UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

NoisyNER - Named entity recognition in social media

Presentado por Malte Janssen
en Universidad de Burgos — July 3, 2019
Tutor: Bruno Baruque Zanón

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. D. Bruno Baruque Zanón, profesor del departamento de Ingenería Civil, Lenguajes y Sistemas Informaticos. 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, July 3, 2019

V^{o}. B^{o}. del Tutor:

D. Bruno Baruque Zanón

Resumen

Hoy en día, alrededor de 3.5 millardos de personas utilizan las redes sociales de forma regular. 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 de aprendizaje profundo. Ambos enfoques se comparan para seleccionar el mejor enfoque y la mejor configuración para el problema. Este trabajo actúa más como un estudio científico que como el desarrollo de una aplicación, aunque se desarrolla un simple web-app para mostrar el modelo final.

Descriptores

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

Abstract

Today about 3,5 billion people use social media on a regular basis. 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 which 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 deep learning model is proposed. Both approaches are then compared in order to select the best approach and configuration for the problem. This work acts more like a scientific study than a development of an application, although a simple web-app is developed to showcase the final model.

Keywords

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

Contents

List of Figures

List of Tables

Abbreviations

NLP Natural Language Processing

NER Named Entity Recognition

CoNLL Conference on Natural Language Learning

SVM Support Vector Machine

CRF Conditional Random Field

LSTM long short term memory

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 see they are able to produce acceptable results. Secondly a deep learning model 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. Furthermore the encountered challenges in the course of the project are documented.

Objectives

This project aims to study the usefulness of several machine learning models on noisy/informal texts. Also a deep learning model is proposed.
During the course of the project the following steps were taken:

- Preparation of data - The data has to be read and turned into a format which the models can handle
- Training - training of the NLTK and scikit-learn classifiers
- Design a deep learning model - propose a more complicated deep learning model to tackle the task
- Parameter study - improve the performance of the models by finding optimal hyperparameters and appropriate features
- 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 its sub-field of artificial intelligence.

Some of today's use cases are information extraction, machine translation, summarisation, searches and human-computer interfaces.[1]

It describes techniques in which natural language is processed into a formal representation, which computers can understand so they can further process the data. In order to do that 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 and common sense to understand texts but rather have to use artificial language algorithms and techniques. One of those techniques is called Named Entity Recognition(NER).

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 which 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.[2]

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

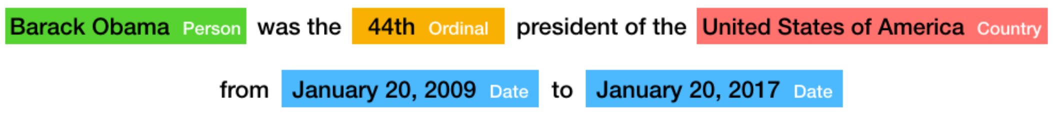

Figure 3.1: Example of entities in a phrase Source: https://github.com/floydhub/named-entity-recognition-template

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 used to have a 140 character limit, so not much context can be extracted in comparison with a newspaper article or a book. Even though now Twitter doubled that limit, most tweets are still under the 140 character mark. For these reasons named entity recognition is a much harder task on informal texts.

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 functions to process natural language (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 task, Flask was chosen to keep the web application as simple as possible.

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.

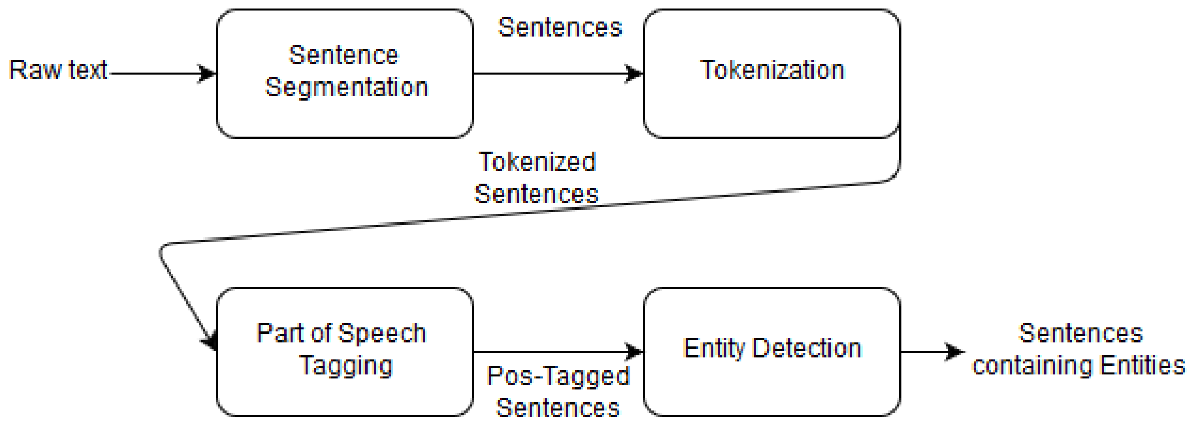

Figure 4.2: NER - Process

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.]

Word - Tokenisation

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

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

Part of Speech Tagging (POS Tagging)

In POS Tagging each word of a text is assigned a part 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[3] 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> 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.

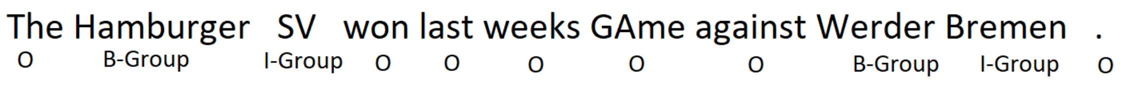
The Hamburger SV won last weeks GAme against Werder Bremen .
O B-Group I-Group O O O O O B-Group I-Group O

Figure 4.3: Example of Chunking

Applied Models

The following section includes some basic descriptions about the theoretical concepts of the applied models without getting into implementation details.

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 sub-sequences of n items. The n amount of previous words and POS-tags are used to guess the tag for the current word. A Unigram Tagger for example doesn't use any context and only uses the POS-tag of the current word to assign a tag. To give an example a Unigram tagger assigning POS-tags would for example not be able to differentiate between "the play" or "to play", which are different types of word and have completely different meaning. A Bigram tagger could detect that "the" and "to" have different meaning though and could therefore potentially detect the difference word type of play.

Feature Chunking

Often it is not enough to just look at the POS-tag of a words, more information needs to be included. This information is determined by a feature extraction function which extracts a so called feature-set for each word. Examples of features contained in a feature-set could be the word itself, the surrounding words, the previously discussed POS-tag or even the shape of a word.

Figure 4.4 shows the classification process using such a feature extraction function. During training each word is transformed into a feature-set, which contains the information on which the word 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 extraction function is used to convert the to be predicted input into features, which are then used by the model to predict the corresponding label.

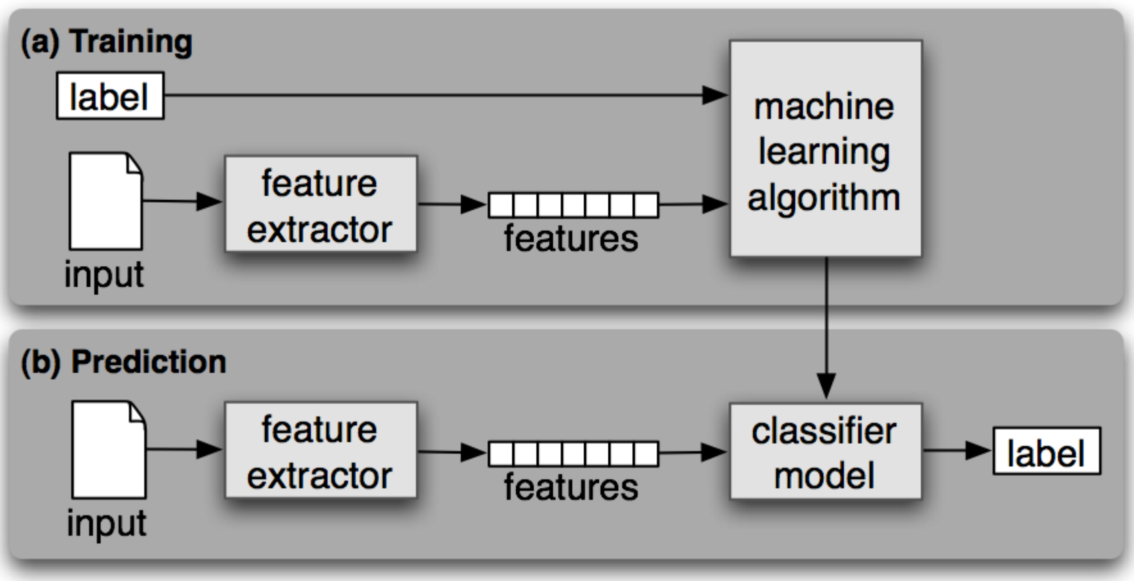

Figure 4.4: Classification Process

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, which could in the simplest case be a true or false decision. The process starts at the root leaf and continues through decision nodes until a leaf node is reached, which assigns a label.[4]

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. [4] The goal is to create a decision tree that contains the decision nodes which represent the most informative features early on.

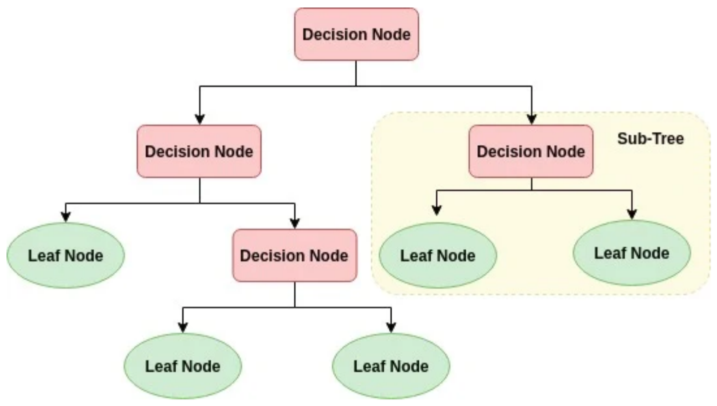

Figure 4.5: Representation of a Decision Tree

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

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).[4]

Multiple different naive Bayes Classifiers, making different assumptions regarding the distribution of the features, have been used.

Multinomial Naive Bayes Multinominal Naive Bayes classifiers assume a multinominal distribution of the features, modelling feature occurrence counts with that distribution.

Bernoulli Naive Bayes The Bernoulli Naive Bayes classifier takes the binary presence or lack of a feature value into account, rather than the number of times it appears using the Bernoulli distribution.

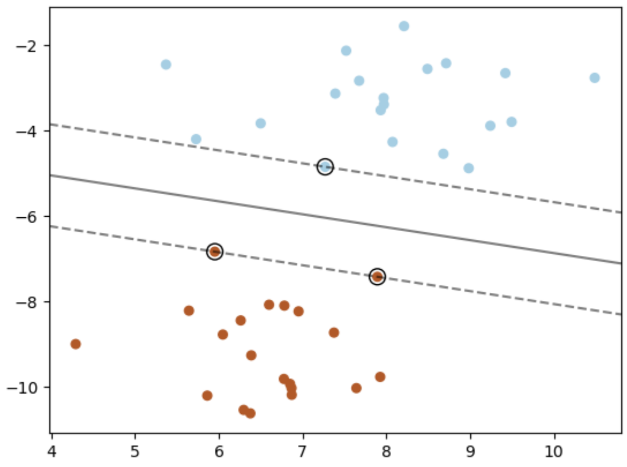

Figure 4.6: Seperation by a hyperplane - Source: https://scikit-learn.org/stable/modules/svm.html

Maximum entropy 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. Rather than looking at each feature indipendently, the model uses a search based optimisation to find weights for the features that maximise the likelihood of the training data. They are discriminative models instead of generative models.[4]

Support Vector Machines

Support Vector machine classifiers crate a hyperplane dividing two classes of data. The hyperplane created has the highest distance possible to the nearest training example of both classes, which enables a good separation. Figure 4.6 visualises such a division.

Looking at figure 4.6 it is obvious that only two classes can be differentiated by a single support vector machine. Two different concepts are applied to make Support Vectors able to deal with multiple classes: One vs. the rest and One vs. one. In one vs. the rest classification a SVM is created for each class classifying training examples into that class or or the other representing all other classes. The other approach creates a SVM for each pair of classes, to separate members of one class from members of the other and applying a voting procedure. The sci-kit learn SVM classifier uses the latter.

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 single strong one. To do that base estimators are being build sequentially and then added to a combined estimator.

Random Forest Classifier

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

Each tree is usually constructed with a bootstrap sample (replacement sample), meaning they are not trained on the whole training data. The decision trees are often 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. [5]

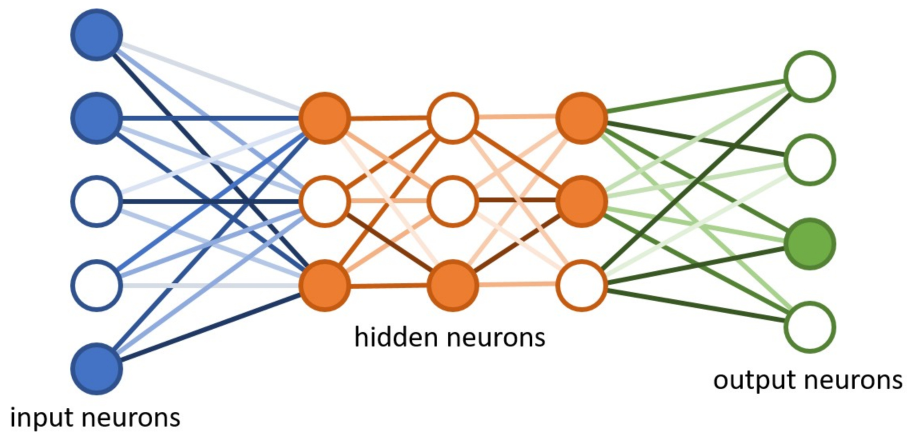

Figure 4.7: Design of a Neural Network Source: https://medium.com/xanaduai/making-a-neural-network-quantum-34069e284bcf

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, 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 dependent 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 fails to improve for a number of iterations). Usual applications are pattern recognition or data classification.

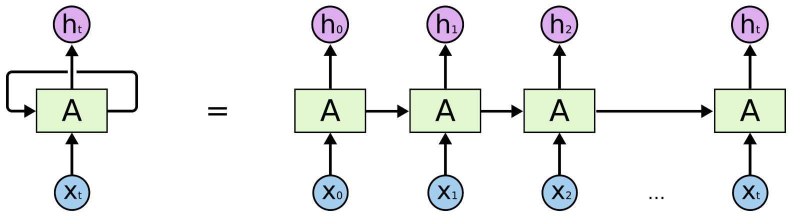

Figure 4.8: RNN network Source:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

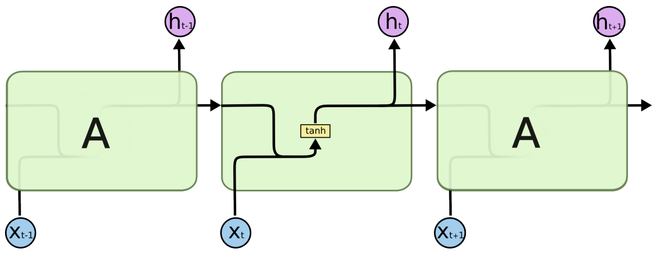

Figure 4.9: RNN network structure source:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

A long-short-term memory network is a recurrent neural network (RNN). It is a network with loops in it, allowing it to keep information. [6] The figure 4.8 shows this structure. Basically they are multiple copies of a network, each of which passes a message to a successor. [7] The problem that simple RNNs face is that they only have recent information available.

That's where LSTM networks come into play. They are a more complex type of RNN capable of learning long-term dependencies.

The figures 4.9 and 4.10 show the difference. Instead of having only one layer like the RNN, LSTM nets have four. The four layers consist of a memory cell, an input gate, an output gate and an forget gate. These doors regulate the flow of information and decide what information should be kept. [8]

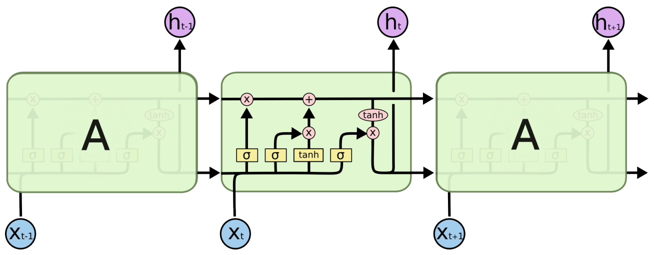

Figure 4.10: LSTM network structure source:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

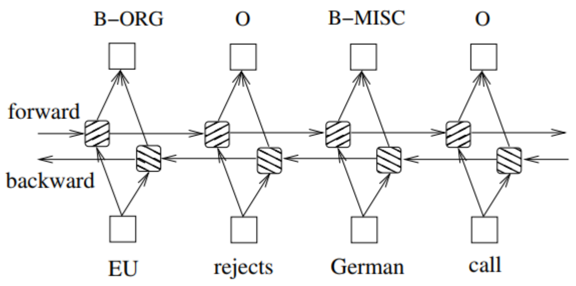

Figure 4.11: bidirection LSTM network source:
https://arxiv.org/pdf/1508.01991v1.pdf

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 4.11 illustrates this model. At the centre of the phrases and labels are the memory cells that communicate in both directions and are making past and future information available that way.

CRF
Conditional random fields (CRF) first proposed by Lefferty et al..[9] They have the ability to consider surrounding output information (sentence level tag information). It takes the discriminative approach of modeling conditional distributions [10]. How this looks like in combination with a LSTM model can be seen in figure 4.12. In comparison with figure 4.11 now not only information about surrounding features is available (represented by the memory cells in the middle), but also the sentence level tag information (other outputs of that sentence). This is represented by the connection

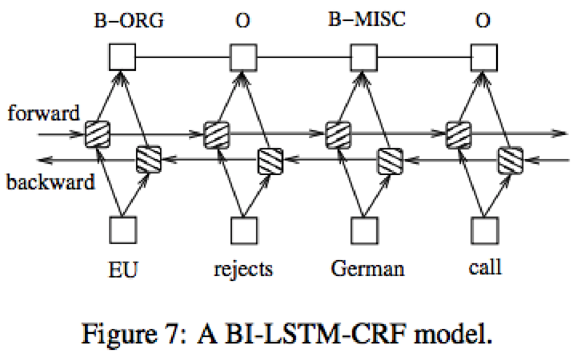

Figure 4.12: BI-LSTM CRF network source:
https://arxiv.org/pdf/1508.01991v1.pdf

between outputs in the diagram.

Dropout
Dropout is a technique used during training, which randomly drops and omits units form the neural network. That means that in each iteration a different thinned out network is trained. That way many different networks can be trained and averaged without adding computational complexity. [11] Figure 4.13 visualises how dropout changes a fully connected network into a only partially connected network.

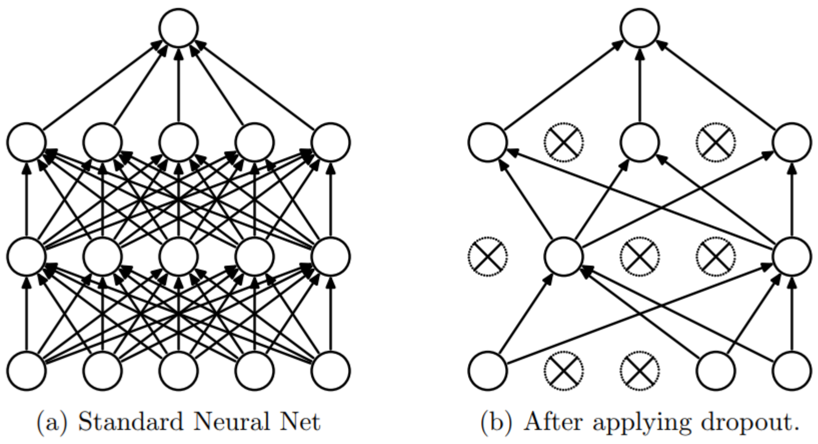

Figure 4.13: Dropout source:
http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Word Embeddings Word embeddings were introduced by Mikolov et al. 2013. [12] Each word in the models vocabulary (words that are in the training dataset) is represented by a word embedding. The word embedding represents the semantic meaning of that word. It is learned based on the usage of the word. Words that are used in similar contexts will have similar word embeddings.

Sequences are fed into the neural network as lists of integers. Each integer represents a word and can be looked up in a lookup table. The neural network used has an Embedding layer, which stores the embeddings in a embedding Matrix with vocabulary size entries. The embedding layer is used on the front end of the network and is fit in a supervised way using backpropagation. The word embeddings are learned jointly with the neural network. The previously described sequence lists act as input to the embedding layer, which updates the word embeddings of the used words based on the contexts in which they have been used in the sentence. The output of the embedding layer is then propagated through the network, updating it's weights.

Character Embeddings Character embeddings work in a similar fashion as word embeddings. Instead of each word having its own vector representation, each character is represented by its own vector representation though. They were first introduced by Zhang et al..[13]

Relevant aspects of the development of the project

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

5.1 Dataset

The WNUT-17 dataset was used to train and evaluate the 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 it 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 (dataset was created befor lifting that limit) characters as these exhibit different writing styles and characteristics.[14]
The data is divided into the following 7 classes:

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

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

- Group
- O (No entity)

Table 5.2 shows how many entities are contained in each data split. 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 which uses 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). A detailed study containing results can be find in Chapter D of the Annex.

Feature-Extraction

Different feature sets were tested to find the most informative features. The tested features were:

- **word**(w) word
- **POS-tag**(p) POS-tag of the token
- **class**(c) entity(class) of the token
- **shape**(s) shape of the word (eg.: Upper-casE -> Xxxxx-xxxX)
- **is digit**(d) whether the is a digit
- **is uppercase**(u) whether the word is uppercased
- **is title**(t) whether the word starts with a uppercased letter and all other letters are lowercase
- **length**(l) length of the word

The results of that study can be find in chapter D of the annex.

5.3 Architecture of the Deep Learning Model

The final model of the deep learning classifier is a BiLSTM CRF word and character embeddings. The dropout technique also has been looked at. The following sections explains the why each of these layers was used.

LSTM

As explained in the last chapter bidirectional LSTM networks can save past and feature information. Having the sequential nature of texts in mind, this is especially useful since this way the context of the text can possibly be captured.

CRF

A conditional random field (CRF) is used after the LSTM layer to determine the most likely output sequence. e Due to the dependencies of elements of a sequence output values depend on each other in a certain way. The CRF is applied to do exactly that. The model now does not only look at the label of the current word, but also at the labels of surrounding words (sentence level tag information).

Dropout

Due to the predictive power of neural networks, they are prone to overfitting. The Dropout technique is used to prevent early overfitting. To check if the network is being fit too tightly to the training data the dropout technique is applied.

Word-Embedding

Since neural networks are designed to learn from numerical data, working with words(strings) is not really an option. Instead words are transformed into vectors, also called word embeddings as discussed in the chapter. These embeddings learn the meaning of a word and are therefore useful in NLP tasks.

Prelearned Embeddings

The proposed network has the ability to load pre-trained word embeddings. This way the embeddings for words inside the vocabulary of the pre-trained word embeddings don't have to be randomly initialised, leading to the model already having knowledge about the significance of some words from the start. Since words can have very different meanings in different contexts these pre-trained embeddings should optimally be trained in a similar context as the task. That is why the pre-trained word embeddings used in the model are the pre-trained GloVe[15] embeddings trained on Twitter Data.

Character-Embedding

Words that are not in the vocabulary, so called out of vocabulary words, are not represented by word embeddings, meaning their significance has to be inferred only by it's surround words. This happens frequently in noisy data, because it contains a lot of misspellings and unusual words. To combat this, additionally to using word embeddings, character embeddings are used.

Character embedding enable to better represent the significance of these misspelled or unusual words. To encode the character-level information, character embeddings and a additional LSTM are used to encode every word to a vector. The resulting vectors are combined with the word embeddings and fed into the main LSTM layer.

Final Architecture

Different combinations of the previously explained techniques have been tested. The results can be found in chapter D of the annex. Combining the best performing approaches results the in figure 5.14 highlighted model. It shows the creation of word and character embeddings, which are then combined into a word representation. This word representation is fed into the models main LSTM layer. The LSTM layer leverages past and future features and acts as input for the CRF layer which also takes sequence level entity output into account.

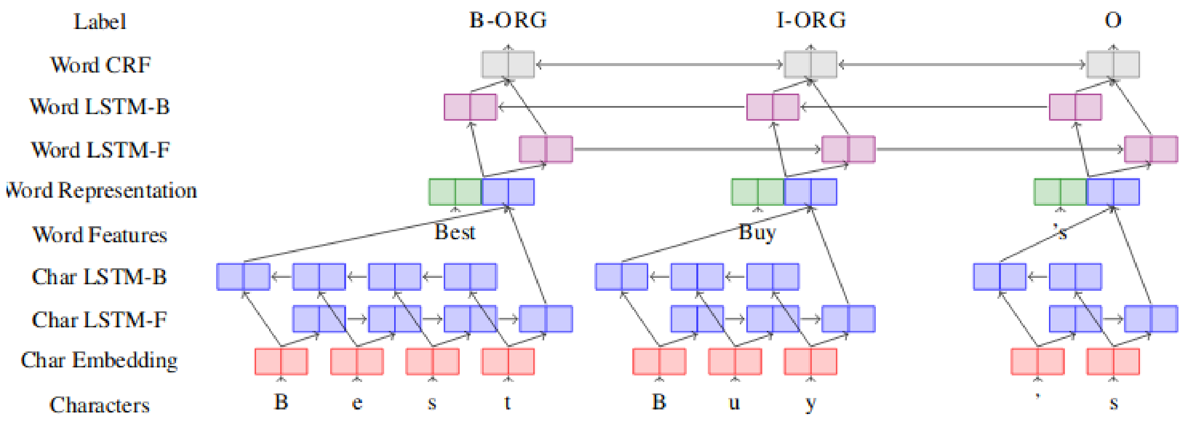

Figure 5.14: Overview of the models architecture source: https://www.aclweb.org/anthology/C18-1182

Related works

Named entities recognition has become a growing field in NLP. In NLP, NER and part-of-speech tagging (POS) are considered examples of the sequence labelling problem. One of the common evaluations for this task is the CoNLL-2003 shared task , which introduced English and German datasets derived from expert-labeled news articles. [16]

Most modern approaches to NER are based on deep learning. While using neural networks for NER is not new (one of the CoNLL 2003 submissions used a LSTM network[17]), the surge of more memory and data has made these approaches the main research direction. Using deep neural networks has led to a reduction of hand tuned features. Instead word embeddings have become the prevalent feature, which was first introduced by Mikolov et al.[12]. Also character embeddings were introduced as an additional feature by Zhang et al..[13] Furthermore Conditional Random field have been thrown into the mix by Lefferty et al. to profit from surrounding outputs.[9]
Finally a similar model to the one proposed was introduced by Huang et al. [8] which also used a bi-directional LSTM CRF model.

Conclusions and future lines of work

While working with the classifiers of the first part of the work it became obvious that the models parameters as well as the hand crafted feature set are of crucial importance. The performance was significantly increased by doing an extensive search of optimal parameters and optimal feature extraction function. But even after tuning parameters it became clear that these models are not adequate for the task at hand. When comparing the results of the more "traditional" classifiers to the deep learning classifier it became obvious that deep learning classifiers have a distinct advantage over most of these "traditional" classifiers, although it has to be noted that just throwing a "random", not appropriate model (in terms of architecture) at the problem will result in inferior results. But when the right type of architecture, with the problem and nature of data in mind, is chosen state of the art deep learning classifiers are the more potent alternative.

The task at hand was a very challenging one due to it's complexity. Not only is named entity recognition a more complex task than other fields of NLP like sentiment analysis (because less information is available about a token than about a whole text), but also the type of data increased complexity. For usual NER problems on for example news corpora state of the art named NER models tend to reach F1 scores in the ninetieth. To date this is not possible for noisy data though. By contrast the best performing model reached a F1 score of only 41.86.

By combining different techniques as described in the document, initial performance of the system was increased, but the model failed to reach the state of art performance for the dataset. A number of different approaches to improve performance will be looked at in the future, including:

- A more extensive parameter search for the deep learning classifier. Due to the very time consuming training process, limited resources and time the parameter search was not the most extensive. A further search could lead to significant improvements.
- Using different pre-trained word embeddings. While the used pre-trained gloVe embeddings improved the performance of the network significantly, the word embeddings are context independent. Other types of word embeddings like ELMo and BERT can generate different word embeddings for a word that captures the context of a word. That way different word embedding are created, depending on the position of the word. Sadly no such pre-trained embedding exist for social media data. Due to them being more powerful it is still worth a shot experimenting with them though.
- Yadav et al. proposed a deep laerning model combining word representations, character embedding and affix features capturing prefixes and suffixes of the word better than just using character embedding. This could work great in the context of noisy data. [https://www.aclweb.org/anthology/S18-2021]
- Another possible way to improve performance would be data preprocessing. A few common methods such as zeroing all numbers and lowercasing all texts were tested but resulted in poorer performance as information got lost. Other possible normalisation approaches could include stemwords and stopwords.

On a personal note, the student acquired many new skills. Nearly all of the topics and techniques used were new before and unheard of before the semester started and had to be learned from zero. The only topic were a little prior knowledge was present was neural networks, but the knowledge was very basic.

Bibliography

[1] Ronan Collobert and Jason Westan. A unified architecture for natural language processing: Deep neural networks with multitask learning. Technical report, NEC Labs America. URL: https://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf.

[2] Behrang Mohit. Natural Language Processing of Semitic Languages. 2014.

[3] Alphabetical list of part-of-speech tags used in the penn treebank project:. Accessed: 2019-04-21. URL: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

[4] Steve Bird, Edward Loper, and Ewan Klein. NLTK Book. 2009. URL: https://www.nltk.org/book.

[5] sckit-learn users guide - ensemble methods. URL: https://scikit-learn.org/stable/modules/ensemble.html.

[6] Alex Graves. Supervised Sequence Labelling with Recurrent Neural Networks. URL: https://www.cs.toronto.edu/~graves/preprint.pdf.

[7] Understanding lstm networks. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[8] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. Technical report, Baidu research, 2015. URL: https://arxiv.org/pdf/1508.01991v1.pdf.

[9] John Lafferty, Andre McCallum, and Fernando C.N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. Technical report, University of Pennsylvania, 2001. URL: https://repository.upenn.edu/cgi/viewcontent.cgi?referer=http://www.albertauyeung.com/post/python-sequence-labelling-with-crf/&httpsredir=1&article=1162&context=cis_papers.

[10] Charles Sutton and Andrew McCallum. An Introduction to Conditional Random Fields. 2012. URL: https://homepages.inf.ed.ac.uk/csutton/publications/crftut-fnt.pdf.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, and Ruslan Sutskever, Ilya Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Technical report, Journal of Machine Learning Research, 2014. URL: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

[12] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. Technical report, Google, 2013. URL: https://arxiv.org/pdf/1301.3781.pdf.

[13] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. Technical report, Courant Institute of Mathematical Sciences, 2016. URL: https://arxiv.org/pdf/1509.01626.pdf.

[14] Leon Derczynski, Eric Nichols, Marieke van Erp, and Nut Limsopatham. Results of the wnut2017 shared task on novel and emerging entity recognition. Technical report, 2017. URL: https://aclweb.org/anthology/W17-4418.

[15] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. Technical report, Stanford University. URL: https://nlp.stanford.edu/pubs/glove.pdf.

[16] Erik F. Tjong, Kim Sang, and Fien De Meuld. Introduction to the conll-2003 shared task:language-independent named entity recognition. Technical report, University of Antwerp, 2003. URL: https://www.aclweb.org/anthology/W03-0419.

[17] James Hammerton. Named entity recognition with long short-term memory. Technical report, 2013. URL: http://delivery.

acm.org/10.1145/1120000/1119202/p172-hammerton.pdf?ip=80.103.249.134&id=1119202&acc=OPEN&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E6D218144511F3437&__acm__=1561735318_e64d9d9c07284edaf994c3f15a451bbf.