Society evolution through Wikipedia's analysis *

Analysis of how society changed during year taking as example Machine Learning

Maria Pia Natale
189160
mariapia.natale@studenti.unitn.it

Alessia Tovo
189192
alessia.tovo@studenti.unitn.it

ABSTRACT

Wikipedia is the largest free encyclopedia whose number of contents grows and is updated continuously. Its expansion may be related to the evolution of our epoch. Articles in Wikipedia are tagged with one or more `Categories` that specify the topics in the page. Depending on the growth of the topic in real life, the number of articles of each category may be different. Indeed, researches, innovation, manufacturing of new products lead to the expansion of some scientific, cultural, artistic fields more than others, like religion, ancient history, ancient literature that can be defined as stuck. As a result, the number of articles per Wikipedia's category changes differently according to these factors.
This paper describes the analysis of Wikipedia's articles regarding a specific category, namely `Category:Machine Learning`, which, nowadays, is one of the most expanding topic in real life. Our purpose is to analyze how fast this category increases during years and also the categories related to it.

1. INTRODUCTION

Nowadays, innovation and evolution are the driving force of our society. The number of academic researches, industrial new products, technologic innovations is more and more large. Not only these fields are subject to the society growth but also music, motion picture industry, sporting events and more others. It is not possible to apply the same "philosophy" to those fields where the knowledge has reached a stuck point. The subjects affected by this saturation are the Ancient literature, Religion, Ancient history, biography of historical figures and others similar. All these topics are included into Wikipedia, which is the first and the largest online free encyclopedia that everyone can edit. It was born in 2001 and, nowadays, has 5,650,187 articles in English. There are other 44 different languages and the number of articles depends on the language. Wikipedia's content are divided into Portals and each portal contains different Categories. For each category there are articles that may belong to other categories.
The number of articles grew during years and some categories are grown more faster than other. Researches in different field, like technology, medical science, physics and more others, lead to the growth of the number of Wikipedia articles concerning that fields. Also categories like Society, Art, Music and Movies are affected by the continuous growth of products. Understanding how Wikipedia's contents grown during years, which are the main categories affected by innovations and discoveries and how fast they changed may help to comprehend which type of advancement our society is taking.
The following document explains the work done for the Big Data and Social Networks course in the University of Trento. The aim of this project is to parse Wikipedia datasets in order to create graphs to analyze how pages were modified during the time and understand what is its advancement. This may help to understand which are the topic in real life that are continuously growing and, vice versa, which are stacked.
The dataset used concerns only a single Category, namely Machine Learning. This decision comes from two main reasons: firstly, the computational power of the available devices was limited; secondly machine learning is a new science in a continuous expansion, so it is an adequate example for our study. In order to obtain historical information the Export pages service of Wikipedia has been used which allows us to download the last 1000 revision of all pages regarding the Machine Learning category. The dataset downloaded is in XML format and it has been pre-processed using the `databricks package` that helped us to easily read the dataset. Before generating the graphs, the dataset has been read and split in order to keep only the useful information for our purpose. The "summarized" dataset has then been saved into Json files in order to simplify the second stage of the project. From the XML dataset a list of directories has been created, one per year found in the files, and into these were put all the information of all the pages for that specific year; each page is a directory that contains the summarized dataset regarding it. This part of the work has been done using a MapReduce approach implementing functions in Apache Spark to cluster-compute the operations and Hadoop HDFS in order to save the result of this process.
After this first stage, Cypher query language has been used to create all nodes and edges and Neo4j database to store the generated graphs. From the dataset, different graphs were generated, one for each year, and were used three different

*(Big data and Social network project AY: 2017/2018)

approaches to create three different type of graph. One approach considers categories as edge between nodes, therefore the pages that belong to the same category are connected to the others. In the second and third graph, an edge were created between pages and each page-link in its text. In particular, the second graph's weight is computed by common words and their occurrences in the texts of the two pages connected. While, the third graph's weight is the number of common links of the two pages connected by the edge.
The connection between HDFS and Neo4j has been done with a Java application. A MapReduce approach were used to obtain useful information from the dataset and compute different weights. In order to extract statistics from graphs, Cypher query language has been used and the results were plotted into graphics to better visualize them.

2. RELATED WORK

The pre-processing phase of the dataset, the creation of the graphs and their analysis were done using different technologies.
The entire project has been built on top of Apache Spark framework, several MapReduce functions have been developed to run the most complex sections of the code, and HDFS has been used to store the results of the pre-processing phase.
The entire code is written in Java with the aid of external packages to read the dataset. Indeed the histories of revisions of Wikipedia's pages originally were in XML format and their contents were read as in a csv file employing the `databricks` package.
Once read the entire dataset, various MapReduce functions have been applied in order to extract only those fields useful for our analysis and for building a new dataset, that will be saved into HDFS.
After that, the apposite queries to create nodes and edges were executed using Cypher, a query language originally made by Neo4j. Then, we store the resulting graphs on the graph database, Neo4j. Other Cypher queries have been applied to extract useful and interesting information from database.
In the following sections, it is possible to better understand which are the technologies used and for which purpose.

2.1 Databricks

Databricks [1] is an external package used to read the XML Wikipedia's history pages. This package has the name of the company that developed it, which was founded by the creators of Apache Spark. Databricks package is available for different programming language, such as Scala, Pyhton and Java. Databricks helped us to read the XML pages like a CSV or JSON format. Moreover, this package has been built specifically for the Apace Spark framework that we used to develop the project.

2.2 Map Reduce

Hadoop MapReduce [2] is a "software framework for easily writing applications which process vast amounts of data [...] in-parallel on large clusters [...] of commodity hardware in a reliable, fault-tolerant manner."
A job in MapReduce is computed using a cluster of computers, divided into Master node and Slave nodes. The job divides the dataset into chunks of the same size and the Master node assigns each chunk to a different Slave node whose performs the required operations on its part of dataset. This phase is called *Map function*. Then, the Master performs the *Shuffle* phase that consists of reordering the output of Slave node which becomes the input of the second phase, the *Reduce function* where other operations are performed by the Slave nodes. Then, each output is collected by the Master node that puts together the single results and stores the final output into a file-system.
For this project, the Spark MapReduce framework has been used which exactly operates as the Hadoop version, but with better performance and more scalability. MapReduce functions were useful to retrieve fields from the dataset, to process the text of the reviews in order to obtain the links and the categories of the page. Moreover, MapReduce has been used to delete the "Stop words" from the text in order to compute a more precise wordcount for each single page, subsequently. To create the final Dataset with tuples composed by title, year, links, wordcount and categories, another MapReduce function has been used and the result of it were saved into HDFS in Json format.

2.3 Neo4j and Cypher

2.3.1 Neo4j

Noe4j [3] is a graph database created by Neo4j, Inc. and, as many others databases, is No-Sql. Neo4j is developed in Java but it is possible to connect it with other programming languages. Since it is a graph database, elements stored are called nodes, edges and attribute. The index-free adjacency of the storage structure of Neo4j allows fast transactions and processing of data. Moreover, the graph's processing ensures zero latency and realtime performances. Neo4j implements ACID protocol that ensures consistency and that data with relationships are natively stored. [4]. Our Java application is connected to Neo4j database through the Bolt protocol that allows a fast and secure connection to our local database. Each Wikiepedia's page is represented as a node with some attributes, and nodes are connected with relationships that may represent physical links between pages or the belonging on the same category, better explained in the 4.3 section.

2.3.2 Cypher

To create nodes and relationships and queries on the resulting graphs, Cypher query language has been used. Cypher is a "a native graph query language that provides the most efficient and expressive way to describe relationship queries". Cypher is a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax. [5]. Nodes are represented with parenthesis, relationships with arrows and both nodes and relationship may have labels and attributes. Cypher has been used to create nodes and relationship, and after this stage, to query the resulting graphs in order to obtain useful information, such as the number of nodes for each page, the most related pages in term of wordcount or link similarity for each year. The 4.4 section will explain how the queries have been done.

[1] https://github.com/databricks/spark-xml
[2] https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
[3] https://neo4j.com
[4] https://neo4j.com/product/
[5] https://neo4j.com/developer/cypher-query-language/

2.4 Google Cloud Platform

In order to execute our code, Google Cloud Platform [6] services have been used, in particular the Dataproc cluster service, which already has Hadoop and Spark framerworks installed. Since the project has been coded on top of Spark, using HDFS as file-system, it was easy and immediate the connection between the code and the Google Platform. The created cluster consists of 1 Master node and 3 Slave nodes. The Master node has 1 vCPU with 3.75 GB of memory, whereas each Slave node has 2 VCPU with 10.0 GB of memory. To this cluster were submitted three different jobs, with same code but different input size, in order to evaluate the scalability of the code. The first job were executed with 1/3 of the original dataset as input, the second job with 2/3 of the original dataset as input and the third job with the whole dataset. The performance were satisfying since the first job required 1 hour and 15 minutes to process 1.1 GB of dataset; the second job took 2 hours and 15 minutes for 1.8 GB of data and the final job taken 2 hours and 53 minutes to process 2.4 GB of data, that is the "complete" dataset.
Performances could be better with a more powerful cluster but unfortunately the cluster made up is the most powerful cluster that Google allows to built with a "free account". The Master has been made less powerful than the slaves since the most important part of the work is made by these last ones.

3. PROBLEM STATEMENT

The evolution of society and knowledge of what happens may be better represented with Web contents, in particular the Wikipedia encyclopedia. What happens in the world, whether it is a musical, theatrical, cinematographic event rather than a political, economic one, could be easily found as a content of a Wikipedia page. With the term "event" we want to include scientific discoveries, new technological products, periodic events, such as Olympic Games or Academy Awards Oscar, the introduction into the society of a famous person that could be a singer, an inventor or a politician. Everything that surrounds us and everything that happens in our society and happened in human's past can be found as a Wikipedia's page content.
Since Wikipedia is the best representation of our society and human knowledge, to analyze its contents means to analyze the whole society and what we, as human, know about it. Our goal is to understand how the society's knowledge changed during years and what are the main topics and fields that increases as number of new "products". The idea is to extend our project to the entire Wikipedia history, but for computational reasons, we took only a small part of Wikipedia contents, namely the Machine Learning category. The reason we took it as category to analyze is that machine learning is a "new" science in continuous growth.
The term "machine learning" was created in 1959 and, nowadays, it is one of the field of computer science most studied and on which a big number of researches have been made. Starting from the history of revisions of all the pages of Wikipedia tagged with Machine Learning as category, we were able to extract information of 17 years, from 2001 to the last revisions of 2018. Revision contents allow us to understand how much a certain argument is known and which is its improvement during years, how much an argument becomes connected to similar arguments of the same or different category, if an argument is in its own right or can be defined as key topic for that field.
Extending our code to the entire Wikipedia history dataset may give an idea of which are the main fields that are continuous growing in term of knowledge and vice versa which are those fields that are stuck.

[6]https://cloud.google.com/?hl=it

4. SOLUTION OF THE PROBLEM

In order to solve the problem described in Section 3, that is understand which are the fields of society that are more subjected to change and growth, the Wikipedia dataset regarding the category Machine Learning has been used. The dataset containing the revisions history has been taken since we needed historical information of the pages and not only the current version. By searching Machine Learning in the toolbar of the Wikipedia:Special:Export, it is possibile to download the last 1000 revisions of all the pages related to Machine Learning category. Since Machine Learning includes others categories, also pages of those categories were downloaded. This allowed us to collect revisions for almost 800 pages and 17 years. Pre-processing the entire dataset helped us to collect only those information necessary to our study, such as the year of the revision, the timestamp in order to obtain the last revision for each year, and the text of the page. Parsing the last field, we obtained all the links of the page to others Wikipedia's pages and the categories the page belong to. Then, the extrapolated information were saved into a new dataset and, from this, were created many graphs, one for each identified year. Moreover, were created 3 different type of graph where the type of relationship and, consequently, the weights vary.

1. **Link Graph:** a graph that has a node for each page found in the dataset and a directed edge if a page has a link to the other page. The weights of the edges are computed as the number of common links between the two pages.

2. **Wordcount Graph:** a graph that has nodes and edges exactly as the previous type of graph but the weights of edges are different. Indeed the `(Min number of wordcount)/(Max number of wordcount)` is used as weight. This mathematical computation is better explained in Section 4.3.

3. **Category Graph:** this third graph has a node for each page of the dataset and an undirected and weightless edge between two nodes is created only if those pages belongs to the same category.

Creating three different graphs allowed us to keep these information separated and easily perform the queries. From the first graph, is possible to understand which are the main pages of the dataset, which have a key role into their categories and, as a result, which is the main topic. From the second graph is possible to identify those pages that describe similar content so they are highly related to each other. The third graph permits to understand which are those categories that are more "populated" and have a biggest growth during years, so which are the key topic of the Machine Learning science. All these information were obtained executing some Cypher queries on the graphs saved into the

Neo4j database. The detail of the queries are explained in Section 4.4.

4.1 Dataset

The used dataset has a size of 2.4 GB and has been obtained from the page Wikipedia:Special:Export [7] that allows to download the last 1000 revisions of the pages that belong to a certain category - Machine Learning in our case. Writing Machine Learning in the text field is possible to obtain all the pages and sub-categories that belong to Machine Learning. Therefore, in order to have a larger dataset, the reviews of the pages from the sub-categories of Machine Learning were added. The total number of categories taken is 32 and the number of pages depends of the year. For example, in 2017 we have data for 774 pages, while for 2018 we have only 365 pages. Worst for the 2001 that include data for only 5 pages. This is a result of the limit of 1000 revisions per page, so revisions of those pages that were largely updated during the latest years are not included into the download. The developed code can be run over the entire dataset of the Wikipedia's history since it not depends strictly on the size of the dataset, as can be found in Section 5.1. The dataset is composed by 32 XML files whose size depends on the contents. The smallest file is 6.0 kB and is about the Machine Learning Portal category, while the largest file is Machine Learning with a size of 366.2 MB.
An example of how is the structure of the file can be found below.

```
<page>
   <title>Log-linear model</title>
   <ns>0</ns>
   <id>15689191</id>
   <revision>
      <id>190439425</id>
      <timestamp>2008-02-10T19:27:32Z</timestamp>
      <contributor>
         <username>Qwfp</username>
         <id>6032993</id>
      </contributor>
      <comment>{comment to the revision}</comment>
      <model>wikitext</model>
      <format>text/x-wiki</format>
      <text ...>{text of the page}</text>
      <sha1>...</sha1>
   </revision>
    <revision>
      [...]
    </revision>
</page>
<page>
   [...]
</page>
```

From this structure has been extracted for each `<page>` the `<title>`, the `<timestamp>` of the revision and the `<text>` of the revision. The content of the `<text>` has then been processed in order to obtain useful information about the page, as described in the following section.

4.2 Data pre-processing

In order to divide data year by year for creating the different graphs, a pre-processing stage has been applied to the dataset. The goal of our pre-processing phase is to have a list of directories named with all the years founded in all the revisions and inside these folders, a list of sub-directories named as the page that has been modified in that year. Into these last directories, there is the Json file containing the information for that page in that year.
Starting from the XML files, the whole list of years has been computed and for each year a directory into the HDFS has been created. How the list of years is collected is shown below.

```
List<String> years = new ArrayList<>();
years = final_df.select(col("Year")).dropDuplicates()
      .sort().as(Encoders.STRING()).collectAsList();
```

From the `final_df`, that is the complete dataset, the column "Year" has been extracted and all the duplicates removed. Then, for each page, the last revision for each year has been taken since it is more plausible to think that the last revision of the year was the most complete one. As shown below, the last revision has been taken through the "max" function of SparkSQL applied on the timestamps. The obtained Dataset, composed by the columns "Year", "Title", "Timestamp", has been joined to the "original" one selecting only the column "Text".

```
Dataset<Row> grouped = final_df.select("title", "Year",
     "Timestamp").groupBy(col("title"),
     col("Year")).agg(functions.max("timestamp")
   .as("Timestamp"));
Dataset<Row> joined = final_df.select(col("text"),
     col("Timestamp"), col("ArrayText")).join(grouped,
     "Timestamp");
```

From the text , the links from the page to other pages have been extracted identifying the link content written between double brackets. The content between these two brackets has been processed since in Wikipedia's pages the link to the page may appear with a different name. So the first element between the brackets identifies the name of the page linked to, while the second element represents the name of the link that appears in the text. The list of the link for each revision is without duplicates, due to the fact that the link information is necessary to create the edge into the graphs. The code below shows the Map function implemented to extract the links from the text and the function `getLinks()` is shown in the Algorithm 1

```
JavaPairRDD<HashSet<String>, Long> links =
     texts.map(text -> getLinks(text)).zipWithIndex();
```

Algorithm 1: Function getLinks(String text)

```
links ← newHashset <>
pattern ← [[(.*?)]]
m ← r.mathcer(text)
while m.find() do
word ← m.group(1)
parsing of word in order to obtain the link
links.add(linkLowerCase)
end while
```

In a similar way, the categories that identify the page have been extracted, since categories are identified with the string "Category:" inside double brackets, the same as for the links. The wordcount of the text has been computed after the removal of the "Stop words" from the text. Following,

[7] https://en.wikipedia.org/wiki/Special:Export

the MapReduce functions to compute the Wordcount of the text is shown, where the `all_word_text` variable contains all the word of the text without the "Stop words"

```
List<JavaPairRDD<String, Integer>> wordcount_text_list
    = new ArrayList<>();
for (JavaRDD<String> t : all_word_text) {
   JavaPairRDD<String, Integer> counts;
    counts = t.flatMap(s ->
         Arrays.asList(s.toString().split("
         ")).iterator()).mapToPair(word -> new
         Tuple2<>(word, 1)).reduceByKey((a, b) -> a +
         b).filter(count -> count._1() != "");
    wordcount_text_list.add(counts);
}
```

These three new information, links, categories, wordcount, were added as columns to the newest dataset. Below, the structure of the new dataset, built as Java class, through the used of `Encoders`, is shown

```
public class RowWikipedia implements Serializable {
    private String Title;
    private String Year;
    private HashSet<String> Links;
    private List<String> Wordcount;
    private List<String> Categories;

    /* getter and setter*/
}
```

Then, the dataset was saved in Json format into the directory named as the `<title>` tag and located inside the directory of the year of the revision. In the code block below it is possible to find the Java code used to implement what has just been described.

```
JavaRDD<Integer> indexes =
     sc.parallelize(IntStream.range(0,wordcount_list.size()).
      boxed().collect(Collectors.toList()));
List<RowWikipedia> tmp = indexes.map(index ->
     setRow(index, titles, k, links_f, cat_f,
     wordcount_list)).collect();
```

The *indexes* structure has been built in order to apply the Map function to all the lists containing dataset information computed and be sure that all the fields are from the same Row, since we have to put them into a new data structure. The Algorithm 2 shows the Map function that creates the rows of the new dataset.

As a final result, into the HDFS, there is the list of directories for each year and in each of these is contained a number of directories for the pages that contain the dataset related to that page in that year.

4.3 Graphs creation

Once the initial dataset has been processed and all the folders created, the graphs have been created. As briefly mentioned in previous section, three different type of graphs have been created.

1. **Link Graph.** Each node of the graphs is a Wikipedia's page and an edge from node A to node B is created if and only if there is a link from page A to page B. The edges are directed and weighted. The weight is computed counting the number of common links between page A and page B, so effectively the number of common neighbors.

2. **Wordcount Graph.** Each node of the graph is a Wikipedia's page and an edge from node A to node B is created if and only if there is a link from page A to page B . The edges are directed and weighted. The weight is computed as the formula below,

$$\sum_{w_i} \frac{min(f_{w_i})}{max(f_{w_i})} \tag{1}$$

where $w_i \in W_{AB}$ and W_{AB} is $W_A \cap W_B$. W_A is the set of wordcount for page A, while W_B is the set of wordcount of page B; f_{w_i} is the frequency of the word w_i.

3. **Category Graph.** Each node of the graphs is a Wikipedia's page and an edge between two pages is created if the two pages belong to the same category. The edge is undirected and with no-weight.

For each file in the HDFS, the function *creationPrincipalNodes(file, sqlContext1, actual_year)*, shown in Algorithm 3, has been applied.

Algorithm 2: Function setRow(Integer index, List<String> titles, List(String) years, List(HashSet(String)) links, List(HashSet(String)) categories, List(List(String)) wordcounts) function

$newRow \leftarrow newRowWikipedia()$
$title\_row \leftarrow titles.get(index)$
$links\_row \leftarrow link.get(index)$
$categories\_row \leftarrow categories.get(index)$
$wordcounts\_row \leftarrow wordcounts.get(index)$
$year\_row \leftarrow years.get(index)$
$newRow.setTitle(title\_row)$;
$newRow.setYear(year\_row)$;
$newRow.setLinks(links\_row)$;
$newRow.setCategories(categories\_row)$;
$newRow.setWordcounts(wordcount\_row)$;

Algorithm 3: Function creationPrincipalNodes(String file, SQLContext sqlContext1, String actual_year)

$final\_df \leftarrow datasetfromfileinHDFS$
connection to Neo4j database
$createNodes(page, title, year, actual\_year)$

The function *createNodes(page, title, year, actual_year)* in Algorithm 3 execute the following Cypher query to create a node.

```
CREATE <page>+<year>:Page_year> {title: '<title>',
     year: '<year>'} RETURN <page>+<year>.title
```

Once all the nodes regarding the pages into HDFS have been created, the pages that are listed in link lists of the dataset are used to create new nodes, if the page does not already exist into the database. In this stage, also the links between the "principal" nodes and these pages were created, if and only if the "secondary" page is a link of the "principal" page.

In Algorithm 4 are defined the steps of this stage.

Algorithm 4: Function creationLinksAndPages(String file, SQLContext sqlContext1, String actual_year)

$final\_df \leftarrow dataset from file in HDFS$
connection to Neo4j database
if page is a directory in HDFS **then**
$list\_links \leftarrow page$
else
$createLinks(page, title, link, link_page, year, 0)$
end if
for page in list_links **do**
$weight \leftarrow computeLinkWeight(page, destLink)$
$createLinks(page, title, link, pageLink, year, weight)$
end for

When a page is not a directory in HDFS, in the *createLinks(...)* function, we put the weight to zero because we have no information about that page. Otherwise, the weight is computed depending on which type of graph we are creating. The *computeWeight(...)* function for the Link Graph type is written below and following the Cypher query to create the link is shown.

```
private static Integer computeWeight(String[] page,
     String[] destLink) {
    Integer weight = 0;
    for (String p : page){
        if (Arrays.asList(destLink).contains(p)){
            weight += 1;
        }
    }
    return weight;
}
```

```
MERGE <pageLink>+<year>:Page<year> {title: '<link>',
    year: 'year'}
WITH <pageLink>+<year>.title AS titleLink
MATCH (p:Page<year> {title: '<title>'}), (l:Page<year>
    {title: '<link>', year: '<year>'})
MERGE (p)-[link_to:LINK_TO_<year> {weight: '<weight>',
    name: 'has_link_to', from: 'title', to:
    'link'}]->(l)
RETURN link_to.name
```

The `MERGE` command allows us to obtain the node with the specified label and attributes and, if the node does not exists, to create it. Then, the `WITH` command permits to save the results of the previous command. The `MATCH` and `MERGE` commands firstly match those nodes to be linked each others and then create a directed edge between those two nodes. These queries and algorithms have been used only for the Link Graph and the Wordcount Graph, which are equal, but with different weights.
To create the Category Graph, a different approach has been used.
For each page in the dataset, a node has been created for each category the page belongs to, using the category and the year as univocal attributes of the node. The Algorithm 5 shows the procedure adopted.

In this algorithm, there is a function called *createNodes(...)* which executes the Cypher query below, where `category_label` is the concatenation of the String "category" and the String "year".

Algorithm 5: Function creationPrincipalNodes(String file, SQLContext sqlContext1)

$final\_df \leftarrow dataset from file in HDFS$
$connection to Neo4j database$
$categories\_dataset \leftarrow final\_df.select("Categories")$
$year \leftarrow final\_df.select("Year")$
for Category in categories_dataset **do**
$createNodes(page, title, year, category, category\_label)$
end for

```
CREATE (<page>+<category>+<year>:<category_label>
    {title: 'title', year: 'year'} RETURN
    <page>+<category>+<year>)
```

Once all the nodes are created for each category and year, a simple Cypher query has been adopted to connect those nodes that belong to the same category in the same year. The edges are undirected and weightless.

```
MATCH (p), (l)
WHERE p <> l and p.year = l.year and p.category =
    l.category
MERGE (p)-[link_to:LINK_TO {category: l.category, year:
    l.year}]-(l)
RETURN count(p)
```

4.4 Queries

In the following subsections are reported all the queries done for each type of graph and results are shown in section 5.2.

4.4.1 Link Graph

Query 1
This query returns the number of nodes for each year and has been applied to **Wordcount Graph** and **Link Graph**.

```
MATCH (n:Page<year>) RETURN count (n)
```

This query has been computed for each year, using the node's label `:Page<year>`. The query allows us to understand how many pages have been updated during years and how many new pages have been created.

Query 2
A similar query has been applied to obtain the number of all relatinships per year. In this way, we know how many links between Wikipedia's pages exist and how the number of these links changes during year.

```
MATCH p=()-[r:LINK_TO_<year>]->() RETURN p
```

Query 3
The following query returns the relationship with the maximum weight for each year. In particular, this relationship in the **Wordcount Graph** represents the nodes with the highest similarity value, while in the **Link Graph** identifies the nodes with the maximum number of common neighbors.

```
MATCH ()-[r:LINK_TO_<year>]-()
WHERE EXISTS(r.weight)
RETURN r.weight AS weight, r.from AS from, r.to AS to
ORDER BY r.weight DESC
```

Query 4
In order to obtain the node with the maximum number of incoming edges, this query has been applied. These results can be compared to the PageRank, an algorithm that permits to rank pages based on the number of links connected to them.

```
MATCH ()-[r:LINK_TO_<year>]->(n)
WITH n, count(r) as rel_cnt
RETURN n.title, rel_cnt
ORDER BY rel_cnt DESC
```

With this query is possible to understand which is the most linked node for each graph, namely the key topic of the graph.

Query 5
Similarly, the node with the maximum number of outgoing edges is retrieved.

```
MATCH ()<-[r:LINK_TO_<year>]-(n)
WITH n, count(r) as rel_cnt
RETURN n.title, rel_cnt
ORDER BY rel_cnt DESC
```

In **Query 3**, **Query 4** and **Query 5** we took only the first row of results, which represents the nodes with the highest value.

4.4.2 *Wordcount Graph*

For the Wordcount graph, the same queries of Link Graphs have been applied to obtain the same information. The only exception is the **Query 3**. Indeed, this particular query applied to the Wordcount Graph gives only information about the edge with the highest weight for each year. Therefore, the result is little bit different. Running that query is possible to know which are the couple of pages that have the highest similarity in the whole graph.
To identify the pages that have the highest number of common neighbors a different query has been used and it is reported below.

Query 6

```
MATCH (source)-->(neighbor)--(target)
WHERE NOT (source) = (target)
AND source.year = '<year>'
RETURN DISTINCT source.title AS source_id, target.title
    AS target_id, count(neighbor) AS common_neighbors
ORDER BY common_neighbors DESC
```

4.4.3 *Category graph*

Query 7
The first query applied to **Category Graph** is the one that returns the number of categories for each year. The query used is the one shown below.

```
MATCH (n) WHERE n.year = '<year>' RETURN DISTINCT
    n.category
```

Query 8
Other important information that can be obtained from this graph is, for each year, the category with more nodes and the result is given by the following query.

```
MATCH (n) WHERE n.year = '<year>'
RETURN DISTINCT COUNT(n.category) AS n_pages,
    n.category AS category, n.year AS year
ORDER BY n_pages DESC
```

From this query, only the first result has been taken since it is the maximum number of nodes for that category.

Query 9
The last query returns a list of category with the number of pages belonging to that category. In order to have a global vision on how the graphs change, we choose to analyze only 4 years with an interval of 4 years, 2005, 2009, 2013 and 2017, since 2017 is the most complete year. The described query is written below.

```
MATCH (n) WHERE n.year = '2005'
RETURN DISTINCT COUNT(n.category) AS n_pages,
    n.category AS category, n.year AS year
ORDER BY n_pages
```

The **Query 10** has been used to graphically generate the graph of the 2005 dataset, in order to show all the sub-graphs that compose the principal graph.

```
MATCH (n)-[r:LINK_TO]-(m) WHERE n.year = '2005' RETURN n
```

5. EXPERIMENTS

5.1 Different datasets

In order to test the scalability of our code, three different size of dataset has been analyzed.
The first sub-dataset, that could be called the "Test-dataset", consists of the 33% of the files in the dataset of 1.1 GB and consist of 10 XML files of different sizes ; the smallest is 7.0 MB while the largest is 308.4 MB. A simply alphabetic order has been used in order to take files in a random manner.
The second dataset has been created using the same criterion of the first one but with the 66% of the files. The final size of this second dataset is about 1.3 GB and the smallest file is 89.9 KB while the largest is 366.2 MB.
The third dataset is the complete dataset whose size is 2.4 GB and the queries were applied to it, since it is the most complete of the three. The computational results have been written in the subsection 2.4 since each sub-dataset is the input of each different Google Cloud Platform job.

5.2 Results

This subsection describes the results of the queries in subsection 4.4.
The result of **Query 1** is shown in Figure 1. As can be seen, the number of pages has an increase during years, except for

the year 2018 since the dataset for that year is not complete. The increasing tread is subject to two different causes: the first one is the effective increase of Wikipedia's pages during years and the second one is that the limit of 1000 revisions by Wikipedia's Special:Export may have cut the revisions of earliest years.

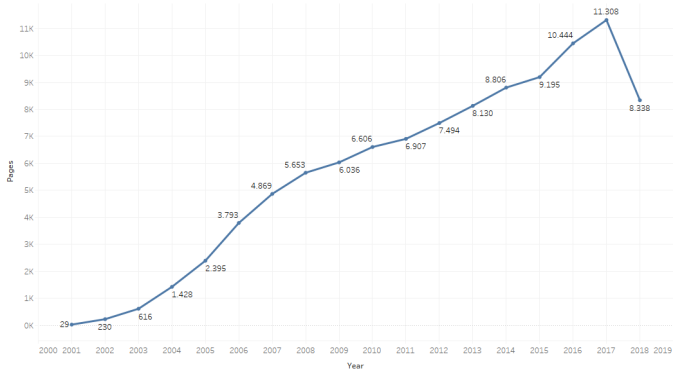

Figure 1: Number of pages during years

The Figure 2 shows the results of the second query that simply returns the number of edges per year, so the total links between Wikipedia's pages year by year.

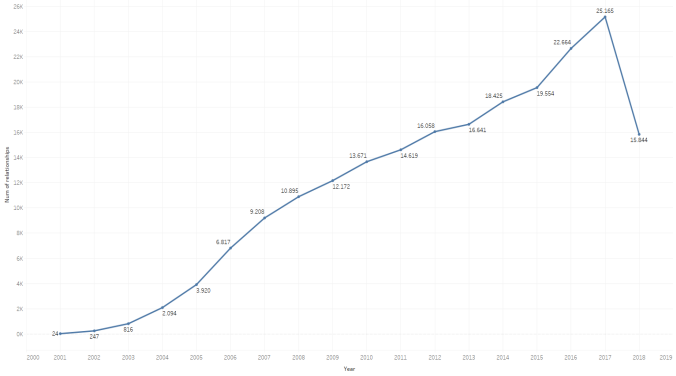

Figure 2: Number of links during years

The third query, that is the one that returns the pages that have the highest edge's weight per year, has been applied both to Link Graph and Wordcount Graph. The results obtained from the first one are reported in the Table 1, that indicate the similarity between the pages. As can be seen, the similarity between *artificial_neural_networks* and *genetic_algorithm* is the more stable in the whole dataset, even if the highest similarity is given by the two pages *deep_learning* and *convolutional_neural_network*.
Table 2 summarizes the results of the **Query 2** applied to the Link Graph, namely the highest Common neighbors value per year.
The results of **Query 4** and **Query 5** are shown in Figure 3 and Table 3 respectively. These queries have been applied both to Link Graph and Wordcount Graph. The results are the same since the structure of the two graphs are the same. The fourth query has been defined as the query that gives the PageRank of a page per year. Since the maximum PageRank for each year is about Machine Learning page, a graphical results of the increasing value of Machine Learning's page-rank is shown in Figure 3.

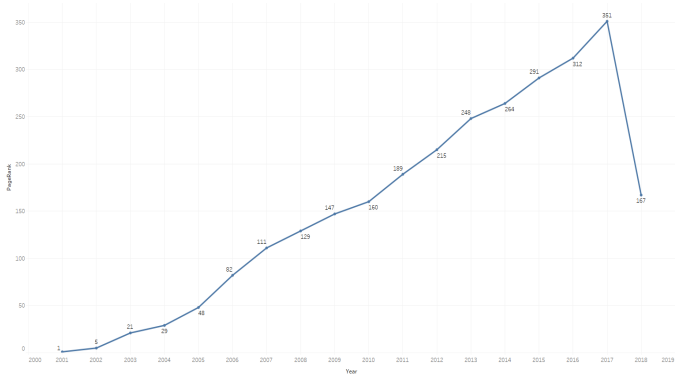

Figure 3: Result of query 4, namely PageRank of Machine Learning page

Also in this case, is possible to see a waning peak in 2018 because of the partial dataset for that year.

Year	Page	Outing edges
2001	genetic_programming	10
2002	supersived_learning	22
2003	artificial_neural_network	52
2004	genetic_algorithm	114
2005	artificial_neural_network	144
2006	artificial_neural_network	188
2007	genetic_algorithm	203
2008	ray_kurzweil	221
2009	genetic_algorithm	213
2010	cluster_analysis	182
2011	cluster_analysis	177
2012	time_series	212
2013	time_series	192
2014	time_series	217
2015	deep_learning	279
2016	glossary_of_artificial_intelligence	302
2017	outline_of_machine_learning	1068
2018	outline_of_machine_learning	1069

Table 1: Maximum number of outgoing edges from a node

The results of **Query 7** are shown in Figure 4. As for the other graphs, the growing trend and the waning peak are subject to the limited number of revisions.
The Figure 5 shows the results of the **Query 8** that returns the category with the highest number of nodes for each year. As can be seen, the main category is almost always Machine Learning. In the figure 6 are reported the number of nodes for each category for year 2005, 2009, 2013, 2017, obtained from **Query 9**, applied to different years.
It is possible to see that a huge number of categories has a single node. In Figure 7, the graph of year 2005 is plotted as sample. A similar graph exists for each year, except for 2001, 2002 and 2003 that have no information about categories. It is possible to identify a various number of subgraphs that represent different categories. The hugest categories are Machine Learning with 45 nodes, Statistics with

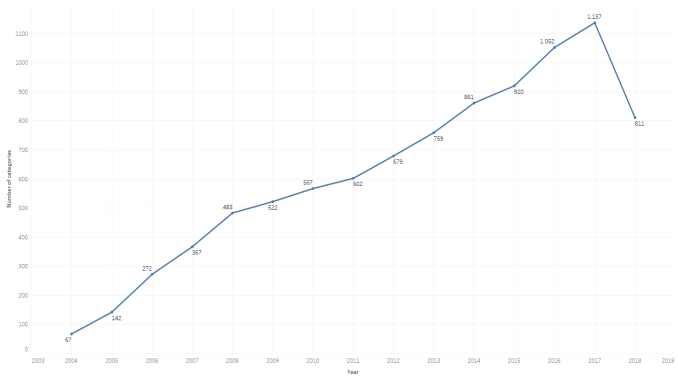

Figure 4: Result of query 7, the number of categories per year

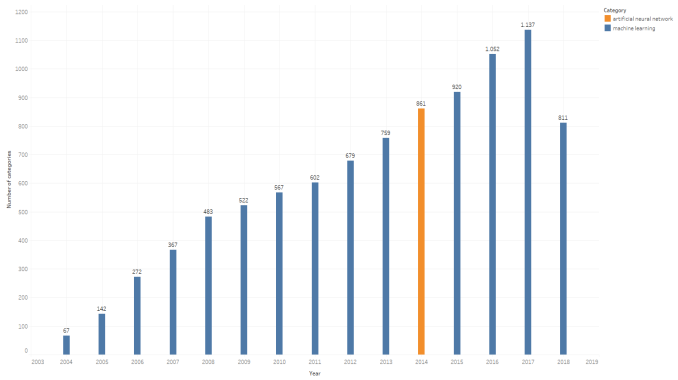

Figure 5: Result of query 8, the categories with more nodes par year

25 nodes, Classification Algorithms with 16 nodes, Neural Networks with 15 nodes, Evolutionary Algorithms with 10 nodes and Stochastic Processes with 9 nodes. The various graphs and sub-graphs are bigger and bigger over the years. Indeed in 2017 Machine Learning has 179 nodes followed by Artificial Neural Network with 116 nodes, then Machine Learning Researches with 80 nodes and Living people with 78 nodes. The Graph for 2017 has not been shown here just for legibility reasons.

6. CONCLUSIONS

At the end of the analisys of the dataset, the obtained results are almost the expected ones. The number of pages, the links between pages and the number of categories is grown during years. The Machine Learning category is on top of all the results, due to the fact that is the category-subgraph with the maximum number of nodes. The pages that are on top based on PageRank measure and the pages that have a key-role are Machine Learning, Outline of Machine Learning and Glossary of Artificial Intelligence.
The main results are the ones obtained from the queries on Category Graph. Applying that queries to the whole Wikipedia's dataset is possible to identify which are the categories with the biggest number of nodes and see its variations during years that is the goal of our problem. From the graph's figure, indeed, is possible to graphically identify which are the categories with the biggest number of nodes and using the Query 8 know this number. Applying the same technique overall, is possible to have a global view of the entire Wikipedia and, as mentioned in the Problem Statement section, a point of view of the change of society and the knowledge we have about it.

Year	From	To	Max_weight
2001	null	null	0.0
2002	genetic_programming	genetic_algorithm	29,746038
2003	perceptron	artificial_neural_network	59,005802
2004	artificial_neural_network	genetic_algorithm	129,005802
2005	artificial_neural_network	genetic_algorithm	163,4788
2006	artificial_neural_network	genetic_algorithm	235,2016
2007	artificial_neural_network	genetic_algorithm	264,40973
2008	artificial_neural_network	genetic_algorithm	284,31168
2009	artificial_neural_network	genetic_algorithm	290,6272
2010	types_of_artificial_neural_networks	recurrent_neural_network	264,5191
2011	types_of_artificial_neural_networks	recurrent_neural_network	274,0883
2012	sensorium_project	ircf360	313,24802
2013	sensorium_project	ircf360	313,0786
2014	sensorium_project	ircf360	310,8286
2015	convolutional_neural_network	deep_learning	408,96887
2016	deep_learning	convolutional_neural_network	442,5204
2017	sensorium_project	ircf360	301,51743
2018	recurrent_neural_network	convolutional_neural_network	296,18182

Table 2: Max Weight of Wordcount Graph, namely the highest similarity of two pages per year

Year	From	To	Max_weight
2001	null	null	0.0
2002	genetic_programming	genetic_algorithm	5
2003	evolutionary_algorithm	genetic_algorithm	8
2004	artificial_neural_network	genetic_algorithm	13
2005	artificial_neural_network	machine_learning	26
2006	artificial_neural_network	machine_learning	35
2007	artificial_neural_network	machine_learning	41
2008	artificial_neural_network	machine_learning	35
2009	artificial_neural_network	machine_learning	35
2010	r_(programming_language)	gnu_octave	37
2011	r_(programming_language)	gnu_octave	37
2012	statistical_classification	pattern_recognition	53
2013	statistical_classification	pattern_recognition	44
2014	statistical_classification	pattern_recognition	40
2015	statistical_classification	pattern_recognition	39
2016	statistical_classification	pattern_recognition	40
2017	outline_of_machine_learning	glossary_of_artificial_intelligence	61
2017	outline_of_machine_learning	pattern_recognition	61
2018	outline_of_machine_learning	supervised_learning	62

Table 3: Max Weight of Link Graph, namely the maximum Common neighbors of two pages

Year	From	To	Max_weight
2001	null	null	0.0
2002	genetic_programming	genetic_algorithm	5
2003	evolutionary_algorithm	genetic_algorithm	8
2004	artificial_neural_network	genetic_algorithm	13
2005	artificial_neural_network	machine_learning	26
2006	artificial_neural_network	machine_learning	35
2007	artificial_neural_network	machine_learning	41
2008	artificial_neural_network	machine_learning	35
2009	artificial_neural_network	machine_learning	35
2010	r_(programming_language)	gnu_octave	37
2011	r_(programming_language)	gnu_octave	37
2012	statistical_classification	pattern_recognition	53
2013	statistical_classification	pattern_recognition	44
2014	statistical_classification	pattern_recognition	40
2015	statistical_classification	pattern_recognition	39
2016	statistical_classification	pattern_recognition	40
2017	outline_of_machine_learning	glossary_of_artificial_intelligence	61
2017	outline_of_machine_learning	pattern_recognition	61
2018	outline_of_machine_learning	supervised_learning	62

Table 4: Max Weight of Link Graph, namely the maximum Common neighbors of two pages

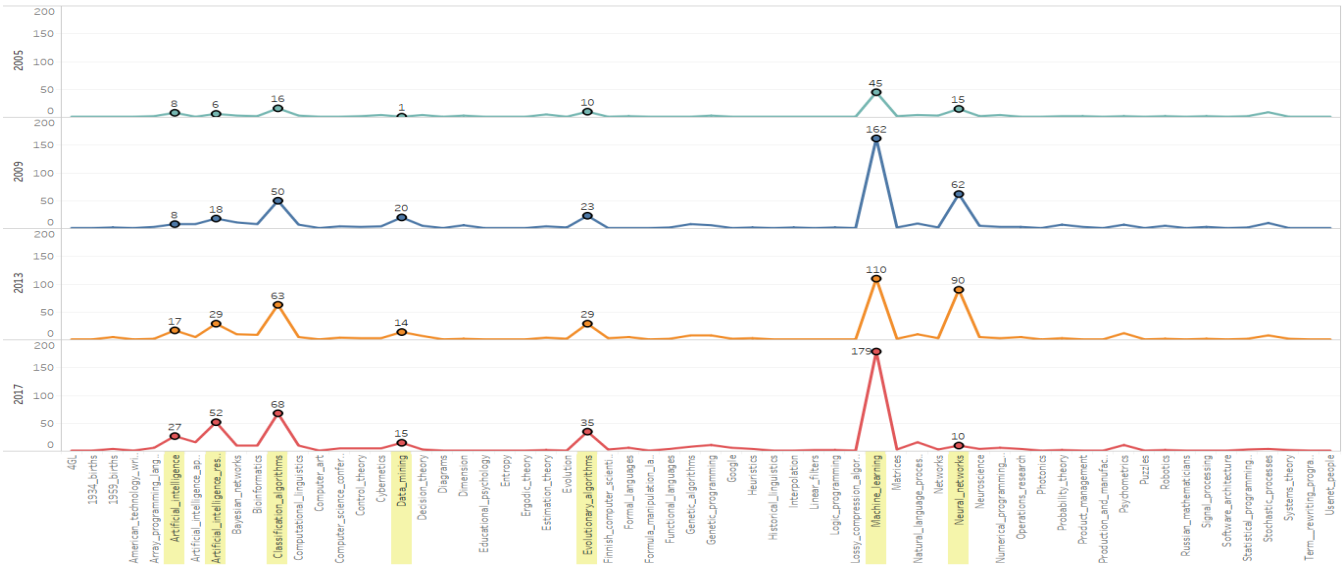

Figure 6: Number of nodes per category

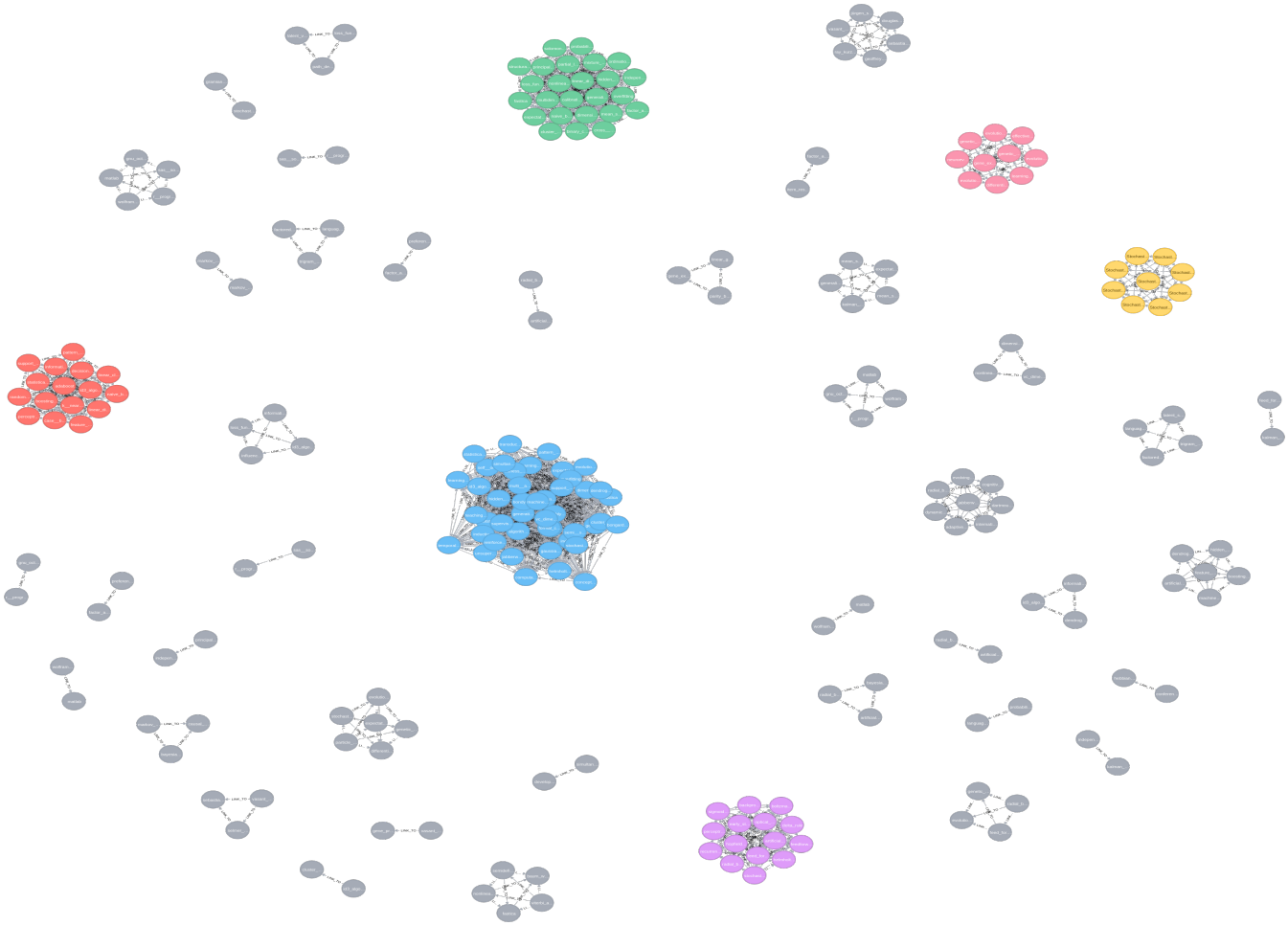

Figure 7: Graph of 2005