Latent Semantic Analysis

Latent Semantic Analysis is a technique of analysing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents & terms.

→ We can generate Keywords using LSA

1	MUSIC	FOOD
2	AR 1 - 85%.	AR 3 - 100%
3		AR 5 - 73%.
4		
5	AR 4 - 100%.	AR 2 - 100%.
6	AR 6 - 100%.	AR 1 - 15%.
	AR 5 - 27%.	
	NEWS	TECH

Bow Model

	W1	W2	W3	...	Wn
Doc 1					
Doc 2					
Doc 3					

→ When Generating LSA there will be a huge corpus of document

SVD (Singular Value Decomposition)

$$A_{[m \times n]} = U_{[m \times r]} * S_{[r \times r]} * \left(V_{[n \times r]}\right)^T$$

A: Input Data Matrix

- m x n matrix (m = number of documents, n = number of words/features)

U = Left Singular matrix

- m x r matrix (m = number of documents, r = number of Concepts)

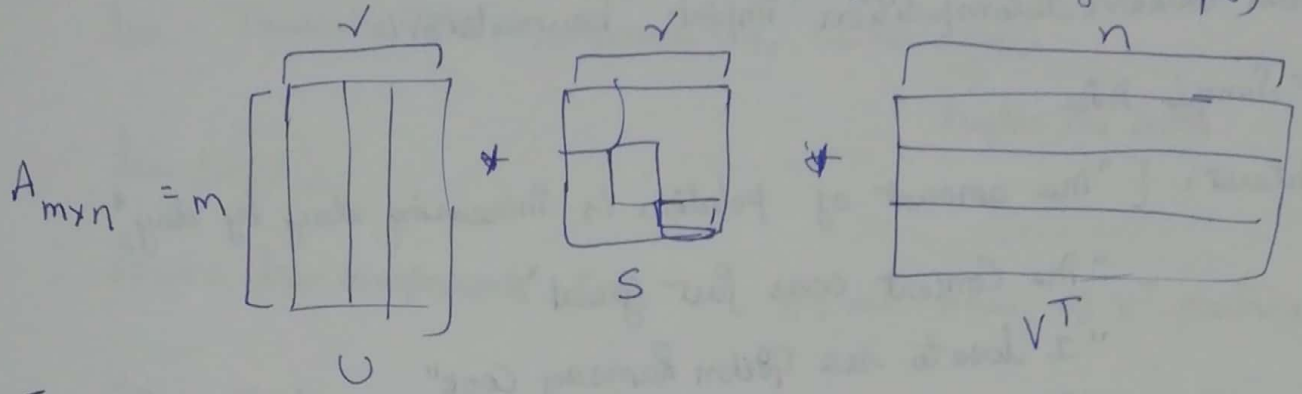
S = Rank matrix

- r x r matrix (r = rank of A)

→ Diagonal matrix

V = Right Singular matrix

→ n×r matrix (n = number of words/features, r = number of concepts)



V^T = r × n matrix.

Application

→ Article Bucketing websites
→ Finding relation b/w articles/words
→ Page indexing in search Engines.

LSA Implementation

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
# Sample Data
dataset = ["The amount of polution is increasing day by day",
           "The Concert was just great"
           "I love to see Gordon Ramsay Cook"
           "Google is introducing new technology"
           "AI Robots are Examples of great technology present today"
           "All of us were singing in the Concert"
           "we have launch campaigns to stop pollution and global warming"]
```

→ we will try to Create Concepts
→ whether it is able to create a Concept
→ ~~use~~ Important words related to a Concept
→ Identify different documents for a specific Context.

→ Preprocessing
  → lower
```
dataset = [line.lower() for line in dataset]
```
→ Creating Bow model (Bow / TFIDF / Count Vectorize)
```
Vectorizer = TfidfVectorizer()
X = Vectorizer.fit_transform(dataset)
print(X[0])
```

o/p
```
(0,34) 0.22786437
(0,2)  0.32114839
(0,24) 0.22786437
```

decomposition

we have to convert the matrix X in ~~to~~ U S V^T

```
lsa = TruncatedSVD(n_components = 4, n_iter = 100)
                                    ↑ higher the better.
lsa.fit(X)
row1 = lsa.components_[0]      //first row of V^T matrix
term = Vectorizer.get_feature_names()  //gives all the words
Concept_words = {}
for i, comp in Enumerate(lsa.components_):
    ComponentTerms = zip(term, comp)
    SortedTerms = sorted(componentTerms, key = lambda x : x[1], reverse = True)
    sortedTerms = sortedTerms[:10]
    print("\nConcept", i, ":")
    for terms in sortedTerms:
        print(terms)
    Concept_words["Concept "+str(i)] = sortedTerms.
```

→ we have 7 documents, we classify which document falls into which Concept

```
for key in Concept_words.keys()
    Sentence_score = [ ]
    For Sentence is dataset
        words = nltk.word_tokenize(sentence)
        score = 0
        for word in words:
            For word_with_score in Concept_words[key]:
                If word == word_with_score[0]:
```

```
        for
            score += word_win_score[1]

        Sentence_score.append(score)
print("\n"+Key+":")
for sentence_score in Sentence_scores:
    print(sentence_score)
```