Progress II presentation

Subject: Mining Massive Datasets

Instructor: Msc **Nguyễn Thành An**

Table of contents

1. Motivating problem

2. Introduction to MinHashLSH algorithm

3. Task 1: In-memory MinhashLSH

4. Task 2: LargeDataMinhashLSH

Motivating problem

Consider the following problem:

We have several billion documents, and we want to identify near-duplicate or similar documents.

How can we solve this efficiently?

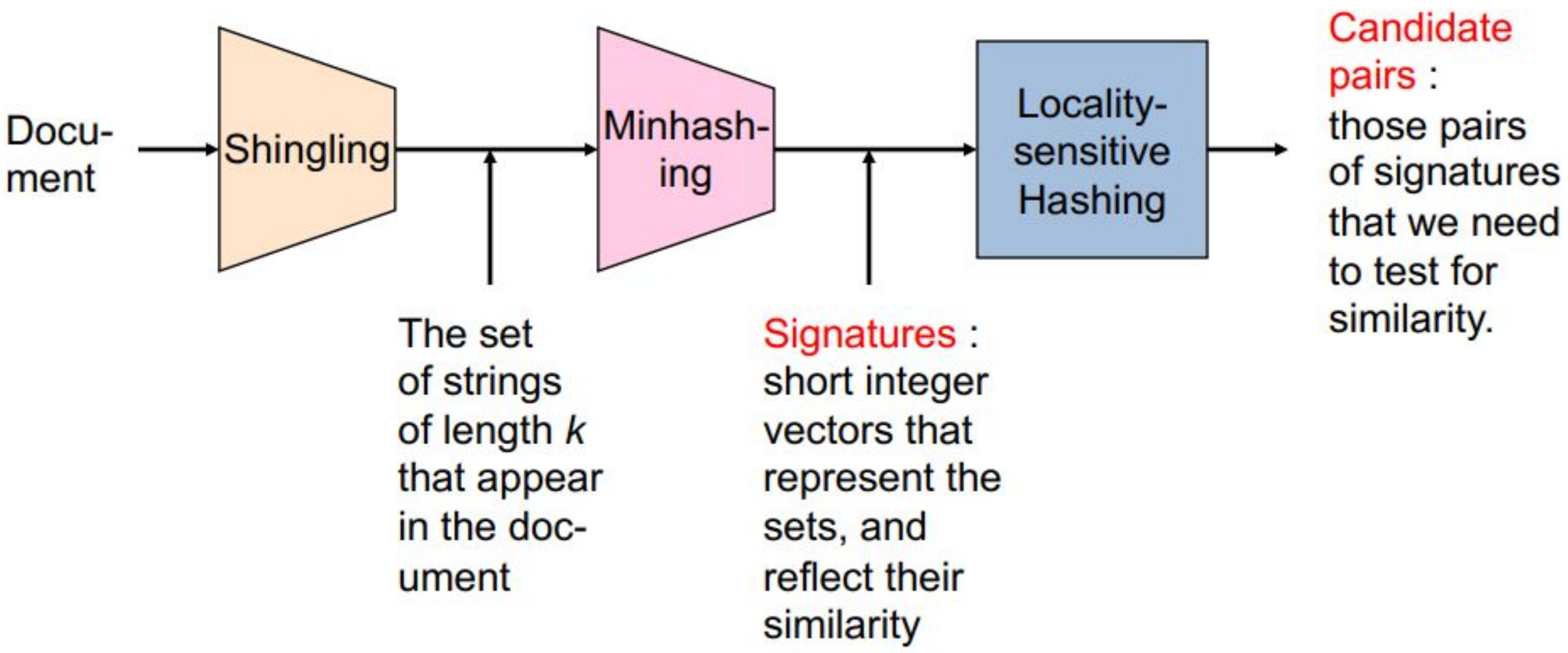
If we do naïve approach, we would go through each pair-wise document in the collection □ but that is too slow

We need an efficient way to cluster similar documents

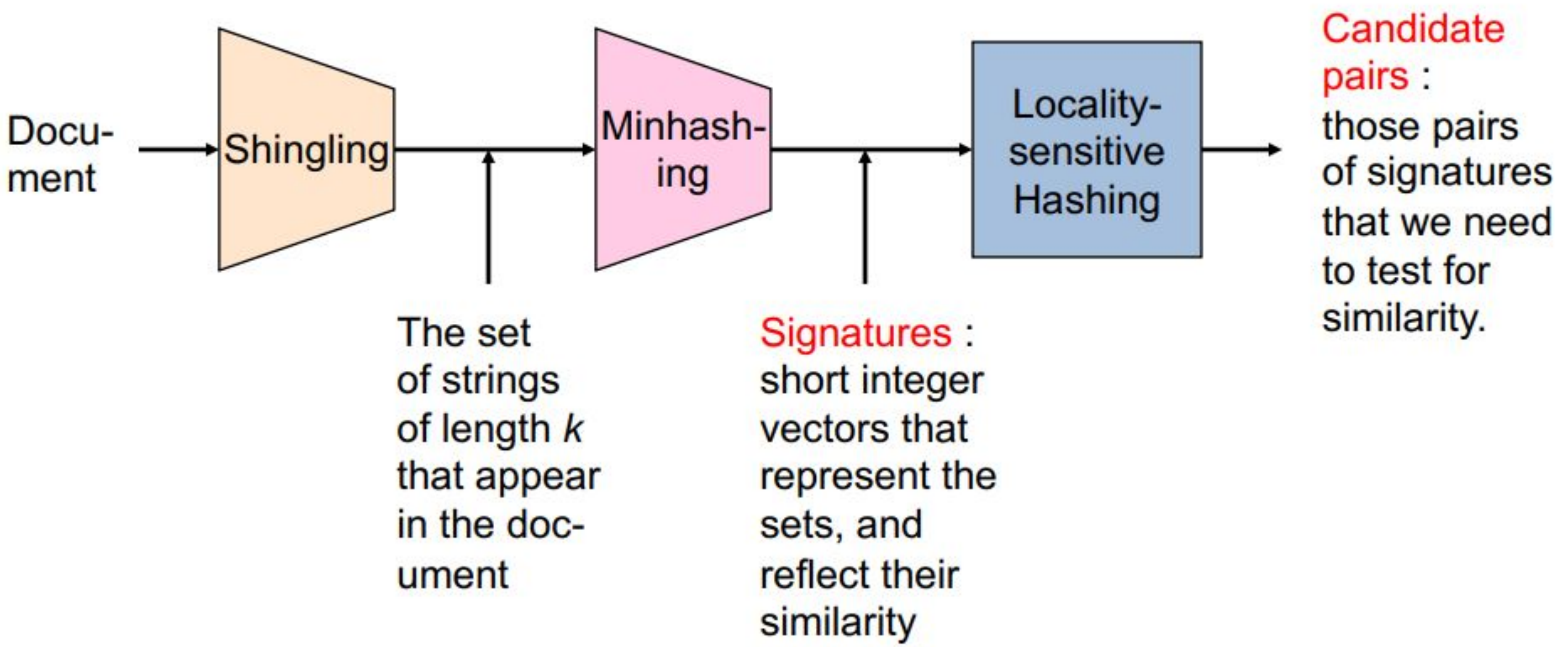
□ Locality sensitive hashing

Motivating problem

The Big Picture

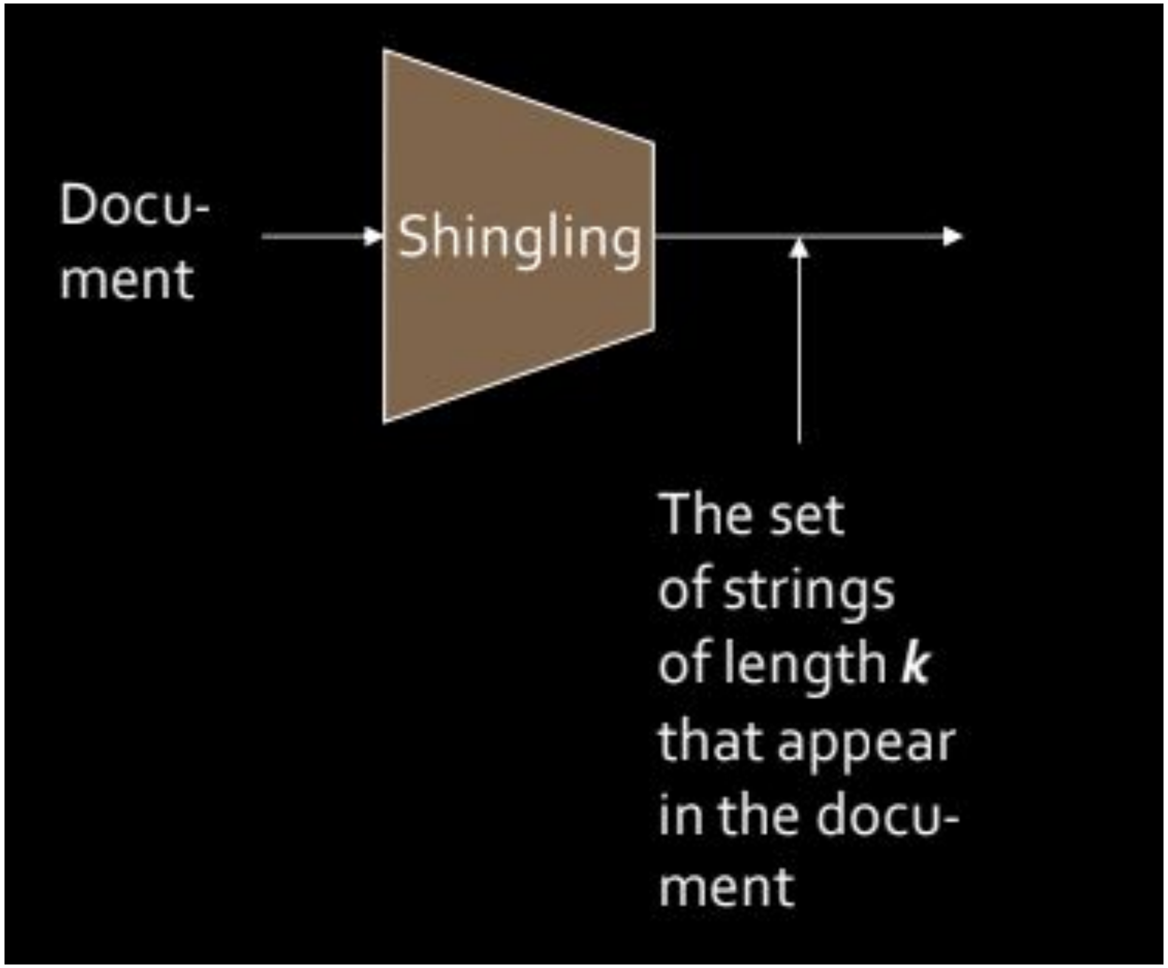


Introduction to MinHashLSH algorithm



Introduction to MinHashLSH algorithm

Shingling: Convert a document into a set



Introduction to MinHashLSH algorithm

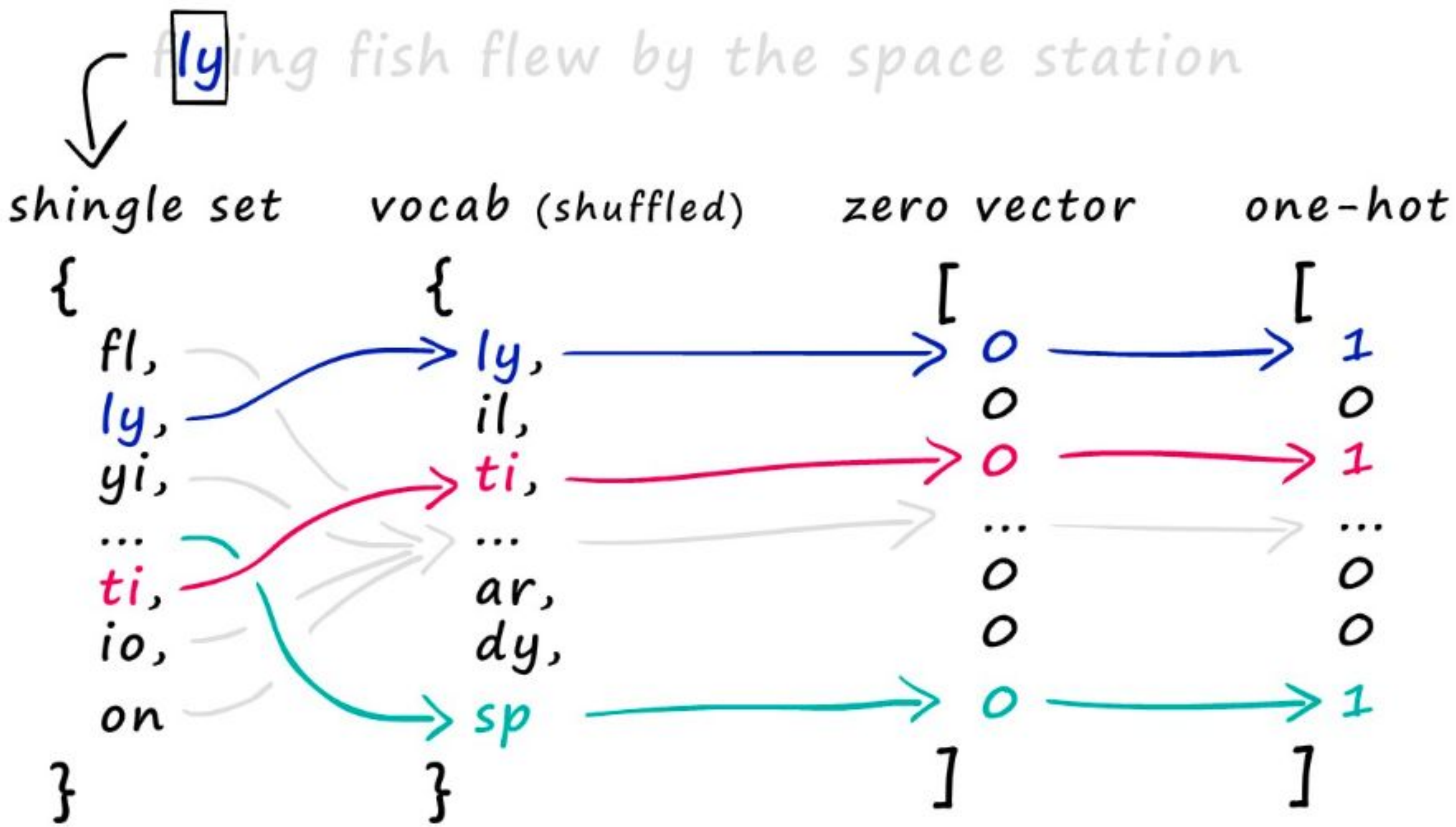
Example of Shingling with k = 2

flying fish flew by the space station

{fl, ly, yi, in, ng, g_, _f, fi, is, sh
h_ … st, ta, at, ti, io, on}

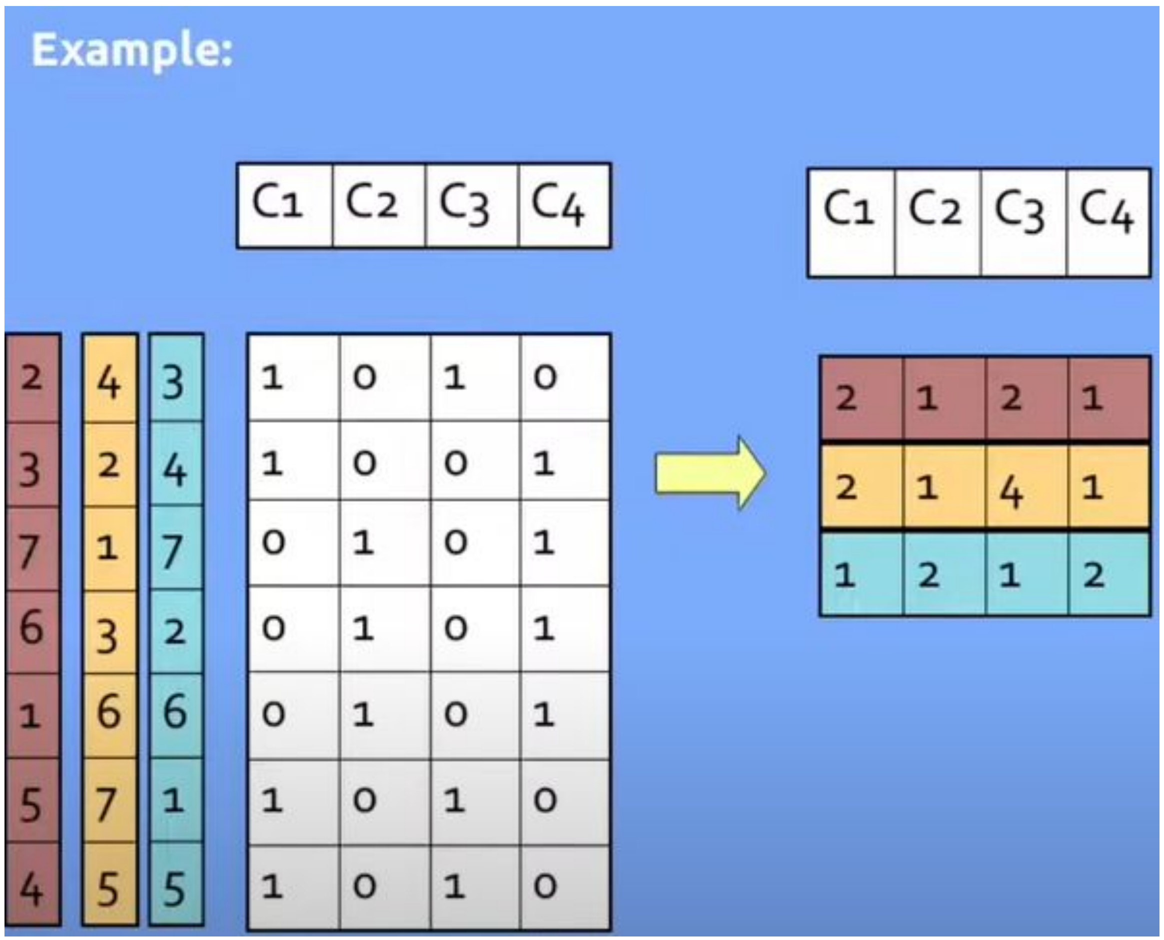
Introduction to MinHashLSH algorithm

From set to boolean matrices



Introduction to MinHashLSH algorithm

MinHashing: Convert *large set* to short signatures, <u>while preserving similarity</u>



At the end of this, we produce our minhash signature — or dense vector.

Introduction to MinHashLSH algorithm

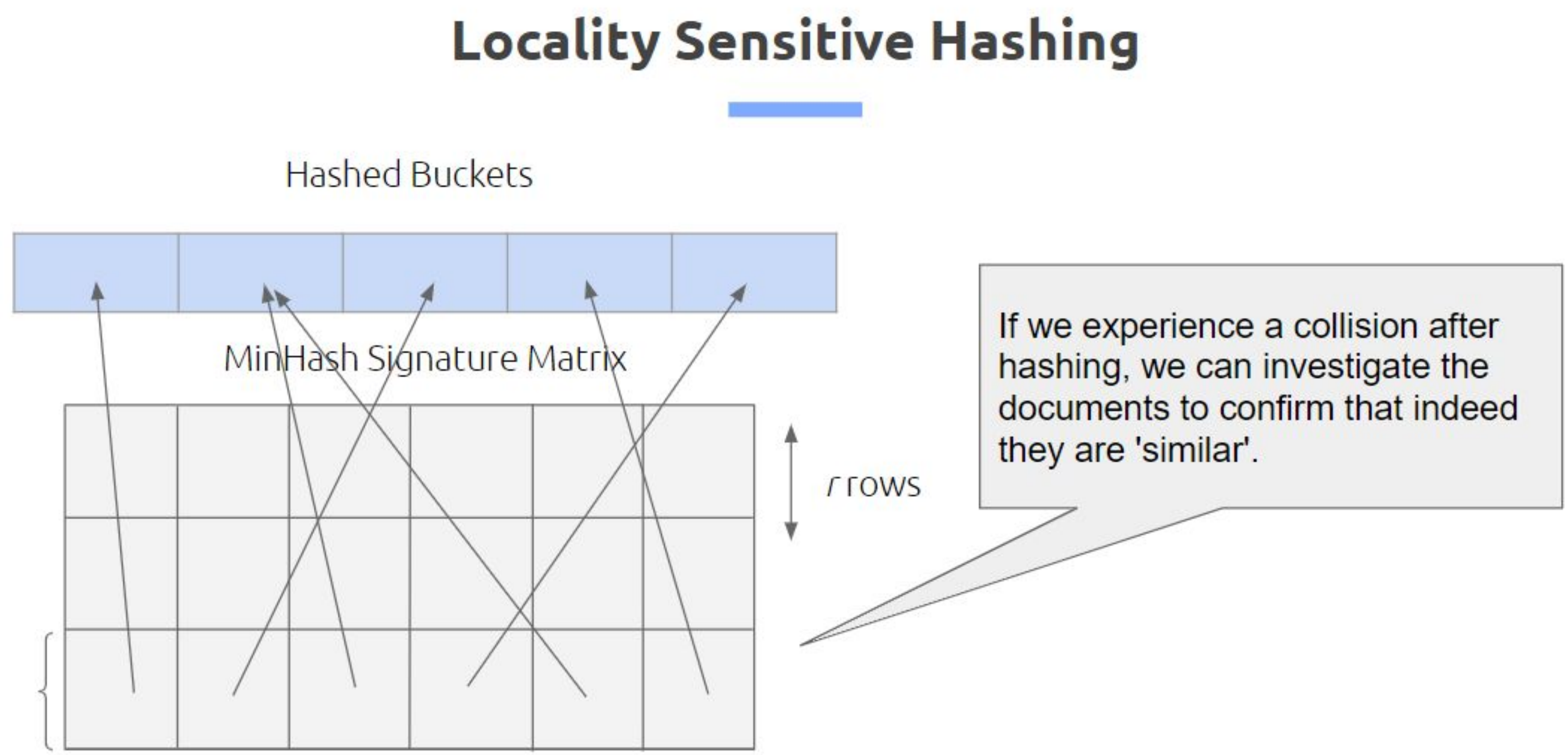
How can we find out two documents are similar?

→ We use Jaccard similarity to calculate the similarity between their signature vectors

Introduction to MinHashLSH algorithm

Locality Sensitive Hashing

- We now obtained a signature matrix, across the different permutations for each document. The goal is to identify across the matrix similar columns.
- The trick is to break down the matrix into bands, and collapse portions of each band, hash it, and then identify others that fall under the same bucket.



Introduction to MinHashLSH algorithm

Advantages of MinHashLSH algorithm

- Scalability: MinHash LSH is highly scalable and efficient for large datasets. It can handle millions or even billions of items efficiently.
- Space Efficiency: MinHash LSH reduces the dimensionality of the data, which leads to significant savings in storage space and computational resources.
- Versatility: It can be applied to various types of data, including text, images, and other high-dimensional data, making it a versatile solution for similarity search problems.
- Speed: MinHash LSH can perform similarity searches in sub-linear time, which means that the search time does not increase linearly with the size of the dataset.

Introduction to MinHashLSH algorithm

Disadvantages of MinHashLSH algorithm

- Approximate Results: While MinHash LSH is efficient, it sacrifices accuracy for speed and scalability. The results provided by the algorithm are approximate and may not always be perfectly accurate.
- Sensitivity to Parameters: The performance of MinHash LSH can be sensitive to the choice of parameters, such as the number of hash functions and the number of hash tables. Tuning these parameters for optimal performance can be challenging.
- False Positives/Negatives: The create Signature method can produce false negatives and false positives

Task 1

We use pure in-memory processing operation and encapsulated it into corresponding classes:

InMemoryMinHashLSH
+ InMemoryMinHashLSH (documents: DataFrame) + shingling(documents: DataFrame): DataFrame + minhashing(bool_vectors: DataFrame): DataFrame + locality_sensity_hashing(signatures: list): DataFrame + run(): void + approxNearestNeighbors(key, n): DataFrame

Pseudocode for Task 1

```python
def shingling(self, documents):
    """Converts documents to sets of k-shingles and maps them to indices."""
    shingle_dict = {}
    all_shingles = set()

    for index, doc in documents.items():
        shingles = set(doc[i:i+self.k] for i in range(len(doc) - self.k + 1))
        shingle_dict[index] = shingles
        all_shingles.update(shingles)

    # Map all shingles to indices
    self.shingle_index = {shingle: idx for idx, shingle in enumerate(all_shingles)}
    self.num_shingles = len(all_shingles)

    # Initialize a sparse matrix
    num_docs = len(documents)
    self.sparse_matrix = lil_matrix((num_docs, self.num_shingles), dtype=bool)
    for doc_id, shingles in shingle_dict.items():
        indices = [self.shingle_index[shingle] for shingle in shingles]
        self.sparse_matrix[doc_id, indices] = True

    # Initialize hash coefficients after shingling
    self.a_coefs = np.random.randint(1, self.num_shingles, size=self.num_perm)
    self.b_coefs = np.random.randint(0, self.num_shingles, size=self.num_perm)

    return self.sparse_matrix, self.shingle_index
```

Pseudocode for Task 1

```
Function shingling(documents):
  Initialize shingle_dict as an empty dictionary
  Initialize all_shingles as an empty set

  For each document in documents:
     Create shingles from the document
     Add shingles to shingle_dict
     Update all_shingles with new shingles

  Create shingle_index mapping each shingle to a unique index
  Set num_shingles to the length of all_shingles

  Initialize sparse_matrix as a sparse matrix with dimensions (number of documents, num_shingles)

  For each document's shingles:
     Get indices for the shingles
     Set corresponding entries in sparse_matrix to True

  Initialize hash coefficients a_coefs and b_coefs with random integers

  Return sparse_matrix and shingle_index
```

Pseudocode for Task 1

```
Function minhashing(sparse_matrix, shingle_index):
        Set num_shingles to length of shingle_index
        Initialize signatures as an array of size (number of documents, num_perm)
    filled with infinity

        For each permutation i:
            Calculate hash values using coefficients a and b
            For each document:
                Get shingle indices from sparse_matrix
                Find minimum hash value for the document
                Update signatures array

        Return signatures as a DataFrame
```

Pseudocode for Task 1

```
Function locality_sensitivity_hashing(signatures):
    Set rows_per_band to num_perm divided by num_bands
    Initialize signature_to_bucket as an empty list

    For each document's signature:
        For each band:
            Get the band_signature
            Calculate bucket_key by hashing the band_signature
            Append document's signature index and bucket_key to signature_to_bucket

    Convert signature_to_bucket to a DataFrame
    Return signature_bucket_df
```

Pseudocode for Task 1

```
Function approxNearestNeighbors(query_doc, n):
    Create shingles from query_doc
    Get indices for query shingles

    If no query_indices match, return empty list

    Initialize query_sparse_matrix with dimensions (1, num_shingles)
    Set corresponding entries in query_sparse_matrix to True for valid indices

    Calculate query_signatures using hash functions

    Initialize similarities as an empty dictionary

    For each document's signature:
        Calculate Jaccard similarity with query signature
        Add similarity to similarities dictionary

    Sort similarities in descending order and return top n results
```

Pseudocode for Task 1

Function jaccard_similarity(set1, set2):

Calculate intersection and union of set1 and set2

Return intersection divided by union if union is not empty, else return 0

Function run():

Execute shingling to get sparse_matrix and shingle_index

Generate signatures using minhashing

Perform locality sensitivity hashing on signatures

Return the resulting signature_bucket_df

Result of Task 1

```
query_doc = 'Phytoplasmas are insect-vectored bacteria that cause disease
top_n_results = minhash_lsh.approxNearestNeighbors(query_doc, 10)
```

```
Nearest Neighbors based on Jaccard Similarity:
Document ID: 0, Similarity: 1.0000
Document ID: 758, Similarity: 0.1800
Document ID: 906, Similarity: 0.1735
Document ID: 3121, Similarity: 0.1717
Document ID: 4654, Similarity: 0.1717
Document ID: 555, Similarity: 0.1709
Document ID: 3249, Similarity: 0.1707
Document ID: 2305, Similarity: 0.1700
Document ID: 1536, Similarity: 0.1683
Document ID: 3274, Similarity: 0.1683
```

Task 2

We re-implement the requirements from task 1 using PySpark

+ shingling(documents: DataFrame): DataFrame
+ minhashing(bool_vectors: DataFrame): DataFrame
+ locality_sensity_hashing(signatures: list): DataFrame
+ run(): void
+ approxNearestNeighbors(key, n): DataFrame

Pseudocode for Task 2

```
function LargeScaleMinHashLSH(spark)
    self.spark ← spark
    self.documents ← null
    self.shingles ← null
    self.signatures ← null
    self.hash_buckets ← null

    function shingling(documents)
        self.shingles ← documents.select("doc_id", "text")
            .rdd.map(lambda x: (x[0], x[1].split()))
            .toDF(["doc_id", "shingles"])
```

Pseudocode for Task 2

```
function minhashing(documents, num_hash_functions=100)
    exploded_shingles ← self.shingles.select("doc_id", explode("shingles").alias("shingle"))
    hash_values ← exploded_shingles.select("shingle")
        .distinct().rdd.map(lambda x: (x[0], [hash(x[0]) % num_hash_functions for _ in range(num_hash_functions)]))
        .toDF(["shingle", "hash_values"])

    self.signatures ← exploded_shingles.join(hash_values, exploded_shingles.shingle == hash_values.shingle)
        .groupBy("doc_id").agg(collect_list("hash_values").alias("hash_values"))

function locality_sensity_hashing(documents, num_hash_buckets=10)
    self.hash_buckets ← self.signatures.rdd.flatMap(lambda x: [(tuple(h), x[0]) for h in x[1]])
        .map(lambda x: ((hash(x[0]), x[1]), x[0]))
        .groupByKey().map(lambda x: (x[0][0] % num_hash_buckets, [x[0][1]]))
        .reduceByKey(lambda x, y: x + y).collect()
function run()
    shingling(self.documents)
    minhashing(self.documents)
    locality_sensity_hashing(self.documents)
```

Pseudocode for Task 2

```
function minhashing(documents, num_hash_functions=100)
    exploded_shingles ← self.shingles.select("doc_id", explode("shingles").alias("shingle"))
    hash_values ← exploded_shingles.select("shingle")
        .distinct().rdd.map(lambda x: (x[0], [hash(x[0]) % num_hash_functions for _ in
range(num_hash_functions)]))
        .toDF(["shingle", "hash_values"])

    self.signatures ← exploded_shingles.join(hash_values, exploded_shingles.shingle ==
hash_values.shingle)
        .groupBy("doc_id").agg(collect_list("hash_values").alias("hash_values"))

function locality_sensity_hashing(documents, num_hash_buckets=10)
    self.hash_buckets ← self.signatures.rdd.flatMap(lambda x: [(tuple(h), x[0]) for h in x[1]])
        .map(lambda x: ((hash(x[0]), x[1]), x[0]))
        .groupByKey().map(lambda x: (x[0][0] % num_hash_buckets, [x[0][1]]))
        .reduceByKey(lambda x, y: x + y).collect()
function run()
    shingling(self.documents)
    minhashing(self.documents)
    locality_sensity_hashing(self.documents)
```

Pseudocode for Task 2

```
function approxNearestNeighbors(documents, query_document, n)
        query_hash ← hash(query_document)
        for bucket_id, documents in self.hash_buckets do
            if hash(query_hash) % len(self.hash_buckets) == bucket_id then
                return documents[:n]

        return []

 function jaccard_similarity(set1, set2)
        intersection ← length(set1.intersection(set2))
        union ← length(set1.union(set2))

        return intersection / union

    return self
```

Pseudocode for Task 2

```
query_document ← " "

n ← 10
results ← lsh.approxNearestNeighbors(file_content_df, query_document, n)
print("Approximate nearest neighbors:", results)
query_shingles ← set(query_document.split())

for result_doc_id in results do
    result_shingle_set ← set(file_content_df.filter(file_content_df['doc_id'] ==
result_doc_id).collect()[0]['text'])
    jaccard_similarity ← lsh.jaccard_similarity(result_shingle_set, query_shingles)
    print(jaccard_similarity)
```

Result of Task 2

```
Approximate nearest neighbors: [1453, 1485, 3407, 3873, 4681, 5306, 26, 63, 63, 100]
```

```
0.009009009009009009
0.009615384615384616
0.008849557522123894
0.01680672268907563
0.017241379310344827
0.015267175572519083
0.009433962264150943
0.018867924528301886
0.018867924528301886
0.008695652173913044
```

Group’s member

Student ID	Full name	Email	Tasks	Complete percentage
520H0536	Lê Quốc Huy	520H0536@student.tdtu.edu.vn	Task 1, slide	100
522H0120	Nguyễn Đình Việt Hoàng	522H0120@student.tdtu.edu.vn	Slide	100
520H0523	Tăng Đại	520H0523@student.tdtu.edu.vn	Task 1, slide	100
521H0072	Nguyễn Thiên Huy	521H0072@student.tdtu.edu.vn	Task 2	100
521H0503	Trương Huỳnh Đăng Khoa	521H0503@student.tdtu.edu.vn	Task 2	100

References

[1]“MIN-HASHING AND LOCALITY SENSITIVE HASHING Thanks to: Rajaraman and Ullman, ‘Mining Massive Datasets’ Evimaria Terzi, slides for Data Mining Course.” Accessed: Apr. 28, 2024. [Online]. Available: https://www.cs.bu.edu/~gkollios/cs660f19/Slides/minhashLSH.pdf

[2]“Locality Sensitive Hashing (LSH): The Illustrated Guide | Pinecone,” www.pinecone.io. https://www.pinecone.io/learn/series/faiss/locality-sensitive-hashing/

[3]“Learn in 5 Minutes: Finding Nearest Neighbor using MinHash,” www.youtube.com. https://www.youtube.com/watch?v=GRHsg0d5X8Y (accessed Apr. 28, 2024).

[4]“Learn in 5 Minutes: Locality Sensitive Hashing (MinHash, SimHash, and more!),” www.youtube.com. https://www.youtube.com/watch?v=R-iFka68ZwM (accessed Apr. 28, 2024).

Subject: Mining Massive Datasets

Thanks for your listening