Named Entity Identification

PAMPO (PAttern Matching and Part Of speech tagging named entity recognition)

Paulo Leal - UP201101503

Faculdade de Ciências da Universidade do Porto

Abstract. The aim of this work is to produce a stand-alone application that can process Artificial Intelligence and Text Mining techniques in order to automatically identify entities in text. As a prototype already exists the main goal would be to find a way to distribute it, either as an application or an online service. Being written in R several different packages were used and with that the issue of version management appeared, seeing as specific versions of both R and each package had to be used to ensure correct functionality simply creating a distribution was not an option. We opted to do one of each having three distinct variants: a local, offline, application that analyzes the text and retrieves the information, having it immediately available, with an interface; the same as before but via console only, analyzing the text and storing the data in a text file of itself consisting only of what was mined; an online service to which you can upload text and the data will be returned. In order for the local version to operate without issues we choose to create a sort of time lapse in which updating is not an immediate concern, thusly "freezing" the packages and resolving version conflicts. After this was done the console option was merely a case of adjusting the input and ouput and the service was created by developing a server that implemented the offline console version and sent the output back to the client. One possible interpretation of this problem and solution is that sometimes simply creating an application isn't the best course of action due to compatibility issues, and providing a service or an alternative method of access may be a better option and that creating something that works without promising extensive features may be feasible while keeping a project in active development may not.

1 Introduction

Named Entity Recognition(NER) is a major task of Natural Language Processing(NLP) and it's been an active area of research for the past twenty years. That being said it becomes obvious that there's an inherent importance (ergo motivation) associated with this work as it may be a contribute of significance to that area, at the very least allowing for a better understanding of the topic.

The aim of this work is to produce a stand-alone application that can process documents and automatically identify Named Entities, in which the application itself can be given a wide use, meaning that it's format must facilitate implementing it as part of a bigger project. This paper describes a multi-faceted deployment solution for a Named Entity Recognition algorithm implemented in R.

This was done be creating an R Script that executed the algorithm without an interface (as it was supplied built on Shiny), followed by the development of a small java application that takes a file or files as input and runs them through the script returning the extracted and parsed data. A java client and server were also done in order to be able to deploy it either locally, on a network, or via a web service.

The result was a package for distribution containing all of the required installation files for R (3.0.2), the packages R uses divided by their dependence upon each other, and the multiple means of deployment (local, client, and server).

2 Named Entity Recognition

Named entities can be considered elements in text that belong to a predefined category, such being a proper name, an organization, a location, and so forth. Named Entity Recognition(NER) is the task of identifying those Named Entities(NE) which are categorized into three types of label by the Sixth Message Understanding Conference(MUC) [1] - a conference initiated by DARPA about information extraction - which follow [1]:

– ENAMEX: person, organization, location;
– TIMEX: date, time;
– NUMEX: money, percentage, quantity;

Example of annotation from MUC-7 data in English [1]:

```
The <ENAMEX TYPE="LOCATION">U.K</ENAMEX> satellite
television broadcaster said its subscribers base grew
<NUMEX TYPE="PERCENT">17.5 percent</NUMEX> during
<TIMEX TYPE="DATE"> the past year</TIMEX> to 5.35 million
```

There are several approaches to NER but they can be divided into three main ones: Supervised methods, Semi supervised methods and Unsupervised methods. Supervised methods learn a model via annotated training examples and are currently the dominant technique, Hidden Markov Models (D. Bikel et al. 1997) [4], Decision Trees (S. Sekine 1998) [5], Maximum Entropy Models (A. Borthwick 1998) [6], Support Machines (M. Asahara and Matsumoto 2003) [7] and Condition Random Fields (A. McCallum and Li 2003) [8] are some of those models; Semi supervised learning algorithms use both labeled and unlabeled corpus to create their own hypothesis[2]. Unsupervised methods are perhaps the most interesting as they use pre-made patterns to try and assume candidate facts which are consequently immediately tested for their plausibility [1, 2]; these patterns usually gather named entities from clusters grouped on the similarity of their context [2].

An example, given the following sentence: ”In 1998, Larry Page and Sergey Brin founded Google Inc.” [3]

```
FounderOf(Larry Page, Google Inc.),
FounderOf(Sergey Brin, Google Inc.),
FoundedIn(Google Inc., 1998 ).
```

This is obtained by further iterating upon the MUC data, evaluating it. A system is scored by its ability to correctly find a TYPE and the ability to find exact TEXT; TYPE is credited if an entity is assigned the correct type regardless of boundaries and TEXT is credited if entity boundaries are correct regardless of type. Both of these axes have three measures and the final MUC score is the micro-averaged f-measure (the harmonic mean of precision and recall over all entity slots over both axes). [2] In MUC precision is calculated and this relation is the percentage of correct relation instances among all discovered. [3]

In conclusion the aim of NER is to extract (and classify in NERC) any rigid designators mentioned in text, otherwise called Named Entities. There is a vast diversity of languages, domains and entity types and each system, most of the time hand-crafted, will have to deal with a specific scenario meaning that there's a high system engineering cost, although this comes at great performance [2]. Although most NER implementations are for the English language the one supplied is in the Portuguese language and it expands over the MUC labels by adding a fourth for events. It's existence, admitting it becomes a good application, serves as a mean of support to other areas that rely on text classification, recommendations, etc. In short any sort of analysis area may utilize this work which is something essential to ensure growth in the quality of pre-processing for applications.

3 Approach

Running the R algorithm presented a problem right at the start due to everything needing to have a specific version, these compatibility issues also prevented us from using several server solutions for R (Rserve, Shinyapps.io, Shiny Server). Updating the algorithm wasn't a choice since using a recent version of R would also require recent packages and this would break the algorithm due to some functions being changed in the packages themselves.

Our approach consisted of creating a small program (in java) that could run input files through the R algorithm since that was something we could adapt for other methods. Using that java implementation as an offline version [Figure 1] we were able to create a server and consequently a client to communicate with it, thus a networked version [Figure 2]. As a result we were left with a multi-faceted deployment for the R algorithm:

- Offline [Figure 1]
    - Compromised of two versions
        - Returns the parsed data
        - Returns file with parsed data
    - Networked [Figure 2]
        - Client
            - Sends file(s) to server and receives data
        - Server
            - Receives file(s) from client, runs them through the offline version (which calls the R script), returns data to client.

Note: Both figures mentioned here can be found in the next page.

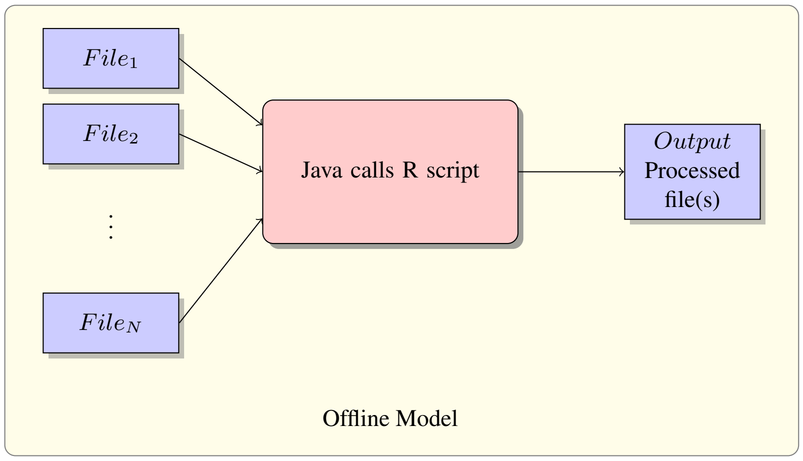


Fig. 1. The architecture of the local (offline) system

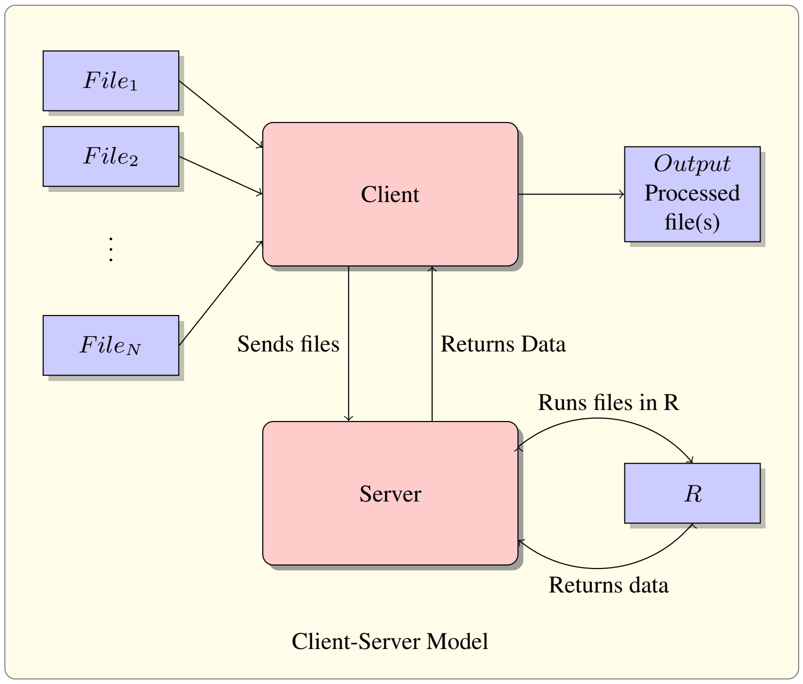


Fig. 2. The architecture of the networked system

4 Development

We needed a way to access R via java and in order to do so an R script was written after analyzing the supplied R algorithm (Shiny's server file) which was built upon Shiny. Shiny applications consist of two main files, the UI and the server (the name might be misleading as it serves only as an indicator of what the UI should do).

The R script can be found in the appendices at the end of this paper as it isn't particularly important since it depends on a lot of other code and doesn't directly affect how this work is distributed.

We were now able to develop a local service that runs offline and works as follows:

- Receive a file or files via input
- Pass them to the R script
- Parse the output
- Display or store the data.

4.1 Running the local NER service

First, arguments are enabled for the R script and packages loaded (although messages are suppressed for output purposes), then it runs for all files in the directory (the file(s) received in input is/are moved to a temporary due to how the algorithm operates and then cleaned when the process is over) and the output is saved in a table that is later caught via a BufferedReader which can now be parsed and analyzed/stored.

This script is called as an argument in Rscript which has the input file as an argument for itself and is executed as a child process in our Java implementation. An example would be:

```
iamfuzzeh@fuzzbox:src$ java Offline
File directory ?
/home/iamfuzzeh/workspace/Projecto/src/tests/CART.236.txt

"Paragraph","Sentence","lista"
"1","1","Bruno Junqueira"
"1","1","Fórmula Mundial-CART"
"1","1","Bruno Junqueira"
"1","1","Fórmula Mundial-CART"
"1","2","Bruno Junqueira"
"1","2","GP de Milwaukee"
"1","3","Ryan Hunter Reay"
"1","4","Patrick Carpentier"
"1","4","Michel Jourdain Jr"
"1","6","Mario Haberfeld"
"1","6","Alex Sperafico"
"1","7","Bruno Junqueira"
"1","7","GP de Portland"
```

```
"1","7","Sebastien Bourdais"
"1","8","Paul Tracy"
"1","9","Mario Haberfeld"
"1","9","Alex Sperafico"
"1","10","Bruno Junqueira"
"1","10","Patrick Carpentier"
"1","10","Sebastien Bourdais"
"1","11","Mario Haberfeld"
"1","11","Alex Sperafico"
"1","13","Aeroporto de Cleveland"
```

Automated Counterpart Being as a local, offline, version was now implemented we could alter it and create a slight variation that immediately stores data in a separate text file. This can be useful for several reasons as we no longer need to iterate over the java execution and can now chose what to do with the data.

As a result this alternate implementation (which is available as well the prompt one) eliminates the need for an interface or a delayed communication with the program (prompts for how many files, directory or location) by simply receiving them as an argument when being called. An example, in which the output is stored in the directory where it's executed, would be:

```
iamfuzzeh@fuzzbox:src$ java Offline /home/iamfuzzeh/
workspace/Projecto/src/tests/CART.236.txt
Done.
```

4.2 Client-Server

We created a server and client (both in java) in order to also have a networked solution, the server merely calls the R Script with the files received from the client as input. They operate as follows:

– Server starts
– Client connects :

```
Type Message ("Bye." to quit)
Available commands:
  Bye. :- Quits program
  Dir dir:- Uploads all files in directory,
   example - dir ./tests
  File dir :- Uploads a file,
   example - dir ./tests/test1.txt
```

– Client sends file(s)
– Server receives and runs them through the script
– Server outputs to client file/text with extracted data.

Implementing a Server for file transfer In order to create a java server we must first begin by creating a ServerSocket object that will listen on a specific port, which has to be enclosed in a try statement. If the server successfully binds its port the Accept object becomes available and a client socket can now be created (Socket s in the below example). The Accept method itself returns a socket that's bound to the same local port but bears both the remote address and port of the client[9] and we can then work upon this connection via InputStream and OutputStream. The benefit of using such objects over BufferedReader is that the content of the file may not be a standard character (any non text file) and will work correctly with Input/OutputStream.

```
    @Override
    public void run() {
        try {
        ServerSocket serverSocket = new ServerSocket(PORT);

         while (true) {
            Socket s = serverSocket.accept();
            saveFile(s);
        }
        } catch (Exception e) { e.printStackTrace(); }
    }
```

Manipulating the client connection (via its socket) is a matter of opening a FileOutputStream with the file as an argument, reading the file byte by byte via a self made buffer (being this an array of bytes) and writing to an output file after each byte. The main part is exemplified below:

```
// 1. Read file name.
        Object o = ois.readObject();
        String dir = "/home/iamfuzzeh/workspace/Projecto/
        src/received/" + o.toString();

        if (o instanceof String) { fos =
        new FileOutputStream( dir); }
        else { throwException("Something is wrong"); }

        // 2. Read file to the end.
        Integer bytesRead = 0;

        do {
            o = ois.readObject();
            if (!(o instanceof Integer)) {
            throwException("Something is wrong"); }

            bytesRead = (Integer)o;
            o = ois.readObject();
            if (!(o instanceof byte[])) {
            throwException("Something is wrong"); }

            buffer = (byte[])o;

            // 3. Write data to output file.
            fos.write(buffer, 0, bytesRead);
```

Multiple Connections Making a server multiplex can be important and so in fact is in our case since a networked solution implies a server and multiple clients, having only client connect at a time implies a delay between a request and its answer. This also eventually leads up to a queue which is not intended.

In order to do so threads must be managed to allow multiple client requests to come into the same port and, consequently, into the same ServerSocket. This signifies that run() would loop forever and in each new connection a saveFileThread() would be called. A very light example in pseudo-code:

```
while (true) {
    accept a connection;
    create a thread to deal with the client;
}
```

Implementing a Client for file transfer Admitting that the server is already running the first thing the client must do is connect to it via a socket with the specified address and port, this must be enclosed in a try statement. In order to transfer a file InputStream and OutputStream need again to be used but it's less complicated to achieve as using the writeObject method from OutputStream will leave only the concern of sending the proper length (size of the file). Again a buffer is created via a byte array. The file is specified via console argument and the main part of this code can be found below:

```
try {
        fileName = args[0];
        } catch (Exception e) {
        System.out.println("Enter the name of the file");
        Scanner scanner = new Scanner(System.in);
        String file_name = scanner.nextLine();

        File file = new File(file_name);
        Socket socket = new Socket("localhost", 3332);
        ObjectInputStream ois = new ObjectInputStream(
        socket.getInputStream());
        ObjectOutputStream oos = new ObjectOutputStream(
        socket.getOutputStream());

        oos.writeObject(file.getName());
```

```
FileInputStream fis = new FileInputStream(file);
byte [] buffer = new byte[Server.BUFFER_SIZE];
Integer bytesRead = 0;

while ((bytesRead = fis.read(buffer)) > 0) {
    oos.writeObject(bytesRead);
    oos.writeObject(Arrays.copyOf(buffer,
    buffer.length));  }
```

5 Results

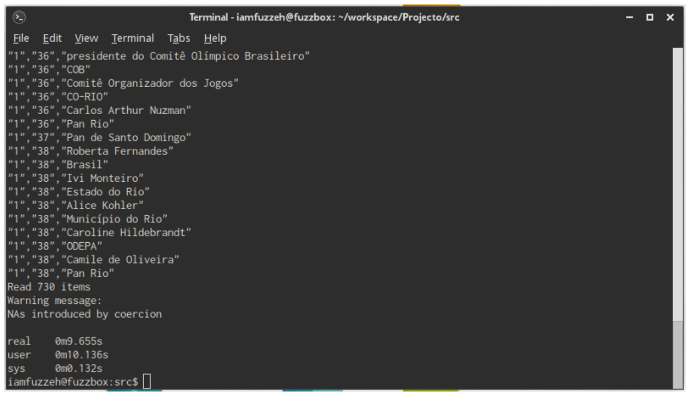
For all of this to work all versions had to be accounted for so each package in each specific version was stored as well as the particular R version (3.0.2) that works in our situation. Everything will be saved upon a single, for distribution, compressed file along with a few installation scripts. The pack assumes the following form:

- Pack
  - Packages
    - Dependencies
    - Dependent Packages
    - Non-Dependent Packages
  - R 3.0.2
    - Installation files.
  - PAMPO
    - Local Version - UI
    - Local Version - Non-UI
    - Client
    - Server
  - Readme
  - package-script

Different users and or installation locales will have different procedures; a hosted server will comprise of installing R, running the package installation script, opening the server and passing the client throughout the network; a local (networked user) will only need to communicate with the server via supplied client, not needing any installing or concern about the structure; a local (solo) user will have to follow the same procedure regarding installation but can utilize either the UI or console version as to gather data. This semblance of solutions allow for versatility given the context in which this work is applied and solves two main concerns: allowing everyone to use it regardless of computer savviness and not limiting the works potential to a specific scenario.

Currently the local and the networked solutions have a similar run time, being the networked one slightly slower due to the file transfer, although this might change after we find a way to keep the packages loaded permanently in the server. Comparing a small number of files to a large file depends on the number of entities available and number of lines; a similar sum of words on the small files to the large file will take approximately the same time.

As an example a file with 781 words (this particular case presents 84 entities) can be takes about 10 seconds to collect entities, as shown in [Figure 3]


Terminal - iamfuzzeh@fuzzbox: ~/workspace/Projecto/src
File Edit View Terminal Tabs Help

```
"1","36","presidente do Comitê Olímpico Brasileiro"
"1","36","COB"
"1","36","Comitê Organizador dos Jogos"
"1","36","CO-RIO"
"1","36","Carlos Arthur Nuzman"
"1","36","Pan Rio"
"1","37","Pan de Santo Domingo"
"1","38","Roberta Fernandes"
"1","38","Brasil"
"1","38","Ivi Monteiro"
"1","38","Estado do Rio"
"1","38","Alice Kohler"
"1","38","Município do Rio"
"1","38","Caroline Hildebrandt"
"1","38","ODEPA"
"1","38","Camile de Oliveira"
"1","38","Pan Rio"
Read 730 items
Warning message:
NAs introduced by coercion

real    0m9.655s
user    0m10.136s
sys     0m0.132s
iamfuzzeh@fuzzbox:src$
```

Fig. 3. Execution example of a file with 84 entities returned

6 Conclusion

We now have a package that can be distributed which allows for both a local (offline) and networked solution and in turn allows us to deploy the software via web as well. There are both strengths and weaknesses to this approach of allowing multiple solutions: Due to time and knowledge constraints the installation part of the process can still be further automated, both client/server can be improved (and do in deed need so in security terms) and although all of this allows for a versatile application of the NER software (PAMPO) as a stand-alone application (or as part of a bigger project) there is room for improvement as there are some features that can still be implemented (even if not necessarily needed): supporting multiple file extensions at the same time, zipped archives, multiple zipped archives, etc.

7 Acknowledgments

I'd like to thank Conceição Rocha for taking her time to help me understand the version management issue and making available all the of the package information, as well as clearing some doubts about the R language itself. I also appreciate the help Alípio Jorge extended in guiding me through this work, paper, and for supplying reading material about the subject (Named Entity Recognition and Text Mining).

References

[1] Rahul, Sharnagat. *Named Entity Recognition: A Literature Survey*. June 30, 2014.
[2] David Nadeau, Satoshi Sekine. *A survey of named entity recognition and classification*. National Research Council Canada / New York University.
[3] Jing, Jiang. *INFORMATION EXTRACTION FROM TEXT*. Singapore Management University. jingjiang@smu.edu.sg.

[4] Marios Skounakis - marios@cs.wisc.edu, Mark Craven - craven@biostat.wisc.edu, Soumya Ray - sray@cs.wisc.edu. *Hierarchical Hidden Markov Models for Information Extraction* University of Wisconsin.

[5] Satoshi Sekine. *Named Entity: History and Future* New York University.

[6] Andrew Borthwick. *A Maximum Entropy Approach to Named Entity Recognition* New York University. September, 1999

[7] Hiroyasu Yamada, Yuji Matsumoto. *Japanese Named Entity Extraction with Redundant Morphological Analysis* Japan Advanced Institute of Science and Technology, Nara Institute of Science and Technology. 2003.

[8] Andrew McCallum, Wei Li. *A Note on Semi-Supervised Learning using Markov Random Fields* University of Massachusetts Amherst. February 3, 2004.

[9] Oracle. *Writing the Server Side of a Socket*. `https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html`

8 Appendix

8.1 R script

```
args <- commandArgs(trailingOnly = TRUE)

suppressMessages( require(tm) )
suppressMessages( require(tools) )
suppressMessages( source("/home/iamfuzzeh/workspace/
Projecto/src/R
/Helpers_Ent.R") )
suppressMessages( source("/home/iamfuzzeh/workspace/
Projecto/src/R
/Helpers_Exp.R") )
suppressMessages( source("/home/iamfuzzeh/workspace/
Projecto/src/R
/Helpers_Desamb4.R") )
suppressMessages( source("/home/iamfuzzeh/workspace/
Projecto/src/R
/Helpers_POS.R") )

    inFile <- scan(args[1], character(0))
    if(is.null(inFile))return(NULL)
    informat<-file_ext(args[1])
    path<-dirname(args[1])
    filename<-basename(args[1])
    if(informat=="pdf"){
      toprint<-Corpus(DirSource( path ),readerControl=
      list(reader=readPDF(), language="pt",id="id1"))
      files <- list.files(path = getwd(), recursive
      = TRUE)
      toprint2<-cleanedtext(toprint)
    } else{
      toprint<-Corpus(DirSource( path ),readerControl=
      list(language="pt",id="id1"))
      files <- filename
      if(informat=="txtpdf"){
      toprint2<-cleanedtext(toprint) }
      else{ toprint2<-toprint }
    }

      n=length(toprint2)
      tab_entity_desambf<-c()
      for(i in 1:n){
        toprint3<-entitytext(toprint2[[i]])
        tab_entity_tag<-entityPOS(toprint3)
        tab_entity_desamb<-entitydesamb4(tab_entity_tag)
        aux<-as.numeric(attr(toprint2[[i]],"ID"))+1
        tab_aux<-merge(files[aux],tab_entity_desamb [,1])
        tab_entity_desamb2<-cbind(tab_aux,
        tab_entity_desamb[,2:3])
        table_entity_doc<-cbind(tab_entity_desamb2,
        tab_entity_desamb[6])
        if(i==1){ tab_entity_desambf<-table_entity_doc }
        else{tab_entity_desambf<-rbind(tab_entity_desambf,
        table_entity_doc)}
      }

      toprint3<-entitytext(toprint2)
      tab_entity_desambf<-c()
      tab_entity_tag0<-entityPOS(toprint3)
      nlen<-length(tab_entity_tag0[,1])
      npa<-tab_entity_tag0[nlen,1]
      for(j in 1:npa){
        dim<-which(tab_entity_tag0$Paragraph==j)
        tab_entity_desamb<-entitydesamb4(
        tab_entity_tag0[dim,])
        tab_entity_desambf0<-cbind(tab_entity_desamb[,1:2],
        tab_entity_desamb[6])#3 em vez de 2
        tab_entity_desambf<-rbind(tab_entity_desambf,
        tab_entity_desambf0)
      }

    write.table(tab_entity_desambf, "", sep = ",", qmethod=
    "double",row.names = FALSE)
```