Aalto University
School of Business

A BUSINESS NEWS EVENT DETECTION ALGORITHM WITH AN APPLICATION TO THE FOREST INDUSTRY

Master's Thesis
Khang Nguyen
Aalto University School of Business
Information Service Management
Fall 2020

Aalto University
School of Business

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of master's thesis

Author Khang Nguyen

Title of thesis A BUSINESS NEWS EVENT DETECTION ALGORITHM WITH AN APPLICATION TO THE FOREST INDUSTRY

Degree Master of Science in Economics and Business Administration

Degree programme lnformation Service Management

Thesis advisor(s) Pekka Malo

Year of approval 2021 **Number of pages** 85 **Language** English

Abstract

The forest industry is an important industry that generates billions of euros and employs millions of workers. However, it lacks a particular type of business intelligence enjoyed by other industries, namely the extraction of knowledge from online articles. Despite many studies on this subject, no relevant study exists for the forestry industry due to the lack of a usable dataset.

This thesis proposes an event detection algorithm for online articles that can be applied to both general business news and forest industry news. To that end, three research questions are examined. Firstly, the creation of a robust dataset that is inclusive of forest industry news. Secondly, establishing the feasibility of building an event detection algorithm to recognize and classify both general business and forest industry news. Lastly, proposing an optimally performing model for the said algorithm.

To build an event detection algorithm, machine learning methods, particularly natural language processing, are used. The proposed solution comprises contextualized word embeddings and a classification model. Those word embeddings are created with BERT, a state-of-the-art model for text handling from Google. For model performance tuning, one approach is implemented to address the class imbalance problem.

The evaluation shows that the proposed solution delivers a strong result, which indicates promising practical implementations in the forest industry. Companies in the industry should be potentially able to enjoy an aspect of business intelligence that has been employed in other industries.

This thesis is the first to empirically examine the links between online news articles, events detection, and the forest industry. The thesis's contributions are twofold. First, the thesis provides an annotated dataset for use with different machine learning methods. Secondly, it complements literature on the feasibility of an event detection algorithm applicable to both business and forestry industry news.

Keywords forest industry news, business news, natural language processing, machine learning, classification, event detection

Table of Contents

List of Tables

List of Figures

Acronyms and Abbreviations

Acronyms	Expansions
AI	Artificial Intelligence
API	Application Programming Interface
AUCROC	Area Under Curve - Receiver Operating Characteristics
BERT	Bidirectional Encoder Representations from Transformers
CPU	Central Processing Unit
DL	Deep Learning
DR	Dimensionality Reduction
ED	Event Detection
FFNN	Feedforward Neural Network
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
LSTM	Long Short Term Memory
ML	Machine Learning
NLP	Natural Language Processing
NSP	Next Sentence Prediction
Odin	Open Domain INformer
RAM	Random Access Memory
ROC	Receiver Operating Characteristics
RNN	Recurrent Neural Network

Seq2seq	Sequence-to-Sequence
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
TF	TensorFlow
TN	True Negative
TP	True Positive
TPR	True Positive Rate
TPU	Tensor Processing Unit
t-SNE	T-distributed Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection

1 Introduction

1.1 Background and motivation

The billion-dollar forest industry occupies a small niche in the business world, yet it holds particular importance. The forest industry is vital to global employment. According to the International Labour Organization, the sector employs around 13.7 million formal workers globally. Besides providing jobs to millions, this industry also possesses considerable economic value. In 2018, the global export of forest products reached 270 billion USD in value, with an increase of 10% from 2017. The industry's growing contribution to the world economy is well-recognized: "*The real value of the forestry sector's contribution has increased over the last three decades at an average annual rate of 2.5 percent*" (Food and Agriculture Organization of the United Nations, 2010). In Europe, the forest industry generated 55,796 million EUR in 2017 (Eurostat, 2020). In Finland, forests are hailed as the country's most valuable natural resource. The Finnish forest industry accounted for about 20% of the country's export value and 5% of the country's GDP (Ministry of Agriculture and Forestry of Finland, n.d.). For an industry at such scale, the need for involved companies to understand the industry landscape and competitors is real.

To understand their industries, companies need to collect and analyze information on the behaviors of various actors in the market. This information may include competitors' activities, production failures, global economic and political events. To be valid, the mentioned information usually needs to represent reality correctly from a point in time (English, 1999). Besides timeliness, there are other properties of information that are of interest to companies, such as availability and accessibility. Among various information sources, online news dwarfs others in terms of availability and accessibility while also being reasonably timely. This abundant and easy-to-access source of information gives rise to a new business intelligence aspect, extracting relevant events from news articles. Business intelligence, by definition, comprises strategies and technologies used by companies for the analysis of business information (Dedić and Stanier, 2016). Detecting relevant events and getting insights from news articles can contribute to creating effective business intelligence strategies.

An event is usually defined as a particular incident with a specific time and location (Verma et al., 2015). Among different types of events, business events are the only type to be considered due to this thesis's scope and focus. Business events differ from other event types because they contain specific incidents that can impact business intelligence, business planning, and decision making. Those events may include announcements from suppliers, changes in government policies, or reactions from consumer protection groups. The process of extracting events from text is called event detection. Most of the existing studies on the application of event detection focus on general business news (Verma et al., 2015; Jacobs et al., 2018; Lefever and Hoste, 2016) or other types of news. Due to the lack of a suitable dataset, there has not been any work concerning forest industry news. Granted, one can apply a general business event detection algorithm to forest industry news, but chances are such an algorithm will not recognize events particular to the forest industry. This erroneous classification will cause companies to miss out on some helpful information altogether. This obstacle has prevented any progress in identifying events from the forest industry. Despite the incompatibility between existing business event detection schemes and forest industry news, an event detection algorithm tailored to the forest industry should be applicable to general business news because of shared business events. This thesis aims to explore the possibility of creating such an algorithm, contributing insights to both the literature and the forest industry.

1.2 Objectives and Research Questions

This thesis aims to develop an event detection algorithm for the forest industry that also works with general business news. After reviewing the literature, it is concluded that the appropriate algorithm should be machine learning based. The introduction of a new dataset is another primary objective. Since a robust dataset for forest industry news does not exist, creating one such dataset is the prerequisite for most machine learning approaches. The thesis will go through dataset creation and the application of various text analysis methods to the generated dataset to create a fitting event detection algorithm.

To meet the thesis aims, the following research questions will be addressed:

1. Establishing an appropriate dataset that is robust enough to be used for training.

2. The feasibility of machine learning solutions as a basis for an event detection algorithm that can be applied to both general business and the forestry industry.

3. Conclusion on an optimally performing model to establish the said event detection algorithm.

1.3 Structure

The structure of this thesis comprises five chapters.

- Chapter 2 introduces the background information that will be useful for understanding this thesis and the technologies employed. Section 2.1 gives a brief overview of event detection while explaining the rationale behind choosing the deep learning approach. Section 2.2 covers some basic concepts in deep learning. Section 2.3 describes some recent advances in natural language processing, such as contextualized web embeddings. Section 2.4 returns to the topic of event detection and goes through the approaches, methods, and existing works. From a board overview, the chapter narrows down to this thesis's focus: news event detection for business with an application to the forest industry.
- Chapter 3 presents the methodologies leading to the proposed solution. It starts with the creation of a custom dataset that is representative of both general business and forest industry news. Since creating a robust dataset is among the research questions, this section is explained in detail. It also mentions the types of metrics and baselines used for evaluation in the subsequent chapter.
- Chapter 4 reviews the results from different proposed models to form an appropriate algorithm. This chapter includes the selection of pre-trained models as well as fine-tuning of those models.
- Chapter 5 concludes the thesis with a summary, contributions, managerial implications, as well as a discussion of limitations and possible future work.

This thesis includes various design choices that affect the final result. Among them, four stand out due to their impact. Termed design decisions, those four design choices are introduced in different chapters to describe the process of converging towards the proposed solution. Each design decision is followed by its rationale and implications.

2 Background

This chapter aims to give a brief overview of the ideas and technologies used in the thesis. It starts by introducing event detection and stating the approach to answer the research questions. Next, it presents the mentioned approach, deep learning, followed by natural language processing. Once the necessary technical background information has been introduced, event detection will be discussed further from a more technical viewpoint.

2.1 Event Detection (ED)

ED is a type of Information Extraction sub-task that gathers information from texts and automatically identifies knowledge on incidents existing in those texts (Wang, 2018). ED works by identifying keywords or triggers that signify events and classify the discovered events into existing groups (Xiang and Wang, 2019). ED is the first step to understanding semantic information of events.

Being a popular sub-task of Information Extraction, detecting events from large volumes of unstructured text data has attracted a sizeable amount of study and research. There are three main approaches to detecting events from literature: data-driven, knowledge-based, and hybrid approach. Data-driven ED requires little domain knowledge and expertise but needs a large amount of data. On the contrary, the knowledge-based approach uses a modest amount of data but requires strong domain knowledge and expertise. The third approach is a compromise between the first two approaches. This approach requires a medium amount of data and domain knowledge; however, it needs strong expertise due to the combination of many techniques (Hogenboom et al., 2011).

There have been many attempts to use the knowledge-based method to detect events. The main drawback of relying on this approach is the time-consuming and challenging process of creating rules and ontologies. Low performance is another downside since a set of rules, no matter how complex, cannot cover all the possible manners that information can be put in text (Hogenboom et al., 2013). This reduced performance gives way to the data-driven approach, which does not require a set of predefined rules. Relying on quantitative methods and a large amount of text, the data-driven approach to ED develops models to capture the underlying semantic information based on statistical relations.

The choice of ED approaches leads to the first design decision of this thesis. The thesis will concentrate on data-driven ED algorithms. The rationale behind this is the constraints associated with the knowledge-based approach, such as degraded performance. The limitations of the author's domain knowledge also contribute to the unfeasibility of non-data-driven models. From this point on, the Background chapter will focus on the data-driven approach to answer the research questions. This data-driven approach corresponds closely to machine learning, or more precisely, its deep learning subset. As for ED, the rest of this subject will be discussed in the ED revisited section (Section 2.4) after the necessary technical foundation has been presented.

2.2 Deep Learning (DL)

Machine learning (ML) plays a huge role in advancing modern society with countless applications such as object identification, language translation, and product recommendation. However, classical ML techniques such as linear models suffer from the inability to work directly with raw data. For an ML system to learn the data's underlying pattern, feature engineering is required (Domingos, 2012). Since feature engineering is both time-consuming and challenging to do, a new class of techniques called DL was introduced.

By mimicking the biological nervous system, DL methods make use of highly interconnected non-linear processors called neurons. This naming gives rise to the term 'neural networks'. Essentially, DL methods stack multiple layers of processors to learn directly from raw data (Goodfellow et al., 2016). DL models can learn different representation levels from the raw input, one at each of their layers. At each layer, non-linear processors are used to transform the given input into representations to pass onto the next layer. Those representations become progressively more abstract as the depth of the model increases. Given sufficient depth, DL models can learn complex functions and the data's underlying patterns. This learning capacity increases DL performance while skipping the manual design of feature engineering (LeCun et al., 2015).

There are various DL models, yet this section only covers a few of them: feedforward neural network, recurrent neural network, and Transformer. Feedforward neural network is included to serve as a starting point for other model types. With recurrent neural network, two submodel types are examined, namely long short-term memory and gated recurrent units. From there, sequence-to-sequence model is briefly touched on to build up to

Transformer. Recurrent neural networks and Transformer are the popular choices for Natural Language Processing tasks and are considered in this thesis. This section also discusses other DL-related topics such as optimization and transfer learning.

2.2.1 Feedforward Neural Networks (FFNN)

The most basic neural network, FFNN, has the following structure: one input layer, *n* number of hidden layers, and one output layer. In FFNNs, the neurons at one layer connect directly to the next layer's neurons. These networks can learn complex functions as well as non-linear decision boundaries (Kriesel, 2007).

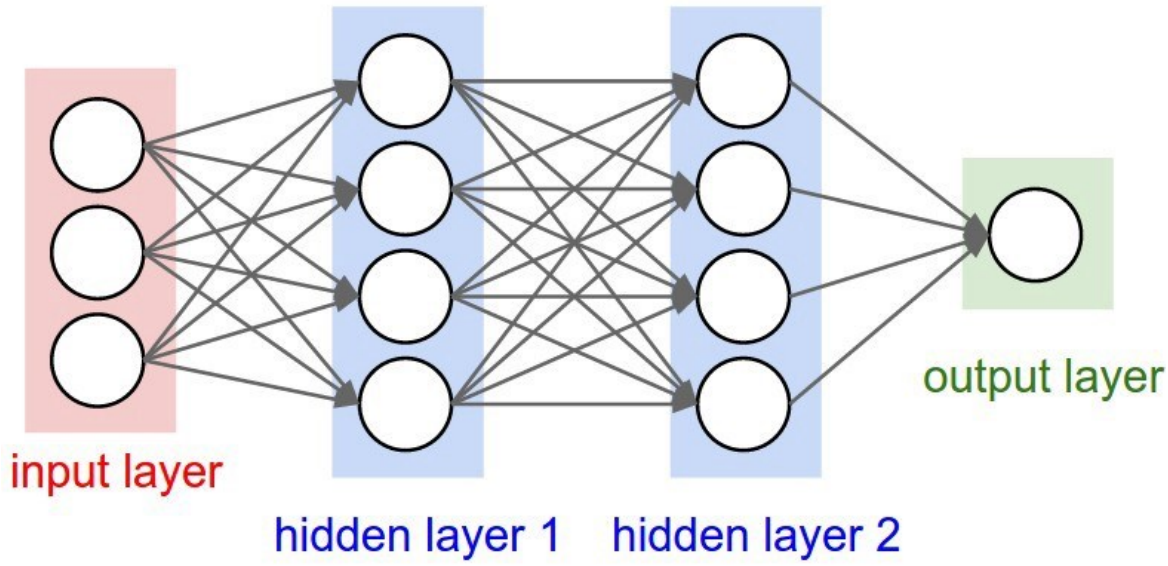

Figure 1. An example of a four layers FFNN. (Stanford University Online, 2020)

Feedforward networks are trained by using forward and backward propagations. Forward propagation is the process of getting input information to create an output. Given the input x, weight w, bias b, and activation function f, the output value of a layer $h(x)$ is computed as:

$$a(x) = b + \sum_i w_i x_i \tag{1}$$

$$h(x) = f\big(a(x)\big) = f\left(b + \sum_i w_i x_i\right) \tag{2}$$

Most often, the function f is a non-linear function. The most commonly used function is the sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{e^{x} + 1} \tag{3}$$

In the output layer *o*, the final activation function f_o depends on the type of task the network is designed to solve. If the network is solving a regression problem, the output is a continuous variable, and there is no activation function. If the network deals with a binary classification problem, the sigmoid function can be reused. In the case of a multi-class classification problem, the softmax activation function can be applied instead:

$$S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{4}$$

Once the network has the outputs, it will compute the difference between those outputs and ground truths with a loss function. The loss function also depends on the type of task. For classification problems, the cross-entropy loss is often utilized:

$$L = -\sum_{c=1}^{C} y_c \log\big(p(y_c)\big) \tag{5}$$

with $p(y_c)$ being the predicted probability of class *c* and *y* being the binary indicator if class label *c* is correct.

For regression problems, the mean squared error is often employed:

$$L = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{6}$$

with $\hat{y}_i$ being the ground truth of the i^{th} training observation.

2.2.2 Recurrent Neural Network (RNN)

Lacking the capacity to store information, FFNNs are not so helpful in tasks that involve more extended input sequences. For such tasks, RNNs are often the preferred choice. Introduced by Elman (1990), RNN is a family of neural networks that specialize in processing sequential data. The main difference between traditional FFNNs and RNNs is that RNNs have a feedback loop to retain information. To work with variable sequence lengths, RNNs also have parameter sharing, which allows them to generalize to different lengths and recognize a repeated piece of information. The feedback loop and parameter

sharing allow RNNs to recognize a relevant word in a sentence based on the previous words (Goodfellow et al., 2016).

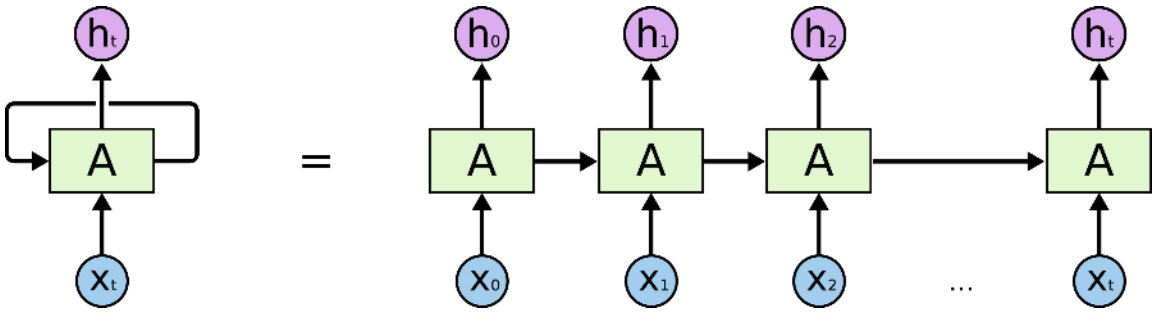

Figure 2. A simple RNN cell and its unfolding (Olah, 2016).

Figure 2 shows a simple RNN and how its loop structure allows information to pass to the next step. With x_t as the input at time *t*, the output at time *t* h_t is updated by:

$$h_t = f(w[h_{t-1}, x_t] + b) \tag{7}$$

where w is the weights, b is the bias, f is a nonlinear function such as sigmoid or tanh, and $[h_{t-1}, x_t]$ denotes the concatenation of the output of the last time step h_{t-1} and the current input x_t.

In cases like ED, future information can also be helpful. The words at the end of the sentence can be indicative of the relevant event. Instead of just looking at the sequence from the beginning to the end, bidirectional RNNs also consider future information by examining that sequence in both forward and backward fashions. This bidirectional structure generally performs better than its unidirectional counterpart (Schuster and Paliwal, 1997). This logic applies to other RNN architectures.

Two RNN types of architectures are discussed below: Long Short-Term Memory and Gated Recurrent Unit.

2.2.3 Long Short-Term Memory (LTSM)

One problem associated with RNNs is learning long-term dependencies by using the gradient. In the context of neural networks, gradient refers to the gradient of the loss function concerning the network's weights. By multiplying the weight *w* by itself many times, RNNs can suffer from vanishing or exploding gradient problems: the gradients can either vanish

(most of the time) or explode (rarely) (Bengio et al., 1994; Pascanu et al., 2013; Goodfellow et al., 2016). Both cases prevent RNNs from learning any pattern from data. To remedy this issue, LSTM, based on the idea of paths that the gradients can flow through without vanishing or exploding, was introduced by Hochreiter and Schmidhuber (1997).

The critical component of LSTM is a memory block with a cell memory and three gates: input, output, and forget gates. Those gates give LSTM units the ability to add to or remove information from the cell memory. Each gate's output is a value between zero and one, designating how much information can pass through the gate.

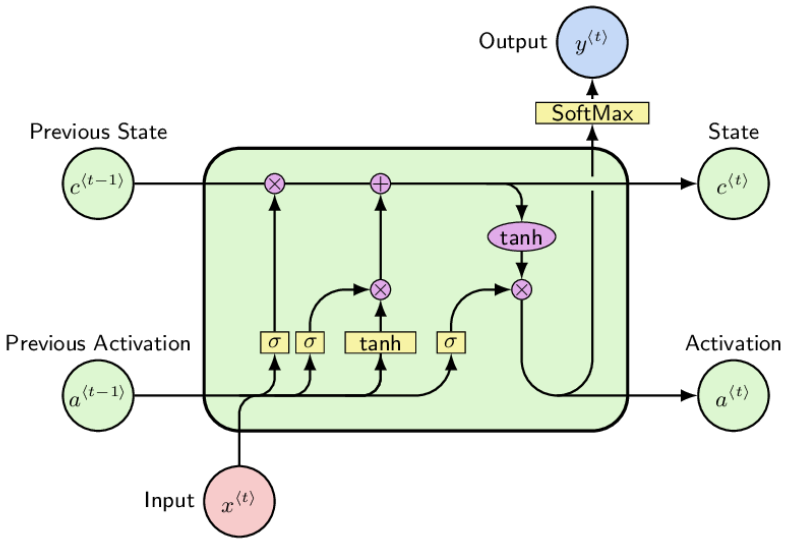

Figure 3. An LSTM cell with its gates (Gago et al., 2019)

The forget gate f_t controls how much of the current information should be forgotten. w_f, b_f are its weights and biases, respectively, and σ is the sigmoid activation function.

$$f_t = \sigma\big(w_f[h_{t-1}, x_t] + b_f\big) \tag{8}$$

Similarly, the input gate i_t to decide how much of the new value should be included in the current cell memory $\tilde{c}_t$. $w_i, b_{i,}$ are the weights and biases of the input gate i_t.

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{9}$$

The newly updated value in the current cell memory $\tilde{c}_t$ is calculated by a tanh activation function. w_c, b_c are the weights and biases of the current cell memory $\tilde{c}_t$.

$$\tilde{c}_t = tanh(w_c[h_{t-1}, x_t] + b_c) \tag{10}$$

Between the chosen memory value from the forget gate f_t and the new value from the input gate i_t, the cell memory c_t can be updated as follows:

$$c_t = f_t \circ \tilde{c}_{t-1} + i_t \circ \tilde{c}_t \tag{11}$$

where $\circ$ represents the Hadamard product.

To calculate the output at time *t,* tanh is applied to the Hadamard product of the cell memory c_t and the output gate o_t.

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{12}$$

$$h_t = o_t \circ \tanh(c_t) \tag{13}$$

where w_o, b_o are the weights and biases of the output gate i_o.

Compared to a traditional RNN cell structure, an LSTM cell is more complex with different gates and a memory cell. This complexity allows LSTM cells to filter out and retain relevant information.

2.2.4 Gated Recurrent Unit (GRU)

Similar to LSTMs, GRUs also use gates to bypass the gradient optimization issue with traditional RNNs. Introduced by Cho et al. (2014), GRU only has two gates, reset and update gates. A smaller number of gates makes GRUs simpler in structure and generally faster to train than LSTMs. For smaller datasets, GRUs are preferred since they enjoy efficiency benefits and can perform better than LSTMs. On the other hand, for more extensive datasets or ones with longer sentences, LSTMs are favored (Yin et al., 2017).

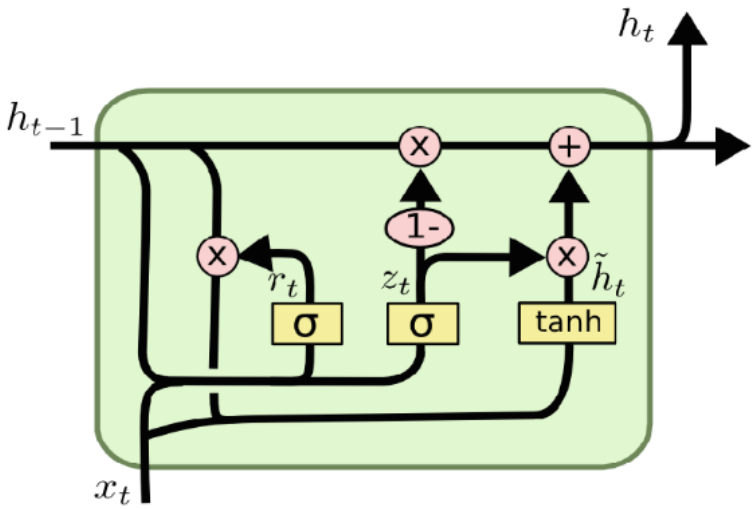

Figure 4. A GRU unit with its reset and update gate (Anderson, 2019)

To get the output h_t from the input x_t at time t with a GRU cell, the following calculations are made:

$$r_t = \sigma(w_r[h_{t-1}, x_t] + b_r) \tag{14}$$

$$z_t = \sigma(w_z[h_{t-1}, x_t] + b_z) \tag{15}$$

$$\tilde{h}_t = tanh(w_h[r_t \circ h_{t-1}, x_t] + b_h) \tag{16}$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t \tag{17}$$

The reset gate r_t can choose to ignore or copy the previous hidden state h_{t-1} to the hidden state $\tilde{h}_t$. The update gate z_t can choose whether to update the memory content h_t with a new hidden state $\tilde{h}_t$. w_r, b_r, w_z, b_z and w_h, b_h are the weights and biases of reset gate, update gate, and new hidden state, respectively.

Thanks to the gated structure, both LSTM and GRU model structures can discard irrelevant or redundant past information, allowing them to learn meaningful temporal relationships from the data. The learned temporal relationships, in turn, make both LSTM and GRU better at capturing long-term dependencies than a simple RNN (Chung et al., 2014; Greff et al., 2016).

2.2.5 From Sequence-to-Sequence model to Transformer

Sequence-to-sequence

Working with text has given rise to many different DL architectures besides RNNs. Among them is sequence-to-sequence (seq2seq) model (Sutskever et al., 2014), consisting of an encoder and a decoder. The encoder takes an input sequence and maps it to a hidden vector representing a higher dimension space. This hidden vector is encoded with all the information from the input source. The decoder will then map the hidden vector to create an output sequence that maximizes the probability P(output|input). Seq2seq model can handle inputs of different lengths and is often made up of LSTMs (with one LSTM acting as the encoder and another as the decoder). However, seq2seq model has a drawback. Since the fixed-length hidden vector is created from all input information, long input sequences can lead to decreased performance.

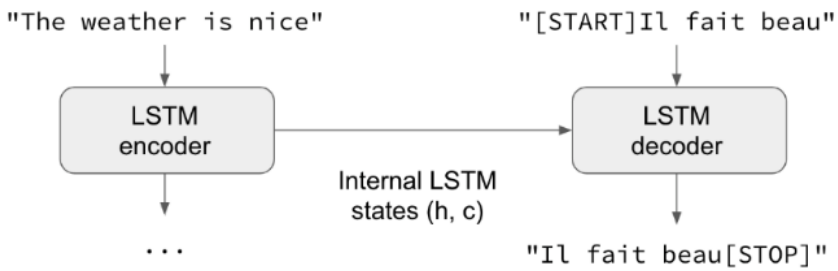

Figure 5. An encoder-decoder model translates from English to French. Here the hidden vector is called Internal LSTM states (Chollet, 2017)

Attention

To deal with long sequences more efficiently, the attention mechanism was introduced by Bahdanau et al. (2014). With RNNs, if two words are ten words apart, then there will be ten computation steps. With attention, those two words can be related in just one step. The influence of each word by all the other words in the sequence is collected as weights. When the model predicts the next output word, instead of using the entire input sentence, attention only uses parts of the input with the most relevant information. Since attention weights are

computed for every input and output combination, the attention mechanism can become costly if there are many words in the sequence.

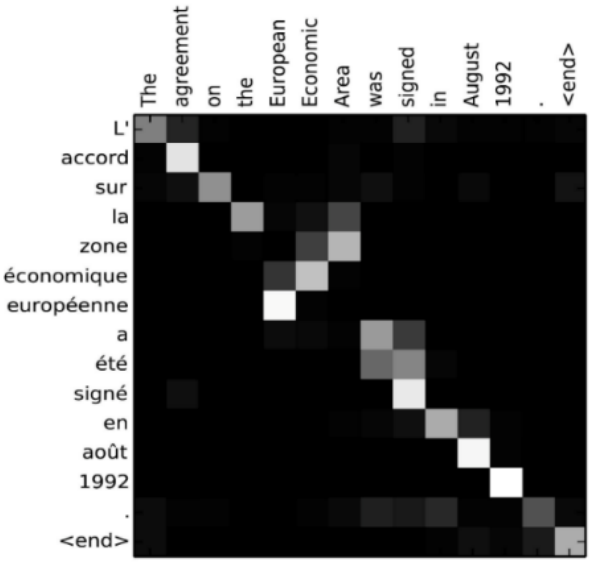

Figure 6. Alignment (attention weight) matrix of a translation from French to English (Bahdanau et al., 2014).

The learned dependencies here are only between inputs and outputs. The Transformer architecture takes this idea further and learns the dependencies between inputs and between outputs.

Transformer

Transformer (Vaswani et al., 2017) is an architecture that aims to learn long-range dependencies without using RNN structure. It only makes use of the attention mechanism, which allows for faster computing. Since it does not use RNNs, Transformer uses positional encoders in the embedded word representation to keep track of the word's order. Unlike the seq2seq architecture, which only has a single encoder and decoder, the Transformer architecture has multiple encoder and decoder cells in its encoding and decoding components. Those cells share the same structure but not the weights.

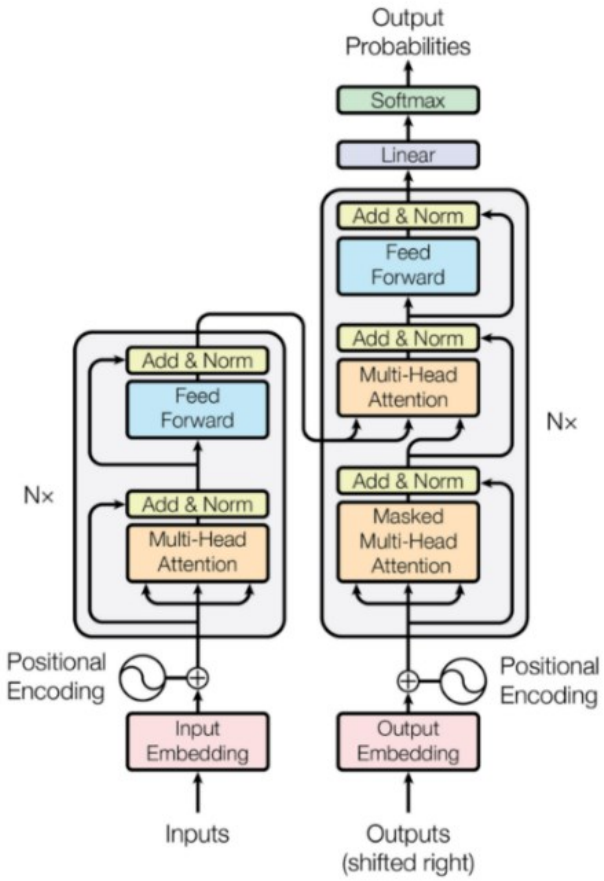

Figure 7. The Transformer architecture (Vaswani et al., 2017).

For other encoder-decoder architectures, the output word at position i^{th} corresponds to the i^{th} word in the decoder input. In Transformer's decoder, the input is shifted right. By shifting the decoder input to the right by one position, the model will have only seen words with $1 \ldots i^{th-1}$ index in the decoder sequence, forcing it to predict the next word at the i^{th} position. Besides attention and feedforward layers, the Transformer's decoder also has an extra attention layer to focus on essential parts of the input sequence. Since Transformer makes up primarily of attention layers, it suffers from attention's drawback. Because attention only works with fixed-length strings, longer input sequences can be split at the middle, leading to loss of context. This context loss is called context fragmentation.

Despite its shortcoming, the Transformer architecture and its variants have led to many advances in DL research. Transformer is often used in NLP and time-series prediction (Devlin et al., 2018; Yang et al., 2019). In natural language processing, the Transformer

architecture has allowed the introduction of many state-of-the-art models like BERT and GPT-3, along with breakthroughs in machine translation and speech recognition tasks (Wang et al., 2019; Liu et al., 2020).

2.2.6 Backpropagation

Moving on from model introduction, the following sections will discuss relevant DL concepts, starting with backpropagation. Once the network finishes computing the output, it can be trained by updating its weights and bias with the backpropagation process. Popularized by Rumelhar et al. (1986), backpropagation determines the error for each weight and bias by calculating their derivatives. The weights and biases are then updated accordingly based on some predefined optimizers. Backpropagation can be better understood with an example. With out_o as the output of an FFNN, net_o as the pre-activation before the final output, w as the weight connecting the first neutron from the last hidden layer to the output, and out_{h1} as the output of that neutron, the effect of a change in w on the total error *E is*:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial out_o} \times \frac{\partial out_o}{\partial net_o} \times \frac{\partial net_o}{\partial w} \tag{18}$$

The components from Equation (18) and their partial derivatives are:

$$E = \frac{1}{2}(target - out_o)^2, \qquad \frac{\partial E}{\partial out_o} = -(target - out_o)$$

$$out_o = \frac{1}{1+e^{-net_o}}, \qquad \frac{\partial out_o}{\partial net_o} = out_o(1 - out_o)$$

$$net_o = w \times out_{h1} + \cdots, \qquad \frac{\partial net_o}{\partial w} = out_{h1}$$

Equation (18) can be written as:

$$\frac{\partial E}{\partial w} = -(target - out_o) \times out_o(1 - out_o) \times out_{h1} \tag{19}$$

Since $\frac{\partial E}{\partial net_o} = \frac{\partial E}{\partial out_o} \times \frac{\partial out_o}{\partial net_o}$ can be denoted as δ_o, equation (19) can be shortened to:

$$\frac{\partial E}{\partial w} = \delta_o out_{h1} \tag{20}$$

Weights and bias are then updated according to an optimizer. With gradient descent, the standard optimizer in training networks, the update equation is as follows:

$$w = w - \alpha \frac{\partial E}{\partial w} \tag{21}$$

with α as the learning rate of the optimizer.

2.2.7 Optimization

Building an optimization model is the core of most ML algorithms. An ML algorithm typically tries to optimize some objective function $f(x, \theta)$ with θ as the parameters. This objective function is also known as the loss function (Goodfellow et al., 2016). The optimum result θ^* is the one that minimizes the function f: $\theta^* = \text{argmin}_\theta f(x, \theta)$. For maximization problems, the optimization task is to minimize $-f$. As a result, choosing an appropriate optimization algorithm and its learning rate plays an integral part in the training of ML models.

While there are many optimizers for ML, this section introduces only three: gradient descent, Adam, and RAadam. The first one is the most common optimizer, while the other two are used in this thesis. Adam and RAadam are chosen since they are less affected by unfitting learning rate choices.

Gradient descent

Gradient descent is an interactive approach that minimizes some function by moving in the direction that brings about the steepest descent as defined by the gradient's negative (Ruder, 2016). As ML algorithms try to minimize loss functions, it is the job of gradient descent to find the minima of those functions. Once the direction to move in is decided, the weights can be updated according to equation 21. The size of the update is called learning rate. A suitable learning rate must be chosen to reach the global minima. A too-large learning rate makes gradient descent overshoot and has trouble reaching the global minima. A too-small learning rate will slow down the network's training or get stuck at local minima.

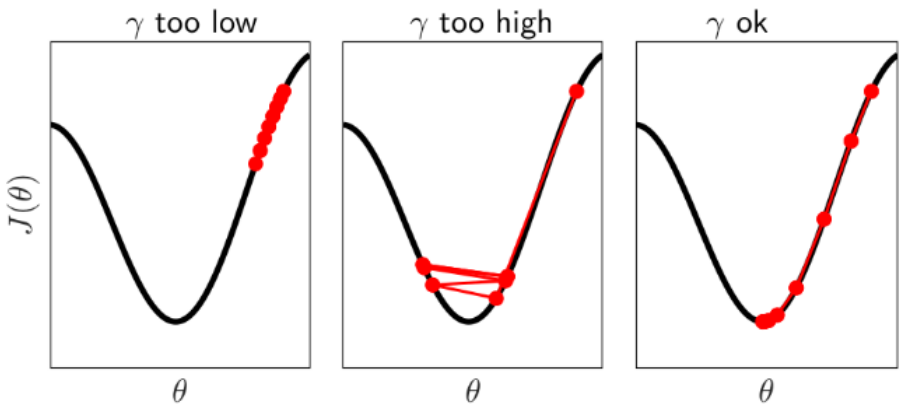

Figure 8. The importance of choosing a reasonable learning rate. (Liquet, B., Moka, S. and Nazarathy, Y., 2021)

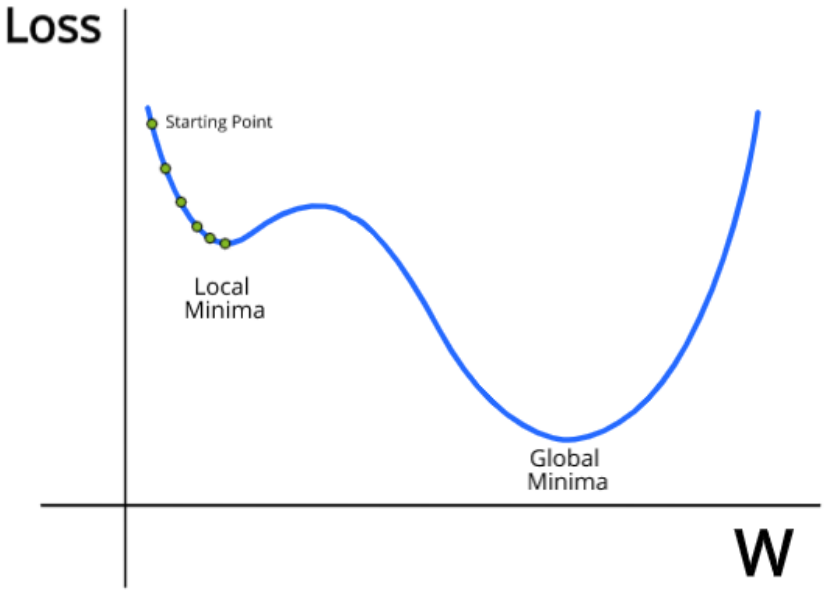

Figure 9. An example of a function with minima (Deep Learning Demystified, 2020)

Gradient descent has three variants depending on the amount of data used to compute the gradient. This difference in data amount can lead to a trade-off between the time to perform a parameter update and the update's accuracy (Ruder, 2016).

Batch gradient descent uses the entire training set to compute one update. By calculating all individual gradients over each sample in the dataset, batch gradient descent returns a precise estimate of the true gradient. However, this type of update is slow and potentially impractical if the training dataset is large. Its biggest issue is that batch gradient descent can get stuck in local minima.

Stochastic gradient descent (SGD) represents another extreme of gradient descent. SGD uses only a single data sample to compute the loss at each iteration, leading to faster learning or updating of parameters. The estimations made by SGD are often unstable, leading to frequent updates with a high variance. This noisiness helps escape from local minima. On the other hand, this noisiness also makes it challenging to find and remain at a global minimum.

Mini-batch gradient descent is a compromise between batch gradient descent and SGD. It performs an update for every *n* training sample. Called batch size, *n* typically ranges from 16 to 128. By performing frequent updates with enough noise in each update, mini-batch gradient descent allows for a faster and more stable convergence. The main drawback of this approach is the introduction of a new hyperparameter, batch size.

Adam

Adam or 'adaptive movement' is one of the more popular optimization algorithms. Introduced by Kingma and Ba (2014), Adam uses momentum and adaptive learning rate to converge faster. Adaptive learning rate, also known as learning rate schedule, varies learning rate during the training process. The learning rate is significant initially and will decay accordingly during training to allow for faster convergence. Besides adaptive learning rate, Adam uses momentum to keep track of an exponentially weighted moving average of the past gradients. If a weight keeps moving in a particular direction, it will gather momentum. When faced with resistance, that weight will temporarily continue in the original direction because of the stored momentum. This mechanism helps to overcome local minima. Furthermore, due to the accumulation of gradients, the weight will converge faster (Deep Learning Demystified, 2020).

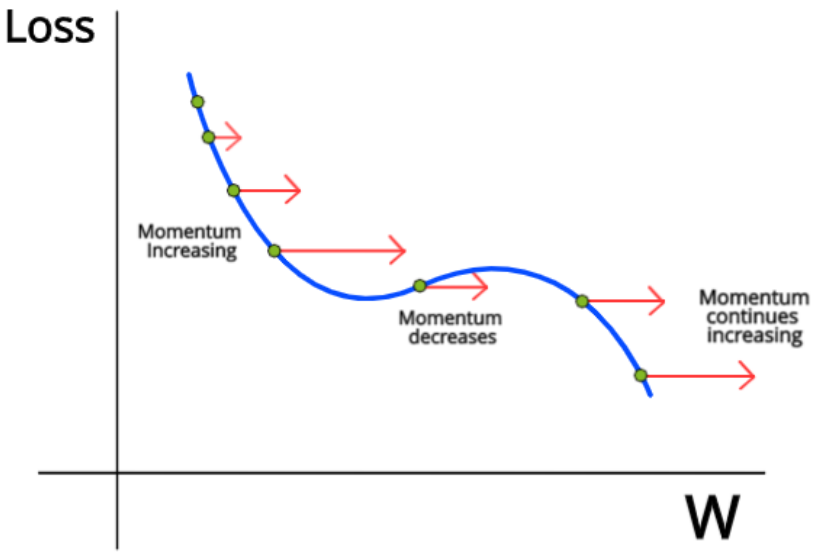

Figure 10. How momentum can overcome local minima (Deep Learning Demystified, 2020)

The combination of momentum and adaptive learning rate makes Adam a robust algorithm that often performs better than other optimizers. With adaptive algorithms such as Adam, the starting learning rate no longer makes or breaks the training process. However, a reasonable starting learning rate is still preferred since it can give the model an extra boost in performance.

RAdam

RAdam (Liu et al., 2019) is a variant of Adam optimizer. Called Rectified Adam, it tries to rectify the variance of adaptive learning rate. It was pointed out that optimizers with adaptive learning rate often need a warm-up heuristic by using a lower learning rate for the first few training iterations (Vaswani et al., 2017; Popel and Bojar, 2018). Without this warm-up stage, adaptive optimizers often end up in local minima due to a significant variance caused by the limited number of training samples in the early training stage. RAdam attempts to rectify this type of variance by measuring the length of the simple moving average of momentum. If the length is less or equal to four, the adaptive learning rate is not activated due to intractable variance. If the length is more than four, the variance rectification term is calculated, and the parameters are updated with adaptive learning rate.

2.2.8 Transfer learning

Due to the size of many DL architectures, training a model from scratch is a substantial undertaking that can be both time- and resource-consuming. In some other cases, the available dataset can be too small to train a full-scale model. To remedy this, pre-trained models are used. Pre-trained models are saved architectures that have been trained on a large amount of data and perform exceptionally well in their respective tasks (Yosinski et al., 2014; Zhuang et al., 2020). The intuition behind using pre-trained models is that if a model is trained on a large and general enough dataset, then that model can be used as a generic model of that field. Among the fields that make frequent use of pre-trained models are object detection and facial detection for computer vision, translation, speech recognition, and text classification for NLP (Dai and Le, 2015). When it comes to text, some of the well-known pre-trained models are Google's BERT, EMLo, or OpenAI's GPT-2 and GPT-3. For those architectures, specialized corpora and the English Wikipedia are often utilized to ensure the quality of data used in pre-training.

Pre-trained models can be used as is, or they can be tuned for some specific tasks. In the fine-tuning process, the base model's final layer is replaced by a layer customized for the current task. The newly added layer and some of the base model's last layers are trained to a specific task while the rest of the model is frozen. This partial updating of the model's parameters allows the new architecture to retain the existing feature representations while making them more relevant for the task at hand (Yosinski et al., 2014). Transfer learning helps models reach good performance in a short amount of time, with a small amount of training data, and within some computational restraints.

Transfer learning is not without its limitations. Currently, one of the most significant drawbacks to transfer learning is negative transfer. For transfer learning to be successful, the original and the end problem should be similar (Williams et al., 2020). If they are not, the model's performance can degrade. Determining the similarity between different kinds of training is still based on speculation. To remedy this, Williams et al. (2020) introduced affinity, which is "*an expected increase or decrease in performance on the recipient problem when using a learning resource*". This concept should provide a clue on how suitable a transfer learning task is. The second issue with transfer learning is overfitting. In this case, the model learns details and noises from the training dataset and negatively affects its

performance. This overfitting issue can be overcome by choosing a test dataset representing the same distribution that produced the training dataset.

2.3 Natural Language Processing (NLP)

NLP is an area of research that focuses on getting a better understating of natural language using computers. This understanding can be achieved by making language text handleable to computers or "*extracting simpler representations that describe limited aspects of the textual information*" (Collobert et al., 2011). NLP researchers aim to create techniques that imitate how humans understand and use languages so that computers can apply them to perform language-related tasks. NLP has applications in many fields, such as machine translation, natural language text processing, summarization, and speech recognition (Vijayarani et al., 2015).

The traditional NLP process often includes several pre-processing steps. Those steps are tokenization, stop word removal, lemmatization, stemming, and representation. Tokenization simply breaks down a string into substrings such as word, character, or subword. Stop words removal removes common stop words such as 'a', 'an', or 'the' that do not contribute anything to the learning process. Stemming is used to reduce words into their base forms. For example, both 'consultants' and 'consulting' are stemmed to 'consult'. On the other hand, lemmatization groups together inflected forms of a word to be analyzed as a single item. For example, 'is' and 'was' are both lemmatized to 'be'. Finally, representation is used to create computer-readable information. Typically, words are usually represented in a one-hot representation. In this representation type, the represented word's index has a value of one while every other index has a zero. Word representation results in a matrix with a size proportional to the number of words and samples in the document. This representation approach leads to the curse of dimensionality: the joint probability of different sequences of words can lead to an enormous number of parameters. To fight the curse of dimensionality, distributed representation of words in a lower-dimensional space was introduced (Bengio et al., 2003). Two approaches to distributed representation, called word embedding and contextualized word embedding, are discussed below, along with one model for each approach.

2.3.1 TF-IDF

TF-IDF is not a word embedding approach, but its popularity in NLP merits a brief introduction. TF-IDF stands for term frequency-inverse document frequency. The normalized Term Frequency is computed by counting how many times a word appears in a document divided by the sum of words in that document. The Inverse Document Frequency is the logarithm of the number of documents in the corpus divided by the number of documents with a specific word (Rajaraman and Ullman, 2011). This approach to text representation lowers the weights of stop words while enhancing the weights of words providing valuable information.

2.3.2 Word Embeddings and word2vec

Based on the idea that similar words are likely to appear in similar contexts, word embedding tries to learn the characteristics of the neighbors of a word (Young et al., 2018). With word embedding, there is no one-to-one mapping between the words and the elements of a vector. Instead, each word is represented by a fixed-length vector, and similar words have comparable representations. These embeddings are better at capturing context analogy and word relationships while being more efficient for NLP tasks. Combining DL with word embedding has led to a state-of-the-art performance in various NLP tasks such as machine translation (Luong et al., 2015), text classification (Lai et al., 2015), or entity matching (Mudgal et al., 2018).

There are two main types of embeddings, frequency-based and prediction-based. The frequency-based embedding method is deterministic and is calculated by counting. This simple method to creating embeddings limits their potential until the introduction of word2vec, which is prediction-based. Introduced by Mikolov et al. (2013), word2vec still treats each word as a densely distributed representation. However, word2vec is more efficient in learning distributed representations of words, especially in large corpora of text. Its performance is due to two new models: the continuous-bag-of-word and skip-gram model. The continuous-bag-of-word model tries to predict the target word based on the context from surrounding words, whereas the skip-gram model does the opposite, predicting the surrounding context or words from the target word.

Being a popular embedding approach, word2vec stands out thanks to its properties. Word2vec embeddings can capture the underlying syntactic and semantic information from

the source words. In terms of visualization, similar words often have their representations appearing close to each other in Euclidean space. Furthermore, thanks to the captured semantics information, learned word2vec vectors can be displayed as linear relationships. Adding two learned vectors will create a semantic composite of the represented words in the form of a new vector. For example, 'man' + 'royal' = 'king' (Young et al., 2018). With this property, word2vec can be used to answer analogy questions by adding or subtracting vectors, then using distance measurements to find the word closest to the resulting vector.

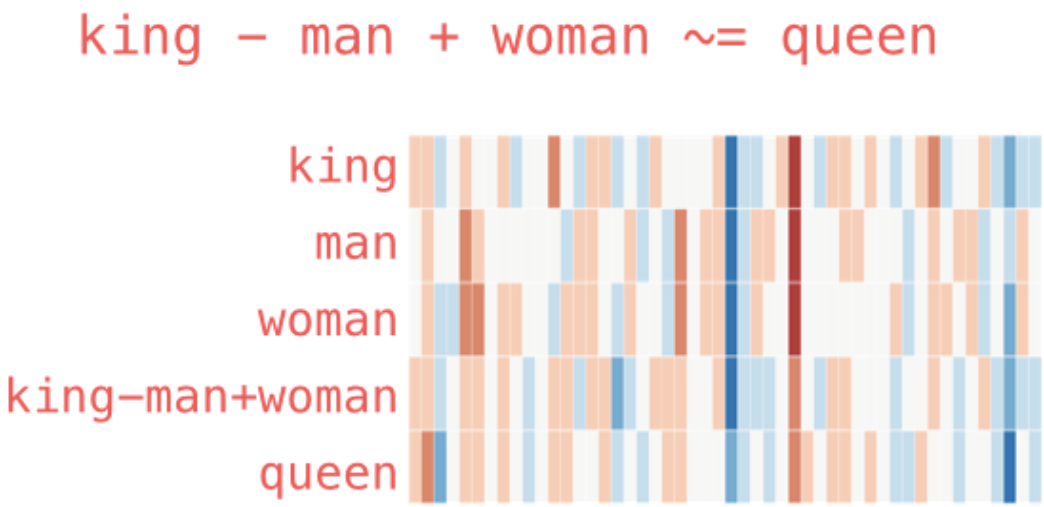

Figure 11. Word2vec vector representations that can be used for analogy tasks, showcased by the last two rows (Alammar, 2019)

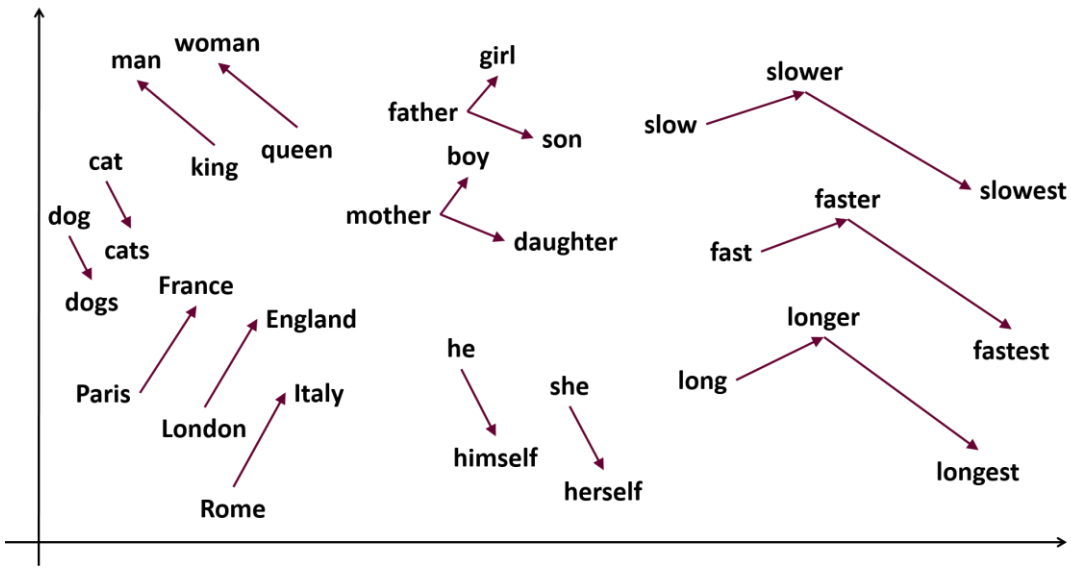

Figure 12. Word2vec vector representations in Euclidean space and their linear relationships (Pal, 2019)

2.3.3 Contextualized Word Embeddings and BERT

Traditional word embeddings such as Word2vec have one major shortcoming. To create a vector representation of a word, those word embeddings methods examine all the sentences containing that word. The resulting embedding is called a global vector representation. However, this embedding approach does not account for words with multiple context-dependent meanings. One example is the word 'bank', which can either means a financial institution or a riverside. Creating a global vector representation for such a word would mean missing out on one or more meaning(s). To account for polysemy, Peters et al. (2018) introduced a newer class of models that considers contextual word embeddings instead. By examining the sequence of every word in the documents, contextual word embeddings can be used to learn sequence-level semantics.

One of the newest models that use contextualized word embeddings is BERT (Bidirectional Encoder Representations from Transformers) from Google AI Language. Introduced by Devlin et al. (2018), BERT has helped many learning algorithms achieve state-of-the-art results in various NLP tasks like machine translation (Yang et al., 2019) and document classification (Adhikari et al., 2019). It is a pre-trained deep model that utilizes Transformer. Traditionally, Transformer includes two separate blocks, an encoding block that reads the input and a decoding block that outputs the prediction (Section 2.2.5). BERT only has the encoding part. BERT also incorporates bidirectional training of the Transformer architecture, which goes through a corpus entirely to learn the context of a specific word based on words on both sides of the target word. By processing a word in relation to all other words in a sentence instead of one by one processing, BERT can identify the context of a given word more effectively.

BERT was pre-trained on an unlabeled corpus (the entire English Wikipedia and the Brown Corpus). The model continues to learn from unlabeled text in its practical applications, e.g., Google search. BERT has two different pre-training tasks for better contextual learning: masked language modeling and Next Sentence Prediction (NSP). In the first task, BERT masks 15% of the word in each sentence and then predicts only the masked words based on other words. This masking avoids the issue that words can indirectly "*see themselves*" in deep bidirectional language models (Devlin et al., 2018). The second task, NSP, has BERT predict a subsequent sentence based on a previous sentence. The second sentence can either be the true subsequent sentence or a sentence chosen randomly from the

corpus. The assumption is that if two sentences are connected, there should be a contextual link between them. NSP is motivated by the importance of understanding the relationship between two sentences in various NLP tasks. Combined, those two pre-training tasks help BERT learn the context within a sentence and between sentences. During training, both tasks are trained together to minimize the pooled loss functions.

By applying transfer learning and fine-tuning, BERT can be used in different tasks such as machine translation, summarization extraction, or text classification. Being transfer learning compatible has increased BERT's popularity among researchers. This recognition will allow for a wide range of practical applications using BERT in the future. BERT has inspired many architectures named after BERT itself: ALBERT, RoBERTa, and StructBERT.

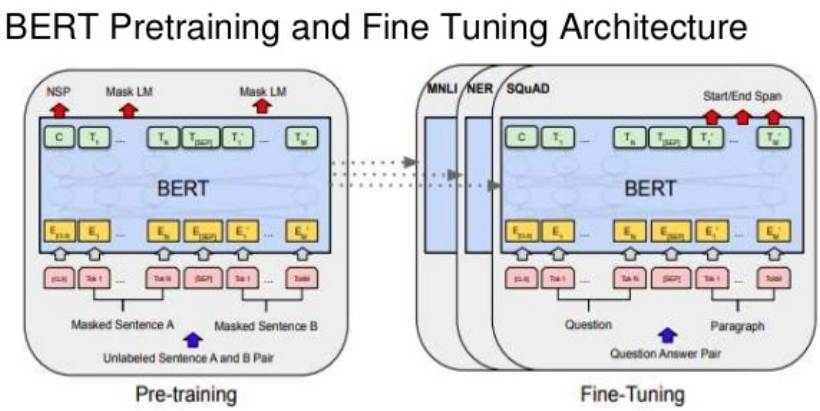

Figure 13. BERT pre-training and fine-tuning. In pre-training, BERT includes both masked words and masked sentence prediction tasks (Devlin et al., 2018)

2.4 ED revisited

After determining the type of ED approach to use and introducing the necessary technical information, this section discusses how ED can be applied in this thesis. It starts by examining different methods for the data-driven ED approach, followed by ED application to news. Finally, the current state of ED concerning the forest industry is discussed.

When it comes to data-driven ED, there are two main modeling methods: clustering and classification, which align with unsupervised and supervised learning techniques, respectively. The third approach is a semi-supervised one which aims to reduce the time and

cost of obtaining labeled data by combining a small amount of original data and generated data (Xiang and Wang, 2019).

2.4.1 Unsupervised ED - Clustering

Clustering is one of the most popular algorithms for working with text data. It has been studied extensively in a wide range of applications such as visualization and document organization (Allahyari et al., 2017). Since clustering algorithms do not require labeled training data or existing groups, they can be referred to as automatic event classification. When used with text data, clustering discovers and groups similar texts or documents from a collection of documents. At its simplest form, clustering models find word co-occurrence count between sequences of text data and create groups containing overlapping sets of words. Moving away from word matching, clustering models can use measures of similarity between documents or sentences to group those items. The idea is to process sequences of text data into representations such as vectors or matrices and then apply a clustering algorithm to those representations. Word representations can be created by either TF-IDF (Bafna et al., 2016), word2vec (Ma et al., 2016), or BERT (Gencoglu, 2018).

There is a variety of clustering algorithms that researchers can choose from:

- Centroid-based algorithms, e.g., K-means (Choromanska and Monteleoni, 2012).
- Hierarchical-based clustering (Dai et al., 2010).
- Density-based clustering, e.g., DBSCAN (Cretulescu et al., 2019).
- Distribution-based, e.g., Latent Dirichlet Allocation (Moro et al., 2015).

2.4.2 Supervised ED - Classification

Text data classification is a popular topic among many communities, such as ML and information retrieval. It has various applications in different domains like document organization, fraud detection, or recruitment. Similar to text clustering, most text data classification approaches start with finding word representations to create computer-readable values. The training step involves having classifiers learn the representations' underlying patterns given some labels. Once trained, the classifiers can predict which label(s) a sequence of texts belongs to. To evaluate the performance of the trained models, predicted labels are compared to actual labels using metrics. Among the evaluation metrics, accuracy is the simplest, and it measures the portion of correctly classified instances to the

total number of instances. The issue of metrics will be revisited in the Methods chapter (Section 3.2.1).

A critical choice in any classification problem is the type of classifier to use. There are different types of classifiers to consider, such as probabilistic, proximity-based, and linear models. However, the most successful models for ED use neural networks. Below are some examples for each of the mentioned types.

- Probabilistic: Naïve Bayes classifier (Ekta et al., 2017).
- Proximity-based: K-NN classifier (Verma et al., 2015).
- Linear: SVM classifier (Jacobs et al., 2018).
- Neural Networks: convolutional neural network (Nguyen and Grishman, 2015), RNN (Chung et al., 2014; Nguyen et al., 2016; Ghaeini et al., 2016), and hybrid network (She and Zhang, 2018).

2.4.3 News ED in general

Due to its usefulness in dissecting an ever-growing amount of online texts, ED has enjoyed a boost in popularity. Among the data-driven studies on ED, the majority focus on either business (Verma et al., 2015; Jacobs et al., 2018; Lefever and Hoste, 2016), politics (Abrishamkar et al., 2018), or other news such as natural disasters (Nugent et al., 2017, Domala et al., 2020). The distinction in topics accompanies the difference in news sources. Not limited to online news articles, ED can be applied to other news sources such as social media platforms. For news from social media such as tweets, detecting disasters is highly relevant since it helps with essential tasks like directing emergency response or managing crisis. For online articles, ED is commonly used for business-related tasks such as business intelligence. In this context, sequences of text are used to provide information for companies to learn of their industrial landscapes and improve their business competitiveness. For example, the sentence *"The housing meltdown is reducing residual softwood woodchip supply and putting some integrated pulp/lumber companies in North America in extreme financial jeopardy."* may help integrated pulp/lumber companies in different parts of the globe plan for a similar situation. Business events also appeal to investors who make their decisions based on reported factors like brand image, corporate stand on environmental or societal issues.

2.4.4 News ED for forest industry

Despite various studies on ED, there has not been any work on identifying events from forest industry news. Arguably, one can apply an ED algorithm for business events to forest industry. Such application would allow for the detection of shared business events such as ‘sale increased’ and ‘stock upgrade’. However, the forest industry has many unique events that signal essential information such as ‘pulp demand’ or ‘paper production’. For example, the ‘pulp demand increased’ event can lead to an increase in sales and possibly an increase in pulp price. An ED algorithm for general business news might not recognize those unique events, leading to erroneous classification. Contrarywise, an ED algorithm tailored to the forest industry should be able to recognize both those unique events and the shared business events. Therefore, an ED algorithm for the forest industry should be applicable to general business news.

The lack of an ED algorithm for the forest industry stems from the lack of an usable dataset. Based on the type of ED task, there are a number of suitable datasets. Two of the more commonly used datasets are the ACE 2005 English corpus and the TAC KBP 2015 data (Ghaeini et al., 2016; Nguyen and Grishman, 2016; Orr et al., 2018; Xiang and Wang, 2019). Other corpora include the Reuter dataset (Mohamad et al., 2010). In a few cases, the working dataset was created by scraping news sites and manual labeling (Wunderwald, 2011; Jacobs et al., 2018). This manual labeling approach is the hardest since it requires a large amount of labor to go through and annotate every sentence. However, it works well for studies with specific topics that have seen little to no research before. Another approach to dataset annotation is topic modeling. Referred to as automatic annotation, topic modeling can be carried out by performing clustering on the text. The resulting topics/clusters can then be used as labels to train a classifier. The meaning of each cluster is interpreted based on the contents of its members. This alternative approach demands less time and labor but is stochastic in nature. Depending on randomization, algorithm, and hyperparameters choices, cluster memberships can vary with each run.

The choices in dataset annotation present the second design decision of this thesis. This thesis will create the proposed dataset by manual labeling. Due to the stochastic nature of various clustering algorithms, topic modeling does not contribute to the creation of a robust dataset. Manual labeling leads to more labor, but the resulting dataset will be appropriate for establishing a forest industry ED algorithm.

3 Data and Methods

This chapter describes the development process and the relevant design decision in constructing the dataset used for model training. Due to its importance as one of the research questions, the data creation process is examined in detail. Method-related issues like computation resources, choice of DL libraries, and baseline models are also discussed.

3.1 Data

3.1.1 Data Requirement

This section discusses the data's origin and pre-processing steps. All the decisions mentioned here were made by the researchers from Aalto University's Information and Service Management Department, who scraped and pre-processed the raw data.

The news data came from two sources: Reuters and RISI. The scraped articles were broken down into sentences. The datasets consist of rows of sentence id, label(s), and the sentence itself. The pre-processed datasets were provided as separate comma-separated values files.

- Reuters: consists of 10 years' worth of new articles from 2003 to 2012. There are approximately 18.8 million articles that include various topics ranging from sport to general news. The following keywords were used to filter out non-forest-related articles: 'forest', 'trees', 'timber', 'biomass', 'woodland', and 'tree farm'. If an article has one of the above keywords in its first 300 characters, it is considered forest-related. After filtering, the total number of articles is 25,188. Despite filtering, this dataset contains many business-related sentences. Non-business articles such as political, sport, and disaster pieces are also present in the given data.
- RISI: a firm in Finland provided the articles to further research in the forest industry. The dataset consists of roughly 19,000 articles that focus mainly on forest-related industries.

An event extraction framework called Odin (Open Domain INformer) was used to annotate the datasets. Developed by the Computational Language Understanding Lab at the University of Arizona as a "*standard way to express rules*", Odin is a rule-based event extractor that is simple and robust (Valenzuela-Escárcega et al., 2015). Similar to other rule-based event extractors, Odin uses different sets of rules to extract events and entities. To

apply its rules to a body of text, Odin goes through each sentence and triggers related events if there are matched criteria. Odin's rules are comparable to regular expressions, and there are thousands of such rules. Odin also allows for the mixing of syntactic patterns and different token-based patterns.

For the given datasets, a custom ruleset was used with Odin to create labels for every sentence. In case of no matched rule, a dummy label 'non-forest' was created to notify that the sentence does not relate to the forest industry. This dummy label denotes sentences mentioning time, location, credit, external link, person, governmental information, or even political news.

2_2	Price_target_raised	Chilean producer Arauco informed Chinese customers that its April prices would increase $ 10 / tonne .
2_3	Price_target	That would bring the firms bleached radiata pine price to $ 480-490 / tonne .

Figure 14. Samples consist of an id, a label, and the base sentence.

20060106180023nN06374742_1	non-forest	Lightning bolt kills five children in Cameroon

Figure 15. A sample of a sentence not related to the forest industry.

3.1.2 Data Processing

Despite several efforts to better rule-based approach to event extraction, it is not optimal (Li and Mao, 2019). This approach tends to have inaccuracies due to the complexity of how causal relations are expressed in natural language (Section 2.1). Similarly, the Odin pre-processed data from Aalto's ISM department is not error-free. Two of the most common errors in the provided datasets are misassigned labels and loss of information. With misassigned labels, a sentence is correctly flagged as related to the forest industry, but the label itself is incorrect. For example, the sentence "*OCC lost Euro 11.11/tonne, falling to Euro 39.45/tonne, while ONP/OMG plummeted Euro 14.57/tonne to Euro 89.87/tonne*" was flagged as 'Loss_reported' due to the word 'lost', even though it actually reports falling price. For this case, the 'Price_decreased' label would be more appropriate. On the other hand, loss of information occurs when a sentence gets assigned the dummy label 'non-forest' incorrectly. This misclassification is the most frequent error due to the limitation of the rule-based extractor. For example, the sentence '*Pulp producers look to kick off 2010 with hikes in Europe*' was labeled 'non-forest' despite the word 'pulp'. Contrastingly, human readers can understand that the sentence reports a planned price increase from wood pulp producers,

a part of the forest industry. These two errors occur in most of the sentences in the original datasets.

Following the second design decision in Section 2.4.4, manual checking of the sentences and their labels has been carried out. As the name suggests, manual checking involves going through every sample in the datasets and correcting the labels as necessary. Manually changing the datasets' labels is a time-consuming and labor-intensive endeavor. Nevertheless, it is necessary for creating a robust dataset to train a news ED algorithm. With manual labeling, there would be human error. However, human error is acceptable in comparison to the errors resulted from a purely rule-based approach.

The Reuters dataset has around 65,000 sentences, while the RISI dataset has more than 69,000 sentences. Both datasets exhibit a high level of replication, with some sentences appearing twice or even more times. This repetition dramatically inflates the size of the datasets. Once duplicated sentences are removed in the manual labeling process, the dataset size is 97,107. The models to be developed will be trained and verified with this processed dataset.

101_0	non-forest	Asian bleached pulp prices up but USK drops .
101_0	Price_increased|Price_decreased	Asian bleached pulp prices up but USK drop

Figure 16. An example of a mislabeled sentence and its newly assigned labels after manual checking.

As seen in Figure 16, a single text can have more than one correct label. Having more than one label represents a multi-label text classification problem instead of a multi-class text classification problem. With multi-class classification, each document is assigned to one label only, and labels are mutually exclusive. With multi-label classification, each document can have multiple labels. This classification type entails a different approach to choosing the model's structure and will be mentioned in the Experiment and Results chapter (Section 4.1.2).

After manual labeling, there are 152 different labels, which can be grouped into 46 different groups. The labels are used for model training, whereas the groups are for visualization purposes. The complete list of groups and labels is presented in appendix A. Based on the processed data, the minimum number of labels a sentence can have is one, and the maximum is seven. The frequency of per sentence labels is presented in Figure 17.

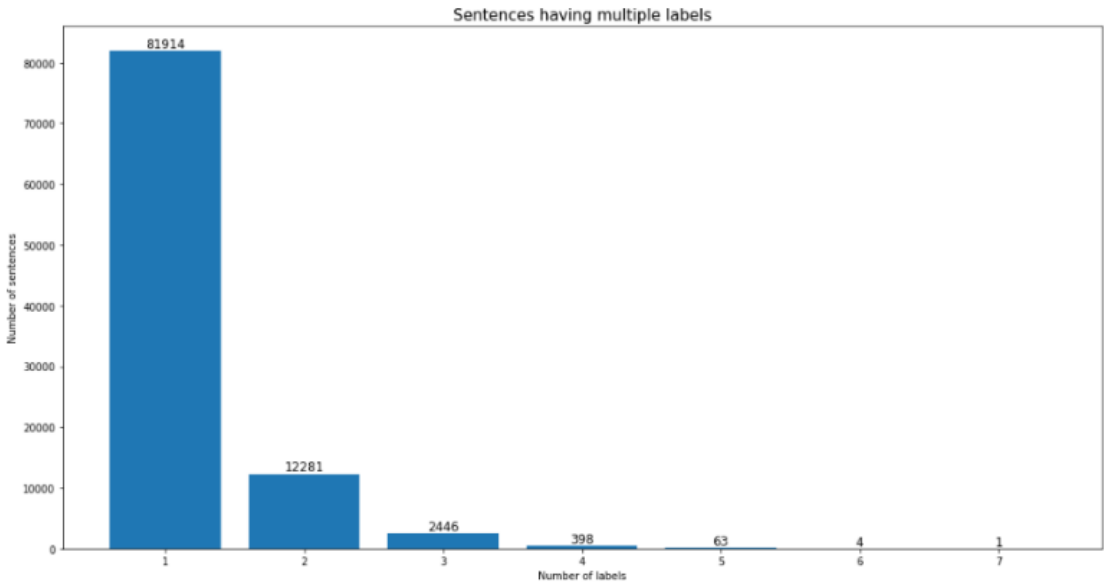

Figure 17. The number of labels for each sentence

The annotated dataset contains a considerable number of non-business and non-forest sentences in various topics such as political, sport, and disaster news. This variety suggests that the existing dataset can be used to detect more than just business and forest industry news, given the correct labels.

The variety of non-business and non-forest sentences leads to the third design decision of this thesis. Due to this thesis's scope, all non-business and non-forest sentences will be given an identical label, 'Other'. This labeling scheme reduces the complexity associated with having more labels. The additional complexity is neither required nor desired since it does not add extra benefit to the stated goals.

As an implication of the third design decision, the number of sentences under the 'Other' label dwarfs the rest of the dataset. While most labels have less than 2,000 sentences, the 'Other' label has more than 20,000 sentences. At another extreme, there are labels with less than ten sentences. This considerable difference in membership leads to a severe class imbalance problem. The mentioned disparity is reflected in the count of label groups in Figure 18.

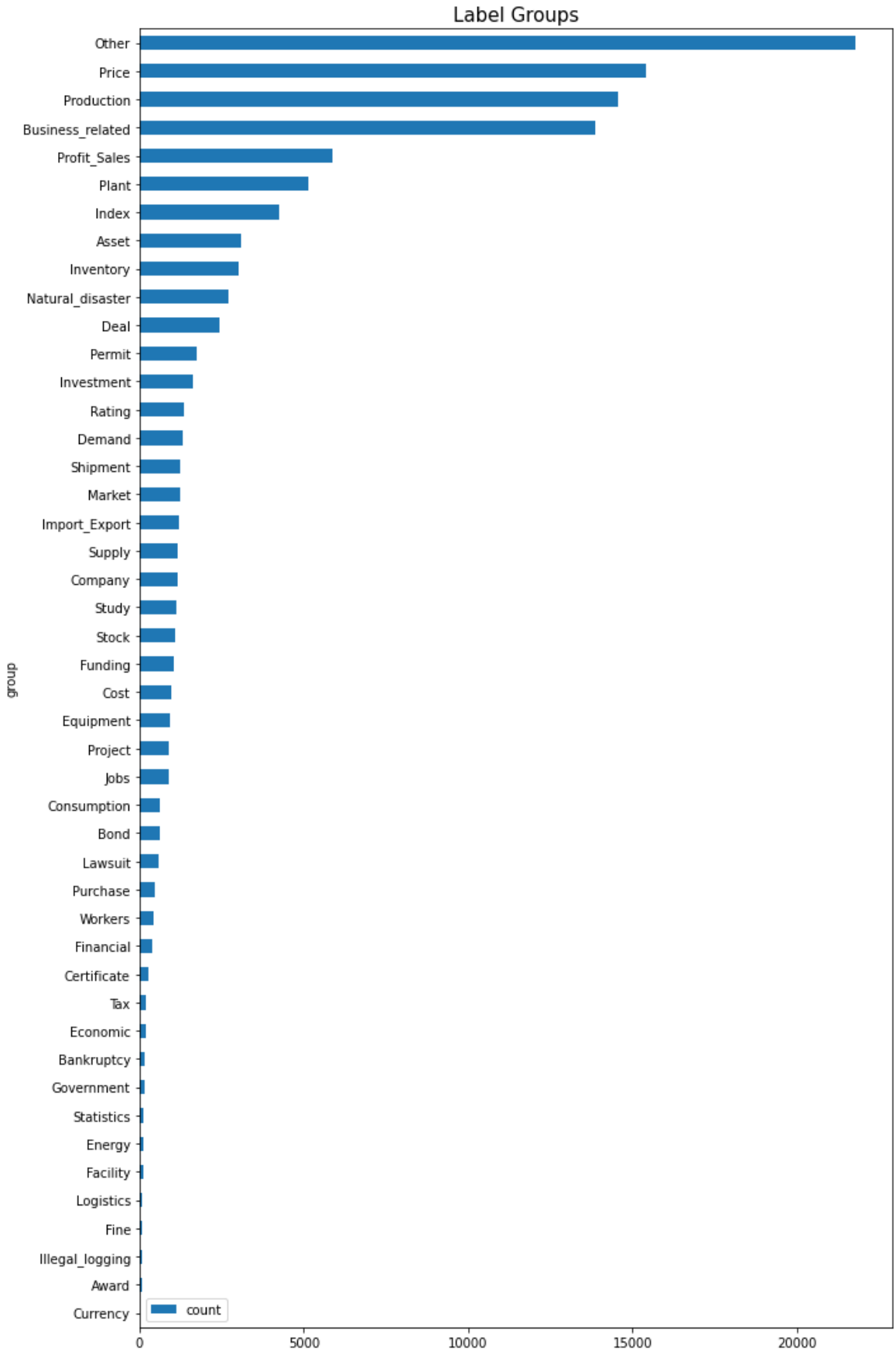

Figure 18. The number of sentences for each label group

The dataset consists entirely of texts. To create valid inputs for ML, those texts are transformed into a computer-readable data type. In this thesis, two approaches are used to create word representations of text data. These approaches correspond to the ideas introduced in Section 2.3, which are TF-IDF vectors and contextualized word embeddings. Non-contextual word embeddings are not considered due to their lowered performance compared to contextualized word embeddings. The first approach uses TF-IDF, and the resulting vectors are used to train baseline models (Section 3.2.2). Before TF-IDF vectors can be created, the dataset underwent several pre-processing steps such as tokenization, lemmatization, and stop words removal (Section 2.3). Afterward, the data is transformed into TF-IDF vectors using scikit-learn's TfidfVectorizer(). This results in a row vector (1 x 24,318) for each sentence, with 24,318 represents the number of unique terms in the dataset. In the second approach, contextualized word embedding vectors are created using BERT's tokenizer. BERT's tokenizer is a part of BERT that handles the encoding of raw data into a BERT-friendly format. For each sentence, the returned embedding is a row vector with the shape (1 x 1,024). The length of the returning vector depends on BERT's architecture. Since BERT large is chosen as the base model for this thesis (Section 4.1.1), its output is a vector of length 1,024. There is no other processing step required for this approach.

The last part of data processing is to create labels in a format understandable to ML models. Strings of labels are transformed into a matrix of 0 and 1 with scikit-learn's MultiLabelBinarizer(). The resulting matrix's rows match the numbers of sentences in the dataset, and its columns represent all existing labels: (97,107 x 152). For each sentence, its final label is a row vector of length 152. If some labels exist for a sentence, the corresponding columns will have a one while the rest will be zero.

3.1.3 Data Visualization

Visualization of text data can be complex due to high dimensionality; however, it is crucial to understand the data better.

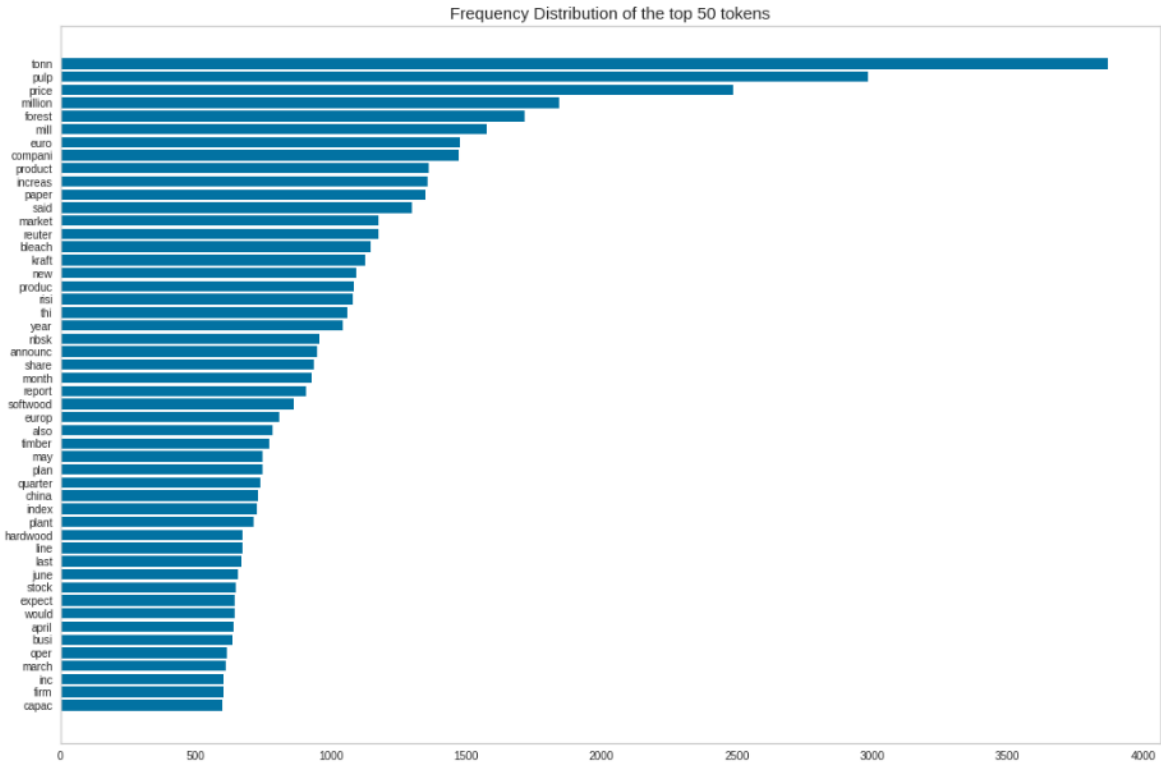

Figure 19. The top 50 tokens

Figure 19 shows the most common tokens in the dataset. Tokens are the words that have been pre-processed (tokenized, lemmatized, and stop words removed) (Section 2.3). With tokens like ‘pulp’ and ‘forest’ being in the top 5, the dataset is clearly related to the forest industry. Other tokens like ‘tonn’, ‘million’, and ‘price’ represent the dataset's business side.

Other visualizations are created with the help of dimensionality reduction (DR) techniques. DR is the transformation of data from a high-dimensional space into meaningful representations in a reduced dimensionality (Van Der Maaten et al., 2009). DR is common in fields involving a large number of variables, such as NLP. Two employed DR methods are mentioned below. The first method is chosen due to its popularity, and the second is considered an improvement over the first. Both are applied with their default settings.

- T-distributed Stochastic Neighbor Embedding (t-SNE): Introduced by Maaten and Hinton (2008), t-SNE is a non-linear dimensionality reduction method that creates two probability distributions. The first one represents the similarity of data points in high-dimension, while the second one captures the same information from the low-dimensional map. To maximize the probability of similarity between the original

objects and their representations, t-SNE tries to minimalize the Kullback-Leibler divergence (Kullback and Leibler, 1951) of the constructed distributions.

- Uniform Manifold Approximation and Projection (UMAP): Introduced by McInnes et al. (2018), UMAP is another non-linear dimensionality reduction method. Like t-SNE, UMAP makes use of data representation in both higher and lower dimensions. The high-dimensional graph represents a weighted graph of points. The points within a certain radius connect to form clusters. The bigger the radius, the smaller the likelihood of connection. This arrangement ensures that the local structure is preserved in balance with the global structure (Coenen and Pearce, n.d.). With a completed high-dimensional map, UMAP adjusts the low-dimensional parallel to be as similar as possible. This process is done in ways similar to t-SNE's. Performance-wise, UMAP is faster than t-SNE.

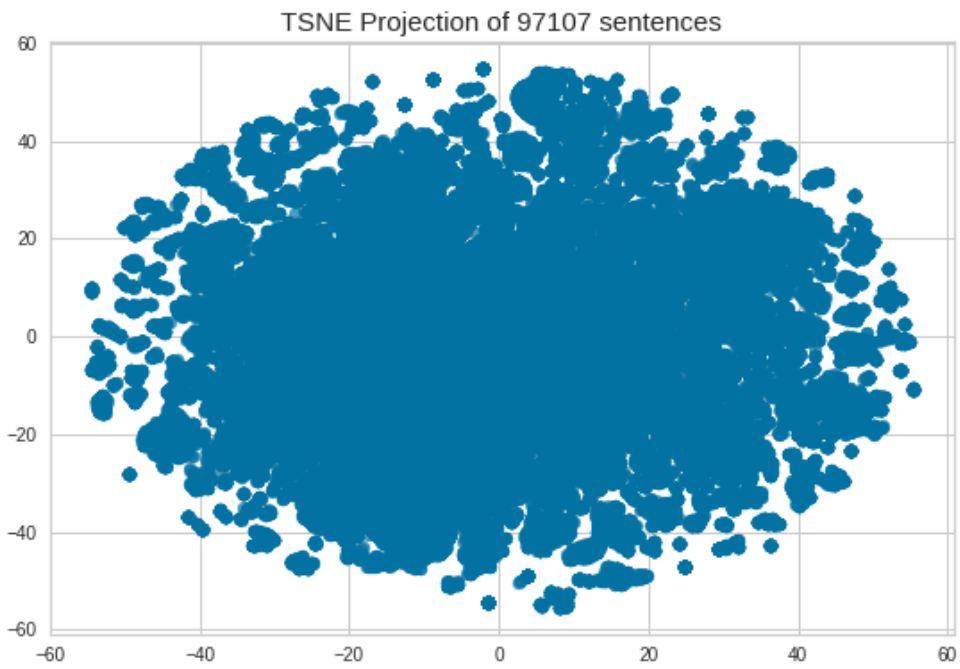

Figure 20. t-SNE visualization of the data

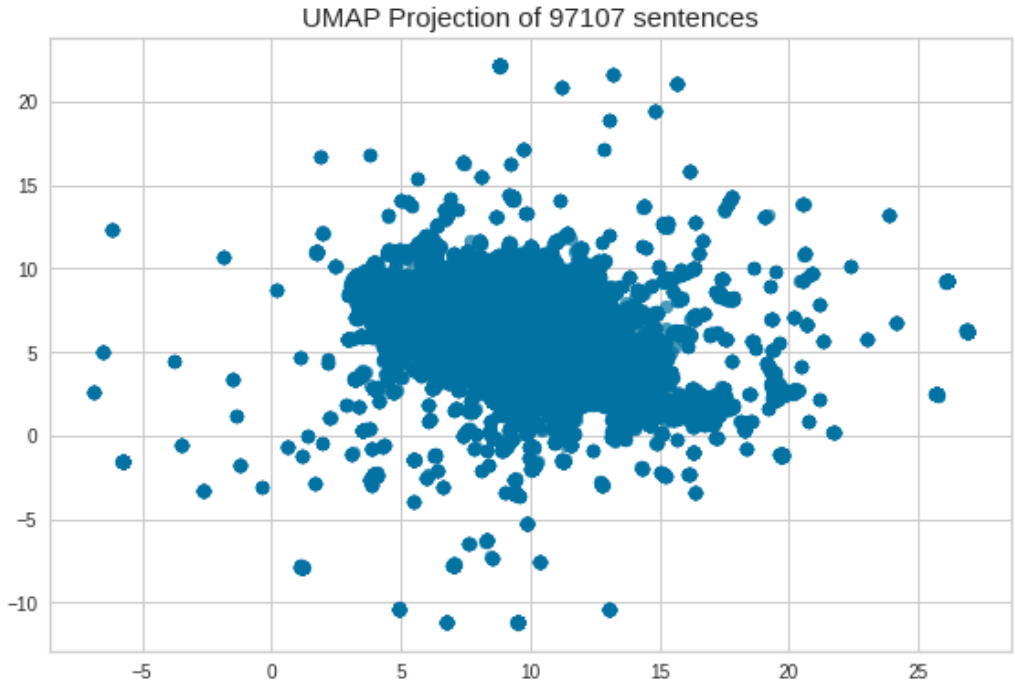

Figure 21. UMAP visualization of the data

With the t-SNE graph, there is a subtle hint that the datasets have some sentences that are not the same as the rest. Most of the data clusters together in the center and are surrounded by groups of smaller clusters. It is hypothesized that the centerpiece stands for forest or business-related sentences, while the smaller pieces represent sentences that are not. With UMAP, the same information is displayed more straightforwardly. The data points that semantically differ from the majority lie on the outlying parts, far from the center.

3.2 Methods

This thesis compares BERT, LSTM, and GRU models on the processed datasets. The scripting language is Python 3. Due to its popularity with DL, Python has many good frameworks to build and implement DL models. Among those frameworks, Google's TensorFlow (TF) is one of the more popular ones. Its popularity was boosted partly by Keras. Keras acts as an interface for TF library and focuses on ease of use. With the release of TF 2.0, TF came up with a strong Keras integration and an intuitive high-level API called tf.keras. This API combines the benefits of Keras and TF's low-level capabilities, such as support for eager execution. There are simplicity and efficiency benefits by adopting Keras and tf.keras in this thesis. This thesis also makes frequent use of scikit-learn, a machine

learning library that provides varying tools for data pre-processing, model fitting, selection, and evaluation.

For computation resources, training is executed on Google Colab. Google Colab provides a notebook environment that allows users to train various ML and DL models on Central Processing Units (CPUs), Graphics Processing Units (GPUs), or Tensor Processing Units (TPUs). With a sufficiently large batch size (Section 2.2.7), TPU gives better performance and shorter training time than GPU (Wang et al., 2019). Since TPU offers performance and time-saving benefits over GPU when used to train large models, Google Colab with TPU runtime has been chosen as the coding environment.

3.2.1 Choosing the metrics

Metrics in ML is the measure of model performance, and their selection can significantly affect the model building process. Depend on the type of task, optimizing the wrong metrics can lead to a not-so-helpful model. For example, for classification tasks, metrics like Mean Squared Error or Mean Absolute Error are unproductive since they are intended for regression. Instead, confusion-matrix-based metrics should be used.

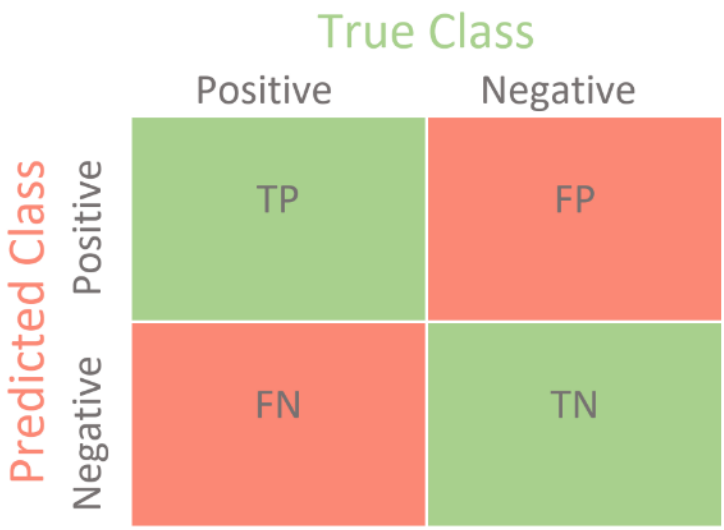

Figure 22. Confusion matrix example with two classes

A confusion matrix is used to summarize the performance of a classifier dealing with labeled data. It shows the correct and incorrect predictions for each class. It comprises true positives (TPs), false positives (FPs), false negatives (FNs), and true negatives (TNs). In the

case of binary classification, the classes are called ‘positive’ and ‘negative’. In multi-class classification, for each class, the class in question is called ‘positive’ while the rest is called ‘negative’. ‘True’ and ‘false’ are used to denote the correctness of the predictions. From those four base elements, a range of classification metrics can be constructed. Accuracy is the fraction of true predictions over all predictions. Precision is the fraction of correct positive predictions over all positive predictions. Sensitivity or recall is the fraction of correct positive predictions over all positive. F1 score is the geometric mean of precision and recall.

$$accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{22}$$

$$precision = \frac{TP}{(TP + FP)} \tag{23}$$

$$recall = \frac{TP}{(TP + FN)} \tag{24}$$

$$F - 1 = 2\frac{percision * recall}{percision + recall} = \frac{2TP}{2TP + FN + FP} \tag{25}$$

To evaluate classifiers graphically, the receiver operating characteristics (ROC) curve can be used. ROC is a plot showing the relationship between the true positive rate (TPR) and the false positive rate (FPR) at different thresholds. TPR is a different name for recall or sensitivity, while FPR denotes how often the model incorrectly predicts a negative as a positive.

$$\text{FPR} = \frac{FP}{(TN + FP)} \tag{26}$$

In a multi-class setting, each class has a score for each metric. The overall classifier metrics are derived by averaging the class scores. Two standard averaging measures are macro-average and micro-average. Macro-average assigns equal weights to every class and then takes the arithmetic mean of the scores. Micro-average assigns equal weights to every sample and then uses all classes' aggregation to compute the average. A key distinction between the two measures is that macro-averaging is per class average, whereas micro-averaging is per sample average. An offset of macro-averaging is weighted averaging, which considers the number of actual instances for each label when computing the mean score.

Theoretically, weighted averaging can address the issue of class imbalance. This benefit must be considered against the potential for skewed results due to majority classes possessing larger weights.

Another metric to use is the area under ROC curve (AUCROC). AUCROC calculates the two-dimensional area beneath the ROC curve in a fashion similar to integral calculus. AUCROC provides an aggregate measure representing how well the classifier can separate different classes at various threshold settings. The higher the AUCROC, the better. The highest value of AUCROC is 1, and the worst is 0.

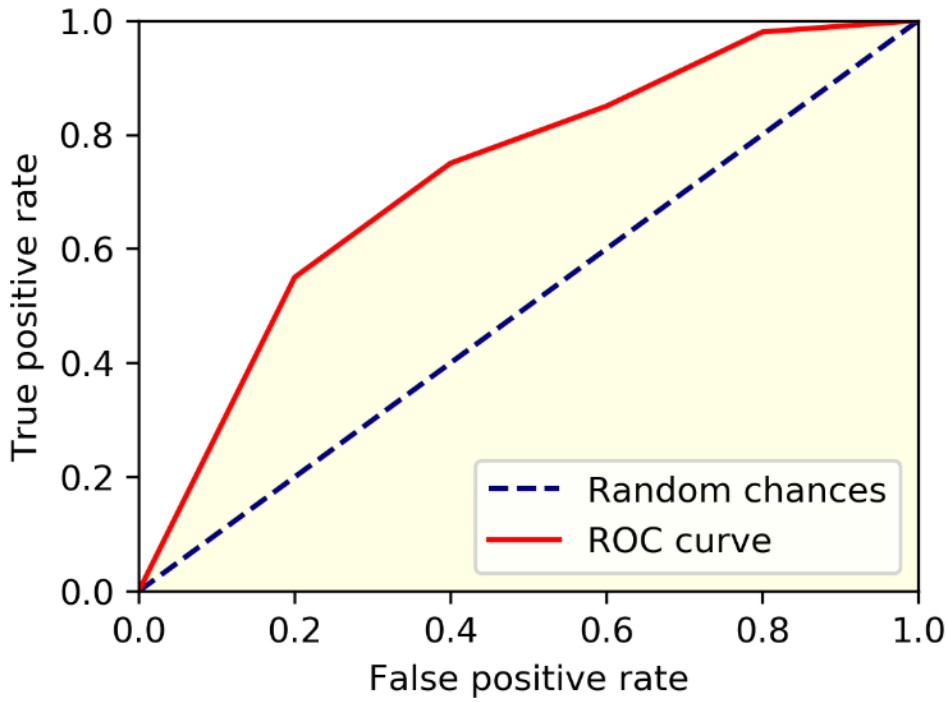

Figure 23. A ROC curve example with AUCROC is the area in shade

In a multi-class setting, the average AUCROC can be calculated from individual AUCROCs:

$$\text{AUCROC} = \frac{2}{C(C-1)} \sum_{i=1}^{C} AUC_i \tag{27}$$

where C is the number of classes.

For this thesis, AUCROC, F1-Micro, F1-Weighted are chosen in hope of avoiding issues from the imbalanced dataset (Section 3.1.2).

3.2.2 Baseline models

Benchmark is often used to gauge the performance of a trained model. Usually selected as a baseline of some metrics, the purpose of benchmarking is to determine if the trained model is performing better than a non-trained model. If the trained model performs at around or less than the selected baseline, that trained model is under-performing and should not be used for evaluation or prediction purposes.

For many studies, reported results from previous studies could be used as benchmarks. Since there has not been any ED study for the forest industry, new benchmarks must be created. For binary classification tasks, an acceptable benchmark is 0.5, representing choosing an answer randomly from two choices. In this thesis, there are 152 classes, some of which can co-exist in the same sentence. Setting up fixed values that represent random guessing, in this case, is overly complex. The next option is to use simple linear models to set up baselines. Benchmarking with linear models can show the improvement to be gained from using more complex models and extra hardware computations. To generate the needed baselines, five different linear models are trained. Based on chosen metrics (Section 3.2.1), the two best performing linear models are Support Vector Machine (SVM) and Bagging.

- SVM: introduced by Boser et al. (1992), SVM is among the more popular binary classifiers. SVM creates a representation of the samples as points in space and then tries to divide those points into two different groups with a gap as large as possible. To predict, SVM maps new samples onto the created sample space and labels them based on the mapped region.
- Bagging: a type of ensemble classifier introduced by Breiman (1996). The bagging algorithm creates many sub-models that are trained on random subsets of the training data. The predictions from those sub-models are then aggregated by either voting or averaging to give the final predictions. Bagging is often used to reduce the error that comes with data (variance).

The linear models use respective scikit-learn's implementations with default settings and are trained with the TF-IDF processed data. Since those linear models are initially meant for binary classification tasks, they are warped in scikit-learn's OneVsRestClassifier() to

enable multi-label classification. One-vs-rest classifier is a strategy that fits one classifier per class. For each classifier, the class in question is fitted against all other classes.

The benchmark scores are calculated using the test dataset and can be found in Table 1. As a by-product of this benchmark study, the feasibility of an ML-based ED algorithm applicable to both general business news and forestry industry news is proven.

Table 1: Benchmark scores using linear models

	Metrics	**Benchmark score**
SVM	AUCROC	0.81
	F1-Micro	0.70
	F1-Weighted	0.68
Bagging	AUCROC	0.79
	F1-Micro	0.68
	F1-Weighted	0.66

4 Experiments and Results

This chapter discusses the results of applying different DL models to the processed dataset and comparing their performances based on the metrics introduced in Section 3.2.1. The structure of the BERT-based proposed solution and the steps taken to tailor it to the forest industry are examined. Besides the proposed architecture, the following DL models are included to provide a performance comparison: LSTM and GRU.

4.1 The proposed Forest Industry ED Scheme

4.1.1 BERT architecture

A robust model is necessary to address the complexity of the task introduced in this thesis. The processed dataset has 97,107 sentences and 152 non-mutually exclusive labels. This complexity is partly reflected in the benchmarking study's limitations to classify the sentences correctly (Section 3.2.2). The robustness requirement leads to the choice of BERT large, uncased with whole word masking, as the base model. This BERT variant is the largest from Google's BERT implementations and also the best performing. The base model has 24 layers (Transformer blocks), 1,024 hidden dim (dimensionality of the encoder layers and the pooler layer), 16 attention heads, and 340 million parameters. The uncased model removes accent markers and formats words into lower case. Whole word masking mitigates the drawbacks of masking partial tokens in pre-training BERT, improving its performance (Cui et al., 2019).

Due to the size of the model, its structure plot is not presented here. The complete model is available on request.

4.1.2 Fine-tuning Process

Due to the stated complexity, fine-tuning is necessary for this classification task. Similar to the training process, several hyperparameters can be chosen during fine-tuning. Those hyperparameters play a crucial role in the performance of ML models. Since BERT is a pre-trained model, the number of configurable options is limited. Sequence length, batch size, learning rate, and epoch are the tunable choices.

Sequence length is a hyperparameter that determines the maximum length of input tokens. The default value from BERT is 512. This value means any sequence longer than 512 tokens will be truncated. The default truncation method is using the first 512 tokens.

Other truncation methods include utilizing the last 512 tokens or a mix between the first and the last tokens. The consequence of having a considerable sequence length along with short sentences is input sequences being filled with padding tokens. Those padding-filled inputs are not helpful to the learning process and can potentially slow down training.

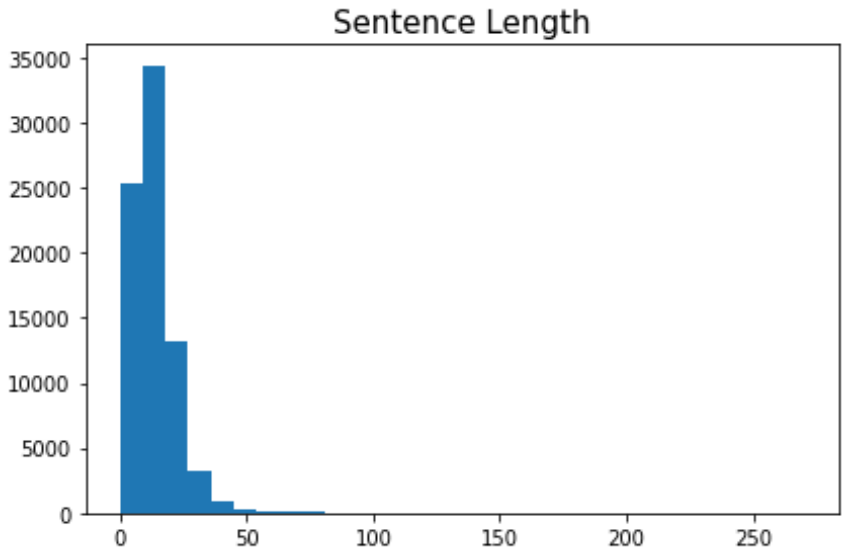

Figure 24. The lengths of the sentences in the dataset

In the available dataset, most sentences are short. From Figure 24, it can be seen that the maximum length is around 270 words. However, the bulk of the sentences have a length of 50 words or less. It can be concluded that an appropriate range of maximum sequence lengths should be between 50 and 270. BERT's performance will be examined with three distinct values from this range, which are 50, 100, and 270.

Batch size is another configurable hyperparameter. Batch size controls the number of training samples used to calculate the error gradient before the model updates its weights (Section 2.2.7). Larger batch sizes allow for faster training since they lead to fewer training steps per training iteration. If batch size is large enough, it will act as a representative sample of the training data and provide a stable estimate of the dataset's gradient. A disadvantage of big batch sizes is that they tend to get stuck in local minima, resulting in lower performance. By contrast, smaller batch sizes prolong training time while providing several benefits. Smaller batch sizes are noisy, which can be helpful in cases of local minima. Small batch sizes also lead to lower generation errors (Kandel and Castelli, 2020). As a side benefit, small batch sizes make it easier to fit a training data batch in the computer’s memory, avoiding out-of-memory error. Smaller batch sizes of 32 (Bengio, 2012; Masters and Luschi,

2018), 64, and 128 have been chosen to train the models. The batch size of 16 is not included due to slow training time resulting in incomplete training in a single Google Colab session.

Learning rate is the next tunable hyperparameter, and it controls how fast the model learns. More precisely, it controls how significant should the weights update be at the end of each training batch. Learning rate plays a vital role in the training process, as mentioned in the Optimization section (Section 2.2.7). According to Sun et al. (2019), a good strategy for fine-tuning BERT is to use Adam with a learning rate of 2e-5 and a warm-up ratio of 0.1. RAdam has been selected to bypass the warm-up heuristic.

Epoch, the last adjustable hyperparameter, is the number of times the model goes through the entire dataset in both forward and backward passes (Section 2.2.6). With too few epochs, there is a possibility that the model has not updated its weights sufficiently. This insufficient update will lead to underfitting. With too many epochs, the model might memorize the training dataset. This memorization, on the other hand, will cause overfitting. Both cases lead to poor performance and are not desirable. Sun et al. (2019) recommend four epochs since BERT is a substantial model and can easily overfit if trained on small datasets for too long. Based on experiments, twenty epochs have been chosen in this thesis since the validation accuracy only stops increasing around the 20^{th} epoch (Section 4.3).

For fine-tuning, the output of BERT's last layer (the NSP Dense layer) is extracted and inputted into a new dense layer. This new dense layer will perform the intended classification. The classifier has an output of 152 units, which is equivalent to the number of labels in the dataset. As noted in Section 2.2.1, sigmoid is often used for binary classification problems, whereas softmax is preferred for multi-class classification problems. Since softmax outputs only one value, it is not a suitable activation function for multi-label classification tasks (Section 3.1.2). For this type of problem, a sigmoid activation function combined with a binary cross-entropy loss function (Equation 5) is favored (Liu et al., 2017). With this combination, the output of each sample is an array of probabilities corresponding to different labels. The higher a probability is, the more likely the sample in question has the matching label. Once the sentences are classified, a threshold (usually 0.5) can be applied to transform the resulting arrays of probabilities into arrays of zeros and ones. For each transformed array, having a one at some indexes means the associated sentence has the labels at those specific indexes.

4.2 LSTM and GRU structures

Even though BERT is chosen as a basis for the proposed ED algorithm, two other DL models, namely LSTM and GRU, are trained to provide performance comparisons. Unlike the proposed model, these recurrent networks are trained from scratch. As parts of the recurrent network family (Section 2.2), they both share a similar structure. This similarity in structure allows for faster prototyping. They only differ from each other in their respective recurrent layers. Those recurrent layers are wrapped in bidirectional layers, allowing the networks to examine sequences from both directions, potentially increasing their performances. There is no difference between the two for non-recurrent layers, e.g., Input, Embedding, Dropout, and the final classification layer. The number of epochs is set to 200, with early stopping implemented. Early stopping is the most commonly used form of regularization to keep models from overfitting (Goodfellow et al., 2016). Both models are trained with Adam optimizer using a 1e-3 learning rate. The loss function for both is binary cross-entropy. Below are short descriptions and settings of the layers. Default values are used for all settings that are not listed.

- Input: Take in the input and return a tensor usable by the model. In this case, the input layer has the shape of (270,) with 270 is the maximum sentence length.
- Embedding: Transform received inputs into embeddings. This layer is Keras built-in solution for creating embeddings from encoded words. Its shape is (270, 200,). 270 is the maximum sentence length, and 200 is the dimensionality of the embeddings.
- Recurrent layer: Have either LSTM or GRU cells. These models use three recurrent layers with 512, 256, and 128 LSTM/GRU units each. Those units represent the hidden state's dimension. The first two recurrent layers have return_sequences set to True to output the entire hidden state sequence.
- BatchNormalization: Make the training of neural networks faster and more stable by re-centering and re-scaling the previous layer. Introduced by Ioffe and Szegedy (2015), batch normalization is thought to increase the model's performance by either reducing internal covariate shift or smoothing the objective function. Internal covariate shift happens when the model's learning rate is affected by either parameter initialization or the differences in input distribution of layers. BatchNormalization layers are placed before every recurrent layer as well as the last layer.

- Dropout: Disables non-output neurons in the network randomly by setting their values to zero. Introduced by Srivastava et al. (2014), dropout is another regularization approach. Dropout prevents neurons from forming co-dependency and makes them handle increased or decreased responsibility from the inputs. The dropout rate is set to 50%. Dropout layers are placed right after the first two recurrent layers
- Classification layer: Return the outputs. This final layer dense layer uses a sigmoid activation and has an output shape of 152.

4.3 Experiments

The processed dataset of 97,107 samples is used to train different models and evaluate their results. The dataset is split according to the Pareto principle. Table 2 shows the splitting ratios.

Table 2: Data split ratio

Dataset	Split ratio (%)	Size
Train	80	77,685
Validation	10	9,711
Test	10	9,711

Training data is used to train the models, while validation data is used to examine their generalization capacity after each training epoch. After training, test data is used for prediction. For each model, its output is arrays of probabilities, with one array for each sample. A 0.5 threshold is applied to the resulting arrays to get the predicted labels. The predicted and actual labels are then inputted into AUCROC, F1-Micro, and F1-Weighted (Section 3.2.1) functions to get the models' performance in those metrics.

To provide a clear comparison between different hyperparameter choices for BERT, the experiments' results are grouped by sequence lengths. This grouping leads to three distinct model groups corresponding to the sequence lengths of 50, 100, and 270. Within each group, there are different BERT models with 32, 64, and 128 batch sizes. The combination of 128 for batch size and 270 for sequence length is not included since it leads to out-of-memory issues. Unlike BERT with different configurations, LSTM and GRU models are limited to one configuration each to enable active comparison. The results for LSTM and GRU are presented in a separate table. The tables are accompanied by charts

showing the Accuracy and Loss of the presented models. For a quick overview, a table of every experiment is shown at the end.

Table 3: Results on the combined dataset, BERT, sequence length of 50

No.	Model	Model Parameters	Recurrent Layers	lr	Optimizer	Sequence Length	Batch Size	AUCROC	F1-Micro	F1-Weig.	Fig.
1	BERT	334,824,600	NA	2e-5	RAdam	50	32	0.81	0.68	0.65	25
2	BERT	334,824,600	NA	2e-5	RAdam	50	64	0.81	0.7	0.66	26
3	BERT	334,824,600	NA	2e-5	RAdam	50	128	0.83	0.72	0.69	27

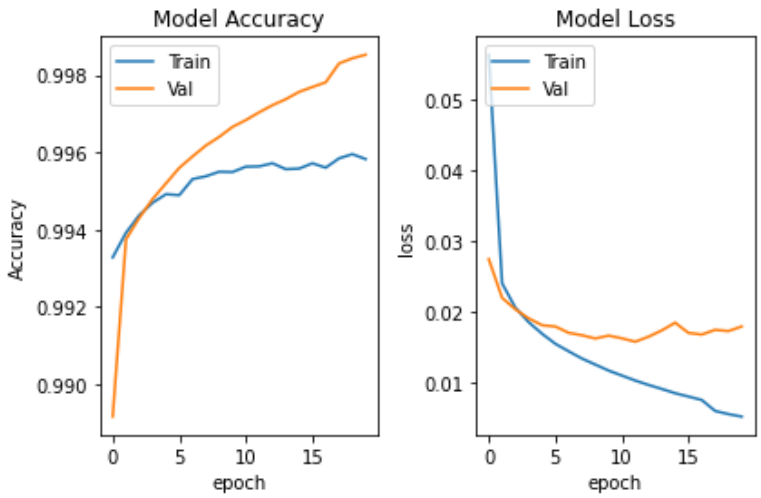

Figure 25. Training and validation history for BERT, experiment 1

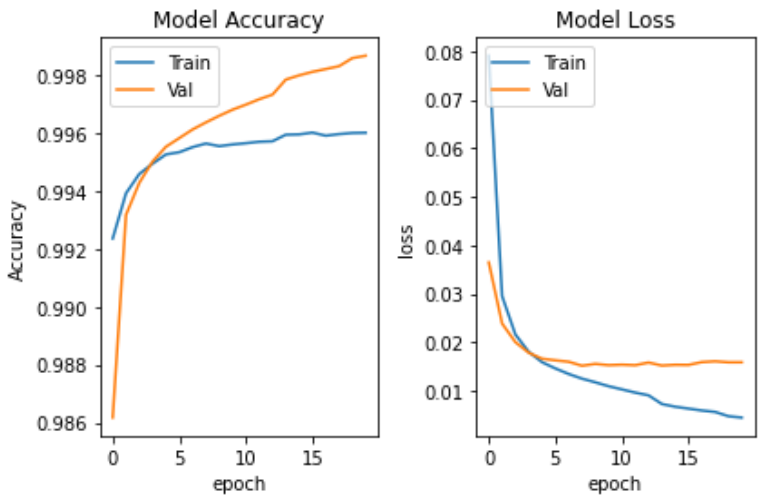

Figure 26. Training and validation history for BERT, experiment 2

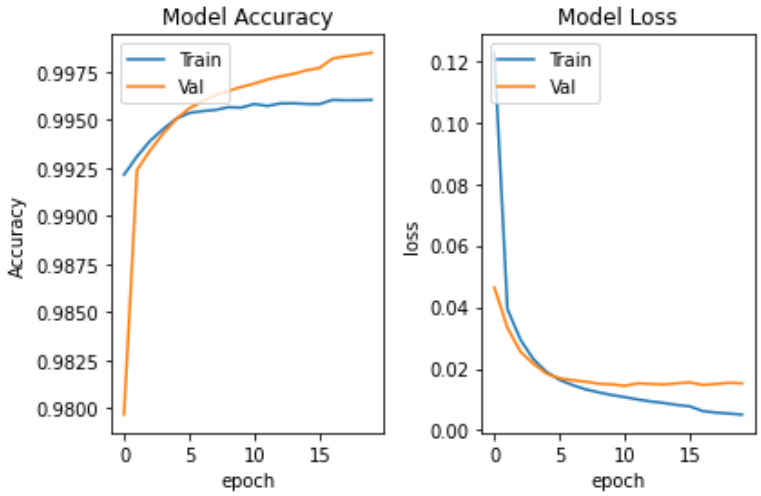

Figure 27. Training and validation history for BERT, experiment 3

Table 4: Results on the combined dataset, BERT, sequence length of 100

No.	Model	Model Parameters	Recurrent Layers	lr	Optimizer	Sequence Length	Batch Size	AUCROC	F1-Micro	F1-Weig.	Fig.
4	BERT	334,875,800	NA	2e-5	RAdam	100	32	0.8	0.69	0.64	28
5	BERT	334,875,800	NA	2e-5	RAdam	100	64	0.84	0.73	0.7	29
6	BERT	334,875,800	NA	2e-5	RAdam	100	128	0.84	0.73	0.71	30

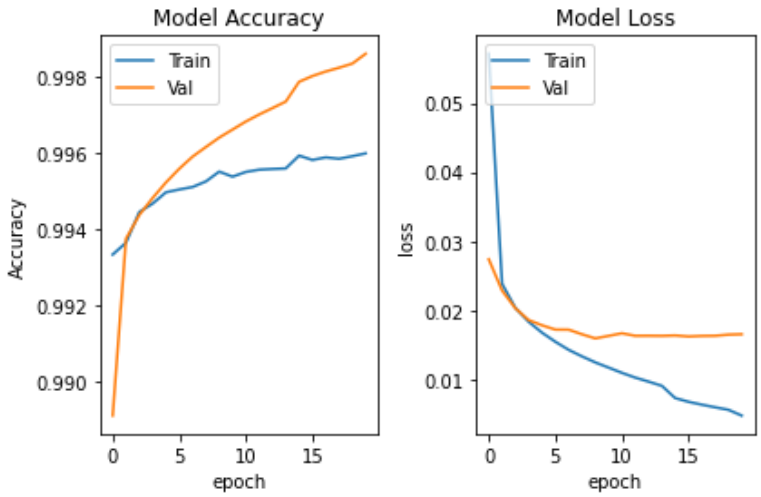

Figure 28. Training and validation history for BERT, experiment 4

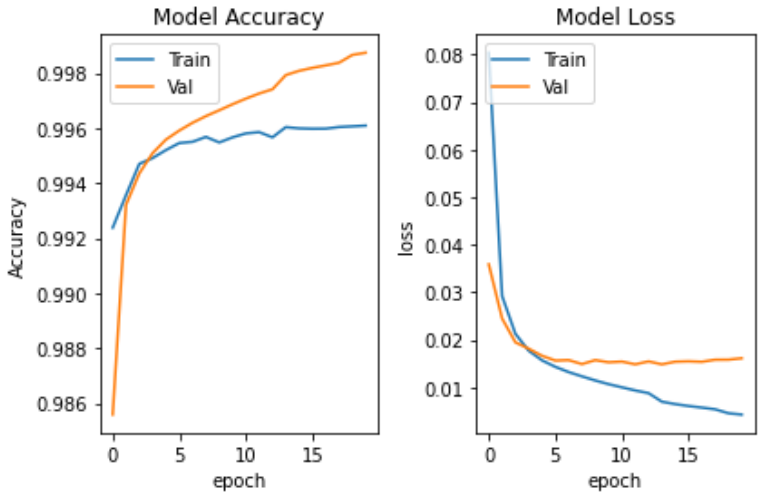

Figure 29. Training and validation history for BERT, experiment 5

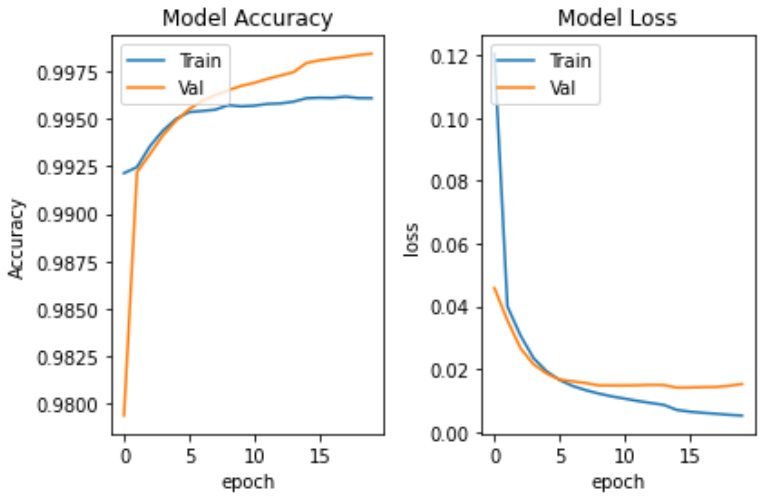

Figure 30. Training and validation history for BERT, experiment 6

Table 5: Results on the combined dataset, BERT, sequence length of 270

No.	Model	Model Parameters	Recurrent Layers	lr	Optimizer	Sequence Length	Batch Size	AUCROC	F1-Micro	F1-Weig.	Fig.
7	BERT	335,049,880	NA	2e-5	RAdam	270	32	0.83	0.72	0.7	31
8	BERT	335,049,880	NA	2e-5	RAdam	270	64	0.84	0.73	0.72	32

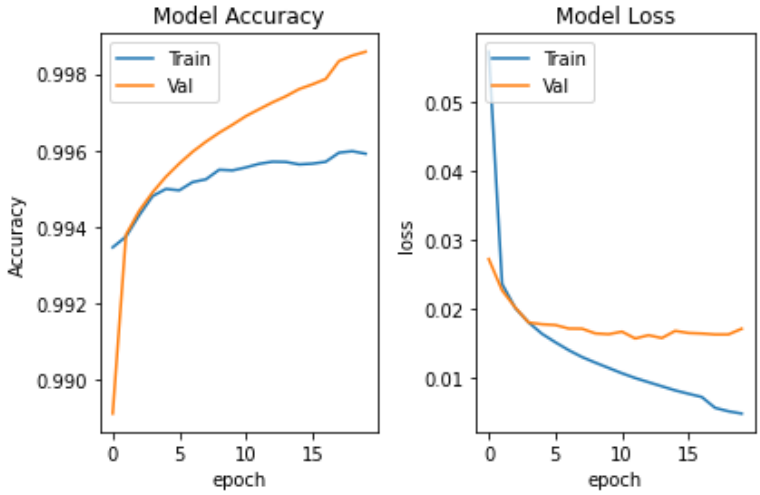

Figure 31. Training and validation history for BERT, experiment 7

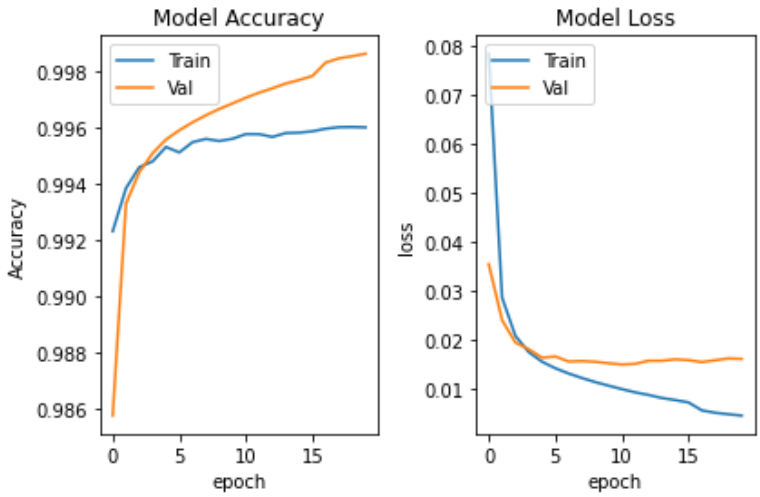

Figure 32. Training and validation history for BERT, experiment 8

Table 6: Results on the combined dataset, LSTM and GRU

No.	Model	Model Parameters	Recurrent Layers	lr	Optimizer	Sequence Length	Batch Size	AUCROC	F1-Micro	F1-Weig.	Fig.
9	LSTM	11,544,152	512 / 256 / 128	1e-3	Adam	NA	512	0.84	0.73	0.71	33
10	GRU	9,999,448	512 / 256 / 128	1e-3	Adam	NA	512	0.83	0.73	0.70	34

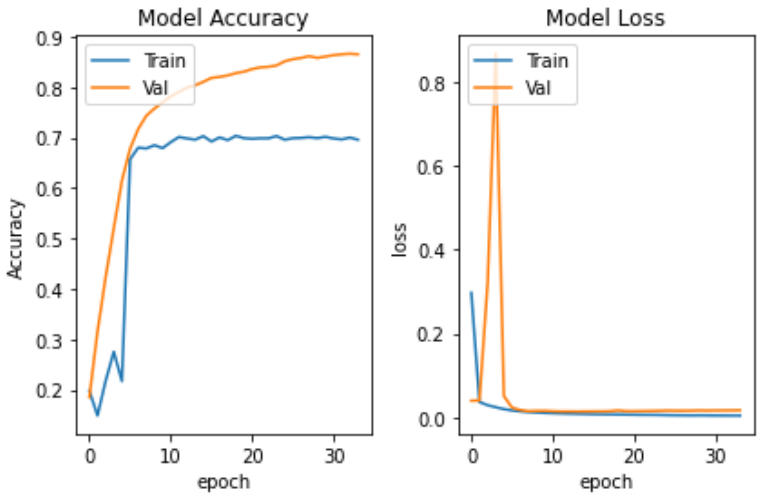

Figure 33. Training and validation history for LSTM, experiment 9

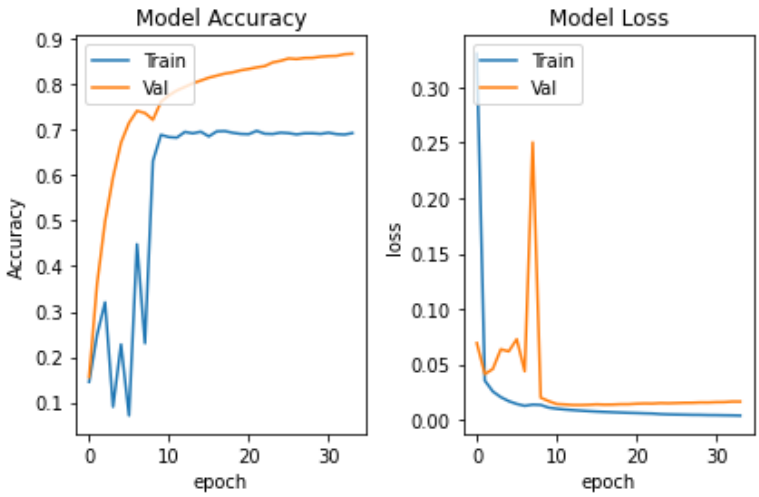

Figure 34. Training and validation history for GRU, experiment 10

Summary of experiments, with highlighted best performing alternatives:

Table 7: Results on the combined dataset, all

No.	Model	Model Parameters	Recurrent Layers	lr	Optimizer	Sequence Length	Batch Size	AUCROC	F1-Micro	F1-Weig.	Fig.
1	BERT	334,824,600	NA	2e-5	RAdam	50	32	0.81	0.68	0.65	25
2	BERT	334,824,600	NA	2e-5	RAdam	50	64	0.81	0.7	0.66	26
3	BERT	334,824,600	NA	2e-5	RAdam	50	128	0.83	0.72	0.69	27
4	BERT	334,875,800	NA	2e-5	RAdam	100	32	0.8	0.69	0.64	28
5	BERT	334,875,800	NA	2e-5	RAdam	100	64	0.84	0.73	0.7	29
6	**BERT**	**334,875,800**	**NA**	**2e-5**	**RAdam**	**100**	**128**	**0.84**	**0.73**	**0.71**	**30**
7	BERT	335,049,880	NA	2e-5	RAdam	270	32	0.83	0.72	0.7	31
8	**BERT**	**335,049,880**	**NA**	**2e-5**	**RAdam**	**270**	**64**	**0.84**	**0.73**	**0.72**	**32**
9	**LSTM**	**11,544,152**	**512 / 256 / 128**	**1e-3**	**Adam**	**NA**	**512**	**0.84**	**0.73**	**0.71**	**33**
10	GRU	9,999,448	512 / 256 / 128	1e-3	Adam	NA	512	0.83	0.73	0.70	34

The results show significant variation in the performance of BERT. It can be inferred that the choice of hyperparameters has a strong impact on the results. Less complex models (with smaller batch sizes or shorter sequence lengths) underperform more complex models. This performance gap can be partly attributed to the total number of labels in the dataset. For smaller batch sizes such as 32 or 64, the model can see only up to 64 samples before calculating error gradient for the weights update. As a result, when the model updates its weights, it has only seen a fraction of the total labels. This limited sampling size contributes to an overall lower performance. A similar limitation is apparent with sequence length choices, where smaller sequence lengths often lead to decreased performance. This lowered performance comes from the loss of information associated with truncated and discarded inputs.

From the model loss plots, the models exhibit signs of overfitting with increasing validation losses. It can be observed from the plots that validation losses stop decreasing around the 4^{th} epoch. This observation can be used to form an appropriate strategy to optimize the proposed model.

Both LSTM and GRU models post comparable performance to some BERT configurations despite having smaller structures. Although their number of epochs is set to

200, the recurrent models' training processes are stopped early at 50 epochs. This early stopping provides a benefit unseen in BERT experiments. The recurrent models do not exhibit any substandard performance issues associated with overfitting. For both LSTM and GRU, the validation losses blow up while the training accuracies fluctuate enormously during the first few epochs. They are the byproducts of batch normalization, which induce severe gradient explosion at initialization (Yang et al., 2019). The results also show that LSTM exhibits a better performance than GRU, confirming that LSTMs outperform GRUs on large datasets (Section 2.2.4).

Compared with the baselines presented in Section 3.2.2, the improvements in performance from using BERT are not substantial. Even though the BERT models have higher AUCROC scores than the baselines, their F1-Micro scores show only marginal increases. In some cases, BERT models have lower metrics scores than the baselines. This disparity reconfirms that hyperparameter choice has a significant influence on the training process.

Among the tested models, the best performing are:

- BERT with batch size of 128 and sequence length of 100 (AUCROC: 0.84, F1-Micro: 0.73, F1-Weighted: 0.71);
- BERT with batch size of 64 and sequence length of 270 (AUCROC: 0.84, F1-Micro: 0.73, F1-Weighted: 0.72);
- LSTM (AUCROC: 0.84, F1-Micro: 0.73, F1-Weighted: 0.71).

BERT performs worse than expected despite benefiting from a more robust structure. Furthermore, similar results across different types of models (linear, recurrent, and BERT) suggest that there is a reason that more powerful and complex models cannot capitalize on their increased complexity. The common denominator between those three model types is the dataset. After dataset processing (Section 3.1.2), no further step was taken to deal with the class imbalance issue seen in Figure 18. The rationale was that choosing the right metrics should be enough to limit the impacts of working with imbalanced data. Since the results are not as expected, the next section will discuss the issue of data imbalance in detail.

4.4 Class Imbalance and the proposed model

In ML classification tasks, one of the more significant issues is class imbalance. Class imbalance happens when a dataset has uneven distribution between multiple classes. Some classes may have a significantly higher number of samples (majority classes), while other classes have far fewer samples (minority classes). If such a dataset is used for training, the model will often bias toward the majority classes. This behavior is undesirable since the minority classes can contain valuable information and are often critical. Therefore, it is crucial to employ measures to achieve an acceptable model performance when working with imbalanced data.

Class imbalance is inherent in the processed dataset: the most prominent class has over 20,000 samples, whereas the smallest classes have around ten samples (Section 3.1.2). Since this problem could not be addressed by choosing suitable metrics alone, additional measures are necessary. A straightforward approach to balance the dataset is to resample it. There are two resampling options: undersampling the majority classes and oversampling the minority classes.

Undersampling removes samples from the majority classes. Undersampling can reduce performance by discarding important information from removed samples. In this particular case, undersampling the dataset would mean removing the majority of the 'Other' class, leaving it at around 2,000 samples (Section 3.1.2). However, due to the sizes of some minority classes, this resampling method would still not be enough to create an even distribution between classes and is not considered.

Oversampling duplicates samples from the minority classes. Oversampling can cause the model to overfit to rare samples, leading to an increase of false-positive/type I error in prediction. Type I error is highly undesirable in many classification cases, including detecting defects, frauds, or cancer, due to its catastrophic effect. With news ED, type I error will still result in misclassification but without the heavy impact associated with other tasks. Consequently, the mentioned concern is considered negligible in this thesis, and oversampling is chosen to tackle the issue of class imbalance.

scikit-learn's RandomOverSampler() is used to oversample the dataset. By employing this approach, the dataset explodes in size. From the original 97,107 samples, the resampled dataset has 3,313,024 samples. The split described in Table 2 is no longer valid

with the resampled dataset since it leads to out-of-memory errors during training. The revised splitting ratio is presented in Table 8.

Table 8: Resampled data split ratio

Dataset	Split ratio (%)	Size
Train	48	1,590,144
Validation	6	198,784
Test	46	1,524,096

The ratio choice here reflects computational constraints. 48% is the maximum amount of training data that can be used in a single Google Colab session. Despite being less than half of all the available data, the new training dataset is roughly twenty times the number of sentences from the original training data. 1.5 million is a sizeable number that will allow BERT to learn the underlying patterns from the data. For validation data, the 6% ratio is chosen for two reasons. It helps shorten the training time since the validation process is performed right after the training process and is counted toward training time. Having small validation data also allows for more extensive test data to examine the model's generalization capacity.

To ensure stratification, scikit-multilearn's iterative_train_test_split() is applied. Based on an algorithm proposed by Sechidis et al. (2011) called Iterative Stratification, iterative_train_test_split() splits a multi-label dataset by considering each label separately and return splits containing a balanced number of examples from each label. This splitting method provides balance across splits while retaining the uniqueness of each label set. This splitting approach is well-suited for a multi-label dataset.

With the newly expanded dataset, it is necessary to re-examine the two BERT results from Section 4.3 to choose the best alternative. This reinspection leads to the fourth and final design decision of this thesis. Among those two BERT models, the model with a batch size of 128 and a sequence length of 100 will be used as the proposed architecture. It is the alternative that completes the training process using resampled data. The other option with a batch size of 64 and a sequence length of 270 has a much longer training time, leading to Google Colab disconnection and incomplete training.

4.5 Final result

This section presents the thesis's result. The final model has a batch size of 128 and a sequence length of 100. It is trained with RAdam optimizer using default parameters. The learning rate is 2e-5. The total number of epochs is four (Section 4.3).

Table 9 shows the configuration of the final model along with its test data result.

Table 9: Final model and result

No.	Model	Model Parameters	Recurrent Layers	lr	Optimizer	Sequence Length	Batch Size	AUCROC	F1-Micro	F1-Weig.	Fig.
1	BERT	334,875,800	NA	2e-5	RAdam	100	128	0.99	0.96	0.99	35

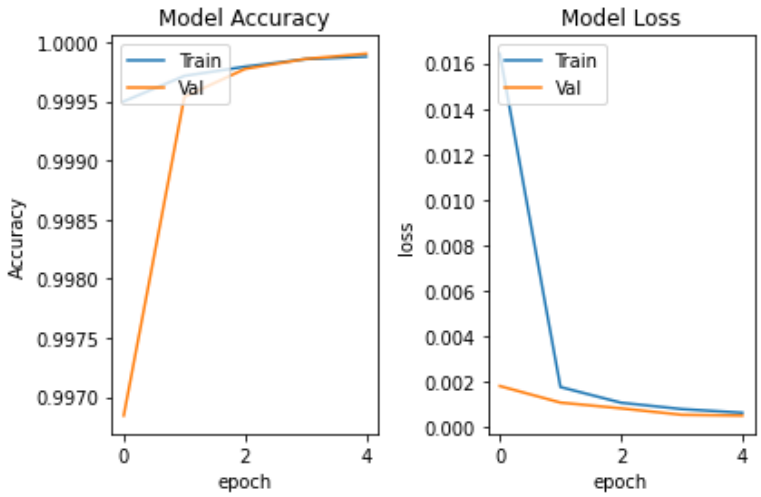

Figure 35. Training and validation history for BERT, final result

From Table 9, the model exceeds 0.9 in all defined metrics. The F1-Micro score is slightly lower than the AUCROC and F1-Weighted scores. Recall that micro-averaging considers every sample to have the same weights (Section 3.2.1); a lowered F1-Micro score means that the model did not correctly classify some samples across several different classes. Those misclassifications are not strongly reflected in the other two metrics due to different methods of calculation. F1-Weighted calculates the F1 score for each class independently and then adds them together using weights from class memberships. AUCROC shows how well the model can classify different classes at different ROC thresholds. F1-Weighted and AUCROC focus on per-class misclassification. Therefore, with F1-Weighted and AUCROC, misclassifying a small number of samples from different classes might not be as

readily apparent as with F1-Micro. Nevertheless, the error displayed by F1-Micro is negligible, and it can be concluded that the model has successfully learned the underlying distribution of the data. Concerning overfitting, the small number of epochs helps the revised model avoid the problem seen in previous models.

From this result, it can be concluded that the examined model based on BERT can be used as a basis for an ED algorithm for both general business and forest industry news.

5 Conclusions

5.1 Summary of Key Findings

This thesis's objective was to investigate and develop an ED method applicable to both general business and forest industry news.

A robust dataset for training algorithms was established from news articles collected from RISI and Reuters. These articles were pre-processed into different topics like forest industry-related, business-related, or other news. To confirm that the labeling was correct, a manual annotation process was carried out. This process also remedied other sentences related issues to create a dataset suitable for training ML and DL models.

It is feasible to use ML solutions as a basis for an ED algorithm applicable to both general business and the forestry industry. Without any advanced data processing, model building, and hyperparameter tuning, linear classifiers such as SVM could create a credible ED algorithm, yielding an F1 score of 0.7. By using a multitopic dataset, this result also indicates that an ED algorithm based on ML or DL can be applied to different subjects simultaneously, given sufficient labeling and training data availability.

Three types of DL models were trained and compared to obtain an optimally performing algorithm. Those models comprise recurrent-based LSTM and GRU, as well as Transformer-based BERT. BERT large, uncased with whole word masking, being the most extensive BERT architecture from Google, was chosen as the BERT model base. Among the BERT experiments, one of the best performing configurations had a batch size of 128 and a sequence length of 100. Despite being more complex and advanced, BERT models only performed marginally better than their recurrent and linear counterparts. This sub-optimal outcome came from class imbalance. Skewing and severe class imbalance were the primary limitations of the processed dataset. The most prominent class had an enormous number of samples, while some classes of interest had substantially fewer samples. Resampling was considered to provide a potential solution to this class imbalance problem. Labels with a small number of samples were oversampled to ensure a similar membership count for every label. Implementing this solution yielded an improvement of at least 15% in test performance. With the resampled dataset, the chosen BERT configuration's evaluation results jumped to 0.99, 0.96, and 0.99 for AUCROC, F1-Micro, and F1-Weighted, respectively. This promising result provides sufficient evidence to conclude that the

proposed architecture based on BERT can classify different topics for general business and forest industry news.

5.2 Contributions

The contributions of this thesis are twofold. Firstly, it produces a robust dataset that can be used to train various ML models to explore the application of ED in the forest industry. The dataset has more than 90,000 sentences with 152 distinct labels. This dataset is the first for the stated purpose. Secondly, the thesis proposes a DL solution as a basis for an ED algorithm applicable to both general business and the forestry industry. The proposed model and its result can be used as a benchmark in future work.

5.3 Limitations and Suggestions for Future Study

Although the research objectives have been fulfilled, there are caveats to the proposed solution and opportunities for future work.

The first limitation in this thesis is the lack of prior studies on the application of ED in the forest industry. This absence creates a gap in the theoretical background of this thesis. There is also no existing benchmark for comparison. However, this is not a considerable drawback as it contributes to some design decisions that shape the proposed solution.

Data-wise, there are two main limitations. The first limitation lies in the labeling process. Since the author had to manually label more than 90,000 sentences in a period of several months, there would be human error such as inconsistency. This inconsistency can lead to missing or incorrect labels. Such limitation opens an opportunity for future researchers to continue to refine the dataset. One recommendation is to use crowdsourcing in combination with independent checking. Another data-related issue that deserves attention is class imbalance. Even though oversampling was implemented, it is not considered an efficient solution since the total number of samples increased by more than thirty times. Furthermore, the resampled dataset may not have been of an appropriate difficulty level for applying BERT. Future work may consider exploring other alternatives to resampling, such as implementing cost-sensitive learning or focal loss. By not using the resampled data, the model building process would enjoy two main benefits: efficiency gain during training and additional possible hyperparameters configurations.

Finally, the proposed ED algorithm is restricted to only general business and forest industry news. A large portion of the Reuters dataset contains news from other topics such as sport, political, or natural disasters (Section 3.1.1). With the third design decision (Section 3.1.2), non-business and non-forest sentences were given a simple 'Other' label, discarding a large part of information from the Reuters dataset. To further utilize the processed dataset, the 'Other' label can be split into different labels such as 'political news', 'sports news', or 'general news'. Coming across sentences in those topics, a trained ED algorithm will able to differentiate them and return the correct label(s) instead of outputting a non-descriptive and generic 'Other' label. With this labeling scheme, an ED algorithm will indubitably be more useful when applied to news in general.

5.4 Managerial Implications

Despite being a billion-dollar industry, the forest industry lacks some competitive intelligence tools that other industries enjoy. This thesis showcases that implementing one such tool is possible. With news ED, companies will have an additional source of information and will be better informed to consider strategies based on the industry landscape and competitors' activities. This expanded business intelligence capacity potentially allows for a more streamlined and faster decision-making process. There is a question of the need for such extra capacity. One can argue that due to the excess of available data, there is not much point in giving companies even more data. However, the data from such an ED algorithm would be of value since it can be converted into consumable knowledge for decision-makers (Forbes, 2018).

The simplest way of applying this thesis is to use the existing code and model to classify news articles. However, since this thesis is a proof of concept, the realistic way to apply its findings is for companies to follow the same steps taken in this thesis. They include selecting a relevant dataset, adjusting that dataset, and tuning the model according to a set of requirements. Dataset selection can be carried out similarly to this thesis: scrapping a news source of interest then annotating the sentences. Companies can add to or remove parts of the selected dataset based on their needs. Data annotation can be crowdsourced and checked to ensure quality. For model building and tuning, transfer learning can be used to take advantage of existing powerful pre-trained models. After tuning, an ED model can be used in a pipeline to generate business value. The pipeline would start by scrapping for articles from various news outlets and optionally performing a word matching search to filter

out non-related articles. Once articles of interest have been gathered, they would be broken down into sentences and then processed to fit the model's format. The trained model would then predict those sentences' labels and trigger an alert if there are sentences with specific labels. In some instances, human experts could perform a validity check on received alerts. This pipeline can be set to run on a schedule to ensure companies are kept up to date on current market movements and competitors' activities.

However, since most companies do not have a sizable department dedicated to data, except for more prominent corporations, the primary applicants of this scheme would likely be agencies specializing in data integration solutions. These agencies would follow the same steps outlined above to produce a scalable ED system for enterprises. The deliverable would be either the algorithm's results or an actual working version of the system in the form of software as a service. Nevertheless, companies in the forest industry would still be the end-users and beneficiaries of this development. With the accelerating change of technology, the practicality of an ED algorithm for enterprises is likely to improve. Over time, more companies within the forest industry will get to enjoy this aspect of business intelligence.

References

Abrishamkar, S., Khonsari, F., An, A., and Huang, J. (2018). Predicting forced population displacement using news articles. *Manuscript Submitted for Publication*

Adhikari, A., Ram, A., Tang, R., and Lin, J. (2019). Docbert: Bert for document classification. *arXiv preprint arXiv:1904.08398*

Alammar, J. (2019). *The Illustrated Word2vec*. Retrieved January 20, 2021, from https://jalammar.github.io/illustrated-word2vec/

Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E.D., Gutierrez, J.B. and Kochut, K. (2017). A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*

Anderson, R. (2019). *RNN, Talking about Gated Recurrent Unit*. [Online Image]. Retrieved January 18, 2021, from https://technopremium.com/blog/rnn-talking-about-gated-recurrent-unit/

Bafna, P., Pramod, D. and Vaidya, A. (2016). Document clustering: TF-IDF approach. *International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016, 61-66. doi: 10.1109/ICEEOT.2016.7754750

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks, 5*(2), 157-166. doi:10.1109/72.279181

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research, 3*, 1137-1155. doi: 10.5555/944919.944966

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. *Neural networks: Tricks of the trade*, 437-478. Springer, Berlin, Heidelberg

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on Computational learning theory*, 144-152

Breiman, L. (1996). Bagging predictors. *Machine learning, 24*(2), 123-140

Cho, K., Merrienboer, B. V., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint arXiv:1406.1078*

Choromanska, A. and Monteleoni, C. (2012). Online clustering with experts. *Artificial Intelligence and Statistics*, 227-235

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv*:*1412.355*

Coenen, A. and Pearce, A. (n.d.). *Understanding UMAP*. Retrieved January 19, 2021, from https://pair-code.github.io/understanding-umap/

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research, 12*, 2493-2537

Cretulescu, R. G., Morariu, D., Breazu, M., and Volovici, D. (2019). DBSCAN Algorithm for Document Clustering. *International Journal of Advanced Statistics and IT&C for Economics and Life Sciences, 9*, 1

Cui, Y., Che, W., Liu, T., Qin, B., Yang, Z., Wang, S., and Hu, G. (2019). Pre-training with whole word masking for chinese bert. *arXiv preprint arXiv:1906.08101*

Dai, A. M., and Le, Q. V. (2015). Semi-supervised sequence learning. *arXiv preprint arXiv:1511.01432*

Dai, X.Y., Chen, Q.C., Wang, X.L. and Xu, J. (2010). Online topic detection and tracking of financial news based on hierarchical clustering. *2010 International Conference on Machine Learning and Cybernetics, 6*, 3341-3346. IEEE

Deep Learning Demystified (2020). *Understanding Optimizers*. Retrieved January 18, 2021, from https://deeplearningdemystified.com/article/fdl-4

Dedić, N., and Stanier, C. (2016). Measuring the success of changes to existing business intelligence solutions to improve business intelligence reporting. In *International conference on research and practical issues of enterprise information systems* (225-236). Springer, Cham.

Devlin, J., Chang, M. W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional Transformers for language understanding. *arXiv preprint arXiv:1810.04805*

Domala, J., Dogra, M., Masrani, V., Fernandes, D., D'souza, K., Fernandes, D., and Carvalho, T. (2020). Automated Identification of Disaster News for Crisis Management using Machine Learning and Natural Language Processing. *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 503-508. IEEE

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*(10), 78-87

Ekta, J. and Roopesh, S. (2017). Data Mining: Document Classification using Naive Bayes Classifier. *International Journal of Computer Applications. 167*(6), 13-16. doi:10.5120/ijca2017913925.

Elman, J. (1990). Finding Structure in Time. *Cognitive Science, 14*(2), 179-211. doi:10.1207/s15516709cog1402_1

English, L. P. (1999). *Improving data warehouse and business information quality: methods for reducing costs and increasing profits*. John Wiley & Sons, Inc.

Eurostat (2020). *Forests, forestry and logging*. Retrieved November 20, 2020, from https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Forests,_forestry_and_logging#Economic_indicators_and_employment

Food and Agriculture Organization of the United Nations (1995). *Forest industries: crucial for overall socio-economic development*. Retrieved November 19, 2020, from http://www.fao.org/3/v6585e/v6585e08.htm

Food and Agriculture Organization of the United Nations (2018). *Forest Products Statistics 2018*. Retrieved November 20, 2020, from http://www.fao.org/forestry/statistics/80938/en/

Forbes (2018*). Today's Companies Are Overwhelmed With Data. Here's How The Smart Ones Will Use It*. Retrieved April 23, 2021, from https://www.forbes.com/sites/quora/2018/11/26/todays-companys-are-overwhelmed-by-data-heres-how-the-smart-ones-will-use-it/

Gago, J. J., Vasco, V., Łukawski, B., Pattacini, U., Tikhanoff, V., Victores, J. G., and Balaguer, C. (2019). Sequence-to-sequence natural language to humanoid robot sign language. *arXiv preprint arXiv:1907.04198*

Gencoglu, O. (2018). Deep representation learning for clustering of health tweets. *arXiv preprint arXiv:1901.00439*

Ghaeini, R., Fern, X., Huang, L., and Tadepalli, P. (2016). Event nugget detection with forward-backward recurrent neural networks. *arXiv preprint arXiv:1802.05672*

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning* (Vol. 1, No. 2). Cambridge: MIT Press. Retrieved December 21, 2020, from https://www.deeplearningbook.org/

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222-2232

Hochreiter S., and Schmidhuber J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735-1780. doi:10.1152/neco.1997.9.8.1735

Hogenboom, F., Frasincar, F., Kaymak, U., and De Jong, F. (2011). An overview of event extraction from text. *DeRiVE@ ISWC*, 48-57

Hogenboom A., Hogenboom. F, Frasincar F., Schouten K., and Van Der Meer, O. (2013). Semantics-Based Information Extraction for Detecting Economic Events. *Multimedia Tools and Applications, 64*(1), 27–52

International Labour Organization (n.d.). *Forestry, wood, pulp and paper sector*. Retrieved November 19, 2020, from https://www.ilo.org/global/industries-and-sectors/forestry-wood-pulp-and-paper/lang--en/index.htm

Ioffe, S., and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*

Jacobs, G., Lefever, E., and Hoste, V. (2018). Economic event detection in company-specific news text. *Proceedings of the First Workshop on Economics and Natural Language Processing*, 1-10

Chollet, F. (2017). *A ten-minute introduction to sequence-to-sequence learning in Keras*. [Online Image]. Retrieved January 25, 2021, from https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

Kandel, I., and Castelli, M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT express*, *6*(4), 312-315

Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*

Kriesel, D. (2007). *A Brief Introduction to Neural Networks*. Retrieved December 20, 2020, from http://www.dkriesel.com

Kullback, S. and Leibler, R.A. (1951). On information and sufficiency. *Annals of Mathematical Statistics. 22*(1): 79–86. doi:10.1214/aoms/1177729694

Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. *Twenty-ninth AAAI conference on artificial intelligence*

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436-444. doi:10.1038/nature14539

Lefever, E., and Hoste, V. (2016). A classification-based approach to economic event detection in dutch news text. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)* (330-335)

Li, P., and Mao, K. (2019). Knowledge-oriented convolutional neural network for causal relation extraction from natural language texts. *Expert Systems with Applications, 115*, 512-523

Liquet, B, Moka, S. and Nazarathy, Y. (2021). *The Mathematical Engineering of Deep Learning*. [Online image]. Retrieved February 3, 2021, from https://deeplearningmath.org/tricks-of-the-trade.html

Liu, J., Chang, W. C., Wu, Y., and Yang, Y. (2017). Deep learning for extreme multi-label text classification. *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 115-124

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2019). On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*

Liu, X., Duh, K., Liu, L., and Gao, J. (2020). Very deep transformers for neural machine translation. *arXiv preprint arXiv:2008.07772*

Luong, M. T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*

Ma, S., Zhang, C. and He, D. (2016). Document representation methods for clustering bilingual documents. *Proceedings of the Association for Information Science and Technology, 53*(1), 1-10. doi:10.1002/pra2.2016.14505301065

Masters, D., and Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*

Maaten, L. V. D., and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research, 9*(Nov), 2579-2605.

McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*

Ministry of Agriculture and Forestry of Finland (n.d.) *Forest industry in Finland*. Retrieved May 3, 2021, from https://mmm.fi/en/forests/use-of-wood/forest-industry

Mohamad, A. Y., Mustapha, S. S., and Razali, M. S. (2010). Automatic event detection on Reuters news

Moro, S., Cortez, P., and Rita, P. (2015). Business intelligence in banking: A literature analysis from 2002 to 2013 using text mining and latent Dirichlet allocation. *Expert Systems with Applications, 42*(3), 1314-1324

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., ... and Raghavendra, V. (2018). Deep learning for entity matching: A design space exploration.

Proceedings of the 2018 International Conference on Management of Data, 19-34. doi:10.1145/3183713.3196926

Nguyen, T. H., and Grishman, R. (2015). Relation extraction: Perspective from convolutional neural networks. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, 39-48

Nguyen, T. H., Cho, K., and Grishman, R. (2016). Joint event extraction via recurrent neural networks. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 300-309

Nielsen, M. (2015). *Neural Networks and Deep Learning* (Vol. 2018). San Francisco, CA: Determination Press. Retrieved February 3, 2021, from http://neuralnetworksanddeeplearning.com/

Nugent, T., Petroni, F., Raman, N., Carstens, L., and Leidner, J. L. (2017). A comparison of classification models for natural disaster and critical event detection from news. *2017 IEEE International Conference on Big Data (Big Data),* 3750-3759. IEEE

Olah, C. (2015). *Understanding LSTM Networks*. Retrieved December 10, 2020, from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Orr, J. W., Tadepalli, P., and Fern, X. (2018). Event detection with neural networks: A rigorous empirical evaluation. *arXiv preprint arXiv:1808.08504*

Pal, S. (2019*). Implementing Word2Vec in Tensorflow*. [Online Image]. Retrieved March 15, 2021, from https://medium.com/analytics-vidhya/implementing-word2vec-in-tensorflow-44f93cf2665f

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. *International conference on machine learning* (1310-1318). PMLR

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*

Popel, M., and Bojar, O. (2018). Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics, 110*(1), 43-70

Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*

Rajaraman, A., and Ullman, J. D. (2011). *Mining of massive datasets*. Cambridge University Press

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533-536. doi:10.1038/323533a0

Schuster, M., and Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing, 45*(11), 2673-2681. doi:10.1109/78.650093

Sechidis, K., Tsoumakas, G., and Vlahavas, I. (2011). On the stratification of multi-label data. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 145-158. Springer, Berlin, Heidelberg

She, X., and Zhang, D. (2018). Text classification based on hybrid CNN-LSTM hybrid model. In *2018 11th International Symposium on Computational Intelligence and Design (ISCID)* (Vol. 2, 185-189). IEEE

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958

Stanford University Online (2020). *Convolutional Neural Networks for Visual Recognition*. [Online image]. Retrieved March 15, 2021, from https://cs231n.github.io/neural-networks-1/

Sun, C., Qiu, X., Xu, Y., and Huang, X. (2019). How to fine-tune bert for text classification?. *China National Conference on Chinese Computational Linguistics*, 194-206. Springer, Cham

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems, 27*, 3104-3112

Valenzuela-Escárcega, M. A., Hahn-Powell, G., Surdeanu, M., and Hicks, T. (2015). A domain-independent rule-based framework for event extraction. *Proceedings of ACL-IJCNLP 2015 System Demonstrations*, 127-132

Van Der Maaten, L., Postma, E., and Van den Herik, J. (2009). Dimensionality reduction: a comparative. *J Mach Learn Res*, *10*(66-71), 13

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 5998-6008

Verma, A. and Virk, H. K. (2015). A hybrid genre-based recommender system for movies using genetic algorithm and knn approach. *International Journal of Innovations in Engineering and Technology, 5*(4), 48-55

Vijayarani, S., Ilamathi, M. J., and Nithya, M. (2015). Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks, 5*(1), 7-16

Wang, Q., Li, B., Xiao, T., Zhu, J., Li, C., Wong, D. F., and Chao, L. S. (2019). Learning deep transformer models for machine translation. *arXiv preprint arXiv:1906.01787*

Wang, Y. E., Wei, G. Y., and Brooks, D. (2019). Benchmarking TPU, GPU, and CPU platforms for deep learning. *arXiv preprint arXiv:1907.10701*

Wang, W. (2018). *Event detection and extraction from news articles* (Doctoral dissertation). Virginia Tech

Williams, J., Tadesse, A., Sam, T., Sun, H., and Montanez, G. D. (2020). Limits of Transfer Learning. *International Conference on Machine Learning, Optimization, and Data Science* (382-393). Springer, Cham

Wunderwald, M. (2011). NewsX. *Event Extraction from News Articles* (Diploma Thesis). Dresden University of Technology. Dept. of Computer Science

Xiang, W., and Wang, B. (2019). A Survey of Event Extraction From Text. *IEEE Access, 7*, 173111-173137. doi:10.1109/ACCESS.2019.2956831

Yang, G., Pennington, J., Rao, V., Sohl-Dickstein, J., and Schoenholz, S. S. (2019). A mean field theory of batch normalization. *arXiv preprint arXiv:1902.08129*

Yang, J., Wang, M., Zhou, H., Zhao, C., Yu, Y., Zhang, W., and Li, L. (2019). Towards making the most of BERT in neural machine translation. *arXiv preprint arXiv:1908.05672*

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*

Yin, W., Kann, K., Yu, M., and Schütze, H. (2017). Comparative study of CNN and RNN for natural language processing. *arXiv preprint arXiv:1702.01923*

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks?. *arXiv preprint arXiv:1411.1792*

Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *IEEE Computational intelligenCe magazine, 13*(3), 55-75. doi:10.1109/MCI.2018.2840738

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., ... and He, Q. (2020). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, *109*(1), 43-76

Appendix A: Labels and groups

Group	Label
Asset	Asset_acquisition
	Asset_acquisition_failed
	Asset_on_sale
Award	Award_received
Bankrupt-cy	Bankruptcy_avoided
	Bankruptcy_filed
Bond	Bond_or_share_issue
	IPO_events
Business related	Business_related
Certificat-e	Certificate_audited
	Certificate_received
	Certificate_required
Company	Company_organizational_change
Consumption	Consumption_decreased
	Consumption_increased
	Consumption_unchanged
Cost	Cost_cutting
	Cost_high
	Cost_low
Currency	Currency_revalued
Deal	Deal_cancelled
	Deal_expired
	Deal_signed
	Offer_rejected
Demand	Demand_high
	Demand_low
	Demand_outlook_down
	Demand_outlook_up
	Demand_unchanged
Economic	Economic_growth
	Economic_slowdown
Energy	Energy_efficient
Equipme-nt	Equipment_or_service_supply
Facility	Facility_closed
	Facility_construction
	Facility_opened
Financial	Financial_trouble_or_debt
Fine	Fine_issued

Group	Label
Funding	Funding_secured
	Funding_seeking
Governm-ent	Government_subsidy
Illegal logging	Illegal_logging
Import Export	Export_high
	Export_low
	Import_high
	Import_low
	Import_export_unchanged
Index	Index_decreased
	Index_high
	Index_increased
	Index_low
	Index_unchanged
Inventory	Inventory_decreased
	Inventory_high
	Inventory_increased
	Inventory_low
	Inventory_report
	Inventory_unchanged
Investme-nt	Investment_detail
	Investment_reduced
	Investment_withdrawn
Lawsuit	Lawsuit_appealed
	Lawsuit_decision
	Lawsuit_dismissed_or_overruled
	Lawsuit_filed
Logistics	Logistics_improvement
Market	Market_balanced
	Market_expansion
	Market_recovering
	Market_strong
	Market_uncertainty
	Market_weak
Natural disaster	Natural_disaster
Jobs	Jobs_created_or_retained
	Jobs_cut

Group	Label
Other	Other
Permit	Permit_appealed
	Permit_received
	Permit_required
	Permit_suspended_or_denied
Plant	Plant_closure
	Plant_construction
	Plant_conversion
	Plant_expansion_or_maintenance
	Plant_relocation
	Price_below_effective_list_price
	Price_decrease_expected
	Price_decreased
	Price_discount
	Price_high
	Price_increase_expected
	Price_increased
	Price_low
	Price_negotiation
	Price_target
	Price_target_raised
	Price_target_reduced
	Price_unchanged
Productio-n	Production_capacity
	Production_decreased
	Production_high
	Production_increased
	Production_low
	Production_mix_alteration
	Production_outlook_down
	Production_outlook_up
	Production_plan
	Production_restart
	Production_startup
	Production_stop
	Production_test
	Production_unchanged
	Production_unchanged
Rating	Rating_events

Group	Label
	Profit_decreased
	Profit_increased
	Profit_negative
	Profit_outlook_down
	Profit_outlook_up
	Profit_report_or_expectation
	Profit_unchanged
	Sales_decreased
	Sales_increased
	Sales_outlook_down
	Sales_outlook_up
	Sales_unchanged
Project	Project_completed
	Project_delay
	Project_dropped
	Project_in_progress
Purchase	Purchase_high
	Purchase_low
	Purchase_unchanged
Shipment	Shipment_decreased
	Shipment_delay
	Shipment_increased
	Shipment_to_capacity_ratio_decreased
	Shipment_to_capacity_ratio_increased
	Shipment_to_capacity_ratio_unchanged
	Shipment_unchanged
Statistics	Statistics_changed
Stock	Stock_buyback
	Stock_decreased
	Stock_increased
	Stock_unchanged
Study	Study_or_investigation
Supply	Supply_outlook_down
	Supply_outlook_unchanged
	Supply_outlook_up
	Supply_high
	Supply_low
Tax	Tax_imposed
Workers	Workers_strike