Sample for Graph Based Data

Alice (ID 1)

Bob (ID 2)

Charlie (ID 3)

Dave (ID 4)

Eve (ID 5)

1: [2, 3] # Alice is friends with Bob and Charlie

2: [1, 4] # Bob is friends with Alice and Dave

3: [1, 4] # Charlie is friends with Alice and Dave

4: [2, 3, 5] # Dave is friends with Bob, Charlie, and Eve

5: [4] # Eve is friends with Dave

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 |
| 3 | 1 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 |

Pyton Program

import networkx as nx

import matplotlib.pyplot as plt

# Create a new graph object

G = nx.Graph()

# Add nodes (people)

G.add\_nodes\_from([1, 2, 3, 4, 5])

# Add edges (friendships)

G.add\_edges\_from([(1, 2), (1, 3), (2, 4), (3, 4), (4, 5)])

# Draw the graph

nx.draw(G, with\_labels=True, node\_size=1000, node\_color="lightblue", font\_size=12)

plt.show()

**Example: Document-Term Matrix (DTM)**

Suppose we have the following 3 documents:

* **Document 1**: "I love machine learning."
* **Document 2**: "Machine learning is fun."
* **Document 3**: "I love programming and machine learning."

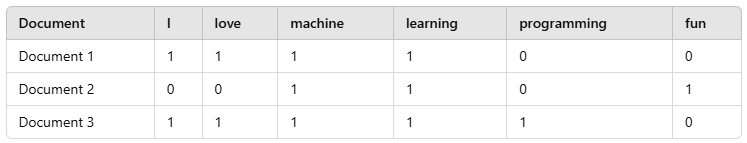
**Step 1: Identify the unique terms (words) across all documents**

We'll first extract the unique words from all documents. Let's consider a simple **bag-of-words** approach (ignoring stop words like "is", "and", etc.):

* Unique terms (vocabulary): **["I", "love", "machine", "learning", "programming", "fun"]**

**Step 2: Create the Document-Term Matrix (DTM)**

* The rows of the matrix correspond to the documents, and the columns correspond to the unique terms. The values in the matrix represent the frequency of each term in each document.

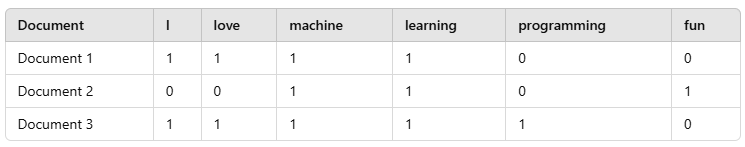


**Explanation:**

* **Document 1**: "I love machine learning."
  + **I** appears once.
  + **love** appears once.
  + **machine** appears once.
  + **learning** appears once.
  + **programming** and **fun** do not appear, so their counts are 0.
* **Document 2**: "Machine learning is fun."
* **machine** appears once.
* **learning** appears once.
* **fun** appears once.
* **I**, **love**, and **programming** do not appear, so their counts are 0.
* **Document 3**: "I love programming and machine learning."
* **I** appears once.
* **love** appears once.
* **machine** appears once.
* **learning** appears once.
* **programming** appears once.
* **fun** does not appear, so its count is 0.

**Other Variations of Document-Term Matrix**

1. **Binary Representation (Presence/Absence)**: Instead of counting the number of times a word appears, you could represent the matrix with 1 for presence and 0 for absence. This is useful in certain text classification problems where only the existence of terms matters.



**TF-IDF (Term Frequency-Inverse Document Frequency)**: Instead of raw term frequencies, you can use TF-IDF to weight terms based on their importance in a document relative to their frequency in the entire corpus. Words that are common across many documents will have lower TF-IDF scores, while rare words will have higher TF-IDF scores.

**Creating a Document-Term Matrix in Python (with scikit-learn)**

Here's a simple code snippet to generate a **Document-Term Matrix** using Python and the **scikit-learn** library:

python

from sklearn.feature\_extraction.text import CountVectorizer

import pandas as pd

# List of documents

documents = [

"I love machine learning.",

"Machine learning is fun.",

"I love programming and machine learning."

]

# Create the CountVectorizer model

vectorizer = CountVectorizer()

# Fit and transform the documents into a Document-Term Matrix

dtm = vectorizer.fit\_transform(documents)

# Convert the matrix to a DataFrame for better visualization

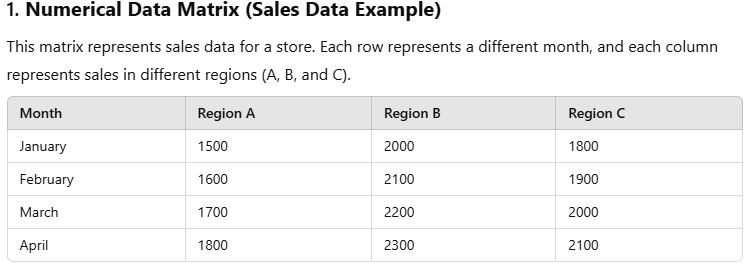
dtm\_df = pd.DataFrame(dtm.toarray(), columns=vectorizer.get\_feature\_names\_out())

# Display the Document-Term Matrix

print(dtm\_df)

. **Numerical Data Matrix (Sales Data Example)**

Matrix data is commonly used to represent structured data in a grid-like format where each element in the matrix can represent different types of information depending on the context (such as numerical values, relationships, or other attributes).



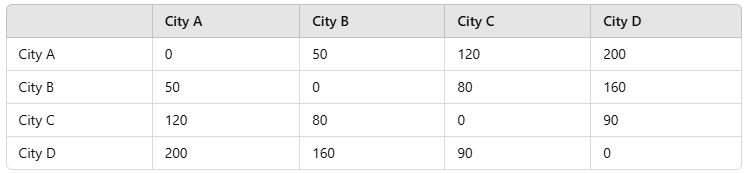
**Rows**: Represent different time periods (months).

**Columns**: Represent different regions or categories.

**Matrix elements**: Represent the sales figures in those regions during each month.

2. **Distance Matrix (Geographical Distances)**

A distance matrix represents the distances between various locations. Each row and column represent different locations, and the values in the matrix represent the distance between them.



**Rows and Columns**: Represent cities.

**Matrix elements**: Represent the distances between cities. For instance, the distance between City A and City B is 50 miles.

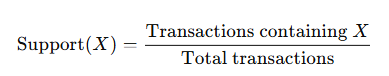
Market Basket Data

**1. Association Rule Mining:**

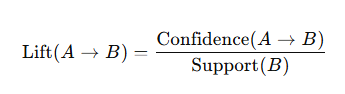
Market Basket Analysis is often implemented through association rule mining, which finds interesting relationships between products in large datasets. The relationships are typically expressed as "If-Then" statements:

* **If a customer buys Product A, they are likely to buy Product B**.

**2. Key Metrics in Market Basket Analysis:**

* **Support**: The frequency with which a product set appears in the dataset. It’s the percentage of transactions that include a specific item or combination of items.
  + Formula:
  + 
* **Confidence**: The likelihood that a customer who buys Product A will also buy Product B. This measures the strength of the association.
  + Formula:



* **Lift**: This metric measures how much more likely two items are to be bought together than would be expected by chance. A lift value greater than 1 indicates that the items are often bought together.
  + Formula:
* 

**3. Applications of Market Basket Analysis:**

* **Product Placement**: Retailers can place products that are frequently bought together in proximity to increase cross-selling.
* **Cross-Promotions**: Offering discounts on complementary products that are often bought together.
* **Recommendation Systems**: Online stores can use market basket analysis to recommend products to customers based on past purchasing patterns.

**4. Algorithms Used:**

* **Apriori Algorithm**: One of the most popular algorithms for market basket analysis, which generates frequent itemsets and association rules.
* **FP-growth (Frequent Pattern Growth)**: A more efficient algorithm for mining frequent itemsets, which doesn’t require candidate generation like Apriori.

# Import necessary libraries

import pandas as pd

from mlxtend.frequent\_patterns import apriori, association\_rules

from mlxtend.preprocessing import TransactionEncoder

# Sample data (each list represents a transaction)

transactions = [

    ['milk', 'bread', 'butter'],

    ['milk', 'diaper', 'beer', 'bread'],

    ['milk', 'bread', 'butter', 'diaper'],

    ['milk', 'bread', 'butter', 'beer'],

    ['milk', 'diaper', 'beer', 'bread']

]

# Step 1: One-Hot Encoding the transactions

te = TransactionEncoder()

te\_ary = te.fit(transactions).transform(transactions)

oht\_df = pd.DataFrame(te\_ary, columns=te.columns\_)

# Display the one-hot encoded DataFrame

print("One-Hot Encoded DataFrame:")

print(oht\_df)

# Step 2: Apply the Apriori algorithm to find frequent itemsets

frequent\_itemsets = apriori(oht\_df, min\_support=0.4, use\_colnames=True)

print("\nFrequent Itemsets (Support >= 0.4):")

print(frequent\_itemsets)

# Step 3: Generate Association Rules

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1.5)

print("\nAssociation Rules (Lift >= 1.5):")

print(rules)

# Step 4: Display details of association rules

print("\nDetails of Association Rules:")

for index, row in rules.iterrows():

    print(f"Rule {index + 1}:")

    print(f"  Antecedent: {row['antecedents']}")

    print(f"  Consequent: {row['consequents']}")

    print(f"  Support: {row['support']:.2f}")

    print(f"  Confidence: {row['confidence']:.2f}")

    print(f"  Lift: {row['lift']:.2f}")

    print("------")