

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ МОСКОВСКИЙ  
ГОСУДАРСТВЕННЫЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ (ФГБОУ ВПО «МГИУ»)**

**КАФЕДРА «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ»**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

по специальности *«Математическое обеспечение и  
администрирование информационных систем»*  
студента *Васина Александра Дмитриевича*

на тему ***«Разработка системы анализа текста  
на наличие заимствований»***

Руководитель работы: *доцент  
Бургонский Дмитрий Сергеевич*

Студент-дипломник	_____	А. Д. Васин
Руководитель работы, доцент	_____	Д. С. Бургонский

**ДОПУСКАЕТСЯ К ЗАЩИТЕ**

Зав. Кафедрой 36, к. ф.-м. н., доцент	_____	Е. А. Роганов
---------------------------------------	-------	---------------

МОСКВА 2011

## **Аннотация**

В данной работе разработана система, позволяющая различными алгоритмами анализировать текст и выявлять в нем заимствования. В работе 64 страницы, 5 рисунков, 7 таблицы, 14 элементов в списке литературы.

Ключевые слова: шингл, мегашингл, супершингл, MinHash, I-Match, LongSet, Ruby on Rails, TF, TF-IDF.

# Содержание

1. Введение.....	4
2. Обзор методов нахождения нечетких дубликатов.....	6
2.1    Локальные методы.....	7
2.1.1    LongSent метод.....	7
2.1.2    Методы на основе меры TF.....	7
2.1.3    Методы, использующие понятия шинглов.....	7
2.1.4    MinHash метод.....	9
2.1.5    Методы, использующие семантические сети.....	10
2.2    Глобальные методы.....	11
2.2.1    Методы на основе меры TF-IDF.....	11
2.2.2    I-Match метод.....	12
2.2.3    Метод «опорных» слов.....	13
2.3    Выводы.....	16
3. Постановка задачи.....	17
4. Сравнение методов нахождения нечетких дубликатов.....	18
5. Проектирование собственной системы нахождения нечетких дубликатов....	20
5.1    Обоснование выбора средств разработки.....	20
5.2    Структура базы данных.....	22
5.3    Реализация используемых алгоритмов.....	24
5.4    Интерфейс взаимодействия с пользователем.....	31
6. Анализ полученных результатов.....	34
7. Выводы.....	40
8. Приложение.....	41
9. Литература.....	64

## 1. Введение

Человечество из поколения в поколение часто сталкивается и обсуждает одни и те же проблемы и жизненные ситуации. Людям свойственно выражать схожие мысли и идеи разными способами, но суть при этом является одинаковой. Подобная ситуация может стать серьезной проблемой во многих сферах. Именно поэтому такое положение вещей делает решение данной задачи актуальным.

Наиболее широко с этой проблемой сталкиваются поисковые системы, поскольку с развитием интернет-технологий количество информации, передающей одинаковый смысл, колоссально увеличилось. В повседневной жизни мы наблюдаем подобную ситуацию при поиске новостей или другого рода информации в сети. Один за другим нам попадаются сайты-клоны, которые дублируют информацию предыдущего источника с незначительно измененным контентом. Информация на подобных ресурсах является вторичной. Это вынуждает поисковые системы выявлять такие нечеткие дубликаты и скрывать их от конечного пользователя.

В дальнейшем под понятием «нечеткого дубликата» будем понимать документ, контент которого частично или полностью заимствован из других документов.

С данной проблематикой мы сталкиваемся и в других отраслях. Часто такая ситуация складывается в научно-образовательной сфере. Большинству педагогов ежедневно приходится проверять ученические работы, которые были взяты недобросовестными студентами из всемирной паутины — интернет. Лень, нехватка знаний и отсутствие уверенности в собственные силы толкают учащегося обращаться к сети интернет в поисках уже готовых решений: написанных рефератов, курсовых и дипломных работ. Студенты отмечают простоту и доступность текстов, находящихся в сети, а их широкий ассортимент обеспечивает наличие необходимой информации на любые тематики. Скопировав материал из

интернета, ученик может пойти на некоторые хитрости. Текст работ при этом, возможно, будет отредактирован и обработан, или произойдет замена абзацев местами, однако подобные модификации и адаптации не меняют смысловую нагрузку работы. Зачастую выявить некачественную работу не способен даже самый опытный педагог. Для повышения качества образования такие работы должны отвергаться преподавателями. Однако, как определить что отвергать, а что оставить? Именно такой вопрос подтолкнул специалистов к разработке методов, которые способствовали бы решению поставленной задачи.

На поиски оптимального решения проблемы нахождения нечетких дубликатов было потрачено немало времени и сил, при этом использовались различные технологии. Ключевым аспектом в решении данного конфликта являлась необходимость обработки гигантских объемов данных. Именно поэтому сравнение всех термов одного документа с терминами другого не может стать приемлемым методом решения проблемы.

В последнее время большое внимание уделяется алгоритмам, которые позволяют снизить требуемые вычислительные ресурсы за счет использования различных эвристик. Результатом таких работ являются алгоритмы, позволяющие определить степень идентичности документов.

## 2. Обзор методов нахождения нечетких дубликатов

Если говорить о методах выявления нечетких дубликатов, то все их можно разделить на два больших класса. Алгоритмы, которые используют определенные знания всей рассматриваемой коллекции документов, мы будем называть глобальными, в противном случае – локальными.

Под «коллекцией документов» мы будем подразумевать набор текстовых файлов. Для того, чтобы оценивать эффективность методов, введем два параметра: точность и полнота.

Точность – способность системы выдавать в списке результатов только документы, действительно являющиеся дубликатами. Вычисляется по формуле:

$$precision = \frac{a}{a+b}$$

где  $a$  – количество найденных пар дубликатов, совпадающих с релевантными парами;  $b$  – количество найденных пар дубликатов, не совпадающих с релевантными парами.

Полнота – способность системы находить дубликаты, но не учитывать количество ошибочно определенных недубликатов.

$$recall = \frac{a}{a+c}$$

где  $a$  – количество найденных пар дубликатов, совпадающих с релевантными парами;  $c$  – количество не найденных пар дубликатов, совпадающих с релевантными парами.

## **2.1 Локальные методы**

Рассмотрим, для начала, локальные алгоритмы. Основная идея таких методов сводится к синтаксическому анализу документа. На основе этого анализа документу ставится в соответствие определенное количество сигнатур.

### **2.1.1 LongSent**

Простейшим примером может служить алгоритм, который вычисляет хеш-функцию (MD5, SHA-2, CRC32) от конкатенации двух самых длинных предложений в документе. Это и будет являться его сигнатурой. Точность такого алгоритма достаточно большая, но он обладает существенным изъяном в безопасности. Такой алгоритм легко обмануть. Достаточно откорректировать всего лишь два самых длинных предложения.

### **2.1.2 Методы на основе меры TF**

Более эффективным способом нахождения нечетких дубликатов может стать метод, основанный на понятие TF (term frequency — частота слова).

TF - отношение числа вхождения некоторого слова к общему количеству слов документа.

Таким образом оценивается важность слова в пределах отдельного документа. Для каждого слова в документе вычисляется его вес, равный отношению числа вхождения этого слова к общему количеству слов документа. Далее сцепляются  $n$  упорядоченных слов с наибольшим значением веса и вычисляется хеш-функция. Такой подход позволяет улучшить ситуацию, но для решения реальных задач этот способ не подходит.

### **2.1.3 Методы, использующие понятия шинглов**

Один из первых методов, который был применен на практике (компанией

AltaVista), основываясь на понятие шинглов. Данный подход был предложен А. Broder [3].

Документ представлялся в виде последовательности перекрывающихся подстрок определенной длины. Визуально это изображено на Рис. 1:



**Рис. 1**

Такие подстроки были названы шинглами. В качестве меры идентичности двух документов использовался тот факт, что схожие документы имеют существенное количество одинаковых шинглов.

Чтобы достичь более эффективных показателей, при сравнении таких множеств для каждого шингла вычислялась хеш-функция. Поскольку число шинглов в документе приблизительно равно длине самого документа, автором было предложено несколько модификаций алгоритмов, позволяющих уменьшить вычислительную трудоемкость задачи.

Изначально был использован метод, выбирающий только те шинглы, чьи хеш-значения делились без остатка на некоторое число  $m$ . В свою очередь, для документов малой длины такой подход не годился, так как выборка могла оказаться очень маленькой или вообще пустой. Вторым способом определял ограниченное число шинглов  $s$  с наименьшими значениями хеш-функций или оставлял все, если их число не превышало  $s$ .

Некоторое время спустя идею этого метода стали развивать D. Fetterly [8] и основоположник А. Broder [4], которым удалось значительно модифицировать и усовершенствовать ее. В своих трудах они предложили с помощью взаимно-однозначных и независимых функций (min-wise independent) вычислять 84 хеш-значения для каждого шингла. В итоге, документ был представлен в виде 84



шинглов, которые имели минимальное хеш-значение из вычисленных 84 min-wise independent функций. Все 84 шингла делились на 6 групп, каждая из которой состояла из 14 независимых шинглов. Эти группы впоследствии получили название супершинглы. На практике данный метод продемонстрировал, что для определения нечетких дубликатов хватало совпадений минимум двух супершинглов. Это означало, что для наилучшего поиска совпадений как минимум двух супершинглов документ был представлен различными попарными соединениями из 6 супершинглов. Такие группы стали называться «мегашинглами». Общее количество мегашинглов составляет 15. Согласно данному методу выявления почти дубликатов документы являются между собой идентичными, если у них совпадает минимум один мегашингл.

#### 2.1.4 MinHash метод

MinHash является разновидностью Locality-sensitive hashing (LSH) алгоритма. Его основная идея заключается в определении семейства хеш-функций  $H$ . Семейство таких хеш-функций образует систему LSH для функции близости  $\text{sim}(x, y)$ , если:

$$\text{sim}(x, y) = P_{h \in H} [h(x) = h(y)]$$

Под «функцией близости» здесь понимаем некую функцию, ставящую в соответствие двум объектам число от 0 до 1, где 0 означает, что объекты абсолютно не похожи, а 1 — они полностью совпадают.

Не все функции близости соответствуют вышеприведенному свойству. Однако часть весьма полезных и применимых на практике это выполняют. Одной из такой функций является коэффициент Жаккарда:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

В качестве множеств  $A$  и  $B$  могут служить те же шинглы или какой-либо

другой заранее определенный терм (нормализованное слово).

С LSH и семейством хеш-функций тесно связано такое понятие, как min-wise independent permutations. Пусть есть некоторое множество простых чисел от 1 до N

$$E = \{1 \dots N\}$$

Будем рассматривать перестановки  $\Pi: E \rightarrow E$ . Подмножество  $K$  таких перестановок будем называть min-wise independent permutations, если выполняется условие:

$$\forall X \in E \quad P_{\substack{\Pi \in K \\ y \in K(X)}} [\min \Pi(X) = y] = \frac{1}{|X|}$$

Очевидно, что всевозможные перестановки  $\Pi$  задают min-wise independent permutations, но нас они не интересуют, так как их количество слишком велико — для этого нам потребуются дополнительные вычисления. Поэтому мы будем рассматривать именно небольшое подмножество  $K$  таких перестановок, удовлетворяющих условию.

Такие min-wise independent permutations функции задают функцию близости для системы LSH. Следующие математические выкладки доказывают это:

$$\begin{aligned} P_{\Pi} [\min \Pi(X) = \min \Pi(Y)] &= P_{\Pi} [\min \Pi(X \cup Y) \in \Pi(X) \cap \Pi(Y)] = \\ P_{\Pi} [\min \Pi(X \cup Y) \in \Pi(X \cap Y)] &= \sum_{a \in \Pi(X \cap Y)} P_{\Pi} [\min \Pi(X \cup Y) = a] = \frac{|A \cap B|}{|A \cup B|} \end{aligned}$$

Идея алгоритма MinHash заключается в выборе минимального значения, вычисленного для каждого элемента множества значений всех min-wise independent permutations функций. Для определения, того является ли один документ дубликатом или нечетким дубликатом другого документа, потребуется ограниченное число операций сравнения вне зависимости от размеров самих документов.

## 2.1.5 Методы, использующие семантические сети

Также интересным подходом является использование семантической сети.

Задача определения факта заимствования сводится к сравнению моделей, отражающих смысловую нагрузку текстов. Анализ ведется с использованием алгоритмов на графах, модифицированных и оптимизированных для применения в рамках данной задачи.

Использование схем анализа данных в этом методе может позволить выявлять факт заимствования, даже если оригинал был определенным образом модифицирован (выполнен перевод, слова были заменены на синонимы, текст был изложен с использованием другой лексики и т.д.).

## **2.2 Глобальные методы**

### **2.2.1 Методы на основе меры TF-IDF**

Дальнейшее развитие метода, использующего меру TF, стал алгоритм, анализирующий документы всей коллекции. В нем используются мера TF-IDF.

IDF (inverse document frequency — обратная частота документа) — инверсия частоты, с которой некоторое слово встречается в документах коллекции.

$$IDF = \log\left(\frac{|D|}{|d_i \ni t_i|}\right)$$

где  $|D|$  — количество документов в коллекции;  $|d_i \ni t_i|$  — количество документов, в которых встречается терм  $t_i$ .

Вес широкоупотребительных слов можно сократить при учете IDF, то есть мера TF-IDF состоит из произведения TF и IDF. В мере TF-IDF больший вес будет у тех слов, которые часто использовались в одном документе и реже — в другом.

Соответственно, при вычислении веса для каждого терма, алгоритм использует формулу  $TF * IDF$ . После этого в строку в алфавитном порядке сортируются 6 слов, которые имеют наибольшее значение веса. Контрольная сумма CRC32 полученной строки вычисляется в качестве сигнатуры документа.

Существуют различные модификации формулы вычисления веса слова. В поисковых системах широко известно семейство функций BM25. Одна из распространенных форм этой функции приведена ниже.

$$score(D, Q) = \sum_{i=1}^n IDF \frac{(q_i) \cdot f(q_i, D) \cdot (k_1 - 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \frac{|D|}{avgdl})}$$

где  $f(q_i, D)$  — это частота слова  $q_i$  в документе  $D$ ,  $|D|$  — это длина документа (количество слов в нём), а  $avgdl$  — средняя длина документа в коллекции.  $k_1$  и  $b$  — свободные коэффициенты, обычно их выбирают как  $k_1 = 2.0$  и  $b = 0.75$ .  $IDF(q_i)$  — обратная документная частота слова  $q_i$ .

### 2.2.2 I-Match метод

Еще один сигнатурный метод в 2002 году предложил А. Chowdhury [5]. Идея подхода тоже заключалась на знании всей коллекции документов. Предложенную методику автор усовершенствовал в 2004 году [6, 7].

Ключевая идея данного метода основывалась на вычислении дактилограммы I-Match для демонстрации содержания документов.

Изначально для исходной коллекции документов строился словарь  $L$ , который включает слова со средними значениями IDF. Именно они позволяли добиться наиболее точных показателей при выявлении нечетких дубликатов. Отбрасывались при этом те слова, которые имели большие и маленькие значения IDF.

После этого для каждого документа создавалось множество  $U$  различных слов, состоящих в нем, и высчитывалось пересечение  $U$  и словаря  $L$ . Экспериментальным методом вычислялся минимальный порог и если размер пересечения превышал его, то список входящих в пересечение слов упорядочивался. Далее нужно посчитать I-Match сигнатуру (хеш-функция SHA1).

Следовательно, здесь мы видим следующую ситуацию: два документа будут

считаться одинаковыми, если у них совпадут I-Match сигнатуры. В отличие от алгоритма, предложенного А. Broder., данный метод имеет больший потенциал. Он демонстрирует значительно улучшенную вычислительную способность. Опять же, если сравнивать с алгоритмом А. Broder, еще одним достоинством в пользу этого алгоритма становится то, что он значительно эффективнее проявляет себя при сравнении небольших по объему документов.

К сожалению, у данного алгоритма есть и свой недостаток — при небольшом изменении содержания он показывает свою неустойчивость.

Чтобы исключить данный недостаток, авторы решили подвергнуть алгоритм изменению и усовершенствовать его. Была предложена новая техника многократного случайного перемешивания основного словаря.

Суть модификаций заключается в следующем: к основному словарю  $L$  создаются  $K$  различных словарей  $L_1-L_K$ , которые образуются методом случайного удаления из исходного словаря определенной закреплённой части  $p$  слов. Эта небольшая группа  $p$  слов составляет приблизительно 30%-35% от исходного объема  $L$ .

Для каждого документа вместо одной, вычисляется  $(K+1)$  I-Match сигнатур по алгоритму, который описан выше. Получается, что документ демонстрируется как вектор размерности  $(K+1)$ . В таком случае два документа между собой будут считаться одинаковыми, если одна из координат у них совпадает. На практике, в качестве самых оптимальных значений параметров хорошо зарекомендовали себя такие показатели:  $p = 0.33$  и  $K = 10$ .

Данный алгоритм продемонстрировал эффективную способность фильтрации спама при использовании в приложениях веб-поиска.

### **2.2.3 Метод «опорных» слов**

Существует еще один способ выявления почти дубликатов, основанный на сигнатурном подходе. Данный метод тоже заключается в использовании

лексических принципов, то есть на основе словаря. Метод был предложен С. Ильинским и др. [9] и получил название — метод «опорных» слов. Рассмотрим более детально принцип данного алгоритма.

Изначально из индекса по правилу, описанному ниже, мы выбираем множество из  $N$  слов, называемых «опорными». В данном случае,  $N$  определяется экспериментально. В дальнейшем, каждый документ выглядит  $N$ -мерным двоичным вектором, в котором  $i$ -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 и является сигнатурой документа. Соответственно, два документа считаются идентичными при совпадении сигнатур.

При построении множества «опорных» слов используются следующие соображения:

1. Множество слов должно охватывать максимально возможное число документов.
2. Число слов в наборе должно быть минимальным.
3. При этом «качество» слова должно быть максимально высоким.

Рассмотрим принцип построения множества алгоритма и выбор пороговых частиц. Допустим «частота» — это нормированная внутридокументная частота слова в документе  $TF$ , которая находится в диапазоне  $0...1$ , единица в данном случае будет соответствовать наиболее частому слову в документе  $TF$ . Распределение документов по данной внутридокументной «частоте» строится для каждого слова однократно.

Рассмотрим несколько этапов, каждый из которых состоит из двух фаз. В первой фазе увеличивается покрытие документов в индексе при фиксированной (ограниченной снизу) точности. Во второй фазе уже увеличивается точность при

фиксированном покрытии.

В данном случае точность будет максимально высокой, если повторение слова в дельта-окрестности значения относительной частоты минимально.

Частота, которая имеет наибольшую точность, получила название пороговой.

Поэтапно сортируются самые неподходящие слова, а когда наступает последняя стадия, то остаются только группы слов, которых достаточно для обеспечения качественного покрытия. Получается, что благодаря этому алгоритму, можно отфильтровать несколько тысяч слов и оставить только 3-5 тысяч.

## **2.3 Выводы**

Каждый из рассмотренных выше алгоритмов имеет свои достоинства и недостатки. Для определенных задач оптимальными могут оказаться алгоритмы на основе шинглов, так как позволяют показывать, какие именно фрагменты были заимствованы. Для других задач приоритетнее могут оказаться скорость работы и потребляемые ресурсы. Для таких задач, скорее всего, больше подойдут лингвистические алгоритмы.

Выбор алгоритма зависит от представленных требований к конкретной задаче. Следующие две главы будут посвящены постановке задачи и анализу эффективности методов нахождения нечетких дубликатов.



### **3. Постановка задачи**

В связи с тем, что количество недобросовестных студентов, сдающих работы, контент которых полностью или большей частью был заимствован из других источников, увеличивается, институтом была поставлена задача найти решение, позволяющее предупреждать такие попытки сдачи работ.

На российском рынке существует программное обеспечение, решающее такую проблему. Все оно, к сожалению, являются коммерческими. Поэтому было принято решение разработать собственную систему, главными требованиями которой являются точность и наглядность полученных при проверке документа результатов. Также система должна решать поставленную задачу за адекватные временные рамки. Под наглядностью понимается интерфейс, позволяющий показывать, откуда были получены заимствованные части и чему равно значение уникальности проверяемого документа. Дополнительным требованием является возможность проверки документа на наличие заимствований в Интернете.

#### 4. Сравнение методов нахождения нечетких дубликатов

За основу оценки эффективности рассмотренных алгоритмов возьмем результат эксперимента, проведенный в статье [2]. В ней в качестве исследуемой коллекции документов была взята база веб-сайтов РОМИН объемом порядка 500 000 документов. Перед прогоном коллекции через каждый алгоритм, все документы подвергались препроцессингу — очистке от html тегов, удалению стоп-слов, знаков пунктуации и прочей информации, которая не несла информационный характер.

В таблице 1 приведены результаты эксперимента из работы:

**Таблица 1**

<b>Метод</b>	<b>Полнота</b>	<b>Точность</b>
TF	0.6	0.94
TF-IDF	0.54	0.96
Мегашинглы	0.36	0.91
I-Match	0.5	0.97
Метод «опорных» слов	0.44	0.77
LongSent	0.84	0.8

В работе [1], использующей семантическую сеть для выявления дубликатов, к сожалению, не приводится численных результатов, но удастся понять, что скорость работы оставляет желать лучшего. Это связано с примененным там алгоритма нахождения изоморфности графов, который является NP-полной задачей.

Единственный из рассмотренных алгоритмов, который способен показать какие именно части контента документа были заимствованы и откуда, является первый алгоритм на основе простых шинглов.

Исходя из поставленных критериев и приведенных результатов эксперимента, нами были выбраны следующие алгоритмы: алгоритмы на основе

шинглов, I-Match, LongSent и MinHash. Каждый из алгоритмов обладает определенными свойствами. MinHash, алгоритмы на основе супершинглов и мегашинглов позволяют ограничить количество сигнатур для документа произвольной длины, что значительно сказывается в лучшую сторону на производительности. LongSent является простым в реализации алгоритмом, но несмотря на это, он показал хорошие результаты в эксперименте. I-Match — очень интересный и перспективный метод. У него имеется несколько модификаций, которые позволяют улучшить результат. Метод на основе простых шинглов — единственный метод, который позволяет показать заимствованные фрагменты проверяемого документа.

На основе этих алгоритмов планируется разработать систему, которая способна удовлетворить требованиям поставленной задачи.

## 5. Проектирование собственной системы нахождения нечетких дубликатов

### 5.1 Обоснование выбора средств разработки

В связи с растущей популярностью различных веб-приложений было принято разработать систему нахождения нечетких дубликатов, используя в качестве основной среды разработки язык ruby и хорошо себя зарекомендовавший фреймворк Ruby on Rails. Веб-приложения позволяют почти полностью убрать ограничения на платформы, где будет использоваться разработанная система.

Основные преимущества Ruby и Ruby on Rails:

- **Скорость разработки:**

Главным достоинством языка программирования Ruby и фреймворка Ruby on Rails является скорость разработки. Если сравнивать с другими языками программирования или фреймворками, то скорость разработки проектов на RoR существенно увеличивается.

- **Культура и стандарты:**

Ruby on Rails — это фреймворк. Принято считать, что фреймворк не дает возможности сгенерировать что-то свое. Хочется отметить, что используя Ruby on Rails, можно отойти от стандарта и действовать в своем ключе, но, как показывает практика, в этом нет необходимости. Любая задача в проекте будет значительно структурирована, если действовать согласно стандартам, предложенным фреймворком Ruby on Rails. Благодаря использованию правил размещения файлов или написания кодов, проект становится более читаемым.

- **Тестирование:**

Опять же, если сравнивать с другими языками программирования и фреймворками, то хочется отметить, что Ruby on Rails характеризуется наличием качественных средств автоматизированного тестирования,

которых нет у других. Это означает, что используя RoR, код проекта пишется только тогда, когда под него созданы тесты, но это в идеале. В принципе, изначально в основе концепции Ruby on Rails лежит использование методов BDD (Behavior Driven Development) или TDD (Test Driven Development).

- **Кеширование:**

Ruby on Rails в его базовой комплектации имеет штатные средства кеширования данных. В начале работы с RoR нам предлагается стандартный набор инструментов для кеширования данных на проекте. Интересной особенностью является то, что в подавляющем большинстве случаев для кеширования на Ruby on Rails, нет необходимости в поиске дополнительных решений, а достаточно воспользоваться стандартными готовыми решениями. Кешировать можно блоки кода или целые страницы. При кешировании можно воспользоваться memcached или же другими средствами.

- **Валидации:**

Хочется обратить внимание, что в ruby on rails качественно созданы инструменты, позволяющие валидировать входящие данные.

- **Миграции и работа с базой данных:**

В наборе стандартных инструментов Ruby on Rails есть так называемые «миграции», которые работают с базами данных. Сама структура базы данных конфигурируется из проекта и хранится в коде приложения.

- **Безопасность:**

Фреймворк Ruby on Rails изначально настроен на максимальную степень защиты проекта. Если использовать инструменты RoR, то можно не бояться SQL инъекций и XSS атак. Автоматически все входные параметры

экранируются, как и выводимые переменные в шаблонах, однако это в том случае, если вы не отменяли опции.

## 5.2 Структура базы данных

Результаты работы выбранных нами алгоритмов мы будем хранить в базе данных (БД) PostgreSQL. Выбор именно этой БД обусловлен ее широким применением в институте. PostgreSQL зарекомендовала себя как быстрая и надежная БД.

Структура БД для всего проекта приведена на Рис. 2:

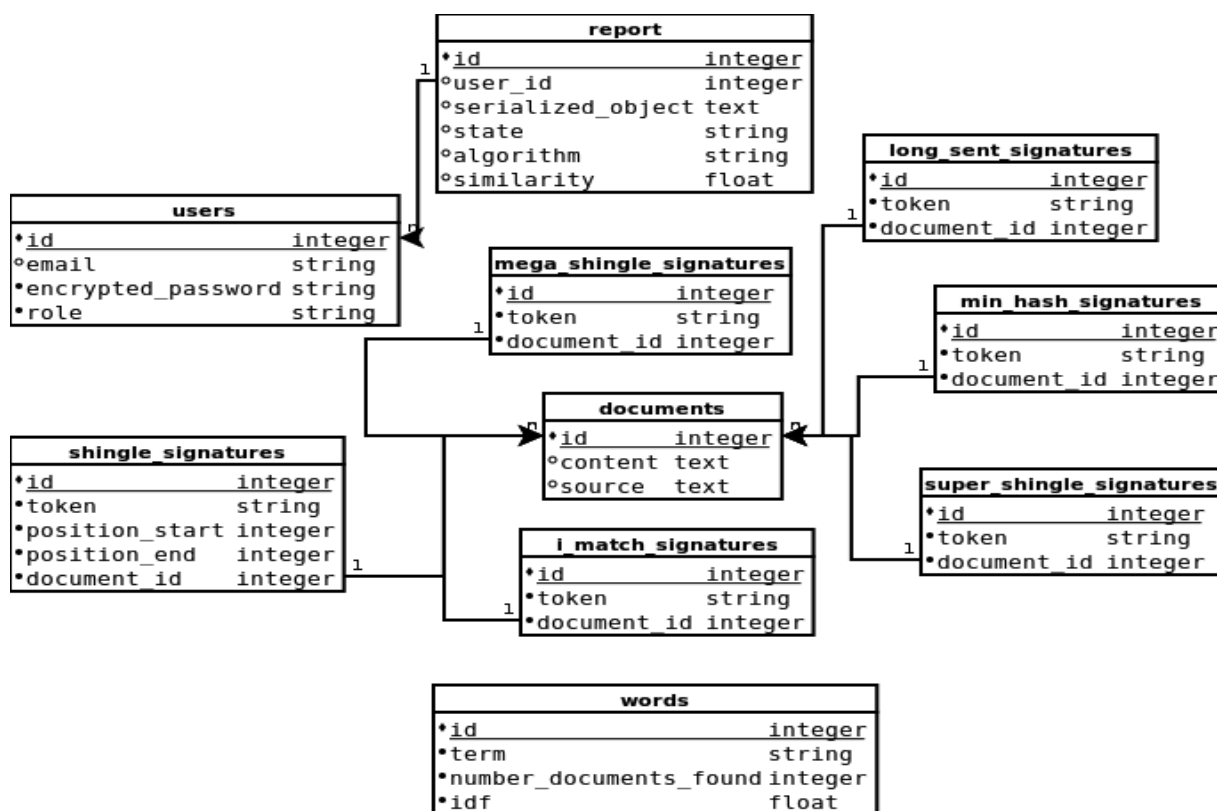


Рис. 2

При проверке очередного документа будет производиться поиск по полю token из таблиц i\_match\_signatures, mega\_shingle\_signatures, min\_hash\_signatures, shingle\_signatures, super\_shingle\_signatures. Чтобы эти запросы обрабатывались как можно быстрее, по этому полю в БД был построен уникальный индекс. Также

поле `idf` из таблицы `words` было проиндексировано, чтобы быстро получать словарь для алгоритма I-Match.

## 5.3 Реализация используемых алгоритмов

### Методы на основе шинглов

Одной из важных составляющих алгоритма на основе шинглов является канонизация документа и его разбиение на шинглы (шинглирование). Под канонизацией документа мы будем понимать процесс очистки текста документа от служебных знаков, которые не несут значимой информации. От правильности реализации этой части зачастую зависит точность и скорость работы алгоритма, так как этот процесс влияет на качество и количество шинглов в документе.

Для этой цели был разработан класс `Shingling`. Главным его методом является `each_shingles`. С помощью него происходит обработка текста и получение ШИНГЛОВ.

```
def each_shingles
  word = ""
  char_flag = false
  shingle = []
  position_end_words = []
  position_start, position_end = 0, 0

  @content.each_char do |char|
    if char !~ /[А-ЯЁа-яё]/
      char_flag = true
    else
      if char_flag
        if !stop_word?(word) && word =~ /\S+/
          shingle << (@downcase ? Unicode::downcase(word) : word)
          word = ""
          position_end_words << position_end
          if shingle.size == @shingle_length
            yield(shingle.join(" "), position_start, position_end)
            position_start = position_end_words.shift
            shingle.shift
          end
        end
      end
    end
  end
end
```



```

        end
      end
      char_flag = false
    end
    word << char
  end
  position_end += 1
end
end
end

```

Недобросовестные студенты могут пойти на различные ухищрения, чтобы обмануть систему, поэтому в алгоритме мы постарались учесть все такие возможности и пресечь их. Первым делом, мы делаем замену определенных символов. Такими символами могут служить английская буква 'a'. Поскольку в тексте ее визуалью не отличить от русской 'а', документ будет казаться полностью нормальным, но эти буквы представляются в компьютере абсолютно разным набором бит. В результате этого, при дальнейшем хешировании, мы получим разные значения для визуалью одинаковых строк, что является некорректным.

Так как знаки препинания, пробельные символы, стоп-слова, а так же специальные символы несут малую смысловую нагрузку, мы будем их игнорировать. Набор стоп-слов взят из общедоступного источника [11]. Слова из этого набора также отбрасываются из анализируемого текста.

В результате работы алгоритма мы получаем шингл определенной длины и два значения: позиция первого и последнего символа относительно шинглируемого текста (это нам понадобится в дальнейшем).

Первый алгоритм, который мы рассмотрим, получает информацию о каждом шингле и вычисляет по ней MD5-хеш.

```

def build_shingle_signatures
  shingling = Shingling.new(
    content,
    :stop_words => Text::STOP_WORDS,
  )
end

```

```

      :shingle_length => ShingleSignature::SHNINGLE_LENGTH,
      :downcase => true
    )

    shingling.each_shingles do |shingle, position_start, position_end|
      shingle_signatures.new(
        :token => Digest::MD5.hexdigest(shingle),
        :position_start => position_start,
        :position_end => position_end
      )
    end
    self.flag_build_shingle_signatures = true
  end
end

```

Для вычисления супершинглов нам понадобится семейство хеш-функций (min-wise permutatations), которые подробно описаны в главе 2.1.4. В качестве таких функций было взято аффинное преобразование вида:

$$h(x) = a \cdot x + b \bmod p$$

где  $a$ ,  $b$  - произвольные случайные, но заранее вычисленные числа, а  $p$  — большое простое число.

Этот метод был написан исходя из соображений, которые представлены в главе 2.1.4. Он возвращает массив фиксированной длины, равный количеству выбранных min-wise independent функций. Массив содержит минимальные значения, полученные при применении этих функций к MD5 хеш-значению шингла.

```

def self.find_min_numbers
  raise ArgumentError unless numbers.is_a?(Array)
  mins = []
  MIN_WISE_FUNCTIONS.each do |min_wise_function|
    min = min_wise_function.call numbers.first.hex if numbers.first
    numbers.each do |number|
      current = min_wise_function.call number.hex
    end
  end
end

```

```

        min = current if min > current
      end
      mins << min
    end

    return mins
  end

```

Далее, полученный массив разбивается на группы и вычисляется по ним MD5-хеш.

Для вычисления мегашинглов нам понадобился метод, генерирующий всевозможные попарные сочетания из вычисленных хешей супершинглов:

```

def generate_combinations_for_mega_shingle
  array = super_shingle_signatures.map(&:token)
  r = 2
  n = array.length
  indices = (0...r).to_a
  final = (n - r...n).to_a
  while indices != final
    yield indices.map {|k| array[k]}
    i = r - 1
    while indices[i] == n - r + i
      i -= 1
    end
    indices[i] += 1
    (i + 1...r).each do |j|
      indices[j] = indices[i] + j - i
    end
  end
  yield indices.map {|k| array[k]}
end

```

Тогда получение мегашинглов сводится к вычислению MD5-хеша от

полученных всевозможных попарных сочетаний.

## LongSent

Реализация данного метода является элементарной по сравнению с другими рассмотренными алгоритмами. Контент каждого документа разбивается на предложения относительно знаков пунктуации '!', '!', '?' и запоминаются два самых длинных предложения. Вычисленный MD5-хеш от конкатенации этих предложений и является сигнатурой для документа.

## I-Match

Для вычисления I-Match сигнатур требуется составление словаря со средним значением IDF. С этой целью был написан rake task, анализирующий всю загруженную коллекцию документов:

```
namespace :documents do
  desc "Составление словаря для I-Match и перевычисление сигнатуры."
  task :re-i-match => :environment do
    global_words = {}
    Document.all.each do |document|
      document.i_match_signatures.destroy_all
      current_words = document.content.split(/^[А-ЯЁа-яё]+/).to_set
      current_words.each do |word|
        if global_words.has_key?(word)
          global_words[word] += 1
        else
          global_words.merge!({ word => 1 })
        end
      end
    end
  end

  document_count = Document.count.to_f
```

```

Word.destroy_all
global_words.each_pair do |word, number_documents_found|
  Word.create(:term => word,
              :number_documents_found => number_documents_found,
              :idf => Math.log2(document_count / number_documents_found))
end

Document.all.each do |document|
  document.create_i_match_signatures
end
end
end

```

Заводится структура типа хеш. В нее в качестве ключа будут помещены слова из всех документов, а значением станет количество документов, в которых это слово встретилось. Все слова из хеш помещаются в таблицу words с вычисленным значением IDF.

После построения словаря для каждого документа вызывается метод вычисления I-Match сигнатур. Задачей этого метода является нахождение пересечения двух словарей: все слова из таблицы БД words и словаря для текущего документа. По найденному пересечению вычисляется MD5 хеш и сохраняется в таблицу i\_macth\_signatures БД.

## MinHash

Как и в случае вычисления MinHash сигнатур мы возьмем ранее построенное семейство хеш-функций и метод для нахождения минимальных значений этих функций. В качестве терма будем использовать шингл. Тогда построение MinHash сигнатуры сводится к вычислению MD5 хеша для полученных минимальных значений min-wise independent функций.

```

def build_min_hash_signatures
  MinWise::find_min(shingle_signatures.map(&:token)).each do |min|

```

```
min_hash_signatures.new(:token => Digest::MD5.hexdigest(min.to_s))  
end  
end
```

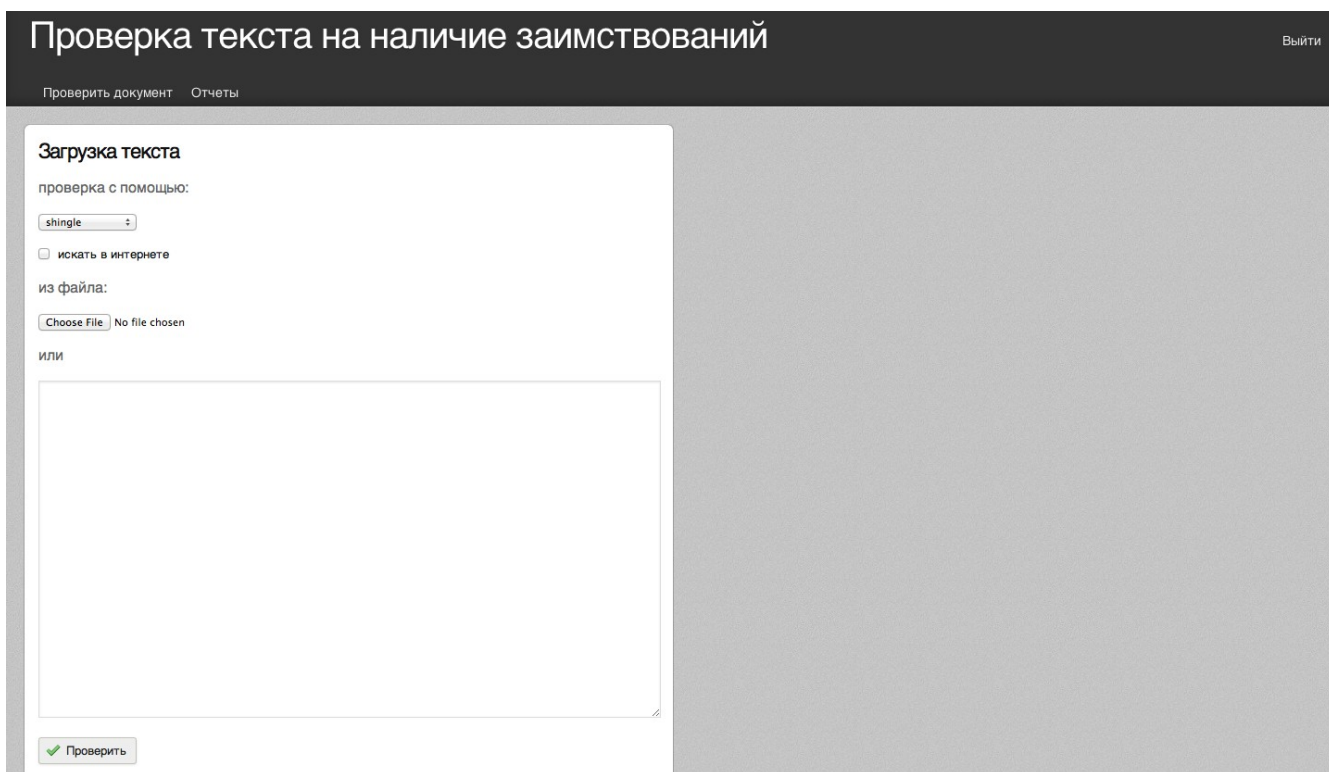
### **Поиск заимствований в Интернет**

Дополнительным требованием к поставленной перед нами задаче, является поиск заимствований в Интернете. С этой целью в качестве основного механизма, мы использовали крупные поисковые системы (Google, Yandex). При таком алгоритме проверки документа часть проверяемого текста отправлялась в качестве запроса в поисковики. Полученный ответ синтаксически анализируется с помощью библиотеки Nokogiri [14]. Результатом анализа является набор ссылок на страницы сайтов, максимально подходящий (с точки зрения поисковой системы) отправленному нами запросу. Далее происходит http запрос по каждой из ссылок. Тело полученного ответа очищается от html тегов и подвергается обработке реализуемых нами алгоритмов, которые описывались в этой главе.

## 5.4 Интерфейс взаимодействия с пользователем

Для удобной проверки документов на наличие заимствований был разработан веб-интерфейс. Доступ к нему осуществляется по заранее полученному логину и паролю. Пользователи наделяются определенными правами в зависимости от выбранной роли при их создании. В системе существует два типа ролей: администратор и преподаватель. Администратору принадлежат права на просмотр, создание, редактирование и удаление для всех объектов в реализуемой системе, а также сама проверка документа. Роли преподавателя разрешается проводить только проверку документа и просмотр отчета о наличии заимствований.

Пользователь, прошедший аутентификацию, попадает на главную страницу веб-сайта, изображенную на Рис. 3. На ней предоставляется основная возможность реализуемой системы — проверка документа на заимствование, а также просмотр отчетов по ранее загруженным работам:



The screenshot shows a web application titled "Проверка текста на наличие заимствований" (Text checking for plagiarism). The interface has a dark header with the title and a "Выйти" (Logout) link. Below the header, there are two tabs: "Проверить документ" (Check document) and "Отчеты" (Reports). The main content area is divided into two sections. The left section, titled "Загрузка текста" (Text upload), contains a dropdown menu for "shingle", a checkbox for "искать в интернете" (search on the internet), and a file upload section with a "Choose File" button and "No file chosen" text. Below this is a large text area for input. The right section is a large, empty gray area. At the bottom left of the text area, there is a green checkmark icon and a "Проверить" (Check) button.

Рис. 3

В выпадающем меню на главной странице можно выбрать различные варианты реализуемых нами алгоритмов для проверки документа. Также предусмотрена возможность загрузки документа из файла текстового формата.

Пользователь заполняет данными форму и отправляет ее на веб-сервер. Если обработка документа на сервере происходит быстро, то открывается страница отчета. В противном случае происходит перенаправление на страницу отчетов, где пользователь может следить за состоянием загруженной работы. Сама работа в таком случае обрабатывается отдельным процессом в системе. Как только статус работы станет «Обработано», преподавателю предоставится возможность просмотреть отчет. В зависимости от выбранного алгоритма проверки, отчеты могут различаться. Если алгоритм позволяет показать, какие фрагменты текста откуда были заимствованы, то применяются визуальные эффекты при формировании отчета, изображенные на Рис. 4:

## Проверка текста на наличие заимствований

Выйти

Проверить документ    Отчеты

### Отчет

**Схожесть 72 %, алгоритм shingle**

Как я работал в SEO-компани... Еще совсем недавно я работал в seo-компани, и теперь хочется немного рассказать миру о том, как происходит процесс работы над проектами в seo-компани. Уверен, что многим будет интересно: как тем, кто ищет себе подрядчика на seo, так и тем, кто уже продвигается с помощью компании. Я работал в двух seo-компаних на руководящих должностях, у всех есть свои небольшие особенности, но попробую расписать что-то среднее, но при этом, чтобы было связано и последовательно. Идеальный бизнес-процесс 1. Потенциальному клиенту предлагается продвижение в поисковых системах, чтобы увеличить свои продажи. 2. Изучается бизнес и целевая аудитория клиента. Исходя из этого составляется семантическое ядро (список запросов, по которым надо продвигаться). 3. Сем. ядро составляется не от балды, и не просто по запросам «кондиционеры» (это пример тупого запроса) — клиенту нужно объяснить, что продвижение по таким запросам ничего не даст, надо целиться уже, потому что и продвижение будет стоить дешевле, и конверсия с узконаправленных запросов больше. Т.е. не просто «кондиционеры», а «установка кондиционеров москва». 4. Просчет итоговых запросов, заключение договора, начало работы. 5. Работа над сайтом клиента. Оптимизация под поисковые запросы, аудит юзабилити для улучшения конверсии. Как-то я прочел в книге Огилви «О рекламе» про то, как потенциальный клиент рассказывал ему, как надо проводить рекламную кампанию. Огилви его внимательно выслушал, а потом проводил за дверь. Эта поучительная история не про seo-компани. Никто не будет никого выпроваживать за дверь, если он готов платить деньги. Он будет сидеть на голове у ваших сотрудников? Ничего страшного, ведь именно за это им и платятся деньги, нет так ли? 6. Закупка внешних ссылок. 7. Разъяснение клиенту особенной поискового продвижения. 8. Мониторинг статистики и ее анализ. Предложения по развитию сайта, улучшающих продажи. Вот она, идеальная схема. Может какие-то детали я упустил, но это не столь принципиально для дальнейших рассуждений. Что происходит на самом деле Про сейлзов Пункты 1-4 из нашего списка осуществляются менеджерами по продажам. Менеджеры по продажам сидят на маленьком окладе и большом проценте, это должно их мотивировать делать продажи. И это реально работает: им в итоге по фиг, что продавать и как, лишь бы продать, потому что они получают за это свой процент. А начинают получать они его в тот момент, когда после

Document id: 1

Еще совсем недавно я работал в seo-компани, и теперь хочется немного рассказать миру о том, как происходит процесс работы над проектами в seo-компани. Уверен, что многим будет интересно: как тем, кто ищет себе подрядчика на seo, так и тем, кто уже продвигается с помощью компании. Я работал в двух seo-компаних на руководящих должностях, у всех есть свои небольшие особенности, но попробую расписать что-то среднее, но при этом, чтобы было связано и последовательно. Идеальный бизнес-процесс 1. Потенциальному клиенту предлагается продвижение в поисковых системах, чтобы увеличить свои продажи. 2. Изучается бизнес и целевая аудитория клиента. Исходя из этого составляется семантическое ядро (список запросов, по которым надо продвигаться). 3. Сем. ядро составляется не от балды, и не просто по запросам «кондиционеры» (это пример тупого запроса) — клиенту нужно объяснить, что продвижение по таким запросам ничего не даст, надо целиться уже, потому что и продвижение будет стоить дешевле, и конверсия с узконаправленных запросов больше. Т.е. не просто «кондиционеры», а «установка кондиционеров москва». 4. Просчет итоговых запросов, заключение договора, начало работы. 5. Работа над сайтом клиента. Оптимизация под поисковые запросы, аудит юзабилити для улучшения конверсии. 6. Закупка внешних ссылок. 7. Разъяснение клиенту особенной поискового продвижения. 8. Мониторинг статистики и ее анализ. Предложения по развитию сайта, улучшающих продажи. Вот она, идеальная схема. Может какие-то детали я упустил, но это не столь принципиально для дальнейших рассуждений.

Рис. 4



Цветом подсвечены фрагменты текста, которые были заимствованы. При нажатии на любое слово из такого фрагмента, оно выделяется и справа показывается документ, а также конкретное место, показывающее откуда этот фрагмент был взят. Документ в правой части подсвечивается тем же цветом, что и выделенный фрагмент из исходного текста.

Если выбранный алгоритм выявления не позволяет визуализировать заимствованные фрагменты текста, то отчет показывается в более консервативном виде, изображенном на Рис. 5:

## Проверка текста на наличие заимствований

[Выйти](#)

[Проверить документ](#)
[Отчеты](#)

### Отчет

**Схожесть 73 %, алгоритм min-hash**

Как я работал в SEO-компании. Еще совсем недавно я работал в seo-компаниях, и теперь хочется немного рассказать миру о том, как происходит процесс работы над проектами в seo-компаниях. Уверен, что многим будет интересно: как тем, кто ищет себе подрядчика на seo, так и тем, кто уже продвигается с помощью компании. Зачастую свояшник или аккаунт-менеджер по приходу проекта в работу должны тут же предложить поменять список запросов на более вменяемый. Но клиента это может смутить: «Вы нам только что продали, а теперь тут же предлагаете менять? Как-то странно, вы же из одной компании». Мда, ну черт с ним тогда, будем продвигать, что есть. Я работал в двух seo-компаниях на руководящих должностях, у всех есть свои небольшие особенности, но попробую расписать что-то среднее, но при этом, чтобы было связано и последовательно. Идеальный бизнес-процесс 1. Потенциальному клиенту предлагается продвижение в поисковых системах, чтобы увеличить свои продажи. 2. Изучается бизнес и целевая аудитория клиента. Исходя из этого составляется семантическое ядро (список запросов, по которым надо продвигаться). 3. Сем. ядро составляется не от балды, и не просто по запросам «кондиционеры» (это пример тупого запроса) — клиенту нужно объяснить, что продвижение по таким запросам ничего не даст, надо целиться уже, потому что и продвижение будет стоить дешевле, и конверсия с узконаправленных запросов больше. Т.е. не просто «кондиционеры», а «установка кондиционеров москва». 4. Просчет итоговых запросов, заключение договора, начало работы. 5. Работа над сайтом клиента. Оптимизация под поисковые запросы, аудит юзабилити для улучшения конверсии.

20 % Еще совсем недавно я работал в seo-компаниях, и теперь хочется немного рассказать миру о том, как происходит процесс работы над проектами в seo-компаниях. Уверен, что многим будет интересно: как тем...

29 % Что происходит на самом деле. Про сейлзов. Пункты 1-4 из нашего списка осуществляются менеджерами по продажам. Менеджеры по продажам сидят на маленьком окладе и большом проценте, это должно их...

24 % Про клиентов и всех-всех-всех 1. Клиент, который все знает. Клиент может оказаться таким, который сам начнет тебе рассказывать, как надо делать, и вообще имеет свое мнение по тому, как продвиг...

Рис. 5

Из такого типа отчета виден общий процент заимствования, а также информация по конкретному документу, в котором был найден дублирующий контент. Перейдя по ссылке такого документа, показывается полный текст контента для дальнейшей текстуальной верификации.

## **6. Анализ полученных результатов**

Для апробирования написанных алгоритмов были сгенерированы несколько тестовых коллекций дубликатов с различными типами изменений:

1. Обработка 60% текста синонимайзером.
2. Обработка 40% текста синонимайзером.
3. Обработка 20% текста синонимайзером.
4. Обработка 5% текста синонимайзером.
5. Изменение порядка следования предложений.
6. Изменение порядка следования параграфов.
7. Замена русских букв на их визуальные аналоги из латинского алфавита.

Синонимайзер — программа или сервис, осуществляющая замену слов или фраз во введённом тексте на синонимы, находящиеся в базе данных с целью видоизменения текста и придания ему уникальности. В качестве синонимайзера был выбран Интернет-сервис [seogenerator.ru](http://seogenerator.ru) [13]. Во всемирной паутине существуют много аналогичных ресурсов, однако этот сервис представляет удобный API для обработки текстов. Данный сервис имеет некоторые недостатки. Во-первых, существуют ограничения на размер исходного текста. Во-вторых, обрабатывается не более 20 запросов в минуту. Нужно иметь в виду, что ситуация, когда обрабатываемый текст составляет более, чем 40% по отношению к исходному, может привести к потере смысловой нагрузки по причине того, что синонимы могут быть заменены в неправильном контексте.

Было написано несколько скриптов, изменяющих исходный текст документа в соответствии с вышеописанными пунктами 1 - 7. Чтобы максимально приблизить результаты тестирования к реальности, в качестве документов, с которыми будут производиться сравнения, были взяты 100 реальных работ студентов. После обработки этих документов, они подвергались проверке эталонной функцией схожести. В качестве такой функции было взято расстояние

Левенштейна с пороговым значением сходства 80%. Ограничение в виде 100 работ связано с большими временными затратами на получение коллекции дубликатов с характеристиками из пунктов 1 — 4, а также нахождение расстояния Левенштейна.

После импорта исходных текстов на вход программе подавались тексты дубликатов. Каждый из реализованных алгоритмов обрабатывал дубликаты и сохранял в БД полученные результаты в виде отчета. Пороговое значение заимствования текста является субъективным понятием и зависит зачастую от тематики работы. Мы в качестве этого значения взяли 70%.

В таблицах 2 - 7 представлены результаты работы алгоритмов. В качестве длины одного шингла для алгоритмов «шинглы», «супершинглы», «мегашинглы», «MinHash», было взято значение 3 термина. Словарь для I-Match алгоритма составлялся из термов, значения IDF которых лежали в диапазоне от  $(IDF\_MEAN - 0.8)$  до  $(IDF\_MEAN + 0.8)$ , где  $IDF\_MEAN$  — это среднее значение IDF по всему словарю.

**Обработка 60% текста синонимайзером. Таблица 2**

Алгоритм	Полнота	Точность
Шинглы	0.02	1
Супершинглы	0	0
Мегашинглы	0	0
MinHash	0	0
I-Match	0	0
LongSent	0.38	0.94

**Обработка 40% текста синонимайзером. Таблица 3**

Алгоритм	Полнота	Точность
Шинглы	0.33	0.96
Супершинглы	0.01	1

Алгоритм	Полнота	Точность
Мегашинглы	0	0
MinHash	0.01	1
I-Match	0.8	0.95
LongSent	0.55	0.96

**Обработка 20% текста синонимайзером. Таблица 4**

Алгоритм	Полнота	Точность
Шинглы	0.91	0.86
Супершинглы	0.25	0.94
Мегашинглы	0.06	0.89
MinHash	0.64	0.93
I-Match	0.18	0.96
LongSent	0.7	0.98

**Обработка 5% текста синонимайзером. Таблица 5**

Алгоритм	Полнота	Точность
Шинглы	0.97	0.98
Супершинглы	0.92	0.96
Мегашинглы	0.79	0.94
MinHash	0.97	0.99
I-Match	0.35	0.8
LongSent	0.79	0.78

**Изменения порядка следования предложений параграфов. Таблица 6**

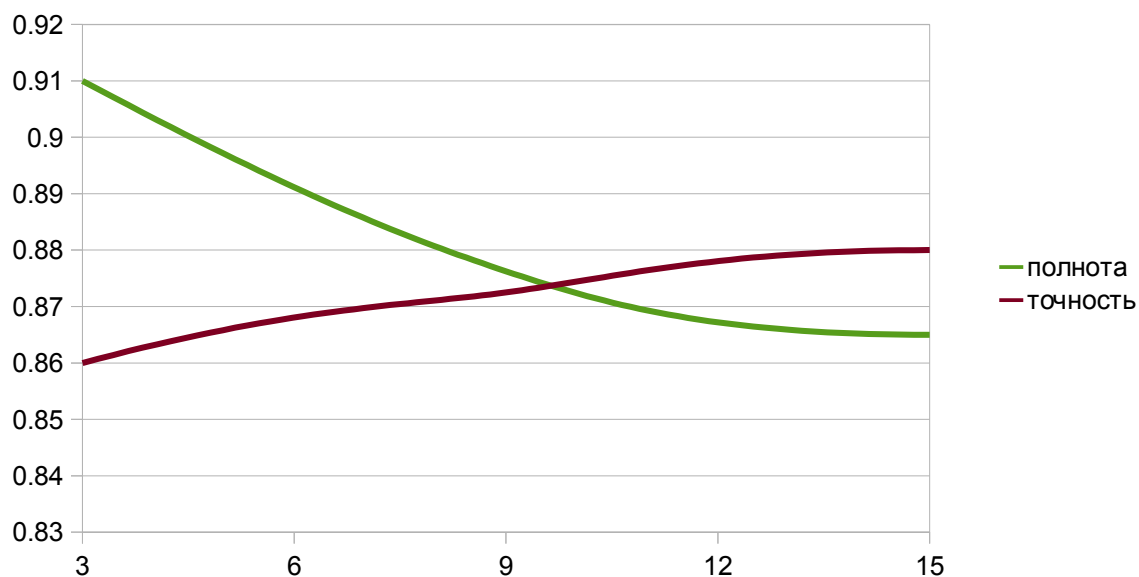
Алгоритм	Полнота	Точность
Шинглы	0.96	0.77
Супершинглы	0.94	0.77
Мегашинглы	0.88	0.79

Алгоритм	Полнота	Точность
MinHash	0.97	0.75
I-Match	0.94	0.71
LongSent	0.98	0.72

**Изменения порядка следования предложений. Таблица 7**

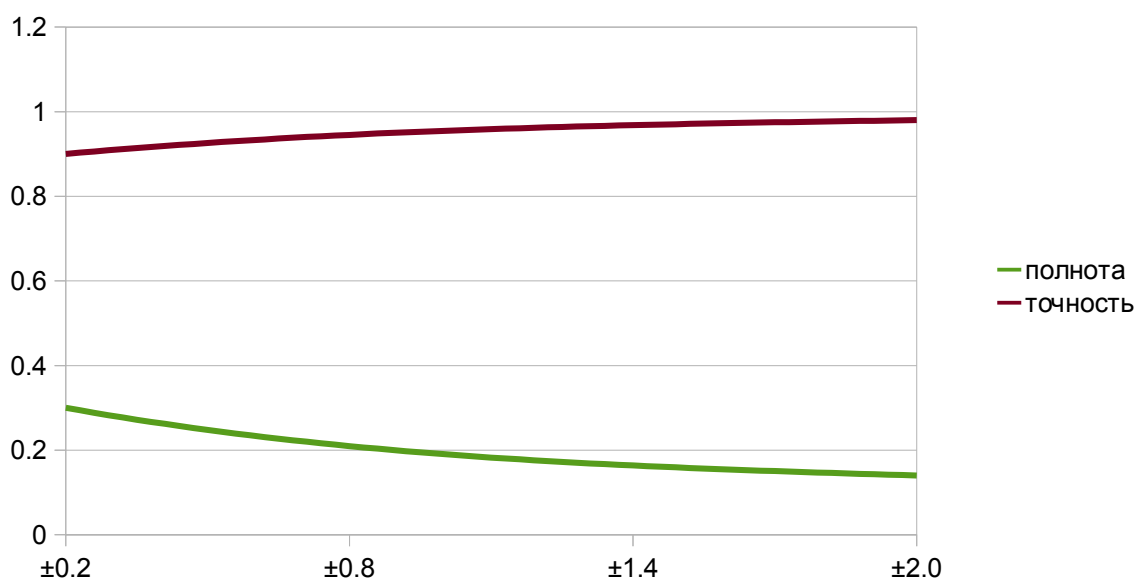
Алгоритм	Полнота	Точность
Шинглы	0.97	0.53
Супершинглы	0.79	0.59
Мегашинглы	0.68	0.6
MinHash	0.98	0.54
I-Match	0.96	0.54
LongSent	1	0.55

Также были проведены аналогичные измерения для длины одного шингла, равной 6, 9, 12, 15 термам. На графике ниже показана зависимость полноты и точности от длины шингла после обработки 20% текста синонимайзером.



Как мы видим, с увеличением количества термов в одном шингле, показатель полноты падает, а точность увеличивается. Брать минимальное значение длины шингла невыгодно, так как увеличивается количество сравнений, производимых при проверке документа на наличие заимствования. К тому же увеличивается размер занимаемой памяти БД.

Для алгоритма I-Match были произведены дополнительные тесты с разными значениями диапазонов IDF для выборки по словарю. Результаты приведены в виде графика зависимости точности и полноты от размера диапазона для типа изменения текста «Изменения порядка следования предложений».



В таблицах 2 - 7 специально не приведены значения для типа изменения текста «Замена русских букв на их визуальные аналоги из латинского алфавита». Это связано с тем, что все тексты проходят предобработку, в которой заменяются латинские буквы на их аналоги из русского алфавита. Таким образом, алгоритмы всегда выдавали значение полноты и точности, равные 1.

Также было произведено сравнение с другой популярной системой, решающей подобную задачу. Ресурс [antiplagiat.ru](http://antiplagiat.ru) [10] позволяет с определенными

ограничениями произвести проверку документа на наличие заимствования. Было выбрано несколько общедоступных в Интернет рефератов. Эти работы были импортированы в реализованную нами систему. Часть текстов была обработана синонимайзером, отослана на проверку на сайт [antiplagiat.ru](http://antiplagiat.ru) [10] и в нашу систему. Разница полученных результатов варьировалась в диапазоне 5-10%.

## 7. Выводы

Исходя из полученных в предыдущей главе результатов, можно сказать, что была реализована система, отвечающая всем требованиям из главы 3, а именно:

1. Разработана система, позволяющая выявлять заимствования в тексте.
2. Система показала хорошие результаты при тестировании.
3. Время работы реализованных алгоритмов является приемлемым.
4. Система имеет удобный и наглядный интерфейс.
5. Существует возможность проверки на наличие заимствований в сети Интернет.

Также система показала хороший результат в сравнении с коммерческими аналогами.

Существует ряд идей, которые смогли бы улучшить эффективность работы системы. В первую очередь, с помощью программы *mystem* [12] или аналогичных программ можно перед обработкой слов производить их канонизацию. Это должно позволить увеличить показатель полноты для подвергшихся небольшим изменениям документов. Из результатов тестовых прогонов было видно, что замена слов на синонимы может значительно ухудшить эффективность алгоритмов. Поэтому для шинглов можно записывать не одну запись в БД, как мы это делали, а несколько записей с замененными синонимами. Для метода I-Match вычисление значения IDF лучше производить не по классической формуле, которую мы использовали, а воспользоваться одним из ее «сглаженных» аналогов. Это должно позволить делать более качественную выборку словаря.



## 8. Приложение

Исходный текст app/models/ability.rb

```
# -*- encoding : utf-8 -*-  
class Ability  
  include CanCan::Ability  
  
  def initialize(user)  
    if user.role? :admin  
      can :manage, :all  
    elsif user.role? :teacher  
      can :read, Document  
      can :manage, Report, :user_id => user.id  
    end  
  end  
end
```

Исходный текст app/models/coloring.rb

```
# -*- encoding : utf-8 -*-  
class Coloring  
  attr_reader :position_start, :position_end, :color, :token  
  
  def initialize(options = {})  
    @position_start = options[:position_start]  
    @position_end = options[:position_end]  
    @color = options[:color]  
    @token = options[:token]  
  end  
end
```

end

Исходный текст app/models/document.rb

```
# -*- encoding : utf-8 -*-
```

```
require 'csv'
```

```
require "net/http"
```

```
class Document < ActiveRecord::Base
```

```
  has_many :shingle_signatures, :dependent => :destroy, :order =>  
'position_start asc'
```

```
  has_many :i_match_signatures, :dependent => :destroy
```

```
  has_many :min_hash_signatures, :dependent => :destroy, :order =>  
:created_at
```

```
  has_many :super_shingle_signatures, :dependent => :destroy
```

```
  has_many :mega_shingle_signatures, :dependent => :destroy
```

```
  has_many :long_sent_signatures, :dependent => :destroy
```

```
  validates :content, :presence => true
```

```
  after_create :create_signatures
```

```
  after_initialize :initialize_for_match
```

```
  attr_accessor :similarity, :paint, :matched_documents
```

```
  def initialize_for_match
```

```
    @similarity = 0
```

```
    @paint = []
```

```
    @matched_documents = []
```

```

end

def create_signatures
  Document.benchmark('create_shingle_signatures') do
    create_shingle_signatures
  end
  Document.benchmark('create_min_hash_signatures') do
    create_min_hash_signatures
  end
  Document.benchmark('create_super_shingle_signatures') do
    create_super_shingle_signatures
  end
  Document.benchmark('create_mega_shingle_signatures') do
    create_mega_shingle_signatures
  end
  Document.benchmark('create_long_sent_signatures') do
    create_long_sent_signatures
  end
end

def build_long_sent_signatures
  if long_sent_signatures.empty?
    max1, max2 = '', ''

    content.split(/[[[:cntrl:]][:punct:]]/).each do |sent|
      if max1.length < sent.length
        max1 = sent
        max2 = max1
      end
    end
  end
end

```

```

        end
    end

    long_sent_signatures.new(:token => Digest::MD5.hexdigest([max1,
max2].sort.join))
    end
end

def build_mega_shingle_signatures
    if mega_shingle_signatures.empty?
        generate_combinations_for_mega_shingle do |mega_shingle|
            mega_shingle_signatures.new(:token =>
Digest::MD5.hexdigest(mega_shingle.join))
        end
    end
end

def build_super_shingle_signatures
    if super_shingle_signatures.empty?
        super_shingle_signatures.new(:token =>
Digest::MD5.hexdigest(min_hash_signatures[0...6].map(&:token).join))
        super_shingle_signatures.new(:token =>
Digest::MD5.hexdigest(min_hash_signatures[6...12].map(&:token).join))
        super_shingle_signatures.new(:token =>
Digest::MD5.hexdigest(min_hash_signatures[12...18].map(&:token).join)
        )
        super_shingle_signatures.new(:token =>
Digest::MD5.hexdigest(min_hash_signatures[18...24].map(&:token).join)
    end
end

```

```

)
    super_shingle_signatures.new(:token =>
Digest::MD5.hexdigest(min_hash_signatures[24...30].map(&:token).join)
)
    super_shingle_signatures.new(:token =>
Digest::MD5.hexdigest(min_hash_signatures[30...36].map(&:token).join)
)
    end
end

def build_min_hash_signatures
  if min_hash_signatures.empty?
    MinWise::find_min(shingle_signatures.map(&:token)).each do |
min|
      min_hash_signatures.new(:token =>
Digest::MD5.hexdigest(min.to_s))
    end
  end
end

def build_shingle_signatures
  if shingle_signatures.empty?
    shingling = Shingling.new(
      content,
      :replace_chars => Diploma::Application::ALPHABETIC,
      :stop_words => Text::STOP_WORDS,
      :shingle_length => ShingleSignature::SHINGLE_LENGTH,
      :downcase => true
  end
end

```

```

    )

    shingling.each_shingles do |shingle, position_start,
position_end|
        shingle_signatures.new(
            :token => Digest::MD5.hexdigest(shingle),
            :position_start => position_start,
            :position_end => position_end
        )
    end
end
end
end

def build_i_match_signatures
    current_words = content.split(/[^A-ЯЁa-яё]+/).to_set
    i_match_signatures.new(:token =>
Digest::MD5.hexdigest((current_words &
Diploma::Application::DICTIONARY).to_a.sort.join))
end

def similarity_super_shingle_signatures
    @matched_documents = Document.select("DISTINCT ON (documents.id)
documents.*").joins(:super_shingle_signatures).where(:"super_shingle_
signatures.token" =>
super_shingle_signatures.map(&:token).map(&:to_s))#.group(:"documents
.id")
    @similarity = @matched_documents.size >= 2 ? 95 : 0
end

```

```

def similarity_long_sent_signatures
  @matched_documents = Document.select("DISTINCT ON (documents.id)
documents.*").joins(:long_sent_signatures).where(:"long_sent_signatur
es.token" =>
long_sent_signatures.map(&:token).map(&:to_s))#.group(:"documents.id"
)
  @similarity = @matched_documents.empty? ? 0 : 100
end

def similarity_i_match_signatures
  @matched_documents = Document.select("DISTINCT ON (documents.id)
documents.*").joins(:i_match_signatures).where(:"i_match_signatures.t
oken" =>
i_match_signatures.map(&:token).map(&:to_s))#.group(:"documents.id")
  @similarity = @matched_documents.empty? ? 0 : 100
end

def similarity_mega_shingle_signatures
  @matched_documents = Document.select("DISTINCT ON (documents.id)
documents.*").joins(:mega_shingle_signatures).where(:"mega_shingle_si
gnatures.token" =>
mega_shingle_signatures.map(&:token).map(&:to_s))#.group(:"documents.
id")
  @similarity = @matched_documents.size >= 2 ? 95 : 0
end

def similarity_min_hash_signatures

```

```

    equal_count = 0.0
    global_equal_count = 0.0
    min_wise_function_is_equal = {}
    @matched_documents = Document.select("DISTINCT ON (documents.id)
documents.*").joins(:min_hash_signatures).where("min_hash_signatures
.token" =>
min_hash_signatures.map(&:token).map(&:to_s))#.group("documents.id")

    matched_documents.each do |document|
      MinWise::FUNCTION_NUMBER.times do |i|
        if min_hash_signatures[i].token.to_s ==
document.min_hash_signatures[i].token.to_s
          equal_count += 1
          unless min_wise_function_is_equal.has_key?(i)
            global_equal_count += 1
            min_wise_function_is_equal.merge! i => true
          end
        end
      end
      document.similarity = (equal_count / MinWise::FUNCTION_NUMBER *
100).round(0)
      equal_count = 0.0
    end

    @similarity = (global_equal_count / MinWise::FUNCTION_NUMBER *
100).round(0)
  end

```



```

def similarity_shingle_signatures
  initialize_for_match
  number_matched = 0
  hash_shingle_signatures = nil
  in_database = []
  local_matched_documents = {}

  Document.benchmark("Create hash from array for
shingle_signatures") do
    hash_shingle_signatures = Hash[shingle_signatures.map {|s|
[s.token, s]}]
  end

  Document.benchmark("shingle_match") do
    in_database = ShingleSignature.where(:token =>
shingle_signatures.map(&:token))
  end

  Document.benchmark("Coloring") do
    in_database.each do |shingle_match|
      shingle_signature =
hash_shingle_signatures[shingle_match.token]
      number_matched += 1
      color = ColorForDocument.get(shingle_match.document_id)

      @paint << Coloring.new(:token => shingle_signature.token,
                             :position_start =>
shingle_signature.position_start,
                             :position_end =>

```

```

shingle_signature.position_end,
                                :color => color)

    matched_document =
local_matched_documents[shingle_match.document_id]
    if matched_document.blank?
        local_matched_documents.merge! shingle_match.document_id =>
shingle_match.document
        @matched_documents << shingle_match.document
        matched_document = shingle_match.document
    end
    matched_document.paint << Coloring.new(:token =>
shingle_signature.token,
                                :position_start =>
shingle_match.position_start,
                                :position_end =>
shingle_match.position_end,
                                :color => color)

    end
end

    @similarity = number_matched > 0 ? (number_matched * 100.0 /
(shingle_signatures.size)).round(0) : 0
end

def paint_sort
    @paint.sort! { |a, b| a.position_start <=> b.position_start }
end

```

```
def create_super_shingle_signatures
  build_super_shingle_signatures
  super_shingle_signatures.map(&:save)
end
```

```
def create_mega_shingle_signatures
  build_mega_shingle_signatures
  mega_shingle_signatures.map(&:save)
end
```

```
def create_min_hash_signatures
  build_min_hash_signatures
  min_hash_signatures.map(&:save)
end
```

```
def create_i_match_signatures
  build_i_match_signatures
  i_match_signatures.map(&:save)
end
```

```
def create_long_sent_signatures
  build_long_sent_signatures
  long_sent_signatures.map(&:save)
end
```

```
def create_shingle_signatures
  build_shingle_signatures
```

```

    conn = ActiveRecord::Base.connection_pool.checkout
    raw = conn.raw_connection
    in_database = ShingleSignature.where(:token =>
shingle_signatures.map(&:token))
    tmp = in_database.map(&:token).to_set

    raw.exec("COPY shingle_signatures (token, position_start,
position_end, document_id) FROM STDIN DELIMITERS ','")
    shingle_signatures.each do |shingle_signature|
        unless tmp.include?(shingle_signature.token)
            raw.put_copy_data "#{shingle_signature.token},
#{shingle_signature.position_start},
#{shingle_signature.position_end},
#{shingle_signature.document_id}\n"
            tmp.add shingle_signature.token
        end
    end
    raw.put_copy_end

    while res = raw.get_result
        # Говорят что важно
    end

    ActiveRecord::Base.connection_pool.checkin(conn)
end

def search_from_web
    query = query_for_search_from_web
    search_from_yandex query
end

```

```

        search_from_google query
    end

    private

    def search_from_yandex query
        Document.benchmark("#search_from_yandex") do
            documents = nil

            Document.benchmark("#search_from_yandex
ScrapingYandex.search('\#{query}\")") do
                documents = ScrapingYandex.search(:query => "\#{query}\")
            end

            Document.benchmark("#search_from_google Document.create") do
                documents.each_pair do |link, content|
                    Document.create(:content => content, :source => link)
                unless Document.find_by_source(link)
                    end
                end
            if documents.empty?
                Document.benchmark("#search_from_yandex
ScrapingYandex.search('\#{query}')") do
                    documents = ScrapingYandex.search(:query => query)
                end
                documents.each_pair do |link, content|
                    Document.create(:content => content, :source => link)
                unless Document.find_by_source(link)
                    end
                end
            end
        end
    end
end

```

```

        end
    end
end

def search_from_google query
    Document.benchmark("#search_from_google") do
        documents = nil

        Document.benchmark("#search_from_google
ScrapingGoogle.search('#{query}')) do
            documents = ScrapingGoogle.search(:query => "#{query}")
        end

        Document.benchmark("#search_from_google Document.create") do
            documents.each_pair do |link, content|
                Document.create(:content => content, :source => link)
            unless Document.find_by_source(link)
                end
            end

            if documents.empty?
                Document.benchmark("#search_from_google
ScrapingGoogle.search('#{query}')) do
                    documents = ScrapingGoogle.search(:query => query)
                end

                documents.each_pair do |link, content|
                    Document.create(:content => content, :source => link)
                unless Document.find_by_source(link)
                    end
                end
            end
        end
    end
end

```

```

    end
end

def query_for_search_from_web
  length = 300
  if content.length > length
    index = rand(content.length - length)
    index_first_word = index
    index_last_word = index + length

    while content[index_first_word] !~ /\s/ && index_first_word <
content.length
      index_first_word += 1
    end

    while content[index_last_word] !~ /\s/ && index_last_word <
content.length
      index_last_word += 1
    end

    position_start = index_first_word
    position_end = index_last_word
  else
    position_start = 0
    position_end = content.length
  end

  return content[position_start..position_end].strip

```

```

end

def generate_combinations_for_mega_shingle
  array = super_shingle_signatures.map(&:token)
  r = 2
  n = array.length
  indices = (0...r).to_a
  final = (n - r...n).to_a
  while indices != final
    yield indices.map {|k| array[k]}
    i = r - 1
    while indices[i] == n - r + i
      i -= 1
    end
    indices[i] += 1
    (i + 1...r).each do |j|
      indices[j] = indices[i] + j - i
    end
  end
  yield indices.map {|k| array[k]}
end
end

```

Исходный текст app/models/report.rb

# -\*- encoding : utf-8 -\*-

Document.class

ShingleSignature.class



Coloring.class

MinHashSignature.class

IMatchSignature.class

MegaShingleSignature.class

SuperShingleSignature.class

LongSentSignature.class

```
class Report < ActiveRecord::Base
```

```
  belongs_to :user
```

```
  state_machine :state, :initial => :new do
```

```
    event :initial do
```

```
      transition any => :new
```

```
    end
```

```
    event :process do
```

```
      transition :new => :processed
```

```
    end
```

```
    event :complite do
```

```
      transition :processed => :complited
```

```
    end
```

```
  end
```

```
  def cancel_generate
```

```
    initial!
```

```
  end
```

```

def serialized_object=(object)
  Report.benchmark("Report#serialized_object=") do
    write_attribute :serialized_object,
ActiveSupport::Base64.encode64(Marshal.dump(object))
  end
end

def serialized_object
  Report.benchmark("Report#serialized_object") do
    Marshal.load(ActiveSupport::Base64.decode64(read_attribute :serialized_object))
  end
end

def generate_and_save options = {}
  generate options
  save!
end

def similarity
  (read_attribute :similarity).round(0)
end

def generate options = {}
  process!
  @document = options[:document].present? ? options[:document] :
Document.new(:content => options[:content])

```

```

@document.search_from_web if options[:web] == true
case algorithm
when 'shingle'
  @document.build_shingle_signatures
  @document.similarity_shingle_signatures
when 'super-shingle'
  @document.build_shingle_signatures
  @document.build_min_hash_signatures
  @document.build_super_shingle_signatures
  @document.similarity_super_shingle_signatures
when 'mega-shingle'
  @document.build_shingle_signatures
  @document.build_min_hash_signatures
  @document.build_super_shingle_signatures
  @document.build_mega_shingle_signatures
  @document.similarity_mega_shingle_signatures
when 'min-hash'
  @document.build_shingle_signatures
  @document.build_min_hash_signatures
  @document.similarity_min_hash_signatures
when 'i-match'
  @document.build_i_match_signatures
  @document.similarity_i_match_signatures
when 'long-sent'
  @document.build_long_sent_signatures
  @document.similarity_long_sent_signatures
end

```

```

    self.similarity = @document.similarity
    self.serialized_object = @document
    complite!
  end
end

```

Исходный текст app/models/shingle\_signature.rb

```

# -*- encoding : utf-8 -*-
class ShingleSignature < ActiveRecord::Base
  SHNINGLE_LENGTH = 9
  belongs_to :document

  validates :token, :presence => true, :uniqueness => true
  validates :position_start, :presence => true, :numericality => true
  validates :position_end, :presence => true, :numericality => true
  validates :document_id, :presence => true

  def range
    self.position_start...self.position_end
  end

  def cut_content
    self.document.content[range]
  end
end

```

Исходный текст app/models/user.rb

```

# -*- encoding : utf-8 -*-

```

```

class User < ActiveRecord::Base
  devise :database_authenticatable, :registerable, :recoverable, :rememberable,
:trackable, :validatable

  has_many :reports

  attr_accessible :email, :password, :password_confirmation, :remember_me

  def role? argv
    role == argv.to_s
  end
end

```

Исходный текст min\_wise.rb

```

# -*- encoding : utf-8 -*-

module MinWise
  def self.find_min numbers
    raise ArgumentError unless numbers.is_a?(Array)
    mins = []
    MIN_WISE_FUNCTIONS.each do |min_wise_function|
      min = min_wise_function.call numbers.first.hex if numbers.first
      numbers.each do |number|
        current = min_wise_function.call number.hex
        min = current if min > current
      end
      mins << min
    end
  end
end

```

```
    return mins
  end
end
```

Исходный текст scraping\_google.rb

```
# -*- encoding : utf-8 -*-
```

```
require 'nokogiri'
```

```
require 'open-uri'
```

```
require "cgi"
```

```
require "iconv"
```

```
module ScrapingYandex
```

```
  def self.search(options = {})
```

```
    raise ArgumentError, "Query is not a Hash" unless options.is_a? Hash
```

```
    raise ArgumentError, "Is not empty query" unless options[:query]
```

```
    ic = Iconv.new('UTF-8//IGNORE', 'UTF-8')
```

```
    documents = {}
```

```
    issuance = Nokogiri::HTML(open("http://yandex.ru/yandsearch?
```

```
text=#{CGI::escape(options[:query])}").read)
```

```
    issuance.css('a.b-serp-item__title-link').each do |link|
```

```
      begin
```

```
        href = link.attributes["href"].content
```

```
        html = open(href)
```

```
        if html.content_type == "text/html"
```

```
          doc = Nokogiri::HTML(html.read)
```

```
          doc.css('script').remove
```

```
    doc.css('style').remove
    content = ic.iconv(doc.content)
    documents.merge! href => content
  end
rescue Exception => e
  Rails.logger.debug { "#{e.message} #{e.backtrace}" }
next
end
end

return documents
end
end
```

## 9. Литература

1. А. А. Евсеев Анализ текстов на заимствование методом построения семантических моделей.
2. Ю. Г. Зеленков, И.В. Сегалович. Сравнительный анализ методов определения нечетких дубликатов для Web-документов.
3. 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.
4. A. Broder, M. Charikar et al. Min-wise independent permutations, Proceedings of the thirtieth annual ACM symposium on Theory of computing, 1998
5. 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).
6. A. Chowdhury. Duplicate Data Detection.  
<http://ir.iit.edu/~abdur/Research/Duplicate.html>
7. 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>
8. D. Fetterly, M. Manasse, M. Najork. A Large-Scale Study of the Evolution of Web Pages, WWW2003, May 20-24, 2003, Budapest, Hungary.
9. 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.
10. <http://www.antiplagiat.ru/index.aspx>
11. <http://snowball.tartarus.org/algorithms/russian/stop.txt>
12. <http://company.yandex.ru/technology/mystem/>
13. <http://seogenerator.ru/>
14. <https://github.com/tenderlove/nokogiri>