Docx Anonymizer

Languages and Algorithms for Artificial Intelligence

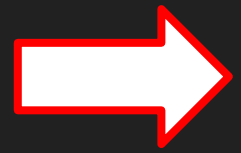
Lorenzo Mario Amorosa
Mattia Orlandi
Giacomo Pinardi

What does DocxAnonymizer do?

This is a test! Here we have Lorenzo Mario Amorosa, Mattia Orlandi and Giacomo Pinardi.

Matricola	Persona
948133	Lorenzo Amorosa
946744	Mattia Orlandi
931130	Giacomo Pinardi

input.docx



This is a test! Here we have x54ydf7g, 8erdf34d and pok6on21.

Matricola	Persona
948133	x54ydf7g
946744	8erdf34d
931130	pok6on21

output.docx

Pattern detection via Regular Expression

- Just-in-time compilation of regex
- Anonymization using dictionaries or given nominatives

```
<regex> = (?<=[^A-Za-zA-OO-
ooyL·l·ČčŘřŠšŽžIJijOEoeŸŐőŰűḂḃĊċḊḋḞḟĠġṀṁṠṡṪṫĀāĒēĪīŌōŪūİıĞğŞşẀẁẂẃŴŵỲỳØøß]|^)
(A((?i)morosa)\s+(L((?i)orenzo)\s+M((?i)ario)|M((?i)ario)\s+L((?i)orenzo)|
L((?i)orenzo)|M((?i)ario))|(L((?i)orenzo)\s+M((?i)ario)|M((?i)ario)\s
+L((?i)orenzo)|L((?i)orenzo)|M((?i)ario))\s+A((?i)morosa))
(?=[^A-Za-zA-OO-
ooyL·l·ČčŘřŠšŽžIJijOEoeŸŐőŰűḂḃĊċḊḋḞḟĠġṀṁṠṡṪṫĀāĒēĪīŌōŪūİıĞğŞşẀẁẂẃŴŵỲỳØøß]|$)
```

Full nominative

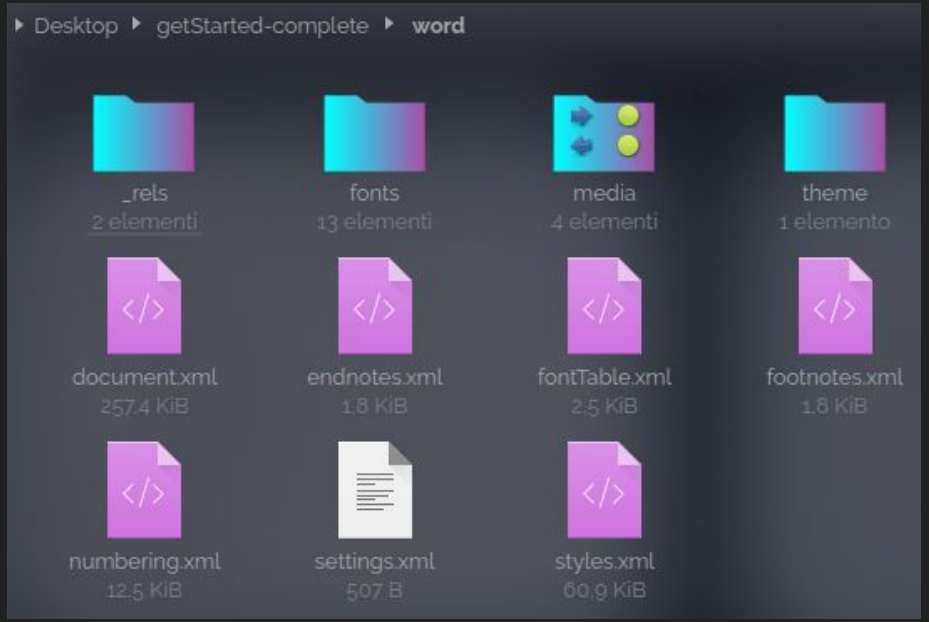
Nominative delimiter

Partial nominative

Docx Hierarchy - Open Packaging Convention

document.xml

```
−<w:document mc:Ignorable="w14 wp14">
  −<w:body>
    −<w:p>
      +<w:pPr></w:pPr>
      −<w:r>
          <w:rPr/>
          <w:t>This is a test.</w:t>
        </w:r>
      </w:p>
    +<w:sectPr></w:sectPr>
  </w:body>
</w:document>
```



XML nodes ⇒ PlainText class ⇒ *Sentence* concept with nominatives

Exploiting parallelism

1 sentence ⇒ 1 s, due to ~15K regex to parse (EN-IT dictionary of names)

Problems of the sequential solution:

- 10K sentences ⇒ 10K s (common in large PA documents)

Advantages of the parallel version (Scala + Spark):

- 10K sentences ⇒ 10K s / *N*, with *N* number of workers
- *Dividi et impera* paradigm:
  - Document split between available nodes through shared data structures
  - Parallel and independent anonymization
  - Partial results are finally merged together to produce complete statistics and the anonymized Docx (saved remotely on S3)

ID association

With anonymization every name is associated with an unique ID, so that by using the list of correspondences it is possible to revert the anonymization process.

Mario Rossi  vkj74rto

Problem: a **mutable** shared data structure is needed to store the correspondences between real names and anonymous IDs

Solution: IDs are generated on the go with a deterministic process (MD5 hashing function)

- every worker computes a local association and frequency map
- all local association and frequency maps are reduced in two single global maps, thanks to the very low collision probability

Processing with FoldLeft

The process starts from a list of tuple of strings (regex, plainName) that represent the list of people to anonymize.

[(regex1, Luca), (regex2, Carlo)]

The initial value is the preprocessed string which contains the document's string slightly modified to facilitate the manipulation, but not anonymized yet.

Carlo and Luca studied AI at Unibo...

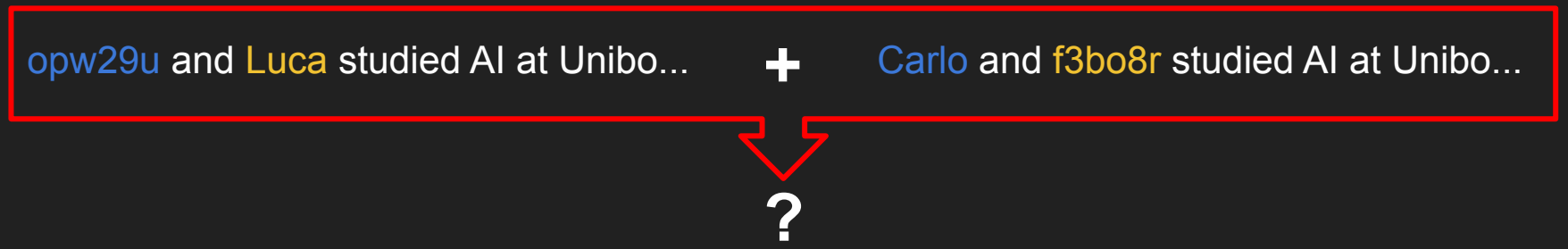
For each successive iteration the preprocessed string is analyzed and through the regex all the matching element are anonymized.

opw29u and Luca studied AI at Unibo...  opw29u and f3bo8r studied AI at Unibo...

Is a full parallel version possible?

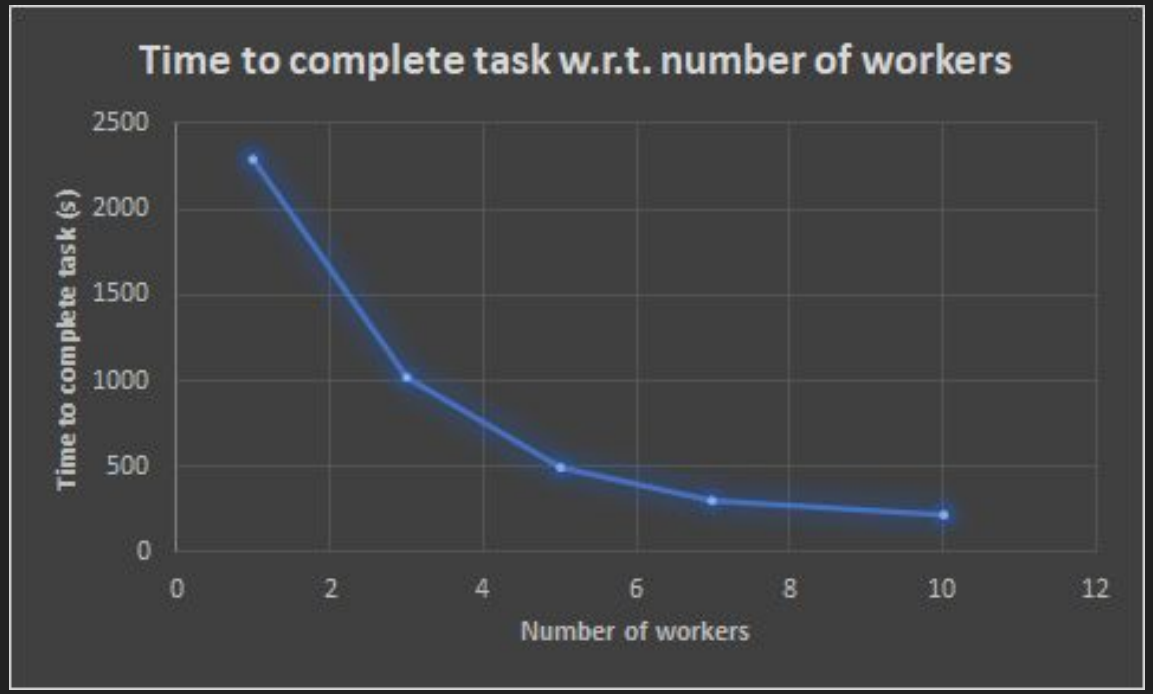
We decided to use foldLeft because of the difficulty to anonymize the string in a completely parallel fashion.

We could have split the (regex, plainName) list to multiple nodes, but then how to recompose the partially anonymized strings into a single completely anonymized result?



The entry point information is lost and difficult to recombine.

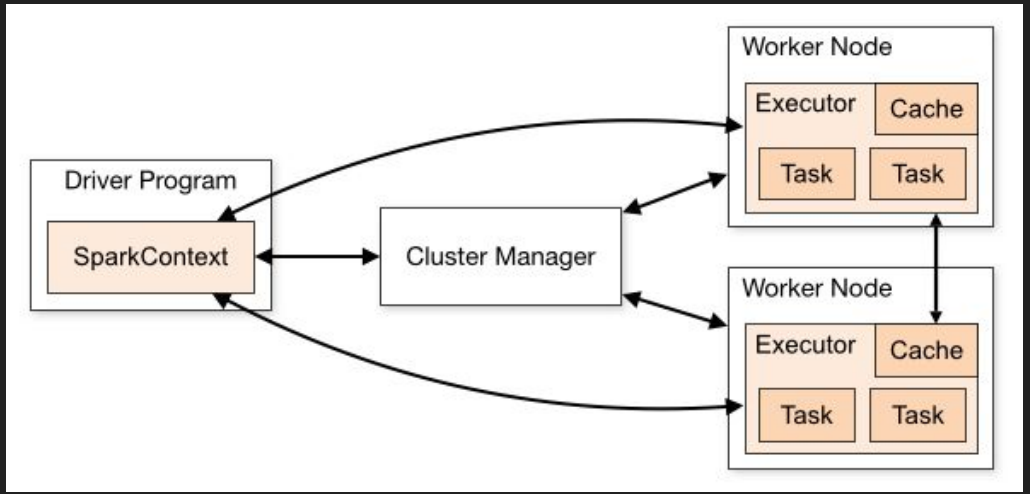
Comparing results



All tests are made with the same file “Graduatoria.docx”

Distributed computation

- One *RDD* contains sections of the whole document ⇒ every worker anonymizes a section
- All other variables are *broadcasted*: a copy is stored in the cache of each worker
- Broadcasted variables are read-only



Scala and Spark features - 1

DocxAnonymizer makes use of several Scala + Spark features:

- *Option[T]* for optional values + *match-case-None/Some[T]*:

```
s3Bucket = Option(commandLine.getOptionValue("s3"))
s3Bucket match {
    case None => ... // Code in case of local files
    case Some(bucketName: String) => ... // Code in case of S3 files
```

- *Either[T, U]* for fallible functions + *match-case-Left[T]/Right[U]*:

```
cmdArgsRetriever(args) match {
    case Left(message: String) => println(s"ERROR: $message")
    case Right(_: Unit) => ... // Code to preprocess docx file
```

- *Try[T]* to encapsulate exceptions + *match-case-Failure/Success[T]*:

```
Try(WordprocessingMLPackage.load(new File(inputFilePath))) match {
    case Failure(_: Docx4JException) => Left(s"could not load $inputFilePath")
    case Success(wordMLPackage: WordprocessingMLPackage) => ... // Code to read docx file
```

Scala and Spark features - 2

- *RDD* to distribute computation + *map()* and *collect()* to transform data and collect results:

```
val processed: RDD[(String, Seq[EntryPoint], mutable.Map[String, Int], mutable.Map[String, String])] = sc.parallelize(inputs.toSeq.sortBy(_._1)) map { ... // Code to transform data }
...
val globalAssociations: Map[String, String] =
processed.collect().map(_._4).reduce(_++_).toMap
```

- *Source.fromFile()* to read files in a functional fashion + Loan pattern to close resources:

```
def using[A <: { def close(): Unit }, B](resource: A)(f: A => B): B =
    try {
        f(resource)
    } finally {
        resource.close()
    }
...
val toProcess: List[String] = using(Source.fromFile(filePath)) { source =>
source.getLines().toList }
```

Java Integration - 1

Seamless integration with Java codebase thanks to *scala.collection.JavaConverters* package:

- *.asJava* to convert from Scala to Java collection:

```
yield new Persona(surname, names.toList.asJava, id)
```

- *.asScala* to convert from Java to Scala collection:

```
val allEntryPoints: Map[Int, Seq[EntryPoint]] =
AnonymUtils.getEntryPoints(elaborator).asScala.groupBy(_.getIndex_PlainText)
```

Java Integration - 2

Use of Java 8 Streams in auxiliary Java class *AnonymUtils* to further improve integration:

```
public class AnonymUtils {
	...
	public static List<String> preprocess(Elaborator elaborator) {
		return elaborator.preprocess().stream().map(StringBuilder::toString)
			.collect(Collectors.toList());
	}
	...
	public static List<EntryPoint> getEntryPoints(Elaborator elaborator) {
		return elaborator.getPlainTexts().getEntryPoints().stream()
			.map(e -> new EntryPoint(null, e.getIndex_PlainText(), e.getFrom(), e.getTo()))
			.collect(Collectors.toList());
	}
	...
}
```

Greetings

Thanks for your attention!