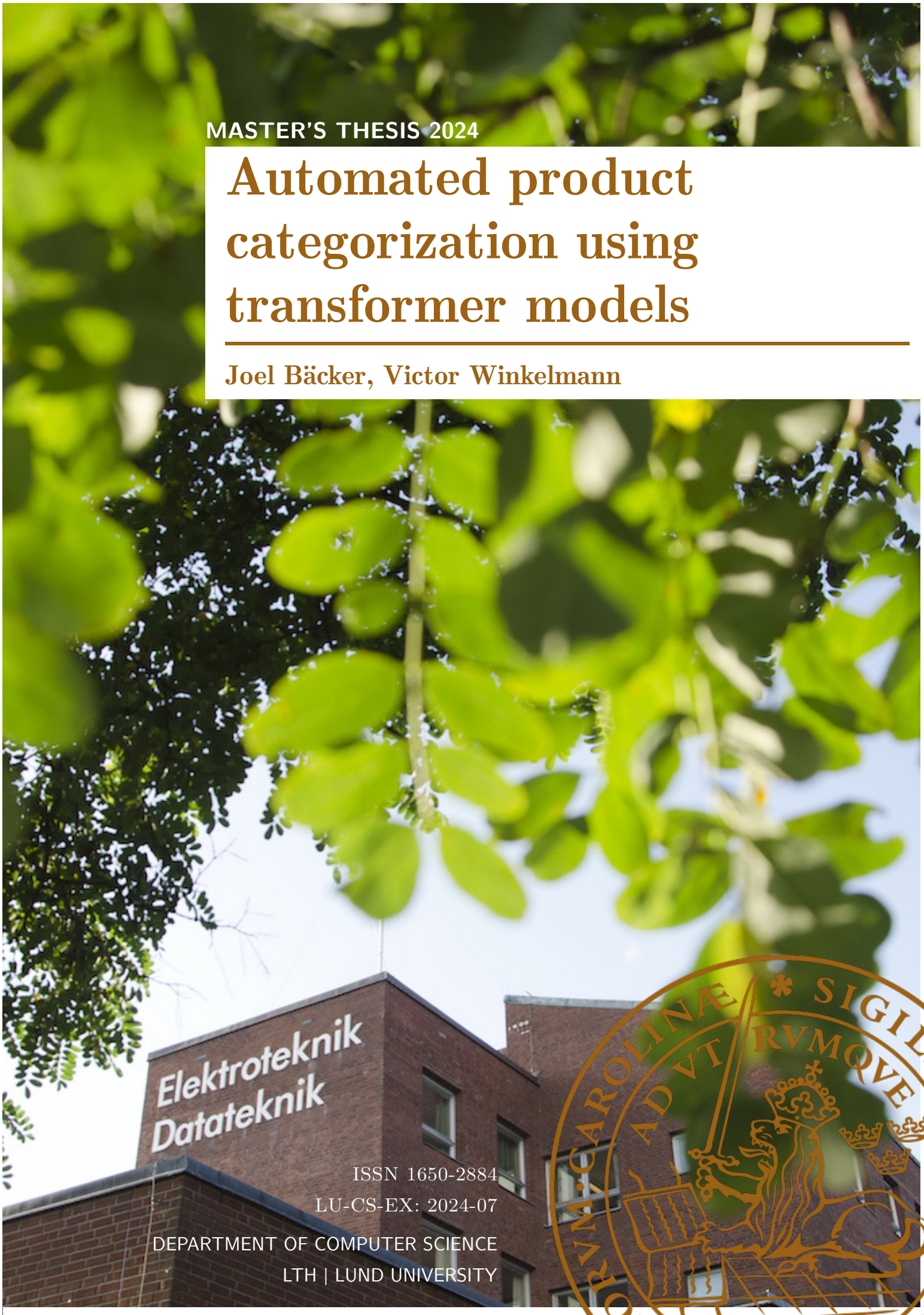


MASTER'S THESIS 2024

Automated product categorization using transformer models

Joel Bäcker, Victor Winkelmann

ISSN 1650-2884
LU-CS-EX: 2024-07

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-07

Automated product categorization using transformer models

Automatiskt produktkategorisering genom transformermodeller

Joel Bäcker, Victor Winkelmann

Automated product categorization using transformer models

Joel Bäcker
jo4383ba-s@student.lu.se

Victor Winkelmann
vi6253wi-s@student.lu.se

February 2, 2024

Master's thesis work carried out at Theca Systems AB.

Supervisors: Marcus Klang, marcus.klang@cs.lth.se
Rasmus Ros, rasmus.ros@theca.com

Examiner: Krueger Volker, volker.krueger@cs.lth.se

Abstract

This thesis explores the effectiveness of fine-tuning BERT models for product classification. There is a growing interest in enabling accurate automatic product hierarchical categorization following the rapid growth of e-commerce. This work explores the effectiveness of BERT models in the Swedish e-commerce sector, leveraging the models renowned capabilities in language understanding. The pre-trained BERT models used in this work where Distilled KB-BERT.

This thesis has concluded that a fine-tuned BERT model with a data set specifically designed for product classification shows promising results. With an average precision of 82% across multiple product categories. The fine-tuned BERT models demonstrated the ability to capture meaningful representations and to make accurate predictions for classifying products into their respective categories.

However, this study acknowledges the need for further investigation into the generalization capability of fine-tuned BERT models. While the accuracy achieved in our experiments is encouraging, it remains unclear how well these models perform when applied to unseen or out-of-domain product data. Future research should focus on evaluating the robustness and generalization of fine-tuned BERT models across different product taxonomies.

Keywords: Machine Learning, NLP, automated categorization, hierarchical categorization, BERT

Acknowledgements

We want to thank our supervisor at LTH, Marcus Klang for all the valuable input, guidance, and support.

We would also like to extend our gratitude toward Theca Systems AB and especially Rasmus Ros for being our supervisor and helping us during the project. We are also grateful for the hardware that was provided to us, without it, this thesis would never have been possible.

Contents

Chapter 1
Introduction

1.1 Background

Classifying product titles/descriptions into an appropriate category is a crucial part of many industries, for example, e-commerce. Effectively organizing products not only facilitates more efficient navigation for customers but also enables a company to conduct a better analysis of consumer behavior. Categorizing products according to some pre-determined taxonomy is a time-consuming task and even more so the impossible task to re-categorize every product if the taxonomy is changed. To address this challenge, the application of artificial intelligence (AI) based methods could be a suitable solution. By the use of algorithms and state-of-the-art machine learning models, AI has the potential to learn complex relationships between text and categories and thereby automate categorization.

Imagine being able to search for a product by describing it and getting results for all products that could belong to that type of text. This is another example of how AI could help create a more comfortable browsing experience.

If one searches for *Vatten- och isdispenser* on Elgiganten's website[1] a large number of relevant products is returned in the search result. However, if you search for *Kyl med vatten och is* only one relevant product can be found. With the use of AI both searches could get a similar range of relevant search results. AI implementation would also enable companies to re-classify a product to another category structure. This could help retail companies lower their expenses in terms of marketing now spent on Google's ad services (Lindberg and Facht, 2022). Such a model would enable companies to send potential companies directly to the most relevant site without the need to rely on Google's services. This work set out to explore the possibility to implement an AI model that could solve such tasks. It also aimed to develop, train and evaluate models that could accurately classify products according

[1] `https://www.elgiganten.se/`

to some taxonomy. Various factors such as accuracy, requirements, scalability, and generalization are taken into consideration while designing the solution. Potential challenges, limitations, and ethical aspects are also discussed.

In conclusion, the automation of product classification holds tremendous potential for enabling a better consumer experience, a lower amount of tedious manual labor and lower expenses to tech giants such as Google. By overcoming these limitations, this research aims to contribute to the growing field of AI-driven tools that could help to streamline companies' product management.

1.2 Problem Statement

This thesis aims to explore how an AI-based model could be constructed and implemented to automatically generate categories for product text from a given taxonomy. A taxonomy is a method of structuring categories in a hierarchical way by pairing parent and child categories. The idea of using a taxonomy is that a parent category is broader than the child, which is more specific. This helps us group similar things and separate those that are different. Using a taxonomy to structure relationships between parent and child categories is common in, for example, web-based stores. The resulting model should ultimately be able to produce ordered categories in a hierarchical manner, similar to a web-based store.

To develop a model capable of generating hierarchical categories, fine-tuning of a large language model (LLM) is carried out. Fine-tuning is the process of adapting an LLM to a specific task using a vast amount of product data. The data is collected from several different stores, all of which are among the largest e-commerce platforms in Sweden. During fine-tuning, some of the data will not be seen and will not be present in the taxonomy; this is done to assess the model's performance on products and categories it has not encountered before. The model can predict unseen categories thanks to the methods employed during fine-tuning, such as zero-shot learning. By leveraging this data, the project will explore which model architecture is most suitable for category prediction. The research aims to be able to answer the following questions:

1. How well can a Machine Learning model predict the category to which a product belongs in different stores with different taxonomies?

2. How does publicly available technology/models perform when compared to our proposed model?

3. What is the performance on unseen stores and taxonomies that are not part of the training set?

1.3 Scientific Contributions

Previous research has shown successful methods for accomplishing automatic hierarchical classification on single-domain products and categories. One such example

is presented in *'Improving pre-trained models for zero-shot multi-label text classification through reinforced label hierarchy reasoning.'* (Liu et al., 2021). In this paper, however, the goal is to expand upon these methods in order to be usable for cross-domain categorization with different taxonomies. This implies that new products from known or unknown domains can be automatically categorized once the model is created.

1.4 Related Work

The approach for the report is to use a transformer-based model called Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019a). As mentioned previously has there have been successful methods to solve parts of the problem, one such is solution is Liu et al. (2021), where only one domain is analyzed at once. This report aims to expand the method in the paper in order to get it to work for several domains.

1.5 Limitations

Even though a large language model like BERT can understand and use several languages, the data in this project will only be in one language: Swedish. Constructing a model in another language or one that is multilingual, i.e., understanding more than one language, is far from impossible. However, considering the scope of this work, which aims to construct the framework for hierarchical product categorization, only one language is used.

Before the data set was created, containing product information fetched from some of the largest e-commerce platforms in Sweden, a search was conducted to find another one. The search yielded nothing that fit into this thesis problem; thus, no benchmarking and comparisons were possible. Comparing the results from the models with something similar might be challenging, as a proper baseline data set was not available at the time.

1.6 Contribution statement

The project was designed and written by Joel Bäcker and Victor Winkelmann, see Tab. 1.1 for detailed contributions.

Table 1.1: Contribution statement.

	Joel Bäcker	Victor Winkelmann
	Contribution [%]	
Data preparation	60	40
Baseline	40	60
Liu et al. (2021) model	40	60
Other transformer models	40	60
Report	40	60

Chapter 2
Theory

This chapter will provide relevant theory about the model that is used for automatic categorization.

2.1 Machine Learning

Computer problem-solving requires an algorithm, a recipe of instructions that a machine can follow in order to produce a solution to the problem at hand. Algorithms can be used to solve problems like the shortest path in a graph, e.g. by using Dijkstra's Algorithm, or sorting a list of numbers, maybe using Quick Sort (Dijkstra, 1959; Hoare, 1961). However, there are problems that prove to be harder to construct a set of instructions, i.e. it is hard to construct a function that can transform the input into a desired output if both are not given explicitly.

Instead of explicitly writing an algorithm to solve a problem, we would like a program to extract the features of an input and associate those with an output. Given enough data, the idea is that a machine could hopefully learn the most important parts of an input and relate these to the correct output and to also make accurate prediction on both similar and new data. This is called machine learning (ML) (Alpaydin, 2010).

The typical steps to solve a machine learning problem is to first find or create data for the problem. Data could be, e.g. time series data, text, pictures, numbers etc., but it needs to be relevant for the problem at hand. From this, the data is prepared so that an ML model can be trained on the gathered data. During the training phase some parameters are tweaked in order to maximize the performance of the model. A part of the data goes to model training, often it is most of the data, another part of the data, i.e. not seen during training, is used for model evaluation (Mitchell, 1997; Brown, 2021).

Depending on the task and data that is available for the problem a ML strategy

needs to be picked, where the most common are Supervised, Unsupervised and Reinforcement Learning.

2.1.1 Supervised Learning

Supervised Learning is a ML paradigm where an algorithm is trained on a labeled data set, i.e. a set that contains pairs of input and correct labels. This could be pictures of animals where the animal in the picture is the label, or handwritten characters where the label is the correct character (Brown, 2021). During training the algorithm identifies patterns in the data and tries to optimize the predictions to be as close to the correct ones. Evaluating the model's performance is done on data that have not been seen during the training phase.

2.1.2 Unsupervised Leaning

Instead of telling a machine what patterns or answers to look for, the program attempts to do it automatically. For instance, let us say we have some sort of sales data, the patterns present in the data set can sometimes be hard to extract and/or find. Thus, the machine can be used to find patterns in the data, it could perhaps find what type of person buys what products or shopping trends such as a correlation between buying two products or holiday shopping trends (Brown, 2021). Here, the main difference is that the data has no definite right answer (unlabeled), the goal of the model is used to find trends and relationships within the input data.

2.1.3 Reinforcement learning

Reinforcement learning (RL) differs from the other methods since it is based on a trial and error system and not needing a labeled data set. The model is instead trained by optimizing a reward function, the reward is calculated from some expression of the current state. The state is some sort of environment to which the model is connected to. Depending on the usage of the model the environment could be a video game or roads used for traffic. In these cases, the reward function could be how far the model gets from the starting point, and the goal is to reach some goal or target.

In addition to the state and reward function the available actions that the model can take also need to be provided. The chosen action for some state during training will affect the state during the next iteration. If the model is trying to play a video game then the action it could take would be the different buttons and other inputs from a controller (Brown, 2021; Kaelbling et al., 1996).

A reinforcement learning process can be summarized into five step:

1. Observe state
2. Choose an action
3. Interact with the state

4. Get reward

5. Repeat

2.1.4 Train, Validation and Test

Before constructing any models, it is often good practice to divide the data set into three parts, which are used during different stages of model creation, as suggested by Ng (2023) (note that this is not applicable for RL). These three parts are:

1. train, used when fine-tuning a model in order to tailor it to a task,

2. validation, used to tune the hyper-parameters in a model to ensure the best performance on the final model

3. test set which is used to evaluate the performance of the model, not seen during training.

To make these parts different criteria are put upon them to maximize the efficiency of the corresponding part. A typical consideration taken when making these are e.g. randomizing or random sampling the data in order to mitigate any potential relation between two data points.

2.1.5 Loss Function

ML algorithms always involve some sort of optimization, referring to the task of either maximizing or minimizing some sort of function or state. When training a neural network, like in a NLP setting, it is called *Loss* function or objective function. This function measures how well the model is performing. There are many different types of loss functions, and they are dependent on the type of problem. Binary Cross-Entropy Loss (BCE) is a loss function that is used for binary classification tasks and Categorical Cross-Entropy (CCE) is used for multi-class problems (Goodfellow et al., 2016).

2.1.6 Top-k Learning

Often when performing ML tasks only one prediction is accepted. Top-k aims to extend this criterion to accept k predictions instead. This method can be used in this project to predict hierarchical categories by doing it one step at a time. A model would predict the best k categories given input and the possible next steps (Petersen et al., 2022).

This is visualized in Tab. 2.1. In the top-1 scenario, only the first prediction is counted as correct. In the top-3 scenario, it is considered correct if the right answer is within the first three predictions. Similarly, for top-5, the answer is counted as correct if it appears within the first five predictions.

Table 2.1: Examples of Top-k for various values of k. Correct guesses are highlighted in bold. Predictions are ordered according to the most probable class as determined by some model. The 'Yes/No' column indicates whether the predictions are included within the corresponding top-k.

Input	Predictions	top-1	top-3	top-5
Dog	**Dog**, Cat, Bird, Rabbit, Fish, Mouse	Yes	Yes	Yes
Cat	**Cat**, Bird, Fish, Mouse, Rabbit, Dog	Yes	Yes	Yes
Bird	Cat, Dog, **Bird**, Fish, Mouse, Rabbit	No	Yes	Yes
Rabbit	Fish, Cat, Mouse, Bird, **Rabbit**, Dog	No	No	Yes
Mouse	Rabbit, Cat, Dog, **Mouse**, Bird, Fish	No	No	Yes
Fish	Bird, Rabbit, Dog, Mouse, Cat, **Fish**	No	No	No

2.1.7 Tools

There are many tools available for constructing ML based model. For this thesis the implementations where made in Python and used the packages mention in this section below.

PyTorch

PyTorch is a machine learning library that shows that two goals are in fact compatible: enabling GPU accelerated training and therefore increasing the speed of training whilst still maintaining a user-friendly interface. PyTorch is one of many popular frameworks. An alternative is TensorFlow developed by Google. PyTorch is a framework developed to train various types of neural networks (Paszke et al., 2019).

Scikit-learn

Scikit-learn is a Python module that integrates a wide range of state-of-the-art machine learning algorithms. It can be used both for supervised and unsupervised problems at a medium scale. The idea behind Scikit-learn is to provide state-of-the-art implementations of many well-known algorithms for classification, regression, clustering, etc. whilst still keeping a user-friendly interface (Pedregosa et al., 2011). Scikit-learn is not GPU enabled and therefore used for medium-scaled problems that have limited complexity. Components used from SciKit-learn:

- MultiLabelbinarizer - transforms the data into a binary format. A binary matrix where the column represents a unique category and the rows represent a unique instance
- OneVsRestClassifier - A machine learning strategy that is used for multi-class classification problems. This method allows you to transform a multi-class problem into multiple binary classification problems, one for each class

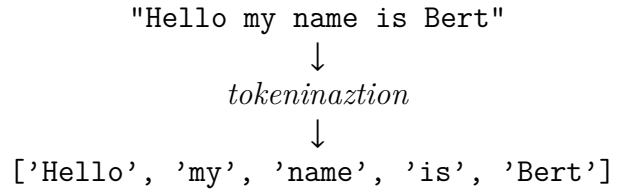


Figure 2.1: Example of how a tokenizer might work.

- LinearSVC - An algorithm used for binary classification problems. It is designed to classify products into one of two possible classes
- LabelEncoder - A utility class that encodes texts to numerical values, this is needed since algorithms often require input data to be in numerical form.
- SGDClassifier - is a classification algorithm. It is used for multi-class classification and binary tasks and is known for its efficiency and ability to handle large datasets.

2.2 Natural Language Processing

Natural Language Processing (NLP) can be described as computers trying to understand and analyze human language, such as English, Swedish, German etc. A task might be used to summarize, translate (between and among languages) or categorize an input such as speech, text, or any other format of language that we use in our everyday life (Allen, 2003).

Although there are several core areas in NLP, the focus and most important one for this report is Language Modeling (LM). The idea behind LM is that single words contain limited information on their own. It is in a sentence or in a context the true value of a word takes shape. To comprehend the essence of words, it is integral to determine the interaction between words. The meaning of words is influenced by the combination and relation to other words in a sentence or in a larger text (Otter et al., 2021a).

2.2.1 Tokenization

A crucial part of a NLP task is to tokenize a text. Tokenization is a method that splits a text into semantically useful parts called tokens. A tokenizer can be used on different part of the text such as sentences or words, in the context of this project word tokenization is used. This is a essential part since the data needs to be in a form that is suitable for a ML model to understand (Trim, 2013; Jurafsky and Martin, 2009; MonkeyLearn, 2023).

An example of how a tokenizer could work is shown in Fig. 2.1. The usual next step after tokenization is to convert the tokens into something a computer can read. One such method is to have each token correspond to an id, e.g. an integer.

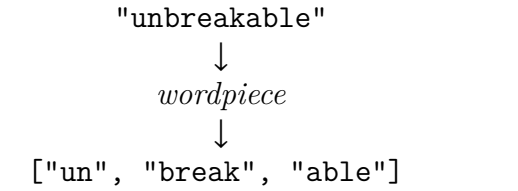


Figure 2.2: Example of how a wordpiece algorithm might work.

WordPiece

Some NLP models operate using a fixed vocabulary, meaning that they can only tokenize words present in that vocabulary. This becomes problematic when a word not in the vocabulary needs to be tokenized. Several methods exist to handle these so-called out-of-vocabulary (OOV) words, one of which is WordPiece (Devlin et al., 2019b).

WordPiece is a method to break down words into smaller pieces, known as *sub-words*. This approach is particularly useful for handling uncommon or unfamiliar words that are OOV. Instead of the model getting confused by an unrecognized word, the word is broken down into smaller, more manageable pieces that the model is likely to know. An example of how this algorithm works is presented in Fig. 2.2.

2.2.2 Word Embedding

Word embeddings are a central component in many neural networks and LM. The concept revolves around being able to convert text, which we humans understand, to a vector, that a computer understands. One such algorithm is Word2Vec which can be used to create word embeddings (Mikolov et al., 2013). Uses for word embeddings varies, for example, it could be used to find semantic similar words, cluster related words or displaying the correlation between word/text (Goldberg, 2017).

2.2.3 Self-Attention

Attention is sometimes described as a method that tries to mimic human approach of paying (cognitive) attention to important path of e.g. sentences (Otter et al., 2021b). If the following document is given as context *The green frog jumped over the rock.* and a question like *What did the frog jump over?* or *What color is the frog?* we instinctively know which parts of the text that contains the answer and which part do not.

There are many attention methods, like Luong et al. (2015) and Bahdanau et al. (2015), but the one relevant in this thesis is self-attention since they are used in Transformers as stated in Vaswani et al. (2017). The self-attention mechanism allows parts of a sequence, e.g. tokens or words in a sentence or text, to determine its connection and weight with other parts within that sequence. It means that this mechanism helps to captures dependencies and relationships within a sequence by assigning an attention score to individual parts based on the importance of it in the

Table 2.2: Example of calculating the TF-IDF score for the term *'Machine'* for 4 documents.

Document	TF	IDF	TF-IDF of term *'machine'*
1	$1/4 = 0.25$	$\log(4/3) \approx 0.1249$	$0.25 \cdot 0.1249 \approx \mathbf{0.0312}$
2	$0/7 = 0$		$0 \cdot 0.1249 = \mathbf{0}$
3	$1/6 \approx 0.1667$		$0.1667 \cdot 0.1249 \approx \mathbf{0.0208}$
4	$0/4 = 0$		$0 \cdot 0.1249 = \mathbf{0}$

sequence (Galassi et al., 2021).

2.2.4 Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) is a combination of term frequency and inverse document frequency. The first one (TF) measures how frequently a term is present in a document relative to the total amount of terms. This means that a term that has many occurrences gets a higher score compared to other terms. The latter (IDF) measures the importance/meaningfulness of a term and is calculated by dividing the total amount of documents by the number of documents that contain this term. This introduces an attenuating effect for terms that are present in multiple documents, the more documents the term is present in the higher the dampening effect becomes. When combined, the weights for each term are created for each document, thus creating the TF-IDF vector (Manning et al., 2008). To calculate the TF-IDF for a term t in a document d it is done by

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \cdot \text{IDF}(t)$$

where IDF is found by

$$\text{IDF}(t) = \log\left(\frac{n}{\text{DF}(t) + 1}\right),$$

assuming $\text{TF}(t, d) \neq 0$, n = number of documents and DF (document frequency) is the number of documents that contain t (Scikit-learn, 2023).

Consider the four sentences (1) *Machine learning is fascinating.*, (2) *Natural language processing is a subfield of AI.*, (3) *AI and machine learning are closely related.* and (4) *AI can help research.*. To find the TF-IDF scores for "machine" in these documents is done in three steps, first find the TF for the word in each documents then caluculate the IDF and lastly calculate the TF-IDF. For this example the TF is simply found by counting the occurrences of "machine" divided by the total number of words in each document and the IDF using the formula above. Calculations are laid-out in Tab. 2.2. This logic is applied on the specific term *'Machine'* in the documents. In reality TF-IDF is a vector that represents all term frequencies of a document.

2.3 Transformer

The transformer is a machine learning architecture developed by Vaswani et al. (2017) and was created in order to process sequential data, such as text. Instead of using long recurrent neural networks (RNN), the transformer uses a self-attention mechanism which allows it to selectively focus on certain parts of an input. This is made possible since the transformer processes the entire input instead of just a smaller part of it like an RNN, and such long-range relationships can be found (Vaswani et al., 2017).

The Transformer model consists of two main components: the encoder and the decoder, as illustrated in Fig. 2.3 showing the entire model architecture. The encoder processes the input data, encoding it in a way that goes beyond simply understanding the meanings of individual words; it captures the essence of the entire input. Each word, represented as a vector, embodies not only its own meaning but also its relationship to all other words in the input, achieved through the self-attention mechanism.

The decoder then generates the output by interpreting the encoded input. This decoding process also employs the self-attention mechanism, producing the output sequentially, word by word. The decoder uses the self-attention mechanism in two distinct ways: one to focus on relevant parts of the input data and another to concentrate on the relevant portions of the output generated thus far.

Both the encoder and decoder are composed of several identical layers, enabling the Transformer to capture complex relationships within the data.

The Transformer model can be applied to tasks such as translation. For example, it could translate a sentence from Swedish to English, processing the input sentence in Swedish and generating the corresponding sentence in English as the output.

2.3.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a modern NLP model based on transformer architecture, however it only uses the encoder part of the transformer model. Unlike other models, BERT is trained on a bidirectional context, meaning the model is trained to predict both the preceding and the following words in a sentence. The bidirectional training enables BERT more nuanced and deeper understanding of language in general (Devlin et al., 2019a).

To train a BERT model, vast quantities of text data, including articles, books, and web pages, are required. To mitigate the problem of OOV words that may arise from such diverse sources, BERT employs the WordPiece tokenization algorithm. The training phase, also called pre-training, consists of predicting words that have been masked out (hidden), a job which demands the model to understand both the meaning of words as well as the context of the text. Once the model training is complete, can a BERT model be fine-tuned for a NLP specific task, such as text classification or question answering. This fine-tuning step adjusts the pre-trained model according to the characteristics of the data set from the specific task. The data set required for fine-tuning is much smaller than the data for pre-training phase. The ability to fine-tune a BERT model allows the model to be versatile and having

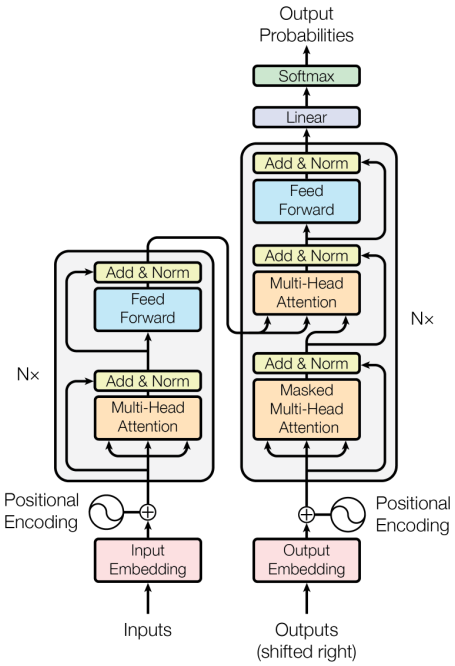


Figure 2.3: Transformer model architecture, provided by (Vaswani et al., 2017). The left block is the encoder and the right the decoder.

the ability to be adapted to several different problems, as mentioned previously (Devlin et al., 2019a).

Layers

As mention previously is BERT model based on the transformer architecture, which is known for its effectiveness in handling sequential data. A BERT model stacks multiple of these transformers on top of each other, where more layers allows for more complex relations to be captured. Between each layer flows information from the input text, where each step refines the word representations. Different variants of BERT contain different number of layers, where more layers implies a larger model and thus more computational complex and need more time training (Devlin et al., 2019a).

In Devlin et al. (2019a) two models are suggested, BERT-base with 12 layers and BERT-large with 24 layers.

2.3.2 Pre-trained and fine tuning models

One of the many advantages of a model like BERT is that once trained on large quantities of data, it can be reused and expanded upon, this is what one would call a pre-trained model. Once created it can be used to solve domain-specific problems by further training on data that is specific for that problem, called fine-tuning. This enables the NLP model to learn complex patterns and structures in the data that

were not present in the original data set. The amount of data needed in order to fine-tune a model is often much lower than the original data set, which implies that the fine-tuning session takes fewer resources (Devlin et al., 2019a). There are many different types of pre-trained BERT models, The models presented in *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* have either 12 or 24 layers. BERT-base has 12 and BERT-large has 24 layers (Devlin et al., 2019a).

Distilling BERT

The resulting BERT model created after training can be quite large, in terms of parameters, so when fine-tuning a model it can take quite some time to get it to perform well. But if one is willing to trade some accuracy with speed then a distilled version of the model can be used. It works by using Knowledge distillation, described by Hinton et al. (2015), transferring the knowledge from one bigger model called the teacher model to a smaller one called the student model where the teacher model in this case would be BERT. It functions by feeding the class probabilities from the teacher model as *soft targets* to the student model. This way the student model learns to mimic the same behavior as the teacher model. After the initial training is done, the student model is fine-tuned with the true labels instead of the soft targets (Hinton et al., 2015). DistilBERT has 6 layers compared to the normal BERT which has 12 or 24 layers.

2.3.3 Zero-shot classification

Zero-shot classification and training is a sub-field of machine learning where the aim of a model is to be able to classify classes even though they have not been present/seen during training. When such model is trained, it is only allowed to see data not containing any labels that have been put in an unseen set. For a model that have been trained to classify animals, e.g. elephants, lions, horses, etc., if a new animal, like a zebra, then the model could still recognize that it is similar to a horse. Zero-shot learning is a technique used to train a model to recognize items that are conceptually related to its existing knowledge (Xian et al., 2018).

2.4 Category Path

To represent the categories to which each product in the data set belongs, this thesis introduces a new term: *category path*. A category path is a collection of categories that a product belongs to, organized hierarchically from the most general to the most specific. Table Tab. 2.3 presents a few examples of category paths. In these examples, the most generic category is `"hem"`, and subsequent categories become progressively more specific.

Table 2.3: Examples category paths.

Category paths
`["hem", "kläder", "jackor", "regnjackor", ...]`
`["hem", "inredning", "möbler", "soffor", ...]`

2.5 Model proposed by Liu et al. (2021)

This thesis has taken great inspiration from the paper Liu et al. (2021). In this paper, the authors are investigating the possibility of how to improve pre-trained models to correctly classify a product towards a taxonomy. The model in the paper is divided into two major parts: fine-tuning and reinforcement learning. First, the model is fine-tuned on data specific to text classification, using the methods mentioned in Sect. 2.3.2. This fine-tuning step takes a text description as input, and the output consists of the category paths (as described in Sect. 2.4). The data used during the fine-tuning phase includes both true categories and categories to which the text does not belong, referred to as *false data*. False data is randomly sampled from a category path, meaning that the path may contain some correct categories but eventually leads to false categories. The random sampling can be achieved by selecting children from a previous category in the path. By combining both true and false data during fine-tuning, the model learns to associate text with a category path and is trained using a sequence classification architecture.

Once the pre-training step is complete, the next phase begins. The model produced during fine-tuning undergoes a second training cycle using reinforcement learning as the training paradigm. During this second training phase, the model is expected to learn the necessary steps to reach the final correct category path. To accomplish this training step, a policy method is introduced. The policy method is used to determine the next action the model should take in a category path. This method takes into account the current state, category path, and action space (possible categories), and it returns a score for each available action. The possible actions that the model can take are determined by the category tree, as each category may have zero or more children in the tree. The available actions are the children of the last category in the path. The model is then updated, and new paths are selected for investigation based on the prediction scores from the policy

The policy selects the next category by choosing the most probable child of the last category in the predicted path. The predicted category is then added to the previously predicted categories, thereby updating the state. This process results in the accumulation of a path over time, starting from the root.

At each stage of this accumulated path, the model receives a positive reward, and for every incorrect prediction, it incurs a penalty. If the correct path consists of categories denoted as c_n, where n is an unique ID for each category, the entire correct path is written as (c_1, c_2, c_3). If this complete path is the desired final output, it means that the model will receive a positive reward during the second training phase if it predicts (c_1), (c_1, c_2), or the entire path, as the elements in the predicted path represent the beginning of the correct path. However, if the model selects (c_5), (c_2, c_1), or (c_1, c_2, c_4), it incurs a penalty because none of these paths are part of the

correct path.

This is a brief introduction to the workings of the model presented in the paper, we urge the interested reader to refer to the paper to get a deeper understanding of the theory and methodology used in this approach.

2.6 Scrapy

Scrapy is one of many open-source application frameworks that enable web crawling, a method of going from site to site and gather structured data. At the core of Scrapy there exists something called spiders. They specify what URLs to start crawling from and how it should follow links as well as specifying how to parse the retrieved website data, which often comes in the form of HTML. The spiders work by sending out an request and receives a response in the form of raw data that are then parsed according to a predetermined method. Sitemap spiders are used to crawl the product data. They work by following the sitemap file for a certain site (Scrapy, 2023). The sitemap file contains a list of links that lead to the different parts of the website. The sitemaps are located within the `robots.txt` file which stands for Robust Exclusion Protocol. Both of them are standardized practices used by websites to indicate to web crawlers and other web robots what parts of a website the crawler is allowed to crawl (Sitemaps.org, 2020; Central, 2023; Koster, 2007).

2.7 Evaluation metrics

In order to measure the performance of the models in this thesis, some measurement metrics need to be introduced. When a model makes a prediction in a classification task it will either produce a correct or an incorrect guess. As a performance metric two measures can be constructed, *recall* and *precision*. Recall is the proportion of times a label were correctly predicted out of number of occurrences of that label, and precision is the proportion of correct predicted labels out of total predictions for that label (c) Chinchor and Sundheim (1993); van Rijsbergen (1979). In detail they are defined as:

$$\text{recall}(c) = \frac{|\text{correct prediction}(c)|}{|\text{occurrences}(c)|} \tag{2.1}$$

$$\text{precision}(c) = \frac{|\text{correct prediction}(c)|}{|\text{predictions}(c)|} \tag{2.2}$$

where $|\cdot|$ is the count operator.

Another measurement is also introduced that combines recall and precision called F_β, and it is defined as:

$$F_\beta = \frac{\left(\beta^2 + 1\right) \cdot \text{precision}(c) \cdot \text{recall}(c)}{\left(\beta^2 \cdot \text{precision}(c)\right) + \text{recall}(c)}, \tag{2.3}$$

where β is the relative importance between recall and precision. In this thesis the value of β will be 1 since recall and precision are weighted the same (Chinchor and

Sundheim, 1993). The F_1-score is thus:

$$F_1 = 2 \cdot \frac{\text{precision}(c) \cdot \text{recall}(c)}{\text{precision}(c) + \text{recall}(c)}. \tag{2.4}$$

Lastly, the measure *accuracy* will be used which will be implemented using Scikit-learn (2023) as:

$$\text{accuracy} = \frac{|\text{correct predictions}|}{|\text{predictions}|} \tag{2.5}$$

note that accuracy and precision is similar but precision per label and accuracy is for all predictions.

Chapter 3
Method

This chapter aims to motivate the design decisions that have been made during this project. The chapter aims to give the reader a good understanding of the methodology that was set up and used. The chapter will be divided into different sections explaining the different parts of the project.

In Fig. 3.1 the broad method pipeline is shown, *Data Preparation* relates to data crawling and Sect. 3.1. Then, *Model Design* is both the design of baseline in Sect. 3.2 and model design in Sect. 3.3. The training process of the models also includes hyperparameter optimization. After the training phase, the models are evaluated and training progression is also presented.

Figure 3.1: The method outline.

3.1 Data Set

One of the goals of this thesis is to implement a method that takes product information and outputs the hierarchical category it belongs to. Thus, a data set was created prior to this project at Theca Systems by Victor Winkelmann with product data retrieved from some of the largest e-commerce in Sweden (Gunnilstam, 2021). In order to create the best setting for a model to generalize on different types of products, a broad and diverse data set is needed since the idea is that the more different products a model sees during training the better. Research to find existing

data sets that met this criterion failed to yield adequate results, thus a data set was created.

Creation of the data set was done by crawling some of the sites mentioned above, however, there were both practical issues and some sites made them unavailable for scraping. As mentioned above the goal is to create a method of automated categorization from title and description, these two are together with the category vital information to gather from each site and product. However, some metadata were also collected, those were Manufacturer Part Number (MPN), Global Trade Item Number (GTIN) and European Article Number (EAN). The plan was to use these in order to find products from different sites that had the same product but different categories so that the model could learn that a product description could belong to different category hierarchies. Another strategy to find common products between sites where to scrape price comparison sites (PCS) which contained sources of common products.

More details on the contents of the data set can be seen in Appendix A and how the data sets where created can be found in Sect. 3.5.2.

3.1.1 Taxonomy

A taxonomy is the collection of relationships between the categories available in the training data, a systematic way to organize the categories. In this taxonomy the parent and child relation is stored, created by examining all the category paths in the training data. For this project, the main use of the taxonomy is to create training examples when fine-tuning a model.

In this project, we create a taxonomy by examining the category path of each product. Since paths are hierarchically organized, starting from the most general category and progressing to the most specific. This hierarchical arrangement facilitates the construction of a tree-like structure for the taxonomy. In this structure, the first category in the path serves as a parent to the second, the second category is a parent to the third, and so forth. For visual representations of how this tree is constructed and its overall structure, refer to Tab. 3.1 and Fig. 3.2.

During this project, two different types of taxonomies were created: one incorporating category paths from all stores, and another containing category paths from only a single store. While developing the taxonomy that included all stores, we encountered an issue with cycles forming in the taxonomy tree, as visually represented in Fig. 3.3. These cycles posed a challenge for the model's ability to predict the next path, as it could potentially lead to endless predictions.

Various strategies were attempted to eliminate these cycles, including removing edges where cycles occurred and renaming categories. However, these methods proved to be ineffective, as they often disrupted the correct paths for some products. As a solution, we implemented a maximum depth limit. This means that any category path generated by the model cannot exceed this set limit. This limit were set to 13 for all models and were chosen based on the longest category path.

Table 3.1: Example category paths.

Category paths
`["hem", "kläder", "jackor", "regnjackor", ...]`
`["hem", "kläder", "jackor", "dunjackor", ...]`
`["hem", "kläder", "byxor", "jeans", ...]`
`["hem", "kläder", "byxor", "chinos", ...]`

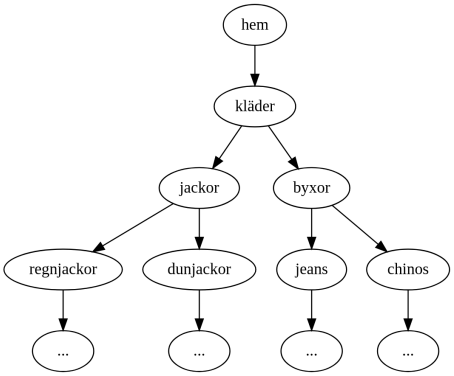


Figure 3.2: Taxonomy created from the example category paths in Tab. 2.3.

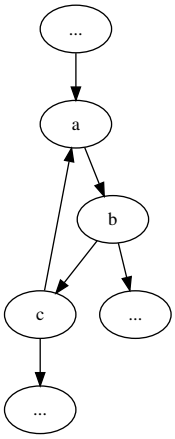


Figure 3.3: A cycle present in the taxonomy, *a*, *b* and *c* are three different categories.

3.1.2 Common Product Handling

When training a model with this data, it is of interest to find potential overlap of products between different stores, e.g. if two store sells the same video game console. Since the text of these products is assumed to be semantically similar to each other the model should be trained, so that for a single product, several possible category paths might be correct. By enabling the model to be trained on all different possible paths for the same text, it can learn complex relationships between categories that are otherwise lost.

To enable this possibility, the products which had multiple entries were grouped together using one of the PCSs available in the data set and the metadata tags MPN, GTIN, and EAN. In other words, there have been four distinct, but not necessarily, independent groups created. Groups that share one or more members are joined together since one can assume they are the same if they share some common metadata element. When the grouping of products is done, new training examples are created for each product in a group adding the pair product text and category paths from each other product.

In Table Tab. 3.2, examples of products are displayed, each with associated metadata values. Product A has associated MPN and EAN values, while Product B has MPN and GTIN values, and so on. It is important to note that some values may be missing. Products A and B are grouped together because they share the same MPN value. Similarly, Products B and C form a group as they have the same GTIN number. Product D, lacking a common metadata tag with the other products, is not grouped with any other product. Since Products A and B form one group and Products B and C form another, and Product B is common to both, these groups are combined. This results in two distinct groups: the first comprising Products A, B, and C, and the second consisting solely of Product D.

The idea of creating these new training examples is to further train the model with more correct paths to reach a product. The hypothesis is that this enables the model to find better correlations between texts and potential categories to enable prediction on unseen category taxonomies with higher accuracy.

Table 3.2: Example of products and meta data associated with them.

Product	MPN	GTIN	EAN
Product A	mpn_1		ean_1
Product B	mpn_1	gtn_1	
Product C		gtn_1	
Product D	mpn_2	gtn_2	ean_2

3.1.3 Creating Train, Validation and Test Sets

As mentioned in Sect. 2.1.4, is it a often a good practise to split the data into three parts: train, validation, and test. Let's start with the training set. To create the

most diverse and comprehensive data set, all available stores are used, as detailed in Appendix A. In Sect. 3.5.2, the contribution from each store to the training data.

The validation and test sets are created similarly but are separated by stores. After creating each part, the data set is shuffled to remove any accidental dependencies between two products that could interfere with the model.

During the data set creation process, two steps were taken to clean the data as thoroughly as possible. First was to filter out any data that was not in Swedish. Secondly, data that was missing a category path was also filtered out since it would be of no use for this task.

It is important to mention that there is no overlap between the train, validation, test data sets, meaning that no product is present in more than one part.

3.2 Baseline Models

A baseline method is supposed to be a naive method that will act as a progress check. The goal of a newly developed model should always be to perform better than this baseline. The approach for this work is to use well-established methods to create such baselines, these methods are TF-IDF, a pre-trained, but not fine-tuned, BERT and zero-shot classification.

3.2.1 TF-IDF

The reasoning for using two variations of TF-IDF as one of the baseline approaches is that it is an easy-to-implement method that yields fast results without the need to do complex machine learning computations and training. In the sections below the two variations will first be presented and then further discussed. The methods were implemented with TF-IDF and its components as introduced in Sect. 2.2.4 and 2.1.7.

These baselines using TF-IDF were made:

- *TF-IDF: multi-label classifier* which baseline uses Scikit-learn's OneVsRestClassifier with LinearSVC.
- *TF-IDF: top-k* which uses Scikit-learn's SGDClassifier and top-k for prediction.

The difference between these models lies in the classification training and evaluation. In the sections below, they will be further presented.

TF-IDF: multi-label classifier

In this approach the data was divided in the following manner:

- Document 1: Consisted of the title and text for each product
- Document 2: Consisted of all possible categories that were present in the data set

This was the easiest baseline to implement since it required little to no data preparation work. The only things that are required are to concatenate the title and description and create a big set containing all possible categories. The decision of Document 2 meant that the hierarchical structure was lost. The first document was first fitted and transformed into a vector using the TF-IDF vectorizer. The second document, containing the categories was thereafter, fitted and transformed using the MultiLabelbinarizer. For training the *OneVsRestClassifier* was used with LinearSVC. It was evaluated by running the test set through the trained model and then comparing if the prediction was the same as the correct answer. In this approach, only the top prediction was checked since that is the output from the model. The model used the following parameters `max_features=10000`, `min_df=100`, and `max_df=0.5`.

TF-IDF: Top-k

In this approach the goal was to create a baseline that dynamically generates the next category based on the current state. In order for this to be possible a new program was written to prepare the data for training. Assuming a product has category length 3, in the Tab. 3.3 the creation is shown, i.e. we get 4 inputs that get added to the input list. Two tokens were therefore needed to be introduced: beginning of sentence `<BOS>` and end of sentence `<EOS>`. The approach enables the baseline to learn when it needs to stop as well as enables dynamic path prediction for top-k predictions. The text was first fitted and transformed into a vector using the TF-IDF vectorizer. The labels in the input data were fitted and transformed using the `MultiLabelbinarizer` and the output was generated by using `LabelEncoder`.

Table 3.3: Input creation for TF-IDF-top-k

Input	Output
text + <BOS>	c_1
text + <BOS>, c_1	c_2
text + <BOS>, c_1, c_2	c_3
text + <BOS>, c_1, c_2, c_3	<EOS>

The algorithm chosen for this task was Stochastic Gradient Decent with the parameters, `max_features=10000`, `min_df=100`, and `max_df=0.5`.

During the evaluation, the top-k path will be saved and predicted until there exist 10 paths that have the `<EOS>` token.

3.2.2 Zero-shot Classification

One approach researched for automatic product categorization was the use of zero-shot classification. Zero-shot classification, introduced in Sect. 2.3.3, has the advantage of classifying categories, even if some were not present during training. However, this approach did not provide a solution to the presented problem. Therefore, it became interesting to evaluate how well the developed model could perform compared to a model designed to predict product categories even without training.

Table 3.4: input creation for the zero-shot model

Approach	**Prediction for depth 2**
Non-concatenated	[kök-och-tvättstuga, Vitvaror, Badrum]
Concatenated	[Kök-och-bad/kök-och-tvättstuga, Kök-och-bad/Vitvaror, Kök-och-bad/Badrum]
Approach	**Prediction for depth 3**
Non-concatenated	[Avfallshantering, Köksflakt, Badrumstillbehör]
Concatenated	[Kök-och-bad/kök-och-tvättstuga/Avfallshantering, Kök-och-bad/kök-och-tvättstuga/Köksflakt, Kök-och-bad/kök-och-tvättstuga/Badrumstillbehör]

Zero-shot classification was added as a baseline using two different approaches. Both baselines only considered the top prediction, i.e., top-1, in every iteration. These approaches were also implemented in a way that takes the hierarchical structure into account. The following sections will provide further introductions to the two baselines.

Non concatenated

The first approach was to let the model in a step-wise approach predict labels of a product text, the labels were in the form of a category starting at *root*. When the baseline ran it would explore all possible children of category x at depth I. The baseline would then choose the child y that was most probable to belong to the text. It would then continue by exploring all possible children to y and continue until the selected child had no more children. All selections made by the program would be added to the result and then be compared to the correct answer. A problem arose during testing since all context was lost due to the program only comparing possible categories at a depth I with the text and not taking the accumulated path into account. In Tab. 3.4 this is shown in the Non-concatenated rows.

Concatenated

This baseline was made to fix the context issue discussed in the previous section. In this approach, the program would take the context into account by concatenating the accumulated path. At every iteration, the accumulated path would be concatenated with all its plausible children and then be predicted and the top prediction would be chosen. This entailed that the context of a label would never be lost. In Tab. 3.4 this can be seen when looking at the Concatenated row.

3.2.3 Pre-trained BERT with Sequence Classification

When the inference program for the developed model was written it became interesting to see how well a fine-tuned model would compare to a pre-trained model. This was an easy task to explore since the same inference program could be used

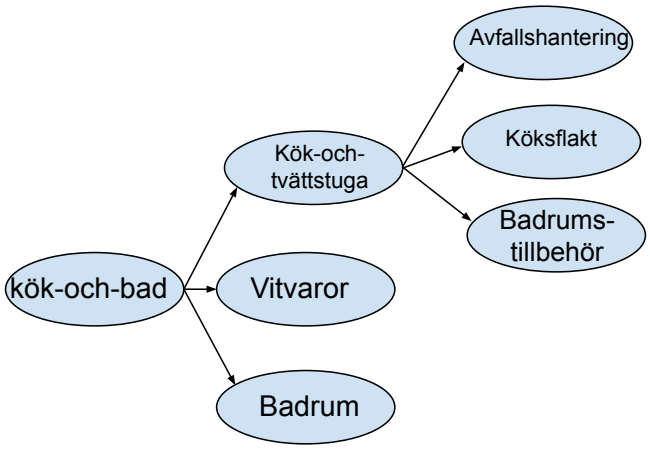


Figure 3.4: Figure to clarify how zero-shot baseline was used

with the same pre-trained model as the developed model was initiated with. The inference program is explained in 3.4.

3.3 Transformer Models

For automatic product categorization using transformer models, the thesis set out to further analyze, develop and adapt the model presented in (Liu et al., 2021). The majority of the allocated research and development time was put into adapting this model to suit this paper's needs. Since this model was not specifically created to solve this thesis' problem, some changes were needed to make it work. Unfortunately, all attempts did not yield a working model and will be further discussed in chapter 5. This also led to the new model being created, which is introduced in the next section.

3.3.1 Sequence Classification Model

Since the model proposed by Liu et al. (2021) did not work, a new model was needed. This new model takes great inspiration from the presented model but uses a new approach with simpler complexity. The model that was decided upon was to use sequence classification combined with hierarchical structures to enable product classification. The new approach utilizes dynamic classification and exploration using top-10 most predictable paths. In the next few sections the components to this model will be introduced. The way this was implemented was by introducing `<CONT>` and `<STOP>` symbols for the model, as introduced in 3.3.1, where the main reason for doing this is to teach the model step-wise behaviors to reach the goal.

Path Sampling

When fine-tuning an ML model the goal is to enable the model to correctly classify products to the according category. For this to be possible the train data needs to consist of both true and false examples. Recall, that true examples are the correct answer that has been gathered during crawling, and negative examples are categories to which a product does not belong, similar to Sect. 2.5. The data set, introduced in Sect. 2.1.4, only contains true values, thus requiring the model to create a negative example for the training phase. This was implemented by letting the program create new examples containing pairs of product text and new false categories for every product. These examples had the same product text but this time with a new false category path and labeled as false. The false examples are also referred to as negative examples.

Furthermore, since the model uses dynamic classification it also needs to learn whether or not it has reached the "final" destination. To enable the model to learn this behavior, two new tokens are introduced, (`<CONT>` and `<STOP>`). These tokens are used when sampling paths to enable the model to learn, this can be seen in 3.6.

Two more variables were introduced as a result of the negative path sampling, *Negative Path Depth* (NPD), how deep the negative examples go, and *Number of Negative Examples* (NNE), the number of false paths created for each level. These variables together decide how many negative examples are created for every positive entry during the training phase. By introducing NNE and NPD unlocks the possibility to control of the ratio between positive and negative examples by tweaking these parameters. The domain of the variables are NNE, NPD $\in \mathbb{N} \cup \{0\}$ and can be independently chosen. However note, that if any of them were to be chosen to be 0 no negative examples will be generated.

The main idea of the method of creating false examples is that for each category in a category path, it creates NNE number of additional negative paths. The length of these additional paths is controlled by NPD, see the example in Tab. 3.5. In addition to this, the model also has to choose between the continue or stop token. Since all of these created paths are false the program randomly decide what token there should be. Given a true category path $(c_1, c_2, ..., c_n)$ of length n, the resulting training data can be seen in Tab. 3.5, the path used for training is the true path concatenated with the false one.

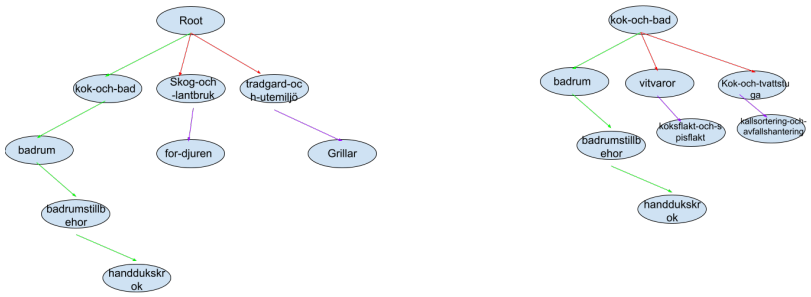
In Fig. 3.5 an example product is presented with a green line, a red line, and a purple line. In this example, both NNE and NPD are equal to 2. The red line represents how many negative examples are created for every level. The purple line represents the depth of the negative example. For every creation, the end token is chosen randomly. This false path creation happens at every depth and in the right figure, one can see the same idea being applied. The false path gets chosen randomly every time which ensures that the whole taxonomy tree is covered when running multiple epochs or having many products belonging to the same category.

Finding Optimal NNE and NPD

When training a transformer-based model both the true and false examples are important to create a well-functioning model. By altering the NNE and NPD parame-

Table 3.5: Example of false path creation for a single product using NNE=1 and NPD=2, c_n is the true category and $fp_{n,i}$ is a false path at level n. Note, for each level the same false paths is used and that the End token are randomly chosen.

Level	True path	False path	End token
0		$fp_{0,1}$	<CONT>
0		$fp_{0,1}, fp_{0,2}$	<STOP>
1	c_1	$fp_{1,1}$	<CONT>
1	c_1	$fp_{1,1}, fp_{1,2}$	<CONT>
2	c_1, c_2	$fp_{2,1}$	<STOP>
2	c_1, c_2	$fp_{2,1}, fp_{2,2}$	<CONT>
		...	
n	$c_1, c_2, ..., c_n$	$fp_{n,1}$	<STOP>
n	$c_1, c_2, ..., c_n$	$fp_{n,1}, fp_{n,2}$	<CONT>



(a) depth at root

(b) depth 1

Figure 3.5: example product

ters the ratio between negative and positive examples changes, which can impact the performance of the model. With this in mind, a method of finding the optimal pair of NNE and NPD was created. However, due to time constraints, the experiment is perhaps smaller than it should be for a more thorough result. The experiment works by first deciding the ranges for the two variables, recall that the domain of the variables are NNE, NPD $\in \mathbb{N} \cup \{0\}$.
Next, a training set is created by taking a sub-set of the data set, this is done so that all tests are done using the same data. A training with all the possible combinations of NNE and NPD is done and once completed inference is done to measure the performance of the different models.

Additional Training Data

Similar to the previous model, the idea of training is based on rewarding the model when it is on the right track as well as displaying what paths are false and how not to create new paths. By always having these continue and stop tokens, the model can learn how to decide whether or not it wants to continue guessing. In Tab. 3.6

one can see how the true paths are created for the trainer, note that the last entry should have a stop token not a continuing one.

Table 3.6: How positive examples are structured during training. There is no value for a false path or end token when its value does not affect if it is negative or not.

True path	End token
c_1	<CONT>
c_1, c_2	<CONT>
c_1, c_2, c_3	<CONT>
...	
$c_1, c_2, ..., c_n$	<CONT>
$c_1, c_2, ..., c_n$	<STOP>

Lastly, the model also needs to see when the correct path is stopped too early, i.e. any continue token replaced with a stop one in Tab. 3.6, and when it predicted a true path too far. The latter of them can happen in two cases, either an end token in the last row in Tab. 3.6 is replaced with a continue one, or the last row is appended with further categories. The probability of generating too-short paths is 5% and creating paths beyond the correct answer generates a probability of 10 %. In Tab. 3.7, an example of how this works is presented.

The positive paths in Tab. 3.6 are combined with the negative ones in Tab. 3.5 and Tab. 3.7 to create the complete set of training examples.

Buffering

To maximize training efficiency the model needs mixed batches where there are both positive and negative examples, but also mixed products. This would not happen if one would feed the output from the path sampler into the model. To solve this problem two buffers were introduced, one for positive and one for negative examples. The path sampler fills these buffers with the training examples until they have a certain number of entries each. They are then shuffled to maximize the spread of product data and then fed to the trainer. When the buffers go below a certain threshold they get refilled with more examples until they both have at least a certain number of entries again. The buffers sizes can vary depending on the product that

Table 3.7: How early and late stopping are generated.

True path	False path	End token
c_1	-	<STOP>
c_1, c_2	-	<STOP>
	...	
$c_1, c_2, ..., c_{n-1}$	-	<STOP>
$c_1, c_2, ..., c_n$	fp_1	-
$c_1, c_2, ..., c_n$	fp_1, fp_2	-
	...	

gets sampled, which the model keeps track of and creates the training batch with this ratio in mind. In figure 3.6 this logic is shown.

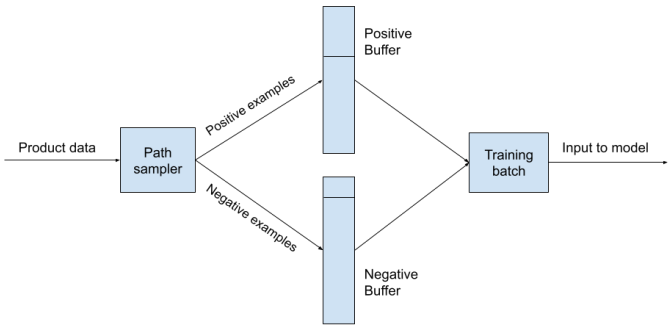


Figure 3.6: Buffer for model trainer

Training of the model

With everything in place, the model can now be trained, and the buffer will continuously feed the trainer with batches containing training examples that follow the predefined ratio of positive and negative examples. The batch sizes can also vary depending on the training hardware. In figure 3.7 the training process is visualized. When the batch has gone through the model the prediction is compared to the actual true values. This information is used in the optimizer and then parameters and weights are updated to minimize the training loss.

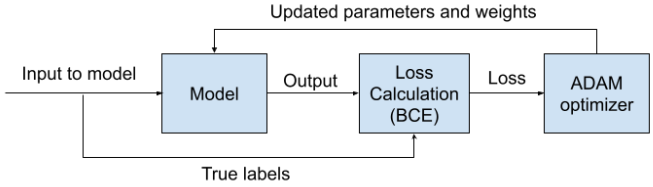


Figure 3.7: Visualization of the trainer for the model

3.4 Evaluation

This section will explain how the inference and evaluation works for the models.

3.4.1 Sequence Classification Model

To verify how well the model performs an inference program was written. This program works by running through a test set. The program starts by extracting all possible categories from the root node, and for every category, it adds a `<CONT>` symbol. It then proceeds to predict which category it is most likely to belong to. It works by extracting the positive class from the output logits and applying a sigmoid

function to the values. After the output logits have been transformed it sorts the input list by score and places both the prediction with the corresponding score in a list.

The program continues to predict until one of two things happens, all top 10 predictions have a <STOP> symbol or two top 10 lists are identical between two iterations in a row. At every iteration, a prediction list is created and the top 10 paths are investigated. One of three things happens for each path.

1. a path has a <CONT> symbol and there exists a child/children to that node: the path gets removed and the following is added to the prediction list. The removed path concatenated with each child individually. It also adds itself with a <STOP> symbol, to enable the model to decide if it wants to stop

2. a path has a <CONT> symbol and there exist no child/children to that node. The node gets placed back into the list with a <STOP> symbol

3. a path already has a <STOP> symbol, nothing happens

The possibilities presented above can be seen in Fig. 3.8 where the green line represents the first possibility, purple the second, and red the third possibility. The

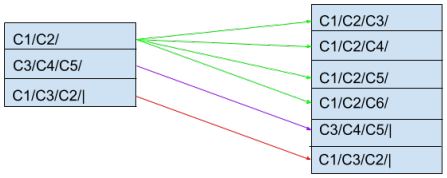


Figure 3.8: Inference program

score is saved every step of the way and updated when a path is further explored. It gets updated by multiplying the old score with the new predicted score.

When the program finishes it saves the top 10 predictions and the correct answer in a result list. This way, one can track how precise the model is if only looking at the top prediction but also the accuracy when looking further down the list. This is illustrated in Fig. 3.9

3.4.2 Evaluation Methods

To measure the performance of the model some evaluation methods are needed, the ones relevant for this thesis have been introduced in Sect. 2.1.5 and Sect. 2.7. To evaluate the models after training, the accuracy metric is used. For accuracy to be a valid measure, there must be clear right and wrong answers, which is the case in our task of predicting product categories. Another method is constructed from Sect. 2.1.6 using the method described below.

Top-k Rank

In some models, top-k learning has been applied, where the top k guesses from each state always is stored. From this learning method, a new evaluation method

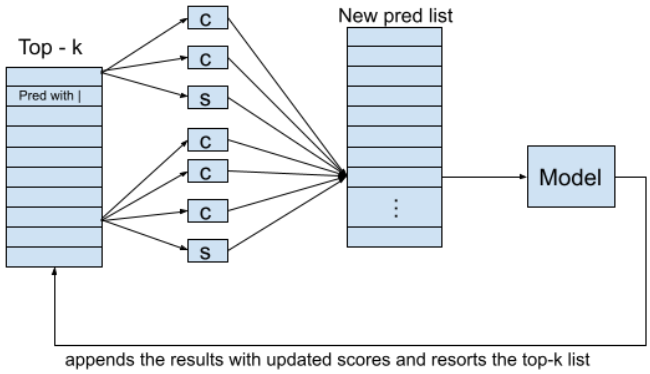


Figure 3.9: Top-10 workflow Inference program. The *c* in the boxes stands for the children to a specific parent node with an added <CONT> symbol. The *s* represents that no children are added and only a <STOP> symbol.

is introduced: top-k rank or simply just rank. The rank is a measure of at which position the correct prediction is located at. Let us say that the correct answer is found at rank x, where $1 \leq x \leq k$. If the correct answer is first in the list the rank is 1, the second rank is 2, and if the correct answer is not found then the rank is set to 10.

3.5 Experimental Setup

In this section, the experimental setup will be presented, and the choices of parameters and hardware available for the thesis will be listed in the following sections.

3.5.1 Hardware

The hardware used in this project was either a Nitro 5 AN515-46-R0EQ laptop[1] or a machine learning computer with a GTX 4090 installed. Both computers used Ubuntu as the operating system. The batch size greatly differed depending on which machine the model was trained on since the Acer only has 8 GB of VRAM as compared to the machine learning computer where the GTX 4090 has 24 GB of VRAM. Due to the limited number of available slots to run the experiments for this thesis, the majority of the results were generated using laptops.

3.5.2 Data Set

The training and validation data set, comprising data from various Swedish e-commerce sources, is detailed in Appendix A. The combined data sets contain approximately 20 million products; however, these products are unevenly distributed

[1] https://www.acer.com/us-en/laptops/nitro/nitro-5-amd/pdp/NH.QH1AA.001

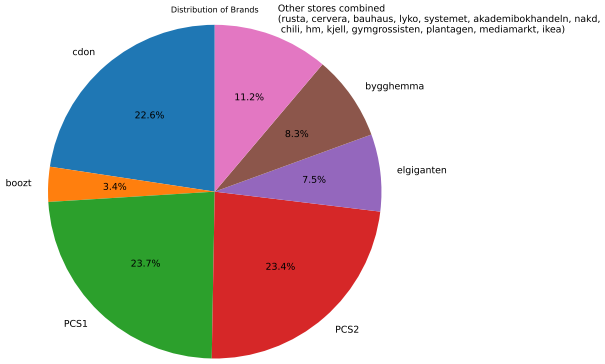


Figure 3.10: Data set Alpha. Product distribution with 1 million products from each store. PCS stands for Price Comparing Site

among the various stores that were crawled. For instance, the Bygghemma data set includes 300,000 products, whereas one of the price comparison sites has over 5 million products. If model were to be trained with this skewed product distribution, Bygghemma's contribution would be minimal compared to that of the larger store. To address this imbalance, it's necessary to form a more evenly distributed subset of products from different stores. Consequently, we have created two balanced data sets for this purpose.

Data set *Alpha*

This data set has a limit of 1 million products as a maximum from each store. This limit resulted in a training file of nearly 4 million products and 20 test data sets with 20% of the products from each store. The reason that the amount is 4 million is that only a few stores have over a million products in total. The decision to have a fixed maximum number of products was to ensure that the model does not get over-fitted for one specific store. In Fig. 3.10 the distributions for the products are shown.

Data set *Beta*

Another smaller training data set was also constructed. This data set was constructed using only 500.000 products from each store which resulted in a train set of 2.7 million products. The distribution is shown in Fig. 3.11

The distribution of products among the different stores is more balanced in Data set Beta.

Validation Set

The validation set is created by sampling 100 products from each store. Since there are 20 stores present the resulting validation set contains 20 different sets with 100

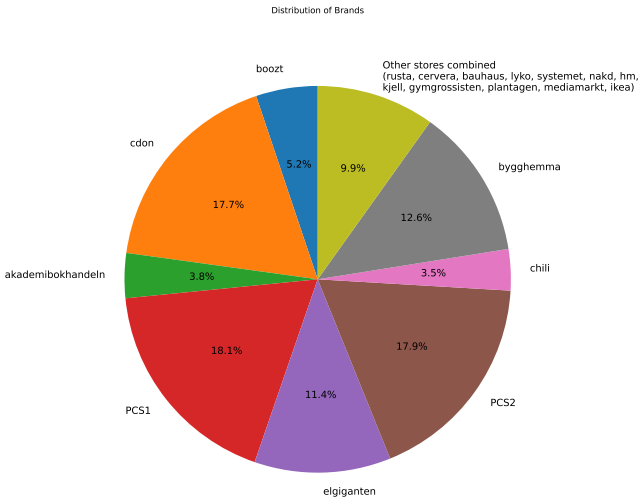


Figure 3.11: Data set Beta. The distribution when using the 500.000 products from each store. PCS stands for Product Comparing Site

products each. The inference program is a computationally heavy one and execution time increases with the number of products in the validation set. By sampling 100 products from each store the time to run the inference program of the models is kept at a reasonable time.

The reason for dividing the validation set into separate sets is that this thesis aims to explore how well a product can be categorized for a given taxonomy. This means that it is interesting to see how a model would classify a product within the taxonomy where it was originally present. Every model in Sect. 3.6 had the same validation set.

Test Set

The test set contains 1000 products from each store, divided into separate sets in order to measure the performance of each store separately. Since the stores were separated, it enabled the model to be tested on a store that was not present during training or validation. Every model in Sect. 3.6 had the same test set.

3.5.3 Source of Pre-Trained Models

The pre-trained transformer models and tokenizers used for this experiment were loaded using the Huggingface interface[2].

[2] `https://huggingface.co/`

Table 3.8: Pre-trained models from Huggingface.

Name	Huggingface
KB-BERT	`KB/bert-base-swedish-cased`
Distilled KB-BERT	`Addedk/kbbert-distilled-cased`

KB BERT

The National Library of Sweden and KBLab[3] released two pre-trained models based on BERT. This model was trained on Swedish texts aiming to provide a representative BERT model for Swedish texts (Malmsten et al., 2020). Furthermore, Distilled KB-BERT was created using knowledge distillation as explained in Sect. 2.3.2.

3.5.4 Model and Parameters

Since this research had a restriction on the hardware it was decided to use Distilled KB-BERT (see Tab. 3.8). This is to ensure that the model could be trained with a higher batch size without having to use gradient accumulation (Jiao et al., 2021). The restriction is based on the VRAM in the GPU. With the laptops, a batch size of 12 was the highest possible number for the VRAM. When using the machine learning computer the batch size could be increased to 56. When running the hyperparameter optimization it was chosen to investigate the parameter combinations found in Appendix B. When performing the training phase, the buffer size was chosen to be 10.000. The different models that were trained all used some variations of the parameters, batch size, NNE, NPD, and train set. This is a result of the different hardware that was presented above. It was also decided to still use the distilled model as the pre-trained model been when running the training on the machine learning computer. The reason was that we wanted to be able to compare the results and see how batch size could impact the accuracy. When the last model was trained, the standard KB-BERT model was loaded but since the model has more layers the batch size had to be reduced to 30.

3.6 Model Summary

Four models will be trained and evaluated, these choices of hyperparameters are the results of the hyperparameter tuning in Sect. 4.2. To evaluate the performance of these models the the metrics accuracy, rank, and loss are used, see Sect. 3.4.2 for more.

3.6.1 Model α

The first model, called *model* α, is trained with dataset Alpha and with the combinations NNE = 2 and NPD = 2. As introduced later this became a hybrid model that was also trained with NNE = 1, NPD = 2.

[3] `https://kb-labb.github.io/`

3.6.2 Model β

The model β used data set Beta and has the combination NNE = 1, NPD = 2, and will be called *model* β.

3.6.3 Model γ

This model also used data set Beta but were trained on the machine learning computer with a bigger batch size. This model will be called *model* γ.

3.6.4 Model δ

The fourth model uses a BERT model instead of a distilled model and has the same combinations as the two above, the batch size will be set to 30. This model will be called *model* δ.

Chapter 4
Results

This chapter presents the results of the experiments and will help answer the research questions presented earlier. Unfortunately results from Liu et al. (2021) were unable to be yielded since adapting the model for this problem was unsuccessful, this is discussed in Sect. 5.3. In Tab. 4.1 an overview of the accuracy's are presented. In Tab. 4.2 the rank-specific scores are presented for all trained models. All models will be separated into sub-sections and more in-depth results will be presented.

The evaluation methods used in this chapter are described in Sect. 2.7.

4.1 Baselines

The following section presents the results of the baselines following the setup described in Sect. 3.5.

4.1.1 TF-IDF

TF-IDF: multi-label classifier

Since TF-IDF is not GPU enabled the decision was made to train the model on a smaller subset based on data set ***beta*** 3.5.2. The baseline was trained on 180000 products and validated using 20000 products, the result is presented in Tab. 4.3. This was also decided since more data could not be loaded into memory and the time constraint disallowed the project to adapt the code to stream the data to the trainer. The model would also have taken way to long to train with all data present in the data set.

Table 4.1: The model top-10 accuracy of model α, model β, model γ and model δ from each store. The average accuracy of the models is also presented. PCS2 and PCS1 are price-comparison sites.

Model	**model α**	**model β**	**model γ**	**model δ**
Stores	**Accuracy (%)**			
PCS2	**98.3**	97.4	96.6	86.8
Lyko	**98.0**	96.8	95.4	96.9
Elgiganten	**95.8**	95.2	86.4	83.2
MediaMarkt	**93.5**	91.2	84.1	90.2
Akademibokhandeln	93.1	90.8	82.5	**95.8**
Chili	**92.5**	89.1	82.1	81.6
Cdon	**89.1**	88.5	75.1	41.9
PCS1	**88.9**	86.0	69.5	86.7
Bygghemma	88.4	**85.4**	68.4	75.8
Ikea	83.2	**84.4**	67.8	60.8
Cervera	**79.8**	79.6	63.7	70.0
Gymgrossisten	**79.7**	78.9	62.2	59.2
Rusta	78.4	**78.7**	60.1	37.0
Bauhaus	**77.4**	73.6	55.9	31.9
Kjell & Company	**77.3**	70.1	40.3	37.9
Systembolaget	**75.9**	67.6	31.3	73.6
Plantagen	**75.7**	65.4	30.7	69.3
NA-KD	**68.6**	64.8	25.3	43.9
Boozt	62.0	54.0	22.8	**75.2**
H&M	45.9	**47.8**	13.2	37.1
Average accuracy	**81.9**	79.3	60.6	66.8

TF-IDF-top-k

The implementation and hypothesis of this baseline were that it should be able to correctly classify products to some extent. However, during evaluation, this method was not able to classify a path correctly, as further demonstrated and discussed in Sect. 5.2.1.

When allowing the baseline to classify a prediction as correct, even when the correct answer started with either `"Startsida"` or `"Hem"`, it was occasionally able to classify products (see Tab. 4.4 for these scores). Without making this exception, no correct predictions could be made, even when allowing the top-10 predictions for each iteration, resulting in an accuracy of 0.00%.

Table 4.2: Rank specific accuracy (%) for all trained models (model α, model β, model γ, model δ). The highest accuracy is shown in bold.

Rank	**model α**	**model β**	**model γ**	**model δ**
10	**81.9**	79.2	60.6	66.8
9	**81.2**	78.3	59.7	65.9
8	**80.1**	77.1	58.5	64.7
7	**78.9**	75.8	57.3	63.2
6	**77.4**	74.4	55.7	61.6
5	**75.7**	72.2	53.3	58.5
4	**73.3**	69.7	50.8	55.8
3	**69.6**	66.2	46.1	50.7
2	**64.5**	60.9	41.0	43.6
1	**53.6**	50.3	30.6	30.4

Table 4.3: Scores from *TF-IDF: multi-label classifier* baseline

Metrics	**%**
Accuracy	13.3
Precision	83.6
Recall	24.0
F1-Score	3.74

Table 4.4: Scores from TF-IDF-top-10 baseline

Metrics	**%**
Accuracy	9

Table 4.5: BERT Zero-shot classification score

	Accuracy (%)
Concatenated path	7.49
Not concatenated	4.22

4.1.2 BERT zero-shot classification

The results were based on the test set used for all models but were interrupted after 12 hours each, the reason for interrupting them is that it would have taken too long to run the model on the whole set since it has not been trained for this specific task, this lead to the model nearly having to investigate nearly every path before coming to a final prediction. The prediction was carried out in two different ways, as explained in Sect. 3.2.2. The results can be seen in Tab. 4.5 below.

4.1.3 Using Pre-trained Model

The results were created by running the inference program but instead using a non-fine-tuned model. The pre-trained Distilled KB-BERT model was loaded into the inference program which was run on the test set and the result can be seen in Tab. 4.6

A recurring theme observed across all baseline models is their limited capacity to produce satisfactory results, highlighting the need for further exploration of more effective methodologies.

4.2 Hyperparameter Tuning

As mentioned previously, several steps are undertaken to ascertain that the model is equipped with optimal training parameters. The initial step towards achieving this optimization is the performance of hyperparameter tuning.

During the hyperparameter optimization process, 12 models were individually trained using 50,000 batches, with parameter adjustments as outlined in Sect. 3.6. The resulting losses are presented in Fig. 4.1. The table within the figure illustrates the combination of NNE and NPD values for each trained model.

After the training was complete a validation inference was run on all different models, this was done to figure out the best parameters for final training. In Tab. 4.7 the results are presented. For each test the average accuracy was created over all stores that the model was validated on.

Since the learning rate (LR) of 1e-05 proved to have the best score and all combinations were made with LR 1e-06, a new hyper-parameter optimization was made, this time with the second parameter combinations found in Appendix B. A problem with this approach is however that the trainer stops at 50.000 batches since the different parameter combinations lead to different amounts of training data for the same amount of products, due to time constraints this could not be taken into further consideration. The Loss for the second combinations is shown in Fig. 4.2

Table 4.6: Store-specific accuracy for pre-trained model baseline. Note no product count is included since this model is not trained.

Stores	**Accuracy** (%)
plantagen	42.0
boozt	23.0
akademibokhandeln	6.0
cervera	2.0
gymgrossisten	2.0
nakd	1.0
chili	1.0
hm	0.0
PCS1	0.0
kjell	0.0
ikea	0.0
rusta	0.0
elgiganten	0.0
cdon	0.0
bygghemma	0.0
lyko	0.0
PCS2	0.0
mediamarkt	0.0
clasohlson	0.0
systemet	0.0
bauhaus	0.0
Average accuracy	4.0

Table 4.7: The average accuracy score for every combination. The averaging is done over the individual store accuracies. For NNE and NPD optimization a learning rate of 10e-6

Hyper-parameters		**Accuracy (%)**
Learning rate		
1e-05		67
1e-06		52
1e-04		35
NNE	**NPD**	
1	3	61
1	2	61
3	3	60
2	2	59
2	3	58
3	2	58
2	1	56
1	1	55
3	1	53

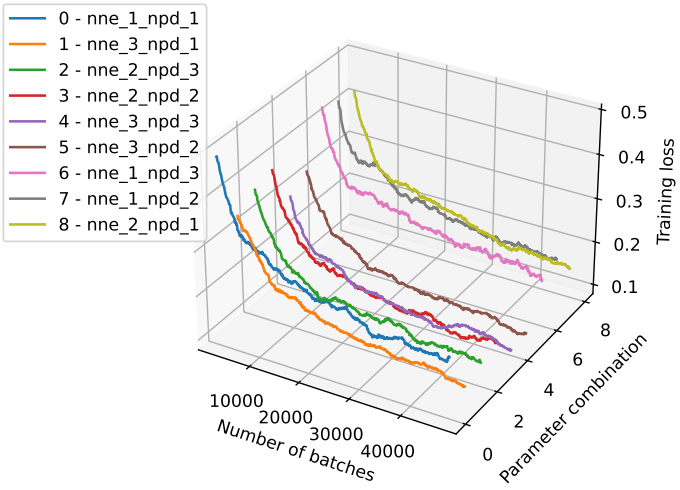


Figure 4.1: The different losses during training.
Labels `nne_x_npd_y` means that the model used NNE=`x` and NPD=`y`.

and the scores from that run are shown in Tab. 4.8. The second run instead used 50.000 products from the training set without stopping and then validated using the same validation set. These changes were made due to time constrictions.

The different parameter combinations will generate different amounts of negative data, which then directly affects the ratio of positive and negative examples for the training set as explained in Sect. 3.3.1. In Fig. 4.2, one could see that the amount of batches changes depending on the combinations. The positive/negative ratio is presented in Tab. 4.8. The time needed for training, therefore, changes relative to the combinations and was, therefore, an influencing factor. In Tab. 4.8 the time needed for training for the different combinations is presented. After the optimization was finished the findings were analyzed and the final training was started.

4.3 Training and Evaluation

In this section, the models will be presented. To create a better outline the models will be divided into their own subsection. The models that were trained are presented in Sect. 3.6. The reasoning for the parameters that were chosen will be further discussed in the next chapter. The models' evaluation will also be presented in its own subsection. To see the progress of each step of the training, the inference was run on as many checkpoints saved as possible. This is due to the time needed to perform one validation. The results are also based on the last saved model when training was finished and use the evaluation described in Sect. 3.6.

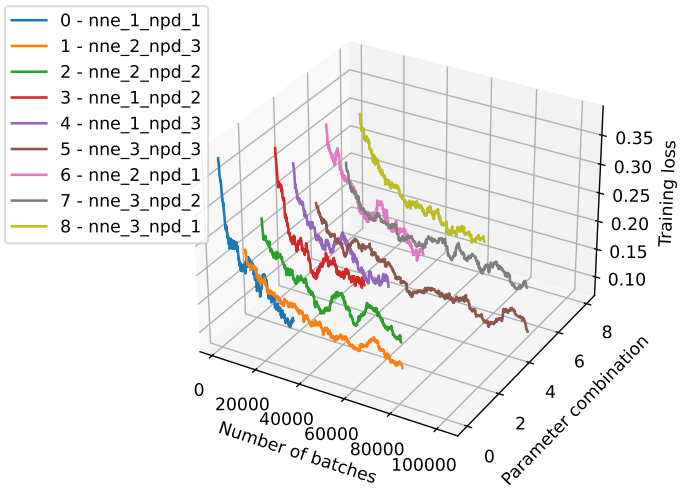


Figure 4.2: Training loss during second hyper-parameter optimization. Labels `nne_x_npd_y` means that the model used NNE=`x` and NPD=`y`.

Table 4.8: Training data from the Second hyper-parameter optimization. +/- stands for positive to negative batch ratio. The accuracy averaging is done over the individual store accuracies.

Hyper-parameters				
NNE	**NPD**	**Accuracy (%)**	**+/- ratio**	**Training time (hours)**
1	2	59	0.41	1.3
1	1	59	0.58	1
2	1	55	0.37	1.5
1	3	54	0.39	1.4
3	1	49	0.29	1.9
2	2	44	0.26	2
3	3	40	0.17	3.1
2	3	37	0.24	2.2
3	2	34	0.21	2.7

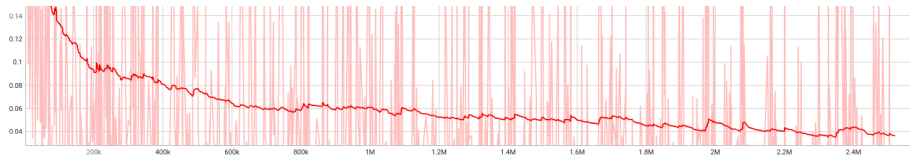


Figure 4.3: Training loss during training.

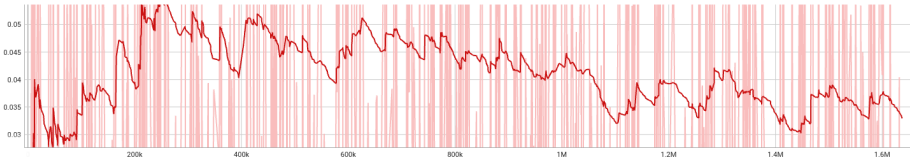


Figure 4.4: Training loss loading the last checkpoint and changing the parameters.

4.3.1 Model α

Results from training and evaluating model α.

Training

The training was done with the parameters found in Appendix C and the training loss for model α can be seen in Fig. 4.3 and Fig. 4.4. Unfortunately, the computer training model α unexpectedly crashed after approximately 2.5 million batches or close to 72 hours. The last checkpoint was loaded and a new training was started around the same place in the data set. The parameter was also changed to NNE = 1, NPD = 2. This change in combination was a mistake but proved to be better in accuracy as well the training time would have been too long with the old combination of NNE = 2 and NPD = 2. Since the parameters were changed the model becomes a hybrid of the combinations and thus a new name was needed. The decision was to name the final model as model α and the partially trained model as pre-model α.

Evaluation

The progress of this training is represented in Fig. 4.5. the start of the training after the unexpected is viewed from checkpoint 2500000 and onwards.

The saved model was loaded into the inference program and tested using the test set described in Fig. 3.5.2. The total training took 129 hours where the pre-model α ran for 72 hours and the rest of the training took 57 hours. In Tab. 4.9 and Tab. 4.10 the accuracy and rank are presented for the pre-model α. In Tab. 4.11 and Tab. 4.12 the final result for model α is presented.

Unseen-data

The model was also run on Biltema and Clasohlsons which was not present during training, the results are presented in Tab. 4.13.

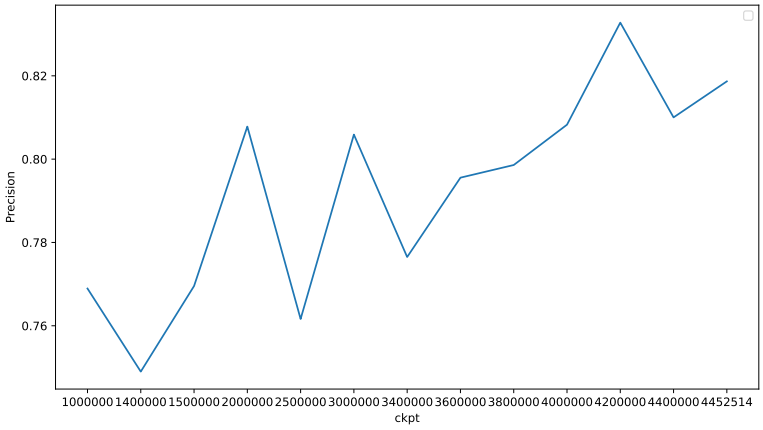


Figure 4.5: Progress of training for model α.

Table 4.9: The store-specific accuracy and average for pre-model α. Products indicate the number of products from a store in the training set.

Stores	**Accuracy (%)**	**Products**
PCS2	98.4	98300
Lyko	94.8	67900
Elgiganten	94.6	314000
Mediamarkt	92.8	14700
Akademibokhandeln	91.6	103000
Chili	90.8	95000
Cdon	89.8	950000
PCS1	83.8	997000
Bygghemma	83.0	347000
Ikea	75.8	30000
Cervera	74.5	16300
Gymgrossisten	74.1	2280
Rusta	72.1	6160
Bauhaus	70.9	55800
Kjell & Company	68.9	9150
Systembolaget	68.5	23200
Plantagen	63.5	7780
NA-KD	49.9	23100
Boozt	46.5	142000
H&M	38.9	15500
Average accuracy	76.2	

Table 4.10: The rank-specific accuracy for pre-model α.

Rank	Accuracy (%)
10	76.2
9	75.4
8	74.1
7	72.8
6	71.4
5	69.4
4	67.1
3	63.3
2	57.5
1	46.9

Table 4.11: The store-specific accuracy and average for model α. Products indicate the number of products from a store in the training set.

Stores	Accuracy (%)	Products
PCS2	98.3	98300
Lyko	98.0	67900
Elgiganten	95.8	314000
Mediamarkt	93.5	14700
Bygghemma	93.1	347000
Chili	92.5	95000
PCS1	89.1	997000
Gymgrossisten	88.9	2280
Akademibokhandeln	88.4	103000
Rusta	83.2	6160
Systembolaget	79.8	23200
Plantagen	79.7	7780
Cervera	78.4	16300
Bauhaus	77.4	55800
Kjell & Company	77.3	9150
Cdon	75.9	950000
Ikea	75.7	30000
NA-KD	68.6	23100
Boozt	62.0	142000
H&M	45.9	15500
Average accuracy	81.9	

Table 4.12: The rank-specific accuracy and average for model α.

Rank	Accuracy (%)
10	81.9
9	81.2
8	80.1
7	78.9
6	77.4
5	75.7
4	73.3
3	69.6
2	64.5
1	53.6

Table 4.13: Accuracy on unseen stores and taxonomies.

Store	Accuracy (%)
Biltema	16
Clasohlson	16

4.3.2 Model β

Results from training and evaluating model β.

Training

The model β can be seen in Fig. 4.6. The training of this model took 72h on the hardware introduced in Sect. 3.5. The training was done with the parameters found in Appendix C and the training loss for model β can be seen in Fig. 4.6. The training was finished after 72 hours

Evaluation

The progress of this training is represented in Fig. 4.7.

The accuracy is presented in Tab. 4.14 and accuracy at every rank is shown in Tab. 4.15.

Unseen-data

The prediction accuracy on the unseen stores is presented in Tab. 4.16.

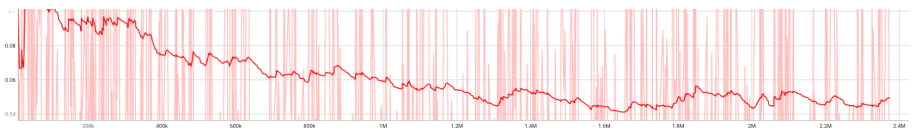


Figure 4.6: Training loss during training of model β.

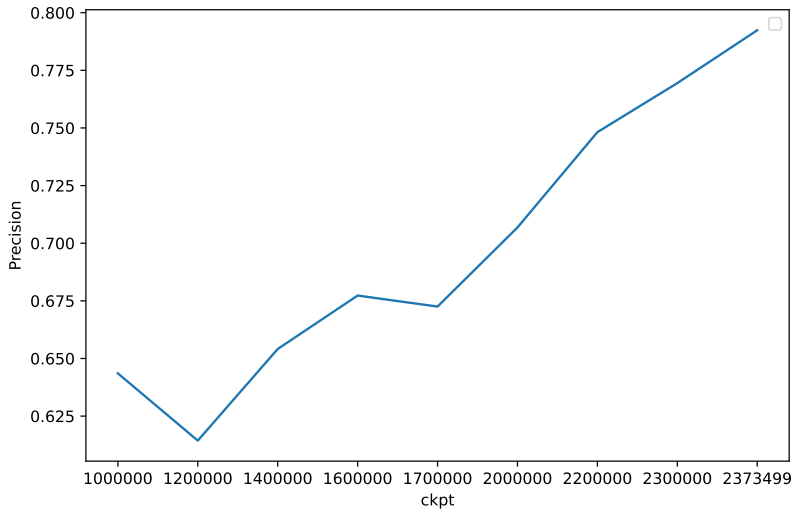


Figure 4.7: Progress of training for model β.

Table 4.14: The store-specific accuracy and average for model β. Products indicate the number of products from a store in the training set.

Stores	**Accuracy (%)**	**Products**
Lyko	97.4	67900
PCS2	96.8	494000
Elgiganten	95.2	314000
Akademibokhandeln	91.2	103000
Bygghemma	90.8	347000
PCS1	89.1	499000
Mediamarkt	88.5	14700
Chili	86.0	95000
Gymgrossisten	85.4	2280
Cdon	84.4	486000
Cervera	79.6	16300
Systembolaget	78.9	23200
Ikea	78.7	30000
Bauhaus	73.6	55800
Plantagen	70.1	7780
Boozt	67.6	142000
Kjell & Company	65.4	9150
Rusta	64.8	6160
NA-KD	54.0	23100
H&M	47.8	15500
Average accuracy	79.3	

Table 4.15: The rank-specific accuracy for model β.

Rank	Accuracy (%)
10	79.2
9	78.3
8	77.1
7	75.8
6	74.4
5	72.2
4	69.7
3	66.2
2	60.9
1	50.3

Table 4.16: accuracy on unseen stores and taxonomies.

Store	Accuracy (%)
Biltema	10
Clasohlson	7

4.3.3 Model γ

Results from training and evaluating model γ.

Training

The training loss is illustrated in Fig. 4.8. The training process was finished after 36 hours.

Evaluation

The accuracy can be seen in Tab. 4.17 and the accuracy at every rank can be seen in Tab. 4.18. Furthermore, the progress graphs are presented in Fig. 4.9.

Unseen-data

4.3.4 Model δ

Results from training and evaluating model δ.

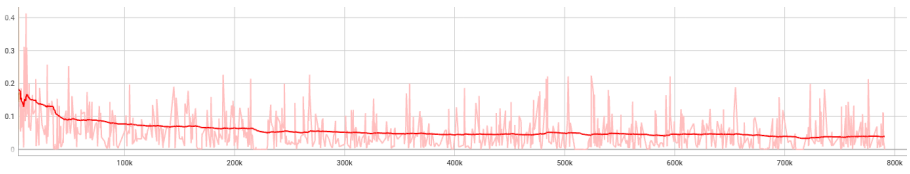


Figure 4.8: Training loss when training on 4090 graphic cards.

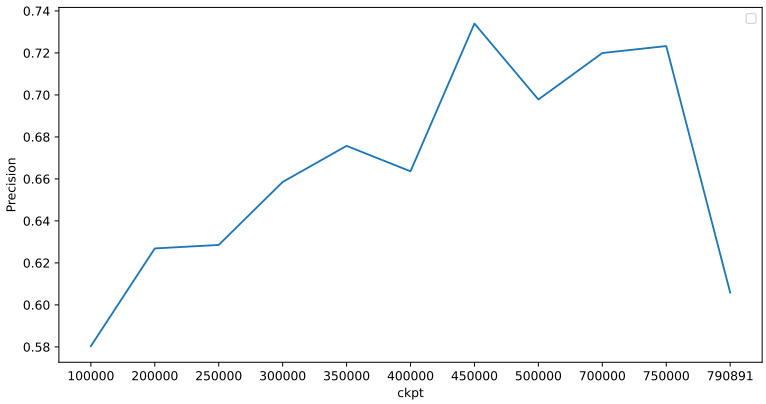


Figure 4.9: The training progression for the model trained on GTX 4090

Table 4.17: The store-specific accuracy and average for model γ. Products indicate the number of products from a store in the training set.

Stores	**Accuracy (%)**	**Products**
Akademibokhandeln	96.6	103000
Lyko	95.4	67900
PCS2	86.4	494000
Elgiganten	84.1	314000
PCS1	82.5	499000
Chili	82.1	95000
Gymgrossisten	75.1	2280
Boozt	69.5	142000
Mediamarkt	68.4	14700
Cdon	67.8	486000
Systembolaget	63.7	23200
Cervera	62.2	16300
NA-KD	60.1	23100
Plantagen	55.9	7780
H&M	40.3	15500
Bygghemma	31.3	347000
Ikea	30.7	30000
Rusta	25.3	6160
Kjell & Company	22.8	9150
Bauhaus	13.2	55800
Average accuracy	60.6	

Table 4.18: The rank-specific accuracy from model γ

Rank	Accuracy (%)
10	66.8
9	65.9
8	64.7
7	63.2
6	61.6
5	58.5
4	55.8
3	50.7
2	43.6
1	30.4

Table 4.19: Accuracy on unseen stores and taxonomies for model γ.

Store	Accuracy (%)
Biltema	3.00
Clasohlson	2.20

Training

This model was trained with the same parameters as model β but as explained in Sect. 3.6 this model used the pre-trained model *KB/bert-base-swedish-cased* instead of a Distilled KB-BERT model. It was possible to use this 12-layered BERT model since it was trained on the 4090 graphic card. The logs during training were somehow they got split after 30 hours, this explains why there are two graphs. the model was finished after 42 hours of training Fig. 4.10 and Fig. 4.11 . The accuracy and rank are shown in Tab. 4.20 and Tab. 4.21.

Evaluation

The progress during training is shown in Fig. 4.12, and the accuracy and rank-specific accuracy is found in Tab. 4.20 and Tab. 4.21.

Unseen-data

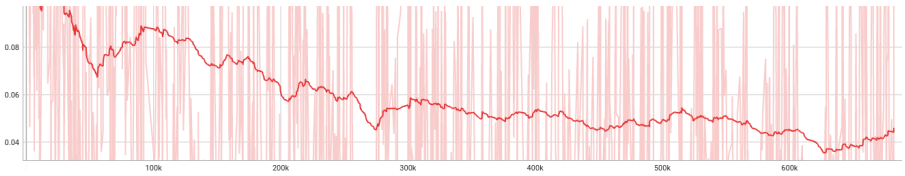


Figure 4.10: Training loss during the training of BERT model (4090) before interruption

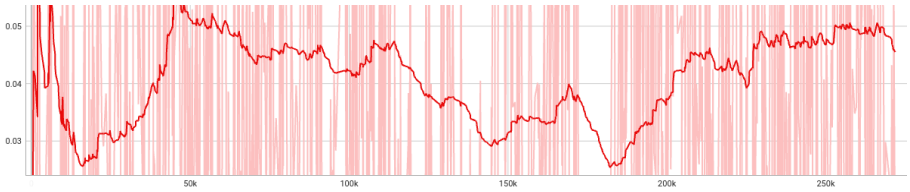


Figure 4.11: Training loss during the training of BERT model (4090) after it was resumed

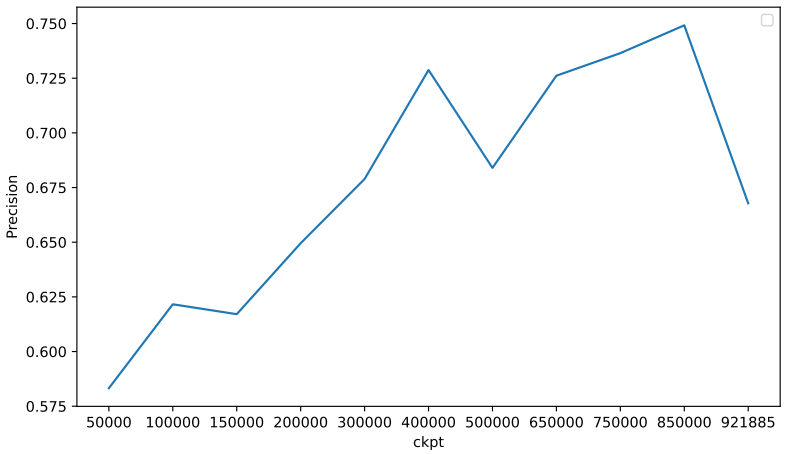


Figure 4.12: Training loss during the training of model δ

Table 4.20: The store-specific accuracy and average for model δ. Products indicate the number of products from a store in the training set.

Stores	**Accuracy (%)**	**Products**
Lyko	96.9	103000
Akademibokhandeln	95.8	67900
Mediamarkt	90.2	494000
PCS2	86.8	314000
PCS1	86.7	499000
Elgiganten	83.2	95000
Chili	81.6	2280
Bygghemma	75.8	142000
Boozt	75.2	14700
Systembolaget	73.6	486000
Cervera	70.0	23200
Plantagen	69.3	16300
Ikea	60.8	23100
Gymgrossisten	59.2	7780
NA-KD	43.9	15500
Cdon	41.9	347000
Kjell & Company	37.9	30000
H&M	37.1	6160
Rusta	37.0	9150
Bauhaus	31.9	55800
Average accuracy	66.8	

Table 4.21: The rank-specific accuracy from model δ.

Rank	**Accuracy (%)**
10	66.8
9	65.9
8	64.7
7	63.2
6	61.6
5	58.5
4	55.8
3	50.7
2	43.6
1	30.4

Table 4.22: Precision on unseen stores and taxonomies for model δ.

Store	**Precision**
Biltema	15.4
Clasohlson	4.50

Chapter 5
Discussion

In this chapter, we will discuss the results, what did not work, and the possible sources of error. Discussion regarding ethical aspects will also be conducted.

5.1 Data Set

When training a model the data quality is a key component to be able to generate a well-performing model. In this section discussion about the data set that was provided by Theca Systems will be performed. The data set was created prior to this project to enable this research and contains as described earlier different product data, see Sect. 3.1. To make sure the data is of high quality, data cleaning is an important aspect of model training which includes making sure that the data is suitable for the training. This section will discuss other problems that might have affected the training. It is hard to guarantee that the data set holds a high standard since there are so many products, when the products were crawled. There are methods of removing HTML tags and other Unicode data that should not be in the final data set. During the research, it was found that some HTML code and Unicode had slipped through during the cleaning process. But even with these flaws, it seems like the models learned other complex relationships in the data and could produce accurate predictions.

Another potential problem in the data set was discovered during the training of the models. It became clear the product distributions across the stores were unbalanced since three stores contributed to 70% of the training data. This was due to the different amounts of products present in each store. It was therefore decided to create another training set that included 500,000 from each store instead, which led to a much more well-balanced product distribution.

Table 5.1: A few examples of products predictions from top-k baseline

Predicted path	`["gaming & underhållning", "böcker"]`
Correct path	`["hem", "gaming & underhållning", "underhållning", "böcker"]`
Predicted path	`["golv, vägg & tak", "malarfarg-och-tapet", "fototapet"]`
Correct path	`["renovering & bygg", "golv, vägg & tak", "tapeter"]`
Predicted path	`["sortiment", "rött vin"]`
Correct path	`["hem", "sortiment", "vin", "rött vin"]`
Predicted path	`["fordon & tillbehör", "bildäck", "fordon & tillbehör", "sommardäck"]`
Correct path	`["hem", "fordon & tillbehör", "däck och fälgar", "sommardäck"]`

5.2 Baselines

In this section the results from the baselines will be discussed and analyzed.

5.2.1 TF-IDF

As explained in Sect. 2.2.4, TF-IDF is one way of classifying texts to a certain taxonomy, the implementation was explained in Sect. 3.2.1. The reason why there were so many attempts to create a working baseline was to enable better model comparisons with the developed models and already existing methods. One could see in Sect. 4.1.1 that all baseline approaches performed very poorly.

TF-IDF: multi-label classifier was the only one that actually managed to correctly classify products. It however had, one serious flaw: this model did not check top-k predictions and, it did not predict category paths dynamically, since it only made one prediction and checked the top predicted path. A top -k approach should have been implemented to enable equal comparison with the developed model since the accuracy score changes quite rapidly when looking at what rank the correct answer is found. This was the main reason for creating other baselines.

TF-IDF-top-k used the same approach as our developed model during evaluation and showed that it was not able to produce paths even when continuously exploring the top-k predictions, that is dynamically predicting category paths. In Tab. 5.1 one can see a few examples from the developed model.

Both of the approaches have a flaw in common: they both lose their hierarchical structure since the taxonomy is flattened during training. This means that both models could potentially predict paths that can not possibly exist.

5.2.2 Zero-shot Classification and Pre-trained model

The reason to use zero-shot classification as a baseline is to test how well a pre-trained zero-shot classification model would compare to a fined-tuned sequence classification model.

Furthermore, it was also tested how well a pre-trained sequence classification model would compare to the final model in terms of accuracy. This could be easily achieved since the pre-trained model could use the same inference program that was written for the final models

Table 5.2: Product example from plantagen

Title	Description	Category
Amaryllis	Vad vore väl vinter och jul utan amaryllis...	Hem/växte innomhus/julblommor/Amaryllis

As seen in the results for both models Sect. 4.1.2 and Sect. 4.1.3, both of them were not able to correctly predict categories. Again this was no surprise since none of the models have been taught the specifics of the task on which they have been used to make predictions. The results that zero-shot classification and a non-fine-tuned model produce, are comparable to wildly guessing what category a product would belong to. It was however interesting to see that the latter could correctly categorize products belonging to Plantagen, It is believed that this was possible since their category taxonomy is the naming of plants for example, which enables the model to correctly classify a text towards that category. in Tab. 5.2 we can see an example where the categories are well described in the description of the text

5.2.3 Conclusion of baselines

As discussed above, all baselines created showed poor results. The results show that an already existing method can not be easily applied and produce good scores. Even though some baselines had some flaws, eg. not predicting dynamically or only checking the top prediction, this paper has shown that already existing techniques are not sufficient to solve the problem statement of this paper This is further discussed in chapter 6

5.3 Model proposed by Liu et al. (2021)

As stated earlier, much time was spent trying to get the model suggested in Liu et al. (2021), to work, unfortunately, no results could be yielded using the model. In this section, the paper aims to explore and discuss the issues as to why this model did not work. From here **model based on paper (MBP)** will be used as an abbreviation.

5.3.1 Fine-tuning

During the fine-tuning of MBP, another problem came to light. After a couple of iterations, the loss became zero and would not change. Even though, a low loss could indicate that a model is well enough trained it was unexpected to see it go down so far after just a couple of iterations. There was no way for MBP to learn what paths are correct at that speed due to the size of the taxonomy, i.e. the number of different paths that exist, and the number of products in the training set. Our hypothesis is that there is an unidentified error in how the model creates these paths and therefore, this behavior occurs. However, it is possible to train the model without fine-tuning it first and it was therefore decided to focus on getting the other step to work first.

5.3.2 Reinforcement Learning

Another problem was discovered in the second step of the training, i.e. the reinforcement learning step. When the training was underway, it became clear that the policy was not working correctly. After a couple of hundred steps, it started predicting categories with a path length of 1. As stated in Sect. 2.5 the path gets rewarded if the predicted path is a strict subset of the correct path. It seemed like the model was penalized when trying to further predict the path and therefore learned that it should stop early since this meant a lower risk of punishment. Even when the policy was changed to reward the correct nodes and penalized the wrongly chosen ones, the model could not be trained using our data. The policy works by running a prediction on certain nodes and then checking the probability of these nodes and whether it should continue exploring that path or not. It became clear that the policy was not working correctly since there existed iterations where no nodes were chosen for further exploration which led to the model crashing. Even after a new code was added that made sure that the right path was added to every exploration, the policy still performed in the same manner. The code was further modified to skip the steps where no further nodes were returned to be explored. This led to the model running for about 1% of the data and then skipping the rest.

5.3.3 Hardware Restrictions

The MBP was also clearly developed with no hardware restrictions in mind, the research was conducted at a company with greater assets than available for this thesis. The architecture was not built to be able to run on a normal laptop with a consumer-grade graphics card. Due to the above-discussed problems, a decision was made to create a new model built from scratch to fit this task's needs. Unfortunately, this should have been done a lot earlier since it constricted the amount of time that was available to develop and tweak the new model.

5.4 Sequence classification model

In this section, the implemented models will be discussed. It will be divided into subsections in order to discuss the different sections regarding training and validation more clearly.

5.4.1 Model training

Hyper-parameter Tuning

Finding the optimal hyper-parameters was one of the biggest challenges for this task. The Equation 5.1 gives an estimate of how many entries the model would have with different parameters. The estimation is based on the maximum number of possible path combinations. In the equation 5.1, p = entries for each product, n = length of the path.

$$\begin{aligned} p &= n + 1 + (n + 1) \cdot (NNE \cdot NPF) + 1 = \\ &= (n + 1)(1 + NNE \cdot NPD) + 1 \end{aligned} \tag{5.1}$$

The Equation 5.1 is the absolute max that could be generated. It was not possible to estimate the correct number of entries since certain paths do not have multiple children and some paths do not have the desired depth. Another reason is that the model only generates false paths beyond the final path 10% of the time and early stopping with 5% of the time for the products in the data set as explained in Sect. 3.3.1. In Equation 5.1 it is assumed that the generation always goes beyond the final path and has one early stop path for every product. The motivation for having probabilities is to keep the number of entries to the model as low as possible to reduce training time, but at the same time, ensure that the model learns the behavior. This leads to the entries being lower than the estimated amount of entries.

The numbers presented in the first hyper-parameter optimization, in Tab. 4.7, are therefore misleading since the models with a higher number on NNE and NPD, see a lot more different negative examples for each product compared to the NNE and NPD with lower numbers. This also led to the models not seeing the same number of examples since every training was stopped after 50.000 batches. However, it was later identified that the scores in Tab. 4.7 were not correct. This was mainly due to the fundamental architecture error of the implementation of NNE. The error entailed that the creations of the negative paths were not correct, no matter the value of NNE it only generated one negative example with desired depth per level that was plausible. Two other errors were also found during the second optimization process and the optimization was therefore canceled. The two other errors that were found were first, the model did not generate false paths beyond the correct answer, which meant that the model could potentially continue even though it should not. The second error was that the model was never trained on paths that went too short with respect to the correct path. These errors make the scores in table 4.7 obsolete since they do not correspond to the task they were meant to discover. This does however explain why the accuracy is highest for all combinations with NPD = 3 and the lowest for all combinations with NPD = 1, since the trainer saw less data with NNE higher than one with the exclusion of the combination NNE = 1, NPD = 1.

Due to these errors, a third hyper-optimization was performed after the bug was fixed. However, due to the time constraint of the project, the number of products was lowered from 100.000 to 50.000 to ensure a faster process. As mentioned above the time constraint restricted the number of products that were used during the final hyper-parameter optimization, which explains why the accuracy score is significantly lower. It was however not expected that a higher number of NNE and NPD would produce lower accuracy since the model was allowed to finish its training no matter the combination. This shows that the positive/negative ratio of the training batch is important to create a precise model and that more negative examples do not equal a better model. Instead, the models seem to prefer having a more balanced training set. The reason for this result could be that the model gets somewhat over-fitted and therefore does not produce the right answers.

When the optimization was run, the time needed for training was captured as illustrated in Tab. 4.8. The time needed for training was taken into account when

the parameters were chosen for the final training because time is a limited asset and it is assumed that more training data implies a better model.

Model training

The models were trained according to Sect. 3.6. The reason for choosing to train models with different parameters was to be able to compare the difference in accuracy. The combinations were chosen since the models were only trained on a subset of 50.000 when in reality the complete training file consists of approximately 4 million product entries and for the latter model, 2.7 million. Since the model randomly generates false paths it is theorized that a low number is sufficient since there is a lot more data to train on and therefore all paths will be explored. Additionally, having too high NNE and NPD seems to have over-fitted the model since it performed lower in the validation test, as seen in Tab. 4.8. This could correlate to the importance of the positive/negative ratio when training.

As briefly mentioned in Sect. 4.3.1 the model α crashed unexpectedly. Enough information was fortunately saved to be able to restart the training. A mistake was made and the NNE and NPD parameters were changed to NNE = 1, NPD = 2, this was not initially the plan but it proved to be reasonable since the training from the recovery point took 57 hours, this would have been a lot higher if the previous parameter combination was used. The results were also surprising since it performs as well or even better than the model with NNE = 1, NPD = 2. These models do however differ in which data set they are trained on since the first used 1 million products from each store whereas the other used 500.000. One could also see that the model seems to be performing better on the stores that had close to 1 million products, this is shown in Tab. 4.9. This could however be dangerous since over-fitting the model for these stores could happen.

The model β finished after 72h and as one can see in Fig. 4.7 it never fluctuated across the checkpoints, one could theorize that the accuracy could go a bit higher if more products were used during training as seen for the model α in Fig. 4.5.

The training data and taxonomy are constructed in a way that ensures the broadest possible training. Since taxonomies are merged from multiple sites the model is enabled to be trained on potential paths that do not exist in the crawled data. The model also becomes more robust as it gets random products from multiple stores at the same time, which helps to reduce over-fitting on certain stores. By doing this, it also enables the model to be trained in a more general manner.

When model γ was trained on the machine learning computer at the end of this project the bigger batch size was chosen but with the same parameters as the first trained model. This was done in order to be able to compare how and if the batch size had a noticeable effect on the accuracy. As one can see in the results presented in Tab. 4.17 the accuracy did not increase with a bigger batch size, it seems to have performed worse than the models trained on the laptops with a smaller batch size, this was an unexpected result since the hypothesis would be that it would perform at least as good as the others. The results from training with different batch sizes indicated that a bigger batch size seems to impact the accuracy in a negative way.

All of the models that were trained used a fixed learning rate of 1e-5. If more

time was available, the next step would have been to implement a scheduler that changed the learning rate during training. This is something that will be presented as future work.

5.4.2 Model Evaluation

This work aimed to research how well a model could predict a product to a belonging category taxonomy, as stated in the problem statement (Sect. 1.2). This is the fundamental reason why the test sets and taxonomies are divided into separate stores. When predicting a product the user would only be interested in how the model would classify the product to a given taxonomy. For example, it could be interesting to see how the model would predict a product from a certain store and then see what category it would predict given the taxonomy was changed. A great example is Pricerunner. Pricerunner has products on their websites that are categorized according to their own category taxonomy. Since Pricerunner does not sell any products they will link to websites that sell the product and by following a link the same product is found but categorized to that store's taxonomy instead. This model would enable a user to take a product text and predict what category it belongs to with regard to any store's taxonomy.

Allowing the model to continuously explore the 10 most probable paths enables the model to correctly predict the right path even if there are multiple highly likely paths. The further down in a taxonomy tree, the lower the possibility gets since fewer and fewer products belong to each category. This makes sense when looking at products on a website-based store. For example, *dishwashers* and *fridges* both belong to kitchen appliances whereas they most likely belong to their own leaf node, *Fridges/freezers*, and *Dishwashers* are examples taken from elgiganten.se[1]. This means that a product text for a dishwasher or a fridge would have a higher possibility of belonging to kitchen appliances compared to further down in the tree. The trained models resolve this issue by always getting rewarded for partly correct paths as well as a `<STOP>` when the end is reached, details of the implementation are presented in Sect. 3.3.1.

The inference is based on the same idea as above of predicting the 10 most probable paths and storing them at the end of each inference iteration. The reason why the 10 most probable predictions are stored was after discussions among the people involved. The rank measurement captures how the accuracy changes further down the list that is accepted as a correct answer. This corresponds to the data in table 4.10.

Comparing DistilBERT vs BERT

All the models except model δ used a distilBERT pre-trained model. Our hypothesis was that a larger BRET model with more layers would be able to find more complex relationships during training and then perform better than the other models. This was however not correct, as illustrated in Tab. 5.3 we can see that the BERT-based model (model δ) proved to be less accurate in nearly all store predictions. What

[1] `https://www.elgiganten.se/`

the comparison indicates is that model distillation works and one can keep the performance of a BERT even with a smaller model.

Table 5.3: Accuracy comparison between the best DistilBERT (model α) and BERT (model δ).

Stores	**Model**	
	model β	model δ
PCS2	**97.4**	86.8
Lyko	96.8	**96.9**
Elgiganten	**95.2**	83.2
MediaMarkt	**91.2**	90.2
Akademibokhandeln	90.8	**95.8**
Chili	**89.1**	81.6
Cdon	**88.5**	41.9
PCS1	86.0	**86.7**
Bygghemma	**85.4**	75.8
Ikea	**84.4**	60.8
Cervera	**79.6**	70.0
Gymgrossisten	**78.9**	59.2
Rusta	**78.7**	37.0
Bauhaus	**73.6**	31.9
Kjell & Company	**70.1**	37.9
Systembolaget	67.6	**73.6**
Plantagen	65.4	**69.3**
NA-KD	**64.8**	43.9
Boozt	54.0	**75.2**
H&M	**47.8**	37.1
Average accuracy	**79.3**	66.8

5.4.3 Summary of Results

As one can see in Tab. 4.1, all the produced models are very close to each other regarding accuracy. This shows that with a large enough data set, enough of the paths get explored, and therefore higher NNE and NPD are not needed. As they only entail longer training times, one could instead use the lower combinations and run the model through multiple epochs to get the same outcome. It is also seen that the model could not predict some stores very well which can be due to a few reasons. One reason could be that the crawled text is not sufficient and therefore the model does not have much to predict on. Another reason could be that the model does not find good relationships between the text and the category. This could be the reason why H&M and NA-KD are so low. Trying to classify a text about a t-shirt is a very hard problem since it could belong to many different categories such as women, men, children, etc. When looking at the results all stores selling clothes are performing right at or below average. One can also see that number of products during training does not have to yield better performance. One example can be seen

in Tab. 4.11, where CDON was trained with 950000 products and still performed lower than Gymgrossiten which was trained with 2280 products.

One can also see by looking at the progression graphs, that a better model could be loaded and produce better results than what the tables present. For this research, it was decided to produce the tables based on the last saved models which was done when the training was completed. Furthermore one could see that a higher batch size resulted in a lower average accuracy.

This thesis was also able to do a brief comparison of a fine-tuned DistilBERT model and a BERT model. The findings show that the DistilBERT almost outperforms the regular BERT, which proves that the performance of a distilled model can be quite good compared to the regular one.

This approach proved to work very well compared to all the baselines that were made. Even when comparing the top-1 scores, the models outperformed all baselines with an accuracy of 53.6% compared to the best baseline which had an accuracy of 13.3%. These numbers are found in Tab. 4.10 and Tab. 4.3. However, it was unfortunate to see that the model was not able to predict unseen stores very well as seen above, and therefore not viable to use as a general model. We do believe, however, that more testing should be done to confirm this behavior. The reasoning for this is that the stores that were tested as unseen was removed from the training due to the crawled data containing a lot of errors. By applying this model to some other store that was not present during training one could contradict our findings.

5.5 Ethics

Constructing an AI model can be both a curious and technical challenge, where new solutions are found for unsolved problems. But while doing this, one needs to consider some ethical aspects. The following are a few aspects relevant to the work in this thesis.

5.5.1 Carbon Emission

The evermore popular transformer-based models which have headlined the news during the past month have an overlooked cost, which many might not reflect upon. In Hao (2019) it is stated that training of such models has an immense environmental impact, where the data is the fossil fuel and the model is the motor that burns it. Training a transformer model with 213 million parameters could release as much as $284,000$ kg CO_2, which is more than the lifetime emissions for many of us. When planning to create these models, one should therefore be aware that training these largest models can have a major impact on the environment.

However, once these models are created they can be used again, e.g. BERT, and maybe most importantly they can be fined tuned. So instead of training a large model from scratch each time, it can be developed to solve a specific problem.

5.5.2 Affect on Consumer Behaviour

As mentioned in the introduction could this model be part of a solution where traffic is directed between e.g. web-based stores. The idea behind this approach is to lower the amount of money spent on advertisement services from Google. But an unwanted side effect of this could be the impact on consumer behaviors. If such a model would instead increase consumption by enabling easier access to find products. Assuming that this is true, then this could harm the environment where the model might contribute to over-consumption.

5.5.3 Biased Model

A transformer-based model's capabilities are limited to the information it is given during training and fine-tuning. This also applies to the values present in the texts, intentional or not. When constructing a model it is important to create one that does not offend or mistreat an ethnic, vulnerable, or LGBTQ+ group. There is a real risk of dire consequences if a book, containing intolerant material towards a certain group, would be labeled as *Course Literature*, i.e. regarded as the truth. Thus when creating these model, one need to take into consideration the contents of the data present in the data set used during training as biased data could very well lead to a biased model. The resulting model is not better than what it was fed.

Chapter 6
Conclusion

Product categorization is no easy task to automate and a time-consuming task to perform as a human due to the amount of data needed to be categorized. This thesis aimed to investigate how one could fine-tune BERT models to correctly classify products based on their descriptions. The approach in this work was to fine-tune BERT models with sequence classification tasks by using data collected from large e-commerce.

The research questions for this thesis are:

1. How well can a Machine Learning model predict the category to which a product belongs in different stores with different taxonomies?

2. Is the constructed model better than technology/models publicly available today?

3. Does the model predict categories as well on stores and taxonomies that were not present during training?

The results show that a pre-trained BERT model can be used and fine-tuned and produce higher accuracy than all of our baselines, however, the reliability of these baselines is questionable. We are well aware that these models lack a strong meaningful baseline to being able to conclude its performance. But the models have proved to be able to correctly classify products with an accuracy of 81.9% as an average across all stores that were crawled and therefore show great potential for further research. However it was not possible to get Liu et al. (2021) to work, this is unfortunate since this meant that no comparison could be made with other solutions. Lastly, the model proved to be able to correctly classify products from unseen stores but unfortunately with accuracy scores far too low to be used in real life.

With the data gathered, we have shown that the model can be used as an efficient tool since the user could choose between the top 10 predicted categories predicted for a given product description, but there is still much to be improved upon.

Future Work

If one were to replicate the models in this thesis, a new approach or refining approach to the baselines needs to be taken. To ensure that all transformer models perform better than naive ones and that it is necessary to use such a complex model.

This thesis has shown that it is possible to use BERT for sequence classification to categorize product text. In the future, it would be good to develop a policy and train the model using reinforcement training instead of BCE as a loss function which would enable the model to better classify products with unseen labels during training. This was the idea of the model in Liu et al. (2021). It is believed that something went wrong in policy when the code was modified for this research's needs and therefore another approach would be to find the error and get the model working. The reasoning behind the decision of writing another model was due to time constraints and the purpose of generating some sort of numerical statistics on the research topic.

The working model was developed during a lot shorter time and has a lot of future potentials. One could use dynamic false path generation instead of random generation to make sure that the whole taxonomy tree is explored and all possible incorrect paths are penalized. This would also help reduce train time since the entries to the model would become shorter over time. We also strongly believe that more testing on unseen data and generalization should be done since the unseen data set contained a lot of errors. One could investigate if fine-tuning based on zero-shot classification could yield a model better for generalization. We also encourage further development during fine-tuning such as learning rate scheduler.

References

Allen, J. F. (2003). *Natural Language Processing*, page 1218–1222. John Wiley and Sons Ltd., GBR.

Alpaydin, E. (2010). *Introduction to Machine Learning, Second Edition (Adaptive Computation and Machine Learning)*. Adaptive Computation and Machine Learning. The MIT Press, 2 edition.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015*. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

Brown, S. M. (2021). mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. *MIT Sloan School of Management*.

Central, G. S. (2023). Sitemaps. `https://developers.google.com/search/docs/advanced/sitemaps`. 2023-06-01.

Chinchor, N. and Sundheim, B. (1993). Muc-5 evaluation metrics. In *Proceedings of the 5th Conference on Message Understanding*, MUC5 '93, page 69–78, USA. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019a). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019b). Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269.

Galassi, A., Lippi, M., and Torroni, P. (2021). Attention in natural language processing. *IEEE Transactions on Neural Networks and Learning Systems, Neural Networks and Learning Systems, IEEE Transactions on, IEEE Trans. Neural Netw. Learning Syst*, 32(10):4291 – 4308.

Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing*. Morgan & Claypool Publishers.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA. `http://www.deeplearningbook.org`.

Gunnilstam, J. (2021). Här Är sveriges 100 största e-handlare. `www.ehandel.se/har-ar-sveriges-100-storsta-e-handlare`. 2023-05-30.

Hao, K. (2019). Training a single ai model can emit as much carbon as five cars in their lifetimes. *MIT Technology Review*.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hoare, C. A. R. (1961). Algorithm 64: Quicksort. *Commun. ACM*, 4(7):321.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. (2021). Lightmbert: A simple yet effective method for multilingual bert distillation.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson Education.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *J. Artif. Intell. Res.*, 4:237–285.

Koster, M. (2007). The web robots pages: The web robots exclusion protocol. *The Web Robots Pages*. 2023-06-01.

Lindberg, T. and Facht, U. (2022). *2.3 REKLAMMARKNADENS UTVECKLING*, volume 2022, page 11–12. Myndigheten för press, radio och tv.

Liu, H., Zhang, D., Yin, B., and Zhu, X. (2021). Improving pretrained models for zero-shot multi-label text classification through reinforced label hierarchy reasoning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1051–1062, Online. Association for Computational Linguistics.

Luong, M., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025.

Malmsten, M., Börjeson, L., and Haffenden, C. (2020). Playing with words at the national library of sweden – making a swedish bert.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). Introduction to information retrieval. *Cambridge University Press*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Mitchell, T. M. (1997). *Machine learning.* McGraw-Hill series in artificial intelligence. McGraw-Hill.

MonkeyLearn (2023). Natural language processing. `https://monkeylearn.com/natural-language-processing/`. Accessed on 2023-06-01.

Ng, A. (2023). Splitting into train, dev and test sets. `https://cs230.stanford.edu/blog/split/`. Accessed 1 June 2023.

Otter, D., Medina, J., and Kalita, J. (2021a). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems, Neural Networks and Learning Systems, IEEE Transactions on, IEEE Trans. Neural Netw. Learning Syst*, 32(2):604 – 624.

Otter, D., Medina, J., and Kalita, J. (2021b). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems, Neural Networks and Learning Systems, IEEE Transactions on, IEEE Trans. Neural Netw. Learning Syst*, 32(2):604 – 624.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8024–8035.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Petersen, F., Kuehne, H., Borgelt, C., and Deussen, O. (2022). Differentiable top-k classification learning. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S., editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 17656–17668. PMLR.

Scikit-learn (2023). 3.3. metrics and scoring: quantifying the quality of predictions. `https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score`. Accessed 7 June 2023.

Scrapy (2023). Scrapy documentation. `https://doc.scrapy.org/en/latest/intro/overview.html`. 2023-06-01.

Sitemaps.org (2020). Sitemaps.org. `https://www.sitemaps.org`. 2023-06-01.

Trim, C. (2013). The art of tokenization. *Developer Works*.

van Rijsbergen, C. J. (1979). *Information Retrieval*. Butterworth-Heinemann, 2nd edition.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Xian, Y., Lampert, C. H., Schiele, B., and Akata, Z. (2018). Zero-shot learning - a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2251–2265.

Appendices

Appendix A
Data set details

We chose to exclude Biltema, Apotea, and Clas Ohlsson from the data set since they had incorrect category paths that resulted in the model not being able to train. Two of these stores were later used as unseen data tests instead.

Store	Title	Brand	Description	Description plus	Category	SKU	GTIN	MPN	EAN
Akademibokhandlen	x	x	x	x	x	x			
Apotea	x	x	x	x	x		x		
Bauhaus	x	x	x	x	x	x	x		
Biltema	x	x	x		x			x	
Boozt	x	x	x	x	x	x	x		
Bygghemma	x	x	x	x	x			x	
Cdon	x	x	x	x	x	x	x		
Cervera	x	x	x	x	x	x		x	
Chili	x	x	x	x	x	x			
Clas Ohlson	x		x	x	x	x			
Elgiganten	x	x	x	x	x	x	x		
Gymgrossisten	x		x	x	x	x		x	x
H&M	x	x	x	x	x	x			
Ikea	x	x	x	x	x	x			
Kjell	x	x	x	x	x	x	x		
Lyko	x	x	x		x	x	x		
Mediamarkt	x	x	x	x	x	x			x
NA-KD	x	x	x		x	x			
Plantagen	x	x	x	x	x	x	x		
Rusta	x		x	x	x	x			
Systemet	x	x	x	x	x	x			
PSC1	x	x	x						
PSC2	x	x	x	x	x	x			

Tab. A.1 displays a table listing the number of products from each store used to create the data sets.

Table A.1: Each store and the number of products in each dataset.

Store	Number of products
Akademibokhandeln	103258
Apotea	43940
Bauhaus	55891
Biltema	29325
Boozt	147910
Bygghemma	346948
Cdon	6733696
Cervera	16333
Chili	146895
Clas Ohlson	45129
Elgiganten	314071
Gymgrossisten	2482
H&M	26492
Kjell	9156
Ikea	31903
Lyko	68094
Mediamarkt	14744
NA-KD	32215
Plantagen	7777
Rusta	6408
Systemet	23306
PCS1	1602846
PSC2	3709534

Appendix B

Hyper-parameter optimization

B.1 Hyper-parameter optimization 1

In the first hyper-parameter optimization study these combinations were made.

Table B.1: Caption

Iterations	Learning rate	NNE	NPD
1	1e-06	1	1
2	1e-06	1	2
3	1e-06	1	3
4	1e-06	2	1
5	1e-06	2	2
6	1e-06	2	3
7	1e-06	3	1
8	1e-06	3	2
9	1e-06	3	3
10	1e-04	2	2
11	1e-05	2	2
12	1e-06	2	2

B.2 Hyper-parameter optimization 2

In the second hyper-parameter optimization study these combinations were made.

Table B.2: Caption

Iterations	Learning rate	NNE	NPD
1	1e-05	1	1
2	1e-05	1	2
3	1e-05	1	3
4	1e-05	2	1
5	1e-05	2	2
6	1e-05	2	3
7	1e-05	3	1
8	1e-05	3	2
9	1e-05	3	3

Appendix C

Training parameters

The parameters that were given to the model during the final training are presented in the table below.

Parameter name	Value
learning_rate	0.00001
gradient_accumulation_steps	1
save_steps	50000
logging_steps	10000
batch_size	12
max_length	512
num_train_epochs	1
number_of_negative_examples	1 & 2
negative_example_depth	2
pre-trained model_name	Addedk/kbbert-distilled-cased

The models that were trained on the machine learning computer with the GTX 4090 used the following parameters and settings:

Parameter name	Value
learning_rate:	0.00001
gradient_accumulation_steps:	1
save_steps:	50000
logging_steps:	10000
batch_size:	56
max_length:	512
num_train_epochs:	1
number_of_negative_examples:	2
negative_example_depth:	2
pre-trained model_name:	Addedk/kbbert-distilled-cased

Parameter name	Value
learning_rate:	0.00001
gradient_accumulation_steps:	1
save_steps:	50000
logging_steps:	10000
batch_size:	30
max_length:	512
num_train_epochs:	1
number_of_negative_examples:	1
negative_example_depth:	2
pre-trained model_name:	KB/bert-base-swedish-cased

EXAMENSARBETE Automated product categorization using transformer models
STUDENTER Joel Bäcker, Victor Winkelmann
HANDLEDARE Marcus Klang (LTH), Rasmus Ros (Theca Systems)
EXAMINATOR Volker Krueger (LTH)

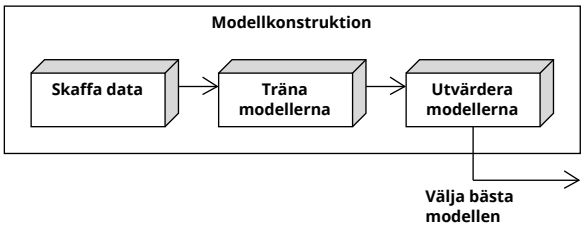
Automatiskt produktkategorisering genom transformermodeller

POPULÄRVETENSKAPLIG SAMMANFATTNING **Joel Bäcker, Victor Winkelmann**

Produktkategorisering är en viktigt del av flera industrier, bland annat e-handels företag. Att effektivt organisera produkter ger enklare navigering för en användare samt underlättar för företag att analysera kundmönster. Detta arbete har tagit fram metoder för att automatisk kategorisera produkter.

Idag läggs mer och mer pengar för att synas tidigt bland resultaten från en sökmotor för att öka trafiken till en hemsida. Då summorna för att synas ökat väldigt har det medfört att företag intresserar sig allt mer för att istället skicka relevant data mellan varandra direkt istället för en mellanhand så som en sökmotor. En del av att möjliggöra trafiken mellan företag är att implementera automatisk produktkategoriering som dessutom kan kategoriera produkter till en annan butiks kategorier. Att kategorisera produkter görs genom att träna artificiell intelligens (AI) modeller, så kallade transformers. Transformers modeller har fördelen att de kan tränas på stora mängder text av någon annan för att sedan tränas en andra gång och då specialiserar sig på en specifik uppgift. Den första träningen görs vanligtvis på mycket stora mängder text.

I examensarbetet har olika modeller utvecklats och utvärderas som ska med så hög pricksäkerhet förutspå vilka kategorier som en produkt tillhör. Det första som gjordes var att skaffa data från flera butiker, som ska användas för att modellen ska kunna tränas på produktinformation för att specialisera sig på kategorisering. Datan som införskaffades innehöll titel, beskrivning, kategori med mera. När datan hade insamlats kunde konstruktion och träning av modellerna göras som i sin tur följdes av modellernas prestanda utvärderades. Utvärdering gjordes genom att jämföra hur bra olika modeller var, de två typer av modeller som fanns var ett par simpla modeller och några transformer modeller.



Resultatet från modellerna visar att den bästa modellen gissade rätt ungefär 82% av alla gånger, jämfört med de simpla modellerna som hade som bäst rätt 14% av gångerna. Detta visar på att problemet med att automatiskt kategorisera produkter inte kan lösas med en simpel modeller utan att det faktiskt behövs tillämpas mer komplexa modell som transformers.