Hàm build\_shingles(document: DataFrame):

Shingles = [] // Khởi tạo một mảng lưu trữ các shingle

For i từ 0 đến (độ dài document - k+1) (k: số lượng từ trong mỗi shingle)

//Thêm một đoạn văn bản từ vị trí i đén i+k và thêm vào shingles[]

shingles.append(document[i : i + self.k])

Return một tập shingles

**hàm shingling(self, documents):**

# Tạo một DataFrame mới để lưu trữ kết quả shingling

tạo một DataFrame mới có tên shingled\_documents

# Thêm cột mới vào DataFrame mới với các shingles

Apply một cột mới có tên 'shingles' vào shingled\_documents

For mỗi tài liệu trong documents[0]:

áp dụng hàm build\_shingles lên tài liệu và thêm kết quả vào cột 'shingles'

# Tạo danh sách các shingles từ DataFrame mới

Tạo một danh sách có tên elements bằng cách lấy cột 'shingles' từ shingled\_documents và chuyển nó thành một danh sách

# Tính toán từ vựng từ danh sách shingles

Set self.vocab chứa các phần tử không trùng trong danh sách elements

# Tạo các vector boolean từ danh sách shingles và từ vựng

Apply một cột mới vào shingled\_documents có tên boolean\_vectors

với mỗi hàng trong shingled\_documents['shingles']:

tạo một danh sách các 0 và 1 trong đó 1 chỉ ra rằng phần tử tương ứng trong self.vocab có mặt trong hàng

thêm danh sách này vào cột boolean\_vectors

Minhashing

Hàm support:

hàm build\_array\_minhash(self, len\_vocab):

array\_minhash = [] //Khởi tạo danh sách minhash rỗng

For self.resolution lần:

one\_minhash = tạo một danh sách các số từ 1 đến len(vocab)

trộn ngẫu nhiên danh sách one\_minhash

thêm one\_minhash vào array\_minhash

trả về array\_minhash

hàm get\_signature(self, shingle, len\_vocab):

signature = [] //Tạo mảng signature rỗng

For hàm trong self.minhash\_array:

For từ 1 đến len\_vocab:

idx = chỉ số của phần tử hiện tại trong hàm

Signature\_val = giá trị tại chỉ số idx trong shingle

If Signature\_val bằng 1:

thêm idx vào signature

thoát khỏi vòng lặp hiện tại

trả về signature

Hàm chính:  
hàm minhashing(self, bool\_vectors):

self.minhash\_array = gọi hàm build\_array\_minhash với tham số là độ dài của self.vocab

signatures = áp dụng hàm get\_signature lên từng hàng của bool\_vectors

trả về DataFrame chứa các signatures

hàm locality\_sensity\_hashing(self, signatures):

# Khởi tạo các bucket

buckets = []

counter = 0

# Khởi tạo các bucket

For self.band lần:

Append một dictionary mới vào buckets

# Thêm các signatures vào các bucket

For từng dòng trong signatures:

L = độ dài của signature[0]

kiểm tra xem độ dài có chia hết cho self.band không

tính số lượng ký tự trong mỗi sub-signature, r = độ dài / self.band

For self.band lần:

sub-signature = giá trị từ vị trí i\*r đến (i+1)\*r trong signature[0]

chuyển sub-signature thành chuỗi và lưu vào biến subvec

If subvec chưa có trong bucket[i]:

buckets[i][subvec] = []

// tạo một danh sách mới trong bucket[i] với key là subvec

Append counter vào danh sách trong bucket[i][subvec]

tăng counter lên 1

trả về buckets

hàm cosine\_similarity(self, vector1, vector2):

# Chuyển các phần tử của vector1 và vector2 thành số nguyên

vector1 = [chuyển x thành số nguyên nếu x là số]

vector2 = [chuyển x thành số nguyên nếu x là số]

# Tính toán độ tương tự

dot\_product = tính tổng của [i \* j cho mỗi i, j trong zip(vector1, vector2)]

magnitude\_vector1 = căn bậc hai của tổng của [i \* i cho mỗi i trong vector1]

magnitude\_vector2 = căn bậc hai của tổng của [i \* i cho mỗi i trong vector2]

If magnitude\_vector1 bằng 0 hoặc magnitude\_vector2 bằng 0:

trả về 0

ngược lại:

similarity = dot\_product / (magnitude\_vector1 \* magnitude\_vector2)

trả về similarity

hàm approxNearestNeighbors(self, key, n):

similarities = {}

# Lặp qua từng bucket trong các bucket bands

For bucket\_band trong self.buckets:

# Lặp qua từng bucket hits trong bucket band hiện tại

For bucket\_hits trong bucket\_band.values():

# Lặp qua từng chỉ số tài liệu trong bucket hits

For doc\_index trong bucket\_hits:

# Lấy nội dung tài liệu tương ứng với chỉ số doc\_index

doc\_text = self.documents.iloc[doc\_index][0]

# Tính độ tương tự giữa key và doc\_text

similarity = self.cosine\_similarity(key, doc\_text)

# Lưu chỉ số tài liệu và độ tương tự vào từ điển similarities

similarities[doc\_index] = similarity

# Sắp xếp các cặp (chỉ số tài liệu, độ tương tự) giảm dần theo độ tương tự

sorted\_similarities = sắp xếp similarities.items() theo key=lambda x: x[1], reverse=True

# Lấy n kết quả đầu tiên

top\_n\_results = lấy n phần tử đầu tiên từ sorted\_similarities

# Trả về danh sách (tài liệu, độ tương tự) của n kết quả gần nhất

trả về [(self.documents.iloc[doc\_index][0], similarity) cho mỗi (doc\_index, similarity) trong top\_n\_results]