Vietnam National University HCMC
International University
School of Computer Science and Engineering

REPORT

TOPIC:
LATENT SEMANTIC ANALYSIS

Subject: *Scalable & Distributed Computing*
Lecturer: *Tran Manh Ha*
Semester/Year: *Sem1/2021-2022*
Group members:

1. *Le Thi Thu Tra - ITDSIU19058*
2. *Nguyen Minh Trang - ITDSIU19020*
3. *Phan Võ Phương Tùng - ITDSIU19025*

TABLE OF CONTENTS:

I. INTRODUCTION

1. What is Latent Semantic Analysis?

- Latent Semantic Analysis (LSA) also known as Latent Semantic Index (LSI).
- It can **extract hidden features** of a document that can't be directly seen
- LSA **uses a bag of words model**, which results in a term-document matrix (occurence of terms in a document). Rows represent terms and columns represent documents *(example below)*

Venue / Keyword	CAMAD	EUNICE	HAISA	HPCC-ICESS	IJESMA	ISCA	KMIS	NMR	SPRINGL	SSV
algorithm	2	8	0	24	0	5	0	2	1	1
cellular	2	1	0	1	0	0	0	0	0	0
game	1	1	0	1	0	0	1	1	0	0
hardwar	1	0	1	4	0	18	0	0	1	0
internet	2	6	2	0	2	0	0	0	0	0
mobil	10	8	0	6	17	5	2	0	2	0
network	58	60	4	38	2	25	12	0	3	0
search	0	1	0	1	2	4	1	0	0	0
secur	4	4	29	5	1	12	3	0	4	0
web	0	2	0	3	3	1	13	0	2	0

- LSA learns latent topics by performing a matrix decomposition on the document-term matrix using Singular Value Decomposition
- LSA is typically used as a dimension reduction or noise reducing technique

2. What is Singular Value Decomposition?

- Singular Value Decomposition (SVD) is a matrix factorization method that represents a matrix in the product of two matrices. It offers various useful applications in signal processing, psychology, sociology, climate, and atmospheric science, statistics and astronomy

3. **Identifying problems**

   File formats: 3 zip files, each contains 5 CSV files

   Size: 2.05GB

   List of attributes:

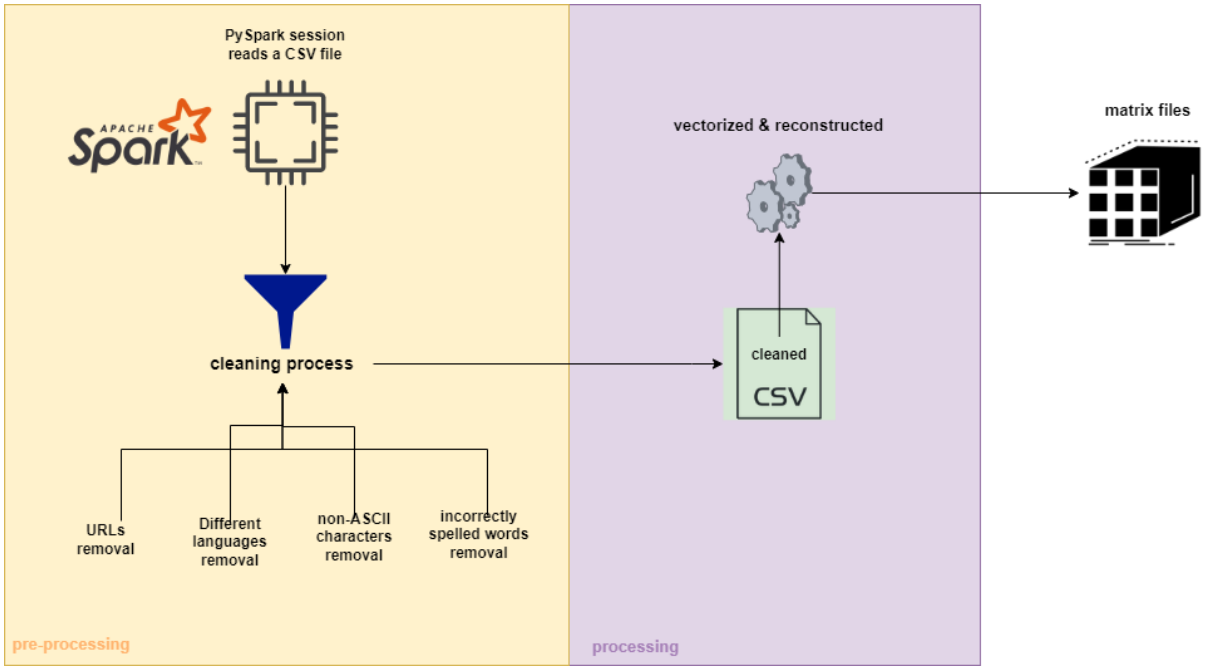
   - `doi` (Digital Object Identifier): a persistent identifier or handle used to identify objects uniquely
   - `articleType`
   - `citationType`
   - `citationStringAnnotated` (main feature to be processed)

   Problems raised:

   - How to read multiple **HUGE** datasets in a **short** to moderate amount of time?
   - How to utilize multiprocessing and multithreading to handle the reading/manipulating processes in parallel?
   - How to handle as much data as possible?
   - Is there any alternative for the traditional pandas library?

4. **Solution**

   - We did some research on scalable and distributed performance and we came across Dask. Dask ships with schedulers designed for use on personal machines. Many people use Dask today to **scale computations on their laptops**, using **multiple cores** for computation and their disks for **excess storage**.
   - However, we finally use PySpark to do the pre-processing larger datasets in a distributed cluster, to save more time and the working memory of our system. Moreover, it helps us to speed up our Python code. We wanted to do the distributed computation in PySpark, we had to perform operations on PySpark Dataframe



click here to enlarge

II. PRE-PROCESSING DATA

1. Reading data

```
1 from pyspark.sql import SparkSession
2 spark = SparkSession.builder.appName("PreProcessingFile").getOrCreate()
3
4 df = spark.read.csv('/content/Billion Citation
  Dataset.part001/*.csv',header=True).select('citationStringAnnotated')
5
```

2. Cleaning data

When working with text, we need to make sure we are dealing with standard text according to what we expect: correctly-spelled text, stopwords removal, stemming, lemmatization, HTML tag removal, and other languages/non-ASCII characters removal

```
df_clean = df.select('citationStringAnnotated',
(lower(regexp_replace('citationStringAnnotated', "http[s]?://(?:[a-zA-Z]|[0-9]|[$-
_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))", "")).alias('cleantext')))

df_clean = df_clean.select('citationStringAnnotated',(lower(regexp_replace('cleantext',
"<.*?>", "")).alias('cleantext')))

df_clean = df_clean.select('citationStringAnnotated',(lower(regexp_replace('cleantext',
"[^a-zA-Z\\s]", "")).alias('cleantext')))
```

III. PROCESSING DATA

1. TF-IDF Vectorizer

TF-IDF (term frequency-inverse document frequency) is a statistical measure that **evaluates how relevant a word is to a document** in a collection of documents.

This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

```
vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(text_list_head[:11000])

feature_names = vectorizer.get_feature_names()

dense = vectors.todense()
denselist = np.float32(dense.tolist())

matrix_df = pd.DataFrame(denselist, columns=feature_names)

```

2. Performing SVD

```
# SVD
U, s, VT = svd(matrix_df)

# create m x n Sigma matrix
Sigma = zeros((matrix_df.shape[0], matrix_df.shape[1]))

# populate Sigma with n x n diagonal matrix
Sigma[:matrix_df.shape[0], :matrix_df.shape[0]] = diag(s)

# select 2 largest singular values
n_elements = 2
Sigma = Sigma[:, :n_elements]
VT = VT[:n_elements, :]

# reconstruct
B = U.dot(Sigma.dot(VT))
```

IV. OUTPUT

We pre processed successfully **69,195,478 rows** with multithread.

V. CONCLUSION

1. Contribution

	Analyze the problem & come up with solution	Pre-processing	Processing	Writing report
Trang	x			x
Tra	x	x		
Tung	x		x	

2. What we learned after this mini-project?

- How to handle large data with Spark and clean big data with Spark
- Using Dask to read large data in processing instead using Pandas.
- How to apply a large dataset into the TF-IDF and SVD model.
- Acknowledge the importance of scalability and distribution when processing the large dataset.
- Acknowledge the importance of NLP in real-life scenarios.
- Learn about multiprocessing and multithreading when preprocessing data.