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 29, 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, June 29, 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

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

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

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 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 deep learning model is proposed.
The following steps were 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 deep learning model to tackle the task. To do this a LSTM 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 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.[1]

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).

[1] 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.[2]

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

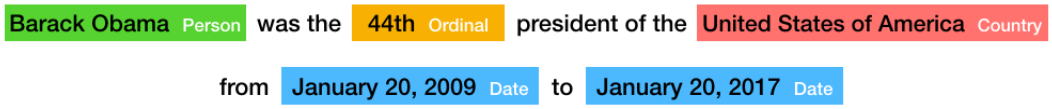


Figure 3.1: Example of entities in a phrase Source: https://github.com/floydhub/named-entity-recognition-template

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 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.

[2]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)[3] 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[4] 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.

[3] *NLTK.*
[4] *Scikit-learn.*

Pytorch

PyTorch[5] 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[6]. 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[7] 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 brackeds are not to be seen as lists but rather as indicators of seperate 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.]

[5] *Pytorch.*
[6] *Flask.*
[7] *Heroku.*

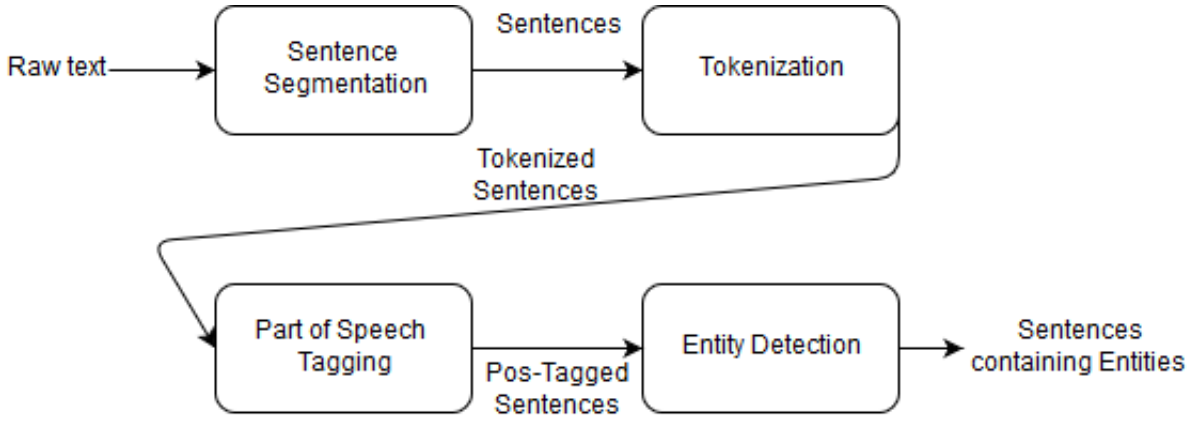


Figure 4.2: Process

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[8] 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

[8] *Alphabetical list of part-of-speech tags used in the Penn Treebank Project:*

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

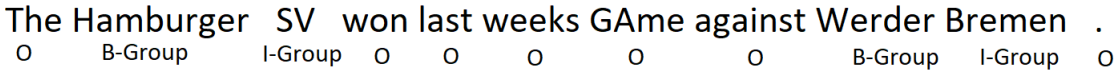


Figure 4.3: Example of Chunking

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.

Applied Models

The follwoing 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 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 th 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

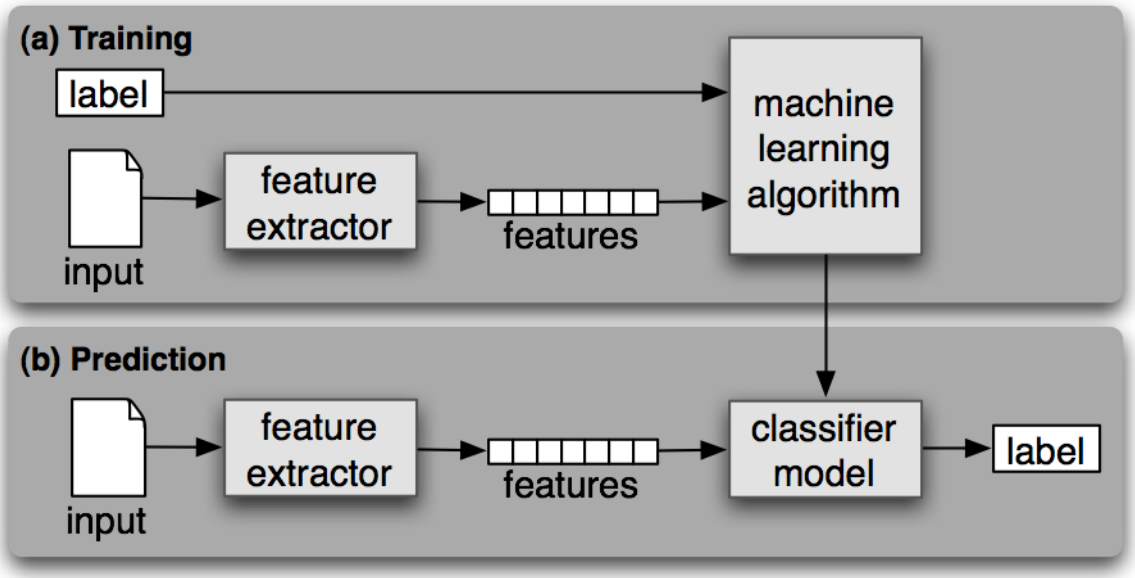


Figure 4.4: Classification Process

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

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.

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.[9]

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.[10]

[9] Cgl. Edward; Klein Ewan Bird Steven; Loper. *NLTK Book*. 2009. Chap. 6.
[10] Cgl. Bird, *NLTK Book*.

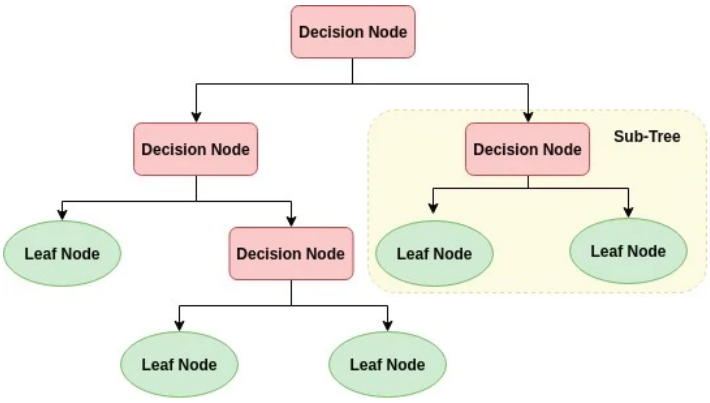


Figure 4.5: Representation of a Decision Tree

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

$$entropy = -\sum p(label)logp(label) \tag{4.1}$$

The entropy formula (4.1) shows that either a high amount of inputs from the same label (low $logP(l)$) 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.[11]

$$InformationGain = Originalentropy - newentropy \tag{4.2}$$

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.[12]

Discriminative vs Generative Models

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

[11] Cgl. Bird, *NLTK Book*.
[12] Cgl. Bird, *NLTK Book*.

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).[13]

The Bayes Theorem states the following:

$$posterior = \frac{priori * likelihood}{evidence} \tag{4.3}$$

Knowing that the following equation can be made.

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

Features is a dependent feature vector of size n:

$$features = (f1, f2, f3, ..., fn) \tag{4.5}$$

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: $P(f1|Class) * ... * P(fn|Class)$

This gets us the following:

$$P(Class|f1, ..., fn) = \frac{P(f1|Class)...P(fn|Class)P(Class)}{P(f1)...P(fn)} \tag{4.6}$$

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

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

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

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

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

[13] Cgl. Bird, *NLTK Book*.

Gaussian Naive Bayes Gaussian Naive Bayes assumes that the likely hood of features is Gaussian.

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

σ_y and μ_y are estimated using the maximum likelyhood method, which is an estimation method in which the values are estimated by taking a sample.[14]

Multinomial Naive Bayes Multinominal Naive Bayes asuumes a multinominal distrubution of the data. The vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ parameterize the distrbution for each class y where n is the number of features and θ_{yi} is the probability $P(x_i \mid y)$ of a feature i to appear in class y.[15]

θ_y is estimated using relative frequency counting:

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

$N_{yi} = \sum_{x \in T} x_i$ 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 α makes sure there are no zero probabilities (eg. Leplace Smoothing).[16]

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.[17]

While building the model it's 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 likely-hood defined by:

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

[14] Cgl. *scikit-learn Users Guide - Naive Bayes.*
[15] Cgl. *scikit-learn Users Guide - Naive Bayes.*
[16] Cgl. *scikit-learn Users Guide - Naive Bayes.*
[17] Cgl. Bird, *NLTK Book.*

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

$$P(label|features) = P(label, features)/\sum_{l} abelP(label, features) \quad (4.12)$$

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.[18]

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 seperation. 4.6 visualises such a devision.

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 it's 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 it's 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

[18] Cgl. Bird, *NLTK Book*.

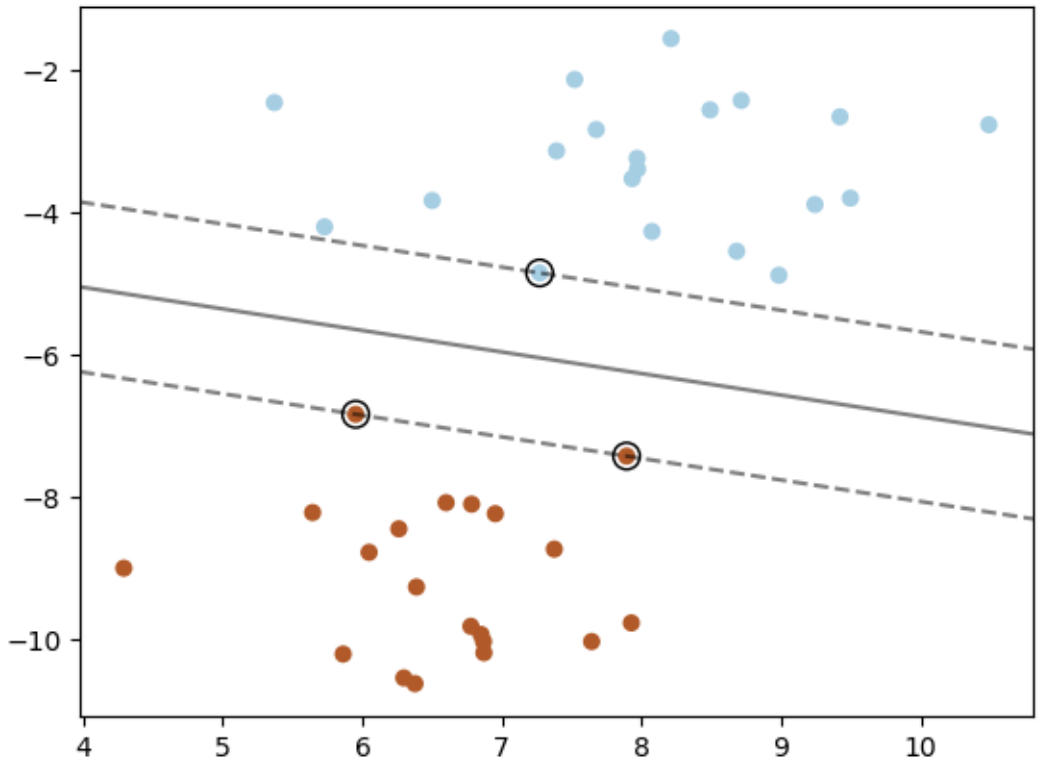


Figure 4.6: Seperation by a hyperplane - Source: https://scikit-learn.org/stable/modules/svm.html

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 usually constructed with a bootstrap sample (replacement sample). 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.[19]

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 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.

[19] Cgl. *sckit-learn Users Guide - Ensemble methods.*

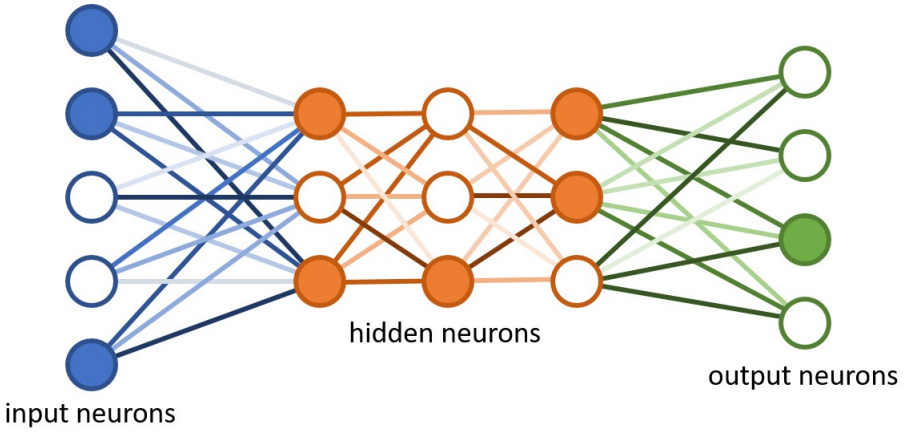


Figure 4.7: Design of a Neural Network Source: https://medium.com/xanaduai/making-a-neural-network-quantum-34069e284bcf

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 characters as these exhibit different writing styles and characteristics.[20]
The data is divided into the following 7 classes:

- Person
- Location
- Corporation (includes Geopolitical Entities)

[20]Cgl. Eric; van Erp Marieke; Limsopatham Nut Derczynski Leon; Nichols. *Results of the WNUT2017 Shared Task on Novel and Emerging Entity Recognition.* 2017.

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

- Product (tangible goods, or well-defined services)
- 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

classifier	F1 score
MultinomialNB	0.0
BernoulliNB	0.0
SVC	0.0
LogisticRegression	0.358
DecisionTree	0.352
GradientBoosting	0.292

Table 5.3

description of each hyperparameter.
The results are merely summaries 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`
The portrayed f1 scores are computed by the default sckit-learn f1 score and do not keep the sequential nature of the data in mind. This is done to find the parameters which encourage the classifier to find as many as possible of the entities, even if only partially.

Baseline Results

Before doing a extensive gridsearch the F1 score for baseline models was calculated. Table 5.3 shows the results. Three of the tested classifiers failed to capture any entity. They classified every single token as "O" (other). The best performing baseline configuration is the Logistic Regression classifier with a token-level f1 score of 0.358 on the validation data.

SVM

Tuned parameters:

- **Kernel** kernel type
- **C** Penalty parameter of the error term - how much classification errors are 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

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

Table 5.4

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 5.5

- **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 5.4 shows that rbf and linear kernels yield the best results. The best result is yielded by a rbf support vector machine with a C value of 10 and a gamma of 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 5.5. A lower degree resulted in better results, but didn't increase the previously found best score of 0.399.

By doing a parameter search it was possible to make the SVM, that failed to capture any entities with the baseline configuration, work for the given problem. The performance was increased by nearly 40%. **Best set of Parameters:**
C: 10
Gamma: 0.1
Kernel: rbf

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 (5.6) 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 in a node with just one example was ideal (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 5.7. 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 5.8. Using the minimum amount of 2 yielded the best results (0.382 F1 score). The F1 score was increased from 0.352 with default parameters to 0.382 with the best set of found parameters.

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 5.6

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 5.7

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 5.8

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."[21] **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

[21] *sklearn Documentation - sklearn.ensemble.GradientBoostingClassifier.*

max_depth	min_samples_leaf	min_samples_split	F1-score
20	5	8	0.383
20	5	2	0.383
20	5	4	0.383
10	5	8	0.375
10	5	2	0.375
10	5	4	0.375
8	5	8	0.371
8	5	2	0.371
8	5	4	0.371
20	3	8	0.361
10	3	8	0.368
8	3	8	0.354

Table 5.9

- **learning_rate** learning rate shrinks the contribution of each tree
- **loss** loss function to be optimised

First a smaller number of estimators (80) was used to tune the tree parameters. This allows for more tests, since training takes less time. In table 5.9 it can be observed that increasing the maximum depth of trees from the default of one had a major effect on performance. Furthermore it can be observed that higher numbers of minimum leaf samples (min_samples_leaf) did better than lower numbers. The parameter min_samples_split barely had an effect though.

After the first general tests the tree parameters were further tuned as can bee seen in table 5.10. Even further increasing the min_samples_leaf parameter to 30 increased performance a little.

Next the learning rate was decreased and the amount of estimators increased. By decreasing the learing rate to 0.03 and at the same time increasing the amount of estimators the performance was slightly increased as can be seen in table 5.11. Increasing the amount of estimators makes the model more powerful and vulnerable to overfitting. To counteract this the importance of each estimator has to be decreased by decreasing the learning rate.

max_depth	min_samples_leaf	min_samples_split	F1-score
20	5	8	0.387
20	10	8	0.362
20	20	8	0.388
0	30	8	0.391
20	50	8	0.386
20	100	8	0.389

Table 5.10

learning_rate	n_estimators	max_feats	F1-score
0.1	80	None	0.391
0.08	100	None	0.383
0.08	500	None	0.384
0.08	250	None	0.393
0.05	500	None	0.390
0.05	250	None	0.391
0.03	250	None	0.391
0.03	500	None	0.394
0.03	500	log2	0.291

Table 5.11

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.

Before doing a parameter search both Naive Bayes classifiers failed to predict any entities and only predicted the majority class O. Decreasing the alpha value increased results in both cases. Tables 5.12 and 5.13 suggested that doing close to no smoothing at all resulted in the best performance.

Logistic Regression

tuned parameters:

- **penalty** norm used in the penalization.

alpha	F1-score
1	0.0
0.8	0.0
0.6	0.0
0.4	0.0
0.2	0.0
0.1	0.086
0.000001	0.297

Table 5.12: Bernoulli Naive Bayes

alpha	F1-score
1	0.0
0.8	0.0
0.6	0.001
0.4	0.024
0.2	0.099
0.1	0.189
0.000001	0.285

Table 5.13: Multinominal Naive Bayes

- **C** penalty term - see SVM parameter description
- **solver** optimisation method
- **max_iter** maximum number of iterations

The results, shown in table 5.14 show that most solvers performed similary. Whenever the C value was smaller than 1 the classifier failed to produce decent results (see complete results). The best results were produced by a classifier with the saga solver, a C value of 1000, and the l1 penalty function. With these parameters the baseline performance was improved from by almost 5% from 0.358 to 0.410 F1 score.

Feature-Extraction

Different feature sets were tested to find the most informative features. These tests were only done on a relatively fast classifier, because testing on all of them would not have been time feasible. The classifier used was

solver	C	penalty	F1 score
saga	10.0	l2	0.389
saga	100.0	l2	0.401
saga	1000.0	l2	0.403
saga	1000.0	l1	0.410
sag	10.0	l2	0.389
sag	100.0	l2	0.402
sag	1000.0	l2	0.401
newton cg	100.0	l2	0.399

solver	C	penalty	F1 score
liblinear	100.0	l2	0.400
liblinear	1000.0	l2	0.4002
liblinear	1000000.0	l2	0.397
liblinear	1000.0	l1	0.365
lbfgs	1000.0	l2	0.407
lbfgs	100000.0	l2	0.408
lbfgs	1000000.0	l2	0.404
newton cg	1000.0	l2	0.400

Table 5.14

the Naive Bayes NLTK classifier. The F1 scores portrayed are the f1 scores calculated by NLTK not the f1 scores from the official wnut eval script, which will be used in the results section.

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

A -/+1 describes the previous/next token.

The results of the tests are visible in table 5.15. They showed that the word itself is important, but that the surrounding words do not increase performance. In contrast the POS-tags of surrounding words did contribute to better results. The reasons for this are kind of obvious because languages are structured, so the type of word of surrounding words contains valuable information. Despite of the nature of data, which has in contrast to formal

Feature-set	F1 Score
w, w-1, w+1	2.4
w, w-1, w+1, p, p-1, p+1	7.9
w, w-1, w+1, p, p-1, p+1, c-1	9.7
w, w-1, w+1, p, p-1, p+1, c-1, s	11.5
w, w-1, w+1, p, p-1, p+1, c-1, s, s+1	9.4
w, w-1, w+1, p, p-1, p+1, c-1, s, s-1, s+1	8.2
w, w-1, w+1, p, p-1, p+1, c-1, s, s-1	9.1
w, w-1, p, p-1, p+1, c-1, s	13.1
w, p, p-1, p+1, c-1, s	15.2
p, p-1, p+1, c-1, s	10.0
w, p, p-1, c-1, s	10.3
w, p, p-1, p+1, c-1, s, d	15.4
w, p, p-1, p+1, c-1, s, d, u	15.5
w, p, p-1, p+1, c-1, s, d, u, t	17.8
w, p, p-1, p+1, c-1, s, d, u, t, l	17.2
w, p, p-1, p+1, c-1, d, u, t, l	17.7

Table 5.15: Results of feature-set tets

text less structure, some structure does remain. The entity of the prior word also had an influence. This is due to that an entity I-Person is more likely to follow B-Person than for example I-corporation. Furthermore the shape of a word had an influence on performance. This can likely be contributed to very short words not likely being a entity. The shape also captures whether a word contains special characters and information about lower/upper casing. Despite that information already partially represented by the shape the feature t (is title) had a big influence, increasing the F1 score by over 2%. Factors slightly increasing the score were whether a word is a digit and uppercased which can be explained by knowing that a word is a digit, makes it very unlikely being an entity. The only proposed feature not included in the final feature selection function was the length of a word, which does not mean that this feature does not contain any valuable information. On the contrary it added a few percent (visible in the last row), but only if the shape feature was not included. This can be explained by the shape feature already covering the length of a word. Including the shape feature instead resulted in a slight increase though (0.1%) because it also contained additional information as explained earlier. The shapes of the surrounding word did not increase the results.

Classifier	F1-score
Unigram	0.051
Bigram	0.005
Trigram	0.002
Bigram - backoff	0.0530
Trigram - backoff	0.0531

Table 5.16

n-gram Taggers - Backoff

The n-gram backoff functionality was tested. The resulting F1 scores have the sequential nature of sentences in mind. Here no parameter had to be tuned. Table 5.16 suggests that unigram taggers are far superior to bi- or tri-gram taggers, but that they were able to classify some entities that unigram taggers couldn't, meaning the backoff functionality slightly increased the F1 score.

5.3 Architecture of Neural Network

The final model of the neural network classifier is a BiLSTM CRF featuring dropout and character embedding. The following sections explain it's individual layers.

LSTM

A LSTM network is a recurrent neural network (RNN). It is a network with loops in them, allowing them to keep information.[22] he figure 5.8 shows this structure. Basically they are multiple copies of a network, each of which passes a message to a successor.[23] The problem with simple RNNs is that they only have recent information available.

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.

[22] Cgl. Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, p.2.

[23] Cgl. "Understanding LSTM Networks". In: (2015).

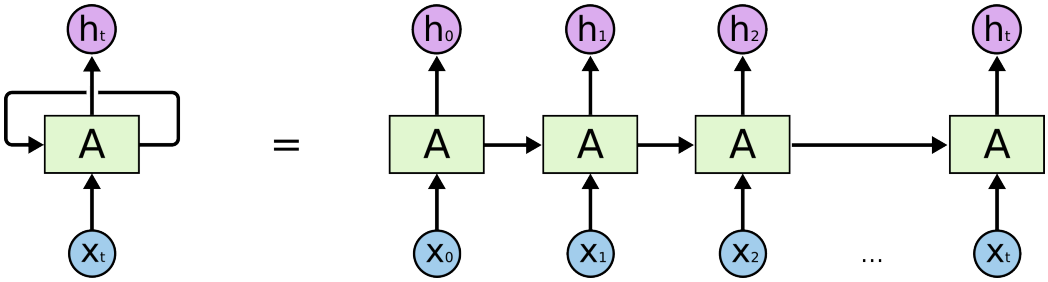


Figure 5.8: RNN network Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

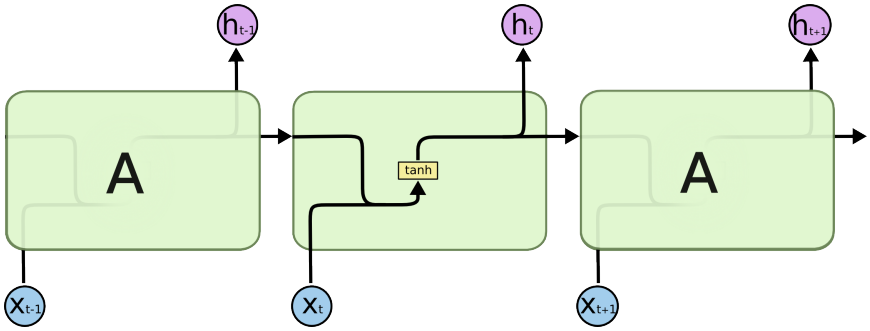


Figure 5.9: RNN source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

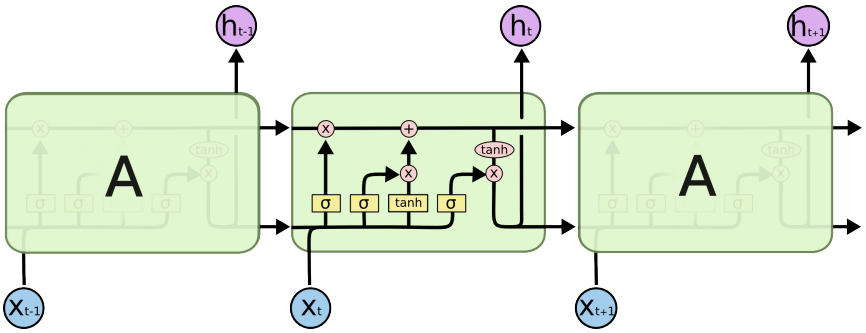


Figure 5.10: LSTM source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

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 an forget gate. These doors regulate the flow of information and decide what information should be kept.[24]

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 σ is the logistic sigmoid function, i, f, o, c are the input gate, forget gate, output gate and cell vectors[25]

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. 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

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 proposed by Lefferty et al..[26]

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,

[24] Cgl. Wei; Yu Kai Huang Zhiheng; Xu. *Bidirectional LSTM-CRF Models for Sequence Tagging*. Baidu research, 2015, p.2/3.

[25] Huang, *Bidirectional LSTM-CRF Models for Sequence Tagging*, p.2.

[26] Andre; C.N. Pereira Fernando Lafferty John; McCallum. *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*. University of Pennsylvania, 2001.

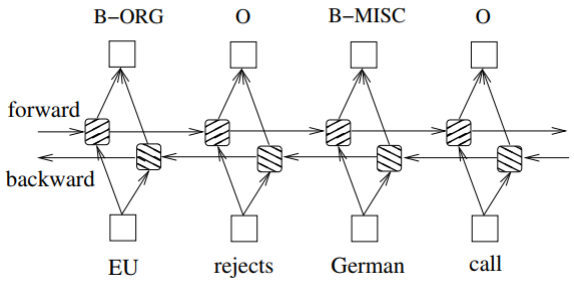


Figure 5.11: bidirection LSTM network source: https://arxiv.org/pdf/1508.01991v1.pdf

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 tackes the discriminative apporch of modeling conditional distributions p(y|x).[27] How this looks like in combination with a LSTM model can be seen in figure 5.12. In comparison with figure 5.11 now not only information about surrounding features is available, but also the sentence level tag information (other outputs of that sentence).

Dropout

Due to the predictive power of neural networks, they are prone to overfitting. The Dropout technique is used to prevent early overfitting. Dropout is only applied during training. In pytorch the dropout is automatically turned off when the model is set to eval mode. During training units of the neural network are randomly dropped and omitted from training. 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.[28] Figure 5.13 visualises how dropout changes a fully connected network into a only partially connected network.

Hinton et al. suggested a dropout of 0.5.[29] Testing different values did not improve the model.

[27] Cgl. McCallum Andrew Sutton Charles. *An Introduction to Conditional Random Fields.* 2012, p.269.

[28] Cgl. Geoffrey; Krizhevsky Alex; Sutskever Ilya; Salakhutdinov Ruslan Srivastava Nitish; Hinton. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research, 2014, p.1930.

[29] Cgl. Krizhevsky Sutskever Salakhutdinov Hinton Srivastava. *Improving neural networks by preventing co-adaptation of feature detectors.* University of Toronto, p.1.

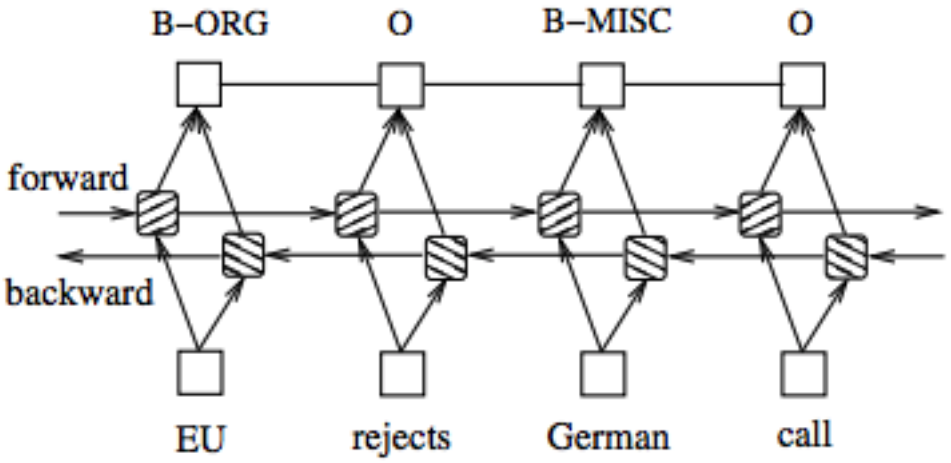
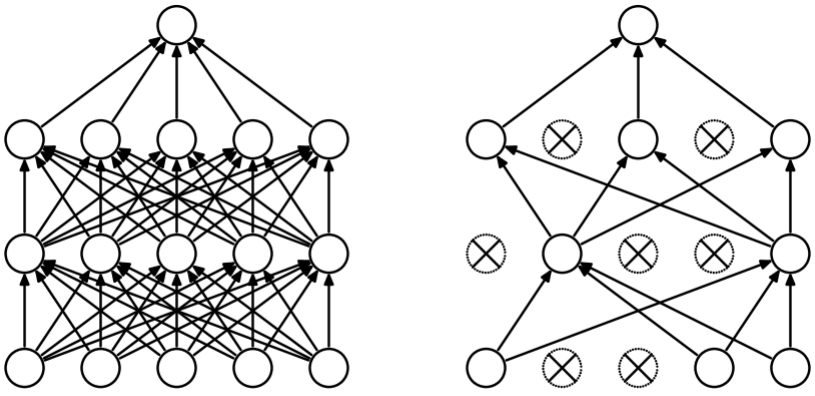


Figure 7: A BI-LSTM-CRF model.

Figure 5.12: Caption



(a) Standard Neural Net (b) After applying dropout.

Figure 5.13: Dropout source:
http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Word-Embedding

Since neural networks are designed to learn from numerical data, working with words(strings) is not really an option. Before Instead words are transformed into vectors, also called word embeddings. They were introduced by Mikolov et al. 2013.[30] Each word in the vocabulary is represented by a word embedding. The word embedding represents the semantic meaning of that word. The word embedding 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 integeers. 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 the Backpropagation algorithm. 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 embedding of the used words based on the context which it was used in the sentence. The output of the embedding layer is then propagated through the network, updating it's weights.

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 vocabulary from the start. 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[31] 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

[30] Kai; Corrado Greg; Dean Jeffrey Mikolov Tomas; Chen. *Efficient Estimation of Word Representations in Vector Space*. Google, 2013.

[31] Richard; D. Manning Christopher Pennington Jeffrey; Socher. *GloVe: Global Vectors for Word Representation*. Stanford University.

this additionaly to using word embeddings, character embeddings are used. They were first introduced by Zhang et al..[32]

Character embedding enables to represent the significance of these misspelled or unusual words itself. To encode the character-level information, character embeddings and a LSTM are used to encode every word to a vector. The resulting vectors are then fed into the main LSTM together with the learned word embedding.

Custom-Initialisation

Model Architectures

Different model setups were tested to observe the influence of the different techniques used. Portrayed F1-scores are validation scores. The results shown in table 5.17 show that the best performance is reached with the BI-LSTM CRF model utilising pre-trained embeddings and a char embedding layer. The biggest factors in terms of f1 score were using pre-trained word embeddings and character embeddings. The pre-trained word embeddings contributed additional information which couldn't have been captured by looking at the training data alone, while the character embeddings improved the model because the data contains a lot of misspellings and rare words. Using a bidirectional LSTM also had a small effect. The BI-LSTM with with pre-trained embedding performed 2.6% better than the unidirectional one. The dropout functionality generally did not increase the F1 score. In one case (BI-LSTM pretrainedEmbed + charEmbed + Dropout) it increased the score by 0.2%, but usually it decreased the score. It results in a network that converges slower in therefore needs more iterations to overfit on the training data, but that didn't increase overall performance. Using a crf layer only slightly increased f1 score (1%), suggesting that the surrounding tag information is not that informative.

5.4 Parameter Search

The model was tuned testing different learning rates and LSTM dimensions. Further tests were not possible because of limited resources. Portrayed F1 scores are validation scores. The model used for the tests was the best performing model from the Architecture section (BI-LSTM CRF pretrainedEmbed + charEmbed). The results of testing different learning rates

[32] Junbo; LeCun Yann Zhang Xiang; Zhao. *Character-level Convolutional Networks for Text Classification.* Courant Institute of Mathematical Sciences, 2016.

Model	F1-score
LSTM	9.0
LSTM pretrainedEmbed	21.8
BI-LSTM pretrainedEmbed	24.4
Bi-LSTM pretrainedEmbed + Dropout	16.0
BI-LSTM pretrainedEmbed + charEmbed	35.4
BI-LSTM pretrainedEmbed + charEmbed + Dropout	35.6
BI-LSTM CRF pretrainedEmbed + charEmbed + Dropout	34.7
BI-LSTM CRF pretrainedEmbed + charEmbed	36.4

Table 5.17

showed in table 5.14 show that the initially picked learning rate of 1e-3 was by far the best learning rate. A lower learning rate took too long to converge anywhere near a local optima while the higher learning rate (blue line) was too high and got stuck at just under a F1 score of 20.

The hidden dimension of the LSTM layer appeared to not have a huge influence(5.15. Usually a higher dimension leads to a more powerful network that is at the same time more prone to overfitting. The poorest results were produced by a model with a dimension of 1200, suggesting that a model with a dimension over 1000 is too powerful. Although this has to be taken with caution because of random aspect of neural nets another run could have led to different results. IF more resources would have been available a averaged test could have produced more reliable results. The best score was produced by a model with a hidden dimension of 1000 (36.4), closely followed by dimensions 300, 600 and 800. The score of 36.4 was previously also achieved by a network with a LSTM dimesion of 600 as can be seen in 5.17.

5.5 Comparison

The scikit-learn classifiers, the NLTK n-gram classifiers and the deep learning classifier were trained with the best found setups and the official wnut-17 evaluation script was executed on the results. Table 5.18 shows the result of the evaluation. The Bernoulli Naive Bayes classifer did the best with a F1 score of 17.61. The proposed deep learning classifier only came in third with a F1 score of 13.18.

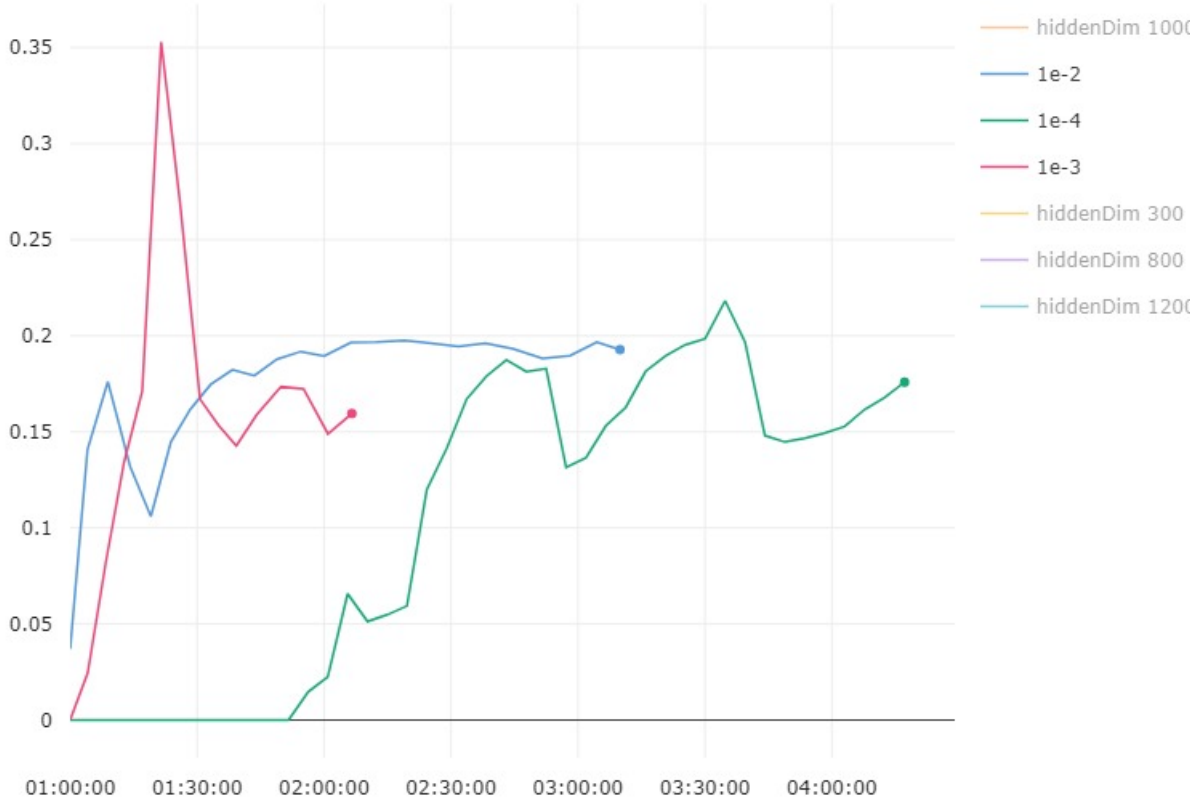


Figure 5.14: Comparison of learning rates
xAxis: F1 yAxis: Training time

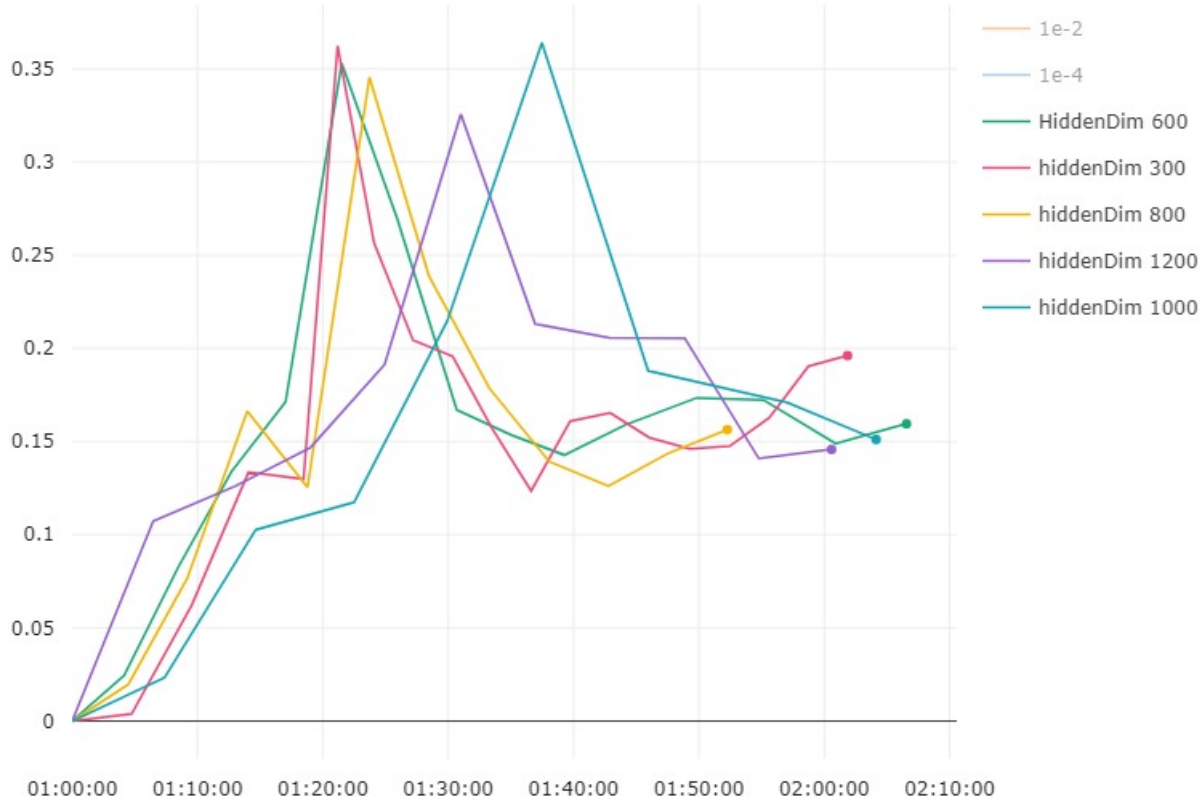


Figure 5.15: Comparison of LSTM hiden layer dimensions
xAxis: F1 yAxis: Training time

Classifier	Accuracy	Precision	Recall	F1-score
Unigram	92.54	18.18	2.97	5.10
Bigram - backoff	92.57	19.64	3.06	5.29
Trigram - backoff	92.56	19.53	3.06	5.29
Decision Tree	91.66	14.00	5.75	8.15
Bernoulli NB	91.91	21.11	15.11	17.61
Multinomial NB	92.00	19.40	10.84	13.91
SVM	92.71	35.27	7.88	12.88
GradientBoosting	92.63	32.11	6.49	10.79
LogisticRegression	92.53	32.72	6.42	10.79
BI-LSTM CRF pretrainedEmbed + charEmbed	90.13	13.23	13.13	13.18

Table 5.18

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 labeling 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.

Most state-of-the art NER approaches are now based on neural networks and deep learning. While using neural networks for NER is not new (one of the CoNLL 2003 submissions used a LSTM network[33]), the availability of more memory and data has made these approaches the dominant research direction. Recent advances in deep neural networks have led to a reduction in the number of hand-tuned features required for achieving state-of-theart performance. Insead word embedding have become the prevalent feature, which was first introduced by Mikolov et al.[34]. Also character embeddings were introduced as an additional feature.Zhang et al..[35] Furthermore Conditional Random field have been thrown into the mix by Lefferty et al. to profit from surrounding outputs. .[36]
Finally a similar model to the one proposed was introduced by Huang et al. **??** which also used a bi-directional LSTM CRF model.

[33] James Hammerton. *Named Entity Recognition with Long Short-Term Memory.* 2013.
[34] Mikolov, *Efficient Estimation of Word Representations in Vector Space*.
[35] Zhang, *Character-level Convolutional Networks for Text Classification*.
[36] Lafferty, *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*.

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 featureset are of crucial importance. The performance was significantly increased by doing an extensive search of optimal parameters and featureset. 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, 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 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. 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 network, but the knowledge was very basic.

Bibliography

[1] Jason Collobert Ronan; Westan. "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.

[2] Behrang Mohit. *Natural Language Processing of Semitic Languages*. 2014, p. 221.

[3] *NLTK*. URL: https://www.nltk.org/.

[4] *Scikit-learn*. URL: https://scikit-learn.org.

[5] *Pytorch*. URL: https://pytorch.org/.

[6] *Flask*. URL: http://flask.pocoo.org/.

[7] *Heroku*. URL: https://www.heroku.com/.

[8] *Alphabetical list of part-of-speech tags used in the Penn Treebank Project:* URL: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

[9] Edward; Klein Ewan Bird Steven; Loper. *NLTK Book*. 2009. Chap. 6. URL: https://www.nltk.org/book.

[14] *scikit-learn Users Guide - Naive Bayes*. URL: https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes.

[19] *sckit-learn Users Guide - Ensemble methods*. URL: https://scikit-learn.org/stable/modules/ensemble.html#forest.

[20] Eric; van Erp Marieke; Limsopatham Nut Derczynski Leon; Nichols. *Results of the WNUT2017 Shared Task on Novel and Emerging Entity Recognition*. 2017. URL: https://aclweb.org/anthology/W17-4418.

[21] *sklearn Documentation - sklearn.ensemble.GradientBoostingClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

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

[23] "Understanding LSTM Networks". In: (2015). URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[24] Wei; Yu Kai Huang Zhiheng; Xu. *Bidirectional LSTM-CRF Models for Sequence Tagging*. Baidu research, 2015. URL: https://arxiv.org/pdf/1508.01991v1.pdf.

[26] Andre; C.N. Pereira Fernando Lafferty John; McCallum. *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*. 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.

[27] McCallum Andrew Sutton Charles. *An Introduction to Conditional Random Fields*. 2012.

[28] Geoffrey; Krizhevsky Alex; Sutskever Ilya; Salakhutdinov Ruslan Srivastava Nitish; Hinton. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research, 2014. URL: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

[29] Krizhevsky Sutskever Salakhutdinov Hinton Srivastava. *Improving neural networks by preventing co-adaptation of feature detectors*. University of Toronto. URL: https://arxiv.org/pdf/1207.0580.pdf#.

[30] Kai; Corrado Greg; Dean Jeffrey Mikolov Tomas; Chen. *Efficient Estimation of Word Representations in Vector Space*. Google, 2013. URL: https://arxiv.org/pdf/1301.3781.pdf.

[31] Richard; D. Manning Christopher Pennington Jeffrey; Socher. *GloVe: Global Vectors for Word Representation*. Stanford University. URL: https://nlp.stanford.edu/pubs/glove.pdf.

[32] Junbo; LeCun Yann Zhang Xiang; Zhao. *Character-level Convolutional Networks for Text Classification*. Courant Institute of Mathematical Sciences, 2016. URL: https://arxiv.org/pdf/1509.01626.pdf.

[33] James Hammerton. *Named Entity Recognition with Long Short-Term Memory*. 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.