TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**TIỂU LUẬN MÔN XỬ LÝ DỮ LIỆU LỚN**

**SỬ DỤNG PHƯƠNG PHÁP LOCALITY SENSITIVE HASHING VÀ MÔI TRƯỜNG PYSPARK ĐỂ TÌM KIẾM CÁC VĂN BẢN TƯƠNG TỰ TRONG TẬP VĂN BẢN**

*Người hướng dẫn*: **THẦY LÊ ANH CƯỜNG**

*Người thực hiện*: **THÂN TRỌNG HUỲNH NHÂN – 51800590**

**HUỲNH TẤN LỢI – 51800574**

**VÕ TẤN LỰC - 51800900**

Lớp **: 18050302**

Khoá  **: 22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**TIỂU LUẬN MÔN XỬ LÝ DỮ LIỆU LỚN**

**SỬ DỤNG PHƯƠNG PHÁP LOCALITY SENSITIVE HASHING VÀ MÔI TRƯỜNG PYSPARK ĐỂ TÌM KIẾM CÁC VĂN BẢN TƯƠNG TỰ TRONG TẬP VĂN BẢN**

Người hướng dẫn: **THẦY LÊ ANH CƯỜNG**

Người thực hiện: **THÂN TRỌNG HUỲNH NHÂN**

**HUỲNH TẤN LỢI**

**VÕ TẤN LỰC**

Lớp **: 18050302**

Khoá  **: 22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

LỜI CẢM ƠN

Trong thời gian làm tiểu luận này, chúng em đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy và bạn bè.

Nhóm em xin gửi lời cảm ơn chân thành đến thầy Lê Anh Cường, giảng viên Bộ môn Xử lý dữ liệu lớn trường ĐH Tôn Đức Thắng, người đã tận tình hướng dẫn, chỉ bảo nhóm em trong suốt quá trình làm tiểu luận cũng như trong việc giảng dạy.

Nhóm em cũng xin chân thành cảm ơn các thầy cô giáo trong trường ĐH Tôn Đức Thắng nói chung, các thầy cô trong khoa CNTT nói riêng đã dạy dỗ cho em kiến thức về các môn đại cương cũng như các môn chuyên ngành, giúp chúng em có được cơ sở lý thuyết vững vàng và tạo điều kiện giúp đỡ nhóm em trong suốt quá trình học tập.

Cuối cùng, nhóm em xin chân thành cảm ơn gia đình và bạn bè, đã luôn tạo điều kiện, quan tâm, giúp đỡ, động viên em trong suốt quá trình học tập và hoàn thành bài tiểu luận này.

**TIỂU LUẬN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm tiểu luận của riêng tôi / chúng tôi và được sự hướng dẫn của thầy Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong tiểu luận còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung tiểu luận của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 31 tháng 10 năm 2020*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Thân Trọng Huỳnh Nhân*

*Huỳnh Tấn Lợi*

*Võ Tấn Lực*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

* 1. Lý do chọn đề tài

Vấn đề bản quyền sở hữu trí tuệ, đạo văn trong các tác phẩm văn học, bài báo khoa học, các công trình nghiên cứu,… hiện nay đang rất nhức nhói và càng được đề cao nhầm bảo vệ quyền lợi cho các tác giả, nhà khoa học, đồng thời cũng góp phần nhận biết, đánh giá năng lực một cách khách quan và chính xác hơn.

Chúng em quyết định chọn đề tài “Sử dụng phương pháp Locality Sensitive Hashing và môi trường pyspark để tìm kiếm các văn bản tương tự trong tập văn bản” để tìm hiểu về quá trình tìm kiếm các thành phần văn bản tương tự nhau tròn tập văn bản. Bên cạnh đó, việc chọn đề tài này còn giúp chúng em hiểu hơn về phương pháp LSH – Locality Sensitive Hashing và cách sử dụng LSH để tìm kiếm văn bản tương tự. Đồng thời, đề tài giúp chúng em có thể nâng cao hiểu biết về kỹ năng nghiệp vụ, cũng như các vấn đề khi sử dụng LSH để giải quyết vấn đề này gặp phải.

* 1. Mục đích của tài liệu

Hiểu rõ hơn về LSH – Locality Sensitive Hashing và quá trình tìm kiếm văn bản tương tự trong tập văn bản bất kỳ. Trên cơ sở đó, xây dựng chương trình lọc các bản tin, văn bản tương tự trong các tập văn bản trên web tin tức hoặc các tập cho trước. Học hỏi và trau dồi kiến thức thông qua quá trình nghiên cứu và tiến hành. Nâng cao khả năng nghiệp vụ và khả năng phân tích đặt vấn đề.

* 1. Các vấn đề nghiên cứu

Để tiếp cận cũng như thực hiện đề tài này, chúng em đã nghiên cứu và tìm hiểu ctheo nhiều khía cạnh của vấn đề, cụ thể như sau:

*Chương 1 – Phương pháp Locality Sensitive Hashing (LSH):* Nhằm trang bị kiến thức cho người đọc một cách tổng quan nhất về cách mà phương pháp LSH được sử dụng trong chương trình hoạt động, các khái niệm, công thức cũng như trình tự các bước của phương pháp LSH.

*Chương 2 – Thực nghiệm chương trình lọc văn bản*: Kết hợp kiến thức tổng quan về LSH và kỹ thuật lập trình trên môi trường PySpark để xây dựng nên một chương trình lọc văn bản hoàn thiện. Chương này sẽ giới thiệu các thành phần có trong chương trình được xây dựng (code, hình ảnh minh họa, kết quả,….).

* 1. Quá trình thực hiện và kết quả nghiên cứu

Để giải quyết các vấn đề đặt ra trong đề tài này, nhóm đã đề ra lịch trình và từng bước thực hiện như sau:

* Lên kế hoạch họp vào mỗi thứ 3, 5, 7 hàng tuần
* Phân công tra cứu, thu nhập thông tin từ các nguồn trên internet
* Tiến hành tổng hợp các thông tin đã thu nhập được
* Tiến hành xác định nền tảng triển khai chương trình
* Hoàn thành chương trình
* Hoàn thành và chỉnh sửa để phù hợp với đề tài
* Kết quả của nhóm đã đạt được thồn qua đề tài:

CÁC THÀNH VIÊN ĐÃ HOÀN THÀNH TỐT NHIỆM VỤ ĐƯỢC GIAO

MỤC LỤC

[LỜI CẢM ƠN 1](#_Toc67856008)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN 3](#_Toc67856009)

[TÓM TẮT 4](#_Toc67856010)

[1.1 Lý do chọn đề tài 4](#_Toc67856011)

[1.2 Mục đích của tài liệu 4](#_Toc67856012)

[1.3 Các vấn đề nghiên cứu 4](#_Toc67856013)

[1.4 Quá trình thực hiện và kết quả nghiên cứu 5](#_Toc67856014)

[MỤC LỤC 6](#_Toc67856015)

[DANH MỤC CHỮ VIẾT TẮT 8](#_Toc67856016)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 9](#_Toc67856017)

[BẢNG PHÂN CÔNG NHIỆM VỤ 10](#_Toc67856018)

[CHƯƠNG 1 – PHƯƠNG PHÁP LOCALITY SENSITIVE HASHING (LSH) 11](#_Toc67856019)

[1.1 Đôi nét về Locality Sensitive Hashing 11](#_Toc67856020)

[1.2 Các bước xử lý 11](#_Toc67856021)

[1.2.1 Shingling 12](#_Toc67856022)

[1.2.2 Min-Hashing 14](#_Toc67856023)

[1.2.3 Locality sensitive Hashing 16](#_Toc67856024)

[CHƯƠNG 2 – THỰC NGHIỆM CHƯƠNG TRÌNH LỌC VĂN BẢN TƯƠNG TỰ 21](#_Toc67856025)

[2.1 Tổng quan về Crawl dữ liệu 21](#_Toc67856026)

[2.2 Chương trình thực nghiệm không áp dụng PySpark 22](#_Toc67856027)

[2.2.1 Quá trình chuẩn bị 22](#_Toc67856028)

[2.2.2 Các hàm hỗ trợ có trong chương trình thực nghiệm 22](#_Toc67856029)

[2.2.3 Quá trình Shingling 26](#_Toc67856030)

[2.2.4 Quá trình Min Hashing 28](#_Toc67856031)

[2.2.5 Quá trình Locallity Sensitive Hashing (LSH) 31](#_Toc67856032)

[2.3 Chương trình thực nghiệm có áp dụng PySpark 32](#_Toc67856033)

[2.3.1 Quá trình chuẩn bị 32](#_Toc67856034)

[2.3.2 Các hàm hỗ trợ có trong chương trình thực nghiệm 33](#_Toc67856035)

[2.3.3 Quá trình Shingling 36](#_Toc67856036)

[2.3.4 Quá trình Min Hashing 38](#_Toc67856037)

[2.3.5 Quá trình Locallity Sensitive Hashing (LSH) 39](#_Toc67856038)

[TÀI LIỆU THAM KHẢO 41](#_Toc67856039)

DANH MỤC CHỮ VIẾT TẮT

**CÁC CHỮ VIẾT TẮT**

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

**DANH MỤC HÌNH**

[Hình 1 – So sánh KNN với LSH 11](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856040)

[Hình 2 – Đồ thị thể hiện độ tương tự theo LSH 20](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856041)

[Hình 3 – Crawl dữ liệu 21](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856042)

[Hình 4 – Shingling-Tìm ma trận boolean với k-grams = 10 26](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856043)

[Hình 5 – Output of Shingling 28](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856044)

[Hình 6 – MinHashing và so sánh tốc độ tính similarity của shingles và signatures 28](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856045)

[Hình 7 – Output của bước tìm kiếm và tính toán similarity Jaccard của các Shingles 30](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856046)

[Hình 8 – Output của bước tìm kiếm và tính toán similarity Jaccard của Min-Hashing 31](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856047)

[Hình 9 – Output của quá trình LSH 32](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856048)

[Hình 10 – Output của bước tìm kiếm và tính toán similarity Jaccard của Shingles (chương trình thực nghiệm có áp dụng PySpark) 38](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856049)

[Hình 11 – Output của bước tìm kiếm và tính toán similarity Jaccard của MinHashing (chương trình thực nghiệm có áp dụng PySpark 39](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856050)

[Hình 12 – Thời gian tìm kiếm LSH (chương trình thực nghiệm có áp dụng PySpark) 40](file:///C:\Users\MSI%20GV72\OneDrive\Máy%20tính\51800590_51800574_51800900.docx#_Toc67856051)

**DANH MỤC BẢNG**

[Bảng 1 – Bảng phân công công việc 10](#_Toc67856052)

[Bảng 2 – Công thức tính Jaccard Similary và Jaccard Distance 12](#_Toc67856053)

BẢNG PHÂN CÔNG NHIỆM VỤ

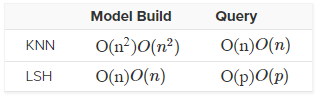
|  |  |  |
| --- | --- | --- |
| **Tên thành viên** | **Nhiệm vụ** | **Mức độ hoàn thành** |
| Thân Trọng Huỳnh Nhân | - Phụ trách mục 1.1 “Đôi nét về Locality Sensitive Hashing” | Hoàn thành tốt |
| - Phụ trách mục 1.2.1 “Shingling” | Hoàn thành tốt |
| - Tổng hợp và hoàn tất báo cáo | Hoàn thành |
| Huỳnh Tấn Lợi | - Phụ trách mục 1.2.2 “Min-Hashing” | Hoàn thành tốt |
| Võ Tấn Lực | - Phụ trách mục 1.2.3 “Locality Sensitive Hashing” | Hoàn thành tốt |
| Các thành viên trong nhóm tham gia code | | Hoàn thành |

Bảng 1 – Bảng phân công công việc

CHƯƠNG 1 – PHƯƠNG PHÁP LOCALITY SENSITIVE HASHING (LSH)

* 1. Đôi nét về Locality Sensitive Hashing

Phương pháp Locality Sensitive Hashing (LSH) là một tập hợp các kỹ thuật sử dụng hàm băm giúp tăng tốc quá trình tìm kiếm lân cận hoặc phát hiện sự trùng lặp gần trên dữ liệu một cách đáng kể. Cách tiếp cận chung đối với LSH là “băm” (hash) các mục nhiều lần. LSH sẽ sử dụng các hàm băm (LSH Families) để hash các dữ liệu vào từng bucket sao cho các dữ liệu gần nhau nhất sẽ có xác suất cao vào cùng một bucket được phân, ngược lại thì chúng sẽ bị băm vào hai bucket khác nhau.

LSH là một loại phương pháp dựa trên lân cận giống như k-nearest neighbors (KNN) nhưng lại có quy mô ưu việt hơn vì có thể mở rộng bằng cách sử dụng kỹ thuật Forest khi số lượng item tăng lên.

Hình 1 – So sánh KNN với LSH

Trong đó: n là số items

P là số permutation

* 1. Các bước xử lý

Bài toán tìm kiếm văn bản tương tự trong tập dữ liệu bằng phương pháp LSH nhìn chung sẽ trải qua ba quá trình chính để có thể đưa ra một kết quả:

* Shingling
* Min-Hashing
* Locality sensitive Hashing
  + 1. Shingling

**Đây là bước đầu tiên của quá trình, ở bước này, dữ liệu của document sẽ được biến đổi thành một tập hợp các ký tự có độ dài k (k-shingles hay k-gram).**

**Ví dụ: Xét tập dữ liệu D = “Hello”:**

* **Với 2-grams thì kết quả sẽ là 🡪 result = {He, el, ll, lo}**
* **Với 3-grams thì kết quả sẽ là 🡪 result = {Hel, llo}**

Việc sắp xếp lại các đoạn văn bản trong tài liệu nếu có bị thay đổi từ ngữ thì vẫn không ảnh hưởng nhiều đến shingles. Và trong thực tế thì giá trị k thường được sử dụng từ 8-10 vì nếu sử dụng giá trị nhỏ sẽ dẫn đến hầu hết tài liệu đều chứa phần tử của shingles (kết quả sẽ sai lệch **🡪** lọc văn bản, so sánh độ tương đồng bị sai). Bên cạnh đó, việc thay đổi từ chỉ ảnh hưởng đến k-shingles trong khoảng cách k-1 từ (sự ảnh hưởng khá thấp).

Sau khi dữ liệu của văn bản cần kiểm tra độ tương đồng và các tập dữ liệu đã được biến đổi thành các tập shingles với độ dài các phần tử k-shingles, việc kiểm tra độ tương tự sẽ được tính bằng chỉ số Jaccard theo công thức:

|  |  |
| --- | --- |
| Sim(D1,D2) =  ***Jaccard Similarity*** | d(D1,D2) = Sim(D1,D2)  ***Jaccard Distance*** |

Bảng 2 – Công thức tính Jaccard Similary và Jaccard Distance

Tuy có thể chọn giá trị k là bất kỳ hằng số nào nhưng phải đảm bảo phù hợp với quá trình sàn lọc vì nếu chọn k quá nhỏ, thì việc chuyển đổi sang định dạng số sẽ mang lại hầu hết các ký tự trong document. Nếu vậy, thì việc so sánh giữa các document khác nhau sẽ mang lại tính tương đồng rất cao (Jaccard cao), điều ấy làm gây nhiễu trong bài toán.

Nếu sử dụng k = 1, hầu hết các trang web hiện nay sẽ có hầu hết các ký tự phổ biến và một vài ký tự khác, vì vậy hầu như tất cả các trang web sẽ có sự tương đồng cao, do đó sẽ gây nhiễu trong bài toán tương đồng. Chọn k phải phụ thuộc vào độ dài của document. Điều quan trọng cần nhớ chính là: *k nên được chọn đủ lớn để xác suất của bất kỳ shingle nào xuất hiện trong bất kỳ document nào không được thấp*.

Bên cạnh đó, để có thể nâng cao kết quả của quá trình so sánh để tìm ra văn bản tương tự trong thực tế thì ở bước shingling này sẽ lấy các câu có trong văn bản sau đó sẽ chia ra các shingles chứa các phần tử là các từ (word). Và dựa vào đặc điểm của các bài báo, các documents hiện hành thường có các từ dừng (stop word) khá phổ biến (“như”, “và”, “bạn”,…). Thêm vào đó, việc xác định các shingle là các từ dừng kết hợp với hai từ theo sau đã tạo nên các shingle khá hữu ích. Cách này được gọi là xây dựng các shingle với các word.

Việc xây dựng các shingles theo word sẽ làm cho các bài báo sau đó sẽ đóng góp nhiều các shingles hơn cho tập hợp đại diện cho trang Web hơn là các yếu tố xung quanh, giúp tiết kiệm chi phí (vì các khi tách các shingles theo ký tự sẽ tốn nhiều chi phí hơn).

**Ví dụ:** Xét các tập dữ liệu

Ta có:

**🡪**

**🡪 sim(A,B) =**

* + 1. Min-Hashing

Sau khi biểu diễn các document dưới dạng ma trận singles. Đên đây vẫn chưa tối ưu vì vẫn phải lưu trữ ma trận singles rất tốn bộ nhớ. Vì vậy ta dùng hàm băm – MinHashing để chuyển đổi các “convert large” thành các “short signatures” mà trong đó vẫn giữ được sự tương đồng. Quá trình thực hiện Min-Hashing diễn ra theo từng bước cụ thể sau:

* *Hoán vị ngẫu nhiên (Permutation ) các dòng index của ma trận singles mà phần trước đã xây dựng*.

|  |  |  |  |
| --- | --- | --- | --- |
| Index | A | B | C |
| 1 | 1 | 1 | 1 |
| 2 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 |
| 4 | 1 | 1 | 0 |
| 5 | 0 | 0 | 0 |
| 6 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 |
| 8 | 0 | 0 | 0 |
| 9 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 |

* + Ví dụ ta có ma trận singles 3 document A, B và C như sau:
  + Hoán vị ngẫu nhiên các index dòng của ma trận

|  |  |  |  |
| --- | --- | --- | --- |
| Index | A | B | C |
| 3 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 9 | 0 | 1 | 1 |
| 5 | 1 | 1 | 0 |
| 2 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 |
| 6 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 |
| 10 | 1 | 0 | 1 |
| 8 | 0 | 1 | 0 |

* *Bắt đầu hash trong MinHashing là ở mỗi cột tương ứng với các document, duyệt đến khi tìm được ô có value = 1 mà tại đó index là nhỏ nhất. Giá trị của hash chính là giá trị của index nhận được (hπ(C) = minππ(C).*

|  |  |  |  |
| --- | --- | --- | --- |
| Index | A | B | C |
| 3 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 9 | 0 | 1 | 1 |
| 5 | 1 | 1 | 0 |
| 2 | 0 | 0 | 1 |
| 4 | 1 | 1 | 1 |
| 6 | 1 | 1 | 1 |
| 7 | 0 | 0 | 0 |
| 10 | 1 | 0 | 1 |
| 8 | 0 | 1 | 0 |

* Sau khi hoàn thành, ta hash ra được một dòng trong ma trận chữ ký là [1,3,2].
* Tương tự các bước trên, làm lại các bước trên nhiều lần với mỗi lần là cột index được hoán vị ngẫu nhiên.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Index 1 | Index 2 | Index 3 | Index 4 |  | A | B | C |
| 5 | 10 | 3 | 5 |  | 1 | 1 | 0 |
| 3 | 5 | 1 | 3 |  | 1 | 1 | 0 |
| 2 | 1 | 9 | 1 |  | 0 | 1 | 1 |
| 4 | 4 | 5 | 4 |  | 1 | 1 | 0 |
| 1 | 6 | 2 | 6 |  | 0 | 0 | 1 |
| 9 | 9 | 4 | 8 |  | 1 | 1 | 1 |
| 10 | 3 | 6 | 7 |  | 1 | 1 | 1 |
| 7 | 2 | 7 | 9 |  | 0 | 0 | 0 |
| 8 | 7 | 10 | 2 |  | 1 | 1 | 0 |
| 6 | 8 | 8 | 10 |  | 0 | 1 | 0 |

* Ma trận chữ ký sau khi hash:

|  |  |  |
| --- | --- | --- |
| A (C1) | B (C2) | C (C3) |
| 3 | 2 | 1 |
| 3 | 1 | 1 |
| 1 | 1 | 2 |
| 2 | 1 | 1 |

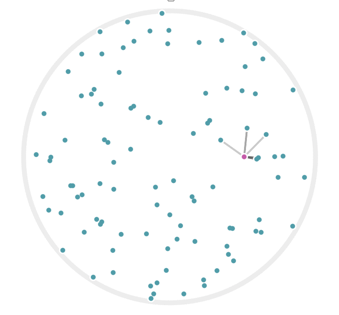
* Bước 3: Tính độ tương đồng giữa các cột (document) của ma trận singles và ma trận chữ ký và so sánh.
* Similarity giữa các chữ ký sẽ là tỉ lệ mà số hàm MinHash (các hàng) có cùng giá trị với nhau.

|  |  |  |  |
| --- | --- | --- | --- |
|  | C1 – C2 | C1 – C3 | C2 – C3 |
| Singles | 6/8 | 2/8 | 3/9 |
| signatures | 1/3 | 0/3 | 2/2 |

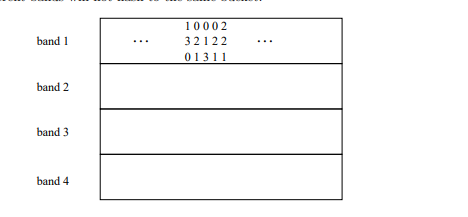
Nhận xét: Thấy có sự khác biệt gía trị giữa hai loại phép tính vì độ dài ma trận chữ ký ở đây là 4. Nếu tăng độ dài đó lên thì độ tương đồng sẽ được nâng cao.

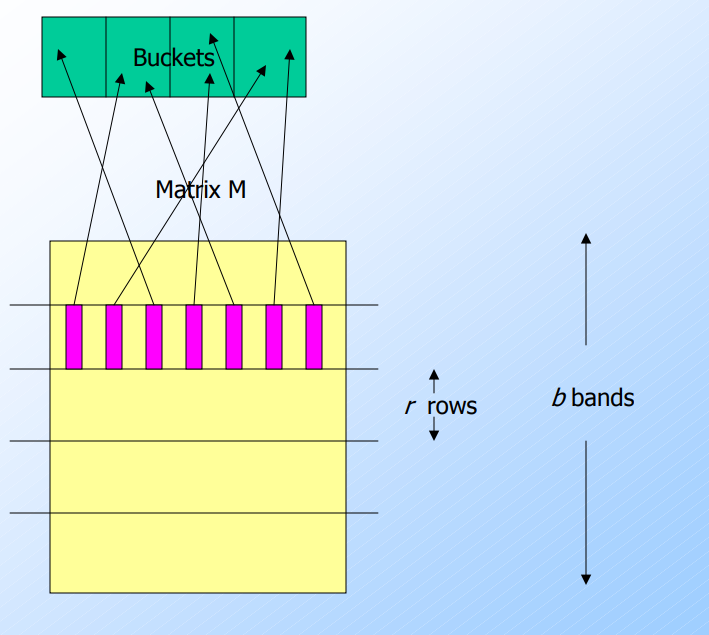
* + 1. Locality sensitive Hashing

Locality-sensitive hashing là một tập hợp các kỹ thuật giúp tăng tốc đáng kể tìm kiếm lân cận hoặc phát hiện gần trùng lặp trên dữ liệu. Các kỹ thuật này có thể được sử dụng, chẳng hạn, để lọc ra các bản sao của các trang web đã được với tốc độ ấn tượng hoặc để thực hiện tra cứu các điểm lân cận theo thời gian gần như liên tục từ tập dữ liệu không gian địa lý.



Một cách sử dụng truyền thống cho các hàm băm là trong các bảng băm. các hàm băm được sử dụng trong bảng băm được thiết kế để ánh xạ một phần dữ liệu thành một số nguyên có thể được sử dụng để tìm trong một nhóm cụ thể trong bảng băm để truy xuất hoặc xóa phần tử đó. Nhiều vùng chứa có khóa chuỗi. Mặc dù bảng băm có thể không đảm bảo tra cứu thời gian liên tục, nhưng trên thực tế, chúng cung cấp chúng một cách hiệu quả.

Một cách tiếp cận chung đối với LSH là "băm" các mục nhiều lần, theo cách mà các mục tương tự có nhiều khả năng được băm vào cùng một nhóm hơn là các mục “dissimilarity”. Sau đó, coi bất kỳ cặp nào được băm vào cùng một nhóm cho bất kỳ băm nào là một cặp “candidate” kiểm tra các cặp ứng cử viên để tìm sự giống nhau. Những cặp không giống nhau nào thực hiện băm cho cùng một nhóm là False positive. Nếu có chữ ký MinHash cho các mục, một cách hiệu quả để chọn các băm là chia ma trận chữ ký thành b dải bao gồm r hàng mỗi dải. Đối với mỗi dải, có một hàm băm nhận vectơ của r số nguyên (phần của một cột trong dải đó) và băm chúng thành một số nhóm lớn. Có thể sử dụng cùng một hàm băm cho tất cả các dải, sử dụng một mảng nhóm riêng biệt cho mỗi dải, vì vậy các cột có cùng vectơ trong các dải khác nhau sẽ không được băm vào cùng một nhóm

Ví dụ trong band 1 thì cột thứ 2 và cột thứ 4 sẽ vào cùng 1 bucket

Hai cột càng giống nhau thì càng có nhiều khả năng chúng sẽ giống hệt nhau trong một dải nào đó. Do đó, về mặt trực giác, chiến lược phân dải làm cho các cột tương tự có nhiều khả năng trở thành các candidate pairs hơn dissimilar pairs

Giả sử chúng ta sử dụng mỗi dải b gồm r hàng và giả sử rằng một cặp tài liệu cụ thể có Jaccard similarity s. xác suất các chữ ký MinHash cho các tài liệu này đồng ý trong bất kỳ hàng cụ thể nào của ma trận chữ ký là s. Chúng ta có thể tính xác suất để các tài liệu này (hay đúng hơn là chữ ký của chúng) trở thành một cặp candidate như sau:

columns C1 và C2 có similarity s

chọn 1 band (r rows)

Xác suất để tất cả các hàng trong dải bằng nhau = sr

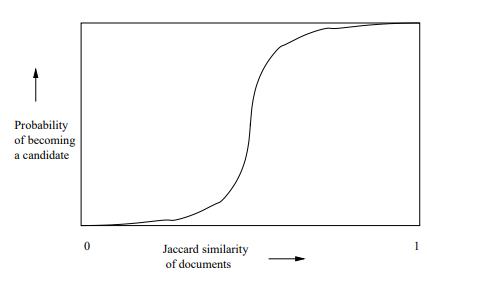
Xác suất để tất cả các hàng trong dải ko bằng nhau = 1-sr

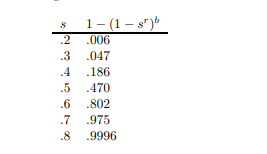
Xác suất không có band nào giống hệt nhau: (1 –sr) b

Xác suất ít nhất 1 band nào giống hệt nhau: 1- (1 –sr) b

Bất kì các hằng số b và r đã chọn, hàm này có dạng đường cong S. threshold là giá trị của độ tương đồng s mà tại đó mức tăng trở nên dốc nhất, là một hàm của b và r. Giá trị gần đúng của threshold là (1/b) ^ (1/r).

Ví dụ, nếu b = 16 và r = 4, thì threshold xấp xỉ là ½.

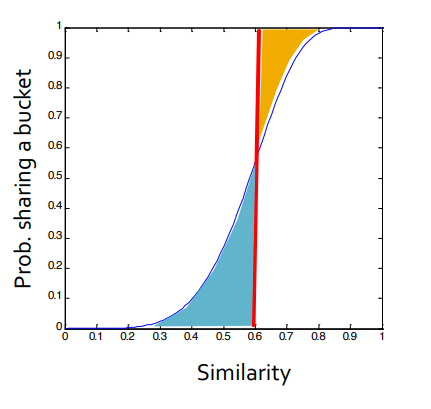


**Ví dụ:** b = 20 và r = 5 thì xác suất có ít nhất 1 band giống hệt nhau

Điều chỉnh M, b, r để có được hầu hết các cặp có chữ ký giống nhau, nhưng loại bỏ hầu hết các cặp không có chữ ký tương tự (M là signature matrix).

Chọn r và b để có S-curve 50 hash-functions (r=5, b=10)

|  |  |
| --- | --- |
| s | 1-(1-sr) b |
| .2 | 0.003 |
| .3 | 0.02 |
| .4 | 0.09 |
| .5 | 0.27 |
| .6 | 0.55 |
| .7 | 0.84 |
| .8 | 0.98 |



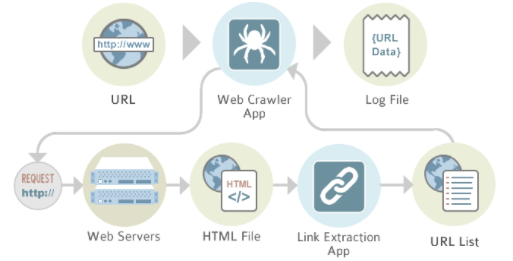
Trong đó:

Yellow area: False Negative rate

Blue area: False Positive rate

Hình 2 – Đồ thị thể hiện độ tương tự theo LSH

CHƯƠNG 2 – THỰC NGHIỆM CHƯƠNG TRÌNH LỌC VĂN BẢN TƯƠNG TỰ

* 1. Tổng quan về Crawl dữ liệu

Hình 3 – Crawl dữ liệu

Kỹ thuật crawl dữ liệu website là kỹ thuật để thu thập dữ liệu khá phổ biến, Google bot cũng là một hình thức của crawler. Kỹ thuật crawler có rất nhiều ứng dụng thực tế như: Xây dựng ứng dụng đọc báo bằng cách crawl dữ liệu website từ các báo lớn, crawl các thông tin tuyển dụng như [ITNavi](https://www.itnavi.com.vn/).v.v… Để tạo ra được một web crawler có rất nhiều cách, và cũng có vô số framework hỗ trợ. Ví dụ như Python thì có Scrapy rất nổi tiếng. Trong bài viết này chúng ta sẽ cùng nhau tìm hiểu kỹ thuật crawler dữ liệu website sử dụng kỹ thuật phân tích cú pháp XML bằng PHP.

 Web crawlers bắt đầu từ một danh sách các URL đã biết. Trước tiên, chúng thu thập dữ liệu webpage tại các URL đó. Từ các page này, chúng sẽ tìm thấy các siêu liên kết đến nhiều URL khác và thêm các liên kết mới tìm được vào danh sách các trang cần thu thập thông tin tiếp theo.

Trong tiểu luận này, sẽ thực hiện crawl dữ liệu từ trang VNExpress.net, vietnam.net và thanhnien.vn

* 1. Chương trình thực nghiệm không áp dụng PySpark
     1. Quá trình chuẩn bị

Sau khi đã crawl dữ liệu hoặc đã có sẵn văn bản cần kiểm tra độ tương tự (ở đây là file *data3.txt*) có trong tập văn bản, tiến hành đọc file văn bản cần kiểm tra bằng phương thức open() và cắt bỏ việc xuống dòng bằng split(“\n”) để chuẩn bị dữ liệu để tiến hành thực hiện chương trình.

import re

from random import randint, seed, choice, random

import time

import binascii

import numpy as np

import itertools

import math

from decimal import \*

from google.colab import drive

drive.mount('/content/drive',force\_remount=True)

f = open("/content/drive/MyDrive/BIGDATA/BTL/data3.txt", "r", encoding="utf-8")

documents = f.read().split("\n")

* + 1. Các hàm hỗ trợ có trong chương trình thực nghiệm
* *Hàm split\_k\_gram(document, shingleNo)*: được dùng để chia, cắt ghép chuỗi dữ liệu trong file văn bản cần kiểm tra độ tương tự thành các shingles theo giá trị của k-grams

def split\_k\_gram(document, shingleNo):

  shingle = []

  shinglesInDocInts = set() # chỉ định lấy duy nhất

  for index in range(len(document) - k\_gram + 1):

    shingle = document[index:index + k\_gram]

    shingle = ' '.join(shingle)

    # năm shingle thành số nguyên (32-bit integer)

    hashed = binascii.crc32(shingle.encode())

    # lọc ra duy nhất

    if hashed not in shinglesInDocInts:

      shinglesInDocInts.add(hashed)

      # tính số lượng shingles

      shingleNo = shingleNo + 1

  return shinglesInDocInts,shingleNo

* *Hàm nextPrimefunc(N)*:Trả về giá trị nhỏ nhất số nguyên tố lớn hơn N

def nextPrimefunc(N):

  # Base case

  if (N <= 1):

      return 2

  prime = N

  found = False

  while(not found):

      prime = prime + 1

      if(isPrime(prime) == True):

          found = True

  return prime

* Hàm *signatureFunction(shingleIDSet)*: tính signature phục vụ cho min hashing

def signatureFunction(shingleIDSet):

  signature = []

  for i in range(0, numHashesFunction):

    minHashCode = nextPrime + 1

    for shingleID in shingleIDSet:

      hashCode = (coeffA[i] \* shingleID + coeffB[i]) % nextPrime

      if hashCode < minHashCode:

        minHashCode = hashCode

    signature.append(minHashCode)

  return signature

* Và các hàm cần thiết khác

def isPrime(n):

  # Corner cases

  if(n <= 1):

      return False

  if(n <= 3):

      return True

  if(n % 2 == 0 or n % 3 == 0):

      return False

  for i in range(5,int(math.sqrt(n) + 1), 6):

      if(n % i == 0 or n % (i + 2) == 0):

          return False

  return True

def pickRandomCoeffs(k):

  randList = []

  while k > 0:

    randIndex = randint(0, maxShingleID)

    while randIndex in randList:

      randIndex = random.randint(0, maxShingleID)

    randList.append(randIndex)

    k = k - 1

  return randList

def FindSimilarity\_Singles(doc, n\_neighbor):

  doc = re.sub("[^\w]", " ", doc).split()

  singlesInput,n = split\_k\_gram(doc,0)

  neighbors\_of\_given\_document\_Test = {}

  for j in range(len(docsAsShingleSets)):

    singles2 = docsAsShingleSets[j]

    s1 = set(singlesInput)

    s2 = set(singles2)

    J = len(s1.intersection(s2)) / float(len(s1.union(s2)))

    # kiểm tra Jaccard similarity có lớn hơn 0 hay ko

    if (float(J) > 0):

        percJ = J \* 100

        neighbors\_of\_given\_document\_Test[j] = percJ

  neighbors\_of\_given\_documentSIGNATURES\_Test= sorted(neighbors\_of\_given\_document\_Test.items(), key=lambda x: x[1], reverse=True)

  return list(neighbors\_of\_given\_documentSIGNATURES\_Test)[:n\_neighbor]

def FindSimilarity\_minhash(doc, n\_neighbor):

  doc = re.sub("[^\w]", " ", doc).split()

  doc,n = split\_k\_gram(doc,0)

  signatureInput = signatureFunction(doc)

  neighbors\_of\_given\_documentSIGNATURES\_Test ={}

  for j in range(len(signatures)):

    signature2 = signatures[j]

    s1 = set(signatureInput)

    s2 = set(signature2)

    J = len(s1.intersection(s2)) / float(len(s1.union(s2)))

    if (float(J) > 0):

        percJ = J \* 100

        neighbors\_of\_given\_documentSIGNATURES\_Test[j] = percJ

  neighbors\_of\_given\_documentSIGNATURES\_Test= sorted(neighbors\_of\_given\_documentSIGNATURES\_Test.items(), key=lambda x: x[1], reverse=True)

  return list(neighbors\_of\_given\_documentSIGNATURES\_Test)[:n\_neighbor]

def get\_band\_hashes(np\_signature, numBand):

  np\_signature  = np.array\_split(np\_signature, numBand)

  hash\_sig = []

  for i in np\_signature:

    j = [hash(x) for x in i]

    hash\_sig.append(sum(j))

  return hash\_sig

def LSH(Test\_input,n\_neighbor):

  Test\_input = re.sub("[^\w]", " ", Test\_input).split()

  Test\_input,n = split\_k\_gram(Test\_input, 0)

  signatureInput = signatureFunction(Test\_input)

  input\_band = get\_band\_hashes(signatureInput, numBand)

  idDoc\_samebucket = []

  result = []

  for i in range(len(input\_band)):

    if input\_band[i] in bucket\_value:

      id = bucket\_value.index(input\_band[i])

      idDoc\_samebucket += [x for x in bucket\_index[id] if x not in idDoc\_samebucket]

  for i in idDoc\_samebucket:

    s1 = set(input\_band)

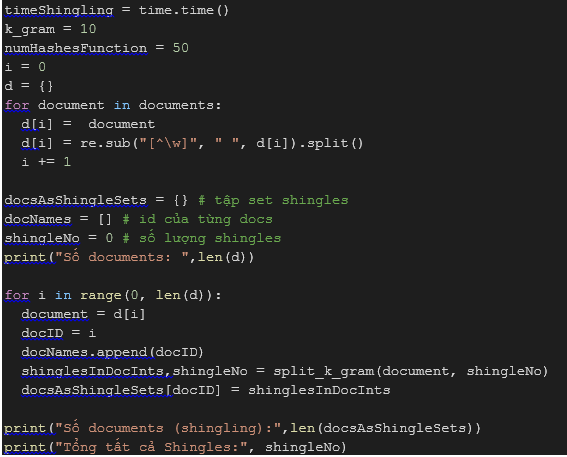
    s2 = set(get\_band\_hashes(signatures[i], numBand))

    sim = len(s1.intersection(s2)) / float(len(s1.union(s2)))\*100

    result.append((sim, i))

  result= sorted(result, key=lambda x: x[0], reverse=True)

  return list(result)[:n\_neighbor]

* + 1. Quá trình Shingling

Hình – Shingling-Tìm ma trận boolean với k-grams = 10

Việc tìm ma trận booleans sẽ là mục tiêu chính của quá trình này, trong thực nghiệm này sẽ chọn giá trị của k-grams là 10.

timeShingling = time.time()

k\_gram = 10

numHashesFunction = 50

i = 0

d = {}

docsAsShingleSets = {} # tập set shingles

docNames = [] # id của từng docs

shingleNo = 0 # số lượng shingles

Sau khi đã chuẩn bị các biến cần thiết, tiến hành cắt data trong file data3.txt thành các word và thay thế các ký từ hoặc ký tự bắt đầu không phải là chữ cái và chữ số bằng “ ” thông qua biểu thức chính quy "[^\w]"

for document in documents:

  d[i] =  document

  d[i] = re.sub("[^\w]", " ", d[i]).split()

  i += 1

Thực hiện gán giá trị cho các biến, tính tổng số các Shingles và thời gian chuyển đổi shingling

for i in range(0, len(d)):

  document = d[i]

  docID = i

  docNames.append(docID)

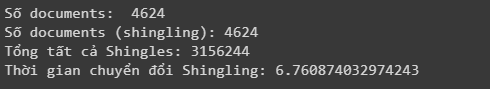
  shinglesInDocInts,shingleNo = split\_k\_gram(document, shingleNo)

  docsAsShingleSets[docID] = shinglesInDocInts

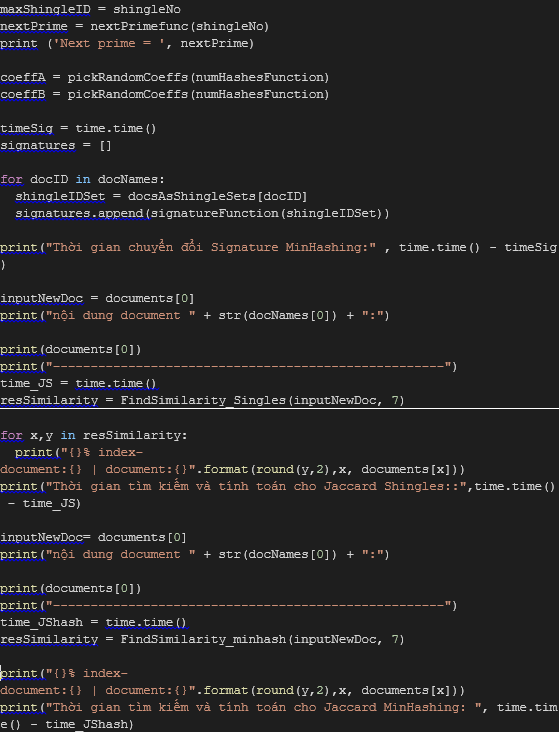
print("Số documents (shingling):",len(docsAsShingleSets))

print("Tổng tất cả Shingles:", shingleNo)

print("Thời gian chuyển đổi Shingling:" , time.time() - timeShingling)

Output của thực nghiệm ở quá trình này là:

Hình – Output of Shingling

* + 1. Quá trình Min Hashing

Hình – MinHashing và so sánh tốc độ tính similarity của shingles và signatures

Trước hết tạo ma trận signature với giá trị của hash function là 50

maxShingleID = shingleNo

nextPrime = nextPrimefunc(shingleNo)

print ('Next prime = ', nextPrime)

coeffA = pickRandomCoeffs(numHashesFunction)

coeffB = pickRandomCoeffs(numHashesFunction)

timeSig = time.time()

signatures = []

for docID in docNames:

  shingleIDSet = docsAsShingleSets[docID]

  signatures.append(signatureFunction(shingleIDSet))

print("Thời gian chuyển đổi Signature MinHashing:" , time.time() - timeSig)

Và thời gian quá trình chuyền đổi này là:



Tính Jaccard similarity của shingles

inputNewDoc= documents[0]

print("nội dung document " + str(docNames[0]) + ":")

print(documents[0])

print("----------------------------------------------------")

time\_JS = time.time()

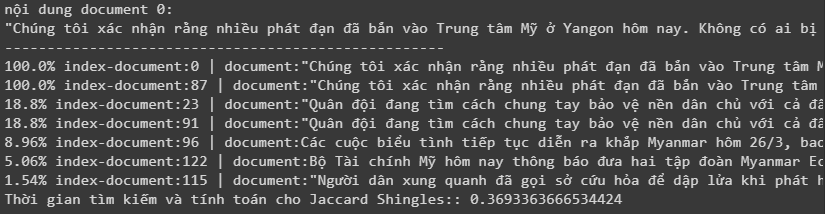
resSimilarity = FindSimilarity\_Singles(inputNewDoc, 7)

for x,y in resSimilarity:

  print("{}% index-document:{} | document:{}".format(round(y,2),x, documents[x]))

print("Thời gian tìm kiếm và tính toán cho Jaccard Shingles::",time.time() - time\_JS)

Output của bước này như sau:

Tính Jaccard similarity của MinHashing theo signatures

Hình – Output của bước tìm kiếm và tính toán similarity Jaccard của các Shingles

inputNewDoc= documents[0]

print("nội dung document " + str(docNames[0]) + ":")

print(documents[0])

print("----------------------------------------------------")

time\_JShash = time.time()

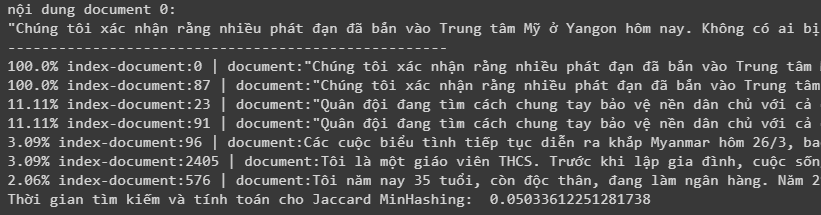
resSimilarity = FindSimilarity\_minhash(inputNewDoc, 7)

for x,y in resSimilarity:

  print("{}% index-document:{} | document:{}".format(round(y,2),x, documents[x]))

print("Thời gian tìm kiếm và tính toán cho Jaccard MinHashing: ", time.time() - time\_JShash)

Output sau khi thực hiện đoạn code trên như sau:

Dự vào output của việc tìm kiếm và tính toán similarity Jaccard của từng quá trình, ta có thể dễ dàng thấy rằng, việc sử dụng min-hashing dữ liệu trước sẽ giúp cho việc tìm kiếm và so sánh sự tương đồng của văn bản nhanh hơn Shingles nhiều lần.

Hình – Output của bước tìm kiếm và tính toán similarity Jaccard của Min-Hashing

* + 1. Quá trình Locallity Sensitive Hashing (LSH)

time\_LSH = time.time()

numBand = 50

bucket\_value = []

bucket\_index = []

for k in range(len(signatures)):

  for i in get\_band\_hashes(signatures[k], numBand):

    if i not in bucket\_value:

      bucket\_value.append(i)

      bucket\_index.append([k])

    else:

      index = bucket\_value.index(i)

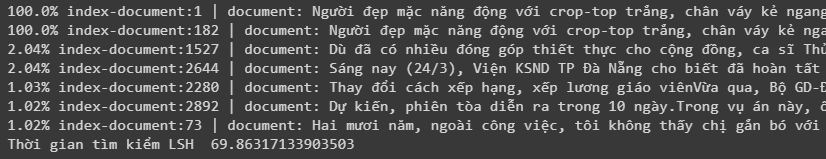
      bucket\_index[index].append(k)

lst\_result = LSH(documents[1], 7)

for x,y in lst\_result:

  print("{}% index-document:{} | document: {}".format(round(x,2),y, documents[y]))

print("Thời gian tìm kiểm LSH ", time.time() - time\_LSH)



Hình – Output của quá trình LSH

* 1. Chương trình thực nghiệm có áp dụng PySpark
     1. Quá trình chuẩn bị

Khai báo các thư viện cần thiết phục vụ cho việc thực nghiệm PySpark

from bs4 import BeautifulSoup

import sys

import os.path

import string

import os

import re

import random

import time

import binascii

import math

import numpy as np

import pyspark

from pyspark import SparkConf, SparkContext

from pyspark.sql import SparkSession

import collections

import time

import itertools

from google.colab import drive

drive.mount('/content/drive',force\_remount=True)

f = open("/content/drive/MyDrive/BIGDATA/BTL/data3.txt", "r", encoding="utf-8")

data = f.read().split("\n")

* + 1. Các hàm hỗ trợ có trong chương trình thực nghiệm
* Hàm *split\_k\_gram(document, shingleNo)*: được dùng để chia, cắt ghép chuỗi dữ liệu trong file văn bản cần kiểm tra độ tương tự thành các shingles theo giá trị của k-grams

def split\_k\_gram(document, shingleNo):

  shingle = []

  shinglesInDocInts = set() # chỉ định lấy duy nhất

  for index in range(len(document) - k\_gram + 1):

    shingle = document[index:index + k\_gram]

    shingle = ' '.join(shingle)

    hashed = binascii.crc32(shingle.encode())

    if hashed not in shinglesInDocInts:

      shinglesInDocInts.add(hashed)

      shingleNo = shingleNo + 1

  return shinglesInDocInts,shingleNo

* Hàm *nextPrimefunc(N)*:Trả về giá trị nhỏ nhất số nguyên tố lớn hơn N

def nextPrimefunc(N):

  # Base case

  if (N <= 1):

      return 2

  prime = N

  found = False

  while(not found):

      prime = prime + 1

      if(isPrime(prime) == True):

          found = True

  return prime

* Hàm *signatureFunction(shingleIDSet)*: tính signature phục vụ cho min hashing

def signatureFunction(shingleIDSet):

  signature = []

  for i in range(0, numHashesFunction):

    minHashCode = nextPrime + 1

    for shingleID in shingleIDSet:

      hashCode = (coeffA[i] \* shingleID + coeffB[i]) % nextPrime

      if hashCode < minHashCode:

        minHashCode = hashCode

    signature.append(minHashCode)

  return signature

* Và các hàm cần thiết khác

def isPrime(n):

  # Corner cases

  if(n <= 1):

      return False

  if(n <= 3):

      return True

  if(n % 2 == 0 or n % 3 == 0):

      return False

  for i in range(5,int(math.sqrt(n) + 1), 6):

      if(n % i == 0 or n % (i + 2) == 0):

          return False

  return True

def pickRandomCoeffs(k):

  randList = []

  while k > 0:

    randIndex = randint(0, maxShingleID)

    while randIndex in randList:

      randIndex = random.randint(0, maxShingleID)

    randList.append(randIndex)

    k = k - 1

  return randList

def FindSimilarity\_Singles(doc, n\_neighbor):

  doc = re.sub("[^\w]", " ", doc).split()

  singlesInput,n = split\_k\_gram(doc,0)

  neighbors\_of\_given\_document\_Test = {}

  for j in range(len(docsAsShingleSets)):

    singles2 = docsAsShingleSets[j]

    s1 = set(singlesInput)

    s2 = set(singles2)

    J = len(s1.intersection(s2)) / float(len(s1.union(s2)))

    # kiểm tra Jaccard similarity có lớn hơn 0 hay ko

    if (float(J) > 0):

        percJ = J \* 100

        neighbors\_of\_given\_document\_Test[j] = percJ

  neighbors\_of\_given\_documentSIGNATURES\_Test= sorted(neighbors\_of\_given\_document\_Test.items(), key=lambda x: x[1], reverse=True)

  return list(neighbors\_of\_given\_documentSIGNATURES\_Test)[:n\_neighbor]

def FindSimilarity\_minhash(doc, n\_neighbor):

  doc = re.sub("[^\w]", " ", doc).split()

  doc,n = split\_k\_gram(doc,0)

  signatureInput = signatureFunction(doc)

  neighbors\_of\_given\_documentSIGNATURES\_Test ={}

  for j in range(len(signatures)):

    signature2 = signatures[j]

    s1 = set(signatureInput)

    s2 = set(signature2)

    J = len(s1.intersection(s2)) / float(len(s1.union(s2)))

    if (float(J) > 0):

        percJ = J \* 100

        neighbors\_of\_given\_documentSIGNATURES\_Test[j] = percJ

  neighbors\_of\_given\_documentSIGNATURES\_Test= sorted(neighbors\_of\_given\_documentSIGNATURES\_Test.items(), key=lambda x: x[1], reverse=True)

  return list(neighbors\_of\_given\_documentSIGNATURES\_Test)[:n\_neighbor]

def get\_band\_hashes(np\_signature, numBand):

  np\_signature  = np.array\_split(np\_signature, numBand)

  hash\_sig = []

  for i in np\_signature:

    j = [hash(x) for x in i]

    hash\_sig.append(sum(j))

  return hash\_sig

def LSH(Test\_input,n\_neighbor):

  Test\_input = re.sub("[^\w]", " ", Test\_input).split()

  Test\_input,n = split\_k\_gram(Test\_input, 0)

  signatureInput = signatureFunction(Test\_input)

  input\_band = get\_band\_hashes(signatureInput, numBand)

  idDoc\_samebucket = []

  result = []

  for i in range(len(input\_band)):

    if input\_band[i] in bucket\_value:

      id = bucket\_value.index(input\_band[i])

      idDoc\_samebucket += [x for x in bucket\_index[id] if x not in idDoc\_samebucket]

  for i in idDoc\_samebucket:

    s1 = set(input\_band)

    s2 = set(get\_band\_hashes(signatures[i], numBand))

    sim = len(s1.intersection(s2)) / float(len(s1.union(s2)))\*100

    result.append((sim, i))

  result= sorted(result, key=lambda x: x[0], reverse=True)

  return list(result)[:n\_neighbor]

* + 1. Quá trình Shingling

Khởi tạo biến *conf, sc, rdd* lần lượt sử dụng SparkConf(), SparkContext().getOrCreate(conf=con*f)* và *sc.parallelize(data)* để phục vụ cho quá trình thực hiện song song. Đồng thời, sau khi đã crawl dữ liệu hoặc đã có sẵn văn bản cần kiểm tra độ tương tự (ở đây là file *data3.txt*) có trong tập văn bản, tiến hành đọc file văn bản cần kiểm tra bằng phương thức open() và cắt bỏ việc xuống dòng bằng split(“\n”) để chuẩn bị dữ liệu để tiến hành thực hiện chương trình – quá trình này sẽ được thực hiện song song

f = open("/content/drive/MyDrive/BIGDATA/BTL/data3.txt", "r", encoding="utf-8")

data = f.read().split("\n")

# ----- Ket thuc qua trinh doc file va split xuong dong ----

conf = SparkConf().setMaster("local").setAppName("BigDataLSH")

sc = SparkContext.getOrCreate(conf=conf)

rdd = sc.parallelize(data)

d =rdd.map(lambda x: re.sub("[^\w]", " ", x).split())

timeShingling = time.time()

k\_gram = 10

numHashesFunction = 50

docsAsShingleSetsRDD = d.map(lambda x : split\_k\_gram(x))

shingleNo = sum(docsAsShingleSetsRDD.map(lambda x: len(x)).collect())

print("Thời gian chuyển đổi Shingling:" , time.time() - timeShingling)

Thời gian chuyển đổi Shingling sẽ mang giá trị như sau:



* + 1. Quá trình Min Hashing

Bước vào quá trình min hashing, dữ liệu sẽ được chuyển đổi với ma trận signature với giá trị của hash function là 50

docsAsShingleSets = docsAsShingleSetsRDD.collect()

maxShingleID = shingleNo

nextPrime = nextPrimefunc(shingleNo)

print ('Next prime = ', nextPrime)

coeffA = pickRandomCoeffs(numHashesFunction)

coeffB = pickRandomCoeffs(numHashesFunction)

timeSig = time.time()

docNames = [i for i in range(len(docsAsShingleSets))]

signatures = docsAsShingleSetsRDD.map(lambda x: signatureFunction(x)).collect()

numDocs = len(docsAsShingleSets)

print("Thời gian chuyển đổi Signature MinHashing:" , time.time() - timeSig)

Và thời gian của quá trình chuyển đổi này là:



Tính Jaccard similarity của shingles

inputNewDoc= data[0]

print(data[0])

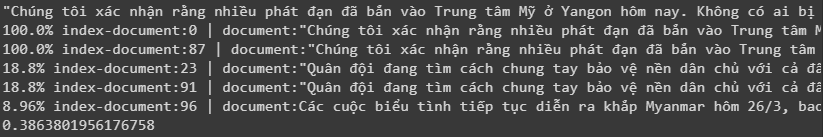
time\_JS = time.time()

resSimilarity = FindSimilaryty\_Singles(inputNewDoc, 5)

for x,y in resSimilarity:

  print("{}% index-document:{} | document:{}".format(round(y,2),x, data[x]))

print(time.time() - time\_JS)

Output của bước này như sau:

Hình – Output của bước tìm kiếm và tính toán similarity Jaccard của Shingles (chương trình thực nghiệm có áp dụng PySpark)

Tính Jaccard similarity của signature

inputNewDoc= data[0]

inputNewDoc= data[0]

print(data[0])

time\_JShash = time.time()

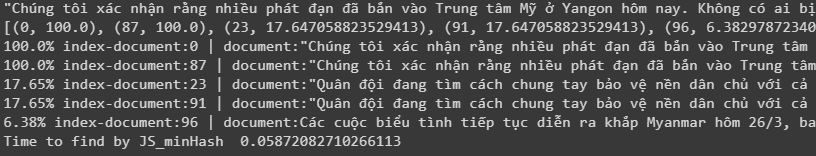
resSimilarity = FindSimilaryty(inputNewDoc, 5)

print(resSimilarity)

for x,y in resSimilarity:

  print("{}% index-document:{} | document:{}".format(round(y,2),x, data[x]))

print("Time to find by JS\_minHash ", time.time() - time\_JShash)

Output của bước này như sau:

Hình – Output của bước tìm kiếm và tính toán similarity Jaccard của MinHashing (chương trình thực nghiệm có áp dụng PySpark

* + 1. Quá trình Locallity Sensitive Hashing (LSH)

time\_LSH = time.time()

numBand = 50

bucket\_value = []

bucket\_index = []

# test = [signatures[0], signatures[0], signatures[0],signatures[1]]

for k in range(len(signatures)):

  for i in get\_band\_hashes(signatures[k], numBand):

    if i not in bucket\_value:

      bucket\_value.append(i)

      bucket\_index.append([k])

    else:

      index = bucket\_value.index(i)

      bucket\_index[index].append(k)

Test\_input = data[1]

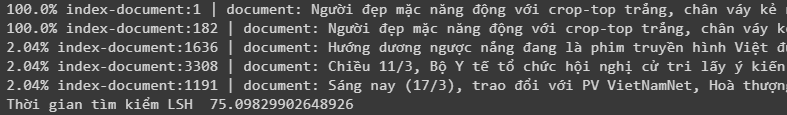
lst\_result = LSH(Test\_input, 5)

for x,y in lst\_result:

  print("{}% index-document:{} | document: {}".format(round(x,2),y, data[y]))

print("Thời gian tìm kiểm LSH ", time.time() - time\_LSH)

Thời gian tìm kiếm LSH là



Hình – Thời gian tìm kiếm LSH (chương trình thực nghiệm có áp dụng PySpark)

TÀI LIỆU THAM KHẢO

**Tiếng Anh**

1. <http://infolab.stanford.edu/~ullman/mmds/ch3.pdf>
2. <https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134>
3. <https://www.learndatasci.com/tutorials/building-recommendation-engine-locality-sensitive-hashing-lsh-python/>