# 

# Artificial Intelligence and Data Engineering - Cloud Computing

IMD Rating (Bloom Filter):  
Hadoop, Spark implementation

# *One Ring to rule them all, One Ring to find them, One Ring to bring them all and in the darkness bind them.*

The Lord of the Rings - **J. R. R. Tolkien**

# **Frodo** *group: Simone* ***Bensi*** *Paolo* ***Falsini*** *Tommaso* ***Nocchi***

# IMDb database

The file title.ratings.tsv contains more than one million records, with the following fields:

* tconst **string**: alphanumeric unique identifier of the title.
* averageRating **double**: weighted average of all the individual user ratings.
* numVotes **int**: number of votes the title has received.

We have 10 rating values, and we must compute 10 different bloom filter with the same false positive rate. In both Hadoop and Spark implementation we first build the Bloom filter, then we have a further step to test the resulting false positive rate.

# Bloom filter

A bloom filter can be seen as a classifier used to predict if an object is a member of a set. Given an element, the classifier prediction can be:

* 1 the element belongs to the set.
* 0 the element doesn’t belong to the set.

The bloom filter classifier is trained adding the object which belong to the set, after that phase the bloom filter can be used as a classifier. The bloom filter prediction will never consist in a false negative. To design a bloom filter for the titles rating (with a given false positive rate 𝑝), we need to estimate the number of keys 𝑛 to be added to the bloom filter.

# Bloom Filter Java class

The Bloom Filter corresponds to the java class BloomFilter (internally, each bit array is implemented using an array of WORD (32 bits)). The class exports the following methods:

* public BloomFilter (int n\_itemNumber, double p\_falsePositiveRate)  
  creates an empty object to store n\_itemNumber items and manage a false positive rate equal to p\_falsePositiveRate
* public vou addItem(Text item)  
  add the item item to the BitArray
* public boolean checkItem(Text item)   
  check if the title item in in the BitArray
* public String getString()  
  return a string representing the BitArray[[1]](#footnote-1)
* public void setString(String integerList)  
  initialize the Bloom Filter

# Hadoop Map-Reduce

In the Hadoop implementation we use three Map–Reduce jobs, coded in a single class IMDRating. The jar command is reported:

hadoop jar target/IMDRating-1.0-SNAPSHOT.jar IMDRating 0.01 data.tsv count bloom check

The requested false positive rate is passed as the first parameter. The job configuration is used to make this value available in all the Map-Reduce stages.

# IMDRating COUNT stage

To define the size of each Bloom Filter we must know the number of titles for each average rating, we design a Map-Reduce stage just for this. The output file of this stage is a list of ten pairs (average rate, number of titles). At the end of this stage, this file is read, and those pairs are stored in the job configuration to be used in the two subsequent stages.

**IMDRating COUNT**

Procedure MAP (Text **t**, Average **a**, numVotes **v**)  
**ra** = round(**a**)  
EMIT (**ra** (as Text), **i = 1** (as Int))

Procedure REDUCER (ra a, Links [i1,i2, …im])  
**for** **all** n **in** Links **do**:  
 *count += 1*  
EMIT(**a** (as Text), **count** (as Int))

## IMDRating POPULATE stage

For every possible rating we build the Bloom Filter of the appropriate size, then we add all the selected titles to the Bloom filter. The output file of this stage is a list of 10 pairs (*average rating*, *string representing the Bloom Filter*).

**IMDRating POPULATE**

Procedure MAP (Text **t**, Average **a**, numVotes **v**)  
**ra** = round(**a**)  
EMIT (**ra** (as Text), **t** (as Text))

Procedure REDUCER (ra a, Links [t1,t2, …tm])  
*Bloom-Filter-Create* (n\_itemNumber,p\_falsePositiveRate)  
**for** **all** t **in** Links **do**:  
 *Bloom-Filter-AddItem*(**t**)  
EMIT(**a** (as Text), **Bloom Filter** (as Text))

## IMDRating CHECK stage

For every possible rating, we select the titles that don’t have that rating, we test them, and count the number of false positives. The ratio between the number of false positives and the tested titles gives the obtained false positive rate.

**IMDRating CHECK**

Procedure MAP (Text **t**, Average **a**, numVotes **v**)  
for I in range (10)  
 *Bloom-Filter-Create* (n\_itemNumber(I),p\_falsePositiveRate)  
 *Bloom-Filter-Initialize* (String ) **ra** = round(**a**)  
for I in range (10)  
 *if (I + 1 != ra && Bloom-Filter-CheckItem*(ra, **t**)  
 EMIT(**a** (as Text), **1**)

Procedure REDUCER (ra a, Links [t1,t2, …tm])  
**for** **all** n **in** Links **do**:  
 **count** *+= 1*

**fpr** = **count** */* n\_itemNumber(ra)  
EMIT(**a** (as Text), **fpr** (as Double))

# Spark Pyspark

The Bloom Filter corresponds to the Python class BloomFilter (internally each bit array is implemented using an array of Boolean). The class exports the following methods:

* **BloomFilter(items\_count, fp\_prob)**creates an empty object to store count titles and to manage a false positive rate equal to fpr
* **public addItem(t)**add the title t to the BitArray
* **public boolean checkItem(t)**check if the title t in in the BitArray

The spark execution is obtained with the following command:

**spark-submit IMDRating.py 0.01 data.tsv**

Reading the file from the Hadoop distributed file system:

**file = sc.textFile(sys.argv[2])  
temp = file.map(lambda x: (x.split("\t")))  
titles = temp.map(lambda x: (x[0],java\_round(float(x[1]))))**

Initializing the fpr parameter:

**fpr = float(sys.argv[1])**

Creating the Bloom Filters and populate them:

**bloom\_filter = [0 for \_ in range(10)]**

**for rating in range(10):**

**titles\_collected = titles.filter(lambda x : x[1] == rating + 1).collect()**

**bloom\_filter[rating] = BloomFilter(len(titles\_collected), fpr)**

**for title in range(0, len(titles\_collected)):**

**bloom\_filter[rating].add(titles\_collected[title][0])**

**print("{0:02d} {1:12d}".format(rating + 1, len(titles\_collected)))**

Testing the titles and getting the results:

**for rating in range(10):**

**titles\_filtered = titles.filter(lambda x : x[1] != rating + 1)**

**titles\_fp = titles\_filtered.filter(lambda x: bloom\_filter[rating].check(x[0]) > 0)**

**print("{0:02d} {1:12.10f}"**

**.format(rating + 1, titles\_fp.count()/titles\_filtered.count()))**

1. The string representing the Bloom Filter is obtained converting each internal word into a string, then a ; is used as separator. [↑](#footnote-ref-1)