Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Волгоградский государственный технический университет»

СЕМЕСТРОВАЯ РАБОТА
ПО КУРСУ «МОДЕЛИРОВАНИЕ СИСТЕМ»

ТЕМА

Математические методы систем поиска информации

Выполнил:
студент группы ИВТ-360
Фамилия И.О. Титов А.К.
(№ зачетки 20131041)

Проверил:
Профессор кафедры
«САПРиПК»
Фоменков С.А.

Волгоград, 2015

Содержание

Введение

Информационный поиск (англ. Information retrieval) — процесс поиска в большой коллекции (хранящейся, как правило, в памяти компьютеров) некоего неструктурированного материала (обычно - документа), удовлетворяющего информационные потребности.

Классическая задача ИП, с которой началось развитие этой области, — это поиск документов, удовлетворяющих запросу, в рамках некоторой статической коллекции документов.

Список задач ИП постоянно расширяется и теперь включает:
a) Классификацию документов;
b) Фильтрацию документов;
c) Кластеризацию документов;
d) Проектирование архитектур поисковых систем и пользовательских интерфейсов
e) Извлечение информации, в частности аннотирования и реферирования документов;
f) Языки запросов и др.

Также, перед движками ИП ставятся некоторые задачи по обработке естественных языков, что включает в себя морфологический анализ, разрешение лексической многозначности и так далее.

В этом реферате я рассмотрю математические методы, помогающие решить некоторые из вышеперечисленных задач.

Ссылки я оформлмил таким образом: в начале главы стоят номера ссылок на документы по данной тематике, с которыми я ознакомился в процессе написания реферата.

Жирным выделены номера, непосредственно использованные в данной (той, в которой они находятся) главе. Остальные номера указывают на ссылки, так или иначе использованные в данной работе.

Модели информационного поиска
Булевская модель

Источники: **[1]**,[2]

Булева модель стала использоваться в информационно-поисковых системах достаточно давно. Это одна из старейших моделей поиска. Основным достоинством такой модели является ее простота, способность работать с большими объемами информации и высокая скорость выполнения поисковых запросов. По этой причине на основе булевой модели построено большое количество поисковых систем.

В булевой модели запросы пользователей представляют собой логические выражения, в которых слова связаны операторами AND, OR и NOT. Для того чтобы документ был найден, в нем должны содержаться все слова, связанные оператором AND, или хотя бы одно из слов, связанных оператором OR. Не трудно заметить, что при сложных запросах, состоящих из нескольких слов, и большом количестве документов в поисковой базе может наблюдаться некий дисбаланс результатов поиска:

- список результатов поиска при использовании оператора AND может оказаться слишком коротким, так как из результатов поиска исключаются все документы, в которых отсутствует хотя бы одно из слов запроса;
- список результатов поиска при использовании оператора OR может оказаться слишком большим, так как в результаты поиска включаются все документы, в которых встречается хотя бы одно из слов запроса.

Кроме того, у булевой модели есть существенный недостаток – в ней нет возможности установить веса термов (слов) и, соответственно, нельзя провести ранжирование результатов поиска. По сути, при поиске документы делятся на две группы – соответствующие и несоответствующие запросу. Так, при использовании оператора AND документы, не содержащие по крайней мере одного из слов запроса, являются столь же несоответствующими запросу, как и документы, не содержащие ни одного из слов запроса. Аналогично при использовании оператора OR: документы, содержащие одно из слов запроса, в равной степени соответствуют запросу, как и документы, содержащие все слова запроса.

Из-за этого современные информационно-поисковые системы практически перестали строиться на основе булевой модели. Современные системы чаще всего используют различные варианты векторной модели, которые позволяют производить ранжирование результатов поиска, обладают неплохими скоростными характеристиками, но требуют большего числа вычислений. Многие информационно-поисковые системы, использующие булеву модель, в настоящее время утрачивают конкурентоспособность. Перевод же на другие модели поиска означает практически полную их замену. Процедура эта довольно трудоемкая и дорогостоящая. По этой причине интерес представляет модификация существующей булевой модели для обеспечения дополнительной гибкости систем.

Решением проблемы является расширение булевой модели, дающее возможность назначать весами термов, осуществлять поиск с частичным соответствием и производить ранжирование результатов поиска.

Впервые расширенная модель была предложена Дж. Солтоном и Е. Фоксом. Основная идея состоит в комбинации булевой формулировки запроса и элементов векторной модели. В сущности, предлагается расширить булеву модель элементами векторной модели.

Эта модифицированная модель получила название расширенной булевой модели

(Extended Boolean Model). В дальнейшем модель была дополнена другими исследователями.

Рассмотрим конъюнктивный (оператор AND, пересечение) логический запрос $q = k_x \wedge k_y$. Согласно булевой модели, если документ содержит только терм k_x или терм k_y, то он не соответствует запросу так же, как документ, не содержащий ни одного из термов. Однако такой бинарный критерий выбора из двух альтернатив является недостаточно гибким. Аналогичная ситуация наблюдается при дизъюнктивных запросах (оператор OR, объединение).

Если рассматриваются только два слова в запросе (терма), то можно отобразить запросы и документы на двумерной карте (см ниже). Документ d_j позиционируется в этом пространстве на основе весов $w_{x,j}$ и $w_{y,j}$, связанных с парами $[k_x, d_j]$ и $[k_y, d_j]$. После проведения нормализации, эти веса будут принимать значения между 0 и 1. Например, эти веса могут вычисляться как TF-IDF коэффициенты (аналогично векторной модели):

$$w_{x,j}=tf_{norm\,x,j} \times idf_{norm\,x},$$

$$tf_{norm\,i,j} = \frac{tf_{i,j}}{tf_{\max x,j}},$$

$$idf_{norm\,x} = \frac{idf_x}{idf_{\max x}},$$

где $tf_{norm\,x,j}$ - нормализованная частота терма k_x в документе d_j;

$idf_{norm\,x}$ - нормализованная инверсная частота документов для терма k_x;

$tf_{x,j}$ - частота терма k_x в документе d_j;

$tf_{max\,x,j}$ - максимальная частота терма k_x в документе d_j;

idf_x - инверсная частота документов для терма k_x;

$idf_{max\,x}$ – максимальная инверсная частота документов для терма k_x;

$f_{x,j}$ - нормализованная частота терма k_x в документе d_j.

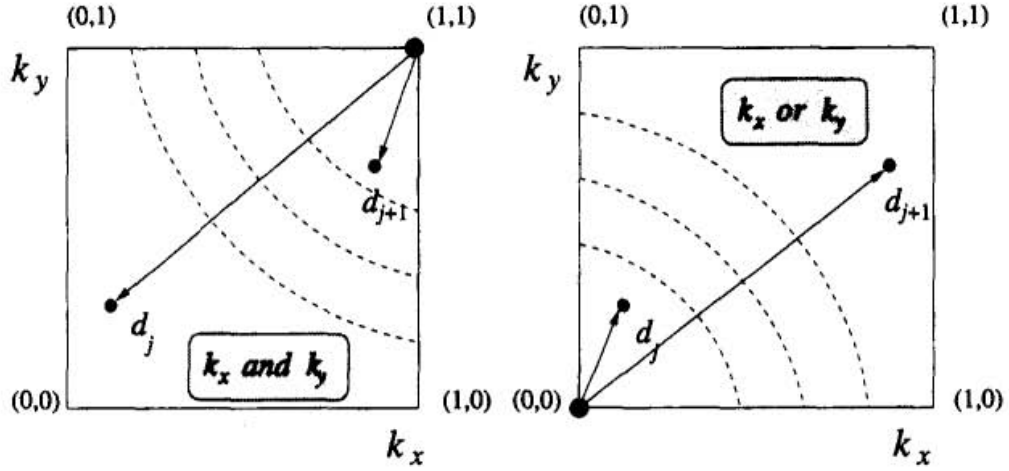


Рис. 1 Пространство, составленное из двух термов k_x и k_y

На рисунке видны две интересные особенности. Во-первых, для дизъюнктивного запроса $q_{or} = k_x \vee k_y$ точка (0,0) является самой нежелательной. Это позволяет считать расстояние от (0,0) как меру подобия запросу q_{or}. Во-вторых, для конъюнктивного запроса $q_{and} = k_x \wedge k_y$ точка (1,1) является самой желательной. Это дает возможность считать расстояние от точки (1,1) как меру подобия запросу q_{and}. Кроме того, такие расстояния могут быть нормализованы.

В результате меры подобия документа запросу будут выглядеть следующим образом:

$$sim(q_{or}, d) = \sqrt{\frac{w_{x,j}^2 + w_{y,j}^2}{2}}$$

$$sim(q_{and}, d) = 1 - \sqrt{\frac{(1 - w_{x,j})^2 + (1 - w_{y,j})^2}{2}}$$

Если веса – логические (то есть $w_{x,j} \in \{0,1\}$), то документ будет располагаться в одном из четырех углов – (0,0), (0,1), (1,0) или (1,1). Тогда значения для $sim(q_{or},d)$ ограничены 0, $1/\sqrt{2}$ и 1. Аналогично, значения для $sim(q_{and},d)$ ограничены 0, $1-1/\sqrt{2}$ и 1. Если учесть, что число термов в коллекции документов равно t, булева модель может быть расширена, чтобы рассматривать евклидовы расстояния в t-мерном пространстве.

Проводя обобщение, можно принять теорию векторных норм. Модель p-нормы обобщает понятие расстояния, включающего не только евклидовы расстояния, но и p-расстояния, где $1 \le p \le \infty$ – вновь введенный параметр, значение которого определяется во время формирования запроса. Обобщенный дизъюнктивный запрос тогда можно представить как

$$q_{or} = k_1 \vee^p k_2 \vee^p \ldots \vee^p k_m$$

Аналогично, обобщенный конъюнктивный запрос можно представить как

$$q_{and} = k_1 \wedge^p k_2 \wedge^p \ldots \wedge^p k_m$$

Тогда функции соответствия документов запросу будут

$$sim(q_{or}, d_j) = \left(\frac{w_{1,j}^p + w_{2,j}^p + \ldots + w_{m,j}^p}{m} \right)^{1/p}$$

$$sim(q_{and}, d_j) = \left(\frac{(1 - w_{1,j})^p + (1 - w_{2,j})^p + \ldots + (1 - w_{m,j})^p}{m} \right)^{1/p}$$

P-норма обладает несколькими интересными свойствами [Baeza-Yates, and et. 1999]. Во-первых, когда $p=1$, получаем

$$sim(q_{or}, d_j) = sim(q_{and}, d_j) = \frac{w_{1,j} + w_{2,j} + \ldots + w_{m,j}}{m}$$

Во-вторых, при $p=\infty$ можно записать

$$sim(q_{or}, d_j) = max(w_{i,j})$$
$$sim(q_{and}, d_j) = min(w_{i,j})$$

Таким образом, для $p=1$ конъюнктивные и дизъюнктивные запросы определяются суммой весов термов в документах, подсчитанных на основе формул подобия в векторном пространстве. Для $p=\infty$ запросы оцениваются согласно терминам нечеткой логики. Изменяя параметр p между 1 и бесконечность, мы можем изменить p-норму, варьируя ранжирование между векторной и булевой моделями. Это весьма хороший аргумент в пользу использования расширенной булевой модели.

Обработка более общих запросов выполняется путем группировки операторов в предопределенном порядке. Например, рассмотрим запрос $q = (k_1 \wedge^p k_2) \vee^p k_3$.

Подобие $sim(q, d_j)$ между документом d_j и этим запросом вычисляется как

$$sim(q,d) = \left(\frac{\left(1 - \left(\frac{(1-w_{1,p})^p + (1-w_{2,p})^p}{2} \right)^{1/p} \right)^p + w_{3,j}^p}{2} \right)^{1/p}$$

Эта процедура может быть применена рекурсивно независимо от числа операторов AND/OR. Еще одна интересная особенность расширенной булевой модели – возможность использования комбинаций различных значений параметра p в том же самом запросе. Например, запрос

$$(k_1 \vee^2 k_2) \wedge^\infty k_3$$

показывает, что k_1 и k_2 должны считаться как в векторной модели, но k_3 должна присутствовать обязательно (то есть конъюнкция интерпретируется как логическая операция).

Надо заметить, что расширенная булева модель ослабляет булеву алгебру, интерпретирующую логические операции в терминах алгебраических расстояний. В этом смысле это – гибридная модель, которая включает свойства и теоретических, и алгебраических моделей.

Приведенные модификации дают возможность обойти ограничения булевой модели и дать новые возможности ее использования в информационно-поисковых системах. Появившаяся возможность осуществления ранжирования делает расширенную булеву модель более конкурентоспособной.

Векторная модель
Источники: [4]

Главная идея векторной модели семантики (vector space model, VSM) – это представление каждого документа коллекции в качестве точки в многомерном пространстве (вектора в векторном пространстве). Близко лежащие друг к другу точки соответствуют семантически схожим документам. В поисковых системах эта идея реализуется следующим образом. Запрос пользователя рассматривается в качестве вектора (как псевдодокумент). Затем, документы сортируются в порядке возрастания расстояния до псевдодокумента и выдаются пользователю.

Векторная модель основана на гипотезе о статистической семантике (statistical semantics hypothesis): статистические зависимости употребления слов человеком могут быть использованы для нахождения заложенного в них смысла.

Сходство документов: матрица термин-документ

При наличии большой коллекции документов и, следовательно, большого числа соответствующих им векторов удобно сгруппировать последние в матрицу. Каждая её строка определяет отдельный термин, а каждый столбец соответствуют некоторому документу.

Пусть имеется документ, который содержит некоторое множество терминов $\{a,a,b,c,c,c\}$ (их порядок в наборе не важен). Тогда ему соответствует вектор $x=(2,1,3)$, где первый элемент соответствует числу вхождений в документ термина a, второй – b, третий – c.

Информационный поиск с использованием матрицы термин-документ (term–document matrix) основывается на следующей гипотезе: оценивание релевантности документа запросу можно производить путем представления документа и запроса в виде набора слов (bag of words). Иными словами, по частоте вхождения слов в документ мы можем судить о его релевантности запросу.

Пусть $\boldsymbol{X}$ – матрица термин-документ. Предположим, что коллекция состоит из n документов и m уникальных терминов. Матрица $\boldsymbol{X}$ будет иметь m строк и n столбцов. Обозначим через w_i – i-ый термин в словаре, d_j – j-ый документ в коллекции. Тогда каждый элемент x_{ij} матрицы $\boldsymbol{X}$ – это количество вхождений термина w_i в документ d_j.

Как правило, большинство значений элементов матрицы $\boldsymbol{X}$ равны 0 (матрица является разреженной). Это связано с тем, что документы содержит лишь малую долю терминов из всего словаря.

Сходство слов: матрица слово-контекст

Матрица термин-документ используется для нахождения сходства среди отдельных документов. Иногда задача может заключаться в рассмотрении близости тех или иных слов в рамках не всего документа, а каких-либо его частей. Матрица слово-контекст (word–context matrix) отражает частоту вхождения терминов в некоторую фразу, предложение, параграф или главу.

Согласно *гипотезе распределения* в лингвистике, слова, встречающиеся в схожих контекстах, стремятся иметь близкий смысл. Эта гипотеза оправдывает использование векторной модели для определения схожести слов.

Слово может быть представлено в виде вектора, элементы которого соответствуют числу вхождений в некоторый контекст. Близость векторов определяет семантическое сходство.

Сходство отношений: матрица пара-модель

В матрице пара-модель (pair–pattern matrix) каждая строка описывает определенную пару слов (такие как каменщик: камень или столяр: дерево), а каждый столбец соответствует модели, определяющей отношение между парами слов (например, X работает с Y).

Важное место здесь занимает *расширенная гипотеза распределения*, согласно которой, отношения, происходящие вместе со схожими парами слов, стремятся иметь близкий смысл. Например, отношение «X решил (что?) Y» и «Y решена (кем?) X» имеет тенденцию употребляться со схожими парами X:Y. В связи с этим, можно предположить, что приведенные отношения имеют схожий смысл.

Согласно гипотезе о скрытых связях, пары слов, которые встречаются в похожих моделях, стремятся иметь близкую семантическую зависимость [1]. Близость векторов-строк матрицы пара-модель определяет степень сходства отношений между словами.

Сходство

Матрицы пара-модель пригодны для измерения сходства семантических отношений между парами слов. В этом случае речь идет о реляционном сходстве (relational similarity). С другой стороны, матрицы слово-контекст используют для измерения атрибутивного сходства (attributional similarity).

Сходство атрибутов между двумя словами a и b, $sim_a(a,b) \in \mathbb{R}$, зависит от степени соответствия свойств a и b. Чем больше соответствие, тем выше атрибутивное сходство. Реляционное сходство между двумя парами слов a:b и c:d, $sim_r(a{:}b,c{:}d)$, зависит от степени соответствия между отношениями a:b и c:d. Чем она выше, тем сильнее реляционное сходство.

Рассмотрим пример. Между словами *собака* и *волк* имеется сравнительно высокая степень атрибутивного сходства, в то время между парами слов *собака:лай* и *кот:мяуканье* существует реляционное сходство.

Вероятностная модель
Источники: [3]**, [5]**

Сначала у пользователей есть информационные потребности, которые они затем переводят в форму запросов. Аналогично существуют документы, которые преобразовываются в представления документов (которые могут отличаться от оригинального документа, содержать много меньше информации, если, например, теряется порядок терминов). Основываясь на этих двух представлениях (запроса и документа), система пытается 17 определить насколько хорошо документы удовлетворяют информационным потребностям. В моделях булева поиска и векторного пространства сопоставление осуществляется в рамках формально определенного, но семантически неточного исчисления индексных терминов. Имея только запрос, система информационного поиска неточно понимает информационную потребность пользователя. Зная представления запроса и документов, система может лишь угадывать, является ли содержание документа релевантным данной информационной потребности. Для того чтобы принимать решения в условиях неопределѐнности, необходим математический аппарат теории вероятностей.

Рассмотрим ранжированный поиск в коллекции документов, при котором пользователь отсылает запросы и получает в ответ упорядоченный список документов. Предположим далее, что оценки релевантности являются бинарными, то есть каждый документ может быть либо релевантным, либо нерелевантным данной информационной потребности. Для запроса q и документа d обозначим $R_{d,q}$ случайную величину – индикатор релевантности документа d по отношению к запросу q. Эта величина равна единице, если документ релевантный, и нулю – в противном случаю.

В рамках вероятностной модели естественно ранжировать результаты поиска по оцененным вероятностям их релевантности информационной потребности: P(R=1|d,q). Эта идея лежит в основе принципа вероятностного ранжирования (Probability Ranking Principle), сформулированного ван Рийсбергеном (van Rijsbergen, 1979) :
«Если в ответ на каждый запрос поисковая система ранжирует документы в коллекции в порядке убывания вероятности их релевантности для пользователя, отославшего запрос, где вероятности оценены как можно более точно на основе доступных данных, то общее качество системы является наилучшим на основе доступных данных.»

Схема взвешивания BM25, или Okapi была разработана как способ построения вероятностной модели, чувствительной к частоте термина и длине документа, но не использующей слишком много дополнительных параметров. В соответствии с ней каждый документ получает рейтинг, определяемый следующей формулой

$$S_{q,d} = \sum_{t \in q} w_{q,t} \cdot w_{d,t}, \qquad \text{где}$$

$$w_{q,t} = \ln\left(\frac{N - f_t + 0.5}{f_t + 0.5}\right) \cdot f_{q,t}$$

$$w_{d,t} = \frac{(k_1 + 1) f_{d,t}}{K_d + f_{d,t}}$$

$$K_d = k_1 \left((1 - b) + b \frac{W_d}{W_A}\right)$$

здесь k1 и b - параметры, устанавливаемые в 1.2 и 0.75 соответственно; Wd и WA – длина документа и средняя длина документа.

Ранжирование
Источники: [3], [5]
Модель случайного блуждания

Рассмотрим сеть из вершин (страницы) и ориентированных ребер (ссылки). Будем моделировать пе- редвижение пользователя по сети следующим образом: пользователь стартует в случайной вершине. С вероятностью ε пользователь переходит в случайную вершину, а с вероятностью 1 − ε он переходит по одному из случайных исходящих ребер. На практике предполагают, что ε ≈ 0,15. Представим себе, что этот пользователь бродит так бесконечно долго. Для каждого k можно опреде- лить $PR_k(i)$ как вероятность оказаться в вершине i через k шагов. Пусть пользователь делает перемещение один раз в секунду. Тогда для каждой страницы существует какая-то вероятность, что пользователь ока- жется в ней через, например, миллиард секунд. Тогда предельная вероятность оказаться в i-й вершине и есть PageRank:

$$PR(i) = \lim_{k \to \infty} PR_k(i).$$

Ранжирование на основе ссылок

Если происходит поиск в Интернет, то расширение методов ранжирования за счет учета ссылочной структуры веба может дать существенное улучшение результатов поиска. Для начала введем понятие веб-графа. Статическая часть веба, состоящая из HTML-документов и гиперссылок между ними может быть представлена в виде направленного графа, в котором каждый узел является веб-страницей, а каждое направленное ребро - гиперссылкой. Совокупность таких узлов и направленных ребер называется веб-графом.

Ссылки в веб-графе не распределены случайным образом. Число ребѐр, входящих в узел, распределено скорее степенному закону, а не по закону Пуассона, как было бы, если бы ссылки были расставлены случайным образом. Рассмотрим два интуитивных предположения, на основе которых базируется изложение методов анализа ссылок в веб-графе.

1. Текст ссылки, указывающей на страницу B, является хорошим описанием страницы B.
2. Гиперссылка со страницы A на страницу B представляет собой признание авторитетности страницы B со стороны создателя страницы A. Это не всегда 20 так; например, многие ссылки со страницы на страницу внутри одного сайта обязаны своим появлением общему шаблону сайта. Например, большинство корпоративных веб-сайтов имеют на каждой странице ссылки на уведомление об авторском праве. Ясно, что это не является свидетельством одобрения. Соответственно, алгоритмы анализа ссылки учитывают такие ссылки с меньшим весом.

Метод PageRank
Источники: [5]

Рассмотрим методы вычисления весов и ранжирования, вытекающие исключительно из структуры ссылок. Первый метод присваивает каждому узлу веб-графа вес в диапазоне от 0 до 1, известный как PageRank. Вес узла зависит от структуры ссылок.

Рассмотрим процесс случайного перемещения по сети, начинающееся с какой-либо страницы (узла веб-графа), и подчиняющееся следующим правилам. В каждый момент времени пользователь с страницы A переходит на случайную страницу, на которую со страницы B ведет гиперссылка. Например, если со страницы A гиперссылки ведут на страницы B, C и D, то "путешественник", находясь в узле A, в следующий момент времени с равной вероятностью (p = 1/3) перейдет на одну из страниц B, C и D.

Ясно, что путешественник будет попадать в одни узлы чаще других; интуитивно понятно, что на эти узлы имеется много ссылок, находящихся на других часто посещаемых узлах. Идея алгоритма PageRank заключается в том, что узлы, часто посещаемые в результате случайного блуждания по сети, имеют большую важность, чем редко посещаемые узлы.

Если в узле A нет исходящий ссылок, то происходит телепортация (teleport) в случайный узел сети. Эта операция эквивалентна тому, что путешественник набирает URL страницы в строке браузера. Вероятность попасть в каждый узел равно 1/N, где N - общее число узлов в графе.

Операция телепортации используется двояко:

1. Если узел не имеет исходящих ссылок, то "путешественник" вызывает операцию телепортации.
2. Если узел имеет исходящие ссылки, то телепортация происходит с вероятностью α, где α обычно равно 0.1, а с вероятностью 1- α переходит по случайной исходящей ссылке.

Применяя теорию марковских цепей, можно показать, что "путешественник", следующий по описанному комбинированному алгоритму, находится в каждом узле ν 21 фиксированную долю времени $\pi(\nu)$, которая зависит от 1) структуры веб-графа и 2) от значения α. Величина $\pi(\nu)$ называется весом PageRank узла ν.

Для подсчета PageRank можно использовать степенной метод (power iteration). Он имитирует случайное блуждание: начав с определенного состояния и выполнив большое число шагов t, мы отслеживаем частоты посещения каждого состояния. После большого количества шагов t эти частоты "устанавливаются", так что разница между вычисленными частотами опускается ниже заданного порогового значения. Эти частоты объявляются весами PageRank.

Значения весов PageRank не зависят от запроса пользователя. Таким образом, вес PageRank является мерой статического качества веб-страницы, не зависящей от запросов пользователей.

Метод HITS (Hyperlink-Induced Topic Search)
Источники: [5]

В этом методе по заданному запросу каждая веб-страница получает два показателя: показатель портальности (hub score) и показатель авторитетности (authority score). В этом методе вычисляется два ранжированных списка страниц, а не один. Метод основывается на разделении страниц на два основных типа: порталы (hub) и авторитетные источники (authority). Такое разделение особенно уместно при широком поиске (broad-topic search), когда запросы имеют вид "Я хочу знать о МГУ". Авторитетным источником является официальный сайт университета. Порталами являются сайты, содержащие ссылки на авторитетные источники. Эти страницы не содержат в себе необходимую инвормацию, но содержат ссылки на нее, собранные заинтересованными людьми.

Показатель портальности будем обозначать h(ν), а показатель авторитетности – как a(ν). Гиперссылку со страницы ν на страницу α будем обозначать $\nu \mapsto \alpha$. В начале для всех узлов h(ν) = a(ν) = 1 рассмотрим итеративный процесс.

$$h(v) \leftarrow \sum_{v \mapsto y} a(y)$$

$$a(v) \leftarrow \sum_{y \mapsto v} h(y)$$

Таким образом, показатель портальности страницы определяется как сумма показателей авторитетности страниц, на которые она ссылается. С другой стороны, если на страницу ссылаются хорошие порталы, то показатель еѐ авторитетности увеличивается. Исследуем результат многократного итеративного применения указанных выше формул. 22 Пусть A – квадратная матрица смежности подмножества веб-графа, в которой каждой странице соответствует одна строка и один столбец. Элемент Aij равен единице, если существует гиперссылка со страницы i на страницу j, и нулю – в противном случае. Тогда формулы перепишутся следующим образом:

или, после взаимной подстановки

Если заменить символы символами = и ввести неизвестное собственное значение, то получим уравнения для собственных векторов.

$$\vec{h} = \frac{1}{\lambda_h} A A^T \vec{h}$$

$$\vec{a} = \frac{1}{\lambda_a} A^T A \vec{h}$$

Здесь λ_h - собственное значение матрицы AA^T, а λ_a – собственное значение матрицы $A^T A$.

Отсюда следуют важные следствия.

1. Результат итеративного вычисления стремится к стационарному значению, определенному структурой графа.
2. Для вычисления результата можно применять любой метод вычисления собственного вектора стохастической матрицы, не обязательно ограничиваться итеративным алгоритмом.

Метод HITS имеет следующий вид:

1. Выбираем целевое множество веб-страниц, создаем веб-граф, индуцированный гиперссылками, и вычисляем матрицы AA^T и A^TA.
2. Вычисляем главные собственные вектора и создаем вектор показателей портальности и вектор показателей авторитетности.
3. Возвращаем веб-страницы с высоким показателем портальности и высоким показателем авторитетности.

Также опишем алгоритм выбора подмножества веб-страниц, соответствующих определенной теме.

1. По заданному запросу с помощью текстового индекса находим страницы, содержащие термины из запроса. Это множество назовем корневым (**root set**).
2. Построим базовое множество (**base set**) страниц, включающее в себя корневое множество, а также любую веб-страницу из корневого множества, которая либо сама ссылается на страницу из корневого множества, либо на нее ссылается какая- то страница из корневого множества.

Базовое множество будет использоваться для создания списков авторитетности и списков портальности.

Нечеткий поиск
Источники: [7]

Алгоритмы нечеткого поиска (также известного как *поиск по сходству* или *fuzzy string search*) являются основой систем проверки орфографии и полноценных поисковых систем вроде Google или Yandex. Например, такие алгоритмы используются для функций наподобие «Возможно вы имели в виду …» в тех же поисковых системах.

В этой статье я рассмотрю следующие понятия, методы и алгоритмы:

- •Расстояние Левенштейна
- •Расстояние Дамерау-Левенштейна
- •Алгоритм Bitap с модификациями от Wu и Manber
- •Алгоритм расширения выборки
- •Метод N-грамм
- •Хеширование по сигнатуре
- •BK-деревья

А также будет проведено сравнительное тестирование качества и производительности алгоритмов.

Нечеткий поиск является крайне полезной функцией любой поисковой системы. Вместе с тем, его эффективная реализация намного сложнее, чем реализация простого поиска по точному совпадению.

Задачу нечеткого поиска можно сформулировать так:
«По заданному слову найти в тексте или словаре размера **n** все слова, совпадающие с этим словом (или начинающиеся с этого слова) с учетом **k** возможных различий».

Например, при запросе «Машина» с учетом двух возможных ошибок, найти слова «Машинка», «Махина», «Малина», «Калина» и так далее.

Алгоритмы нечеткого поиска характеризуются *метрикой* — функцией расстояния между двумя словами, позволяющей оценить степень их сходства в данном контексте. Строгое математическое определение *метрики* включает в себя необходимость соответствия условию неравенства треугольника (X — множество слов, p — метрика):

$$\rho(x,y) \leqslant \rho(x,z) + \rho(z,y), \quad x,y,z \in X.$$

Между тем, в большинстве случаев под метрикой подразумевается более общее понятие, не требующее выполнения такого условия, это понятие можно также назвать *расстоянием*.

В числе наиболее известных метрик - расстояния *Хемминга*, *Левенштейна* и *Дамерау-Левенштейна*. При этом расстояние Хемминга является метрикой только на множестве слов одинаковой длины, что сильно ограничивает область его применения.

Впрочем, на практике расстояние Хемминга оказывается практически бесполезным, уступая более естественным с точки зрения человека метрикам, о которых и пойдет речь ниже.

Расстояние Левенштейна

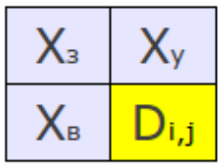
Наиболее часто применяемой метрикой является расстояние Левенштейна, или расстояние редактирования, алгоритмы вычисления которого можно найти на каждом шагу.

Тем не менее, стоит сделать несколько замечаний относительно наиболее популярного алгоритма расчета — метода Вагнера-Фишера.

Исходный вариант этого алгоритма имеет временную сложность **O(mn)** и потребляет **O(mn)** памяти, где **m** и **n** — длины сравниваемых строк. Весь процесс можно представить следующей матрицей:

		А	Р	Е	С	Т	А	Н	Т
	0	1	2	3	4	5	6	7	8
Д	1	1	2	3	4	5	6	7	8
А	2	1	2	3	4	5	5	6	7
Г	3	2	2	3	4	5	6	6	7
Е	4	3	3	2	3	4	5	6	7
С	5	4	4	3	2	3	4	5	6
Т	6	5	5	4	3	2	3	4	5
А	7	6	6	5	4	3	2	3	4
Н	8	7	7	6	5	4	3	2	3

Если посмотреть на процесс работы алгоритма, несложно заметить, что на каждом шаге используются только две последние строки матрицы, следовательно, потребление памяти можно уменьшить до **O(min(m, n))**.



$$D_{ij} = min\left(X_в + 1, X_у + 1, X_з + C_{замены}\right)$$

$$C_{замены} = \begin{cases} 1, & если\ S_1[i] \neq S_2[j] \\ 0, & иначе \end{cases}$$

Но это еще не всё — можно дальше оптимизировать алгоритм, если стоит задача нахождения не более k различий. В этом случае нужно вычислять в матрице лишь диагональную полосу шириной **2k+1** (отсечение Укконена), что сводит временную сложность к **O(k min(m, n))**.

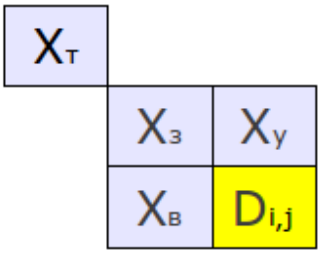
Префиксное расстояние

Также бывает необходимо вычислять расстояние между префиксом-образцом и строкой — т. е. найти расстояние между заданным префиксом и ближайшим префиксом строки. В этом случае необходимо взять наименьшее из расстояний от префикса-образца до всех префиксов строки. Очевидно, что префиксное расстояние не может считаться метрикой в строгом математическом смысле, что ограничивает его применение.

Зачастую при нечетком поиске важно не столько само значение расстояния, сколько факт того, превышает оно или нет определенную величину.

Расстояние Дамерау-Левенштейна

Эта вариация вносит в определение расстояния Левенштейна еще одно правило — *транспозиция* (перестановка) двух соседних букв также учитывается как одна операция, наряду со вставками, удалениями и заменами.
Еще пару лет назад Фредерик Дамерау мог бы гарантировать, что большинство ошибок при наборе текста — как раз и есть транспозиции. Поэтому именно данная метрика дает наилучшие результаты на практике.



$$D_{ij} = min\left(X_в + 1, X_y + 1, X_з + C_{замены}, X_т + C_{транспозиции}\right)$$

$$C_{замены} = \begin{cases} 1, & если\ S_1[i] \neq S_2[j] \\ 0, & иначе \end{cases}$$

$$C_{транспозиции} = \begin{cases} 1, & если\ \ S_1[i] = S_2[j-1]\ \ и\ \ S_1[i-1] = S_2[j] \\ \infty, & иначе \end{cases}$$

Чтобы вычислять такое расстояние, достаточно немного модифицировать алгоритм нахождения обычного расстояния Левенштейна следующим образом: хранить не две, а три последних строки матрицы, а также добавить соответствующее дополнительное условие — в случае обнаружения транспозиции при расчете расстояния также учитывать и её стоимость.

Кроме рассмотренных выше, существует еще множество других, иногда применяющихся на практике расстояний, таких как метрика Джаро-Винклера, многие из которых доступны в библиотеках SimMetrics и SecondString.

Алгоритмы нечеткого поиска без индексации

Эти алгоритмы предназначены для поиска по заранее неизвестному тексту, и могут быть использованы, например, в текстовых редакторах, программах для просмотра документов или в веб-браузерах для поиска по странице. Они не требуют предварительной обработки текста и могут работать с непрерывным потоком данных.

Линейный поиск

Простое последовательное применение заданной метрики (например, метрики Левенштейна) к словам из входного текста. При использовании метрики с ограничением, этот метод позволяет добиться оптимальной скорости работы. Но, при этом, чем больше **k**, тем сильнее возрастает время работы. Асимптотическая оценка времени — **O(kn)**.
Bitap (также известный как Shift-Or или Baeza-Yates-Gonnet, и его модификация от Wu-Manber)

Алгоритм *Bitap* и различные его модификации наиболее часто используются для нечеткого поиска без индексации. Его вариация используется, например, в unix-утилите agrep, выполняющей функции аналогично стандартному grep, но с поддержкой ошибок в поисковом запросе и даже предоставляя ограниченные возможности для применения регулярных выражений.

Впервые идею этого алгоритма предложили граждане **Ricardo Baeza-Yates** и **Gaston Gonnet**, опубликовав соответствующую статью в 1992 году.
Оригинальная версия алгоритма имеет дело только с заменами символов, и, фактически, вычисляет расстояние *Хемминга*. Но немного позже **Sun Wu** и **Udi Manber** предложили модификацию этого алгоритма для вычисления расстояния*Левенштейна*, т.е. привнесли поддержку вставок и удалений, и разработали на его основе первую версию утилиты agrep.

Операция Bitshift

$$R = \begin{matrix}0\\0\\0\\0\\0\end{matrix} \xrightarrow{Bitshift} \begin{matrix}1\\0\\0\\0\\0\end{matrix} \xrightarrow{Bitshift} \begin{matrix}1\\1\\0\\0\\0\end{matrix} \xrightarrow{Bitshift} \begin{matrix}1\\1\\1\\0\\0\end{matrix}$$

Вставки

$$R^k_{\text{вставки } j+1} = (\,Bitshift(R^k_j) \wedge s_x) \vee R^{k-1}_j$$

Удаления

$$R^k_{\text{удаления } j+1} = (Bitshift\,(R^k_j) \wedge\ s_x) \vee Bitshift\,(R^{k-1}_{j+1})$$

Замены

$$R^k_{\text{замены } j+1} = (Bitshift\,(R^k_j) \wedge\ s_x) \vee Bitshift\,(R^{k-1}_j)$$

Результирующее значение

$$R^k_{j+1} = R^k_{\text{вставки } j+1} \vee R^k_{\text{удаления } j+1} \vee R^k_{\text{замены } j+1}$$

Где **k** — количество ошибок, **j** — индекс символа, **s**x — маска символа (в маске единичные биты располагаются на позициях, соответствующих позициям данного символа в запросе).

Совпадение или несовпадение запросу определяется самым последним битом результирующего вектора R.

Высокая скорость работы этого алгоритма обеспечивается за счет битового параллелизма вычислений — за одну операцию возможно провести вычисления над 32 и более битами одновременно.
При этом тривиальная реализация поддерживает поиск слов длиной не более 32. Это ограничение обуславливается шириной стандартного типа *int* (на 32-битных архитектурах). Можно использовать и типы больших размерностей, но это может в некоторой степени замедлить работу алгоритма.

Не смотря на то, что асимптотическое время работы этого алгоритма **O(kn)** совпадает с таковым у линейного метода, он значительно быстрее при длинных запросах и количестве ошибок **k** более 2.

Тестирование

Тестирование осуществлялось на тексте 3.2 млн слов, средняя длина слова — 10.

Точный поиск
Время поиска: 3562 мс

Поиск с использованием метрики Левенштейна
Время поиска при *k=2*: 5728 мс
Время поиска при *k=5*: 8385 мс

Поиск с использованием алгоритма Bitap с модификациями Wu-Manber
Время поиска при *k=2*: 5499 мс
Время поиска при *k=5*: 5928 мс

Очевидно, что простой перебор с использованием метрики, в отличие от алгоритма Bitap, сильно зависит от количества ошибок **k**.

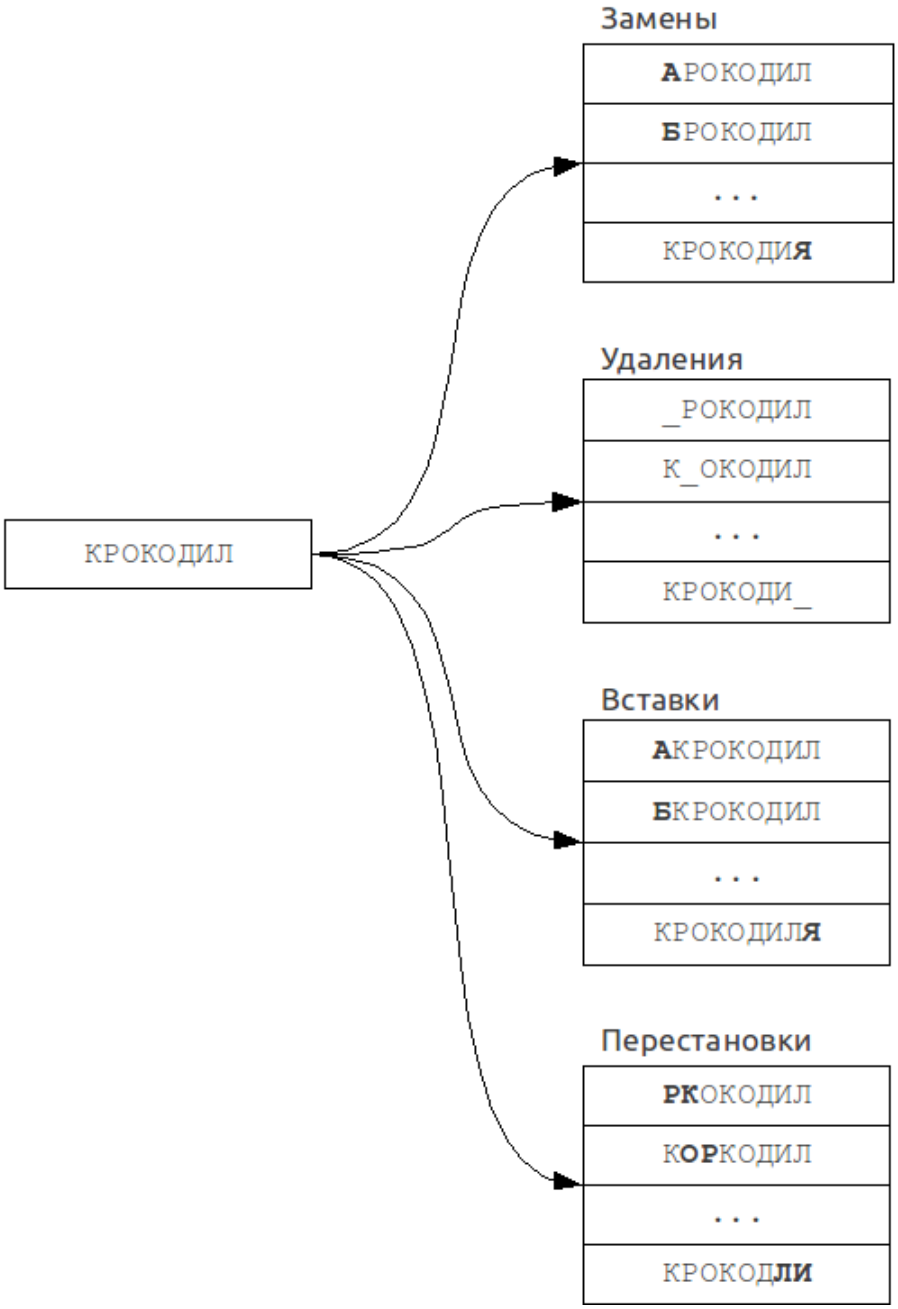
Тем не менее, если речь заходит о поиске в неизменных текстах большого объема, то время поиска можно значительно сократить, произведя предварительную обработку такого текста, также называемую *индексацией*.

Алгоритмы нечеткого поиска с индексацией

Источники: [7]

Особенностью всех алгоритмов нечеткого поиска с индексацией является то, что индекс строится по словарю, составленному по исходному тексту или списку записей в какой-либо базе данных.

Эти алгоритмы используют различные подходы к решению проблемы — одни из них используют сведение к точному поиску, другие используют свойства метрики для построения различных пространственных структур и так далее.

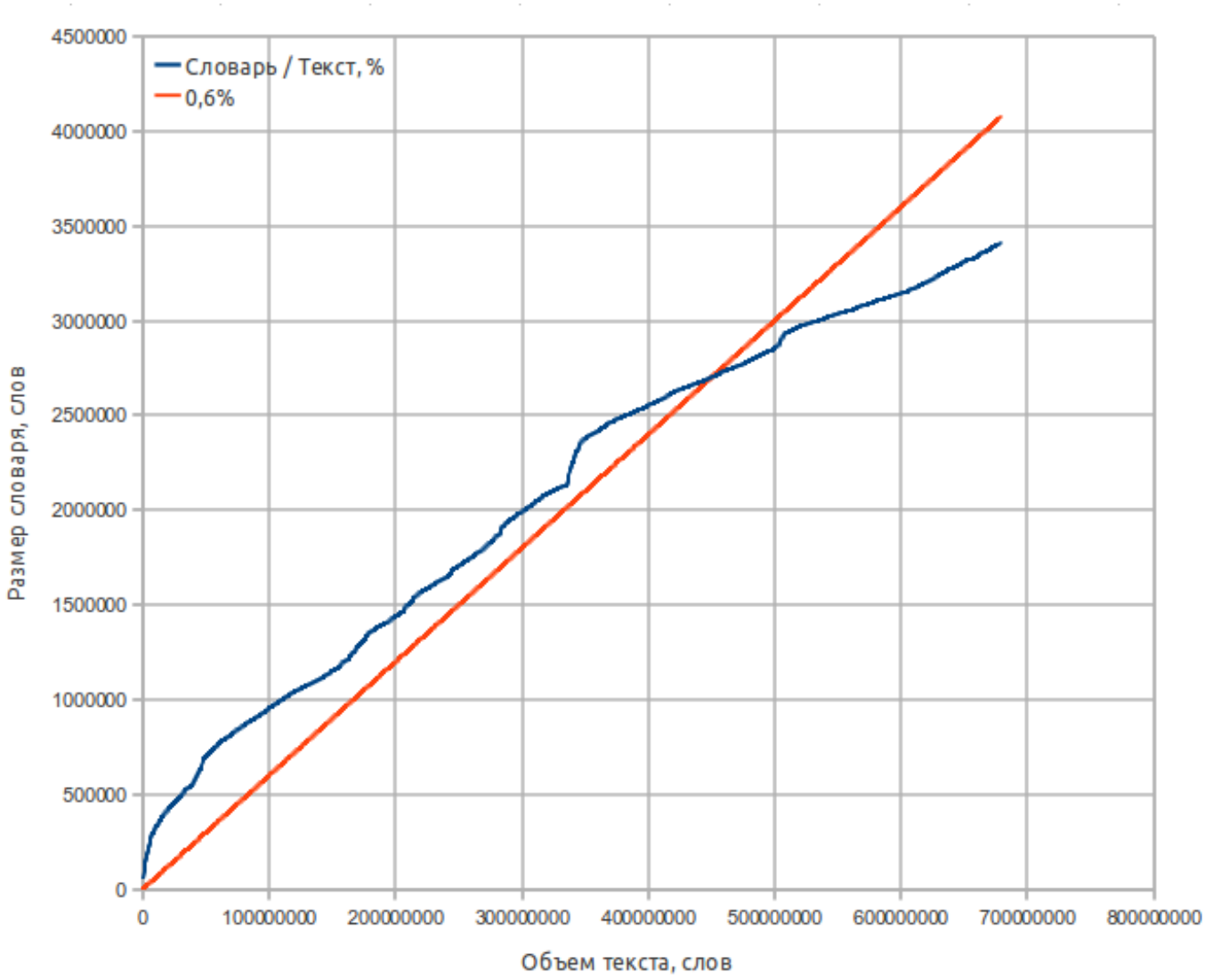


Прежде всего, на первом шаге по исходному тексту строится словарь, содержащий слова и их позиции в тексте. Также, можно подсчитывать частоты слов и словосочетаний для улучшения качества результатов поиска.
Предполагается, что индекс, как и словарь, целиком загружен в память.

Тактико-технические характеристики словаря:

- •Исходный текст — 8.2 гигабайта материалов библиотеки Мошкова (lib.ru), 680 млн слов;
- •Размер словаря — 65 мегабайт;
- •Количество слов — 3.2 млн;
- •Средняя длина слова — 9.5 символов;
- •Средняя квадратичная длина слова (может быть полезна при оценке некоторых алгоритмов) — 10.0 символов;
- •Алфавит — заглавные буквы А-Я, без Ё (для упрощения некоторых операций). Слова, содержащие символы не из алфавита, не включены в словарь.

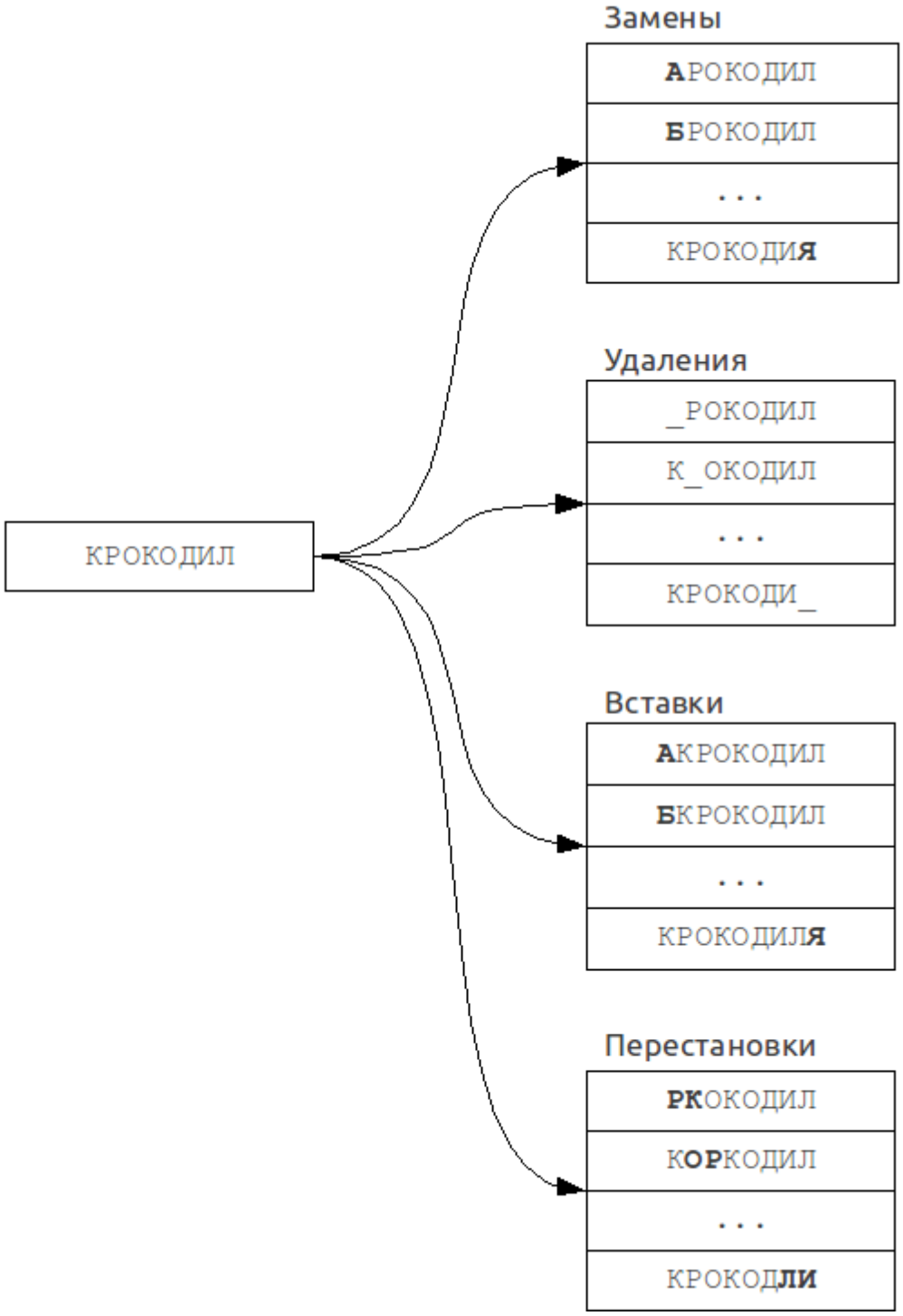
Зависимость размера словаря от объема текста не является строго линейной — до некоторого объема формируется базовый каркас слов, составляющий от 15% на 500 тысячах слов до 5% на 5 миллионах, и затем зависимость приближается к линейной, медленно убывая и доходя до 0.5% на 680 млн слов. Последующее сохранение роста обеспечивается в большинстве своем за счет редких слов.



Алгоритм расширения выборки

Этот алгоритм часто применяется в системах проверки орфографии (т.е. в spell-checker'ax), там, где размер словаря невелик, либо же где скорость работы не является основным критерием.
Он основан на сведении задачи о нечетком поиске к задаче о точном поиске.

Из исходного запроса строится множество «ошибочных» слов, для каждого из которых затем производится точный поиск в словаре.



Время его работы сильно зависит от числа k ошибок и от размера алфавита A, и в случае использования бинарного поиска по словарю составляет: $O((m|A|)^k \cdot m \cdot \log n)$

Например, при $k = 1$ и слова длины 7 (например, «Крокодил») в русском алфавите множество ошибочных слов будет размером около 450, то есть будет необходимо сделать 450 запросов к словарю, что вполне приемлемо.

Но уже при $k = 2$ размер такого множества будет составлять более 115 тысяч вариантов, что соответствует полному перебору небольшого словаря, либо же 1 / 27 в нашем случае, и, следовательно, время работы будет достаточно велико. При этом не нужно забывать еще и о том, что для каждого из таких слов необходимо провести поиск на точное совпадение в словаре.

Особенности:

Алгоритм может быть легко модифицирован для генерации «ошибочных» вариантов по произвольным правилам, и, к тому же, не требует никакой предварительной обработки словаря, и, соответственно, дополнительной памяти.

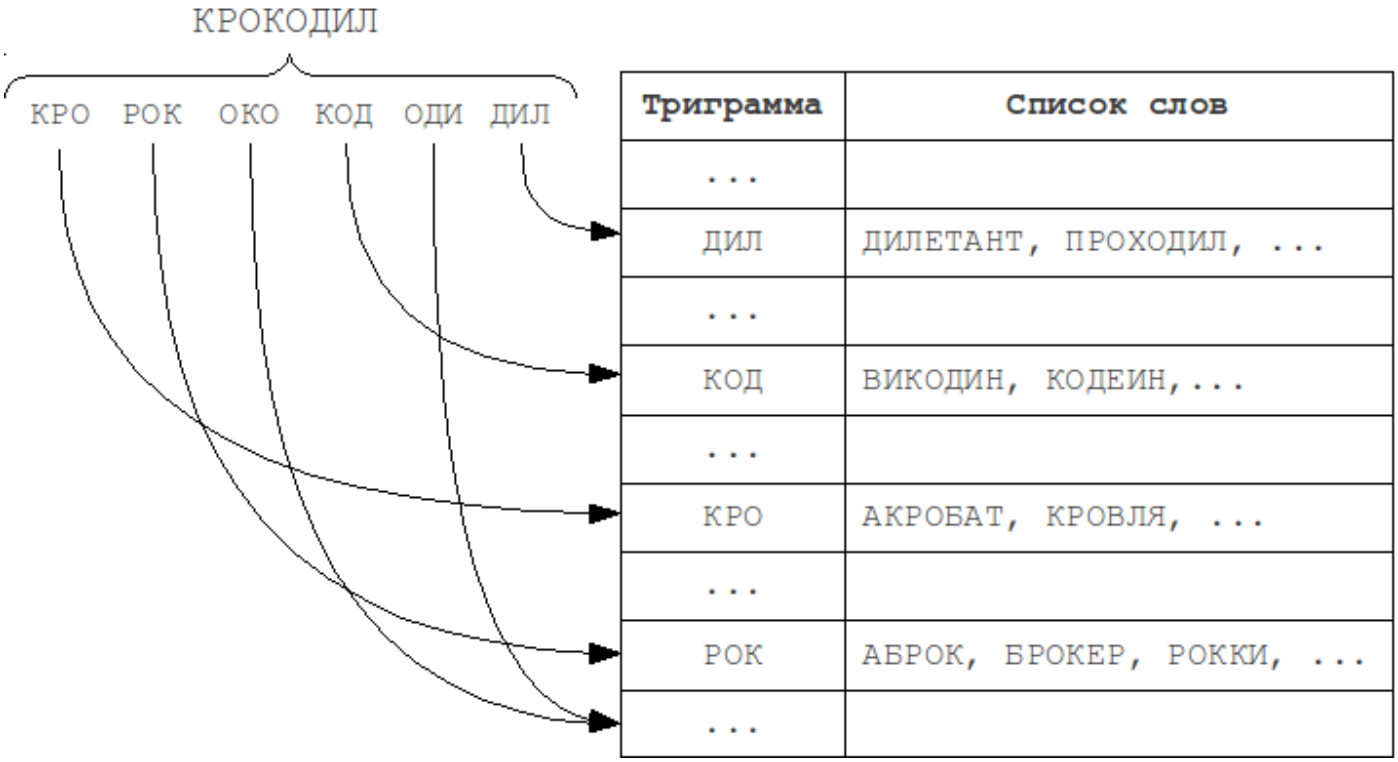
Возможные улучшения:

Можно генерировать не всё множество «ошибочных» слов, а только те из них, которые наиболее вероятно могут встретиться в реальной ситуации, например, слова с учетом распространенных орфографических ошибок или ошибок набора.

Метод N-грамм
Источники: [7]

Этот метод был придуман довольно давно, и является наиболее широко используемым, так как его реализация крайне проста, и он обеспечивает достаточно хорошую производительность. Алгоритм основывается на принципе:
«Если слово А совпадает со словом Б с учетом нескольких ошибок, то с большой долей вероятности у них будет хотя бы одна общая подстрока длины N».
Эти подстроки длины N и называются N-граммами.

Во время индексации слово разбивается на такие N-граммы, а затем это слово попадает в списки для каждой из этих N-грамм. Во время поиска запрос также разбивается на N-граммы, и для каждой из них производится последовательный перебор списка слов, содержащих такую подстроку.



Триграмма	Список слов
...	
ДИЛ	ДИЛЕТАНТ, ПРОХОДИЛ, ...
...	
КОД	ВИКОДИН, КОДЕИН,...
...	
КРО	АКРОБАТ, КРОВЛЯ, ...
...	
РОК	АБРОК, БРОКЕР, РОККИ, ...
...	

Наиболее часто используемыми на практике являются триграммы-подстроки длины 3. Выбор большего значения N ведет к ограничению на минимальную длину слова, при которой уже возможно обнаружение ошибок.

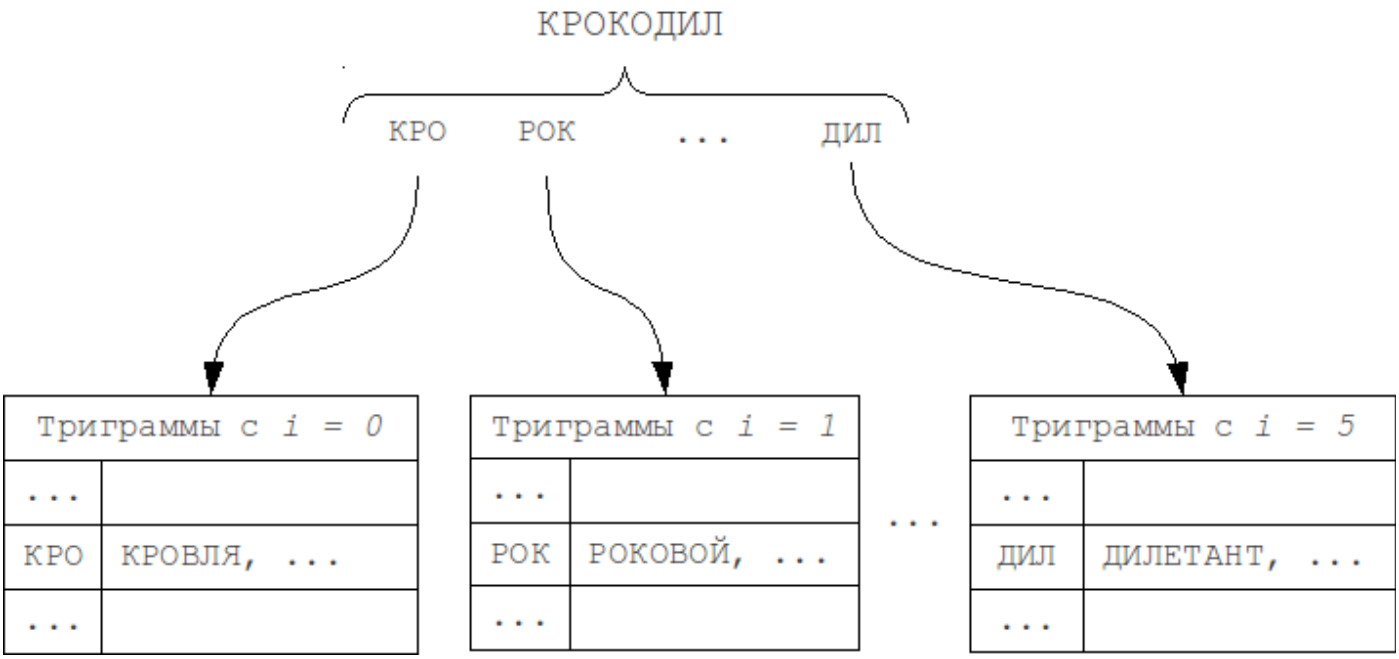
Особенности:

Алгоритм N-грамм находит не все возможные слова с ошибками. Если взять, например, слово ВОТКА, и разложить его на триграммы: ВО**Т**КА → ВО**Т** О**ТК** **Т**КА — то можно заметить, что они все содержат ошибку Т. Таким образом, слово «ВОДКА» найдено не будет, так как оно не содержит ни одной из этих триграмм, и не попадет в соответствующие им списки. Таким образом, чем меньше длина слова и чем больше в нем ошибок, тем выше шанс того, что оно не попадет в соответствующие N-граммам запроса списки, и не будет присутствовать в результате.

Между тем, метод N-грамм оставляет полный простор для использования собственных метрик с произвольными свойствами и сложностью, но за это приходится платить — при его использовании остается необходимость в последовательном переборе около 15% словаря, что достаточно много для словарей большого объема.

Возможные улучшения:

Можно разбивать хеш-таблицы N-грамм по длине слов и по позиции N-граммы в слове (модификация 1). Как длина искомого слова и запроса не могут отличаться более чем на **k**, так и позиции N-граммы в слове могут различаться не более чем на k. Таким образом, необходимо будет проверить лишь таблицу, соответствующую позиции этой N-граммы в слове, а также k таблиц слева и k таблиц справа, т.е. всего **2k+1** соседних таблиц.



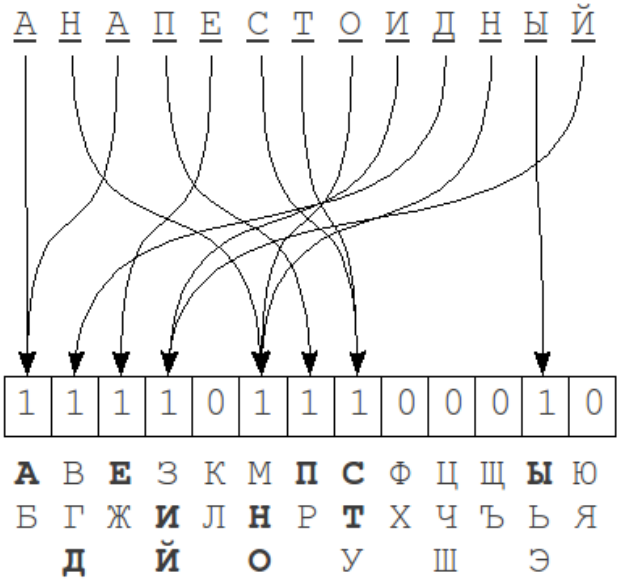
Можно еще немного уменьшить размер необходимого для просмотра множества, разбив таблицы по длине слова, и аналогичным образом просматривая только соседние **2k+1** таблицы (модификация 2).

Хеширование по сигнатуре
Источники: [7]

Этот алгоритм описан в статье Бойцова Л.М. «Хеширование по сигнатуре». Он базируется на достаточно очевидном представлении «структуры» слова в виде битовых разрядов, используемой в качестве хеша (сигнатуры) в хеш-таблице.

При индексации такие хеши вычисляются для каждого из слов, и в таблицу заносится соответствие списка словарных слов этому хешу. Затем, во время поиска, для запроса вычисляется хеш и перебираются все соседние хеши, отличающиеся от исходного не более чем в k битах. Для каждого из таких хешей производится перебор списка соответствующих ему слов.

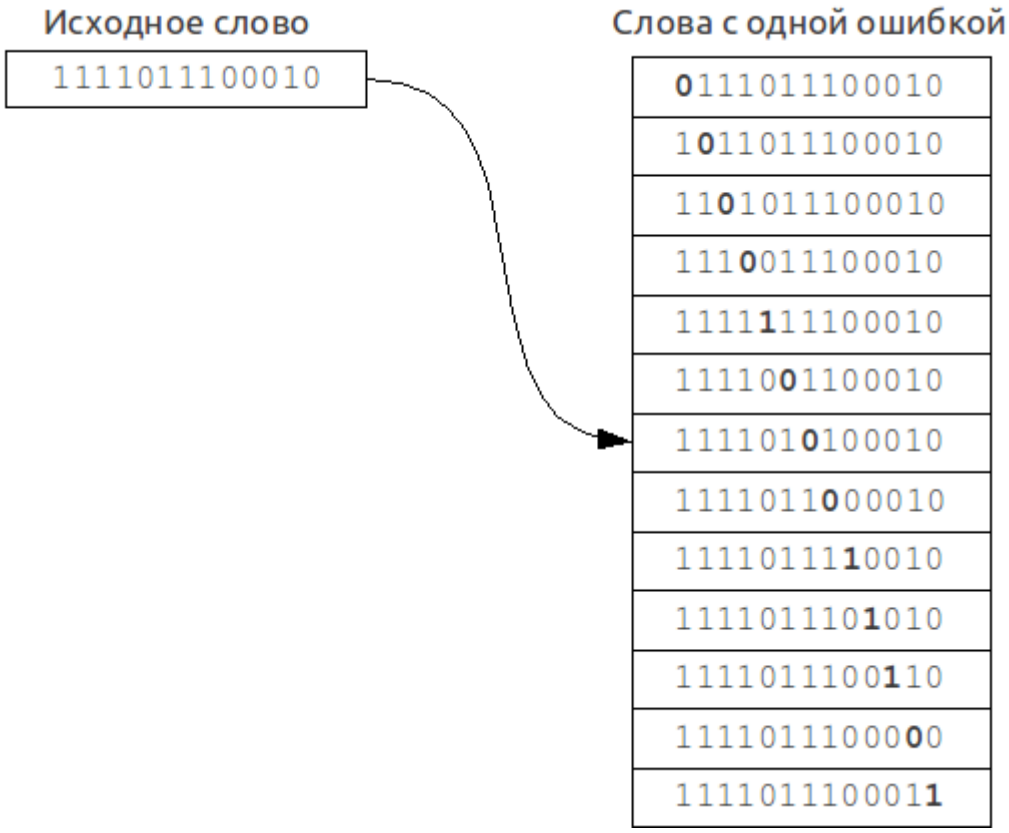
Процесс вычисления хеша — каждому биту хеша сопоставляется группа символов из алфавита. Бит 1 на позиции *i* в хеше означает, что в исходном слове присутствует символ из *i-ой* группы алфавита. Порядок букв в слове абсолютно никакого значения не имеет.



Хеш	Список слов
0000000000000	-
...	
1111011100001	АВТОПРЕДПРИЯТИЕ, БЕЗОТРАДНАЯ, ВЕТЕРИНАРИЯ, ...
1111011100010	ПРЕВРАТНЫЙ, БЕЗРАССУДНЫЙ, АНАПЕСТОИДНЫЙ, ...
1111011100011	СОРИЕНТИРОВАТЬСЯ, БЕСПРЕПЯТСТВЕННЫЙ, ...
...	
1111111111111	ЛЕГКОИСЧЕРПЫВАЮЩИХСЯ, ВЫСОКОРАЗРЕШАЮЩИХ, ...

Удаление одного символа либо не изменит значения хеша (если в слове еще остались символы из той же группы алфавита), либо же соответствующий этой группе бит изменится в 0.

При вставке, аналогичным образом либо один бит встанет в 1, либо никаких изменений не будет. При замене символов всё немного сложнее — хеш может либо вовсе остаться неизменным, либо же изменится в 1 или 2 позициях. При перестановках никаких изменений и вовсе не происходит, потому что порядок символов при построении хеша, как и было замечено ранее, не учитывается. Таким образом, для полного покрытия k ошибок нужно изменять не менее **2k** бит в хеше.



Время работы, в среднем, при k «неполных» (вставки, удаления и транспозиции, а также малая часть замен) ошибках:

$$O(|H|^k \cdot \frac{n}{2^{|H|}})$$

Особенности:

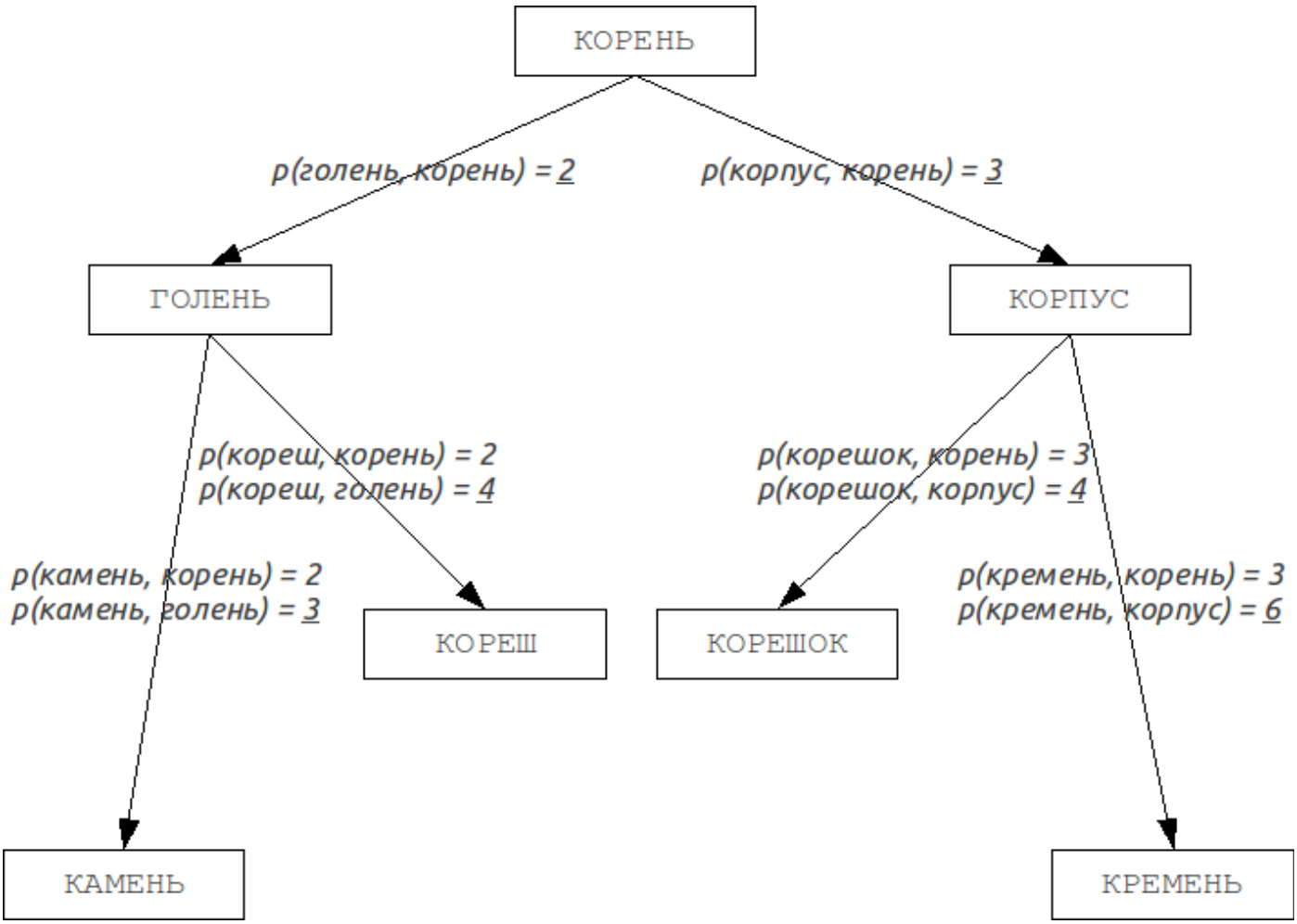
Из того, что при замене одного символа могут изменятся сразу два бита, алгоритм, реализующий, например, искажения не более 2 битов одновременно в действительности не будет выдавать полного объема результатов из-за отсутствия значительной (зависит от отношения размера хеша к алфавиту) части слов с двумя заменами (и чем больше размер хеша, тем чаще замена символа будет приводить к искажению сразу двух бит, и тем менее полным будет результат). К тому же, этот алгоритм не позволяет проводить префиксный поиск.

BK-деревья

Деревья *Burkhard-Keller* являются метрическими деревьями, алгоритмы построения таких деревьев основаны на свойстве метрики отвечать неравенству треугольника:

$$\rho(x,y) \leqslant \rho(x,z) + \rho(z,y), \quad x,y,z \in X.$$

Это свойство позволяет метрикам образовывать метрические пространства произвольной размерности. Такие метрические пространства не обязательно являются *евклидовыми*, так, например, метрики *Левенштейна* и *Дамерау-Левенштейна* образуют *неевклидовы* пространства. На основании этих свойств можно построить структуру данных, осуществляющую поиск в таком метрическом пространстве, которой и являются деревья Баркхарда-Келлера.



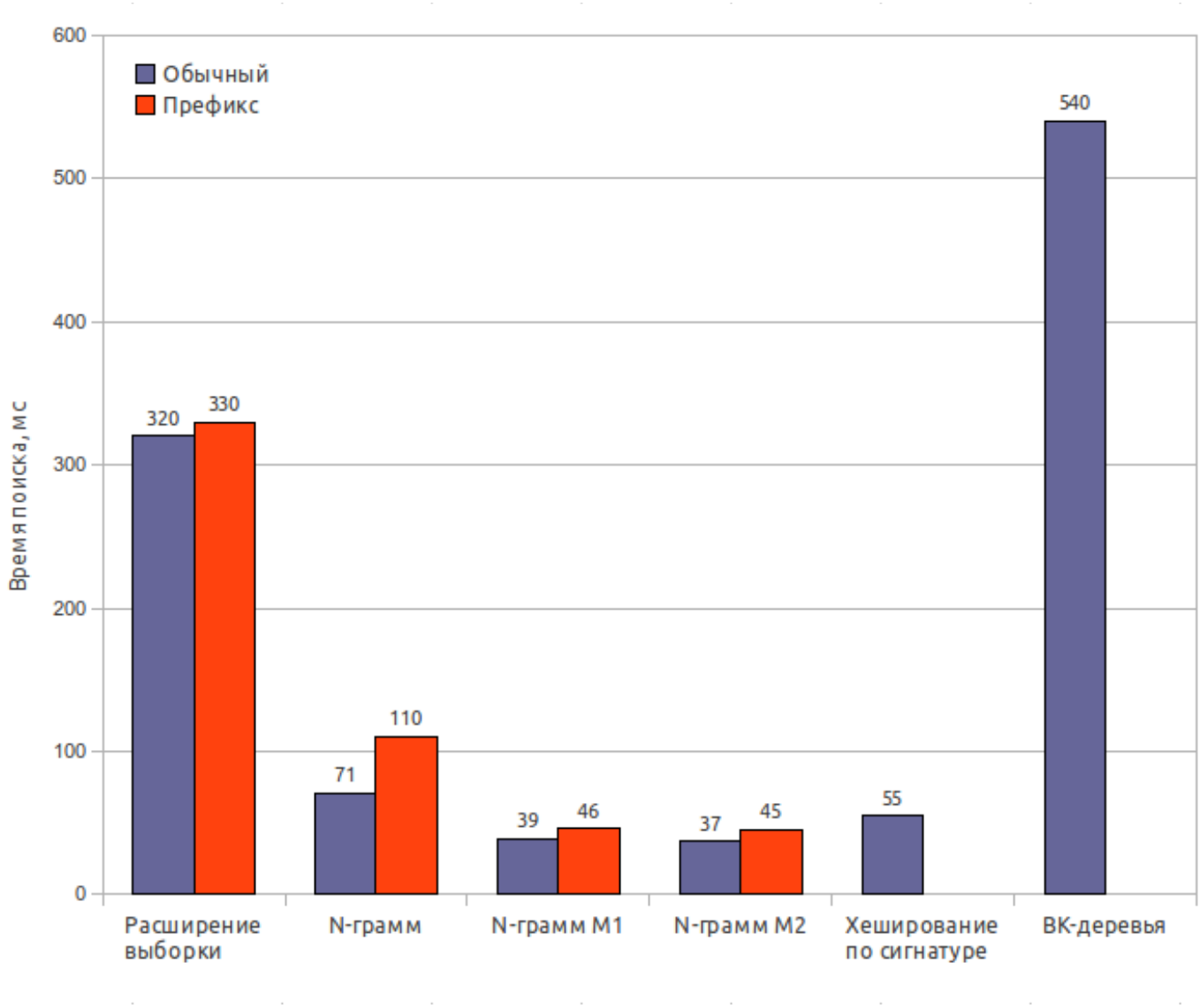
Улучшения:

Можно использовать возможность некоторых метрик вычислять расстояние с ограничением, устанавливая верхний предел, равный сумме максимального расстояния к потомкам вершины и результирующего расстояния, что позволит немного ускорить процесс:

$$P_{limit} \leqslant max_{children}\, d(this, child) + k$$

Тестирование

Тестирование осуществлялось на ноутбуке с Intel Core Duo T2500 (2GHz/667MHz FSB/2MB), 2Gb ОЗУ, ОС — Ubuntu 10.10 Desktop i686, JRE — OpenJDK 6 Update 20.



Тестирование осуществлялось с использованием расстояния Дамерау-Левенштейна и количеством ошибок $k = 2$. Размер индекса указан вместе со словарем (65 Мб).

Расширение выборки
Размер индекса: 65 Мб
Время поиска: 320 мс / 330 мс
Полнота результатов: 100%

N-грамм (оригинальный)
Размер индекса: 170 Мб
Время создания индекса: 32 с
Время поиска: 71 мс / 110 мс
Полнота результатов: 65%

N-грамм (модификация 1)
Размер индекса: 170 Мб
Время создания индекса: 32 с
Время поиска: 39 мс / 46 мс
Полнота результатов: 63%

N-грамм (модификация 2)
Размер индекса: 170 Мб
Время создания индекса: 32 с
Время поиска: 37 мс / 45 мс
Полнота результатов: 62%

Хеширование по сигнатуре
Размер индекса: 85 Мб
Время создания индекса: 0.6 с
Время поиска: 55 мс
Полнота результатов: 56.5%

BK-деревья
Размер индекса: 150 Мб
Время создания индекса: 120 с
Время поиска: 540 мс
Полнота результатов: 63%

Численная оценка информационного поиска
Источники: [8]

Accuracy

В простейшем случае такой метрикой может быть доля документов по которым классификатор принял правильное решение.

$$Accuracy = \frac{P}{N}$$

где, P – количество документов по которым классификатор принял правильное решение, а N – размер обучающей выборки. Очевидное решение, на котором для начала можно остановиться.
Тем не менее, у этой метрики есть одна особенность которую необходимо учитывать. Она присваивает всем документам одинаковый вес, что может быть не корректно в случае если распределение документов в обучающей выборке сильно смещено в сторону какого-то одного или нескольких классов. В этом случае у классификатора есть больше информации по этим классам и соответственно в рамках этих классов он будет принимать более адекватные решения. На практике это приводит к тому, что вы имеете accuracy, скажем, 80%, но при этом в рамках какого-то конкретного класса классификатор работает из рук вон плохо не определяя правильно даже треть документов.

Один выход из этой ситуации заключается в том чтобы обучать классификатор на специально подготовленном, сбалансированном корпусе документов. Минус этого решения в том что вы отбираете у классификатора информацию об остностельной частоте документов. Эта информация при прочих равных может оказаться очень кстати для принятия правильного решения.

Другой выход заключается в изменении подхода к формальной оценке качества.

Точность и полнота

Точность (precision) и полнота (recall) являются метриками которые используются при оценке большей части алгоритмов извлечения информации. Иногда они используются сами по себе, иногда в качестве базиса для производных метрик, таких как F-мера или R-Precision. Суть точности и полноты очень проста.

Точность системы в пределах класса – это доля документов действительно принадлежащих данному классу относительно всех документов которые система отнесла к этому классу. Полнота системы – это доля найденных классфикатором документов принадлежащих классу относительно всех документов этого класса в тестовой выборке.

Эти значения легко рассчитать на основании таблицы контингентности, которая составляется для каждого класса отдельно.

Категория i		Экспертная оценка	
		Положительная	Отрицательная
Оценка системы	Положительная	TP	FP
	Отрицательная	FN	TN

В таблице содержится информация сколько раз система приняла верное и сколько раз неверное решение по документам заданного класса. А именно:

- TP — истино-положительное решение;
- TN — истино-отрицательное решение;
- FP — ложно-положительное решение;
- FN — ложно-отрицательное решение.

Тогда, точность и полнота определяются следующим образом:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Confusion Matrix

На практике значения точности и полноты гораздо более удобней рассчитывать с использованием матрицы неточностей (confusion matrix). В случае если количество классов относительно невелико (не более 100-150 классов), этот подход позволяет довольно наглядно представить результаты работы классификатора.

Матрица неточностей – это матрица размера N на N, где N — это количество классов. Столбцы этой матрицы резервируются за экспертными решениями, а строки за решениями классификатора. Когда мы классифицируем документ из тестовой выборки мы инкрементируем число стоящее на пересечении строки класса который вернул классификатор и столбца класса к которому действительно относится документ.

	0.91	0.96	0.94	0.75	1.00	0.83	0.85	0.97	1.00	0.86	1.00	0.79	1.00	0.75	1.00	1.00	0.96	0.90	0.81	0.89	0.94	0.98	0.86	0.89	0.94	0.92	0.96
0.80		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
0.95	1	94	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
1.00	2	0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.29	3	0	0	6	0	0	3	2	0	1	0	0	0	0	0	0	1	1	0	0	1	0	1	3	0	2	0
1.00	4	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.50	5	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	1	1
0.92	6	1	0	0	0	0	152	0	0	1	0	0	0	0	0	0	0	1	4	2	3	0	0	0	0	2	0
0.97	7	1	0	1	0	0	0	256	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	2	0
0.33	8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
0.97	9	0	0	0	0	0	0	0	0	69	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
0.82	10	0	0	0	0	0	2	0	0	0	18	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0.87	11	0	0	0	0	0	0	0	0	0	0	34	0	4	0	0	0	0	0	0	0	0	0	1	0	0	0
1.00	12	0	0	0	0	0	0	0	0	0	0	0	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.57	13	0	0	0	0	0	0	0	0	0	0	9	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0
0.63	14	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	3	0	0	0	0	0	0	0	0	0
0.50	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	1	0	0	0	0	0	0
0.77	16	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	47	0	1	3	4	0	0	2	0	1	0
0.87	17	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	69	1	2	5	0	0	0	0	0	0
0.97	18	0	0	0	0	1	4	0	0	1	0	0	0	0	0	0	0	0	197	1	0	0	0	0	0	0	0
0.78	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	35	183	13	0	0	2	0	1	0
0.97	20	0	0	0	0	0	10	3	0	1	0	0	0	0	0	0	0	0	0	4	702	0	0	0	0	6	0
0.93	21	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56	0	2	0	0	0
0.29	22	0	0	1	0	0	2	0	0	6	0	0	0	0	0	0	0	0	1	1	1	0	6	2	0	1	0
0.91	23	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	3	6	0	0	115	0	0	0
1.00	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0
0.93	25	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	4	5	0	0	0	1	196	0
0.98	26	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	78

Матрица неточностей (26 классов, результирующая точность – 0.8, результирующая полнота – 0.91)

Как видно из примера, большинство документов классификатор определяет верно. Диагональные элементы матрицы явно выражены. Тем не менее в рамках некоторых классов (3, 5, 8, 22) классификатор показывает низкую точность.

Имея такую матрицу точность и полнота для каждого класса рассчитывается очень просто. **Точность** равняется отношению соответствующего диагонального элемента матрицы и суммы всей строки класса.

Полнота – отношению диагонального элемента матрицы и суммы всего столбца класса.

Формально:

$$Precision_c = \frac{A_{c,c}}{\sum_{i=1}^{n} A_{c,i}}$$

$$Recall_c = \frac{A_{c,c}}{\sum_{i=1}^{n} A_{i,c}}$$

Результирующая точность классификатора рассчитывается как арифметическое среднее его точности по всем классам. То же самое с полнотой. Технически этот подход называется macro-averaging.

F-мера

Понятно что чем выше точность и полнота, тем лучше. Но в реальной жизни максимальная точность и полнота не достижимы одновременно и приходится искать некий баланс. Поэтому, хотелось бы иметь некую метрику которая объединяла бы в себе информацию о точности и полноте нашего алгоритма. В этом случае нам будет проще принимать решение о том какую реализацию запускать в production (у кого больше тот и круче). Именно такой метрикой является F-мера.

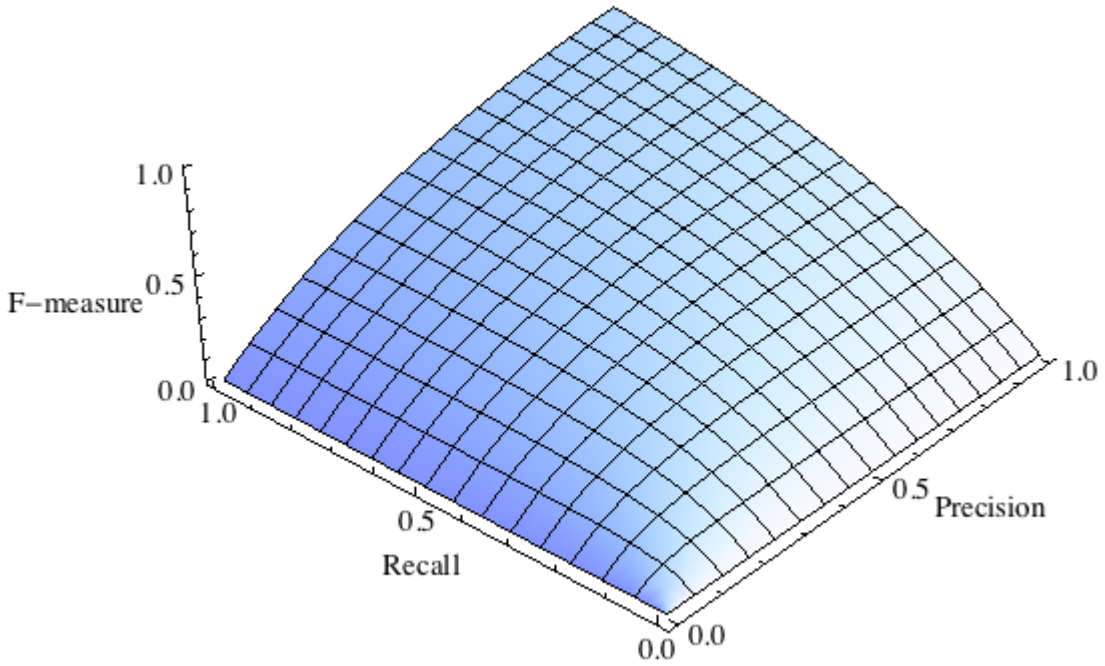
F-мера представляет собой гармоническое среднее между точностью и полнотой. Она стремится к нулю, если точность или полнота стремится к нулю.

$$F = 2\frac{Precision \times Recall}{Precision + Recall}$$

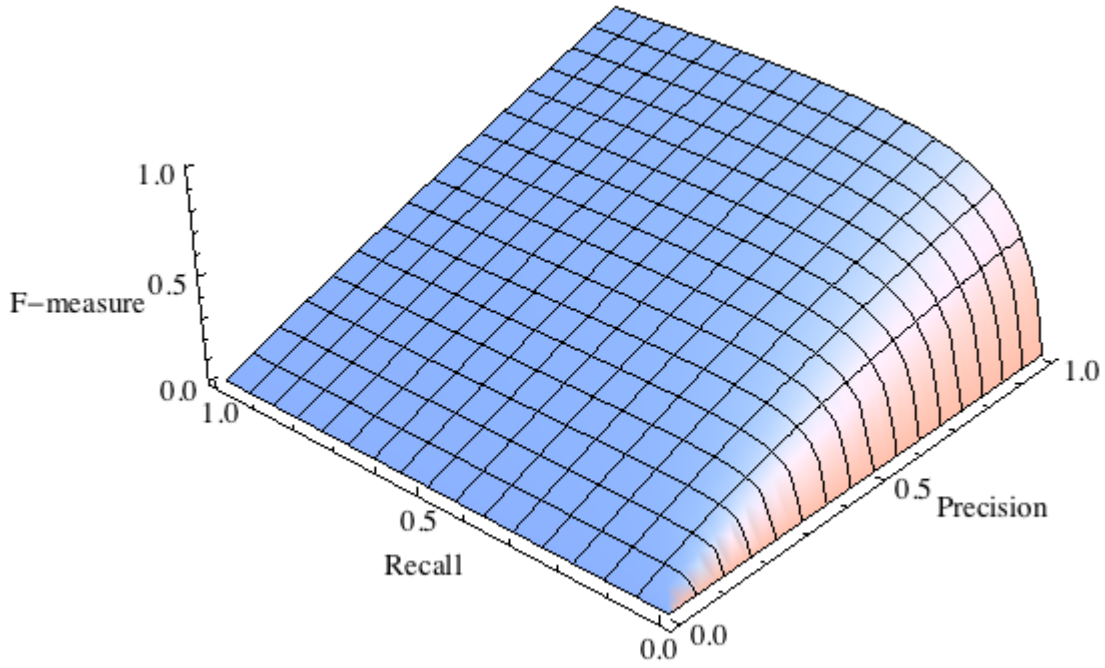
Данная формула придает одинаковый вес точности и полноте, поэтому F-мера будет падать одинаково при уменьшении и точности и полноты. Возможно рассчитать F-меру придав различный вес точности и полноте, если вы осознанно отдаете приоритет одной из этих метрик при разработке алгоритма.

$$F = \left(\beta^2 + 1\right)\frac{Precision \times Recall}{\beta^2 Precision + Recall}$$

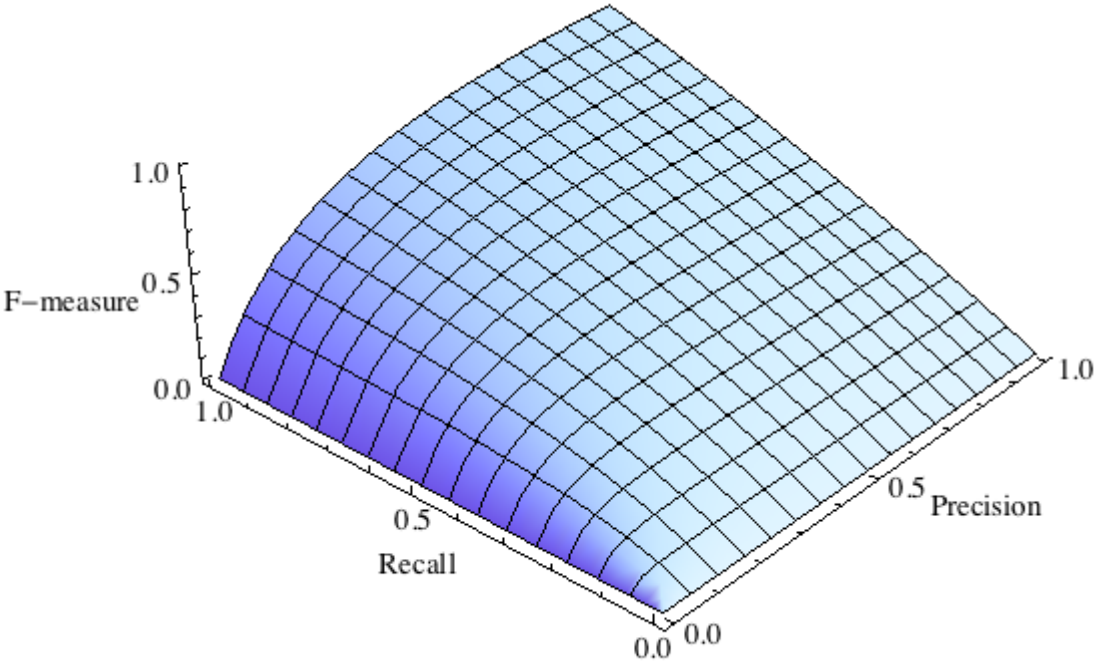
где β принимает значения в диапазоне 0<β<1 если вы хотите отдать приоритет точности, а при β>1 приоритет отдается полноте. При β=1 формула сводится к предыдущей и вы получаете сбалансированную F-меру (также ее называют F_1).



Сбалансированная F-мера



F-мера с приоритетом точности (β2=14)



F-мера с приоритетом полноты (β2=2)

F-мера является хорошим кандидатом на формальную метрику оценки качества классификатора. Она сводит к одному числу две других основополагающих метрики: точность и полноту.

Список источников

0) Классический информационный поиск: реализация и методы
http://www.machinelearning.ru/wiki/images/d/d2/IR.pdf

1) Расширенная модель булевского поиска
http://textualheritage.org/content/view/83/68/lang,russian/

2) "Интернетика" [30-39]
http://bourabai.ru/library/internetica.pdf

3) Модели информационного поиска
http://yury.name/internet/03ianote.pdf

4) Векторная модель информационного поиска
http://goo.gl/pqVwoU

5) Ранжирование в информационно-поисковых системах на основе социальных сервисов
http://seminar.at.ispras.ru/wp-content/uploads/2011/05/Kiyko_thesis.pdf

6) Вероятностные модели в задачах машинного обучения
http://www.machinelearning.ru/wiki/images/6/6f/BMMO11_1.pdf

7) Нечеткий поиск в тексте и словаре
http://habrahabr.ru/post/114997/

8) Оценка классификатора (точность, полнота, F - мера)
http://bazhenov.me/blog/2012/07/21/classification-performance-evaluation.html

Другие:
***)Лекции от Техносферы**
http://habrahabr.ru/company/mailru/blog/257119/
***)Лингвистика в поисковых системах**
http://habrahabr.ru/company/yandex/blog/224579/
***)Лекции из ШАД**
https://yandexdataschool.ru/edu-process/courses
***)Технологии Яндекса**
https://tech.yandex.ru/