THE UNIVERSITY OF SYDNEY

School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT

Unit of Study: COMP5349: CLOUD COMPUTING

Assignment name: SPARK ASSIGNMENT

Tutorial time: THU (4-6 PM) **Tutor name:** CHENHAO HUANG

DECLARATION

We the undersigned declare that we have read and understood the *University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy*, an, and except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that failure to comply with the *Academic Dishonesty and Plagiarism in Coursework Policy* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Project team members				
Student name	**Student ID**	**Participated**	**Agree to share**	**Signature**
1. SANNA NAZIR	490517677	✓ Yes / No	✓ Yes/No	SANNA NAZIR
2. ANSHU KUMAR	490517666	✓ Yes / No	✓ Yes / No	ANSHU KUMAR
3.		Yes / No	Yes / No	
4.		Yes / No	Yes / No	
5.		Yes / No	Yes / No	
6.		Yes / No	Yes / No	
7.		Yes / No	Yes / No	
8.		Yes / No	Yes / No	
9.		Yes / No	Yes / No	
10.		Yes / No	Yes / No	

SIT Building, J12
The University of Sydney
NSW 2006 Australia

T +61 2 9351 3423
F +61 2 9351 3838
E sit.info@sydney.edu.au
sydney.edu.au/it

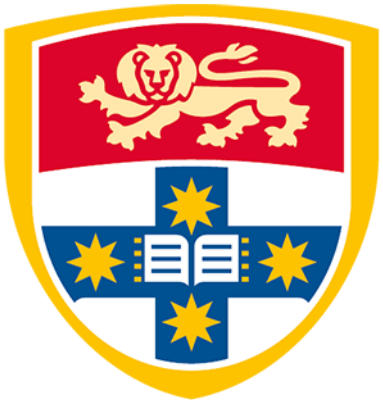
ABN 15 211 513 464
CRICOS 00026A

CLOUD COMPUTING ASSIGNMENT

Lecturer: Ying Zhou

Tutor: Chenhao Huang

Spark Application: Development and Deployment (Using AWS EMR Clusters)



THE UNIVERSITY OF SYDNEY

Anshu Kumar (490517666)
Sanna Nazir (490517677)

May 2020

Contents

1 INTRODUCTION

1.1 Aim of the project

The goal of this assignment/project is to successfully build spark applications for the prescribed workloads. Further, the aim is to successfully deploy and host these application on a Spark-enabled Cluster that can compute the workloads for the immense dataset provided to us. The underlying goal is to facilitate the students understanding of Cloud Computing Systems, Parallelism for large workloads, understanding Spark framework as well as getting acquainted with AWS services.

1.2 Application technical details

To develop the Spark Applications, we have used the **PySpark Language** i.e. the python API written in Python to support Apache Spark. The workload execution is performed using both **Spark RDD API** as well as the **Spark SQL API**. Conversions between the two are intuitive and as required.

Additionally, the **NLTK software package** has been used to execute tokenization as well as stopwords removal. The **Tensorflow package** has been used for enabling the Google Universal Sentence Encoder. The **Spark Machine Learning library** has been used to enable K-means and TF-IDF encoding processes.

The environment used for creating and hosting the applications is the AWS EMR Cluster environment hosted by Amazon. Table 1 highlights the detailed configuration of the EMR environment.

Environment Attribute Name	Configuration Details
EMR	Release label:emr-5.29.0
Hadoop	Distribution:Amazon 2.8.5
Spark	Spark 2.4.4
Livy	Livy 0.6.0
Zepplin	Zeppelin 0.8.2
Tensorflow	TensorFlow 1.14.0

Table 1: Environment Configuration for AWS EMR Clusters

The number of nodes (single or multi) used within the EMR cluster varies depending on the workload. Specific details on the cluster composition are present within 'Implementation Details' section of each workload description. However, across all clusters, configuration for maximizing resource allocation has been activated. That is, while launching any cluster, *"maximizeResourceAllocation"* setting has been set to "true."

Furthermore, each cluster is bolstered with a custom bootstrap action while launching. The script used for bootstrapping has been taken from the lab sessions of COMP5349 and provides the cluster wide installation of **tensorflow-hub package, matplotlib package as well as the data available in the nltk package (includes the tokenizer, stopwords, punctuation list etc.)**

1.2.1 Steps for environment setup to run and debug the applications

Please refer to readme.md file attached with each code file for details about environment setup and configuration changes for each workload.

2 VOCABULARY EXPLORATION

2.1 Requirement 1: Match and Mismatch Corpora

2.1.1 Implementation Details

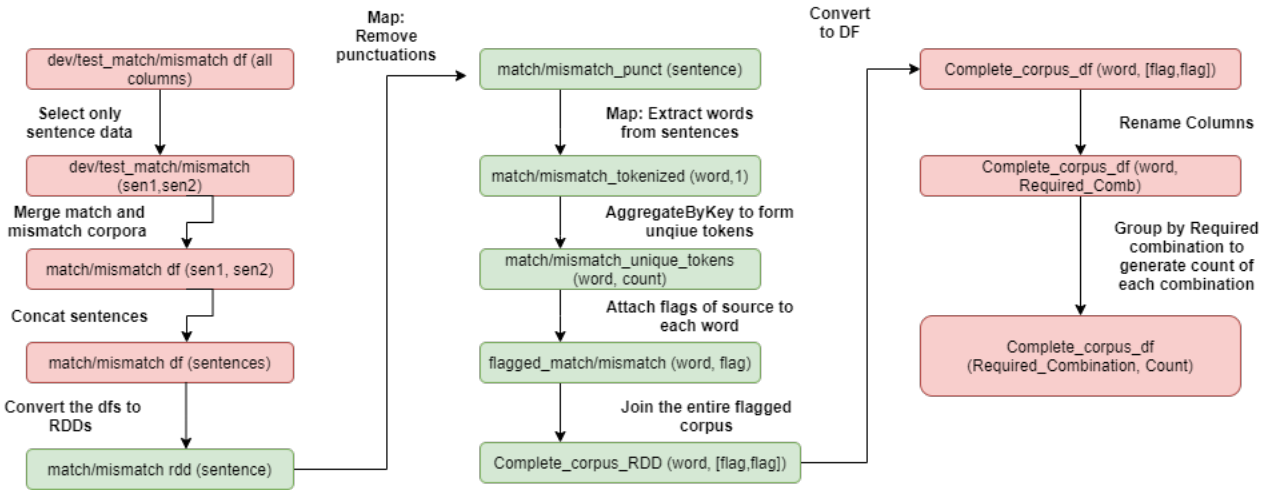


Figure 1: Flowchart for design of vocabulary exploration workload (Match and Mismatch Datasets) *(*green for type RDD, *red for type DF)*

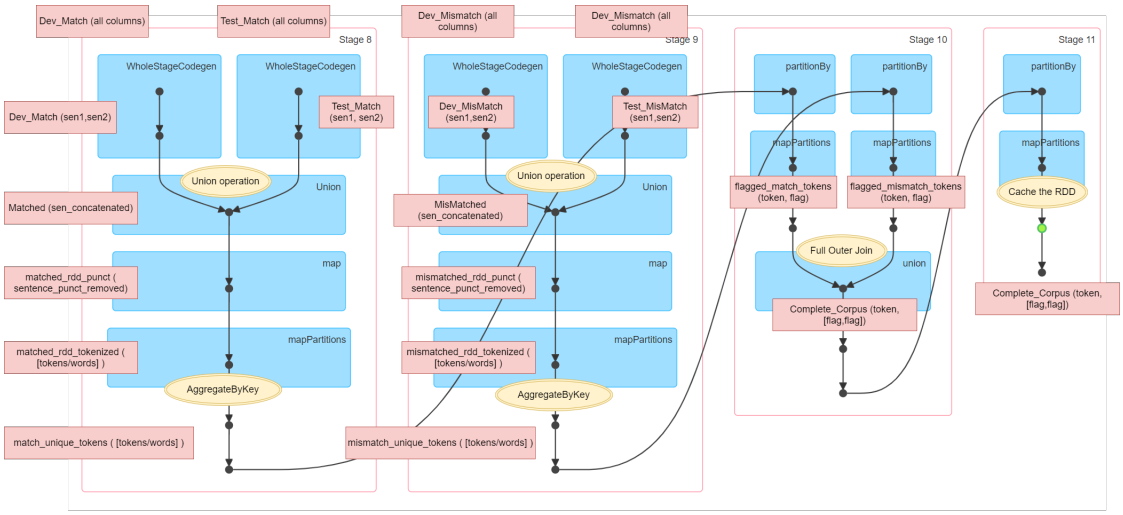
VOCABULARY EXPLORATION WORKLOAD (MATCH/MISMATCH CORPUS)			
ENVIRONMENT USED			
Cluster Type	m4.xLarge		
Master Nodes	1	Slave Nodes	0
CONFIGURATIONS USED			
spark.scheduler.mode	FIFO	spark.executor.memory	8960M
spark.driver.memory	2048M	spark.dynamicAllocation.enabled	TRUE
spark.default.parallelism	16	spark.executor.cores	8
EXECUTION STATISTICS			
Total execution time	30 seconds	Total jobs	15
CUSTOM FUNCTIONS CREATED			
punctuation_remover	Removes punctuations (punct list from string package) from sen.		
custom_tokenizer	Extracts words from a sentence using NLTK word_tokenizer		
pairFlagToWord	Attaches a flag to each word indicating the vocab source		

Figure 2: Implementation Details

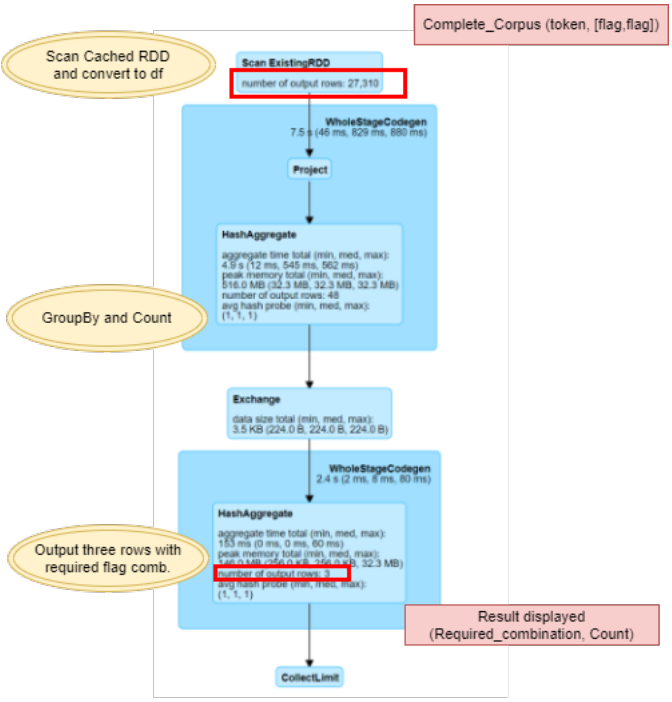
To execute the vocabulary exploration workload for match/mismatch corpora, our design and data flow can be understood from figure 1. Furthermore, all implementation details for the application are listed in figure 2.

The choice of these design considerations was made because of efficient time in executing the application as well as minimalistic use of resources for cost optimization.

2.1.2 Design Flow using Annotated Directed Acyclic Graphs



(a) Till corpus full outer join



(b) Calculating statistics from complete corpus

Figure 3: DAG for match/mismatch vocabulary exploration

Figure 3(a) depicts the DAG till the full outer join of the entire corpus is performed. The corpus consists of unique words in the combined vocabulary, each attached with a vocab source flag. Following this, it is converted to a dataframe and simple grouping and aggregation (Figure 3(b)) is performed to achieve and project the results.

2.1.3 Results

VOCABULARY EXPLORATION WORKLOAD	
MATCH/MISMATCH CORPUS	
Words common between match and mismatch sets	9276
Words unique to match vocabulary set	10305
Words unique to mismatch vocabulary set	7729
Total Unique words in the combined corpus	27310

Figure 4: Results of Match/Mismatch Data Vocabulary Exploration

2.2 Requirement 2: Training Data Corpus

2.2.1 Implementation Details

To execute the vocabulary exploration workload for training corpus, our design and data flow can be understood from figure 5. Furthermore, all implementation details for the application are listed in figure 6. (Only parts different from Q1,Part1 are mentioned here. Details not mentioned are same as in Figure 2.)

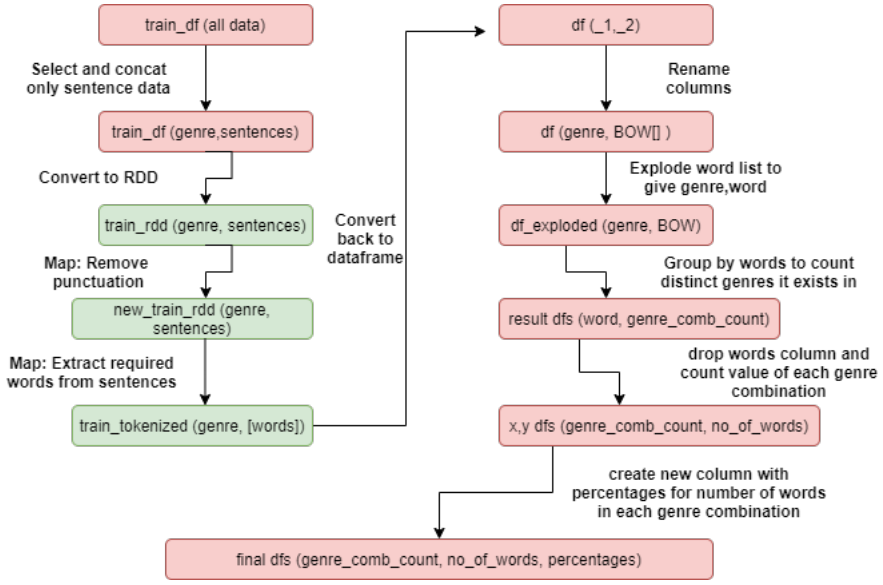


Figure 5: Flowchart for design of vocabulary exploration workload (Training Data Corpus) *(*green for type RDD, *red for type DF)*

VOCABULARY EXPLORATION WORKLOAD (TRAINING DATA)			
EXECUTION STATISTICS			
Total execution time	3.3 min	Total jobs	8
CUSTOM FUNCTIONS CREATED			
punctuation_remover	Removes punctuations (punct list from string package) from sen.		
tokens_with_stopwords	Extracts all words from a sentence using NLTK word_tokenizer		
tokens_without_stopwords	Extracts words from a sentence using NLTK word_tokenizer that are not present in a given stopwords list (imported from NLTK)		

Figure 6: Implementation Details

2.2.2 Design Flow using Annotated Directed Acyclic Graphs

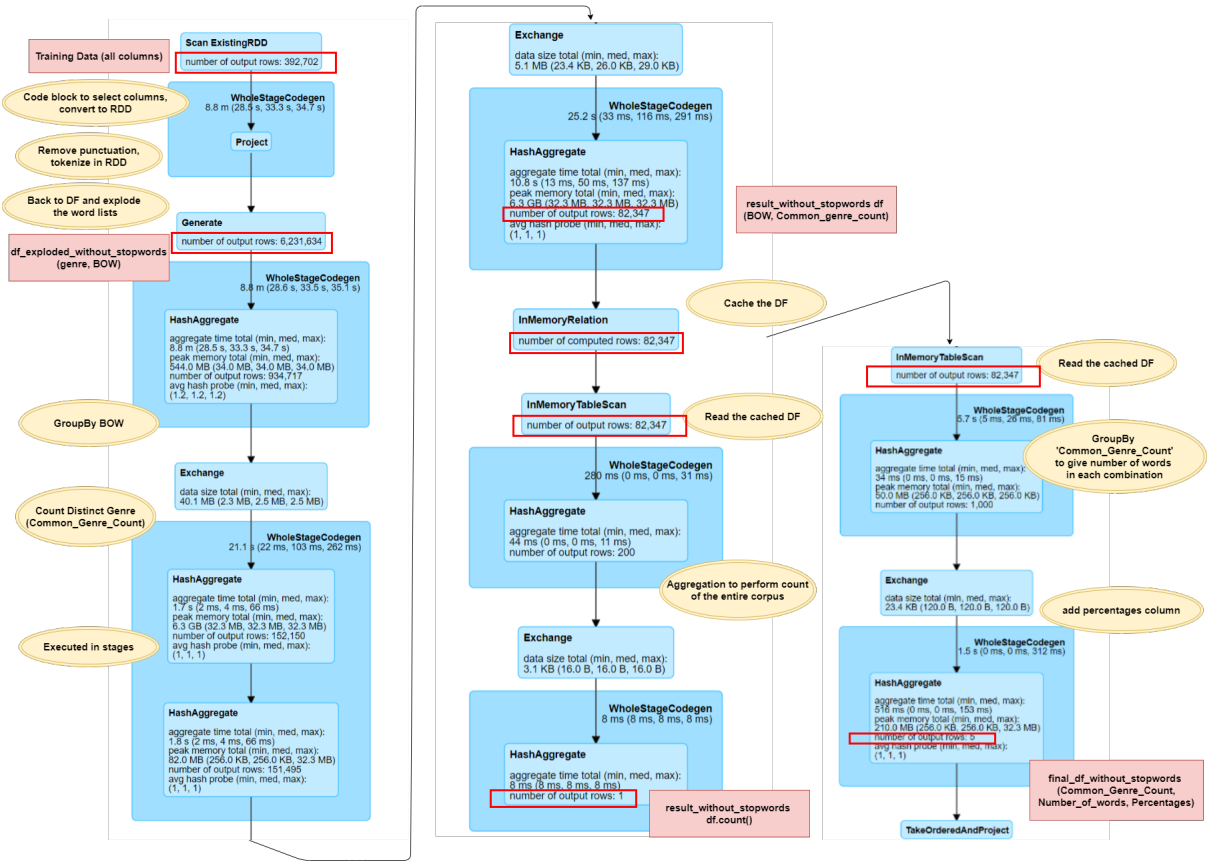


Figure 7: DAG for the training data vocabulary exploration

2.2.3 Results

VOCABULARY EXPLORATION WORKLOAD		
TRAINING DATA CORPUS (with stopwords)		
Words in just one genre (genre_count: 1)	51918	62.94%
Words in two genres (genre_count: 2)	11189	13.56%
Words in three genres (genre_count: 3)	6600	8%
Words in four genres (genre_count: 4)	5177	6.28%
Words in five genres (genre_count: 5)	7606	9.22%
Total Unique Words	82490	100.00%

(a) Corpus With Stopwords

VOCABULARY EXPLORATION WORKLOAD		
TRAINING DATA CORPUS (after removing stopwords)		
Words in just one genre (genre_count: 1)	51914	63.04%
Words in two genres (genre_count: 2)	11187	13.59%
Words in three genres (genre_count: 3)	6597	8%
Words in four genres (genre_count: 4)	5174	6.28%
Words in five genres (genre_count: 5)	7475	9.08%
Total Unique Words	82347	100.00%

(b) Corpus Without Stopwords

Figure 8: Results of Training Data Corpus Vocabulary Exploration

3 SENTENCE VECTOR EXPLORATION

The workload for sentence vector encoding task was carried out in two ways: by generating sentence encoding using TFIDF method and by using Google's Universal Sentence Encoder (USE).

3.1 Design and Data Flow for clustering based on TF-IDF Encoding

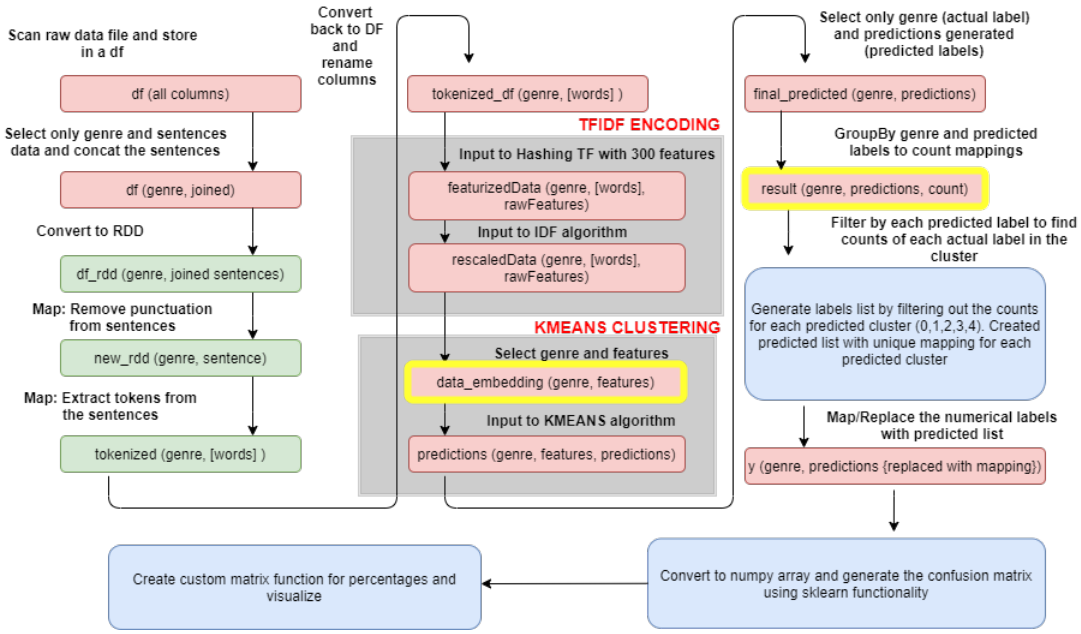


Figure 9: Flowchart for sentence vector encoding using TF-IDF *(*yellow for cached df, *green for type RDD, *red for type DF, *blue for regular python data structures and functions)*

3.2 Design and Data Flow for clustering based on Google Universal Sentence Encoder (USE)

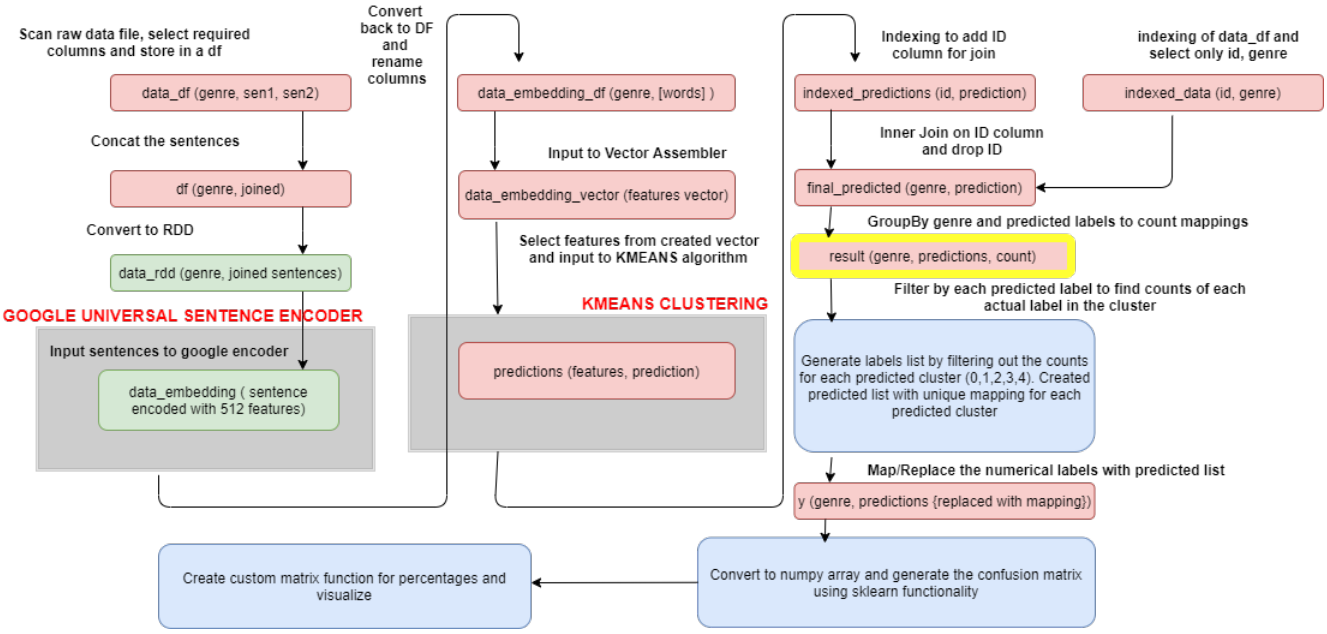


Figure 10: Flowchart for sentence vector encoding using Google USE*(*yellow for cached df, *green for type RDD, *red for type DF, *blue for regular python data structures and functions)*

3.3 Implementation Details

The below tables highlight the environment and configuration settings required to implement the task of clustering using the two encoding methods.

K-MEANS CLUSTERING USING TF-IDF ENCODING			
ENVIRONMENT USED			
Cluster Type	m4.xLarge		
Master Nodes	1	Slave Nodes	4
CONFIGURATIONS USED			
spark.scheduler.mode	FIFO	spark.executor.memory	8960M
spark.driver.memory	2048M	spark.dynamicAllocation.enabled	TRUE
spark.default.parallelism	16	spark.executor.cores	8
EXECUTION STATISTICS			
Total execution time	30 seconds	Total jobs	15
Completed Stages	73	Skipped Stages	19
PACKAGES USED			
VectorAssembler (Spark ML)	IDF (Spark ML)	Concat (Spark SQL)	Punctuation (String)
KMeans (Spark ML)	StopWords (NLTK)	HashingTF (Spark ML)	Seaborn
Lit (Spark SQL)	Sklearn	Word_Tokenize (NLTK)	Matplotlib

(a) TFIDF Method

K-MEANS CLUSTERING USING GOOGLE USE ENCODING			
ENVIRONMENT USED			
Cluster Type	m4.xLarge		
Master Nodes	1	Slave Nodes	4
CONFIGURATIONS USED (CHANGES FROM THE DEFAULT VALUES)			
yarn.nodemanager.pmem-check-enabled	FALSE	yarn.nodemanager.vmem-check-enabled	FALSE
spark.default.parallelism	70	spark.dynamicAllocation.enabled	DISABLED
spark.driver.memory	7372M	spark.executor.memory	7372M
spark.executor.instances	7	spark.executor.cores	3
spark.network.timeout	10000s	spark.executor.heartbeat.Interval	80s
spark.yarn.driver.memoryOverhead	819M	spark.yarn.executor.memoryOverhead	819M
EXECUTION STATISTICS			
Total execution time	36 min	Total jobs	82
PACKAGES USED			
VectorAssembler (Spark ML)	Sklearn	Lit and Concat (Spark SQL)	Matplotlib
KMeans (Spark ML)	StopWords (NLTK)	Tensorflow & Tensorflow Hub	Seaborn

(b) USE Method

Figure 11: Implementation and Environment details for the Sentence Vector Exploration methods

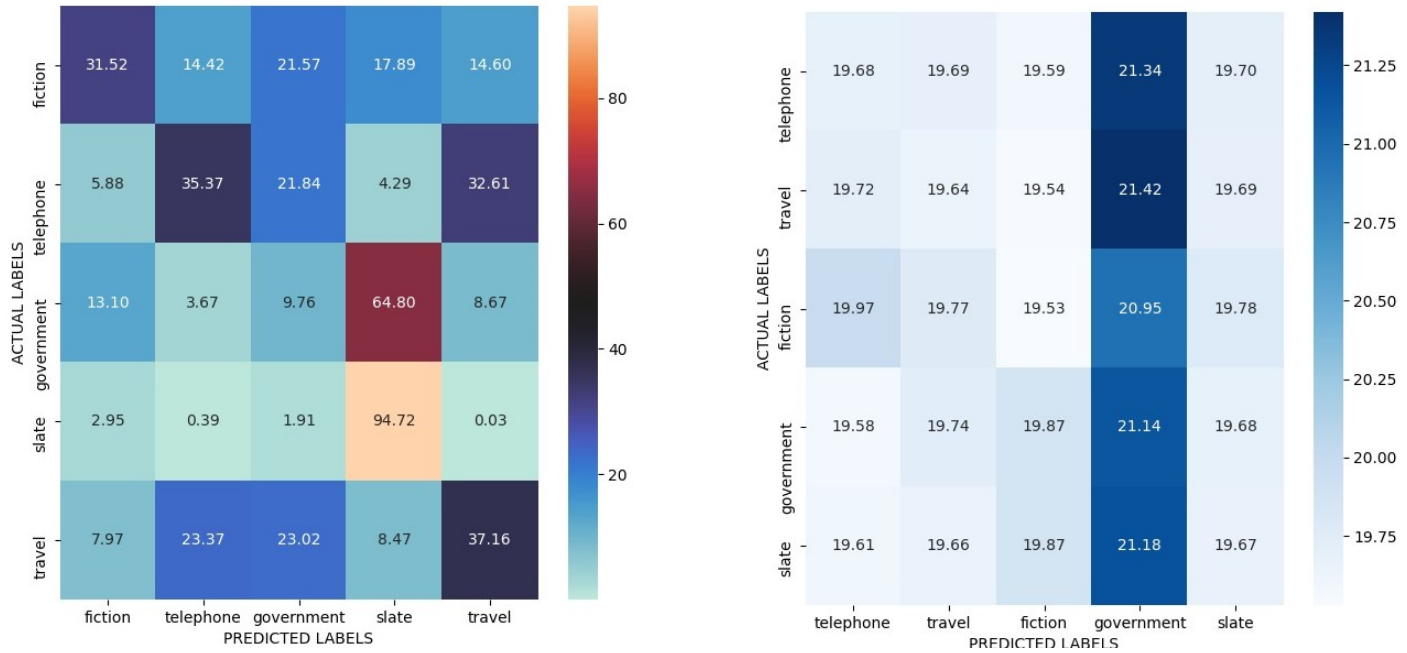
3.3.1 Optimization and configuration changes for the USE-Clustering

On running the USE method for the clustering task, we encountered the problem of "container getting killed because memory limits were getting exceeded." This happened on running the default configurations for the m4.xlarge 1-Master,4-Core/Slave Cluster architecture. To eliminate this problem, we introduced various changes to the "Hadoop Configuration" as well as the "Spark Configuration." These changes, made using the SSH connection to the cluster, are as below:

Configuration	Default Value	Custom value
yarn.nodemanager.vmem-check-enabled	true	false
yarn.nodemanager.pmem-check-enabled	true	false
spark.dynamicAllocation.enabled	true	(Hashed out the line)
spark.executor.instances	4	7
spark.executor.cores	8	3
spark.driver.memory	11171M	7372M
spark.yarn.executor.memoryOverhead	10%	819M
spark.yarn.driver.memoryOverhead	10%	819M
spark.default.parallelism	64	42

Table 2: Configuration Changes and Optimization for USE-Clustering

3.4 Results (Confusion Matrix)



(a) TFIDF Method

(b) USE Method

Figure 12: Results of Sentence Encoding and Clustering Workload

4 PERFORMANCE EVALUATION

The below table highlights the different configurations we have tested for the purposes of the performance evaluation. Each configuration has been established on a cluster of 5 nodes (1 Master and 4 Slaves), with the m4.xlarge provisioning for each node. Furthermore, the evaluation is based on running the K-Means Clustering (using TF-IDF encoding) Workload. Code used is the same as submission code for Question2_Part2_GoogleEncoder.

CONFIGURATION DETAILS				
PARAMETER	CONFIG 1	CONFIG 2	CONFIG 3	CONFIG 4
spark.executor.instances	4	8	32	6
spark.executor.cores	8	4	1	8
spark.driver.memory	11171M	5529M	1382M	11171M
spark.executor.memory	10356M	5529M	1382M	10356M
spark.default.parallelism	64	64	64	96
spark.driver.memoryOverhead	1000M	614M	153M	1000M
spark.executor.memoryOverhead	1900M	614M	153M	1900M
spark.dynamicAllocation.enabled	TRUE	FALSE	FALSE	TRUE
yarn.nodemanager.pmem-check-enabled	TRUE	FALSE	FALSE	TRUE
yarn.nodemanager.vmem-check-enabled	TRUE	FALSE	FALSE	TRUE
MEASURED STATISTICS				
execution time taken	2m 43s	3m 5s	5m 10s	2m 18s
Total jobs	43	43	43	43
Total stages	73 (19 skipped)	73 (19 skipped)	73 (19 skipped)	73 (19 skipped)
Number of threads	8	4	1	8

Figure 13: Comparison of different configurations for the same workload.

4.1 Comparative Analysis

The key points for drawing comparisons in the above table is the behavior of the executors (determined by their first two properties in the table). Increasing the number of executors entails increasing the instances for the executor. And increasing the capacity for the executor entails increasing the cores of each executor (consequently, affecting the memory for each as well.) The major points of evaluation as drawn from the above table are as below:

- Increasing the number of executors (4-32) but decreasing their capacity (1 core, 1382M memory) leads to an increase in the time for execution of the workload. The increase is almost two fold. (Config3)
- Simultaneously, decreasing the number of executors while increasing capacity for each results in better performance in lesser time. (Config1)
- On the other hand, if we increase the number of core instances while maintaining the capacity for each (config4), we achieve better performance as parallelism gets increased and thus, execution time decreases.

5 CONCLUSION

The aim of this project was to enhance our capabilities around Spark, AWS, Hadoop and the greater cloud computing domain. Viewed microscopically, we have successfully executed all tasks at hand and have been able to appreciate the knowledge delivered in class with hands-on practice. Macroscopically, the project has prepared us for the real-world application of cloud computing and imbibed in us the understanding of how data-heavy, intensive applications that employ complex ML algorithms can be deployed in the market.