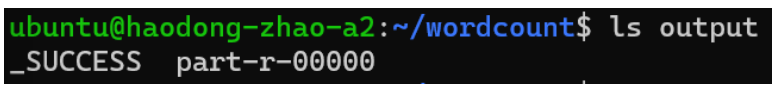
Assignment 2

Task 1

Task 1.1

```
/usr/local/hadoop-3.3.1/bin/hadoop jar /usr/local/hadoop-3.3.1/share/hadoop/mapreduce/hado
op*examples*.jar wordcount /home/ubuntu/wordcount/input /home/ubuntu/wordcount/output
```

1. ***Look at the contents of the folder "output" - what are the files placed in there? What do they mean?***

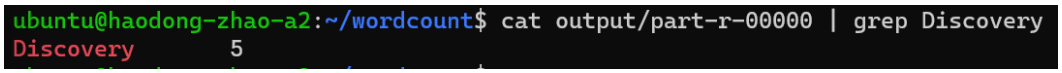


   File `_SUCCESS` and file `part-r-00000` are placed in the folder `output`

   - File `_SUCCESS` shows that there is an successful job.

- File `part-r-00000` is the output file. `r` means that the job is an reduce job, `00000` is the task number.

2. ***How many times did the word 'Discovery' (case-sensitive) appear in the text you analyzed?***



   5 times.

3. ***In this example we used Hadoop in "Local (Standalone) Mode". What is the difference between this mode and the Pseudo-distributed mode?***

   - **Local (Standalone) Mode**
     - No custom configuration is required (`mapred-site.xml`, `core-site.xml`, `hdfs-site.xml` W).
     - When need to debug, we usually choose local(Standalone) mode.
     - Standalone mode is usually **the fastest** mode of Hadoop, since it just uses the **local file system** for all the input and output.
     - HDFS is not used in local(standalone) mode.
   - **Pseudo-distributed Mode**
     - Configuration is required
     - Pseudo-distributed cluster is a cluster where all daemons are running on one node itself. All the daemons will be running as a separate process on separate JVM(Java Virtual Machine) or we can say run on different java processes.

> 💡 **Reference**
>
> 1. https://data-flair.training/forums/topic/what-are-the-modes-in-which-hadoop-run/
> 2. https://www.geeksforgeeks.org/hadoop-different-modes-of-operation/
> 3. https://www.edureka.co/community/3444/difference-between-single-pseudo-distributed-mode-hadoop

---

Task 1.2

1. ***What are the roles of the files `core-site.xml` and `hdfs-site.xml`?***

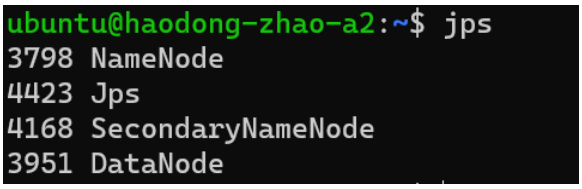
   - The `core-site.xml` file informs Hadoop daemon where **NameNode** runs in the cluster. It contains the configuration settings for Hadoop Core such as I/O settings that are common to HDFS and MapReduce.
     - For example,

       ```
       <property>
           <name>fs.default.name</name>
           <value>hdfs:// ip:port</value>
       </property>
       ```

       it defines the name of the default file system, so that the other nodes can visit the cluster.
   - The `hdfs-site.xml` file contains the configuration settings for Hadoop Distributed File System(HDFS).

     Configure `hdfs-site.xml` can define the permission checking and default block replication on HDFS.

2. ***Describe briefly the roles of the different services listed when executing `jps`.***



- **NameNode**
  - executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes.
- **DataNode**
  - Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes.
  - usually one per node in the cluster, which manage storage attached to the nodes that they run on.
  - the DataNodes are responsible for serving read and write requests from the file system's clients; perform block creation, deletion, and replication upon instruction from the NameNode.
- **Jps**

  stands for Java Virtual Machine Process Status tool, it can lists all java processes on the machine.
- **SecondaryNameNode**
  - stores modifications to the file system as a log appended to a native file system file, edits.
    - **editlogs:** It keeps track of the changes have been made in HDFS.
    - **fsimage:** It stores the snapshot of the file system.
  - When a NameNode starts up, it reads HDFS state from an image file, `fsimage`, and then applies edits from the edits log file.
  - it will communicate with NameNode periodically. If the NameNode fails, the Secondary NameNode can be used as a backup NameNode.

💡 **Reference**

1. https://www.edureka.co/blog/explaining-hadoop-configuration/
2. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
3. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html#Secondary_NameNode
4. https://data-flair.training/forums/topic/what-is-secondary-namenode-in-hadoop/

Task 1.3

1. ***Explain the roles of the different classes in the file*** `WordCount.java`**.**

   There are 2 classes in the in the file `WordCount.java`

   - class `TokenizerMapper`
     - It's the Mapper class, its job is to maps input key/value pairs to a set of intermediate key/value pairs.
     - In this program, class `TokenizerMapper` extends `org.apache.hadoop.mapreduce.Mapper`, it splits the input value into words, and make it as `(word, count)` pair.
   - class `IntSumReducer`
     - It's the Reducer class, its job is to reduce a set of intermediate values which share a key to a smaller set of values.
     - In this program, class `IntSumReducer` extends `org.apache.hadoop.mapreduce.Reducer`.

       It adds the values of pair with the same keys, and generate the key-value pair as the result.

2. ***What is HDFS, and how is it different from the local filesystem on your virtual machine?***
    - HDFS, Hadoop Distributed File System, is a distributed file system that handles large data sets running on commodity hardware. It is used to scale a single Apache Hadoop cluster to hundreds (and even thousands) of nodes.
    - HDFS vs local file system
        - Data retrieval in HDFS is fast, while in the local file system, it is slow. Thus, HDFS is suitable for analysis of big data.
        - Local file system stores data as a single block, while HDFS chunks the data file into blocks and save the chunks in to different DataNodes.
        - Local file system does not replicate data, while HDFS replicates data into different DataNodes. Therefore, HDFS is reliable.
        - The structure of the local file system is tree, while HDFS is the mater-slave structure.

Reference

1. https://www.ibm.com/topics/hdfs
2. https://www.geeksforgeeks.org/difference-between-local-file-system-lfs-and-distributed-file-system-dfs/

Task 1.4

1. Compile and make a jar-file

```
cd /home/ubuntu/FirstLetterCount
javac -cp `/usr/local/hadoop-3.3.1/bin/hadoop classpath` FirstLetterCount.java
jar -cvf firstlettercount.jar *.class
```

2. Since the input file is already put on the HDFS in the previous steps, so I don't need to put it to the HDFS. Otherwise, we need to use the command:

```
/usr/local/hadoop-3.3.1/bin/hdfs dfs -put /home/ubuntu/FirstLetterCount/input
```

3. Run the program, and save the result to folder `fl_output` in HDFS.

```
/usr/local/hadoop-3.3.1/bin/hadoop jar firstlettercount.jar FirstLetterCount input fl
_output
```

4. Check the result

```
/usr/local/hadoop-3.3.1/bin/hdfs dfs -ls fl_output
/usr/local/hadoop-3.3.1/bin/hdfs dfs -cat fl_output/part-r-00000
```

5. Get the result from HDFS

```
/usr/local/hadoop-3.3.1/bin/hdfs dfs -get fl_output/ /home/ubuntu/FirstLetterCount/ou
tput
```

```
scp -r -i /mnt/d/UU-IT.pem ubuntu@130.238.28.93:/home/ubuntu/FirstLetterCount/output
 /mnt/d
```

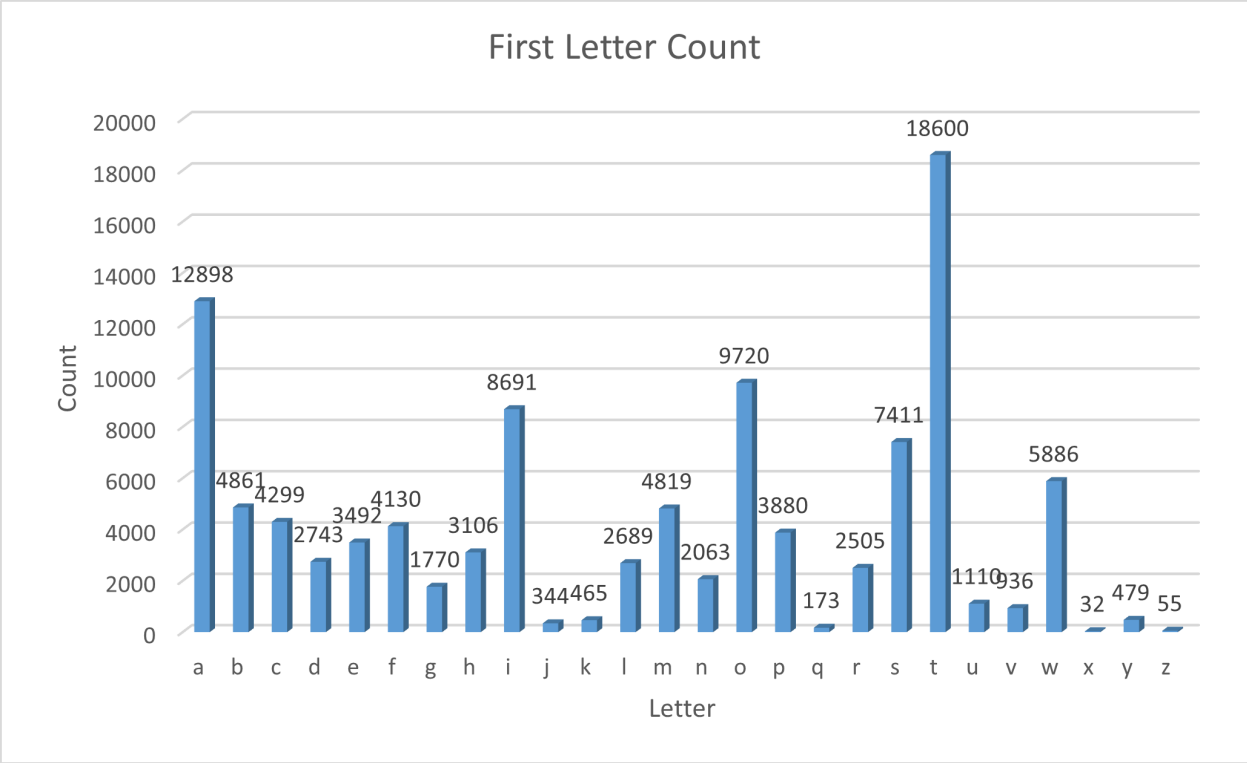


Figure 1. First Letter Count

The result is shown in Figure 1 above. We can conclude that in the analyzed text, the most words starting with the letter t, with 18600 words, while only 55 words starting with the letter z.

Task 1.5

1. ***One example of JSON formatted data is Twitter tweets: https://dev.twitter.com/overview/api/tweets.***

   Based on the twitter documentation, how would you classify the JSON-formatted tweets - structured, semi-structured or unstructured data?

   The JSON-formatted tweets are semi-structured data.

   The JSON format data has some degree of structure, but it's not as strict as the structured data. This kind of data uses tags-like things to separate the elements and divide the hierarchies. The size of the elements are different.

2. ***Elaborate on pros and cons for SQL and NoSQL solutions, respectively. Give some examples of particular data sets/scenarios that might be suitable for these types of databases. (expected answer length: 0.5 A4 pages)***

   SQL:

   - Pros:
     - SQL is a standardized language.
     - SQL is highly suitable for relational databases.
     - SQL does not need code skills
     - SQL has ACID properties(Atomicity, Consistency, Isolation, Durability)
   - Cons:
     - Handling Big data is very costly as you will have to increase the hardware for scaling.
     - A SQL database's schema must be defined before use. So, it's inflexible and hard to modify.

   NOSQL:

   - Pros
     - NOSQL is suitable for big data and building up distributed systems.
     - NOSQL does not need to define schema first, it's flexible.
     - Easier and low-cost scalability.
   - Cons
     - There is no standardized language for NOSQL
     - NOSQL is hard to do some complex queries, since some of them has no JOIN
     - NOSQL is possible to receive inconsistent information.
     - Smaller community, less community support

Examples:

- **NOSQL**
  1. Panama Papers.

     It's suitable to use graph-based NOSQL to analyze Panama Papers. We can store the relationships in the graph database. The person is suspicious if has high connectivity.

     For example, if one person chair of a large number of companies and the companies have the same address, then the person is suspicious.

     It's easy for us to find out these person, companies and addresses by using graph-based NOSQL.

  2. Twitter Data

     Twitter generates a large amount of daily tweets. It might be suitable to save the tweets as semi-structured data like JSON to NOSQL System.

- **SQL**
  - Bank System

    In a bank system, the SQL is suitable to manage customer's transaction and bank accounts. Since the bank system needs ACID properties to make sure the consistency of the transactions and the states of the accounts are correct.

  - Management systems

    Most of management systems are object-oriented, data have well-defined structures. It's suitable to save this kind of data into relational databases and manage the data by using SQL.

Reference

1. https://www.softwaretestinghelp.com/sql-vs-nosql/
2. https://www.datastax.com/blog/sql-vs-nosql-pros-cons

Task 2

Task 2.1

Preparation

- get tweets data

```
tar xvf ./tweets.tar -C TwitterAnalyze/tweet_input/
```

- save data to hdfs

```
/usr/local/hadoop-3.3.1/bin/hdfs dfs -put /home/ubuntu/TwitterAnalyze/tweet_input/
```

Implementation

1. **Mapper (mapper.py)**
    a. list the keywords and save the keywords into a list.
    b. read the JSON file line by line, and for each line:
        i. convert data to JSON data
        ii. if the JSON data has `retweeted_status`, which means it's not the unique tweet, exclude it, otherwise
            1. extract the text in the `text` field and split it in to words.
            2. If the word is in the keywords list, make a `(keyword, 1)` pair.
2. **Reducer (reducer.py)**
    - read the data from the Mapper, and group multiple word-count pairs by word
    - in each group, calculate the sum of the counts and save it as a new `(keyword, sum)` pair.

Execute

```
/usr/local/hadoop-3.3.1/bin/hadoop jar /usr/local/hadoop-3.3.1/share/hadoop/tools/lib/hadoop-*streaming*.jar \
-file /home/ubuntu/TwitterAnalyze/mapper.py    -mapper /home/ubuntu/TwitterAnalyze/mapper.py \
-file /home/ubuntu/TwitterAnalyze/reducer.py   -reducer /home/ubuntu/TwitterAnalyze/reducer.py \
-input tweet_input/* -output tweet_output
```

Result

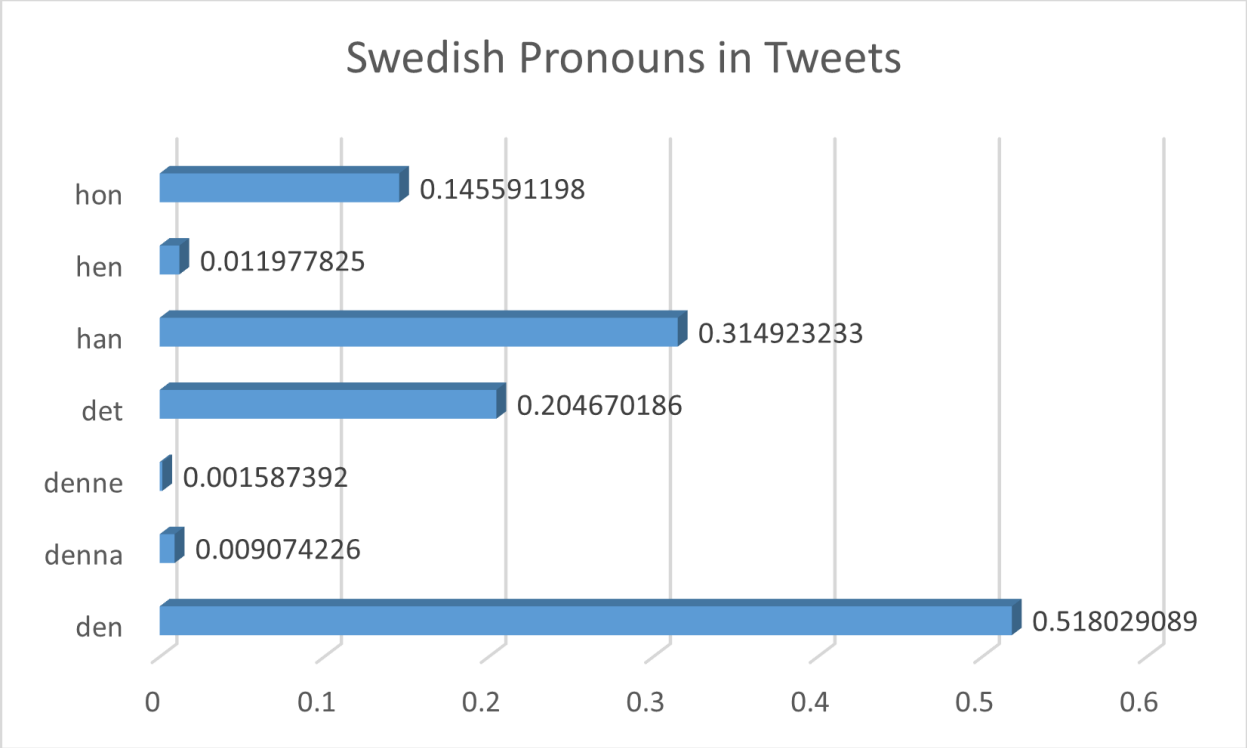


Figure 2. Swedish Pronouns in Tweets

The result is shown in Figure 2 above. We can conclude that `den` dominates Swedish pronouns in Swedish tweets, it appears in more than half of the unique tweets. While `hen`, `denne` and `denna` are the least common Swedish Pronouns.

Task 2.2

Motivate your chosen implementation and how you did it.

1. Load tweets data into MongoDB (`mydb.tweets`)

```
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_1.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_2.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_3.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_4.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_5.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_6.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_7.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_8.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_9.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_10.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_11.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_12.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_13.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_14.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_15.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_16.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_17.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_18.txt
mongoimport -d mydb -c tweets --file TwitterAnalyze/tweet_input/tweets_19.txt
```

2. Count the required Swedish pronouns
   - Unique tweets

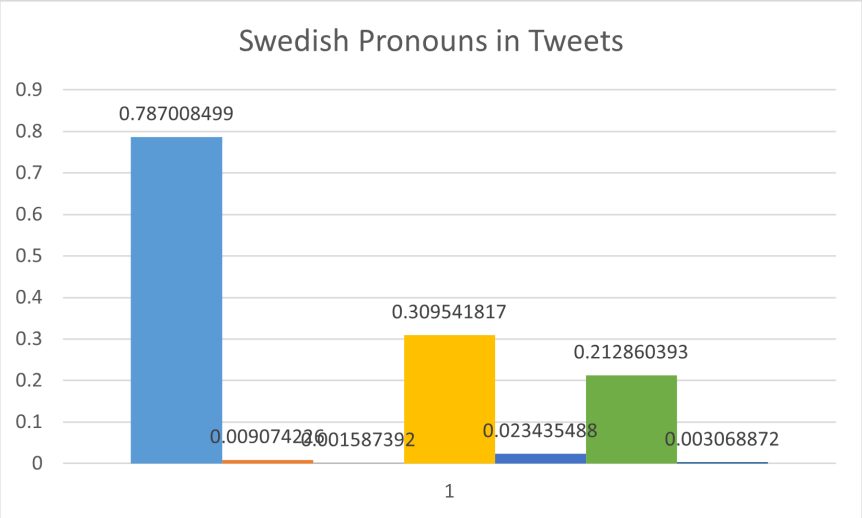
```
db.tweets.find({ retweeted_status: { $exists: false } }).count()
```

   - Swedish Pronoun Count by using `db.collection.aggregate`
     - use regex to filter out the needed keywords
     - use `"$addFields":{"count":1}` to make them as `(key, value)` pairs
     - use `$group` to do the reduce
     - use `$out` to save data into `tweets_result`.

```
db.tweets.aggregate(
  [
    {
      "$project":{"matches":{"$regexFindAll":{"input":"$text", "regex":"(*UCP)\\b
```

```
(han|hon|den|det|denna|denne|hen)\\b", "options":"i"}}}
    },

    {
      "$set":{"matches": "$matches.match"}
    },
    {
      "$unwind": {"path": "$matches", "preserveNullAndEmptyArrays": false}
    },
    {
      "$addFields":{"count":1}
    },
    {
      "$group":{"_id":{"$toLower":"$matches"}, "total":{"$sum":"$count"}}
    },
    {
      $out: "tweets_result"
    }
  ]
)
```



What are some pros and cons of the MongoDB solution compared to the implementation you did in Task 2.1?

Pros:

1. easy to insert JSON file into MongoDB.
2. do not need to parse JSON objects
3. easy to use regex

Cons:

1. In my view, it’s not easy to write MongoDB search command using `db.collection.aggregate`
2. It needs to load the data into MongoDB first.
3. It’s really slow to insert data into MongoDB compared with Hadoop HDFS.
4. It’s slower than Hadoop to get the result .