной точки зрения алгоритмов определения нечетких дубликатов. Другой целью являлась разработка новых методов в этой области, учитывающих недостатки и ограничения существующих подходов.

В качестве основных показателей качества работы алгоритмов были выбраны *полнота*, *точность* и *F-мера*. Предполагалось сравнить алгоритмы по этим параметрам, а также определить их взаимную корреляцию и совместное покрытие разными сочетаниями алгоритмов исходного множества пар нечетких дубликатов.

Результаты работы каждого алгоритма (пары нечетких дубликатов) могут быть агрегированы так, что для любой пары исходных документов *(docX, docY)* указывался список алгоритмов *(A1, A2, ...)*, с точки зрения которых эти документы являлись нечеткими дубликатами. Все пары-кандидаты, полученные хотя бы одним алгоритмом, можно проверить прямым текстовым сравнением, что позволит оценить точность каждого алгоритма. Для сравнительной оценки полноты алгоритмов можно использовать идею общего пула, как максимального множества найденных и верифицированных дублей.

4 Описание методов и алгоритмов

Для практического исследования были выбраны методы и алгоритмы определения нечетких дубликатов, перечисленные в *Табл. 1*.

Таблица 1

Методы и алгоритмы определения нечетких дубликатов

Имя	Название
A0	MD5
A1	TF
A2	TF*IDF
A3	TF*RIDF
A4	Long Sent
A5	Heavy Sent
A6	Megashingles
A7	Lex Rand
A8	Log Shingles
A9	Descr Words
A10	Opt Freq

Ниже приводятся краткие описания исследуемых алгоритмов, в которых используются следующие обозначения:

N — число документов в коллекции;

tf — частота слова в документе;

tf_max — максимальная частота слова в документе;

df — число документов в коллекции, содержащих данное слово;

cf — суммарная частота слова в коллекции.

dl — длина документа;

dl_avg — средняя длина документа в коллекции.

4.1 A0 — MD5

Данный алгоритм на самом деле определяет «точные» дубликаты и включен в список с целью сравнения итоговых статистик для полных и нечетких дубликатов. В качестве сигнатуры документа используется хеш-функция *MD5* [19], вычисленная для всего документа.

4.2 A1 — TF

Идея этого и ряда следующих алгоритмов (A1-A5) достаточно очевидна, в качестве аналога приведем работу [18], в которой анализируется семейство подобных алгоритмов для задач нахождения перемещенных документов в вебе. В соответствии с этой работой назовем «локальными» алгоритмы, не использующие общую статистику коллекции (A1, A4) и «глобальными» те, которые опираются на частотные характеристики по всей коллекции (A2, A3, A5).

Строится частотный словарь документа, упорядоченный по убыванию частот. Затем выбираются и сцепляются в алфавитном порядке в строку 6 слов с наибольшими значениями *tf*. В качестве сигнатуры документа вычисляется контрольная сумма *CRC32* полученной строки.

*4.3 A2 — TF*IDF*

По всей коллекции строится словарь, ставящий каждому слову в соответствие число документов, в которых оно встречается хотя бы один раз (*df*) и определяется средняя длина документа (*dl_avg*). Затем строится частотный словарь документа и для каждого слова вычисляется его «вес» *wt* по формуле *Okapi BM25* с параметрами $k = 2$ и $b = 0.75$ [14]:

$wt = TF * IDF$, где

$$TF = tf / [\ 2 * (0.25 + 0.75\ (dl / dl\_avg\ )\ ) + tf\ ]$$
$$IDF = log\ [\ (N - df + 0.5)/(df + 0.5)\ ]$$

Затем выбираются и сцепляются в алфавитном порядке в строку 6 слов с наибольшими значениями wt. В качестве сигнатуры документа вычисляется контрольная сумма *CRC32* полученной строки.

*4.4 A3 — TF*RIDF*

Краткое теоретическое введение. Основная идея *RIDF* (*Residual IDF*) [15] состоит в сравнении двух способов подсчета количества информации (в смысле определения *К. Шеннона*), содержащейся в сообщении о том, что данное слово входит в некоторый документ (по меньшей мере один раз). Первый способ, статистический, это обычный $IDF = -log(df / N)$. Второй способ, теоретический, основан на модели распределения *Пуассона*, предполагающей, что слова в коллекции документов распределяются случайным и независимым образом, равномерно рассеиваясь с некоторой средней плотностью. В этом случае соответствующее количество информации равно $P\_IDF = -log(1 - exp(-cf / N))$. Тогда,

$$RIDF = IDF - P\_IDF = -log(df / N) + log(1 — exp(-cf / N))$$

показывает прирост информации, содержащейся в реальном распределении слова в коллекции по сравнению с равномерно случайным пуассоновским, т.е. *ценность* слова. Другими словами «хорошие» (значимые, осмысленные) слова должны быть распределены неравномерно среди относительно небольшого числа документов (обладать «редкостью»), а «плохие» (бессодержательные) будут равномерно рассеяны по всей коллекции (встречаться, что называется, «на каждом шагу»).

Практическая реализация. По всей коллекции строится словарь, ставящий каждому слову в соответствие число документов, в которых оно встречается хотя бы один раз (*df*) и определяется суммарная частота каждого слова в коллекции (*cf*). Затем строится частотный словарь документа и для каждого слова вычисляется его «вес» *wt* по формуле:

$wt = TF * RIDF$, где

*TF = 0.5 + 0.5 * tf / tf_max*
RIDF = -log(df / N) + log(1 - exp(-cf / N))

Затем выбираются и сцепляются в алфавитном порядке в строку 6 слов с наибольшими значениями *wt*. В качестве сигнатуры документа вычисляется контрольная сумма *CRC32* полученной строки.

4.5 A4 — Long Sent

Документ разбивается на предложения, которые упорядочиваются по убыванию длины, выраженной количеством слов, а при равенстве длин — в алфавитном порядке. Затем выбираются и сцепляются в строку в алфавитном порядке 2 самых длинных предложения. В качестве сигнатуры документа вычисляется контрольная сумма *CRC32* полученной строки.

4.6 A5 — Heavy Sent

Документ разбивается на предложения. Для каждого предложения подсчитывается его «вес» *ws* равный сумме весов *wt* всех входящих в него слов. Веса *wt* подсчитываются по формулам алгоритма A2. Затем выбираются и сцепляются в строку в алфавитном порядке 2 самых «тяжелых» предложения. В качестве сигнатуры документа вычисляется контрольная сумма *CRC32* полученной строки.

4.7 A6 — Megashingles

Алгоритм реализован в соответствии с принципами, изложенными в [9].

Сначала вычисляется множество всех 5-словных шинглов. Так как сходство 0.95 представлялось нам слишком сильным, мы задавались сходством 0.75. Прямая схема Fetterly (с 84 функциями и 14 шинглами в 1-м мегашингле) давала полноту практически 0.00. Поэтому мы остановились на 36 функциях, построенных с помощью случайных полиномов и реализующих дактилограммы *Карпа-Рабина* [4]. Выбираются 36 шинглов, минимизирующих значения соответствующих функций.

Далее 36 шингла разбиваются на 6 групп по 6 и определяются 6 «супершинглов», из которых составляются всевозможные парные сочетания, называемые «мегашинглами». Число таких сочетаний из 6 по 2 равно 15. Все эти 15 мегашинглов составляют сигнатуру документа, которые давали полноту 33% для порогового отличия 0.75, 42% для 0.80 и 50% для 0.85. ($p^6 \sim 0.80^6 \sim 0.26$ (26%); $1-(1-0.26)^6-6\cdot0.26\cdot(1-0.26)^5 \sim 0.42$(42%)).

4.8 A7 — Lex Rand

Алгоритм реализован в соответствии с принципами, изложенными в [12]. Сначала по всей коллекции строится словарь, аналогичный использованному в алгоритме *A2* из которого удаляются слова с наибольшими и наименьшими значениями *IDF*. Затем основе этого словаря генерируются 10 дополнительных словарей, содержащих примерно на 30% меньше слов, чем исходный. Слова удаляются случайным образом.

Для каждого документа строится 11 *I-Match* сигнатур (см. выше обзорную часть статьи). Дубликатами считаются документы хотя бы с одной совпавшей сигнатурой. Оказывается, что такой подход весьма существенно по сравнению с *A2* (более чем в 2 раза) повышает полноту обнаружения дубликатов при снижении относительной точности всего на 14%.

4.9 A8 — Log Shingles

Метод основан на «супершинглировании» *логарифмической* [7] выборки из исходного полного множества шинглов, такой, которая оставляет шинглы, делящиеся без остатка на степени небольшого числа.

Сначала вычисляется множество всех 5-словных шинглов (слова в конце документа «заворачиваются» на начало). Затем из этого множества отбираются шинглы, делящиеся на степени числа 2. Они и составляют точную сигнатуру документа.

4.10 A9 — Descr Words

Алгоритм реализован в соответствии с принципами, изложенными в [13] и техникой, описанной выше. Сначала по всей коллекции строится словарь «опорных» слов и вычисляются двоичные векторные сигнатуры документов, которые затем кластеризуются в группы нечетких дубликатов на основе признака совпадения сигнатур.

4.11 A10 — Opt Freq

Краткое пояснение. Алгоритм реализует метод «оптимальной поисковой частоты», предложенный *М. Масловым (Яндекс* [17]*)*, и использующийся для поиска похожих документов в широком спектре приложений, от веб-поика до кластеризации новостей. Суть его заключается в следующем. Вместо классической метрики *TF*IDF* (см. алгоритм *A2*) предлагается ее модифицированный вариант. Вводится эвристическое понятие «оптимальной частоты» для слова равное – *ln(10/1000000) = 11.5*, т.е. «оптимальным» считается вхождение слова в 10 документов из 1000000. Если реальное значение *IDF* меньше «оптимального», то оно немного (по закону параболы) повышается до *IDF_opt = SQRT(IDF / 11.5)*, а если больше, то существенно (как гипербола) снижается до *IDF_opt = 11.5 / IDF*.

Практическая реализация. По всей коллекции строится словарь, ставящий каждому слову в соответствие число документов, в которых оно встречается хотя бы один раз (*df*). Далее строится частотный словарь документа и для каждого слова вычисляется его «вес» *wt* по формуле:

*wt = TF * IDF_opt*, где

*TF = 0.5 + 0.5 * tf / tf_max*
IDF = -log(df / N)

IDF_opt = SQRT(IDF / 11.5), если IDF меньше чем 11.5, иначе
IDF_opt = 11.5 / IDF

Затем выбираются и сцепляются в алфавитном порядке в строку 6 слов с наибольшими значениями wt. В качестве сигнатуры документа вычисляется контрольная сумма *CRC32* полученной строки.

5 Результаты проведенных экспериментов

Для проведения экспериментальных исследований алгоритмов *A0-A10* была использована следующая коллекция документов:

- набор полных веб-сайтов *РОМИП*. Объем ~ *500 000* документов;

В качестве эталона 100%-й полноты для данной коллекции использовалось объединение дубликатов, найденных всеми алгоритмами. В качестве эталона 100%-й точности использовались результаты, полученные с помощью функции *Perl String::Similarity* с порогом сходства 80%.

Перед обработкой выполнялся необходимый *препроцессинг* — очистка от html-тегов, удаление стоп-слов (служебной лексики и коротких слов длиной 3 и менее букв), символов новой строки, лишних пробелов, специальных знаков и т.п.

В нижеследующих таблицах приведены результаты обработки алгоритмами *A1-A10* веб-коллекции РОМИП.

Коллекция веб-сайтов РОМИП

Объем коллекции ~ *500 000* документов

Количество пар дублей, обнаруженных всеми алгоритмами – *17 471 200,* только 68% из которых верифицировалось по результатам текстуального сравнения.

Таблица 2

Эффективность алгоритмов определения нечетких дубликатов (результаты упорядочены по убыванию F-меры)

Имя	Полнота	Точность	F-мера
A4	0.84	0.80	0.82
A1	0.60	0.94	0.73
A10	0.59	0.94	0.73
A3	0.59	0.95	0.73
A5	0.62	0.86	0.72
A2	0.54	0.96	0.69
A7	0.50	0.97	0.66
A9	0.44	0.77	0.56
A8	0.39	0.97	0.56
A6	0.36	0.91	0.51
A0	0.23	1.00	0.38

Таблица 3

Количество пар дублей, обнаруженных отдельными алгоритмами

Имя	Число дублей
A0	2 791 226
A1	7 540 460
A2	6 745 096
A3	7 373 208
A4	12 635 770
A5	8 544 468
A6	4 696 699
A7	6 124 207
A8	4 791 883
A9	6 927 589
A10	7 531 378

Таблица 4

Матрица совместных корреляций некоторых алгоритмов (в смысле *Dice*-коэффициента)

	A1	A2	A3	A4	A5	A6	A7	A9
A1	-	0.83	0.79	0.40	0.68	0.54	0.90	0.65
A2		-	0.75	0.38	0.76	0.60	0.88	0.71
A3			-	0.32	0.60	0.48	0.82	0.59
A4				-	0.76	0.54	0.54	0.55
A5					-	0.59	0.69	0.68
A6						-	0.56	0.53
A7							-	0.66
A9								-

Dice-коэффициент для двух множеств A и B рассчитывается по формуле:

$$Dice(A,B) = 2 \cdot A \cdot B / (A + B)$$

6 Экспериментальные алгоритмы определения нечетких дубикатов

Основная задача, которую ставили перед собой авторы при разработке новых алгоритмов обнаружения нечетких дубликатов — существенное (в 2-4 раза) увеличение показателя "полноты" по сравнению с существующими алгоритмами, при сохранении максимально возможного показателя «точности».

6.1. Метод «декомпозиции» (A11)

В этом методе мы экспериментировали с полными наборами шинглов и методами

сокращения квадратичной зависимости, которая возникает, когда общим признаком обладает большое количество документов. Идея основана на (1) том наблюдении, что размер документа в словах служит хорошим разделяющим свойством, а также (2) на нахождении общих списков документов, обладающих одинаковым признаком.

Работа алгоритма состоит в следующем:

- Для каждого документа вычисляем не повторяющиеся шинглы (можно не все, а выборочно) и сохраняем в файле в формате *<shingle, doc_id, doc_len>*.
- Для одинаковых шинглов строим цепочки в формате *<doc_id1, doc_len1> <doc_id2, doc_len2> ...*, упорядоченные *по возрастанию doc_len*.
- Разделяем цепочки на более мелкие, если у соседних длин отношение большей длины к меньшей превышает некоторый порог, определяемый минимальным уровнем сходства для дубликатов (например, для уровня сходства *0.85* можно практически без потери полноты использовать порог *1.15*).
- Удаляем дубли цепочек и цепочки, которые целиком входят в другие. В результате число цепочек сокращается *в сотни раз*, а оставшиеся цепочки в подавляющем большинстве являются достаточно короткими (*2 — 10* элементов).
- Документы внутри цепочки сравниваем попарно (например, путем использования функции *Perl Similarity* или, еще проще, с помощью каких-нибудь дополнительных числовых характеристик документов). Причем сравнение идет опять же не по всей цепочке, а только в пределах небольшой локальной окрестности, определяемой все тем же порогом отношения длин. Поэтому общее число реальных сравнений *мало*.
- Для исключения дублей проверок в разных цепочках списки уже обработанных пар сохраняем в хеше.

Основным недостатком алгоритма, снижающим его производительность, является необходимость вычисления шинглов для документов и использование функции *Similarity*. Хотя, как указывалось ранее, можно вычислять не все шинглы, а только некоторые, в соответствии с какой-либо эвристикой и заменить *Similarity* более простыми средствами. Здесь необходимы дополнительные исследования.

Экспериментальная проверка

Алгоритм показал практически 100% полноту и точность на тестовой коллекции, одновременно при вполне приемлемой производительности.

6.2. Метод «3+5» (A12)

Подбирая параметры этого метода, мы экспериментировали с признаками, требующими минимальных вычислительных ресурсов, в том числе, мы не определяли контрольные суммы всех подстрок текста. Кроме того, мы пытались опираться только на «локальные» свойства текстов, то есть не использовали глобальные параметры коллекции.

Работа алгоритма состоит в следующем.

- Для каждого документа коллекции формируется *не более трех* записей следующего вида:
  <ss1,id,len,num,ss1,ss2,ss3,<ws1..5>>
  <ss2,id,len,num,ss1,ss2,ss3,<ws1..5>>
  <ss3,id,len,num,ss1,ss2,ss3,<ws1..5>>
  где
  <ss1,ss2,ss3> — сигнатуры трех самых длинных предложений документа, упорядоченные по убыванию длины предложений (а при равенстве длин — по сигнатурам),
  <id> — идентификатор документа,
  <len> — длина документа (количество слов длиной три и более букв),
  <num> — число предложений в документе.
  <ws1..5> = *<ws1,ws2,ws3,ws4,ws5>* — сигнатуры пяти самых длинных слов документа, упорядоченные по убыванию длины слов (а при равенстве длин — по сигнатурам).
- Для *коротких* документов, состоящих из одного или двух предложений, создаются одна или две записи вида:
  <sis1,id,len,sis1,0,0,<ws1..5>>,
  или
  <sis1,id,len,sis1,sis2,0,<ws1..5>>
  <sis2,id,len,sis1,sis2,0,<ws1..5>>
  соответственно.
- Полученный файл сигнатур сортируется и для записей с совпадающим первым полем формируются цепочки вида (само первое поле отбрасывается):
  <id1,len1,num1,ss11,ss21,ss31,<ws11..51>>
  <id2,len2,num2,ss12,ss22,ss32,<ws12..52>> ...,
  упорядоченные по возрастанию поля *<len>* — длины документа (а при равенстве длин — по *<id>*). При этом отношение длин двух *соседних* элементов цепочки не должно превышать некоторого порога, определяемого заданным минимальным коэффициентом сходства документов

(например, для коэффициента *0.85,* оптимальное значение порога равно *1.15*).

- При превышении порога формирование текущей цепочки заканчивается и начинается формирование новой, несмотря на то, что первое поле может оставаться тем же самым. В результате исходный файл сигнатур преобразуется в множество сравнительно коротких цепочек, содержащих компактные (с небольшим разбросом) монотонно неубывающие последовательности длин документов.

- Файл цепочек сортируется, и из него исключаются дубли и цепочки, которые целиком входят в другие, более длинные. В результате такой нормализации получается небольшого размера файл цепочек, в котором коэффициент избыточности (вхождение элементов в разные цепочки) составляет не более 10%.

- Поскольку внутри цепочки элементы упорядочены по возрастанию длин документов, для каждого элемента существует локальная окрестность относительно небольшого размера, определяемого тем же пороговым значением отношения длин.

- Просматриваем файл цепочек и для каждого элемента цепочки находим дубликаты по следующим правилам:

    - ищем только в пределах локальной окрестности;
    - сравниваем пары, только если отношение числа предложений в них не превышает некоторого порога (1.20) и из пяти сигнатур слов <*ws1..5*> совпадает *не менее двух*, неважно в каком порядке;
    - два документа считаем дубликатами, если у них совпадают сигнатуры самых длинных предложений <*ss1*> или (для документов, содержащих более 5 предложений, а таких — подавляющее большинство) из трех сигнатур <*ss1, ss2, ss3*> совпадают *две, неважно в каком порядке*.

Экспериментальная проверка

Таблица 7

Характеристики алгоритма *A12* на коллекции *РОМИП*

Имя	Полнота	Точность	F-мера
A12	**0.96**	**0.95**	**0.95**

Как видно, на коллекции РОМИП алгоритм обнаруживает нечеткие дубликаты с показателем полноты около 96%, что в 2-3 раза превышает полноту всех других известных алгоритмов. Точность алгоритма составляет порядка 95%.

7. Выводы и обсуждение результатов

Проведенные экспериментальные исследования наиболее популярных современных алгоритмов определения нечетких дубликатов показали следующее:

- все методы, опирающие на внутри-документную частоту слов, показали приемлемую точность (*TF*IDF*, *TF*RIDF*, *Opt Freq*, *TF*), однако не смогли приблизиться к существенно более представительной по числу включаемых в нее слов лексической сигнатуре документа *Lex Rand*. В этом семействе методов лучшие показатели полноты у методов *TF*, *Opt Freq*, *TF*RIDF*, не включающих редкие слова в сигнатуру.
- Метод *Lex Rand*, хотя и является «чемпионом» по точности, обладает не лучшей полной, возможно из-за того, что требует точного совпадения слишком большого подмножества словаря документа.
- Из «синтаксических» (использующих последовательности слов) методов метод *Log_Shingle* стоит особняком, приближаясь по своему поведению к MD5 – самая высокая точность и самая низкая полнота, а метод *Megashingles* показал близкий к предсказанному результат (36% против заказанного» 0.42).
- Очень хорошую полноту (лучшую в прогоне) показал метод длинного предложения *Long Sent.* При этом точность обоих методов, основанных на выделении лучшего предложения, предсказуемо самая низкая.
- Метод Descr Words показал низкие результаты, возможно, из-за технической ошибки.

Мы посчитали, что в алгоритмах имеются неиспользованные возможности по увеличению *полноты*, которые можно интегрировать в единую схему и попытаться преодолеть недостатки каждого отдельного алгоритма в этом отношении.

С учетом сказанного, авторами были предложены новые алгоритмы (A11 и A12), которые в определенной степени реализуют эту схему и на тестовых коллекциях имеют хорошие показатели.

Литература

[1] A. Broder, S. Glassman, M. Manasse and G. Zweig. Syntactic clustering of the Web. Proc. of the 6th International World Wide Web Conference, April 1997.

[2] U. Manber. Finding Similar Files in a Large File System. Winter USENIX Technical Conference, 1994.

[3] N. Heintze. Scalable document fingerprinting. In Proc. of the 2nd USENIX Workshop on Electronic Commerce, Nov. 1996.

[4] Д. Гасфилд. Строки, деревья и последовательности в алгоритмах. СПб.: Невский диалект, 2003.

[5] A. Broder. On the resemblance and containment of documents. Compression and Complexity of Sequences (SEQUENCES'97), pages 21-29. IEEE Computer Society, 1998. http://ftp.digital.com/pub/Digital/SRC/publications/broder/positano-final-wpnums.pdf

[6] A. Broder. Algorithms for duplicate documents. http://www.cs.princeton.edu/courses/archive/spr05/cos598E/bib/Princeton.pdf

[7] И. Сегалович, Д. Тейблюм, А. Дилевский. Принципы и технические методы работы с незапрашиваемой корреспонденцией. http://company.yandex.ru/articles/spamooborona.html

[8] W. Pugh. Detecting duplicate and near — duplicate files. http://www.cs.umd.edu/~pugh/google/Duplicates.pdf

[9] D. Fetterly, M. Manasse, M. Najork. A Large-Scale Study of the Evolution of Web Pages, WWW2003, May 20-24, 2003, Budapest, Hungary.

[10] A. Chowdhury, O. Frieder, D. Grossman, M. McCabe. Collection statistics for fast duplicate document detection. ACM Transactions on Information Systems (TOIS), Vol. 20, Issue 2 (April 2002). http://ir.iit.edu/~dagr/2002collectionstatisticsfor.pdf

[11] A. Chowdhury. Duplicate Data Detection. http://ir.iit.edu/~abdur/Research/Duplicate.html

[12] A. Kolcz, A. Chowdhury, J. Alspector. Improved Robustness of Signature-Based Near-Replica Detection via Lexicon Randomization. KDD 2004. http://ir.iit.edu/~abdur/publications/470-kolcz.pdf

[13] S. Ilyinsky, M. Kuzmin, A. Melkov, I. Segalovich. An efficient method to detect duplicates of Web documents with the use of inverted index. WWW Conference 2002. http://www2002.org/CDROM/poster/187/

[14] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, M. Gatford. Okapi at trec-3. The Third Text REtrieval Conference (TREC-3), 1995.

[15] K. Church, W. Gale. Poisson mixtures. Natural Language Engineering, 1995, 1(2):163–190.

[16] A. Broder, M. Charikar et al. Min-wise independent permutations, Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998

[17] Яндекс. http://www.yandex.ru

[18] S.-T. Park, D. Pennock, C. Lee Giles, R. Krovetz, Analysis of Lexical Signatures for Finding Lost or Related Documents, SIGIR'02, August 11-15, 2002, Tampere, Finland

[19] Berson, Thomas A. (1992). "Differential Cryptanalysis Mod 2^{32} with Applications to MD5". EUROCRYPT

Comparative analysis of near-duplicate detection methods of Web documents

© Zelenkov Yuri. G., Segalovich Ilya V.
Yandex

The work covers comparative experimental investigation of most popular modern methods of near-duplicate detection for textual documents. The quantitative evaluation of indices of *completeness*, *precision* and *F-measure* is given.

The test data which is used in experiments, include the *ROMIP*.

A new algorithm, having higher quality factors than the existing approaches is proposed.