Universitatea "Alexandru Ioan Cuza" din Iași
Facultatea de Informatică

Named Entity Recognition

For The Romanian Language

Drumea Alexandru-Daniel
Sesiunea: Iulie, 2019
Coordonator științific: Prof.Dr. Cristea Dan

Contents

1 Introduction

1.1 Abstract

Computer science is the study of processes that interact with large volumes of data, that can be represented in the form of programs. It enables the use of algorithms to manipulate, compute, store and communicate digital information. The above mentioned field is relatively new in the scientific community, as it employs the use of computers in order to ease human labour and aid in completing activities which otherwise would take significantly longer time to complete.

Such activities include natural language processing, a topic of research which is concerned with the interactions between computers and human (natural) language, in particular how to program computers to process and analyze large quantities of natural language data. One of the challenges in this field is "named entity recognition", a topic of great research in modern times.

Named entity recognition is a subtask of information extraction that seeks to locate and classify mentions of known named entities in unstructured texts into pre-defined categories, such as person names, organization names, geographical locations and miscellaneous such as date and time, geo-political relations etc.

Presently, conferences such as CoNLL address the issues and challenges of the task of classifying named entities and award prizes for papers that have most contributed to perfecting human language analysis in general and NER in particular.

The scientific community in this field has currently reached a high level precision and recall for much studied languages such as English and German. This can stand as a base level on which to place other more sophisticated technologies.

Techopedia, the leading online technical dictionary, explains "Named-entity recognition is a state-of-the-art intelligence system that works with nearly the efficiency of a human brain. NER systems are structured in such a way that they are capable of finding entity elements from raw data and can determine the categories to which the elements belong. The system reads the sentence and highlights the important entity elements in the text. A NER system might be made sensitive to different types of entities depending on the project. This means that the NER system designed for one project may not be reused for another task. Similarly, NER faces many challenges which include the extraction of correct information for specific but closely related categories". In addition to those previously stated, the problem of recog-

nizing named entities may be broken down, conceptually, in two problems, which are the detection of names and classification of said names by the type of entity they refer to (e.g. person, organization, location and miscellaneous).

In order to evaluate the results of such systems, one must define the concepts of precision, recall and F-measure functions, as defined by academic conferences such as CoNLL.

Precision is the number of predicted entity name spans that line up exactly with spans in the gold standard evaluation data. I.e. when [Person Hans] [Person Blick] is predicted but [Person Hans Blick] was required, precision for the predicted name is zero. Precision is then averaged over all predicted entity names. Recall represents the number of names in the gold standard that appear at exactly the same location in the predictions, while the F-measure (or the F1 function) is the harmonic mean of these two.

The technology currently available for this type of information extraction is comprised of Python libraries and Java applications, which offer support for the aforementioned languages (German and English), with great F-measure scores. To describe the available resources for said languages, one needs to understand the Conditional Random Fields Classifiers, which are described Sutton and McCallum (2010) paper, "An Introduction to Conditional Random Fields" as follows "A solution(n.r Named entity recognition) to this problem is to model the conditional distribution $p(y|x)$ directly, which is all that is needed for classification. This is a conditional random field (CRF). CRFs are essentially a way of combining the advantages of classification and graphical modeling, combining the ability to compactly model multivariate data with the ability to leverage a large number of input features for prediction. The advantage to a conditional model is that dependencies that involve only variables in x play no role in the conditional model, so that an accurate conditional model can have much simpler structure than a joint model". A corpus is a large data set consisting of bodies of text, which are either annotated or not. Seen from the perspective of building a NER system, a corpus should include annotations of name entities, together with their classes, out of which a learning system can be trained to recognise name entities.

Stanford NER is a Java implementation of a Named Entity Recognizer. Stanford NER is also known as a CRF Classifier. The software provides a general implementation of the linear chain Conditional Random Field (Sutton *et al*, 2011) sequence models. That is, by training ones own models on labeled data, one can actually use this code to build sequence models for NER or any other task. The English corpus is included in the Stanford NER package and will correctly tag above 90% of named entities.

NLTK (Natural Language Toolkit) is a "Python package providing a series of natural language corpora and APIs of wide varieties of NLP algo-

rithms and it works in three stages: word tokenization, part-of-speech tagging and named entity recognition"("Introduction to NER" article, 2018).

SpaCy is a natural language processing library in Python, known for its industrial-strength classification capabilities. "SpaCy supports numerous entity types and can be trained on a multitude of languages from scratch"("Introduction to NER" article, 2018).

All of the technological solutions above are benchmarks for either the independent, hobbist researcher or for the international community restlessly working to improve the processing of human language with the use of computers. However, even with a language-independent approach, getting very good results on these platforms needs large quantities of training data, not easily achievable in languages having rather scarce annotated resources. Among these - Romanian, despite the fact that some offer multi-language and language-independent support.

1.2 Motivation

The writer of this paper is highly motivated by the problem of creating a model of NER for the Romanian language, as it may serve as a template for other languages with Latin roots such as French, Spanish, Portuguese etc. The steps taken to achieve the task at hand include the use of the aforementioned technologies, in order to create a web application which provides one with statistics of the training process, pre-trained corpora as well as a classifier for unlabeled texts.

In addition to the added flexibility of the topic and of the more nuanced variety of expressions, the Romanian language, being a much less known language than English benefits from a much smaller international research community to create models, gazetteers, training data, or named entity recognition software. The previous argument is to be added to the author's motivation to create a template for the aforementioned research community in the field of Natural Language Processing in general and Named Entity Recognition in particular.

NER models are used in a wide range of applications in the field of Natural Language processing and Information Retrieval. One such example is in automatically summarizing CVs, which consists of summarizing a CV, thus enabling Human Resources departments to quickly compile large volumes of personnel data. Running a NER model on a large collection of articles once and storing the entities associated with them permanently, would result in optimization of a search engine, which would not have to search the whole collection for the entered entity that occurs in a query each time a query is initiated. Another application of NER is simplifying customer support

as it can be used in recognizing relevant entities in customer complaints and feedback such as Product specifications, department or company branch details, so that the feedback is classified accordingly and forwarded to the appropriate department responsible for the identified product.

2 Theoretical Considerations, State-of-the-Art Technologies and Data Formats

2.1 Data Formats Used in the Current Thesis

This chapter describes a series of theoretical notations and the metadata used in the process of named entity recognition. The files used contain specialized annotations used by the compiler in the categorization of the entities and prediction of their type.

The CoNLL-U Plus Format can encode any kind of annotation using a combination of the sentence-level comments and the MISC attributes, this format can have any number of non-zero columns, as well as a sentence-level comment (depicted by the use of # at the beginning of the line, used in order to either set a sentence id, or to write the sentence as raw text). This extends the CoNLL-U format which is widely used in Natural Language Processing, but is impractical in Named Entity Recognition due to its fixed number of columns (ten).

QuoVadis is a corpus displaying semantic relations in free text. As stated in the paper, "Developing a lexical-semantic knowledge base to be used in Natural Language Processing (NLP) applications is the main goal of the research described in this chapter (n.r. paper). Such resources are not available for many languages, mainly because of the high cost of their construction. The knowledge base is built in such a way as to facilitate the training of programs aiming to automatically recognize in text entities and semantic relations.[...] it includes annotations for the spans of text that display mentions of entities and relations, for the arguments (poles) of the relations, as well as for the relevant words or expressions that signal relations"(Cristea *et al*, 2014). The data that comprises this corpus will be used in the training of the NER tagger this paper intents to build. The layers of annotation in the aforementioned corpus are as follows: segmentation at text level (marks the sentence boundaries in the raw book text), tokenization (demarcates words or word compounds, but also numbers, punctuation marks, abbreviations etc), part-of-speech tagging, lemmatization (determines lemmas of words) and noun-phrase chunking, which explores the previously generated data and adds information regarding noun phrase boundaries and

their head words. In the scope of training the NER, the application to be presented in the following sections uses the part-of-speech tagging provided by the QuoVadis corpus, as well as the noun phrase chunking to create tab separated values (.tsv), a file to be trained by the Stanford CRF classifier. The data of the aforementioned corpus, which is initially comprized in an .xml type file, will be extracted and each "entity" tag will be added to the .tsv file, as highlited in Figure 1.

```
<S id="1" offset="0">
    <W EXTRA="NotInDict" LEMMA="Henryk" MSD="Rg" POS="ADVERB" head="0" id="1" offset="0">Henryk</W>
  <W Case="oblique" Definiteness="no" EXTRA="NotInDict" Gender="feminine" LEMMA="Sienkiewicz" MSD="Npfpon"
  Number="plural" POS="NOUN" Type="proper" deprel="a.adj." head="4" id="2" offset="7">Sienkiewicz</W>
  <W Case="oblique" Definiteness="no" EXTRA="NotInDict" Gender="feminine" LEMMA="Quo" MSD="Npfpon"
  Number="plural" POS="NOUN" Type="proper" deprel="a.adj." head="4" id="3" offset="19">Quo</W>
  <W Case="direct" Definiteness="no" EXTRA="NotInDict" Gender="masculine" LEMMA="vadis" MSD="Ncmsrn"
  Number="singular" POS="NOUN" Type="common" head="0" id="4" offset="23">vadis</W>
  </S>
```

Figure 1: Data of the QuoVadis corpus

RONEC (Dumitrescu *et al*, 2018) is a corpus of annotated data consisting of 5127 sentences, classified in 16 classes with a total of 26376 entities. It is used as a template for the training files, as well as for the named entity data it contains. From this file, the classificator will use the template of the file (CoNLL-UP format), as well as the data classified as person, organization, geographical location, or miscellaneous. The RONEC.conllup file is depicted in figure 2.

```
# sent_id = 5
# text = Flankerul selecționatei de rugby a României și al echipei franceze din Grenoble, Florin Corodeanu, și-a reluat antrenamen
1   Flankerul   Flankerul   PROPN   Ncmsry   _   19  nsubj   _   _   1:PERSON
2   selecționatei   selecționată   NOUN   Ncfsoy   Case=Dat,Gen|Definite=Def|Gender=Fem|Number=Sing   1   nmod   _   _   *
3   de   de   ADP Spsa   AdpType=Prep|Case=Acc   4   case   _   _   *
4   rugby   rugby   NOUN   Ncms-n   Definite=Ind|Gender=Masc|Number=Sing   2   nmod   _   _   *
5   a   al   DET Tsfs   Gender=Fem|Number=Sing|Poss=Yes|PronType=Prs   6   det _   _   *
6   României   României   PROPN   Npfsoy   Case=Dat,Gen|Definite=Def|Gender=Fem|Number=Sing   4   nmod   _   _   2:GPE
7   și   și   CCONJ   Crssp   Polarity=Pos   9   cc   _   _   *
8   al   al   DET Tsms   Gender=Masc|Number=Sing|Poss=Yes|PronType=Prs   9   det _   _   *
9   echipei   echipă   NOUN   Ncfsoy   Case=Dat,Gen|Definite=Def|Gender=Fem|Number=Sing   2   conj   _   _   *
10  franceze   francez ADJ Afpfson Case=Dat,Gen|Definite=Ind|Degree=Pos|Gender=Fem|Number=Sing 9   amod   _   _   3:NAT_REL_POL
11  din din ADP Spsa   AdpType=Prep|Case=Acc   12  case   _   _   *
12  Grenoble   Grenoble   PROPN   Np   _   9   nmod   _   SpaceAfter=No   4:GPE
13  ,   ,   PUNCT   COMMA   _   14  punct   _   _   *
14  Florin   Florin   PROPN   Np   _   1   appos   _   _   5:PERSON
```

Figure 2: Data of the RONEC corpus

After the processing of the data in the two aforementioned corpora, the custom file to be used in the process of training, will consist of roughly 195000 entries, with their named entity recognition label, which is detailed below.

In Figure 3, one can observe the form of the processed corpus is of the form entity ¡TAB¿ type. The types of entities are: person, organization,

```
Născut  O
în  O
1369    MISC
într-   O
un  O
orășel  O
la  O
75  MISC
km  O
de  O
Praga   GPE
,   O
el  O
a   O
ajuns   O
decanul O
Facultății  ORGANIZATION
de  O
Filosofie   ORGANIZATION
din O
Praga   GPE
,   O
iar O
apoi    O
rectorul    O
Universității   ORGANIZATION
Carol   PERSON
din O
acest   O
oraș    O
.   O
```

Figure 3: Data of the processed corpus

o (others, which describes an entity which is not a named one), GPE which depicts countries, cities, states, as well as MISC, which will cover geo-politic entities, date and time, facilities and works of art.

2.2 Deploying Web Applications Using Python

As the result of the application of the theoretical considerations and algorithms presented in this thesis will be comprised in a web application, it is very important to note some of the frameworks and libraries to be used.

One of the applications which provide one with the best ease of use is called Flask.

Flask is a microframework for Python, that comes with built-in development, server and debugger, integrated unit testing support, RESTful request dispatching, is Unicode based, and is extensively documented.

A use-case of this framework is depicted in the figure below.

The aforementioned use-case is as follows: a page that consists of three elements, a navigation bar, contents and the footer respectively. To be noted is that the page will present in the content section of its body a text-box in which the user will insert a sentence that will be sent to the server, the server catching that will execute a Python function using the parameters

```python
from flask import Flask
from flask import Flask, request, render_template
import os
import Vote

app = Flask(__name__)
PEOPLE_FOLDER = os.path.join('static', 'images')
app.config['UPLOAD_FOLDER'] = PEOPLE_FOLDER

@app.route('/')
def home():
    menu = os.path.join(app.config['UPLOAD_FOLDER'], 'menu.png')
    index = os.path.join(app.config['UPLOAD_FOLDER'], 'index.png')
    sect1 = os.path.join(app.config['UPLOAD_FOLDER'], 'sectiunea1.png')
    footer = os.path.join(app.config['UPLOAD_FOLDER'], 'footer.png')
    submit = os.path.join(app.config['UPLOAD_FOLDER'], 'SUBMIT.png')
    if len(request.args)!=0:
        string = "<p>" + str(Vote.vote(request.args['sentence'])) + "</p>"
        return render_template('index.html', menu_img = menu, index_img = index, sect1_img = sect1, footer_img = footer, submit_img = submit, data = string)

    return render_template('index.html', menu_img = menu, index_img = index, sect1_img = sect1, footer_img = footer, submit_img = submit)


if __name__ == '__main__':
    app.run(debug=True)
```

of the request. Finally, the server will use the results of said Python function and will alter the front-end accordingly.

2.3 The Stanford Named Entity Recognition Software (Sutton *et al*, 2015)

2.3.1 The Conditional Random Field Algorithm, Theoretical Considerations of the Stanford Software

It is denoted as $x = (x_1, \ldots, x_n)$ as the input sequence, i.e. the words of a sentence and $s = (s_1, \ldots, s_n)$ the sequence of output states i.e. the named entity tags. In conditional random fields, the conditional probability $p(x_1, \ldots, x_n | s_1, \ldots, s_n)$ is modeled. This feat is accomplished by defining a feature map function $\Phi(\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}, \mathrm{s}_1, \ldots, \mathrm{s}_\mathrm{n}) \in \mathrm{R}^\mathrm{d}$, this maps the input sequence x paired with an entire state sequence s to some d-dimensional vector. Then one can model the probability as a log-linear model with the parameter vector $\omega \in \mathrm{R}^\mathrm{d}$:

$$p(s|x;\omega) = \frac{exp(\omega \cdot \Phi(x,s))}{\sum_{s'} exp(\omega \cdot \Phi(x,s'))},$$

where s' ranges over all possible output sequences. For the estimation of ω, it is assumed there is a set of n labeled examples $(\mathrm{x}^\mathrm{i}, \mathrm{s}^\mathrm{i})$, where i ranges between 1 and n. Now, the log-likelihood function is computed:

$$\sum_{i=1}^{n} log(p(s^\mathrm{i}|x^\mathrm{i},\omega))$$

The parameter vector ω^* is then estimated as:

$$\omega^* = argmax_{\omega \in \mathrm{R}^\mathrm{d}} L(\omega)$$

The statement above means that each sequence will be tagged with the tagging associated with the highest probability.

Now, the most likely tagging of a sentence x may be computed as:

$$s^* = argmax_s p(s|x, \omega^*)$$

Having understood the aforementioned algorithm, one may apply said algorithm both for part-of-speech tagging, as well as for named entity recognition. This algorithm is most prominently used by the Stanford NLP team as the nucleus for their named entity recognition software, on which this thesis has based its research into NER tagging in the Romanian language.

2.3.2 The Operation of the Stanford Named Entity Recognition Software

This model uses the Conditional Random Field Algorithm in order to predict the features of an unknown word.

All word features are stored in a window, that being comprised of the current word, previous word, next word, as well as orthographic features such as capitalization and overall form of a word (Jenny is generically depicted as Xxxxx, while AH1-N1 as XX#-X#). Additionally, features of high importance are also stored, features such as prefixes and suffixes, which are also stored in a window (Jenny is depicted as <Jen, <...ny> >), as well as label sequences as feature conjunctions.

This method of recognition also employs the use of the Distributional Similarity Features on a large unannotated corpus, which will provide context of named entities, which will then be clustered on how similar their distributions are within the corpus in order to combat sparsity.

The Stanford Named Entity Recognition Software is highly used in the academical due to its multi-language and language-independent support, as well as to its speed and its open-source availability.

2.4 The SpaCy Entity Recognition Module

SpaCy is a free, open source library that has made Natural Language Processing (NLP) much simpler in Python.

It provides users with a statistical system which is highly efficient for named entity recognition, which can assign labels to groups of tokens which are contiguous. It can accurately recognize a wide variety of named entities or numerical entities, and can enable the addition of arbitrary classes to the model, by training it to enable bespoke examples.

Because this program is widely used in the industry, the company has not yet provided many technical specifications, other than it uses convolutional neural networks (CNN).

A convolutional neural network is a class of deep neural networks, which are used in a wide variety of tasks in deep learning, ranging from analyzing visual imagery, to sentence classification and named entity recognition.

"Convolutional neural networks (CNN) utilize layers with convolving filters that are applied to local features"(Kalchbrenner *et al*, 2014). Originally invented for computer vision, CNN models have subsequently been shown to be effective for NLP and have achieved excellent results in semantic parsing (Yih *et al*, 2014), sentence modeling (Liu *et al*, 2015), and other traditional NLP tasks.

Convolutional Neural Networks are used in Natural Language Processing in general and Named Entity Recognition in particular because of their ability to extract features. These networks are applied to embedding vectors of a given sentence with the aim of extracting useful features such as relationships between words that are closer together in a sentence, or the relation between sentences. Moreover, Convolutional Neural Networks will capture the relationships and general meanings of a sentence, which also represents an important advantage for good classification.

The use of CNNs in NER is very similar to their use in image classification. Given a sequence of words $w_{1:n} = w_1,\ldots,w_n$, where each is associated with an embedding vector of dimension d.

A 1D convolution of width-k is the result of moving a sliding-window of size k over the sentence, and applying the same convolution filter or kernel to each window in the sequence, i.e., a dot-product between the concatenation of the embedding vectors in a given window and a weight vector u, which is then followed by a non-linear activation function g.

Considering a window of words $w_i,\ldots,w_{i+k}$, the concatenated vector of the ith window is then:

$$x_{\mathrm{i}} = [w_{\mathrm{i}}, , w_{\mathrm{i+1}}, \ldots, w_{\mathrm{i+k}}] \in R^{\mathrm{k}\,\cdot\,\mathrm{d}}$$

The convolution filter is applied to each window, resulting in scalar values r_i, each for the ith window:

$$r_{\mathrm{i}} = g(x_{\mathrm{i}} \cdot u) \in R$$

As previously stated in this thesis, SpaCy is an industrial-strength classifier, meaning that it is currently used by companies in order to achieve different tasks of a large scope, tasks which include product management, human resource management as well or search engine tuning. According to

the lead-developer of SpaCy, companies such as Airbnb (which is an accomodation booking company), Quora (an online technology forum) or Allen Institute for Artificial Intelligence currently employ this software to extend and improve their services.

2.5 The Theoretical Foundations of the Gazetteer Method in Named Entity Recognition

2.5.1 The Brute Force Method

The first step in implementing this method implies the creation of a different list of items for each entity one desires to recognize, on which search operations will be applied in a later step in order to classify names.

Secondly, a large corpus of data needs to be appended to each list in order to achieve a great level of precision and recall for each of the named entities.

This method is quite restrictive due to the difficulties of comprising a corpus large enough to accomodate the varieties of named entities one may encounter, as well as the difficulties in keeping it up to date automatically. Moreover, the ambiguity resolution factor (the interpretation of metaphors, entities of the same name etc) is rather low, a fact which also leads to low precision.

Despite its disadvantages, the Gazetteer Method is a highly powerful method in named entity recognition, due to its speed, ease of manual improvement and relative ease of use.

2.5.2 Improving the Brute Force Method

In order to improve the method presented in the previous subsection, one must use a part-of-speech tagger as to extract only words that are proper nouns, thus optimizing the speed of the search algorithm and improve the overall performance. After having taken these steps, the search algorithm may be applied on each proper noun which is an output of the part-of-speech tagger.

To achieve those previously stated, the application presented in this thesis uses a part of the Stanford NER model, the Stanford POS Tagger. This tool applies the Conditional Random Field algorithm on the set of untagged data for obtaining the Part-Of-Speech Tagging necessary. The input data for the POS Tagger is comprised of a Romanian language model available on the Stanford platform, said model containing over 125000 tagged parts of speech.

2.6 State-of-the-Art Methods with Respect to The Romanian Language

The aforementioned methods of tagging named entities are widely used by companies and researchers alike, being highly appreciated for their reliability and overall correctness. They have proved to produce good F-measure score in languages which are highly restrictive with respect to the topic of the sentence.

For example, in English, one may represent a noun-phrase by the following regular expression:

$$pattern =' NP : < DT >? < JJ > * < NN >',$$

in which NP depicts noun-phrase, DT is a determiner, JJ an adjective and NN a noun. The previously stated regular expression translates as: a noun phrase, NP, should be formed whenever the chunker finds an optional determiner, DT, followed by any number of adjectives, JJ, and then a noun, NN. One needs a noun phrase in order to more efficiently classify words as chunks, as well as to get a better understanding of the context and reduce the number of entities to be classified. For example in the following sentence: *"Google, the giant American company, has been fined* $*5.1 billion dolars."*, the chunk "the giant American company" is a noun phrase which will offer the named entity recognition software the context needed to classify Google as an organization.

Meanwhile, in the Romanian language, the above stated sentence may be translated as *"Google, giganta companie americana, a fost amendata cu 5.1 miliarde de dolari."*, which is a natural context having a natural topic of the sentence. One may observe that the order in which the adjectives are placed is highly flexible, ranging from the above stated <DT>? <JJ> $*$ <NN><JJ> $*$ to <DT>? <NN><JJ> $*$("compania giganta americana"), a fact which is much harder to represent using regular expressions because there exist much more variations in the topic of a sentence rather than in English.

3 The Romanian Named Entity Recognition Software

The object of the current thesis is to combine the three methods described in the previous chapter in a reliable and fast Named Entity Recognition Software for the Romanian language in the form of a web application.

The resources that will be used are to be detailed and exemplified in the current chapter, as well as the application logic and the results of the processing.

3.1 Data Preprocessing

In order to obtain fast, reliable data, the corpora described in the the first section of the previous chapter in the current thesis has to be preprocessed in order to obtain a ConLL-UP formatted file, the one that is needed for the training of the customly-built named entity tagger described in this thesis.

Firstly, one has to extract the relevant data from the RONEC file, in order to comprise the training file for the Stanford NER software. This feat is to be accomplished by making use of the Python script in Figure 4. After data has been processed, the ronec.conllu is presented as in Figure 5.

```
import os

f = open("../Resources/ronec.conllu", 'r', encoding='utf8').read()

preprocessed = ""

for line in f.split('\n'):
    if len (line) > 1 and '#' not in line:
        tab = line.split('\t')
        if len(tab) >= 10:
            preprocessed += tab[1] + '\t' + tab[10][2:] + '\n'

```

Figure 4: Preprocessing the RONEC corpus

One may notice that the shape of the file to be used for training is not ideal at this point, as there are words without tagging, mainly those that are parts of noun phrases. In the interest of simplicity, the following Python script will be used in order to complete the file with the missing tags. One is inclined to see that if between two tagged words there are words which are not tagged, there is a high probability that the words between the two tags are of the same tag as the former. Errors are insignificant to the training, as the corpus contains 120000 entries.

This script will loop 500 times to ensure that the Romanian corpus is complete and contains no white spaces in the column assigned to the named entity type. This procedure is described in Figure 6.

```
Grupul  ORGANIZATION
Școlar
"
M.Eminescu
"
Jimbolia
(
205     NUMERIC_VALUE
)
,
Grupul  ORGANIZATION
Școlar
Sânnicolau
Mare
```

Figure 5: Output of the Python Script in Figure 4.

```python
def loop_forever():
    f = open ("../Resources/ronec.collu", 'r', encoding='utf8')

    data = f.read()
    lines = data.split("\n")
    f.close()

    # print(lines[672].split("\t"))
    to_write = ""
    for idx in range(0, len(lines)):

        if len(lines[idx].split('\t')) == 1:
            lines[idx] += '\t'
        if lines[idx].split("\t")[1] == '':
            to_write += lines[idx] + lines[idx-1].split("\t")[1] + '\n'
        else:
            to_write += lines[idx] + '\n'


    g = open ("D:/Anul_3/Lic/stanford/train/dummy-romanian-corpus.tsv", 'w', encoding='utf8')
    g.write(to_write)

for i in range(0, 500):
    loop_forever()
```

Figure 6: Completing the types of the entities which are part of noun phrases.

```
în  O
special O
Ucrainei    GPE
,   O
Georgiei    GPE
și  O
Republicii  GPE
Moldova GPE
,   O
unde    O
Moscova GPE
și- O
a   O
pierdut O
influența   O
după    O
revoluțiile O
oranj   O
.   O
Se  O
asigură O
condiții    O
deosebite   O
pentru  O
asimilarea  O
cunoștințelor   O
,   O
C.P.P.P.    ORGANIZATION
```

Figure 7: The output of the Python script in Figure 6.

One may proceed with the extraction of data from the QuoVadis corpus, with the mention that a python xml parser will be needed to extract the words containing the tag <entity>< /entity>, as well as to extract the "Type" attribute and the text of the entry in the .xml file.

```
<ENTITY ID="E000200000" TYPE="PERSON">
    <W Case="oblique" Definiteness="no" EXTRA="NotInDict" Gender="feminine"
    LEMMA="Petronius" MSD="Npfpon" Number="plural" POS="NOUN" Type="proper" deprel="sbj."
    head="7" id="5" offset="20">Petronius</W>
</ENTITY>
```

Figure 8: The exemplification of an "entity" type entry in the QuoVadis corpus.

In the aftermath of those above, a complete and comprehensive training file is comprised and ready to serve both as a gazetteer and an input for the Conditional Random Field classification algorithm.

3.2 Stanford Named Entity Recognition Method

Having comprised the training data, one is ready to apply the Stanford NER, by far the most reliable Natural Language Processing library on the market. The aforementioned package is written in Java, so one will need a proper Java Development Kit installed in order to run it.

Having prepared the environment needed, one needs to load the Stanford NER engine, available online for free for academic purposes.

The classification file compiled at the previous step (Subsection 3.1 Data Preprocessing) will now serve as a training file in the context of the program itself, as well as the two vectors (x - word vector, s - state vector) presented in Subsection 2.1.1 - "The Conditional Random Field Algorithm, Theoretical Considerations of the Stanford Software", the respective file is to be placed in the folder "./train" of the home folder of the Stanford classifier software. Moreover, the classifier software needs to be given a list of proprieties, most of which are fixed for the Named Entity Recognition, some of which may be changed. Said proprieties are presented in Figure 9.

The trainFile field represents the location of the .tsv (tab separated variables) file presented in Section 2.0.1 and built at the previous section, serializeTo field is the location to which the serialized, trained file will be written, while the map field is the exact layout of the columns of the .tsv file, in this case the numbering of the columns will start from 0, the first column being named as "word", whilst the second column (1) will be named as "answer", meaning the type of the named entity. It is important to note that the first column will be the vector of x (words) in the Conditional Random Field Algorithm (Subsection 2.1.1), while the second will be the vector of s (states) in the same algorithm.

```
trainFile = D:/Anul_3/Lic/stanford/train/dummy-romanian-corpus.tsv
serializeTo = D:/Anul_3/Lic/stanford/train/dummy-ner-model-romanian.ser.gz
map = word=0,answer=1

useClassFeature=true
useWord=true
useNGrams=true
noMidNGrams=true
maxNGramLeng=6
usePrev=true
useNext=true
useSequences=true
usePrevSequences=true
maxLeft=1
useTypeSeqs=true
useTypeSeqs2=true
useTypeySequences=true
wordShape=chris2useLC
useDisjunctive=true
```

Figure 9: The list of proprieties of the Stanford NER.

Having set the proprieties, one has to run the command in Figure 10

in the home folder of his Stanford NER software.

```
PS D:\Anul_3\Lic\stanford> java -cp "*" -mx4g edu.stanford.nlp.ie.crf.CRFClassifier -prop train/prop.txt
```

Figure 10: The command to run the Stanford Trainer.

Having run the command, the training will output the serialized, trained file, having applied the Conditional Random Field Classifier on all the windows it produced for over 100 iterations. The training process is time and memory consuming. For the latter, one may observe that the fourth parameter of the command in Figure 10 is the memory allocated for the training process (in the current case 4 GB of memory have been allocated).

In Figure 11, the training time for 5000 data entries is presented, which is around 6 minutes, the training having been done after 131 iterations and will produce satisfying results at tagging level. However, for a higher F-measure score, this thesis will use all of the pretrained data that has been comprised at the previous subsection, adding up to 195000 entries.

```
Iter 120 evals 141 <D> [M 1.000E0] 9.456E2 353.80s |5.058E0| {2.421E-4} 2.491E-4 -
Iter 121 evals 142 <D> [M 1.000E0] 9.453E2 356.50s |1.222E1| {5.851E-4} 2.353E-4 -
Iter 122 evals 143 <D> [M 1.000E0] 9.451E2 359.17s |7.131E0| {3.413E-4} 2.241E-4 -
Iter 123 evals 144 <D> [M 1.000E0] 9.449E2 361.84s |4.657E0| {2.229E-4} 2.113E-4 -
Iter 124 evals 145 <D> [M 1.000E0] 9.448E2 364.50s |1.229E1| {5.883E-4} 1.866E-4 -
Iter 125 evals 146 <D> [1M 2.810E-1] 9.447E2 368.31s |8.443E0| {4.042E-4} 1.777E-4 -
Iter 126 evals 148 <D> [M 1.000E0] 9.445E2 371.13s |4.352E0| {2.083E-4} 1.729E-4 -
Iter 127 evals 149 <D> [M 1.000E0] 9.444E2 374.08s |3.023E0| {1.447E-4} 1.549E-4 -
Iter 128 evals 150 <D> [M 1.000E0] 9.443E2 376.94s |4.487E0| {2.148E-4} 1.498E-4 -
Iter 129 evals 151 <D> [M 1.000E0] 9.443E2 380.35s |1.053E1| {5.041E-4} 1.363E-4 -
Iter 130 evals 152 <D> [M 1.000E0] 9.442E2 384.24s |5.090E0| {2.437E-4} 1.247E-4 -
Iter 131 evals 153 <D> [M 1.000E0] 9.441E2 387.83s |3.260E0| {1.560E-4} 1.033E-4 -
QNMinimizer terminated due to average improvement: | newest_val - previous_val | / |newestVal| < TOL
Total time spent in optimization: 391.42s
CRFClassifier training ... done [395.8 sec].
Serializing classifier to D:/Anul_3/Lic/stanford/train/dummy-ner-model-romanian.ser.gz... done.
```

Figure 11: Result for 5000 entries.

In order to run the classifier on all 195000 entries, various tweaks had to be made to the training file, such as the removal of some irrelevant punctuation (such as "():;"), because otherwise the Java stack would be exceeded.

```
Iter 236 evals 268 <D> [M 1.000E0] 7.432E3 624.17s |4.904E1| {2.306E-4} 1.142E-4 -
Iter 237 evals 269 <D> [M 1.000E0] 7.432E3 626.54s |2.377E1| {1.118E-4} 1.064E-4 -
Iter 238 evals 270 <D> [M 1.000E0] 7.431E3 628.89s |7.996E1| {3.760E-4} 1.003E-4 -
QNMinimizer terminated due to average improvement: | newest_val - previous_val | / |newestVal| < TOL
Total time spent in optimization: 631.27s
CRFClassifier training ... done [643.3 sec].
Serializing classifier to D:/Anul_3/Lic/stanford/train/dummy-ner-model-romanian.ser.gz... done.
```

Figure 12: Result for all 195000 entries.

In order to run the now-trained classifier on raw, unannotated data, one must implement a Python script, using the StanfordNERTagger class of the NLTK (Natural Language Toolkit) Python library this thesis has mentioned in the Introduction section.

An example of such Python script is presented in Figure 13, this exact script will be used in the web application this thesis aims to implement. Moreover, in Figure 14, one notices the results of a tagging example for the purpose of visualizing the results. These results are to be used as data for the Statistics section of this thesis, where one will be introduced to the numerical facts of this classification, compared to the other two that will be presented in the following subsections.

```python
import nltk
from nltk.tag.stanford import StanfordNERTagger

def run(sentence):
    import os
    java_path = "C:/Program Files/Java/jdk-12.0.1/bin"
    os.environ['JAVAHOME'] = java_path

    jar = 'D:/Anul_3/Lic/stanford/stanford-ner.jar'
    model = 'D:/Anul_3/Lic/stanford/train/dummy-ner-model-romanian.ser.gz'

    ner_tagger = StanfordNERTagger(model, jar, encoding='utf8')

    words = nltk.word_tokenize(sentence)
    raw_data = ner_tagger.tag(words)
    to_return = []
    for tuplu in raw_data:
        if tuplu[1] != 'O':
            to_return.append(tuplu)

    return to_return
```

Figure 13: Example of Python script to run the trained classifier.

```
PS D:\Anul_3\Lic\Training_Methods> python .\ner_romanian.py
[('Tudor', 'PERSON'), ('Arghezi', 'PERSON'), ('Ion', 'PERSON'), ('Nae', 'PERSON'), ('Theodorescu', 'PERSON'), ('Academia', 'PERSON'), ('Romana', 'PERSON'), ('21', 'MISC'), ('mai', 'MISC'), ('1880', 'MISC'), ('14', 'MISC'), ('iulie', 'MISC'), ('1967', 'MISC'), ('scriitor', 'PERSON'), ('roman', 'MISC')]
```

Figure 14: Output of the Classifier.

One may notice that the output of the classification in this case is a list of tuples having the following format [(word_text1, NER_tag1),(word_text2, NER_tag2), ..., (word_textn, NER_tagn)]. This format is the one to be used

in the manual training of the test data used in the Statistics section of the application, as well as for the other classifiers.

3.3 The Practical Application of the Gazetteer Method

3.3.1 Creation of the List of Entities

The creation of this corpus is highly time consuming. The internet is the best resource as to gain as much information as possible in order to complete the list necessary for a satisfactory classification of some more complex sentences.

Data from the Wikipedia article dedicated to Romanian surnames have been extracted, a list of the most common 2000 first names in the language, as well as data from the previously presented RONEC and QuoVadis corpora have been extracted in order to comprise the "PERSON.txt" file for named entities of this particular types. The same procedure has been applied for the other 3 categories of named entities (ORGANIZATION, MISC, GPE), with the mention that for the geographical entities, data has also been extracted from the gazetteer comprised in the study "A Mixed Approach in Recognising Geographical Entities in Texts"(Cristea *et al*, 2015).

The format of the above mentioned files is one-per-line in a simple text file, for easier access and reduced complexity of the search algorithm. Figure 15 depicts a snippet of the "PERSON.txt" file, comprising of 43727 entries.

```
43718  Țăroi
43719  Țăroiu
43720  Țăruș
43721  Țărușsi
43722  Țărână
43723  Țărîndă
43724  Țărăngoiu
43725  Țărăpănel
43726  Țărăscu
43727  Țărău
```

Figure 15: Snippet of the PERSON.txt file.

3.3.2 The Operation of the Gazetteer Method

First and foremost, one must run the Stanford POS Tagger in order to appropriately tag the parts of speech in the given test text. The code of said tagging is depicted in Figure 16, while the output in Figure 17. The test data to which the tagger is applied is the sentence: *"Tudor Arghezi, pseudonimul lui Ion Nae Theodorescu, Academia Romana, nascut la 21 mai 1880, Bucuresti decedat in 14 iulie 1967 a fost un scriitor român, cunoscut pentru contribuția sa la dezvoltarea liricii românești sub influența baudelairianismului. Opera sa poetică, de o originalitate exemplară, reprezintă o altă vârstă marcantă a literaturii române. A scris, între altele, teatru, proză (notabile fiind romanele Cimitirul Buna Vestire și Ochii Maicii Domnului), pamflete, precum și literatură pentru copii. A fost printre autorii cei mai contestați din întreaga literatură română".*

```
def extract_pos(doc):
    result = []
    for sent in doc.sentences:
        for wrd in sent.words:
            result.append((wrd.text, wrd.upos))
    #return a dataframe of pos and text
    return result
```

Figure 16: The function for the Part of Speech Tagger.

```
[('Tudor', 'PROPN'), ('Arghezi', 'PROPN'), (',', 'PUNCT'), ('pseudonimul', 'N
), (',', 'PUNCT'), ('nascut', 'VERB'), ('la', 'ADP'), ('21', 'NUM'), ('mai',
7', 'NUM'), ('a', 'AUX'), ('fost', 'AUX'), ('un', 'DET'), ('scriitor', 'NOUN'
tarea', 'NOUN'), ('liricii', 'NOUN'), ('românești', 'ADJ'), ('sub', 'ADP'), (
('de', 'ADP'), ('o', 'DET'), ('originalitate', 'NOUN'), ('exemplară', 'ADJ'),
, 'NOUN'), ('române', 'ADJ'), ('.', 'PUNCT'), ('A', 'AUX'), ('scris', 'VERB')
, ('notabile', 'ADJ'), ('fiind', 'AUX'), ('romanele', 'NOUN'), ('Cimitirul',
(',', 'PUNCT'), ('pamflete', 'NOUN'), (',', 'PUNCT'), ('precum', 'ADV'), ('și
 ('autorii', 'NOUN'), ('cei', 'DET'), ('mai', 'ADV'), ('contestați', 'VERB'),
```

Figure 17: A snippet of the output, when run on test data.

As the gazetteer method usually is insensitive to the context, which means that it sees the words in a sentence as a list, one may filter the results presented above and extract only said words that are nouns and proper nouns, and then proceed to the tagging itself, with the returned result being of the

same type as that of the one outputted by the classifier in the previous section. These facts are depicted in Figures 18 and 19.

```
def runGazeteer(sentence):
    f = open(r"D:\Anul_3\Lic\Resources\PERSON.txt", "r", encoding="utf8").read()
    g = open(r"D:\Anul_3\Lic\Resources\GPE.txt", "r", encoding="utf8").read()
    h = open(r"D:\Anul_3\Lic\Resources\MISC.txt", "r", encoding="utf8").read()
    j = open(r"D:\Anul_3\Lic\Resources\ORGANIZATION.txt", "r", encoding="utf8").read()
    # #extract pos
    # print(extract_pos(doc))
    person = []
    gpe = []
    misc = []
    organization = []
    for line in f.split('\n'):
        person.append(line)
    for line in g.split('\n'):
        gpe.append(line)
    for line in h.split('\n'):
        misc.append(line)
    for line in j.split('\n'):
       organization.append(line)
    sent = nlp(sentence)
    pos_tagged = extract_pos(sent)
    ner_list = []
    # print ("Arghezi" in person)

    for word in pos_tagged:
        if word[1] == 'NOUN' or word[1] == 'PROPN':
            if word[0] in person:
                ner_list.append((word[0], "PERSON"))
            elif word[0] in gpe:
                ner_list.append((word[0], "GPE"))
            elif word[0] in organization:
                ner_list.append((word[0], "ORGANIZATION"))
            elif word[0] in misc:
                ner_list.append((word[0], "MISC"))

    return ner_list
```

Figure 18: The function for the classification of the proper nouns.

One may see that the proper nouns that describe persons have been correctly classified, as well as those describing geographical location, meanwhile those describing miscellaneous named entities or organizations named entities, due to the vast scope of their respective entities are not perfectly recognized.

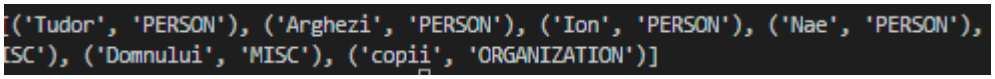

Figure 19: A snippet of the output when run on test data.

3.4 SpaCy Named Entity Recogniser

In order to run the SpaCy software on a language that does not have any pretrained model, one needs to download the multi language model available on the SpaCy website in order to access data extracted from various Wikipedia pages, comprised into a training data set for the convolutional neural network (described in Subsection 2.2). The aforementioned model supports identification of PERSON, GPE, ORGANIZATION and MISC entities, similar to the other two methods described in the previous chapters.

Having downloaded the resources necessary for the execution, one has to import and load the model into the memory, as described in Figure 20.

```
import spacy
import xx_ent_wiki_sm
nlp = xx_ent_wiki_sm.load()
```

Figure 20: SpaCy imports

The SpaCy NER tagger will then work automatically, tagging the instances, the output being formatted to the format used in the previous two sections of the thesis (list of tuples of the form (word, entity), where entity is one of PERSON, ORGANIZATION, GPE or MISC). The execution of the SpaCy tagger is shown in Figure 21, while the output is in Figure 22, the input data being the same sentence as the aforementioned two other methods of classification.

It is to be observed that the classification is correct to a certain degree and becomes satisfying when taking into account that this classifier has no specific model for the Romanian language.

3.5 Statistics

In order to achieve comprehensive statistics, one must run the three aforementioned methods of classification on multiple examples from different

```python
def run_english(sent):
        import spacy
        import xx_ent_wiki_sm
        nlp = xx_ent_wiki_sm.load()

        sentence = nlp (sent)

        l =  [(X.text, X.label_) for X in sentence.ents]
        # print(l)
        to_return = []

        # print(l)
        for item in l:
                if len(item[0].split(' ')) == 1:
                        if item[1] == 'PER':
                                new_tuple = (item[0], 'PERSON')
                        elif item[1] == 'ORG':
                                new_tuple = (item[0], 'ORGANIZATION')
                        elif item[1] == 'GPE':
                                new_tuple = (item[0], 'GPE')
                        else:
                                new_tuple = (item[0], 'MISC')
                        to_return.append(new_tuple)
                else:
                        for tab in item[0].split(' '):
                                if item[1] == 'PER':
                                        new_tuple = (tab, 'PERSON')
                                        to_return.append(new_tuple)
                                elif item[1] == 'ORG':
                                        new_tuple = (tab, 'ORGANIZATION')
                                        to_return.append(new_tuple)
                                elif item[1] == 'GPE':
                                        new_tuple = (tab, 'GPE')
                                        to_return.append(new_tuple)
                                else:
                                        new_tuple = (tab, 'MISC')
                                        to_return.append(new_tuple)

        return to_return
```

Figure 21: SpaCy execution and formatting the data.

spheres and contexts. For doing so, a list of 20 pieces of text has been comprised, the contents of which varying from scientific text, journalistic text, social media, academic or online article.

The format chosen for this collection of data is a dictionary with two

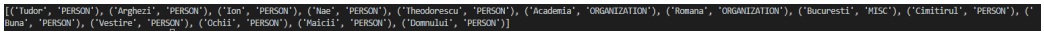

Figure 22: SpaCy results

keys, the 'raw' tab, and the 'tagged' tab, with the raw tab containing the text as it is, unannotated, while the tagged tab contains the Named Entity Recognition done manually by the author. Figure 23 presents a snippet of the file mentioned above.

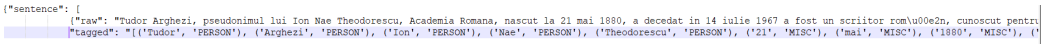

Figure 23: Dictionary of test data.

In order to present the precision and recall functions, one must define some key terms.

The first of those is the term of true positive. One calls a word a true positive, if the word is a named entity, the classifier tagged the word as a named entity, and the category attributed the word is correct.

Secondly, one calls a false positive a word which is not a named entity, but has been classified as such.

Thirdly, a false negative is a either a named entity which has not been classified at all, or one which has been wrongly classified.

Lastly, a true negative is a word that is neither a named entity, nor has it been classified as such.

One computes the following metrics, which have theoretically addressed in Section 1 of the current thesis, for calculating the performance of each classifier:

1. $$precision = \frac{true\_positives}{true\_positives + false\_positives}$$

2. $$recall = \frac{true\_positives}{true\_positives + false\_negatives}$$

3. $$F - measure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Each of these three metrics are relevant to the overall performance of one's Named Entity Recognition software. In this current thesis we address all 3 of them in order to asses the level of corectness each method of classification previously described achieves.

For each of the methods, one will compute a function in order to compute the parameters previously stated, as in the example in Figure 24.

```
elif method.lower() == 'blind':

    for sentence in sentences:
        tagged_sentence =ner_english.run_english(sentence['raw'])
        for item in tagged_sentence:
            if str(item) in sentence['tagged']:
                true_positives += 1
            elif str(item[0]) in sentence['tagged']:
                false_negatives += 1
            else:
                false_positives += 1
            false_negatives += (all_correct_tagged - true_positives)
            true_negatives = all_words - all_correct_tagged
            # print(all_words, all_correct_tagged)
```

Figure 24: The statistics function for the SpaCy method.

It is highly important to mention that variables "all_words" and "all_correct_tagged" have been previously computed, the former being the sum of all words in the test examples (be them named entities or not), while the latter the sum of all manually tagged words (all of the named entities).

The result of these calculations is an excellent indicator to how the system performs, and which of the methods is better on this particular data set.

Figure 25 depicts the output of the precision algorithm presented above, similarly, Figures 26 and 27 present the output of the recall and F-measure algorithms respectively.

```
The precision of the trained Stanford CRFClassifier is: 0.8390804597701149
The precision of the Gazetteer Classifier is: 0.5858585858585859
The precision of the SpaCy multi-language model Classifier is: 0.75
```

Figure 25: The output of the precision algorithm.

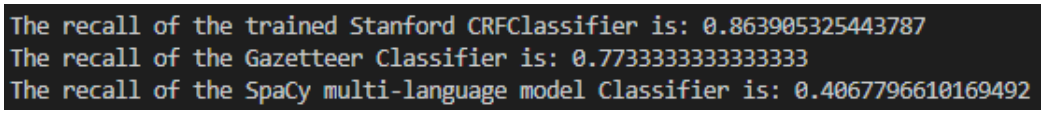

Figure 26: The output of the recall algorithm.

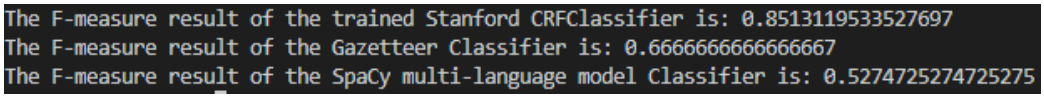

Figure 27: The output of the F-measure algorithm.

One must note that the results are bounded by 1 (it being the maximum precision, recall and F-measure achievable). If we were to scale the result to percentages, the most exact overall classifier (the metric of choice is the F-measure score), would be the trained Stanford Conditional Random Field Classifier, which has been previously presented in Subsection 2.1 from a theoretical standpoint and in Subsection 3.2 from a practical standpoint.

The results show that inevitably the best classifier is, with a score of 85.1% is the trained Stanford classifier, while the least accurate is the SpaCy, an expected result, due to its lack of Romanian language-specific models.

The lack of higher accuracy in the context of the Gazetteer method is due to its lack of context data, as well as due to the lack of sufficient data in order to interpret different ORGANIZATION and MISC entities, despite the relatively high number of instances present in the training data.

3.6 The Voting Function

In order for the application to achieve the best possible classification comprised of the three methods, one must implement a voting system in order to best choose the classification needed for the untagged sentence.

This thesis proposes a voting function which combines the F-measure results computed in the previous subsection, making each of the F-measure scores ratios of their sum.

The general formula for each of the weights is:

$$\frac{f_{\mathrm{i}}}{\sum_{i=1}^{3} f_{\mathrm{i}}}$$

One will then run each of the classifiers, and presuming the same noun phrase is caught by all classifiers in the same way, will compute the

overall ratio of said noun phrase.

All of the weights will be saved in a list as proper numbers. So, the weight of the first classifier is 0.42 (the result of $\frac{0.86}{2.04}$, where 0.86 is the F-measure score of the trained Stanford NER Classifier and 2.04 is the sum of all classifiers), that of the second classifier is 0.32 (the result of $\frac{0.66}{2.04}$, where 0.66 is the F-measure score of the Gazetteer Classifier and 2.04 is the sum of all classifiers), and the weight of the third classifier is 0.25 (the result of $\frac{0.52}{2.04}$, where 0.52 is the F-measure score of the SpaCy Classifier and 2.04 is the sum of all classifiers). One might add that the sum of all these weights is 1, meaning that each classifier has been assigned the weight corresponding to how much it contributes to the sum of F-measure scores.

For example, if the word "Google" is present in the output of the first classifier, its score will be 0.42, if the same word is present in the second classifier, one will add to its score 0.32 and so on.

Having stated those above, one will compare the sum of the weights left in the list of weights (those in which the word is not present in) to the score it has after the aforementioned computations have been performed. If the score is higher than the sum of ratios, it will be present in the final result list.

```
def vote(sentence):

    result = []
    trained = ner_romanian.run(sentence)
    gazetteer = stanford.runGazeteer(sentence)
    spacy = ner_english.run_english(sentence)



    for data in trained:
        result.append(data)
    for data in gazetteer:
        coefs = [0.85, 0.09, 0.06] #according to each method's F-measure score
        score = 0.09
        coefs.remove(0.09)
        if data in trained and data not in result:
            score += 0.85
            coefs.remove(0.85)
        if data in spacy and data not in result:
            score += 0.06
        if score > sum(coefs):
            result.append(data)

    for data in spacy:
        coefs = [0.85, 0.09, 0.06] #according to each method's F-measure score
        score = 0.06
        coefs.remove(0.06)
        if data in trained and data not in result:
            score += 0.85
            coefs.remove(0.85)
        if data in spacy and data not in result:
            score += 0.09
        if score > sum(coefs):
            result.append(data)


    return result
```

Figure 28: The Python implementation of the voting function.

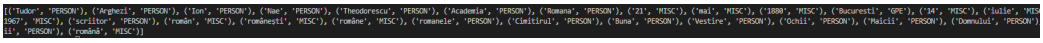

Figure 29: The output of the voting function.

Each classifier behaves differently with respect to each of the classes (meaning that one of them may be better with PERSON entities, one with GPE entities and so on). This suggests that for each of the identified entities one needs to combine the results of the three classifiers into one score.

3.7 The Web Application

For the user interface, this thesis has adopted the use of a web application with a user-friendly front-end, as well as all the necessary information about the thesis. One can observe that a minimalist approach has been chosen, as depicted in Figure 30.

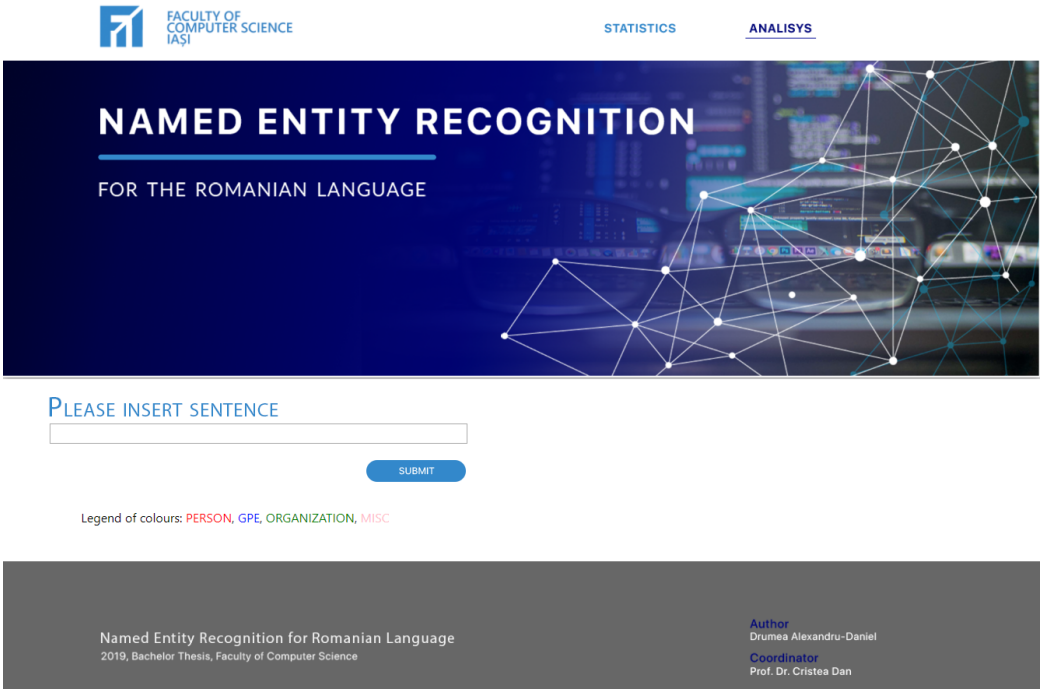

Figure 30: The homepage of the web application.

The operation of the aforementioned application is fairly simple, as the user has to input the sentence he wants to label, and the result of said classification will appear on the right side of the box, just as Figure 31 depicts.

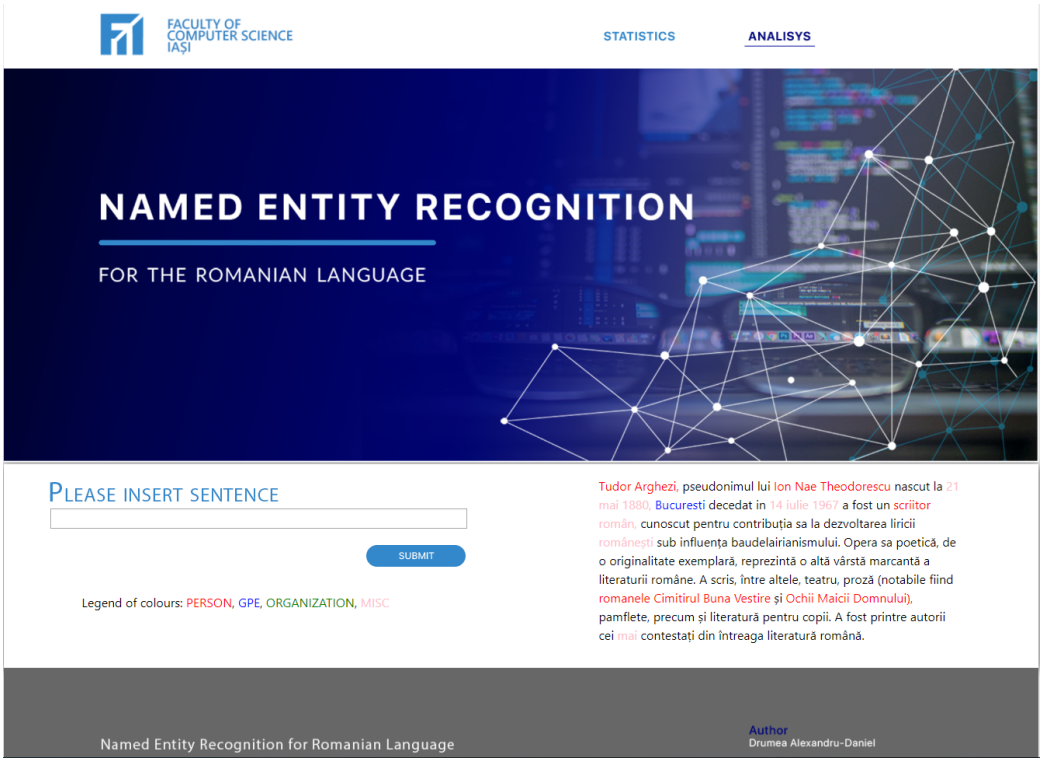

Figure 31: The homepage of the web application with labeled text.

The logic of this application is as follows, the user input is extracted from the front-end, it being passed on to the voting function described previously. The voting function will do the named entity recognition as described in the previous sections, then call the beautify function (Figure 32), which will return the newly formed text, with the words labeled, using the colours described in the legend section of the page.

In order for all of the above to be fulfilled, Flask, the microframework described at Section 2.2 has been used, employing the same approach described in great detail there.

```
def beautify(sentence, ner_tags):

    to_return = ""
    for word in sentence.split(' '):
        aux = word
        new_word = ""
        for tupl in ner_tags:
            if tupl[0] == word.strip(',').strip(")").strip("("):
                # print(word)
                if tupl[1] == 'PERSON':
                    # print(tupl[0])
                    new_word = "<span style=\"color: red\">" + aux + "</span>"
                    break
                elif tupl[1] == 'GPE':
                    new_word = "<span style=\"color: blue\">" + aux + "</span>"
                    break
                elif tupl[1] == 'ORGANIZATION':
                    new_word = "<span style=\"color: green\">" + aux + "</span>"
                    break
                else:
                    new_word = "<span style=\"color: pink\">" + aux + "</span>"
                    break
        if len(new_word) == 0:
            new_word = aux
        to_return += new_word + ' '

    return to_return
```

Figure 32: The beautify function.

4 Conclusions

The methods used in the current thesis have returned results that are satisfying as well as comprehensive. One has managed to achieve an F-measure score of 85%, which consistently makes reliable classifications of sentences when it comes to named entities. If on a relatively short training data set the results have been such satisfying, one may assume that on a much more complex data set, the metrics will improve, the application having to reach a higher level of accuracy.

As one may have noticed in the course of this brief documentation, the premise of having a named entity recognizer has been achieved in a very comprehensive manner.

The first steps taken in the beginning of this thesis were to take the state of the art technologies such as SpaCy and test them on an unannotated,

Romanian text. This procedure achieves an F-measure score of just 52%, which despite not being satisfying, is not a negligible score either.

The Gazetteer method, which in some areas provide one with quite accurate classifications, has its drawbacks, having achieved an F-measure score of 66%,, which primarily comes from the difficulty of keeping a gazetteer up-to-date, as well as its lack of context awareness, thus making it not such a reliable, long-term tool.

The aim of this thesis has been reached by using the Conditional Random Field Algorithm of the Stanford Named Entity Recognition Software, which has obtained a score of 85% accuracy. Even though it is not quite at the level of the industry standard, it is a fine attempt in solving the problem of named entity recognition for the Romanian language.

Improvements are required in order for the application to become a reliable, powerful, industry standard tool, these improvements being: bigger contexts, more data, more ambiguity applied to said data, as well as much more powerful computers than the one used in the current thesis.

In conclusion, the aim set at the beginning of this paper has been achieved, the knowledge gathered has been documented and the improvements needed have been identified and, therefore, the premises for further studies with more ambitious goals have been set.

5 Bibliography

1. Charles Sutton, Andrew McCallum (2011), "An Introduction to Conditional Random Fields", Vol. 4, No. 4, 267–373

2. Anca-Diana Bibiri, Mihaela Colhon, Paul Diac, Dan Cristea (2014). Statistics Over A Corpus Of Semantic Links: "QuoVadis". In Mihaela Colhon, Adrian Iftene, Verginica Barbu Mititelu, Dan Cristea, Dan Tufiş (eds.) (2014). Proceedings of the 10th International Conference "Linguistic Resources And Tools For Processing The Romanian Language, Craiova, 18-19 September 2014", "Alexandru Ioan Cuza" University Publishing House, pag. 33-44

3. Y. LeCun, L. Bottou, Y. Bengio and P. Haffner (November 1998), Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324

4. Wen-tau Yih, Xiaodong He, Christopher Meek (June 2014), Microsoft Research, Association for Computational Linguistics, "Semantic Parsing for Single-Relation Question Answering",

5. Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, Ye-Yi Wang (May 2015), NAACL, "Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval"

6. Nal Kalchbrenner, Edward Grefenstette, Phil Blunsom (2014), Department of Computer Science, "A Convolutional Neural Network for Modelling Sentences", University of Oxford

7. Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, Pavel Kuksa (2011), "Natural Language Processing (Almost) from Scratch", "Journal of Machine Learning Research 12" 2493-2537

8. https://github.com/dumitrescustefan/ronec

9. https://nlp.stanford.edu/software/CRF-NER.shtml

   Cristea D., Gîfu D., Pistol I., Sfirnaciuc D., Niculiţă M. (2016) A Mixed Approach in Recognising Geographical Entities in Texts. In: Trandabăţ D., Gîfu D. (eds) Linguistic Linked Open Data. RUMOUR 2015. Communications in Computer and Information Science, vol 588. Springer, Cham

10. $https://spacy.io/models/xx$

11. $https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da$

12. $https://towardsdatascience.com/precision-vs-recall-386cf9f89488$

13. Charles Sutton, Andrew McCallum (2011), "An Introduction to Conditional Random Fields", Foundations and Trends in Machine Learning Vol. 4, No. 4, 267–373

14. Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky (2014). The Stanford CoreNLP Natural Language Processing Toolkit In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.

15. https://blog.sicara.com/train-ner-model-with-nltk-stanford-tagger-english-french-german-6d90573a9486

16. https://profs.info.uaic.ro/ dcristea/papers/RUMOUR-Cristea%20et%20al.pdf

17. https://pythonprogramming.net/named-entity-recognition-nltk-python/

18. Manning, Christopher D. and Surdeanu, Mihai and Bauer, John and Finkel, Jenny and Bethard, Steven J. and McClosky (2014), "The Stanford CoreNLP Natural Language Processing Toolkit", Association for Computational Linguistics, 55-60

19. http://www.aclweb.org/anthology/P/P14/P14-5010

20. https://pythonhow.com/how-a-flask-app-works/

21. https://profs.info.uaic.ro/ dcristea/cursuri/IA/2018-2019 /Curs13-%20Analiza%20limbajului%20natural.ppt.pdf

22. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

23. https://universaldependencies.org/

24. Doug Cutting and Julian Kupiec and Jan Pedersen and Penelope Sibun (1992) ,"A Practical Part-of-Speech Tagger", ANLC '92 Proceedings of the third conference on Applied natural language processing Pages 133-140

25. https://medium.com/@jrodthoughts/ambiguity-in-natural-language-processing-15f3e4706a9a

26. https://www.quora.com/How-does-CNN-work-with-NLP

27. https://en.wikipedia.org/wiki/Convolutional_neural_network #Natural_language_processing

28. http://flask.pocoo.org/

29. https://www.techopedia.com/definition/13825/named-entity-recognition-ner

30. https://medium.com/explore-artificial-intelligence/introduction-to-named-entity-recognition-eda8c97c2db1