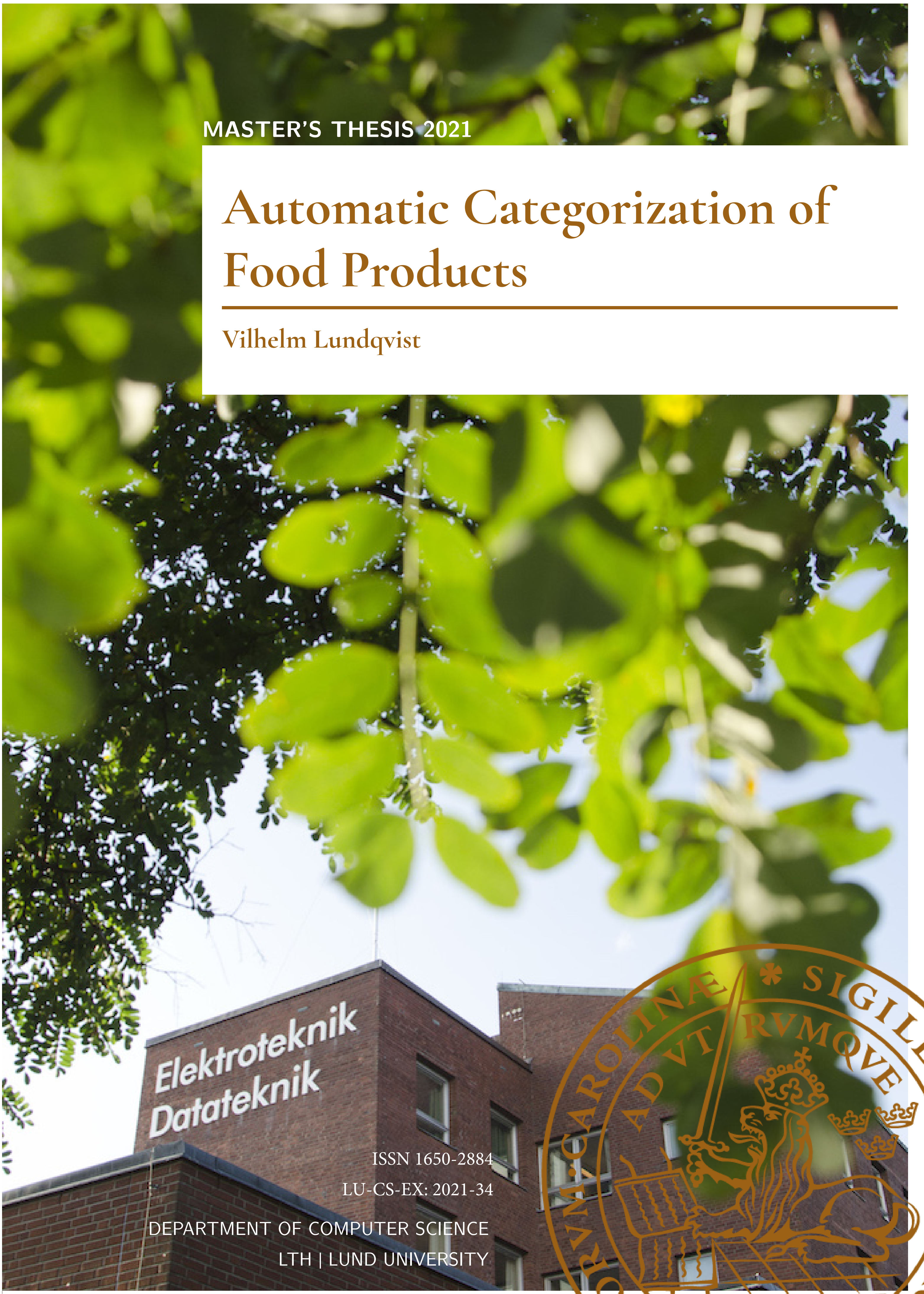
MASTER'S THESIS 2021

Automatic Categorization of Food Products

Vilhelm Lundqvist



ISSN 1650-2884
LU-CS-EX: 2021-34

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2021-34

Automatic Categorization of Food Products

Automatisk Kategorisering av Livsmedel

Vilhelm Lundqvist

Automatic Categorization of Food Products

Vilhelm Lundqvist
`dat13vlu@student.lth.se`

June 10, 2021

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Pierre Nugues, `Pierre.Nugues@cs.lth.se`
Johnny Zackrisson, `johnny.zackrisson@matildafoodtech.com`

Examiner: Jacek Malec, `Jacek.Malec@cs.lth.se`

Abstract

As a part of its daily operations, the company *Mashie* receives datasheets on food and food-related products from suppliers. These datasheets describe various details about their products, such as product names, ID's, packaging formats, weight and more. This data is often semi-structured and the structure and terminology used may vary between individual manufacturers. Additionally, important details are sometimes embedded in other fields, such as a product's weight being described in its name. *Mashie* works to process this data, structuring and storing it as annotated objects in a database.

While routine and well-mastered, this process is currently done by hand and very labor-intensive. *Mashie* thus seeks to explore if parts of this process can be automated. In this thesis, I explore if one of the tasks, product categorization, can be viably performed using only product names, an easily extractible feature.

To build the categorizer, I used the *BERT* transformer models. The pretrained models *mBERT* (multilingual) and *KB-BERT* (Swedish), were fine-tuned on this text classification task, obtaining macro-average F1-scores of 96.76% and 94.73%, respectively. *KB-BERT* mostly suffers from systemic misclassification among a few taxonomies. Overall the results indicate that modern pretrained language models such as *BERT* are seemingly well suited to the task, and thus shows promise for use on similarly unnatural text sources.

Keywords: MSc, BERT, Text classification, Food products, NLP, Machine learning, categorization

Acknowledgements

I would like to thank Pierre Nugues for his excellent guidance and vision in both technical and planning matters, Johnny Zackrisson for proving great support during the thesis and providing many interesting ideas and perspectives, Mashie for providing such an interesting project, my friends for all of their refreshing dank memes as well as my family and loved ones for supporting me throughout my degree and my life.

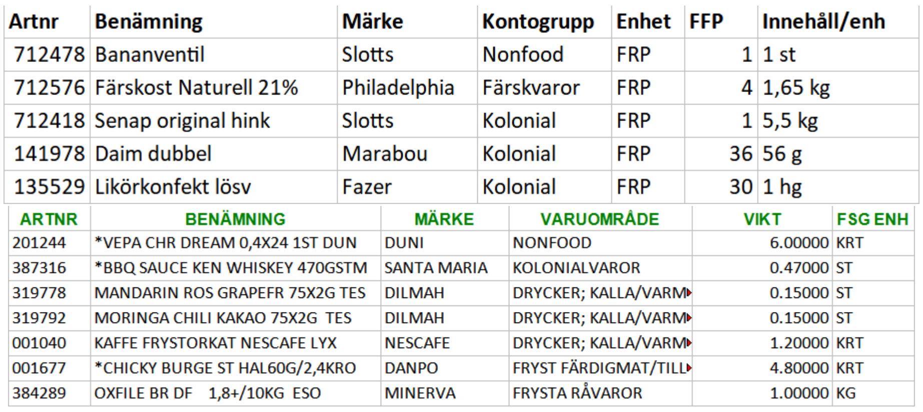
Contents

Chapter 1

Introduction

1.1 Overview and Background

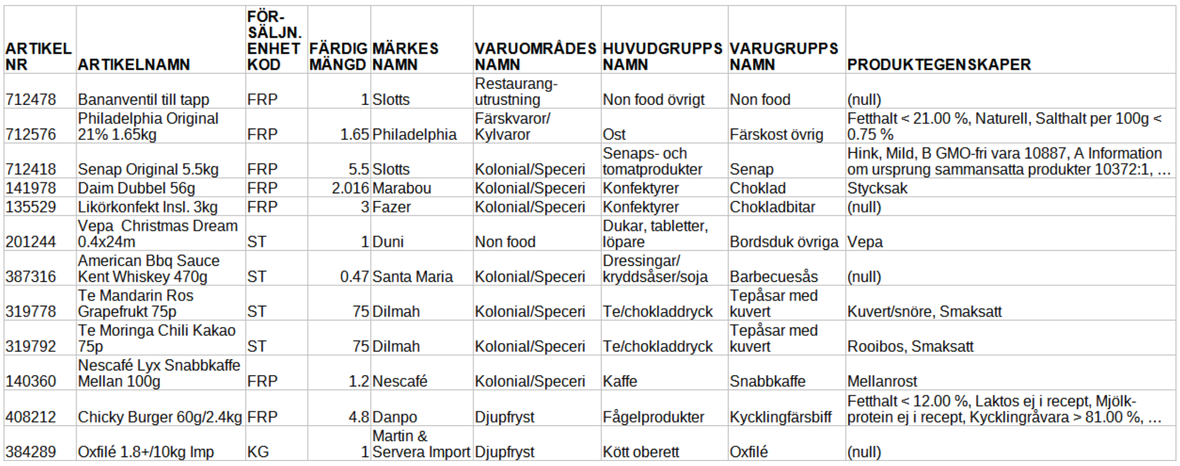
As a part of its daily operations, the *Mashie* company receives datasheets from manufacturers containing details about food and food-related products that are typically sold in grocery stores. *Mashie* works to categorize, annotate, and describe these products and their details, storing the data in a structured format. Currently, this is done manually by industry experts. These datasheets contain a mix of information understandable by laymen such as product names or packaging formats, but also more complex tags, like abbreviations of production details or certifications. These datasheets thus contain a lot of semi-structured information, but the structuring of the data itself varies widely, even between individual datasheets from the same manufacturer.



Artnr	Benämning	Märke	Kontogrupp	Enhet	FFP	Innehåll/enh
712478	Bananventil	Slotts	Nonfood	FRP	1	1 st
712576	Färskost Naturell 21%	Philadelphia	Färskvaror	FRP	4	1,65 kg
712418	Senap original hink	Slotts	Kolonial	FRP	1	5,5 kg
141978	Daim dubbel	Marabou	Kolonial	FRP	36	56 g
135529	Likörkonfekt lösv	Fazer	Kolonial	FRP	30	1 hg

ARTNR	BENÄMNING	MÄRKE	VARUOMRÅDE	VIKT	FSG ENH
201244	*VEPA CHR DREAM 0,4X24 1ST DUN	DUNI	NONFOOD	6.00000	KRT
387316	*BBQ SAUCE KEN WHISKEY 470GSTM	SANTA MARIA	KOLONIALVAROR	0.47000	ST
319778	MANDARIN ROS GRAPEFR 75X2G TES	DILMAH	DRYCKER; KALLA/VARM	0.15000	ST
319792	MORINGA CHILI KAKAO 75X2G TES	DILMAH	DRYCKER; KALLA/VARM	0.15000	ST
001040	KAFFE FRYSTORKAT NESCAFE LYX	NESCAFE	DRYCKER; KALLA/VARM	1.20000	KRT
001677	*CHICKY BURGE ST HAL60G/2,4KRO	DANPO	FRYST FÄRDIGMAT/TILL	4.80000	KRT
384289	OXFILE BR DF 1,8+/10KG ESO	MINERVA	FRYSTA RÅVAROR	1.00000	KG

Figure 1.1: Excerpts from two product sheets from different manufacturers. They have different structure and labeling, and some important information is embedded in the product names in the second picture.



ARTIKEL NR	ARTIKELNAMN	FÖR-SÄLJN. ENHET KOD	FÄRDIG MÄNGD	MÄRKES NAMN	VARUOMRÅDES NAMN	HUVUDGRUPPS NAMN	VARUGRUPPS NAMN	PRODUKTEGENSKAPER
712478	Bananventil till tapp	FRP	1	Slotts	Restaurang-utrustning	Non food övrigt	Non food	(null)
712576	Philadelphia Original 21% 1.65kg	FRP	1.65	Philadelphia	Färskvaror/ Kylvaror	Ost	Färskost övrig	Fetthalt < 21.00 %, Naturell, Salthalt per 100g < 0.75 %
712418	Senap Original 5.5kg	FRP	5.5	Slotts	Kolonial/Speceri	Senaps- och tomatprodukter	Senap	Hink, Mild, B GMO-fri vara 10887, A Information om ursprung sammansatta produkter 10372:1, …
141978	Daim Dubbel 56g	FRP	2.016	Marabou	Kolonial/Speceri	Konfektyrer	Choklad	Stycksak
135529	Likörkonfekt Insl. 3kg	FRP	3	Fazer	Kolonial/Speceri	Konfektyrer	Chokladbitar	(null)
201244	Vepa Christmas Dream 0.4x24m	ST	1	Duni	Non food	Dukar, tabletter, löpare	Bordsduk övriga	Vepa
387316	American Bbq Sauce Kent Whiskey 470g	ST	0.47	Santa Maria	Kolonial/Speceri	Dressingar/ kryddsåser/soja	Barbecuesås	(null)
319778	Te Mandarin Ros Grapefrukt 75p	ST	75	Dilmah	Kolonial/Speceri	Te/chokladdryck	Tepåsar med kuvert	Kuvert/snöre, Smaksatt
319792	Te Moringa Chili Kakao 75p	ST	75	Dilmah	Kolonial/Speceri	Te/chokladdryck	Tepåsar med kuvert	Rooibos, Smaksatt
140360	Nescafé Lyx Snabbkaffe Mellan 100g	FRP	1.2	Nescafé	Kolonial/Speceri	Kaffe	Snabbkaffe	Mellanrost
408212	Chicky Burger 60g/2.4kg	FRP	4.8	Danpo	Djupfryst	Fågelprodukter	Kycklingfärsbiff	Fetthalt < 12.00 %, Laktos ej i recept, Mjölk-protein ej i recept, Kycklingråvara > 81.00 %, …
384289	Oxfilé 1.8+/10kg Imp	KG	1	Martin & Servera Import	Djupfryst	Kött oberett	Oxfilé	(null)

Figure 1.2: The same products as shown in Figure 1.1 but with excerpts of the standardized structure used by Mashie's databases, viewed in an Excel file.

However, this is very labor-intensive work, and *Mashie* thus seeks to explore if parts of this process can be automated, reducing the workload and improving throughput. Due to the volatile structure of the data, it is difficult to apply and maintain rule-based methods for information extraction.

Parts of *Mashies* structurization is shown in **Figure 1.2**, including their product categorization scheme with the fields *varuområdesnamn*, *huvudgruppsnamn* and *varugruppsnamn*. In this thesis, I explore if one of the tasks, product categorization, can be viably performed with the modern NLP model *BERT* (Devlin et al., 2019) using only product names, an easily extractible feature. This *BERT* has been proven to produce good results for text classification on lengthier, natural and complete sentences and text, such as analyzing the sentiment of a comment (Devlin et al., 2019), or classifying knowledge-dense texts in the BioMed domain (Baker et al., 2015). It has not, however, been used for classification on highly structured and short texts, such as product names.

Using a reduced set of 40 product categories, I fine-tuned two pretrained *BERT* models, multilingual *mBERT* (Google, 2018) and Swedish *KB-BERT* (Malmsten et al., 2020) on the dataset of over 23,000 products. The models achieve very good performance for the given task, obtaining macro-average F1-scores of 96.76% and 94.73%, respectively. Both models achieved F1-scores over 95% in a majority of the categories.

Contrary to expectations (de Vargas Feijo and Moreira, 2020), the multilingual *mBERT* performed better overall and has less variance in F1-score across the various product categories. *KB-BERT* performed only slightly worse (~1% unit lower F1-score) for most categories, but particularly suffered from systemic misclassification among select categories. This lowered the average performance of the the latter model substantially.

The required training support in order to achieve high classification performance was found to vary enormously across categories; some required as little as 4 training examples of a category in order for the model to achieve F1-scores above 90%, whilst others required ~1000 training examples.

The results indicate that modern pretrained language models such as *BERT* are well suited to categorizing food products solely based on their names. However, selection of which pretrained models to use still has an impact as model behavior will vary. In general, this thesis shows that *BERT* is suitable for use on more esoteric text data, rather than just complete sentences and texts.

1.2 Outline

Chapter 1 provides an introduction to the thesis as well as motivation and a general overview of it. This thesis involves terminology and techniques from *natural language processing*, a sub-field of *machine learning* and utilizes a model called *BERT* (Devlin et al., 2019) that recently revolutionized how many *natural language processing* tasks are performed. Required background knowledge on these topics are described in **Chapter 2**. **Chapter 3** elaborates on the thesis methodology; the structure of the data given by *Mashie*, how it was processed, how the categories were selected, along with details on the how the two *BERT* models were trained on this data. **Chapter 4** discloses the results of the trained models and an investigation into their behaviour and capabilities. These results and their details are then discussed in **Chapter 5**, along with suggestions for future experiments and improvements.

1.3 Related Work

While there are papers on food-related NLP tasks, such as *FoodIE* (Popovski et al., 2019) and Kalra et al. (2020) which both perform named-entity recognition to extract food ingredients and products from text, I found only one other paper that has performed a categorization of foods.

Food Ingredient Classification

Wiegand et al. (2014) presented a method to automatically categorize food items such as *raspberry*, *bananas*, *almonds* into a predefined ontology of 11 categories. Using a corpora of recipes, they generated a weighted relational graph of food items by identifying relations and similarity using semantic rules such as:

*You may use **food1** $\underline{\text{instead of}}$ **food2**,*

from which the nodes *food1* and *food2* are extracted, connected and then inserted to the graph, increasing the connection weight if the nodes and connection was already present in the graph.

Using this graph as a measure of similarity between food items, they employed semi-supervised clustering algorithms to classify products into the predefined categories. While our goal is similar, the input data is very different. Furthermore, their method does not apply powerful modern feature transformations such as *word embeddings*. This would enable their system to capture important semantic and relational information about the input ingredients, which in their paper is constructed in a restrictive rule-based manner.

Exam Questions Classification

Outside of the realm of food, Abduljabbar and Omar (2015) performed categorization of short exam questions. The task is vaguely similar to this thesis, as the exam questions in their dataset were short and doesn't always "flow" naturally in contrast to normal language, like an article. The questions relation to their categories is also somewhat elusive; deeper language and word perception as well thinking is required for humans to categorize them, in contrast to just flagging for specific words.

The dataset consisted of questions like *Define Inheritance concept.* and *Write a JAVA program to show the Overloading concept..* Such exam questions were then classified into a set of categories: *Knowledge*, *Comprehension*, *Application*, *Analysis*, *Synthesis* and *Evaluation*.

The approach to transform their data into usable input for their classifiers was to ascribe a set of keywords to each category; the question category *Knowledge* was allotted keywords such as *arrange, recognize, relate, label, list, memorize, recall, define* and so forth. Exam questions were transformed into input features based on their usage of these keywords and this input data was further preprocessed using the methods *Mutual Information*, *Chi-square statistic* and *Odd Ratio* to construct newer features to allow for better distinction between categories.

Classifiers based on *SVM*, *Naive-Bayes* and *k-Nearest Neighbors* were then fitted on their data, achieving 78.19%, 74.95% and 80.82% macro-averaged F1-scores respectively. While their task is indeed similar, such an approach is not feasible for our input data save for a few categories and, similarly to Wiegand et al. (2014), does not use of modern *Transformer* models.

Text Classification with BioMed-trained Transformer models

Lewis et al. (2020) review several *Transformers* models pretrained on BioMed corpora by fine-tuning them on several down-stream tasks. Among these tasks is the *HOC* task (Baker et al., 2015), which involves detecting if BioMed abstracts contains text describing hallmark behavior of cancer cells and classifying the abstract according to that specific cancer cell behavior.

Among several models, they found the *BioMed-RoBERTa* model (Gururangan et al., 2020) to perform the best, achieving an 86.7% macro-averaged F1-score. This task is similar in the sense that both the text and the categories belong to quite a narrow domain while utilizing *Transformer* models. However the models in question were pretrained at least in part on corpora specific to their BioMed domain, which differs from the models used in this thesis.

Comparison of Monolingual and Multingual BERT models

As this thesis evaluates the multilingual model *mBERT* and the monolingual model *KB-BERT*, the work of de Vargas Feijo and Moreira (2020) is of relevance. They measured and compared the performance of *mBERT*; the *BERT* model trained on a multilingual corpus (of which details are shown in **Section 2.5.7**), with two monolingual models introduced in their paper: *BertPT* and *AlbertPT* (de Vargas Feijo and Moreira, 2020). Both models were pretrained on a monolingual Portuguese corpus.

For clarification, *AlbertPT* stems from the *ALBERT* model (Lan et al., 2019) which is based on *BERT*, contrary to *BertPT* which is based on regular *BERT*. These models were fine-tuned on several NLP tasks.

Their paper indicates that all models perform quite well on many tasks, but which model is best differs on a task-to-task basis. The text classification task most similar to this thesis, *Offensive Comment Identification* (de Pelle and Moreira, 2017), was found to be handled best by the monolingual *BertPT*, although not by a wide margin.

1.4 Contributions

All parts of the thesis were performed by Vilhelm Lundqvist.

Chapter 2
Theoretical Background

Throughout this chapter, I will explain concepts and terminology required to understand the contents and methodology employed in the thesis. The two models *mBERT* and *KB-BERT* use the *BERT* architecture, but in order to understand how it works there are a couple of underlying concepts that must be explained first. This chapter is structured to serially introduce these topics in order of complexity and dependence, so the reader can be build a stack of understanding that allows them to grasp why and how *BERT* works.

Starting out, I will explain the basics of what *machine learning* is, introduce how it is applied to human language with *natural language processing* and explain how *neural networks* conceptually uses stacks of machine learning to facilitate more powerful models. Following that, I will describe the neural-network based constructions called *embeddings* and *encoders* along with their purpose. Finally, I will explain the architecture and methodology of *BERT*.

A section on terminology is appended to the end of this chapter. There I explain certain words that are used in this chapter but not defined. This is done to avoid cluttering and maintain continuity. It also includes some concepts and words that are important for the thesis, but do not require entire sections of their own.

2.1 Machine Learning

Machine learning can broadly be described as utilizing computers to solve certain tasks that are not easily solved using explicit rules, algorithms or equations, by approximating solutions in an automated, iterative fashion. One of the simplest examples of this is fitting a straight line to a large number of data points. This problem can be solved without machine learning, but it provides quite a simple demonstration of the iterative methodology of machine learning.

2.1.1 Linear Regression Example

A linear $y = ax + b$ line is created with random parameters, which we can see in **Figure 2.1** along with the data it is supposed to approximate. We can see it is not doing a very good job and we can in fact measure how poorly it is doing, by calculating the sum of all errors. The vertical distance from a data point to the line is calculated, which represents its "error from the line". All these errors are then squared and summed together, giving us this sum of all errors; this value is called the *loss* value in machine learning terminology.

This *loss* is thus a measure of how poorly the line approximates the data. If the line describes the data perfectly by somehow passing through each data point, the *loss* will be 0. The worse the line is at approximating the data, the higher the *loss* will be.

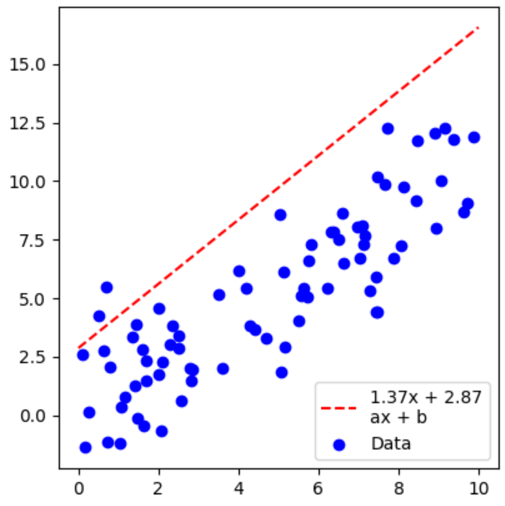


Figure 2.1: The initial state prior to linear regression, with a random line making a poor attempt at approximating the data. The parameters of the line $ax = b$ are $a = 1.37$ and $b = 2.87$.

If we change the line by varying the line parameters a and b, the *loss* value (error sum) changes. If we try to generalize the *loss* by calculating it with *variables* representing a and b, instead of actual numbers, it instead becomes a *function* of a and b that we call the *loss function*. The *loss function* can then be used to calculate what the *loss* will be for a particular combination of a and b. In fact, we calculated what the *loss* will be for lots of these possible pairs of a and b and displayed it as a 3D-graph in **Figure 2.2**.

In this plot of the *loss function*, the reader can see how the *loss* changes when the line's slope (a) or elevation (b) is changed.

Looking at **Figure 2.1**, we can see there is a minimum at $a = 1$ and $b = 0$. However, finding the minimum this way by looking at a complete illustration of the *loss function* is unfeasibly expensive for almost all real-life scenarios, and furthermore impossible to visualize when you have more than two parameters (N parameters leads to an $N + 1$-dimensional graph).

However, it is still cheap to get the *loss* and the slope of the *loss function* for a single point. Using these two facts, we can slowly step down the steepest slope of the *loss function*, as if we were on a mountainside in a blizzard, tapping around with our foot to find the downward slope, taking a step in that direction and repeating the process over and over until we reach the bottom.

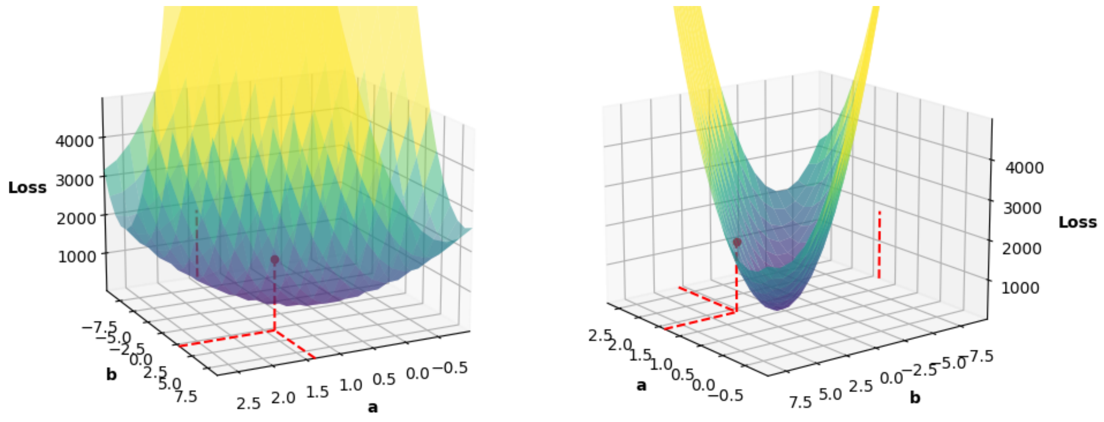


Figure 2.2: The *loss function* from the line and the data given in Figure 2.1, given in two angles. The line's initial parameters $a = 1.37$ and $b = 2.87$ are highlighted in red, along with its corresponding *loss* value. Looking at this graph you can see that if you increased b, the *loss* would increase. This stems from the fact that b corresponds to raising the line vertically in Figure 2.1, which in turn worsens the approximation by increasing the sum of all errors – the *loss* value. a similarly affects the *loss*; by manipulating a you affect the slope of the line, affecting its approximation of the data.

To check the slope, we differentiate this *loss function* based on the line parameters a and b and input the lines' current values $a = 1.37$ and $b = 2.87$ to get the slope of the *loss function* at that point.

Finally, we take this slope, multiply it with a small number called the *learning rate* and use the result to update a and b. In doing this, we've taken a small step towards the minimum of this loss function; meaning we're moving towards the specific combination of a and b that gives the lowest possible *loss* value/error sum. By repeating this step, or iteration, we slowly converge to the minimum. The minimum of this particular *loss function* is at $a = 1, b = 0$.

To recap, the minimum of *loss function* being at $a = 1, b = 0$ means that, of all the possible combinations of a and b, the line that best describes the given data (by having the smallest possible *loss*) is the line $y = 1 \cdot x + 0$. This line is shown in **Figure 2.3**, which shows that it indeed approximates the pattern of the data well.

2.1.2 Artificial Intelligence vs. Machine Learning

This process of moving towards the minimum of the *loss function* is known as *gradient descent* and it is the foundational building block of many machine learning algorithms; almost equivalent to being the definition of what makes an algorithm a *machine-learning* algorithm.

While the general field of *artificial intelligence* also involves computers making decisions based on some numerical input, *artificial intelligence* algorithms are usually constructed using predefined rules. *Machine learning* inherently differs from this in the sense that, while designers of machine learning algorithms and models specify how the data is generally processed, what type of operations are performed on it and in what order; the parameters of these operations are left unspecified and are "learned" through iteratively updating them using a *loss*

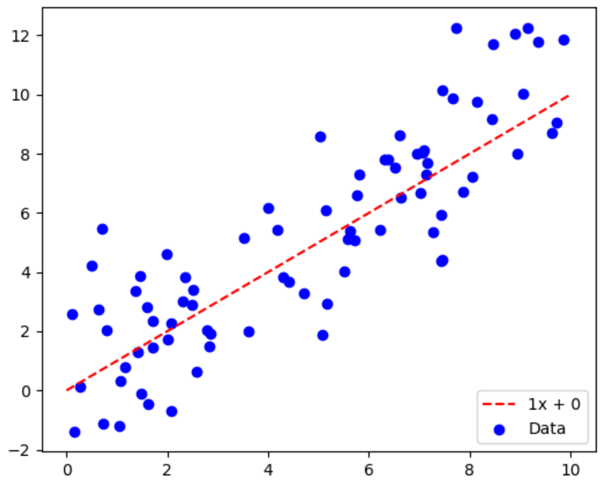


Figure 2.3: An example of finished linear regression.

function, with the goal of finding the optimum set of parameters.

2.1.3 Further Capabilities of Machine Learning

This idea of fitting a model to data through iteration, often using a *loss function*, is generally what defines the field of machine learning. There are many types of tasks and categories of tasks in machine learning, both supervised (where the data has been annotated with an answer) and unsupervised learning (where a model tries to perform a task on pure unlabeled data).

An example of a supervised learning task is *classification*; e.g. a model trying to classify if a patient has a tumor or not using X-Ray images as input data, where each image in the training dataset has an associated correct answer. A typical task in unsupervised learning is *clustering*, which could be trying to separate friend groups on Facebook, by trying to create clusters of people that have all friended each other with minimal connections to other friend groups.

2.1.4 Overfitting and Underfitting

Throughout the iterations of the training process, the machine algorithm is trying to select model parameters that minimize the *loss function* as much as possible. Depending on the model, data and other factors, this can cause unintended consequences – by optimizing on the *loss function* too much, the model may end up fitting the training data *too well* and not work generally for future examples. This phenomenon is called overfitting.

Conversely, the programmer may have set the *learning rate* too low, causing the training/*gradient descent* process to never actually reach the minimum of the *loss function*. This phenomenon is analogously called *underfitting*. There are many techniques to combat these problems, some of which utilize splitting the dataset into three parts:

Training set the data the model is actually trained on,

Evaluation set on which the model is tested against during training, to make sure the model is actually moving in the right direction and not overfitting,

Test set which is used to evaluate the final model after the training process.

As the *evaluation* and *test* sets contain data that is not present in the *training* set, they provide sober indicators of how the model is performing. With these datasets, there are many techniques to detect overfitting, for example by checking if the *evaluation loss* is steadily increasing whilst *training loss* is decreasing.

It is also possible to do a *hyperparameter grid search*, where different combinations of model and training settings such as the *learning rate* (called *hyperparameters* in machine learning) are used to train models. These parameters are subsequently tested against the *evaluation dataset*. The model with the lowest *evaluation loss* is then selected and is finally measured against the *test set*, avoiding any possible biases the *evaluation set* might have that would increase the score of a particular setting of *hyperparameters*.

2.2 Natural Language Processing

Natural Language Processing, called *NLP* for short, is a branch of machine learning that uses machine learning tools to perform tasks on human language such as text or other data sources like speech or images of text.

Some widely known examples of NLP tasks include language translation of text, speech recognition and optical character recognition – parsing an image of text into text strings usable in a computer. While many techniques and methods from ordinary machine learning can and are used for NLP on text directly, text almost always requires preprocessing to turn it some form of numerical data, as machine learning algorithms can only run and be trained on numbers.

There are many ways to process text into numerical data, including techniques like *TF/IDF vectors*, *word embeddings* and *bag-of-words vectors*. To illustrate a simple example of how this processing procedure relates to machine learning algorithms, I will present an example of the machine learning task *classification* with the algorithm *logistic regression*, using the *bag-of-words* processing technique.

2.2.1 Text Classification Example with Bag-of-Words

Dataset Description

In this example of the *supervised learning* task *text classification*, we are trying to classify movie reviews as either *positive* or *negative*. We have a small dataset of six movie reviews shown in **Figure 2.1**, where each data point is *labeled*, meaning it has an annotated answer. There are three positive movie reviews which are labeled as having the sentiment 1. The other three reviews are negative, represented with a sentiment of 0.

The Bag-of-Words Vector

We're going to use logistic regression (explained later on) to classify these reviews, but as that method requires numbers as input, we will need to convert the text into numbers. For this example, we're going to be representing the text as a numerical vector, using a *bag-of-words* vector.

Comment	**Sentiment**
Good movie. Didn't suck.	1
Amazing movie, I thoroughly enjoyed it.	1
Really good. Very entertaining, enjoyed it alot.	1
Fell asleep halfway, boring movie.	0
This movie sucks, really boring. Don't watch it.	0
Sucks.	0

Table 2.1: Some hypothetical reviews of a movie. Sentiment of 1 indicates a positive review, sentiment 0 indicates a negative review.

A *bag-of-words (BOW)* vector involves collecting all the N unique words used in the dataset, assigning each word w its own index number i_w. A sentence is then represented with a vector of N bits, where the i:th bit is 1 if the word w is present in the sentence.

An example of this for the first review is shown in **Table 2.2**. An important process to note is how these words were converted into numbers by giving each possible word a unique index, which is called *One-Hot-Encoding*.

Word w	**Index** i	***BOW* vector**
amazing	0	0
alot	1	0
asleep	2	0
boring	3	0
don't	4	0
didn't	5	**1**
enjoyed	6	0
entertaining	7	0
fell	8	0
good	9	**1**
halfway	10	0
i	11	0
it	12	0
movie	13	**1**
really	14	0
suck	15	**1**
thoroughly	16	0
this	17	0
very	18	0
watch	19	0

Table 2.2: A *bag-of-words* vector for the sentence *Good Movie. didn't suck*. Note the 1's at indices 5, 9, 13 and 15.

Using the BOW Vector for Logistic Regression

Using this representation, we now have a fully numerical representation of our text data, shown in **Table 2.3**, which can be fed to a machine learning algorithm.

A suitable machine learning algorithm for this case is *Logistic Regression*, which involves multiplying each of these bits in a *BOW* vector with a corresponding weight w_i, calculating the sum of these terms and finally feeding it into what is known as an *activation* function. For *logistic regression*, this is the *logistic* function:

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

also known as the *sigmoid* function.

This function has the important properties that $\sigma(0) = 0.5$, $\sigma(\infty) = 1$, $\sigma(-\infty) = 0$, along with being differentiable.

If we can make the weighted sum of the *BOW*-vector be *larger* than 0 for all *positive* reviews and inversely have that weighted sum be *less* than 0 for all *BOW*-vectors corresponding to *negative* reviews, we will have a model that can classify reviews.

***BOW* vectors**																				**Sentiment**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	
0	0	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**	0	**1**	0	0	0	0	1
1	0	0	0	0	0	**1**	0	0	0	0	**1**	**1**	**1**	0	0	**1**	0	0	0	1
0	**1**	0	0	0	0	**1**	**1**	0	**1**	0	0	**1**	0	**1**	0	0	0	**1**	0	1
0	0	**1**	**1**	0	0	0	0	**1**	0	**1**	0	0	**1**	0	0	0	0	0	0	0
0	0	0	**1**	**1**	0	0	0	0	0	0	0	**1**	**1**	**1**	**1**	0	**1**	0	**1**	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0

Table 2.3: Each review in Table 2.1, converted to *BOW* vectors using the same indexing as in Table 2.2.

Analogously to the linear regression example shown in **Section 2.1.1**, we can create a *loss function* that compares the *logistic function*'s output with the sentiment that review was associated with.

Since the *logistic function* is differentiable, the *loss function* in turn is differentiable. We can thus differentiate the *loss function* relative to the weights w_i and perform *gradient descent*. For brevity, we will not describe the *gradient descent* process here, but instead show how such a model could work with manually selected weights, presented in **Table 2.4**.

2.2.2 Multi-Class Classification

This thesis involves solving the task of *multi-class classification* which is very similar to the task of *classification* shown above.

The difference is that instead of just having 2 classes, as in the above example with *positive* or *negative*, there are *several* classes that a piece of data can belong to. Instead of representing the assigned class as a single value *sentiment* as shown above, N bits representing the N possible classes are used and the bit corresponding to a data point's class is set to 1 and all others 0.

									Weights													
2	0	0	**-3**	0	0	**2**	**2**	0	**4**	0	0	0	0	0	**-2**	0	0	0	0			
									***BOW* vectors**											**W.S**	**$\sigma(W.S)$**	**Sentim.**
0	0	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**	0	**1**	0	0	0	0	2	0.8808	1
1	0	0	0	0	0	**1**	0	0	0	0	**1**	**1**	**1**	0	0	**1**	0	0	0	4	0.9820	1
0	**1**	0	0	0	0	**1**	**1**	0	**1**	0	0	**1**	0	**1**	0	0	0	**1**	0	8	0.9997	1
0	0	**1**	**1**	0	0	0	0	**1**	0	**1**	0	0	**1**	0	0	0	0	0	0	-3	0.0474	0
0	0	0	**1**	**1**	0	0	0	0	0	0	0	**1**	**1**	**1**	**1**	0	**1**	0	**1**	-5	0.0067	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	-2	0.1192	0

Table 2.4: The *BOW* vectors of each review, along with a logistic regression model that successfully classifies each comment. The model is represented by the weights vector shown at the top. The weights vector is multiplied element-wise with each *BOW* vector to produced the weighted sum shown in the column **W.S**. Each weighted sum is sent as input to the *activation function*, which for *logistic regression* is the *logistic/sigmoid* function. The output, known as the *activation*, is shown in the column $\sigma(W.S)$. By checking if this value is larger than or less than 0.5, the comment is classified as being positive or negative. As shown in this figure, the *activations* are close to the *target* values shown in *Sentiment*. Thus, the model is currently a perfectly decent classifier of the data.

2.3 Neural Networks

Neural networks, also called *artificial neural networks*, are various types of networks that mostly consist of layers and networks of *perceptrons*. A perceptron is the same thing as described in **Section 2.2.1**, where a weighted sum of input numbers is sent through some activation function (there are more activation functions than just the *logistic/sigmoid* function).

What makes *Neural Networks* different is that, in contrast to the example in **Section 2.2.1**, the inputs are sent to several *perceptron* nodes that each have different settings, which in turn similarly send their outputs to a second layer of *perceptrons*, repeating the process until the final *output* layer is reached.

What makes *neural networks* so powerful is that conceptually, these in-between layers (called *hidden layers*) are creating new, more advanced inputs than what was possible with just the original inputs. This structure mimics the structure of networks of biological neurons, hence the name *artificial neural networks*.

A visualization of an example *artificial neural network* with three layers is shown in **Figure 2.4**. All layers between the input and output layers are called *hidden layers*.

2.3.1 Training and Backpropagation

The process of training these *neural networks*, called *backpropagation*, is very similar to how the desired parameter updates were found in **Sections 2.1.1** and **2.2.1**.

Starting from the output layer, the *loss function* is differentiated relative to the individual output nodes, using their current values to get the slope (called the *gradient*) of the the *loss*

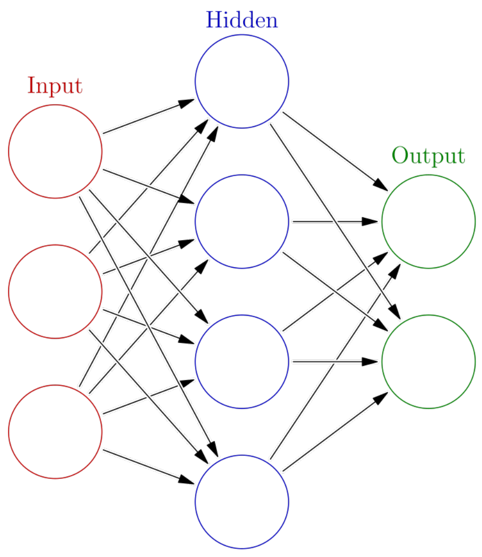


Figure 2.4: A visualization of a neural network by Glosser.ca (2013).

function. This *gradient* thus describes how each output node should be altered to decrease the *loss* value.

Similarly, the *loss function* is differentiated relative to that layer's preceding layer nodes and their weights, using their current values to get how they should be modified to achieve the desired changes in the nodes of the output layer. This procedure is then repeated for all layers, moving backwards throughout the network until the input layer is reached.

2.3.2 Architectures

Furthermore, there are several architectures of neural networks that have been shown to be suited to different tasks. The network described above, featuring input that is propagated through layers of only perceptrons, is called a *feed-forward network*.

To handle sequential data such as language better, *recurrent neural networks (RNNs)* were created, in which each node also has its previous value as input. To improve problems that sometimes occur when training these *RNNs* (the most important one being the *vanishing gradient problem*), the network architecture *Long Short-Term Memory (LSTM) networks* (Hochreiter and Schmidhuber, 1997) was created, which could handle long sequences of data. The *LSTM network* proved instrumental for several tasks including speech recognition.

However, due to the design of *LSTMs* (and *RNNs* as well), each piece in a sequence of data needs to be processed sequentially. In order to parse a sequence of 4 items, an LSTM cannot arbitrarily start processing item 3; it has to process item 1 first, then item 2 before finally processing item 3. This inherent sequential dependency means parallelizing the training process is difficult, making these networks slow to train.

A new neural architecture called the *Transformer* (Vaswani et al., 2017), which parses sequences and their ordering utilizing an entirely different mechanism, was able to produce models that outperform previous architectures for many text-related tasks, whilst offering parallelizable and thus faster training.

2.4 Embeddings

Embeddings are a specific type of *projection*. A *projection* is basically a lossy compressed version of any (usually binary) input data, where that compressed version was generated using a neural network.

What makes *embeddings* different from *projections* is that *embeddings* aim to somehow also represent the meaning of the data and how they relate to each other, in some sense. Understanding what they are and how they are useful requires quite a bit of context, so I will instead lead by discussing one type of *embeddings* called *word embeddings*, framing them as an alternative to *bag-of-words* vectors.

2.4.1 Word Embeddings

While techniques like *BOW* vectors shown in **Section 2.2.1** and TF-IDF vectors are conceptually simple and are effective for many tasks, they don't encode any actual *meaning* of words and sentences. They just restructure the data by labeling each possible word with a unique index. Furthermore, these vectors are often very sparse, containing a lot of 0's and just a few 1's.

Word embeddings are instead pre-generated numerical vector representations of words, with the property that words with a similar meaning are close to each other in vector space. A famous example of the power of *word embeddings* from Mikolov et al. (2013) is to look up the *word embedding* vector v_{king} that representing the word *king*, subtract the vector v_{man} that represents *man* and add the vector v_{woman} for *woman* and search the list of embeddings for the vector most similar to the result, which is *queen*.

$$v_{king} - v_{man} + v_{woman} \approx v_{queen} \tag{2.1}$$

Using these *word embeddings* in models for tasks like *text classification*, you can reduce the complexity of the model and still achieve good, possibly even better results. With the example shown in **Table 2.1**, a model that has *word embeddings* as input does not need a separate input and weight for each possible positive or negative word to do it's job; the *word embedding* vectors corresponding to negative words like *sucks* and *terrible* will likely be very similar, and positive words will analogously be similar to each other.

The model can be far simpler, and conceptually classify comments based on meaning instead of checking if a word is "in the *negative* or *positive* word lists", as was conceptually done with the hand-picked weights in **Table 2.4**.

What's been shown here specifically is *static word embeddings*, where each word has one single vector representation. Some popular *static word embeddings* include *word2vec* by Mikolov et al. (2013) or *GloVe* by Pennington et al. (2014). An innate disadvantage of using static word embeddings is that words can mean different things depending on context; the word *bank* can refer to the financial institution or a region of land alongside a body of water. There are *dynamic* or *context-sensitive word embeddings* that exist to solve this problem, which is one of the outputs that *BERT* produces. This will be elaborated on in **Section 2.5.4**.

2.4.2 Encoder-Decoder models

Words embeddings may seem a bit a bit magical, but they are normally created as a byproduct of a neural network built with the *Encoder-Decoder* architecture. *Encoder-Decoder* models are neural networks that are conceptually two separate neural networks blocks, called the *encoder* and the *decoder*. They are connected to each other in a specific way and specifically trained to perform slightly different tasks – *encoding* and *decoding*.

The intention of the *encoder* sub-network is to compress large, high-dimensional input vectors into a singular small vector that hopefully works as a useful, meaningful representation of the input. Hence the name: it *encodes* the input. This compact vector representation is in turn sent to the *decoder* block, whose purpose is to then somehow decompress or create new data from this compact vector, where the creation of this new data must be made with a "deep understanding" of the input data; i.e. the *decoder* block *decodes* the data.

Autoencoder

The *autoencoder*, visualized in **Figure 2.5**, is a simple network that serves as a good depiction of how these *encoder* and *decoder* blocks come to actually interact and work together.

The *autoencoder* is a neural network specifically designed and trained to work like a lossy compression algorithm. It takes a vector of bits as inputs, feeds them to a layer of much smaller dimension (less nodes) than the input, and sends the outputs of this small in-between layer to the final output layer. During training, the targets of the output nodes are set as the original input vector; the *decoder* is thus trying to recreate the original input from the smaller hidden layer, the *embedding*.

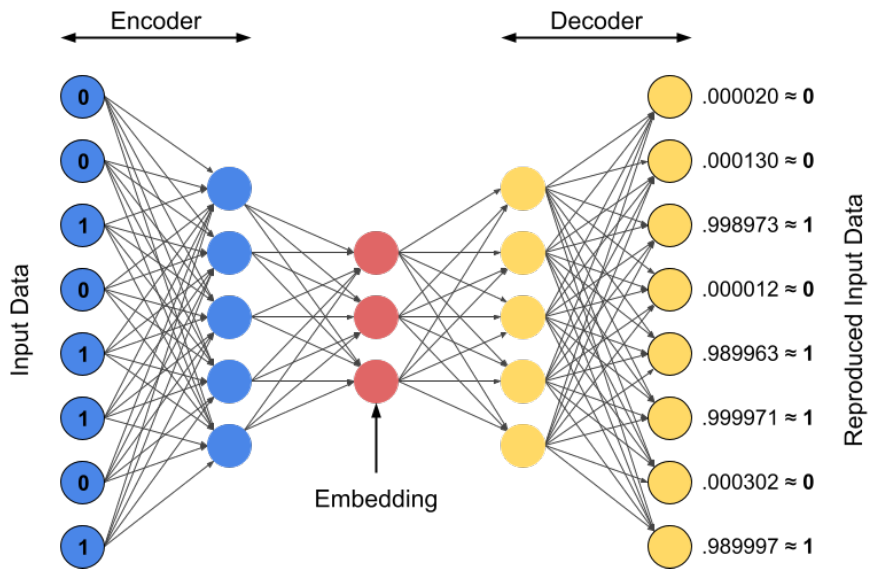


Figure 2.5: Illustration of a well-trained *autoencoder* network.

In the example shown in **Figure 2.5**, the input vector of eight bits is passed through smaller and smaller layers, to a layer that contains only three nodes. The outputs of these three nodes are then sent to larger and larger layers until the output layer is reached, which is hopefully recreating the input vector with its output nodes.

If the network is trained successfully, the outputs of this smallest layer of the nodes is seen as being the *embedding*, the *encoded* version of our input. Thus, through the training process, the *encoder* block becomes trained to transform the input into a usable *embedding* that can be fed as input to the *decoder* block, which is simultaneously trained to able to reconstruct the input data.

Separating the Tasks of Encoders and Decoders

This technique of structuring a network into an *encoder* and a *decoder* block is very powerful. While the autoencoder network is useful in itself for compressing data, tweaking what the encoder and/or decoder is doing allows a network to perform much more advanced tasks.

I will revisit this concept of a *decoder* performing a slightly different task in **Section 2.4.3**, where the *encoder* and *decoder* blocks will still function similarly as in the *autoencoder* example.

While a full explanation will not be provided here, the word embeddings in *word2vec* (Mikolov et al., 2013) were generated using such an *encoder-decoder* network. Compared to the compression and decompression tasks for the *encoder* and *decoder* blocks of the *autoencoder*, their goals are tweaked in *word2vec*. Instead, they were in tandem trained to guess a word that was removed from a sentence and additionally doing the reverse; predict the surrounding words of a given word. After training was completed, a word was input and the values smallest layer in the network was then collected, representing that word's *word embedding*.

2.4.3 Sentence Embeddings

While individual word embeddings are useful, it is hard to use them out-of-the-box for various NLP tasks. Relating to the example in **Section 2.2.1**, the comments differ in length, making it hard to format their *word embeddings* as input to a model. A good solution to this is to somehow obtain a fixed-sized vector that can represent the sentence as a whole.

A simple method that achieves this is to take the average of the *word embeddings* corresponding to the words in each comment/sentence. However, this approach does not work well in practice (Conneau et al., 2017). Additionally, given the two sentences:

That's not a bad idea, its a good one
That's not a good idea, its a bad one

Table 2.5

We see that they contain the exact same words. However by switching *good* and *bad*, the sentences mean different things. Alas, since the sentences have the same words, the average of their *word embeddings* would be the same.

Using a Encoder-Decoder Network for Translation

As shown in **Section 2.4.2** for *word2vec*, *encoders* and *decoders* can be designed to perform different tasks through a powerful *embedding* vector. Expanding on this thought, we can try to encode entire sentences into one vector. For example, we will imagine an *encoder* that sequentially encodes *word embeddings* into a new output, a "hidden state" vector. If the *encoder* is correctly trained, this "hidden state" will be increasingly embedded with information from the sentence. Upon processing the last word, the *encoder* will produce what can be considered a *sentence embedding*. An example usage of this is a basic *neural machine translation* network, shown in **Figure 2.6**, which was inspired by the model architecture used for an earlier version of *Google Translate*. (Wu et al., 2016)

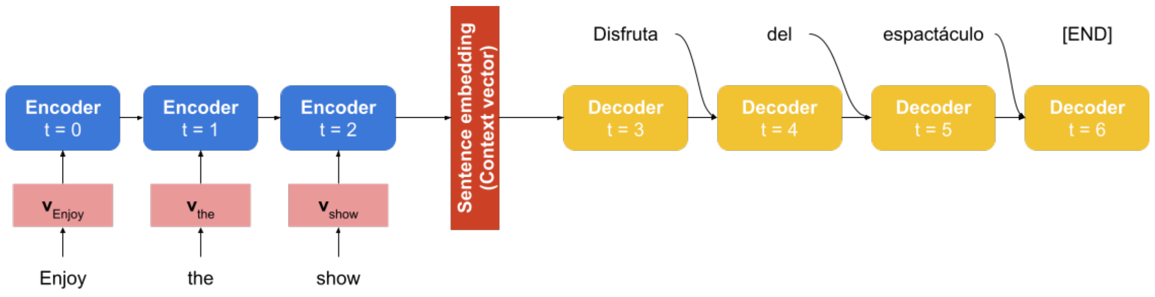


Figure 2.6: An *encoder-decoder* network that encodes words into a *sentence embedding* (usually called *context vectors* in these networks), which is used by a decoder that is trained to output the sentence in a different language; which in this example is Spanish.

The *embedding* of each word in the sentence *Enjoy the show* is retrieved and sent back to the *encoder* one word at a time. After parsing the first word and producing an output, that *encoder* output, called a *hidden state*, is sent back into the same encoder along with the next word. With each new word, this *hidden state* from the encoder is hopefully continually embedded with more and more "meaning" until the encoder has parsed all words, at which point the *hidden state* output of the *encoder* can be perceived as the *sentence embedding*.

In this translation model, the *sentence embedding* is then sent to a *decoder*, which continually outputs words while piping the remains of the *hidden state* back into itself, until the *decoder* declares the output sentence finished by outputting an "end token".

Harkening back to the *decoder* in the *autoencoder* example (**Section 2.4.2**), the *decoder* is in this case almost trying to "decompress" the *sentence embedding* back to its original words, although this time in a different language. This example is clearly similar to what the *autoencoder* network is doing, however it should now be apparent that we can train networks that intelligently represent words, sentences and almost any other data as meaningful fixed-size numerical vectors, as a byproduct of training on other tasks.

2.5 BERT

BERT (Devlin et al., 2019) is a *Transformer*-esque model that can be thought of as being an excellent "model of language" that, while not useful in itself, performs extremely well when altered to be used for other NLP tasks like text classification, question answering, entity-recognition and more. As *BERT* is the underlying architecture of the models used in this thesis, *BERT* will be explained in relatively deep detail. This description formally starts at **Section 2.5.3**, but is prepended with clarifications on the terms *Language Model* and *Transformer* as they can help frame an understanding of the model.

2.5.1 Language Model

Before I move on, we will take notice of the term *language model*, which has a very specific definition in NLP that is quite abstract and doesn't seem too useful at first glance. Given the two tasks it solves, the name *language model* seems to be a bit of a misnomer and could instead be described as a *next-word* and *sentence-probability* predictor.

The first task a *language model* performs is to, given a set of words, provide a probability distribution of what the next word is. For example, *language models* are used when typing something on a modern smartphone that gives a suggestion of the next word is. When typing *I'm going to the*, the *language model* in a smartphone might suggest *store*.

The second task a *language model* performs is to, given a sentence, compute the probability that it is actually a real, grammatically correct sentence. A good language model will give a high score to the sentence *I'm going to the store*, and a low score to *Store to going I'm the* .

While direct usages of language models have useful applications as shown above, they can also be perceived as having "learned how language works" in some sense.

By training a *language model* but then ignoring the outputs that actually make it a "next word" or "actual sentence" predictor, it can be viewed as being an *encoder* of not just words or sentences, but as an *encoder* of "language" in the broadest sense. The reason that a general *language encoder* is considered useful is that a model that "understands language" well can probably be a good interim step for later *downstream* NLP tasks. This is precisely what *BERT* has been shown to be. A "*language encoder*" model is indeed a very vague and ambiguous definition, but I will need to describe a few more things about *BERT* in order to understand how it fulfills this purpose.

2.5.2 Transformer and Attention

As mentioned, *BERT* is a *Transformer* model, which is a *neural network architecture* created by Vaswani et al. (2017). The *Transformer* is a rather complex architecture and a complete explanation will not be provided here. Instead I will highlight two of it's main contributions that are particularly relevant to understanding how *BERT* is faster and learns language better.

The *attention mechanism* is a mathematical procedure that can be applied in NLP to encode how relevant one word is to another. When the *decoder* in **Figure 2.6** is producing the first Spanish token, *attention* can be implemented, which would let the decoder look at each input word embedding and weigh their relevance for the to-be-generated first Spanish token. The computation of *attention* is a series of matrix multiplications, and the matrix parameters are learned through training so that the resulting *attention* values work as intended. to act An important property is that each *attention* calculation is independent from each other and thus parallelizable.

The *Transformer* uses this *attention mechanism* when encoding the input sequence as well, replacing the sequential dependence of encoders like in **Figure 2.6**, allowing for parallelization during training.

Furthermore, *encoders* utilizing *attention* allows truly bidirectional nuances in encoding. This is different from earlier pseudo-bidirectional encoders, which usually employs one encoder that processes words left-to-right like in **Figure 2.6**, a second encoder that parses the sentence in reverse (right-to-left) and then simply concatenates the two encoders' embeddings for each word.

2.5.3 BERT Architecture

In upcoming sections, I will describe how each of the parts work in detail, but as a starting point I will show the general architecture of *BERT* to be used as reference for how the network looks, is structured and how data flows through it.

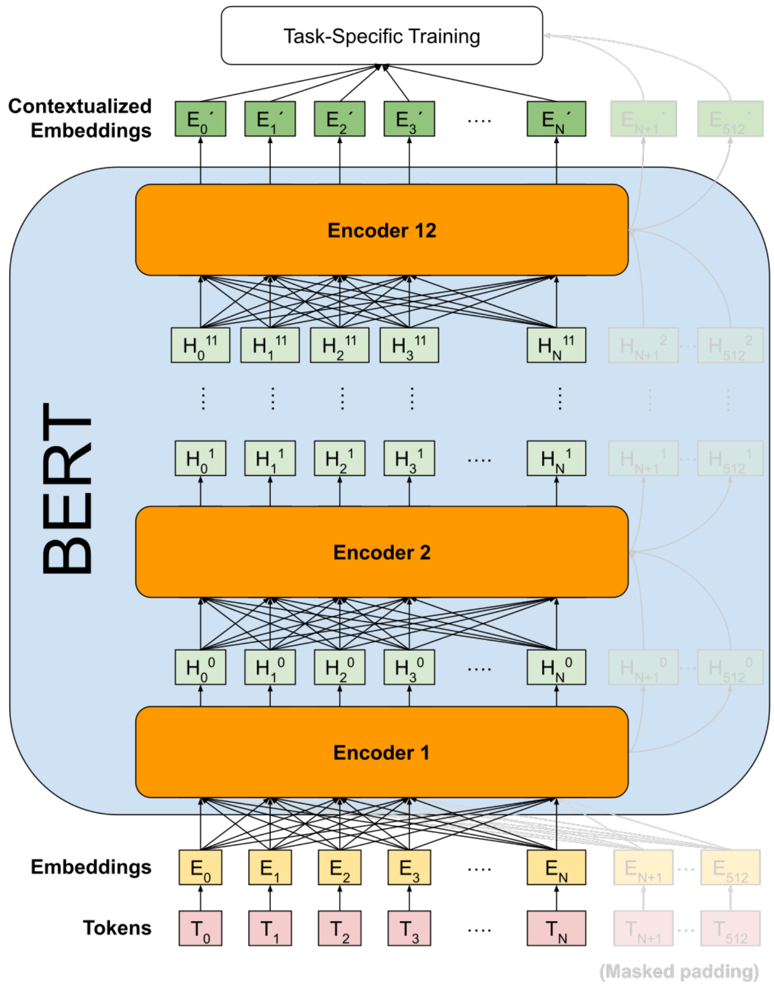


Figure 2.7: An overview on the architecture of *BERT*.

BERT parses a sentence by splitting the sentence into individual words, also called tokens, which are all sent in parallel into a linear stack of encoders. To be specific, words are represented using “interim” word embeddings called *WordPiece* embeddings, described in **Section 2.5.4**. Important to note is that *BERT* receives all the words as parallel individual inputs, in contrast to the network shown in **Figure 2.6**.

Sentences are obviously variable in length, however *BERT* handles this by adding padding to sentences shorter than 512 words, truncating sentences longer than that and applying an attention mask that essentially zeroes out all padding tokens so they don’t affect the encoder stack.

This input is sent to a stack of 12 serially connected encoders, where each encoder outputs 512 new word embeddings. Each encoder is thus receiving all word embeddings of a sentence and generating a new embedding for each word. The final output of the model is thus 512 word embeddings, where the first N output vectors of a sentence that is N words long are the word embeddings for each word and the remaining vectors are to be ignored.

2.5.4 Word Embeddings from BERT

We will start off by looking at the primary outputs of *BERT* that is seen in the model architecture. One of the primary purposes of *BERT* is to generate word embeddings: specifically, context-aware word embeddings.

BERT takes a full sequence of words represented by "interim" static *subword* (explained later) embeddings as input. It passes each of these "interim" word embeddings through layers of encoders in parallel, subtly evolving these embeddings after each encoder until it finally outputs enriched embeddings for each of these words.

The term "enriched" in "enriched word embedding" refers to the fact that word embeddings generated by *BERT* are not static, but are informed by the other words in the sentence. As an example, the word *bank* in the sentences *I went to the bank by the river.* and *What is the balance of my bank account?* will be represented with different word embeddings when using BERT, in contrast to those from *word2vec* (Mikolov et al., 2013) or *GloVe* (Pennington et al., 2014).

When an *encoder* in *BERT* is generating a word embedding, it is not just looking at that word itself, but is also simultaneously using all the other words as input to the *encoder*. However, it does not do this blindly; it assigns an *attention* value to each other word. This *attention* value for a word can be thought of as describing *how relevant is this other word to the word we're currently focusing on?*. For example, in the sentence *I saw a dog, it was happy.*, the word *dog* is highly relevant to *it*, and *dog* while thus have a higher *attention* value when encoding the word *it*.

Details on Subword Tokenization and Layered Embeddings

An interesting feature of BERT is how it represents words using the aforementioned "dumb" static *word embeddings*.

BERT uses *WordPiece* tokenization (Wu et al., 2016), where an input sentence sentence string is broken down (tokenized), not just into separate tokens (words), but sometimes into *subwords* (parts of words). The word *embedding* may be broken into *em* and *bedding*, for example. Additionally, a word that was broken into subwords is made distinct from real words by using the prefix *"##*. *embedding* is thus actually broken into *em* and *##bedding*, to preserve that it is part of a larger word and to not mistake it with the other real word *bedding*.

Each possible word and subword is assigned its own unique static embedding, also stemming from Wu et al. (2016). This subword dictionary is extremely expensive, to the point where the *out-of-vocabulary* issue, which is when the embedding of a word is not present in the embeddings list, does not occur.

Furthermore, the vectors passed as input to *BERT*'s stack of encoders are not solely the *WordPiece* embeddings. Each word embedding is added with a vector called a *position "embedding"*, which represents if the word is the 1st, 2nd, 3rd or i:th word in the sentence. This allows the encoder to not only take surrounding other words into account when encoding a word, but also its position. The cruciality of this information was shown in the examples of **Table 2.5**.

While adding this *position embedding* to the *WordPiece embeddings* may seem like it is disrupting the relational nature of word embeddings, the math involved in computing the *attention* $\alpha_{i,j}$ that word w_i pays to w_j conceptually separates the *position* and *word* embedding, allowing the *encoder* to treat the words and their relative positions separately during training.

2.5.5 Pretraining and Tasks

BERT is pretrained on two simultaneous tasks in order to develop a deep understanding of language: *masked language modelling* and *next-sentence prediction*, both of which can be performed without *labeled data*. The originally proposed and published models *bert-base* and *bert-large* were thus trained on a very large corpus, a text dump of the entire English Wikipedia (Devlin et al., 2019). We will follow up the above section on word embeddings by describing its pretraining process for the first task.

Masked Language Modelling

BERT is trained to generate rich, context-sensitive word embeddings by training it on the task *masked language modelling*. This task means that a certain percentage of words fed to *BERT* during pretraining will be replaced with a special blank *[MASK]* token, and a *decoder* is attached to the *word embedding* of that masked word. The ultimate goal of the task and the *decoder* is then to predict what that missing word was.

As all encoders are connected to all of the input words simultaneously, the result of this of this task is that the encoder stack becomes trained to use the surrounding words when inferring the missing word, which in turn trains them to create context-aware word embeddings as side effect.

Next-Sentence Prediction

However as mentioned earlier, *BERT* is not being trained on just one task. A training example loaded from the corpus of *BERT* actually includes not just one, but two sentences A and B, that are fed to the model simultaneously.

This second task of the model during pretraining is to predict if these sentences follow each other, i.e. if sentence B follows sentence A. While this is indeed a *supervised* task, labeling the training examples is easily automated; 50% of the time when a sentence A was selected from the corpus, the actual succeeding sentence was selected as B, whereas the other 50% of the time a completely random sentence from the corpus was selected.

Sentence pairs

Sentence A	Sentence B	IsNext
The man went to the store.	*He bought a gallon of milk.*	**True**
The man went to the store.	*Penguins are flightless birds.*	**False**

Table 2.6: Example of Next-Sentence prediction on two sentence pairs from Devlin et al. (2019)

To give the model the ability to complete this task, it needs to be able to understand that these are two separate sentences and obtain an overarching grasp of the two complete sentence to be able to compare them.

The first requirement is enabled via two features: a special token *[SEP]* that is placed between sentences ***A*** and ***B***, as well as a *segment embedding*, a predefined value that is added to each word's embedding, just like the *position embedding* described in **Section 2.5.4**. These two features provide the ability for the model to understand that these are two different sentences.

The latter requirement, actually grasping the sentences and comparing them, is facilitated by a single special token *[CLS]* that is prepended to the entire sequence of ***A*** and ***B***. After being fed through the encoder stack, the output embedding of this *CLS* token is fed to a perceptron that represents the models prediction of *IsNext*. Thus the *[CLS]* token is trained to be an embedding that represents an understanding of both sentences and if they follow each other.

A detail that might further illuminate how *BERT* is trained to understand language is that sentences ***A*** and ***B*** don't have to be complete sentences. They can be arbitrary strings of text picked from the corpus that either directly follow each other, wherein *IsNext* is *True*, or are picked from two completely different sources upon which *IsNext* is set to *False*.

Full Input and Pretraining

Now that both pretraining tasks have been shown, we can review how one training example looks in its entirety, shown in **Table 2.7**.

The two sentences are tokenized using the *WordPiece* tokenizer with some words randomly masked. They are then concatenated, separated with the insertion of the special token *[SEP]* and finally the entire sequence is prepended by the special *[CLS]* token. Each token in the sequence is processed in parallel by the encoder stack and a decoder is attached to the final encoder output of masked tokens' embedding, where the decoder is trained to predict the missing word. The output embedding of the *[CLS]* token from the encoder stack is attached to a perceptron that is trained to represent if sentence ***B*** follows ***A***, training *[CLS]* to become sort of an embedding of both sentences.

Sentence pairs	IsNext
[CLS] *the man went to* ***[MASK]*** *store* ***[SEP]*** *he bought a gallon* ***[MASK]*** *milk* ***[SEP]***	**True**
[CLS] *the man* ***[MASK]*** *to the store* ***[SEP]*** *penguin* ***[MASK]*** *are flight ##less birds* ***[SEP]***	**False**

Table 2.7: A typical training example supplied to *BERT* during pre-training, from Devlin et al. (2019). Here we can see how the input is prepared so the model can train on the two simultaneous tasks, as well as how *WordPiece* subword tokenization is handled in how *"flightless"* is split into *"flight"* and *"##less"*.

2.5.6 Fine-Tuning for Text Classification

We should now be able to understand how *BERT* is trained to understand language on a broader level: for individual words, sequences of words and a general grasp on textual and language continuity.

BERT's power in this regard is mainly shown when it is used as an intermediate step when training on other tasks. It can by trained to perform different tasks by changing the structure of the input and/or attaching an additional output layer on the embedding outputs of *BERT*. The model weights can either be left as is while training only the added output layer or the model can alternatively be *fine-tuned*, meaning the internal parameters of *BERT* are adjusted slightly for the desired task. While *fine-tuning* is indeed more expensive, *BERT*'s pretraining often produces an encoded understanding of language effective enough that internal weights rarely need to be adjusted in any large capacity.

While many tasks can use the frozen weights of *BERT* and only train an output layer, *text classification* usually requires some sort of fine-tuning, as nothing we've seen so far in the *BERT* architecture can be directly viewed as an encoding of a full sentence. However, the special token *[CLS]* has been pretrained to hold vaguely similar properties.

As directly proposed by the authors of *BERT* Devlin et al. (2019) in their paper, a class/label decoder can be attached to the *[CLS]* output embedding and then be trained and fine-tuned for the task, causing the [CLS] embedding to morph into a *sentence embedding*. As previously mentioned, the similarity of properties between the *[CLS]* embedding and a *sentence embedding* means the *fine-tuning* process should not require too many drastic changes to *BERT*'s internal parameters.

This architecture is shown in **Figure 2.8** and was used by Devlin et al. (2019) when benchmarking *BERT* on the down-stream tasks *SST-2* (Socher et al., 2013) and *COLA* (Warstadt et al., 2018). It is also the task-specific architecture used on the two *BERT* models described below in **Sections 2.5.7** and **2.5.8**.

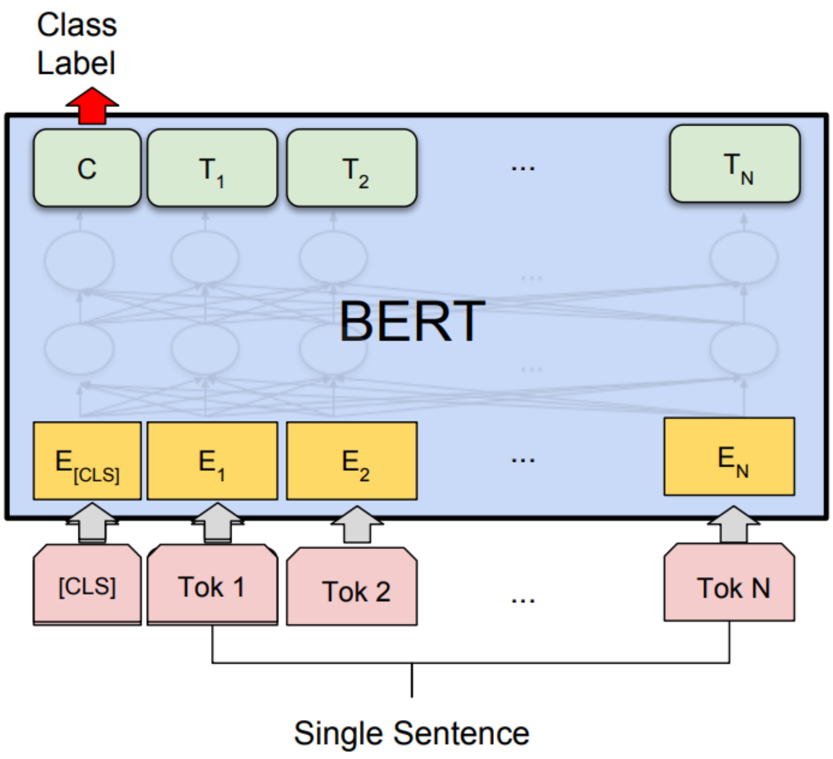


Figure 2.8: A modification of the *BERT* model architecture designed to perform sentence classification, from Devlin et al. (2019).

2.5.7 mBERT: a Multilingual BERT Model

Using the *BERT* architecture, Google published a *BERT*-model called *mBERT* (Google, 2018), trained on text data in over 104 languages. *mBERT*'s corpus was the text data from all Wikipedia articles belonging to the top 104 languages on Wikipedia with the most textual data.

Flair

Flair (Akbik et al., 2019) is a software library that facilitates easy usage of NLP techniques to solve some typical NLP tasks, such as text classification. It supports a number of techniques, algorithms and models, including several pretrained models, one of which is *mBERT*. *Flair* is used in this thesis to load *mBERT* and fine-tune it for classification on the corpus of this thesis, the food product names and classes described in **Section 3.1**.

2.5.8 The Swedish KB-BERT Model from Kungliga Biblioteket

Swedens national library *Kungliga Biblioteket* published a monolingual *BERT* model for the Swedish language, called *KB-BERT* (Malmsten et al., 2020). It was trained on a vast Swedish corpus procured by *Kungliga Biblioteket*, including text from digitized archives of newspapers, government publications and reports, ebooks, social media, online forums and all Swedish articles on Wikipedia.

HuggingFace Transformers

HuggingFace Transformers is a software library that hosts a multitude of modern *Transformer* models such as *BERT*, as well as versions of these models pretrained on various corpora. It also hosts community submissions of pretrained models such as *KB-BERT*. *HuggingFace Transformers* also facilitates a range of pre-made extensions of models to accommodate popular NLP tasks such as text classification.

2.6 Terminology

This section contains definitions of some additional important terms that either weren't fully elaborated on in earlier sections, or were not mentioned so far, that are used in this paper.

Training Example A *training example* is a data point used by a machine learning model during training to improve or evaluate its performance on a given task. in **Table 2.1**, each of the comments is a data point and each data point used to train the model is a *training example*.

Corpus A corpus is a collection of data used for a machine learning task. In this thesis, Mashies *products document* (described further in **Section 3.1**), containing a vast collection of product names and their associated product category, is the *corpus* used for this thesis. A corpus may be *annotated*, meaning each individual data point has metadata that may be used as the *target values* (answers) for the task at hand.

Epoch An epoch is the same thing as the *training iteration* described in **Section 2.1.1**, a full pass of model parameter updates from all data points in the *training dataset*.

Training Batch and **Batch Size** When training very large models on large datasets, it is rarely possible to pass the entire dataset through the model in a single step due to memory constraints, particularly on GPUs. To handle this, the *training dataset* can be split up into smaller pieces known as *batches* that are passed through the model. The model parameters are then usually updated to minimize the *loss function* after each *batch*. Alternatively, parameter and weight updates can be accumulated and performed after either all or a predefined number of *batches* have been passed through. The *batch size* specified the number of training examples contained in a single *batch*.

Optimizer The optimizer is the scheme/method used to calculate how the parameters of a model should be updated during training. In **Section 2.1.1**, I described the optimizer called *stochastic gradient descent* (SGD), which updates parameters by subtracting a fraction of the *loss function*'s gradient, where the gradient's multiplier is a small number called the *learning rate*. There are more optimizers such as *adaptive optimizers*, that employ custom learning rates for each parameter that are adjusted individually throughout the training process.

Learning Rate Scheduler Instead of just using the same *learning rate* throughout all iterations, it is possible to use a *learning rate scheduler* that deterministically varies the learning rate after each training iteration. There are many schedulers, such as the cycle scheduler, sinoid scheduler, the step scheduler and more. In this thesis, the *annealing step scheduler* was used, which reduces the learning rate with a multiplicative *annealing factor*, after a specified amount of *bad epochs*, meaning *epochs* where no improvement was observed.

Support In *classification* tasks, *support* refers to the number of training examples that belong to a class. In the example shown in **Table 2.1**, there were 3 comments with positive sentiment, thus the class *positive sentiment* had a support of 3.

Precision is a measure of how confident you can be that a prediction made by a model is true. Using **Table 2.1** as an example, if a classifier model had predicted the first 3 comments as positive, incorrectly predicted the 4:th comment as positive and correctly classified the rest as negative, the positive class would have a *precision* of **3/4**, or 75%, as 3 out of the 4 training examples were actually positive.

Recall is a measure of how many of the training examples were captured by the classifier and put in the correct class. If only the first comment in **Table 2.1** was predicted to be positive, the *precision* of the positive class would be 100%, as it did not actually incorrectly classify anything as positive when it was actually negative. However, the *recall* of the positive class would only be **1/3** or 33.3...%, as it "missed" the other 2 positive comments.

F1-score As both *precision* and *recall* are needed to properly describe the performance of a model, the *F1-score* is often used to combine both metrics into a single number. The *F1-score*, a value between 0 and 1 just like *precision* and *recall*, is the harmonic mean of these two metrics, as shown in **2.2**. For the model example shown in the definition of *precision*, the *F1-score* of the positive class is 85.7%. Analogously for the model in the definition of *recall*, the *F1-score* of the positive class is 50%.

$$F1\text{-}score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.2}$$

Macro-average The *macro-average* of a metric is the direct mean of these metrics across all classes, with no weighting given to any of them. Given a model that identifies the presence of tumors in x-ray images, most datasets will probably have a very high ratio of non-tumor images, we will use an example of 99 non-tumor images and 1 image with a tumor. The model will likely be trained to solely predict *does not contain tumor* for every image, giving the *tumour* and *no tumour* ~100% and 0% F1-scores, respectively. Although the model obtains low *loss*, this is probably not desired behavior and the *macro-average* F1-score will indicate that, as it will be $\frac{\sim 100\% + 0\%}{2} \approx 50\%$.

Confusion Matrix In the example presented in the definition of *Precision*, there were 3 positive comments which were all predicted as positive and 3 negative comments, 2 of which were predicted as being negative. While the metrics *precision*, *recall* and *F1-score* are good overall metrics to see how the model is behaving, a *confusion matrix* allows us to see how the model is classifying these training examples in further detail. A *confusion matrix* is a matrix of cells, where each cell described how many training examples with a particular true class $class_{true}$ was *predicted* to belong to a class $class_{pred}$. The cell's row describes which class it actually belongs to, $class_{true}$, while its column describes which class it was *predicted* to belong to, $class_{pred}$. **Table 2.8** show the confusion matrix that represents this example. These *confusion matrices* are particularly useful when classifying among multiple classes in order to detect if there is any *systemic misclassification*, e.g. when training examples are regularly being predicted as belonging to a particular but incorrect class.

Actually Positive	3	0
Actually Negative	1	2
	Predicted Positive	**Predicted Negative**

Table 2.8: Confusion matrix for the example where 3 positive comments were correctly predicted as *positive*, 2 negative comments were correctly predicted as *negative* and 1 negative comment was misclassified as *positive*.

Chapter 3
Methodology

The original task proposed by Mashie was to investigate how many details and properties of a product could be accurately predicted, using unrefined product names and descriptions recieved from manufacturers of food products and related goods. This task was reformulated as an investigation into the efficacy of predicting a products category with NLP models, using only the product names as input data. This problem was thus interpreted as a *multi-class text classification* task.

This chapter describes methodology of this investigation and evaluation, including the data, data processing and model selection.

3.1 Input Data from Mashie

Mashie provided a large Excel document containing details of 114,158 food and food-related products produced by various Swedish manufacturers, hereinafter denoted as the *products document*. Each row in this document describes an individual product and its properties, including the product's name and category.

Originally, this data was sourced by Mashie from various manufacturers via product datasheets. These manufacturer product sheets often use a wide variety of shorthands and include irregular tokens and typographical errors. Part of Mashie's operations involve cleaning, transcribing and structuring this data. In this thesis, only Mashie's refined versions of product names were used and thus the only preprocessing deemed necessary was transforming it to lowercase and consolidating whitespace tokens. These product names are akin to what you would find on an online grocery store.

Document and Product Details In the *products document*, the category of a product is described in three levels:

1. The top level is a broad grouping of goods called *varuområde*. This entails groups such as *frozen goods*, *fresh produce*, and *non-foods*.

2. The next level, *huvudgrupp*, denominates a group of goods within the aforementioned *varuområde*. Examples of a *huvudgrupp* under *fresh produce* are *meat products*, *unprocessed vegetables* and *berries*.
3. The final level of categorization is a product-type level of categorization called *varugrupp*. This specifies quite narrow ranges of products, such as *carbonated soda*, *mineral water*, *cereal*, *cucumbers*, *pork* and more.

Table 3.1 contains a few example products from the *products document* and their associated *varuområde*, *huvudgrupp* and *varugrupp*.

Table 3.1: Examples of a product's 3 levels of categorzation and the product names.

Varuområde	Huvudgrupp	Varugrupp	Product name
Färskvaror/Kylvaror	Frukt/bär	Bananer	Bananer (5kg) (25-30st/krt)
Djupfryst	Kött oberett	Blandfärs	Blandfärs 70/30 Lf
Kolonial/Speceri	Baktillbehör	Bakchoklad	Mjölkchoklad pellets Lait Entier 37%

3.2 Data Processing

The three levels of categorization for a product were combined and interpreted as one unit, hereinafter called the *taxonomy*. The *taxonomy* is thus a text string with the format *<varuområde><huvudgrupp><varugrupp>*. This provided a clear set of classes which products belonged to. With these definitions, the *products document* contains 114,158 products that belong to 2,808 unique taxonomies.

3.2.1 Procedure

Of these 2,808 taxonomies, a distribution count was made and the top 40 taxonomies associated with the largest number of products were selected. This number of taxonomies was selected as it was small enough to train relatively quickly and is deemed a large enough number to contain a diverse selection of taxonomies. This diversity thus provides an indication how the model would work when classifying products among all taxonomies and categories.

Following this, the list of all products was split into training, validation and test sets, using an 80%, 10% and 10% ratio. This was performed by picking a random product and placing into the validation set until the validation set contained 11,415 products, e.g. 10% of the total number of products. This process was repeated for the test set, and the remaining products were used as the training set. To ensure comparable results across both models, the random-number generator used for the set splitting process was initialized with a fixed seed, seed 0.

The training set was then optionally reduced when studying the effectiveness of the model on lower amounts of training data. It was reduced in a similar fashion, by deleting random products until the desired amount of products was left, using a random number generator that also used seed 0 to ensure comparable results between models. The validation and test sets were not reduced in this process, so the classification results for models trained on reduced datasets can be considered as valid as for the model trained on 100% of the data.

Finally, all products in the training, validation and test sets which did not belong to the aforementioned top 40 selected taxonomies were removed. Initially these products were included as an extra *unknown* taxonomy, but the resulting performance of the models was very poor and this idea was quickly dismissed.

3.2.2 Resulting Datasets

The resulting datasets were quite imbalanced, with training support for a taxonomy ranging from over 1600 training examples down to 300 training examples, as a direct result of the distribution of products in Mashie's *products document*. A table of each taxonomy and its training and test support can be viewed in **Table 3.2**.

No action was taken to resolve this imbalance, as the training support of a taxonomy did not seem to correlate with its classification performance when using 100% of the training data. For example, the taxonomies *hushållsglass* and *äpplen*, with 411 and 1765 training examples each, achieved the similar F1-scores, as shown in **Table 4.1**. This imbalance also offered an opportunity to granularly inspect the impact of training support on F1-score when decreasing the training dataset, further described in **Sections 3.3** and **4.2**.

Table 3.2: Table of the 40 chosen taxonomies for evaluation in this thesis. The right hand side shows the number of products/training examples in each taxonomy for the test dataset and training dataset; also known as *test support* and *training support*.

	Dataset support	
Taxonomy	**Test**	**Training**
<djupfryst><glass><glass stycksortiment>	47	398
<djupfryst><glass><hushållsglass>	31	411
<färskvaror/kylvaror><ägg><ägg färska>	34	329
<färskvaror/kylvaror><frukt/bär><apelsiner>	45	395
<färskvaror/kylvaror><frukt/bär><äpplen>	192	1765
<färskvaror/kylvaror><frukt/bär><melon>	68	542
<färskvaror/kylvaror><frukt/bär><päron>	95	833
<färskvaror/kylvaror><frukt/bär><småcitrus>	57	448
<färskvaror/kylvaror><frukt/bär><vindruvor>	42	415
<färskvaror/kylvaror><grönsaker obehandlade> - <groddar/skott/småblad>	84	676
<färskvaror/kylvaror><grönsaker obehandlade> - <grönsaker övriga>	31	305
<färskvaror/kylvaror><grönsaker obehandlade><gurka>	78	500
<färskvaror/kylvaror><grönsaker obehandlade><lök gul>	40	323
<färskvaror/kylvaror><grönsaker obehandlade><morötter>	83	637
<färskvaror/kylvaror><grönsaker obehandlade><paprika	66	664
<färskvaror/kylvaror><grönsaker obehandlade><sallad övrig>	154	1125
<färskvaror/kylvaror><grönsaker obehandlade><tomater>	148	1215
<färskvaror/kylvaror><kaffebröd/konditori><bakelse>	49	469
<färskvaror/kylvaror><kaffebröd/konditori><bullar>	49	321
<färskvaror/kylvaror><kaffebröd/konditori><tårtor>	166	1073
<färskvaror/kylvaror><kaffebröd/konditori><vetebröd>	44	323
<färskvaror/kylvaror><matbröd färdigt><bröd>	59	356
<färskvaror/kylvaror><matbröd färdigt><portionsbröd>	32	302
<färskvaror/kylvaror><mejerivaror><yoghurt smaksatt>	50	337
<färskvaror/kylvaror><potatis><potatis skalad>	33	364
<färskvaror/kylvaror><potatis><potatis>	131	1212
<kolonial/speceri><kolsyrade drycker><kolsyrad läsk övrig>	47	406
<kolonial/speceri><kolsyrade drycker><mineralvatten>	41	328
<kolonial/speceri><konfektyrer><choklad>	51	415
<kolonial/speceri><konfektyrer><konfektyrer>	176	1300
<kolonial/speceri><te/chokladdryck><tepåsar med kuvert>	46	363

Taxonomy	Test	Training
<non food><bägare,grepp,koppar engångs> - <bägare,grepp,koppar engångs>	30	346
<non food><formar,uppläggningsfat engångs><plastform/lock>	52	479
<non food><servetter,servettställ,dispenser><servetter>	63	545
<restaurangutrustning> - <hushålls och storköksmaskiner> - <hushålls och storköksmaskiner>	44	483
<restaurangutrustning><köksredskap><köksredskap>	90	787
<restaurangutrustning><köksutrustning><kantiner>	40	323
<restaurangutrustning><textil><textilier och kläder>	60	531
<vin/sprit/öl klass 3><vin><vin rött>	74	643
<vin/sprit/öl klass 3><vin><vin vitt>	76	496

3.3 Varying Training Data Amount

To further study the performance of the models, the option to reduce the training set was utilized by training the models on 100%, 50%, 20%, 10%, 5% and 1% of the training set. The resulting classification reports were then studied by generating the graphs shown in **Section 4.2**.

As mentioned in **Section 3.2**, the test set was unchanged for all models, to ensure valid comparisons. In these graphs, the F1-score was selected as the main metric for evaluation, as neither precision nor recall could be dismissed as an unimportant metric for the task.

3.4 Fine-Tuning the BERT Models

Initially, I chose the software library *Flair* to host the text classification model and pipeline, as it was advertised to deliver near state-of-the-art levels of performance (Akbik et al., 2019) whilst being relatively easy to use. Furthermore, *Flair* recommended using *Transformer* models specifically for text classification. Thus I selected the multilingual model *mBERT* (Google, 2018), as it was the only model compatible with *Flair* that was trained on Swedish text. This was important as most of the product names are in Swedish and are sourced from Swedish manufacturers.

Comparison with a monolingual Swedish BERT model was desired to see if the expected slightly higher performance of a monolingual model (de Vargas Feijo and Moreira, 2020) was replicated here.

Therefor, I selected the *BERT* model *KB-BERT* from *Kungliga Biblioteket* (Malmsten et al., 2020), as it is a recent *BERT* model trained on a very large Swedish corpus. *KB-BERT* is provided via the software library *HuggingFace Transformers*, which was thus used for fine-tuning.

Hyperparameters To ensure fair comparison of both models, I modified the trainer utility and settings provided by HuggingFace to be as similar as possible to *Flair*'s default training process and settings. Both training procedures used the following settings:

- Fine-tuning (as described in **Section 2.5.6**)
- 100 maximum epochs
- 0.1 learning rate
- A batch size of 32
- An SGD optimizer
- An annealing learning rate scheduler with:
  - A patience of 4
  - 0.5 annealing factor
  - Minimum learning rate of 10^{-4}

The annealing learning rate scheduler was set to halve the learning rate (annealing factor 0.5) if the evaluation loss had not improved after 4 epochs. If the learning rate fell below the minimum prior to reaching 100 epochs, training was terminated.

After training was completed, I loaded the model from the epoch with the lowest detected evaluation loss. The learning rate and number of epochs were set at *Flair*'s default values, which are rather high in comparison to the original suggestion made by Devlin et al. (2019), but serves to avoid underfitting. The subsequent possibility of divergence was avoided by using the annealing learning rate scheduler, which additionally counteracts overfitting by design. Both models terminated training at around 60-70 epochs due to reaching the minimum learning rate.

Chapter 4

Results

This chapter describes the test results and other diagnostics of the models. In the tables and diagrams, the multilingual model *mBERT* (Google, 2018) is denoted as *BERT Multilingual* or *Multilingual*. The Swedish model *KB-BERT* from Kungbib (Malmsten et al., 2020) is denoted as *BERT Swedish* or *Swedish*.

4.1 Overall Comparison

4.1.1 Classification Report

In **Table 4.1**, the results of the classification test on the test set is presented on a per-taxonomy basis, along with a macro-average across all taxonomies.

The taxonomies were shortened to aid readability, only containing the last part of the taxonomy, the *varugrupp*. They have been grouped and separated by lines according to their *varuområde*, which are *djupfryst*, *färskvaror/kylvaror*, *kolonial/speceri*, *non food*, *restaurangutrustning*, *vin/sprit/öl*, in the same order as presented in the table. *Varuområde*, *varugrupp* and the structuring of the taxonomy is described in **Sections 3.1** and **3.2**.

F1-score is selected as the main metric to compare performance of taxonomy prediction between the models, as both precision and recall are important to the task. Macro-average F1-score is selected as the metric to compare the overall performance of the models, as F1-score is important for the reasons mentioned above and the imbalance of the dataset should not be accounted for as all classes are of equal importance.

Table 4.1 shows that both models perform very well in general. Of the 40 taxonomies, 38 and 33 taxonomies have an F1-score over 90% and 25 and 26 taxonomies exceed an F1 score of 95% for the multilingual and Swedish models, respectively. The multilingual model outperforms the Swedish model with a 2.03% higher macro-average F1-score, and individually outperforms the Swedish model on a clear majority of the taxonomy predictions.

Both models achieve high macro F1-scores, with the multilingual model slightly exceeding the Swedish one.

Specific Results to Note

Both models share a relatively low F1-score for the taxonomies *choklad* and *hushålls och storköksmaskiner*. For the multilingual model, they are 8.04 and 9.17 percentage points below the macro-average F1-score, respectively. For the Swedish model, they are analogously 9.26 and 12.04 percentage points lower than the average.

Additionally, the Swedish model has a certain set of taxonomies that perform particularly poorly, both relative to their respective performance in the multilingual model as well as the macro-average of the Swedish model.

The taxonomies *vin rött*, *kolsyrad läsk*, *vin vitt*, *bullar* and *portionsbröd* are 5.32%, 5.67%, 9.55%, 10.12% and 25.09% points lower than their respective F1-scores in the multilingual model. In comparison to the Swedish model's macro-average F1-Score, these taxonomies are respectively 5.56, 5.06, 10.42, 12.53, 26.06 percentage points below the average.

Both models underperform in taxonomies *choklad* and *hushålls och storköksmaskiner*. The Swedish model particularly suffers from a specific set of taxonomies underperform, both in comparison to the macro-average and to their respective performances in the Multilingual model.

Table 4.1: Classification Report. Best result marked in **bold**.

	BERT Multilingual (Google)			**BERT Swedish** (Kungbib)		
Taxonomy (varugrupp)	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
glass stycksortiment	**0.9643**	0.9474	0.9558	0.9483	**0.9649**	**0.9565**
hushållsglass	**0.9811**	1.0	**0.9905**	0.963	1.0	0.9811
ägg färska	0.9783	1.0	0.989	0.9783	1.0	0.989
apelsiner	1.0	1.0	1.0	1.0	1.0	1.0
äpplen	0.9947	1.0	0.9973	0.9947	1.0	0.9973
melon	0.988	1.0	0.9939	**1.0**	1.0	**1.0**
päron	0.9921	**1.0**	0.996	**1.0**	0.9921	0.996
småcitrus	1.0	0.975	0.9873	1.0	**1.0**	**1.0**
vindruvor	0.9818	1.0	0.9908	1.0	1.0	1.0
groddar/skott/småblad	**0.989**	**1.0**	**0.9945**	0.9888	0.9778	0.9832
grönsaker övriga	0.9111	**0.9535**	**0.9318**	**0.9737**	0.8605	0.9136
gurka	1.0	1.0	1.0	1.0	1.0	1.0
lök gul	1.0	1.0	1.0	1.0	1.0	1.0
morötter	1.0	1.0	1.0	1.0	1.0	1.0
paprika	1.0	1.0	1.0	1.0	1.0	1.0
sallad övrig	1.0	**0.9774**	**0.9886**	1.0	0.9398	0.969
tomater	1.0	0.9933	0.9966	1.0	0.9933	0.9966
bakelse	1.0	**0.9636**	**0.9815**	1.0	0.9455	0.972
bullar	**0.8824**	0.9677	**0.9231**	0.7143	0.9677	0.8219
tårtor	**1.0**	0.9851	**0.9925**	0.9925	0.9851	0.9888
vetebröd	**0.925**	0.9487	**0.9367**	0.9024	0.9487	0.925
bröd	**0.9592**	**0.9038**	**0.9307**	0.92	0.8846	0.902
portionsbröd	**1.0**	**0.8824**	**0.9375**	0.697	0.6765	0.6866
yoghurt smaksatt	1.0	1.0	1.0	1.0	1.0	1.0
potatis skalad	**1.0**	1.0	**1.0**	0.9375	1.0	0.9677
potatis	1.0	**0.9932**	**0.9966**	1.0	0.9658	0.9826
kolsyrad läsk övrig	**0.9444**	0.9623	**0.9533**	0.8254	**0.9811**	0.8966
mineralvatten	0.9697	0.9143	0.9412	**1.0**	0.9143	**0.9552**
choklad	0.873	**0.9016**	**0.8871**	**0.8929**	0.8197	0.8547
konfektyrer	**0.8947**	0.9379	**0.9158**	0.8537	**0.9655**	0.9061
tepåsar med kuvert	1.0	1.0	1.0	1.0	1.0	1.0
bägare, grepp, koppar engångs	**0.9565**	0.9362	**0.9462**	0.8519	**0.9787**	0.9109
plastform/lock	0.95	**0.95**	**0.95**	**0.9811**	0.8667	0.9204
servetter	0.9872	1.0	0.9935	0.9872	1.0	0.9935
hushålls och storköksmaskiner	0.8689	**0.8833**	**0.876**	**0.9773**	0.7167	0.8269
köksredskap	**0.9551**	0.914	0.9341	0.9271	**0.957**	**0.9418**
kantiner	**0.9744**	0.9744	**0.9744**	0.95	0.9744	0.962
textilier och kläder	0.9437	0.9306	0.9371	**0.9459**	**0.9722**	**0.9589**
vin rött	**0.9167**	**0.9747**	**0.9448**	0.8506	0.9367	0.8916
vin vitt	**0.9683**	**0.9104**	**0.9385**	0.9444	0.7612	0.843
macro avg	**0.9687**	**0.967**	**0.9676**	0.9499	0.9487	0.9473

4.1.2 Evaluation of Confusion Matrices

To further understand the performance of various taxonomies in the classification report, I generated normalized confusion matrices shown in **Figures 4.1** and **4.2**. For readability, decimal percentages are only shown for values below 10% and taxonomies are shortened, similarly to **Table 4.1**.

These confusion matrices will make possible patterns of systemic misclassification clearly visible. The raw confusion matrices showing the actual number of products placed in the various taxonomies are not as easily digestible due to data imbalances. However, they are included in the Appendix as **Figures A.1** and **A.2** as they are only used later for minor comments on precision. In the confusion matrix for the Swedish model, **Figure 4.2**, a number of anomalies can be observed.

- 20% of *portionsbröd* products are mislabeled as *bullar*.
- 19% of *vin vitt* products are mislabelled as *vin rött*, but there is little confusion vice-versa.
- 13% of products belonging to the *choklad* taxonomy are mislabeled as belonging to the *konfektyrer*.
- 12% of products in *grönsaker övriga* were mislabelled as *bröd* and *konfektyrer* in even distribution.
- 11% of *plastform/lock* products are mislabelled under the taxonomy *bägare, grepp, koppar engångs*.
- 8.6% of *mineralvatten* products are mislabelled as *kolsyrad läsk*.
- *hushålls och storköksmaskiner* is an oddity, where 29% of products were mislabelled as a variety of taxonomies; the most common of them being *köksredskap*, *konfektyrer*, *kantiner* and *textiler och kläder* in descending order. All of these except *konfektyrer* are in the same *varuområde* called *restaurangutrustning*.

For the multilingual model displayed in **Figure 4.1**, there aren't as many pronounced misclassification anomalies of note. The few of note are:

- *vin vitt*, of which 9.0% are misclassified as *vin rött*,
- *choklad*, of which 6.6% are misclassified as *konfektyrer*,
- *portionsbröd*, of which 5.9% were misclassified as *konfektyrer*,
- *mineralvatten*, of which 5.7% are misclassified as *kolsyrad läsk övrig*,
- *hushålls och storköksmaskiner*, of which 5.0% are misclassified as *konfektyrer* and 6.7% are misclassified as taxonomies belonging to the *varuområde* named *restaurangutrustning*.

Comparing these results with **Table 4.1**, it is clear that these systemic misclassifications are largely responsible for underperforming taxonomies. Due to how the normalization was performed (row-wise), each cell where $i = j$ is equal to the taxonomies recall. Many of the aforementioned taxonomies with systemic misclassifications in the Swedish model have low recall scores which deteriorate the F1-score. Furthermore, there are also taxonomies that have lowered precision scores from products in other taxonomies being systemically misclassified. This is best illustrated in the raw confusion matrices, which is shown for the Swedish model in the appendix as **Figure A.2**. Some prominent examples are:

- *bullar*, where a large number of *portionsbröd* products were misclassified as *bullar*.
- *vin rött*, where a substantial portion of *vin vitt* products were misclassified into *vin rött.*

The most significant outlier from this pattern is *portionsbröd.* As visible in the raw confusion matrix in **Figure A.2** , the low precision of this taxonomy stems from products in a large variety of taxonomies being predicted to belong to this taxonomy.

The poor performance of the aforementioned taxonomies in the Swedish model largely stems from systemic misclassification. While the feature selection of product names in combination with taxonomy similarity may seem to be the intuitive explanation of the phenomenon, the substantially better results of the multilingual shows that is not the full explanation.

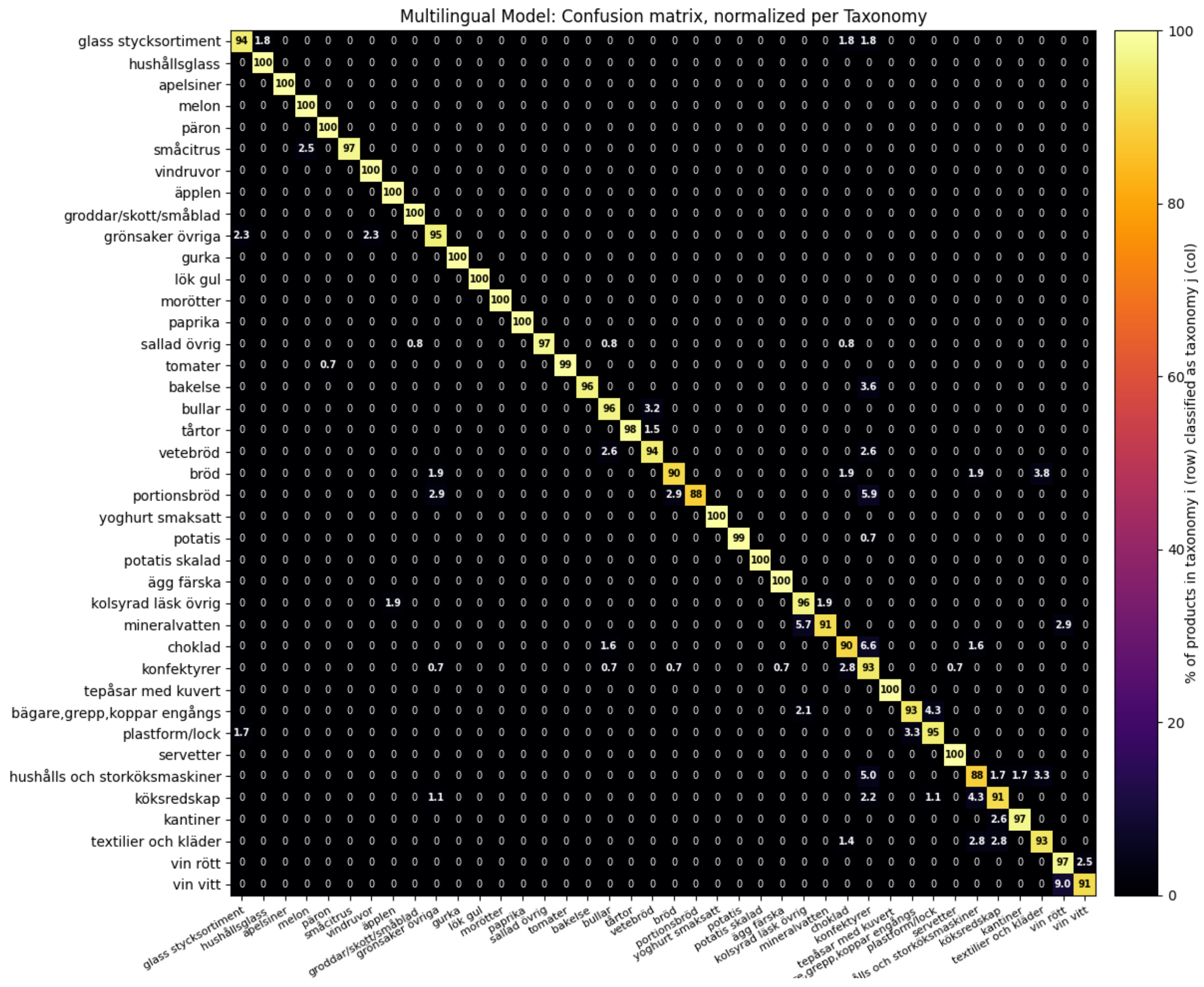


Figure 4.1: Confusion Matrix for the **Multilingual Model**, normalized on the number of total products actually within each taxonomy (normalized row-wise). Each cell shows the percentage of products that belonged to taxonomy i (row i), that were predicted to belong to taxonomy j (column j).

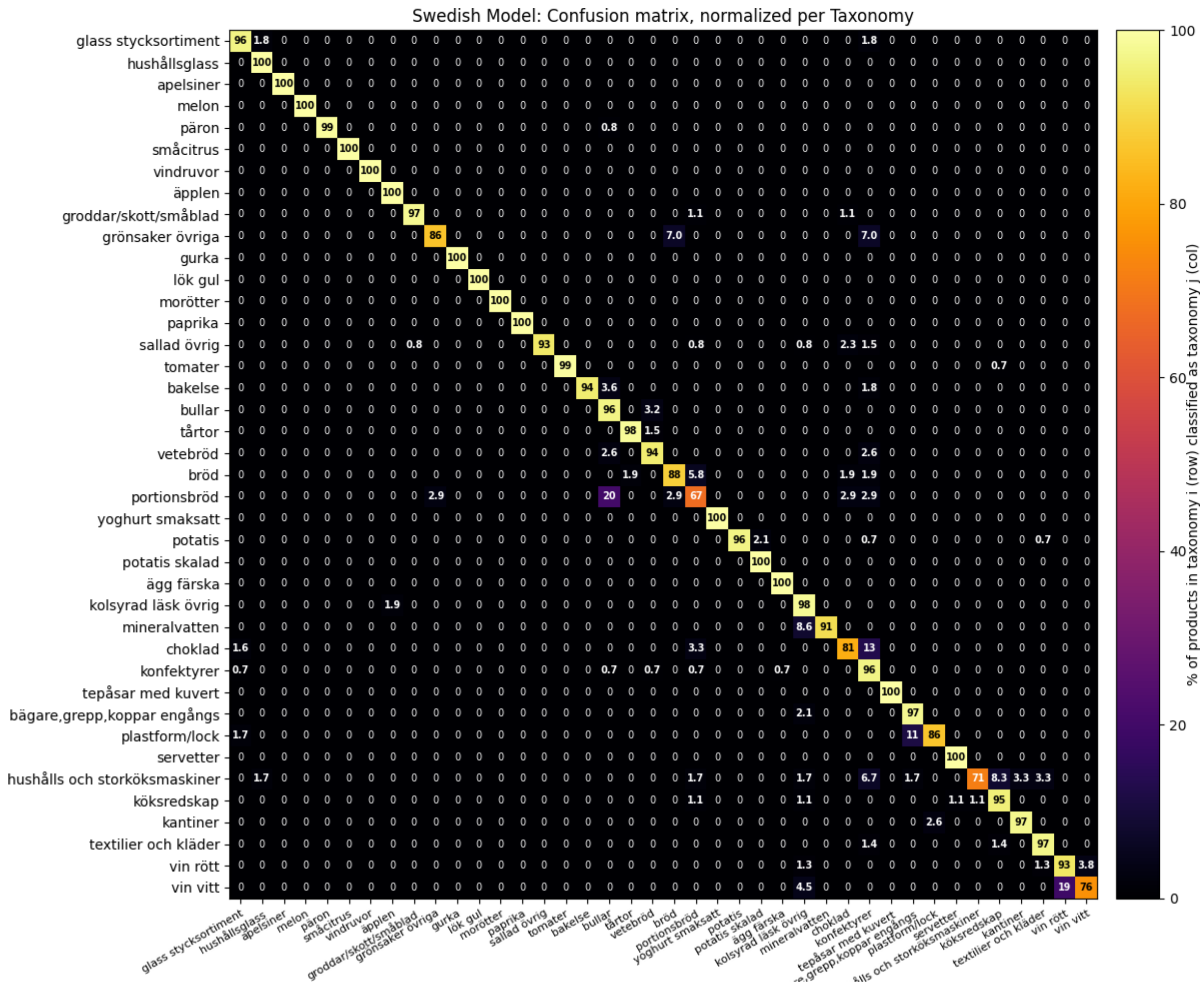


Figure 4.2: Normalized Confusion Matrix for the **Swedish Model**, normalized on the number of total products actually within each taxonomy (normalized row-wise). Each cell shows the percentage of products that belonged to taxonomy *i* (row *i*), that were predicted to belong to taxonomy *j* (column *j*).

4.2 Effects of Reduced Training Data

I made an investigation into how well the models performed when training on different amounts of training data. After training on different percentages of training data as explained in **Section 3.3**, I analyzed the data using three plots. Firstly I will discuss the scatter plots, where the F1-score of each taxonomy was plotted against its training support. These metrics were obtained from the two models being trained on 1%, 5%, 10%, 20%, 50% and 100% of the training data. The metrics were grouped together according to these training data proportions and displayed on a figure for each model (*mBERT* and *KB-BERT*).

The scatter plots for the multilingual and the Swedish model are shown in **Figures 4.3** and **4.4**, respectively. As expected, F1-score generally increases as training support increases. However, there is vast variation: some taxonomies still achieve F1-scores above 0.9, even with as little as 3-5 training examples.

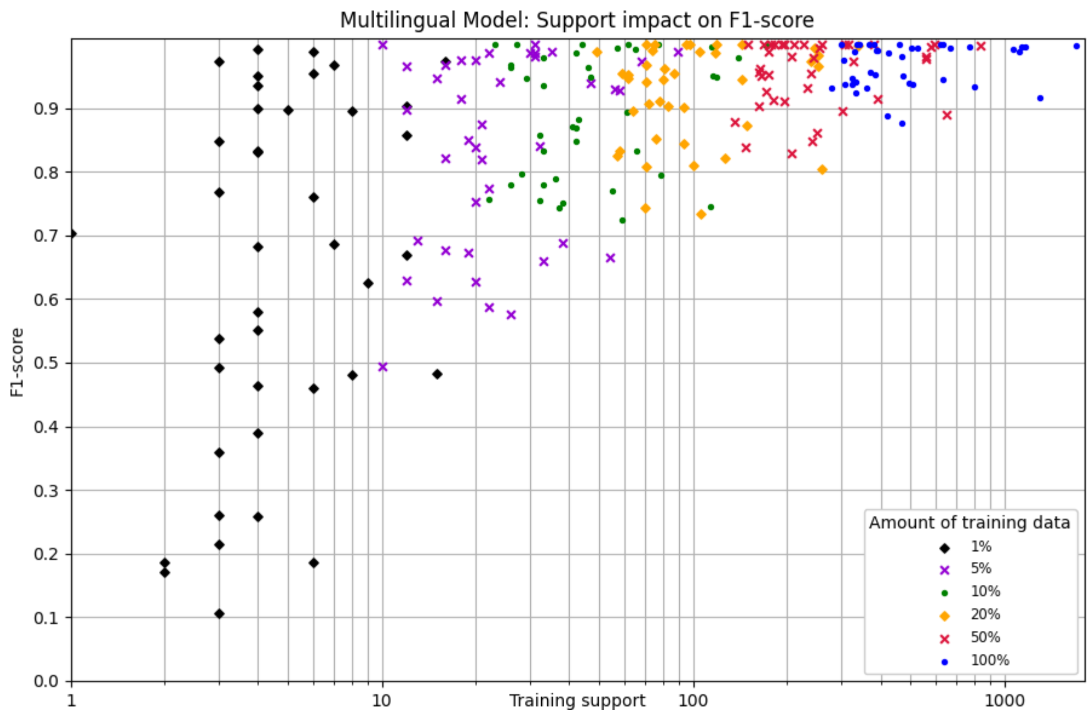


Figure 4.3: Training data support impact for the **Multilingual model**, trained with 100%, 50%, 20%, 10%, 5% and 1% of the training data. Each point represents the F1-score and training data support (amount of products belonging to a taxonomy in the training data) of a single taxonomy. Horizontal axis is scaled logarithmically for readability.

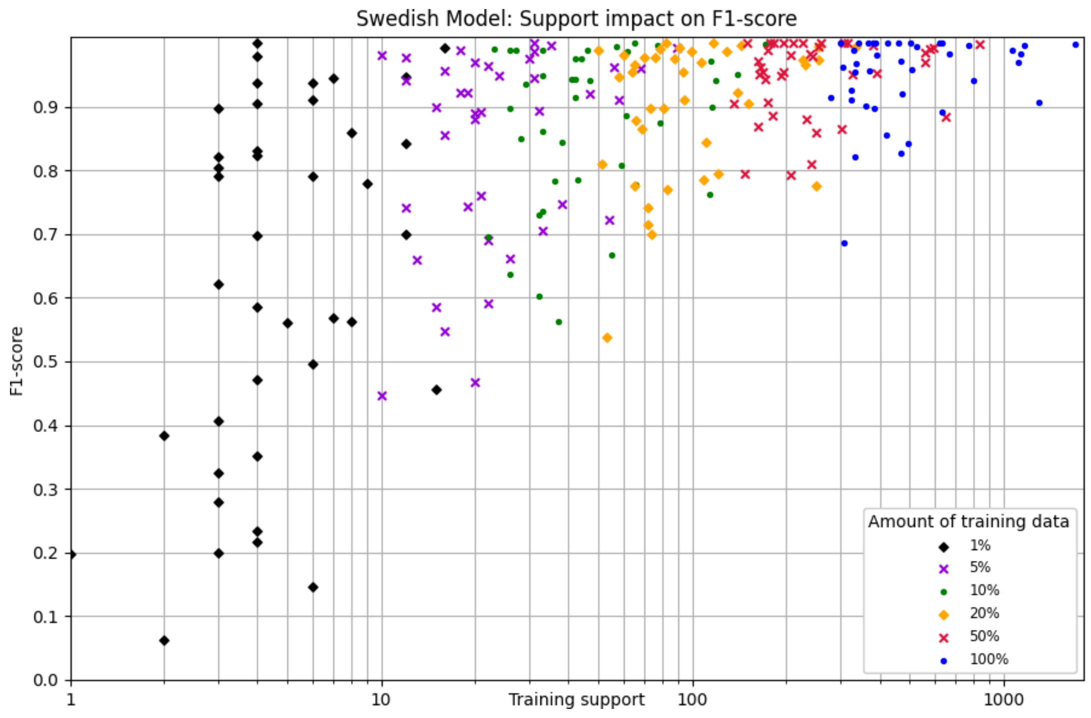


Figure 4.4: Analogous graph to Figure 4.3, for the **Swedish Model**.

To further illustrate this disparity, I plotted a line graph for each taxonomy across the 6 levels of training support. These graphs, shown in **Figure 4.5** and **4.6** respectively, are near illegible on a taxonomy-by-taxonomy basis, but it is clearly visible that the amount of training support impacts the F1-score very differently for different taxonomies.

4.2.1 Impact of Training Support for each Taxonomy

To clearly visualize the effects of reduced training support for each taxononomy's performance in the model, I generated a final pair of graphs that plot the difference in F1-score for each taxonomy when using 100% versus 1% of the training data, which are shown in **Figure 4.7** for the Multilingual model and **Figure 4.8** for the Swedish model.

These graphs thus give a sense of how much training support affects the F1-score of a particular taxonomy, which will be denoted as *sensitivity* or *sensitivity to training support*. In these graphs, taxonomies were colored in groups of their *varuområde* to aid visual detection of any patterns across taxonomies. An exception to this was made for the taxonomies in *färskvaror/kylvaror*, which was thus further split and coloured according to the *huvudgrupp* of each taxonomy, as *färskvaror/kylvaror* contained a very large majority of taxonomies.

Looking at these graphs, we can see there is not necessarily a hard split between the taxonomies in regards to their training support sensitivity. Sorted by increasing F1-diff, there is a near-linear increase in sensitivity across the various taxonomies in both models. However, there are a few observable phenomenon. The following taxonomies and *varuområde* are on the lower side of sensitivity to training support in both models, gaining little improvement when increasing training support. They also achieve high F1-scores.

- *yoghurt smaksatt*

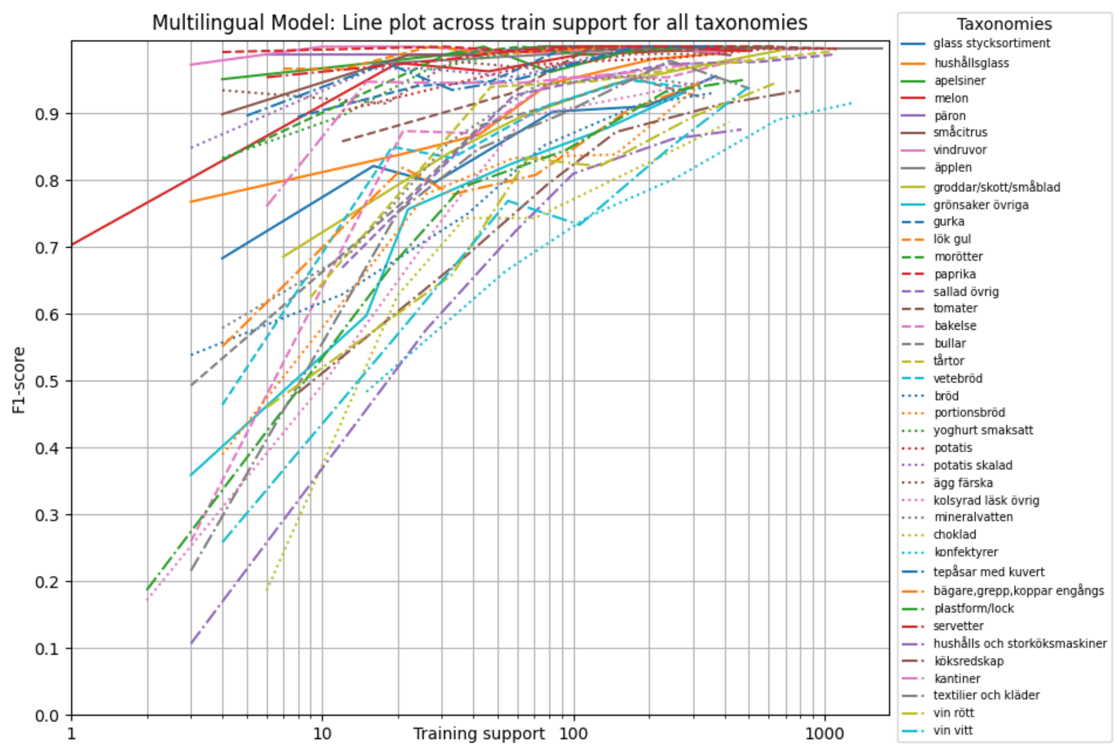


Figure 4.5: Taxonomy F1-support graph for the **Multilingual model**, where each line is a taxonomies F1-score plotted over its training support from 1% to 100% of the training data.

- The majority of taxonomies in *färskvaror/kylvaror* that represent a singular type of vegetable/fruit, such as *paprika*, *päron*, *potatis*, etc; distinct from taxonomies that are groupings of product categories such as *grönsaker övriga* and *sallad övrig*,

- The *varuområde djupfryst*,

- The taxonomy *servetter*, in contrast to the other taxonomies in the *varuområde non food* that are more sensitive training support.

The multilingual and Swedish models both share a few taxonomies and *varuområden* that require a lot of training data to perform well, such as:

- The taxonomy *kolsyrad läsk övrig*,

- The taxonomy *hushålls och storköksmaskiner*,

- The *varuområde vin/sprit/öl*,

- Most taxonomies in the *huvudgrupp kaffebröd/konditori*, under the *varugrupp färskvaror/kylvaror*,

- Most taxonomies in *kolonial/speceri*.

However, there are a few taxonomies, *varuområden* and *huvudgrupper* that behave very differently in the two models. The most pronounced differences are:

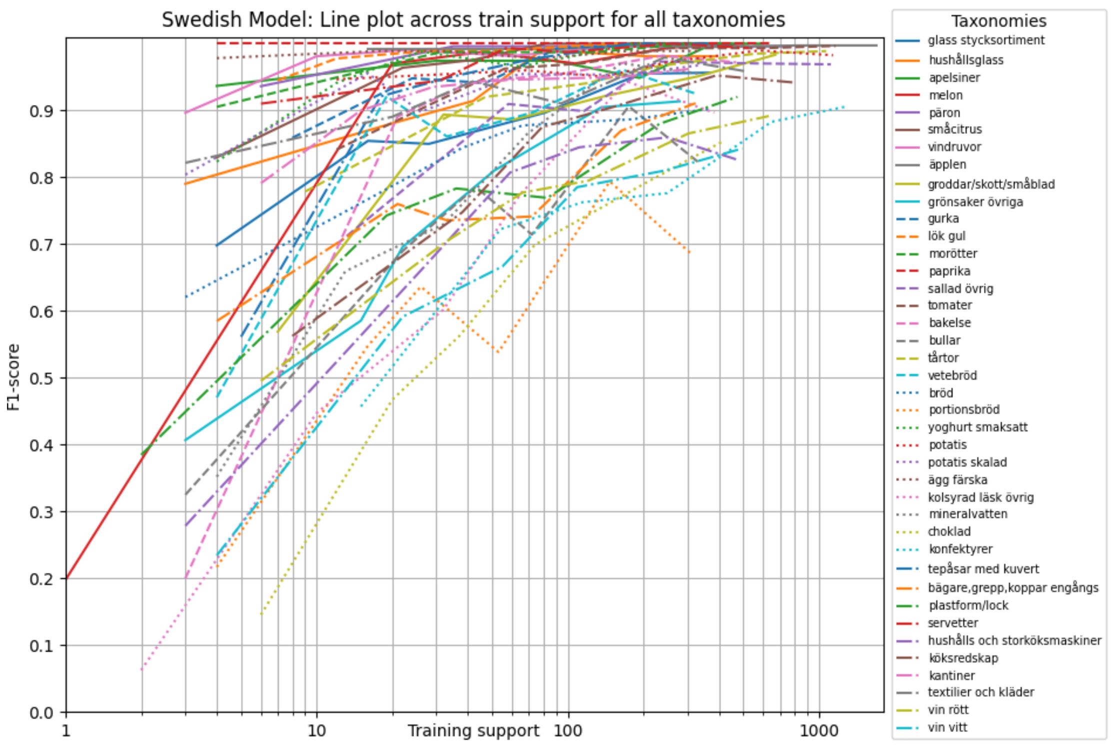


Figure 4.6: Taxonomy F1-support graph for the **Swedish model**, where each line is a taxonomies F1-score plotted over its training support from 1% to 100% of the training data.

- *melon*, which has an F1-diff of ~0.8 in the Swedish model but is under ~0.3 in the multilingual model;
- The *varuområde* of *restaurangutrustning*, which overall is less sensitive to training support in the Swedish model for all taxonomies;
- The taxonomy *tepåser med kuvert* has a low F1-diff of ~0.1 in the multilingual model, whereas it is almost 0.45 in the Swedish model.

While more training data overall improves the F1-score across all taxonomies of the model, there is immense variation between taxonomies in much improvement is made, and how much training data is required to achieve high F1-scores. Some taxonomies require over 1000 training examples to achieve an F1-score over 0.9, while some require just 3-5. Overall, requirements of training support seem mostly inherent to the taxonomies themselves and are mostly not affected by model selection.

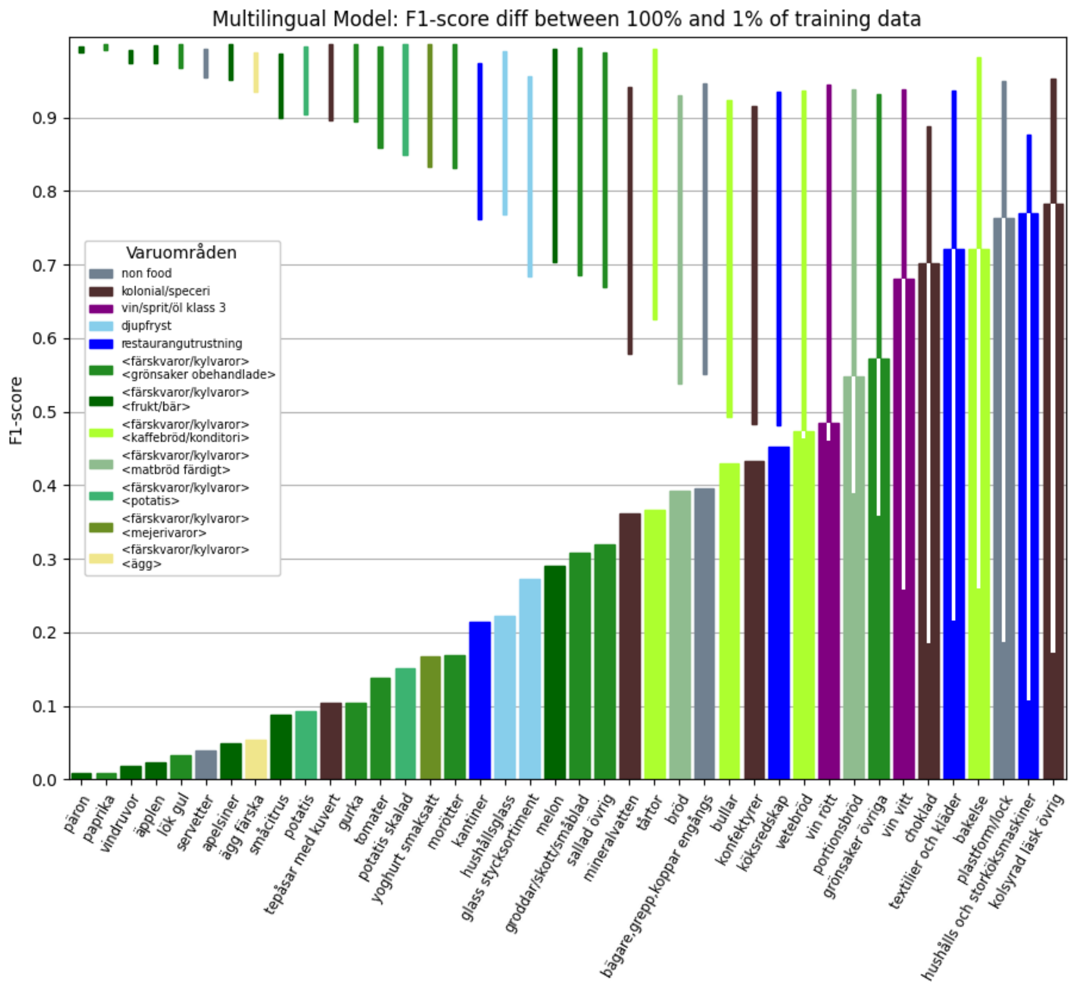


Figure 4.7: Difference in F1-score between models trained on 100% and 1% of training data, for the **Multilingual model**. Thick bars show the absolute difference in F1-score. Thinner bars are the actual F1-score ranges from 100% to 1% of training data. Taxonomies are grouped in color according to their *varuområde*. Taxonomies in the *varuområde färskvaror/kylvaror* are split into subcategories according to their *huvudgrupp*, as the majority of taxonomies were under *färskvaror/kylvaror*.

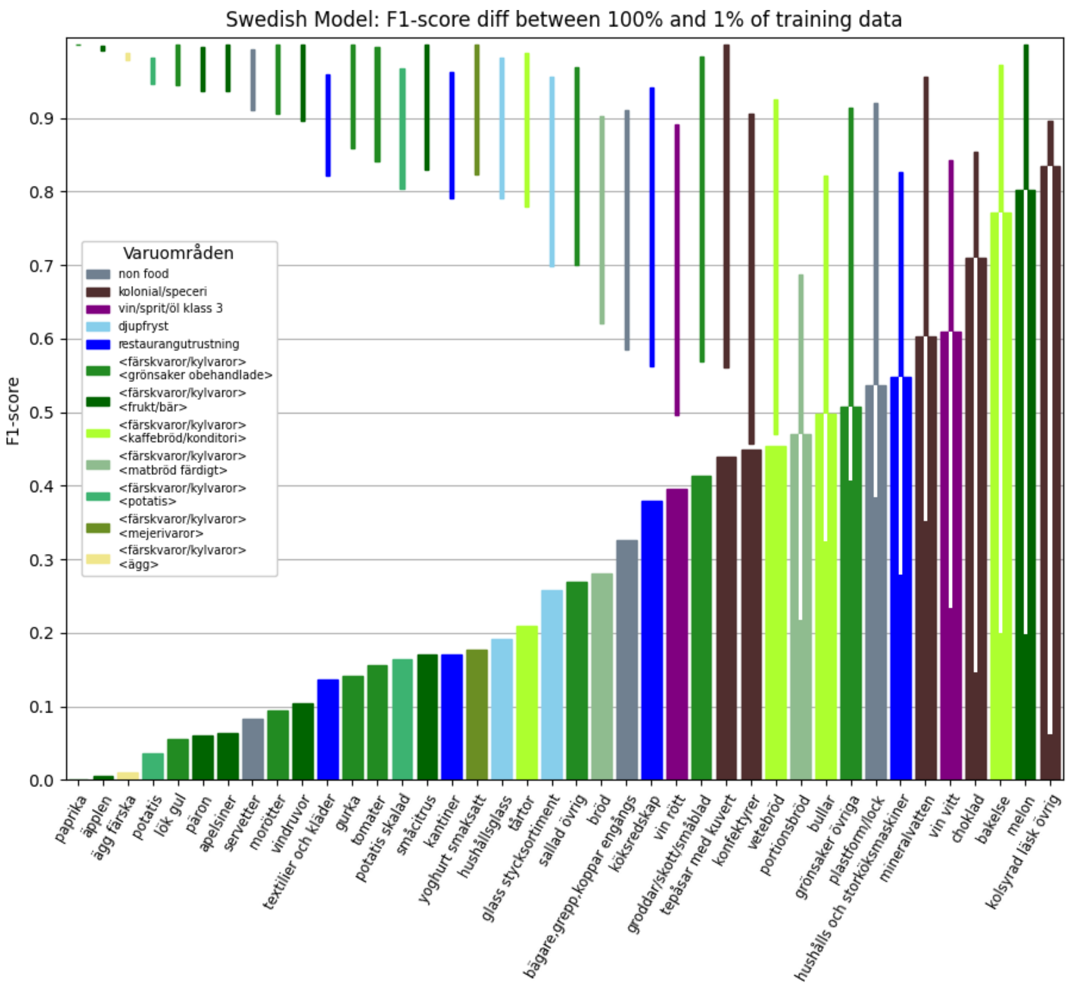


Figure 4.8: Analogous graph to Figure 4.7, for the **Swedish model**.

Chapter 5
Discussion

5.1 Reflections on Results

5.1.1 Overall Model Performance

Overall, both models produced very good results. A majority of taxonomies achieved F1-scores over 95% in both models. However, contrary to other experiments reported by de Vargas Feijo and Moreira (2020), the multilingual model outperformed the Swedish one slightly overall, as measured by the macro-average F1-score. While there are specific cases where the difference in performance appears counterintuitive, which are discussed below, a possible explanation for the slight advantages the multilingual model holds in a majority of the taxonomies could be that names of food products occasionally contains words and abbreviations from different languages, such as English or French.

The performance of the Swedish model was mostly deteriorated by systemic misclassification of products belonging to a select group of taxonomies, as seen in **Section 4.1.2**. Similarity in product names and taxonomies among these seems to be a reasonable explanation for this phenomenon; it is easy to imagine that it is hard for the models to separate certain types of products when solely using the product name, as in the case of:

- Wine (*vin rött - vin vitt*)
- Some chocolates (*choklad*) and other generic sweets (*konfektyrer*) that may be chocolate flavoured,
- Differentiating kitchen equipment products (the taxonomies in the *varuområde restaurangutrustning*).

However, this reasoning clashes with the fact that the Multilingual model did not suffer from systemic misclassification in the same capacity among these taxonomies. It is possible that the source material of the multilingual model is more comprehensive in areas closely related to these taxonomies. For example, it is likely that there are a lot of Wikipedia articles on wine and wine-related products in non-Swedish languages, as many are internationally produced and usually only parts of their product names are translated, if at all. The multilingual model, which uses Wikipedia of several languages as its corpus, would then be better at understanding and producing embeddings for these products. Furthermore, names of wine products tend to also be in a variety of languages, which would explain why the *vin rött* and *vin vitt* taxonomies perform better; the multilingual model can utilize its powerful embeddings in this area. In contrast to the international corpus of the multilingual model, a vast majority of the Swedish model's corpus is based on newspaper articles dating back to the 1940's (Malmsten et al., 2020).

Simultaneously, this does not explain the multilingual model's large performance advantage for the taxonomy *portionsbröd* relative to the Swedish model, especially as a majority of those products are either endemic to Sweden or are described with Swedish product names. Given that the Swedish Model was trained on a large and varied Swedish corpus, mentions of *portionsbröd* products such as *fralla*, *rågkaka* and *brytbröd*, as well as *bullar* products like *kanelbulle* and *gifflar*, should reasonably be present, for example in newspaper articles. Product names among these taxonomies should not have been misclassified as often as they were, following the line of reasoning for the Swedish model's poor performance on *vin rött* and *vin vitt* products.

The source code of the trainers facilitated by *Flair* and *HuggingFace* were read and surgically modified where applicable to make the training settings and procedures as similar as possible. However it is still possible that the training procedures still contain differences that were missed, causing the two models to train differently.

In summary, for the majority of taxonomies where a substantial difference in performance exists, the root cause of the difference is unclear and warrants further investigation.

5.1.2 Variability in Performance on Reduced Training Data

As presented in **Section 4.2**, there is enormous variation in how much training data is required to achieve high F1-scores. A highly plausible explanation is that taxonomies with little variability in their product names require much less training data.

For example, most food products that are vegetables or other raw food goods, such as *potatis*, *ägg*, *paprika*, *gul lök*, are all probably named similarly, whereas a taxonomy like *kolsyrad läsk övrig* (soft drinks) probably contains a large and diverse pool of product names. Cartons of onions produced by different manufacturers are probably are probably named very similarly, whereas an orange-flavored soft drink can be named very differently, such as *Zingo*, *Fanta*, *Jaffa*, *Mirinda Orange*, etc.

This would explain the training support sensitivity for a majority of taxonomies, but there are cases where the issue seems to stem from other factors; the taxonomy *melon* improved from an F1-score of ~0.2 to 1.0 in the Swedish model when using 1% versus 100% of the training data, whereas the multilingual the range of improvement was only from ~0.7 to 0.9939 in F1-score. In this specific case, it is possible that the multilingual model is equipped with a more nuanced understanding of melons and varieties of them, as it was trained on a large corpus of Wikipedia articles.

5.2 Future Improvements

5.2.1 Different Definition of Taxonomy

In this thesis, the construction and definition of the selected taxonomies led to very fine-grain classes. Some taxonomies were extremely specific, as in the case of most taxonomies in the *huvudgrupp grönsaker obehandlade* and *frukt/bär*, in contrast to more general taxonomies such as *kolsyrad läsk övrig*. It would be interesting to see how the performance of these models would be affected if trained on a lower granularity of taxonomy definition, utilizing only *varuområde* and *huvudgrupp*, for example. Alternatively, only taxonomies that share cases of systemic misclassification could be merged into a broader taxonomy, retaining fine-grained taxonomies with good performance where possible. My hypothesis is that it would perform better, as most cases of systemic misclassification were within the same *huvudgrupp*.

5.2.2 Hyperparameters

For this thesis, hyperparameters were not tuned or experimented with. Some proposed changes would be to modify the learning rate scheduler to reload the best previous model when annealing the learning rate, as there was a couple of cases where the the learning rate was continually annealed with no improvements, possibly having "jumped over" the local minimum of the loss function. This may however just be a limit of the model's generalizable performance and would instead lead to overfitting on the evaluation dataset. Another suggestion is to use more recent optimizers such as AdamW (Loshchilov and Hutter, 2019) or AdaBound (Luo et al., 2019) in place of the SGD optimizer, which supposedly converge faster and might provide better results than the current setup.

5.2.3 Model Selection

In this thesis, only *BERT-Base* models were used. More models could be added to see if better performance is possible, for example with *BERT-Large* models or *MT5* (Xue et al., 2021), which used a vastly larger corpus during pretraining. Furthermore, comparison with a baseline text classifier should be included. A suitable candidate for such a baseline would be a pipeline with TF-IDF vectorization of product names sent to a logistic regression classifier.

5.2.4 Inspection of Differences in Support Sensitivity

Some sort of measure of product name variability should be calculated and compared with the F1-sensitivities shown in **Figures 4.7** and **4.8**, perhaps using some sort of edit-distance algorithm. A good suggestion would be the Jaro-Winkler distance (Winkler, 1990), since it weights string similarity with an emphasis towards the start of strings which applies well to how product names are often formulated, both specifically in Mashie *products document* and generally for product names.

5.2.5 Minor Improvements and Modifications

As described in **Section 3.2.1**, all products were split into training, evaluation and test datasets before unrelated products (which did not belong to the top 40 selected taxonomies) were removed. These steps should have been reversed to ensure a proper 80%/10%/10% split, but given the large number of products, along with the fact that products were separated into these datasets fully randomly, the impact on the results in this thesis should be insignificant.

Similarly, training set reduction was not explicitly done proportionally on a taxonomy-by-taxonomy basis, but instead by removing a random product until the training set in its whole was of the desired size. This may result in certain taxonomies not containing strictly x% of the number of products that existed for the taxonomy in the full training dataset. However, given the large number of products in the training set, any effect of this should also be insignificant.

References

Abduljabbar, D. and Omar, N. (2015). Exam questions classification based on bloom's taxonomy cognitive level using classifiers combination. *Journal of Theoretical and Applied Information Technology*, 78:447–455.

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. (2019). FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, Minneapolis, Minnesota. Association for Computational Linguistics.

Baker, S., Silins, I., Guo, Y., Ali, I., Högberg, J., Stenius, U., and Korhonen, A. (2015). Automatic semantic classification of scientific literature according to the hallmarks of cancer. *Bioinformatics*, 32(3):432–440.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364.

de Pelle, R. and Moreira, V. (2017). Offensive comments in the brazilian web: a dataset and baseline results. In *Anais do VI Brazilian Workshop on Social Network Analysis and Mining*, Porto Alegre, RS, Brasil. SBC.

de Vargas Feijo, D. and Moreira, V. P. (2020). Mono vs multilingual transformer-based models: a comparison across several language tasks. *arXiv:2007.09757 [cs.CL].*

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). 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. Association for Computational Linguistics.

Glosser.ca (2013). Artificial neural network with layer coloring. (`https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg` Accessed: 2021-06-10).

Google (2018). Bert-base, multilingual cased. (`https://github.com/google-research/bert/blob/master/multilingual.md` Accessed: 2021-04-28).

Gururangan, S., Marasovic, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., and Smith, N. A. (2020). Don't stop pretraining: Adapt language models to domains and tasks. *CoRR*, abs/2004.10964.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Kalra, J., Batra, D., Diwan, N., and Bagler, G. (2020). Nutritional profile estimation in cooking recipes. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*, pages 82—-87. IEEE.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942.

Lewis, P., Ott, M., Du, J., and Stoyanov, V. (2020). Pretrained language models for biomedical and clinical tasks: Understanding and extending the state-of-the-art. In *Proceedings of the 3rd Clinical Natural Language Processing Workshop*, pages 146–157. Association for Computational Linguistics.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations*. ICLR.

Luo, L., Xiong, Y., Liu, Y., and Sun, X. (2019). Adaptive gradient methods with dynamic bound of learning rate. In *Proceedings of the 7th International Conference on Learning Representations*. ICLR.

Malmsten, M., Börjeson, L., and Haffenden, C. (2020). Playing with Words at the National Library of Sweden – Making a Swedish BERT. *arXiv:2007.01658 [cs.CL]*.

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

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, volume 14, pages 1532–1543.

Popovski, G., Kochev, S., Seljak, B., and Eftimov, T. (2019). Foodie: A rule-based named-entity recognition method for food information extraction. In De Marsico, M., Sanniti di Baja, G., and Fred, A., editors, *The International Conference on Pattern Recognition Applications and Methods*, volume 8, pages 915–922. SciTePress.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS.

Warstadt, A., Singh, A., and Bowman, S. R. (2018). Neural network acceptability judgments. *CoRR*, abs/1805.12471.

Wiegand, M., Roth, B., and Klakow, D. (2014). Automatic food categorization from large unlabeled corpora and its impact on relation extraction. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 673–682, Gothenburg, Sweden. Association for Computational Linguistics.

Winkler, W. E. (1990). String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. In *Proceedings of the Section on Survey Research Methods*, page 354–359. American Statistical Association.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., and Raffel, C. (2021). mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv:2010.11934 [cs.CL]*.

Appendices

Appendix A
Figures

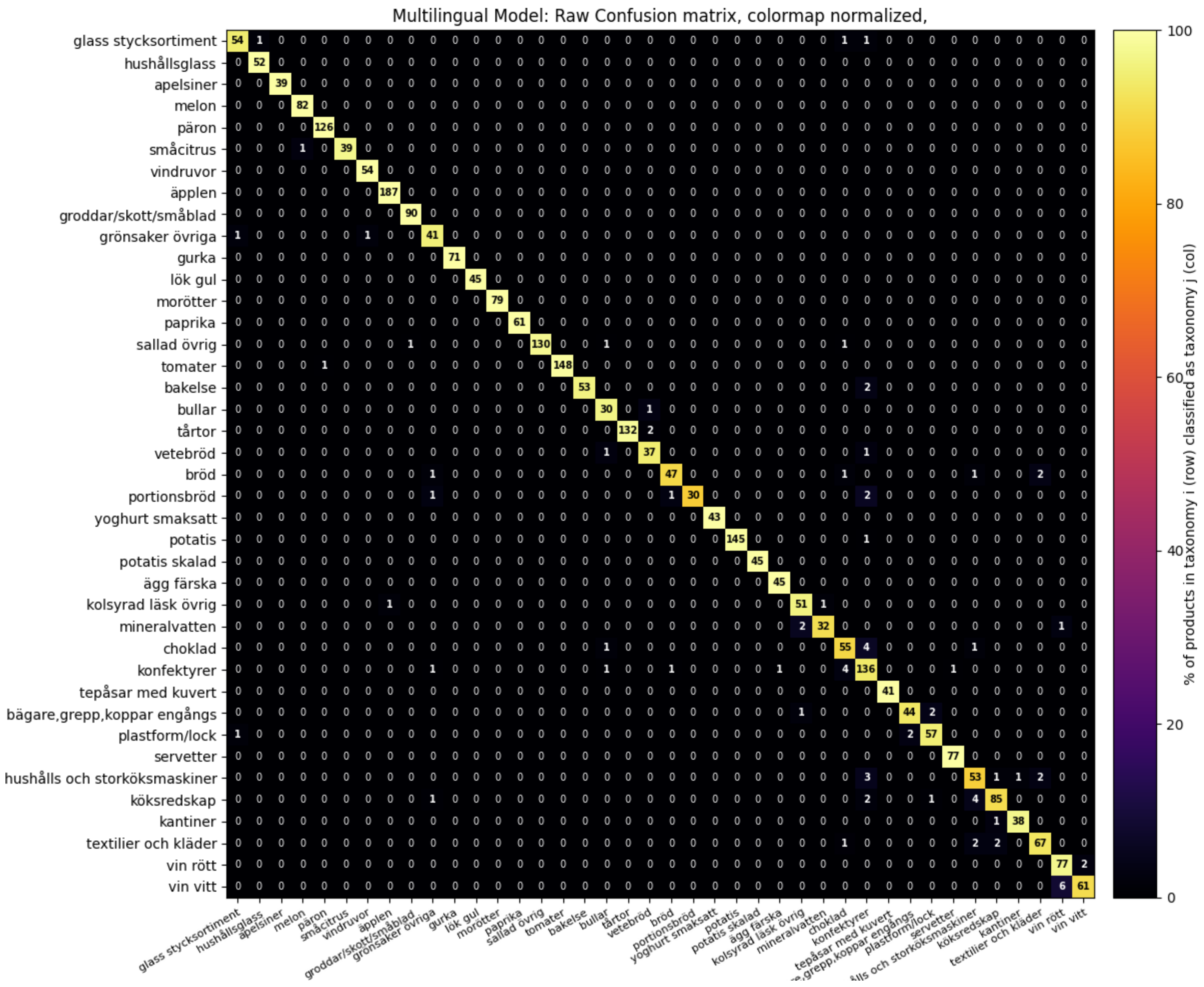


Figure A.1: Raw Confusion Matrix for the **Multilingual Model**, with the same color map as in Figure 4.1.

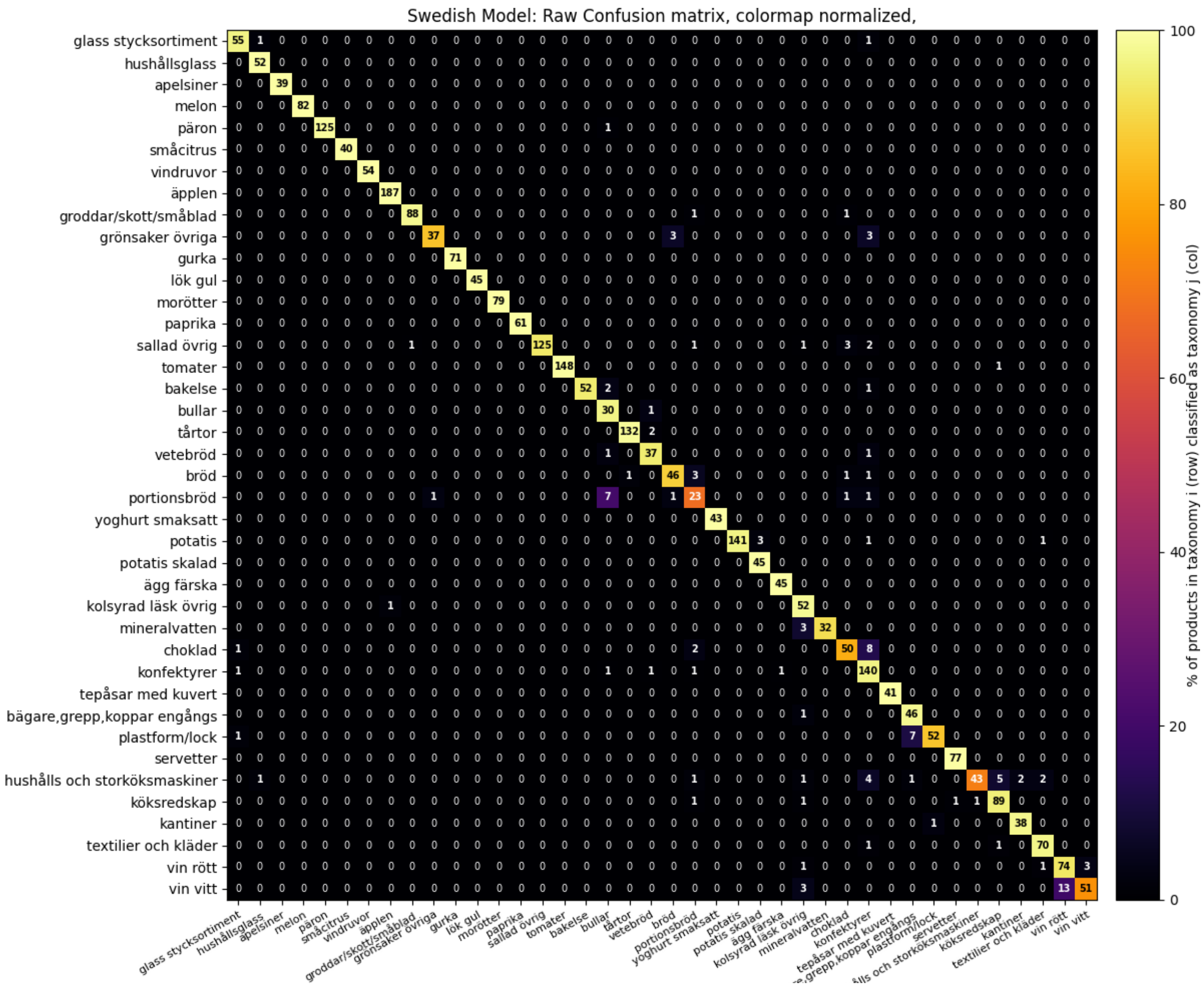


Figure A.2: Raw Confusion Matrix for the **Swedish Model**, with the same color map as in Figure 4.2.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2021-06-09

EXAMENSARBETE Automatic Categorization of Food Products
STUDENT Vilhelm Lundqvist
HANDLEDARE Pierre Nugues (LTH), Johnny Zackrisson (Mashie)
EXAMINATOR Jacek Malec (LTH)

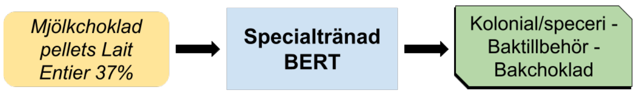
Automatisk kategorisering av livsmedelsprodukter utifrån produktnamn

POPULÄRVETENSKAPLIG SAMMANFATTNING **Vilhelm Lundqvist**

Företaget Mashie arbeterar med livsmedelsprodukter samt lägger stora resurser på att hantera och dokumentera data från dessa. Detta arbete undersöker om en vara kan kategoriseras rätt enbart utifrån dess produktnamn med hjälp av språkmodellen BERT.

Företaget Mashie arbetar med IT-infrastruktur till måltidsbranschen som exempelvis restauranger. Detta kräver en aktuell och strukturerad databas av livsmedelsartiklar. I nuläget uppdateras databasen manuellt av experter.

I mitt examensarbete undersöks möjligheten till automatisk kategorisering av livsmedel genom en specialtränad BERT-modell. BERT är en avancerad språkmodell som förhandstränas på enorma mängder obehandlad text. Med sin inbyggda språkförståelse fungerar den väldigt väl som ett mellansteg i textanalys då den anpassas till uppgiften genom specialiserad träning.


Mjölkchoklad pellets Lait Entier 37%
Specialtränad BERT
Kolonial/speceri - Baktillbehör - Bakchoklad

På materialet tillämpades två varianter av BERT; en förhandstränad på alla Wikipedia-artiklar skrivna på de 104 största språken *(mBERT)* och en tränad på svenska texter från Kungliga Bibliotekets arkiv *(KB-BERT)*. Modellerna specialtränades att läsa namnen på livsmedelsprodukter och placera dem i fördefinierade kategorier, såsom *"Kolonial/speceri - Kolsyrade drycker - Läsk"*. På grund av hårdvarubegränsingar utvärderades modellerna på de 40 största kategorierna. Deras beteende undersöktes ytterligare genom att tränas på reducerade mängder data.

F1-score är ett mått som mäter modellens pricksäkerhet och "gissningsvilja". Genom F1-score uppmättes prestandan av *mBERT* och *KB-BERT* till 96.76% respektive 94.73%. Majoriteten av kategorierna uppnår höga resultat. Ett fåtal kategorier har sämre prestanda där produkter tenderade att placeras i varandras kategorier. Exempelvis tror modellen ibland att produkter är röda viner när de egentligen är vita viner. Främst *KB-BERT* drabbas av detta vilket är den särskilt bidragande faktorn till modellens lägre prestanda.

Kategorierna visade sig dessutom kräva en mycket varierande mängd data för att ge bra resultat. Båda modellerna behövde tränas på 3 produktnamn av typen vindruvor för att ge >90% F1-score samtidigt som nästan 1000 konfektyrprodukter krävdes för att uppnå samma resultat inom sina respektive kategorier.