TRABAJO DE FIN DE GRADO

Título:	Desarrollo de un Sistema de Reconocimiento de Entidades de Nombre basado en Algoritmos de Aprendizaje Automático por Ensamblado Selectivo
Título (inglés):	Development of a Named Entity Recognition System based on Ensemble Machine Learning Algorithms
Autor:	Constantino Román Gómez
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Mercedes Garijo Ayestarán
Vocal:	Tomás Robles Valladares
Secretario:	Carlos Ángel Iglesias Fernández
Suplente:	Amalio Francisco Nieto Serrano

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE GRADO

DEVELOPMENT OF A NAMED ENTITY RECOGNITION SYSTEM BASED ON ENSEMBLE MACHINE LEARNING ALGORITHMS

Constantino Román Gómez

Julio de 2015

Resumen

El Reconocimiento de Entidades de Nombre (NER) es una tarea primordial en el Procesamiento de Lenguajes Naturales. Consiste en detectar y clasificar expresiones que identifican claramente un elemento. NER ha sido investigado durante más de veinte años, pero sigue siendo un importante desafío.

Los sistemas NER usan diferentes enfoques, desde algoritmos hechos a mano hasta el aprendizaje automático, incluyendo el aprendizaje supervisado y no supervisado.

Los algoritmos de aprendizaje automático por ensamblado selectivo han sido investigados recientemente. La principal idea detrás de este enfoque consiste en combinar un conjunto de múltiples clasificadores para obtener mejores resultados. Cada clasificador es entrenado secuencialmente. Este método puede mejorar los resultados del NER de un clasificador individual.

En este proyecto, nos centraremos en NER aplicado al dominio de Twitter. Twitter es un dominio particularmente desafiante debido a su brevedad, falta de contexto, estilo informal y naturaleza inmediata, con nuevas entidades apareciendo continuamente.

El proyecto también abordará el desafío de adaptar sistemas NER existentes al español, dado que la mayoría de sistemas NER disponibles han sido desarrollados y ajustados para el inglés.

Nuestro enfoque estará basado en el uso de métodos de ensamblado selectivo para combinar varios sistemas NER disponibles. El proyecto será evaluado con datasets de tweets en el dominio educativo.

Palabras clave: Named Entity Recognition, Natural Language Processing, NIF, Ensemble Learning, Senpy, FOX, Weka, Webanno, Python, Java

Abstract

Named Entity Recognition (NER) is a major task of Natural Language Processing. It consists in detecting and classifying phrases that clearly identify one item. NER has been researched for more than twenty years, but it still remains a big challenge.

NER systems use different approaches, ranging from hand-crafted algorithms to machine learning, including supervised and unsupervised approaches.

Ensemble machine learning algorithms have been researched recently. The main idea behind them is that a set of multiple classifiers can be combined to achieve better performances. Each classifier is trained sequentially. It can improve NER performance of a single classifier.

In this project, we will focus on NER applied to the Twitter domain. Twitter as a domain is particularly challenging due to its shortness, lack of context, informal style and real-time nature, with new entities appearing constantly.

The project will also address the challenge of adapting existing NER systems to Spanish, given that most available NER systems have been specially developed and tuned for English.

Our approach will be based on the use of ensemble methods to combine several available NER systems. The project will be evaluated in a dataset of tweets in the education domain.

Keywords: Named Entity Recognition, Natural Language Processing, NIF, Ensemble Learning, Senpy, FOX, Weka, Webanno, Python, Java

Agradecimientos

En primer lugar, quiero dar las gracias a Carlos, mi tutor, por la oportunidad que me ha brindado para desarrollar este proyecto y por la ayuda que me ha proporcionado para llevarlo a buen puerto.

A mis compañeros de laboratorio, por ayudarme con el proyecto cuando lo he necesitado y por el gran entorno de trabajo que han creado.

A mis padres por apoyarme todos estos años, especialmente en mi decisión de venir a Madrid a estudiar este grado.

A mis amigos y a mis compañeros de la Escuela, por su apoyo y ayuda para terminar este grado y, sobre todo, por acompañarme durante estos años.

Gracias.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Context

Natural Language Processing is a field of Artificial Intelligence and linguistic committed to the interactions between computers and documents written in human languages, as [4] defines. A natural language can be defined as the language used by people for general communications. NLP systems have evolved from hand-written rules to supervised and unsupervised learning. This field includes numerous areas of research, including Named Entity Recognition, which is the task on which this project focuses.

The Named Entity Recognition (NER) task is essential to extract information from unstructured documents [13]. It consists on recognizing entities like person, organization and location names. Most work in NER research is dedicated to news article documents and the English language. However, new languages and domains, such as social media, are receiving more attention recently. Twitter is one of the domains where new research is being focused, and numerous classifiers are being developed specifically for it.

Moreover, research has shown that classifiers with diverse strengths can be combined in various ways to build better-performing systems [16]. This approach is called ensemble learning. While this learning method has already been applied to NER, its performance on

different languages and domains has not been fully researched yet.

Taking this context into account, we present Nerdy: a NER service that integrates classifiers for English and Spanish languages and for both Twitter and the general domain. It uses the format NIF as interface to make it easy to integrate and extend with other tools. In addition, it includes modules for ensemble NER classifiers training and evaluation.

1.2 Project goals

We have a series of goals that we wanted to achieve with this project. These include:

- Develop a tool that provides NER as a service in the NIF format.
- Look into research on NER in Spanish and on the Twitter domain and evaluate tools on this cases.
- Integrate utilities for document annotation and NER training and evaluation.
- Review Ensemble Learning research for Named Entity Recognition classifiers and implement ensemble classifiers.

1.3 Structure of this document

In this section we summarize the chapters included in this document. The document has the following structure.

Chapter 1 introduces the project, explaining its context and the main objectives that the projects aims to achieve. **Chapter 2** describes the theoretical aspects the project is based on and the technologies that it uses. **Chapter 3** offers a description of the NER part of the project, including its architecture and a detailed view of each of the modules it includes. **Chapter 4** describes the Ensemble Learning part of the project, explaining its implementation and offering an in-depth view on all its functionalities and how it is linked to the NER part of the project. **Chapter 5** offers several evaluations performed on the project and analyzes the results that were obtained. Finally, **Chapter 6** summarizes the conclusions extracted from the development of the project and proposes paths for future work on the project.

CHAPTER 2

Enabling Technologies

2.1 Introduction

In this chapter, we will define Named Entity Recognition, expound its characteristics and look into the research that has been carried out in last years. We will also explain various technologies that will be used in this project and the reasons to use them.

As said in [15], NER is a major task in Natural Language Processing (NLP). It is essential to recognize named entities, and numeric expressions including time, date and money. NER has been researched for more than twenty years and while methods have evolved a lot, it still remains a big challenge.

The problem is often broken down in two different sub-tasks, named entity detection (also called segmentation) and classification by the type of entity to which they refer. Another step that can follow NER is Named Entity Disambiguation (NED), which consists in linking entity mentions within the same document, or in other resources, such as Wikipedia or Freebase.

Named Entity Segmentation

Barack Obama is the President of the United States.

Named Entity Classification

Barack Obama PERSON is the President of the United States ORGANIZATION.

Sharnagat (2014) [15] defines named entities as words or phrases that clearly identify one item from a set of other items with similar attributes. In natural language, these entities are usually classified in three types: persons, locations and organizations. These types can be divided into sub categories, but in practice it is very difficult to identify them.

The type "miscellaneous" is used in the CoNLL conferences and includes proper names that doesn't fall in the previous categories. It can be divided in many types like "product", "timex" types (date and time), "numex" types (money and percent), and other types for specific needs ("phone number", "URL", "Twitter username").

Other critical aspects of NER include features and evaluation methods. Some other important factors reported in [13] in this task are:

- Language factor: A large proportion of research has been dedicated to the study of English but many other languages have been studied, mainly German in CoNLL-2003, Spanish and Dutch in CoNLL-2002, Japanese, Chinese and many others.

- Textual genre or domain factor: The impact of these factors has been less studied, with few studies specifically dedicated to diverse genres and domains. Experiments have proved that any domain can be basically supported, but porting a system to a new domain or genre remains a major challenge. (Drop in performance reported)

- Entity type factor: The most studied types are names of persons, locations and organizations. Each type can be divided in multiple subtypes (locations: city, state...; person: politician, entertainer).

2.2 NER

An essential part of NER systems is the ability to recognize previously unknown entities. Early systems used hand-crafted rule-based algorithms, while most recent studies use supervised machine learning as a way to automatically induce rule-based systems or sequence labelling algorithms starting from a collection of training examples.

2.2.1 Supervised methods

Currently, the dominant technique in the NER task is supervised learning. It consists in studying the features of positive and negative examples of NE over a large collection of annotated documents and design rules that capture instances of a given type. These systems usually memorize lists of entities and create rules for disambiguation. Some of the most used supervised learning algorithms are Hidden Markov Models, Decision Trees, Maximum Entropy Models, Support Vector Machines and Conditional Random Fields.

- Hidden Markov Models: in this model, the system is assumed to be a Markov process, a process without memory, with hidden states but visible output. It makes three assumptions: the next state depends only on the current state, state transition probability do not depend of the time at which the transitions are made, and the output is independent of previous outputs, given the current state. HMMs have been applied to part-of-speech tagging, text-segmentation, information extraction and named entity recognition, with successful results.

- Maximum Entropy: this is a flexible technique based on the Principle of Maximum Entropy, defined by Jaynes (1957), which says that the correct distribution is the one that maximizes entropy, or uncertainty; any other distribution would make arbitrary assumptions of information we do not have. Therefore, these methods model all known information but don't assume anything we don't know.

- Support Vector Machine: a SVM builds a hyperplane where it maps the training examples that it is given, separating the examples of different categories by the widest gap possible. Then, it can map new examples on the hyperplane and predict their categories based on the side of the gap where they are mapped.

- Conditional Random Fields: CRF classifiers can use the context of the samples, taking the previous and the next tags into account, for example. This allows them to predict sequences of labels from inputs made of sequences of samples.

The main disadvantage of supervised learning (SL) is the requirement of a large annotated corpus. The unavailability and the high cost of creating these resources lead to two alternative learning methods: semi-supervised learning (SSL) and unsupervised learning (UL).

2.2.2 Semi-supervised methods

The usual approach for SSL is called "bootstrapping", which requires a small degree of supervision for starting the learning process. This technique might ask the user to provide a few examples, then search for sentences that contain them and identify contextual clues common to the examples. These clues will be used by the system to find other instances in similar contexts, and the learning process is repeated, finding new examples and contexts. Eventually, the system will be able to find a large amount of instances similar to the ones provided in the examples, which could be diseases, book titles, etc.

2.2.3 Unsupervised methods

In UL, no labels are given to the learning algorithm. The main technique for UL is clustering. For example, named entities can be collected based on the similarity of context. There are other methods too, that usually rely on lexical resources like WordNet, on lexical patterns and on statistics computed on large unannotated corpora.

2.2.4 The Twitter domain

Social media have experienced enormous growth in recent years, with millions of people interchanging information constantly. Twitter is one of the leading platforms, with 302 million active users and over 500 million tweets sent per day. Organizations are more and more willing to analyse automatically this massive amount of data.

Most NER systems were developed and trained on news articles or other well written content. As we have discussed earlier, NER systems are not easily ported to new domains, experimenting important drops in their performance. Twitter is a particularly challenging domain for three main reasons.

First, tweets are short, limited to 140 characters, lacking enough context to determine entities types. Secondly, Twitter text style is usually informal, with grammatical errors, misspelled or abbreviated words, and frequent use of hashtags, emoticons and abbreviations. Finally, tweets also contain very different entity types, and new entities appear constantly due to Twitter's real-time nature. This makes traditional NER methods underperform, having 30-50% accuracy while these methods have 85-90% on longer texts, as stated in [5].

To address these problems, numerous techniques have been proposed. Many of them use knowledge bases such as Wikipedia, Freebase or DBpedia to provide large dictionaries

of entities.

Most NER systems are conducted in a supervised manner.

In [14], Ritter et al. introduce a NLP system based on a distantly supervised approach. They use LabeledLDA applying constraints based on Freebase as a source of supervision. This system contains NLP tools adapted to tweets, forming a pipeline. These tools include POS tagging, noun-phrase chunking, capitalization classification and named entity recognition.

Evaluating this system with a dataset collected by B. Locke (2009) [12], we obtain better results than the one reported by Locke using his NER system, which used a small amount of labeled data and a large amount of unlabeled data. These results are shown in Table 2.1. Locke obtained an F1 measure of 31.05 percent, while Ritter et al. system accomplish an F1 measure of 54.89 or 59.15 percent, depending on whether we only consider full match or also partial match for multi-word entities. These concepts will be further explained in Chapter 3.

Table 2.1: Twitter NLP Tools evaluation

Measure	Full	Full+Partial
Precision	0.642	0.694
Recall	0.479	0.515
F1	0.548	0.591

Gattani et al. [8] describe a system that uses Wikipedia as a knowledge base, and also uses its traffic as a social signal to improve the accuracy of the tasks.

It also interleaves the four tasks (extraction, linking, classification and tagging), so that the output of each one help the others.

The system uses tweets context to improve its accuracy. This context includes other tweets from the same user or hashtags, but also websites linked in the tweets and social signals, such as traffic in Pinterest or Wikipedia, as said earlier.

These supervised systems have a main disadvantage: they require the availability of annotated data, which is hard to obtain. To avoid this problem, other proposed NER systems for Twitter take an unsupervised approach.

In this category we find TwiNER [11], which consists of two steps: tweet segmentation

and segment ranking.

The first step relies on the collocation of words in a named entity. Unlike capitalization and other linguistic features, the correct collocation of a named entity is preserved in tweets. Leveraging on this feature, they segment tweets and obtain a set of phrases, which will be candidate named entities. The segmentation is based on two ideas. Firstly, they use Microsoft Web N-Gram corpus to count the appearance of each possible segment and estimate its probability of being a candidate named entity. The second idea consists in searching the segments in Wikipedia article titles, disambiguation pages, etc. to refine the probability of being a named entity.

The second step assigns a confidence score to each candidate named entity based on its probability of being a true named entity. The idea consists in building a segment graph using all the candidate named entities and applying a random walk model to derive the probabilities. Global context is also integrated using Wikipedia to refine the scores.

2.3 Ensemble methods

2.3.1 Introduction

As explained earlier, a supervised learning algorithm takes a set of training examples and outputs a classifier. An ensemble of classifiers is a set of classifiers where their predictions are combined to achieve a more precise output. This combination is typically made by weighted or unweighted voting.

Ensemble methods has been one of the most active areas in supervised learning, and it has been discovered that this methods often outperform the original individual classifiers.

One of the most important conditions for an ensemble to be more accurate than its component classifiers is that these classifiers are diverse. Classifiers are diverse if they make different errors on new data. This way, when one of them is wrong, the others may be correct, so the a voting algorithm will classify the entity correctly.

Another important condition is that the individual classifiers have an error rate below 0.5, or the ensemble system will underperform the classifiers.

Therefore, it is essential to obtain individual classifiers with error rates lower than 0.5 whose errors are uncorrelated.

As it has been proved, ensemble methods work in practice, being possible to develop

ensembles with much better performance that their classifiers. There are three main reasons that explain this.

The first reason is statistical. When we have little training data compared to the size of the problem, many classifiers can be trained that give similar accuracy on the training data. If we construct an ensemble with these classifiers, the ensemble algorithm is able to reduce the risk of choosing the wrong classifier. In other words, if we average the vote of different accurate classifiers, we can obtain a better classifier.

The second reason is computational. Several learning algorithms work by minimizing errors over the training data, and they often get stuck in local points. An ensemble based on a search from different starting points can produce a classifier that outperforms the individual classifiers.

The third reason is representational. In many learning problems, the true function that the classifiers are trying to represent is impossible to obtain. This is because most learning algorithms only explore a limited series of hypotheses, and stop when a hypothesis that works well on training data is found. However, if we produce a weighted sum of many hypothesis, is is possible to expand the space of representable functions and obtain a hypothesis closer to the true function.

2.3.2 Methods

Lots of methods have been developed to construct ensembles. Some of them are the following:

- Bayesian voting: if we see each classifier as a hypothesis of the true function, each one of them defines a conditional probability distribution. The true function can be expressed as a weighted sum of all the hypotheses, and this can be viewed as an ensemble method. This method is optimal in problems where we can completely list all possible hypotheses and compute their probabilities. However, it is often difficult in practice.
- Manipulating the training data: this method consists in altering the training examples to produce multiple hypotheses. Then, the learning algorithm is run many times, each one with a different set of the training data. There are three main ways of manipulating the training set: bagging, cross-validation and boosting.
  - Bagging consists in training the algorithm with a set of the same size of the total training sample that contains, on the average, 63.2% of the original training set,

the rest being duplicates.

- In cross-validation, training sets are built by randomly dividing the original set in a number of subsets, 10 for example. Then, one of them is left out in each subset, obtaining 10 different subsets that can be used to train 10 different classifiers.
- Finally, boosting involves constructing an ensemble by training each model emphasizing the importance of the classes that the previous models classified wrongly. The most popular implementation of boosting is the AdaBoost algorithm, which applies a series of weights to the training examples. In each iteration, the algorithm it is called to reduce the weighted error on the training set, returning a hypothesis. In the next iteration, the weights are updated based on the weighted error, giving more importance to the training examples that were misclassified. The final classifier is built by a weighted vote of all the classifiers. These weights are based on the accuracy on the training sets the classifiers were trained on.

These methods work best with unstable learning algorithms, such as decision-tree, neural network and rule learning algorithms.

- Manipulating the input features: this method consists in dividing the set of input features in subsets that are used to train different classifiers.This method only works well when the input features are very redundant.

- Manipulating the output targets: in this technique, the total number of classes can be divided in many subsets and all classes in a subset are given a label, 0 and 1 if we divide the classes in two, for example. The relabeled data is used to train a classifier, and this process is repeated many times. When we need to classify new data, each classifier is applied to the data. If the classifier returns 0, each class labeled with it receives a vote. After all the classifiers have voted, the class with the most votes is the prediction that the ensemble returns.

- Injecting randomness: this method can be applied to numerous types of learning algorithms. Randomness can be injected into initial weights in neural networks training, into the selection of the best prediction in decision trees, or into the input features, for example.

2.3.3 Comparison

Many experiments have been conducted to compare the performance of different ensemble methods. Bauer and Kohavi (1999) and Dietterich (2000) carried out the largest studies in

this matter. The results indicate that AdaBoost usually performs best in datasets with little noise. Bagging and randomized trees give similar performances, although randomization shows better results in very large datasets. However, AdaBoost doesn't work well with noisy datasets, while bagging shows very good performances in these datasets.

2.3.4 Weka

WEKA (Waikato Environment for Knowledge Analysis) [10] is a free machine learning workbench developed at the University of Waikato, New Zealand. It was developed to provide a simple and quick way to test and compare different algorithms on different datasets, and it also provides with mechanisms to extend it easily. It includes numerous machine learning algorithms and data preprocessing tools. However, we will only use the following 13 machine learning algorithms:

- AdaBoostM1 with J48 as base classifier (ABM1)
- Bagging with J48 as base classifier (BG)
- Decision Table (DT)
- Functional Trees (FT)
- J48
- Logistic Model Trees (LMT)
- Logistic Regression (Log)
- Additive Logistic Regression (LogB)
- Multilayer Perceptron (MLP)
- Naïve Bayes (NB)
- Random Forest (RF)
- Support Vector Machine (SVM)
- Sequential Minimal Optimization (SMO)

We will use these algorithms through the FOX framework, which we will explain next, to create ensemble classifiers.

2.3.5 FOX

FOX is an open-source NER framework written in Java that implements ensemble learning to leverage on the variety of NER classifiers available. It also provides a REST-ful service to easily access to its resources.

The FOX framework creates NER ensemble classifiers by using 15 learning algorithms. 13 of them are the previously listed algorithms from WEKA, and the other 2 are voting algorithms: CVote, which selects the classifier with best performance for each entity class, and Vote, which doesn't apply weights, relying on the Majority Vote Rule.

To train ensemble classifiers with the previous learning algorithms, FOX implements four NER single classifiers: the Stanford NER classifier, the Illinois Named Entity Tagger, the Ottawa Baseline Information Extraction (Balie) and the Apache OpenNLP classifier. Each classifier is implemented as a Java class, making it easy to extend FOX with new classifiers. We added several Spanish NER classifiers, as we will explain in Chapter 4.

2.4 NIF

NIF (NLP Interchange Format) is an RDF/OWL-based format developed in the University of Leipzig. Its goal is interoperability between NLP tools, language resources and annotations.

The motivation behind it, given the numerous NLP tools and services that are appearing recently, is to simplify the combination of these tools. These often are developed in a pipeline with many different tools (POS tagging, named entity recognition, etc) and it is often desirable to reuse parts of different pipelines in new tools, like ensembles, for example. These pipelines use very different formats, and a common format (NIF) makes much simpler their combination. This way, complex NLP applications can be built by combining many tools.

Another important task that makes interoperability so important is to use enormous quantities of Linked Data available on the Web, so that NLP tasks can be boosted when they use this data as background knowledge or gazetteers, for example. Simple interoperability between NLP tools also allows to compare results, helping in all this ways to achieve better scores in precision and recall in NLP tasks.

NIF has a core that consists in a vocabulary that is used to represent strings as RDF resources. Annotations are referenced to parts of documents by URIs. These URIs can be

used to publish annotations on the Web as Linked Data, or to interchange them between NLP tools. NIF has three basic components:

Structural Interoperability: annotations are anchored to documents with URI schemes that also use String and Structured Sentence Ontologies to define the type of the URIs and their relations.

Conceptual Interoperability: NIF is easy to extend, and it already integrates many ontologies, such as DBpedia for entity linking.

Access Interoperability: NIF supports the interaction between NLP tools by a REST interface description. Listing 2.1 shows an example of NIF applied to NER annotations.

Listing 2.1: NIF TTL example

```
<https://github.com/badiehm/TwitterNEED/archive/MasterBranch.zip/
    Microposts2014_Collection_train.xml/10320787951072460 8#char=0,>
  a nif:String , nif:Context , nif:RFC5147String ;
  nif:isString """Oh dear . Kun Aguero has scored more for his new club in half an hour
      than Fernando Torres has in seven and a half months . Just saying ."" ;
  nif:beginIndex "0"^^xsd:nonNegativeInteger ;
  nif:endIndex "137"^^xsd:nonNegativeInteger .

<https://github.com/badiehm/TwitterNEED/archive/MasterBranch.zip/
    Microposts2014_Collection_train.xml/10320787951072460 8#char=10,20>
  a nif:String , nif:RFC5147String ;
  nif:anchorOf """Kun Aguero""" ;
  nif:beginIndex "10"^^xsd:nonNegativeInteger ;
  nif:endIndex "20"^^xsd:nonNegativeInteger ;
  a nif:Phrase ;
  itsrdf:taIdentRef "http://dbpedia.org/resource/Sergio_Ag%C3%BCero" .

<https://github.com/badiehm/TwitterNEED/archive/MasterBranch.zip/
    Microposts2014_Collection_train.xml/10320787951072460 8#char=75,90>
  a nif:String , nif:RFC5147String ;
  nif:anchorOf """Fernando Torres""" ;
  nif:beginIndex "75"^^xsd:nonNegativeInteger ;
  nif:endIndex "90"^^xsd:nonNegativeInteger ;
  a nif:Phrase ;
  itsrdf:taIdentRef "http://dbpedia.org/resource/Fernando_Torres" .
```

2.5 Senpy

Senpy [1] is a tool originally developed to create sentiment and emotion analysis servers easily. It is written in Python and it uses NIF+JSON-LD as interface, making it suitable to communicate different NLP tools. Its goal is to provide a simple way to turn sentiment analysis algorithms into servers. However, in Nerdy we will use it to create a NER server.

To create a web service, a new plugin must be defined. Each plugin only needs two files: a .senpy file where the service is defined, including the port where it will be accessible and other options, and a .py file where the service is implemented and the algorithm is called with the options required in the call parameters. Finally, in this Python file the response is processed, creating an "Entry" with various attributes for each element, converting them to NIF+JSON-LD and sending the final response.

[1] `https://github.com/gsi-upm/senpy`

CHAPTER 3

Nerdy Architecture

3.1 Introduction

In this chapter, we present the design of the Nerdy service and all the tools that we have developed in it, including the system architecture. Firstly, we present the Nerdy pipeline with all its modules. Then, we explain each module in depth and how they communicate with each other.

The main goal of Nerdy (Named Entity Recognition and Disambiguation in pYthon) is to offer a plethora of NER classifiers as a service in a common format, NIF. There are numerous NER systems in the public domain, but they usually work with different input or output formats. This makes it difficult to compare them, evaluate them or combine them in ensembles. Nerdy aims to makes these processes easier by integrating numerous NER systems and offering a number of useful tools, like training, evaluation and NIF conversion modules, in addition to the web service.

To take advantage of these features, we also used the FOX framework to train ensemble classifiers based on the NER systems implemented in Nerdy, as we will explain later. In order to train these ensembles, we communicate FOX with Nerdy to use its classifiers. Nerdy is also communicated with FOX to provide its ensembles as another classifier available in

the web service, like the ones previously implemented.

Figure 3.1 shows a diagram of this architecture, but we will explain each module further along this chapter.

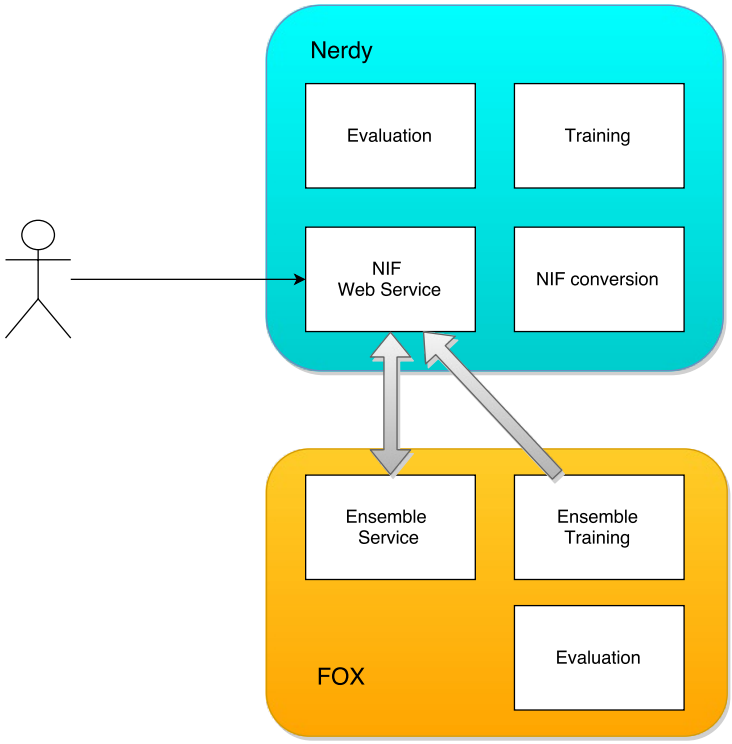

Figure 3.1: Nerdy architecture

3.2 Pipeline

The Nerdy pipeline consists of several components, as Figure 3.2 shows.

The first task is the annotation of documents, and it's optional to the pipeline. It is implemented with a Webanno [17] service, which supports multi-user annotation, curation and monitoring with many layers of annotations. New layers and tagsets can also be created. When the annotation is finished, the results are exported in the Webanno TSV format, although it also supports many others. Then, a converter to NIF-TTL is applied, allowing its use for testing or training the NER classifiers.

We implemented many NER classifiers available in the public domain. Nerdy includes the Stanford CRF NER [6], the classifier developed for tweets by Ritter et al. [14], the

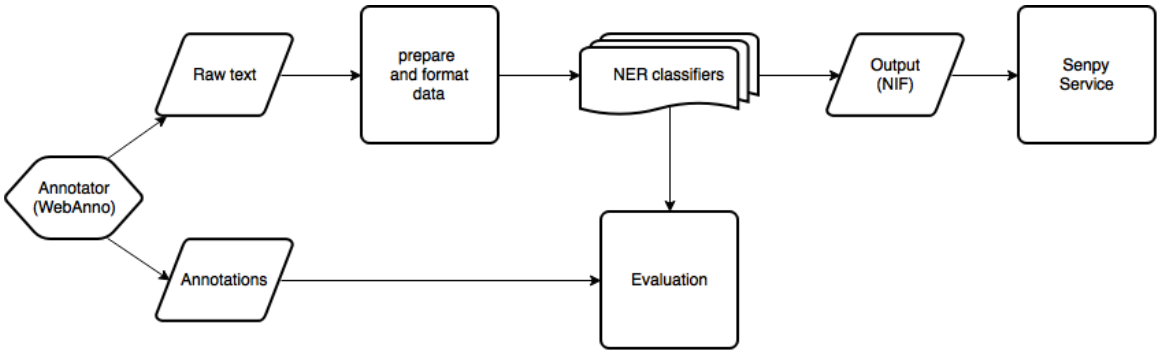

Figure 3.2: Nerdy pipeline

CitiusTagger classifier from the University of Santiago de Compostela [7], and the Polyglot-NER classifier [1]. The Stanford NER includes models for Spanish and English, while the CitiusTagger supports also Portuguese. The Polyglot-NER classifier includes NER annotator for 40 languages. A wrapper was developed to interact with these classifiers and convert their output to a common format, NIF-TTL.

The results from these NER systems can be evaluated using the evaluation module we developed. It provides with precision, recall and F1 scores for only full matches or taking partial matches into consideration.

The NER classifiers can be easily called from a web service that we developed as a plugin for Senpy. Senpy is a tool that allows easy creation of web services using NIF+JSON-LD as interface.

3.2.1 Annotator

The main reason to integrate an annotator in Nerdy is to create annotated documents that will be used to train and evaluate the classifiers. It is optional to the pipeline, given the several annotated datasets for the NER task that are freely available. The annotator that we decided to implement is based on Webanno [17]. We considered other web annotators, but they did not work well with very big datasets and had fewer options and usability.

Webanno is a general purpose web-based annotation tool that provides a wide range of different types of annotations. It includes POS tags, named entities and many other layers of annotations. It also support custom annotation layers and allows to annotate a text on multiple layers simultaneously. New tagsets can be easily created and assign to new or already existent layers. Webanno also supports numerous input formats for documents, from plain text to their own Webanno TSV format.

Figure 3.3 shows the user interface of the web service, with class tags for each named entity. The document is part of the CoNLL-2002 Spanish dataset. As it can be seen on the right side of the figure, the selected annotation from the text "La Coruña" corresponds with the Named Entity layer. Other layers can be selected for annotation in the drop-down menu "Layer". La Coruña is annotated as LOC, which indicates Location. Many named entities classes included in the Webanno Named Entity tagset can be selected on the "value" drop-down menu. New values can also be created and included in the tagset.

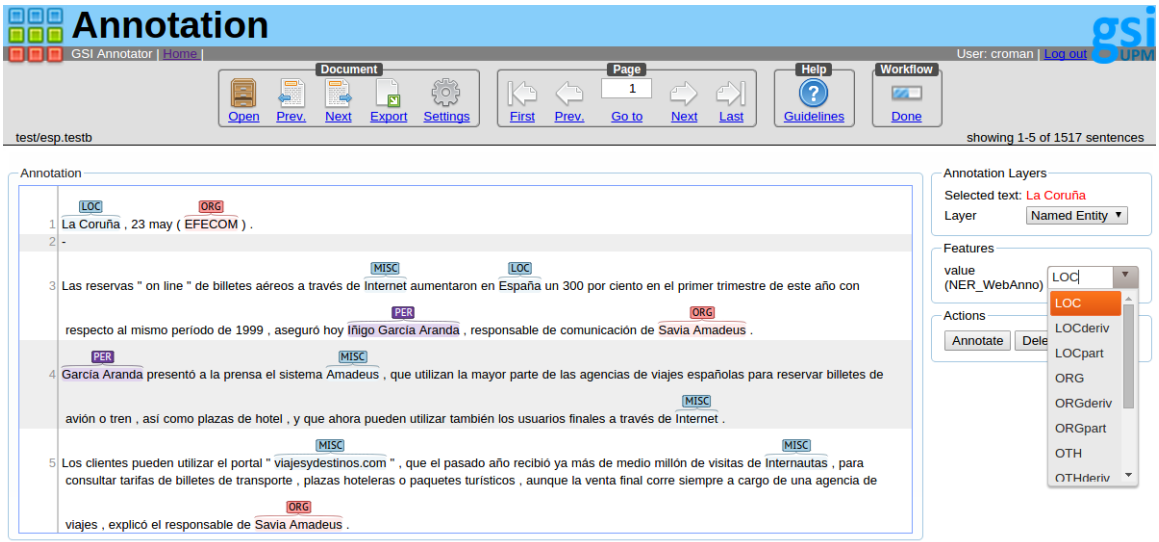

Figure 3.3: Webanno user interface

Webanno supports multi-user annotation, providing different roles like annotator, curator or project manager. Numerous annotator can be assigned with different texts to annotate, and the curator can decide which annotation is correct when there are conflicts between annotations made by different users. It also supports monitoring of the progress of the annotations, and agreement between annotators. The annotator has a pipeline task architecture.

In the first place, a project manager must create a project, assign users with roles and documents to annotate, and define guidelines to the annotators. Types and tagsets can also be configured, including custom ones. When the project is set up, the annotators can begin to annotate the documents. Their progress can be monitored by the project manager, who can also assign workloads to the annotators, being possible to have different levels of redundancy.

When at least two annotators have finished their annotations on the same document, the curator can start his work. It consists on comparing the annotations, correcting them or solving conflicts when needed. When the curator has finished his work, the final annotated

documents can be exported in different formats. The format that we used is the Webanno TSV format, for which we developed a converter to NIF-TTL to fully integrate it in the Nerdy pipeline. This way, it can be used by the training and evaluation modules.

The annotator is completely web-based and does not require advance knowledge to use it. We implemented a Webanno service, creating new tagsets and layers for Sentiment Analysis. The service can be accesed in `http://demos.gsi.dit.upm.es/webanno/`. We wrote a guide in Spanish that can be found in `https://www.gsi.dit.upm.es/mediawiki/ssl/index.php/GSI_Annotator`, explaining its different uses and possibilities.

3.2.2 NER classifiers

Nerdy includes numerous classifiers in Spanish and English languages. Among the implemented NER classifiers, we can find:

- The Stanford NER [6] is a Named Entity Recognizer implemented in Java, although Python interfaces have also been developed. It is based on Conditional Random Field sequence models and it includes various models, with 3 named entities classes (Person, Organization, Location) but also with 4 (including Misc) or even 7 (dividing Misc in Time, Date, Money and Percent). It also has feature extractors for NERs, and options to define new ones.

  Nerdy currently supports a 3 class English model trained for the CoNLL and MUC datasets, and a Spanish one trained on a Spanish CoNLL news dataset. However, new models can easily be trained on new training data, providing options for adding a gazetteer, for example. In fact, a model was trained with a tweets dataset and a gazetteer for a challenge. This will be further explained later.

- The Ritter classifier [14] was presented earlier. Ritter et al. presented a NLP pipeline especially adapted to tweets that includes POS tagging, chuking and NER. As reported by them, their NER system doubles the F1 score of Stanford NER on tweets. This NER system has two separate tasks: segmentation and classification.

  The Named Entity Segmentation task use Conditional Random Fields for learning and inference. It uses orthographic, contextual and dictionary features,. The dictionary consists of a set of type lists obtained from Freebase. The outputs of other tasks (POS, chunking and capitalization) are also used as features. This system is trained with a set of 2,400 tweets. This task uses IOB encoding to represent the segments, so that each word either is inside, outside or begins a named entity. The segmentation task has a reported 52% raise on F1 score over the Stanford NER news-trained model.

The Named Entity Classification task uses lists of entities and their types gathered from Freebase to perform distant supervision. However, many entities that are found on tweets do not appear in Freebase, so LabeledLDA is applied to model these entities and its classification.

- The Polyglot [1] NER system includes classifiers for 40 different languages. To create these, they use language-independent techniques with minimal human intervention or annotated training data. Firstly, the system learns word embeddings. Then, Wikipedia link structure is used to detect named entities. Finally, they rely on Freebase attributes to identify the entities that will be used as training data.

  Polyglot is distributed as a natural language pipeline for Python with many other functions, like language detection, POS tagging, sentiment analysis and many more. We will only use the NER models on two languages: Spanish and English.

- Citius Tagger is a Named Entity Recognition and Classification tool developed by Gamallo et al. (2014) [7]. It is adapted to Portuguese, English and Spanish. This system classifies entities into four types: Person, Location, Organization and Miscellaneous. It is mainly based on queries to resource files extracted from FreeBase and DBpedia.

  Then, disambiguation rules are applied to previously identified entities to take a decision when there are many classes associated to an entity or there are unknown entities. These rules rely on gazetteers of entities extracted from DBpedia and FreeBase, and on triggers, which are key names used to classify entities into a class.

  The disambiguation algorithm firstly searches the entity in the gazetteers. If ambiguities or unknown classes are found, the algorithm searches for triggers in the entity context. If the class remains ambiguous, the more probable class is assigned, or if the entity is unknown, partial matches with gazetteers and triggers are explored, assigning the class "Miscellaneous" if all these steps fail.

  The validation of this system shows stable results independent from the type of corpus used, and overperforms the FreeLing and OpenNLP classifiers in corpus different from the CoNLL one, which they use for training.

3.2.2.1 NER training

We also implemented a module for NER training in Nerdy. We decided to adapt it to the Stanford CRF NER classifier because of its ease to use and integrate. Although the Stanford NER includes many models for English and Spanish, these models are trained on

news corpora and do not work well on other domains, like tweets. Therefore, it was decided to train new models adapted to tweets.

To train new models, we need to provide training data and define a properties file to select the features that we want to use.

The training data is expected to be in tab-separated columns, with a "word" column for the words and a "answer" column for the annotations. However, some feature extractors used by the Stanford NER can also use other column names like "tag". The meaning of the columns is defined on the "map" property of the properties file. In our module, we implemented the two basic columns, due to the fact that named entities datasets only have one layer of annotations.

To convert the data to this tab-separated format, we used Docon [1]. It allows us to create a converter by only defining a simple template. We created several converters to support conversion from different formats.

Once we have the training data in the adequate format, a properties file must be created. In this file, we define the location of the training dataset that we want to use, the meaning of its columns and the location where the classifier will be saved. We can also select the features we want to use in the training.

In addition to the basic features, Stanford also supports the use of gazetteers. We can create a simple list of entities that we want the trainer to use and define it in the properties file. In the gazetteer we only need to write two columns, one with the entities and the other one with the classes to which each entity belongs. Two types of gazetteers can be selected: clean gazetteers, which give a feature match only if the whole entity appears in the text, and sloppy gazetteers, that make the feature fire if any of the words match with an entity from the gazetteer. Listing 3.1 shows an example of a properties file with a sloppy gazette.

Listing 3.1: Stanford NER training properties file

```
#location of the training file
trainFile = datasets/esp.train.tsv
#location where you would like to save (serialize to) your
#classifier; adding .gz at the end automatically gzips the file,
#making it faster and smaller
serializeTo = ner-conll-sloppy.gaz.ser.gz

#structure of your training file; this tells the classifier
#that the word is in column 0 and the correct answer is in
#column 1
```

[1] https://github.com/gsi-upm/docon

```
map = word=0,answer=1

#these are the features we'd like to train with
#some are discussed below, the rest can be
#understood by looking at NERFeatureFactory
useClassFeature=true
useWord=true
useNGrams=true
#no ngrams will be included that do not contain either the
#beginning or end of the word
noMidNGrams=true
useDisjunctive=true
maxNGramLeng=6
usePrev=true
useNext=true
useSequences=true
usePrevSequences=true
maxLeft=1
#the next 4 deal with word shape features
useTypeSeqs=true
useTypeSeqs2=true
useTypeySequences=true
wordShape=chris2useLC

sloppyGazette=true
gazette=caps-socialtv.gaz.txt
```

We also implemented in the training module a k-fold cross-validation tool. In this method, the training dataset is randomly divided in k equally sized subsets, 10 for example. Then, all the folds but one are used to train a classifier and the left fold is used to test the resulting classifier. This process is performed k times, with each fold used once to test. We also average the results from all the iterations.

We used this module to train a classifier for the TASS 2015 challenge, that will be further explained in Chapter 5. We obtained high scores in the cross-validation, with an average precision of 94,09% and a recall of 88,35%, that produced an average F1 of 91,05%.

3.2.3 Service

Nerdy was implemented as a service using Senpy, which allows to easily build web services based on NLP tools using NIF+JSON-LD as interface. For this purpose, a plugin must be created. It only requires two files: a .senpy file, which defines the service and contains information such as the port where the service will be accessible or extra parameters that can be processed and their different options, and a Python file that implements the service, obtaining the parameters, calling the algorithm and processing the response.

The plugin developed for nerdy takes the text to analyse as the input, and supports the previously mentioned NER classifiers as an extra parameter in the call. It sends the input text to the selected classifier and the results are sent in the response in NIF+JSON-LD. Each entity is a new entry in the response, and includes the input text, the entity text, its class and its starting and ending indexes. A Nerdy call follows this structure: `http://localhost:5000/?i=BarackObamaeselpresidentedeEEUU&classifier=stanford-es&algo=nerdy`. The "i" parameter is the input text to be analyzed, the "classifier" parameter indicates which classifier will be used on the text, and the "algo" parameter is used to identify the Senpy plugin we want to use. A part of the Senpy JSON response is presented in Listing 3.2.

Listing 3.2: Nerdy JSON response example

```
entries: [
{
  nif:taClassRef : "dbo:PERSON"
  text : "Barack Obama es el presidente de EEUU",
  nif:endIndex : 12,
  nif:startIndex : 0,
  nif:anchorOf : "Barack Obama",
  @id : "Entry1"
},
{
  nif:taClassRef: "dbo:LOCATION",
  text: "Barack Obama es el presidente de EEUU",
  nif:endIndex: 37,
  nif:startIndex: 33,
  nif:anchorOf: "EEUU",
  @id: "Entry1"
}
]
```

3.2.4 Evaluation

We developed an evaluation module in Python. It takes as input the golden set with the correct annotated entities and the results from applying a classifier to the text of the golden set. Both files are expected to follow the NIF format on TTL. The function firstly parses the golden set and the results file using the rdflib Python library. Then, entities are extracted from the golden set and precision and recall are calculated by counting entities that appear on both sets, that we call "matches".

A problem appears then with multiword entities when some but not all the words from the entity are annotated by the classifier as an entity. For example, in the sentence "I love

San Francisco", the correct entity is "San Francisco", but the classifier can extract just "Francisco" as an entity.

To solve this problem, we decided to count two types of matches, as Gupta et al. [9] suggest. If all the words from the golden set entity are identified as an entity, we count it as a full match. If only some words are extracted as an entity, but not the full phrase, we count it as a partial match. All the evaluation results obtained with this function show the full match score, and the sum of full and partial match score.

This module provides with three different measures: precision, recall and F1. Precision is the fraction of the extracted entities that are correct, that is, the number of correct entities divided by the number of extracted entities. Recall represents the fraction of the golden set entities that are correctly extracted by the NER classifier. It is calculated by dividing the number of correct entities by the number of entities in the golden set. Finally, the F1 score combines both measures in the following formula:

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{3.1}$$

To calculate these measures, we use the entities extracted from the golden set and the NER classifier. The precision and recall functions are described in Algorithm 1, as the F1 score is trivial to obtain.

We use these functions on both full match and full+partial match. Listing 3.3 shows an example of the output of the evaluation module.

Listing 3.3: Evaluation output example

```
Full Mentions Precision: 0.787723785166
Full+Partial Mentions Precision: 0.892583120205
Full Mentions Recall: 0.607843137255
Full+Partial Mentions Recall: 0.676470588235
Full Mentions F1: 0.686190664415
Full+Partial Mentions F1: 0.76964379884
```

Algorithm 1 Counting full and partial matches to evaluate

Precondition: *set* represent the entities from the golden set and *results*, the entities extracted by the NER classifier we want to evaluate

```
1: function PRECISION(set, results)
2:     fm ← 0
3:     pm ← 0
4:     for entity in results do
5:         if entity in set then
6:             fm ← fm + 1
7:         else
8:             for entity′ in set do
9:                 if entity in entity′ then
10:                    pm ← pm + 1
11:                end if
12:            end for
13:        end if
14:    end for
15:    return fm, fm + pm
16: end function
17: function RECALL(set, results)
18:    fm ← 0
19:    pm ← 0
20:    for entity′ in results do
21:        if entity′ in set then
22:            fm ← fm + 1
23:        else
24:            for entity in set do
25:                if entity′ in entity then
26:                    pm ← pm + 1
27:                end if
28:            end for
29:        end if
30:    end for
31:    return fm, fm + pm
32: end function
```

CHAPTER 4

Ensemble Methods

4.1 Introduction

In this chapter, we explain the implementation of ensemble methods on the Named Entity Recognition task. Firstly, we present the FOX framework, its functioning and its architecture. Then, we expound the changes that we made to extend it and how we use it.

In order to apply ensemble methods to NER, the FOX framework [16] was chosen. Although there are tools like scikit-learn that work include several ensemble algorithms on Python and could be used for our purposes, FOX already implements several NER classifiers and includes ensembles training and evaluation modules. It implements 15 different algorithms for ensemble learning. The drawback is that it is written in Java, so we decided to communicate it with the Nerdy Senpy service to implement its classifiers.

FOX originally integrates the Stanford NER, the Illinois Named Entity Tagger, the Ottawa Baseline Information Extraction (Balie), the Apache OpenNLP classifier, and the DBpedia Spotlight tool. We decided to create an ensemble of Spanish NERs because FOX did not include any Spanish classifier. FOX implementation of the Stanford NER allowed to easily include Spanish models and Balie also has a Spanish named entity tokenizer, so they were implemented along with the Spanish classifiers from Nerdy.

Balie divides sentences into tokens (words or punctuations). POS tagging and capitalization are applied and used as features for the machine learning algorithm, which is based on Weka. Only three classes are considered: Person, Organization and Location, mapping other entity types from each classifier into these three. OpenNLP and DBpedia Spotlight proved to be difficult to implement, so they were not used in this work but could be implemented in future work.

FOX also uses Weka to implement its ensemble algorithms, which are 13: AdaBoostM1 and Bagging, which use J48 as its base classifier, Decision Table, Functional Trees, J48, Logistic Model Trees, Logistic Regression, Additive Logistic Regression, Multilayer Perceptron, Naïve Bayes, Random Forest, Support Vector Machine and Sequential Minimal Optimization. Two more ensemble methods are also implemented using voting: CVote (Class Vote) and Vote.

4.2 Architecture

FOX includes modules for training and evaluation of ensemble classifiers, in addition to the web service that makes the classifiers accessible. Both models apply k-folding cross-validation, which we will explain in the following sections.

To train or evaluate a classifier, many options can be selected. We can choose the dataset, the NER classifiers that the ensemble will use, the learning algorithm that will be used to train the ensemble, the number of runs and folds of the cross-validation and many other parameters. These options are selected in a properties file.

FOX uses a web service to make the classifiers accessible. It uses a REST-ful API that allows to call individual classifiers or ensembles on text sent in the call. The API accepts the following parameters:

- Input: this parameter includes the URL where the input text is located or the actual text, depending on the selected type.
- Type: this parameter can take the value "text", if the input is raw text, or "url", if the input text is located at an URL.
- Task: this parameter can only take the value "NER" for the moment.
- Output: with this parameter we can select the output format. It currently supports JSON-LD, N-Triples, RDF/JSON/XML, Turtle, TriG and N-Quads.

- Foxlight: it selects the NER classifier that we want to use to analyze the input text.

4.2.1 Training

To train the ensemble classifiers, they take annotated data and run all the basic classifiers to obtain their results and compare them. This way, a list of all the tokens with their golden set annotations and the classifiers ones is produced. As it can be seen in the example, each row represents a token or word, and each column is an annotation. Annotations made by the same classifier are divided in four columns, one for each class (including the Null class to represent non-named entity tokens). In these columns, a '1' indicates that the classifier identifies the correspondent token with the class that the column represent, where a '0' indicates the opposite. The rightmost column contains the golden set annotation. The ensemble learning algorithm takes this data as input, and uses it to train the ensemble classifiers.

```
(Richard Branson), chairman of (Virgin Atlantic Airways).

0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,PERSON

0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,PERSON

0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,NULL

0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,NULL

1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,ORGANIZATION

1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,ORGANIZATION

1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,ORGANIZATION
```

FOX uses k-fold cross-validation to evaluate the ensemble that it is training. This method consists on splitting the data in K folds (i.e., 10 folds) and use part of these folds as training data (i.e., 9 of the 10 folds) and the remaining part of the dataset as testing data. Training and testing are performed 10 times to predict how will the system generalize over independent datasets. When the process is finished, the ensemble classifier model is serialized and the scores from the cross-validation are returned. These scores give a measure on how the ensemble works in different datasets, checking that the ensemble classifier is not overfitting the training set. Similar scores over the different folds are desirable and usually show that the classifier is not overfit. Figure 4.1 shows a sequence diagram of the training and cross-validation process.

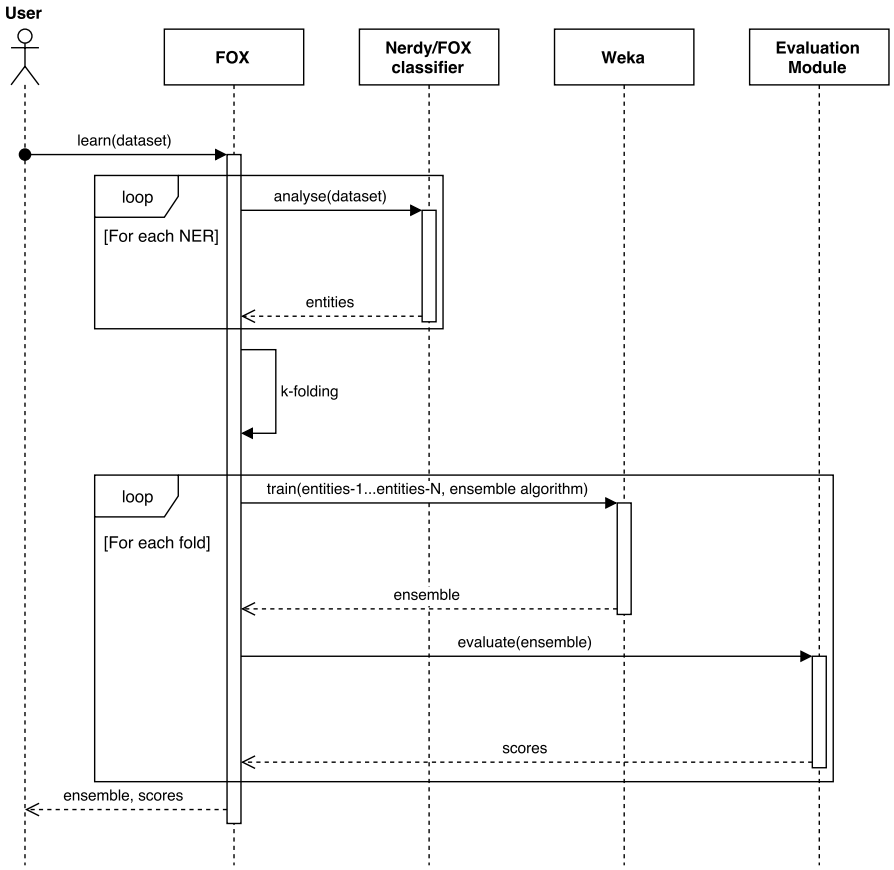

Figure 4.1: Training sequence diagram

The simplest algorithm used by FOX is voting. This algorithm consists in assigning to each input word the class that was predicted by the majority of classifiers. The other voting algorithm implemented by FOX is Class Vote. The strategy that it follows relies on selecting for each class the NER tool with best results obtained in the evaluation on that particular class.

4.2.2 Evaluation

FOX also has an evaluation pipeline. Firstly, they preprocess the input dataset, extracting the text, which will be used by the NER classifiers, and the correct named entities, which

are needed for evaluating the ensemble. Then, the classifiers are called to perform NER on the input text. The total output from the classifiers is randomly split in a number of folds (10 by default) to carry out a k-fold cross-validation. In addition to the number of folds, the number of repeats of the cross-validation is also adjustable. Each run, the k-folding will split differently the dataset, due to its random character.

FOX also carries out two types of evaluation: token-based and entity-based. The token-based evaluation takes partial matches into account. For example, if "Universidad Politécnica de Madrid" is annotated in the golden set as a Organization entity and a classifier predicts only "Universidad Politécnica" as an Organization, the evaluation function will count 2 true positives and 2 false negatives. However, entity-based evaluation only will count exact matches, so that the annotation from the previous example will be considered incorrect.

Evaluation carried out by Speck and Engomo (2014) [16] shows that Random Forest, Multilayer Perception, AdaBoostM1 and J48 are the best classifiers on the datasets used by them. However, MultilayerPerception and AdaBoostM1 are the only algorithms that obtain best results on both token-based and entity-based evaluation.

4.3 Extending FOX

To adapt FOX to our needs, Java classes were created for each NER classifier. Stanford and Balie were implemented locally, using their Spanish models to use them on this language. The Spanish model for Stanford was available in their website, and was trained on news wire corpora from the EFE news agency. The Balie model was already included in the tokenizer and we only had to select the Spanish language when the tokenizer was called. As said earlier, we decided to use the classifiers implemented on Nerdy in the ensembles, so their Java classes had to be communicated with the Nerdy Senpy service. They communicate with Nerdy by POST HTTP requests. GET requests could not be used, due to the length of the datasets we used for training and evaluation. The classifiers from Nerdy that we use on FOX are the CitiusTagger and the Polyglot-NER, because Stanford was locally implemented to save time and resources, and the other classifiers do not have Spanish models.

For the training and evaluation tasks, we used two Spanish datasets from CoNLL-2002. These datasets contain annotated text from news wire articles of the EFE News Agency, from 2002. FOX input format for training and validating datasets is different from the formats we had used before. To convert the datasets to this format, we used Docon [1], a

[1] https://github.com/gsi-upm/docon

tool that allows easy conversion using templates. We defined a template specifically to carry out conversion from the CoNLL-2002 datasets format to the format used by FOX.

We trained ensembles using all the 15 machine learning algorithms provided on the FOX framework. To train each ensemble we defined in the FOX properties file the individual classifiers, the ensemble learning algorithm and the training datasets we were going to use. When each ensemble training finished, scores from the cross-validation were provided. We will analyze these results in Chapter 5.

Figure 4.2 shows our ensemble training pipeline. We use the data from the CoNLL datasets, the NER classifiers from Nerdy and the ones adapted from FOX, and FOX ensemble training algorithms, finally obtaining an ensemble classifier.

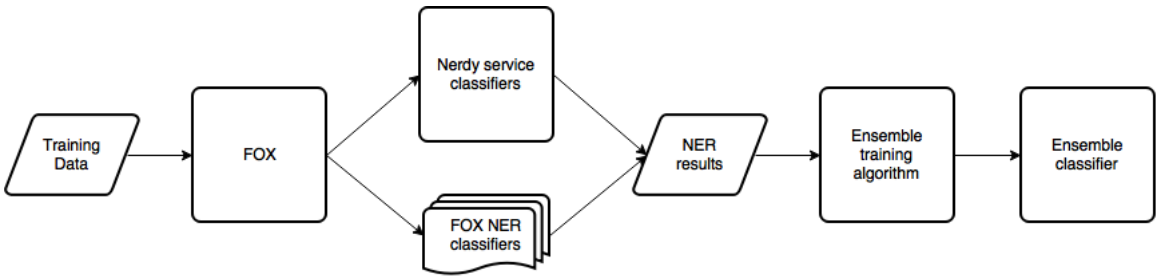

Figure 4.2: Training pipeline

We have explained earlier how we communicated FOX with Nerdy to use the classifiers implemented in Nerdy to train ensembles with FOX. We have also linked FOX with Nerdy to make the ensemble classifiers we trained in FOX available in Nerdy. This way, we can make requests to the Nerdy service to use ensemble classifiers and obtain a response in NIF.

In the process that will follow a request to Nerdy using an ensemble classifier FOX and Nerdy will communicate two times. First, FOX will use the Nerdy classifiers and its own integrated ones to retrieve the entities from a text. Next, the ensemble model will use this output to provide its analysis of the input text. Then, the results will be communicated to the Nerdy service, which will finally transmit them in the response in NIF format. Figure 4.3 shows this process.

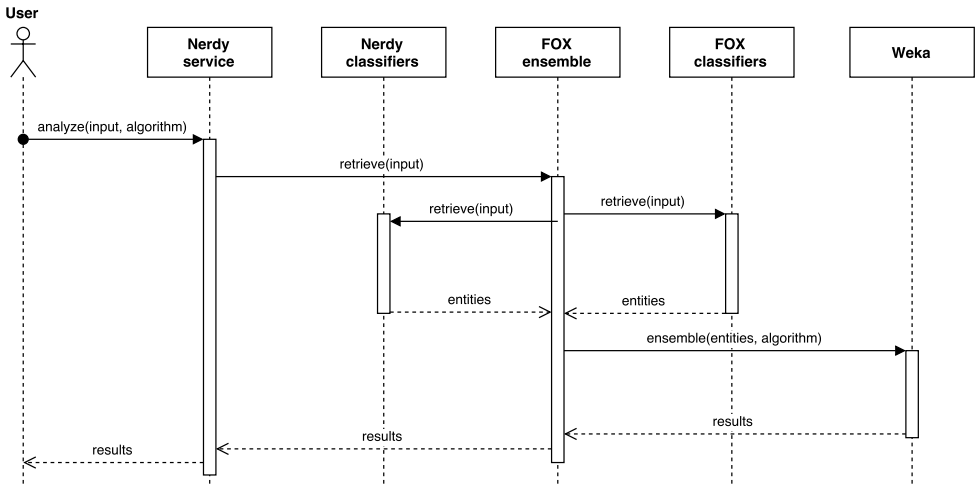

Figure 4.3: Nerdy request of FOX ensemble

CHAPTER 5

Evaluation

5.1 Introduction

The goal of this chapter is to show an analysis and comparison of the various classifiers we integrated or developed over different datasets. We will also provide the result of the tests we performed on the NER classifiers that we have described in previous chapters.

Firstly, we will analyze the scores obtained with the classifiers integrated in Nerdy. Next, we will present a Sentiment Analysis challenge for which we trained a NER classifier, and expound the results. Finally, we will explain the ensembles that we trained with the FOX framework and the results that were obtained on the cross-validation performed in the training process.

5.2 Nerdy Results

We tested classifiers from Nerdy using three datasets that contain tweets in English. The first dataset is called *Brian* [12], the second one is named *Mena* [2] and the third one is *Microposts2014* [3]. Table 5.1 shows the number of tweets and entities of each dataset. Microposts2014 is the largest, followed by Brian dataset. Mena dataset has very few tweets,

but contains many more entities per tweet.

Table 5.1: Number of tweets and entities of each dataset by class

	Brian Dataset	Mena Dataset	Microposts2014 Dataset
Tweets	1603	162	2339
Entities	1585	510	3819

We used the evaluation module of Nerdy to test Ritter and Polyglot NER classifiers with the three previous datasets. Figures 5.1, 5.2 and 5.3 show the scores on each dataset. We obtained results for full mention matches and full+partial mention matches. Mena was the dataset with best scores, with Ritter NER leading in full match F1 (68.62%) and Polyglot NER performing better in full+partial match F1 (78.08%).

The other two larger datasets produced lower scores. In the Brian dataset Ritter NER had the best results, with 54.97% F1 in full match and 59.01% F1 in fuull+partial match. In the Microposts2014 dataset Ritter NER led again with 48.51% F1 in FM and Polyglot NER performed best in FM+PM match, with an F1 of 54.1%.

In general, Ritter NER performed better than Polyglot, especially in full match evaluation. However, Polyglot obtained better results in full+partial match evaluation in Mena and Microposts2014 datasets. This can be explained by the fact that Ritter NER includes Twitter-specific features.

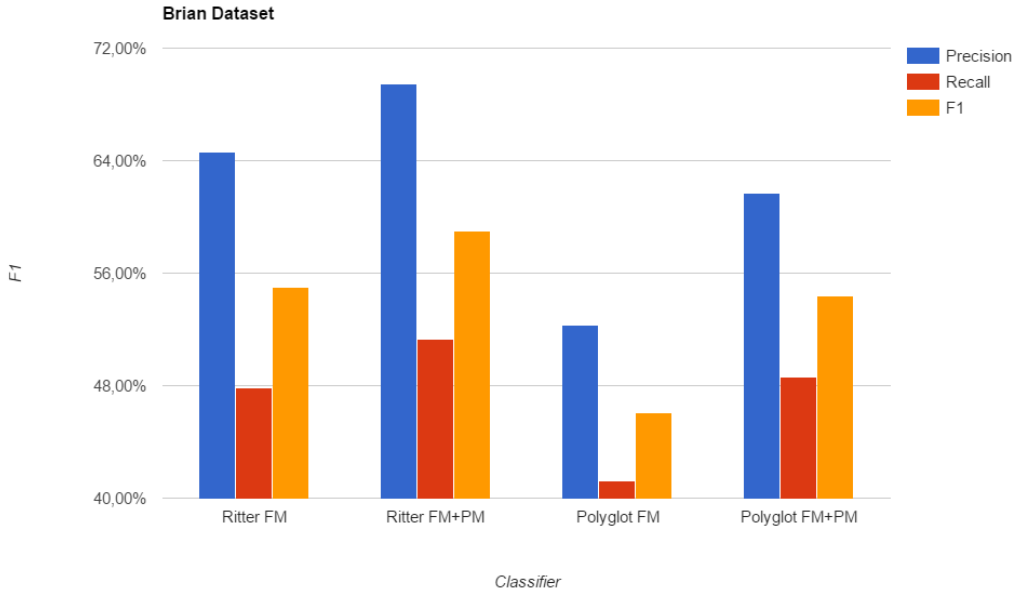

Figure 5.1: Brian Dataset scores

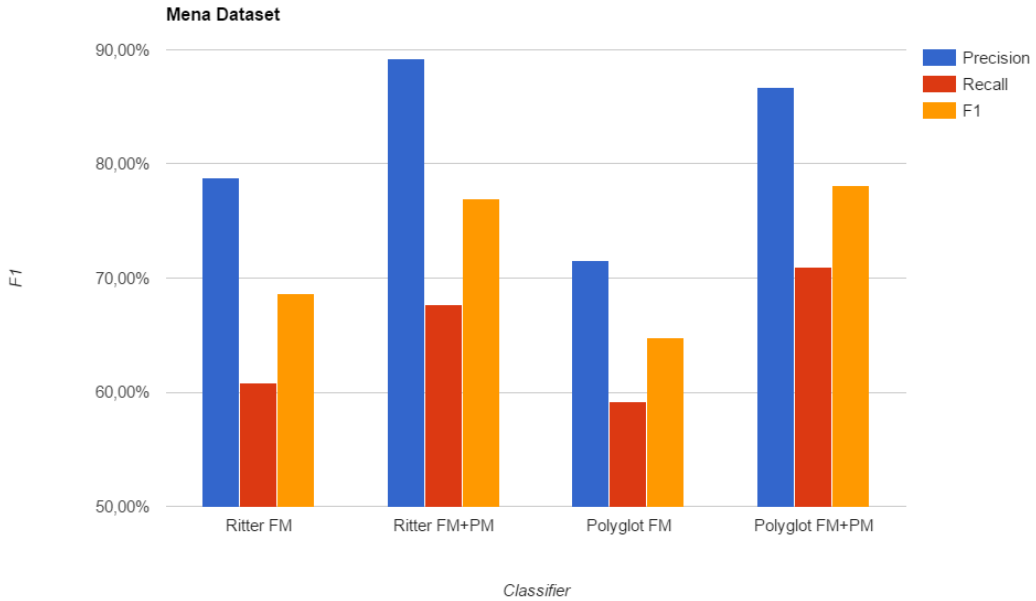

Figure 5.2: Mena Dataset scores

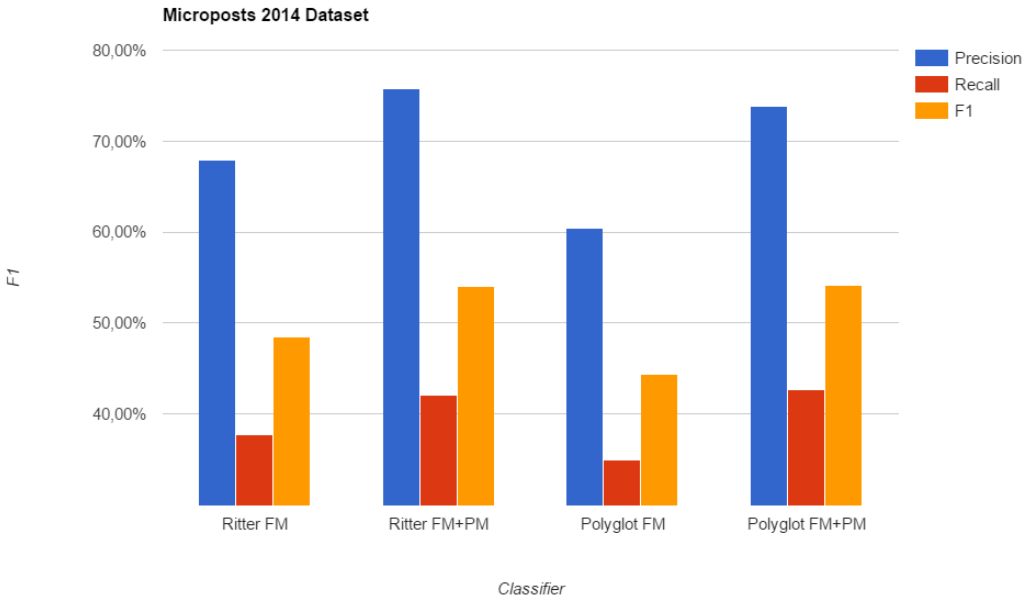

Figure 5.3: Microposts2014 Dataset scores

5.3 TASS 2015 Challenge

TASS 2015 is an experimental evaluation challenge for sentiment analysis focused on Spanish language and social media texts, specifically tweets. This challenge consisted in analysing sentiments, by full tweets or by aspects, in three corpus that were provided.

Two tasks were proposed. The first one consisted in analysing sentiments at a global level on each tweet, while the second one involved aspect-based sentiment analysis, where systems needed to identify sentiment polarity for each aspect of each tweet. The aspects were players, coaches, match, broadcast, referee and many more. Two corpora were provided for this task: the Social-TV corpus, which included tweets collected during a football match, and the STOMPOL corpus, that consisted on tweets related to a political election campaign.

The task I assumed was to identify named entities from the Social-TV corpus for the aspect-based sentiment analysis task. The named entities I had to identify were footballers, coaches and teams names. In addition to the NER task, I was also assigned with a NED task. This consisted in not only recognizing entities that fell into one of the classes mentioned before, but to identify which specific player, coach or team they referenced.

For the NER task, we decided to use the Stanford CRF NER. It includes a Spanish model, but as it was trained on news article corpora, it performed poorly on tweets. Therefore, we decided to train a model with it, using the training files provided by the organizers and a gazette. The gazette entities were collected from the training file, resulting in a list of all the ways these players, coaches or teams were named. Two types of gazettes were tested: clean and sloppy. Clean gazettes give a match when the whole text from an entity is detected, and sloppy gazettes give it when a word from the entity appears. They produced similar F1 scores, with clean gazettes resulting in a better recall, while sloppy ones gave better precision.

Then, k-fold cross-validation was applied to the training corpus, each iteration giving similar scores, with an average precision of 94,09% and a recall of 88,35%, resulting in an average F1 of 91,05%.

The aspect-based task of the challenge required that the footballers or teams were classified, identifying the specific player or team. For this purpose, we used the list of all the ways these entities were named, and we searched for full or at least partial matches. The reference (the player or the team) with more matches was assigned as the correct one.

5.4 Ensemble Results

As said earlier, we used FOX to train ensemble classifiers. We focused on Spanish NER, given the fact that the FOX framework did not include any Spanish classifier, but English and German ones.

We used two different datasets from CoNLL-2002, that were provided as testing sets. We will call them CoNLL-a and CoNLL-b. Table 5.2 shows the number of entities by class that each dataset contains. The datasets have a similar size and while the CoNLL-a has a more balanced proportion of entities, CoNLL-b contains fewer Location entities and more Organization and, especially, Person entities.

Table 5.2: Number of entities of each dataset by class

Class	CoNLL-a	CoNLL-b
Location	404	282
Organization	558	671
Person	503	782
Total	1465	1735

We used all the 15 algorithms that were analyzed in [16]. Therefore, we obtained 15 different ensemble classifiers for each dataset. As explained in Chapter 4, the FOX training module also performed a k-fold cross-validation, returning the scores. These scores included precision, recall and F1. However, these scores were provided by class and did not include global scores. To obtain the global scores, we calculated the weighted average of the class scores, using the number of entities of each class as weights.

Tables 5.3 and 5.4 show the global precision, recall and F1 obtained in the cross-validation of the 15 ensembles and the 4 single classifiers. Figures 5.4 and 5.5 present the f-scores divided by classes from the 4 single classifiers, the simple Vote algorithm and the two best classifiers on each dataset.

As the tables show, the ensemble algorithms clearly outperform the base classifiers and the Voting approaches. Simple Vote obtains poor results, being even outmatched by the Stanford NER, which was the best base classifier on all the datasets and entity classes. Due to this fact, CVote obtained the same measures as Stanford, as its approach consisted in selecting the classifier with best results for each class.

The best results were obtained in the CoNLL-a dataset. In this dataset, LMT and RF were the best performing algorithm. LMT leads in precision (80.46%) and F1 (77.99%), outperforming RF by +0.11% more F1. However, the ensemble with best recall was the Naïve Bayes with 81.35%, although its results were harmed by its low precision (68.05

If we compare the ensemble scores with the ones obtained by the base classifiers, we observe that the best ensemble, LMT, achieves +6.56% more F1 than the best single classifier, Stanford. If we choose simple Vote as the baseline classifier, as [16] did, we see a +34.34% difference in precision, a +1.15% difference in recall and a +21.02% difference in F1 between LMT and this algorithm, with LMT leading the comparison in all measures.

With the CoNLL-b dataset, the results were a bit poorer. In this dataset, the best performing ensemble algorithm is LMT again, followed by MLP, BG and DT before RF. LMT outperforms MLP by +1.69% in precision and +0.34% in F1. However, it falls behind MLP in recall by a -1.08% difference. LMT leads again in F1 (76.25%) obtaining +0.34% more than MLP, as NB does in recall (78.50%), outperforming LMT by +1.94%. However, this time the classifier with the best precision is RF (80.91%), outperforming LMT by +4.96

The best performing single classifier was Stanford again, leading in all measures and entity classes. It outperforms the simple Vote algorithm again with +22.44% more precision, -3.36% in recall and +12.81% in F1. Ensembles perform better than the single classifiers again, with LMT obtaining +7.72% more F1 than Stanford. LMT also outperforms Vote by +30.85% in precision, +3.66% in recall and +20.53% in F1.

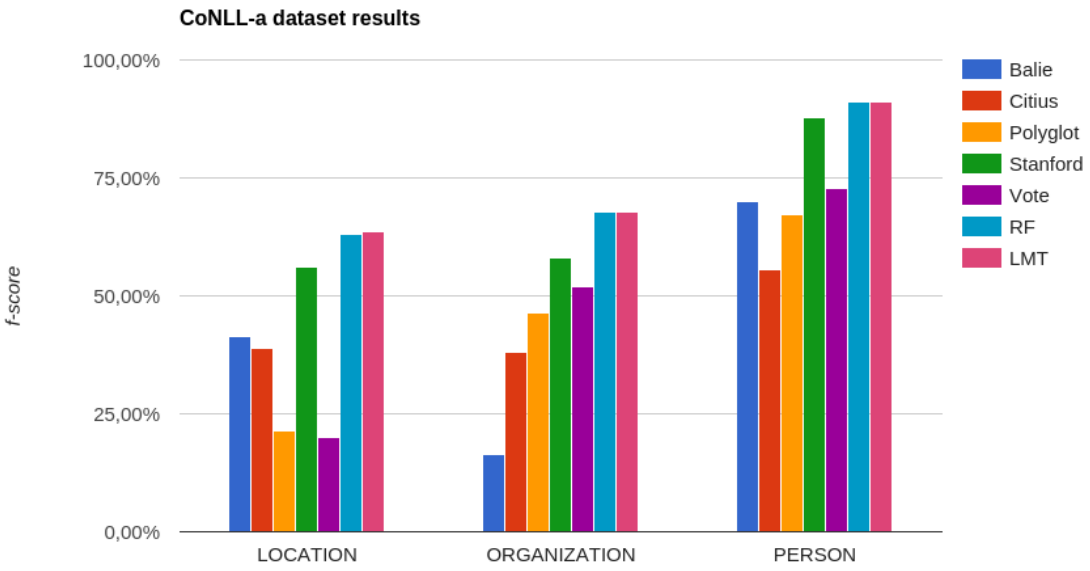

Figure 5.4: CoNLL-a scores

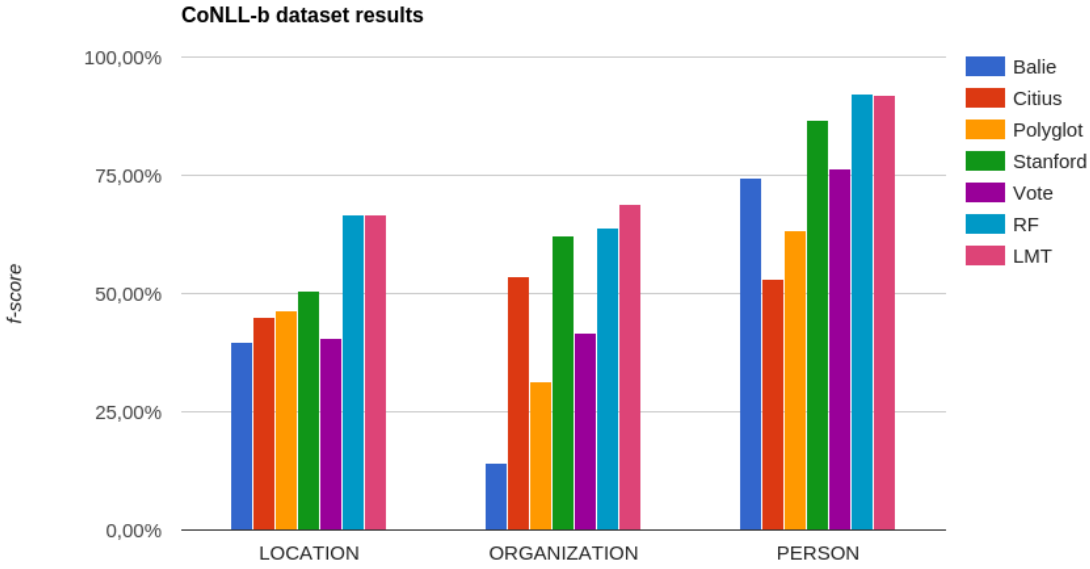

Figure 5.5: CoNLL-b scores

Table 5.3: CoNLL-a ensemble results

Classifier	Precision	Recall	F1
LMT	**80.46**	75.66	**77.99**
RF	80.16	75.73	77.88
BG	79.76	75.62	77.64
J48	79.93	75.27	77.53
ABM1	79.93	75.27	77.53
FT	78.90	76.06	77.46
DT	80.06	74.94	77.42
MLP	79.20	74.96	77.02
NB	68.05	**81.35**	74.11
SVM	78.98	69.66	74.03
SMO	75.82	69.67	72.62
Log	77.62	67.70	72.32
CVote	67.01	76.46	71.43
Stanford	67.01	76.46	71.43
LogB	75.44	67.61	71.31
Vote	46.12	74.51	56.97
Polyglot	55.87	53.68	54.75
Citius	47.79	63.61	54.57
Balie	69.57	37.62	48.83

Table 5.4: CoNLL-b ensemble results

Classifier	Precision	Recall	F1
LMT	75.95	76.56	**76.25**
MLP	74.26	77.64	75.91
BG	75.49	76.31	75.90
DT	75.37	76.33	75.85
RF	**80.91**	70.18	75.17
J48	80.47	69.77	74.74
ABM1	80.47	69.77	74.74
FT	80.17	69.85	74.65
SVM	75.06	72.57	73.79
SMO	73.24	73.35	73.29
NB	65.33	**78.50**	71.31
LogB	75.55	64.78	69.75
Log	75.01	65.05	69.68
CVote	67.54	69.54	68.53
Stanford	67.54	69.54	68.53
Vote	45.10	72.90	55.72
Citius	58.95	47.97	52.89
Polyglot	44.34	53.71	48.58
Balie	64.55	35.05	45.43

CHAPTER 6

Conclusions and future work

In this chapter, we analyze the conclusions that we extracted from the project, and propose some lines for future work.

6.1 Conclusions

In this project we have developed Nerdy, a tool that provides different NER classifiers as a service in a common format, NIF. By using this common format, we make it easier the training, evaluation and combination of these classifiers. We have also included in this tool specific annotation, evaluation and training functionalities, although they can be substituted or extended easily thanks to the use of the NIF format. Users can extract named entities from text they provide or use annotated documents to evaluate and compare the classifiers.

We also provide ensemble classifiers as a service included in Nerdy. We used the FOX framework to train and evaluate these ensembles. To perform Named Entity Recognition in Spanish, we had to extend this tool. Therefore, we included new NER classifiers and trained new ensembles using them and also Spanish datasets. We also communicated FOX with Nerdy so that FOX could use Nerdy classifiers and Nerdy could offer the ensemble classifiers trained with FOX as a service.

Nerdy uses various technologies and during the development of this project, we have learnt to use them and take advantage of them. Nerdy use Senpy to create the web service, Docon to convert document formats, Webanno to annotate documents, FOX to train and evaluate ensembles and Weka to provide the ensemble learning algorithms. We have also used NIF as a common format, and numerous NER classifiers in Spanish and English to provide the Nerdy service and to train ensemble classifiers.

The project is divided in multiple modules, including annotation, NER training and evaluation, ensemble training, web service and so on. By developing the project in separate parts, we make it easy to extend or even substitute any of these parts in new projects. As we have said above, using a common format contributes to this purpose.

6.2 Achieved goals

- **Create a web service that provides Named Entity Recognition** The main goal of this project was to develop a service that allows the user to extract Named Entities from documents. The entities extracted by the NER classifiers are provided in the NIF format to allow easy integration and comparison. The implementation of this service is detailed in Chapter 3.
- **Evaluate classifiers on Spanish NER and on the Twitter domain.** We looked into research on these fields and integrated and evaluated specific tools. The research review can be found in 2.2 and the evaluation is detailed in Chapter 5.
- **Develop utilities for NER annotation, training and evaluation.** Nerdy integrates various tools to annotate documents, train new classifiers and evaluate and compare them with different datasets. These tools facilitate the integration with other classifiers and datasets. These utilities are detailed in 3.2.
- **Train and evaluate ensemble NER classifiers.** The FOX framework was integrated in Nerdy to train new ensembles and test them. This framework was extended with Spanish NER classifiers and the ensemble classifiers were trained on Spanish datasets. The implementation and extension of FOX is expound in Chapter 4.

6.3 Future work

Nerdy can be extended and improved in several areas. In this section we present some of the paths to extend Nerdy that could not be implemented due to time limitations.

- Extend Nerdy with different NER classifiers and evaluate them.
- Train new NER classifiers on new domains.
- Train and test new ensembles using additional single classifiers or training data from other domains, such as Twitter.
- Add the possibility of training new ensembles or single NER classifiers through the web service interface, making them available later from the service.
- Include format converting utilities through the web service to fully integrate this task with the NER functionalities.
- Integrate the Webanno-based annotator in the service, so that results from the NER classifiers can be displayed on this tool.

Bibliography

[1] Rami Al-Rfou, Vivek Kulkarni, Bryan Perozzi, and Steven Skiena. Polyglot-NER: Massive multilingual named entity recognition. *Proceedings of the 2015 SIAM International Conference on Data Mining, Vancouver, British Columbia, Canada, April 30 - May 2, 2015*, April 2015.

[2] Mena Badieh Habib Morgan and Maurice van Keulen. Named entity extraction and disambiguation: The missing link. In *Proceedings of the sixth international workshop on Exploiting semantic annotations in information retrieval*, pages 37–40. ACM, 2013.

[3] Amparo E Cano, Giuseppe Rizzo, Andrea Varga, Matthew Rowe, Milan Stankovic, and Aba-Sah Dadzie. Making sense of microposts:(# microposts2014) named entity extraction & linking challenge. In *CEUR Workshop Proceedings*, volume 1141, pages 54–60, 2014.

[4] Abhimanyu Chopra, Abhinav Prashar, and Chandresh Sain. Natural language processing. *International Journal of Technology Enhancements and Emerging Engineering Research*, 1(4):131–134, 2013.

[5] Leon Derczynski, Diana Maynard, Giuseppe Rizzo, Marieke van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak, and Kalina Bontcheva. Analysis of named entity recognition and linking for tweets. *Information Processing & Management*, 51(2):32–49, 2015.

[6] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[7] Pablo Gamallo, Juan Carlos Pichel, Marcos Garcia, José Manuel Abuín, and Tomás Fernández Pena. Análisis morfosintáctico y clasificación de entidades nombradas en un entorno big data. *Procesamiento del Lenguaje Natural*, 53:17–24, 2014.

[8] Abhishek Gattani, Digvijay S Lamba, Nikesh Garera, Mitul Tiwari, Xiaoyong Chai, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. Entity extraction, linking, classification, and tagging for social media: a wikipedia-based approach. *Proceedings of the VLDB Endowment*, 6(11):1126–1137, 2013.

[9] Parth Gupta, Khushboo Singhal, and Paolo Rosso. Multiword named entities extraction from cross-language text re-use.

[10] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[11] Chenliang Li, Jianshu Weng, Qi He, Yuxia Yao, Anwitaman Datta, Aixin Sun, and Bu-Sung Lee. Twiner: named entity recognition in targeted twitter stream. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 721–730. ACM, 2012.

[12] Brian William Locke. Named entity recognition: Adapting to microblogging. 2009.

[13] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[14] Alan Ritter, Sam Clark, Oren Etzioni, et al. Named entity recognition in tweets: an experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534. Association for Computational Linguistics, 2011.

[15] Rahul Sharnagat. Named entity recognition: A literature survey. 2014.

[16] René Speck and Axel-Cyrille Ngonga Ngomo. Ensemble learning for named entity recognition. In *The Semantic Web–ISWC 2014*, pages 519–534. Springer, 2014.

[17] Seid Muhie Yimam, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. Webanno: A flexible,web-based and visually supported system for distributed annotations. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (System Demonstrations) (ACL 2013)*, pages 1–6, Stroudsburg, PA, USA, August 2013. Association for Computational Linguistics.