



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería  
Informática**

**Automatic Data Recognition  
System in Natural Language**



Presentado por Malte Janssen  
en Universidad de Burgos — June 25, 2019  
Tutor: Bruno Baruque Zanón







UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



D. D. Bruno Baruque Zanón, profesor del departamento de Ingeniería Civil, Lenguajes y Sistemas Informáticos. Expone:

Que el alumno Malte Janssen, con DNI L1W10HMVC, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, June 25, 2019

Vº. Bº. del Tutor:

D. Bruno Baruque Zanón





## Resumen

Hoy en día, alrededor de 3.500 millones de personas utilizan las redes sociales de forma regular[<https://hootsuite.com/pages/digital-in-2019>]. Con tanta gente usando los medios sociales, mucha información fluye a través de las plataformas sociales. Esto lleva a un creciente interés por extraer y analizar esa información.

El reconocimiento de entidades nombradas (NER) es una subtarea del Procesamiento del Lenguaje Natural (NLP). Se centra en la extracción y clasificación de entidades nombradas de textos.

El reconocimiento de las entidades nombradas en los textos informales plantea varios retos. Este trabajo trata de abordar esos retos. Se compone de dos enfoques. El primer enfoque del trabajo consiste en aplicar diferentes métodos de aprendizaje automático de las bibliotecas Scikit-learn y NLTK. En la segunda parte se propone un modelo más complejo utilizando deep learning. Ambos enfoques se comparan para seleccionar el mejor enfoque y la mejor configuración para el problema.

Además, se desarrolla un sencillo web-app para mostrar el modelo final.

Traducción realizada con el traductor [www.DeepL.com/Translator](http://www.DeepL.com/Translator)

## Descriptores

Procesamiento del lenguaje natural, reconocimiento de entidades nombradas, aprendizaje automático, redes neuronales

## **Abstract**

Today about 3,5 billion people use social media on a regular basis[<https://hootsuite.com/pages/digital-in-2019>]. With that many people using social media a lot of information flows through social platforms. This leads to increasing interest to extract and analyse that information.

Named Entity Recognition (NER) is a sub-task of Natural Language Processing (NLP). It focuses on extracting and classifying named entities from text.

The recognition of named entities in informal texts poses several challenges. This work tries to tackle those challenges. It is made up by two approaches. The first approach of the work consists of applying different machine learning methods from the libraries Scikit-learn and NLTK. In the second approach a more sophisticated model is proposed using deep learning. Both approaches are then compared in order to select the best approach and configuration for the problem. Furthermore a simple web-app is developed to showcase the final model.

## **Keywords**

Natural Language Processing, Named Entity Recognition, Machine Learning, Neural Networks



---

# Contents

---

<b>Contents</b>	<b>iii</b>
<b>List of Figures</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>Abbreviations</b>	<b>ix</b>
<b>Introduction</b>	<b>1</b>
<b>Objectives</b>	<b>3</b>
<b>Theoretical concepts</b>	<b>5</b>
3.1 Natural Language Processing(NLP) . . . . .	5
3.2 Named Entity Recognition(NER) . . . . .	6
3.3 Named Entity Recognition on informal Texts . . . . .	6
<b>Techniques and Tools</b>	<b>7</b>
4.1 Tools/Libraries . . . . .	7
Model Libraries . . . . .	7
NLTK . . . . .	7
Scikit-learn . . . . .	7
Pytorch . . . . .	8
Web-app . . . . .	8
Flask . . . . .	8
Heroku . . . . .	8
4.2 Techniques . . . . .	8
NER - Process . . . . .	8

Sentence Segmentation . . . . .	8
Word - Tokenization . . . . .	9
Part of Speech Tagging (POS Tagging) . . . . .	9
Penn Treebank Tag Set . . . . .	9
Chunking (Entity Detection) . . . . .	9
Applied Models . . . . .	10
N-gram Chunking . . . . .	10
Feature Chunking . . . . .	11
Decision Tree . . . . .	12
Discriminative vs Generative Models . . . . .	13
NaiveBayes . . . . .	13
Gaussian Naive Bayes . . . . .	14
Multinomial Naive Bayes . . . . .	15
Maxent . . . . .	15
Support Vector Machines . . . . .	15
One vs the rest . . . . .	16
One vs. one . . . . .	17
Gradient Boosting Classifier . . . . .	17
Random Forest Classifier . . . . .	17
Neural Networks . . . . .	17
Conditional Random Fields . . . . .	18
<b>Relevant aspects of the development of the project</b>	<b>19</b>
5.1 Dataset . . . . .	19
5.2 NLTK and Scikit-learn . . . . .	20
Parameter Search . . . . .	20
SVM . . . . .	21
Decision Tree . . . . .	22
Random Forest . . . . .	23
Gradient Boosting . . . . .	24
Bernoulli Naive Bayes . . . . .	24
Multinomial Naive Bayes . . . . .	25
Logistic Regression . . . . .	25
Feature-Extraction . . . . .	25
Results . . . . .	25
5.3 Architecture of Neural Network . . . . .	25
CRF . . . . .	25
LSTM . . . . .	26
bidirectional LSTM . . . . .	28
Dropout . . . . .	28
Word-Embedding . . . . .	28

CONTENTS

v

Character-Embedding . . . . .

28

Custom-Initialisation . . . . .

28

Parameter Search . . . . .

28

Results . . . . .

28

5.4 Comparison . . . . .

28

. . . . .

28

. . . . .

28

Related works

29

Conclusions and future lines of work

31

Bibliography

33

---

## List of Figures

---

3.1	Example of entities in a phrase Source: <a href="https://github.com/floydhub/named-entity-recognition-template">https://github.com/floydhub/named-entity-recognition-template</a> . . . . .	6
4.2	Process . . . . .	9
4.3	Example of Chunking . . . . .	10
4.4	caption . . . . .	12
4.5	caption . . . . .	13
4.6	Seperation by a hyperplan - Source: <a href="https://scikit-learn.org/stable/modules/svm.html">https://scikit-learn.org/stable/modules/svm.html</a> . . . . .	1
4.7	Design of a Neural Network Source: <a href="https://medium.com/xanaduai/making-a-neural-network-quantum-34069e284bcf">https://medium.com/xanaduai/making-a-neural-network-quantum-34069e284bcf</a> . . . . .	18
5.8	Red RNN . . . . .	26
5.9	RNN . . . . .	27
5.10	LSTM . . . . .	27
5.11	Red LSTM bidirectional . . . . .	28

---

# List of Tables

---

4.1	Penn Treebank Tagset . . . . .	10
5.2	. . . . .	20
5.3	. . . . .	22
5.4	. . . . .	22
5.5	. . . . .	23
5.6	. . . . .	23
5.7	. . . . .	24
5.8	. . . . .	25



---

# Abbreviations

---

**NLP** Natural Language Processing

**NER** Named Entity Recognition

**CoNLL** Conference on Natural Language Learning

**SVM** Support Vector Machine

**CRF** Conditional Random Field





---

# Introduction

---

A common task in information extraction is the identification of named entities from text, called Named Entity Recognition (NER). NER is typically formulated as a sequence prediction problem, where for a given sequence of tokens, an algorithm or model needs to predict the correct sequence of labels. Typically such models are able to produce good results on traditional, formal texts (eg.: newspaper corpora). However, noisy and user generated informal texts, which are common on social media, pose several challenges for generic NER systems, such as shorter texts and improper grammar. Two approaches to tackle this challenging task were taken. First traditional machine learning techniques from the libraries scikit-learn and NLTK were used to if they are able to produce acceptable results. Secondly a more sophisticated model using a deep learning was applied in hopes of better results.

In the following paper, the used models are presented by examining the main objectives, underlying theories and way of proceeding as well as describing encountered challenges in the course of the project.



---

# Objectives

---

This project aims to study the usefulness of several machine learning models on noisy/informal texts. Also a more sophisticated approach is proposed in order to improve performance.

The following steps are taken:

- Prepare and read the Data so it can be fed into the classifiers
- Train the NLTK and scikit-learn classifiers. To do this a script is written.
- Improve the performance of the models. This includes a parameter search, finding a appropriate feature extraction function, and text preprocessing.
- Propose a more sophisticated model to tackle the task. To do this a neural network was chosen and combined with a CRF.
- Build a web-app to showcase the proposed model



---

# Theoretical concepts

---

This section will contain descriptions of fundamental concepts applied to the project.

## 3.1 Natural Language Processing(NLP)

Natural language processing is a field of computer science and plays a big part in it's sub-field of artificial intelligence.

Some of today's use cases are information extraction, machine translation, summarization, search and human-computer interfaces.<sup>1</sup>

It describes techniques in which natural language is processed into a formal representation which computer can understand so they can further work with the data. In order to do that the spoken or written human language has to be analysed. It is not enough for the computer to know the significance of each word or even each sentence. Otherwise a simple dictionary could be easily used. In order to understand a text the capture of complete text correlations is necessary. This is a challenge for computers because of the complexity of human language and it's ambiguity. Computer unlike humans can't use their experience to understand a text but rather have to use artificial language algorithms and techniques. One of those techniques is called Named Entity Recognition(NER).

---

<sup>1</sup>Cf. Jason Collobert Ronan; Westan. *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning*. NEC Labs America.

## 3.2 Named Entity Recognition(NER)

Named entity recognition refers to the extraction of important information from a text. In named entity recognition the task is to identify which information is important and to then categorise this information. This results in named entities, that are particular terms in a text that are more informative than others or have a unique context. It can be used as a source of information for different NLP applications, such as answering questions, automatic translation or information retrieval.<sup>2</sup>

Figure 3.1 shows a sentence and it's possible entities.

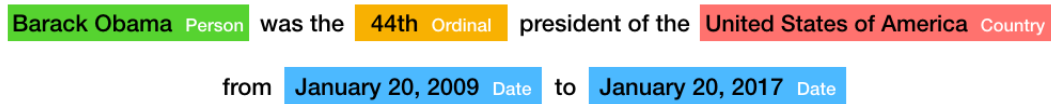


Figure 3.1: Example of entities in a phrase Source:  
<https://github.com/floydhub/named-entity-recognition-template>

## 3.3 Named Entity Recognition on informal Texts

The recognition of named entities in informal texts (eg.: social media posts) is a challenge. This is because they are written inherently differently than formal texts. Unlike formal texts, most social media posts do not follow strict rules, have different grammatical structures, contain misspelled words, informal abbreviations and multilingual vocabulary. They are also relatively short. For example tweets have a 140 character limit, so not much context can be extracted. For these reasons named entity recognition is a much harder task on informal texts.

---

<sup>2</sup>Cf. Behrang Mohit. *Natural Language Processing of Semitic Languages*. 2014, p. 221.

---

# Techniques and Tools

---

In this chapter, the tools and technologies used during the course of the project are presented. The different types of models used to do NER as well as the steps taken to process text data are described.

## 4.1 Tools/Libraries

This section introduces the main libraries that are used in the project.

### Model Libraries

#### NLTK

The Natural Language Toolkit (NLTK) is a collection of libraries and programs for linguistic applications in the programming language Python. It was chosen because it provides a practical introduction to language processing as well as it doesn't only provide machine learning models, but also includes several function to process data (eg.: tokenisation).

#### Scikit-learn

Scikit-learn is another machine learning library for the programming language Python.

On top of using NLTK's machine learning models Scikit-learn's models were applied. They were added because scikit-learn provides a wider selection of supervised learning models than NLTK.

## Pytorch

PyTorch is a machine learning library for the programming language Python that is based on the Torch library. It is used for deep learning and natural language processing applications.

## Web-app

### Flask

To develop the web-app two popular web-frameworks were looked at. Django and Flask. Django is a full-stack web framework, whereas Flask is a micro and lightweight web framework. Since the web-application is to only act as a demo, to visualise the named entity recognition, Flask was chosen to keep the web application simple.

### Heroku

Heroku is a platform-as-a-service that enables users to run their applications in the cloud. It is used to host the demo application.

## 4.2 Techniques

### NER - Process

Figure 4.2 highlights the process of Classifying words given as raw text. The diagram is applicable for the NLTK and Scikit-learn Classifiers. The only difference in the proposed deep learning model is that the part of speech tagging is omitted since it uses different features. Each step will be expanded on in the following subsections.

### Sentence Segmentation

Sentence segmentation is the division of a text into its component sentences. Eg. (the squared brackets are not to be seen as lists but rather as indicators of separate elements): [Hamburg won the Champions League 1983. It was their biggest feat to date.] -> [Hamburg won the Champions League 1983.], [It was their biggest feat to date.]



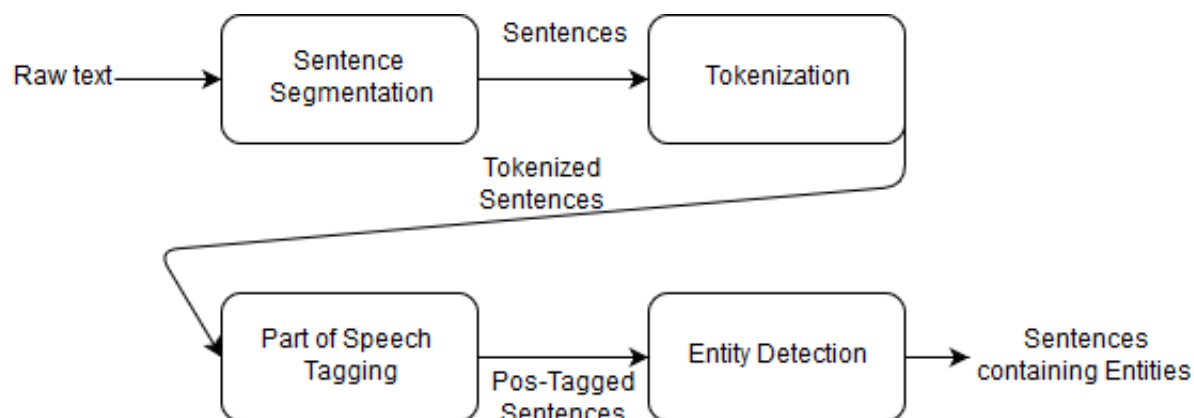


Figure 4.2: Process

### Word - Tokenization

Word Tokenization describes the process of dividing Sentences into it's component words. Eg.: [Hamburg won the Champions League 1983.] -> [Hamburg], [won], [the], [Champions], [League], [1983], [.]

### Part of Speech Tagging (POS Tagging)

In POS Tagging each word of a text is assigned a parch of speech token. POS stands for part of speech, and provides syntactic information about a word. The tagset used by the NLTK Tagger is the Penn Treebank tag-set.

**Penn Treebank Tag Set** The Penn Treebank Tag set uses the tags shown by table 4.1.

### Chunking (Entity Detection)

Chunking is a step following POS tagging and structures a sentence into "chunks". Sentences are chunked into the IOB format. Each token is tagged with one of three chunk tags, I (inside), O (outside), B (begin). The B-token marks the beginning of a chunk. All subsequent tokens belonging to the chunk are marked with I. Behind the B and I Tag the chunk type (entity class) is added. The chunk type describes the type of entity (eg.: Person).

The figure 4.3 shows an example. Here the sentence: "The Hamburger SV won last weeks game against Werder Bremen." was chunked into 2 chunks (Hamburger SV and Werder Bremen). In each case the B-<type>

1.	CC	Coordinating conjunction	19.	PRP\$	Possessive pronoun
2.	CD	Cardinal number	20.	RB	Adverb
3.	DT	Determiner	21.	RBR	Adverb, comparative
4.	EX	Existential there	22.	RBS	Adverb, superlative
5.	FW	Foreign word	23.	RP	Particle
6.	IN	Preposition or subordinating conjunction	24.	SYM	Symbol
7.	JJ	Adjective	25.	TO	to
8.	JJR	Adjective, comparative	26.	UH	Interjection
9.	JJS	Adjective, superlative	27.	VB	Verb, base form
10.	LS	List item marker	28.	VBD	Verb, past tense
11.	MD	Modal	29.	VBG	Verb, gerund or present participle
12.	NN	Noun, singular or mass	30.	VBN	Verb, past participle
13.	NNS	Noun, plural	31.	VBP	Verb, non-3rd person singular present
14.	NNP	Proper noun, singular	32.	VBZ	Verb, 3rd person singular present
15.	NNPS	Proper noun, plural	33.	WDT	Wh-determiner
16.	PDT	Predeterminer	34.	WP	Wh-pronoun
17.	POS	Possessive ending	35.	WP\$	Possessive wh-pronoun
18.	PRP	Personal pronoun	36.	WRB	Wh-adverb

Table 4.1: Penn Treebank Tagset

The Hamburger SV won last weeks GAME against Werder Bremen .  
O B-Group I-Group O O O O O B-Group I-Group O

Figure 4.3: Example of Chunking

marks the beginning of the chunk, while the next word in the chunk is marked with the I-<type> tag. All other tokens(words) are tagged as O, meaning they are not in a chunk.

## Applied Models

### N-gram Chunking

N-gram taggers use a simple statistical tagging algorithm. Each token is assigned the tag that is most likely it's type based on it's POS-tag. It uses so called n-grams which are subsequences of n items. The n amount of previous words and POS-tags are used to guess the tag for the current word. This information is saved inside a context Dictionary maintained by the tagger. A Unigram Tagger for example doesn't use any context and only uses the POS-tag of the current word to assign a tag. Such a Unigram tagger would for example not be able to differentiate between "the wind" or "to wind", which are different types of word and have completely different meaning. A Bigram tagger detects that "the" and "to" have different POS-tags though and could therefore potentially detect the difference in meaning.

### Feature Chunking

Often it is not enough to just look at the POS-tag of a words, more information need to be included. This information is determined by a feature extraction-function. One example is shown in listing 4.1. In this example the feature function generates a feature set that contains the current, previous, and next word and part-of-speech tag, along with the previous IOB tag.

Listing 4.1: example of feature function

```
def prev_next_pos_iob(tokens, index, history):
    word, pos = tokens[index]

    if index == 0:
        prevword, prevpos, previob = ('<START>',) * 3
    else:
        prevword, prevpos = tokens[index - 1]
        previob = history[index - 1]

    if index == len(tokens) - 1:
        nextword, nextpos = ('<END>',) * 2
    else:
        nextword, nextpos = tokens[index + 1]

    feats = {
        'word': word,
        'pos': pos,
        'nextword': nextword,
        'nextpos': nextpos,
        'prevword': prevword,
        'prevpos': prevpos,
        'previob': previob
    }
    return feats
```

Figure 4.4 shows the classification process using such a feature extraction-function. During training each input is transformed into a feature set, which contains the information on which the input is to be classified on. The machine learning algorithm takes these feature-sets with their corresponding labels and creates a model.

During prediction the same feature extractor-function is used to convert the input into features, which are then used by the model to predict the corresponding label.

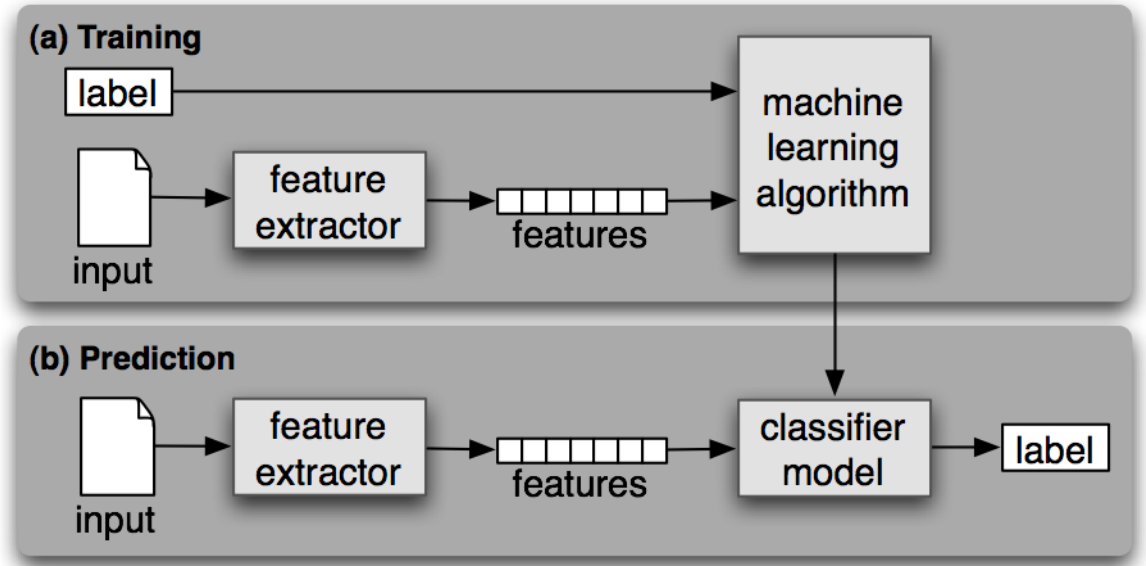


Figure 4.4: Classification Process

source: <https://www.nltk.org/book/ch06.html>

### Decision Tree

A decision tree (4.5) is a flowchart-like structure that can be used for classification. The Tree is made up of decision- and leaf-nodes. Decision nodes correspond to a feature which is checked inside. The process starts at the root leaf and continues through decision nodes until a leaf node is reached, which assigns a label.

The decision tree is build in the training phase through entropy and information gain. Information gain measures how much the organisation of input values increase. To do that the entropy (disorganisation) of their labels has to be calculated. Low entropy would mean that most input values correspond to the same value while high entropy would mean that the labels of the input labels vary widely.

$$entropy = - \sum p(label) \log p(label) \quad (4.1)$$

The entropy formula (4.1) shows that either a high amount of inputs from the same label (low  $\log P(l)$ ) or a low amount of input from the same label (small  $P(l)$ ) lead to low entropy. While a wide amount of lables with medium frequency would lead to a high entropy.

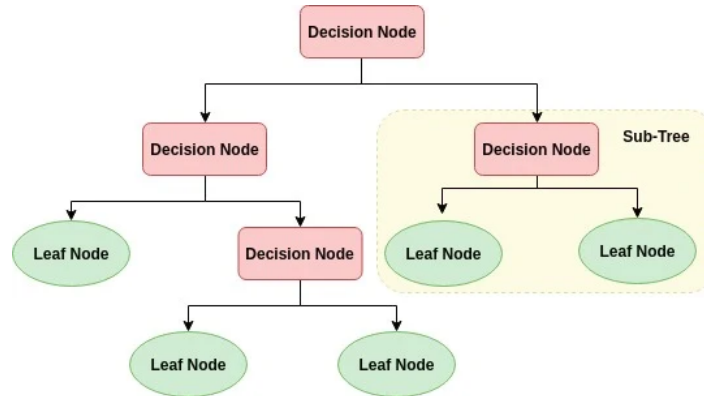


Figure 4.5: Representation of a Decision Tree

source: <https://www.datacamp.com/community/tutorials/decision-tree-classification-python>

$$InformationGain = Originalentropy - newentropy \quad (4.2)$$

The decision Tree is made up by single decision trees called decision stumps. To construct the tree these decision stumps are added. To decide on which feature is used in the first decision node the entropy of all input attributes is calculated. To do that the average entropy of it's leafs is calculated. Then the information gain can be calculated using formula (4.2). The decision tree is then build by selection selecting the decision stumps with the highest information gain.

### Discriminative vs Generative Models

Discriminative models calculate conditional probabilities ( $P(label|input)$ ) instead of the joint probability ( $P(input, label)$ ) generative models calculate.

**NaiveBayes** The Naive Bayes Classifier is a discriminative model and uses an algorithm based on the Bayes Theorem. The algorithm assumes that features are independent of each other (class-conditional independence), meaning that each feature contributes independently towards the probability of a certain result, ignoring possible correlations. (naive)

The Bayes Theorem states the following:

$$posterior = \frac{priori * likelihood}{evidence} \quad (4.3)$$

Knowing that the following equation can be made.

$$P(Class|features) = \frac{P(Class)P(features|Class)}{P(features)} \quad (4.4)$$

Features is a dependent feature vector of size n:

$$features = (f1, f2, f3, \dots, fn) \quad (4.5)$$

The Priori is the prior probability of a class. It gets calculated by simply determining the frequency of it in the training set.

Due to the assumed class conditional independence among the features  $P(features|Class)$  can be rewritten as such:  $P(f1|Class)*\dots*P(fn|Class)$

This gets us the following:

$$P(Class|f1, \dots, fn) = \frac{P(f1|Class)\dots P(fn|Class)P(Class)}{P(f1)\dots P(fn)} \quad (4.6)$$

For all inputs, the denominator remains static. Therefore, it is removed and a proportionality is introduced.

$$P(Class|f1, \dots, fn) \propto P(Class) \prod_{i=1}^n P(x_i|Class) \quad (4.7)$$

To determine a class the maximum probability needs to be determined:

$$y = \underset{y}{\operatorname{argmax}} P(y) \prod_{i=1}^n P(x_i|Class) \quad (4.8)$$

Multiple different naive Bayes Classifiers, making different assumptions regarding the distribution of  $P(x_i|Class)$  have been used.

**Gaussian Naive Bayes** Gaussian Naive Bayes assumes that the likelihood of features is Gaussian.

$$P(x_i|y) = \frac{1}{\sigma_y \sqrt{2\pi}} e^{-(x_i - \mu_y)^2 / 2\sigma_y^2} \quad (4.9)$$

$\sigma_y$  and  $\mu_y$  are estimated using the maximum likely-hood method, which is an estimation method in which the values are estimated by taking a sample.

**Multinomial Naive Bayes** Multinomial Naive Bayes assumes a multinomial distribution of the data. The vectors  $\theta_y = (\theta_{y1}, \dots, \theta_{yn})$  parameterize the distribution for each class  $y$  where  $n$  is the number of features and  $\theta_{yi}$  is the probability  $P(x_i | y)$  of a feature  $i$  to appear in class  $y$ .

$\theta_y$  is estimated using relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n} \quad (4.10)$$

$N_{yi} = \sum_{x \in T} x_i$  is the number of times a feature  $i$  appears in the class  $y$ , while  $N_y = \sum_{i=1}^n N_{yi}$  is the count of all features of the class  $y$ . The smoothing parameter  $\alpha$  makes sure there are no zero probabilities (eg. Laplace Smoothing).

**Maxent** Maximum entropy classifiers work in a similar fashion as naive Bayes classifiers. The key difference is that they don't assume that the features are independent. They are discriminative models instead of generative models.

While building the model its parameters are not determined by probabilities like in the Naive Bayes Classifier, but instead it uses search techniques to find the parameter that maximises the total likelihood defined by:

$$P(features) = \sum_x |in|_{corpus} P(label(x) | features(x)) \quad (4.11)$$

$P(label | features)$  is defined by:

$$P(label | features) = P(label, features) / \sum_l P(label, features) \quad (4.12)$$

Dependent features can be complex. That is why the model parameters are determined using iterative optimisation. The parameters are initialised randomly and then optimised in each iteration. The algorithm calculates the empirical frequency (frequency in training set) for each joint feature. It then searches for the distribution which maximises entropy, while still predicting the correct frequency for each joint-feature. Not being able to directly calculate the parameters makes training these models time costly.

## Support Vector Machines

Support Vector machine classifiers create a hyperplane dividing two classes of data. The hyperplane created has the highest distance possible to the

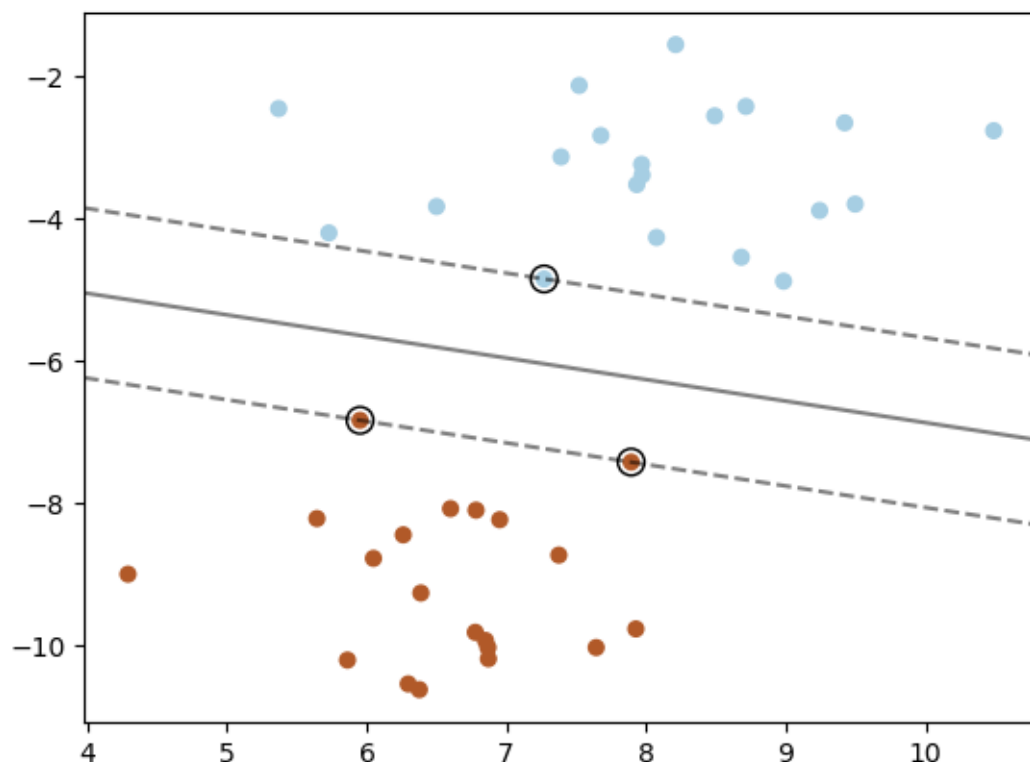


Figure 4.6: Separation by a hyperplan - Source: <https://scikit-learn.org/stable/modules/svm.html>

nearest training example of both classes, which enables a good separation. 4.6 visualises such a deviation.

Two Different concepts are applied to make Support Vectors able to deal with multiple classes: One vs. the rest (as used in scikit LinearSVM) and One vs. one (used in scikit SVC and NuSVC).

**One vs the rest** For each class a binary SVM classifier is trained. One class represents the class while the other represents all other classes. That means that each classifier determines whether an example belongs to its class or any other class. These classifiers don't only return a class label, but also a confidence score to avoid ambiguity. The class of the classifier that classified its own class with the highest confidence score is selected.



**One vs. one** A binary classifier is trained for each pair of classes. Each receives the samples of the pairs target classes from the training set, and learns to distinguish these two classes. While predicting a voting procedure is used to combine the outputs.

### Gradient Boosting Classifier

Gradient Boosting Classifiers are ensemble methods which combine the predictions of several base estimators built with a given learning algorithm in order to improve generalisability. The idea of boosting methods is to build several weak classifiers and to combine them into a strong one. To do that base estimators are being build sequentially and then added to a combined estimator to reduce bias. Gradient Boosting Classifiers sequentially build base classifiers to predict the labels of samples, and calculate the gradient of the loss function in regards to the prediction (difference between the outcome of the learner and the real value).

### Random Forest Classifier

Another ensemble classifier is the Random Forest Classifier. The ensemble is made up by randomised decision trees.

Each tree is constructed with a bootstrap sample (replacement sample). The decision trees are build slightly differently to classical decision trees. Instead of considering all features at the splits only a random subset of features is considered. This randomness results in a higher bias of individual trees.

But since the prediction of the ensemble is given as the averaged prediction of the individual classifiers this increased bias is usually more than compensated, yielding a better result.

### Neural Networks

Neural Networks are networks inspired by the human nervous system. They simulate how the brain processes information. Like the brain they are composed of a large number of highly interconnected processing elements (neurons) working together to solve specific problems. These neurons can generally be differentiated into input-neurons, hidden-neurons and output-neurons. The input-neurons "perceive" information in form of patterns or signals. Hidden-neurons are situated in between input- and output-neurons and map internal information patterns. The output-neurons output the information gained. These neurons are connected by weights,

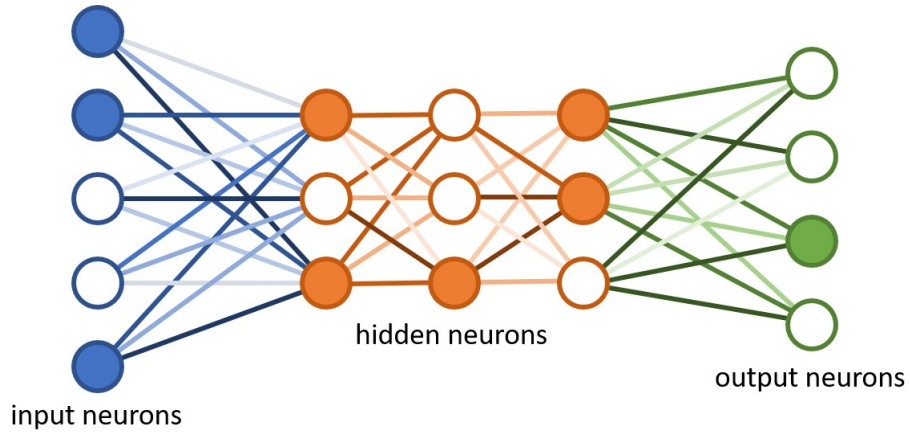


Figure 4.7: Design of a Neural Network Source:  
<https://medium.com/xanaduai/making-a-neural-network-quantum-34069e284bcf>

mapping the output of one neuron to the input of another. A visualisation can be seen in figure 4.7.

The Behaviour of the network is depending on the design of these connections and their strength, which are updated during the training process depending on a learning rule until a stopping criteria is reached (eg. the network performs well on the given task). Usual applications are pattern recognition or data classification.

### Conditional Random Fields

---

# Relevant aspects of the development of the project

---

This chapter of the documentation examines the most relevant aspects of the project development.

## 5.1 Dataset

The WNUT-17 dataset was used to train and evaluate classifiers. The dataset is focused on emerging and rare entities and contains very few surface forms which mostly don't repeat more than once. Most of the surface forms also are not shared by Training and Test Data. The dataset is split into a training, development, and test dataset. All of them are quite a bit different. The train data contains only tweets, while the test data also contains posts from Reddit, Youtube and Stackexchange. The validation data contains Twitter and Youtube data. The reason for these differences is to have posts with more than 140 characters as these exhibit different writing styles and characteristics.

`cite[http://aclweb.org/anthology/W17-4418]`

The data is divided into the following 7 classes:

- Person
- Location
- Corporation (includes Geopolitical Entities)
- Product (tangible goods, or well-defined services)

Metric	Train	Test	Validation
Documents	1000	1,287	1,008
Tokens	65,124	23,394	15,734
Entities	660	1,070	835
person	660	429	470
location	548	150	74
corporation	221	66	34
product	142	127	114
creative-work	140	142	104
group	264	165	39

Table 5.2

- Creative work
- Group
- O (No entity)

Table 5.2 shows how many entities are in the data splits. It is obvious that most words do not belong to an entity (eg.: training data: only 660 entities in 65.124 tokens). Also the entities are not balanced in terms of frequency. It can be said that the dataset is very unbalanced.

## 5.2 NLTK and Scikit-learn

### Parameter Search

A gridsearch was performed for each scikit-learn classifier to get optimal performances. A slightly adapted version of the scikit-learn gridsearch was used, using the available validation dataset instead of scikit-learn's GridCV's standard cross validation using only training data. This was done to save time and also to get more realistic results (as described in the previous section, the validation data looks different than the training data).

In every classifier that gave the option the random state was set, so result are comparable.

The following subsections contain the results of the gridsearch for each classifier. F1 scores were rounded. Furthermore, it also provides a small description of each hyperparameter.

The results are merely a summary of the results, as to include complete

tables would take up too much space with little to no information gain. For the complete results look at the following location: `\NLTK_SKLEARN\results\gridtest`

## SVM

### Tuned parameters:

- **Kernel**
- **C** Penalty parameter of the error term - how much are classification errors penalised. This influences the smoothness of the decision boundary (low c: smooth decision boundary, high C: unsmooth decision boundary (more samples as support vectors)). Bias Variance Tradeoff: large C: low bias, high variance; small C: high bias, low variance
- **Gamma** Gamma defines how far the influence of single training example reaches. If the value of Gamma is high, then the decision boundary will depend on points close to the decision boundary and nearer points carry more weights than far away points.
- **degree** only applies to polynomial kernel function. Defines the flexibility of decision boundary (1 = linear).

Table 5.3 shows that rbf and linear kernels yield the best results. The best result is yielded by a support vector machine with a C value of 10 and a gamma or 0.1. These values result in the best bias-variance tradeoff. As expected too high C values and too low gamma values (lower than 0.1) resulted in worse results, as they lead to high variance. The same can be said for low C values (smaller than 1) and high gamma values (usually bigger than 0.1) in relation to the bias.

The poly Kernel was also tested with different degree values as can be seen in table 5.4. A lower degree resulted in better results, but didn't increase the previously found best score of 0.399.

### Best set of Parameters:

C: 10

Gamma: 0.1

Kernel: rbf

Kernel	C	Gamma	F1-Score	Kernel	C	Gamma	F1-Score
linear	1.0	-	0.393	sigmoid	1	0.1	0.314
linear	10.0	-	0.391	sigmoid	1	1.0	0.163
linear	100.0	-	0.384	sigmoid	1	0.01	0.085
rbf	1	0.1	0.344	sigmoid	10.0	0.1	0.351
rbf	1	0.01	0.170	sigmoid	100.0	0.1	0.326
rbf	1	1.0	0.076	sigmoid	1.0	0.1	0.314
rbf	10.0	0.1	0.399	poly	1	10.0	0.273
rbf	1.0	0.1	0.344	poly	1	1.0	0.273
rbf	0.1	0.1	0.084	poly	1	0.1	0.138

Table 5.3

Kernel	C	Gamma	degree	F1 Score
poly	1.0	10.0	1	0.391
poly	1.0	10.0	2	0.359
poly	1.0	10.0	3	0.273
poly	1.0	10.0	5	0.145

Table 5.4

## Decision Tree

### Tuned parameters:

- **criterion** The function to measure the quality of a split.
- **max\_depth** maximum depth of the tree
- **min\_samples\_leaf** minimum number of samples required to split an internal node
- **min\_samples\_split** minimum number of samples required to be at a leaf node. `min_samples_leaf` and `min_samples_split` decide if a split is made.

The first search (5.5) showed that using information gain (entropy) to measure the quality of a split is slightly better than using gini impurity, although it took a little longer to compute (presumably because of calculating a logarithm).

Furthermore it can be observed that allowing splitting nodes even if it results

criterion	max_depth	min_samples_leaf	F1 score
entropy	100	1	0.375
entropy	100	5	0.369
entropy	50	1	0.373
entropy	50	5	0.316
gini	100	1	0.370
gini	100	5	0.370
gini	50	1	0.356
gini	50	5	0.370

Table 5.5

criterion	max_depth	min_samples_leaf	F1 score
entropy	500	1	0.382
entropy	250	1	FFCCC90.382
entropy	100	1	0.375
entropy	50	1	0.373

Table 5.6

in a node with just one example was ideal ( $\text{min\_samples\_leaf} = 1$ ). Having this parameter small is important in this very unbalanced dataset since in most regions the minority classes (entities) will be very few.

Next the maximum depth of a tree was looked at. The results are showed in table 5.6. High values did not lead to overfitting (at least the ones tested up to 500). Values lower than 250 did not capture all useful pattern. A depth of 250 and 500 yielded the best results. Since the f1 score of both is the same 250 was chosen as the ideal value due to potentially lower time complexity.

The minimum samples to do a split was also looked at as can be seen in table 5.7. Using the minimum amount of 2 yielded the best results.

## Random Forest

**tuned parameters:** For parameters  $\text{min\_samples\_leaf}$ ,  $\text{min\_samples\_split}$ ,  $\text{max\_depth}$  see Decision Trees.

- **n\_estimators** number of trees in the forest

criterion	max_depth	min_samples_leaf	min_samples_split	F1 score
entropy	500	1	2	0.382
entropy	250	1	3	0.382
entropy	100	1	5	0.375
entropy	50	1	10	0.373

Table 5.7

- **max\_features** number of features to consider when looking for the best split
- **bootstrap** whether to use out-of-bag sample or not

## Gradient Boosting

The criterion parameter was not tuned to save time and the scikit-learn documentation says: "The default value of "friedman\_mse" is generally the best as it can provide a better approximation in some cases." <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>  
**tuned parameters:** For parameters min\_samples\_leaf, min\_samples\_split, max\_depth see Decision Trees.

- **subsample** fraction of samples to be used for fitting the individual base learners
- **learning\_rate** learning rate shrinks the contribution of each tree
- **loss** loss function to be optimised

## Bernoulli Naive Bayes

**tuned parameters:**

- **alpha** Additive (Laplace) smoothing parameter. This is basically a 'fail-safe' probability in case a word is not seen before.
- **binarize** Threshold for binarising of sample features



solver	C	penalty	F1 score	solver	C	penalty	F1 score
saga	10.0	l2	0.389	liblinear	100.0	l2	0.400
saga	100.0	l2	0.401	liblinear	1000.0	l2	0.4002
saga	1000.0	l2	0.403	liblinear	1000000.0	l2	0.397
saga	1000.0	l1	0.410	liblinear	1000.0	l1	0.365
sag	10.0	l2	0.389	lbfgs	1000.0	l2	0.407
sag	100.0	l2	0.402	lbfgs	100000.0	l2	0.408
sag	1000.0	l2	0.401	lbfgs	1000000.0	l2	0.404
newton cg	100.0	l2	0.399	newton cg	1000.0	l2	0.400

Table 5.8

## Multinomial Naive Bayes

tuned parameters:

- **alpha** Additive (Laplace) smoothing parameter. This is basically a 'fail-safe' probability in case a word is not seen before.

## Logistic Regression

tuned parameters:

- **penalty** norm used in the penalization.
- **C** penalty term - see SVM parameter description
- **solver** optimisation method
- **max\_iter** maximum number of iterations

## Feature-Extraction

## Results

## 5.3 Architecture of Neural Network

### CRF

A CRF is used after the LSTM layer to determine the most likely output sequence. CRF's are commonly used for labeling sequences. This was first

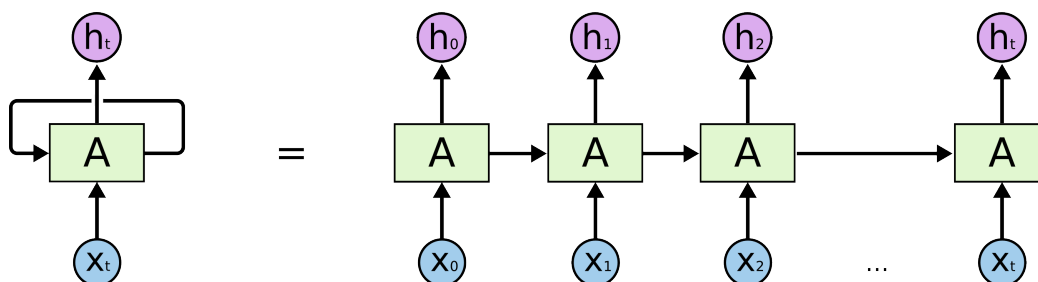


Figure 5.8: Red RNN

proposed by Lefferty

Due to the dependencies of elements of a sequence the predictor should have in mind the way in which output values depend on each other. This means that the model should not only look at the label of the current word, but also at the labels of surrounding words (sentence level tag information). One way to represent those dependencies are graphical models. A CRF is such a model. It takes the discriminative approach of modeling conditional distributions  $p(y|x)$ . [<http://homepages.inf.ed.ac.uk/csutton/publications/crftut-fnt.pdf>]

## LSTM

A LSTM network is a recurrent neural network (RNN). It is a network with loops in them, allowing them to keep information.<sup>3</sup> The figure 5.8 shows this structure. Basically they are multiple copies of a network, each of which passes a message to a successor.<sup>4</sup> The problem with simple RNNs is that they only have recent information available. Figura 5.8 muestra el diseño de un modelo RNN.

That's where LSTM networks come into play. They are a type of RNN capable of learning long-term dependencies. What's the difference between them? Basically, a RNN is much simpler.

The figures 5.9 and 5.10 show the difference.

Instead of having only one layer LSTM nets have four. The four layers consist of a memory cell, an input gate, an output gate and a forget gate. These doors regulate the flow of information and decide what information

<sup>3</sup>Cgl. Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, p.2.

<sup>4</sup>Cgl. "Understanding LSTM Networks". In: (2015).

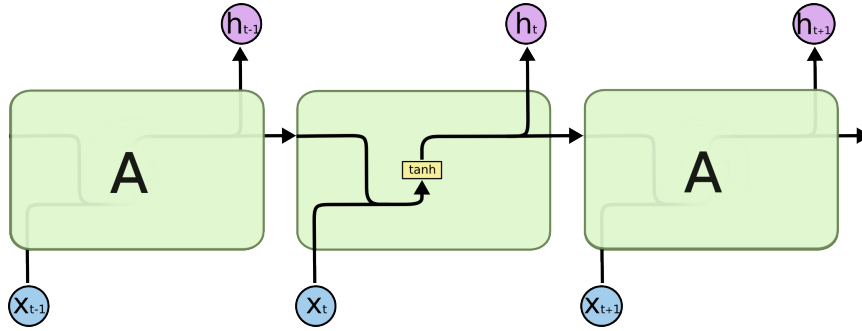


Figure 5.9: RNN

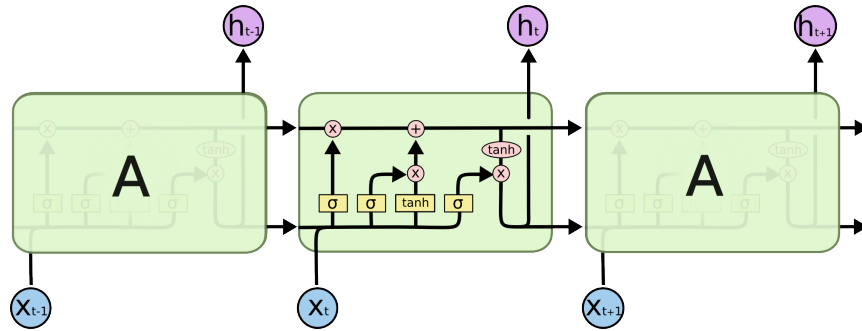


Figure 5.10: LSTM

should be kept.<sup>5</sup>

The output of a LSTM unit is defined the following way:

$$\begin{aligned}
 i_t &= \sigma_g(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
 f_t &= \sigma_g(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
 c_t &= f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
 o_t &= \sigma_g(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
 h_t &= \tanh(c_t)
 \end{aligned}$$

where  $\sigma$  is the logistic sigmoid function,  $i, f, o, c$  are the input gate, forget gate, output gate and cell vectors

<sup>5</sup>Cgl. Wei; Yu Kai Huang Zhiheng; Xu. *Bidirectional LSTM-CRF Models for Sequence Tagging*. Baidu research, 2015, p.2/3.

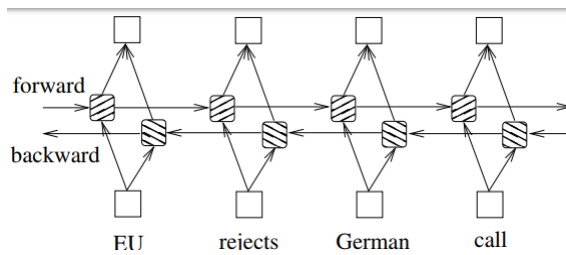


Figure 5.11: Red LSTM bidirectional

**bidirectional LSTM**

A Unidirectional LSTM only preserves information of the past because the only inputs it has seen are from the past. Using a bidirectional LSTM will run the inputs in two directions, one from past to future, and the other from future to past. This way information from the past and future can be preserved. The figure 5.11 illustrates this model in combination with a CRF network. At the centre of the phrases and labels are the memory cells that communicate in both directions.

**Dropout****Word-Embedding****Character-Embedding****Custom-Initialisation****Parameter Search****Results****5.4 Comparison**

---

## Related works

---

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.



---

## **Conclusions and future lines of work**

---

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.





---

## Bibliography

---

- [1] Jason Collobert Ronan; Weston. *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning*. NEC Labs America. URL: [https://ronan.collobert.com/pub/matos/2008\\_nlp\\_icml.pdf](https://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf).
- [2] Behrang Mohit. *Natural Language Processing of Semitic Languages*. 2014, p. 221.
- [3] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. URL: <https://www.cs.toronto.edu/~graves/preprint.pdf>.
- [4] “Understanding LSTM Networks”. In: (2015). URL: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [5] Wei; Yu Kai Huang Zhiheng; Xu. *Bidirectional LSTM-CRF Models for Sequence Tagging*. Baidu research, 2015. URL: <https://arxiv.org/pdf/1508.01991v1.pdf>.