Выступление

на тему

"Фонетические алгоритмы. Обзор и сравнение"
"Phonetic algorithms. Overview and comparison".

Автор: Карасик Андрей Максимович
Место учебы: Механико-математический факультет МГУ,
кафедра прикладной механики и управления,
лаборатория управления и навигации
TG, VK: @dthedog
GitHub: `https://github.com/AMKarasik`

СтудКон. Просто о сложном

31 октября 2022 г.

Содержание

1 Мотивация **3**

2 Исторический экскурс **3**

3 Базовые методы **4**
3.1 SoundEx 4
3.2 NYSIIS 5
3.3 Русский Metaphone 8

4 Фонетическое расстояние **9**
4.1 Расстояние Левенштейна 9
4.2 Расстояние на основе N-грамм 10
4.3 Расстояние Джаро 10

5 Применение фонетических алгоритмов в разных сферах **11**

1 Мотивация

Каждый из нас сталкивался когда-то с автозаменой в поисковой строке, с исправлением слов при отправке сообщения и тд. Для таких вещей (и не только) существуют фонетические алгоритмы. Попробуем разобраться с тем, как они работают, с их различиями, преимуществами и недостатками.

2 Исторический экскурс

С ростом количества информации всегда росло и количество ошибок, допущенных в ней, которые нужно искать и исправлять. Базы данных играют важную роль во многих организациях. И поиск совпадений записей в базе данных — постоянная и известная проблема на протяжении многих лет.

Один из методов улучшения качества данных использует вариации звука для обнаружения ошибочных данных с фонетическим соответствием. Техника получения слов с помощью звуков использовалась в США при переписи населения с конца 1890-х гг., но конкретное решение было запатентовано Робертом Расселом в 1912 году как алгоритм SoundEx. Позднее многие алгоритмы были разработаны на основе различных спецификаций и языковых ограничений.

Несмотря на появление и развитие новых моделей и методов распознавания речи, таких как вероятностные и статистические алгоритмы, нейронные сети, машинное обучение и др., алгоритмы фонетического кодирования не утратили своей актуальности, так как являются базовыми для применения этих моделей и методов на практике.

Также несмотря на все улучшения, общим недостатком алгоритмов фонетического кодирования все еще остается наличие в получаемых ложноположительных и ложноотрицательных результатах. Это объясняется тем, что эти алгоритмы работают не на последовательности элементарных звуков, из которых состоят слова, а на их текстовом представлении, которое искажено правилам написания того или иного естественного языка.

3 Базовые методы

Существует десятки фонетических алгоритмов, среди них SoundEx, NYSIIS, Metaphone, Caverphone, Daitch-Mokotoff, Polyphone и многие другие. Рассмотрим первые три.

3.1 SoundEx

В основном, этот алгоритм стал известен после того, как был опубликован в книге Дональда Кнута "Искусство программирования".

Этот алгоритм сопоставляет словам некий численный индекс. Принцип его работы основан на разбиении согласных букв на группы с порядковыми номерами, из которых затем и составляется результирующее значение.

Алгоритм:

1. Сохранить первую букву слова.
2. Удалить из слова буквы A, E, I, O, U, Y, W и H.
3. Заменить оставшиеся буквы на цифры следующим образом:

   b, f, p, v → 1

   c, g, j, k, q, s, x, z → 2

   d, t → 3

   l → 4

   m, n → 5

   r → 6
4. Если в коде имеются группа из одинаковых цифр, то заменить эту группу первой цифрой, за исключением цифр, разделяющих буквы W и H в исходном слове.
5. Сформировать результирующий код из первой буквы слова и трех первых цифр, полученный на предыдущих шагах. Если в коде менее трех цифр, то дополнить код нулями.

Примеры работы алгоритма:

	шаг 1	шаг 2	шаг 3	шаг 4	шаг 5
Robert & Rupert	R & R	brt & prt	163 & 163	-	R163 & R163
Andrei & Andrew	A & A	ndr & ndr	536 & 536	-	A536 & A536

Реализация алгоритма на Python:

```
def soundex_generator(word):
    # Convert the word to upper case for uniformity
    word = word.upper()

    soundex = ""

    # Retain the first letter
```

```
    soundex += word[0]

    # Create a dictionary which maps letters to respective soundex codes
    # Vowels and 'H', 'W' and 'Y' will be represented by '.'
    dictionary = {"BFPV": "1",
                  "CGJKQSXZ": "2",
                  "DT": "3",
                  "L": "4",
                  "MN": "5",
                  "R": "6",
                  "AEIOUHWY": ""}

    for char in word[1:]:
        for key in dictionary.keys():
            if char in key:
                code = dictionary[key]
                if code != '':
                    if code != soundex[-1]:
                        soundex += code

    # Trim or pad to make SoundEx a 4-character code
    soundex = soundex[:4].ljust(4, "0")

    return soundex
```

Первый недостаток алгоритма SoundEx состоит в том, что существуют близкие по звучанию слова, которые имеют разный кодирующий текст. Например, для слов ***Lee*** и ***Leigh***, имеющих одно и то же звучание, получаем различные коды: "L000" и "L200" соответственно.

Второй недостаток, обратный первому: существуют различные по звучанию слова, которые имеют одинаковый кодирующий текст. Например, слова ***Gauss*** и ***Ghosh*** с разным звучанием имеют код "G200".

Ну и главным недостатком оригинального алгоритма SoundEx является его низкая точность: например, на одно значение кода может приходиться более 20 фамилий.

Алгоритм SoundEx сильно зависит от языка. Поэтому разработано множество модификаций этого алгоритма для различных языков: русского, китайского, испанского, персидского, арабского и т.д. Благодаря своей простоте и низкой вычислительной сложности алгоритм SoundEx стал стандартом и встроен в механизм поиска почти всех известных систем управления базами данных.

3.2 NYSIIS

Разработанный в 1970 году как часть системы *New York State Identification and Intelligence System*, этот алгоритм дает несколько лучшие результаты относительно оригинального SoundEx, используя более сложные правила преобразования исходного слова в результирующий код. Этот алгоритм разработан для работы именно с американскими фамилиями.

Алгоритм:

1. Преобразование начала слова путем следующих подстановок:
   MAC → MCC;
   KN → N;
   K → C;
   PH, PF → FF;
   SCH → SSS.
2. Преобразование конца слова путем следующих подстановок:
   EE → Y;
   IE → Y;
   DT, RT, RD, NT, ND → D.
3. Преобразование слова в целом путем следующих подстановок:
   EV → AF;
   A, E, I, O, U → A;
   Q → G; Z → S;
   M → N;
   KN → N;
   K → C;
   SCH → SSS;
   PH → FF;
   W → A.
4. Удаление H после гласных и S, A в конце слова.
5. Преобразование суффикса слова путем подстановки AY → Y.
6. Ограничить полученный код 6 знаками.

Сравним SoundEx и NYSIIS. Видно, что не очень похожие фамилии в Soundex имеют один код, а вот в NYSIIS эти фамилии будут иметь разную запись. NYSIIS преобразует к одному и тому же коду немногим более двух фамилий.

	SoundEx	NYSIIS
Ackermann Azuron	A265 A265	ACARNAN ASARAN
Dyatlov Detlov	D341 D341	DYATLAV DATLAV
Nikonov Nozhnov Nesmeev	N251 N251 N251	NACANAV NASNAV NASNAF
Archipcev Archipychev Archipkov	A621 A621 A621	ARCAPCAF ARCAPYCAF ARCAPCAV

Реализация алгоритма на Python:

```
_vowels = 'AEIOU'


def replace_at(text, position, fromlist, tolist):
    for f, t in zip(fromlist, tolist):
        if text[position:].startswith(f):
            return ''.join([text[:position],
                            t,
                            text[position+len(f):]])
    return text


def replace_end(text, fromlist, tolist):
    for f, t in zip(fromlist, tolist):
        if text.endswith(f):
            return text[:-len(f)] + t
    return text


def nysiis(name):
    name = re.sub(r'\W', '', name).upper()
    name = replace_at(name, 0, ['MAC', 'KN', 'K', 'PH', 'PF', 'SCH'],
                               ['MCC', 'N',  'C', 'FF', 'FF', 'SSS'])
    name = replace_end(name, ['EE', 'IE', 'DT', 'RT', 'RD', 'NT', 'ND'],
                             ['Y',  'Y',  'D',  'D',  'D',  'D',  'D'])
    key, key1 = name[0], ''
    i = 1
    while i < len(name):
        n_1, n = name[i-1], name[i]
        n1_ = name[i+1] if i+1 < len(name) else ''
        name = replace_at(name, i, ['EV'] + list(_vowels), ['AF'] + ['A']*5)
        name = replace_at(name, i, 'QZM', 'GSN')
        name = replace_at(name, i, ['KN', 'K'], ['N', 'C'])
        name = replace_at(name, i, ['SCH', 'PH'], ['SSS', 'FF'])
        if n == 'H' and (n_1 not in _vowels or n1_ not in _vowels):
            name = ''.join([name[:i], n_1, name[i+1:]])
        if n == 'W' and n_1 in _vowels:
            name = ''.join([name[:i], 'A', name[i+1:]])
        if key and key[-1] != name[i]:
            key += name[i]
        i += 1
    key = replace_end(key, ['S', 'AY', 'A'], ['', 'Y', ''])
    return key1 + key
```

3.3 Русский Metaphone

Оригинальный Metaphone (1990) отличается от предыдущих алгоритмов немного иным подходом к процессу кодирования: он преобразует исходное слово с учетом правил английского языка, используя заметно более сложные правила, и при этом теряется значительно меньше информации. Русская версия же появилась в 2002 году.

Алгоритм:

1. Для всех гласных букв проделать следующие операции:

   ЙО, ИО, ЙЕ, ИЕ, Е, Ё, Э → И;

   О, Ы, Я → А;

   Ю → У.

2. Для всех согласных букв, за которыми следует любая согласная, кроме Л, М, Н или Р, либо же для согласных на конце слова, провести оглушение:

   Б → П;

   З → С;

   Д → Т;

   В → Ф;

   Г → К.

3. Удаление повторяющихся букв

4. Склеиваем ТС и ДС в Ц: ТС → Ц

5. Удаление букв Ъ, Ь и дефиса

Этот алгоритм преобразует к одному коду в среднем 1-2 фамилии. Примеры работы:

Витавский, Витовский → Витафский,
Пермаков, Пермяков, Перьмяков → Пирмакаф.

4 Фонетическое расстояние

Основным методом, реализуемым в рассмотренных ранее алгоритмах фонетического кодирования, является метод эквивалентных преобразований слова по звучанию, при котором часть слова, принадлежащая некоторому классу эквивалентности заменяются кодом этого множества или его типичным представителем. При этом заметно, что части слов из одного множества, близких по звучанию, также близки по написанию. При введении соответствующей метрики на словах можно поставить задачу определения схожести слов по звучанию путем подсчета расстояния между словами по написанию.

4.1 Расстояние Левенштейна

Метрика названа в честь великого советского математика, выпускника мехмата МГУ Владимира Иосифовича Левенштейна.

Расстояние Левенштейна (редакционное расстояние) — метрика сходства между двумя строковыми последовательностями. Чем больше расстояние, тем более различны строки. Для двух одинаковых последовательностей расстояние равно нулю. По сути, это минимальное число односимвольных преобразований (удаления, вставки или замены), необходимых, чтобы превратить одну последовательность в другую.

Расстояние Левенштейна активно используется для исправления ошибок в словах, поиска дубликатов текстов, сравнения геномов и прочих полезных операций с символьными последовательностями.

Расстояние Левенштейна Lev(i, j) между двумя словами a и b длиной |a|=i и |b|=j при min(i, j) = 0 по определению равно max(i, j), а при min(i, j) > 0 находится из следующего рекуррентного уравнения:

$$\mathrm{Lev}(i,j) = \min \begin{cases} \mathrm{Lev}(i, j-1) + 1, \\ \mathrm{Lev}(i-1, j) + 1, \\ \mathrm{Lev}(i-1, j-1) + m(i,j), \end{cases}$$

где $m(i,j) = 0$, если i-я буква слова a равна j-й букве слова b, и единице — в противном случае. Разберемся в данной формуле и попробуем составить матрицу, отвечающую это метрике для конкретного примера.

$D(i,j) =$

		л	е	с	т	н	и	ц	а
	0	1	2	3	4	5	6	7	8
к	1	1	2	3	4	5	6	7	8
о	2	2	2	3	4	5	6	7	8
л	3	2	3	3	4	5	6	7	8
е	4	3	2	3	4	5	6	7	8
с	5	4	3	2	3	4	5	6	7
н	6	5	4	3	3	3	4	5	6
и	7	6	5	4	4	4	3	4	5
ц	8	7	6	5	5	5	4	3	4
а	9	8	7	6	6	6	5	4	3

Lev(колесница, лестница) = 3

Реализация алгоритма на Python:

```
def lev_dist(str_1, str_2):
    n, m = len(str_1), len(str_2)
    if n > m:
        str_1, str_2 = str_2, str_1
        n, m = m, n

    current_row = range(n + 1)
    for i in range(1, m + 1):
        previous_row, current_row = current_row, [i] + [0] * n
        for j in range(1, n + 1):
           add, delete, change = previous_row[j] + 1, \
                                 current_row[j - 1] + 1, \
                                 previous_row[j - 1]
            if str_1[j - 1] != str_2[i - 1]:
                change += 1
            current_row[j] = min(add, delete, change)

    return current_row[n]
```

4.2 Расстояние на основе N-грамм

N-граммой называется последовательность из N элементов (букв).

Для определения фонетической близости двух слов подсчитывается число общих N-грамм. Обычно N принимается равным трем. Рассмотрим на конкретном примере. Два слова, которые имеют одинаковое звучание: ***Thomson*** и ***Thompson***. Разобьем эти слова на триграммы. В результате получим, что слово ***Thomson*** включает триграммы THO, HOM, OMS, MSO и SON, а слово ***Thompson*** — THO, HOM, OMP, MPS, PSO и SON. Общими триграммами этих слов являются триграммы THO, HOM и SON. Доля общих триграмм составляет 3/6, что определяет расстояние между словами, равное трем (подсчет велся относительно слова с большим числом триграмм).

Очевидно также, что вычисление расстояния между словами на основе N-грамм дает лучший результат для более длинных слов, чем для коротких.

Обычно N-граммы используются для нечеткого сравнения слов, которое не затрагивает фонетических аспектов. Например, для идентификации языка, так как установлено, что на достаточно длинных текстах каждый язык имеет свое распределение N-грамм, для сжатия текстов, для "угадывания" следующих букв и т.п.

4.3 Расстояние Джаро

Неформальное определение расстояния Джаро между двумя словами — это минимальное число однобуквенных изменений, которое необходимо выполнить для преобразования одного слова в другое. Чем меньше расстояние Джаро, тем более схожи сравниваемые слова.

Расстояние Джаро $D(a, b)$ между двумя словами a и b при определяется так:

$$\mathrm{D}(i,j) = \begin{cases} 0 & m = 0 \\ w_1 \frac{m}{|a|} + w_2 \frac{m}{|b|} + w_3 \frac{m-t}{m} & m > 0 \end{cases}$$

где w_1, w_2, и w_3 — весовые коэффициенты, $w_1 + w_2 + w_3 = 1$;

m — число совпадающих букв (число букв, разнесенных не более чем на половину длины самого короткого слова);

t — половина числа транспозиций (половина числа совпадающих букв, отличающихся порядковыми номерами).

Рассмотрим в качестве примера два имени: ***Marik*** и ***Karim***. Число совпадающих символов: $m = 5$, половина транспозиций: $t = 2/2 = 1$, $D = \frac{2}{3} + \frac{4}{5} \cdot \frac{1}{3} = \frac{14}{15}$. При округлении $d = 1$ видим, что, действительно, всего при помощи одной транспозиции между первой и последней буквой получить одинаковые имена.

5 Применение фонетических алгоритмов в разных сферах

Возможно, из всех вышеприведенных алгоритмов и примеров могло показаться, что применение ограничено исправлением слов или поиском схожих фамилий, но это далеко не так.

Приведу лишь часть списка, где могут использоваться фонетические алгоритмы:

- для устранения дублирования данных;
- для фонетического поиска;
- для распознавания этнической принадлежности;
- в поисковых сервисах сети интернет;
- для первичной обработки голосовых запросов;
- для ускоренного ввода текста с клавиатуры;
- для исправления орфографических ошибок;
- для работы с последовательностями ДНК.

И подробнее рассмотрим одно из применений, а именно, как алгоритм Левенштейна помогает генетикам.

ДНК (Дезоксирибонуклеиновая кислота) — некоторая молекула, в которой хранится информация о генетическом материале. Эта молекула состоит из повторяющихся блоков — нуклеотидов. В ДНК встречается четыре вида азотистых оснований (аденин (A), гуанин (G), тимин (T) и цитозин (C)).

ДНК записывается в виде некой последовательности азотистых оснований нуклеотидов (A, G, T, C), например:

ATTCAAAAGACCTCGCTAAAAATCTCGCAGTCAACTATCTTTAGCGTTAAATCACGCAA
CATATTTCAACCGCATTGGAGAGTCGAGGCAGCTAAGCCCGGTAACCCCTTTCATATCTGA

TCCTACGGGATCTTGGGTTTGTCCGCCATTCTGATTGTGAGAACGGGGTGTGTCCGCAGA
ACCCCTCTCTAGACAACTAGACCATTCGACTCAG

Попробуем найти расстояние Левенштейна для двух ДНК-последовательностей, чтобы посмотреть, насколько они похожи и отличаются.

A	T	C	A	A	G	G	G	A	C	C
+	+	+	-	-	-	-	-	+	-	+
A	T	C	G	C	A	A	T	A	G	C

Видим, что расстояние Левенштейна равно 6.

Таким образом, если уже имеется некая база данных с последовательностями ДНК и требуется проверить, является ли новая ДНК похожей на что-то, самый простой вариант — просто найти минимум расстояний Левенштейна с другими последовательности.