Big Data Project

Clara Maine (s1032005)
26 June 2022

Is the internet more “good” or “bad”? Analyzing the frequencies of opposites in the CommonCrawl.

Description of the basic idea that motivates your project

Anthropologists, linguists, and philosophers have noticed that there is a universal human tendency to organize our world into binary distinctions. Up vs Down, Left vs Right, Love vs Hate, Good vs Bad. These are essentially universal across cultures, but what about how often we use them or read them? With this I project analyzed the absolute frequency of pairs of antonyms in a section of the CommonCrawl dataset.

The Process

(how divide and conquer, smaller steps to eventually get a project result)

I approached the project iteratively, beginning first with a basic look at the frequencies of “good” vs ”bad”. I then then extending my analysis to include a more comprehensive list of antonyms and ran it first on a single webpage, then on a warc file. I unfortunately did not have sufficient time to wait for the results of the single-warc-segment for this project.

(what we’ve done during the project and what we’ve executed on the cluser)

Before I even got a WARC file, I ran a test case on the Wikipedia page for “Morality”, “Good” and “Evil”:—thinking that those pages would probably have an interesting split between using “good” and “bad”. I followed the “WARC for Spark” Zeppelin notebook and created a fake `warcfile` with the contents of the requested Wikipedia page. I then ran the following code

```
// Create a container for the warcfile contents
val warcs = sc.newAPIHadoopFile(
              warcfile,
              classOf[WarcGzInputFormat],             // InputFormat
              classOf[NullWritable],                  // Key
              classOf[WarcWritable]                   // Value
    ).cache()

//separate out the warc records from the crawl metadata
val warc_records = warcs.map{ wr => wr._2 }
                        .filter{_.isValid()}
                        .map{_.getRecord()}

//Separate out the english text content from the webpage
val warc_text = warc_records
            .map(wr => wr.getHttpStringBody() )
            .map(wr => Jsoup.parse(wr))
            .filter(wr => wr.select("html").first().attr("lang") == "en")
            .map(_.body().text())
            .filter(_.length > 0)

//Do some basic NLP to split on words
val warc_words = warc_text
            .flatMap(_.split(" "))
            .map(_.replaceAll(
                "[^a-zA-Z]", "")
                .toLowerCase
            )

//Finally,
val wordcounts = warc_words
            .filter(w => w == "good" || w == "bad")
```

```
            .map(word => (word, 1))
            .reduceByKey(_+_)

// As a small aside, in an earlier verion of this I counted the
// frequency of every word and then filtered out everything but
// good or bad, but I realized that was probably an inefficient
// way to do it since I think that aggregating a smaller number
// of words is less expensive than filtering

wordcounts.take(2).foreach(println)
```

The Results

Wikipage	Mentions of “good”	Mentions of “bad”
Morality - Wikipedia	15	0
Good - Wikipedia	54	3
Evil - Wikipedia	48	7

This seemed surprising, especially the infrequent mentioning of bad in the Evil wikipedia page, but when I did another check to see how many times “evil” was mentioned, it was `13` for Morality, `33` for Good, and `216` for Evil. This revealed the first major difficulty of this project which is that even things which appear to be direct opposites can conflict. Even in Wikipedia’s database, there is no singular webpage for “bad” like there is for “good”. Rather, it is a collection of pages reffering to the word’s common meanings, acronyms, uses in media etc. This made the project a bit more interesting, since it revealed that “good” seemed to have a more concrete meaning than “bad”—enough to deserve its own wikipedia page.

Importing Anyonym Data

From this realization, I realized it might be useful to seek external sources for determining antonyms. I found this Large-Scale Collection of English Antonym and Synonym Pairs across Word Classes and had to import this dataset into spark. Since this dataset was split into real and fake antonym pairs, I had to filter out the fake pairs with this python code.

```
# Python Code in a Jupyter Notebook
import csv
import numpy as np

antonym_pairs = []

with open('./ant_syn_pairs/nouns.txt') as data:
    data_reader = csv.reader(data, delimiter='\t')
    for pair in data_reader:
        antonym_pairs.append(pair)

with open('./ant_syn_pairs/adjectives.txt') as data:
    data_reader = csv.reader(data, delimiter='\t')
    for pair in data_reader:
        antonym_pairs.append(pair)

with open('./ant_syn_pairs/verbs.txt') as data:
    data_reader = csv.reader(data, delimiter='\t')
    for pair in data_reader:
        antonym_pairs.append(pair)

filtered_antonym_pairs = antonym_pairs.copy()

for i in range(len(antonym_pairs)):
    pair = antonym_pairs[i]
    if int(pair[2]) == 0:
        filtered_antonym_pairs.remove(pair)

filtered_antonym_pairs = np.array(filtered_antonym_pairs)[:,:2]
np.savetxt("antonym_data.csv", filtered_antonym_pairs, delimiter=",", fmt='%s')
```

In the end, I had a dataset with 5,466 antonym pairs. I now had to upload this into spark and be able to filter my wodcount program to only count these words.

To do this I uploaded my new `antonym_data.csv` file to the big-data cluster (and eventually to redbad) and read it into a zeppelin notebook with the following code

```
%spark
import org.apache.spark.sql.types._
val schema = new StructType()
      .add("ant1",StringType,true)
      .add("ant2",StringType,true)

val antpairs = spark.read
        .format("csv")
        .option("header", false)
        .option("multiLine", true)
        .schema(schema)
        .load("file:////opt/hadoop/rubigdata/antonym_data.csv").cache() //faster local

antpairs.printSchema()
antpairs.show(5)

///// Output: //////
root
 |-- ant1: string (nullable = true)
 |-- ant2: string (nullable = true)

+-------------+-----------+
|         ant1|       ant2|
+-------------+-----------+
|infinitesimal|   infinity|
|     grandson|grandfather|
|   angiosperm| gymnosperm|
|   illiteracy|   literacy|
|      defence|prosecution|
+-------------+-----------+
only showing top 5 rows
```

Counting Antonym Pairs

From this, I had to incorporate this `antpairs` dataframe into the wordcount code I had written above. It was pretty difficult, first to figure out how to filter the `wordcounts` rdd to only count the words in either column of the `antpairs` list. I tried various operations working with the columns of the dataframe like `.filter(w => antpairs("ant1").contains(w)` but spark doesnt let me call contains on a column, so eventually I converted each column of antpairs into a separate list with `.toList` and used those lists for comparison. In the end, this solution feels pretty inefficient since it has to do two `.contains` for every single word (which is going to be a LOT when we get to the full WARC files), but I wasn't sure of a better way to do it.

I also converted the RDD to a dataframe since that made the join in the next section much easier. The output

```
val warc_text = warc_records
            .map(wr => wr.getHttpStringBody() )
            .map(wr => Jsoup.parse(wr))
            .filter(wr => wr.select("html").first().attr("lang") == "en")
            .map(_.body().text())
            .filter(_.length > 0)

val warc_words = warc_text
            .flatMap(_.split(" "))
            .map(_.replaceAll(
                "[^a-zA-Z]", "")
                .toLowerCase
            )

val antpairslist1 = antpairs.select("ant1").map(f=>f.getString(0)).collect.toList
val antpairslist2 = antpairs.select("ant2").map(f=>f.getString(0)).collect.toList

val wordcounts = warc_words
            .filter(w => antpairslist1.contains(w) || antpairslist2.contains(w))
            .map(word => (word, 1))
            .reduceByKey(_+_)
            .sortByKey()

val wordcountdf = wordcounts.toDF("word", "count")

wordcountdf.show(5)
```

```
/////// Output //////// (These are counts from https://en.wikipedia.org/wiki/Good)
+-----------+-----+
|       word|count|
+-----------+-----+
|  practical|    2|
|    poverty|    1|
|    eastern|    1|
|rationalism|    1|
|       pain|    1|
+-----------+-----+
only showing top 5 rows
```

After I got `wordcountsdf` with the counts of all the relevant terms, I wanted to project those counts back onto the original antonym pairs dataframe, essentially using this project as an opportunity to expand the antonym dataset with a rough approximation of their usage frequencies.

This was also really difficult. I had to join `wordcounts` twice on each column of `antpairs` and continually rename the count columns so the join wouldn't get confused. This took me a good few hours, first trying to just join the pure RDD of wordcounts, then trying various joins and selections, but eventually my final solution worked. It is included as the code below. The last step was saving the modified data to a single csv file.

```
val antpairscounts = antpairs.join(
                                wordcountdf.withColumnRenamed("word1", "count1").alias("c1"),
                                $"c1.word" === antpairs("ant1"),
                                "right"
                                )
                           .withColumnRenamed("count", "count1")
                           .join(wordcountdf.withColumnRenamed("word2", "count2").alias("c2"), $"c2.word" === antpairs("ant2"))
                           .withColumnRenamed("count", "count2")
                           .select("ant1", "count1", "ant2", "count2")

bigantpairs.show(5)

val output_path = s"hdfs:///user/s1032005/antonym_counts.csv"
  antpairscounts.coalesce(1)
      .write
      .option("header","true")
      .option("sep",",")
      .mode("overwrite")
      .csv(output_path)
  print("Antonym pair counts Saved!")

///// Output ////// (These are counts from https://en.wikipedia.org/wiki/Good)

+----------+------+--------+------+
|      ant1|count1|    ant2|count2|
+----------+------+--------+------+
|      good|    54|    evil|    33|
|  goodness|     1|    evil|    33|
|    simple|     1| complex|     1|
|immorality|     2|morality|    10|
|   western|    10| eastern|     1|
+----------+------+--------+------+
only showing top 5 rows
```

Expanding to a WARC file

I was pretty happy with this output, and the cvs file sucessfully saved, so I was now ready to run my code on a WARC file in the cluster.

I used the shell of the RUBigDataApp from one of the homework assignments and changed the directory to point to a single WARC file I had downloaded from the crawl-data folder to my personal redbad directory. The full file is this

```
package org.rubigdata

// Old imports I dont want to delete
import org.apache.spark.sql.types._
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.scalalang.typed

// Class definitions we need in the remainder:
```

```
import org.apache.hadoop.io.NullWritable
import de.l3s.concatgz.io.warc.{WarcGzInputFormat,WarcWritable}
import de.l3s.concatgz.data.WarcRecord

// Overrule default settings
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession

// needed for processing text
import org.jsoup.Jsoup
import org.jsoup.nodes.{Document,Element}
import collection.JavaConverters._

// needed for creating the schema
import org.apache.spark.sql.types._

object RUBigDataApp {
  def main(args: Array[String]) {
    //val warcfile = s"hdfs:///single-warc-segment"
  val warcfile = s"hdfs:///user/s1032005/CC-MAIN-20210417102129-20210417132129-00338.warc.gz"

  val spark = SparkSession.builder.appName("RUBigDataApp").getOrCreate()
  spark.sparkContext.setLogLevel("WARN")

    import spark.implicits._

  val sc = spark.sparkContext
  val warcs = sc.newAPIHadoopFile(
              warcfile,
              classOf[WarcGzInputFormat],             // InputFormat
              classOf[NullWritable],                  // Key
              classOf[WarcWritable]                   // Value
    )

  //sanity check
  val nHTML = warcs.count()
  println(nHTML)
  println(s"HELLO ITS WORKING AND THE OUTPUT IS HERE AHHHHHHHH")

  val warc_records = warcs.map{ wr => wr._2 }
                        .filter{_.isValid()}
                        .map{_.getRecord()}

  val schema = new StructType()
      .add("ant1",StringType,true)
      .add("ant2",StringType,true)

  val antpairs = spark.read
        .format("csv")
        .option("header", false)
        .option("multiLine", true)
        .schema(schema)
        .load("hdfs:/user/s1032005/antonym_data.csv") //.cache()

  // sanity check
  antpairs.printSchema()
  antpairs.show(5)


  val warc_text = warc_records
        .map(wr => wr.getHttpStringBody() )
        .map(wr => Jsoup.parse(wr))
        .filter(wr => wr.select("html").first().attr("lang") == "en")
        .map(_.body().text())
        .filter(_.length > 0)

  val warc_words = warc_text
        .flatMap(_.split(" "))
        .map(_.replaceAll(
          "[^a-zA-Z]", "")
          .toLowerCase
        )

  val antpairslist1 = antpairs.select("ant1").map(f=>f.getString(0)).collect.toList
  val antpairslist2 = antpairs.select("ant2").map(f=>f.getString(0)).collect.toList

  val wordcounts = warc_words
        .filter(w => antpairslist1.contains(w) || antpairslist2.contains(w))
        .map(word => (word, 1))
        .reduceByKey(_+_)

  val wordcountdf = wordcounts.toDF("word", "count")
```

```
  wordcountdf.show(5)

  val antpairscounts = antpairs.join(
                                wordcountdf.withColumnRenamed("word1", "count1").alias("c1"),
                                $"c1.word" === antpairs("ant1"),
                                "right"
                                )
                           .withColumnRenamed("count", "count1")
                           .join(wordcountdf.withColumnRenamed("word2", "count2").alias("c2"), $"c2.word" === antpairs("ant2"))
                           .withColumnRenamed("count", "count2")
                           .select("ant1", "count1", "ant2", "count2")

  antpairscounts.show(5)

  val output_path = s"hdfs:///user/s1032005/antonym_counts"
  antpairscounts.coalesce(1)
      .write
      .option("header","true")
      .option("sep",",")
      .mode("overwrite")
      .csv(output_path)

  println("Antonym pair counts Saved!")

  spark.stop()
  }
}
```

Following the same process as in previous assignments, I copied `RUBigDataApp.scala` to the redbad cluster, compiled it with `sbt assembly` and ran it. It took a couple tries to get it working, due to some directory issues, but eventually it ran and I was able to download the output csv file to my local machine and look at it.

Results

I will attach the csv file in this submission, but I thought I would share some of the most fun and interesting results here.

ant1	count1	ant2	count2
good	16	bad	8
more	570	less	1
all	436	none	2
winter	85	summer	14
used	55	new	285
global	48	local	25
buy	26	sell	28
peace	10	war	2
right	28	left	11

There were, surprisingly, no mentions of “love” or “hate” in thie warc file we looked at.

Analysis & Conclusion

(conclusions about output, what went well what went wrong)

I thought about expanding my code to run on the `single-warc-segment` which contains around 600 warc files, but given my timeframe for this project I wouldn’t have enough time to wait for it to complete its run, since my code is fairly costly (a single WARC file takes betweek 30 seconds to a minute). The results will not be done by the deadline of this project, but I did run it, so if you’re interested in the full csv you can contact me. If I were to expand this project I would try to find a more efficient way to do it and run it on the segment. Overall, I think the results are fun to see how much emphasis is placed on one opposite over another even if they are used the same way/frequency linguistically, but given the fact it was run on only one warc file i dont think my results show the whole picture.