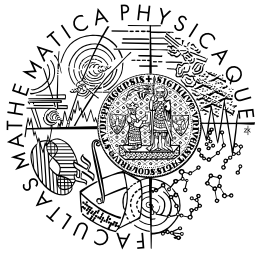


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Tomáš Pop

Named Entity Recognition and Its Application to Phishing Detection

Department of Software Engineering

Supervisor of the bachelor thesis: prof. RNDr. Tomáš Skopal, Ph.D.

Study programme: Computer Science

Study branch: Programming and Software Development

Prague 2022

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I want to thank my supervisor, Tomáš Skopal, and advisors, Jan Brabec, and Radek Starosta, for their professional guidance and valuable insights while writing the thesis. Moreover, I thank Cisco for providing the resources to conduct the experiments mentioned in this thesis. Lastly, I thank my family for their support and patience.

Title: Named Entity Recognition and Its Application to Phishing Detection

Author: Tomáš Pop

Department: Department of Software Engineering

Supervisor: prof. RNDr. Tomáš Skopal, Ph.D., Department of Software Engineering

Abstract: This thesis focuses on named entity recognition applied to email phishing detection. Named entity recognition is a classification task that aims to extract information from a text into a predefined set of categories (named entities), such as organizations, person names, or locations. The thesis describes various named entity recognition approaches, ranging from simple utilizations of neural networks to current state-of-the-art architectures. The most prevalent libraries and their models in named entity recognition are compared against each other from the computational and predictive performance perspective on the publicly available Enron email dataset. Moreover, differences in terms of named entities between positive (including phishing) and negative emails are measured on a proprietary dataset. Ultimately, the proprietary dataset is used for an experiment where a phishing email classification workflow is enriched with named entities to conclude whether named entities are helpful for the classifier to improve predictive performance. According to the experiment outcomes, a noticeable dissimilarity was measured regarding named entities in positive and negative emails. However, in the phishing email classification experiment with the provided dataset, it was concluded that named entities do not offer a convincing benefit. Though, it is possible that other conclusions would have been drawn using a different dataset.

Keywords: phishing detection, named entity recognition, neural networks, natural language processing, transformer

Contents

Introduction

The digital world is developing and expanding rapidly, and so do cybercriminals who utilize it for unjust enrichment by inflicting damage ranging from individuals to large corporations. Although cybercriminals have developed various methods to steal confidential information from victims, social-engineering-based attacks remain the most prevalent approach [1]. One of such social-engineering attacks is *phishing*, which according to the FBI's crime report in 2021, is the most common cybercrime in general [2].

Phishing, one of the initial access techniques [3], exploits the human factor to either trick the victim to reveal confidential information used for further exploitation or to infiltrate the internal infrastructure. Adversaries utilize various media to conduct attacks ranging from spoofed websites, social media phishing, and phone phishing to email phishing. This thesis focuses more thoroughly on phishing encountered in email correspondence, which comprises 96% of all phishing cases [4].

In recent years, there have been significant breakthroughs in the machine learning field regarding natural language processing (NLP), a computational technique for representing human language. Several attempts toward phishing detection based on natural language processing have already been conducted [5].

Moreover, NLP successfully contributed to the field of phishing email detection, for instance, by being able to recognize a sense of urgency, call-to-action (a suspicious request, such as a request to click a link, download an attachment, or update login credentials), or grammar errors abundant in mass phishing emails [6].

One of such NLP techniques whose usability for phishing detection will be thoroughly examined in this thesis is *named entity recognition* (NER). It is a task that aims to extract information from a text classified into a predefined set of categories (named entities), such as organizations, person names, or locations. NER may prove helpful since such named entities are typically abundant in emails (Figure 1), and various observations can be made, for instance, whether named entity distributions of positive (including phishing) emails differ from negative, harmless ones. Furthermore, the presence of certain named entities or combinations of them found in an email may prove to be a potential signal for phishing.

Figure 1: An example of a phishing email with highlighted named entities regarding a phony research opportunity at Cornell University coming from a non-Cornell account [7]

In recent years, named entity recognition experienced many advances in predictive performance. The thesis mainly focuses on neural network NER approaches aiming toward current state-of-the-art architectures. Moreover, off-the-shelf libraries used for named entity recognition are thoroughly described. Various libraries with their NER models are compared from computational and predictive perspectives to provide an overview of their tradeoffs.

Afterward, an experiment is conducted on a proprietary email dataset where named entity distributions of positive and negative emails are compared. Ultimately, the proprietary email dataset is further utilized to conclude whether named entities produced by the best performing model in terms of predictions added as an enrichment to a phishing email classification workflow prove beneficial in predictive performance.

1. Phishing

Phishing is a malicious technique that exploits the human factor to infiltrate the network. Targets ranging from individuals to large companies are contacted in the vast majority of cases by email either to obtain and misuse sensitive targets' data or, as an initial access technique [3] for further exploitation, to deploy malicious software, such as ransomware [4]. This chapter examines phishing techniques encountered in various communication media.

1.1 Email Phishing

Email phishing, making 96% out of all phishing cases, is the most prevalent phishing technique [4]. Phishing emails can be broken down into two categories: *high-volume phishing* emails sent in masses and *spear phishing* emails subtly designed for a targeted victim.

1.1.1 High-Volume Phishing

The most common approach in email phishing is to design a fraudulent email to trick a victim into revealing confidential data. Such data can range from credit card credentials, personal account credentials, contacts, or confidential data about the company where the victim is employed, for instance, by sending a spoofed link to a website to fill in a form with confidential data [8].

Another option adversaries frequently use phishing for is to deliver malicious software through a spoofed link or a malicious attachment, for instance, ransomware, to the internal infrastructure, potentially resulting in even more damage.

Phishing adversaries developed the emails and their following scenarios so sophisticated that it may be difficult to recognize them from regular legitimate emails. The adversaries typically register fake domain names which mimic existing organizations by slightly misspelling the organization name [9]. Furthermore, the design can be visually as convincing as possible. Typically, the message content may create a sense of urgency or threat to lower the chances that the victim looks for the email's authenticity. Since such emails are primarily targeted to be sent to as many victims as possible, these emails are usually not personalized, as seen in Figure 2.

1.1.2 Spear Phishing

Spear phishing is targeted primarily at a specific person or a group from an organization [11]. The adversary typically collects personal information about the victim beforehand, such as the full name, job position, specific information about the position, colleagues, family members, and other contacts. With collected profile information, the adversary aims to further increase email credibility, for instance, by impersonating a colleague or superior of the victim. Such emails are difficult to catch by an anti-phishing filter compared to the high-volume phishing method sent in masses.

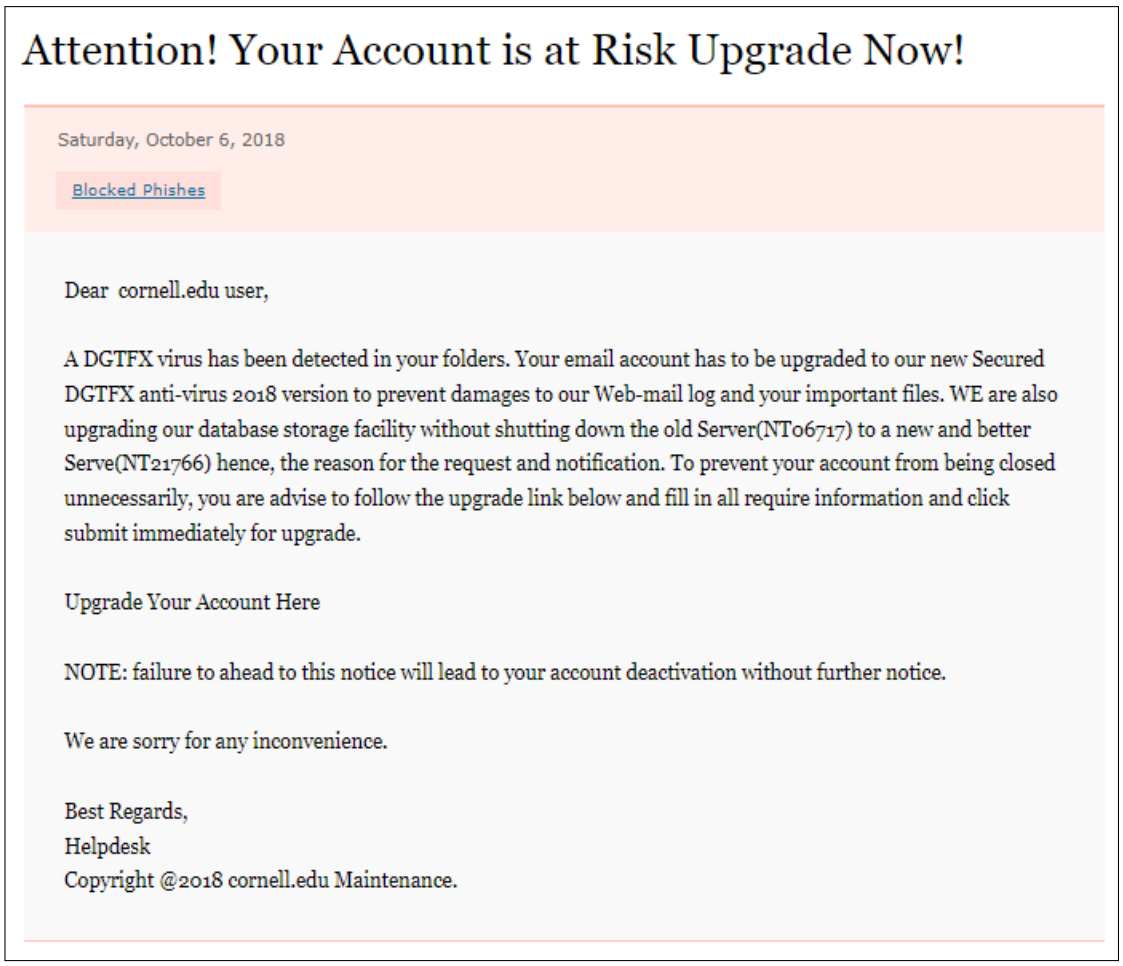


Figure 2: An example of a high-volume phishing email (impersonal, poorly written, containing grammar errors and urgency requesting confidential data) [10]

1.2 Social Media Phishing

Information shared over social media provide a powerful tool for anyone to perform a personal information survey. Furthermore, with the rapid growth of social media users, the social media phishing attack count is also increasing [12]. Adversaries found many opportunities for exploitation, such as account hijacking, impersonation, or malware distribution [1].

1.2.1 Personal Account Impersonation

Information surveying by cybercriminals, besides spear phishing, can be used for impersonation directly on a social media platform, for instance, by instant messaging. It was shown that users are more than four times more likely to be deceived when criminals impersonate to be an acquaintance of the target [13]. The adversaries utilize collected information to form their copy of the impersonated victim's profile to contact a friend or family member for requests, including credentials or financial aid.

1.2.2 Brand Impersonation

Another common tactic where fake accounts play their role is the impersonation of a well-known legitimate business. One of the options for how to proceed is to offer a phony promotion leading to a spoofed link where credentials are requested to confirm identity to further participate, for example, to claim a prize [12]. The accounts hijacked this way can be sold or even investigated further whether the victim uses similar credentials for other social media, email, or banking. Alternatively, the accounts can be used for malicious software spreading or to lure an acquaintance of the victim into the identic trap.

1.3 Phone Phishing

Phishing techniques mentioned above are applicable independently of the platform. Nevertheless, phishing over the phone offers inherently connected vulnerabilities to the mobile platform: phone calls and text messages.

1.3.1 Vishing

During a voice-over IP phishing, also known as vishing, phone call, adversaries may impersonate a trusted source, such as the victim's bank, claiming that there are problems regarding their account to convince a victim into revealing personal information and financial details.

1.3.2 SMishing

SMS phishing, also known as SMishing, is a form of phishing where the adversary entices the victim into revealing confidential information via a text message. For instance, the message may contain a security alert message from a bank that links to a malicious website or a fraudulent mobile application.

2. Named Entity Recognition

Named entity recognition is a natural language processing task that aims to extract information from text. The extracted information (named entities) consists of words in the text, such as person names, organizations, locations, dates, or geopolitical entities (Figure 3).

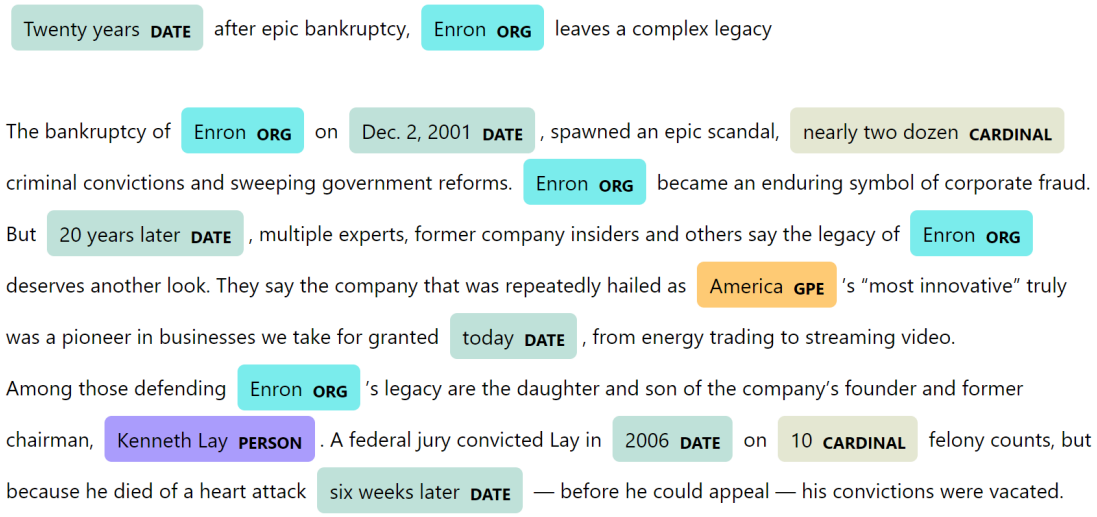


Figure 3: Visualizing named entities found in a news article [14] via spaCy [15]

Initially, named entity recognition as a *classification* task was approached by traditional *supervised machine learning* algorithms such as *Support vector machines* [16] or *Conditional random fields* [17].

Supervised machine learning uses datasets annotated with expected outcomes in advance to train a model using an underlying algorithm to classify data into a predefined set of discrete categories (classification) or predict real value outcomes accurately (regression) [18]. Each sample from the training dataset usually contains various features of the sample. Training means the model gradually finds the relationship between the features and the expected outcome annotations.

Moreover, additional features can be formed using *feature engineering* to discover new data by transforming, selecting, or manipulating input data, providing extra information for better pattern recognition [19]. Nevertheless, patterns found in the training data should be transferable to previously unseen data. In other words, generalize well. After training, the model's predictive performance is tested on an unlabeled dataset on which its generalization ability is examined.

Extensive feature engineering in many traditional supervised machine learning approaches for named entity recognition was crucial. Feature vector representation provided the abstraction over text where each word was represented using various boolean, numeric, and nominal values based on linguistic domain knowledge, such as the word's prefix, suffix, whether the word is capitalized, whether it ends with a period, or the length of the word [20].

Traditional supervised machine learning systems have retreated as deep neural networks emerged and dominated many natural language processing tasks [21]. Deep neural network approaches proved beneficial due to their ability to automatically learn feature representations from raw input data effectively, compared to traditional supervised machine learning approaches.

Initially, deep neural network successes in NER have been made using recurrent neural networks utilizing various methods for text representation [22]. However, with the rise of attention mechanisms, recurrent neural networks were displaced by state-of-the-art NER models utilizing a transformer architecture that proved to be a breakthrough throughout many natural language processing tasks [23].

2.1 Neural Networks

Neural networks are a machine learning method whose inspiration was seen in the biological principles of neurons in the human brain [24]. Many machine learning tasks, such as image recognition, speech recognition, natural language processing, and others, are nowadays approached using deep learning methods with a neural network backbone [25].

2.1.1 Perceptron

The first algorithmically described neural network is a perceptron (Figure 4), presented by Frank Rosenblatt [26]. The single-layer perceptron is the simplest neural network model that consists of a single neuron and is suited for binary classification with a linearly separable dataset.

To briefly introduce how the computation is made, consider feature variables as an input vector x and model weights w with a bias b. Net input $o(x)$ (Equation 2.1) is computed as a dot product between the input vectors and weights.

$$o(x) = x^T w + b = \sum_{i=1}^{n} x_i w_i + b \tag{2.1}$$

In order to obtain the output signal y (Equation 2.2), the net input is introduced to a non-linear activation function f, deciding whether a neuron will be activated or not [27].

$$y = f(\sum_{i=1}^{n} x_i w_i + b). \tag{2.2}$$

A non-linear activation function is required since a network with only linear activation functions would be equivalent to a linear regression model. Thanks to the non-linearity of the activation functions, neural networks can accurately capture complex relationships in the data [28]. In the perceptron case (Figure 4), a step function as an activation function f is applied, returning 0 for negative and 1 for positive values.

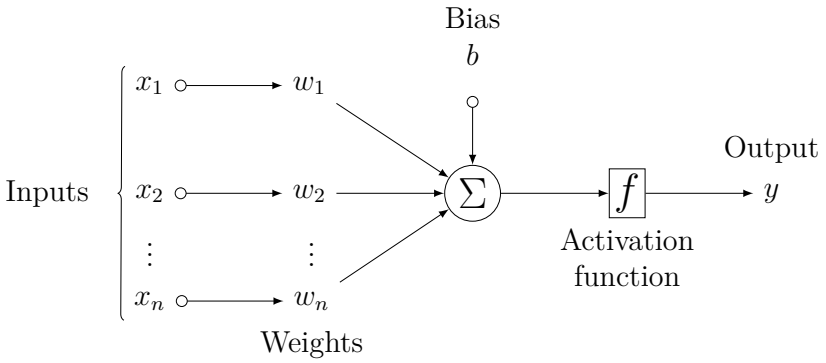


Figure 4: Perceptron

2.1.2 Multilayer Perceptron

Multilayer perceptron (MLP) tackles the limitation of the single-layer perceptron in linear separability. It consists of 3 types of layers: the input layer, one or more hidden layers, and the output layer. Moreover, each layer contains multiple perceptrons mentioned above (also called units) stacked together. Each unit is fully connected across the neighboring layers (Figure 5). Such a layer with fully connected units will be further referred to as a *fully connected layer*.

The input layer contains the number of units proportional to the input features count. When considering a classification task, such as named entity recognition, the unit count in the output layer equals the number of classes desired to classify. On the other hand, a single unit in the output layer is used with regression. Consequently, the unit count in the hidden layer can vary according to the needs of the task.

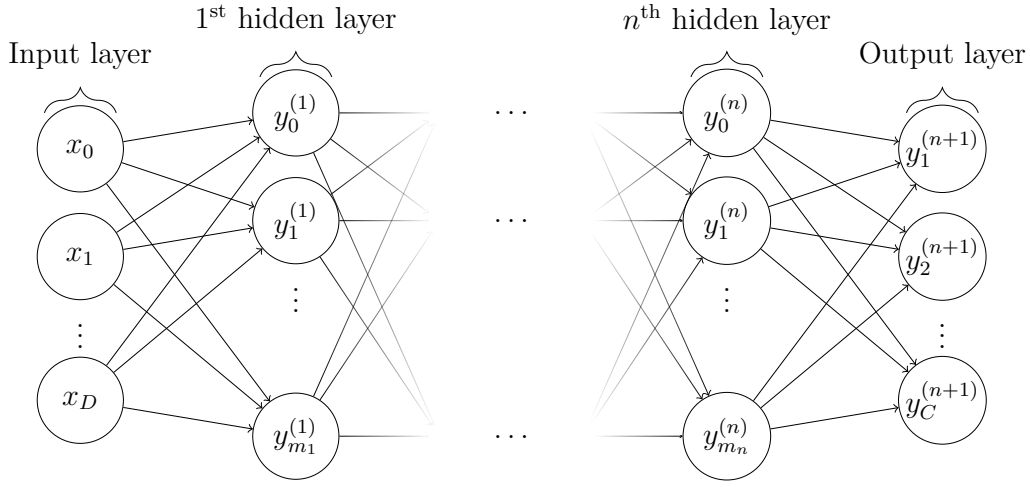


Figure 5: Multilayer perceptron classification with D input features to C classes

Universal Approximation

MLPs with a proper choice of activation function f, for instance, *Rectified linear unit* (ReLU)

$$f(x) = \text{ReLU}(x) = \max(0, x), \tag{2.3}$$

can approximate any function using a single hidden layer given enough units [29]. Nevertheless, the main concern is efficiency since, for some tasks, the number of units can be absurdly large. Therefore, it may be suitable for such neural network computations to opt for more sophisticated neural network architectures, as shown in Section 2.3.

2.1.3 Neural Networks Training

In supervised learning, the training goal is to perform well on previously unseen data with the lowest generalization error. The training process can be described as an iterative optimization problem where minor updates of the neural network model's weights are performed to minimize the *loss function* using the training dataset, changing the model's performance with each iteration [30].

Gradient Descent

An optimization algorithm is required to adjust the weights and change the loss function accordingly. Gradient descent (Equation 2.4) is one of the simplest optimization algorithms for iterative function minimization.

$$w^{(n+1)} = w^{(n)} - \alpha \nabla_{w^{(n)}} E(w^{(n)}) \tag{2.4}$$

α	learning rate, a tunable parameter, determining the step size at each iteration while moving towards the loss function minimum
$w^{(n)}$	weights in the n^{th} step of the algorithm
$\nabla_w E(w)$	the gradient of the loss function

Learning Rate

Learning rate defines how quickly or slowly the neural network updates its learned concepts [31]. On the one hand, a low learning rate may require plenty of time since the weight changes are too insignificant. On the other hand, the rapid changes with fewer epochs may result in divergent behavior and possibly never reaching an optimum (Figure 6) [32]. Therefore, the optimal learning rate should be low enough to reach convergence but high enough to be trained within a reasonable time [33].

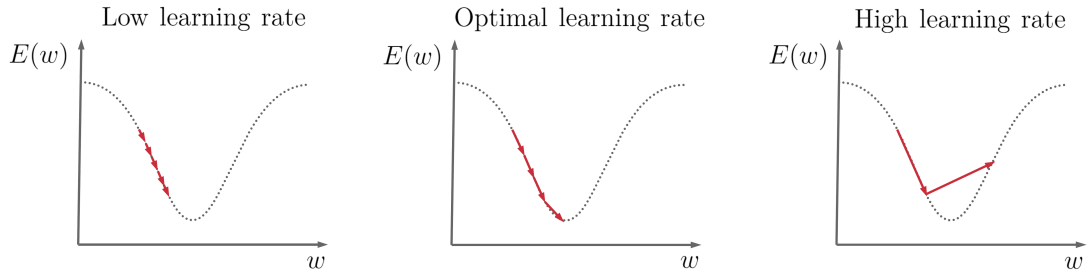


Figure 6: Impact on reaching an optimum with different learning rates [32]

Loss Function

A loss function usually consists of a sum over training examples of some per-example loss function [30]

$$\nabla_w E(w) = \frac{1}{m} \nabla_w \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, w). \tag{2.5}$$

L	a per-example loss
$x^{(i)}$	an example from the training dataset
$y^{(i)}$	an expected target value for the aforementioned example $x^{(i)}$

The approach is called *standard gradient descent* if all training examples from the dataset are used. Nevertheless, when dealing with large datasets where the training dataset grows to billions of examples denoted by m, the gradient computation can be inefficient since *O(m)* time complexity for each iteration is required. When considering the loss function computation as an expectation value (Equation 2.5), an estimation of *E(w)* can be made using a smaller set of training examples known as *mini-batch stochastic gradient descent* (Equation 2.6) [30]. In other words, for each step of the algorithm, a batch of B random independent examples $\{x^{(1)}, x^{(2)}, ..., x^{(B)}\}$ is sampled, resulting in time complexity reduction to a constant B.

In practice, the recommended values of B are of the power of two ranging from 32 to a few thousand, while satisfying that B fits the memory requirements of CPU/GPU [34]. For instance, when aiming for execution efficiency while making a computation with GPU, Nvidia suggests setting the batch size to 64, 128, or 256 since it helps better divide work between multiple parallel processes [35].

$$\nabla_w E(w) \approx \frac{1}{B} \nabla_w \sum_{i=1}^{B} L(x^{(i)}, y^{(i)}, w) \tag{2.6}$$

Multiple improvements to gradient descent algorithms have been made primarily by introducing the adaptive learning rate [36], such as AdaGrad [37], RMSProp [38], and the most used optimization algorithm in practice called Adam [39].

Softmax

For multiclass classification, as named entity recognition is, an output layer activation function called softmax is required, which maps a vector of size n of real numbers to a vector containing n real numbers that sum to one to be interpreted as probabilities (Equation 2.7).

$$\text{softmax} : \mathbb{R}^n \to (0,1)^n \tag{2.7}$$

The softmax is computed for each value in a vector z

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.8}$$

where $z = (z_1, ..., z_n) \in \mathbb{R}^n$, $n \geq 2$.

2.2 Text Representation

Before moving onto more sophisticated neural network architectures and proceeding to the NER state-of-the-art models, it is worth clarifying how to proceed when representing text. Since words are not a concept that most deep neural network models understand, converting the words into a numeric representation is necessary.

2.2.1 Bag of Words

One common way to represent text is using a bag-of-words model. It is a feature extraction method that preprocesses the text into a vector containing occurrences of words in the text given [40]. Therefore the size of the vector depends on the number of unique words encountered in the corpus, as shown in Table 1, which can be computationally expensive for training with large datasets. Moreover, other details about the words, such as their position in a document, position in the sentence, and context, are ignored.

	another	corpus	first	sentence	word
First corpus sentence.	0	1	1	1	0
Another sentence.	1	0	0	1	0
Word, word, word.	0	0	0	0	3

Table 1: Bag-of-words feature extraction for a simplistic three-sentence corpus

2.2.2 Word Embedding

Word embedding is one of the text representation techniques that found popularity in NLP, in general. It could be described as word mapping to real-number vectors (further referred to as *word vectors*) of a fixed size (Figure 7). Word embedding aims to train word representations via word vectors to capture the meaning, context, and relationship between words via their values. Furthermore, these word vectors are given as inputs to a classifier, for instance, a deep neural network.

To ensure that valuable embeddings are learned, a large amount of text data, for instance, millions or even billions of words, is required. A typical approach uses pre-trained word vectors learned via a word embedding algorithm on a massive corpus in advance, known as *pre-training* [41]. Afterward, the learned weights in word vectors are transferred to the original task, in this thesis, named entity recognition.

Various word embedding algorithms have been introduced, such as word2vec [42], GloVe [43], and fastText [44], each of them developed as open-source tools targeted for practical use.

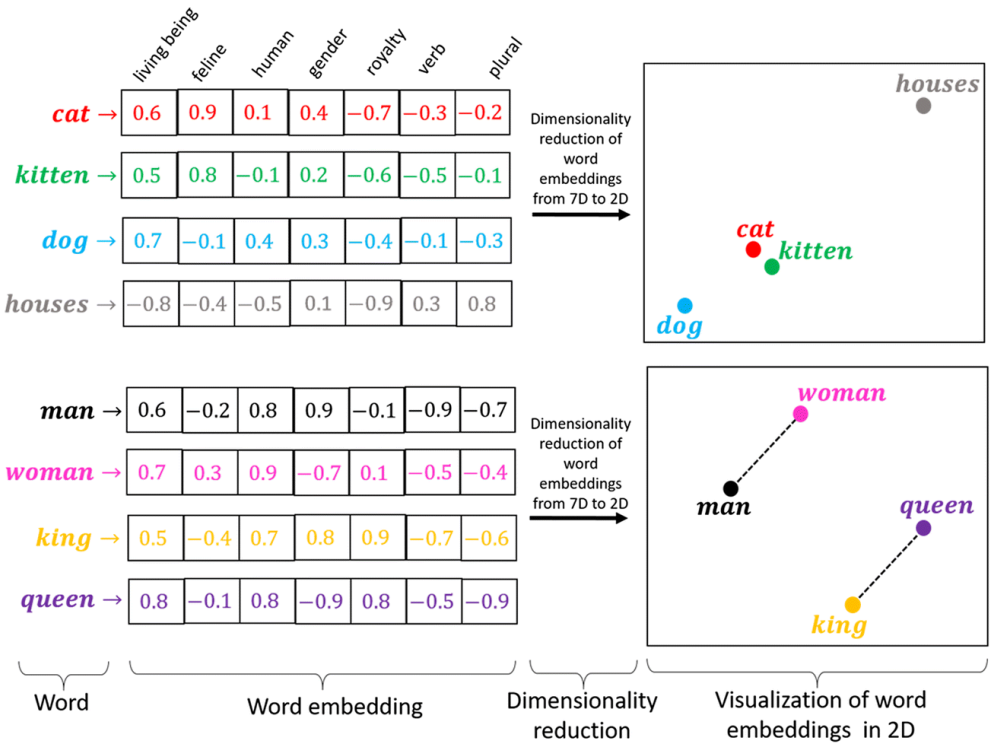


Figure 7: An example of word embeddings [45]

2.2.3 Word2vec

A word embedding algorithm that moved the NLP a big step forward is *word2vec* [42]. Word2vec was primarily created to learn word vectors from large datasets (billions of words) coming in two self-supervised variants: the *continuous bag-of-words* model (CBOW) and the *skip-gram* model (Figure 8). CBOW predicts the center word based on a window of surrounding context words. On the other hand, the skip-gram model uses the center word to generate its neighbor context independently.

Whether to choose one over the other was pointed out in the original word2vec paper. According to the original paper [42], the skip-gram model works well with smaller datasets and handles less frequent words better than CBOW. Nevertheless, CBOW takes less time to train and represents frequent words better. Furthermore, approximation training methods were introduced to speed up further the computation process, namely negative sampling and hierarchical softmax [46].

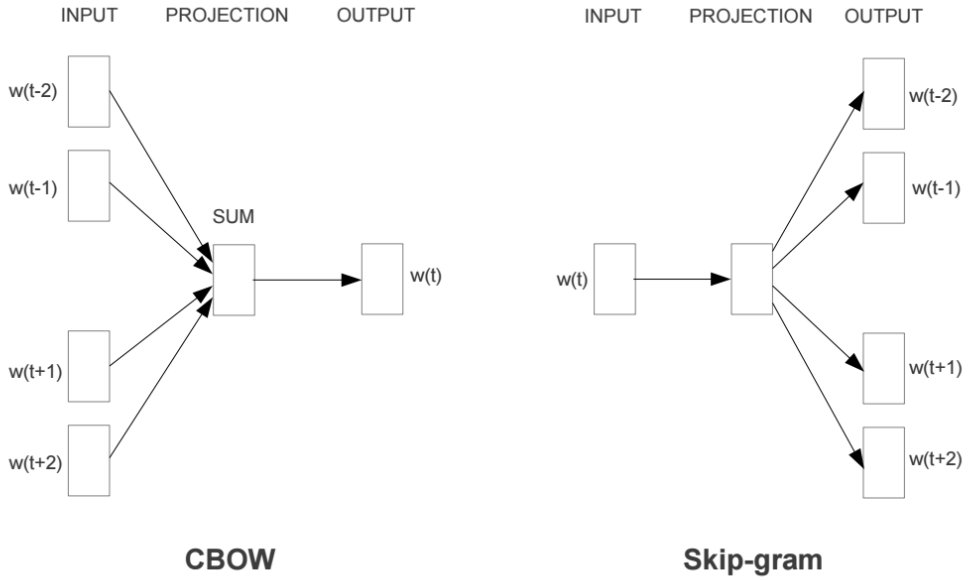


Figure 8: Word2vec models [47]

2.2.4 GloVe

Global vectors (GloVe) [43] word embedding algorithm extends the skip-gram word2vec by introducing the use of global statistical information regarding word co-occurrences. The GloVe model learns word vectors by examining word co-occurrences within a training dataset. A global co-occurrence matrix is constructed by counting how often a word w_i is found in the context (adjustable via the window size) of the word w_k. This process is performed across all training examples. Afterward, co-occurrence probabilities can be derived from the matrix where $P(w_k \mid w_i)$ denotes the probability of the word w_k appearing in the context of the word w_i.

GloVe showed that the ratio of word-word co-occurrences probabilities could capture the potential relationship between words [48]. Suppose target words *ice* and *steam* with their co-occurrence probabilities are given based on a few example context words from an otherwise large corpus containing 6 billion tokens, as shown in Table 2.

Probability and ratio	solid	gas	water	fashion
$P(w_k \mid \text{ice})$	$1.9 \cdot 10^{-3}$	$6.6 \cdot 10^{-5}$	$3 \cdot 10^{-3}$	$1.7 \cdot 10^{-5}$
$P(w_k \mid \text{steam})$	$2.2 \cdot 10^{-4}$	$7.8 \cdot 10^{-4}$	$2.2 \cdot 10^{-3}$	$1.8 \cdot 10^{-5}$
$P(w_k \mid \text{ice})/P(w_k \mid \text{steam})$	8.9	0.085	1.36	0.96

Table 2: Word-word co-occurrence probabilities and their ratios [43], [46]

For the context word $w_k = \text{solid}$, which is associated with *ice* but not *steam*, a large ratio is expected. Conversely, a small ratio is awaited for $w_k = \text{gas}$ connected to *steam* but not *ice*. For the word, $w_k = \text{water}$ linked with both the words in italics, the co-occurrence probabilities ratio is expected to be close to 1. Similarly, for the completely unrelated word $w_k = \text{fashion}$, the ratio should be again close to 1.

Nevertheless, the initial goal is to work with word vectors, so it is demanded to present the ratio, a scalar information value, in the word vector space [43]. Since the ratio depends on three words, it is necessary to come up with a function f which contains three word vectors, which eventually map into the desired ratio

$$f(v_{w_i}, v_{w_j}, v'_{w_k}) = \frac{P(w_k \mid w_i)}{P(w_k \mid w_j)}. \tag{2.9}$$

$v_{w_i}, v_{w_j} \in \mathbb{R}^d$ generated word vectors
$v'_{w_k} \in \mathbb{R}^d$ a word vector typically (given a co-occurrence matrix is symmetric) generated from the same set with a different random initialization

One of the options how to reduce the dimensionality via the function f is to perform a difference between the first two word vectors which does not change the dimensionality of the word vectors in any way and then to perform a dot product between the difference and the last word vector

$$f((v_{w_i} - v_{w_j})^T v'_{w_k}) = \frac{P(w_k \mid w_i)}{P(w_k \mid w_j)}. \tag{2.10}$$

2.3 Deep Neural Networks

Deep neural networks, neural networks that have several stacked layers of neurons, have achieved many successes in natural language processing [49]. Deep learning proved beneficial because it can automatically and effectively learn feature representations from unstructured data, such as text, compared to traditional supervised machine learning approaches where feature engineering typically based on domain knowledge is required.

Moreover, the predictive performance of deep neural network architectures can increase when provided with more data. In contrast, traditional machine learning methods struggle with predictive performance scaling beyond a certain data amount [50].

As a standard, how to approach named entity recognition was by utilizing deep neural networks while obtaining semantic content of words using various pre-trained word embeddings for textual representation from a massive corpus [22]. In recent years, NER state-of-the-art has utilized a transformer architecture which has been a breakthrough in various natural language processing tasks.

2.3.1 Recurrent Neural Networks

Considering sequential data, one of the main drawbacks of feed-forward neural network architectures, such as multilayer perceptron, is the lack of information persistence. Recurrent neural networks (RNN) address this issue by extending a feed-forward neural network with loops that allow information from one or many previous steps to be remembered via the *hidden state* [46].

In order to determine current outputs denoted by h_t, the network uses current inputs x_t with the hidden state h_{t-1} using the repeating module A (Figure 9).

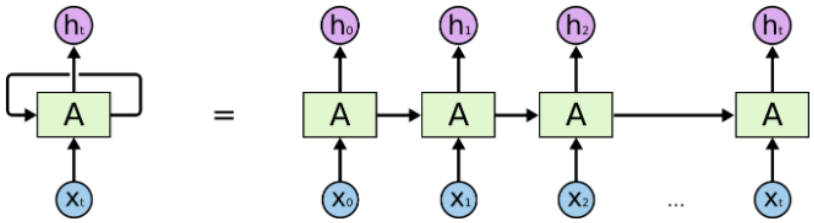


Figure 9: Chain-like nature of a recurrent neural network [51]

For instance, the repeating module of the *simple RNN* (Figure 10) is a *tangent hyperbolic activation function*

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.11}$$

Although, in theory, simple RNNs can learn long-term dependencies, they struggle to do so in practice [51]. Therefore different architectures emerged that still use the RNN chain; nevertheless, the inner repeating module A contains more *gates* (Figure 10) used to regulate incoming and outcoming information. Modern recurrent networks are the *Long short-term memory* (LSTM) [52] and its more computationally efficient variant, the *Gated recurrent unit* (GRU) [53].

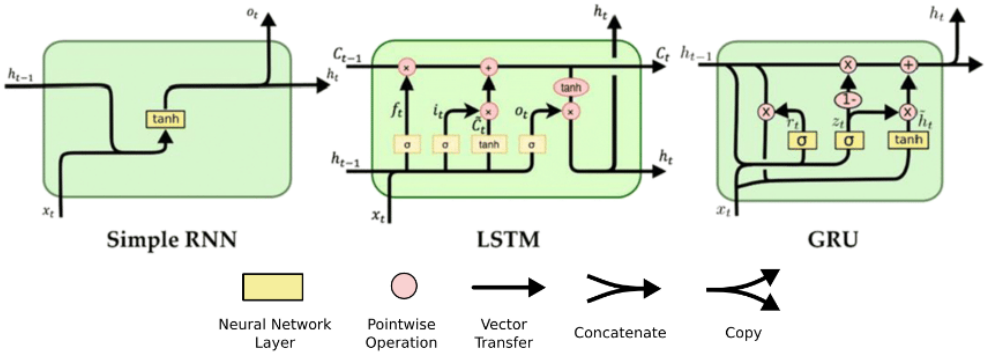


Figure 10: Recurrent neural networks cells overview [54]

Long Short-Term Memory

To handle long-term dependencies, which RNNs struggled with, Hochreiter and Schmidhuber [52] introduced a neural network called the Long short-term memory, where the chain RNN structure is still applied. Nevertheless, the significant difference was with adjustments in the repeating module which targets the long-term dependencies issue. The authors use a memory cell handled by an input gate i_t, forget gate f_t, and output gate o_t (Figure 11).

The gates consist of a *sigmoid activation function* σ (Equation 2.12), followed by a multiplication operation [51].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

An input gate is required if read data are supposed to be memorized and added to the memory cell. Furthermore, a forget gate is needed to reset the cell's content. In addition, an output gate governs reading entries from the cell. With this mechanism, the architecture should be able to distinguish between memorable and ignorable inputs in the hidden state.

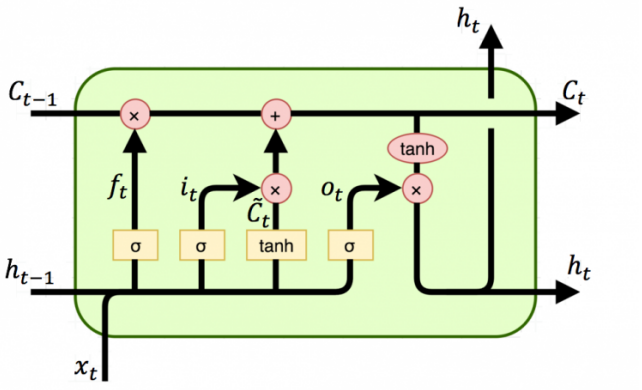


Figure 11: Long short-term memory [55]

Gated Recurrent Unit

Gated recurrent unit architecture [53] tackles the issue of long-term dependencies containing an update gate z_t and a reset gate r_t (Figure 12). The update gate plays a similar role as forget and input gates do in LSTM. It decides which information shall be preserved. A reset gate determines how much past information is supposed to be forgotten. Since GRU contains fewer gates and also fewer parameters than LSTM, the model is faster to train and execute. In small-scale datasets with not too long sentences, it is considered a preferred variant according to the comparison conducted by Chung et al. [56]. Nevertheless, the tradeoff between expressive power and computational efficiency remains unclear.

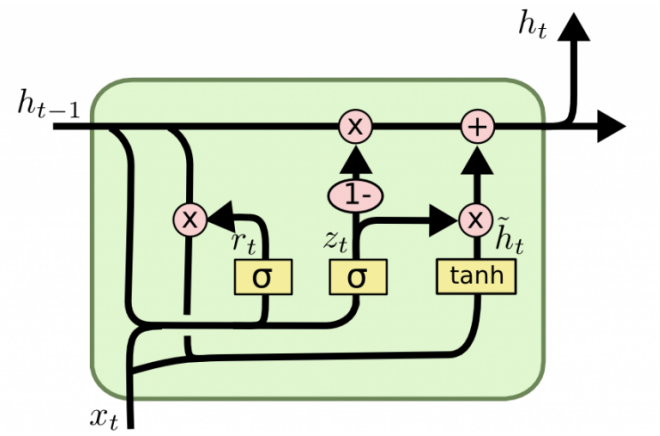


Figure 12: Gated recurrent unit [57]

2.3.2 Bidirectional Recurrent Neural Networks

Although there is no silver bullet regarding whether LSTM or GRU should be chosen, the results in many NLP tasks have shown that it is a suitable option to use a bidirectional variant of the recurrent network, especially for named entity recognition [58]. A bidirectional RNN (BiRNN) consists of two RNNs (LSTM or GRU of the same type): the first, which takes the input forward, and the second, in a backward direction. Afterward, the output vectors from both directions are added together. The added-up vector is then introduced to a fully connected layer and a softmax layer for classification, such as named entity recognition.

In the case of named entity recognition, a popular model which utilizes BiLSTM is shown in Figure 13. This model used word representations capturing both word-level and character-level features [59]. Generally, until transformers emerged, BiLSTMs with various word representations modifications were NER state-of-the-art back then [22].

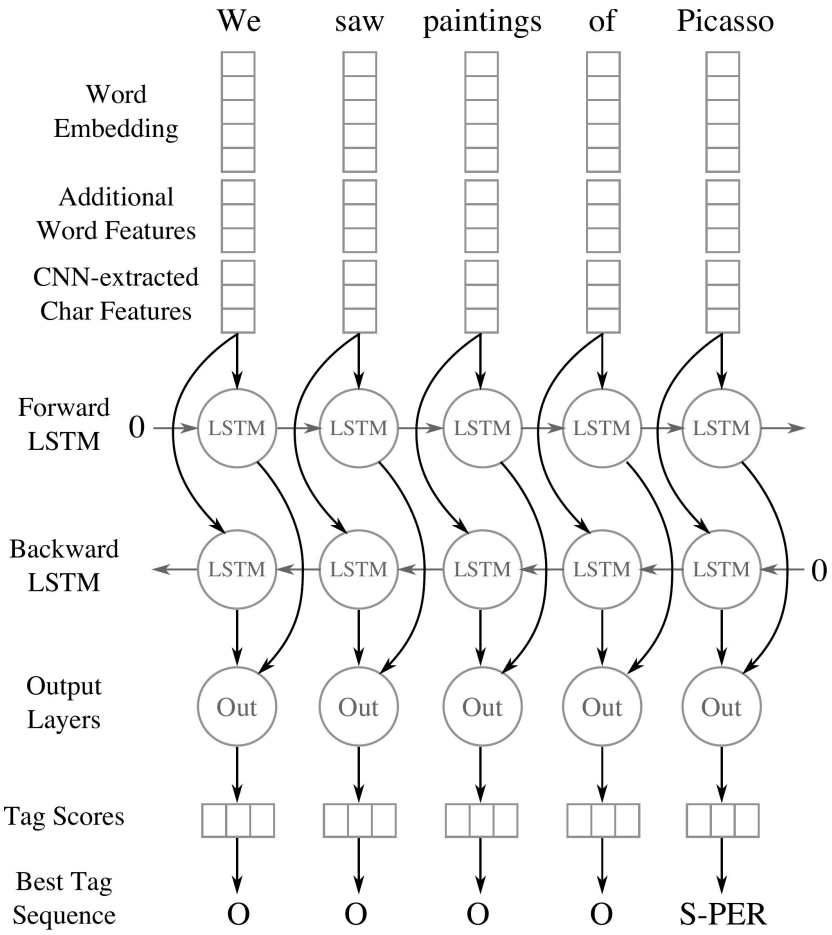


Figure 13: Named entity recognition using BiLSTM [59]

2.3.3 Transformers

Since the introduction of transformers in 2017, transformers have been the model of choice for many natural language processing tasks. a transformer architecture can be described as a *sequence-to-sequence* architecture that uses the *attention* mechanism to weigh the significance of parts of the input data [60].

Sequence to Sequence Learning

Sequence-to-sequence (seq2seq) learning was initially applied to neural machine translation [61]. There are two main components in the seq2seq architecture: an encoder and a decoder. In terms of RNN-based seq2seq architectures, the encoder and decoder are essentially RNN layers placed on top of each other. The encoder part gradually processes each item in the input sequence and encodes it into a fixed-length context vector (typically 256, 512, or 1024), further sent to the decoder. Consequently, the decoder proceeds to produce the output sequence item by item.

Although this approach had been state-of-art in terms of neural machine translation until transformer architecture was introduced, it does not deal with long sentences properly due to the fixed-length nature of the context vector. The RNN's long sequences processing issue was pointed out by Bahdanau et al. [62]. The authors introduce the attention, allowing the model to focus on relevant parts of the input sequence as needed.

The model does not require perfectly encoding a long sentence into a fixed-length context vector when using attention. However, it encodes only the parts of the input sentence of a particular word where the most relevant information is concentrated.

Attention

The inspiration for the attention mechanism in neural networks was found in natural human behavior when inspecting a visual scene. The optic nerve receives far more information than a human brain can process and the capability of the brain allows to allocate resources only to a fraction of information of interest while diminishing the rest. In the context of natural language processing, the goal is to be able to focus on relevant parts of the input sequence as needed [46].

Attention was introduced using recurrent neural networks as described earlier [62]. Moreover, the authors of the transformer architecture then further contributed by having no recurrence involved, relying solely on the attention mechanism.

Scaled Dot-Product Self-Attention

The initial self-attention mechanism introduced in the proposing transformer article [60] was *scaled dot-product self-attention*. Scaled dot-product self-attention is calculated using three vectors from each of the encoder's input vectors. A query, key, and value are created for each input vector by multiplying the input vector with trainable weight matrices W^Q, W^K, and W^V (Figure 14, on the left).

Considering a matrix X which contains input vectors packed together, query, key, and value matrices (Q, K, V) are obtained by performing dot products between the matrix X and the weight matrices mentioned above (Figure 14, on the right) [63].

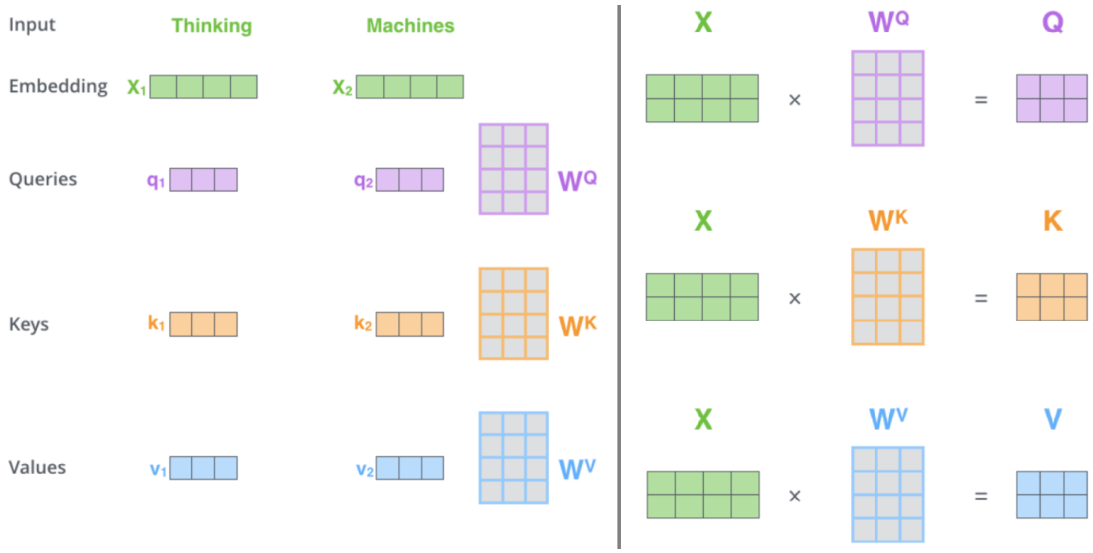


Figure 14: Scaled dot-product self-attention illustration [63]

The output matrix of the self-attention layer is computed as

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.13}$$

where the dot product QK^T is scaled by $\frac{1}{\sqrt{d_k}}$ mainly to stabilize gradients [60].

Multi-head Self-Attention

The authors also introduced the *multi-head self-attention* mechanism that helps with information extraction from multiple representation subspaces, which the scaled dot-product self-attention is incapable of. This mechanism has multiple sets of randomly initialized query, key, and value weight matrices. The number of sets is also known as heads, denoted by h. Afterward, the scaled dot-product self-attention is performed for each head separately (Figure 15), opening the opportunity for parallelization [60].

Since it is necessary to end up with the dimensions that a forthcoming feed-forward neural network accepts, the resulting matrices must be concatenated. The result of the concatenation is then multiplied by an additional matrix W^O (Equation 2.14) trained jointly with the model.

$$\text{Multihead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.14}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.15}$$

Q	a query matrix
K	a key matrix
V	a value matrix
$head_i$	the i^{th} attention head
W_i^Q	the query weight matrix for the i^{th} attention head
W^O	an additional trainable matrix used for dimensionality reduction

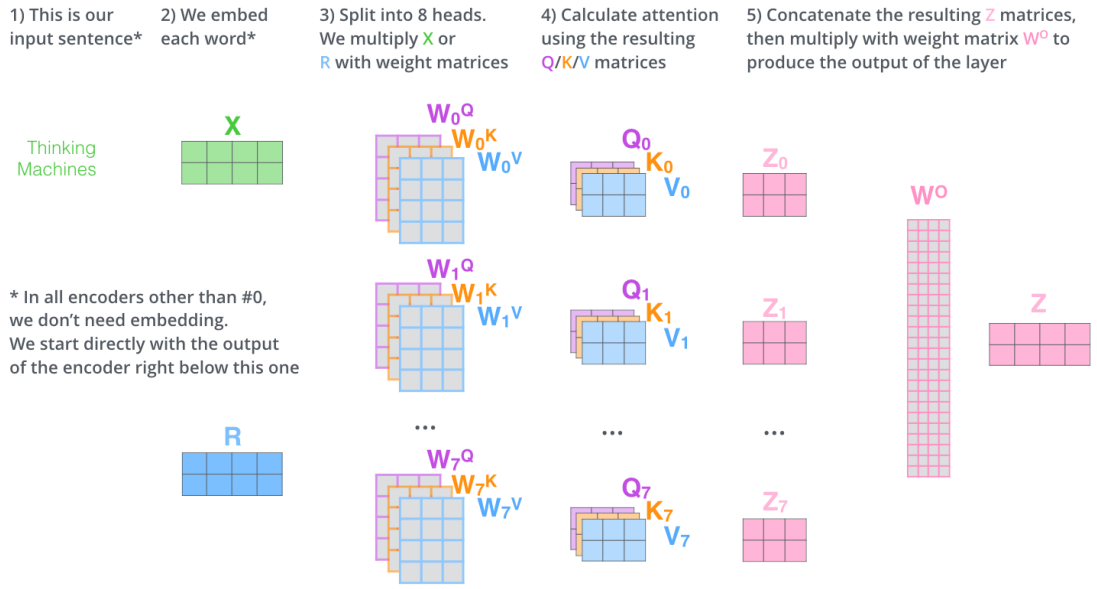


Figure 15: Multi-head self-attention illustration [63]

Model

The transformer model as a seq2seq architecture contains N encoders and decoders. Each encoder and decoder contains sub-layers. The encoder's inputs flow through a self-attention layer. The outputs of the self-attention layer are sent to a feed-forward neural network. The decoder contains both these layers, while between them is a special attention Encoder-Decoder layer which will be further described later. Each sub-layer has a residual connection around itself, followed by a layer normalization step that recenters and rescales across the features (the Add & Normalize step in Figure 16). This technique is mainly used since it consistently helps with convergence acceleration [46].

In order to represent words for the model, it is required to start by transforming each input word into a vector via word embedding. Since the model does not contain any recurrence, to store information about the relative

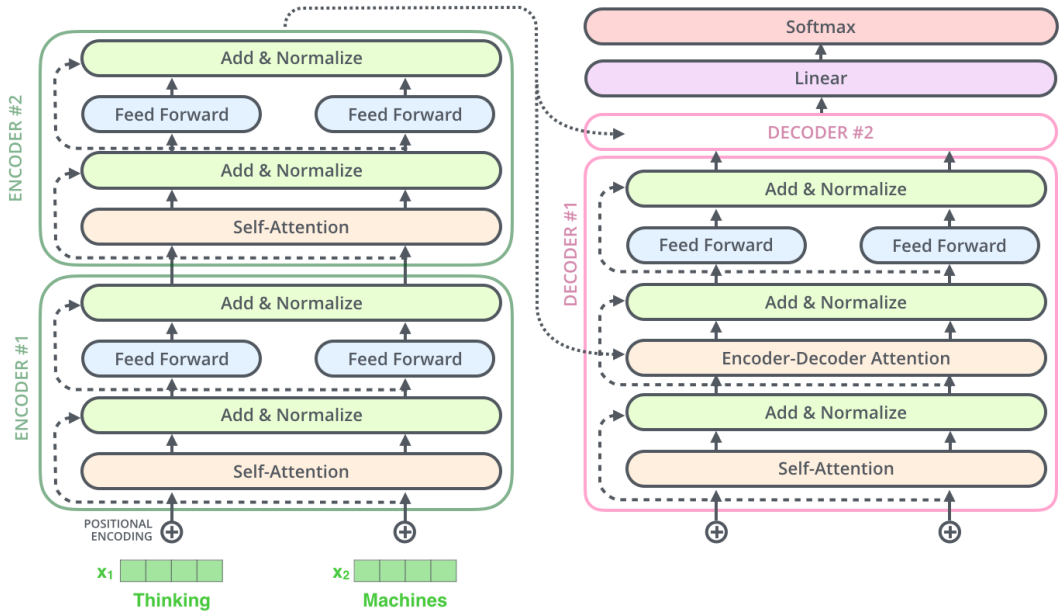


Figure 16: Transformer architecture with two encoders and decoders [63]

position of each token in the sequence, positional encoding is added to the embedded vectors (the lower part of Figure 16). The encoder receives a list of vectors as an input. These vectors are passed into a self-attention layer and afterward into a feed-forward neural network, where the output is further sent to the next encoder.

As the model proceeds to the top encoder, its output is transformed into a set of attention matrices K and V, which is further used in all Encoder-Decoder attention layers in decoders as their keys and values (the center of Figure 17).

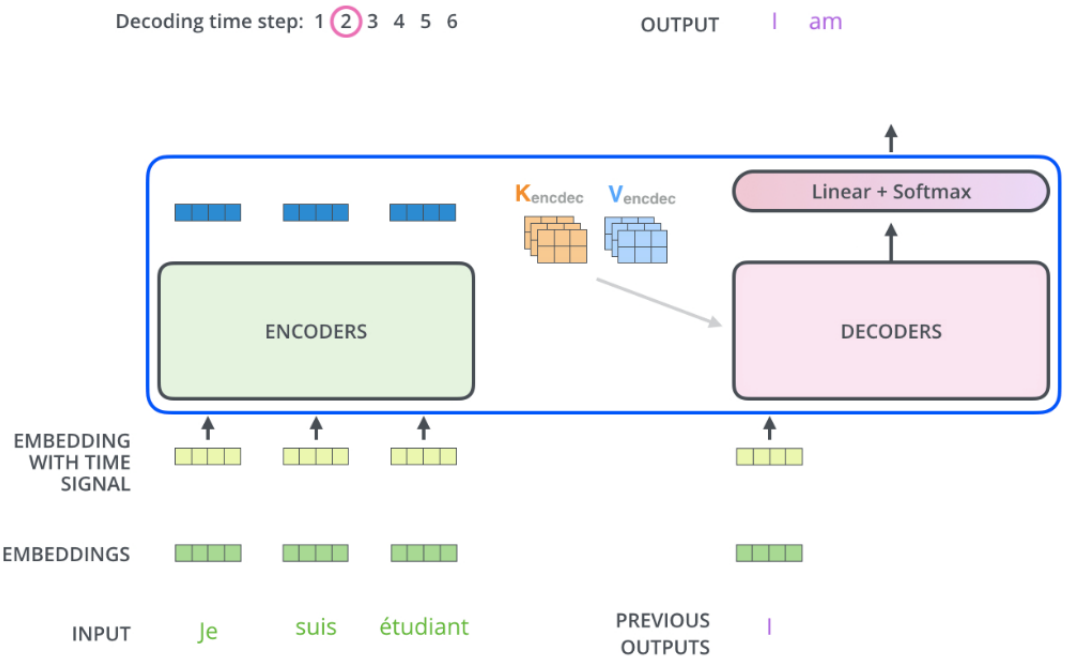


Figure 17: Transformer decoding [63]

In the decoder part, the self-attention layer can only attend in the back direction in the output sequence since future positions are masked before the softmax step in the self-attention calculation. The Encoder-Decoder attention layer works similarly to multi-head self-attention. With the exception, it does not take the keys and values matrix from the previous layer but from the top-most encoder [63].

Consequently, the vector needs to be represented as a word again. In the end, the linear and softmax layers turn the vector into probabilities. Naturally, the cell with the highest probability is chosen, and the word associated with it is produced as the output, which finalizes the transformer architecture (Figure 18).

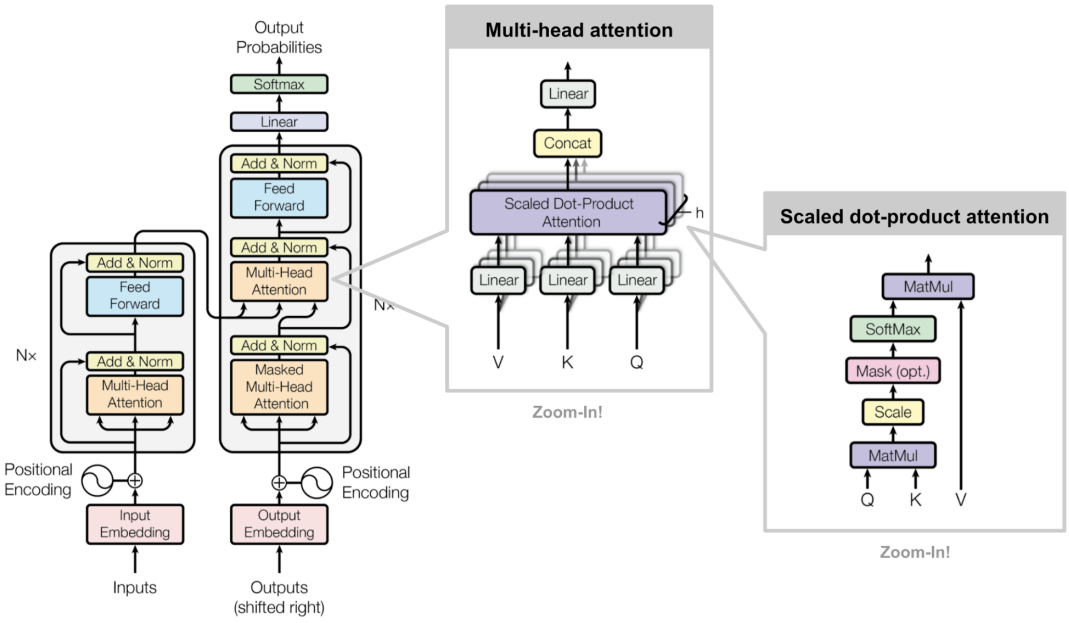


Figure 18: Transformer architecture with its self-attention mechanisms [63]

2.3.4 Bidirectional Encoder Representations from Transformers

Bidirectional encoder representations from Transformers (BERT) [64] come with the innovation of using the transformer above's attentive training for language representations [65]. Since the transformer consists of the encoder for reading the text input and the decoder for the translation, BERT needs only the encoder, respectively multiple stacked encoders.

Pre-Training

The BERT model is pre-trained on unannotated data from the BookCorpus [66], consisting of 985 million words and another 2.5 billion words from English Wikipedia. The input representation is done as in the transformer model (via word embedding and positional encoding). The pre-training phase remains the same for each natural language processing task. It consists of two tasks executed simultaneously: masked language modeling and next sentence prediction.

Bidirectional context encoding for each input representation is performed using **masked language modeling**. BERT randomly masks a fraction of tokens (in the proposing article, 15% of them). Furthermore, it uses tokens from the bidirectional context to predict the masked tokens. Moreover, 80% of the masked tokens are replaced by a special [MASK] token, a random word replaces 10% of the masked tokens, and the rest 10% remain without any changes [65].

Next sentence prediction (NSP in Figure 19), the second pre-training task, helps better understand the relationship between multiple sentences [64]. Sentence pairs generated exclusively for pre-training purposes are half of the time valid consecutive pairs. In the rest, the second sentence is randomly sampled from the corpus, which has no relationship with the first sentence. BERT is afterward given a task to correctly classify whether the following sentence is related to the leading one or not [46]. The result of the pre-training is the weights containing contextual embeddings for words.

Fine-Tuning

Using the pre-trained weights, BERT can be fine-tuned by adding a classifier (fully-connected layer followed by a softmax layer) to accomplish the initially intended natural language processing task, here named entity recognition (Figure 19). The fine-tuning for NER consists of training on a much smaller annotated NER-specific dataset, such as CoNLL 2003 [67] or OntoNotes v5 [68], which is supposed to accurately recognize named entities from a predefined set via a softmax layer. BERT tags each token in a sentence either with an entity type or a blank symbol, for instance, by O, meaning that the token is not a named entity.

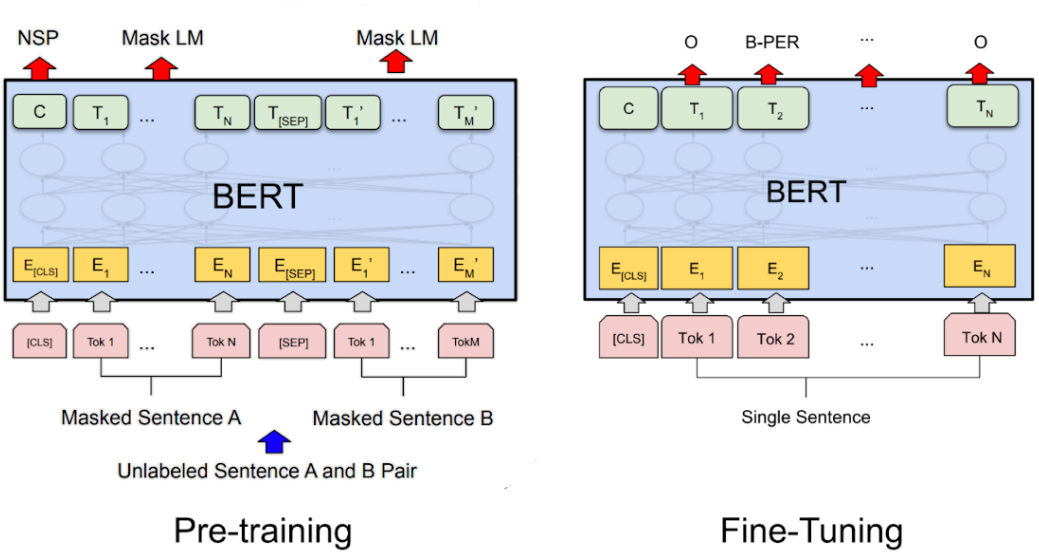


Figure 19: BERT for named entity recognition (E denotes the input representation, T the output vectors) [64]

3. Software for Named Entity Recognition

Many open-source tools that support named entity recognition already exist. This section will describe available datasets, libraries, and models used for experiments on email data, as described in Chapter 4.

3.1 Datasets

There are many datasets designed for fine-tuning NER models. Nevertheless, the most prevalent NER English datasets [22] are CoNLL 2003 [67] and OntoNotes v5 [68].

3.1.1 CoNLL 2003

CoNLL 2003 (Conference on Natural Language Learning) [67] is a NER dataset released in English and German versions. The English data was collected from Reuters news ranging from August 1996 and August 1997. CoNLL 2003 recognizes four types of entities (Table 3).

3.1.2 OntoNotes v5

An alternative dataset for NER fine-tuning is OntoNotes v5 [68], released in 2013, comprising various text genres (news, web blogs, broadcast news, talk shows, and others) in English, Arabic, and Chinese. Compared to CoNLL 2003, it offers a broader range of 18 entity types (Table 4).

3.2 SpaCy

SpaCy [15] is an open-source natural language processing Python library. SpaCy contains various linguistic features, ranging from tokenization, sentence segmentation, and the word vectors similarity to part-of-speech tagging or extraction of named entities in multiple language pipelines [69]. The library contains two variants of models targeted for English: a simpler model which uses a transition-based approach and a more complex model based on the transformer architecture. Both models are fine-tuned on OntoNotes v5.

Entity	Description
O	Outside of a named entity
PER	Person's name
ORG	Organization
LOC	Location
MISC	Miscellaneous entity

Table 3: Named entity types in CoNLL 2003

Entity	Description
CARDINAL	Numerals that do not fall under another type
DATE	Absolute or relative dates or periods
EVENT	Named hurricanes, battles, wars, sports events, etc.
FAC	Buildings, airports, highways, bridges, etc.
GPE	Countries, cities, states
LANGUAGE	Any named language
LAW	Named documents made into laws.
LOC	Non-GPE locations, mountain ranges, bodies of water
MONEY	Monetary values, including unit
NORP	Nationalities or religious or political groups
ORDINAL	”first”, ”second”, etc.
ORG	Companies, agencies, institutions, etc.
PERCENT	Percentage, including ”%”
PERSON	People, including fictional
PRODUCT	Objects, vehicles, foods, etc. (not services)
QUANTITY	Measurements, as of weight or distance
TIME	Times smaller than a day
WORK_OF_ART	Titles of books, songs, etc.

Table 4: Named entity types in OntoNotes v5 [68]

3.2.1 Transition-Based Model

The transition-based model [70] included in the spaCy library is a multitask model that utilizes a pre-training and fine-tuning learning approach.

Pre-Training

During pre-training, the input text is tokenized into words where each word is embedded. The embedding in this model works on a sub-word level and consists of four attributes: norm, prefix, suffix, and shape. The norm is essentially a lowercase variant of the input word. The shape maps lowercase characters of the input word to w, uppercase characters to W, and numbers to d. These four lexical attributes of an i^{th} word are separately embedded to vectors [71] of size n (in spaCy, 128), and the attributes are afterward concatenated. Since it is required to end up with dimensionality n, the concatenated result is passed through a fully connected layer (FC_i) of n units with a Maxout activation function (Equation 3.1) which, according to the library author, compared to ReLU, produced slightly better results.

$$\mathrm{Maxout}(x) = \max(w_1^T x + b_1, w_2^T x + b_2) \tag{3.1}$$

The output of the fully connected layer FC_i can be described as a vector v_i of size n. Nevertheless, v_i lacks context. Context encoding in the spaCy transition model is constructed using four previous and four following words. In the first iteration, the previous, current, and following word vectors are concatenated, forming a *trigram* $v_{i-1}v_iv_{i+1}$. Again, the trigram is reduced via another fully

connected layer of n units denoted by w_i. Afterward, the vectors v_i and w_i of size n are summed, denoted by x_i.

In the next iteration, x_i is used as input, and its immediate neighbors are v_{i-2} and v_{i+2}, whose concatenation forms the trigram $v_{i-2}x_iv_{i+2}$. After the reduction via a fully connected layer to n classes, a vector y_i is created. The sum of x_i and y_i is used as an input in the next iteration. Eventually, the output after the fourth iteration is the desired word vector for the i^{th} word with both sub-word and contextual features.

Fine-Tuning

With the pre-trained weights, the model is fine-tuned for NER using a transition-based approach [72]. The model transfers its knowledge to a NER-specific OntoNotes v5 dataset. The inference is made using a state machine with an entity stack, input buffer, and output buffer. From the input buffer, the model takes a word in each iteration. If the input buffer and entity stack are empty, the state machine enters a final state. The machine takes one word in each step, representing it using the approach described in the pre-training step combined with pre-trained weights.

Obtained features are then the input into a multilayer perceptron classifier, which, based on the features, decides whether to add an entity to the entity stack, move the entity stack with the entity type to the output buffer, or continue without changes. Therefore, the final fine-tuning task lies in predicting the correct sequence of actions supposed to be undertaken in a state machine.

The transition-based model comes in three variants: small model `en_core_web_sm` (12 MB), which does not use static word vectors. On the other hand, the medium `en_core_web_md` (31 MB) and large `en_core_web_lg` (382 MB) models use GloVe [43]. The difference between the medium and large model is that the medium model keeps only 20 000 most frequent word vectors compared to the large which includes the full set of 343k word vectors [73].

3.2.2 Transformer-Based Model

With the release of spaCy v3, the transformer-based model `en_core_web_trf` for NER was introduced [74]. The model is *Robustly optimized BERT approach* (RoBERTa) [75], which utilizes BERT architecture with a modified pre-training procedure. The optimization lies primarily in the masked language modeling task and the devotion of more quantity of data to pre-training.

In the original BERT article [64], it is proposed that random masking is performed only once at the beginning, and the model is learning the contextualized word representations. In practice, it was shown that it is suitable to duplicate the pre-training data with different masking patterns not to get the same mask when the same training sequence is encountered over multiple training epochs [75]. RoBERTa further extends this temporary solution by introducing *dynamic masking*, which generates a new masking pattern every time a sequence is fed to the model. While training on larger datasets (RoBERTa uses approximately 10x more data than the original BERT), this method proved to be a viable option.

Furthermore, RoBERTa restricts that each input contains complete sentences contiguously sampled from one or more documents, for instance, an article, of length at most 512 tokens. When the end of the document is reached, the model proceeds to the next one, with a separator token placed between them. It was shown that with this setup, the next sentence prediction task during pre-training did not help performance and was removed completely [75].

The spaCy model comes in a $\text{RoBERTa}_{\text{base}}$ variant which uses $\text{BERT}_{\text{base}}$ architecture (12 encoder layers stacked on top of each other, 12 attention heads) with the modified pre-training approach mentioned earlier.

3.3 Hugging Face

Hugging Face is a platform with state-of-the-art machine learning models, mainly focused on transformers [76]. Hugging Face offers many BERT model variants and modifications for various tasks via their transformers library [77]. However, in the thesis, there have been utilized only the implementations of the original BERT paper [64] fine-tuned for named entity recognition, namely `dslim/bert-base-NER` ($\text{BERT}_{\text{base}}$) [78] and `dslim/bert-large-NER` ($\text{BERT}_{\text{large}}$)[79].

The only difference between the models is that $\text{BERT}_{\text{base}}$ contains 12 encoder layers stacked on top of each other, whereas $\text{BERT}_{\text{large}}$ contains 24 [80]. Furthermore, $\text{BERT}_{\text{base}}$ has 12 attention heads, while $\text{BERT}_{\text{large}}$ contains 16. Both models are fine-tuned on a CoNLL 2003 dataset.

3.4 Flair

Flair [81] is an open-source natural language processing library developed in Python. It contains fine-tuned models both on CoNLL 2003 and OntoNotes v5.

3.4.1 BiLSTM Model

Initially, Flair library mainly contained BiLSTM models, such as `ner-english-fast` for CoNLL 2003, `ner-english-ontonotes-fast` for OntoNotes v5. The models are pre-trained using GloVe embeddings concatenated with their own contextualized string embeddings [82].

3.4.2 Transformer-Based Model

Flair also contains transformer-based models `ner-english-large` and `ner-english-ontonotes-large` with state-of-the-art efficacy [83]. Both models use a multilingual RoBERTa (XLM-R) pre-trained on cleaned 2.5TB of Common Crawl data [84] for 100 languages [85]. Besides using the word representations from the transformer model, static word embeddings are concatenated with them (for English, GloVe [43], for other languages, fastText [44]). The flair large model comes in a $\text{RoBERTa}_{\text{large}}$ variant which uses $\text{BERT}_{\text{large}}$ architecture with the modified pre-training approach.

4. Experiments

This chapter describes the experiments conducted aiming toward phishing detection using named entity recognition. Initially, it is expressed how a named entity recognition task can be approached and how it can be used on a publicly available dataset, such as the *Enron email dataset* [86]. Since various options can be utilized, as mentioned in Chapter 3, performance and accuracy measurements of specific models are investigated.

Furthermore, the named entity recognition approach is applied to a proprietary dataset. Additionally, this part compares named entities found in positive and negative emails. Its mutual similarity or difference is assessed via statistical methods.

In the last experiment, features extracted via named entity recognition are added to a proprietary phishing classifier, where their results are compared with and without the features. Eventually, a conclusion is made whether or not their presence is a helpful factor in phishing detection.

4.1 Enron Email Dataset

One of the most prevalent public datasets containing emails is the Enron email dataset [86]. The dataset contains 517 401 email messages written or received by 150 Enron Corporation employees.

4.1.1 Sentence Parsing

Since the models from the libraries mentioned earlier are fine-tuned on sentences, the context they are allowed to capture needs to be within the sentence. Two main options were considered to parse the emails into sentences. The first option, Sentencizer [87], available with spaCy installed, provides simple sentence parsing using a rule-based approach. A viable alternative was to use a SentenceRecognizer [88] by spaCy, a trainable sequence tagger, which offers a more accurate sentence parsing, especially in sentences where punctuation is missing or incorrect. However, the difference between the methods in terms of accuracy was too insignificant. Therefore, Sentencizer was favored due to approximately 2x faster runtime on average.

4.1.2 Model Choice

Various off-the-shelf solutions suited for named entity recognition already exist. There have been multiple aspects that were taken into consideration when filtering candidate models for further experiments.

Fine-Tuning Dataset

One of the initial decisions was to choose the models that were fine-tuned on OntoNotes v5. Although both datasets support person, organizations, and location, OntoNotes further offers a greater diversity of entities that may be

		Predicted	
		Positive	Negative
Actual	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Table 5: Confusion Matrix

potentially useful in terms of phishing detection, which the CoNLL 2003 dataset does not support, such as money, cardinal, date, or time (Table 4). For this reason, only spaCy and flair were taken into consideration.

Evaluation Metrics

Evaluation metrics are required to measure a model's accuracy on a dataset. In binary classification tasks, the desired detections consist of a positive and a negative class. Assuming a dataset annotation with gold data for classification has been made, the predictions form a confusion matrix (Table 5).

One of the standard metrics which utilizes the confusion matrix is ***F_1-score*** which is used as the evaluation metric on the OntoNotes v5 dataset. F_1-score can be described as the harmonic mean of *precision* and *recall* (Equation 4.1).

$$F_1\text{-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.1}$$

Precision is calculated as the ratio between the number of correctly classified positive examples in the dataset and the number of samples classified as positive (Equation 4.2) [89].

$$\text{precision} = \frac{TP}{TP + FP} \tag{4.2}$$

On the other hand, recall, also known as the *true positive rate* (TPR), is calculated as the ratio between the number of correctly classified positive examples in the dataset and the total number of actually positive examples (Equation 4.3).

$$\text{recall} = TPR = \frac{TP}{TP + FN} \tag{4.3}$$

Conversely, the *false positive rate* is calculated as the ratio between the dataset's incorrectly classified negative examples and the total number of actually negative examples (Equation 4.4).

$$FPR = \frac{FP}{FP + TN} \tag{4.4}$$

Another standard evaluation metric for binary classification is the **receiver operating characteristic** (ROC) curve used in Section 4.3. The ROC curve is a graph that shows the classification performance of a classifier at all classification thresholds using the true positive rate (TPR, Equation 4.3) and false positive rate (FPR, Equation 4.4) [90].

The ROC curve then visualizes the tradeoff between TPR and FPR. The smaller values on the x-axis express lower false positives and higher true negatives.

Model	Precision	Recall	F_1	Relaxed F_1	OntoNotes F_1
Transition-based, small (spaCy)	48.4%	37.3%	42.2%	58.6%	84%
Transition-based, medium (spaCy)	58%	48.3%	52.7%	66.3%	85%
Transition-based, large (spaCy)	55.4%	45.4%	49.9%	65.7%	85%
Transformer (spaCy)	78.5%	55.4%	64.9%	72.2%	90%
BiLSTM (flair)	60.6%	41.1%	49%	57%	89.3%
Transformer (flair)	58%	53.3%	55.5%	63%	90.9%

Table 6: NER models evaluation metrics comparison

In contrast, larger values on the y-axis imply a higher number of true positives and lower false negatives. Especially the ROC curve is a useful tool to compare curves of different binary classifiers either at various thresholds or on the entire graph.

Another option to compare classifiers is by comparing the model's *area under curve* (AUC), where a perfect classifier touching the upper-left corner would have AUC equal to 1. On the other hand, a completely random guessing classifier would be represented via a diagonal line from the bottom-left to the top-right on a linear scale, therefore would have AUC of 0.5.

Annotation

On OntoNotes v5, NER models taken into consideration have their F_1-score publicly available. Nevertheless, it was suitable to examine whether and how much the models could convert their knowledge to the Enron dataset. Since no annotation for the Enron dataset exists, a minor subset consisting of 200 sentences of the Enron dataset was manually annotated, and the F_1-score on this minor dataset was computed. The results have shown that all models' F_1-score degraded compared to the public results on the OntoNotes v5 dataset, as shown in Table 6.

It was observed that some examples in the minor dataset were declined, although they were close to the correct assessment. Nevertheless, since such assessment could still be helpful in terms of named entity recognition in emails, an own relaxed F_1-score was introduced. In the relaxed F_1-score, a true positive set was extended by examples where a named entity is only partially found compared to the annotated string, but the entity type is correct. Furthermore, the examples where the entity type was not assessed correctly but the string captured exactly matched the annotation. Compared to the standard F_1-score, each model, when measured with the relaxed F_1-score, managed to gain at least 7% of predictive performance.

Performance

A benchmark on a minor amount of randomly sampled sentences from the Enron email dataset with an identical seed was performed to determine how fast each model runs.

Considering that the model is loaded and the dataset with 10 000 sentences has been prepared in advance, the benchmark consists of sentence tokenization via Sentencizer and named entity recognition on a sentence level (Table 7). The

Model	Runtime	Speedup against Transformer (spaCy)
Transition-based, small (spaCy)	23.34 s	38x
Transition-based, medium (spaCy)	26.06 s	34x
Transition-based, large (spaCy)	26.07 s	34x
Transformer (spaCy)	885.22 s	1x
BiLSTM (flair)	344.53 s	2.5x
Transformer (flair)	268.87 s	3.3x

Table 7: NER models runtime comparison on 10 000 sentences from the Enron email dataset

measured runtimes are an average from three different datasets containing 10 000 randomly sampled sentences from the Enron email dataset. The experiment was computed on a device with AMD Ryzen 7 4800H, 16GB RAM, Nvidia RTX 2060 6GB, transition-based and BiLSTM models utilized CPU, whereas transformer-based models ran on GPU.

4.2 Proprietary Dataset

The proprietary dataset provided by Cisco contains 50 000 positive and 53 059 negative emails encountered from the 1st to the 31st of March 2022. Furthermore, it contains 50 000 positive and 37 225 negative emails collected from 1st April to 26th April 2022. In this dataset, positive emails are a mixture of spam, phishing, and malicious messages. On the other hand, negative emails are considered emails that the customers reported but, according to prior knowledge or manual analysis, proved to be harmless, such as normal conversation, notifications, or marketing.

In the Enron dataset experiment, the output produced consists of found entities and models comparison. However, most importantly, it shows for each model and its entity types found how many times an entity occurred in a sentence in JSON format (Figure 20).

Hence the number of sentences was known in advance, and the number of sentences where no such entity was found can be computed; per-entity probability distributions can be considered. In the data, it was observed that the differences in distributions between datasets could be seen.

The measurements were made on the March and April dataset to exclude other factors from the difference and to deny better that this situation occurred only by chance. Moreover, the probability distributions were compared using the Kullback-Leibler (KL) and the Jensen-Shannon (JS) divergence [91].

Assuming P and Q are discrete probability distributions well defined on the probability space X, the Kullback-Leibler divergence is defined as

$$\mathrm{KL}(P \mid\mid Q) = \sum_{x \in X} P(x) \frac{log(P(x))}{log(Q(x))}. \tag{4.5}$$

The KL divergence score tells how much information is lost when the second distribution approximates the first one. If the logarithm is of base two, the unit of information is called bits, with the base of the Euler number nats. Usually, the KL divergence score is not symmetrical, and the information lost from the distribution

```
{
    "en_core_web_trf": {
        "CARDINAL": {
            "1": 25826,
            "2": 5880,
            "3": 1078,
            "4": 390,
            "5": 203,
            "6": 25,
            "7": 40,
            "8": 50,
            "9": 7,
            "10": 54,
            "11": 3,
            "12": 9,
            "13": 2,
            "14": 2,
            "15": 2,
            "16": 1,
            "17": 1,
            "19": 2,
            "31": 4,
            "34": 1,
            "35": 2,
            "38": 1,
            "39": 2,
            "55": 4
        },
        "DATE": {
            "1": 46434,
            "2": 8065,
            "3": 2344,
            //... the rest of the occurrences
        },
        "EVENT": {
            //... occurrences
        },
        //... the rest of the named entities

        "sentences" : 416860
    }
    //... other models
}
```

Figure 20: Named entity occurrences JSON snippet ("1": 25826 denotes that the *CARDINAL* entity was found once in a sentence in 25 826 cases out of 416 860)

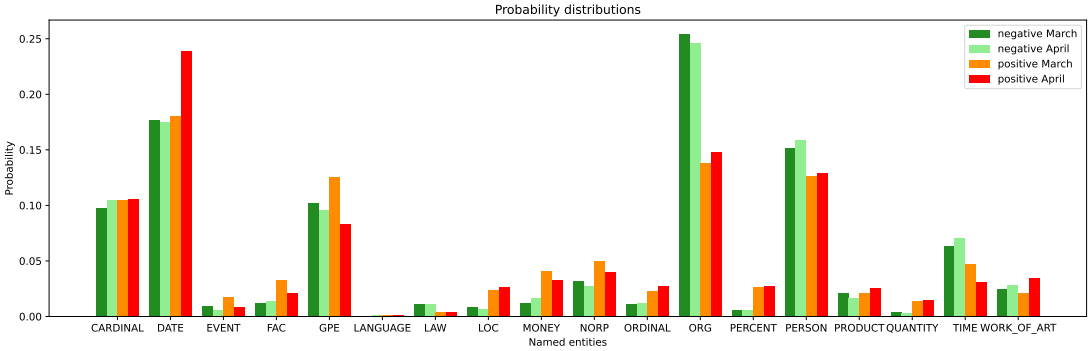


Figure 21: Probability distributions of proprietary dataset components

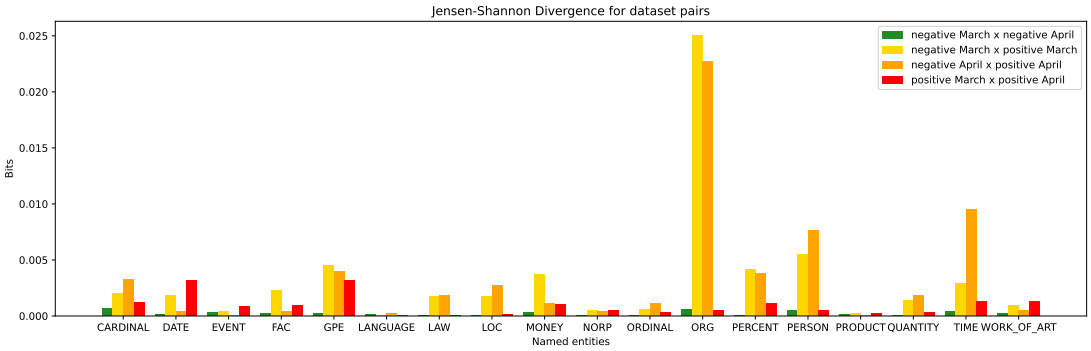


Figure 22: Jensen-Shannon divergence of proprietary dataset components

P to the Q typically differs when considering the information lost from Q to P. This issue is targeted by the Jensen-Shannon divergence (Equation 4.6).

$$\mathrm{JS}(P \mid\mid Q) = \frac{1}{2}\mathrm{KL}(P \mid\mid M) + \frac{1}{2}\mathrm{KL}(Q \mid\mid M). \tag{4.6}$$

$$M = \frac{1}{2}(P + Q) \tag{4.7}$$

The JS divergence directly uses the KL divergence and calculates a normalized symmetrical score which is observed per entity. As mentioned earlier, the computation is derived from probability distributions (Figure 21). With per-entity Jensen-Shannon divergence (Figure 22), the positive emails differ from the negative emails by a large margin regarding organizations, followed by person and time entity types abundant in both positive and negative emails. The rest of the entity types did not show that remarkable results. Despite that, on average, positive versus negative proved to be least similar, according to their per-entity JS divergences.

When comparing negative emails from March and April, this distribution pair proved most alike regarding per-entity JS divergences. Positive email distributions compared against each other were less alike due to the higher potential to be more diverse. For a more thorough analysis, pairwise bar plots are included in Section A.1.

4.3 Phishing Email Classification

Before moving to the contribution of named entity recognition for phishing detection, it is suitable to describe the phishing email classification process (Figure 23). One of the phishing detection techniques utilized by Cisco is conducted using the *Cognitive Anti-Phishing Engine*, containing various detectors for phishing. The engine produces the output as JSON containing detections found in an email. The experiment regarding named entity recognition then compares whether named entities found in the email can benefit the phishing email classifier's predictive performance. Initially, the experiment ran without using named entity recognition. Afterward, named entity recognition is utilized, producing a JSON output containing named entities found on a sentence level. JSON provided is embedded using *JSON2Bag*. Consequently, a classifier based on multiple instance learning (*Instance selection randomized trees* [92]) is used to utilize features from JSON2Bag for phishing email classification.

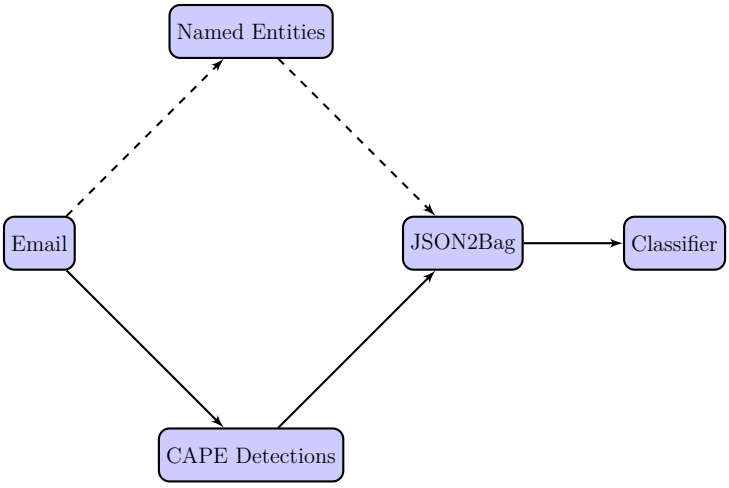


Figure 23: Proposed phishing email classification workflow

4.3.1 Cognitive Anti-Phishing Engine

Cognitive Anti-Phishing Engine (CAPE) is a phishing detection engine developed by Cisco Cognitive Intelligence Department. CAPE is designed as an ensemble of detectors. Initially, CAPE analyzes email headers and text content and watches for potentially malicious signals. CAPE currently supports over 30 distinct heterogeneous detectors, such as call-to-action, impersonation, or credentials phishing detectors. The detectors are then combined using a proprietary machine learning model, which, based on the detections, comes with a verdict on whether the email should be classified as phishing. In production, CAPE is deployed as a REST API server whose response is a JSON object. Detections may contain metadata with additional information, such as the email segment due to which the detection was triggered or information from email headers (Figure 24).

```
{
    "detections": [
        {
            // Link Masquerade Detector
            "code": "DLINKMASQ",
            "meta": [
                {
                    "id": "displayed",
                    "val": "https://google.com"
                },
                {
                    "id": "real",
                    "val": "https://malicious.com"
                }
            ],
            "score": 1.0
        },
        {
            // Cryptocurrency Address Detector
            "code": "DCRYPTOCCY",
            "meta": [
                {
                    "id": "address",
                    "val": "<anonymized address>"
                },
                {
                    "id": "cointype",
                    "val": "bitcoin"
                }
            ],
            "score": 1.0
        },
        {
            // Call-To-Action Detector (Suspicious Request)
            "code": "DREQUEST_CCT",
            "meta": [
                {
                    "id": "segment",
                    "val": "Send 5 BTC to this address:
                    <anonymized address>."
                }
            ],
            "score": 1.0
        }
    ]
}
```


Figure 24: CAPE detections serialized into JSON format from an email

4.3.2 JSON2Bag

JSON2Bag is, at the time of writing the thesis unpublished, a feature extraction algorithm that can transform arbitrary JSON data into a bag of numeric vectors using a set of general feature extractors. Initially, it flattens the structured JSON data into a set of path and value pairs. Each path-value pair is then converted into a single feature vector using two types of feature extraction functions. Firstly, there are path feature extractors, such as the path length, the number of fields in the path, or the number of arrays in the path. Secondly, value feature extractors are considered, such as whether the value is of a given type (null, bool, number, or string). If it is a string, its one-hot encoded representation and its hashed-embedded representation are considered (Figure 25).

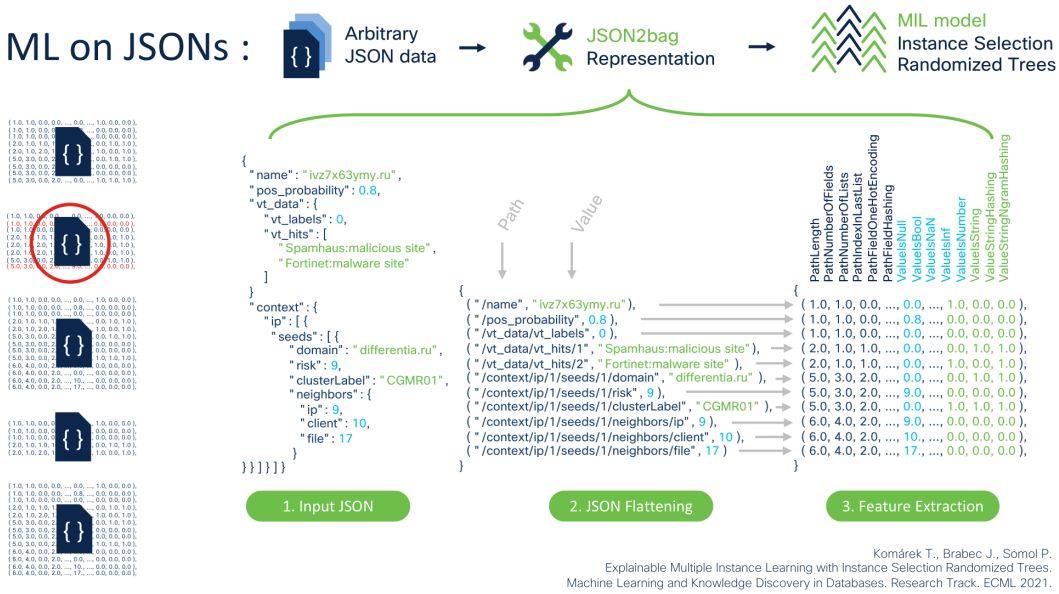


Figure 25: JSON2Bag feature extraction algorithm

4.3.3 Named Entity Recognition Contribution

The experiment involving named entity recognition examines whether named entities found on a sentence level serialized to JSON format can help the phishing email classifier in predictive performance. For this experiment, the proprietary dataset containing positive and negative emails mentioned earlier was utilized for phishing detection. Initially, phishing detection was measured for March and April data separately. The first model undergoes the phishing detection procedure mentioned earlier.

Whereas the second model is the identic model with the difference, it is given additional JSON containing named entities on a sentence level (Figure 26) produced by the spaCy transformer from which other feature vectors are formed. Consequently, the generalization abilities of the phishing email classifier with and without named entities are examined using training data from March and testing data from April.

```
{
    "named_entities": [
        [
            {
                "type": "PERSON",
                "text": "Hazel"
            }
        ],
        [
            {
                "type": "MONEY",
                "text": "27,460"
            }
        ],
        [],
        [
            {
                "type": "DATE",
                "text": "daily"
            }
        ],
        [
            {
                "type": "TIME",
                "text": "the next 12 hours"
            },
            {
                "type": "PERSON",
                "text": "James Smith"
            },
            {
                "type": "GPE",
                "text": "Switzerland"
            }
        ]
    ]
}
```


Figure 26: Named entities serialized into JSON from an email where each pair of square brackets denotes a sentence in the email containing named entities found in the sentence

Monthly Comparison

When comparing predictive performance in March 2022 (Figure 27) and April 2022 (Figure 28) separately, it can be observed that named entities improve the model's predictive performance. The AUC when using NER is visibly more significant than without them in both months. Furthermore, as shown in Table 8, in the ROC curve at critical thresholds, TPR gained at least 7%. In general, the named entities helped predictive performance at all levels of the ROC curve.

The experiment uses *k-fold cross-validation* to reduce overfitting impact, meaning that the model finds patterns that are not generally useful but only for the dataset given. In the k-fold cross-validation, the dataset is randomly shuffled and split into k complementary subsets (folds), where $k-1$ subsets of the dataset are used for training, and the last k^{th} of the dataset becomes a testing (validation) set. The process is repeated until every fold becomes the testing set. Afterward, the average of recorded scores is used. The experiment used 3-fold cross-validation. Since the dataset was shuffled randomly prior, the process was repeated five times (5-round 3-fold cross-validation) to reduce the overfitting impact further [93].

TPR x FPR	at 10^{-3}	at 10^{-2}	at 10^{-1}
Classifier - March	73.74%	84.53%	91.24%
Classifier with NER - March	82.57%	92.32%	98.77%
Classifier - April	67.82%	80.83%	88.09%
Classifier with NER - April	74.46%	90.82%	95.56%

Table 8: Monthly comparison of phishing email classifier with and without named entities at critical thresholds

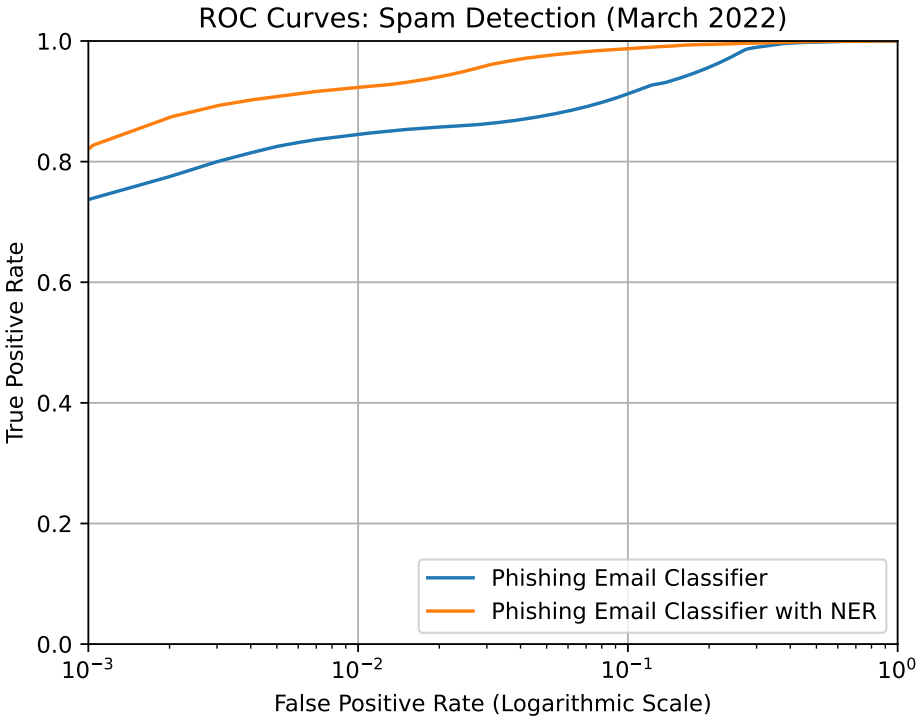


Figure 27: Phishing detection ROC curves from March comparison between the phishing email classifiers, ROC curves as interpolations of 5-round 3-fold cross-validation

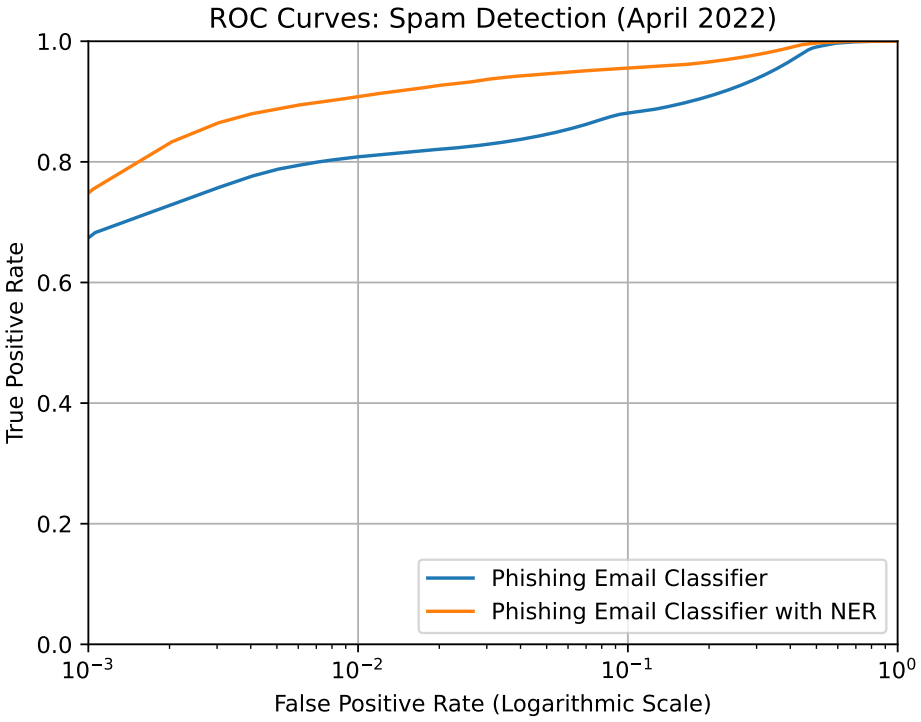


Figure 28: Phishing detection ROC curves from April comparison between the phishing email classifiers, ROC curves as interpolations of 5-round 3-fold cross-validation

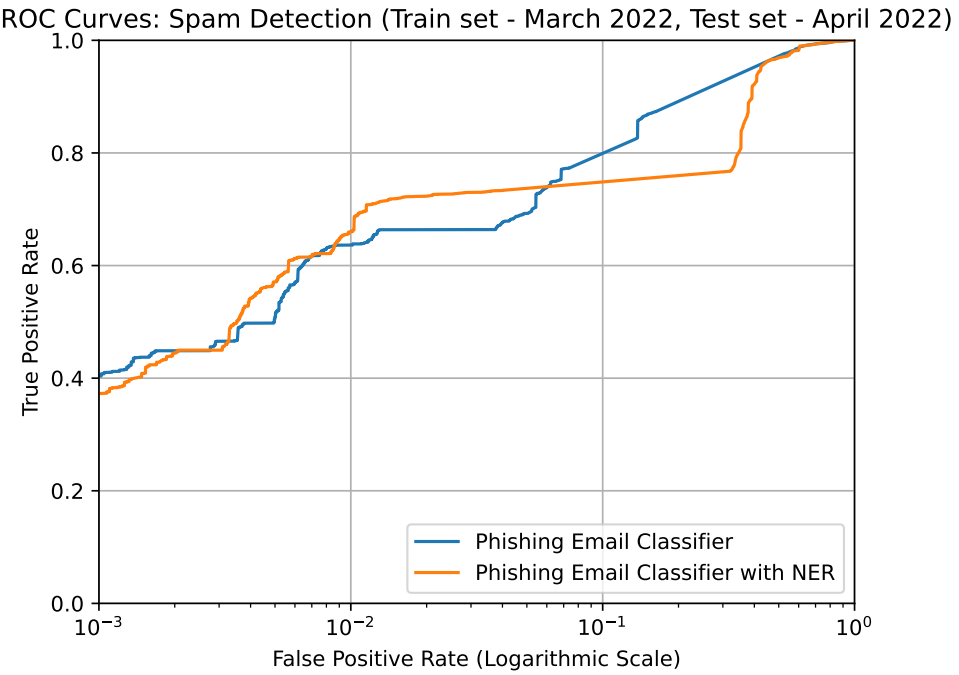


Figure 29: Phishing detection ROC curves comparison between the email phishing classifiers with the training dataset from March applied to the testing dataset from April

Generalization Comparison

Afterward, the model was examined from another perspective. It was given the dataset from March 2022 to train with and the testing dataset from April to be tested to ensure the model generalizes well. As shown in Figure 29, named entities did not prove beneficial since the model lost the true positive rate by a small margin in the leftmost part of the ROC curve. Concrete values at critical thresholds are shown in Table 9. Moreover, the difference in predictive performance is not convincing at the interval of practical use of the ROC curve $[10^{-3}, 10^{-2}]$ either. A time-based train-test split showed that the model still learned some patterns which were not transferable and, therefore, not useful for generalization.

One of the possible reasons overfitting was still observed is that the differences in emails in these two months were way too significant, that the model did not have the opportunity to encounter such emails in its training set. Therefore it could not utilize such knowledge on the testing set. As an option to potentially improve predictive performance on unseen data, the model should encounter a wider variety of emails to train with, ideally from a broader time horizon.

TPR x FPR	at 10^{-3}	at 10^{-2}	at 10^{-1}
Classifier	40.38%	63.63%	79.51%
Classifier with NER	37.25%	66%	74.08%

Table 9: Comparison of phishing email classifier with and without named entities using March dataset for training, April dataset for testing

Conclusion

This bachelor thesis described various named entity recognition approaches regarding phishing detection. Initially, phishing and its deceiving techniques conducted over multiple media types were introduced. Afterward, named entity recognition was presented. Since named entity recognition is a natural language processing task concerned with text extraction, various text representation techniques were described. As the named entity recognition task is mainly approached by neural networks, many concepts regarding neural networks were described from the basics. The thesis then gradually proceeded to the current state-of-the-art named entity recognition models.

Since many libraries with named entity recognition models already exist, and their models are publicly available, model architectures description and thorough examination were conducted before proceeding to their execution. The model choice was based on various aspects, such as predictive performance (F_1-score), runtime specifics, and the dataset on which the model was supposed to be fine-tuned. It was shown that the preferred dataset would be OntoNotes v5 over CoNLL 2003 due to its wider variety of named entity types. Furthermore, from the predictive performance perspective, the best model was the transformer-based model included in the spaCy library. Nevertheless, its runtime performance was not satisfying. For this reason, an alternative transition-based medium model included in spaCy with 34x faster runtime with the second-best F_1-score was also considered.

As the model choice was left for these two, an examination of the differences between positive and negative emails provided by a proprietary dataset by Cisco followed. Initially, named entity occurrences on a sentence level were computed, and a sole comparison of named entity probability distributions was made. Nevertheless, such an approach did not seem to be convincing enough. One of the methods to compare probability distributions is to compare the distributions via Kullback-Leibler divergence, respectively, using its symmetrized normalized variant called Jensen-Shannon divergence, which resulted in more effectiveness.

With per-entity Jensen-Shannon divergence, positive emails differed from the negative emails by a large margin regarding the organization, person, and time entity types which, according to the probability distributions, were abundant in both positive and negative emails. Other entity types did not provide too noticeable differences. Despite that, on average, JS divergences have shown that positive versus negative email distribution pairs proved to be least similar. Conversely, negative email distributions compared against each other showed most similar results regarding per-entity JS divergences. The positive email distribution pair was less alike than the negative distribution pair mainly due to the higher potential to be more diverse.

In the last part of the chapter regarding conducted experiments, named entities were examined to determine whether they can improve predictive performance for phishing detection. Afterward, the phishing email classification workflow utilized in this experiment was described. The provided phishing email detection engine used various detectors which examined the email content

and its header, producing a JSON output that was input into a multiple instance learning model, obtaining features via the JSON2Bag feature extraction algorithm.

A comparison was made whether named entities found in the email on a sentence level serialized to JSON and passed as extra information added to the phishing email classifier can prove beneficial. The classifiers differed in passed JSON data where the first model utilized serialized CAPE detections, and the second one utilized the CAPE detections and additional named entities on a sentence level.

Initially, predictive performance comparison was measured on March and April 2022 datasets separately. It was shown that true positive rates gained at least 7% at critical thresholds of the ROC curve. Moreover, named entities visibly helped predictive performance at all levels of the ROC curve. Afterward, the phishing email classifiers' generalization ability was examined via a time-based train-test split approach where training data were used from March 2022, and testing was conducted on data from April 2022.

The difference in predictive performance on the interval of practical use of the ROC curve was not convincing enough to conclude that additional named entities provide persuasive benefit. According to the time-based split, it was shown that during monthly comparison, the classifier with NER provided better results by learning patterns that were not useful for generalization.

On the other hand, during the time-based split, it was discussed that differences in emails from these two months may have been way too large that the classifier did not have the opportunity to utilize the knowledge on a testing set. An option to potentially improve predictive performance on unseen data would be for the classifier to encounter a wider variety of emails during training, ideally from a broader time horizon. Assuming a different email dataset with more representative training data was used, it could happen that different conclusions would have been drawn.

Bibliography

[1] Zainab Alkhalil, Chaminda Hewage, Liqaa Nawaf, and Imtiaz Khan. Phishing attacks: A recent comprehensive study and a new anatomy. *Frontiers in Computer Science*, 3:563060, 2021.

[2] Federal Bureau of Investigation. Internet Crime Report 2021. `https://www.ic3.gov/Media/PDF/AnnualReport/2021_IC3Report.pdf`, 2021. Accessed: 15/07/2022.

[3] MITRE ATT&CK (MITRE Adversarial Tactics, Techniques, and Common Knowledge). Initial Access. `https://attack.mitre.org/tactics/TA0001/`, 2019. Accessed: 25/05/2022.

[4] Verizon. Data Breach Investigations Report 2020. `https://itb.dk/wp-content/uploads/2020/07/verizon-data-breach-investigations-report-2020.pdf`, 2020. Accessed: 25/05/2022.

[5] Said Salloum, Tarek Gaber, Sunil Vadera, and Khaled Shaalan. Phishing email detection using natural language processing techniques: A literature survey. *Procedia Computer Science*, 189:19–28, 2021. AI in Computational Linguistics.

[6] María Fernanda Cazares, Roberto Andrade, Gustavo Navas, Walter Fuertes, and Jhonathan Herrera. Characterizing phishing attacks using natural language processing. In *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, pages 224–229, 2021.

[7] Cornell University. Phish Bowl. `https://it.cornell.edu/phish/9465`, 2022. Accessed: 15/07/2022.

[8] KnowBe4. Phishing Attacks. `https://www.knowbe4.com/phishing`, 2021. Accessed: 30/06/2022.

[9] Check Point. What is Phishing? `https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-phishing/`, 2020. Accessed: 30/06/2022.

[10] Cornell University. Phish Bowl. `https://it.cornell.edu/phish/6457`, 2018. Accessed: 15/07/2022.

[11] The PhishLabs Team. How Spear Phishing Makes BEC Attacks So Effective. `https://www.phishlabs.com/blog/how-spear-phishing-makes-bec-attacks-so-effective/`, 2019. Accessed: 30/06/2022.

[12] Jetli Chung, Jing-Zhi Koay, and Yu-Beng Leau. A review on social media phishing: Factors and countermeasures. In Mohammed Anbar, Nibras Abdullah, and Selvakumar Manickam, editors, *Advances in Cyber Security*, pages 657–673, Singapore, 2021. Springer Singapore.

[13] Tom N. Jagatic, Nathaniel A. Johnson, Markus Jakobsson, and Filippo Menczer. Social phishing. *Commun. ACM*, 50(10):94–100, oct 2007.

[14] CNBC. Twenty years after epic bankruptcy, Enron leaves a complex legacy. https://www.cnbc.com/2021/12/02/twenty-years-after-epic-bankruptcy-enron-leaves-a-complex-legacy.html, 2021. Accessed: 29/05/2022.

[15] spaCy. Industrial-Strength Natural Language Processing. https://spacy.io, 2016. Accessed: 30/05/2022.

[16] Asif Ekbal and Sivaji Bandyopadhyay. Named entity recognition using support vector machine: A language independent approach. *International Journal of Electrical and Computer Engineering*, 4(3):589 – 604, 2010.

[17] Nita Patil, Ajay Patil, and B.V. Pawar. Named entity recognition using conditional random fields. *Procedia Computer Science*, 167:1181–1188, 2020. International Conference on Computational Intelligence and Data Science.

[18] IBM Cloud Education. What is machine learning? https://www.ibm.com/cloud/learn/machine-learning, 2020. Accessed: 14/07/2022.

[19] Microsoft. Data featurization in automated machine learning. https://docs.microsoft.com/en-us/azure/machine-learning/how-to-configure-auto-features, 2022. Accessed: 14/07/2022.

[20] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30, 08 2007.

[21] Vikas Yadav and Steven Bethard. A survey on recent advances in named entity recognition from deep learning models. *CoRR*, abs/1910.11470, 2019.

[22] Meta AI. Named Entity Recognition. https://paperswithcode.com/task/named-entity-recognition-ner, 2022. Accessed: 25/05/2022.

[23] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[24] Munish Puri, Aum Solanki, Timothy Padawer, Srinivas M Tipparaju, Wilfrido Alejandro Moreno, and Yashwant Pathak. *Introduction to Artificial Neural Network (ANN) as a Predictive Tool for Drug Design, Discovery, Delivery, and Disposition.* Academic Press,, Amsterdam, [Netherlands] :, 2016.

[25] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[26] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, Berlin, Heidelberg, 2006.

[27] Panagiotis Antoniadis. Activation Functions: Sigmoid vs Tanh. https://www.baeldung.com/cs/sigmoid-vs-tanh-functions, 2022. Accessed: 14/07/2022.

[28] Radek Starosta. Phishing detection using natural language processing. Master's thesis, Czech technical university in Prague, 2021.

[29] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[31] Jason Brownlee. Understand the Impact of Learning Rate on Neural Network Performance. `https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/`, 2019. Accessed: 25/05/2022.

[32] Ritchie Ng, Jie Fu. Learning Rate Scheduling. `https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/lr_scheduling/`, 2019. Accessed: 29/05/2022.

[33] Jeremy Jordan. Setting the learning rate of your neural network., 2018. `https://www.jeremyjordan.me/nn-learning-rate/`.

[34] Jason Brownlee. A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size. `https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/`, 2017. Accessed: 25/05/2022.

[35] Nvidia. Get Started With Deep Learning Performance. `https://docs.nvidia.com/deeplearning/performance/dl-performance-getting-started/index.html`, 2022. Accessed: 07/06/2022.

[36] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[37] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[38] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[39] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[40] Jason Brownlee. A Gentle Introduction to the Bag-of-Words Model. `https://machinelearningmastery.com/gentle-introduction-bag-words-model/`, 2017. Accessed: 15/07/2022.

[41] Amit Mandelbaum and Adi Shalev. Word embeddings and their use in sentence classification tasks. *CoRR*, abs/1610.08229, 2016.

[42] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Computer Science*, 2013.

[43] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[44] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[45] David Rozado. Wide range screening of algorithmic bias in word embedding models using large sentiment lexicons reveals underreported bias types. *PLOS ONE*, 15:e0231189, 04 2020.

[46] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[47] Lei Mao. Word2Vec Models Revisited. `https://leimao.github.io/article/Word2Vec-Classic/`, 2019. Accessed: 29/05/2022.

[48] Jeffrey Pennington,Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation. `https://nlp.stanford.edu/projects/glove/`, 2014. Accessed: 15/07/2022.

[49] Nvidia. Deep Learning. `https://developer.nvidia.com/deep-learning`, 2022. Accessed: 16/07/2022.

[50] Md. Zahangir Alom, Tarek Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Nasrin, Mahmudul Hasan, Brian Essen, Abdul Awwal, and Vijayan Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8:292, 03 2019.

[51] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 08 2015. Accessed: 2022-04-20.

[52] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[53] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[54] Jitendra Tembhurne and Tausif Diwan. Sentiment analysis in textual, visual and multimodal inputs using recurrent neural networks. *Multimedia Tools and Applications*, 80:1–40, 02 2021.

[55] Savvas Varsamopoulos, Koen Bertels, and Carmen Garcia Almudever. Designing neural network based decoders for surface codes. 2018.

[56] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[57] Christopher Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. Accessed: 30/05/2022.

[58] Devopedia. Bidirectional RNN. https://devopedia.org/bidirectional-rnn, 2020. Accessed: 27/05/2022.

[59] Jason P. C. Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *CoRR*, abs/1511.08308, 2015.

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[61] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[62] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[63] Jay Alammar. The illustrated transformer. https://jalammar.github.io/illustrated-transformer/, 06 2018. Accessed: 2022-05-08.

[64] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[65] Raman Kumar. All you need to know about bert. https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-bert/, 05 2021. Accessed: 2022-05-13.

[66] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[67] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *CoRR*, cs.CL/0306050, 2003.

[68] Sameer Pradhan Ralph Weischedel, Lance Ramshaw, Jeff Kaufman, Michelle Franchini, and Mohammed El-Bachouti. Ontonotes release 5.0. https://catalog.ldc.upenn.edu/docs/LDC2013T19/OntoNotes-Release-5.0.pdf, 09 2012. Accessed: 2022-05-13.

[69] spaCy. Linguistic Features. https://spacy.io/usage/linguistic-features, 2022. Accessed: 31/05/2022.

[70] Matthew Honnibal. SPACY'S ENTITY RECOGNITION MODEL: incremental parsing with Bloom embeddings and residual CNNs. https://www.youtube.com/watch?v=sqDHBH9IjRU, 2017. Accessed: 26/06/2022.

[71] Matthew Honnibal, Adriane Boyd, Vincent D. Warmerdam. Compact word vectors with Bloom embeddings. https://explosion.ai/blog/bloom-embeddings, 2022. Accessed: 31/05/2022.

[72] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *CoRR*, abs/1603.01360, 2016.

[73] spaCy. Available trained pipelines for English. https://spacy.io/models/en, 2022. Accessed: 31/05/2022.

[74] spaCy. What's New in v3.0. https://spacy.io/usage/v3, 2021. Accessed: 06/06/2022.

[75] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[76] Hugging Face. Models. https://huggingface.co/models, 2020. Accessed: 01/06/2022.

[77] Hugging Face. Transformers. https://huggingface.co/docs/transformers/main/en/index, 2019. Accessed: 03/07/2022.

[78] David S. Lim. bert-base-NER. https://huggingface.co/dslim/bert-base-NER, 2020. Accessed: 01/06/2022.

[79] David S. Lim. bert-large-NER. https://huggingface.co/dslim/bert-large-NER, 2020. Accessed: 01/06/2022.

[80] Zuhaib Akhtar. BERT base vs BERT large. https://iq.opengenus.org/bert-base-vs-bert-large/, 2018. Accessed: 01/06/2022.

[81] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. Flair: An easy-to-use framework for state-of-the-art nlp. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, 2019.

[82] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

[83] Meta AI. Named Entity Recognition on CoNLL 2003 (English). https://paperswithcode.com/sota/named-entity-recognition-ner-on-conll-2003, 2021. Accessed: 25/05/2022.

[84] Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. CCNet: Extracting high quality monolingual datasets from web crawl

data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4003–4012, Marseille, France, May 2020. European Language Resources Association.

[85] Stefan Schweter and Alan Akbik. FLERT: document-level features for named entity recognition. *CoRR*, abs/2011.06993, 2020.

[86] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *Proceedings of the 15th European Conference on Machine Learning*, ECML'04, page 217–226, Berlin, Heidelberg, 2004. Springer-Verlag.

[87] spaCy. Sentencizer. `https://spacy.io/api/sentencizer`, 2022. Accessed: 16/07/2022.

[88] spaCy. SentenceRecognizer. `https://spacy.io/api/sentencerecognizer`, 2022. Accessed: 16/07/2022.

[89] Ahmed Fawzy Gad. Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall. `https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/`. Accessed: 04/07/2022.

[90] Jason Brownlee. How to Use ROC Curves and Precision-Recall Curves for Classification in Python. `https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/`, 2018. Accessed: 09/07/2022.

[91] Jason Brownlee. How to Calculate the KL Divergence for Machine Learning. `https://machinelearningmastery.com/divergence-between-probability-distributions//`, 2019. Accessed: 26/06/2022.

[92] Tomáš Komárek, Jan Brabec, and Petr Somol. Explainable multiple instance learning with instance selection randomized trees. In Nuria Oliver, Fernando Pérez-Cruz, Stefan Kramer, Jesse Read, and Jose A. Lozano, editors, *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 715–730, Cham, 2021. Springer International Publishing.

[93] Jason Brownlee. A Gentle Introduction to k-fold Cross-Validation. `https://machinelearningmastery.com/k-fold-cross-validation/`, 2018. Accessed: 08/07/2022.

List of Figures

List of Tables

List of Terms and Acronyms

Terms

negative emails Emails that the customers reported but, according to prior knowledge or manual analysis, proved to be harmless, such as normal conversation, notifications, or marketing.

positive emails Emails that are a mixture of spam, phishing, and malicious messages.

Acronyms

API Application Programming Interface

AUC Area Under (ROC) Curve

BERT Bidirectional Encoder Representations from Transformers

CAPE Cognitive Anti-Phishing Engine

CoNLL Conference on Computational Natural Language Learning

FPR False Positive Rate

GloVe Global Vectors

GRU Gated Recurrent Unit

JS Jensen-Shannon (divergence)

JSON JavaScript Object Notation

KL Kullback-Leibler (divergence)

LSTM Long Short-Term Memory

MLP Multilayer Perceptron

NER Named Entity Recognition

NLP Natural Language Processing

ReLU Rectified Linear Unit

REST Representational State Transfer

RNN Recurrent Neural Network

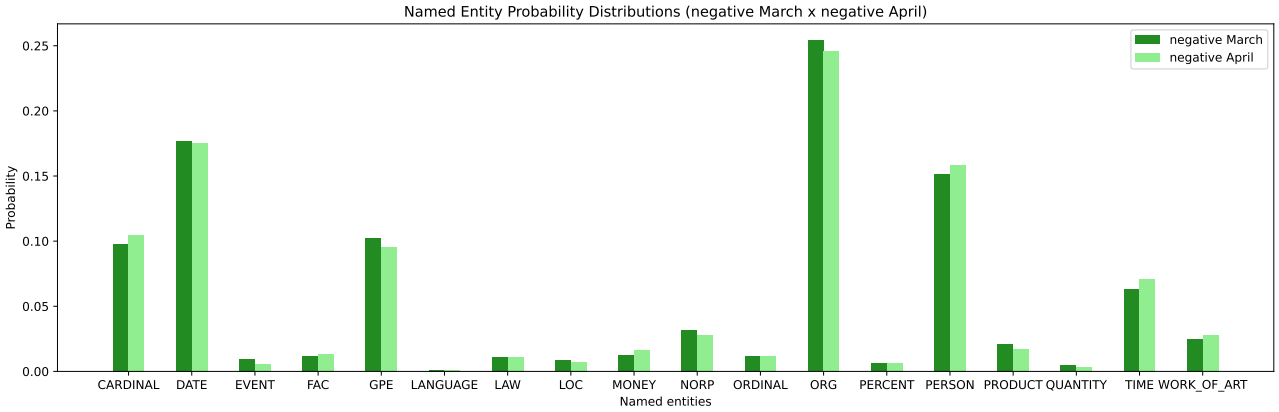
ROC Receiver Operating Characteristic

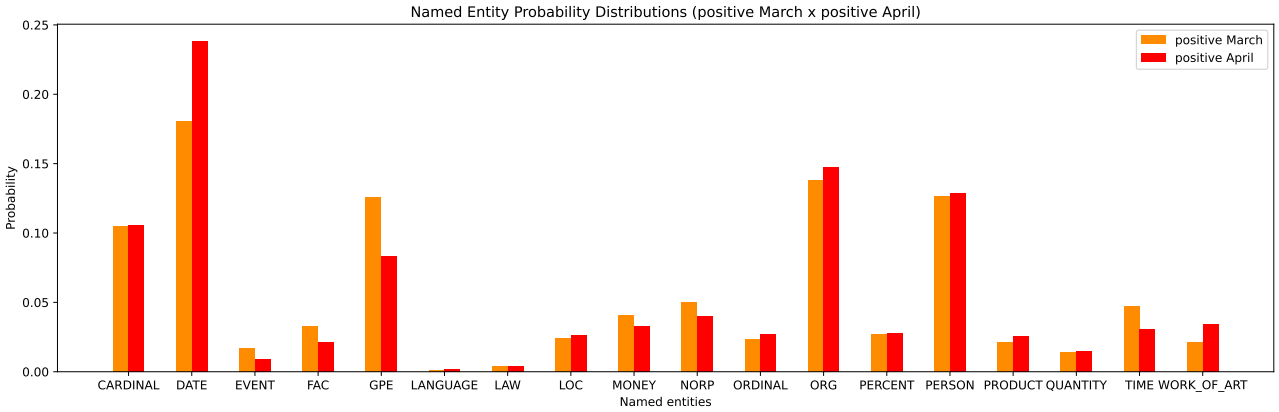
TPR True Positive Rate

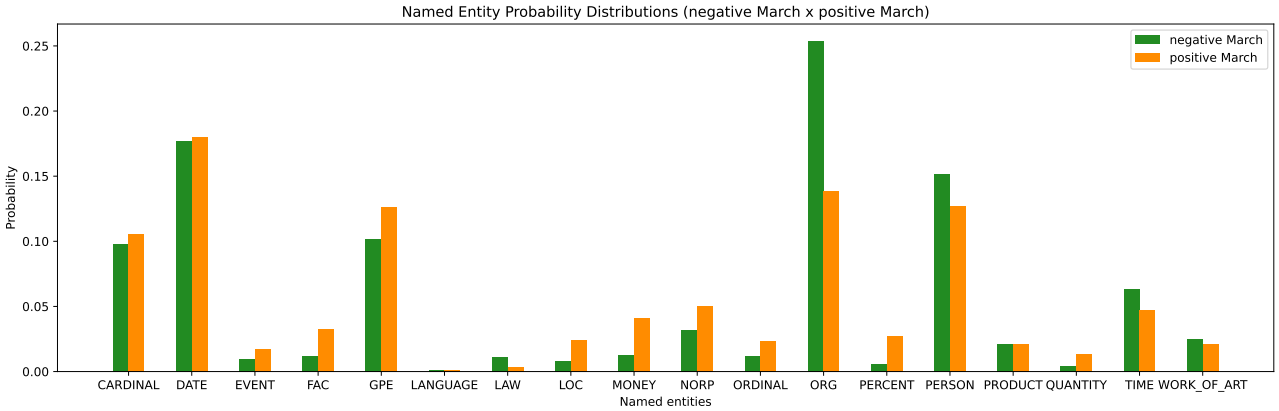
A. Attachments

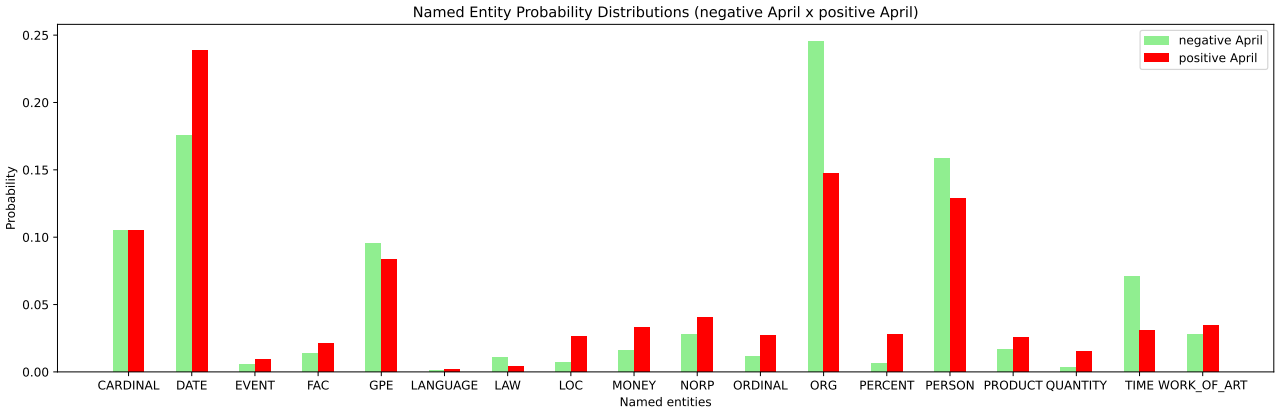
A.1 Proprietary Dataset Experiment

A.1.1 Probability Distributions

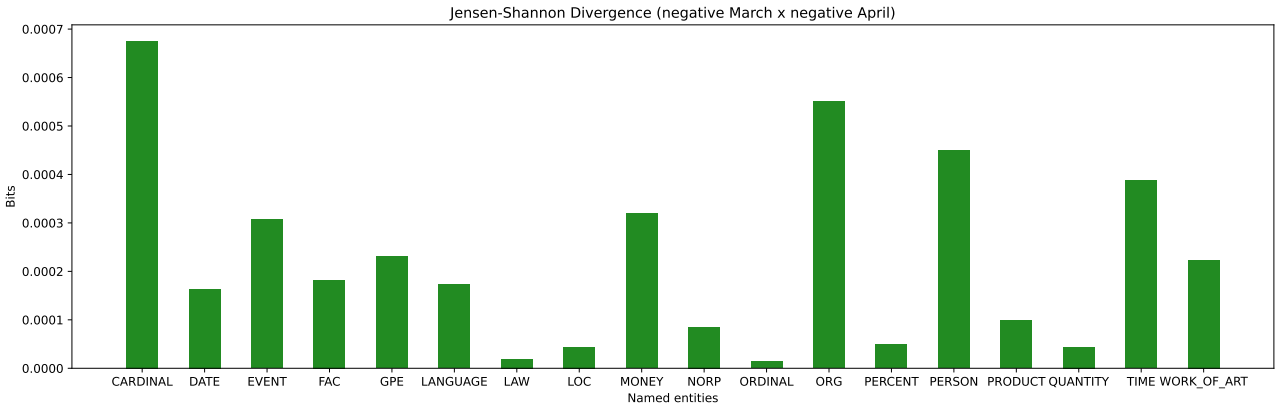


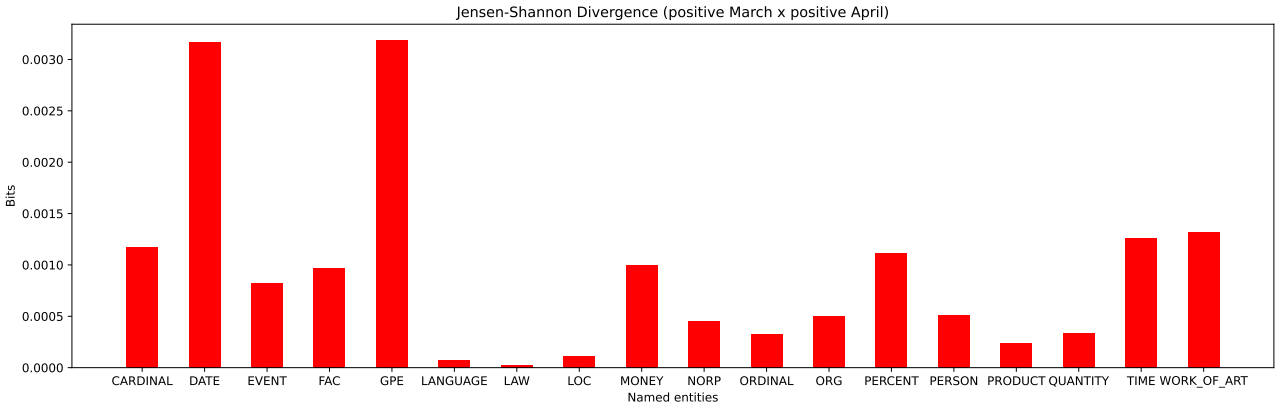


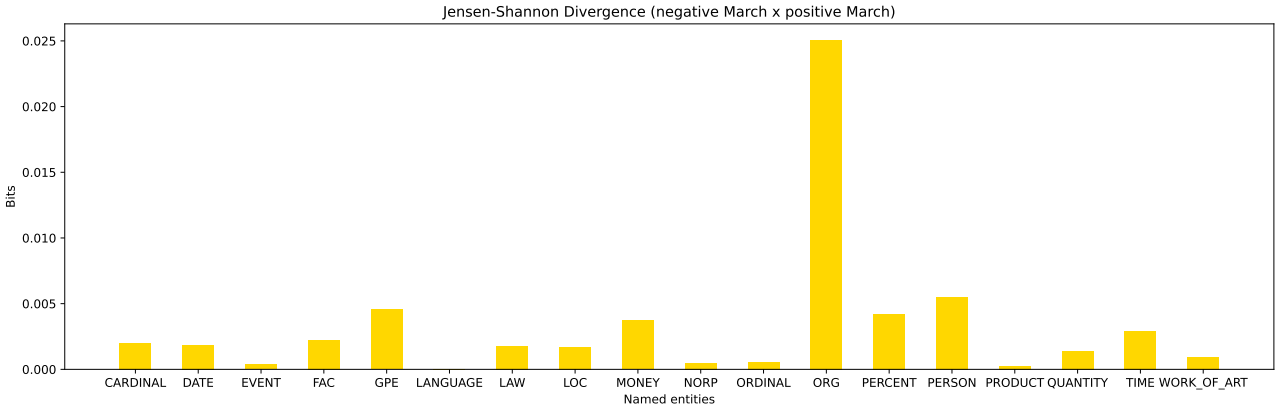

Named Entity Probability Distributions (negative March x positive March)
negative March
positive March
Probability
0.25
0.20
0.15
0.10
0.05
0.00
CARDINAL
DATE
EVENT
FAC
GPE
LANGUAGE
LAW
LOC
MONEY
NORP
ORDINAL
ORG
PERCENT
PERSON
PRODUCT
QUANTITY
TIME
WORK_OF_ART
Named entities

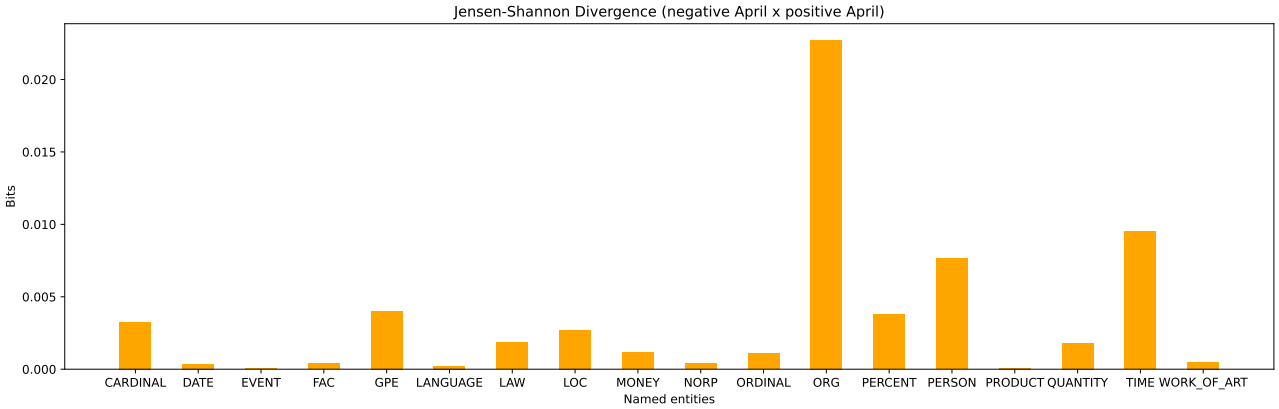

Named Entity Probability Distributions (negative April x positive April)
negative April
positive April
Probability
0.25
0.20
0.15
0.10
0.05
0.00
CARDINAL
DATE
EVENT
FAC
GPE
LANGUAGE
LAW
LOC
MONEY
NORP
ORDINAL
ORG
PERCENT
PERSON
PRODUCT
QUANTITY
TIME
WORK_OF_ART
Named entities

A.1.2 Jensen-Shannon Divergence






Jensen-Shannon Divergence (negative March x positive March)
Bits
0.025
0.020
0.015
0.010
0.005
0.000
CARDINAL
DATE
EVENT
FAC
GPE
LANGUAGE
LAW
LOC
MONEY
NORP
ORDINAL
ORG
PERCENT
PERSON
PRODUCT
QUANTITY
TIME
WORK_OF_ART
Named entities


Jensen-Shannon Divergence (negative April x positive April)
Bits
0.020
0.015
0.010
0.005
0.000
CARDINAL
DATE
EVENT
FAC
GPE
LANGUAGE
LAW
LOC
MONEY
NORP
ORDINAL
ORG
PERCENT
PERSON
PRODUCT
QUANTITY
TIME
WORK_OF_ART
Named entities