Реализация алгоритма шинглов на Node.JS. Поиск нечетких дубликатов для английских текстов

Разработка веб-сайтов *Алгоритмы *Node.JS *
Из песочницы
При работе с информацией часто возникают задачи парсинга веб-страниц. Одной из проблем в этом деле является определение похожих страниц. Хороший пример такого алгоритма — «Алгоритм шинглов для веб-документов».

Часть проекта по парсингу реализована на Node.JS, поэтому и алгоритм нужно было реализовать на нем. Реализаций на javascript или npm-пакетов я не нашел — пришлось писать свою.

Вся работа по коду основана на статье выше, поэтому все пункты алгоритма будет из нее, но с некоторыми поправками.

Для определения схожести 2-х документов необходимо:

1. канонизация текста;
2. разбиение на шинглы;
3. вычисление хешей шинглов с помощью 84х статических функций;
4. случайная выборка 84 значений контрольных сумм;
5. сравнение, определение результата.

Пункты 3,4 для меня оказались довольно проблематичными. 1-е — необходимо найти 84 статических функции для хеширования, а 2-е – случайная выборка 84-х значений контрольных сумм. Если для 1й проблемы – решения найти можно, то второе мне не ясно. Если массив шинглов для текста мы хешируем 84-мя функциями то выходит что на выходе получится 2-х мерный массив размерностью 84xN(кол-во шинглов в документе). Теперь необходимо обойти этот 84-х элементный массив для каждого текста и сравнить случайные хеши шинглов. Можно сравнивать случайные элементы, но такой вариант может не дать совпадений. Если брать минимальные хеши по длинне, то для md5 все хеши равны по длине, а рассчитывать длину по кодам символов – дополнительная нагрузка. Поэтому я решил пункты 3 и 4 заменить на простое хеширование шинглов с помощью crc32 и последовательное сравнение.
Конечный алгоритм:

1. канонизация текста;
2. разбиение на шинглы;

3. вычисление хешей шинглов с помощью crc32;
4. последовательное сравнение, определение результата.

1. Канонизация текста

В моем случае канонизация состоит из:

1. очистка от html сущностей;
2. очистка от лишних пробелов по бокам(trim);
3. очистка от таких спец символов "”", "“", "\n", '\r', ',', '.', ':', '$', '#', "", '(', ')';
4. очистка от ненужных частей речи в предложении

Для начала необходимо подготовить методы для обработки текста.

```
var strWordRemove = function(entry) {
    var regex = new RegExp('(^|\\s)'  + entry + '(?=\\s|$)', 'g');
    text = text.replace(regex, '');
  };

  var strCharacterRemove = function(entry) {
    var escapeRegExp = function (str) {
      return str.replace(/[\-\[\]\/\{\}\(\)\*\+\?\.\\\^\$\|]/g, "\\$&");
    };

    var regex = new RegExp(escapeRegExp(entry), 'g');
    text = text.replace(regex, '');
  };
```

Первый нужен для замено слов в тексте, а второй для замены спец. символов. Далее идет сама обработка:

```
  var withoutTagsRegex = /(<([^>]+)>)/ig;

  text = text.replace(withoutTagsRegex, "");

  text = text.trim();

  ['”', '“', "\n", '\r'].forEach(strCharacterRemove);
```

Для Node.JS есть npm-пакет “pos”, который позволяет находить в тексте части речи. Работает довольно неплохо.

Обработка частей речи с помощью pos

```
var words = new pos.Lexer().lex(text);
  var taggedWords = new pos.Tagger().tag(words);

  var removeWords = [];
  var nounWords = [];

  for (var i in taggedWords) {
    var taggedWord = taggedWords[i];
```

```
    var word = taggedWord[0];
    var tag = taggedWord[1];

    //Adjective

    /*
     JJ Adjective               big
     JJR Adj., comparative      bigger
     JJS Adj., superlative      biggest
     CC Coord Conjuncn          and,but,or
     IN Preposition             of,in,by
     TO ÒtoÓ                    to
     UH Interjection            oh, oops
     DT Determiner              the,some
     */

    //console.log(word + " /" + tag);
    if(tag === 'NNS') {
      nounWords.push(word);
    }

    if(['JJ', 'JJR', 'JJS', 'CC', 'IN', 'TO', 'UH', 'DT'].indexOf(tag) !== -1) {
      removeWords.push(word);
    }
  }

  removeWords.forEach(strWordRemove);
```

Все остальные спец. символы я решил убрать после обработки частей речи.

```
[',', '.', ':', '$', '#', '"', '(', ')'].forEach(strCharacterRemove);
```

Далее осталось привести все существительные к единственному виду и блок канонизации можно считать готовым. Стоить заметить, что pos заносит к множественным существительным такие слова как Command's. Их я решил пропускать.

Существительные к единственному виду

```
// replace all plural nouns to single ones
  nounWords.forEach(function(entry) {
    //parent's || Apple's || Smurf's
    if(entry.length > 2 && entry.slice(-2) === "'s") {
      // now skip it. in future we can test to remove it
      return ;
    }

    var newOne = '';

    if(entry.length > 3 && entry.slice(-3) === "ies") {
      newOne = entry.slice(0, -3) + 'y';
    } else if(entry.length > 2 && entry.slice(-1) === "s") {
      newOne = entry.slice(0,-1);
    } else {
      return ;
    }

    var rexp = new RegExp('(^|\\s)' + entry + '(?=\\s|$)','g')
    text = text.replace(rexp, "$1" + newOne );
  });
```

Убираем все множественные пробелы и передаем текст на следующий уровень.

```
text = text.replace(/ +(?= )/g,'');
callback(text);
```

2. Разбиение на шинглы

С этим пунктом все просто. Делим текст по пробелам и создаем массивы.

```
var makeShingles = function(text, callback) {
  var words = text.split(' ');
  var shingles = [];
  var wordsLength = words.length;
  while(shingles.length !== (wordsLength - shingleLength + 1)) {
   shingles.push(words.slice(0, shingleLength).join(' '));
   words = words.slice(1);
  }

  callback(shingles)
};
```

3. Вычисление хешей шинглов с помощью crc32

В этом пункте мы обходим массив шинглов и хешируем строки. Первый цикл от 0 до 1 остался от попытки хешировать с помощью 84-х функций. Решил не убирать(вдруг вернусь к этой идее).

```
var hashingShingles = function(shingles, callback) {
  var hashes = [];
  for(var i = 0, n = 1; i < n; i++) {
    var hashedArr = [];
    for(var j = 0, k = shingles.length; j < k; j++) {
        hashedArr.push(crc.crc32(shingles[j]));
    }
    hashes.push(hashedArr);
  }

  callback(hashes);
};
```

4. Последовательное сравнение, определение результата

Для примера я взял 2 новости из google news которые тот показал как похожие. Сохранил их в json файле и далее, для более высокой скорости, обрабатывал параллельно с помощью Async utilities. После чего нашел количество совпавших шинглов и рассчитал результат.

Определение результатов для 2-х текстов

```
var fileJSON = require('./article1.json');
var content1 = fileJSON.content;
```

```
var fileJSON2 = require('./article2.json');
var content2 = fileJSON2.content;

var async = require('async');

async.parallel([
  function(callback){
    textCanonization(content1, function(text) {
      makeShingles(text, function(shingles) {
        hashingShingles(shingles, function(hashes) {
          callback(null, hashes);
        });
      })
    });
  },
  function(callback){
    textCanonization(content2, function(text) {
      makeShingles(text, function(shingles) {
        hashingShingles(shingles, function(hashes) {
          callback(null, hashes);
        });
      })
    });
  }
], function(err, results){
    var firstHashes = results[0];
    var secondHashes = results[1];


    var compareShingles = function(arr1, arr2) {
      var count = 0;

      arr1[0].forEach(function(item) {
        if(arr2[0].indexOf(item) !== -1) {
          count++;
        }
      });

      return count*2/(arr1[0].length + arr2[0].length)*100;
    };

    var c = compareShingles(firstHashes, secondHashes);

    console.log(c);
  });
```

Формула `count*2/(arr1[0].length + arr2[0].length)*100` находит процентное соотношение для 2х текстов.

Тексты для сравнения: FTC says Apple will pay at least $32.5 million over in-app purchases и Apple will pay $32.5m to settle app complaints. При количестве слов в шингле, равном 10 — тексты были похожи на 2.16% что очень неплохо.
Из вопросов не ясно, чем вариант использования 84х функций лучше. А также хотелось бы знать какой-то алгоритм для высчитывания оптимального количества слов в шингле(в текущем указано 10).
Весь исходный код алгоритма и пример работы можно посмотреть на github.com

Теги:

- алгоритмы поиска
- алгоритм шинглов
- шинглы
- node.js

Хабы:

- Разработка веб-сайтов
- Алгоритмы
- Node.JS

+12
10K

Редакторский дайджест

Присылаем лучшие статьи раз в месяц

8
Карма
0
Рейтинг
@aslikeyou

Пользователь

Комментарии 8

glebmachine 21.01.2014 в 14:15
Мимо написал комментарий, простите)
0

0decca 21.01.2014 в 14:48
Многие вещи типа дедупликации или даже установления авторства можно сделать вот этим.

Собственно идея крайне проста, если слить два текста и упаковать, то коэффициент сжатия будет тем больше, чем больше одинаковых паттернов в обоих текстах.
Как ни странно — часто работает.
:-)

Правда при сколько-нибудь реальных объемах документов — процессор расплавится, т.к. расчет очень затратен, а для всей базы еще и квадратичен.

Я в похожих задачах вместо шинглов использовал bloom filter на случайных или на полных ngramm'ax.
Как-то работало, в сочетании с aNN методами, чтобы уйти от квадратичности.

У BF есть преимущество — все магические константы (типа 84) достаточно просто считаются под задачу и обоснованы теорвером и прочим матаном.
+1
aslikeyou 21.01.2014 в 22:04
За «Normalized compression distance» спасибо. В любом случае интересно сравнить ресурсоемкость/качество работы алгоритма. А какие aNN методы вы использовали в паре с bloom filter?
0

0decca 21.01.2014 в 22:35
Ну, по ресурсоемкости хуже чем NCD наверное и не найти, компрессия — штука тяжелая, и чем она тяжелее, тем точнее результат.

Я с Bloom filter использовал самописную версию LSH, думал перейти на более хитрые извращения, но потом задачу решил по другому, вообще без BF и дедупликации, т.е. саму постановку задачи перекрутил.

Имхо по дефолту лучше не заморачиваться велосипедописательством и брать например FLANN — вполне приличная библиотека, и с хорошей лицензией.
И там уже выбирать внутри нее.

Для BF кстати лучше использовать Жаккардову метрику, не Хемминга — считается битовыми операциями и как-то «нативнее» для BF — показывает отношения в множестве. Ну и она же и используется в известном MinHash — который тоже может быть применен для дедупликации.
0
aslikeyou 22.01.2014 в 18:10
Спасибо, разобрался с NCD.

Один на github, а второй на одном сайте по c# .

Если вкратце есть формула NCD(x,y) = C(xy) — min{C(x), C(y)} / max{C(x), C(y)}, где C возвращает длинну текста (строк, изображений и тд) после обработки алгоритмом архивации(например: «gzip», «bzip2», «PPMZ»). Причем xy — это два текста склеиных в одну переменную. Например xy = x + ' ' + y.

Выложил свой пример на github. Только не могу понять, почему для двух строк 'hello world' выводит резульат 0.09.
0

0decca 22.01.2014 в 18:57

Так потому что архиватор — это приближение.
В точности должна быть т.н. «Колмоговровская сложность».
А архиватор дает очень приблизительную оценку.

Длина архива с двумя идентичными текстами всегда будет хоть на один бит больше, чем длина архива с одним таким текстом.
Вот эта дельта и играет роль.
В теории чем длиннее тексты, тем меньше будет ошибка.
На практике архиваторы работают по блокам и за размером блока значения начинают плавать.
Хотя сильно зависит от архиватора.

И еще, на маленьких строках портит все заголовок архива.
Т.е. вот Hello world!\n
13 байт
А архив bzip2 — 53 байта — вот такая хреновая компрессия.
А двойной архив — 57 байт.

А с другой стороны — вот берем файл www.lib.ru/GIBSON/neuromancer.txt
485967 байт
Соответственно сжатый отдельно он — 152317
А сжатый с самим собой — 223390

Получаем дистанцию 0.467, что совершенно не похоже на ожидаемый 0.
Это уже артефакт блочной архивации.
Т.е. не все там так просто и не всякий архиватор пойдет.

А вот возьмем например архиватор xz
У него архивированный текст «Нейромантика» — 167532 байт (хуже чем у bzip2)
А архивированный сам с собой — 167680
Что дает дистанцию — 0.0008834

Совсем другое дело.

+1

rinat_crone 22.01.2014 в 19:25
Для поиска дубликатов текста в одном из проектов использую алгоритм LSA (http://blog.netpeak.ru/algoritm-lsa-dlya-poiska-pohozhih-dokumentov/). Реализацию не проблема найти под требуемый язык программирования.
0
aslikeyou 24.01.2014 в 18:22
Можете более детально обьяснить, как расчитывать степень похожести 2-х документов после вычисления матрицы U, V, W?
0
Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

Похожие публикации

- **Захват флага: Практика уязвимости веб-приложений на Node.js (часть 1)**

+4
1.4K

- 0

1. **7 способов улучшить производительность Node.js в масштабе**

+13
2.8K

- 3 +3

1. **Поиск и устранение неисправностей Node.js-приложений под капотом**

+42
12K

- 1 +1

Минуточку внимания

Разместить

Промо

Промокод — твой билет в общество потребления

Мегапост

Что и как с онлайн-образованием в 2022 году

Опрос

Хотите рассказать о себе в наших социальных сетях?

Вопросы и ответы

- После отправки запроса страница перезагружается?
  JavaScriptПростой1 ответ
- Как с помощью pillow обрезать изображение, чтобы оно обрезалось адоптируясь под размер фотографии?

PythonСредний1 ответ

- Как изменить место хранения директории node_modules?
  Node.jsПростой1 ответ
- Как перехватывать 500-е ошибки (в частности ошибку 502) в node.js express?
  Node.jsСредний1 ответ
- Как проверить на соответствие модель таблицы и саму таблицу?
  Node.jsПростой0 ответов

Больше вопросов на Хабр Q&A

Лучшие публикации за сутки

- ### Сгорел сарай, гори и хата, или Месть британского сисадмина

+43
28K

- 59 +59

1. Когда тестирование бессильно. Космические лучи меняют биты памяти чаще, чем принято думать

+40
11K

- 104 +104

1. Kincony KC868-A32: авианосец на DIN-рейку

+35
3.7K

- 18 +18

1. Конь остановлен, изба догорела

+34
5.1K

- 22 +22

1. «Ленивый сахар» PostgreSQL

+32
4.1K

- 5 +5
- Волны гасят ветер: 12 историй про веру разработчиков в российские IT
  Мегапост

Читают сейчас

- **Символы Unicode: о чём должен знать каждый разработчик**

  241K

  47 +47

- **Конь остановлен, изба догорела**

  5.2K

  22 +22

- **В открытый доступ выложены файлы с данными курьеров сервисов доставки «Яндекс.Еда» и Delivery Club**

  5K

  8 +8

- **Сгорел сарай, гори и хата, или Месть британского сисадмина**

  28K

  59 +59

- **Luxoft прекращает деятельность в России**

  9K

  18 +18

- Как попытка улучшить свою жизнь работает на онлайн-образование

  Мегапост

Работа

JavaScript разработчик

378 вакансий

Node.js разработчик

124 вакансии

Все вакансии

Ваш аккаунт

- Войти
- Регистрация

Разделы

- Публикации
- Новости
- Хабы
- Компании
- Авторы
- Песочница

Информация

- Устройство сайта
- Для авторов
- Для компаний
- Документы
- Соглашение
- Конфиденциальность

Услуги

- Корпоративный блог
- Медийная реклама
- Нативные проекты
- Мегапроекты

Техническая поддержка Вернуться на старую версию