Anssi Taskinen

Machine Learning Approach to Classifying Finnish News Articles

Helsinki Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

1 December 2020

Helsinki Metropolia University of Applied Sciences

PREFACE

This study was a big challenge for me. During the writing, I faced several inconveniences that made my writing process a bit complicated. I could not finalize this paper before exiting the company this thesis was created for. This caused a slight hassle in the writing process. The work done for the project went well, but honestly, I am more like a maker and doer guy than an academic writer!

A small piece of luck: The subject was interesting, and actually, it was hard to narrow the scope enough for the paper. Somehow, I managed to keep the focus on the things related to the actual outcome. I learned a lot of machine learning related things during the project.

I have to send a few thank you messages. First goes to an important person related to this work, who worked as a mentor in this project on behalf of the company. He is my long-time colleague Teemu Kuusisto MSc. It was an honor to get a piece of that expertise you have during our co-working years!

Thanks to my Principal lecturer Juha for the extreme help with this study. I have never met a person so passionate about what he does. A huge thanks go to Ville for pushing me to do this work.

Finally, big thanks go to my wife and my company partners for giving me the time required to finish this job. Without your understanding and will to help, I would never complete this.

Espoo, 1.12.20
Anssi Taskinen

Author(s) Title Number of Pages Date	Anssi Taskinen Machine Learning Approach to Classifying Finnish News Articles 41 pages 1 December 2020
Degree	Master of Engineering
Degree Programme	Information Technology
Specialisation option	Networking and Services
Instructor(s)	Juha Kopu, Principal Lecturer Teemu Kuusisto, Software developer MSc

This study investigates the possibility of classifying Finnish news articles with the methods of machine learning. The work aims to both save time otherwise spent in performing the task manually, and to improve the quality of the articles through their automatic classification. The study focuses on determining a single keyword for each article on the basis of its contents.

The outcome of the study is a test application providing an interface for requesting a keyword (class) for the inserted article (textual content). The implementation combines machine learning techniques with those of traditional programming; the latter ones are employed mainly in tasks related to data preformatting. The actual classification model has the form of a neural network combining convolutional layers with standard fully connected layers. Word embedding and other advanced text preprocessing techniques were used to convert the article texts to numerical form.

From the early stages of the work, the article classification task revealed itself to be a difficult one to tackle from the machine learning perspective. Most importantly, only a part of the available article data contained the relevant keyword field, which is necessary for training the machine learning models. Furthermore, most of the articles containing this label turned out to have only a single keyword available. This latter fact was taken into account by restricting the models to also output a unique keyword label for each article input. The obtained results provide insight into the possibility of classifying the text articles automatically.

The test program was implemented successfully, and has been used in a test environment to predict keyword class labels for real news articles. The highest observed success rates were nearly 60%. Finally, some proposals for further development are formulated in the end of the thesis.

Keywords	Natural Language Classification, Neural Networks, Word Embeddings

Table of Contents

List of Abbreviations

CMS	Content management system/software
CBOW	Continuous Bag-of-Words
OS	Operating system
NLTK	Natural language toolkit
CNN	Convolutional neural networks
RNN	Recurrent neural networks
DENSE	Fully connected neural networks
ReLu	Rectified linear unit
XML	Extensible markup language
FSTs	Finite-state transducers
HFST	Helsinki finite-state transducer technology
JSON	JavaScript object notation
NDJSON	New Line delimited JSON
HTTP	Hypertext transfer protocol

1 Introduction

Anygraaf is a software company focusing on solutions for the media field. The company's main product is a content management software (CMS) used worldwide to manage and write news articles at the media companies. Client companies share articles to increase efficiency by reducing the workload. Each article contains metadata that may be context depending, meaning that it might differ depending on the publication the article is published at. Examples of this kind of metadata fields are “department”, “keywords” and “tags”. Filling those for an article taken from another context is currently a manual task. Entering the department to that article is straightforward because there is a fixed number of options to choose from. Adding keywords and tags manually has not been an obvious task. This is because there are many opinions about the correct ones.

Machine learning can potentially provide a solution to solve this problem. Learning the article's keywords based on article text might provide a way to build a model that determines a context depending keyword. The solution allows clients to improve the quality of existing articles by providing a way to enter solid keywords to them afterwards.

This study focuses on developing a machine learning model that can be tested to solve a keyword for the news article. It aims to answer the question, whether the machine learning approach is applicable to solve the problem using the available data. Data analysis revealed the insight that most of the articles either contained no keywords or just a single keyword. This is the reason to focus on solving a single keyword for an article instead of solving multiple keywords.

The outcome is a test application that provides an interface to request the text's keyword. Integrating the outcome to the company's CMS is not included in this thesis since the focus is in the machine learning. However, the last chapter presents the concept for that task. The integration is designed in a way, that enables the continuous learning strategy.

The research process had five steps. It starts with identifying the problem by interviewing the company’s stakeholders and customers. Next, the problem was discussed with the machine learning expert who worked for the same company. This discussion narrowed the scope of this study enough to be realistic to implement. This step's outcome is a research plan with a list of possible tools to build the solution. The tools are selected by

research, and in some cases, based on previous experience. They are introduced in chapter 4.

The next step is collecting available data and implementing tools for data preformatting. Unsuitable data is discarded, and the remainder is preprocessed to a uniform structure. Tools used for this step are small Python applications that can be used later with the data pipelines to re-teach the machine learning models.

In the preformatting step, text content is converted to a suitable format for machine learning models. This includes morphological conversions like converting words to a base form. This reduces the number of different words.

The fourth step is to build models. The first model is called word embeddings. This model can be considered as a part of the data preprocessing step because it preprocesses data to a proper format for an artificial neural network to learn. It provides an algorithm to transform words from natural language or character sequences into numerical representations [1]. Building this model requires iterations back to the preformatting step and even back to collecting data. The more text content there are to build word embeddings, the more accurate embeddings will be. Finally, the actual machine learning model is taught using word vectors from the word embeddings.

The steps related to data processing and model building are repeated iteratively according to the evaluation results. Figure 1 represents the study workflow.

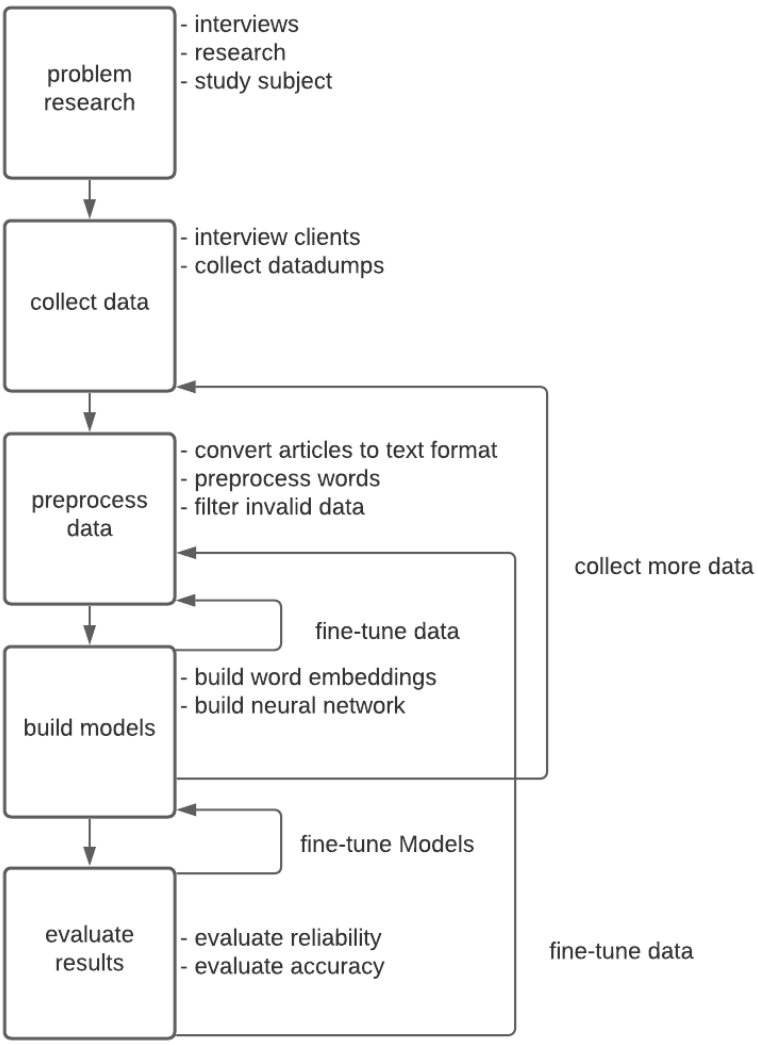

Figure 1. Study workflow.

This paper is divided into six sections. The introduction presents the problem and delimits the scope of the project. It explains how this study is conducted and describes the outcome's main features.

"Techniques and background" part contains theoretical description at a general level for machine learning techniques used in this work. This is required to understand the work that is done in this thesis. The chapter focuses on machine learning techniques. “Tools and libraries” chapter describes the other tools used in this work and explains why each tool is chosen for the project.

“Building a solution” chapter presents the solution to the problem. The “Results and analysis” chapter gives the results. The last chapter, “Conclusions” presents thoughts and future development ideas.

2 Techniques and background

This section introduces the basic knowledge required to understand the work done in this thesis. It explains the machine learning related terminology used in this paper.

2.1 Machine learning

Machine learning consists of computer programs that learn features from data. Those programs try to find out features from data based on input-output relationships and then build rules presenting relations between data and features. Machine learning is a trend nowadays, and companies invest in machine learning research because the available data has grown significantly, and computers have more and more processing power for calculations. It is not a new invention because the first papers available are already from 1956 [2]. Machine learning can be divided into three different branches. Those are reinforcement learning, supervised learning, and unsupervised learning [3].

Reinforcement learning is based on exploration and exploitation. In this learning approach, an algorithm, better known as an agent, attempts to achieve the goal in an uncertain environment. The agent gets a trial situation and an error and should develop an action that avoids the error (is a solution for the problem). The action affects the environment, and the agent gets either rewarded or penalized as feedback for the performed action. A real-life application suitable for reinforcement learning could be a self-driving car. The reward function might estimate how much damage the car caused by the action and whether it got to its destination. Reinforcement learning is a much-used machine learning method in the robotics and virtual gaming industry [4, 5].

Unsupervised learning is concerned about modeling data without training set that includes the correct answers. This is a suitable branch for problems that do not have all known answers yet. A simple real-life use case suitable for unsupervised learning is to find potential customer segments. The algorithm groups the samples, in this case, customers, by finding out common features. One sample could belong to several different groups simultaneously. These groups are more commonly named clusters in

machine learning. Each cluster represents a group of features. In this case, the number of samples in a cluster presents how well those features describe a potential customer segment. Many unsupervised learning applications are used to find features from media files, such as videos and images [6].

The last branch is supervised learning. This method always requires separate training data with the correct answers. The data used for teaching the model is often referenced as a training set, and correct answers are called labels. In supervised learning, the data is used for training, and then an algorithm builds rules that led to the label. With those rules, the model can solve similar problems for samples outside of the learning set. There are several algorithms for supervised learning, like decision trees and neural networks. The machine learning methods used in this thesis belong to this branch. This work aims to study whether it is possible to classify articles with neural networks. Therefore, the program represented in this thesis constitutes a good example of a real-life application for supervised machine learning.

Many kinds of research confirm the neural network's effectiveness in solving the multi-class classification problem [7]. In machine learning, a multi-class classification problem is a problem in which instances are classified into one of three or more classes. Classifying samples into one of two classes is a binary classification. An example of a multi-class classification is to find out the department for a news article based on its text content and other features. There are a finite number of classes (departments), and the article is classified with one of those.

Machine learning provides solutions for problems that would be complicated to be solved with traditional programming. In some cases, the traditional programming aspect can be almost impossible. Thinking about the number of conditions to determine if a sentence contains words, which commonly occurs in articles classified by some specific keyword. There would be too many conditions to write. If one managed to do so, the code would be extremely complex, and there is no way to update it after new articles are written. Of course, some solutions combining traditional programming with other methods, like mixing heuristics to resolve most meaningful words with a full-text search engine, may provide a solution for this problem. There may still be a need for complex logic to find out the most meaningful words. Machine learning enables the possibility to find them out without struggling with complex programming. It happens automatically when a machine learning algorithm finds out the common features by exploring samples.

The machine learning solution cannot always be categorized into only one of the branches. The final solutions for machine learning are rarely constructed from a single model. They are more often combinations of multiple models, including software logic around the models [8].

2.2 Neural networks

Artificial neural networks are built from layers that contain neurons. There are always one or more input layers and one or more output layers. Between those, there are hidden layers. Each layer contains a predefined number of neurons. The neurons in a particular layer receive a number of input values from the preceding layer and perform a definite mathematical operation with them. The final outcome of this operation constitutes the output value of the neuron, which, in turn, is transferred to the next layer as one of its inputs [9].

Commonly used neural network types are fully connected neural networks (dense), convolutional neural networks (CNN), and recurrent neural networks (RNN). One neural network may contain elements from all these [10]. Figure 2 below represents a simple, fully connected neural network with an input layer i1, two hidden layers h1 and h2, and output layer o1. Hidden layer h1 has 4 neurons, and the hidden layer h2 has 2 neurons. A fully connected neural network means that every neuron on a layer connects to all the neurons on the next layer.

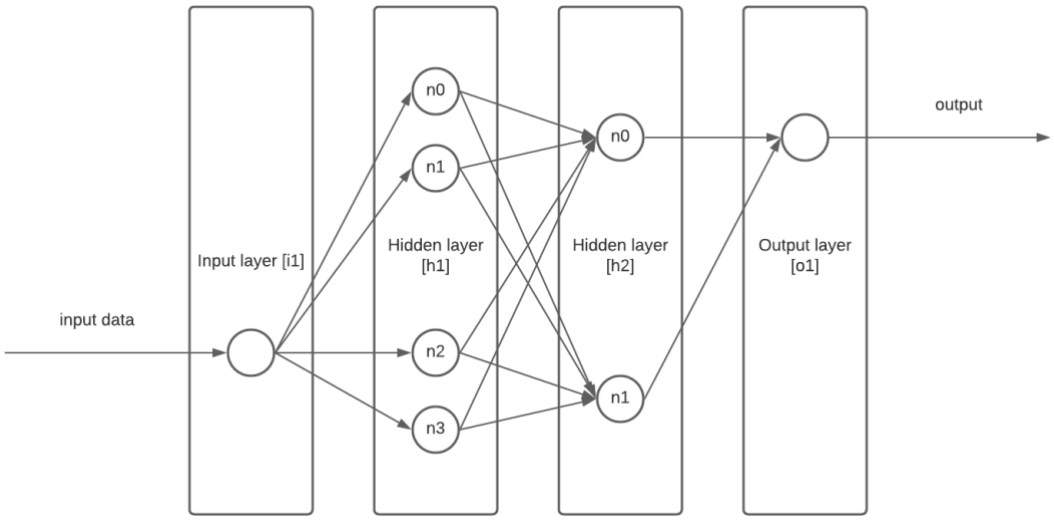

Figure 2. A fully connected neural network with 2 hidden layers.

The biological neural network definitely inspires neural networks due to similarities in functionality and terminology. Original data is given as an input for the input layer. Data flows from the input layer to the hidden layer that performs computations with the data and forwards the result to the next hidden layer or output layer. A neural network with a lot of hidden layers is often called a deep neural network. In a fully connected layer, as in figure 2, the sum operation performed by a single cell is $\sum_{i=1}^{n} C_i X_i + y$, where C_i is input and X_i is a weight, and y is a bias. The bias value is basically included in the general linear expressions to improve the model adaptability.

The cell computes a weighted sum and then passes it through a nonlinear activation function [10]. Nonlinearity is needed for the model to be able to express more diverse input-output relationships. A simple and popular choice for such an activation function is a rectified linear unit (ReLu), in that $f(\mathrm{x}) = \max(0, x)$. That means the final output from a cell may be $f(\sum_{i=1}^{n} C_i X_i + y)$, where f is ReLu. The values for weights and biases are initially initialized randomly. Initial values are adjusted with the model's teaching phase to be more useful under the control of the problem.

Figure 3 below clarifies the above by visualizing what happens in a single cell. In that figure, X_1, X_2, …, X_n are the weights. C_1, C_2, …, C_n are outputs from the previous activities, and y is the bias. The cell calculates a weighted sum with those, and then, a nonlinear activation function is applied.

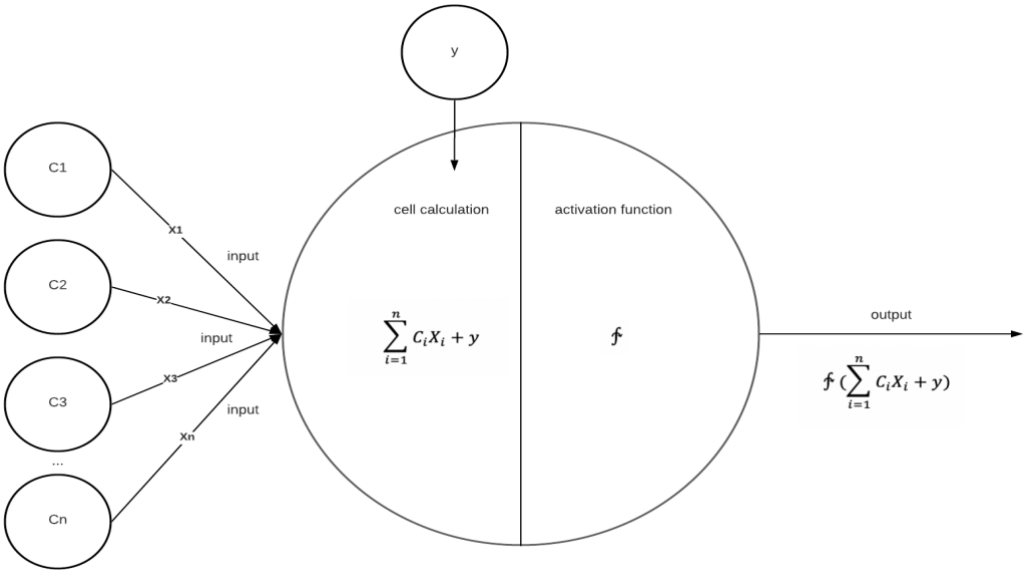

Figure 3. The operation performed by a single neuron.

Convolutional neural networks are based on the fact that there is enough local understanding. Unlike in dense networks, the neurons in convolutional layers are only connected to a restricted number of neurons in the previous layer at a time. This number is determined by the so-called window size. Convolutional layers are developed to provide a solution to a problem in which too many parameters given as an input to the network would cause the learning process to be inefficient [13]. An example of this issue is from this thesis while working with articles and word embeddings. A single article is limited to have one thousand words, each word presented with a 300-dimensional vector generating 300 000 input parameters to a network. That includes input parameters only. The trainable parameters (weights and biases) of the network have to be taken into account as well. The solution provided by the convolutional layers is to view a small window to the data at a time, providing an efficient way to handle a large number of parameters.

Convolutional layers may have one, two, or three dimensions. One dimensional layer (1D layer) is like an array, having only a length dimension. A two-dimensional layer is a matrix, having a height dimension as well. A three-dimensional layer is like a cube, including the depth dimension. Convolutional layers used in this work are 1D layers.

There are two core concepts in convolutional neural networks. The first one is the filters, and the second is pooling. Filters detect important local features like meaningful words. Pooling reduces the size of the output and allows for identifying structures on different hierarchy levels.

As a filter example, imagine an article with five words, each presented with a vector having 3 cells. This article is visualized with an input vector I below. The filter, which sometimes is referred to as the kernel, is a vector with size 3 marked with K.

$$I = [1, 0, 2, -3, 1, 0, 1, -2, 0, 1, 1, 1, 1, 1, 1] \quad K = [1, 0, -1]$$

The kernel (K) is placed at the left edge on the input (I). Then multiplication is performed with K and the first three cells of I. The kernel is moved to the right, and new multiplication is performed. The step size is referred to as stride (S). Below is an example of the results with (I) as an input, (K) as a filter with stride 1 (S1) and 2 (S2).

$$S1 = [-1, 3, 1, -3, 0, 2, 1, -3, 1, 0, 0, 0, 0] \quad S2 = [-1, 1, 0, 1, 1, 0, 0]$$

Figure 4 below visualizes the operations with stride 1 and stride 2 for the five first cells from the input (I) using the kernel (K).

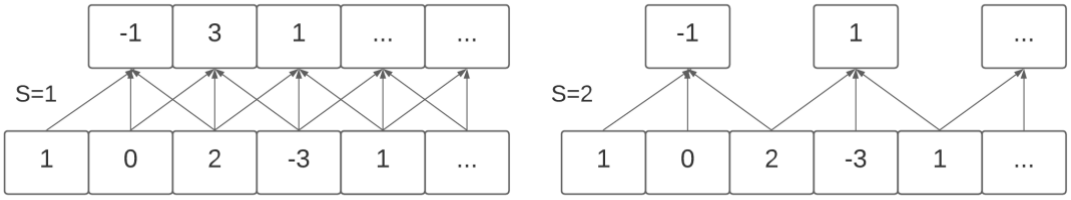

Figure 4. Convolutional neural network layer filtering.

To give an example for pooling, there is a method called max pooling that is used in this work. Max pooling is a technique used to pick the biggest value from the defined window and then replace all values in that window with a single value. The result from max pooling operation with pool size 5 performed with the same input (I) is a vector $[1,1,1]$.

The 1-dimensional convolutional layer is visualized in figure 5. In that example, the input volume consists of a vector with 1000 elements. Then comes one convolutional layer using the ReLu as an activation function and window size 5. Next, max pooling is applied

with pool size 5. Max pooling may be considered as a separate layer or, alternatively, included in the convolutional layer. The output is an array with 199 cells.

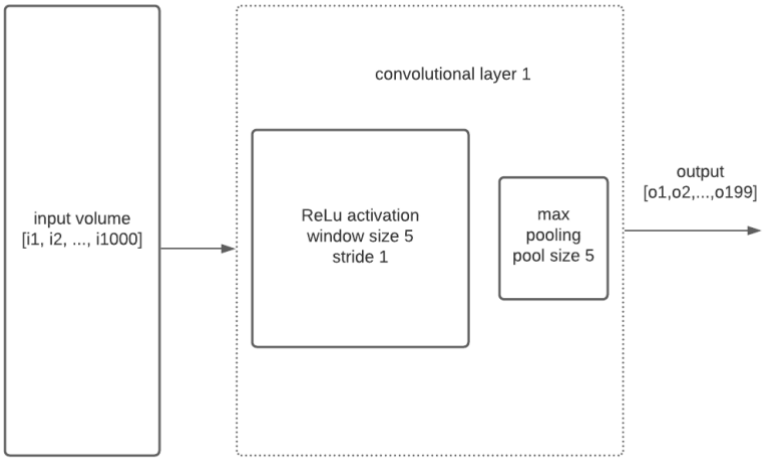

Figure 5. Convolutional neural network layer.

This work's neural network is a combination of convolutional layers and fully connected layers.

2.3 Training neural networks

This chapter covers a simplified description of a neural network's learning process to understand the work done in this thesis. The chapter includes examples of filters and optimizers that can be applied during the learning process. The chapter is written to cover the learning process for this thesis.

When the network structure is designed and the learning process begins, the first step is called forward propagation. In this step, the labeled training data flow through the cells. The correct answers for classifications are called labels. Cells apply calculations according to the existing weights and/or network biases, and finally, the output layer outputs the prediction of the model. This predicted value is compared to a label in supervised learning to know if the output value was correct. To quantify the difference between the prediction and the true (labeled) output value, the so-called loss value is calculated. The loss value is a measure of how accurately the model can represent the input-output relationships of the training set.

In this thesis, a function called SoftMax is applied when making predictions for classification problems. It is a simple function that turns a vector containing predefined number of real values into a vector having the same number of real values that sum to 1. Formula for SoftMax is presented below.

$$\sigma(\mathcal{Z})_i = \frac{e^{z_i}}{\sum_{j=1}^{\mathrm{K}} e^{z_j}}$$

All the z_i values in the formula are input vector elements. They can take any real value. The term on the bottom of the formula is the normalization term, ensuring that all the function's output values will sum to 1, thus constituting a valid probability distribution.

The output from the neural network layer before the SoftMax activation function is an array of input elements (i). SoftMax is used to interpret the output values from 0…1 as probabilities.

After the loss value has been calculated, the values of the weights and biases of the network are adjusted. The optimization algorithm that handles the details of this update procedure is referred to as backward propagation; the name refers to the fact that the parameter adjustments proceed in reverse order (from output layer back towards input layer). There are many optimization algorithms available. One of the most common is gradient descent. The gradient descent finds the gradient of a function to figure out which direction values should be adjusted, to reduce the loss towards a minimum value [10].

A learning rate is a multiplier used to adjust the weights in a controlled manner together with the derivative. If the learning rate is chosen too large, the algorithm might not be able to converge to a local minimum at all, while a too small value might cause the algorithm to take a prohibitively long time to reach the local minimum.

Adam is another optimization algorithm that has been used in this thesis. Adam optimizer uses gradient descent and keeps some of the momentum from the previous calculations combining it with elements from RMSProp optimization. The algorithm aims to reduce movement to an unnecessary direction.

A few additional terms are necessary for assessing the performance of the neural network with the task studied in this thesis. The "Results and analysis" section of this thesis refers to the "confidence" of the model. Confidence is a numeric value from 0 to 1 presenting how confident the model is about the prediction. For example, a neural network can give a keyword for text content with a confidence of 0.6, which means it is 60% sure that this keyword is correct for the given text.

Evaluation metrics for a multi-class classification model may be explained with the context of a binary classification model in which each class is either negative or positive. These are derived from the four categories. The first one is true positive items (TP), where the label is positive, and the class is correctly predicted to be positive. The second category is false positive (FP), where the label is negative, but the class is incorrectly predicted to be positive. The third category is true negative (TN), where the label is negative, and class is predicted correctly to be negative. The last one is a false negative (FN), in which the label is positive, but the class is incorrectly predicted to be negative. A true positive class occurs when the labeled class is predicted to be positive, meaning that there must be multiple classes that should be considered true negative for a given prediction.

In this context, the evaluation term accuracy A can be defined with the form $A = (TP + TN)/(TP + TN + FP + FN)$. It is the fraction of predictions the model got right, calculated simply by dividing the number of true predictions by the total number of predictions. The term precision B is the number of correctly identified positive predictions divided by all positive predictions. Precision can be defined with form $B = TP/(TP + FP)$. The recall C is the number of correctly identified positive predictions divided by the number of all actual positive predictions $C = TP/(TP + FN)$.

It is easy to explain the term F-score used in the “Results and analysis” section with these terms. F-score is calculated from precision and recall, and it represents a measure of accuracy. A traditional F-score is an F1 score that is a harmonic mean of precision and recall. It is calculated with the formula presented below.

$$\mathrm{F1} = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}}$$

The F1 score measures the effectiveness of identification in which as much importance is given to the recall and the prediction.

2.4 Overfitting neural network

When a neural network is too detailed compared to the data complexity, the situation is called overfitting. It causes the neural network to be unable to create any general rules applied to the data. This is the most expected issue that may lead to unsuccessful results with this thesis.

Overfitting is a real problem because only a limited amount of data is available for teaching the neural network. The problem could conceivably be solved with additional data, but such extra data might not be available. An extreme example of such a situation occurs when there are only ten articles to learn, and each of those contains a different set of keywords. No matter how detailed the network is, the result would be that the network can only recognize exactly those ten articles. There is no way to create any general rules that can be applied to any other articles.

To reduce overfitting, there is a popular concept called regularization. One technique that can be listed as regularization is a dropout. In dropout, neurons are left out of the computation with a certain probability [11]. At the training phase, this means the neural network cannot use the same route to give a prediction. It must generalize the weights in a way that allows multiple routes to lead to the same result. Figure 6 below represents the same fully connected neural network at the training phase with dropout applied to the second layer.

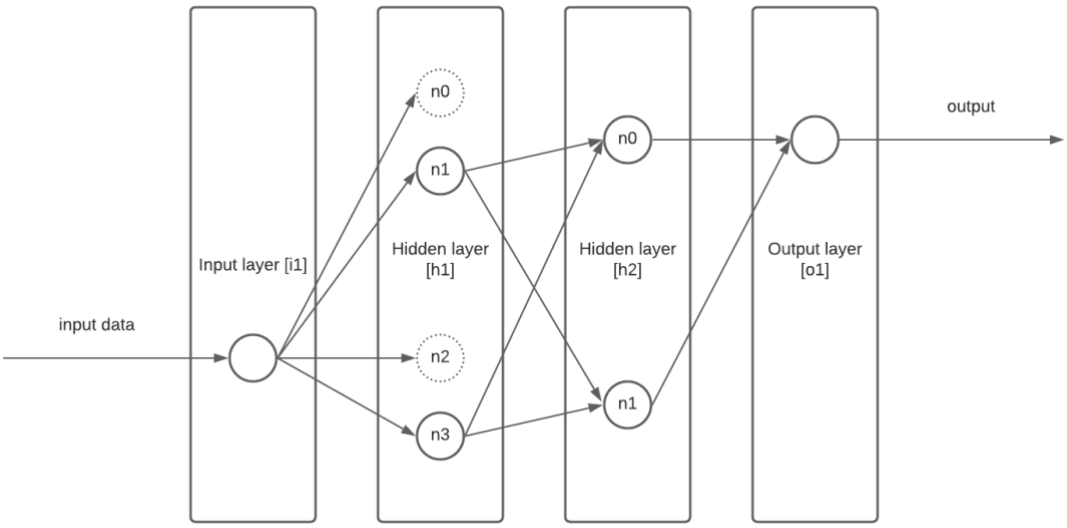

Figure 6. Fully connected neural network with dropout applied to first hidden layer.

2.5 Preprocessing data for neural network

The data has to be cast in the numerical form before it can be manipulated by neural networks. Preprocessing is a commonly used name for the procedures that need to be applied for the input data before it goes to the neural network's input layer.

In typical cases, the input data may already be numerical, but different features might have values ranging over vastly different scales. This causes performance problems. One way to avoid this is normalization. As a rule, that numerical data should always be normalized when possible [12]. .

Normalization is not applicable for all numeric data. In this work, keywords resemble categorical data, for which the concept of ratios has no meaning. With categorical data in mind, there are few commonly used options for preprocessing, and the selection depends on the number of values. The first one is one-hot encoding. It replaces categorical data having n distinct classes with vectors of size n. Table 2 below represents a one-hot encoding approach by applying it to numbers from zero to thousand. The result is a thousand arrays having thousand numeric cells in each.

Table 2. One-hot encoding

0	[1,0,0,0,...,0,0]
1	[0,1,0,0,...,0,0]
2	[0,0,1,0,...,0,0]
999	[0,0,0,0,...,1,0]
1000	[0,0,0,0,...,0,1]

Each cell in the result array presents a single binary value. One-hot encoding is a simple and straightforward way to encode categorical data suitable for machine learning models. The biggest issue with this approach will be memory consumption when there are many categorical options. In this thesis, there will be several thousand words to encode. In this case, a training set with 300 000 lines, each containing a selection from 80 000 different words, would create a matrix with $24 \cdot 10^9$ entries. Available computing power and memory would not be enough to handle that size of data.

The second solution is to use embedding encoding. This is a convenient approach when there are many categorical options to encode. One embedding technique used in this thesis to convert the textual data to a numeric input is word embeddings. Embeddings are multidimensional vectors representing the values with a predefined size of vectors [12]. Embeddings are initialized randomly, and during the training process, each embedding value is adjusted based on the learning set. Adjusting embeddings has similarities with adjusting biases and weights during the neural network learning process. The table below is an example of embeddings where numbers from zero to one thousand have been embedded into a three-dimensional float vector. In the example, numbers between 3 and 999 are skipped.

Table 3. Embedding encoding

0	[0.123, 0.234, 0.356]
1	[0.303, -0.299, 0.996]
2	[0.129, 0.204, 0.380]
1000	[0.333, 0.333, 0.334]

3 Tools and libraries

The chapter represents the tools and libraries chosen for this project. This chapter also contains explanations of why each tool or library is selected.

3.1 Python

Python is a programming language that provides advanced libraries for simple tasks like text processing. It runs on any platform, and many popular machine learning libraries and frameworks provide well documented Python APIs. Because Python code is compiled at runtime, it makes the prototyping process easy and efficient for this work. Previous experience with Python programming also influenced the selection of Python.

3.2 Docker

Docker is a software solution that makes it possible to deliver software wrapped with the environment. Docker also includes multiple other features, and it should not be simplified to only a packing solution, but this thesis only uses those deliverable packages.

The Docker package that contains an application and an environment is called a Docker image. When the Docker engine starts the Docker image, it is called a Docker container. The benefit achieved by using Docker containers to deliver software is that there are hardly any operating system (OS) related compatibility issues with libraries. This work relies on several external libraries that make Docker a valuable and time-saving tool. Figure 7 represents how the Docker container wraps an application with the environment.

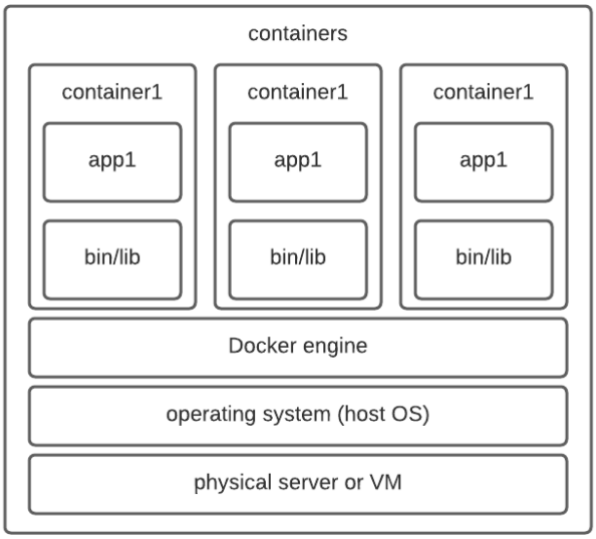

Figure 7. Docker containers example.

3.3 Word embeddings and Word2Vec technique

The word embeddings is a technique of representing text so that words with similar or same meaning have the same representation. The model is demonstrated to successfully capture the semantic similarities between words, even with syntactically different words.

Technically speaking, word embedding represents words or character sequences with real-valued vectors in a fixed-dimensional vector space. This means that each word is represented as a vector having tens or even hundreds of dimensions. This allows calculating the distance between two vectors or the distance between two words [7].

Predefined size vocabulary from a corpus of text is used to teach vector representations. The learning process might be included in a neural network model as a task, or, alternatively, it is an unsupervised process that uses document statistics.

Word2Vec is one of the techniques for teaching word embeddings. The method is based on statistics, and it provides embedding for standalone words from a text [7]. It was developed to make the neural-network-based training of the embedding more efficient. Since Tomas Mikolov developed Word2Vec at Google in 2013, it has gained a standard status for pre-trained embeddings.

Two different learning methods can be perceived as branches of the Word2Vec approach. The first one is Continuous Bag-of-Words (CBOW), and the second is Skip-Gram. The common principle with both is that they learn by predicting word or words in a window that requires knowing their local usage context. The CBOW model learns by predicting the current word based on surrounding words. The continuous Skip-Gram model learns by predicting the surrounding words from the current word [7]. This is clarified in figure 8.

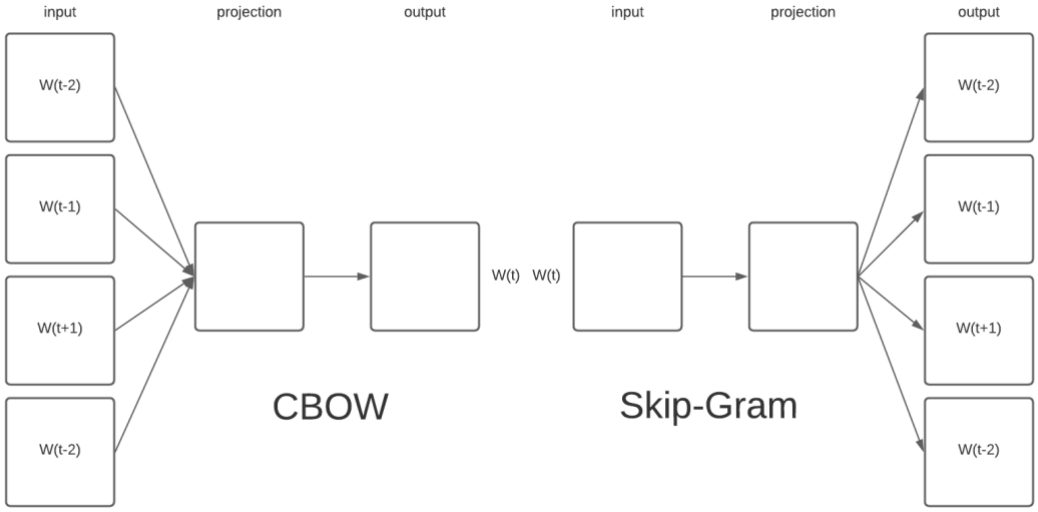

Figure 8. CBOW and Skip-Gram methods.

Word2Vec is one of the most popular models used for training word embedding models [7]. Nowadays, there are many pre-trained models available to be downloaded free of charge. In this work, a separate Word2Vec embedding model was employed to evaluate whether there is enough data available for achieving reasonably accurate word embeddings.

Some custom text preformatting was also used for the solution covered by this thesis, and those pre-trained models would not be suitable for that reason (see the preformatting chapter below for more details). Much available material, including exemplary implementations of word embeddings, also influenced the selection of the Word2Vec technique as a tool for building embeddings in this work.

3.4 TensorFlow

An open-source machine learning framework called TensorFlow is the primary framework used to build machine learning models in the project covered by this study. TensorFlow is the most popular tool for building deep-learning solutions. TensorFlow was chosen due to previous experience and comprehensive documentation, and because of the framework's general popularity. It was especially suited for the task studied in this thesis because it has a Python application programming interface (API), and all the other code evolving this work is written with Python.

According to data scientist Jeff Hale's article, TensorFlow is the most powerful tool for building deep learning solutions [13]. This score is built with 11 data sources across 7 distinct categories to gauge framework usage, interest, and popularity.

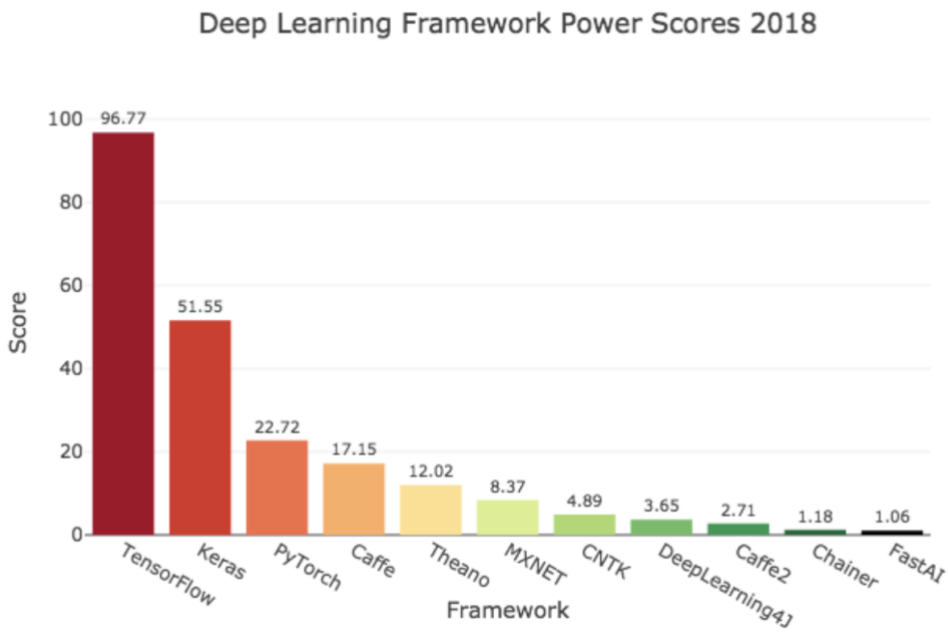

Figure 9. Machine learning frameworks power scores [14].

To understand the work done in this thesis, it is relevant to know how TensorFlow operates. When using TensorFlow with Python, one must first define a graph. A graph describes the structure of the model and what kind of variables, constants, and parameters are involved. After a graph is defined, a TensorFlow session is launched. A session is where actual calculations within the graph are executed [13].

TensorFlow has a saver and loader to store a trained model into a file and continue processing it afterward. That is a widely used feature in this thesis.

3.5 Natural language toolkit

According to python.org, the natural language toolkit (NLTK) is the leading platform for building Python programs to work with human language. It provides several tools for the preformatting phase of this thesis. With this toolkit, the textual content is formatted to be suitable for the word embeddings model to learn. This library was selected to be used in this work based on previous experience.

3.6 NumPy

NumPy is a scientific computing package for Python offering comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more. NumPy mathematical functions are extremely optimized, and it is effective for n-dimensional array manipulation. It is also being used by TensorFlow which is a machine learning framework used in this thesis.

3.7 Finite-state transducers

Finite-state transducers (FSTs) have been used for morphologically analyzing text to convert tokens from the surface form to the lexical one [16]. Morphological analysis is one of the tasks that have been studied for years. Different techniques have been used to develop models for performing morphological analysis. Finite-state transducers are one of the most powerful and most accurate techniques to perform analysis.

FSTs map a set of input sequences to a set of output sequences. A transducer is defined by an input alphabet A, an output alphabet B, a set of states S, a set of initial states I, a set of final states F, and a set of transitions between different states of the transducer T where $I \in S$, $F \in S$, $T \in S \times (A \cup \{\}) \times (B \cup \{\}) \times S$.

Each transition has an input token, output token, source state, and destination state. Given an input sequence, the output sequences correspond to the set of valid paths between the initial and final states passing through a set of intermediate states.

Models based on finite-state transducers have proved to be more suitable for languages with low available resources [16]. That is one reason why finite state transducer technology is suitable for morphological analysis with the Finnish language. In this study, FST based technology is used to reformat text to reduce the number of different words. This is done by converting each word to the base form. This reduces the number of different words successfully, but at the same time, part of the data gets lost. Since this study aims to find similarities between articles in order to classify them, it is a risk that needs to be taken.

In this study, FST technology is used with the Helsinki Finite-State Transducer software (HFST) provided by the department of modern language at the University of Helsinki. This software is intended to implement morphological analyzers and other tools based on weighted and unweighted finite-state transducer technology. Software is licensed under the LGPLV3 license.

4 Building a solution

This chapter presents the solution and how it was built. It covers why the choices were made and how those affected each building phase's outcome.

4.1 Collecting data

The data used in this study came from Anygraaf’s client company. That company has almost one million articles stored in the relational database. Those articles were written in Finnish, and there were no translations available. Each of those articles was stored in a format that is called rich text in the company. Rich text is basically an extensible markup language (XML) document containing text and formatting. Below is a randomly selected example of the rich text document.

```
<richtext version="1.0.0.0">
  <paragraphlayout>
    <paragraph box="1" parstyle="Headline">
      <text>Nokia sai omia parannuks</text><text>ia Windows Phone 7.8
päivitykseen</text>
    </paragraph>
    <paragraph box="2" parstyle="Ingress">
      <text></text>
    </paragraph>
    <paragraph box="3" parstyle="Text">
      <text>Ohjelmistoyhtiö Microsoft on saanut valmiiksi Windows Phone 7.5 -puhelinten
seuraavan päivityksen...</text>
    </paragraph>
  </paragraphlayout>
</richtext>
```

Each article entry on the articles table in the relational database contains fields presented in the following table. The “exists in most” column tells how often an article entry contains a value for the field. The label “yes” means that more than half of the articles are delivered with a value for the field.

Table 4. List of available fields.

key	**type**	**description**	**exists in most**
tname	string	Title of the article	yes
headline	String or XML Document	Headline of the article	yes
subdepartment	string	Name for the sub department	no
department	string	Name for the department	yes
keywords	string	Comma separated list of keywords	no
hierarchykeyword_a	XML Document	Contains IPTC tags for the article	no
texttag_a	XML Document	Contains tags for the article	no

As exhibited in table 4, most of the articles are delivered with incomplete values. This reveals an issue in the learning data leaving the "department" to be the only meta field

that exists in most of the articles. Unfortunately, there is no real-life application for that feature alone since, almost without exception, a journalist writes content for only one department. Filling a department based on the journalist's details always gives the correct result without machine learning.

The” keywords” field is available in 30% of the articles. This narrows the number of articles available for learning to 300 000. Further data investigation showed that most of those having a value for the “keywords” field contained only one keyword. For this reason, the original plan was updated. Originally the plan was to build a model that finds out an unrestricted number of keywords for an article. Having a lot of less data than originally planned and most of the articles having only one keyword provided an opportunity to focus on a model that predicts only one keyword. The smaller data motivated putting an extra effort into the text preformatting phase by including extra steps like transforming words to a base form. These are attempts to increase the possibility to find similarities between articles.

4.2 Preformatting data

This chapter presents how the data was formatted to be suitable for machine learning models to learn.

The preformatting phase started once the data was gathered, and simple analysis of the available fields was done. The outcome of this phase is a program that converts data to a suitable format for machine learning models to learn.

Exploring rich text documents exposed that there was no unified way used with text styles between documents. The previous example of the rich text document in chapter 4.1 demonstrates the article having a headline, ingress, and text sections, but those only exist in part of the articles. Further investigation of the stylesheet's history revealed that the document was updated frequently, and those fields were inconsistent with each other, rendering them unsuitable for this task. If those fields had been unambiguous, there could have been a change to build an algorithm that rearranges those document sections in the correct order by the field priority. Intuitively a headline would have been the most meaningful part of the document. That would have been followed by the ingress and then the rest of the text.

The positive finding was that those sections happened to be in that order in the rich text document almost without exception. For that reason, the least laborious way to transform this document into a text was to strip off all the XML tags and keep only the textual content.

A combination of the "tname" and "headline" fields prepended to the text content ended up being the final format. This was one way to build a uniform representation for all articles where the most meaningful content always becomes first. Below is an example of the source data and the result after applying this step.

```
<richtext version="1.0.0.0">
  <paragraphlayout>
    <paragraph box="1" parstyle="Headline">
      <text>Nokia sai omia parannuks</text><text>ia Windows Phone 7.8
päivitykseen</text>
    </paragraph>
    <paragraph box="2" parstyle="Ingress">
      <text></text>
    </paragraph>
    <paragraph box="3" parstyle="Text">
      <text>Ohjelmistoyhtiö Microsoft on saanut valmiiksi Windows Phone 7.5 -puhelinten
seuraavan päivityksen...</text>
    </paragraph>
  </paragraphlayout>
</richtext>
…
<tname>Nokia parantaa</tname>
<headline>Nokia sai omia parannuksia Windows Phone 7.8 päivitykseen</headline>
```

After stripping the XML markup, the same example takes the following form.

> Nokia parantaa Nokia sai omia parannuksia Windows Phone 7.8 päivitykseen Nokia sai omia parannuksia Windows Phone 7.8 päivitykseen Ohjelmistoyhtiö Microsoft on saanut valmiiksi Windows Phone 7.5 -puhelinten seuraavan päivityksen...

The Keywords field in the articles table holds a comma-separated list of keywords, most of those having only one entry. Dropping all the articles without this field available ended up with less than 300 000 articles left for training. This forced to include more

preformatting steps for the text before building machine learning models. It was expected that there are a lot of different words compared to a number of each word's instances. This makes it nearly impossible to produce any general rules for the keywords, and the model cannot learn the data. The solution was to use regex to strip unwanted characters and whitespaces from sentences in combination with NLTK library. All the steps were wrapped to a small program that exports articles from the relational database and apply the formatting. Below is an example of the same article that is tokenized with NLTK library.

```
['nokia', 'parantaa', 'nokia', 'sai', 'omia', 'parannuksia', 'windows', 'phone', '7.8',
'päivitykseen', 'nokia', 'sai', 'omia', 'parannuksia', 'windows', 'phone', '7.8', 'päivitykseen',
'ohjelmistoyhtiö', 'microsoft', 'on', 'saanut', 'valmiiksi',
'windows', 'Phone', '7.5', 'puhelinten', 'seuraavan','päivityksen']
```

The words are in a list in the same order as they appeared in the original sentence. The next step to reduce the number of different words is to convert each of them to a base form. The library named Helsinki finite-state transducer technology (HFST), provided by the department of modern language at the University of Helsinki was used for this action. That library was referred to in the previous chapter. It provides morphological analyzers and other language-related tools that enable a compelling and reliable way to get the base forms for each word. After applying this for the tokenized article, the result for the same article is in the example below.

```
['nokia', 'parantaa', 'nokia', 'saada', 'omia', 'parannus',
'windows', 'päivitys', 'nokia', 'saada', 'omia', 'parannus',
'windows', 'päivitys', 'ohjelmistoyhtiö', 'microsoft', 'olla', 'saanut', 'valmis', 'windows',
'puhelin', 'seuraava', 'päivitys']
```

HFST library has useful tools to analyze natural language. With this library, it would be interesting to try to calculate weights for the words by finding the most meaningful words based on morphological analysis. Since this thesis's subject is to work from the machine learning perspective, this step is excluded from this paper.

These actions performed to the text reduced the number of different words, but at the same time, part of the data was lost. This thesis's preprocessing part consists of finding the correct balance between improving the quality of the data and losing a part of it.

To measure how the preformatting phase affected the data, a smaller patch with 100 000 articles from another customer was fed through the process. In the first phase, the number of different words was counted from the material after removing special characters and transferring characters to lower case. There were 866857 different words in those articles. The number of instances for each word was from 1 to 492671. Then all the words were transferred to a base form. It decreased the number of different words to 406921. It means that transferring words into a base form narrowed the vocabulary size by 47%. The maximum number of instances did not change.

4.3 Building word embeddings

The first machine learning part of the work started after the data was preformatted to a suitable format. This consisted of training the word embedding model with the Word2Vec technique. This technique is presented with more details in chapter 3.

In this step, all the articles were suitable for the teaching process. Missing or invalid meta fields do not affect this step because they are not used for anything at this phase. This enabled including almost one million articles to build the word embedding models.

After a few iterations of training the word embedding model with a small batch of data, parameters were tuned to 300-dimensional vectors with 32-bit float values. The Skip-Gram method was used for teaching. The depth of the vocabulary was fixed to two million words. A typical starting point for dimensions with natural language embeddings is between 100 and 1000 [7]. After narrowing the number of different words with advanced preformatting, dimensions of that order of magnitude were found to produce satisfactory results, achieving an accuracy that was good enough without causing a time-consuming learning process. Adjusting the size of the dimensions a bit did not significantly affect the result.

Teaching was done with a huge text file containing all the preformatted articles separated by space. All the words were in base form, and all the numeric content and special characters like dots were stripped off. Below is an example of the teaching file containing familiar text content from the same article used as an example in the preformatting chapter.

nokia parantaa nokia saada omia parannus windows päivitys nokia saada omia parannus windows päivitys ohjelmistoyhtiö microsoft olla saanut valmis windows puhelin seuraava päivitys

That file was read programmatically and fed through the teaching process in small batches containing 1000 words. The result was a word embedding model ready to be used. The model was stored in a file for later use. There are multiple solutions to evaluate how successful that model is with the selected parameters.

One example that is easy to understand is to give the model a task to output the word having a distance from the input word i3 that is as close as possible to the distance between input words i1 and i2. The table below explains the results of this task. There are three input words i1, i2, i3, and one output word.

Table 5. Evaluation example for word embeddings

i1	i2	i3	output
mies	kuningas	nainen	kuningatar
suomi	presidentti	ruotsi	kuningas
ihminen	jalka	koira	tassu
suomi	helsinki	viro	tallinna

Occasionally, the outputs from the model turned out to have no sensible meaning. This table contains only examples that humans can understand in order to emphasize the aim behind this task. With this in mind, concatenating articles next to each other causes word embedding to learn that the last word from article n and first word from article n+1 has a connection. This should not be an issue for two reasons. First, the number of these cases is minimal because an article usually contains many words. The second reason is that the application related to this thesis cannot be significantly affected by this because the neural network used to learn the articles mainly focuses on the different words and the number of those words instead of focusing on words' relations or the content itself.

4.4 Mapping keywords

As the earlier chapter explained, it is only possible to input numerical data to a neural network. For this reason, keywords must be in a numeric format too. In the original material, those were a comma-separated list of words as a plain text.

The solution was done with some traditional programming added as a first step in the neural network model's learning process. In this step, all the keywords were converted to a numeric id based on the following conditions.

When a keyword was entered, the program checked if there is already an id for the keyword. If one exists, it was returned, and if it was a new keyword with no existing id, it generated one by incrementing a numeric value by one and returned that number. This id mapping was stored to a file using TensorFlow saver, enabling conversion from a keyword to keyword id and conversion from a keyword id back to the actual keyword.

The mapping must support effective two-way conversion for later use because the neural network's outcome needs to be converted back to an actual word.

The only preprocessing activity performed in the first phase was to lowercase all characters. It turned out later that the neural network only reached weak confidence. After several attempts to adjust the shape of the network and optimizing the learning process by using regularization techniques like dropout, the reason for the symptom was too many keywords compared to the number of articles. There was an attempt in which all the compound words were broken up into multiple keywords to prove that this caused the actual problem. An example of a compound word is “kotimaanmatkailu” which was split into words “kotimaa” and “matkailu”. This idea came from the fact that a surprisingly large number of the keywords were actually compound words. Unfortunately, this did not affect the results.

4.5 Building a neural network

This chapter presents the shape of the network and describes the training process. There is more discussion in chapter 5 that focuses mainly on the results.

All the articles were transferred to a new line delimited JavaScript object notation format (NDJSON). It merely means that each line contains an article text and keywords. Article text was formatted as represented in the preformatting chapter. Below is an example of the document containing two articles. The number on the left represents the line number.

1. {”text”: ”nokia parantaa nokia saada omia parannus windows päivitys nokia saada omia parannus windows päivitys ohjelmistoyhtiö microsoft olla saanut valmis windows puhelin seuraava päivitys”, ”keywords”: [”tekniikka”]}
2. {”text”: ”hallitus aloittaa keskustella…”, ”keywords”: [”politiikka”]}

Next, the data was fed to a model. The maximum length for the text on an article was fixed to 1000 words. The limit is set in order to bring all the input articles to have identical form, and the reason for the specific number is that most news articles fit in that number of words. Those 1000 words were taken from the beginning of the article because that part includes the most important words.

First, the article's words went through the previously-stored word embedding model, and keywords went through the conversion routine. At this stage, each article is a list of word vectors with a list of keyword identifiers.

Next, the actual training process begins. After several test iterations, the neural network was shaped to have one-dimensional convolutional layers combined with dense layers. First, there were three identical convolutional layers, each containing 256 filters. The number of filters was decided because there were 300-dimensional word vectors used with the embeddings. According to a conversation with a machine learning expert, a useful rule of thumb for choosing the number of filters is to pick a number of the same order of magnitude as the embedding dimension. A small adjustment to the number of cells did not affect the results.

Each convolutional layer used window size 5. The learning data contained articles of different lengths. In other words, articles contained a different number of words. From that fact came the idea to use convolutional layers on top of the network because that enables a possibility to handle only a fixed-size window. Zero-filling shorter articles removed the requirement to use fixed-size inputs.

Each convolutional layer used ReLu as an activation function, and on each of those three layers, max pooling was applied with a pool size 5.

These three convolutional layers are followed by two dense layers, the first of which has 1792 cells. The size is the same as the output size from the last convolutional layer after flattening the output. This layer provides a connection between convolutional layers and the final dense output layer. The connection layer used ReLu as an activation function. The last layer was a dense layer with the same number of cells as the number of different keywords. It used SoftMax as an activation function. All the cells at the output layer give a float value for one keyword. Using the same number of cells with the actual output from the output layer is a common strategy for building multi-class classifier models. Figure 10 below visualizes the network used in this work.

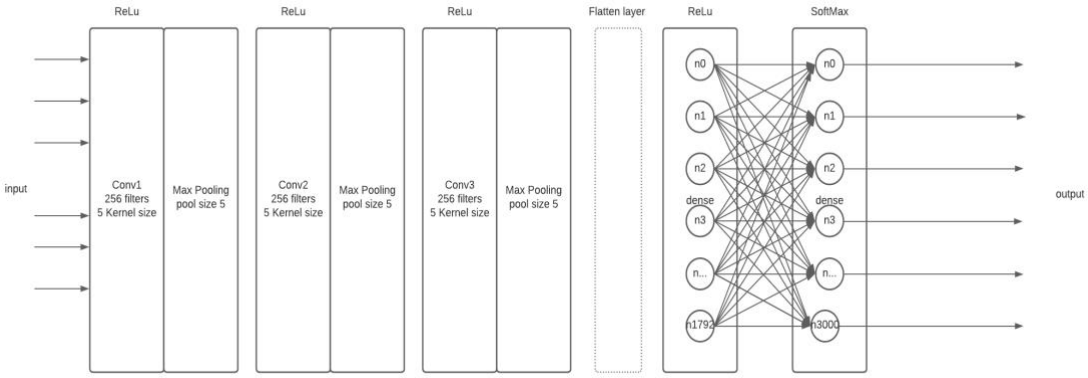

Figure 10. Artificial neural network used in this work.

The input for the network is a vector containing 300 000 cells. This number of inputs is a result of having a thousand words multiplied with word embedding dimensions. Word count for each article is limited to a thousand words, and shorter articles are zero-filled. After the first convolutional layer, the data is $996 \cdot 256 = 254\,976$ sized vector. After the first max pooling, the data is in a vector having $199 \cdot 256 = 50\,944$ cells. The second convolutional layer outputs the data in a vector with a size of $195 \cdot 256 = 49\,920$, and when max pooling is applied, the data is in a vector with size $39 \cdot 256 = 9\,984$. The third convolutional layer shapes the data to a vector with size $35 \cdot 256 = 8960$, and after the max pooling, the data is in a vector with size $7 \cdot 256 = 1792$. Data is flattened for the dense layers. The first dense layer does not change the shape because the layer has 1792 cells. The final layer produces the output in the form of a 3000-dimensional vector,

each cell presenting a keyword. After the SoftMax, the output can be simplified to keyword identifiers with a float number for each of those from 0 to 1, presenting a probability of the keyword for input text. These can be referenced as predictions.

4.6 Building a test application

The test application was built with a small Python HTTP server framework named Express. The whole package was wrapped with Docker to simplify the delivery process. The models were mounted into the Docker container from outside on the Docker start. This enabled access to the models providing a way to replace those with an external program. Figure 11 below visualizes the test application.

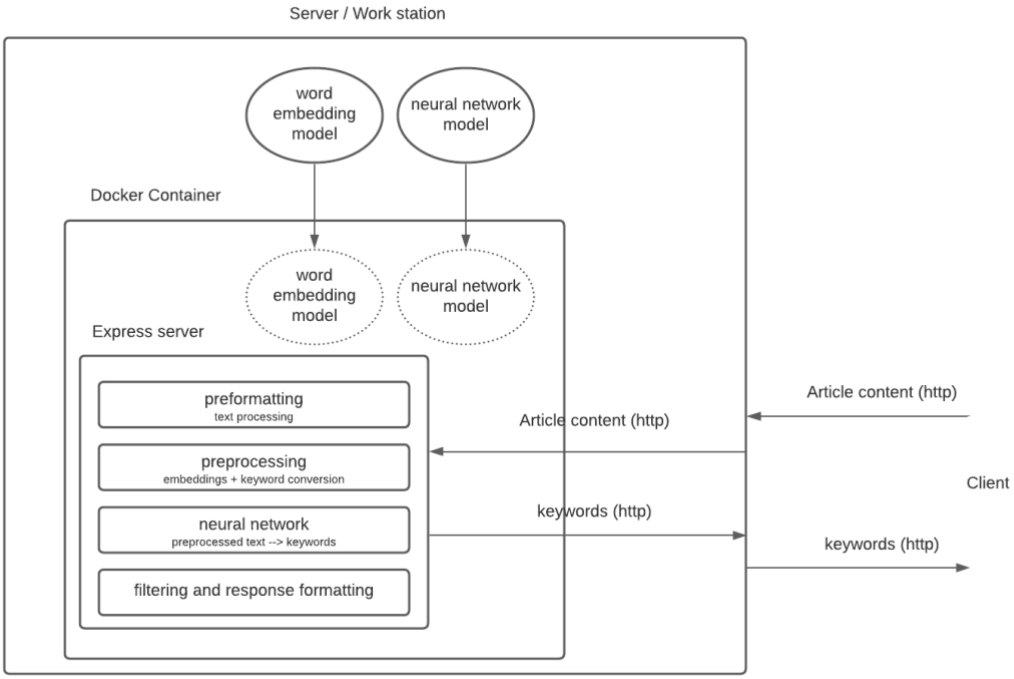

Figure 11. Test application for text classification.

4.7 Data flow in the outcome

This chapter presents the complete picture of the outcome and describes the data flow. In figure 12 below, the process is visualized.

An article text content goes with HTTP from the client to the express server and travels through the preformatting and preprocessing layers. After the initial preprocessing is completed, the preprocessed text content is passed to the word embedding layer. Word vectors go to a neural network model that outputs the answers. The neural network output goes to the filtering layer, which outputs a keyword if any satisfies the filtering threshold. The result is given as a response to the HTTP request through the express server. The whole application is wrapped into a single Docker container.

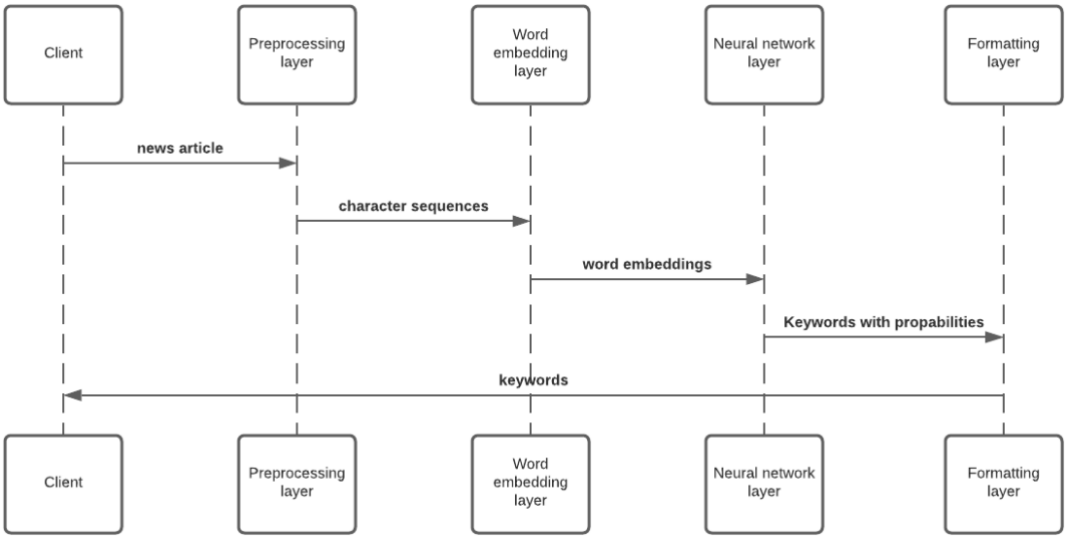

Figure 12. The outcome’s workflow.

5 Results and analysis

This chapter explains the results and analyzes the suitability of the data and the techniques for this work.

The biggest issue that affected all the phases was that only a restricted amount of the available article data contained the keyword fields relevant for the task studied in this thesis. That made teaching the machine learning models difficult and probably affected the results.

5.1 Preformatting

Tools used for preformatting text worked as expected. There were hardly any issues related to the libraries used in this thesis. The FST-based library provided with a pre-trained corpus by Helsinki university worked significantly better for the Finnish language compared to the NLTK library. For example, getting base form for each word with NLTK library got to a 46% of successful results, and the FST-based library was extremely accurate with 99% of correct answers. This evaluation was done with a small batch containing 200 words. Results were compared with Microsoft Excel.

Many of the Python libraries came from the company because Python is one of their main programming languages. The driver used for connecting to the relational database and libraries used to process XML documents are examples of those libraries. Due to previous experience with C and Python programming, this work proceeded with no programming issues.

The preformatting phase was a large part of this work. Finding those solutions to complete that phase came mainly for the reasons caused by the data. In the beginning, there was no plan for any text preformatting steps, but the results revealed a need for improved data quality.

5.2 Preprocessing

Tools used for preprocessing worked for all tasks as planned. Word2Vec model was easy to build and save. After the model was created, it worked with TensorFlow with no issues. Example implementations with many comments by Facebook and Google helped to build that model.

The model was tested with a similar task that is represented in chapter 3. The word embeddings model resolved the word o1, which is the nearest word with the same distance from i3 as the distance between words i1 and i2. The task was executed with two models. The first one only contained the text from the articles, and the second one was taken beyond that, including a batch (30%) of the content from the Finnish Wikipedia dump on 22.2.2019. Words from Wikipedia were converted to a base form, so the results

are comparable. It is essential to know that both embedding models can only work correctly with words in a base form.

There were 50 samples collected from the Internet and people. All of them included the correct answers and were converted to the base form that was verified with the HFST library. The model, with no Wikipedia content, got 38 correct answers with those examples. With Wikipedia content included, there were 43 correct answers. In this case, the correct answer refers to the solution that humans would intuitively end up.

5.3 Neural network model

At the training phase, the process writes the accuracy for each keyword to the file. Part of the keywords had an accuracy of up to 0.96, and the worst accuracies were nearly zero. Attempts to optimize the network using different shapes did not significantly affect these results. Accuracy is calculated, as explained in chapter 2.3.

The training set, containing 80% of the articles, is split into 200 batches, each containing approximately 1200 articles. Accuracy and loss were printed for each batch. Three models were trained with the same data but using different preprocessing and text preformatting. Charts were built for each model with Microsoft Excel using the printed outputs to show the results achieved by this model with its preformatting. The x-axis for all charts uses the same scale, and it represents the number of batches that are run through the training process. Attempts to use Gradient Descent and Adam optimizers led to similar results as without optimizers.

The first attempt used word embeddings built from the article’s texts without any text preformatting. Articles were fed to the learning process without preprocessing text. Therefore, text content includes all the special characters and uppercase letters. The results are visualized in figure 13. The y-axis scale in this figure has been limited to clarify the graph.

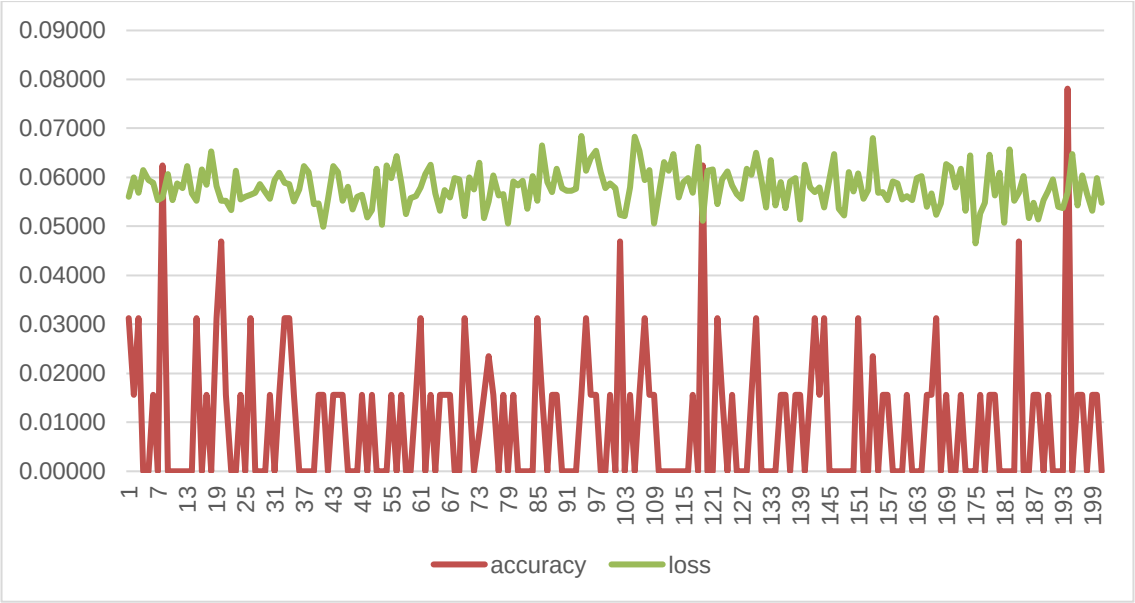

Figure 13. Teaching process with no preformatting, including only word embeddings built from news articles text content.

The results from this first phase were as expected. The neural network was not able to find any general rules between content and keywords. At the end of the training process, the mean accuracy for keywords was 0.62, and the mean F1 score was 0.34.

Running test data through this model did not get any results that would satisfy the requirements. The fixed filtering threshold value needed to be 0.2 to get any keywords out from the model. With the labeled testing set, containing 20% of the data, only 7 % of the articles were answered correctly. Test data in this step contained articles with text content that was not preformatted. The same 20% for testing and 80% for teaching distribution was used in all three cases.

The text was used with no preformatting in the next phase, but word embeddings were improved by including part of the Finnish Wikipedia content. This led to some improvements in the results during the training process. However, at some point in training, the accuracy started to decrease, and shortly after that, the loss increased sharply. This might be because of the complexity of the text data. There are still many different words compared to the number of articles. The results for this step are represented in figure 14.

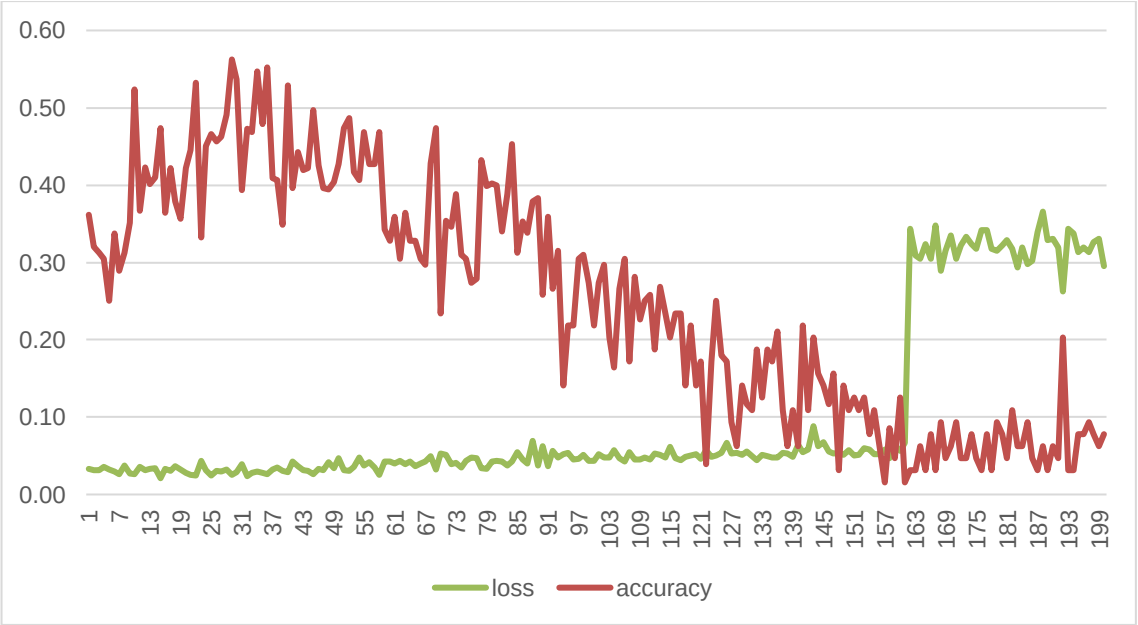

Figure 14. Teaching with no preformatting and Wikipedia content included in word embeddings.

At the end of the training process, the mean accuracy for keywords was 0.87, and the mean F1 score was ~0.44.

Running test data through this model was an improvement compared to the first attempt. The fixed threshold value used for filtering keywords was adjusted to 0.3 to get predictions for most of the articles. With the labeled testing set, 37% of the articles were assigned with correct keywords. Test data in this step contained articles having text content without preformatting.

The third phase was built with preformatted text, and words brought to base form were used to build the word embeddings model. Word embeddings included all available news articles and content from Wikipedia dump. The same training set was used, but now the full preformatting was applied to textual content.

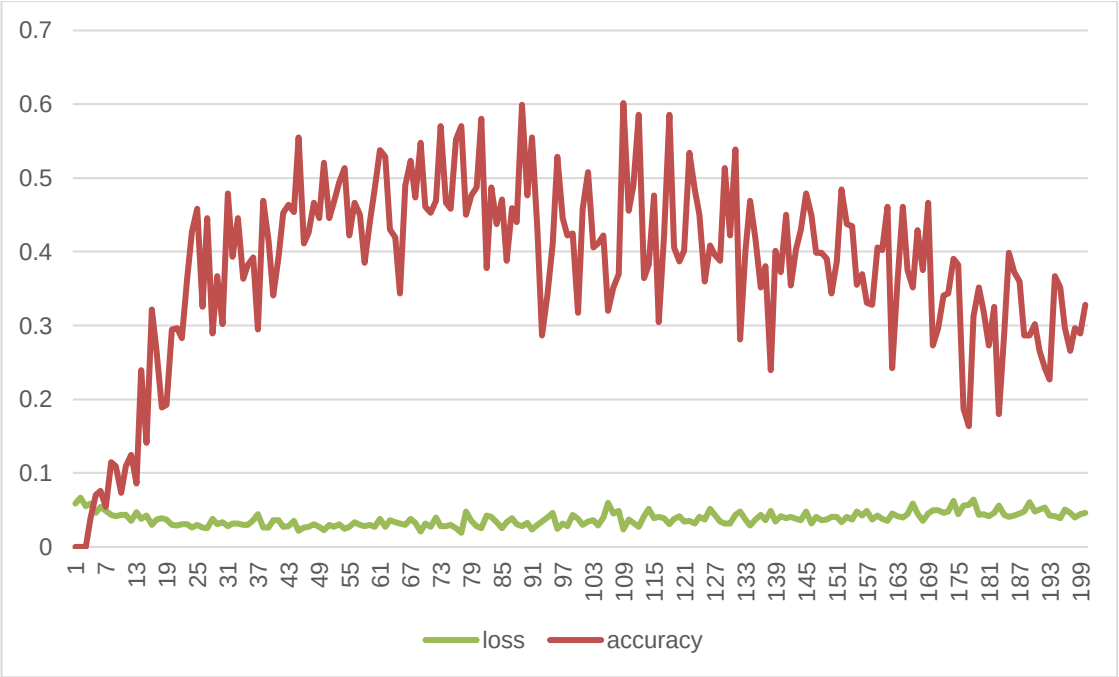

Figure 15. Teaching process with preformatted content and preformatted word embeddings.

At the end of the training process, the mean accuracy for keywords was 0.99, and the mean F1 score was ~0.55.

Running test data through this model was an improvement compared to the previous attempts. The fixed threshold used for filtering keywords was adjusted to 0.5 to get the keyword for many of the articles. Theoretically, with the 0.5 thresholds, SoftMax is only able to output a single keyword at most. With the testing set, the model predicted 57% of the articles correctly.

There is still much scattering between keyword accuracies. That can be seen from the log that printed the accuracy for each keyword and from those figures presenting the output from the batches during the learning process. Some keywords on the material were nonsensical for this use case, like “jääkiekko 2017”, which is only valid for one year and cannot be used for further articles. The biggest issue was that some articles contained keywords, and some similar articles that should have the same keywords were missing those. For example, only ten of the hundred articles about movie reviews actually contains the keyword "elokuva-arvostelu" (movie review). For this particular problem, this application might provide a solution in the future, together with a semi-supervised classifying process.

6 Conclusions

This chapter presents the main conclusions and proposes further development ideas.

6.1 Conclusions about preprocessing

The data used for teaching the word embeddings model was suitable after the advanced preprocessing phase. However, it would have been interesting to try to drop off the reformatting step - at least the part where words are converted to the base form. This would have allowed learning the whole sentences or complete chapters, letting the neural network to learn the text's actual meaning. Learning the order of words and the sentence's actual content would give a possibility to learn the text's meaning instead of essentially learning only the words and number of instances for each word.

In conclusion, word embeddings worked reasonably well for this task, but more data would have improved the results. Word embeddings may work better when there are actual connections between words that are not converted to a base form. This would have required much more training data.

There are downloadable word embeddings models ready for use. For example, a project by Facebook called FastText contains support for 157 languages, including Finnish. Those models were trained using CBOW with position-weights in dimension 300, and those are fully operational with Python because of an included module. Using a ready-built model would have reduced the number of steps done in this work. Competing against those models is challenging. However, in this thesis, the goal was to use the available data to teach models. For this reason, the solution represented in this work, including preformatting, can be regarded as a successful part of the outcome. It also reduces the data requirements, enabling companies with small amounts of data to use the model.

6.2 Conclusions about the neural network model

It would have been interesting to switch the network completely to the RNN model. RNN networks store an internal state, and this would have taken into account all sequential

information included in the data. This means, for example, learning whether some words always occur in the same order. It would provide a possibility to split the article into sentences represented as a collection of word vectors and give each sentence at a time as an input to the network.

The current solution's implementation was simplified by assuming that the specific keyword is connected with some words in the article, which is not entirely accurate. However, this improved the results in this case. The available data did not encourage further model development. In the end, the result is better than expected at the beginning of the project.

This thesis is built to resolve a single keyword for an article. It is quite straightforward to transform the model to predict multiple keywords. The only unsuitable part is the neural network model with its output filtering. With SoftMax as an activation function, the output layer can only give zero to one keyword. This is because the sum for all the keyword probabilities is always one. Updating the output filtering may provide a way to solve multiple keywords for an article. In the current solution, it was adjusted to have a threshold of 0.5. A better solution would be updating the output layer completely. This requires at least updating the activation function to Sigmoid and analyze what would be the best loss function for this use case. The Sigmoid function is essential to be used for the logistic regression model. A logistic regression model is used to estimate the probability of a binary event (such as being healthy or sick) or, generally speaking, whether the sample belongs to a class or not. With the Sigmoid activation function, the output would be a list of zeros and ones, each representing a specific keyword. No further filtering is required with this approach. Running the learning process using this kind of model was also attempted with log loss, but the results were the same as earlier. This may be because most of the data had only one keyword.

A neural network presented as the solution in this thesis has at least one issue that must be kept in mind for future improvements and the integration phase. The network needs to be completely rebuilt when the number of keywords changes. This is mainly because the number of cells at the output layer is fixed to be the same as the number of keywords. This is a commonly used technique to build multi-class classifier neural networks. The whole model could be replaced with an unsupervised model that would discover the keywords with no labels. Alternatively, reconfiguring the output layer, including its loss function and activation function, may provide a way to continue teaching without

rebuilding the whole model. Instead of being worried about the model rebuilding, it might be more important to keep in mind that the performance of a neural network improves with more data. The data often needs to be fed through the learning process several times for the best results. With this in mind, it would be more important to focus on making the learning process as effective as possible than being worried about rebuilding the model in certain situations.

One way to speed up the learning process is to reduce the number of dimensions on the word vectors. As mentioned in the results section, small adjustments for the dimensions did not significantly affect the results. When the embedding dimension was reduced to one third, the training process proceeded with twice the speed.

6.3 Integration design and continuous learning strategy

The integration to the CMS is done with the HTTP interface. This part is the same as represented in the “Building a solution” chapter with the demo application. Article text is sent from the CMS user interface as an HTTP request to the server, and the response contains the keyword(s). This is done at the UI level because, according to the results, there may be a need for manual validation.

New articles are included in the system periodically. They reside in a relational database. The same code used to build the models queries articles that are new or have been updated since the previous execution. Preformatting tasks are applied to all articles. Next, the word embeddings model is updated with the articles. Articles are formatted to a suitable format to be appended to the NDJSON document. The neural network learning process is executed. Existing models are replaced with the new ones, and the previous models are kept as a backup. Then the server providing an HTTP interface is requested to reload the models. Visualization for the complete integration is in figure 16.

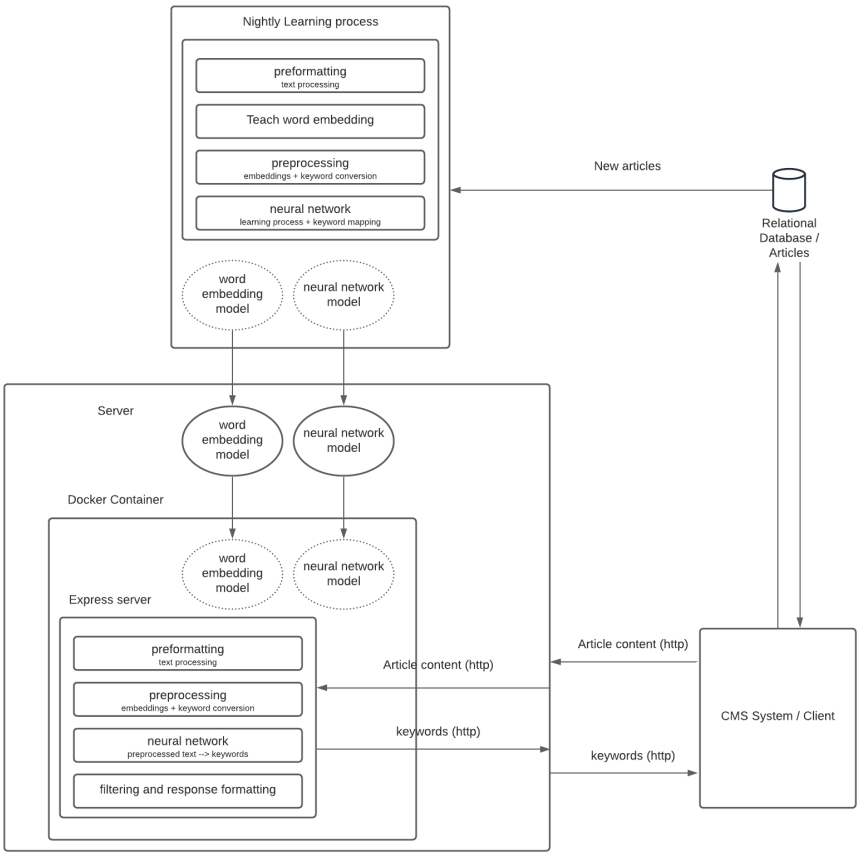

Figure 16. Design to integrate the outcome with the company's CMS.

References

1 I. Sutskever, T. Mikolov, K. Chen, G. Corrado and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” 2013. < https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf> [Accessed 2.12.2018]

2 R.J. Solomonoff "An Inductive inference machine" 1965 <http://raysolomonoff.com/publications/indinf56.pdf> [Accessed 3.11.2020]

3 J. Breuker and others “Emerging Artificial Intelligence Applications in Computer Engineering”. 2014

4 Jeans Kober, J. Andrew Bagnell and Jan Peters, “Reinforcement learning in robotics: A survey” Article 2013

5 Y Chen and others “An unsupervised learning approach to content-based image retrieval” <https://ieeexplore.ieee.org/abstract/document/1224674> [Accessed 3.11.2020]

6 Shaukat Hayat and others “A Deep Learning Framework Using Convolutional Neural Network for Multi-Class Object Recognition” <https://ieeexplore.ieee.org/document/8492777> 2018 [Accessed 15.9.2020]

7 Tomas Mikolov and others “Distributed Representations of Words and Phrases and their Compositionality” <https://arxiv.org/pdf/1310.4546.pdf> [Accessed 10.9.2020]

8 Jonathan Schmidt and others “Recent advances and applications of machine learning in solid-state materials science” Article at “npj Computational Materials” 2019

9 Jason Brownlee “Linear Regression for Machine Learning” 2016

10 V.N. Phat and H. Trinh Exponential Stabilization of Neural Networks With Various Activation Functions and Mixed Time-Varying Delays <https://ieeexplore.ieee.org/document/5484431> 2010 [accessed 3.10.2020]

11 Andrew Ng “Improving Deep Neural Networks: Hyperameter tuning, Requlariza tion and Optimization” 2009

12 Shuntaro Okada and others “Efficient partition of integer optimization problems with one-hot encoding” published at scientific reports [Article number 13036] 2019

13 N. Shukla ”Machine Learning with TensorFlow” Manning Publications Co. Kindle Edition. pp. 284 - 324.

14 Jedd Hale “Deep Learning Framework Power Scores” <https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a> [Accessed 30.11.2020]

15 Kenneth R. Beesley “Finite-State Morphology” 2003

16 Amr Kelg and others “An unsupervised method for weighting finite-state morphological analyzers” <https://www.aclweb.org/anthology/2020.lrec-1.474.pdf> [accessed 3.11.2020]