Distributed Computing and Storage Architectures

Project Map Reduce Tasks Report

Members

Deniz Alp ATUN – 0552182
deniz.alp.atun@vub.be

Siddheswar Mukherjee – 0554647
siddheswar.mukherjee@vub.be

Table of Contents

1. 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, Task 6]**

2. 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.

2.1. 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.

2.2. Task 2: Top 15 keywords for each movie genre

The steps followed are as follows:

mapper1

This returns (genre, title) pairs. It must be noted the genre columns may have multiple genres associated with one movie title. Hence these genres are identified individually and the pairs are created accordingly.

mapper2

This mapper job returns the genre and converts the title into a word count dictionary.

combiner and reducer1

In these steps, the dictionary associated with each genre are combined. These were generated by the previous job – mapper2. Each instance of the reducer receives all pairs that have the same gerne (which is the key in this case).

Finally the data structure consists of each genre and an associated dictionary that contains the word count combined from all titles associated with that genre. Thereafter the top 15 entries in the dictionaries of each genre is isolated and returned by the reducer.

3. Online Retail Tasks

The following tasks were executed using the csv files provided as a part of the exercise. It was observed that some billing entries are repeated in both the files. Hence necessary steps were taken to eliminate redundant lines.

3.1. Task 3: Top 10 best customers for each retail year

mapper, combiner and reducer

The MRjob conssting of *mapper, combiner* and *reducer* are used to preprocess and remove the redundant lines as described above. The reducer finally produces key value pair where the key consists of the customer id and year and value is the revenue generated.

combiner1, reducer1

These steps sum up the revenues generated by customers per year. The reducer1 output consists of key = year and value = a dictionary consisting of customer_id and the total revenue generated for that year.

combiner2 , reducer2

These jobs append the dictionaries generated by reducer1 so that there is one dictionary associated with each year (which in case is 3: 20019, 2010, 2011) .

The top 10 entries from each dictionary is calculated using the *most_common* function, which gives the final key, value output where key = year and value = list of top 10 customers along with the total revenue generated.

3.2. Task 4: Best selling product

This task was very similar to task 3. The preprocessing was done by the jobs: *mapper, combiner, reducer.* The only difference is that the customer ids are no longer tracked for each bill.

reducer1

This reducer produces the total quantities and the revenues generated for each product. In order to facilitate sorting in the next step, the key is yielded as *None* and the value consists of the triples: total quantity, total revenue and stock id.

reducer2

Finally the products generating the maximum revenue and quantity are calculated in this step which produces the final output.

4. Similar Paper Recommendation Tasks

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

4.1. 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.

$$i = sum\ of\ only\ mins\ of\ intersecting\ values$$
$$a = sum\ of\ input\ values\ of\ A;\ \ b = sum\ of\ input\ values\ of\ B$$
$$Jaccard\ Dist = 1 - \frac{i}{a + b - i}$$

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]

4.2. Task 6: Cosine similarity

Similar to the previous section, the first mapper has to *mapper_raw* in order to tackle the JSON input file. The steps followed are as follows:

mapper_raw step
Initially loads json file using json library, then picks one random paper using random library.

Each mapper then returns the id, title and the content of each article. The text content is preprocessed to remove all punctuations and numbers. It is also stemmed in order to facilitate better similarity calculation. Snowball stemmer was preferred here.

Reducer

TF-IDF

The reducer will receive all articles from all mappers. There after the TF-IDF vector is calculated for each article. It must be noted that TF-IDF calculation is different from word counter and requires the presence of all articles together.

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

t = word ,
d = a particular document
D = all documents.
tf(t,d) = term frequency of word t in the document d.
document frequency (df) = The ratio of number of documents in which the word d appears and the total number of documents in the corpora.
Inverse document frequency (idf(t,D)) = log(1/ df)

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

After the calculation of TF-IDF vectors for every article, the cosine similarity can be calculated using these vectors.

Cosine Similarity

The cosine similarity between two vectors is the cosine of the angle between the vectors. The mathematical formulae is as follows: [1]

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Here A and B are the vectors.

Python Implementation

It must be noted that the cosine similarity does not take into consideration the magnitude of the vectors. Also the TF-IDF component outputs the vectors in row normalised format. [2]

Hence the dot product of the vectors is sufficient to calculate the cosine of the angle. This tep is performed by the *linear_kernel* function of the *scikit learn* library. The linear kernel calculation is as follows:

$$k(x, y) = x^\top y$$

This step eliminates the running nested loops and is hence faster.

The linear kernel step produces the final (n x n)cosine similarity matrix where n is the number of articles. Each row of the matrix belongs to a particular article and has the similarity scores with all other articles.

The *randompick* function chooses a random article id (x) and the final output of the reducer is the element in the similarity matrix , corresponding to the row of x, which has the highest similarity score.

The reducer finally returns the (id, title) for the most similar article.

5. Matrix Multiplication Task

5.1. 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.

$$For\ matrix\ A = \ k, ("A", i, A[i][k]) \qquad For\ matrix\ B = \ k, ("B", k, B[k][j])$$

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:

$$\begin{matrix} a & b & c \\ d & e & f \end{matrix} \times \begin{matrix} x & u \\ y & v \\ z & w \end{matrix} = \begin{matrix} ax + by + cz & au + bv + cw \\ dx + ey + fz & du + ev + fw \end{matrix}$$

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

6. REFERENCE

[1] [Online]. Available: https://github.com/taki0112/Vector_Similarity.

[2] [Online]. Available: https://stackoverflow.com/a/12128777/4168707.