Система полнотекстового поиска по длинным запросам

© А.П. Колосов, М.Ю. Богатырев

SmartBear Software, Тульский государственный университет
Alexey.Kolosoff@gmail.com

Аннотация

Рассматривается задача полнотекстового поиска, в которой поисковым запросом является текст, состоящий из нескольких предложений. Такая задача актуальна для сервисов, посвященных ответам на вопросы (техподдержка, форумы и т.п.).
Для решения данной задачи предлагается алгоритм, позволяющий свести поиск по тексту к поиску по словосочетаниям. Приводятся результаты экспериментов, подтверждающие эффективность данного алгоритма.

1. Введение

Электронные ресурсы, посвященные ответам на вопросы (например, системы служб техподдержки или форумы) по мере своего развития накапливают все больше информации, поиск по которой способен с одной стороны помочь пользователям таких систем получить ответ немедленно, не ожидая ответа специалистов, а с другой – снизить количество дублирующихся вопросов и нагрузку людей, отвечающих на вопросы.

Однако практика показывает, что большинство пользователей предпочитает сразу прибегать к помощи других людей, не пытаясь найти ответ среди имеющейся информации. А значит, уменьшение количества дубликатов и снижение нагрузки специалистов может достигаться путем автоматического поиска по тексту вопроса до того, как этот вопрос (состоящий в общем случае из 1-10 предложений произвольного стиля на естественном языке) попадет в базу данных.

В данной статье описывается алгоритм поиска по полным текстам вопросов, решающий данную задачу. В качестве примера электронного ресурса рассматривается портал техподдержки компании SmartBear, занимающейся производством программного обеспечения.

Труды 13[й] Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2011, Воронеж, Россия, 2011.

2. Постановка задачи

2.1 Описание информационного ресурса

Структура рассматриваемого информационного ресурса достаточно типична для любой подобной компании: имеется база данных опубликованной в Интернете справочной документации (Help, FAQ и т.п.) по продуктам, а также база данных техподдержки, в которую собираются вопросы пользователей, поступающие с форумов и по электронной почте. В той же базе данных хранятся ответы на эти вопросы.

Входными данными для поиска являются вопросы пользователей, выходными – список документов, которые могут содержать нужную пользователю информацию. Отметим также, что сообщения состоят из темы (короткого описания) и полного текста с подробным (несколько предложений) описанием проблемы на русском или английском языке.

Реализованная на данный момент система позволяет производить автоматический поиск по ключевым словам из темы сообщения, но собранная статистика показывает, что такой подход недостаточно эффективен. Только 1% пользователей находит нужную информацию среди автоматически подобранных результатов. Еще 4,5% после просмотра предложенных результатов изменяют свой вопрос. Подавляющее большинство все же отправляет свои вопросы в техподдержку без изменений. Объясняются такие результаты достаточно просто: лишь очень немногие пользователи могут достаточно кратко и в нужных терминах сформулировать суть проблемы, с которой они столкнулись. Большинство просто указывает абстрактные темы вроде «Support query» или «Usage problem». А значит, для повышения эффективности поиска необходимо учитывать также и подробное описание вопроса.

2.2 Отличия от поиска в Интернете

Поставленная задача поиска имеет ряд важных отличий от классического поиска в Интернете. Во-первых, поскольку вопрос может оказаться достаточно длинным (например, состоять из 10 предложений) и содержать не влияющие на смысл слова и предложения (например, приветствия, подписи, и т.п.), необходимо извлечь из него

словосочетания, выражающие саму суть вопроса. Во-вторых, все индексируемые страницы принадлежат одной и той же компании, заинтересованной в адекватности результатов поиска. А значит, нет смысла учитывать при поиске ссылки между документами для определения авторитетности той или иной страницы и производить прочие действия, направленные на борьбу с нечестным продвижением сайтов. Также количество индексируемых документов составляет десятки тысяч, т.е. относительно невелико по сравнению с количеством страниц в Интернете, и документы посвящены одной тематике, или ряду близких тематик.

3. Алгоритмическое обеспечение поисковой системы

Алгоритм, предлагаемый для решения поставленной задачи, можно разбить на следующие этапы:

1. Обработка запроса (разбиение на слова, выделение словосочетаний).
2. Поиск (вычисление релевантности по словосочетаниям).
3. Обучение (корректировка веса найденных элементов).

Последняя операция не является обязательной, но позволяет улучшить качество поиска, о чем будет рассказано ниже. Приведем подробное описание алгоритма, решающего каждую из подзадач.

3.1 Обработка запросов

На этапе обработки запросов производится разбор текста на слова и предложения, фильтрация шумовых слов, раскрытие форм слов и выявление словосочетаний. Последнее рассмотрим подробнее.

Выявление словосочетаний производится путем обработки знаков препинания в индексируемых документах, а также путем построения *концептуальных графов* [1], соответствующих предложениям запросов. Концептуальный граф как семантическая модель текста предложения позволяет найти в нем словосочетания в виде пар концептов, связанных определенными отношениями, например, отношением «*атрибут*». В данной технологии применяется программное обеспечение для автоматического построения концептуальных графов для текстов [3].

Сначала рассмотрим обработку знаков препинания.

Поскольку знаки препинания служат для разделения текста на структурные единицы, логично обрабатывать их на этапе индексирования документа. В предлагаемом алгоритме знаки препинания, разделяющие слова, учитываются при вычислении позиции слова относительно начала текста. Отметим, что в индексах используется векторная модель текстов – каждому слову ставится в соответствие его вес, вычисленный по известной формуле tf (term frequency) [6]. Также для каждого слова хранятся порядковые номера, характеризующие расстояние относительно начала текста. Наличие знаков препинания приводит к искусственному увеличению (или уменьшению) расстояния между словами, давая базовое представление о том, какие слова могут быть связаны, а какие, напротив, не должны иметь семантических связей друг с другом. Благодаря этому информация о знаках препинания неявно сохраняется в индексах, хранящих только вес слов и их позицию.

Рассмотрим примеры:

…**issue-tracking** tools... => [N, N+0.5]

Здесь слова *issue* и *tracking* разделены дефисом, а значит, вероятно, связаны между собой. Поэтому расстояние между ними берется равным 0.5.

...the **issue**, but **tracking** changes... => [N, N+3]

Здесь слова *issue* и *tracking* разделены запятой (дающей увеличение позиции на 2) и шумовым словом *but* (увеличивающим ее на 1). Итоговое расстояние между словами показывает, что в данном случае слова слабее связаны между собой.

...**Object.Method()**... **=> [**N, N+0.5**]**

Здесь слова разделены точкой без пробела, что является типичным для программного кода, часто встречающегося в технической документации. В этом случае слова *object* и *method* скорее всего связаны между собой, поэтому расстояние берется равным 0.5.

…some **object**. **Method** A shows… => [N, N+15]

В данном случае слова *object* и *method* находятся в разных предложениях, а значит вряд ли связаны семантически. Поэтому позиция слова *method* искусственно увеличивается на 15.

Приведенные в примерах значения констант, характеризующих увеличение позиции, подобраны экспериментально и использовались при индексировании тестового множества, о котором пойдет речь позднее.

Итак, знаки препинания и расположение слов и позволяют судить о том, какие слова в индексируемых документах могут быть связаны (а значит, присутствует ли в документе искомое словосочетаний), но этого, очевидно, недостаточно для выявления из запроса словосочетаний, по которым следует проводить поиск.

3.2. Применение концептуальных графов

Для более детального анализа предлагается использовать концептуальные графы, учитывающие информацию о формах слов, частях речи и т.п. Здесь используется одно важное наблюдение: – для описания технической сути проблемы пользователи стараются использовать грамматически корректные сочетания слов и термины, которые они видят в используемых продуктах – например, в текстах ошибок, в интерфейсе и т.п. Незначащие предложения (например, приветствия, подписи, и т.п.) часто не являются грамматически корректными, и их можно отфильтровать простым

способом, построив для них концептуальные графы. Такие графы окажутся некорректными, т. е. в них окажутся концепты, не связанные никакими отношениями. Обнаружение таких концептов выполняется также при помощи системы построения концептуальных графов [3]. Рассмотрим пример запроса в техподдержку компании SmartBear:

Hi there!
i have a **script test** with a **bunch of checkpoints**, but when it hits a checkpoint cannot be verified, the **execution of the script stops** and **any tests** after the **failed checkpoint** do not get executed.
Thank's in advance.

Жирным выделены словосочетания, имеющие технический смысл и выявленные в результате разбора. Отметим, что предложения *Hi there* и *Thank's in advance* были отфильтрованы на этапе построения концептуального графа, поскольку между словами не обнаружилась семантическая связь. Фрагмент концептуального графа, построенного автоматически из приведенного выше запроса, приведен на рис. 1:

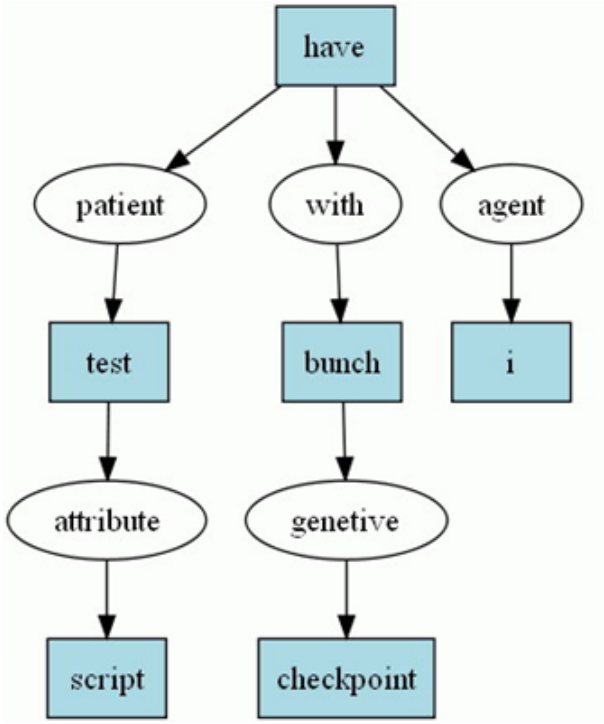

Рис. 1. Фрагмент концептуального графа

Как видно из рисунка, построенный граф позволил выявить взаимосвязь между *I have*, *test script* и *bunch of checkpoints*, хотя в исходном тексте эти словосочетания разделены артиклями и предлогами. Подсчет встречающихся рядом слов, как, например, при латентно-семантическом анализе [4], или в алгоритме TextRank[5],не позволил бы выявить эту связь.

Для построения концептуального графа использовались два внешних ресурса: словарь, хранящий информацию о частях речи, формах слов и т.п. и набор шаблонов, описывающих различные семантические роли, т.е. отношения между словами.

Таким образом, результатом разбора запроса является ряд словосочетаний, состоящих из семантически связанных (а не просто идущих подряд) слов. Если какое-то словосочетание повторяется в тексте несколько раз, ему присваивается больший вес. Далее эти словосочетания объединяются оператором OR (поскольку текст может содержать несколько не связанных друг с другом вопросов) и подаются на вход поисковой системы. В результате задача поиска по тексту сводится к поиску по набору словосочетаний, а эта задача решается, в свою очередь, с помощью предлагаемого алгоритма.

3.3 Поиск по словам и словосочетаниям

Теперь рассмотрим сам алгоритм поиска, использующийся как для поиска по коротким запросам, так и для поиска по текстам. На итоговую релевантность документа запросу влияют следующие факторы:

1. Операторы поиска (AND, AND NOT, OR, "exact phrase", и т.п.).
2. Вес ключевых слов.
3. Позиции слов в документе и знаки препинания.
4. Результаты морфологического разбора запроса.

Сначала производится булевский поиск, т.е. отбор документов, соответствующих имеющимся в запросе операторам поиска (если операторы не заданы явно, все слова считаются объединенными оператором AND). После этого вычисляется релевантность по словосочетаниям. Поскольку проиндексированные документы считаются состоящими из нескольких полей (заголовок и текст, заголовок имеет больший вес), релевантность вычисляется отдельно по каждому полю. После этого, с учетом веса поля, считается итоговая релевантность. В случае нулевой релевантности по словосочетаниям предлагается вычислять релевантность по отдельным ключевым словам. Такой подход позволит получать непустые результаты поиска в случае запросов, не являющихся грамматически корректными сочетаниями слов.

Для вычисления релевантности по ключевым словам предлагается использовать косинус угла между векторами документов. Также рассматривалась и другая популярная формула, Okapi BM25, но эксперименты, проводившиеся на тестовом множестве из 10 000 документов, показали, что оптимальные результаты как по качеству, так и по производительности дает именно косинус угла, часто использующийся для определения смысловой близости при векторной модели текстов:

$$\cos(d_j, q) = \frac{d_j \cdot q}{\|d_j\|\|q\|} = \frac{\sum_{i=1}^{N} w_{i,j} \cdot w_{i,q}}{\sqrt{\sum_{i=1}^{N} w^2{}_{i,j}} \sqrt{\sum_{i=1}^{N} w^2{}_{i,q}}}$$

где $w_{i,j}$ – вес соответствующих слов, вычисленный по формуле tf.

При вычислении релевантности по словосочетаниям учитываются следующие предположения:

1. Словосочетания, повторяющиеся в тексте запроса несколько раз, имеют больший вес (зависящий от количество повторений).
2. Чем больше слов в словосочетании, тем более узкий смысл оно выражает, а значит, тем больше смысловая близость между документами, содержащими это словосочетание.
3. Чем дальше слова расположены друг от друга, тем меньше вероятность того, что они связаны.

Правило подсчета количества искомых словосочетаний в документе основывается на том, что слова, расположенные далеко друг от друга и/или в разных предложениях, скорее всего, не связаны между собой. Соответственно, контекстное окно должно учитывать оба фактора, что легко достигается за счет искусственного увеличения позиции слов при обработке знаков препинания.

Считается, что искомое словосочетание присутствует в документе, если расстояние между каждой парой составляющих его слов меньше некой величины, равной искусственному приращению позиции после конца предложения. Данное условие является более гибким, чем то, что используется в недавно появившемся алгоритме PATeR [2], в котором учитывается только разбиение текста на предложения, но не обрабатываются ситуации, когда два слова находятся на разных концах длинного предложения. Также предлагаемый алгоритм учитывает большее количество факторов (например, количество слов в словосочетании).

Поясним сказанное примером. Допустим, требуется найти словосочетание *AJAX applications testing*.

AJAX web **applications** are, indeed, difficult for **testing**.

В данном случае суммарное расстояние между тремя словами с учетом знаков препинания составляет 7. Соответственно, если после конца предложения позиция увеличивается на 15, считается, что данное предложение содержит искомое словосочетание, поскольку 7 < 15.

No **AJAX applications**. **Testing** desktop applications is another task.

Хотя здесь искомые слова расположены ближе, слово *testing* находится уже в следующем предложении, а значит, суммарное расстояние составляет 16. В этом случае считается, что искомое словосочетание не присутствует в тексте, поскольку 16 > 15.

Релевантность по словосочетаниям предлагается вычислять по следующей формуле:

$$R_{phrase}(q, d_j) = \frac{\sum_{i=1}^{N} R_p(p_i, d_j) w_{p_i}}{N}$$

Иными словами, релевантность документа d_j запросу q вычисляется как среднее арифметическое релевантностей по каждому из рассматриваемых словосочетаний p_i, выделенных из запроса q, с учетом веса каждого словосочетания, вычисляющегося как частота возникновения словосочетания в запросе (аналогично формуле tf). Здесь R_p – релевантность документа словосочетанию p_i, вычисляемая по следующей формуле:

$$R_p(p_i, d_j) = \sum_{k=1}^{m} \frac{2^{2n_{t_i}}}{\Delta_{p_i,k}}$$

где n_{t_i} – количество слов в словосочетании p_i (в общем случае оно может состоять из двух и более слов), $\Delta_{p_i,k}$ – суммарное расстояние между каждым из этих слов в рассматриваемом документе d_j, вычисленное для каждого вхождения в документ словосочетания p_i, m – общее количество вхождений словосочетания p_i в документ d_j. Отметим, что максимальное значение релевантности по ключевым словам не будет превышать 1, а значит документы, имеющие ненулевую релевантность по словосочетаниям (максимальное значение которой не ограничено и может достигать намного больших величин) будут, как правило, стоять в рейтинге выше.

Формула итоговой релевантности учитывает следующие факторы:

1. Релевантность по словосочетаниям должна иметь больший вес, чем релевантность по отдельным словам, но ее отсутствие не должно приводить к нулевому итоговому значению.
2. Релевантность по словам и словосочетаниям вычисляется для каждого проиндексированного поля отдельно, после чего умножается на вес поля.
3. Для того, чтобы сделать формулу итоговой релевантности более универсальной, логично не связывать ее с конкретными алгоритмами, использующимися для вычисления каждой составляющей релевантности.

Вычислять итоговую релевантность предлагается по следующей формуле:

$$R(q, d_j) = \sum_{k=1}^{Nf} w_f R_{phrase}(q, d_j)$$

где *Nf* – общее количество проиндексированных полей (в текущее реализации равно 2), *wf* – вес проиндексированного поля, *Rphrase* – релевантность по словосочетаниям, вычисленная ранее.

3.4 Экспериментальные результаты

Рассматриваемый поисковый алгоритм уже реализован и опробован на практике в системе полнотекстового поиска. Использовалось тестовое множество из 10 000 статей, являющихся документацией по 6 программным продуктам. Поиск проводился по 30 наиболее популярным словосочетаниям, выявленным из статистики поиска по сайту компании SmartBear за 2010 год. В качестве асессоров выступали сотрудники компании, проставлявшие оценки качества первых 10 выданных системой результатов от 0 (не релевантный) до 2 (максимальная релевантность). Для оценки качества предлагаемого поискового алгоритма использовалась формула discounted cumulative gain:

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}$$

где *reli* – проставленная асессором релевантность (от 0 до 2), *i* – порядковый номер результата поиска в выдаче, *p* – количество оцениваемых результатов (в данном эксперименте бралось равное 10). Отметим, что в отличие от популярной метрики p@K (precision at top K), данная формула учитывает как релевантность каждого результата, так и порядковый номер результата поиска в выдаче. График, показывающий усредненные значения результатов, представлен на рис. 2:

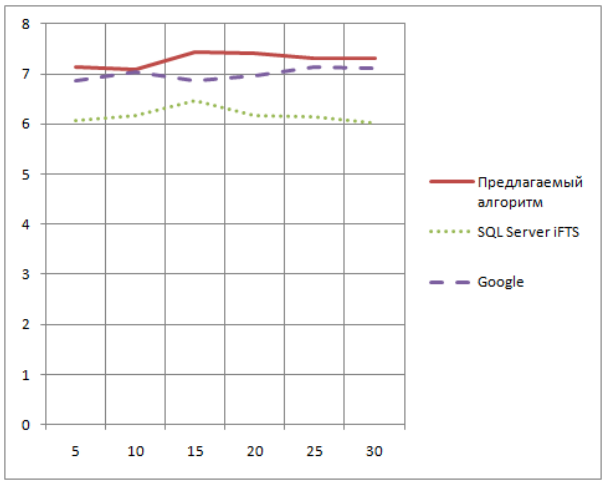

Рис. 2. Качество результатов поиска

Вертикальная ось соответствует качеству результатов поиска (максимально возможное значение равно 10,51). Горизонтальная ось – количеству поисковых запросов, по которым считалась средняя величина DCGp.

Видно, что по мере увеличения количества запросов предлагаемый алгоритм дает для текущей выборки более качественные результаты, чем два других (SQL Server iFTS относится к встроенному поисковому алгоритму SQL Server, используемому в данный момент для поиска по сайту). Низкие результаты этой системы можно объяснить тем, что она использует формулу Okapi BM25, а она, как уже упоминалось, дает на рассматриваемом множестве статей менее качественные результаты. Некоторое улучшение относительно Google можно объяснить тем, что формула релевантности этого поисковика дает очень большой вес «внестраничным» факторам, таким как ссылки на документ, а в случае поиска по документации такой подход, как уже упоминалось ранее, не оправдан.

4. Заключение

Для решения поставленной задачи поиска по длинному запросу, состоящему из нескольких предложений, предлагается алгоритм, выделяющий из текста запроса связанные словосочетания путем построения концептуальных графов. Таким образом, задача поиска по тексту сводится к задаче поиска по словосочетаниям. Для поиска по словосочетаниям предлагается алгоритм, учитывающий большее количество факторов, чем существующие аналогичные алгоритмы. Экспериментальные результаты доказывают эффективность предлагаемого алгоритма поиска по словосочетаниям.

В дальнейшем планируется улучшить качество поиска за счет применения обучения без учителя. Такое обучение на основе статистических данных о поведении пользователей поможет улучшить релевантность выдаваемых результатов. Также планируется ввести обучение распознаванию словосочетаний, которое поможет улучшить качество построения концептуальных графов, и расширит число поддерживаемых языков.

Литература

[1] A World of Conceptual Graphs, http://conceptualgraphs.org/

[2] Bani-Ahmad S.G., Al-Dweik G. A new term-ranking approach that supports improved searching in literature digital libraries // Research Journal of Information Technology, 2011. Volume 3, Number 1, p. 44-52.

[3] Bogatyrev, M.Y., Mitrofanova, O. A., Tuhtin, V.V. Building Conceptual Graphs for Articles Abstracts in Digital Libraries. - Proceedings of the Conceptual Structures Tool Interoperability Workshop (CS-TIW 2009) at 17th International Conference on Conceptual Structures (ICCS'09) - Moscow, Russia, July 2009, - p.p. 50-57.

[4] Landauer, T.K., Foltz, P.W., Laham, D. An Introduction to Latent Semantic Analysis. Discourse Processes, Issue 25, p. 259-284, 1998.

[5] Mihalcea, R., Tarau, P., TextRank: Bringing Order into Texts, - Proceedings of the Conference on Empirical Methods in Natural Language Processing http://www.citeulike.org/user/johnkork/article/430523
[6] TF-IDF, Wikipedia, 10.01.2007 http://en.wikipedia.org/wiki/Tf%E2%80%93idf

A Full-Text Search Algorithm for Long Queries

©Alexey Kolosoff, Michael Bogatyrev

The problem of a full-text search operation with long search queries is considered. A long query is a natural language text which consists of several sentences. Such search operations are of specific interest for questions answering services (forums, Q&A web sites, technical support portals, etc.) to provide for finding answers and/or similar questions automatically.

A full-text search algorithm which solves the problem via reducing the original text to a number of phrases is described. Experimental results prove the effectiveness of the suggested algorithm which performs search on phrases retrieved during query expansion.