ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

«ВЕКТОРИЗАЦИЯ JSON-ОБЪЕКТОВ НА ОСНОВАНИИ ЭТАЛОНА С ВЫЧИСЛЕНИЕМ МЕР СХОДСТВА ИХ ПОЛЕЙ»

Выпускная квалификационная работа бакалавра

Направление 010300

Фундаментальная информатика и информационные технологии

Выполнена студентом

Князевым Сергеем Викторовичем

Санкт-Петербург

2020

Содержание

Введение

Последние годы стремительными темпами набирают популярность сетевые технологии. Облачные вычисления, микросервисы, Интернет Вещей - термины, которые на слуху даже у людей, не связанных с IT. В связи с этим стал крайне популярен формат данных, используемый в web-приложениях - JavaScript Object Notation, или же JSON. JSON используется при обмене данных как между браузером и сервером при помощи технологии AJAX, так и между различными серверами. В этом формате можно представить практически любую структуру данных, будь то сведения о каком-то товаре, информация о человеке в социальной сети или набор характеристик некой системы. Но, как и многие понятные человеку системы, JSON плохо подходит для компьютерного анализа, из-за чего данные в этом формате зачастую необходимо преобразовывать в соответствии с конкретной задачей.

В рамках данной работы рассматривается способ преобразования подобных данных в многомерные векторы, которые лучше поддаются обработке и не требуют Ad-Hoc[1] решений. В частности, предполагается преобразование в вектор размерности, равной количеству свойств у рассматриваемого объекта. Предлагается ввести некоторую меру сходства для сравнения соответствующих друг другу полей разных объектов, и на основании этой меры определять координаты получившегося вектора.

[1] (лат.) “Для этого случая”

Постановка задачи

Целью данной работы является рассмотрение способа векторизации JSON-объектов на основании эталона. Для этого вычисляются меры сходства полей объектов с соответствующими полями эталона, затем вычисленные значения используются в качестве координат результирующего вектора. При написании работы были поставлены следующие задачи:

1. Рассмотреть варианты сравнения свойств и различные меры сходства, а также проверить, как они показывают себя на практике;
2. Разработать программный продукт, позволяющий производить векторизацию набора однородных объектов основываясь на сделанных выводах.

Глава 1. Сравнение свойств и обзор мер сходства.

Сравнение различных типов данных

Свойства объектов JSON состоят из двух частей: ключа и значения. В то время, как ключ всегда представляет собой строку, значение может иметь один из следующих типов[1]:

1. Вложенный объект;
2. Массив из значений;
3. Строка;
4. Число;
5. Логическое true/false;
6. null - отсутствие значения.

Соответственно, при сравнении двух свойств разных объектов с одинаковыми ключами, мы можем сравнивать значение, принадлежащее одному из этих типов со значением, принадлежащем любому другому (или такому же) типу данных.

Очевидно, что в большинстве случаев сравнение данных разных типов не имеет смысла; тогда мы принимаем меру сходства равной 0. Далее перечислены пары типов данных, мера сходства которых рассматривается в данной работе, с кратким её описанием, а также особый случай сравнения null со значением любого другого типа.

Null

Поскольку null обозначает отсутствие значения, то мы можем заключить, что при сравнении null и null они равны (мера сходства равна 1), в то время как при сравнении его с любым другим типом данных мы считаем меру сходства равной нулю. Например, если у двух сравниваемых объектов товаров отсутствует поле “цвет”, то мы полагаем, что по данному критерию между ними нет различий. И, наоборот, если это свойство присутствует только у одного из них, разница принципиальна.

Также стоит заметить, что при сравнении двух объектов мы считаем любые свойства, которые отсутствуют в структуре одного из объектов, но присутствуют в структуре другого, равными null для первого.

Строка и число, строка и true/false

Если строку можно преобразовать во второй тип без потери информации, то производится это преобразование и сравнивается подобное с подобным. Иначе второе значение приводится к строчному типу данных и сравнивается с первым.

Сравнение данных одного типа

Несмотря на то, что данные принадлежат к одному типу данных, их сравнение и вычисление меры сходства может быть нетривиальной задачей. Далее перечислены возможные сравнения однотипных данных и то, каким образом было решено их сравнивать.

Объект

Учитывая контекст работы, логично в качестве меры сходства представить величину, вычисляемую при помощи векторных представлений сравниваемых объектов. В данной работе в качестве такой меры рассматривается косинусное сходство, вычисляемое следующим образом[2]:

$$similarity = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$$, где a и b - векторы размерности n.

Косинусное сходство было выбрано среди различных вариантов представления разницы между векторами в том числе по причине того, что оно представляет собой именно сходство, а не разницу. Также контекст сравнения практически инвалидирует два основных недостатка косинусного сходства, выделяемые авторами [3]:

1. Координаты с большими значениями слишком сильно влияют на меру:

Учитывая, что все координаты сравниваемых векторов - меры сходства, принадлежащие промежутку от 0 до 1, сравниваемые значения крайне редко будут отличаться больше, чем на порядок.

2. Мера не учитывает координаты, которые у обоих векторов равны нулю:

Как следует из постановки задачи, каждая координата рассматриваемых векторов - мера сходства пары свойств. Следовательно, если оба свойства не

похожи на некое третье, это ничего не говорит нам о схожести рассматриваемых свойств и не должно включаться в меру сходства.

В случае сравнения объекта с эталоном, вектор эталона представляется вектором, состоящим из единиц.

Массив

В контексте данной работы предполагается, что каждый массив представляет собой некий набор свойств произвольной длины (например, список возможных цветов футболки в интернет-магазине). Данное предположение исходит из того, что это одна из самых распространённых семантик массивов в JSON, и с точки зрения программы задача отличения её от других семантик не является тривиальной и не рассматривается.

Принимая во внимание вышеописанное, наилучшим решением видится результат следующего алгоритма:

1. Для каждого элемента кратчайшего массива находится наиболее подходящий ему элемент более длинного;
2. Для каждой пары подобранных элементов вычисляется мера сходства (либо берётся уже вычисленная в процессе подбора пар);
3. В качестве результирующей меры сходства берётся арифметическое среднее полученной в п. 2 последовательности, домноженное на некоторый коэффициент $k < 1$, характеризующий разницу в длине массивов.

В качестве k в данной работе предлагается принять следующую величину: $\sqrt{\frac{l_a}{l_b}}$, где l_a - длина более короткого массива, l_b - длина более длинного массива. При таком выборе разница между массивами возрастает при росте разницы между их длинами, но медленнее, чем линейно: например, массив из 3 элементов будет больше схож с массивом, состоящим из одного из

этих элементов, чем с массивом, состоящим из одного из этих элементов и двух совершенно новых элементов.

Для решения задачи о назначениях в п. 1 используется метод полного перебора.

Также возможны случаи, когда при сравнении массивов важен порядок элементов, например, массив, где i-й элемент - это булева величина, отвечающая за наличие или отсутствие скругления i-го угла у некого многоугольника. К сожалению, подобные случаи довольно специфичны, и не видится возможным программно отличить их от вышеописанного списка свойств, поэтому в контексте данной работы они не рассматриваются.

Строка

В отличие от сравнения векторов, для меры схожести двух строк нет очевидного решения. В связи с этим в данной работе предлагаются наиболее популярные варианты решения данной проблемы.

В программной реализации используется сходство Джаро-Винклера, однако реализованы все перечисленные ниже метрики.

1. Расстояние Левенштейна исчисляется минимально возможным количеством операций удаления\замены символа, необходимых для получения одной строки из другой.

   Само по себе расстояние Левенштейна это достаточно примитивная метрика, которая даёт хорошие результаты на строках малой длины, но при увеличении количества символов результаты размываются; так, с точки зрения данной метрики, строки “foo” и “bar” настолько же далеки друг от друга, насколько и строки “beauty” и “beautiful”, несмотря на очевидную схожесть второй пары для человеческого глаза. Из-за этого, а также из-за того, что нам нужна мера схожести, в данной работе будет

рассматриваться модификация данной метрики, вычисляющаяся при помощи следующей формулы:

$sim_{Lm} = 1 - \frac{d_L}{|s|}$, где d_L - расстояние Левенштейна, |s| - наибольшая из длин строк. Таким образом, для одинаковых строк такая мера будет равна 1, а для полностью отличающихся - 0.

2. Метод n-грам заключается в идее разбиения строки на участки из n букв и сравнения количества совпадающих n-грам с их общим количеством. В качестве n чаще всего выбирается 2 или 3. Например, так будет выглядеть разбиение слова “martha” на триграмы: [mar art rth tha].

   Далее возможно применение основанных на сравнении множеств метрик, таких как коэффициент Жаккара, индекс Тверски, а также их обобщения и модификации. В контексте данной работы будет рассмотрена мера схожести на основе коэффициента Сёренсена, вычисляющаяся по следующей формуле:

   $DSC = \frac{2|X \cap Y|}{|X| + |Y|}$, где |X| - количество триграм в строке X, $|X \cap Y|$ - количество совпадающих триграм в строках.

3. Сходство Джаро-Винклера[4] вычисляется в несколько этапов. Вначале получается сходство Джаро:

   $sim_j = \frac{1}{3}(\frac{c}{d} + \frac{c}{r} + \frac{c-t}{c})$, где:

   - c - число *совпадающих* символов;
   - d - длина первой строки;
   - r - длина второй строки;
   - t - количество *транспозиций*.

В контексте данного выражения *совпадающими* считаются только те символы, которые присутствуют в обеих строках и находятся на расстоянии не

более $\frac{Max(d, r)}{2} - 1$. Количество *транспозиций* вычисляется как половина совпадающих, но идущих не по порядку, символов.

Далее задаются префиксные коэффициент p и длина l. Длина префикса рассматривается менее 4 символов; коэффициент же не должен превышать 0.25, иначе сходство может получиться больше 1. Винклер использует 0.1, поэтому данная работа также будет придерживаться этого значения. Итоговая формула для сходства Джаро-Винклера выглядит следующим образом:

$sim_w = sim_j + 0.1 \cdot l(1 - sim_j)$.

4. Ratcliff/Obershelp Pattern Recognition[5] - метрика, вычисляемая следующим образом:

   $sim_{ro} = \frac{2K_m}{|S_1| + |S_2|}$, где K_m - количество *совпадающих* символов, $|S_1|$ и $|S_2|$ - длины сравниваемых строк. K_m вычисляется рекурсивно: складываются длина наибольшей подстроки и K_m подстрок слева и справа от неё.

5. Также имеют место различные фонетические метрики, основанные на звучании произношения строк, и т.н. “простые” метрики, такие как разница в длине, схожесть префиксов\постфиксов и т.д. Метрики из данного пункта не рассматриваются в данной работе по причине специфичности случаев, в которых они могут быть показательными, но могут быть использованы вместо аналогов при работе с этими случаями.

Число

В качестве меры схожести чисел будет использоваться отношение меньшего числа к большему. Такая метрика говорит сама за себя и наилучшим образом выражает схожесть чисел.

Значение true/false

При сравнении двух таких значений будет использоваться также довольно простая и логичная мера сходства, равная 1 при тождественности значений; иначе равная 0.

Глава 2. Векторизация объектов и программная реализация.

Основные принципы векторизации

Для обеспечения единообразия результатов, данная работа придерживается следующих правил при преобразовании объектов в вектора:

1. Единый порядок.

   При преобразовании значений в координаты вектора, для обеспечения порядка ключи сортируются лексикографически. Поскольку JSON-объект не может иметь двух полей с одинаковыми ключами, данная сортировка обеспечит одинаковый порядок ключей для всех рассматриваемых объектов.

   Также перед началом обработки все объекты приводятся к одному и тому же множеству ключей: недостающие поля заполняются значениями null.

2. Эталон.

   Перед началом обработки данных должен быть предоставлен объект-эталон, с которым будут сравниваться все рассматриваемые объекты. Возможны варианты реализации со случайным выбором эталона (либо нескольких) из предоставленного множества объектов, но в данной

работе они не рассматриваются по причине обширности и требовательности. Иными словами, это само по себе достойно отдельной работы, посему в целях упрощения представленная программная реализация алгоритма требует отдельно указанный эталон.

3. Нормированные координаты.

   Общность рассматриваемых данных обязывает нас к тому, чтобы каждое рассматриваемое свойство имело одинаковый вес для дальнейшей работы с получившимися векторами, например, подсчёт длины, расстояния между ними и т.п. Вследствие этого было выдвинуто требование к рассматриваемым мерам сходства: их значения должны лежать между 0 и 1, где 0 - это полное несовпадение сравниваемых значений, а 1 - точное сходство. Это же правило позволяет нам с лёгкостью сравнивать объекты в предыдущей главе.

4. Репрезентативное сравнение строк.

   Перед сравнением строки приводятся к определённому виду с целью сделать результат их сравнения наиболее репрезентативным:

   a. Все символы по возможности приводятся к строчным;
   b. Убираются все знаки пробелов и табуляции в начале и в конце каждой строки.

Набор объектов для примера и тестирования

Для проверки написанного алгоритма используется набор, объекты которого имитируют одежду - товары интернет-магазина. Анализируются 6 объектов, имеющих следующую структуру:

```
{
  "name": "string",
  "type": "string",
```

```
 "price": 0,
 "variants": [
        {
                "color": "string",
                "size": 0
        }
 ],
 "free_delivery": false,
 "manufacturer": "string"
```

}, где:

1. **name** - название товара - *строка*;
2. **type** - тип продукции - *строка*;
3. **price** - цена товара в долларах - *число*;
4. **variants** - возможные вариации товара - *массив объектов*:
    4.1. **color** - цвет варианта - *строка*;
    4.2. **size** - численное выражение размера варианта - *число*;
5. **free_delivery** - есть ли бесплатная доставка - *true/false*;
6. **manufacturer** - изготовитель - *строка*.

Структура была подобрана таким образом, чтобы в описании объектов присутствовали все типы данных, а также чтобы каждое свойство давало чёткую информацию об объекте.

Программная реализация

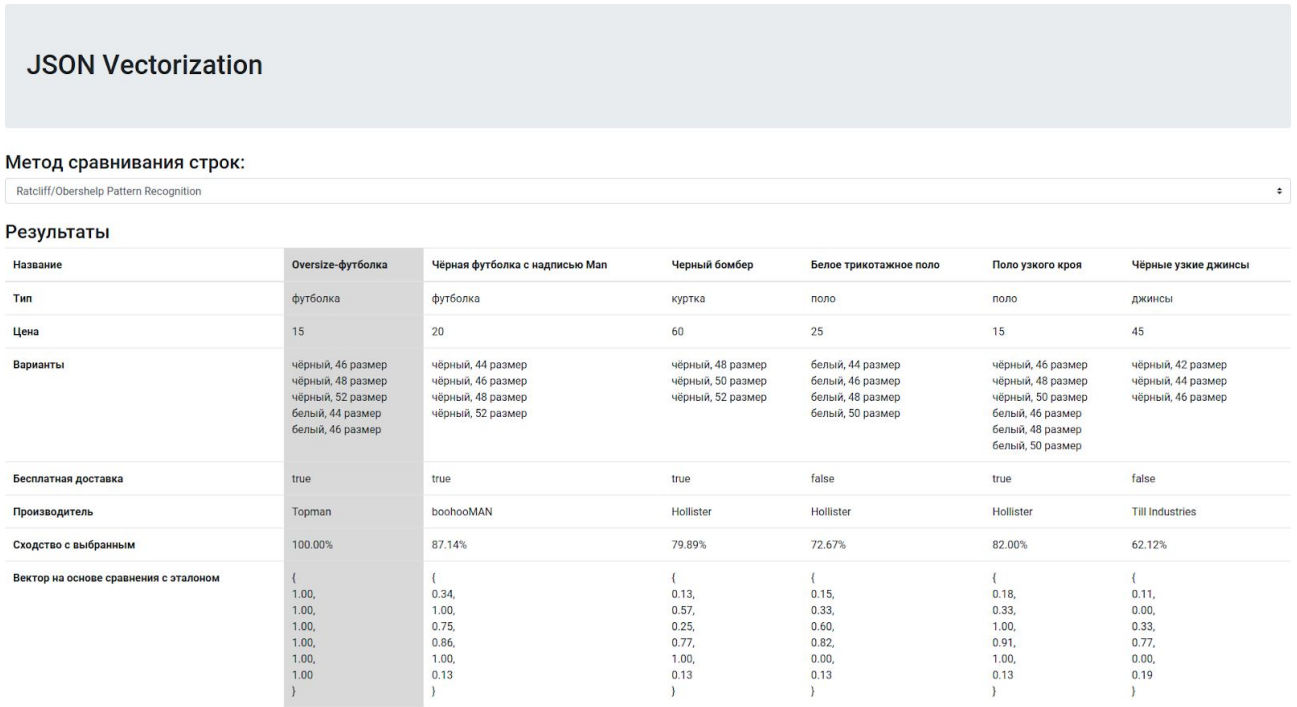
Программный пример реализации описанных в данной работе алгоритмов был выполнен с использованием языка JavaScript и визуализирован при помощи HTML и набора инструментов для стилизации веб-приложений Bootstrap (распространяется по лицензии MIT) [6].

Были реализованы следующие элементы:

- Отображение списка сравниваемых объектов в виде таблицы;
- Возможность выбора эталона для сравнения с остальными объектами;
- Возможность выбора алгоритма сравнения строк;
- Мгновенное отображение результатов сравнения как в виде метрики схожести объекта с эталоном, так и в виде получившегося в результате работы алгоритма вектора.

С исходным кодом можно ознакомиться в приложении А. В файле data.js находится JSON с тестовыми данными, в algo.js - реализация алгоритма, в index.html - визуализация, jaro_winkler.js - адаптация алгоритма нахождения сходства Джаро-Винклера, изначально написанного на C, под JavaScript.

Далее приведены несколько скриншотов, демонстрирующих работу программы:



JSON Vectorization

Метод сравнивания строк:

Ratcliff/Obershelp Pattern Recognition

Результаты

Название	Oversize-футболка	Чёрная футболка с надписью Man	Черный бомбер	Белое трикотажное поло	Поло узкого кроя	Чёрные узкие джинсы
Тип	футболка	футболка	куртка	поло	поло	джинсы
Цена	15	20	60	25	15	45
Варианты	чёрный, 46 размер чёрный, 48 размер чёрный, 52 размер белый, 44 размер белый, 46 размер	чёрный, 44 размер чёрный, 46 размер чёрный, 48 размер чёрный, 52 размер	чёрный, 48 размер чёрный, 50 размер чёрный, 52 размер	белый, 44 размер белый, 46 размер белый, 48 размер белый, 50 размер	чёрный, 46 размер чёрный, 48 размер чёрный, 50 размер белый, 46 размер белый, 48 размер белый, 50 размер	чёрный, 42 размер чёрный, 44 размер чёрный, 46 размер
Бесплатная доставка	true	true	true	false	true	false
Производитель	Topman	boohooMAN	Hollister	Hollister	Hollister	Till Industries
Сходство с выбранным	100.00%	87.14%	79.89%	72.67%	82.00%	62.12%

Вектор на основе сравнения с эталоном	{
	1.00,
	1.00,
	1.00,
	1.00,
	1.00,
	1.00
	}
		{
		0.34,
		1.00,
		0.75,
		0.86,
		1.00,
		0.13
		}
			{
			0.13,
			0.57,
			0.25,
			0.77,
			1.00,
			0.13
			}
				{
				0.15,
				0.33,
				0.60,
				0.82,
				0.00,
				0.13
				}
					{
					0.18,
					0.33,
					1.00,
					0.91,
					1.00,
					0.13
					}
						{
						0.11,
						0.00,
						0.33,
						0.77,
						0.00,
						0.19
						}

JSON Vectorization

Метод сравнивания строк:

Сходство Джаро-Винклера

Результаты

Название	Oversize-футболка	Чёрная футболка с надписью Man	Черный бомбер	Белое трикотажное поло	Поло узкого кроя	Чёрные узкие джинсы
Тип	футболка	футболка	куртка	поло	поло	джинсы
Цена	15	20	60	25	15	45
Варианты	чёрный, 46 размер чёрный, 48 размер чёрный, 52 размер белый, 44 размер белый, 46 размер	чёрный, 44 размер чёрный, 46 размер чёрный, 48 размер чёрный, 52 размер	чёрный, 48 размер чёрный, 50 размер чёрный, 52 размер	белый, 44 размер белый, 46 размер белый, 48 размер белый, 50 размер	чёрный, 46 размер чёрный, 48 размер чёрный, 50 размер белый, 46 размер белый, 48 размер белый, 50 размер	чёрный, 42 размер чёрный, 44 размер чёрный, 46 размер
Бесплатная доставка	true	true	true	false	true	false
Производитель	Topman	boohooMAN	Hollister	Hollister	Hollister	Till Industries
Сходство с выбранным	100.00%	92.97%	87.45%	73.96%	84.49%	70.36%

Вектор на основе сравнения с эталоном	{
	1.00,
	1.00,
	1.00,
	1.00,
	1.00,
	1.00
	}
		{
		0.49,
		1.00,
		0.75,
		0.86,
		1.00,
		0.45
		}
			{
			0.44,
			0.53,
			0.25,
			0.77,
			1.00,
			0.45
			}
				{
				0.55,
				0.00,
				0.60,
				0.83,
				0.00,
				0.45
				}
					{
					0.56,
					0.00,
					1.00,
					0.91,
					1.00,
					0.45
					}
						{
						0.43,
						0.00,
						0.33,
						0.77,
						0.00,
						0.48
						}

JSON Vectorization

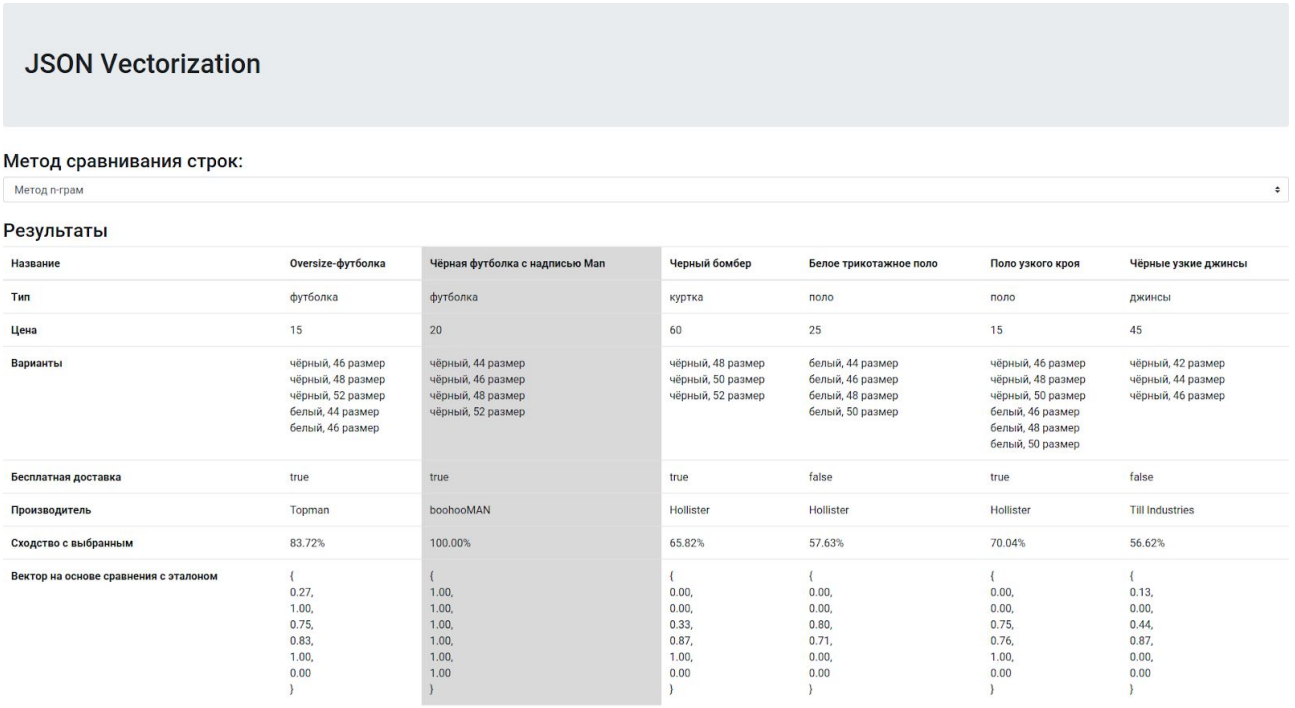
Метод сравнивания строк:

Ratcliff/Obershelp Pattern Recognition

Результаты

Название	Oversize-футболка	Чёрная футболка с надписью Man	Черный бомбер	Белое трикотажное поло	Поло узкого кроя	Чёрные узкие джинсы
Тип	футболка	футболка	куртка	поло	поло	джинсы
Цена	15	20	60	25	15	45
Варианты	чёрный, 46 размер чёрный, 48 размер чёрный, 52 размер белый, 44 размер белый, 46 размер	чёрный, 44 размер чёрный, 46 размер чёрный, 48 размер чёрный, 52 размер	чёрный, 48 размер чёрный, 50 размер чёрный, 52 размер	белый, 44 размер белый, 46 размер белый, 48 размер белый, 50 размер	чёрный, 46 размер чёрный, 48 размер чёрный, 50 размер белый, 46 размер белый, 48 размер белый, 50 размер	чёрный, 42 размер чёрный, 44 размер чёрный, 46 размер
Бесплатная доставка	true	true	true	false	true	false
Производитель	Topman	boohooMAN	Hollister	Hollister	Hollister	Till Industries
Сходство с выбранным	87.14%	100.00%	81.96%	71.80%	81.81%	59.74%

Вектор на основе сравнения с эталоном	{
	0.34,
	1.00,
	0.75,
	0.86,
	1.00,
	0.13
	}
		{
		1.00,
		1.00,
		1.00,
		1.00,
		1.00,
		1.00
		}
			{
			0.28,
			0.57,
			0.33,
			0.87,
			1.00,
			0.11
			}
				{
				0.15,
				0.33,
				0.80,
				0.83,
				0.00,
				0.11
				}
					{
					0.17,
					0.33,
					0.75,
					0.78,
					1.00,
					0.11
					}
						{
						0.37,
						0.00,
						0.44,
						0.87,
						0.00,
						0.00
						}



Название	Oversize-футболка	Чёрная футболка с надписью Man	Черный бомбер	Белое трикотажное поло	Поло узкого кроя	Чёрные узкие джинсы
Тип	футболка	футболка	куртка	поло	поло	джинсы
Цена	15	20	60	25	15	45
Варианты	чёрный, 46 размер чёрный, 48 размер чёрный, 52 размер белый, 44 размер белый, 46 размер	чёрный, 44 размер чёрный, 46 размер чёрный, 48 размер чёрный, 52 размер	чёрный, 48 размер чёрный, 50 размер чёрный, 52 размер	белый, 44 размер белый, 46 размер белый, 48 размер белый, 50 размер	чёрный, 46 размер чёрный, 48 размер чёрный, 50 размер белый, 46 размер белый, 48 размер белый, 50 размер	чёрный, 42 размер чёрный, 44 размер чёрный, 46 размер
Бесплатная доставка	true	true	true	false	true	false
Производитель	Topman	boohooMAN	Hollister	Hollister	Hollister	Till Industries
Сходство с выбранным	83.72%	100.00%	65.82%	57.63%	70.04%	56.62%

Вектор на основе сравнения с эталоном	{
	0.27,
	1.00,
	0.75,
	0.83,
	1.00,
	0.00
	}
		{
		1.00,
		1.00,
		1.00,
		1.00,
		1.00,
		1.00
		}
			{
			0.00,
			0.00,
			0.33,
			0.87,
			1.00,
			0.00
			}
				{
				0.00,
				0.00,
				0.80,
				0.71,
				0.00,
				0.00
				}
					{
					0.00,
					0.00,
					0.75,
					0.76,
					1.00,
					0.00
					}
						{
						0.13,
						0.00,
						0.44,
						0.87,
						0.00,
						0.00
						}

Заключение

В рамках данной работы были получены следующие результаты:

- Рассмотрены различные меры сходства однотипных данных и выбраны лучшие для каждого типа, либо приведено несколько равноправных альтернатив. Для некоторых типов были рассмотрены возможности сравнения с другими типами данных;
- Разработано приложение, реализующее вычисление вышеописанных мер сходства и использование их в целях векторизации JSON-объектов на основании эталона;
- Сгенерирован набор тестовых данных, содержащий список JSON-объектов, состоящих из разнотипных полей таким образом, чтобы каждый из рассматриваемых типов присутствовал в этих объектах;
- Реализовано дополнение к основным алгоритмам, предоставляющее визуализацию результатов их работы на тестовом наборе данных.

Полученные результаты имеют обширную область применения. Так, их можно использовать как для различных рекомендательных систем, от подбора похожих товаров в интернет-магазине до поиска друзей по интересам в социальных сетях, так и для последующего анализа полученных данных. Векторный вид результата позволяет использовать множество различных методов для разностороннего анализа.

Приложения

Приложение А. Примеры кода.

A.1. algo.js: реализация алгоритмов вычисления мер сравнения и векторизации.

```
function prepare(model, obj) {
  Object.keys(obj).forEach(prop => {
    if (!model.hasOwnProperty(prop)) {
      delete obj[prop];
    }
  });

  Object.keys(model).forEach(prop => {
    if (!obj.hasOwnProperty(prop)) {
      obj[prop] = null;
    }
  });
}

function compareValues(val1, val2) {
  switch (typeof val1) {
    // #TYPE string
    case 'string':
      if (typeof val2 === 'number' || typeof val2 === 'boolean') {
        val2 = val2.toString();
      }
      if (typeof val2 !== 'string') return 0;
      return compareStrings(val1, val2);
    // #TYPE number
    case 'number':
      if (typeof val2 === 'string') {
        val2 = parseFloat(val2);
      }
      if (isNaN(val2 - val2)) return 0;
      return Math.min(val1, val2) / Math.max(val1, val2);
    // #TYPE boolean
```

```
      case 'boolean':
        if (typeof val2 === 'string') {
          val2 = (val2 == true);
        }
        return val1 == val2 ? 1 : 0;
      case 'object':
        if (Array.isArray(val1)) {
          // #TYPE array
          if (!Array.isArray(val2)) {
            return 0;
          }
          return compareArrays(val1, val2);
        } else if (val1 === null) {
          // #TYPE null
          return val2 === null ? 1 : 0;
        } else {
          // #TYPE object
          if (typeof val2 !== 'object' || val2 === null || Array.isArray(val2)) {
            return 0;
          }
          return compareObjects(val1, val2);
        }
    }
  }

  function vectorize(model, obj) {
    prepare(model, obj);

    const res = new Array(Object.keys(model).length);

    Object.keys(model).forEach((propName, index) => {
      res[index] = compareValues(model[propName], obj[propName]);
    });

    return res;
  }

  function compareArrays(arr1, arr2) {
    let table = Array.apply(null, new Array(arr1.length)).map(() => new
Array(arr2.length));
    arr1.forEach((item1, index1) => {
      arr2.forEach((item2, index2) => {
        table[index1][index2] = compareValues(item1, item2);
      });
    });

    if (arr1.length > arr2.length) {
      table = transpose(table);
    }

    let indexes = table[0].map((_, index) => index);
    let assignments;
    let assignmentsSum = 0;

```

```
    do {
      let intermediateSum = 0;
      const intermediateArray = new Array(table.length);
      table.forEach((row, rowIndex) => {
        intermediateArray[rowIndex] = indexes[rowIndex];
        intermediateSum += row[indexes[rowIndex]];
      });

      if (intermediateSum > assignmentsSum) {
        assignments = intermediateArray;
        assignmentsSum = intermediateSum;
      }

      indexes = nextArrayTransmutation(indexes);
    } while (indexes);

    const res = assignments.map((item, index) => table[index][item]).reduce((acc,
value) => acc + value, 0) / assignments.length;
    return res * Math.sqrt(table.length / table[0].length);
  }

  function compareObjects(obj1, obj2) {
    const vec = vectorize(obj1, obj2);

    return (vec.reduce((acc, item) => acc + item, 0)) / (
      Math.sqrt(vec.reduce((acc, item) => acc + item * item)) *
Math.sqrt(vec.length)
    );
  }

  function nextArrayTransmutation(arr) {
    arr = arr.slice();
    const reversed = arr.slice().reverse();
    const j = arr.length - 1 - reversed.findIndex((_, index, arr) => index > 0 &&
arr[index] < arr[index - 1]);
    if (j === arr.length) return false;

    const k = arr.length - 1 - reversed.findIndex(item => item > arr[j]);

    [arr[j], arr[k]] = [arr[k], arr[j]];

    let l = j + 1, r = arr.length - 1;
    while (l < r) {
      [arr[l], arr[r]] = [arr[r], arr[l]];
      ++l;
      --r;
    }

    return arr;
  }

  function transpose(arr) {
    const res = Array.apply(null, new Array(arr[0].length)).map(() => new
Array(arr.length));
```

```
    arr.forEach((row, rowIndex) => {
      row.forEach((item, itemIndex) => {
        res[itemIndex][rowIndex] = item;
      });
    });

    return res;
  }

  function compareStrings(str1, str2) {
    const algo = window.__stringComparisonAlgo || RatcliffObershelp;

    return algo(str1, str2);
  }

  function Levenshtein(a, b) {
    if (a.length === 0) return b.length
    if (b.length === 0) return a.length
    let tmp, i, j, prev, val, row
    // swap to save some memory O(min(a,b)) instead of O(a)
    if (a.length > b.length) {
      tmp = a
      a = b
      b = tmp
    }

    row = Array(a.length + 1)
    // init the row
    for (i = 0; i <= a.length; i++) {
      row[i] = i
    }

    // fill in the rest
    for (i = 1; i <= b.length; i++) {
      prev = i
      for (j = 1; j <= a.length; j++) {
        if (b[i-1] === a[j-1]) {
          val = row[j-1] // match
        } else {
          val = Math.min(row[j-1] + 1, // substitution
                Math.min(prev + 1,     // insertion
                         row[j] + 1))  // deletion
        }
        row[j - 1] = prev
        prev = val
      }
      row[a.length] = prev
    }
    return 1 - (row[a.length] / Math.max(a.length, b.length));
  }

  function nGram(str1, str2) {
    str1 = '_' + str1 + '_';
```

```
    str2 = '_' + str2 + '_';
    const str1grams = [];
    const str2grams = [];
    const n = 3;

    for (let i = 0; i < str1.length - n; ++i) {
      str1grams.push(str1.substring(i, i + n));
    }

    for (let i = 0; i < str2.length - n; ++i) {
      str2grams.push(str2.substring(i, i + n));
    }

    let commonCount = 0;

    str1grams.forEach(gram => {
      if (str2grams.includes(gram)) ++commonCount;
    });

    return 2 * commonCount / (str1grams.length + str2grams.length);
  }

  function RatcliffObershelp(str1, str2) {
    const stack = [];
    stack.push(str1, str2);

    let score = 0;

    while (stack.length > 0) {
      const s1 = stack.pop();
      const s2 = stack.pop();

      let longestSequenceLength = 0;
      let longestSequenceIndex1 = -1;
      let longestSequenceIndex2 = -1;

      for (let i = 0; i < s1.length; ++i) {
        for (let j = 0; j < s2.length; ++j) {
          let k = 0;
          while (i + k < s1.length && j + k < s2.length && s1.charAt(i + k) ===
  s2.charAt(j + k)) {
            ++k;
          }

          if (k > longestSequenceLength) {
            longestSequenceLength = k;
            longestSequenceIndex1 = i;
            longestSequenceIndex2 = j;
          }
        }
      }

      if (longestSequenceLength === 0) {
        continue;
```

```
      }

      score += longestSequenceLength * 2;
      if (longestSequenceIndex1 !== 0 && longestSequenceIndex2 !== 0) {
        stack.push(s1.substring(0, longestSequenceIndex1), s2.substring(0,
longestSequenceIndex2));
      }
      if (longestSequenceIndex1 + longestSequenceLength !== s1.length &&
longestSequenceIndex2 + longestSequenceLength !== s2.length) {
        stack.push(s1.substring(longestSequenceIndex1 + longestSequenceLength),
s2.substring(longestSequenceIndex2 + longestSequenceLength));
      }
    }
    return score / (str1.length + str2.length);
  }
```

A.2. jaro_winkler.js: реализация вычисления сходства Джаро-Винклера, из-за своей сложности вынесенная в отдельный файл.

```
  var jaro_winkler = {};

  function jaroWinkler(a, b) {

    if (!a || !b) { return 0.0; }

    a = a.trim().toUpperCase();
    b = b.trim().toUpperCase();
    var a_len = a.length;
    var b_len = b.length;
    var a_flag = []; var b_flag = [];
    var search_range = Math.floor(Math.max(a_len, b_len) / 2) - 1;
    var minv = Math.min(a_len, b_len);

    // Looking only within the search range, count and flag the matched pairs.
    var Num_com = 0;
    var yl1 = b_len - 1;
    for (var i = 0; i < a_len; i++) {
      var lowlim = (i >= search_range) ? i - search_range : 0;
      var hilim  = ((i + search_range) <= yl1) ? (i + search_range) : yl1;
      for (var j = lowlim; j <= hilim; j++) {
        if (b_flag[j] !== 1 && a[j] === b[i]) {
          a_flag[j] = 1;
          b_flag[i] = 1;
          Num_com++;
          break;
        }
      }
    }

    // Return if no characters in common
    if (Num_com === 0) { return 0.0; }

    // Count the number of transpositions
```

```
    var k = 0; var N_trans = 0;
    for (var i = 0; i < a_len; i++) {
      if (a_flag[i] === 1) {
        var j;
        for (j = k; j < b_len; j++) {
          if (b_flag[j] === 1) {
            k = j + 1;
            break;
          }
        }
        if (a[i] !== b[j]) { N_trans++; }
      }
    }
    N_trans = Math.floor(N_trans / 2);

    // Adjust for similarities in nonmatched characters
    var N_simi = 0; var adjwt = jaro_winkler.adjustments;
    if (minv > Num_com) {
      for (var i = 0; i < a_len; i++) {
        if (!a_flag[i]) {
          for (var j = 0; j < b_len; j++) {
            if (!b_flag[j]) {
              if (adjwt[a[i]] === b[j]) {
                N_simi += 3;
                b_flag[j] = 2;
                break;
              }
            }
          }
        }
      }
    }

    var Num_sim = (N_simi / 10.0) + Num_com;

    // Main weight computation
    var weight = Num_sim / a_len + Num_sim / b_len + (Num_com - N_trans) / Num_com;
    weight = weight / 3;

    // Continue to boost the weight if the strings are similar
    if (weight > 0.7) {
      // Adjust for having up to the first 4 characters in common
      var j = (minv >= 4) ? 4 : minv;
      var i;
      for (i = 0; (i < j) && a[i] === b[i]; i++) { }
      if (i) { weight += i * 0.1 * (1.0 - weight) };

      // Adjust for long strings.
      // After agreeing beginning chars, at least two more must agree
      // and the agreeing characters must be more than half of the
      // remaining characters.
      if (minv > 4 && Num_com > i + 1 && 2 * Num_com >= minv + i) {
        weight += (1 - weight) * ((Num_com - i - 1) / (a_len * b_len - i*2 + 2));
      }
```

```
    }

    return weight

  };

  // The char adjustment table used above
  jaro_winkler.adjustments = {
    'A': 'E',
    'A': 'I',
    'A': 'O',
    'A': 'U',
    'B': 'V',
    'E': 'I',
    'E': 'O',
    'E': 'U',
    'I': 'O',
    'I': 'U',
    'O': 'U',
    'I': 'Y',
    'E': 'Y',
    'C': 'G',
    'E': 'F',
    'W': 'U',
    'W': 'V',
    'X': 'K',
    'S': 'Z',
    'X': 'S',
    'Q': 'C',
    'U': 'V',
    'M': 'N',
    'L': 'I',
    'Q': 'O',
    'P': 'R',
    'I': 'J',
    '2': 'Z',
    '5': 'S',
    '8': 'B',
    '1': 'I',
    '1': 'L',
    '0': 'O',
    '0': 'Q',
    'C': 'K',
    'G': 'J',
    'E': ' ',
    'Y': ' ',
    'S': ' '
  }
```

A.3. data.js: тестовые данные.

```
  window.__data = JSON.parse(`[
    {
      "name": "Oversize-футболка",
```

```
      "type": "футболка",
      "price": 15,
      "variants": [
        {
          "color": "чёрный",
          "size": 46
        },
        {
          "color": "чёрный",
          "size": 48
        },
        {
          "color": "чёрный",
          "size": 52
        },
        {
          "color": "белый",
          "size": 44
        },
        {
          "color": "белый",
          "size": 46
        }
      ],
      "free_delivery": true,
      "manufacturer": "Topman"
    },
    {
      "name": "Чёрная футболка с надписью Man",
      "type": "футболка",
      "price": 20,
      "variants": [
        {
          "color": "чёрный",
          "size": 44
        },
        {
          "color": "чёрный",
          "size": 46
        },
        {
          "color": "чёрный",
          "size": 48
        },
        {
          "color": "чёрный",
          "size": 52
        }
      ],
      "free_delivery": true,
      "manufacturer": "boohooMAN"
    },
    {
      "name": "Черный бомбер",
```

```
      "type": "куртка",
      "price": 60,
      "variants": [
        {
          "color": "чёрный",
          "size": 48
        },
        {
          "color": "чёрный",
          "size": 50
        },
        {
          "color": "чёрный",
          "size": 52
        }
      ],
      "free_delivery": true,
      "manufacturer": "Hollister"
    },
    {
      "name": "Белое трикотажное поло",
      "type": "поло",
      "price": 25,
      "variants": [
        {
          "color": "белый",
          "size": 44
        },
        {
          "color": "белый",
          "size": 46
        },
        {
          "color": "белый",
          "size": 48
        },
        {
          "color": "белый",
          "size": 50
        }
      ],
      "free_delivery": false,
      "manufacturer": "Hollister"
    },
    {
      "name": "Поло узкого кроя",
      "type": "поло",
      "price": 15,
      "variants": [
        {
          "color": "чёрный",
          "size": 46
        },
        {
```

```
          "color": "чёрный",
          "size": 48
        },
        {
          "color": "чёрный",
          "size": 50
        },
        {
          "color": "белый",
          "size": 46
        },
        {
          "color": "белый",
          "size": 48
        },
        {
          "color": "белый",
          "size": 50
        }
      ],
      "free_delivery": true,
      "manufacturer": "Hollister"
    },
    {
      "name": "Чёрные узкие джинсы",
      "type": "джинсы",
      "price": 45,
      "variants": [
        {
          "color": "чёрный",
          "size": 42
        },
        {
          "color": "чёрный",
          "size": 44
        },
        {
          "color": "чёрный",
          "size": 46
        }
      ],
      "free_delivery": false,
      "manufacturer": "Till Industries"
    }
  ]`);
```

A.4. index.html: визуализация применения алгоритмов на тестовых данных.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <link rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
  integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk
  " crossorigin="anonymous">
    <script src="./jaro_winkler.js"></script>
    <script src="./algo.js"></script>
    <script src="./data.js"></script>
    <title>JSON Vectorization</title>
  </head>
  <body>
    <div class="jumbotron">
      <h1>JSON Vectorization</h1>
    </div>

    <h3>Метод сравнивания строк:</h3>
    <select class="custom-select" id="select-string-comp">
      <option value="0">Расстояние Левенштейна</option>
      <option value="1">Метод n-грам</option>
      <option value="2">Сходство Джаро-Винклера</option>
      <option value="3" selected>Ratcliff/Obershelp Pattern Recognition</option>
    </select>

    <h3 class="pt-4">Результаты</h3>

    <table class="table">
      <thead>
        <tr data-property="name">
          <th>Название</th>
          <th data-index="0"></th>
          <th data-index="1"></th>
          <th data-index="2"></th>
          <th data-index="3"></th>
          <th data-index="4"></th>
          <th data-index="5"></th>
        </tr>
      </thead>
      <tbody>
        <tr data-property="type">
          <th>Тип</th>
          <td data-index="0"></td>
          <td data-index="1"></td>
          <td data-index="2"></td>
          <td data-index="3"></td>
          <td data-index="4"></td>
          <td data-index="5"></td>
        </tr>
        <tr data-property="price">
          <th>Цена</th>
          <td data-index="0"></td>
          <td data-index="1"></td>
          <td data-index="2"></td>
          <td data-index="3"></td>
          <td data-index="4"></td>
          <td data-index="5"></td>
```

```
        </tr>
        <tr data-property="variants">
          <th>Варианты</td>
          <td data-index="0"></td>
          <td data-index="1"></td>
          <td data-index="2"></td>
          <td data-index="3"></td>
          <td data-index="4"></td>
          <td data-index="5"></td>
        </tr>
        <tr data-property="free_delivery">
          <th>Бесплатная доставка</td>
          <td data-index="0"></td>
          <td data-index="1"></td>
          <td data-index="2"></td>
          <td data-index="3"></td>
          <td data-index="4"></td>
          <td data-index="5"></td>
        </tr>
        <tr data-property="manufacturer">
          <th>Производитель</td>
          <td data-index="0"></td>
          <td data-index="1"></td>
          <td data-index="2"></td>
          <td data-index="3"></td>
          <td data-index="4"></td>
          <td data-index="5"></td>
        </tr>
        <tr data-property="similarity">
          <th>Сходство с выбранным</td>
          <td data-index="0"></td>
          <td data-index="1"></td>
          <td data-index="2"></td>
          <td data-index="3"></td>
          <td data-index="4"></td>
          <td data-index="5"></td>
        </tr>
        <tr data-property="vector">
          <th>Вектор на основе сравнения с эталоном</td>
          <td data-index="0"></td>
          <td data-index="1"></td>
          <td data-index="2"></td>
          <td data-index="3"></td>
          <td data-index="4"></td>
          <td data-index="5"></td>
        </tr>
      </tbody>
    </table>

    <script>
      const data = window.__data;
      select(0);

      data.forEach((obj, index) => {
```

```
        Object.keys(obj).forEach(key => {
          const elem = document.querySelector(`tr[data-property="${key}"] >
[data-index="${index}"]`);

          elem.innerHTML = obj[key];

          if (key === 'variants') {
            elem.innerHTML = obj[key].map(variant => `${variant.color},
${variant.size} размер`).join('<br>');
          }
        });
      });

      document.querySelectorAll('th[data-index]').forEach((elem, index) => {
        elem.addEventListener('click', () => {
          select(index);
        });
      });

      document.querySelector('#select-string-comp').addEventListener('change', (e)
=> {
        window.__stringComparisonAlgo = ([Levenshtein, nGram, jaroWinkler,
RatcliffObershelp])[e.currentTarget.value];
        select(0);
      })

      function select(index) {
        updateProto(index);
        document.querySelectorAll('*').forEach(elem => {
          elem.classList.remove('active');
        });
        document.querySelectorAll('[data-index="' + index + '"]').forEach(elem => {
          elem.classList.add('active');
        });
      }

      function updateProto(index) {
        const proto = data[index];

        data.forEach((obj, objIndex) => {
          const similarity = compareObjects(proto, obj);
          const vector = vectorize(proto, obj);
          document.querySelector(`tr[data-property="similarity"] >
[data-index="${objIndex}"]`).innerHTML = `${(similarity * 100).toFixed(2)}%`;
          document.querySelector(`tr[data-property="vector"] >
[data-index="${objIndex}"]`).innerHTML = `{<br>${vector.map(val =>
val.toFixed(2)).join(', <br>')}<br>}`;
        });
      }
    </script>
    <style>
      th[data-index] {
        cursor: pointer;
      }
```

```
      .active {
        background-color: rgba(0, 0, 0, 0.15);
      }
    </style>
  </body>
  </html>
```

Список литературы

[1] https://tools.ietf.org/html/rfc8259

[2] Guenther Retscher, Julian Joksch - Comparison of Different Vector Distance Measure Calculation Variants for Indoor Location Fingerprinting

[3] Li, B. & Han, L. (2013). Distance weighted cosine similarity measure for text classification.

[4] Winkler, William E. (1990) - String Comparator Metrics and Enhanced Decision Rulesin the Fellegi-Sunter Model of Record Linkage

[5] Ilya Ilyankou - Comparison of Jaro-Winkler and Ratcliff/Obershelp algorithms in spell check

[6] https://getbootstrap.com/docs/4.5