УДК 004.424.47:004.738.5

ИСПОЛЬЗОВАНИЕ ПЕРЦЕПТИВНОГО ХЭША В НАХОЖДЕНИИ НЕОБХОДИМЫХ СООБЩЕНИЙ НА ПРИМЕРЕ ВЫБОРКИ ИЗ ТВИТТЕРА

Ю.А. Захарик
Студентка БГУ

Д.М. Прокурат
Магистр математики и информационных технологий

Белорусский государственный университет, Республика Беларусь
E-mail: zakharik.julia@gmail.com

Ю.А Захарик
Студентка механико-математического факультета Белорусского государственного университета.

Д.М. Прокурат
Магистр математики и информационных технологий.

Аннотация. В данной работе была рассмотрена проблема нахождения схожих сообщений на основе выборки из Твиттера. Рассмотрены следующие алгоритмы: расстояние Левенштейна, нормализованное расстояние Левенштейна, Soundex, Phonex, алгоритм Джаро, Q-граммы, алгоритм LCS, алгоритм Metric LCS, SIFT4, алгоритм Дамерлау-Левенштейна, алгоритм косинусов и алгоритм подсчета букв. Было произведено сокращение выборки и упрощение твитов. Предложен оптимальный алгоритм решения задачи.

Ключевые слова: перцептивный хэш, сравнение текстов, алгоритм косинусов, алгоритм Джаро, твиттер

Введение. В современном мире количество информации каждые два года удваивается. Со временем появляется все большее число новостей, которые являются явными дубликатами без указания ссылок на оригинал. Из чего вырастает необходимость нахождения плагиата оригинального сообщения. На данный момент известно большое количество алгоритмов, решающих данную проблему на небольших количествах данных, однако, показывающих плохие результаты на большой выборке, из-за чего данная проблема не решена в масштабах Интернета. Была проделана работа по нахождению схожих сообщений на примере 800.000 твитов. Твит – короткое сообщение, длиной до 280 символов, опубликованное в сервисе Twitter.

Целью данной работы является выявление пользы использования перцептивного хэша в поиске схожих сообщений на примере выборки из Твиттера.

Определения. Определим, что значит, что два сообщения схожи. Схожесть определяет, насколько оба текста «близки» друг к другу. Будем считать тексты схожими, если:

- Тексты написаны на одинаковом языке;
- «Смысл» текстов схож;
- Изменён порядок слов;
- В словах допущены опечатки
- Допускаются изменения форм слов (падежи, спряжение и т.д.);
- Добавлены или удалены некоторые слова в предложениях;
- Некоторые слова заменены на синонимы.

– Сообщения практически одинаковы по длине

Рассчеты сложности. На выборке из $N_1 = 800.000 N_1 = 800.000$ твитов необходимо совершить $\sum_{i=1}^{N} i \sum_{i=1}^{N} i$ операций. Зная, что примерно 40% всех твитов являются ретвитами, можем сократить выборку до $N_2 = 480.000 N_2 = 480.000$ твитов. Так как твиты на разных языках не могут быть схожи, можно разделить выборку на языки, которая представлена в таблице 1.1. Согласно [1] количество твитов на английском языке составляет около 70%. В нашей выборке их примерно 80%, поэтому можем рассматривать сообщения только на английском языке без больших потерь объёмов данных.

Таблица 1.1 – Количество сообщений на разных языках в процентах от общего количества

	Английский, %	Русский, %	Другие, %
Изначально	82.5	10	7.5
После удаления ретвитов	62	22	16

Согласно таблице 1.1 можем сократить выборку до $N_3 = 297.600 N_3 = 297.600$ для сообщений, написанных на английском языке. Получается, что из всей выборки для сравнения осталось около 35% всех сообщений.

Далее отсортируем выборку по длине твитов, итого получится 6 классов примерно по $N_4 = 50.000 N_4 = 50.000$ твитов в каждом.

Несмотря на значительное сокращение выборки, количество операций осталось довольно значительным, а именно $6 \cdot \sum_{i=1}^{N_4} i 6 \cdot \sum_{i=1}^{N_4} i$, где $N_4 = 50.000\ N_4 = 50.000$. Поэтому появляется необходимость максимально упростить тексты для обработки, в связи с чем были сделаны следующие изменения:

– Все буквы приведены к нижнему регистру.

– Удалены незначащие символы, такие как восклицательный знак, точка и т.д.

– Удалены все слова, меньшие трёх символов, что очищает сообщение практически от всех предлогов и других незначащих слов.

– Удалены все хэштэги, упоминания, ссылки.

Способы реализации. После сделанных преобразований сообщения все еще остаются в не очень удобной форме для сравнения, поэтому возникает необходимость использования хэшей.

Были выделены следующие способы сравнения:

– Криптографический хэш

– Перцептивный хэш

  – Фонетический хэш

  – Структурный хэш

Криптографический хэш. Данный тип хэша сжимает исходное сообщение, однако, при малейшем изменении текста самого сообщения, хэш меняется кардинально.

Таблица 1.2 – Сравнение хэшей двух исходных строк с использованием криптографического хэша MD5.

я люблю Беларусь	Я люблю Беларусь
90d677e3ac0af3773f10b5be08266e56	**cdfbadad1d75122148a74ef268f2acd0**

Из таблицы 1.2, становится видно, что полученные результаты сильно отличаются при небольшом изменении. Данные типы хэшей хорошо работают при поиске дубликатов, создавая одинаковые хэши для одинаковых сообщений, но оказываются не дееспособными при поиске схожих сообщений.

Перцептивный хэш. В отличие от криптографических хэшей перцептивный хэш генерируется на основе содержания объекта. Поэтому незначительная трансформация объекта не повлечёт сильное изменение самого хэша.

Перцептивный хэш разделяется на:

– Фонетический хэш

– Структурный хэш

Фонетический хэш.

Общим принципом для всех методов фонетического кодирования является то, что алгоритмы преобразовывают строку сообщения в хэш в соответствии с произношением данного текста в оригинальном языке. Естественно, этот процесс зависит от языка. Большинство методов, включая все представленные здесь, разрабатывались в основном для кодирования на английском языке, что применимо для решаемой проблемы, так как используется выборка на соответствующем языке. Несколько методов были адаптированы для других языков.

– Soundex. Алгоритм основан на том, что схожие по звучанию буквы определяются алгоритмом как одинаковые и заменяются на соответствующую цифру или остаются неизменными, если буква стоит в начале слова (напр. «Caffe» – C010). Каждой букве в соответствии с таблицей преобразования соотносится цифра от 0 до 6. Следовательно, существует около 5600 комбинаций результатов данного алгоритма и в среднем 200 различных слов будут определены как одинаковые.

– Phonex (Phonix). Phonex- это разновидность Soundex, которая улучшает качество кодирования путём предварительной обработки слов в соответствии с их английским произношением перед кодировкой (напр. «wr» в «r»). Phonix работает аналогично Soundex, однако, имеет другую таблицу преобразования, которая состоит из 9 цифр (напр. напр. «Caffe» – C070). Результат алгоритма принимает около 19.000 комбинаций, или одну комбинацию на 52 слова, что значительно улучшает качество работы.

Так как твиты – это короткие сообщения, поэтому фонетические хэши сильно не уменьшают длину сообщения, что делает их использование нецелесообразным.

Структурный хэш:

Структурный хэш используется в приближенном сравнении строк, который имеет широкое распространение. Выделяют следующие виды сравнения:

– Расстояние Левенштейна. Расстояние Левенштейна определяется как наименьшее количество операций редактирования (вставки, удаления и замены), необходимых для трансформации одной строки в другую. Расстояние (количество правок) между двумя строками $s_1 s_1$ и $s_2 s_2$ можно рассчитать за время $O(|s_1| \times |s_2|) O(|s_1| \times |s_2|)$. Функция расстояния Левенштейна $dist(s_1, s_2) dist(s_1, s_2)$ возвращает значение 0, если строки одинаковые или положительное количество правок, если они разные.

– Алгоритм Джаро. Алгоритм рассчитывает число cc общих символов и количество транспозиций tt. Мера подобия рассчитывается как:

$$jaro(s_1, s_2) = \frac{1}{3}\left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{t}\right)$$

– Алгоритм Дамерлау-Левенштейна. В этой вариации расстояния Левенштейна транспозиция также считается элементарной операцией редактирования (на расстоянии Левенштейна транспозиция соответствует двум правкам: одна вставка и одно удаление или две замены). Мера алгоритма рассчитывается аналогично $lev(s_1, s_2)lev(s_1, s_2)$.

– Алгоритм LCS или алгоритм наибольшей общей подстроки. Алгоритм в цикле ищет самую длинную общую подстроку в двух сравниваемых сообщениях до минимальной длины (обычно устанавливается в 2 или 3). Например, две строки «gail west» и «vest abigail» самая длинная общая подстрока «geil». После того, как она будет удалена, появятся две новые строки «west» и "vest abi". Во второй итерации удаляется подстрока «est», оставляя "w" и "v abi". Суммарная длина общих подстрок теперь 7. Мера рассчитывается по формуле:

–

$$lcs(s_1, s_2) = |s_1| + |s_2| - 2 \cdot |distLcs(s_1, s_2)|$$

где $distLcs(s_1, s_2)$ – функция, возвращающая расстояние LCS.

– Q-граммы. Q-граммы являются подстроками длины q в более длинных строках. Обычно используемые q-граммы - это униграммы (q = 1), биграммы (q = 2) и триграммами (q = 3). Например, «river» содержит биграммы «ri», «iv», «ve» и «er». Мера q-граммы рассчитывается путем подсчета количества общих q-грамм (т.е. q-грамм, содержащихся в обоих сообщениях) и разделенное либо на число q-грамм в более короткой строке, либо на число в более длинной строке, либо на среднее количество q-грамм в обоих твитах.

– Алгоритм косинусов. Алгоритм считает сходство посредством измерения косинуса угла между двумя векторами, где каждое сообщение представлено в виде вектора и рассчитывается по формуле:

–

$$dist(s_1, s_2) = \cos(s_1, s_2) = \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

– Нормализованное расстояние Левенштейна.

– Алгоритм Metric LCS. Нормализованное расстояние Левенштейна вычивляется аналогично расстоянию Левенштейна, за исключением того, что расстояние делится на длину большей строки и в разных вариациях возможно улучшить качество сравнения текстов, если давать различные коэффициенты редактирования для разных видов ошибок (напр. замене «w» и «e» давать меньший коэффициент из-за их близкого расположения на клавиатуре, чем «q» и «p»). Поэтому Нормализованное расстояние в отличие от обычного не является метрикой и вычисляется по формуле:

$$norm\_lev(s_1, s_2) = 1 - \frac{dist(s_1, s_2)}{\max(|s_1|, |s_2|)}$$

– Алгоритм SIFT4. SIFT4 - это экспериментальный алгоритм, который сочетает в себе признаки алгоритмов Винклера и LCS.

– Алгоритм подсчёта букв. Алгоритм представляет каждое сообщение в виде массива, в котором указаны количества появлений каждой буквы. Расстояние считается по формуле:

–

$$dist(s_1, s_2) = \frac{c}{\max(|s_1|, |s_2|)}$$

где $^{c\ c}$ – количество общих символов в двух строках

Временное и качественное сравнение. Для выбора алгоритма наиболее важным фактором является время его выполнение.

Таблица 1.3 – Время и сложность работы алгоритмов.

Алгоритм	Сложность	Время, мкс
Алгоритм Дамерлау-Левенштейна	$O(m*n)$	72
Расстояние Левенштейна	$O(m*n)$	10
Нормализованное расстояние Левенштейна	$O(m*n)$	11
N-граммы	$O(m*n)$	44
Алгоритм Metric LCS	$O(m*n)$	12
Алгоритм LCS	$O(m*n)$	11
Алгоритм Джаро	$O(m*n)$	12
Алгоритм Винклера	$O(m*n)$	12
Q-граммы	$O(m+n)$	23
Алгоритм косинусов	$O(m+n)$	7
Алгоритм SIFT4	$O(m+n)$	2
Алгоритм подсчета букв	$O(n)$	0.17

Необходимо также выяснить, какой из алгоритмов показывает наилучшие результаты при различных трансформациях сообщений. Результаты работы отображены в таблице 1.5 с условиями из таблицы 1.4.

Таблица 1.4 – Пределы изменения сообщений

Трансформация	Незначительно	Средне	Значительно
Удаление слов	1-2	3-5	5-10
Добавление слов	1-2	3-5	5-10
Замена слов	1-2	3-4	5-7
Изменение формы слов	1-10	10-20	20-30
Изменение порядка слов	1-3	4-6	7-12

Таблица 1.5 – Процент схожести исходного сообщений при значительных и незначительных изменениях (значительные изменения – верхняя часть ячейки, незначительные нижняя)

Алгоритм	Добавление слов	Удаление слов	Изменение порядка слов	Изменение форм слов	Замена слов
Алгоритм Дамерлау-Левенштейна	83	83	81	90	90
	34	43	44	82	61
Расстояние Левенштейна	84	85	81	90	90
	57	63	43	82	61
Нормализованное расстояние Левенштейна	84	80	78	89	89
	57	23	33	80	55
N-граммы	84	80	78	83	87
	56	23	23	75	52
Алгоритм Metric LCS	84	80	87	99	99
	57	23	49	88	59
Алгоритм LCS	83	83	77	99	84
	34	43	13	88	39
Алгоритм Джаро	86	82	99	87	89
	70	57	82	83	75
Алгоритм Винклера					
Q-граммы	79	79	89	50	64
	24	41	65	30	1
Алгоритм косинусов	90	88	99	73	81
	67	56	81	62	39
Алгоритм SIFT4	65		80	94	93
	1		53	85	64
Алгоритм подсчета букв	82	82	100	91	89
	52	31	100	83	56

Будем считать, что при добавлении, удалении и замене слов для сохранения схожести двух сообщений, необходимо, чтобы при незначительных изменениях подобие обоих твитов

было высоким, а при значительных – небольшим. При изменении порядка и форм слов при незначительных и значительных изменениях схожесть обоих сообщений должна быть высокой. Для данных условий наиболее подходящими являются алгоритмы косинусов, MLCS и нормализованное расстояние Левенштейна. В ходе эксперимента, наилучшие результаты по времени и точности показал алгоритм косинусов, который и был определен в качестве решения. Числовая разница между похожими строками и между абсолютно разными должна быть максимальной. Данное ограничение необходимо для избежания коллизий.
Немаловажным фактором при выборе алгоритма является процент ложных срабатываний. Количество коллизий было протестировано на 800 твитах или было произведено около 265.000 сравнений, результаты которых отображены в таблице 1.5.

Таблица 1.6 – Количество колиизий на выборке из 800 твитов.

Алгоритм	Количество коллизий
Алгоритм Дамерлау-Левенштейна	5000
Расстояние Левенштейна	5000
Нормализованное расстояние Левенштейна	7
Алгоритм Metric LCS	1
Алгоритм LCS	1500
Алгоритм Джаро	0
Q-граммы	800
Алгоритм косинусов	7
Алгоритм SIFT4	2000
Алгоритм подсчета букв	500

Как видно из таблиц, алгоритм подсчета букв является наиболее быстрым, но не совсем точным. Положим, что если сообщения являются схожими, то и количество одинаковых букв в них будет примерно одинаковым. Поэтому требуется дополнительный алгоритм для проверки возможных пар схожих сообщений. Наиболее подходящими алгоритмами являются алгоритм косинусов и нормализованное расстояние Левенштейна.

В кандидаты дополнительного алгоритма были выбраны алгоритм косинусов и нормализованное расстояние Левенштейна. Так как по параметрам качественного сравнения они схожи и на практике показали неплохие результаты, но алгоритм косинусов работает быстрее, поэтому ему и было отдано предпочтение.

Наконец, алгоритм состоит из следующих шагов:

– Удаляем все ретвиты.

– Разделяем сообщения на языки

– Производим преобразования строк для уменьшения количества сравниваемых символов

– Используем алгоритм подсчета букв

– Для схожих строк используем дополнительный алгоритм проверки на схожесть (напр. Алгоритм косинусов).

Заключение. После удаления ретвитов, разделения всей выборки на языки и сортировки по длине, удалось значительно сократить количество исходных данных. Также удаление из сообщений незначащих символов и слов получилось сократить время работы алгоритмов. Был использован наиболее быстрый алгоритм и алгоритм, проверяющий его на корректность, что позволило решить задачу за 7 минут 47 секунд с минимальным количеством коллизий.

Список литературы

[1] I. Bartolini, P. Ciaccia, and M. Patella. String matching with metric trees using an approximate distance. In SPIRE, LNCS 2476, pages 271–283, Lisbon, Portugal, 2002.

[2] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In Proceedings of ACM SIGKDD, pages 39–48, Washington DC, 2003.

[3] P. Christen and K. Goiser. Quality and complexity measures for data linkage and deduplication. In F. Guillet and H. Hamilton, editors, Quality Measures in Data Mining, Studies in Computational Intelligence. Springer, 2006.

[4] Peter Christen. A Comparison of Personal NameMatching: Techniques and Practical Issues. September 2006

[5] С.А. Самойленко, Д. Мейсона. Твиттер как разговор через контекст: от Образования 2.0 к Образованию 3.0.

USING PERCEPTUAL HASHING IN FINDING SIMILAR MESSAGES BY THE EXAMPLE OF SAMPLE FROM TWITTER

J.A. Zakharik
Student of BSU

D.M. Prokurat
Master of Mathematics and Information Technology

Mechanics and Mathematics Faculty of the Belarusian State University, Republic of Belarus
Belarusian State University, Republic of Belarus
E-mail: zakharik.julia@gmail.com

Abstract. In this paper, the problem of finding similar messages based on a selection from Twitter was considered. The following algorithms are considered: Levenshtein distance, normalized Levenshtein distance, Soundex, Phonex, Jaro algorithm, Q-grams, LCS algorithm, Metric LCS algorithm, SIFT4, Damerlau-Levenshtein algorithm, cosine algorithm and letter counting algorithm. There was a reduction in sampling and simplification of tweets. An optimal algorithm for solving the problem is proposed.

Keywords: perceptual hash, text comparison, cosine algorithm, Jaro's algorithm, twitter