# http://huisstijl.vub.ac.be/wp-content/uploads/2016/09/het-vub-logo.png

**Distributed Computing and Storage Architectures**

|  |  |
| --- | --- |
|  | **Project Map Reduce  Tasks Report** |
|  | **Members**   |  |  | | --- | --- | | Deniz Alp ATUN – 0552182  [deniz.alp.atun@vub.be](mailto:deniz.alp.atun@vub.be) | Siddheswar Mukherjee – 0554647  [siddheswar.mukherjee@vub.be](mailto:siddheswar.mukherjee@vub.be) | |

Table of Contents

[1](#_Toc29643640)

[1. Introduction 2](#_Toc29643641)

[2. IMDB Tasks 3](#_Toc29643642)

[2.1. **Task 1:** 50 most common keywords for all movies and shorts 3](#_Toc29643643)

[2.2. **Task 2:** Top 15 keywords for each movie genre 3](#_Toc29643644)

[3. Online Retail Tasks 3](#_Toc29643645)

[3.1. **Task 3:** Top 10 best customers for each retail year 3](#_Toc29643646)

[3.2. **Task 4:** Best selling product 3](#_Toc29643647)

[4. Similar Paper Recommendation Tasks 3](#_Toc29643648)

[4.1. **Task 5:** Jaccard similarity coefficient 3](#_Toc29643649)

[4.2. **Task 6:** Cosine similarity 3](#_Toc29643650)

[5. Matrix Multiplication Task 3](#_Toc29643651)

[5.1. **Task 8:** Matrix dot product 3](#_Toc29643652)

# Introduction

In this project we are using a map reduce to complete various real-world tasks.  
The map reduce part is provided by MRJob Library, thus allowing us to focus on logic of the tasks rather than the implementation part of map reduce.

Below you will find members assigned tasks and their respective reports.

|  |  |
| --- | --- |
| Members | Tasks |
| Deniz | **[ Task 1, Task 5, Task 8 ]** |
| Siddheswar | **[ Task 2, Task 3, Task 4]** |

# IMDB Tasks

In this task, we are provided by a .tsv (tab separated values) file, which contains basic information regarding titles that are found on IMDB.

## **Task 1:** 50 most common keywords for all movies and shorts

To do this task, the process is divided into two parts.  
First part is to mapping word frequencies; second part is sorting and yielding top 50.

**First Part:**Since what every column is representing is provided by PDF, it can be deduced that we will only require column[1] for show-types and column[2] for primaryTitles.

Mapper part contains 4 filtering operations to yield a word list to be processed by combiner.

* + 1. An if statement to determine if title type is a movie or short, else skip.
    2. Regular expression that separates all non-whitespace and non-digits with more than 2 characters. 2 character is used as a broad way to reduce short possessives in multiple languages.
    3. Natural language toolkit is used to part-of-speech tagging and removing tags such as prepositions, determiners, articles, punctuation etc for **English** language.
    4. Due to NLTK working on English language, to further reduce input NLTK stopwords for all languages are used.  
       This is done since the input file contains English, French, German, Spanish and more languages that passes through pos-tagging.

After mapper result, combiner combines the word keys with their counts to yield number of times that word was seen.

Then reducer changes the (key,value) pair to (None, (value,key)) to be processed in second part.

**Second Part:**  
Sorter is a reducer function that takes first 50 elements passed into it and then removes minimum value when a higher value is found that will be added instead.

## **Task 2:** Top 15 keywords for each movie genre

# Online Retail Tasks

## **Task 3:** Top 10 best customers for each retail year

## **Task 4:** Best selling product

# Similar Paper Recommendation Tasks

In this task we are provided with a .JSON file that contains information of multiple articles.

## **Task 5:** Jaccard similarity coefficient

Due to file format being json, we have to define a different input method than the provided one since it passes values line by line.

Thus we have 3 steps: mapper\_raw, jaccard\_sim (combiner), reduce\_max\_sim (reducer).

**mapper\_raw step**Initially loads json file using json library, then picks one random paper using random library.

The random picking and rest of the files go through the process of string to list function that removes punctuation of , . ( ) and any whitespace then outputs lowered version of string.  
The output is then passed to list to dictionary which maps words with their occurrences.

**Jaccard\_sim step**  
This step then takes every (id, dictionary) pair and compares to the previously randomly picked paper. Yields the jaccard coefficient and id of that paper.

This step went through many iterations due to the description of jaccard coefficient.

(..) measures similarity as the intersection divided by the union of the objects.   
(...) The Jaccard coefficient compares the sum weight of shared terms to the sum weight of terms that are present in either of the two document   
but are not the shared terms.

First it was implemented in both numpy and base libraries, considering only the intersection divided by union of objects *(can be seen in JacardCoef\_1)*.   
This case didn’t take consideration of repeated words or their respective weights.   
The current implementation uses dictionary and weight values to do that.

**Reduce\_max\_sim step**  
This step then looks into the output of previous step and yields maximum value.  
Providing an output of   
Randomly Picked ID, [ Maximum Jaccard Result, Maximum Matched Result ]

## **Task 6:** Cosine similarity

# Matrix Multiplication Task

## **Task 8:** Matrix dot product

Initially “MapReduce (Advanced Topics): Part 3” pdf is followed as a basis and inspiration, but as soon as mapping started the project started to deviate.

This implementation requires two steps which uses pairs of mappers and reducers.

**In below description;**   
First matrix is referred as A and it’s (row, column) is (i, k).  
Second matrix is referred as B and it’s (row, column) is (k ,j)

**Step 1:**

First to have more control on how mapper is working, mapper\_raw is modified for our purposes.  
It takes two inputs A.txt and B.txt from command line and depending on file name that is in process, elements of the matrix is mapped differently.

When compared to PDF, at combiner step it is combining many duplicate values to be used in multiplication due to the nature of matrix dot product.

Instead of doing that, reducer takes the output of mapper and calculates element multiplication part of dot product. Considering a 2x3 and 3x2 matrix multiplication below:

This would mean instead of mapping a multiple times, first reducer is producing ax, by, cz and so on.  
The yield is then mapped as (i,j), (A[i][k] \* B[k][j])

**Step 2:**  
This step is to map elements with same keys so that they will be summed together.  
Mapper maps the elements with same keys (0,0), (ax,by,cz) for example.  
Then the reducer sums them together (0,0), ax+by+cz