Distributed Computing and Storage Architectures

Project MapReduce 2021-2022

Project Overview

In this project I have used MapReduce in order to solve several tasks including: counting, reverse web-graph link, KNN algorithm and matrix multiplication. Each of the tasks has its own dataset and requirements, as explained in the project description file. MapReduce is a programming model and implementation which, allows us to process and generate large amounts of data, by splitting them in smaller chunks and performing parallel operations on a cluster e.g., Hadoop. All the code is written in Python, by using MRJob framework, which allows us to program with MapReduce.

- Mapper: Takes an input and processes a logical key-value pair to generate a set of intermediate key-values.
- Combiner: Serves as a local reducer for each map.
- Reducer: Merges all intermediate values, associated with the same intermediate key and outputs the result.

The code for the mapper and reducer functions can be written in one single class. We can specify multiple steps that consist of at least one of the functions.

Contents

Task 1

1. Dataset

For this task I have been working on the Movie Lens dataset. The dataset file name in the project folder is **movies.csv**. It consists of over sixty thousand rows and three columns: movieId, title and genres.

2. Task description

As specified in the project description, the goal in this task is to find the top 10 most common keywords in the titles, for each unique genre. Additionally, title words should be filtered to avoid numbers (years), auxiliary verbs, conjunctions, prepositions, and articles as keywords. The output should be each genre and its associated list of the top 10 most used keywords and their counter.

3. Dependencies

- MRJob and MRStep – Python library for writing MapReduce.
- RE – Regular Expression Operations.
- NLTK – Library for human language data processing.
- SimpleJSON – JSON serializer.

4. Run project command

- Python main.py movies.csv

5. Solution

I have created a python class MovieLens that imports MRJob library as a parameter. Inside this class are written all the map and reduce functions needed for this task.

1. First step:

- Mapper – takes as a parameter key: None, and value: each line of the input file. In this case movies.csv.

  Since the data in the dataset has not a fixed format, I have accessed each data of the row according to their columns, by splitting the line. Most of the data have a comma separation between them, so I have split the line with comma. Some other data have nested quotations or other special characters. Therefore, I checked the length of line with split by comma and if it is not three (each column), I split the line by quotes. In this way I have access equally to all the data in the dataset, regardless of their format and I unpacked each line according to the columns: movieId, title, genres.

  Afterwards, I filtered the genres with a regular expression that matches only letters, spaces, and lines. Additionally, I needed to also filter the titles. I made another regular expression that matches only letters and I used NLTK library stop words as suggested in the project task description. It ignores all the common words in English such as: “the”, “a”, “in” etc. I also used the function isalnum, which removes the

special characters. Furthermore, I filtered only the title words that have a length of higher than two, so that I could pick up more significant keywords.

This mapper yields a list of genres and title words as key, and a count of 1 for each as value. The format is: [genre, title], 1.

```
["Musical", "Killing"]  1
["Musical", "Chinese"]  1
["Musical", "Bookie"]   1
["Action", "Run"]       1
["Drama", "Run"]        1
["Thriller", "Run"]     1
["Sci-Fi", "Things"]    1
["Sci-Fi", "Come"]      1
```

Fig 1. Mapper Output

- Combiner – takes as a parameter key: [genre, title] obtained from the mapper and as value counts, which are 1.
  This combiner sums locally the title words we have seen so far, for each genre.
  Returns [genre, title], sum(counts)

```
["Comedy", "People"]    7
["Comedy", "Perched"]   1
["Comedy", "Perfect"]   2
["Comedy", "Perry"]     4
["Comedy", "Pettigrew"] 1
["Comedy", "Philanthropy"]      1
["Comedy", "Phillip"]   1
["Comedy", "Philosopher"]       1
```

Fig 2. Combiner Output

- Reducer - takes as a parameter key: [genre, title] obtained from the combiner and as value the number of occurrences of the word obtained from the combiner. Sums the title words we have seen so far, for each genre.
  Returns: genre, (word, sum)

```
"Documentary"   ["Misconception", 1]
"Documentary"   ["Misery", 1]
"Documentary"   ["Miss", 6]
"Documentary"   ["Missing", 3]
"Documentary"   ["Mission", 7]
"Documentary"   ["Mississippi", 1]
"Documentary"   ["Mist", 1]
"Documentary"   ["Mistaken", 1]
```

Fig 3. Reducer Output

2. Second step:
    - Reducer - takes as a parameter key: all the unique genres obtained from the first step and as value [word, sum]. This reducer unpacks the values and sorts dhe list. Since the value is of type generator, I have used simplejson, in order to serialize it as JSON and perform the sorting in descending order, according to the count. I have outputted only the top 10 most common title keywords in descending order for each genre.


```
"Action"	"[[\"Man\", 132], [\"Death\", 77], [\"Black\", 73], [\"Dragon\", 64], [\"Last\", 64], [\"Blood\", 61], [\"Dead\", 60], [\"Movie\", 60], [\"City\", 55], [\"King\", 55]]"
"Adventure"	"[[\"Man\", 72], [\"Movie\", 64], [\"King\", 58], [\"Adventures\", 57], [\"Island\", 55], [\"Lost\", 54], [\"Last\", 52], [\"World\", 52], [\"Dragon\", 49], [\"Time\", 43]]"
"Animation"	"[[\"Movie\", 141], [\"Little\", 52], [\"Christmas\", 46], [\"Barbie\", 35], [\"Doo\", 35], [\"Scooby\", 34], [\"Time\", 33], [\"Donald\", 31], [\"Dragon\", 31], [\"Legend\", 30]]"
"Children"	"[[\"Christmas\", 119], [\"Movie\", 86], [\"Little\", 69], [\"Adventures\", 45], [\"Doo\", 35], [\"Scooby\", 34], [\"Barbie\", 32], [\"Time\", 32], [\"Secret\", 31], [\"Dog\", 30]]"
"Comedy"	"[[\"Love\", 343], [\"Man\", 234], [\"Movie\", 176], [\"Girl\", 158], [\"Life\", 150], [\"Night\", 136], [\"Big\", 131], [\"Little\", 129], [\"Christmas\", 122], [\"One\", 107]]"
"Crime"	"[[\"Man\", 100], [\"Night\", 68], [\"Murder\", 60], [\"Black\", 56], [\"Death\", 55], [\"Killer\", 51], [\"City\", 50], [\"Crime\", 44], [\"Dead\", 43], [\"Kill\", 43]]"
"Documentary"	"[[\"Story\", 164], [\"Life\", 129], [\"Man\", 91], [\"World\", 84], [\"American\", 68], [\"Love\", 62], [\"Last\", 55], [\"Movie\", 54], [\"Death\", 49], [\"America\", 48]]"
"Drama"	"[[\"Love\", 493], [\"Man\", 384], [\"Life\", 241], [\"Story\", 227], [\"Girl\", 225], [\"Night\", 221], [\"Last\", 206], [\"Woman\", 161], [\"One\", 159], [\"Day\", 158]]"
"Fantasy"	"[[\"Christmas\", 44], [\"Movie\", 38], [\"Time\", 35], [\"King\", 34], [\"Man\", 31], [\"Dragon\", 29], [\"Story\", 28], [\"Night\", 26], [\"One\", 26], [\"Legend\", 25]]"
"Film-Noir"	"[[\"Night\", 14], [\"City\", 12], [\"House\", 10], [\"Woman\", 9], [\"Big\", 8], [\"Murder\", 7], [\"Street\", 7], [\"Angel\", 6], [\"Dark\", 6], [\"Man\", 6]]"
"Horror"	"[[\"Dead\", 171], [\"Night\", 148], [\"Blood\", 129], [\"House\", 121], [\"Dark\", 98], [\"Death\", 82], [\"Evil\", 72], [\"Devil\", 71], [\"Man\", 67], [\"Vampire\", 65]]"
"IMAX"	"[[\"Man\", 7], [\"Harry\", 6], [\"Potter\", 6], [\"Dark\", 5], [\"Dragon\", 4], [\"Fast\", 4], [\"Sea\", 4], [\"Spider\", 4], [\"Star\", 4], [\"Aliens\", 3]]"
"Musical"	"[[\"Love\", 23], [\"Story\", 19], [\"Movie\", 17], [\"Girl\", 16], [\"Broadway\", 14], [\"Girls\", 12], [\"Little\", 12], [\"Rock\", 12], [\"Man\", 11], [\"Opera\", 10]]"
"Mystery"	"[[\"Murder\", 64], [\"Man\", 57], [\"Night\", 54], [\"Death\", 52], [\"Mystery\", 40], [\"House\", 37], [\"Black\", 36], [\"Detective\", 34], [\"Dark\", 32], [\"Charlie\", 31]]"
"Romance"	"[[\"Love\", 424], [\"Girl\", 120], [\"Man\", 105], [\"Christmas\", 71], [\"Time\", 68], [\"Life\", 67], [\"Woman\", 61], [\"Night\", 60], [\"Story\", 58], [\"Last\", 54]]"
"Sci-Fi"	"[[\"Man\", 97], [\"Space\", 74], [\"Time\", 61], [\"Alien\", 56], [\"Planet\", 49], [\"Star\", 49], [\"World\", 48], [\"Earth\", 46], [\"Monster\", 40], [\"Godzilla\", 34]]"
"Thriller"	"[[\"Man\", 144], [\"Night\", 128], [\"House\", 98], [\"Dark\", 92], [\"Dead\", 88], [\"Death\", 82], [\"Blood\", 76], [\"Killer\", 73], [\"Last\", 72], [\"Black\", 71]]"
"War"	"[[\"War\", 77], [\"Battle\", 35], [\"Last\", 25], [\"Red\", 23], [\"Hell\", 18], [\"Love\", 18], [\"Men\", 16], [\"Soldier\", 16], [\"Days\", 15], [\"Story\", 15]]"
"Western"	"[[\"Man\", 54], [\"West\", 46], [\"Gun\", 28], [\"Wild\", 25], [\"Trail\", 22], [\"Django\", 19], [\"Last\", 19], [\"Texas\", 19], [\"Guns\", 18], [\"River\", 18]]"
"no genres listed"	"[[\"Love\", 72], [\"Man\", 63], [\"Story\", 52], [\"Life\", 42], [\"Live\", 34], [\"Last\", 32], [\"One\", 32], [\"Time\", 30], [\"Night\", 28], [\"Christmas\", 26]]"
```


Reducer Final Output

Task 2

6. Dataset

For this task I have worked with the Google Web Graph dataset. In the project folder it has the name web-Google.txt. This dataset consists of a very large number of rows that have unordered node pairs: FromNodeId – ToNodeId. In total there are 875713 Nodes and 5105039 Edges. Nodes represent web pages and edges represent hyperlinks between them.

7. Task description

As specified in the project description, the goal is to reverse the web-link graph from the given dataset. The output should be each node and its corresponding list of other nodes linking to them.

8. Dependencies

- MRJob and MRStep – Python library for writing MapReduce.

9. Run project command

- Python main.py web-Google.txt

10.Solution

I have created a python class ReverseGraph that imports MRJob library as a parameter. Inside this class are written all the map, reduce functions needed for this task. It has only one step.

- Mapper – takes as a parameter key: None, and value: each line of the input file. In this case web-Google.txt.

  The data format is really simple. There are only two columns divided by empty spaces. Therefore, I split each line by the empty spaces and skipped the first couple of lines that describe the dataset. I got a good understanding of the functions input and output by looking at the hand notes that professor Deligiannis has posted on canvas.

  This mapper yields in reverse order: toNode as key and fromNode as value.

```
"28226"  "274257"
"325450"        "274257"
"327884"        "274257"
"375600"        "274257"
"413594"        "274257"
"525053"        "274257"
"539792"        "274257"
"567615"        "274257"
"570066"        "274257"
```

Fig 4. Mapper Output

- Reducer – takes as parameter key: toNode and as a value: fromNode generated from the mapper.

  Yields toNode as a key and its corresponding list of all nodes linking to it as value. Yield: toNode, list(fromNode). In simpler terms, all the nodes in the list are links to the node in the key.

```
"117684"        ["726490", "274837", "114847", "503969", "242225"]
"117685"        ["185601"]
"117686"        ["352138", "341060"]
"117688"        ["391981", "436264", "643006", "764557"]
"117691"        ["883231", "86133", "451814", "622407", "648780", "762709", "795029", "837208", "252990"]
```

Fig 5. Reducer Output

Task 3

11. Dataset

For this task I have worked with the Iris dataset, which in the project folder has the name Iris.csv. This dataset consists of four features that were measured for three Iris species. In total there are 150 rows and 6 columns: Id, SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm and Species classification.

12. Task description

There are some Iris in the dataset that are not classified. As specified in the project description, the goal is to classify those unknown species based on the four features, by implementing the k-nearest neighbours (KNN) algorithm with K=15. To measure the distance in KNN we will use the Euclidian distance:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

The output should be the Id of the unknown species associated by our result classification.

13. Dependencies

- MRJob and MRStep – Python library for writing MapReduce.
- Math (sqrt) – built in function to calculate the mathematical operations.
- Sklearn (preprocessing) – Library for normalizing dhe dataset.
- Pandas – Popular library for reading .csv file.

14. Run project command

- Python main.py Iris.csv

15. Solution

I have created a python class KNN that imports MRJob library as a parameter. Inside this class are written all the map and reduce functions needed for this task.

16. Constructor

As a starter, I have created a class constructor in order to read the csv file with pandas and store them in constructor variables. This way I can access the data anywhere in the code. Pandas returns a data frame that makes it easier to access the data, either by rows or columns. With Pandas you need to specify the absolute file path in order to access it. Therefore, you need to put your own device's absolute file path in order to work. Since, I don't need to normalize all the columns, I selected only the feature columns and passed them to sklearn pre-processing MinMaxScaler function, to normalise the dataset. All the data now are scalable from 0 to 1, where 0 is the minimum value and 1 is the maximum value.

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	0.527778	0.083333	0.593220	0.583333	Iris-versicolor
1	2	0.194444	0.625000	0.050847	0.083333	Iris-setosa
2	3	0.555556	0.541667	0.627119	0.625000	Iris-versicolor
3	4	0.500000	0.375000	0.627119	0.541667	Iris-versicolor
4	5	0.805556	0.416667	0.813559	0.625000	Iris-virginica
..	...	...	...	...	...	...
145	146	0.388889	1.000000	0.084746	0.125000	Iris-setosa
146	147	0.138889	0.458333	0.101695	0.041667	Iris-setosa
147	148	0.388889	0.333333	0.525424	0.500000	Iris-versicolor
148	149	0.944444	0.333333	0.966102	0.791667	Iris-virginica
149	150	0.666667	0.458333	0.779661	0.958333	Iris-virginica

Fig 6. Data Normalization

Finally, I have split the data into training and testing set. Testing data consists of the Iris species that are not classified, while training data consists of the Iris species that are already classified.

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
38	39	0.750000	0.500000	0.627119	0.541667	NaN
53	54	0.555556	0.541667	0.847458	1.000000	NaN
95	96	0.222222	0.625000	0.067797	0.041667	NaN

Fig 7. Test Set

17. First step:

- Mapper – takes as parameter key: None and as parameter value: line but in this case I have not used it, since I read the dataset with Panda's library.

  This mapper calculates the Euclidian distance by iterating over the test and train data.
  Returns (test data id, train data id, train data species) as key and Euclidian distance as value.

```
[54, 146, "Iris-setosa"]        1.2590478983458446
[54, 147, "Iris-setosa"]        1.2865148097634507
[54, 148, "Iris-versicolor"]    0.6518330976662087
[54, 149, "Iris-virginica"]     0.5021120774091343
[54, 150, "Iris-virginica"]     0.1600706560190694
[96, 1, "Iris-versicolor"]      0.977875169889677
```

Fig 8. Mapper Output

- Reducer - takes as parameter keys and values coming from the mapper. It removes data repetitions and returns them in a different format.
  Yield: test data id as key, (train data id, train data species, Euclidian distance) as value.

```
96      [91, "Iris-setosa", [0.11239640912524881]]
96      [92, "Iris-versicolor", [0.9118950363705373]]
96      [93, "Iris-versicolor", [0.8604352956416256]]
96      [94, "Iris-virginica", [1.1672106168146086]]
96      [95, "Iris-virginica", [1.2082532572435958]]
96      [97, "Iris-setosa", [0.12951484504727034]]
96      [98, "Iris-virginica", [1.198616106814525 4]]
96      [99, "Iris-setosa", [0.08784104611578843]]
```

Fig 9. Reducer Output

18. Second step:

- Reducer - takes as parameter key and value coming from the first step. It sorts the data according to the distance in ascending order and gets top 15 neighbours, since it was specified in the task that K=15.
  Yield: test data id as key and (train data id, train data species, Euclidian distance) as value.

```
39      [[20, "Iris-versicolor", [0.07343651849122701]], [92, "Iris-versicolor", [0.10206207261596571]],
         [55, "Iris-versicolor", [0.10614150697393161]], [135, "Iris-versicolor", [0.14790398035188826]],
         [93, "Iris-versicolor", [0.17318794385107014]], [40, "Iris-versicolor", [0.17510849201711304]],
         [57, "Iris-versicolor", [0.1764977940450499]],  [6, "Iris-versicolor", [0.17916292260406583]],
         [3, "Iris-versicolor", [0.21561353744805584]],  [133, "Iris-versicolor", [0.22156553458871633]],
         [24, "Iris-versicolor", [0.22301405614750575]], [5, "Iris-virginica", [0.22745424824002503]],
         [127, "Iris-versicolor", [0.2640676394720663]], [12, "Iris-versicolor", [0.26709587700106446]],
         [71, "Iris-virginica", [0.2681770147023632]]]

54      [[95, "Iris-virginica", [0.08982540048725768]], [17, "Iris-virginica", [0.12219305610175774]],
         [67, "Iris-virginica", [0.14069120897338847]], [63, "Iris-virginica", [0.15339027125979598]],
         [150, "Iris-virginica", [0.1600706560190694]], [78, "Iris-virginica", [0.1681010195812336]],
         [77, "Iris-virginica", [0.1883759932706523]],  [53, "Iris-virginica", [0.1975950335420275]],
         [94, "Iris-virginica", [0.20666136730716034]], [36, "Iris-virginica", [0.23086932420459433]],
         [83, "Iris-virginica", [0.25376320274327274]], [98, "Iris-virginica", [0.25482512080040565]],
         [25, "Iris-virginica", [0.26433831642836686]], [75, "Iris-virginica", [0.26738452491916936]],
         [124, "Iris-virginica", [0.2698923040558335]]]

96      [[88, "Iris-setosa", [0.03254041656427031]], [61, "Iris-setosa", [0.04166666666666666]],
         [82, "Iris-setosa", [0.04498205067597233]], [32, "Iris-setosa", [0.05007710104811097]],
         [131, "Iris-setosa", [0.05007710104811109]], [2, "Iris-setosa", [0.052867663285673355]],
         [117, "Iris-setosa", [0.05286766328567343]], [99, "Iris-setosa", [0.08784104611578843]],
         [15, "Iris-setosa", [0.09914459462408452]], [64, "Iris-setosa", [0.10157823570743627]],
         [144, "Iris-setosa", [0.10296239887184866]], [45, "Iris-setosa", [0.10614150697393179]],
         [91, "Iris-setosa", [0.11239640912524881]], [74, "Iris-setosa", [0.11906369161416717]],
         [122, "Iris-setosa", [0.11906369161416727]]]
```

Reducer Output

19. Third step:

- Reducer - takes as parameter key and value coming from the second step. It classifies the unknown species by finding the mode of the neighbours list.

  I have iterated over the list of neighbours, and I have counted which of the species type is repeated the most. Therefore, the feature with the highest count number will be the classification for the unknown data. Yield: test data id, classification.

```
39      "Iris-versicolor"
54      "Iris-virginica"
96      "Iris-setosa"
```

Fig 10. Reducer Output

Task 4

20. Dataset

For this task I have worked with a matrix dataset, which in the project folder has the name A.txt. This dataset consists of 1000 rows and 50 columns.

21. Task description

As specified in the project description, the goal of this task is te calculate the Frobenius norm of the matrix with the given formula:

$$||A||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

We are required to calculate the sum of the squares of the elements on the same row in one reducer and calculate the Frobenius norm in another reducer.

22. Dependencies

- MRJob and MRStep – Python library for writing MapReduce.
- Math – built in function to perform mathematical operations.

23. Run project command

- Python main.py A.txt

24. Solution

I have created a python class, FrobeniusNorm that imports MRJob library as a parameter. Inside this class are written all the map and reduce functions needed for this task.

25. First step:

- Mapper - takes as parameter key: None and as parameter value: each line of the dataset.

  Each of the data is separated by an empty space. Therefore, I have split the line into empty spaces in order to access each element.
  Yield: row items, None.

- Reducer- takes as parameters key: row items coming from the mapper and as value: none. Sums the absolute value of all elements in a row, in power of 2.
  Yield: None, row sum

```
null    13.44063745405871
null    15.185259170691058
null    13.750650985970926
null    15.30780369166359
null    19.018042645067226
null    16.43637226069459
null    15.695343113261298
null    21.702181054296545
null    15.534048925223665
null    16.205957425147073
null    15.787004912093936
```

Fig 11. Reducer Output

26. Second step:

- Reducer – takes as parameter key: none and as a value: sum of the absolute values of all elements in a row, in power of 2.
  Returns: 'Forbenius Norm is: ', Frobenius Norm.

```
"Frobenius Norm is: "   128.4506152933252
```

Fig 12. Reducer Output

References

- A. (2020, November 25). *K-Nearest Neighbors Algorithm Using Python*. Edureka. Retrieved November 14, 2021, from https://www.edureka.co/blog/k-nearest-neighbors-algorithm/
- *Getting started — pandas 1.3.4 documentation*. (n.d.). Pandas. Retrieved November 14, 2021, from https://pandas.pydata.org/docs/getting_started/index.html
- *Hadoop - MapReduce*. (n.d.). TutorialsPoint. Retrieved November 14, 2021, from https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm
- *mrjob — mrjob v0.7.4 documentation*. (n.d.). MRJob. Retrieved November 14, 2021, from https://mrjob.readthedocs.io/en/latest/index.html
- Narula, M. (n.d.). *Calculate Euclidean Distance in Python*. Delft Stack. Retrieved November 14, 2021, from https://www.delftstack.com/howto/numpy/calculate-euclidean-distance/
- *NLTK :: Natural Language Toolkit.* (n.d.). NLTK. Retrieved November 14, 2021, from https://www.nltk.org/
- *numpy.linalg.norm — NumPy v1.21 Manual.* (n.d.). NumPy. Retrieved November 14, 2021, from https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html
- *re — Regular expression operations — Python 3.10.0 documentation*. (n.d.). Python.Org. Retrieved November 14, 2021, from https://docs.python.org/3/library/re.html
- *simplejson — JSON encoder and decoder — simplejson 3.17.5 documentation*. (n.d.). Simplejson. Retrieved November 14, 2021, from https://simplejson.readthedocs.io/en/latest/
- *sklearn.preprocessing.normalize*. (n.d.). Scikit-Learn. Retrieved November 14, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html