|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Лабораторная работа № 1 | | |
| по дисциплине «Программные системы статистического анализа» | | |
| **.** | | |
|  | | |
|  | Вариант: 52 | ПМИМ-51 БуданцеВ дмитрий |
| . |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Преподаватели | Тимофеева Анастасия Юрьевна |
|  |  |
| Новосибирск,2025 | | |

1. Постановка задачи

Цель

Сравнение метрик, реализованных для вычисления расстояния между строками в stringdist.

Задачи

**Часть 1**: Базовое сравнение на простых примерах

**Часть 2**: Производительность метрик

1. Теоретическая часть
2. Введение

Метрики расстояния меся фундаментальным инструментом в обработке естественного языка, анализе текстов и очистке данных. Они позволяют количественно оценить степень различия между двумя строковыми последовательностями, что особенно важно для задач поиска похожих строк, исправления опечаток и обнаружение дубликатов.

1. Расстояние Левенштейна
2. Определение

Расстояние Левенштейна - это минимальное количество операций редактирования, необходимых для преобразования одной строки в другую.

Формальное определение:

Для двух строк и длиной и соответственно, расстояние Левенштейна вычисляется по формуле:

где:

– расстояние между первыми  символами строки  и первыми  символами строки

– индикаторная функция, равная 0 если символы совпадают, и 1 в противном случае.

1. Операции редактирования

Расстояние Левенштейна учитывает три базовые операции:

1. **Вставка** - добавление символа

- Пример: "кот" -> "кот**а**"

2. **Удаление** - удаление символа

- Пример: "кот**а**" -> "кот"

3. **Замена** - замена одного символа на другой

- Пример: "кот" -> "к**а**т"

1. Пример вычисления

Для строк "kitten" и "sitting":

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | S | I | T | T | I | N | G |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| K | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| I | 2 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| T | 3 | 3 | 2 | 1 | 2 | 3 | 4 | 5 |
| T | 4 | 4 | 3 | 2 | 1 | 2 | 3 | 4 |
| E | 5 | 5 | 4 | 3 | 2 | 2 | 3 | 4 |
| N | 6 | 6 | 5 | 4 | 3 | 3 | 2 | **3** |

Расстояние: 3

1. Расстояние Дамерау-Левенштейна
2. Определение

Расстояние *Дамерау-Левенштейна* является расширением классического расстояния Левенштейна, добавляя четвёртую операцию - **транспозицию** (перестановку двух соседних символов)

Формальное определение:

Для двух строк и длиной m и n соответственно, расстояние Левенштейна вычисляется по формуле:

1. Дополнительная операция

4. Транспозиция - перестановка двух соседних символов

- Пример: "**ab**cd" -> "**ba**cd"

1. Пример вычисления

Для строк "ca" и "abc"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | a | b | c |
|  | 0 | 1 | 2 | 3 |
| c | 1 | 1 | 2 | 2 |
| a | 2 | 1 | 2 | **3** |

Расстояние: 3

Примечание: Если попробовать получить результат от **stringdist**, то результат будет отличаться. Дело в том, что данный алгоритм реализован, через оптимальное расстояние выравнивания струны, а не через расстояние с соседними транспозициями из-за чего значение отличается от истинного.

1. Расстояние Хэмминга
2. Определение

Для двух строк одинаковой длины расстояние Хэмминга равно количеству позиции, в которых соответствующие символы различаются.

где – индикаторная функция, равная 1 если символы различны, и 0 в противном случае.

1. Расстояние Джакарда
2. Определение

Строки представляются как множества n-грамм (обычно биграмм и триграмм).

Формула коэффициента Джакарда:

Расстояние Джакарда:

где:

- множества n-грамм строк

- мощность пересечения множеств

- мощность объединения множеств

1. Пример вычисления

Для строк и с триграммами :

триграммы:

триграммы:

1. Параметр q и его влияние

Параметр определяет размер n-грамм:

Малые (1-2): лучше для коротких строк, чувствительно к опечаткам

Большие (3-4): лучше для длинных текстов, устойчивее к шуму

1. Косинусное расстояние
2. Определение

**Косинусное расстояние** основано на концепции векторного пространства и измеряет угол между векторами, представляющими строки.

Математическая формула:

где:

- – скалярное произведение векторов

- – нормы векторов

- – угол между векторами

1. Векторное представление строк

Строки представляются как векторы в пространстве n-грамм:

Каждая dimension соответствует уникальной n-грамме

Значения – частоты или бинарные индикаторы наличия n-грамм

Пример для "hello" (биграммы):

1. Q-граммное расстояние
2. Определение

**Q-граммное расстояние** измеряет разность между множествами n-грамм двух строк.

Формула:

где – множество q-грамм строки .

1. Расстояние Jaro
2. Алгоритм вычисления

**Шаг 1**: Поиск соответствующих символов

- Символы считаются соответствующими, если они одинаковы и находятся в пределах позиций

**Шаг 2**: Подсчет транспозиций

- Транспозиции - это соответствующие символы в разном порядке

**Формула Jaro**:

где:

* + – количество совпадающих символов
  + – половина количества транспозиций

1. Расстояние Jaro-Winkler
2. Формула вычисления

где:

* – расстояние **Jaro**
* – длина общего префикса (максимум 4 символа)
* – фактор масштаба (обычно 0.1)

1. Расстояние LCS
2. Определение

**Расстояние на основе самой длинной общей подстроки**(Longest Common Substring distance) – это метрика, которая измеряет различие между двумя строками на основе длины их наибольшей общей непрерывной подпоследовательности.

**Формальное определение:**

Для двух строк и , расстояние вычисляется как:

где:

- – длины строк и

-  – длина самой длинной общей подстроки

1. Пример вычисления

Для строк и :

**Шаг 1:** Поиск LCS

- Общие подстроки: (длина 2), (длина 2)

- Longest Common Substring: или (длина 2)

**Шаг 2:** Вычисление расстояния

В **stringdist** другая интерпретация, поэтому результат другой.

1. Практическая часть

Часть 1: Базовое сравнение на простых примерах

**Таблица 1.** Результаты сравнения метрик расстояния

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pair** | **L** | **DL** | **H** | **J Q2** | **J Q3** | **CosQ2** | **CosQ3** | **QGQ2** | **QGQ3** | **Jaro** | **JaroW** | **LCS** |
| **"kitten" "sitting"** | 3 | 3 | NA | 0.778 | 0.875 | 0.635 | 0.776 | 7 | 7 | 0.254 | 0.254 | 5 |
| **"hello" "hell"** | 1 | 1 | NA | 0.25 | 0.333 | 0.134 | 0.184 | 1 | 1 | 0.067 | 0.04 | 1 |
| **"abc" "acb"** | 2 | 1 | 2 | 1 | 1 | 1 | 1 | 4 | 2 | 0.444 | 0.4 | 2 |
| **"book" "back"** | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 6 | 4 | 0.333 | 0.3 | 4 |
| **"data" "date"** | 1 | 1 | 1 | 0.5 | 0.667 | 0.333 | 0.5 | 2 | 2 | 0.167 | 0.117 | 2 |
| **"martha" "marhta"** | 2 | 1 | 2 | 0.75 | 0.857 | 0.6 | 0.75 | 6 | 6 | 0.056 | 0.039 | 2 |
| **"example" "samples"** | 3 | 3 | 7 | 0.5 | 0.571 | 0.333 | 0.4 | 4 | 4 | 0.19 | 0.19 | 4 |
| **"programming" "program"** | 4 | 4 | NA | 0.4 | 0.444 | 0.225 | 0.255 | 4 | 4 | 0.121 | 0.073 | 4 |
| **"identical" "identical"** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **"completely" "different"** | 8 | 8 | NA | 1 | 1 | 1 | 1 | 17 | 15 | 0.567 | 0.567 | 15 |

**Таблица 2.** Матрица корреляции между метриками

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **L** | **DL** | **H** | **J Q2** | **J Q3** | **CosQ2** | **CosQ3** | **QGQ2** | **QGQ3** | **Jaro** | **JaroW** | **LCS** |
| **L** | 1 | 0.875 | 0.853 | 0.638 | 0.652 | 0.526 | 0.558 | 0.773 | 0.739 | 0.488 | 0.530 | 0.857 |
| **DL** | 0.875 | 1 | 0.906 | 0.383 | 0.400 | 0.279 | 0.291 | 0.552 | 0.554 | 0.368 | 0.417 | 0.943 |
| **H** | 0.853 | 0.906 | 1 | 0.163 | 0.181 | 0.050 | 0.060 | 0.377 | 0.472 | 0.192 | 0.261 | 0.731 |

Продолжение таблицы

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **L** | **DL** | **H** | **J Q2** | **J Q3** | **CosQ2** | **CosQ3** | **QGQ2** | **QGQ3** | **Jaro** | **JaroW** | **LCS** |
| **J Q2** | 0.638 | 0.383 | 0.163 | 1 | 0.983 | 0.976 | 0.995 | 0.846 | 0.566 | 0.802 | 0.783 | 0.613 |
| **J Q3** | 0.652 | 0.400 | 0.181 | 0.983 | 1 | 0.923 | 0.976 | 0.850 | 0.626 | 0.739 | 0.708 | 0.626 |
| **CosQ2** | 0.526 | 0.279 | 0.050 | 0.976 | 0.923 | 1 | 0.981 | 0.773 | 0.428 | 0.842 | 0.828 | 0.523 |
| **CosQ3** | 0.558 | 0.291 | 0.060 | 0.995 | 0.976 | 0.981 | 1 | 0.820 | 0.529 | 0.786 | 0.760 | 0.542 |
| **QGQ2** | 0.773 | 0.552 | 0.377 | 0.846 | 0.850 | 0.773 | 0.820 | 1 | 0.897 | 0.431 | 0.440 | 0.720 |
| **QGQ3** | 0.739 | 0.554 | 0.472 | 0.566 | 0.626 | 0.428 | 0.529 | 0.897 | 1 | 0.045 | 0.056 | 0.633 |
| **Jaro** | 0.488 | 0.368 | 0.192 | 0.802 | 0.739 | 0.842 | 0.786 | 0.431 | 0.045 | 1 | 0.994 | 0.517 |
| **JaroW** | 0.530 | 0.417 | 0.261 | 0.783 | 0.708 | 0.828 | 0.760 | 0.440 | 0.056 | 0.994 | 1 | 0.544 |
| **LCS** | 0.857 | 0.943 | 0.731 | 0.613 | 0.626 | 0.523 | 0.542 | 0.720 | 0.633 | 0.517 | 0.544 | 1 |

**Вывод:**

Исходя из преведённых таблиц можно заметить, что пары алгоритмов: (*Levenshtein, Damerau-Levenshtein, LCS*), (*Jaccard, Cossine, Q-gram*), (*Jaro, Jaro-Winkler*) имеют между друг другом высокую корреляцию. Для идентичных строк ("identical - identical") все метрики корректно возвращают 0. Для полностью различных строк ("completely" "different") метрики достигают максимумов. Для строк разной длины Hamming не применим.

Часть 2: Производительность метрик

Таблица 3. Зависимость времени выполнения (мс) от длины строк

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Len | L | DL | H | J2 | J3 | Cos2 | Cos3 | Qgr2 | Qgr3 | Jaro | J-Wink | LCS |
| 64 | 0,1 | 0,29 | 0,13 | 0,21 | 0,21 | 0,2 | 0,21 | 0,21 | 0,21 | 0,14 | 0,14 | 0,2 |
| 128 | 0,36 | 0,71 | 0,13 | 0,22 | 0,22 | 0,22 | 0,22 | 0,21 | 0,22 | 0,15 | 0,16 | 0,24 |
| 256 | 0,69 | 2,74 | 0,14 | 0,31 | 0,33 | 0,3 | 0,33 | 0,31 | 0,32 | 0,19 | 0,18 | 0,54 |
| 512 | 2,48 | 11,3 | 0,14 | 0,49 | 0,54 | 0,49 | 0,54 | 0,48 | 0,53 | 0,3 | 0,31 | 2,01 |
| 1024 | 22,27 | 51,74 | 0,15 | 0,87 | 1,04 | 0,86 | 1,03 | 0,85 | 1,03 | 0,74 | 0,74 | 22,53 |
| 2048 | 95,7 | 214,94 | 0,16 | 1,49 | 2,11 | 1,47 | 2,09 | 1,46 | 2,09 | 2,4 | 2,4 | 95,86 |
| 4096 | 476,28 | 902,2 | 0,19 | 2,63 | 4,58 | 2,6 | 4,55 | 2,61 | 4,53 | 8,93 | 8,94 | 487,09 |

Таблица 4. Анализ роста времени выполнения

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Time (64), мс** | **Time (4096), мс** | **Сложность** |
| *Levenshtein* | 0,1 | 476,28 | O(n^2,01) |
| *Damerau-L* | 0,29 | 902,2 | O(n^1,91) |
| *Hamming* | 0,13 | 0,19 | O(n^0,09) |
| *Jaccard2* | 0,21 | 2,63 | O(n^0,6) |
| *Jaccard3* | 0,21 | 4,58 | O(n^0,73) |
| *Cos2* | 0,2 | 2,6 | O(n^0,61) |
| *Cos3* | 0,21 | 4,55 | O(n^0,73) |
| *Qgram2* | 0,21 | 2,61 | O(n^0,6) |
| *Qgram3* | 0,21 | 4,53 | O(n^0,73) |
| *Jaro* | 0,14 | 8,93 | O(n^0,99) |
| *J-Winkler* | 0,14 | 8,94 | O(n^0,99) |
| *LCS* | 0,2 | 487,09 | O(n^1,85) |

Примечание: В последней строке указано среднее значение.

Таким образом, алгоритмы Hamming, Jaccard2, Jaccard3, Cos2, Cos3, Qgram2, Qgram3, Jaro, Jaro-Winkler имеют линейную сложность, что соответвует теоретическим предположениям. Алгоритмы Levenshtein, Damerau-Levenshtein, LCS имеют квадратичную сложность, что так же соответсвует теоретическим предположениям.

1. Заключение

Полученные результаты позволяют осознано выбирать оптимальные метрики для конкретных практических задач, балансируя между точностью, производительностью и интерпретируемостью результатов.

Исследование показало, что для успешного применения метрик расстояния требует понимания их математических основ, областей применения и ограничений, а также тщательного учета характеристик конкретных данных и требований задачи.

1. Приложение

Программа **baseCase.R**:

# Установка зеркала CRAN

options(repos = c(CRAN = "https://cloud.r-project.org/"))

# Функция для установки и загрузки пакетов

install\_and\_load <- function(package) {

  if (!require(package, character.only = TRUE, quietly = TRUE)) {

    install.packages(package, dependencies = TRUE)

    library(package, character.only = TRUE)

  }

}

# Пытаемся установить только stringdist

tryCatch({

  install\_and\_load("stringdist")

  install\_and\_load("dplyr")

  install\_and\_load("knitr")

}, error = function(e) {

  cat("Ошибка при установке stringdist:", e$message, "\n")

  cat("Использую базовые функции.\n")

})

# Функция для сравнения метрик

compare\_metrics <- function(string1, string2, pair\_name = NULL) {

  if (is.null(pair\_name)) {

    pair\_name <- paste0('"', string1, '" - "', string2, '"')

  }

  results <- data.frame(

    pair = pair\_name,

    levenshtein = stringdist(string1, string2, method = "lv"),

    damerau\_levenshtein = stringdist(string1, string2, method = "dl"),

    hamming = ifelse(nchar(string1) == nchar(string2),

                    stringdist(string1, string2, method = "hamming"),

                    NA),

    jaccard\_q2 = stringdist(string1, string2, method = "jaccard", q = 2),

    jaccard\_q3 = stringdist(string1, string2, method = "jaccard", q = 3),

    cosine\_q2 = stringdist(string1, string2, method = "cosine", q = 2),

    cosine\_q3 = stringdist(string1, string2, method = "cosine", q = 3),

    qgram\_q2 = stringdist(string1, string2, method = "qgram", q = 2),

    qgram\_q3 = stringdist(string1, string2, method = "qgram", q = 3),

    jaro = stringdist(string1, string2, method = "jw", p = 0),

    jaro\_winkler = stringdist(string1, string2, method = "jw", p = 0.1),

    lcs = stringdist(string1, string2, method = "lcs"),

    stringsAsFactors = FALSE

  )

  return(results)

}

# Тестовые пары строк

test\_pairs <- list(

  c("kitten", "sitting"),

  c("hello", "hell"),

  c("abc", "acb"),

  c("book", "back"),

  c("data", "date"),

  c("martha", "marhta"),

  c("example", "samples"),

  c("programming", "program"),

  c("identical", "identical"),

  c("completely", "different")

)

# Создаем имена для пар

pair\_names <- sapply(test\_pairs, function(x) paste0('"', x[1], '" - "', x[2], '"'))

# Сравниваем все пары

all\_results <- data.frame()

for (i in 1:length(test\_pairs)) {

  result <- compare\_metrics(test\_pairs[[i]][1], test\_pairs[[i]][2], pair\_names[i])

  all\_results <- rbind(all\_results, result)

}

# Выводим результаты в виде таблицы

cat("Сравнение метрик расстояния между строками\n")

cat("==========================================\n\n")

kable(all\_results, digits = 3, caption = "Результаты сравнения метрик расстояния")

correlation\_matrix <- all\_results %>%

  select(-pair) %>%

  cor(use = "complete.obs")

kable(correlation\_matrix, digits=3, caption="Матрица корреляции между метриками")

cat("\nАнализ завершен!\n")

Программа **efficiency.R**:

# Установка зеркала CRAN

options(repos = c(CRAN = "https://cloud.r-project.org/"))

# Функция для установки и загрузки пакетов

install\_and\_load <- function(package) {

  if (!require(package, character.only = TRUE, quietly = TRUE)) {

    install.packages(package, dependencies = TRUE)

    library(package, character.only = TRUE)

  }

}

# Устанавливаем необходимые пакеты

tryCatch({

  install\_and\_load("stringdist")

  install\_and\_load("microbenchmark")

}, error = function(e) {

  cat("Ошибка при установке пакетов:", e$message, "\n")

})

# Функция для генерации случайной строки заданной длины

generate\_random\_string <- function(length) {

  paste0(sample(c(letters, LETTERS, " "), length, replace = TRUE), collapse = "")

}

# Функция для тестирования производительности одной метрики

benchmark\_metric <- function(string1, string2, method, q = NULL, p = NULL) {

  if (!is.null(q) && !is.null(p)) {

    result <- microbenchmark(

      stringdist(string1, string2, method = method, q = q, p = p),

      times = 50

    )

  } else if (!is.null(q)) {

    result <- microbenchmark(

      stringdist(string1, string2, method = method, q = q),

      times = 50

    )

  } else if (!is.null(p)) {

    result <- microbenchmark(

      stringdist(string1, string2, method = method, p = p),

      times = 50

    )

  } else {

    result <- microbenchmark(

      stringdist(string1, string2, method = method),

      times = 50

    )

  }

  return(median(result$time) / 1e6) # Возвращаем медианное время в миллисекундах

}

# Функция для тестирования всех метрик на одной паре строк

benchmark\_all\_metrics <- function(string1, string2, length) {

  results <- data.frame(

    length = length,

    levenshtein = benchmark\_metric(string1, string2, "lv"),

    damerau\_levenshtein = benchmark\_metric(string1, string2, "dl"),

    hamming = ifelse(nchar(string1) == nchar(string2),

                    benchmark\_metric(string1, string2, "hamming"),

                    NA),

    jaccard\_q2 = benchmark\_metric(string1, string2, "jaccard", q = 2),

    jaccard\_q3 = benchmark\_metric(string1, string2, "jaccard", q = 3),

    cosine\_q2 = benchmark\_metric(string1, string2, "cosine", q = 2),

    cosine\_q3 = benchmark\_metric(string1, string2, "cosine", q = 3),

    qgram\_q2 = benchmark\_metric(string1, string2, "qgram", q = 2),

    qgram\_q3 = benchmark\_metric(string1, string2, "qgram", q = 3),

    jaro = benchmark\_metric(string1, string2, "jw", p = 0),

    jaro\_winkler = benchmark\_metric(string1, string2, "jw", p = 0.1),

    lcs = benchmark\_metric(string1, string2, "lcs"),

    stringsAsFactors = FALSE

  )

  return(results)

}

# Запускаем бенчмаркинг с кратно увеличивающейся длиной строк

cat("ЗАПУСК АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ С КРАТНЫМ УВЕЛИЧЕНИЕМ ДЛИНЫ\n")

cat("============================================================\n\n")

# Длины строк: от 2^3 до 2^10 (кратное увеличение)

lengths <- 2^(6:12)  # 8, 16, 32, 64, 128, 256, 512, 1024 символов

performance\_results <- data.frame()

for (len in lengths) {

  cat("Тестирование для длины:", len, "символов...\n")

  # Генерируем две случайные строки одинаковой длины

  str1 <- generate\_random\_string(len)

  str2 <- generate\_random\_string(len)

  # Добавляем некоторые различия во вторую строку

  if (len > 10) {

    # Заменяем некоторые символы для создания различий

    positions <- sample(1:len, min(5, len %/% 10))

    for (pos in positions) {

      substr(str2, pos, pos) <- sample(letters, 1)

    }

  }

  result <- benchmark\_all\_metrics(str1, str2, len)

  performance\_results <- rbind(performance\_results, result)

  cat("Завершено для длины", len, "символов\n")

  cat("Пример строк: \"", substr(str1, 1, 20), "...\" vs \"", substr(str2, 1, 20), "...\"\n", sep = "")

  cat(paste0(rep("-", 60), collapse = ""), "\n\n")

}

# Вывод результатов

cat("РЕЗУЛЬТАТЫ АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ\n")

cat("=====================================\n\n")

# Таблица с результатами для всех длин

cat("Зависимость времени выполнения (мс) от длины строк:\n\n")

cat("Длина |  Lev  | D-Lev |  Ham  | Jac2  | Jac3  | Cos2  | Cos3  | Qgr2  | Qgr3  |  Jaro | J-Wink|  LCS  \n")

cat(paste0(rep("-", 110), collapse = ""), "\n")

for (i in 1:nrow(performance\_results)) {

  cat(sprintf("%5d | %5.2f | %5.2f | %5s | %5.2f | %5.2f | %5.2f | %5.2f | %5.2f | %5.2f | %5.2f | %5.2f | %5.2f\n",

              performance\_results$length[i],

              performance\_results$levenshtein[i],

              performance\_results$damerau\_levenshtein[i],

              ifelse(is.na(performance\_results$hamming[i]), "  NA", sprintf("%5.2f", performance\_results$hamming[i])),

              performance\_results$jaccard\_q2[i],

              performance\_results$jaccard\_q3[i],

              performance\_results$cosine\_q2[i],

              performance\_results$cosine\_q3[i],

              performance\_results$qgram\_q2[i],

              performance\_results$qgram\_q3[i],

              performance\_results$jaro[i],

              performance\_results$jaro\_winkler[i],

              performance\_results$lcs[i]))

}