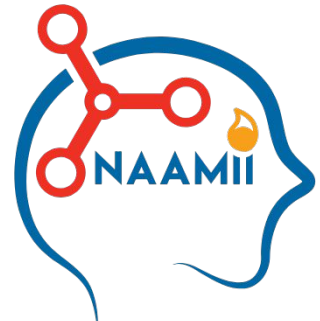


Finding Similar Items in Large Datasets

Big Data Analytics - Gandaki University MSc



Mahesh Shakya in co. with Shiva Ram Dam
Research Associate, NAAMII
Incoming PhD Student @ Hertie AI Center
for Brain Health, Uni. of Tübingen, Germany



Finding Similar items in Large Datasets

Basic Solution to finding text similarity

Issues with Basic Solution if we did not have speed constraint

Fast Solution meaningful and fast

Problem Statement

Your company is building a plagiarism detection engine that must scan millions of student essays submitted each semester. You need to detect near-duplicates in real-time. How do you represent and compare these documents at scale?

Are these text Plagiarized off of one another?

inheritance in object oriented programming is where a new class is formed using classes which have already been defined. These classes have some of the behavior and attributes which were existent in the classes that it inherited from. The purpose of inheritance in object oriented programming is to minimize the reuse of existing code without modification.

Inheritance allows classes to be categorized, similar to the way humans categorize. It also provides a way to generalize due to the "is a" relationship between classes. For example a "cow" is a generalization of "animal" similarly so are "pigs" & "cheaters". Defining classes in this way, allows us to define attributes and behaviours which are common to all animals in one class, so cheaters would naturally inherit properties common to all animals.

The advantage of inheritance is that classes which would otherwise have a lot of similar code, can instead share the same code, thus reducing the complexity of the program. Inheritance, therefore, can also be referred to as polymorphism which is where many pieces of code are controlled by shared control code.

Inheritance can be accomplished by overriding methods in its ancestor, or by adding new methods.

Inheritance in object oriented programming is a way to form new classes using classes that have already been defined. The new classes, known as derived classes, inherit attributes and behaviour of the existing classes, which are referred to as base classes. With little or no modification, it is intended to help reuse existing code. It is typically accomplished either by overriding one or more methods exposed by ancestor, or by adding new methods to those exposed by an ancestor.

Inheritance is also sometimes called generalization, because there is-a relationships represent a hierarchy between classes of objects. A 'fruit', for instance, is a generalization of "orange", "mango", "apples" and many others. One can consider fruit to be an abstraction of apple, orange, etc. Since apples are fruit (i.e., an apple is-a fruit), conversely apples may naturally inherit all the properties common to all fruit, such as being a fleshy container for the seed of a plant.

An advantage of inheritance is that modules with sufficiently similar interfaces can share a lot of code reducing the complexity of the program.

Are these text Plagiarized off of one another?

inheritance in object oriented programming is where a new class is formed using classes which have already been defined. These classes have some of the behavior and attributes which were existent in the classes that it inherited from. The purpose of inheritance in object oriented programming is to minimize the reuse of existing code without modification.

Inheritance allows classes to be categorized, similar to the way humans categorize. It also provides a way to generalize due to the "is a" relationship between classes. For example a "cow" is a generalization of "animal" similarly so are "pigs" & "cheaters". Defining classes in this way, allows us to define attributes and behaviours which are common to all animals in one class, so cheaters would naturally inherit properties common to all animals.

The advantage of inheritance is that classes which would otherwise have a lot of similar code, can instead share the same code, thus reducing the complexity of the program. Inheritance, therefore, can also be referred to as polymorphism which is where many pieces of code are controlled by shared control code.

Inheritance can be accomplished by overriding methods in its ancestor, or by adding new methods.

Inheritance is a basic concept of Object-Oriented Programming where the basic idea is to create new classes that add extra detail to existing classes. This is done by allowing the new classes to reuse the methods and variables of the existing classes and new methods and classes are added to specialise the new class. Inheritance models the "is-kind-of" relationship between entities (or objects), for example, postgraduates and undergraduates are both kinds of student. This kind of relationship can be visualised as a tree structure, where 'student' would be the more general root node and both 'postgraduate' and 'undergraduate' would be more specialised extensions of the 'student' node (or the child nodes). In this relationship 'student' would be known as the superclass or parent class whereas, 'postgraduate' would be known as the subclass or child class because the 'postgraduate' class extends the 'student' class.

Inheritance can occur on several layers, where if visualised would display a larger tree structure. For example, we could further extend the 'postgraduate' node by adding two extra extended classes to it called, 'MSc Student' and 'PhD Student' as both these types of student are kinds of postgraduate student. This would mean that both the 'MSc Student' and 'PhD Student' classes would inherit methods and variables from both the 'postgraduate' and 'student' classes.

How to compare documents based on their content ?

Inheritance in object oriented programming is where a new class is formed using classes which have already been defined. These classes have some of the behavior and attributes which were existent in the classes that it inherited from. The purpose of inheritance in object oriented programming is to minimize the reuse of existing code without modification.

Inheritance allows classes to be categorized, similar to the way humans categorize. It also provides a way to generalize due to the "is-a" relationship between classes. For example a "cow" is a generalization of "animal", similarly so are "pigs" & "cheaters". Defining classes in this way allows us to define attributes and behaviours which are common to all animals in one class, so cheaters would naturally inherit properties common to all animals.

The advantage of inheritance is that classes which would otherwise have a lot of similar code, can instead share the same code, thus reducing the complexity of the program. Inheritance, therefore, can also be referred to as polymorphism which is where many pieces of code are controlled by shared control code.

Inheritance can be accomplished by overriding methods in its ancestor, or by adding new methods.

Inheritance in object oriented programming is a way to form new classes using classes that have already been defined. The new classes, known as derived classes, inherit attributes and behaviour of the existing classes, which are referred to as base classes. With little or no modification, it is intended to help reuse existing code. It is typically accomplished either by overriding one or more methods exposed by ancestor, or by adding new methods to those exposed by an ancestor.

Inheritance is also sometimes called generalization, because there is a relationship that represents a hierarchy between classes of objects. A 'fruit', for instance, is a generalization of "orange", "mango", "apples" and many others. One can consider fruit to be an abstraction of apple, orange, etc. Since apples are fruit (i.e., an apple is-a fruit), conversely apples may naturally inherit all the properties common to all fruit, such as being a fleshy container for the seed of a plant.

An advantage of inheritance is that modules with sufficiently similar interfaces can share a lot of code reducing the complexity of the program.

Highly similar

How to compare documents based on their content ?

How to compare documents based on their content ? **Basic Solution**

How to compare documents based on their content ?

- Solution 1: Count how many words are common
 - Good first solution

How to compare documents based on their content ?

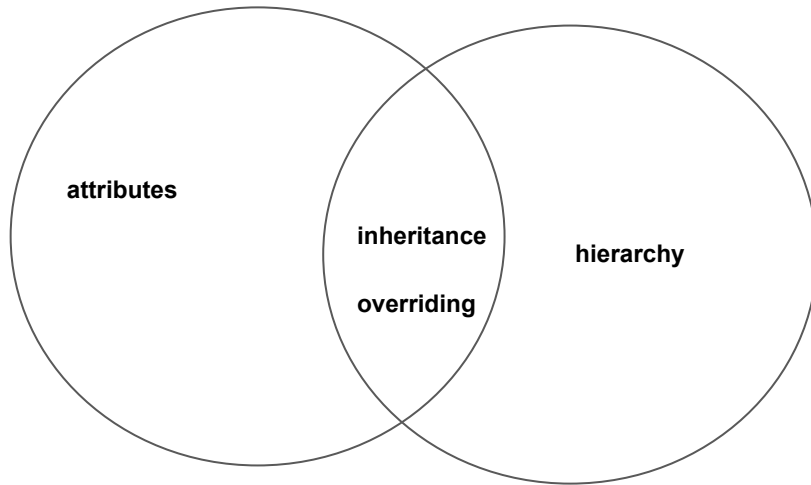
- Solution 1: Count how many words are common
 - Good first solution
 - Challenges
 - Ignores Word order and context e.g. “dog bites man” vs. “man bites dog”
 - Synonyms, paraphrase, case-sensitivity, punctuation
 - Common words such as the, in, of etc. may increase count
 - Scalable Implementation

How to compare documents based on their content ?

- Solution 1: Count how many words are common
 - Good first solution
 - Challenges
 - Ignores Word order and context e.g. “dog bites man” vs. “man bites dog”
 - Synonyms, paraphrase, case-sensitivity, punctuation
 - Common words such as the, in, of etc. may increase count
 - Scalable Implementation
- How do we define “similar documents”?
 - Count common words: Large texts have lot more common words than short texts
 - Percentage of common words between texts?

Count common words

$$C(A, B) = |A \cap B|$$



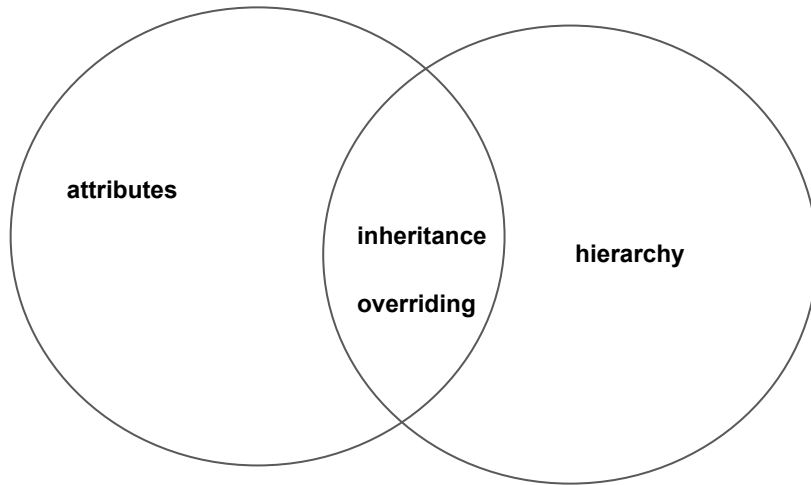
$$C = |\{\text{inheritance, overriding}\}|$$

How many is too many?

Percentage of common words

$$C_A(A, B) = |A \cap B| / |A|$$

$$C_B(A, B) = |A \cap B| / |B|$$

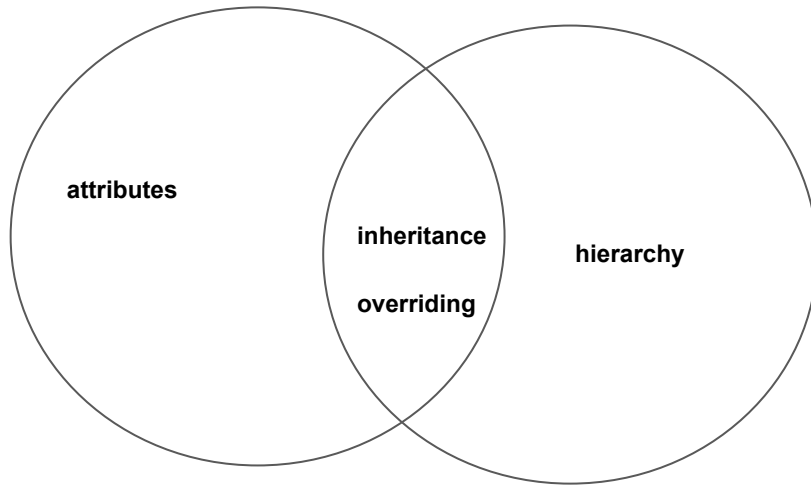


$$C = \{\text{inheritance, overriding}\}$$

85% is too many?

How to make this symmetric - Jaccard Similarity

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$



$$J = \frac{|{\text{inheritance, overriding}}|}{|{\text{attributes, inheritance, overriding, hierarchy}}|}$$

Set to Vector Representation - Bag of Words (BoW)

Step 1: Input Documents

Document 1 (D1):

"ChatGPT is a helpful assistant"

Document 2 (D2):

"ChatGPT is a powerful assistant"

Step 2: Create the Vocabulary

Extract all unique words (after tokenization and lowercasing):

Vocabulary: [chatgpt, is, a, helpful, powerful, assistant]

Step 3: Vector Representation

Word	D1 Count	D2 Count
chatgpt	1	1
is	1	1
a	1	1
helpful	1	0
powerful	0	1
assistant	1	1

D1 → [1, 1, 1, 1, 0, 1]

D2 → [1, 1, 1, 0, 1, 1]

How to compare documents based on their content ?

- Solution 1: Count how many words are common
 - Good first solution
 - Challenges
 - Ignores Word order and context e.g. “dog bites man” vs. “man bites dog”
 - Synonyms, paraphrase, case-sensitivity, punctuation
 - **Common words such as the, in, of etc. may increase count**
 - Scalable Implementation
- How do we define “similar documents”?
 - Count common words: Large texts have lot more common words than short texts
 - Percentage of common words between texts?

How to handle common words like “a of ...” ?

Step 1: Input Documents

Let's take two short documents:

- **D1:** "the cat sat on the mat"
- **D2:** "the dog sat on the log"

Step 2: Create the Vocabulary

Vocabulary:

[the, cat, sat, on, mat, dog, log]

TF (raw frequency) vectors:

- **D1** \rightarrow [2, 1, 1, 1, 1, 0, 0]
- **D2** \rightarrow [2, 0, 1, 1, 0, 1, 1]

Step 3: Term Frequency (TF)

Term	D1 Count	D2 Count
the	2	2
cat	1	0
sat	1	1
on	1	1
mat	1	0
dog	0	1
log	0	1

How to handle common words like “a of ...” ?

TF (raw frequency) vectors:

- **D1** → [2, 1, 1, 1, 1, 0, 0]
- **D2** → [2, 0, 1, 1, 0, 1, 1]

Vocabulary:

[the, cat, sat, on, mat, dog, log]

Document 1 (D1):

- [2×0.693, 1×1.098, 1×0.693, 1×0.693, 1×1.098, 0, 0]
- → [1.386, 1.098, 0.693, 0.693, 1.098, 0, 0]

Document 2 (D2):

- [2×0.693, 0, 1×0.693, 1×0.693, 0, 1×1.098, 1×1.098]
- → [1.386, 0, 0.693, 0.693, 0, 1.098, 1.098]

 **Step 4: Compute Inverse Document Frequency**

(IDF) Let's use **smoothed IDF**:

$$\text{IDF}(t) = \log \left(1 + \frac{N}{df(t)} \right)$$

Term	df(t)	IDF	Value
the	2	$\log(1 + 2/2)$	≈ 0.693
cat	1	$\log(1 + 2/1)$	≈ 1.098
sat	2	$\log(1 + 2/2)$	≈ 0.693
on	2	$\log(1 + 2/2)$	≈ 0.693
mat	1	$\log(1 + 2/1)$	≈ 1.098
dog	1	$\log(1 + 2/1)$	≈ 1.098
log	1	$\log(1 + 2/1)$	≈ 1.098

How to count common words in vector representation - Cosine Similarity

Document 1 (D1):

"ChatGPT is a helpful assistant"

Document 2 (D2):

"ChatGPT is a powerful assistant"

$$\cos(\theta) = \frac{\vec{D1} \cdot \vec{D2}}{||\vec{D1}|| \cdot ||\vec{D2}||}$$

1. Dot Product:

$$(1 \cdot 1) + (1 \cdot 1) + (1 \cdot 1) + (1 \cdot 0) + (0 \cdot 1) + (1 \cdot 1) = 4$$

2. Magnitudes:

$$||\vec{D1}|| = \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 1^2} = \sqrt{5}$$

$$||\vec{D2}|| = \sqrt{1^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2} = \sqrt{5}$$

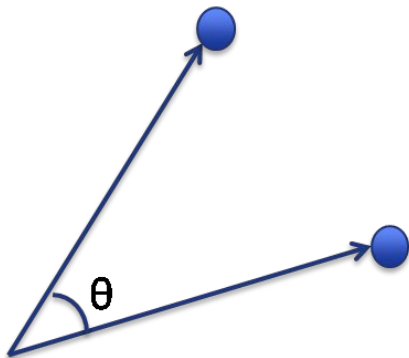
3. Cosine Similarity:

$$\cos(\theta) = \frac{4}{\sqrt{5} \cdot \sqrt{5}} = \frac{4}{5} = 0.8$$

Step 4: Cosine Similarity Calculation

D1 → [1, 1, 1, 1, 0, 1]

D2 → [1, 1, 1, 0, 1, 1]



Similarity Metric

A Similarity metric is a function that takes A and B and returns a scalar R
 $F(A,B) \rightarrow R [0, +\infty]$, higher the better.

Properties

1. A should be similar to A

$$F(A,A) = \text{large}$$

2. Symmetric

$$F(A,B) = F(B,A)$$

3. Distance Inequality

$$F(A,B) + F(B,C) \leq F(A,C)$$

How to compare documents based on their content ?

- Solution 1: Count how many words are common
 - Good first solution
 - Challenges
 - **Ignores Word order and context e.g. “dog bites man” vs. “man bites dog”**
 - Synonyms, paraphrase, case-sensitivity, punctuation
 - Common words such as the, in, of etc. may increase count
 - Scalable Implementation
- How do we define “similar documents”?
 - Count common words: Large texts have lot more common words than short texts
 - Percentage of common words between texts?

Comparison of Methods to Compare documents

Method	Captures Near-Duplicates	Considers Word Order	Handles Synonyms	Scalable to Millions of Documents	Typical Use Case
Exact String Matching	✗	✓	✗	✓	Duplicate detection (identical content)
Bag-of-Words (BoW)	✗	✗	? (partially)	✓	Document classification, clustering
TF-IDF + Cosine Sim.	✓ (semantic similarity)	✗	✓	✓	Information retrieval, search engines

Comparison of Methods to Compare documents

Method	Captures Near-Duplicates	Considers Word Order	Handles Synonyms	Scalable to Millions of Documents	Typical Use Case
Exact String Matching	✗	✓	✗	✓	Duplicate detection (identical content)
Bag-of-Words (BoW)	✗	✗	? (partially)	✓	Document classification, clustering
TF-IDF + Cosine Sim.	✓ (semantic similarity)	✗	✓	✓	Information retrieval, search engines
Edit Distance	✓	✓	✗	✗	Small-scale text diffing (e.g., spell check)

How to compare documents based on their content ? **Preserve Word Order**

Doc 1: “dog bites man”

Doc 2: “man bites dog”

Bag of Words

Cosine Similarity = 1

Doc 1: [1,1,1]

Doc 2: [1,1,1]

Edit Distance = 2

Doc 1: “dog bites man”

**Doc 2: “~~man~~ dog bites
~~dog~~ man”**

How to compare documents based on their content ? Meaningful & Fast!!



Step 1: Input Documents

Doc A: "the cat sat on the mat"

Doc B: "the cat sat on the hat"



Step 2: Create Shingles (word-based)

Doc A shingles: {the cat sat, cat sat on, sat on the, on the mat}

Doc B shingles: {the cat sat, cat sat on, sat on the, on the hat}

Calculate Jaccard similarity:

$$\frac{|A \cap B|}{|A \cup B|} = \frac{3}{5} = 0.6$$

Local Text structure preserved!!
"dog bites man" vs. "man bites dog"

How to compare documents based on their content ? **Fast!!**



Step 1: Input Documents

Doc A: "the cat sat on the mat"

Doc B: "the cat sat on the hat"



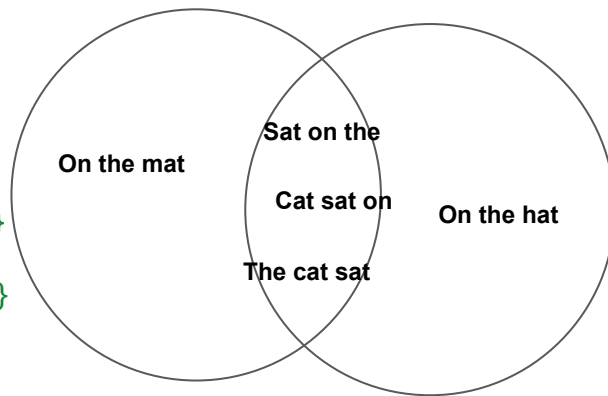
Step 2: Create Shingles (word-based)

Doc A shingles: {the cat sat, cat sat on, sat on the, on the mat}

Doc B shingles: {the cat sat, cat sat on, sat on the, on the hat}

Calculate Jaccard similarity:

$$\frac{|A \cap B|}{|A \cup B|} = \frac{3}{5} = 0.6$$



Set size increases



**Cannot calculate
exact similarity, too
big!**

How to calculate **approximate** similarity between sets? **Fast!!**

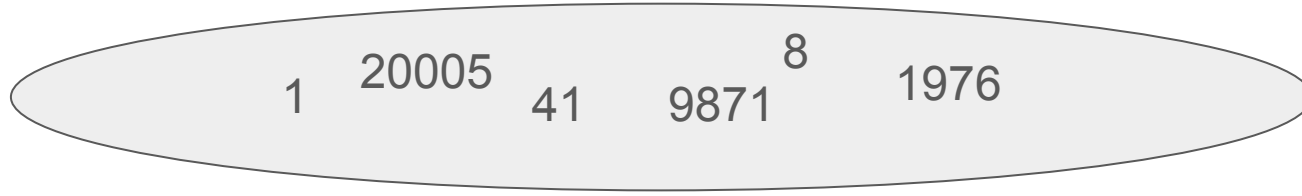
If two documents are similar, their shingle sets will have a high Jaccard similarity. But computing exact Jaccard similarity is expensive for large datasets. How can we estimate similarity efficiently?

How to calculate **approximate** similarity between sets? **Fast!!**

MinHashing

generate compact signatures from sets that probabilistically preserve Jaccard similarity.

Hashing



Arbitrary
Size

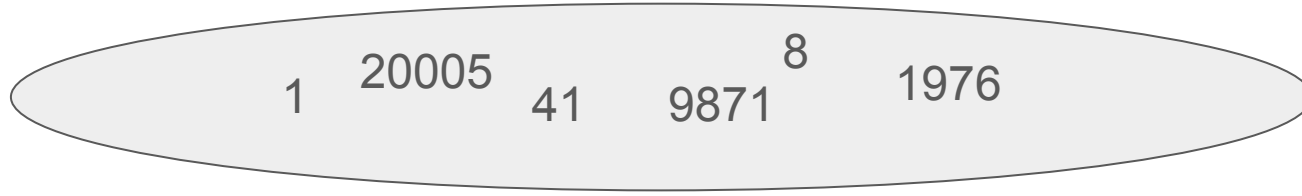


0 1 2 3 4 5 6 7 8 9

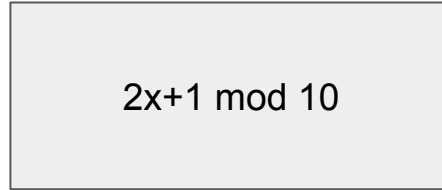


Fixed
Bins

Hashing



Arbitrary
Size



0 1 2 3 4 5 6 7 8 9



Fixed
Bins

MinHashing

Step 1: Define two small shingle sets

Doc A shingles: {the cat sat, cat sat on, sat on the, on the mat}

Doc B shingles: {cat sat on, sat of off, sat on the, on the hat}

(Represent shingles as integers for **simplicity.**)

Doc A shingles = {1, 3, 5, 7}

Doc B shingles = {3, 4, 5, 8}

MinHashing

Step 1: Define two small shingle sets

(Represent shingles as integers for **simplicity.**)

Doc A shingles: {the cat sat, cat sat on, sat on the, on the mat}

Doc A shingles = {1, 3, 5, 7}

Doc B shingles: {cat sat on, sat of off, sat on the, on the hat}

Doc B shingles = {3, 4, 5, 8}

Step 2: Define 3 simple hash functions

For example, define hash functions $h_i(x) = (a_i x + b_i) \bmod 10$:

Hash function	a_i	b_i	Definition
$h_1(x)$	2	1	$h_1(x) = (2x + 1) \bmod 10$
$h_2(x)$	3	2	$h_2(x) = (3x + 2) \bmod 10$
$h_3(x)$	5	1	$h_3(x) = (5x + 1) \bmod 10$

MinHashing

Step 1: Define two small shingle sets

Step 2: Define 3 hash functions

(Represent shingles as integers for **simplicity.**)

Doc A shingles = {1, 3, 5, 7}

Doc B shingles = {3, 4, 5, 8}

Step 3: Compute hash values for all shingles

Shingle	$h_1(x)$	$h_2(x)$	$h_3(x)$
1	$(2*1+1)\%10=3$	$(3*1+2)\%10=5$	$(5*1+1)\%10=6$
3	$(2*3+1)\%10=7$	$(3*3+2)\%10=1$	$(5*3+1)\%10=6$
4	$(2*4+1)\%10=9$	$(3*4+2)\%10=4$	$(5*4+1)\%10=1$
5	$(2*5+1)\%10=1$	$(3*5+2)\%10=7$	$(5*5+1)\%10=6$
7	$(2*7+1)\%10=5$	$(3*7+2)\%10=3$	$(5*7+1)\%10=6$
8	$(2*8+1)\%10=7$	$(3*8+2)\%10=6$	$(5*8+1)\%10=1$

MinHashing

Step 1: Define two small shingle sets

Step 2: Define 3 hash functions

Step 3: Compute hash values for all shingles

(Represent shingles as integers
for **simplicity.**)

Doc A shingles = {1, 3, 5, 7}

Doc B shingles = {3, 4, 5, 8}

Step 4: For each hash function, find the minimum hash value over the shingles in each document

Hash func	Min hash in Doc A (shingles {1,3,5,7})	Min hash in Doc B (shingles {3,4,5,8})
h_1	$\min(3,7,1,5) = 1$	$\min(7,9,1,7) = 1$
h_2	$\min(5,1,7,3) = 1$	$\min(1,4,7,6) = 1$
h_3	$\min(6,6,6,6) = 6$	$\min(6,1,6,1) = 1$

MinHashing

Step 1: Define two small shingle sets

Step 2: Define 3 hash functions

Step 3: Compute hash values for all shingles

Step 4: For each hash function, find the minimum hash values

(Represent shingles as integers for **simplicity.**)

Doc A shingles = {1, 3, 5, 7}

Doc B shingles = {3, 4, 5, 8}

Step 5: MinHash signatures

- Doc A signature = [1, 1, 6]
- Doc B signature = [1, 1, 1]

MinHashing

Step 1: Define two small shingle sets

Step 2: Define 3 hash functions

Step 3: Compute hash values for all shingles

Step 4: For each hash function, find the minimum hash values

Step 5: MinHash Signatures

(Represent shingles as integers for **simplicity.**)

Doc A shingles = {1, 3, 5, 7}

Doc B shingles = {3, 4, 5, 8}

Step 6: Estimate similarity

- Count number of matching components in signatures: 2 (first two hashes match)
- Estimated similarity = $2 / 3 \approx 0.67$

MinHashing

Step 1: Define two small shingle sets

Step 2: Define 3 hash functions

Step 3: Compute hash values for all shingles

Step 4: For each hash function, find the minimum hash values

Step 5: MinHash Signatures

(Represent shingles as integers for **simplicity**.)

Doc A shingles = {1, 3, 5, 7}

Doc B shingles = {3, 4, 5, 8}

Step 6: Estimate similarity

- Count number of matching components in signatures: 2 (first two hashes match)
- Estimated similarity = $2 / 3 \approx 0.67$

Step 7: Calculate actual Jaccard similarity

$$J(A, B) = \frac{|3, 5|}{|1, 3, 4, 5, 7, 8|} = \frac{2}{6} \approx 0.33$$

Note: Our MinHash estimate is an approximation and improves with more hash functions

Summary

Finding similar Items - e.g. Plagiarism Detection

Simple Solution : Count common words

Bag of Words

How to measure similarity between sets/vectors

Cosine Similarity, Jaccard Similarity

- + Preserve word order: Shingling
- + Fast on large datasets: MinHashing