Тимофей Типишев

Отчёт по выполнению ДЗ №2

Это домашнее задание состояло из трёх основных частей, а именно

1. Написать *simhash* функцию, с большой интересной подзадачей равномерно распределить векторы по единичной сфере
2. Избежать большого квадрата при сравнении сигнатур файлов при поиске полудублей
3. Вывести результаты по трём групам порога различия сигнатур (5, 10, и 15 бит)

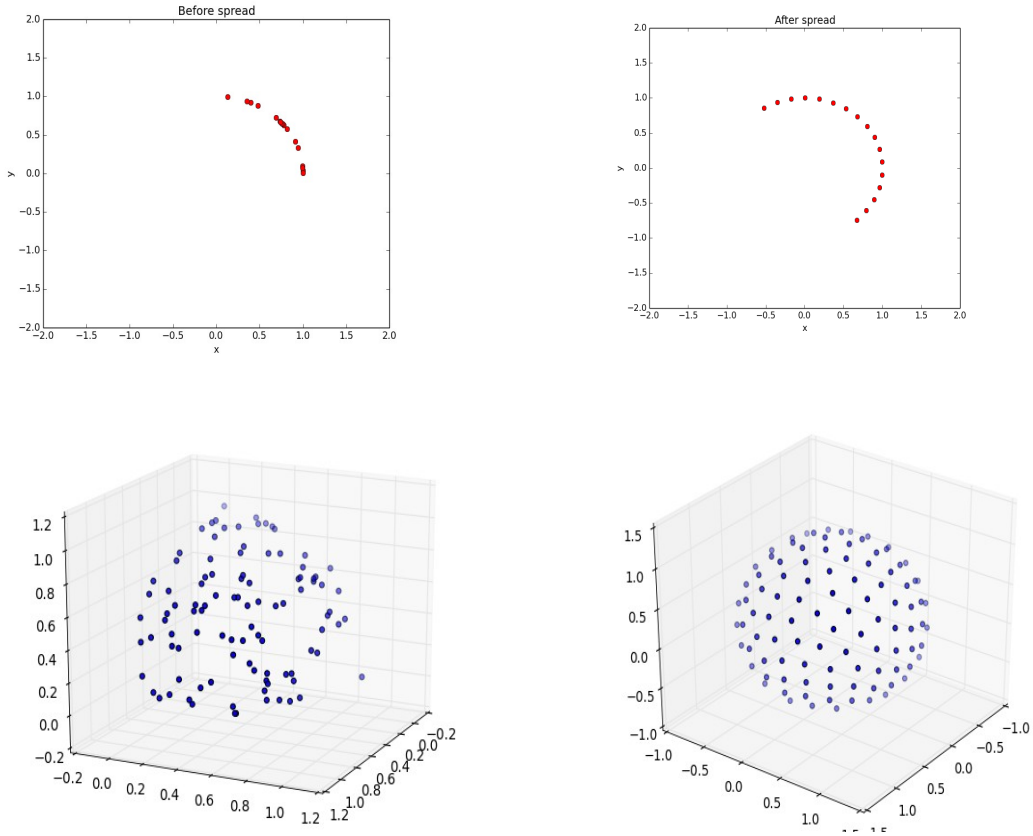
1. Simhash функция

Для того чтобы построить *simhash* функцию каждый документ бился на слова, для каждого слова считался *md5* хэш, который позже переводился в двоичную запись. От этой записи брались первые 42[1] бита, затем в зависимости от символа 0 или 1 на каждой позиции этих 42 бит к отпечатку документа (вектору целых чисел длиной 42) добавлялась либо +1 либо -1 соответственно.

1 Все мы знаем почему

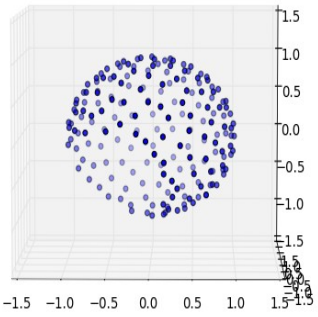
1.push() Равномерно распределённые векторы

Для следующей части нахождения *simhash*'а требовалось построить 64 случайных единичных вектора размерности 42. Для этого я воспроизвёл алгоритм с семинара, а именно моделирование электронов на полусфере с итерированием отталкивания до схождения. Я проверил его корректность на двух и трёх измерениях.



До и после равномерного распределения

Поскольку построение векторов занимает какое-то время, их, как и все операции, которые долго считаются я сериализовал и записывал в файл.



Напрямую к заданию не относится, но ради интереса я посмотрел что происходит если позволить зарядам растечься по всей сфере (без учёта вторых половинок гантели) и остановиться на несошедшейся итерации. Заряды растеклись неравномерно и “освоили” не всю сферу.

1.pop()

После получения векторов я мог быстро получить *simhash* для всех документов путём скалярного умножения их отпечатков на каждый из векторов и записи 0 или 1 в simhash в зависимости от знака скалярного произведения. Для всех документов я посчитал таблицу вида

имя файла	количество слов	симхэш

2. Избегание квадратища

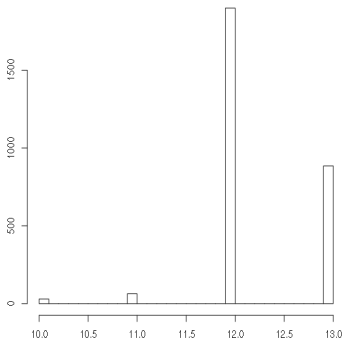
Как и было сказано в задании, сверка все-со-всеми – наивный и очень не эффективный подход к поиску дупликатов потому что никак не использует условие различия в количестве слов и требует десятки миллиардов сравнений даже при скромном корпусе из 158000 документов. Более разумный подход, также обозначенный в задании – упорядочить документы по количеству слов и сравнивать только те документы, количество слов в которых различается не более чем на 20%.
В коде я это реализовал как пару функций: одна бъёт индексы упорядоченного массива чисел на окна внутри которых длины различаются не более чем на 20% (или любое другое значение), а вторая по этому разбиению оценивает сколько сравнений потребуется при подсчёте расстояния Хэмминга, то есть складывает квадратики для каждого окна. При 20% количество разбиений вполне приемлемо – несколько миллионов.

3. Результаты

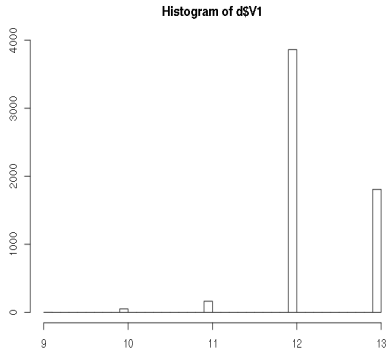
Для всех потенциальных пар полудупликатов было найдено расстояние Хэмминга между их симхэшами. После этого в зависимости от порога похожести (5, 10, 15) бит были найдены списки полудупликатов и записаны в приложенные файлы (*sorted_n10_dupes.txt*, *sorted_n15_dupes.txt*, *sorted_n5_dupes.txt*) для удобства изучения в одной строке записаны главный файл и его потенциальные дупликаты, файлы отсортированы по длине строки для поиска самых топ-N “повторимых” статей. Чаще всего – это пустые статьи и едва заполненные шаблоны о бесчисленных городах Северной Америки и коммунах Франции.

3.1 Гистограммы

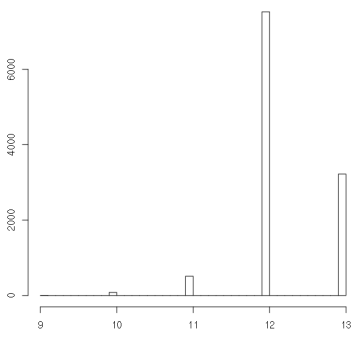
Хотя и встречаются статьи с несколькими сотнями дублей, по большей части размеры кластеров не так велики:



Для n=5
Всего 2877 кластеров



Для n=10
Всего 5880 кластеров



Для n=15
Всего 11340 кластеров