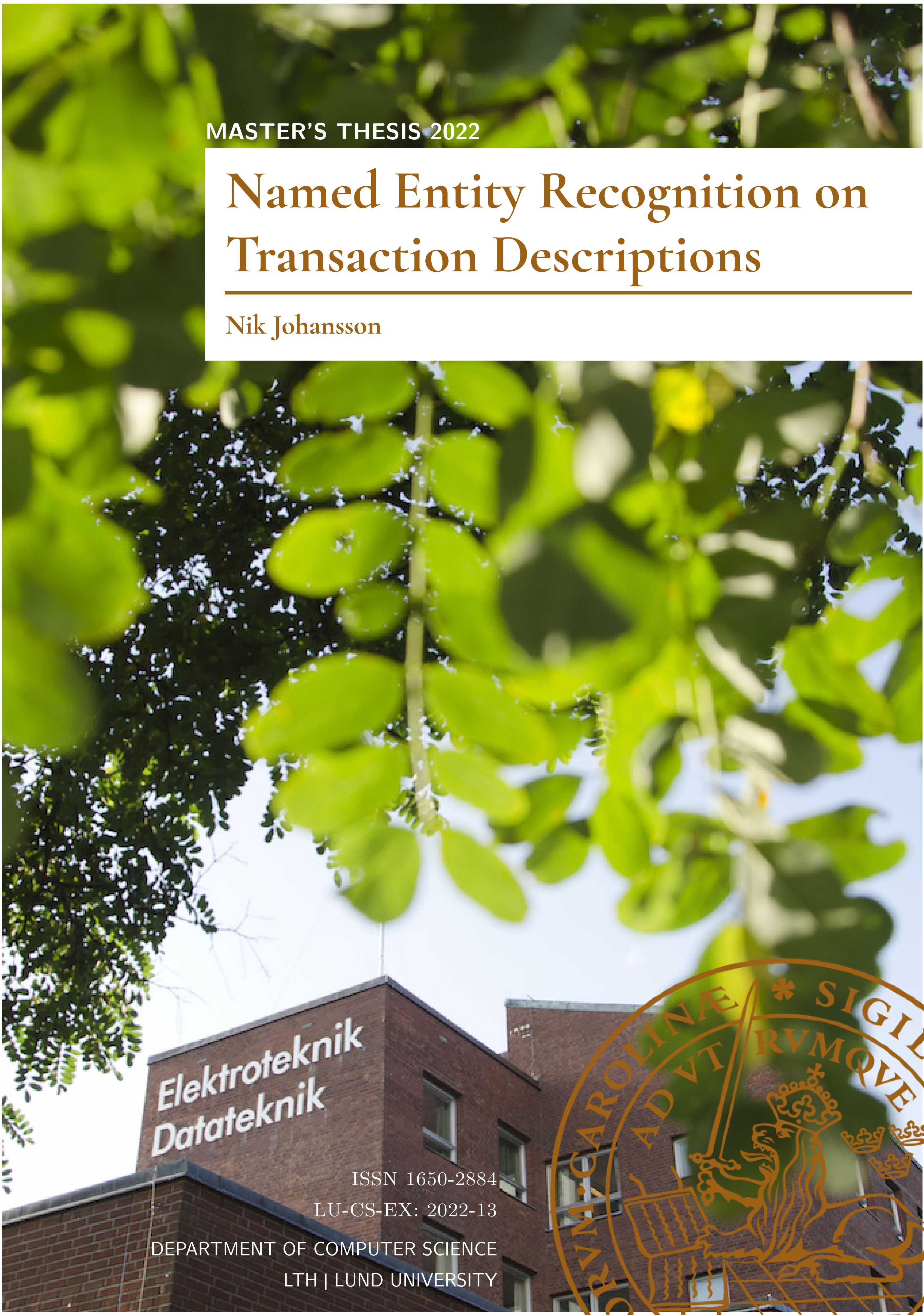
MASTER'S THESIS 2022

Named Entity Recognition on Transaction Descriptions

Nik Johansson



ISSN 1650-2884
LU-CS-EX: 2022-13

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-13

Named Entity Recognition on Transaction Descriptions

Named Entity Recognition på Transaktionsbeskrivningar

Nik Johansson

Named Entity Recognition on Transaction Descriptions

Nik Johansson
ni5812jo-s@student.lu.se

March 30, 2022

Master's thesis work carried out at Tink AB.

Supervisors: Trent Woodbury, trent.woodbury@tink.com
Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Marcus Klang, marcus.klang@cs.lth.se

Abstract

With the surge of open banking, there is a large increase in applications based on transaction data. Therefore, there is a need for being able to extract important information from Swedish transaction descriptions in a structured way.

We designed models for named entity recognition on transaction descriptions that can identify and classify organizations, locations, persons, payment providers (e.g Swish, Klarna, or PayPal) and products/apps. With our best NER model, we reached a chunk F1 score 0.849, despite only using 2200 transactions for training and transaction descriptions being messy. This is, to the best of our knowledge, the first published report on named entity recognition for transaction descriptions.

Finally, we used our named entity recognition model and its output to complement a commercial transaction categorization system and improve the performance of the two existing models by 7.4% and 1.1% respectively.

Keywords: named entity recognition, natural language processing, financial transactions, transaction categorization, open banking, transformers

Acknowledgements

I would like to thank Trent Woodbury and the rest of the Enrichment Categorization team at Tink for taking me on as a thesis student and supplying me with necessary equipment, data, and support.

I would also like to thank Manning Publications for allowing me to use some figures from *Deep Learning with Python* (Chollet, 2017, 2021).

And a special thanks goes out to Pierre Nugues for his guidance and support during this period.

Contents

Chapter 1

Introduction

1.1 Digital Transactions

Financial transactions are something we have all encountered. A transaction is the exchange of money for a product or a service, for example, when we go to a clothing store to buy a piece of clothing. When the customer makes a payment at the merchant, the transaction is then sent to her/his bank or can make its way through a third party provider (TPP), as shown in Figure 1.1.

The transaction can contain different things, but most importantly, it contains a text description and an amount. When the transaction arrives at the bank and/or the TPP, it needs to be processed. The processing can, for example, be registering the transaction, updating balances, or it can be processing to improve user experience, such as cleaning the description or enriching the transaction to provide insight to the consumer. Table 1.1 shows one example of processing, where an original description and amount comes in through a merchant. The description is then cleaned and the transaction is sent through a categorization model to predict what type of purchase has occurred.

Original description	**Formatted description**	**Amount**	**Category**
Sobares Ibiza 1439,PAS284 IBIZA	Sobares Ibiza	-83.46	Restaurant

Table 1.1: An example of transaction processing. *Original description* is the transaction description sent by the merchant, *Formatted description* is the transaction description after cleaning, *Amount* is the cost of the purchase, and *Category* is the category that the transaction categorization model predicted.

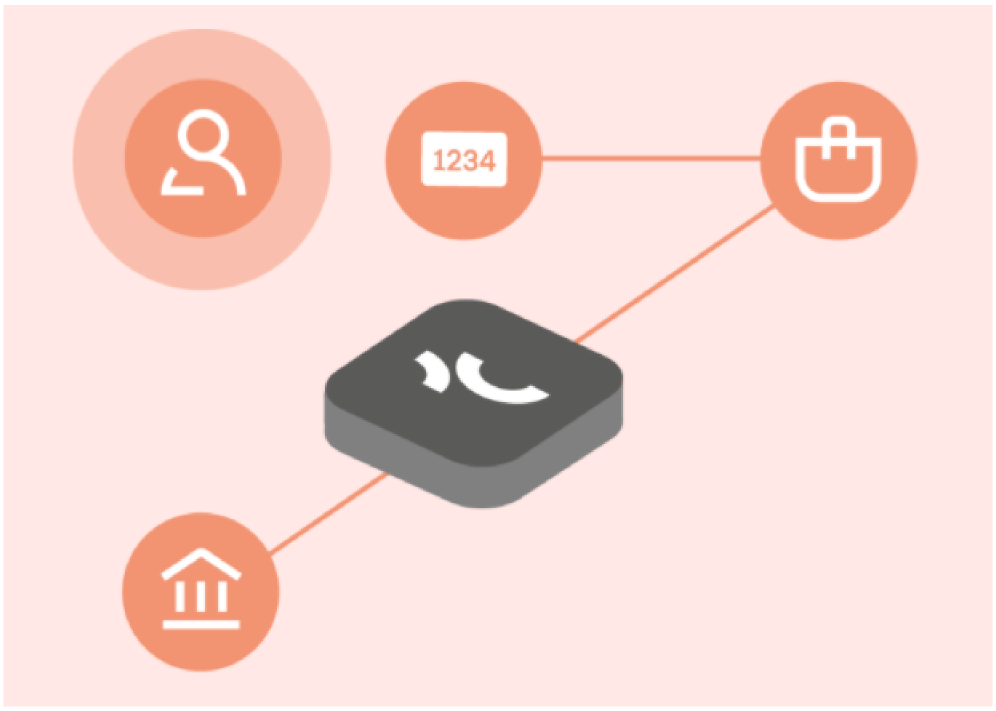


Figure 1.1: The transaction flow of a person making a purchase at a merchant. The transaction is then sent through a third party provider, in this case Tink, and onward to the bank.

1.1.1 PSD2 Regulation and Open Banking

The Revised Payment Service Directive, known as PSD2, is a directive passed at the end of 2015 in the European Union (Schulz and Schmit, 2015). This directive shifted the ownership of some financial data in general and transaction data in particular, from the banks to the consumer. This means that third party providers can now get access to transaction data upon customer consent, which in later years has created a surge in the use of transaction data as well as products, applications, and innovation based on this data. This is commonly referred to as *open banking*, and Figure 1.2 shows an illustration of this.

1.2 Tink

I carried out this Master's thesis at Tink, a company operating within open banking and the PSD2 sphere. Tink is a leading open-banking platform in Europe (Tink, 2022) and has partnered with many of the Swedish and Scandinavian financial institutions. The company provides infrastructure to fetch data through PSD2 APIs and develops applications based on this data, mainly focused on creating tools and providing insight for users, to make it easier for them to make good financial decisions. One of its flagship products is a tool for categorizing transactions and presenting them to provide better knowledge of one's expenses.

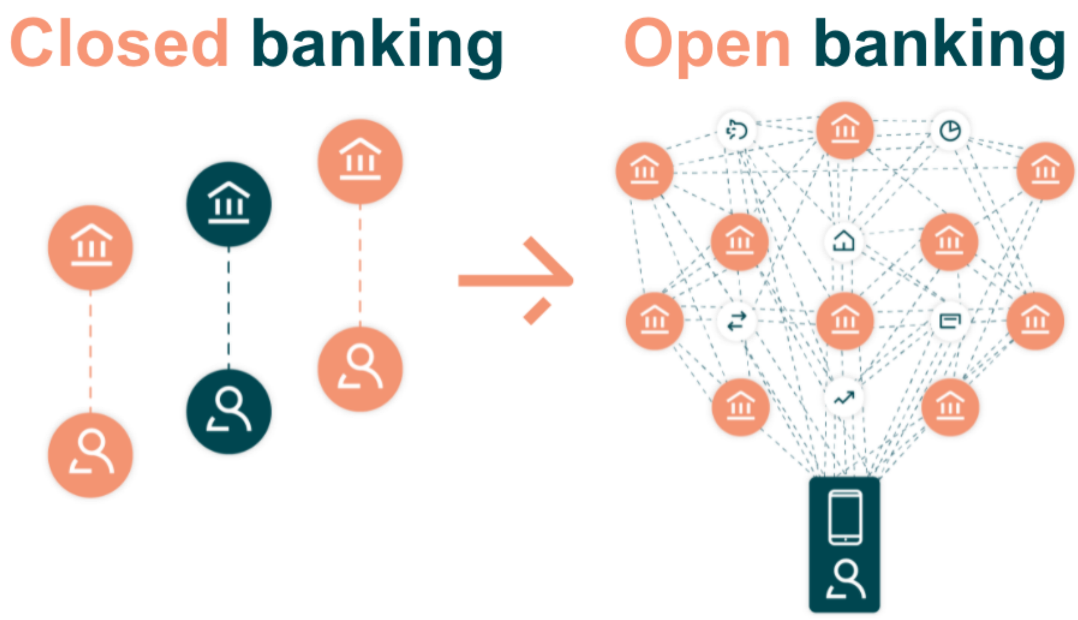


Figure 1.2: Illustration of open banking. On the left part of the figure we see one person being connected to one financial institution, while on the right part the financial institutions as well as the user, are connected in a common network.

1.3 Named Entity Recognition

In Table 1.1, the description contains the names of a restaurant, *Sobares*, and its location, *Ibiza*. Organizations, locations, but also persons and similar are often the most important information in text in general, and transaction descriptions specifically. Extracting this information in a structured way can therefore be valuable for a wide variety of use cases.

Named entity recognition (NER) is a subdomain in natural language processing (NLP), which is the task of identifying and classifying named entities in text. Named entities correspond more or less to *proper nouns*, i.e entities that we can identify with their name, like *Sobares* or *Ibiza*, as opposed to *common nouns*, such as *restaurant* or *island*, which denote classes of entities. This task can sometimes also be extended to temporal expressions and number expressions (Grishman and Sundheim, 1995b,a).

The NER task was first introduced at the *Message Understanding Conference 6* (muc, 1995) in the United States and focused on extracting important information in text. At this time, the tasks were mainly centered around company and defense related activity, with the Defense Advanced Research Projects Agency (DARPA) being heavily involved. In the early days of NER, it was done mainly through hand-crafted lexical rules, but with time and increase in computing power, machine-learning models became more prevalent. One of the milestones in NER was the *Conference on Natural Language Learning* in 2002 and 2003 (Tjong Kim Sang, 2002; Tjong Kim Sang and De Meulder, 2003a), which focused on this task.

The CoNLL conferences provided annotated training, validation, and test sets and introduced a performance metric. Organizations, universities, and others could submit models for the task, and the best performing one took the crown. The conference created a broadly accepted baseline for the task, and the following years of research could use the datasets and evaluation metrics to determine the state-of-the-art in the field.

In this thesis, we focus on named entities consisting of organizations, locations, persons, and we will also have a miscellaneous category for the named entities which are not the aforementioned. For example, in the sentence:

> Nik Johansson does his master's thesis at Tink, which has its office at Vasagatan in Stockholm.

The named entities we find are *Nik Johansson*, which is a person, *Tink*, which is an organization, *Vasagatan* and *Stockholm*, which are both locations.

There has been a lot of research done on NER over the years with current state-of-the-art models being able to perform the task at a high accuracy level (Meta, 2022). However, these models are mainly based on sentences with proper context and proper grammar, such as from newspapers. There has not been much research done on NER for transaction descriptions and this thesis will focus on it.

1.4 Motivation

Named entities are arguably some of the most important components in text in general, and in transaction descriptions specifically. By having a model to extract this information, it opens up for more applications and new products that make use of this information in transaction description data.

It can, for example, be used to detect the top brands in transactions, both on a user level as well as on a market level, to get better knowledge of the market or where an individual spends her/his money. It can be used to mask out sensitive information when applicable. And it can also be used to detect fraudulent transactions by looking at patterns in the detected locations.

Furthermore, we have a hypothesis that NER can in some way be combined with a transaction categorization model and add additional information and therefore improve the performance, which is something we will test.

1.5 Previous Work

As mentioned previously, there has been a lot of research on NER throughout the years. In the CoNLL 2003 shared task, support vector machines and logistic regression were the most popular tools. In this section, we will go through some of the most prominent and effective models as of today.

1.5.1 Neural Architectures for Named Entity Recognition

Out of recent work, researchers from Carnegie Mellon University and Pompeu Fabra University introduced the bidirectional long short-term memory with conditional random fields

(Bi-LSTM-CRF) model in 2016, which later became the most popular architecture type within NER (Lample et al., 2016). In the paper, they used character-level embeddings and trained the model on the CoNLL 2003 English dataset (Tjong Kim Sang and De Meulder, 2003a) to perform a chunk F1 score of 0.909 on the CoNLL2003 test set.

1.5.2 Contextual String Embeddings for Sequence Labeling

Peters et al. (2017) from Zalando's research team continued on the Bi-LSTM-CRF architecture, but added two additional embeddings on top of the character-level embeddings from Section 1.5.1. They added the pre-trained GloVe embeddings (Pennington et al., 2014) and contextualized word embeddings Peters et al. (2018). The model was applied on the CoNLL 2003 English dataset (Tjong Kim Sang and De Meulder, 2003a) and was able to perform an overall chunk F1 score of 0.931 on the CoNLL2003 test set.

1.5.3 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

BERT (Devlin et al., 2018) is a popular transformer architecture introduced by Google AI Language in 2018. BERT is a general purpose model that performs well on many different tasks within natural language processing. The model is first pre-trained to get a deep bidirectional representation by leveraging both the right and left context of the words in the sentences. The pre-training is done using two different unsupervised approaches, masked language modeling and next sentence prediction, and on two different data sources, the English Wikipedia (2500 million words) and BookCorpus (800 million words). The model was then fine-tuned on the CoNLL 2003 English dataset (Tjong Kim Sang and De Meulder, 2003a) and was able to perform an overall chunk F1 score of 0.928 on the CoNLL2003 test set.

Subsequently, the BERT team trained their transformer on a multilingual corpus. Their model, called multilingual BERT or mBERT, can be applied to any language, including Swedish.

1.5.4 Arbetsförmedlingen's Swedish BERT Models

Following the transformers trained on English, Arbetsförmedlingen (The Swedish Public Employment Service) created two models using this architecture (see Section 1.5.3 for reference) (Stollenwerk, 2020). They released one smaller model with approximately 110 million parameters and one larger with 340 million parameters. Both models were trained on approximately 2 million Swedish Wikipedia articles. The models were fine-tuned on SUC 3.0 and were able to perform an overall chunk F1-score of 0.876 using a 70/20/10% training, validation, and test split (Malmsten et al., 2020; Språkbanken, 2012).

1.5.5 Playing with Words at the National Library of Sweden – Making a Swedish BERT

Kungliga Biblioteket (the National Library of Sweden) introduced a second Swedish transformer model, the KB-BERT model (Malmsten et al., 2020), which uses the BERT model and trains it on Swedish language corpora. The model was trained on new articles, Official Reports of the Swedish Government (*Statens offentliga utredningar*), legal e-deposits, social media, and all of the Swedish Wikipedia, summing up to 3,497 million words and 18.341 GB of data. The models were fine-tuned on SUC 3.0 and the best of them was able to perform an overall chunk F1-score of 0.927 using a 70/20/10% training, validation, and test split (Malmsten et al., 2020; Språkbanken, 2012).

1.6 Our contributions

In this thesis, we applied NER on Swedish transaction descriptions. We trained models using transformer architectures, pretrained on Swedish and multilingual corpora.

With our best NER model, we were able to reach an overall chunk F1 score of **0.849** on the Swedish transactions dataset (see Sect. 3.1) despite only using 2200 transactions for training and transaction data being messy. To the best of our knowledge, this is the first published report for NER on transaction descriptions.

Additionally, we were able to use features from our NER model to develop a new model for transaction categorization that outperformed the then-current transaction categorization models at Tink, by increasing the weighted mean KPI score by **7.4%** and **1.1%** respectively.

Chapter 2
Approach

2.1 Overview

As mentioned previously, named entity recognition is the task of identifying and classifying named entities in text. An illustration of NER on transaction descriptions is shown in Fig. 2.1, which components will be explained in coming sections.

In Sect. 1.5, we saw that transformer and Bi-LSTM models are the two most prominent and effective architectures to solve NER. As of 2022, transformer models obtained the best results on the Swedish NER task. Taking this into consideration, as well as the fact that Bi-LSTMs often need more data than what is necessary for a transformer to fine-tune a model, we focused on the transformer architecture.

Since there is no available data for NER on transaction descriptions, neither publicly nor at the company, we had to annotate a corpus ourselves. Due to time limitation, this resulted in a modest dataset. A way to mitigate this data scarcity was to apply transfer learning: take a pre-trained model trained on another dataset and fine-tune it on this annotated dataset.

We evaluated the performance of the model with the *conlleval* script (Tjong Kim Sang and De Meulder, 2003b; Tjong Kim Sang, 2000), which computes the chunk F1-score for the categories and overall. This is the standard evaluation metric for NER, which was introduced at CoNLL and using it will make our results comparable.

As an application of NER, we combined our model for NER with a model for transaction categorization. We could thus develop a new model for transaction categorization that could reach better performance.

2.2 Outline of the Architecture

The processing architecture consists of two main steps: vectorization of the input and data processing.

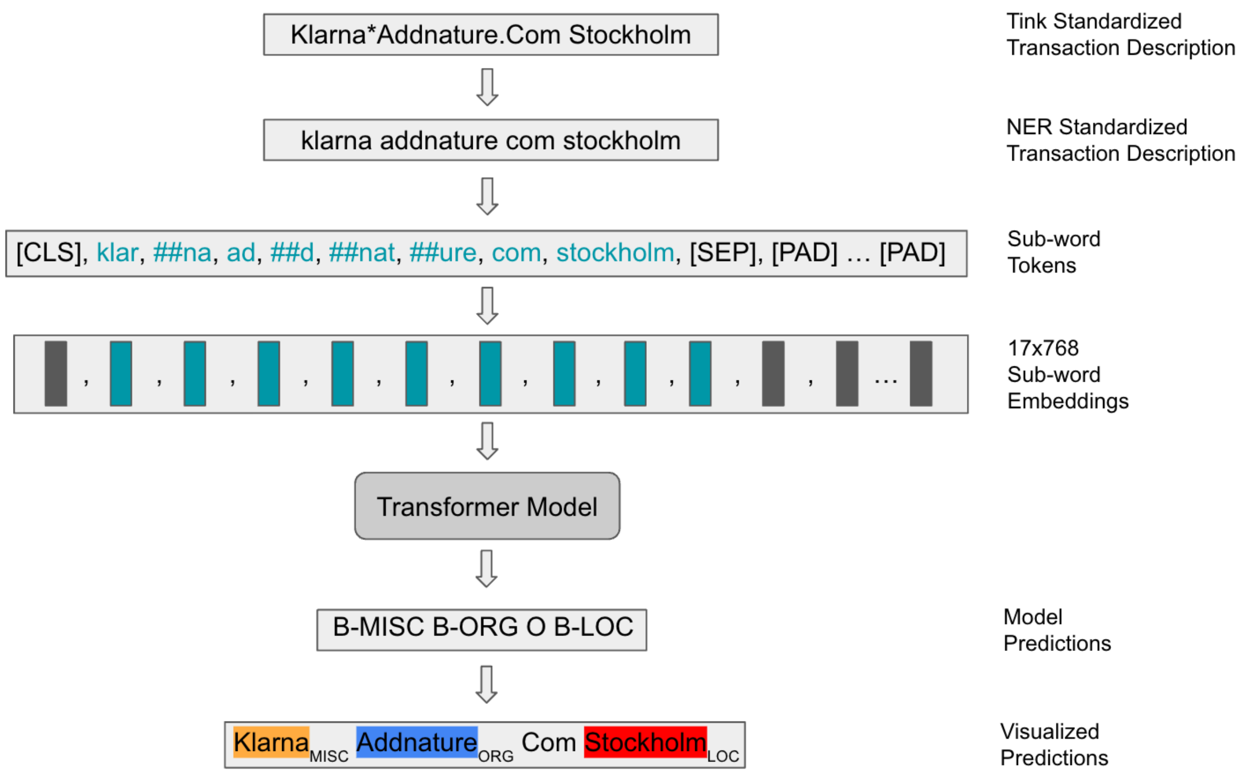

Klarna*Addnature.Com Stockholm
Tink Standardized Transaction Description
klarna addnature com stockholm
NER Standardized Transaction Description
[CLS], klar, ##na, ad, ##d, ##nat, ##ure, com, stockholm, [SEP], [PAD] … [PAD]
Sub-word Tokens
17x768 Sub-word Embeddings
Transformer Model
B-MISC B-ORG O B-LOC
Model Predictions
Klarna MISC Addnature ORG Com Stockholm LOC
Visualized Predictions

Figure 2.1: Illustration of the NER models. A transaction description comes in, which is standardized by Tink. Further NER specific standardization is then applied and then broken up into sub-word tokens. These tokens are then given an embedding representation to then be processed by the transformer model. The transformer model makes its predictions and in this case finds the payment provider *Klarna*, the organization *Addnature*, and the location *Stockholm*. Note that NER standardization, tokenization, and embedding representation are different from model to model.

2.2.1 Vectorization

In general, machine learning models cannot handle text input and therefore need the input to be converted into a numeric representation. This process is called *vectorization* and can be done in different ways. Usually, it includes the following steps, which will be expanded upon in coming sections:

- Standardization
- Tokenization
- Encoding

An example of vectorization of a sentence can be found in Fig.2.2.

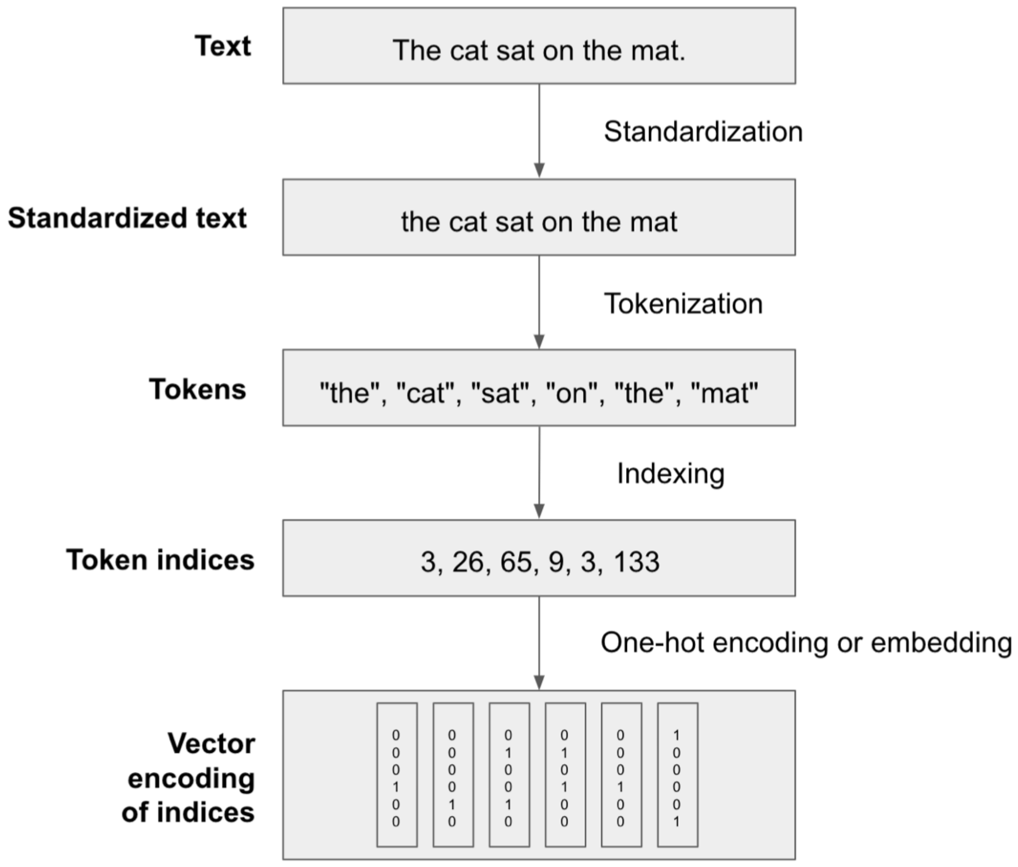


Figure 2.2: An example of vectorization of a sentence. First, the sentence gets standardized by making it lower cased and removing punctuation. The sentence is then broken up into word-level tokens, which are assigned token indices and also vector encodings (either one-hot encodings or embeddings). Image source: Chollet (2021).

2.2.2 Models

Once the text is preprocessed, we trained a transformer. To reach a high performance, a transformer needs a very large training corpus and training it from scratch is only possible with very large resources. Therefore, we started from different pretrained models from Hugging Face 🤗 (Wolf et al., 2019), in a *follow the crowd*-approach. We started out with trying some of the most popular Swedish and multilingual models and, depending on the results and the conclusions we could draw, we iterated with other models.

2.3 Standardization

Standardization is the process of cleaning and working with a dataset, usually with regex rules or similar, before any machine learning is involved. The goal of this process is often to remove unnecessary information and move your task to a smaller feature space to prevent the model to overfit to non-important variations. Some examples of this are:

- Lowercasing the dataset,
- Normalizing special characters like "é" to "e" or "æ" to "ae",
- Removing punctuation, new lines etc.

Standardization can also be used for other reasons. For example, like in this thesis where we use it to remove ambiguity in the dataset, which will be explained further in Sect. 3.1.

2.4 Tokenization and Embeddings

Tokenization and embeddings are important aspects when working with natural language processing. In general, machine learning models cannot handle text input and therefore need the input to be converted into a numeric representation (Chollet, 2021). For this to be achieved, we need to break the text up into smaller pieces. This process is called *tokenization*.

There are different types of tokenization, for example, word tokenization, character tokenization and sub-word tokenization. If we take word tokenization as an example, each word in the corpus is assigned a numeric token. In this case, the set of all words which have a numeric token assigned is known as the *vocabulary* of the model.

Closely related to tokenization, we have embeddings. To have a well functioning NLP model, we need to have a good representation of our tokens. We want to be able to measure semantic distance between tokens to find out which tokens are similar to each other, which ones appear in similar contexts etc. For any complicated NLP task, just having numeric tokens along a one dimensional line will not be sufficient, thus we will need a higher dimensional representation of our tokens. These representations are called *embeddings*.

Similar to tokenization, there are different types of embeddings. If we end up using word tokenization, we will use word embeddings. If we use character tokenization, we will use character embeddings and so on.

One of the more popular embeddings to use are the GloVe embeddings (Pennington et al., 2014). The GloVe embeddings have been trained on large datasets to get good representations of words, where similar words will have similar embeddings, in terms of, for example, cosine similarity. Table 2.1 shows the top 10 most similar words to the word "france", in terms of cosine similarity for the GloVe embeddings.

As we can see in Table 2.1, the list of top 10 words is reasonable. It includes countries arguably being the most similar in terms of geometrical distance, culturally, and in terms of history. It contains the word for either the language of the country or of its people, as well as the capital of the country, and the continent it is placed on. This means that, for any NLP task using the pre-trained GloVe embeddings, exchanging the word "france" for e.g. the word "belgium" in the input will yield similar output.

2.4.1 Types of Tokenization and Embeddings

As mentioned earlier, there are different types of tokenization and embeddings. Which one should you use?

Top 10 cosine similarity to "france"
"france"
"belgium"
"french"
"britain"
"spain"
"paris"
"germany"
"italy"
"europe"
"netherlands"

Table 2.1: The top 10 closest words to "france" in the GloVe embeddings in terms of cosine similarity.

Word Level

The most common way to do tokenization and embeddings is on a word level. In general, this works well, as could be seen above, but has some flaws. One of the flaws is that words can occur in the text that are not part of the vocabulary and thus will get a random embedding that is not a good representation of the word. For text data with common and proper grammar (newspapers etc.), this is less of an issue since most of the words will be in the vocabulary. But using, for example, the GloVe embeddings on text data that either does not have proper grammar or has a lot of words that are unique to the field (i.e. medical data, social media etc.), we will have a lot words that are out of vocabulary.

Another related problem is the size of the vocabulary. If we want to minimize the number of words that are out of vocabulary, we can increase the vocabulary size. As languages may have millions of different words, this can lead to a very large vocabulary. Especially since, for most words, we would need several forms of the word, even though the embedding might be similar. For example, the word "dog" and the word "dogs" will need separate tokenization and embeddings. Having a large vocabulary will result in a large feature space and might lead to overfitting.

Character Level

If we do tokenization and embeddings on a character level, we get rid of some of the flaws described for word level. Using character level tokenization and embeddings, there is always (except rare cases) a nonrandom embedding representation for the text data we are working with. Since we only need to take characters into account, our vocabulary will be small.

However, character-level tokenization and embeddings have some flaws as well. Breaking the text up into so small components might lead to losing too much information. It might be hard to capture the full nuance of a sentence just by looking at the sequence of characters in it.

Character-level tokenization and embeddings reached a high performance in NER specifically, but this is often in combination with other types of embeddings. For example, Akbik et al. (2018) used character-level embeddings together with the GloVe embeddings and contextualized word embeddings (Peters et al., 2017, 2018) in their NER model, which at the time of release was the current state-of-the-art for NER on the English CoNLL 2003 dataset (Tjong Kim Sang and De Meulder, 2003a) and still is among the best performing models for NER (Ruder, 2022).

Sub-word Level

Tokenization and embeddings on a sub-word level is frequently used in many of the state-of-the-art architectures within NLP, e.g. for transformer models, and is a good compromise between word level and character level. The words in the text are split into sub-words based on their character sequence frequencies. This results in a general principle of frequent words remaining intact and infrequent words being split up, which can go all the way down to character level. This ensures there is always a non-random embedding representation for a word, while not losing much of the information and keeping the vocabulary from being too large.

The word “dogs” could, for example, be broken up into “dog” + “s” which would give us the embedding for the word “dog” and the embedding for a general plural form.

2.5 Transformers

The transformer architecture was introduced in 2017 in the paper “Attention is all you need” (Vaswani et al., 2017) and has since then become prominent in many NLP tasks. An illustration of the transformer architecture can be found in Fig. 2.3.

Transformers are general purpose models. In the case of NLP, this means they have a good language understanding in general and can be fine-tuned (see Sect. 2.7) to perform well on several NLP tasks.

2.5.1 Attention

The concept of attention is a key part of the transformer architecture, as one might have guessed from the name of the article, where it was introduced. As opposed to LSTM or other RNN architectures, where the tokens are processed from beginning to end or end to beginning, the transformer architecture is sequence independent. The attention mechanism looks at a token and computes relevance scores between the token and all of the other tokens to see where it should put its attention, resulting in a context-aware representation of the token.

In Fig. 2.4, we see how the attention mechanism comes into play in computing a context-aware vector for the word *station* in the sentence “the train left the station on time”. First, the sentence is tokenized into word tokens and given a vector representation. The attention scores between every word in the vector is then computed, which is then used to create a new context-aware vector representation based on the weighted sum of these word embeddings. This results in a representation where the most important tokens will have the largest weight, and where the representation of a token changes depending on its surrounding context. The word *station* will therefore have a different representation in the context of a *train station*, *radio station*, and *space station*.

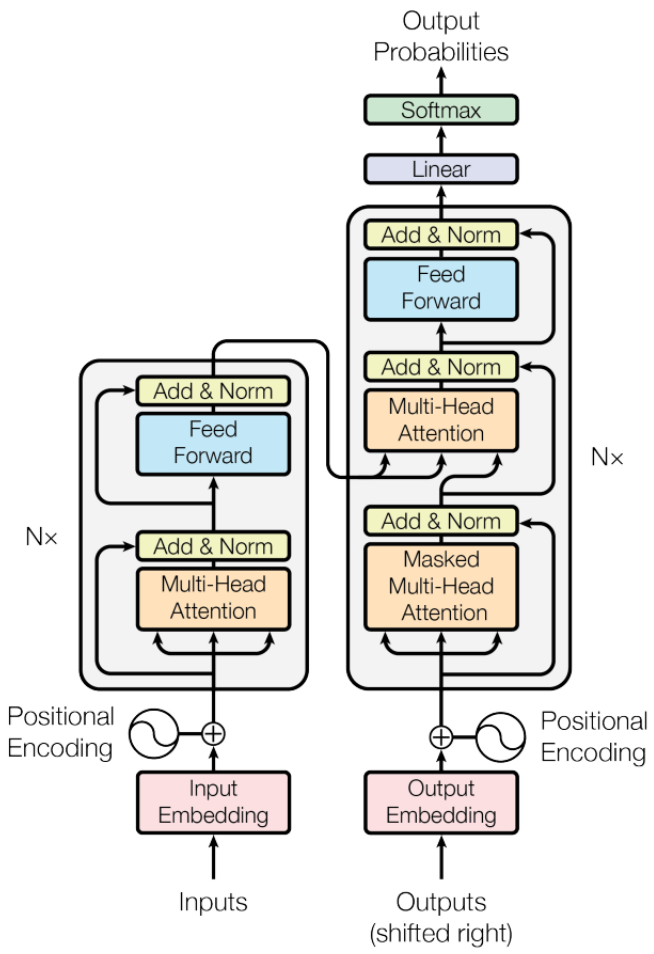


Figure 2.3: An illustration of the transformer architecture. Image source: Vaswani et al. (2017).

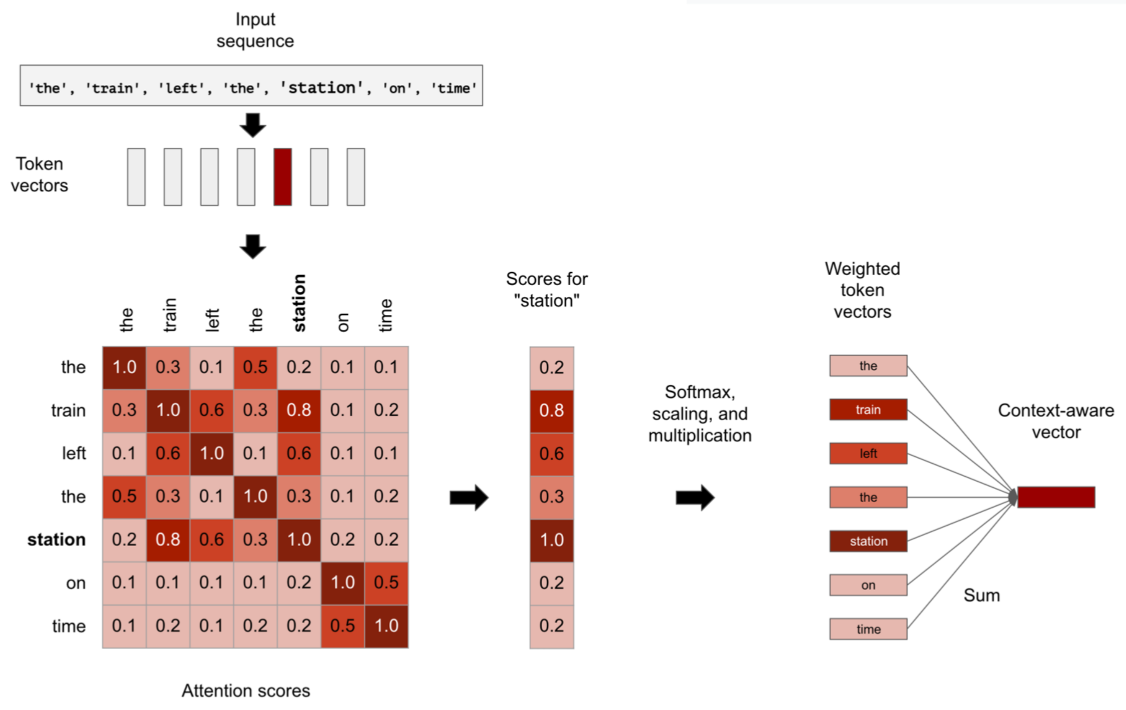
2.5.2 Masked Language Modeling

In the progression of machine learning, there is a strong correlation between added model complexity and better performance. The added complexity enables a better task understanding, but it also requires more data in order to make use of this complexity. But large amounts of high quality labeled data are often hard to come by and thus it becomes important to make use of unlabeled data.

Transformer models are usually pre-trained using an unsupervised training approach. One popular method is masked language modeling (MLM), which is used by BERT (Devlin et al., 2018) as an example. MLM masks out a portion of the tokens in the dataset and have the model predict the masked word. For example, a sentence in the dataset could look like the following:

> Nik Johansson does his master's [MASK] at Tink which has its office at Vasagatan in Stockholm.

where the correct missing word would be "thesis".



	the	train	left	the	**station**	on	time
the	1.0	0.3	0.1	0.5	0.2	0.1	0.1
train	0.3	1.0	0.6	0.3	0.8	0.1	0.2
left	0.1	0.6	1.0	0.1	0.6	0.1	0.1
the	0.5	0.3	0.1	1.0	0.3	0.1	0.2
station	0.2	0.8	0.6	0.3	1.0	0.2	0.2
on	0.1	0.1	0.1	0.1	0.2	1.0	0.5
time	0.1	0.2	0.1	0.2	0.2	0.5	1.0

Figure 2.4: An illustration of the attention mechanism in the transformer architecture. First, a sentence is tokenized and token vectors are generated. The attention scores between each token are then calculated. Looking specifically at the word *station*, a new context-aware vector representation is generated based on the sum of the token vectors in the sentence, weighted by the attention scores. Image source: Chollet (2021).

Since this training method is unsupervised, it enables the model to be trained on large amounts of data and get a good general language understanding. The model can then be fine-tuned on a specific task with a relatively smaller dataset, to get task specific understanding.

2.6 Annotation and Evaluation

Since there is little to no previous work on NER on transaction descriptions, there is no available labeled dataset. Therefore, a dataset needs to be annotated and one needs to think about how to evaluate the performance.

2.6.1 Annotation

We will use a dataset containing Swedish transaction descriptions, that will be annotated by one person (the author of this thesis). To make the results comparable to the majority of NER-models, we will use the same four entity types as in the datasets shared in the CoNLL 2002 and 2003 task (Tjong Kim Sang, 2002; Tjong Kim Sang and De Meulder, 2003a): Organization (ORG), Location (LOC), Person (PER), and Miscellaneous (MISC).

The first three are self explanatory, and the MISC category will be used for all named entities that are not organizations, locations, and persons. For the Swedish transaction descriptions, the MISC category will be almost exclusively payment providers (like Klarna, Swish, and PayPal), products, and apps. Note that temporal expressions and number expressions are excluded here, similarly to the CoNLL datasets.

We will use the BIO tagging scheme. Thus the dataset has the 9 following different NER-tags: B-ORG, I-ORG, B-LOC, I-LOC, B-PER, I-PER, B-MISC, I-MISC and O. Here the prefix “B-” marks the first word in a named entity while “I-” marks the rest of the words in a named entity. O is the label for words that are not named entities.

As an example, the sentence:

Nik Johansson lives in Stockholm

would therefore be annotated as

Nik	Johansson	lives	in	Stockholm
B-PER	I-PER	O	O	B-LOC

2.6.2 Evaluation

As our evaluation metric, we will use chunk F1-score, which is the standard way to evaluate NER that was introduced in the CoNLL-tasks. The chunk F1-score can be retrieved through the original *conlleval* Perl script, of which there is an equivalent in Python (Tjong Kim Sang and De Meulder, 2003b; Tjong Kim Sang, 2000).

The chunk F1-score is the harmonic mean between the precision and recall for an entity chunk (the whole entity). This score is calculated for each of our named entity categories and also an overall score based on the micro-average of the different categories. Using the micro-average will result in the most frequent categories contributing the most towards the overall score. This could be a problem if the categories are imbalanced and the less frequent categories are important for the applications of the NER model.

In order for an entity chunk to be correct, it has to line up exactly with the annotation, e.g. the entity “Ica Maxi Stormarknad” needs to be predicted as

Ica	Maxi	Stormarknad
B-ORG	I-ORG	I-ORG

Any other prediction will be seen as incorrect, thus

Ica	Maxi	Stormarknad
B-ORG	I-ORG	O

does not contribute towards a higher F1-score, even though the partly correct prediction can still be useful.

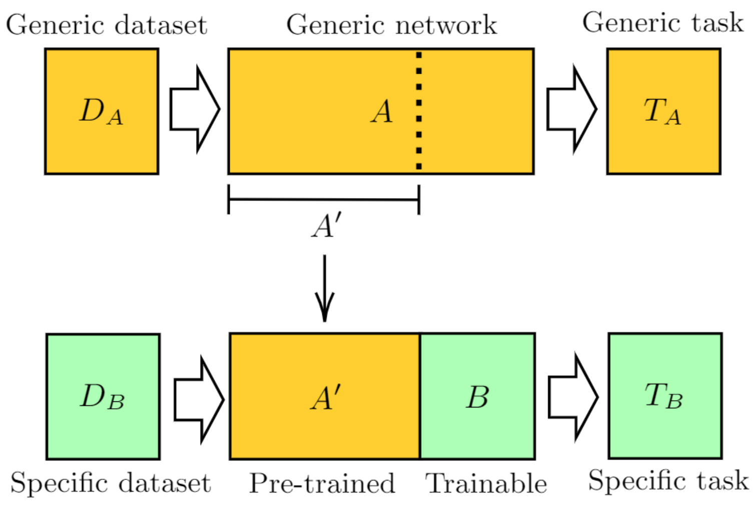


Figure 2.5: An illustration of transfer learning. In the upper part of the figure, we see a trained network, including a task and a dataset. In the bottom part of the network, we see part of the network being reused in another network, for a different task and dataset. Image source: Mari et al. (2020).

2.7 Transfer Learning

Transfer learning, also referred to as fine-tuning, is the process of taking a network that is pre-trained on another dataset and retraining part of the network, as shown in Fig. 2.5. This can be done if we have limited data in relation to the amount of data the network was trained on.

To illustrate, we can take the VGG16 Network (Simonyan and Zisserman, 2015) for image classification. In Fig. 2.6, we can see the filters in some of the early and later layers of the network respectively. In the early layer filters, we can see that these filters are trying to match edges, colors and some basic patterns, while the later filters are trying to match some things more specific, like eyes and feathers, for example. So the earlier layers represent some kind of general image understanding, while the later layers represent specific image understanding. This is also true for NLP. The earlier layers in the network represent some kind of general language understanding, while the later layers represent task and dataset specific language understanding.

If the task and dataset is similar to the pre-trained network, one should keep most of the pre-trained network and just train a new classifier and maybe a couple of the later layers. If there are differences in task and dataset, then some of the earlier layers could still be useful but the rest of the model would need to be retrained. It is also dependent on how much data we have. The more data we have, the more layers should be retrained. Figure 2.7 illustrates how much of the model should be re-trained based on similarity and dataset size.

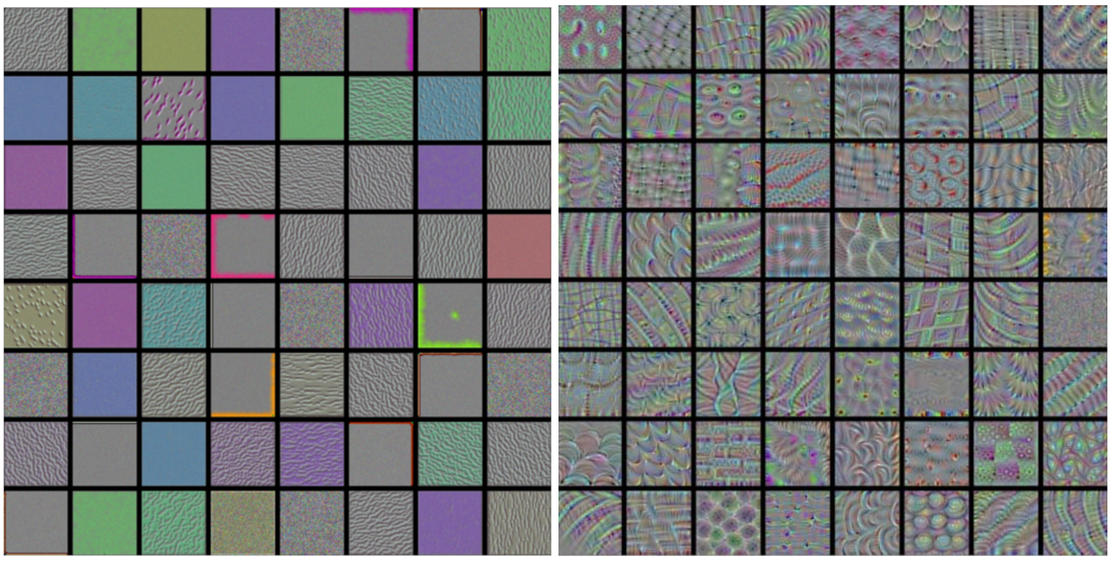


Figure 2.6: An illustration of the filters in an early layer (left) and in a later layer (right). In the early layer filters, we can see that these filters are trying to match edges, colors and some basic patterns, while the later filters are trying to match some things more specific, like eyes and feathers, for example. Image source: Chollet (2017).

2.7.1 Ways of Fine-tuning

TensorFlow

Many of the models on Hugging Face 🤗 can be retrieved as TensorFlow implementations. Because of our familiarity with the framework, we used them in this thesis. Nonetheless, using the TensorFlow implementation limits flexibility in the fine-tuning process. In the TensorFlow implementation, the entirety of the base model is put into one layer called *TFBertMainLayer* or *TFRobertaMainLayer*, etc. This means that our only options, when it comes to freezing layers, are either to freeze the entire base (i.e only the weights between the base and the classifier are trainable), or to unfreeze all the layers.

In terms of classifiers, we can either add a classifier on top of a base model (a model only containing the base, and not a classifier), or we can use a model which already includes a classifier. If we add a classifier on top, we can then specify the NER categories as we want, but the model will not have been fine-tuned for any NER task before our training process starts. If we use a model with an existing classifier, then it will have been fine-tuned on a NER task before we start the training process, but there is no guarantee that the NER categories of the model match the categories one wants to use. In this case, we need to find a translation between the categories of the model from Hugging Face 🤗 and the categories we want to use.

Some of the models that will be used in this thesis have existing classifiers that use the categories from SUC 3.0 (Språkbanken, 2012). In these cases, we translate the categories from SUC 3.0 into ORG, LOC, PER, MISC, and O, in the following way:

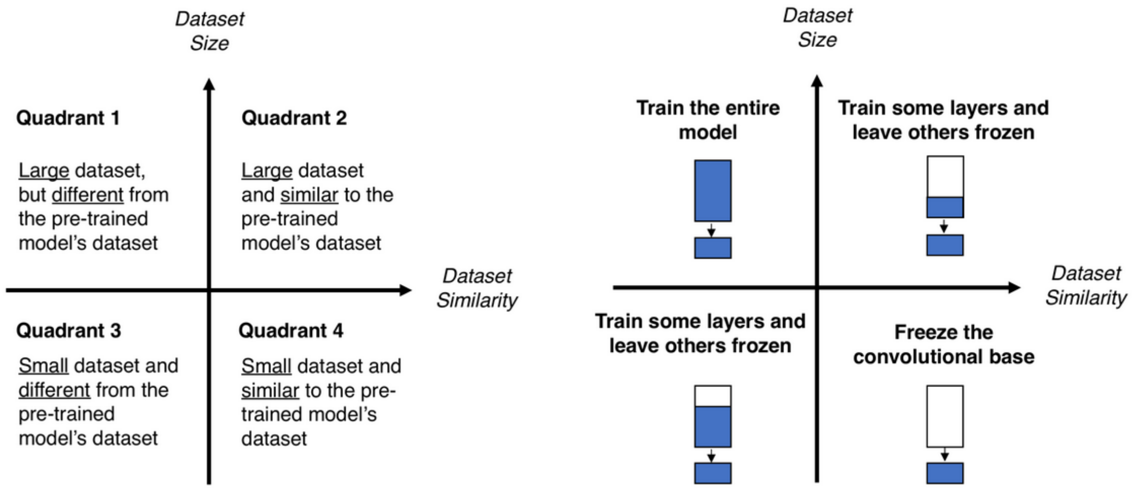

Dataset Size
Quadrant 1
Large dataset, but different from the pre-trained model's dataset
Quadrant 2
Large dataset and similar to the pre-trained model's dataset
Dataset Similarity
Quadrant 3
Small dataset and different from the pre-trained model's dataset
Quadrant 4
Small dataset and similar to the pre-trained model's dataset
Dataset Size
Train the entire model
Train some layers and leave others frozen
Dataset Similarity
Train some layers and leave others frozen
Freeze the convolutional base

Figure 2.7: How much of the network should be retrained based on similarity and dataset size. Image source: Marcelino (2018).

ORG	→	ORG
LOC	→	LOC
PRS	→	PER
OBJ, ORG/PRS, OBJ/ORG, PRS/WRK, WRK, LOC/PRS, LOC/ORG, EVN	→	MISC
O, TME, MSR	→	O

where PSR are persons, OBJ are objects, WRK are work and art, EVN are events, TME are temporal expressions, and MSR are numerical expressions. ORG and LOC are the same as defined previously.

If we want more flexibility in the fine-tuning process, we could use the PyTorch implementation of the Hugging Face 🤗 models, where each layer in the base model can be frozen individually, or we could use other tools like *NerBlackBox*.

NerBlackBox

NerBlackBox is a black-box tool used in this thesis for training and hyperparameter tuning of NER, provided by Arbetsförmedlingen (Stollenwerk, 2021). NerBlackBox can also be used for many of the models on Hugging Face 🤗. The tool takes a model, a dataset, and training parameters as input and produces a fine-tuned model as output.

2.8 Transaction Categorization

Transaction categorization is the task of classifying transactions into different categories, like groceries or public transportation. The classification is mainly based on the transaction description, but the amount of the transaction can also be included as an input feature.

In this thesis, we had a hypothesis that a well-functioning model for NER could in some way be used in the process of transaction categorization to improve performance. Let us take two transactions as examples: "Willys Lund" and "Klarna Apotea". If the model could distinguish that *Willys* is an organization and *Lund* is a location, or that *Klarna* is a payment provider and *Apotea* is an organization, then the hypothesis is that the model would be better at determining the correct category for the transaction.

2.8.1 Models, Performance metrics, and Datasets

At Tink, there are existing models, evaluation metrics, and datasets for transaction categorization. Tink uses different models for different markets and customers, but in general, their least common denominator is that they are based on the Facebook AI fastText model (Joulin et al., 2016). We will therefore use the performance of one of the fastText models for transaction categorization for Swedish data as a baseline when comparing the performance of our model with NER included.

Some models also use XGBoost (Chen and Guestrin, 2016), which takes the output probabilities of the fastText models, adds the amount, and uses that as input. The models have 45 labels that fall under 8 parent categories. As performance metrics, we will use the same metrics that are used at Tink for model performance. The performance metrics and their explanation can be found in Table 2.2.

Performance metrics	**Explanation**
Mode accuracy by frequency	Accuracy of model where every transaction has equal weight. This is the same as precision by frequency × categorization level.
Mode precision by frequency	Precision of model where every transaction has equal weight. This is the same as accuracy by frequency of the transactions which are categorized.
Mode accuracy by amount	Accuracy of model weighted by the amount. This is the same as precision by amount × categorization level.
Mode precision by amount	Precision of model weighted by the amount. This is the same as accuracy by amount of the transactions which are categorized.
Mode parent accuracy by amount	Accuracy for the parent categories weighted by amount.
Categorization level	The model will not predict a category unless it finds the probability of that category to be > x% (50% for fastText and customizable for XGBoost). Categorization level shows the % of transactions that have been categorized.
Weighted mean KPI score	Weighted mean of all of the above, except categorization level.

Table 2.2: The performance metrics and their explanation for the transaction categorization models. Mode, in this case, means mode categories. All transactions with the same description will be assigned the mode category of that description.

For our transaction categorization model, we will use the same dataset Tink uses for its transaction categorization models. Most of the data will be transactions, where the labels have been assigned with different methods: Amazon Mechanical Turk, keyword-based labeling, and through different kinds of user feedback, and some data from OpenStreetMap (OpenStreetMap contributors, 2017), which looks similar to the transaction descriptions (organization and possible location).

In the training step, all of the data sources are available, while for validation and testing only a certain type of user feedback labeled transactions will be used, since this kind of labeling is considered to have the highest quality.

Chapter 3

Dataset

3.1 Swedish Transaction Dataset

The Swedish transactions dataset is a set of annotated transaction descriptions from banks in Sweden which are Tink partners, from 2015 to 2021. We have manually annotated these transaction descriptions with NER tags. In total, 3200 transactions.

Engineers from Tink first preprocessed these transaction descriptions using regexes. We complemented this work with the same techniques. The preprocessing by Tink was mainly done from a user experience (UX) perspective, where they preprocessed it to look good for the end user, while our preprocessing was mainly to remove redundant information and turn ambiguous entities into non-ambiguous entities. For example, the transaction:

```
Systembolaget,Eskilstuna
```

where you have an organization and a location both in one token. In this case, we can split the token at "," to create two non-ambiguous entities.

This dataset uses the BOI tagging scheme explained in Sect. 2.6.1 with the following tags: ORG for organizations, LOC for locations, PER for person, MISC for miscellaneous (almost exclusively products, apps, and payment providers).

3.1.1 Exploratory Data Analysis

Table 3.1 shows some examples of transaction descriptions before preprocessing. We can see that the transactions do include a lot of named entities, but they can be a bit messy. There are, for example, cutoff words, words missing some letters, some of the "åäö" have been exchanged with "aao", removed, or exchanged with white space. Also, note that the first letter of each word is upper cased.

In the dataset, there are a few tag sequences that are very frequent. In addition, the length of the transactions are short, sometimes consisting of only one word:

Id	Transaction description examples
1	Ica Nara Jon
2	Bostadsb
3	V Sttrafik A
4	Jobmeal Jnkping
5	Atm Kontanten1073, Helsingborg
6	12345678910
7	Övf Via Internet 123456789101112
8	Swish Nik Johansson

Table 3.1: Examples of transaction descriptions before preprocessing. Note that any sensitive information in this data like names and account numbers has been replaced.

```
Spotify
Subway
```

or two words:

```
Okq8 Umeå
Postnord Sve
```

Table 3.2 shows the frequency of the 10 most frequent complete tag sequences. We can also plot how much of the dataset is covered by the top X complete tag sequences, as shown in Fig. 3.1. From this, we see that the 5 most frequent complete tag sequences cover 45% of the dataset and the top 10 cover 66% and so on.

Tag sequence	Frequency
B-ORG I-ORG	155
B-ORG	154
O	153
O O O O	144
B-ORG B-LOC	111
O O	87
B-ORG I-ORG B-LOC	83
B-MISC	74
B-MISC B-PER I-PER	49
B-MISC O	49

Table 3.2: The frequency of the top 10 most frequent complete sequences out of the 1600 transactions in the Swedish transactions dataset.

By looking at a sequence, we get a good idea of what type of transaction we are dealing with. For example, if you have an organization followed by a location, it is probably a brick-and-mortar type of store, like a grocery store or a clothing store. If you have a four-word transaction with no entities, it is most likely a transfer between two bank accounts, and if you have a miscellaneous entity followed by a person or something that is not a named entity, then it is most likely a transfer through a service like Swish or similar.

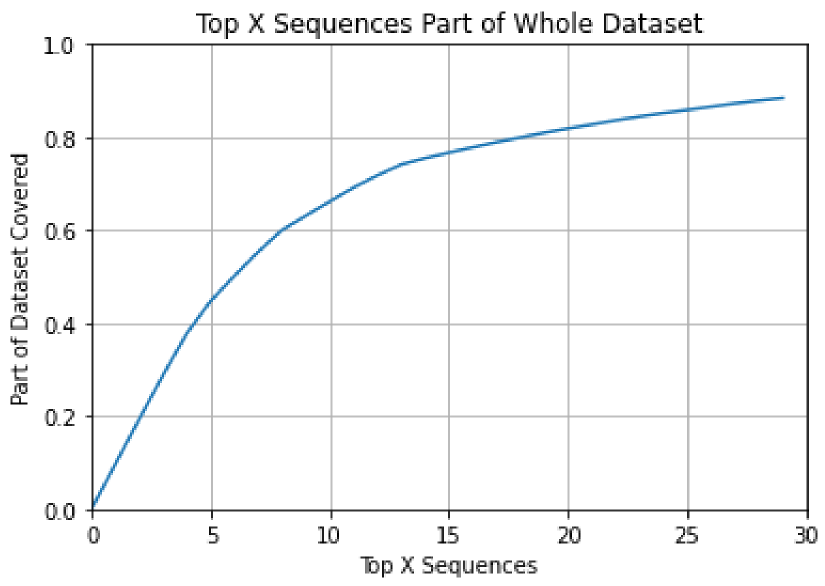


Figure 3.1: This figure shows the frequency of the top X most frequent complete sequences.

3.2 Swedish NER Corpus

The Swedish NER corpus (Klintberg, 2020) is a public dataset that consists of annotated Swedish news from 2012 provided by Språkbanken. The dataset has the following NER tags: ORG, LOC, PER, MISC, and 0.

We converted the dataset into the begin-inside-outside (BIO) scheme explained in Sect. 2.6.1. We applied the conversion by setting a *B* prefix to the first tag and then *I* prefixes to following tags of the same category. While this works well in general, it introduces incorrect labels in the case where two separate named entities of the same type are right next to each other. Nonetheless, this is a rare case, if at all present in the dataset, and can therefore be neglected.

One example of a sentence in this dataset, with the original annotation :

> [...] så (0) jag (0) är (0) inte (0) orolig (0) , (0) säger (0) Ingvar (PER) Kamprad (PER) till (0) Smålandsposten (ORG) . (0)

and after being converted to the BIO scheme:

> [...] så (O) jag (O) är (O) inte (O) orolig (O) , (O) säger (O) Ingvar (B-PER) Kamprad (I-PER) till (O) Smålandsposten (B-ORG) . (O)

Chapter 4

Named Entity Recognition Models

In this chapter, we go through the different named entity recognition models we developed. First of all, we trained multiple models on a smaller dataset (1500 transactions), with the following architectures: Kungliga Biblioteket Swedish BERT, Arbetsförmedlingen AI Swedish BERT, BERT multilingual, and XLM RoBERTa.

We used the Kungliga Biblioteket Swedish BERT model as our baseline model, since this is the most popular model for Swedish NER on Hugging Face 🤗 and it also being the current state-of-the-art for Swedish NER, during the thesis period (2021-2022).

We evaluated the performance, as described in Sect. 2.6.2, on a test set of Swedish transactions dataset, where we will specifically look at the *overall chunk F1 score*, which is the weighted mean of the chunk F1 score for all named entity categories.

When all models had been evaluated on a smaller dataset of 1500 transactions, we took our best performing monolingual model and our best performing multilingual model to see how they performed with added data (3200 transactions).

An overview of the results of the NER models we developed is shown in Table 4.1. Here we can see that for 1500 transactions, the smaller version of the Arbetsförmedlingen AI model was our best performing monolingual model, and the BERT multilingual was our best performing multilingual model. We can also see that when these models are trained on 3200 transactions, we get an overall chunk F1 score of 0.849 and 0.805 respectively.

4.1 Kungliga Biblioteket Swedish BERT

The KB Swedish BERT model (Malmsten et al., 2020) is a BERT based-model developed by the National Library of Sweden (Kungliga Biblioteket). We use one version of the model specific to NER, which has been fine-tuned on SUC 3.0 corpus (Språkbanken, 2012). This is a transformer-based model using sub-word tokenization. The model is case sensitive.

Overview of results – NER models	Precision	Recall	F1
Kungliga Biblioteket	0.515	0.596	0.552
Arbetsförmedlingen AI Small	0.768	0.792	0.780
Arbetsförmedlingen AI Large	0.775	0.761	0.768
BERT Multilingual	0.763	0.751	0.757
XLM RoBERTa	0.274	0.417	0.331
XLM RoBERTa Twitter	0.569	0.581	0.575
Arbetsförmedlingen AI Small 3200	0.847	0.852	0.849
BERT Multilingual 3200	0.816	0.796	0.805

Table 4.1: An overview of the results of the NER models we developed, where the first six are trained using 1500 transactions. The last two are the best performing monolingual and multilingual model, trained using 3200 transactions.

4.1.1 Pretrained – Swedish NER corpus

We evaluated the model that had been fine-tuned on SUC 3.0 on the test set of the Swedish NER corpus (see Sect. 3.2) using *conlleval*. Since SUC 3.0 and the Swedish NER corpus do not use the same NER tags, the tags from SUC 3.0 had to be translated into the tags of the Swedish NER corpus. This was done as explained in Sect. 2.7.1. This gave us an overall chunk F1 score of 0.774, as shown in Table 4.2. Although evaluating the same corpus but with the text in lower casing, we obtain the results shown in Table 4.3, giving us an overall chunk F1 score of 0.345. We can see that the performance drops substantially across the board, giving us a picture of the importance of casing in the model.

Swedish NER – KB Swedish BERT	Precision	Recall	F1
Overall	0.771	0.778	0.774
ORG	0.542	0.758	0.632
PER	0.930	0.871	0.899
LOC	0.913	0.792	0.848
MISC	0.154	0.183	0.167

Table 4.2: The chunk performance of the Swedish BERT model on the Swedish NER corpus using conlleval.

Swedish NER – KB Swedish BERT lower casing	Precision	Recall	F1
Overall	0.238	0.628	0.345
ORG	0.192	0.645	0.296
PER	0.209	0.549	0.303
LOC	0.403	0.773	0.530
MISC	0.016	0.105	0.028

Table 4.3: The chunk performance of the Swedish BERT model on a lower cased version of the Swedish NER corpus using conlleval.

4.1.2 Pretrained – Swedish Transactions Dataset

We evaluated the model that had been fine-tuned on SUC 3.0 on the test set of the Swedish transactions dataset using *conlleval*. As with the previous experiment, since SUC 3.0 and the Swedish transactions dataset do not use the same NER tags, we translated the tags from SUC 3.0 into the tags of the Swedish transactions dataset; see Sect. 2.7.1. This gave us an overall chunk F1 score of 0.243, as shown in Table 4.4. Table 4.5 shows the evaluation the same dataset but with having the text in lower casing, giving us an overall chunk F1 score of 0.030.

Tink – KB Swedish BERT	Precision	Recall	F1
Overall	0.193	0.330	0.243
ORG	0.151	0.597	0.241
PER	0.683	0.204	0.315
LOC	0.265	0.429	0.327
MISC	0.012	0.036	0.018

Table 4.4: The chunk performance of the Swedish BERT model on the Swedish transactions dataset using conlleval.

Tink – KB Swedish BERT lower casing	Precision	Recall	F1
Overall	0.016	0.348	0.030
ORG	0.007	0.400	0.014
PER	0.070	0.273	0.115
LOC	0.029	0.429	0.055
MISC	0.000	0.000	0.000

Table 4.5: The chunk performance of the Swedish BERT model on a lower cased version of the Swedish transactions dataset using conll-eval.

Here we can see that using the non-lower cased version of the dataset vastly improves the precision at the cost of a little bit of recall, compared to the lower cased version. Note that the casing in the dataset is just the initial letter in every word being upper cased and therefore has no real meaning. As we could see in Sect. 4.1.1, the model got affected a lot by the casing. Since a lot of the words in the dataset are named entities and the model seems more prone to predicting a word as a named entity if it starts with an upper case letter, the model performs better on the non-lower cased version.

4.1.3 Fine tuned – KB Swedish Transactions Dataset

We fine-tuned the KB Swedish BERT model on the Swedish transactions dataset. We did this by freezing all the weights from all layers in the original model except the last one. We then translated the NER tags of the Swedish transactions dataset into the tags of SUC 3.0 in the following way:

ORG →ORG LOC →LOC PER →PRS MISC →OBJ O →O

We applied the training procedure to 700 transactions, we used 300 for validation, and 500 for testing. We ran the model for 40 epochs.

After model prediction, the tags were translated back into ORG, LOC, PER, MISC, O. Note that we could not take prefixes ”B-” or ”I-” into consideration since the Swedish BERT model does not use them and the Swedish transactions dataset has a lot of cases, where two different named entities of the same type are next to each other.

Table 4.6 shows the evaluation of the model using *conlleval*. Here we can see that the model performs much better for all the categories than the pretrained model, improving the F1 score by 0.301 overall, by 0.259 for the ORG category, 0.317 for the PER category, 0.105 for the LOC category, and 0.779 for the MISC category. Such results confirm the importance of fine-tuning.

Tink fine tuned – KB Swedish BERT	**Precision**	**Recall**	**F1**
Overall	0.515	0.596	0.552
ORG	0.480	0.521	0.500
PER	0.756	0.544	0.632
LOC	0.330	0.625	0.432
MISC	0.709	0.910	0.797

Table 4.6: The chunk performance of the KB Swedish BERT model fine-tuned on the Swedish transactions dataset (without any changes to casing) using conlleval. The NER tags of the Swedish transactions dataset was translated into the tags of the KB Swedish BERT model.

4.1.4 Learning Curve

We wanted to know if we can expect the model to perform better with more data, since it was only trained on 700 transactions. To evaluate this, we annotated another 100 transactions and trained the model on 0 (i.e no training), 500, 600, 700 and 800 transactions to see the effect on the overall chunk F1 score shown in Figure 4.1.

In the figure, it seems like the overall chunk F1 score reaches a plateau, which makes us believe we would not improve performance massively by annotating more data.

4.2 The Arbetsförmedlingen AI Model

The AF-AI model (Stollenwerk, 2020) is a BERT-based model developed by the Swedish Public Employment Service. The model uses sub-word tokenization and has been trained on Swedish Wikipedia articles. The model is case insensitive.

4.2.1 Fine Tuned – AF-AI Swedish Transactions Dataset

We fine-tuned the smaller version of the AF-AI model on the Swedish transactions dataset using the NerBlackBox tool (see Sect. 2.7.1).

We applied the training procedure to 700 transactions, we used 300 for validation, and 500 for testing. The training was done using early stopping on 2 different hyperparameter

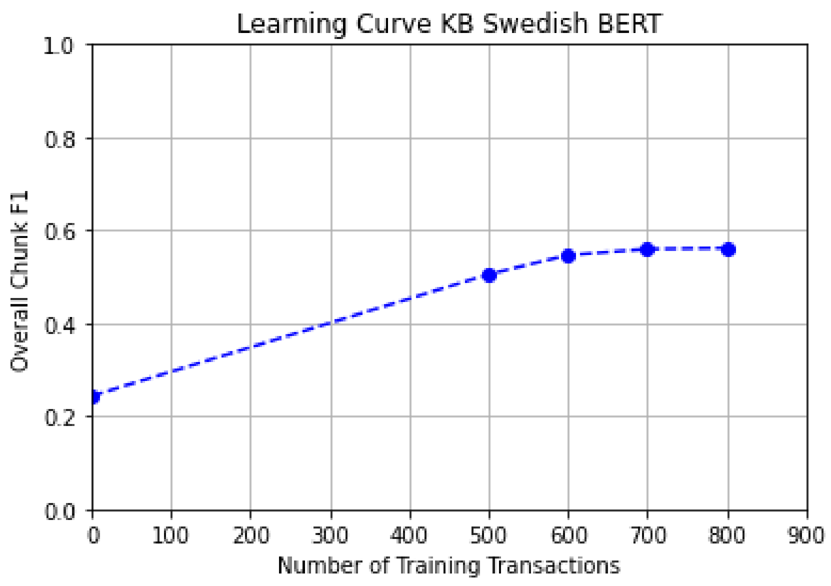


Figure 4.1: The learning curve of the KB model, where we see how the overall chunk F1 score is affected by the number of training transactions.

setups, where each setup was run 3 times with the best one being picked out. The hyperparameters are the default ones from NerBlackBox, but using a patience of 3 epochs in early stopping, a max input sequence length of 16 sub-words, and different setups for learning rates. One setup uses a constant learning rate of $4 \cdot 10^{-5}$, while the other one uses a learning rate of $3 \cdot 10^{-5}$ with cosine annealing.

Table 4.7 shows the evaluation of the model using *conlleval*. Here we can see that it performs better than the fine-tuned KB model in 4.1.3, improving the F1 score by 0.216 overall, by 0.238 for the ORG category, 0.257 for the PER category, 0.336 for the LOC category, and 0.096 for the MISC category.

Fine tuned small – AF Swedish transactions dataset	**Precision**	**Recall**	**F1**
Overall	0.768	0.792	0.780
ORG	0.754	0.721	0.738
PER	0.878	0.900	0.889
LOC	0.716	0.830	0.768
MISC	0.825	0.973	0.893

Table 4.7: The conlleval chunk performance of the small AF-AI model fine-tuned on the Swedish transactions dataset

If we instead fine-tune the larger version of the AF-AI model using the same procedure, we get the results shown in Table 4.8. Here we can see that the larger model does not perform better on the overall F1 score than the smaller one, especially with the PER category performing worse.

Interesting to note is that both AF-AI models perform better than the fine-tuned KB model in 4.1.3, despite the KB model showing better results than the AF-AI model on the

Fine tuned large – AF Swedish transactions dataset	Precision	Recall	F1
Overall	0.775	0.761	0.768
ORG	0.751	0.718	0.734
PER	0.731	0.698	0.714
LOC	0.806	0.755	0.779
MISC	0.837	0.986	0.905

Table 4.8: The conlleval chunk performance of the large AF-AI model fine-tuned on the Swedish transactions dataset

SUC 3.0 corpus (Malmsten et al., 2020). This gives us an indication that the model being case insensitive is of importance.

4.2.2 Learning Curve

We wanted to know if we can expect the model to perform better with more data, since it was only trained on 700 transactions. To evaluate this, we fine-tuned the smaller model on 100, 500, 600, 700, 800, and 900 transactions to see the effect on the overall chunk F1 score. Figure 4.2 shows the results.

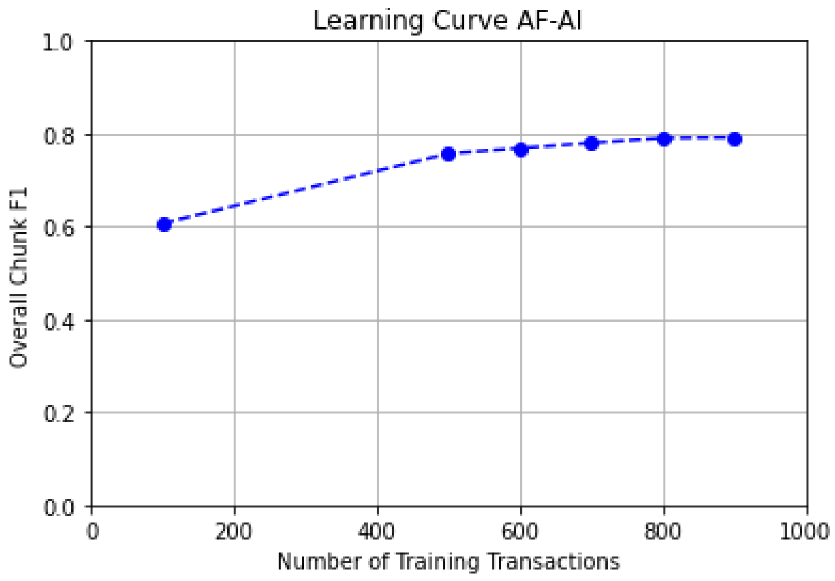


Figure 4.2: The learning curve of the AF-AI model, where we see how the overall chunk F1 score is affected by the number of training transactions.

In the figure, there is no clear sign the overall chunk F1 score seems to plateau, which makes us believe that the model might improve with more data.

4.3 BERT Multilingual

mBERT is a multilingual uncased version of the BERT model (Devlin et al., 2018), which is a transformer-based model developed by Google. The model uses sub-word tokenization and trained on Wikipedia articles for 102 different languages.

4.3.1 Fine Tuned – mBERT Swedish Transactions Dataset

We fine-tuned the mBERT model on the Swedish transactions dataset using the NerBlackBox tool (see Sect. 2.7.1).

We applied the training procedure to 700 transactions, we used 300 for validation, and 500 for testing. The training was done using early stopping on 2 different hyperparameter setups, where each setup was run 3 times with the best one being picked out. The hyperparameters are the default ones from NerBlackBox, but using a patience of 3 epochs in early stopping, a max input sequence length of 16 sub-words, and different setups for learning rates. One setup uses a constant learning rate of $4 \cdot 10^{-5}$, while the other one uses a learning rate of $3 \cdot 10^{-5}$ with cosine annealing.

Table 4.9 shows the evaluation of the model using *conlleval*. Here we can see that the model in general performs slightly worse than the AF-AI model in 4.2.1, decreasing the overall F1 score by 0.023.

Fine tuned – BERT multilingual Swedish transactions dataset	Precision	Recall	F1
Overall	0.763	0.751	0.757
ORG	0.726	0.709	0.718
PER	0.854	0.714	0.778
LOC	0.767	0.731	0.749
MISC	0.837	0.973	0.900

Table 4.9: The conlleval chunk performance of the BERT multilingual model fine-tuned on the Swedish transactions dataset

4.3.2 Learning Curve

We wanted to know if we can expect the model to perform better with more data, since it was only trained on 700 transactions. To evaluate this, we fine-tuned the model on 100, 500, 600, 700, 800, and 900 transactions to see the effect on the overall chunk F1 score. See Figure 4.3.

In the figure, there is no clear sign the overall chunk F1 score reached a plateau, which makes us believe that the model might improve with more data.

We also note that the learning curve shows a bit more of a steep and linear trend than previous models. This gives us an indication the model might scale better with data. This could be the case since the model is multilingual and thus it does not only have to learn transaction data, but also focus on the Swedish language, which is not the case for the monolingual models.

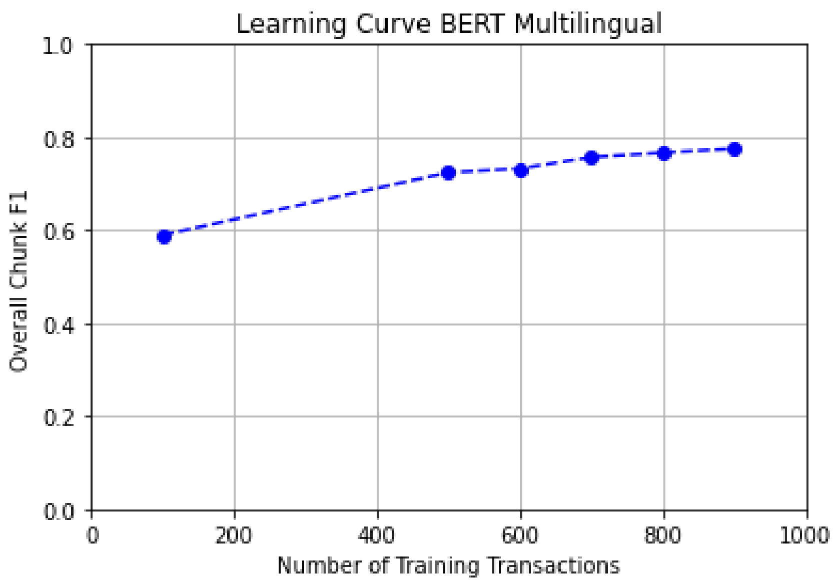


Figure 4.3: The learning curve of the BERT multilingual model, where we see how the overall chunk F1 score is affected by the number of training transactions.

4.4 XLM RoBERTa

XLM-RoBERTa is a multilingual case sensitive model (Conneau et al., 2019), which is transformer-based developed by Facebook. The model uses sub-word tokenization and trained on CommonCrawl data for 100 different languages.

4.4.1 XLM RoBERTa Base

We used the base of the XLM-RoBERTa model where we added a dropout layer as well as a classifier. We tried the model both with and without lower casing the Swedish transactions dataset. As a training procedure, we used 700 transactions as well as 300 for validation and 500 for testing. We froze the model base and ran 40 epochs. We then unfroze the model base and continued training the model for a couple of epochs until the validation loss no longer improved.

Tables 4.10 and 4.11 show the evaluation of the model using *conlleval* after the model head had been fine-tuned for 40 epochs, with and without lower casing respectively. Here we can see that the version that is not lower cased performs slightly better than the model with lower casing, but the performance is still far worse than what we have gotten for our previous models.

Tables 4.12 and 4.13 show the evaluation of the model using *conlleval* after the full training procedure, with and without lower casing respectively. Here we can see that the version that is not lower cased still performs slightly better than the model with lower casing. We managed to increase the performance, but the performance is still far worse than what we had gotten for our previous models.

The relatively bad performance might be due to the model being case sensitive. We can

Fine tuned lower – XLM RoBERTa	Precision	Recall	F1
Overall	0.126	0.310	0.179
ORG	0.228	0.301	0.263
PER	0.000	0.000	0.000
LOC	0.000	0.000	0.000
MISC	0.000	0.000	0.000

Table 4.10: The conlleval chunk performance of the XLM RoBERTa model, with the model head fine-tuned on a lower cased version Swedish transactions dataset.

Fine tuned – XLM RoBERTa	Precision	Recall	F1
Overall	0.157	0.340	0.215
ORG	0.284	0.341	0.310
PER	0.000	0.000	0.000
LOC	0.000	0.000	0.000
MISC	0.000	0.000	0.000

Table 4.11: The conlleval chunk performance of the XLM RoBERTa model, with the model head fine-tuned on the Swedish transactions dataset.

see that the performance improves while unfreezing the whole network, but still without great results. In this case, we would probably need to unfreeze more than just the classifier, but less than the whole network, in order to get optimal performance. Though this is not possible with the TensorFlow implementation of the model.

4.4.2 XLM RoBERTa Twitter

This model is the XLM RoBERTa trained on tweets from different languages (Barbieri et al., 2021). On top of this, we added a dropout layer as well as a classifier. We tried the model both with and without lower casing the Swedish transactions dataset. As a training procedure, we used 700 transactions as well as 300 for validation and 500 for testing. We froze the model base and ran 40 epochs. We then unfroze the model base and continued training the model for a couple of epochs until the validation loss no longer improved.

Tables 4.14 and 4.17 show the evaluation of the model using *conlleval* after the model head had been fine-tuned for 40 epochs, with and without lower casing respectively. Here we can see that the lower cased version performs slightly better than the one that is not lower cased. We can also see that it performs much better than the equivalent models for XLM RoBERTa base in 4.4.1. Although they still have worse performance than non- XLM RoBERTa models in previous sections.

Tables 4.16 and 4.13 show the evaluation of the model using *conlleval* after the full training procedure, with and without lower casing respectively. Here we can see that we do manage to improve the performance by unfreezing the whole network, achieving similar results as the KB model in Sect. 4.1.3 but still being worse than the AF-AI model in Sect. 4.2.1 and the BERT multilingual model in Sect. 4.3.

Whole model lower – XLM RoBERTa	Precision	Recall	F1
Overall	0.273	0.405	0.327
ORG	0.396	0.433	0.414
PER	0.073	0.750	0.133
LOC	0.029	0.500	0.055
MISC	0.255	0.286	0.270

Table 4.12: The conlleval chunk performance of the XLM RoBERTa model, with the whole model trained lower cased version Swedish transactions dataset.

Whole model – XLM RoBERTa	Precision	Recall	F1
Overall	0.274	0.417	0.331
ORG	0.365	0.428	0.394
PER	0.366	0.833	0.508
LOC	0.000	0.000	0.000
MISC	0.256	0.286	0.270

Table 4.13: The conlleval chunk performance of the XLM RoBERTa model, with the whole model trained on the Swedish transactions dataset.

This makes us believe Twitter data may have some similarities with the Tink transaction data, mainly casing not playing as big of a role, shorter context, and messy grammar. But the model still has some of the flaws from the XLM RoBERTa base in Sect. 4.4.1.

4.5 Adding More Data

In this section, we take our best performing monolingual model as well as our best performing multilingual model and we add more data to see what the effects are.

4.5.1 AF-AI 3200 Transactions

We used the same model as the smaller model in Sect. 4.2.1, but with more data. We applied the same training procedure but with 2200 transactions for training, 500 transactions for validation, and 500 transactions for testing. The results of this model are shown in Table 4.18. Here we can see that this model performs better across all categories than the model with less data, increasing the overall chunk F1 score by 0.069.

Looking at the learning curve for the model, shown in Fig. 4.4, we can see that the model performance does not show any clear signs of plateauing, which makes us think the model will keep on improving with more data.

4.5.2 BERT Multilingual 3200 Transactions

We used the same model as in Section 4.3, but with more data. We applied the same training procedure but with 2200 transactions for training, 500 transactions for validation, and 500

Fine tuned Twitter lower – XLM RoBERTa	Precision	Recall	F1
Overall	0.439	0.435	0.437
ORG	0.411	0.321	0.361
PER	0.000	0.000	0.000
LOC	0.398	0.500	0.443
MISC	0.791	0.919	0.850

Table 4.14: The conlleval chunk performance of the Twitter version of the XLM RoBERTa model, with the model head fine-tuned on a lower cased version Swedish transactions dataset.

Fine tuned Twitter – XLM RoBERTa	Precision	Recall	F1
Overall	0.417	0.377	0.396
ORG	0.414	0.267	0.325
PER	0.000	0.000	0.000
LOC	0.281	0.547	0.372
MISC	0.791	0.907	0.844

Table 4.15: The conlleval chunk performance of the Twitter version of the XLM RoBERTa model, with the model head fine-tuned on the Swedish transactions dataset.

transactions for testing. The results of this model are shown in Table 4.19. Here we can see that this model performs better for ORG, PER, MISC and Overall, while actually performing worse for LOC than the model with less data. The overall chunk F1 score increases by 0.048, which is not as much of an increase as for the monolingual model in Sect. 4.5.1.

Looking at the learning curve for the model, shown in Fig. 4.5, we can see that the model performance does show some signs of plateauing, which is different from what we expected by looking at the learning curve in Sect. 4.3.2. If we extrapolate the trend from this plot, then we would not see large increases in model performance by adding more data, and the monolingual model's performance seems to diverge from the multilingual one. Although one would need to annotate more data to be sure.

As a preliminary conclusion, we see that the Arbetsförmedlingen AI model has the best performance over all the other models we evaluated, reaching a F1 score of nearly 85% i.e. 4.4 percentage points better than mBERT. This corresponds to nearly 30% of error reduction.

Whole model Twitter lower – XLM RoBERTa	Precision	Recall	F1
Overall	0.547	0.594	0.570
ORG	0.460	0.512	0.484
PER	0.073	0.061	0.067
LOC	0.709	0.785	0.745
MISC	0.872	0.974	0.920

Table 4.16: The conlleval chunk performance of the Twitter version of the XLM RoBERTa model, with the whole model trained lower cased version Swedish transactions dataset.

Whole model Twitter – XLM RoBERTa	Precision	Recall	F1
Overall	0.569	0.581	0.575
ORG	0.470	0.496	0.483
PER	0.073	0.081	0.077
LOC	0.806	0.686	0.741
MISC	0.860	0.902	0.881

Table 4.17: The conlleval chunk performance of the Twitter version of the XLM RoBERTa model, with the whole model trained on the Swedish transactions dataset.

AF-AI 3200 transactions	Precision	Recall	F1
Overall	0.847	0.852	0.849
ORG	0.839	0.824	0.831
PER	0.805	0.868	0.835
LOC	0.796	0.837	0.816
MISC	0.953	0.953	0.953

Table 4.18: The conlleval chunk performance of the AF-AI model fine-tuned on 3200 transactions from the Swedish transactions dataset.

BERT Multilingual 3200 transactions	Precision	Recall	F1
Overall	0.816	0.796	0.805
ORG	0.775	0.798	0.786
PER	0.854	0.795	0.824
LOC	0.796	0.683	0.735
MISC	0.953	0.943	0.948

Table 4.19: The conlleval chunk performance of the BERT multilingual model fine-tuned on 3200 transactions from the Swedish transactions dataset.

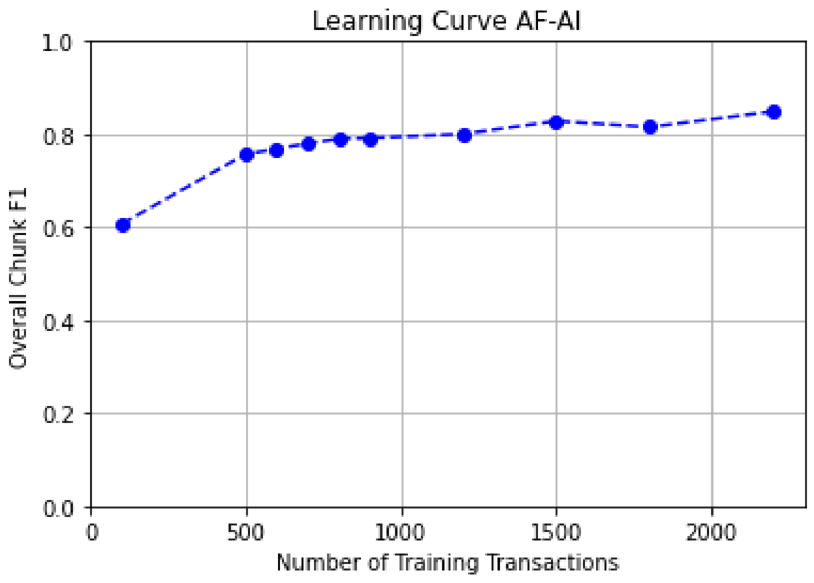


Figure 4.4: The learning curve of the AF-AI model, where we see how the overall chunk F1 score is affected by the number of training transactions.

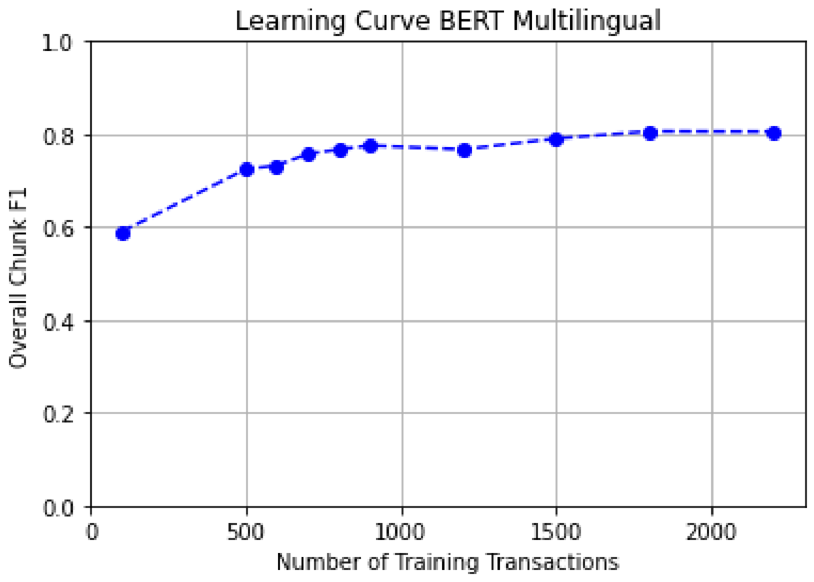


Figure 4.5: The learning curve of the BERT multilingual model, where we see how the overall chunk F1 score is affected by thenumber of training transactions.

Chapter 5

Application to Transaction Categorization Models

In this chapter, we describe an application of our NER model to improve an existing categorization application. More precisely, we combined our best performing NER model with Tink's models for transaction categorization.

1. Firstly, we tried to use the NER model for preprocessing the transaction descriptions, to see if we can decrease the size of our feature space while keeping the important information.

2. Secondly, we developed a multi-input model where we combined the transaction categorization model at Tink with features from our NER.

We used the transaction categorization models, datasets, and performance metrics as described in Sect. 2.8

We started from two existing categorization models used at Tink. Table 5.1 shows the performance of the current fastText model, while Table 5.2, shows the performance of the current XGBoost model. They defined our baselines.

An overview of the results for the different approaches when we apply NER to transaction categorization is shown in Table 5.3. Here we can see that both approaches using NER for cleaning the dataset end up performing worse than the fastText baseline. We can also see that we get a weighted mean KPI score of 0.793 with the multi-input model, which is 0.074 better than the fastText baseline and 0.011 better than the XGBoost baseline.

5.1 Using NER for Cleaning Dataset

In this section, we look at the performance of the Tink fastText model when using the smaller version of the AF-AI model trained on 900 training transactions, described in Sect. 4.2.1, for preprocessing of the transaction descriptions. We applied the following preprocessing rules:

Performance metrics	**Score**
Mode accuracy by frequency	0.696
Mode precision by frequency	0.790
Mode accuracy by amount	0.611
Mode precision by amount	0.758
Mode parent accuracy by amount	0.681
Categorization level	0.881
Weighted mean KPI score	0.719

Table 5.1: The performance metrics for the current fastText model at Tink.

Performance metrics	**Score**
Mode accuracy by frequency	0.769
Mode precision by frequency	0.816
Mode accuracy by amount	0.726
Mode precision by amount	0.795
Mode parent accuracy by amount	0.783
Categorization level	0.943
Weighted mean KPI score	0.782

Table 5.2: The performance metrics for the current XGBoost model at Tink.

1. If the description includes an organization, keep the organizations and remove the rest.
2. If the description does not includes an organization, but includes a miscellaneous entity (i.e payment provider, product, or app), keep the miscellaneous entity and remove the rest.
3. If the description neither includes an organization nor an miscellaneous entity, keep all of the named entities (i.e persons and locations) and remove the rest.
4. If no named entity is found, keep the description as it is.

Table 5.4 shows a couple of transactions after applying the procedure above to transaction descriptions. In the given examples, we can see that the model correctly picks out the correct named entities, that it can separate between payment providers and organizations, and that it does not find named entities when it is not supposed to.

The above procedure was first applied to the test set only, which gave us the results shown in Table 5.5, where we can see that the weighted mean KPI score goes down by 0.036 compared to the current fastText model. We then applied the procedure on the training and validation set as well and retrained the fastText model, which gave us the results shown in Table 5.6. The weighted mean KPI score goes up to 0.704, but is still 0.015 lower than the current fastText model.

We then tried another procedure where we kept all of the named entities the NER model has predicted, and removed everything that is not a named entity (except if there are no

Overview of results – Transaction categorization models	Weighted mean KPI score
NER Cleaning – Prioritized Named Entities	0.704
NER Cleaning – All Named Entities	0.691
Multi-input Model	0.793

Table 5.3: An overview of the results for the different approaches when we apply NER to transaction categorization.

Preprocessed transactions	Transactions after NER cleaning	Rule #
hemköp stockholm hor	hemköp	1.
oob visby 88 visby se	oob	1.
swish svenska spel	svenska spel	1.
betalning bg 123-4567 trängselskatt	betalning bg 123-4567 trängselskatt	4.
betalning bg 1234-5678 if skadeförs	if skadeförs	1.
paypal netflix	netflix	1.
mcd ingelsta norrkoping se	mcd	1.
amazonmktplc 2826v51f4 www amazon se	amazon	1.
ica kvantum taby c taby se	ica kvantum	1.
swish +46700000000	swish	2.
från lönekonto nik	nik	3.

Table 5.4: The preprocessed transactions before and after NER cleaning according to the process in Sect. 5.1. Note that any sensitive information in these examples like account numbers, phone numbers, and names have been replaced.

named entities, in which the description is kept as it is). We applied this procedure on the training, validation, and test sets and we got a weighted mean KPI score of 0.691, which is 0.024 lower than the current fastText model, which is shown in Table 5.7.

From our three experiments, we can see that none of them improved the performance of the transaction categorization. These procedures do not add any information to the model, they only remove information. Ideally, the information removed would be redundant, and removing it would decrease the size of your feature space and thus improve model performance, but this is not the case. This might be due to the model removing too much information. For example words like *hyra* (rent) which is not a named entity, but is important in determining the category. It could also be due to the predictions of the NER model not being accurate enough.

5.2 Multi-Input Model

In Sect. 5.1, we experimented with using the NER model for preprocessing our data, where the goal was to remove redundant information from the model. In this experiment, we instead used the NER model to add information to the model, by building a multi-input model.

In our baseline setup of the multi-input model, we combined the fastText model with the AF-AI NER model from Sect. 4.2.1 trained on 1700 transactions. We used three inputs:

Performance metrics	**Score**
Mode accuracy by frequency	0.675
Mode precision by frequency	0.740
Mode accuracy by amount	0.595
Mode precision by amount	0.686
Mode parent accuracy by amount	0.691
Categorization level	0.913
Weighted mean KPI score	0.683

Table 5.5: The performance metrics for the fastText model when the descriptions in the test set were cleaned using the procedure in Sect. 5.1.

Performance metrics	**Score**
Mode accuracy by frequency	0.677
Mode precision by frequency	0.810
Mode accuracy by amount	0.537
Mode precision by amount	0.770
Mode parent accuracy by amount	0.594
Categorization level	0.805
Weighted mean KPI score	0.704

Table 5.6: The performance metrics for the fastText model when the descriptions in the training, validation and test set were cleaned using the procedure in Sect. 5.1.

1. The first input was the probability score for every label output from the fastText model.

2. The second input was the named entities that the NER model extracted from the description (unless no entities were found, in which it would be the normal transaction), where each sub-word had an 768 long embedding vector extracted from the hidden states in the second last layer of the AF-AI model. This part of the input is padded to handle up to 17 sub-words (including tokens to mark out the beginning and the end of the description with extracted entities), as that was the maximum length of a transaction in the dataset.

3. The third input was the corresponding named entity group (ORG, PER, LOC and MISC) for every sub-word one-hot encoded with one additional tag corresponding to a sentence with no named entities (O). This input is padded to handle up to 15 sub-words, for the same reason as above, but without tokens to mark out the beginning and the end.

We built a model similar to that in chapter 7 of *Deep Learning with Python* (Chollet, 2021). We use Keras functional API where we pass in the 3 vectorized inputs explained above. These are then passed through a concatenation layer, using $axis = -1$ to create one long embedding vector. This vector is then passed into a 45-node fully connected layer with the softmax function as activation. We ran the training with early stopping and with a batch size of 128. Figure 5.1 shows an illustration of the multi-input model.

Performance metrics	**Score**
Mode accuracy by frequency	0.660
Mode precision by frequency	0.846
Mode accuracy by amount	0.513
Mode precision by amount	0.762
Mode parent accuracy by amount	0.559
Categorization level	0.780
Weighted mean KPI score	0.691

Table 5.7: The performance metrics for the fastText model when everything in the description has been removed except the named entities (except for when there are no named entities), in the train, validation, and test set.

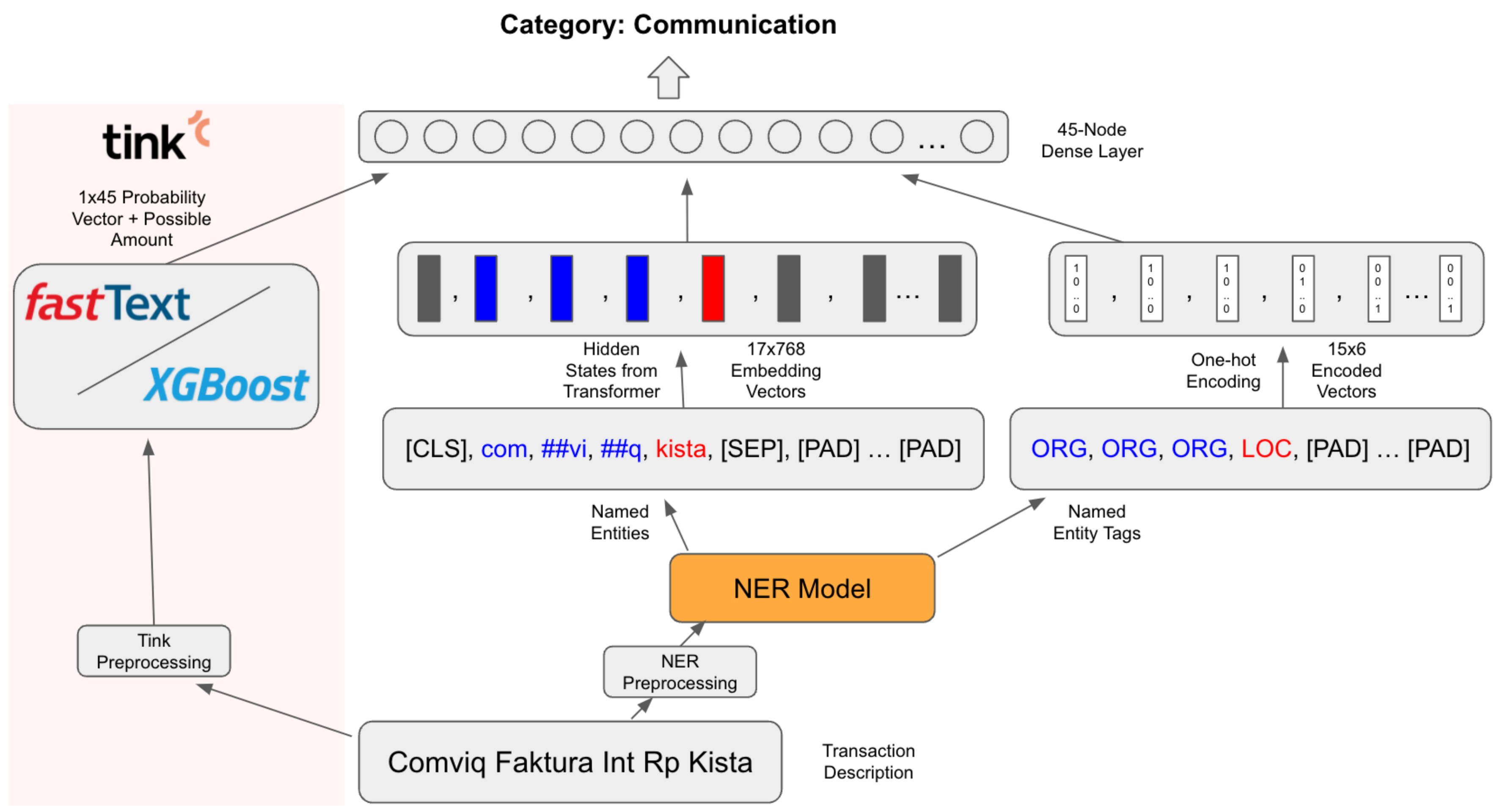


Figure 5.1: Illustration of the multi-input model. The NER model predicts *Comviq* to be an organization (blue), *Kista* to be a location (red) and the rest is discarded. These are then vectorized and sent into a 45 node dense layer together with the probability vector from fastText/XGBoost plus possible amount, to then be predicted as *Communication*.

We experimented with the following in the architecture for our model:

- Dropout
- Truncation
- Monitoring different metrics for early stopping
- Learning rate
- Additional hidden layers
- Including different data sources for training
- Optimizing the cutoff level for when to predict uncategorized
- Turning our sub-word embeddings into entity embeddings
- Adding amount as a feature
- Using XGBoost features instead of fastText
- Using a NER model trained on more data
- Adding an XGBoost model on top of the network, that takes the model outputs as input

Results from the model experimentation process can be found in Appendix A. Some notable findings were that iterating on our highest quality data source rather than using all available data yielded better results. We also found that reducing the feature space in different ways yielded better results. The number of sub-words in a transaction description can vary a lot, but with most of them being short. Thus, the two NER parts of the network needed to be padded to handle 17 sub-words, which resulted in a large part of the embeddings not being useful in a lot of cases. Therefore, we added truncation as well as taking the mean of the sub-word embeddings for a named entity group. This resulted in one embedding vector per named entity, rather than one for every sub-word (see illustration in Fig. 5.2). With this, the transaction descriptions only needed to be padded to handle up to 7 embedding vectors, which gave us better performance.

Another notable finding was that monitoring validation accuracy rather than validation loss in the early stopping yielded better performance. Monitoring validation accuracy instead of validation loss will generally converge towards a model with higher accuracy in predictions, but that is more unsure of its prediction. This is generally not something you want. But if we take into consideration how our performance metrics are set up (as shown in Table 2.2): When the model is unsure, it sets the prediction to *uncategorized*. What we found was when we monitor the validation accuracy instead of validation loss, the model will, in general, be more correct when it is confident in its predictions and when it is not confident, the transaction is set to uncategorized. This results in a higher score in the precision metrics, and a little bit lower score in the accuracy metrics, but with an overall positive effect on the weighted mean KPI score.

The final and best performing model was the baseline model with the following changes:

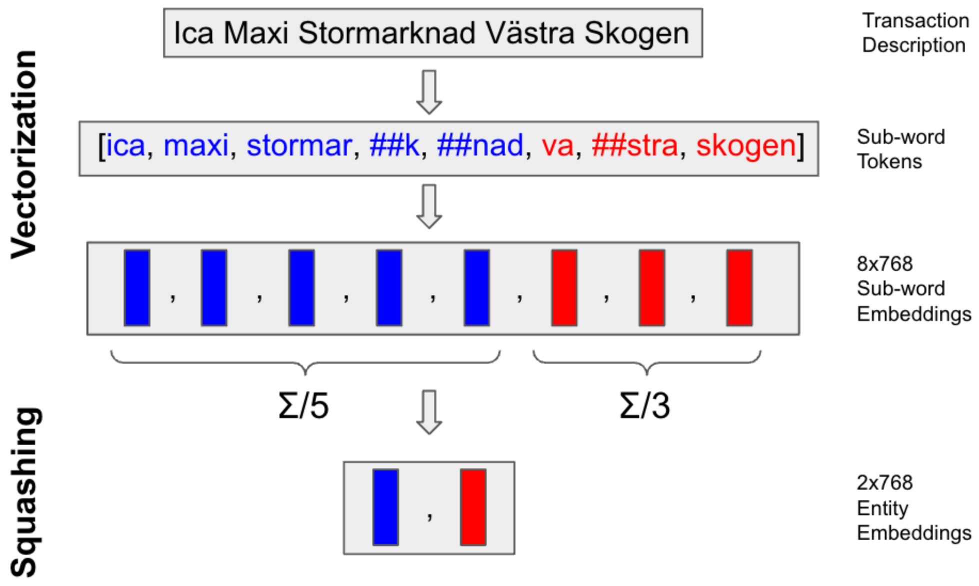

Vectorization
Squashing
Ica Maxi Stormarknad Västra Skogen
Transaction Description
[ica, maxi, stormar, ##k, ##nad, va, ##stra, skogen]
Sub-word Tokens
8x768 Sub-word Embeddings
Σ/5
Σ/3
2x768 Entity Embeddings

Figure 5.2: Illustration of squashing the sub-word embeddings into entity embedding. The transaction description is first standardized and tokenized into sub-word tokens. The tokens are then each assigned an embedding vector. A new entity embedding is then created based on the mean of all sub-word embeddings in the named entity. In this example, the transaction description goes from having eight embedding vectors to having two embedding vectors, one for the organization (blue) and one for the location (red).

- Only using some data sources, as explained above
- Adding truncation and embedding squashing
- Using validation accuracy as metric for early stopping
- Adding dropout
- Adding the amount of the transaction as a feature
- Optimizing the cutoff for when to predict uncategorized
- Using a learning rate of $2 \cdot 10^{-5}$
- Using features from XGBoost rather than fastText

The result of one run of our best model can be found in Table 5.9. Running the training procedure and evaluation of the model 5 times gives the following average and 95% confidence interval seen in Table 5.8. As we can see, the weighted mean KPI score is **0.074** above the Tink fastText model, and **0.011** above the Tink XGBoost model.

Weigted mean KPI score
0.793 ±0.004

Table 5.8: The average weighted mean KPI score and 95% confidence interval for our best model with 5 runs.

Performance metrics	Score
Mode accuracy by frequency	0.735
Mode precision by frequency	0.889
Mode accuracy by amount	0.682
Mode precision by amount	0.888
Mode parent accuracy by amount	0.713
Categorization level	0.827
Weighted mean KPI score	0.799

Table 5.9: The performance metrics for one run of our best model for transaction categorization.

5.2.1 Trade-off – Weighted mean KPI score and Categorization level

Up until this point, we have only been optimizing towards weighted mean KPI score, which balances high precision (i.e many accurate predictions of the transactions that are not set to uncategorized) and high overall accuracy. If the model has a high threshold, it will only make predictions when it is confident in them, resulting in a high precision and relatively lower accuracy. But some customers might have a requirement that the categorization level has to be above a certain level, or it could be the case that you can greatly increase the categorization level, while not really affecting the weighted mean KPI score.

Thus it is interesting to look at the tradeoff between these two metrics, which can be shown for one run of our best model in Figure 5.3. Here we can see that the best weighted mean KPI score is around a categorization level of ~0.83. Although there is just a marginal difference in the weighted mean KPI score if we were to increase the categorization level to 0.86–0.87, and also not that big of a difference if we wanted to get it up to ~0.9.

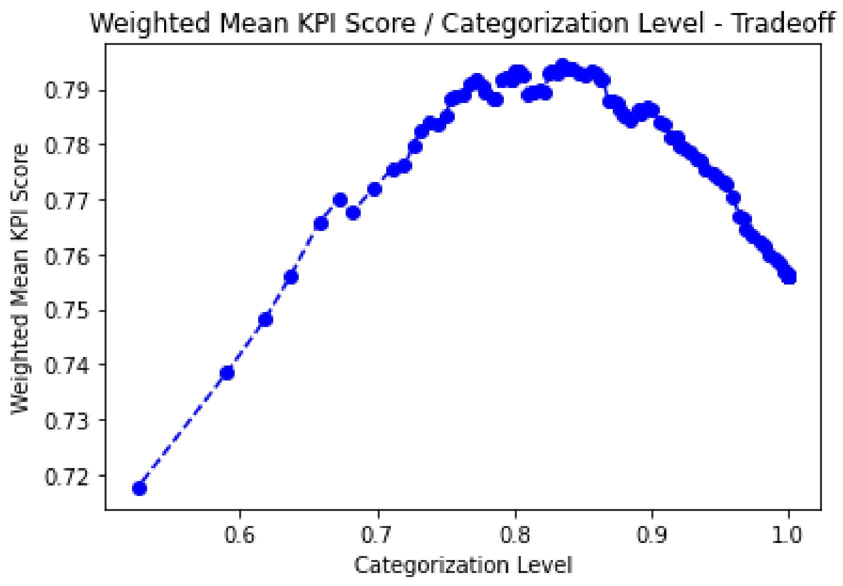


Figure 5.3: The trade-off between weighted mean KPI score and categorization level for one run of our best model.

Chapter 6

Conclusions

6.1 Discussion

When trying out different NER models, we could see that the performance of the model on the transaction description dataset depended a lot on how the vocabulary was set up, and more specifically if the vocabulary was case sensitive or case insensitive. It would therefore be interesting to do the tokenization process from scratch, where you use unlabeled transaction descriptions for creating your vocabulary and pre-training. There is a good chance this would increase the performance of the model, as well as give more flexibility in model choice. This would allow us to use, for example, DistilBERT (Sanh et al., 2019) or similar architectures to reduce inference time while achieving similar performance.

For our transaction categorization, we were able to increase the performance by developing a multi-input model that used features from our NER model. In the model development, we saw that there was a positive effect when reducing the amount of padding in the model, by using truncation and squashing our sub-word embeddings into entity embeddings. Another way to do it would be to properly implement masking, in which the model would not process the padding of the input.

It would also be interesting to include training of the embeddings in our multi-input model, as opposed to just using the pre-trained embeddings as input to the model. We saw great increases in performance when fine-tuning our NER models compared to just using the pre-trained ones. Therefore, there is a good chance we would see similar effects when doing it for the transaction categorization model. However, one can discuss when this goes too far into transaction categorization and too far out of the scope of NER, which was the focus of this thesis.

Finally, one can discuss the ethical aspects when it comes to NER on transaction descriptions. Some of the information extracted in NER can be of a sensitive nature, like personal identifiable information. Thus, one needs to take into careful consideration how this model is being used, and also make sure that the use of it is compliant with existing regulations. But

NER can also strengthen personal integrity, by being used to anonymize sensitive information in applications where transaction descriptions are being used.

6.2 Summary

In this thesis, we applied named entity recognition on Swedish transaction descriptions. We first annotated a dataset of Swedish transaction descriptions. We then used different transformer models that we fine-tuned on it. As pretrained transformer models, we used both Swedish and multilingual ones, including Swedish BERT models from the National Library of Sweden (Kungliga biblioteket) and the Swedish Public Employment Service (Arbetsförmedlingen), mBERT, and XLM-Roberta.

We then evaluated our models. Our best model was the fine-tuned AF-AI model from Sect. 4.5.1, with which we were able to reach an overall chunk F1 score of **0.849** despite only using 2200 transactions for training and transaction data being messy. To the best of our knowledge, this is the first published report for NER on transaction descriptions.

These results can be compared to the overall chunk F1 score from the National Library of Sweden's model of **0.927**, which can be considered as the current state-of-the-art model of Swedish NER, on a dataset that has proper context and proper grammar.

We applied named entity recognition to improve transaction categorization. For this, we combined the NER models we created to existing categorization models. We showed that we could improve the performance of the categorization model by developing a multi-input model, where we added features from the NER model to the current transaction categorization model. We got a weighted mean KPI score of **0.793** which is **0.011** better than the current best model on the same dataset.

6.3 Future work

Some of the topics that would be interesting to dive into were explained already in Sect. 6.1, including doing pre-training and tokenization from scratch with unlabeled transaction descriptions, and implementing masking and training the embeddings for the transaction categorization model. It would also be interesting to expand the NER into different markets and for different clusters (most of the large customers have their own models, and therefore the data can be different), as well as to do further research of the model performance with more labeled data. Furthermore, one could experiment with adding a CRF layer (conditional random fields) on top of the NER model. There were common patterns in the tag sequences which we could see in Table 3.2 and Figure 3.1 that could benefit from such a layer.

Two topics for future work that we want to explain in further detail are *entity linking* and *transformer models for transaction categorization*.

6.3.1 Entity Linking

When doing NER on Swedish transaction descriptions, the model will predict, for example, *Ica Maxi Stormarknad*, *Maxi Ica Stormarknad*, *Ica Nära*, *Ica Supermarket* etc. to be organizations. However, it has no idea that these are all branches within the organization *Ica*. Also, we do not get any information of what type of organization this is. This could be done by expanding into entity linking, where you can have a system that links all Ica branches to the same unique identifier and a knowledge base like Wikimedia.

In the Wikimedia example, you have Wikipedia pages that consists of 3 layers (Altay, 2020), the body of text, the links, and a knowledge graph, as shown in Figure 6.1. This could be used to confirm the entities, or be leveraged in the process of creating training data for your NER system. It could also be used to get additional knowledge of entities in the transaction description. We could, for example, find out the type of organization the model has predicted; we could use the body of text to create a representation of an entity that can be used to find similar entities; and we can leverage the fact that Wikipedia pages usually exist in multiple languages to use already known information about organizations in one market when transitioning into a new market, for example.

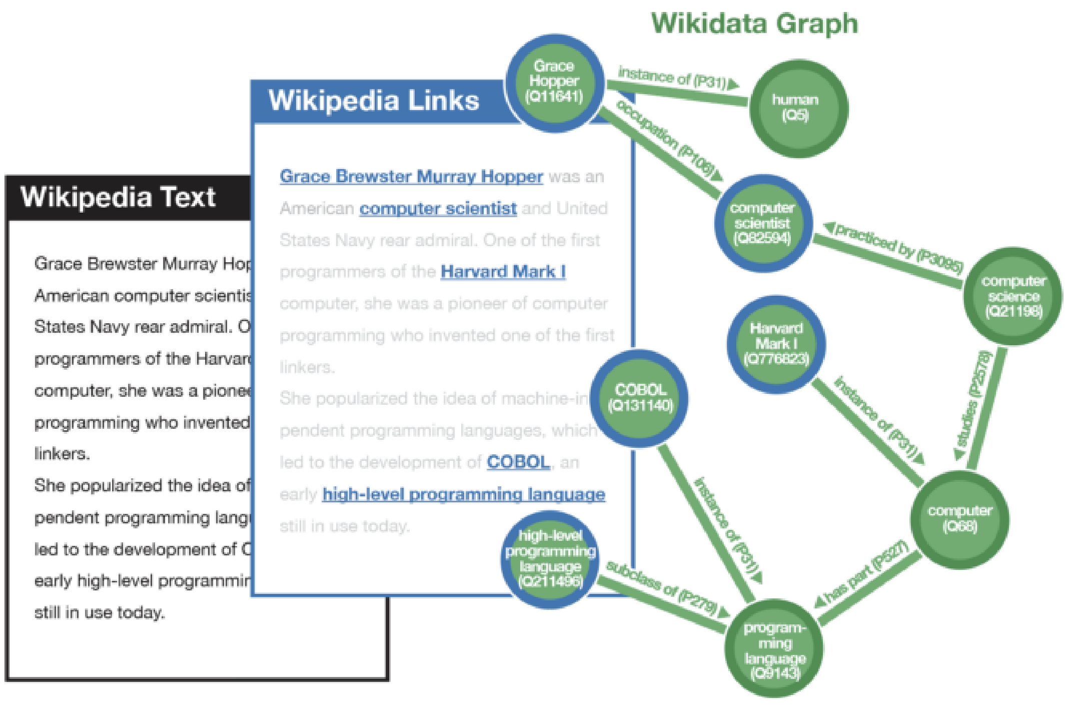

Wikidata Graph
Wikipedia Links
Wikipedia Text
Grace Hopper (Q11641)
instance of (P31)
human (Q5)
occupation (P106)
computer scientist (Q82594)
practiced by (P3095)
computer science (Q21198)
Harvard Mark I (Q776823)
instance of (P31)
studies (P2578)
COBOL (Q131140)
instance of (P31)
computer (Q68)
has part (P527)
high-level programming language (Q211496)
subclass of (P279)
programming language (Q9143)
Grace Brewster Murray Hopper was an American computer scientist and United States Navy rear admiral. One of the first programmers of the Harvard Mark I computer, she was a pioneer of computer programming who invented one of the first linkers.
She popularized the idea of machine-independent programming languages, which led to the development of COBOL, an early high-level programming language still in use today.

Figure 6.1: The Wikipedia page of Grace Hopper divided into 3 layers: the body of text, the links, and a knowledge graph. Image source: Altay (2020).

6.3.2 Transformers for Transaction Categorization

In this thesis, we combined a transaction categorization model with features from our NER model to create a new model for transaction categorization. Another approach for a transaction categorization model would be to train a transformer model without including any NER. This would make the architecture simpler as it does not combine models and tools from a lot of different sources, like the multi-input model. Using a transformer model would, in a natural way, solve the issue with masking, padding, and non-trained embeddings described in Sect. 6.1, as well as not requiring the model to have annotated transactions for both NER and transactions categorization.

There are a couple of things which make us think this would be a successful approach. Firstly, the transformer models are general purpose models and we had good results for the task of NER, which uses the same data. Secondly, François Chollet described in his book *Deep Learning with Python* (Chollet, 2021) a rule of thumb for text classification that when the ratio between the number of data samples to the mean sample length is higher than 1500, transformer models tend to perform better than other alternatives. This is the case for transaction data, since it has a short mean sample length and the number of data samples is generally high. Thirdly, there is a transformer model online where it seems they have gotten good results for transaction categorization for Italian transactions (Grella, 2021), which are – based on own experience – very messy.

References

(1995). *Sixth Message Understanding Conference (MUC-6): Proceedings of a Conference Held in Columbia, Maryland, November 6-8, 1995.*

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Altay, G. (2020). Introducing the kensho derived wikimedia dataset.

Barbieri, F., Anke, L. E., and Camacho-Collados, J. (2021). Xlm-t: A multilingual language model toolkit for twitter.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754.

Chollet, F. (2017). *Deep Learning with Python*. Manning Publications, first edition.

Chollet, F. (2021). *Deep Learning with Python*. Manning Publications, second edition.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2019). Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Grella, M. (2021). Autonlp bank transaction classification. `https://huggingface.co/mgrella/autonlp-bank-transaction-classification-5521155`. [Online; accessed 21-Febuary-2022].

Grishman, R. and Sundheim, B. (1995a). Appendix C: Named entity task definition (v2.1). In *Sixth Message Understanding Conference (MUC-6): Proceedings of a Conference Held in Columbia, Maryland, November 6-8, 1995.*

Grishman, R. and Sundheim, B. (1995b). Design of the MUC-6 evaluation. In *Sixth Message Understanding Conference (MUC-6): Proceedings of a Conference Held in Columbia, Maryland, November 6-8, 1995*.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Klintberg, A. (2020). Swedish manually annotated NER. `https://github.com/klintan/swedish-ner-corpus`. [Online; accessed 23-September-2021].

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.

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

Marcelino, P. (2018). Transfer learning from pre-trained models. `https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751`. [Online; accessed 01-March-2022].

Mari, A., Bromley, T. R., Izaac, J., Schuld, M., and Killoran, N. (2020). Transfer learning in hybrid classical-quantum neural networks. *Quantum*, 4:340.

Meta (2022). Named entity recognition. `https://paperswithcode.com/task/named-entity-recognition-ner`. [Online; accessed 21-March-2022].

OpenStreetMap contributors (2017). Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Peters, M. E., Ammar, W., Bhagavatula, C., and Power, R. (2017). Semi-supervised sequence tagging with bidirectional language models. *CoRR*, abs/1705.00108.

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

Ruder, S. (2022). Nlp-progress: Named entity recognition. `https://nlpprogress.com/english/named_entity_recognition.html`. [Online; accessed 23-Febuary-2022].

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Schulz, M. and Schmit, N. (2015). Directive (eu) 2015/2366. `https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32015L2366`. [Online; accessed 28-Febuary-2022].

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

Språkbanken (2012). Stockholm-umeå corpus 3.0.

Stollenwerk, F. (2020). Arbetsförmedlingen's swedish bert models.

Stollenwerk, F. (2021). nerblackbox: a python package to fine-tune transformer-based language models for named entity recognition.

Tink (2022). This is tink. `https://tink.com/press/`. [Online; accessed 28-Febuary-2022].

Tjong Kim Sang, E. F. (2000). Output example conlleval. `https://www.clips.uantwerpen.be/conll2000/chunking/output.html`. [Online; accessed 28-Febuary-2022].

Tjong Kim Sang, E. F. (2002). Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*.

Tjong Kim Sang, E. F. and De Meulder, F. (2003a). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In Daelemans, W. and Osborne, M., editors, *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada.

Tjong Kim Sang, E. F. and De Meulder, F. (2003b). Language-independent named entity recognition (ii). `https://www.clips.uantwerpen.be/conll2003/ner/`. [Online; accessed 28-Febuary-2022].

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

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Appendices

Appendix A
Experiments – Transaction Categorization

The following tables show the performance of different experiments with the multi-input model, starting from a baseline experiment as explained in 5.2, and gradually adding and changing things. The models are run on only the highest quality data source, and including other data sources was tried, but with no success. We also tried different configurations by adding a hidden layer, but we did not find any success doing this.

Performance metrics	**Score**
Mode accuracy by frequency	0.689
Mode precision by frequency	0.696
Mode accuracy by amount	0.559
Mode precision by amount	0.566
Mode parent accuracy by amount	0.621
Categorization level	0.989
Weighted mean KPI score	0.627

Table A.1: The performance metrics of the baseline model.

Performance metrics	Score
Mode accuracy by frequency	0.702
Mode precision by frequency	0.716
Mode accuracy by amount	0.584
Mode precision by amount	0.600
Mode parent accuracy by amount	0.643
Categorization level	0.980
Weighted mean KPI score	0.651

Table A.2: The performance metrics when adding dropout to the model.

Performance metrics	Score
Mode accuracy by frequency	0.703
Mode precision by frequency	0.727
Mode accuracy by amount	0.597
Mode precision by amount	0.629
Mode parent accuracy by amount	0.659
Categorization level	0.968
Weighted mean KPI score	0.667

Table A.3: The performance metrics when squashing the sub-word embeddings into entity embeddings, as shown in Fig 5.2. This results in the sub-word embeddings being padded to length 8 and the NER tags to length 6. We also exclude dropout.

Performance metrics	Score
Mode accuracy by frequency	0.713
Mode precision by frequency	0.739
Mode accuracy by amount	0.616
Mode precision by amount	0.654
Mode parent accuracy by amount	0.687
Categorization level	0.964
Weighted mean KPI score	0.684

Table A.4: The performance metrics when adding dropout again.

Performance metrics	Score
Mode accuracy by frequency	0.738
Mode precision by frequency	0.756
Mode accuracy by amount	0.656
Mode precision by amount	0.682
Mode parent accuracy by amount	0.718
Categorization level	0.977
Weighted mean KPI score	0.711

Table A.5: The performance metrics when we monitor validation accuracy instead of validation loss in our early stopping.

Performance metrics	Score
Mode accuracy by frequency	0.661
Mode precision by frequency	0.889
Mode accuracy by amount	0.596
Mode precision by amount	0.871
Mode parent accuracy by amount	0.620
Categorization level	0.753
Weighted mean KPI score	0.744

Table A.6: The performance metrics when we optimize the cutoff level (based on weighted mean KPI score) as to when to set a transaction to uncategorized.

Performance metrics	Score
Mode accuracy by frequency	0.715
Mode precision by frequency	0.800
Mode accuracy by amount	0.654
Mode precision by amount	0.750
Mode parent accuracy by amount	0.712
Categorization level	0.894
Weighted mean KPI score	0.734

Table A.7: The performance metrics when adding amount as an input feature.

Performance metrics	Score
Mode accuracy by frequency	0.698
Mode precision by frequency	0.888
Mode accuracy by amount	0.640
Mode precision by amount	0.856
Mode parent accuracy by amount	0.681
Categorization level	0.786
Weighted mean KPI score	0.772

Table A.8: The performance metrics when decreasing the learning rate to 2e-5.

Performance metrics	**Score**
Mode accuracy by frequency	0.689
Mode precision by frequency	0.915
Mode accuracy by amount	0.582
Mode precision by amount	0.919
Mode parent accuracy by amount	0.603
Categorization level	0.754
Weighted mean KPI score	0.771

Table A.9: The performance metrics when using features from Tink's XGBoost model instead of Tink's fastText model. We also monitor validation loss for early stopping, and don't use amount as a feature.

Performance metrics	**Score**
Mode accuracy by frequency	0.739
Mode precision by frequency	0.874
Mode accuracy by amount	0.666
Mode precision by amount	0.858
Mode parent accuracy by amount	0.702
Categorization level	0.846
Weighted mean KPI score	0.784

Table A.10: The performance metrics when we go back to monitoring validation accuracy for early stopping.

Performance metrics	**Score**
Mode accuracy by frequency	0.734
Mode precision by frequency	0.888
Mode accuracy by amount	0.668
Mode precision by amount	0.872
Mode parent accuracy by amount	0.713
Categorization level	0.827
Weighted mean KPI score	0.792

Table A.11: The performance metrics when we use amount as an input feature again.

Performance metrics	**Score**
Mode accuracy by frequency	0.735
Mode precision by frequency	0.889
Mode accuracy by amount	0.682
Mode precision by amount	0.888
Mode parent accuracy by amount	0.713
Categorization level	0.827
Weighted mean KPI score	0.799

Table A.12: The performance metrics when we truncate the transactions, so the maximum sub-word length is 6.

Performance metrics	**Score**
Mode accuracy by frequency	0.765
Mode precision by frequency	0.789
Mode accuracy by amount	0.707
Mode precision by amount	0.752
Mode parent accuracy by amount	0.771
Categorization level	0.969
Weighted mean KPI score	0.759

Table A.13: The performance metrics when we try to add an XGBoost model that takes the output of our model as input (similar to how it is used in the fastText case).

Performance metrics	**Score**
Mode accuracy by frequency	0.762
Mode precision by frequency	0.838
Mode accuracy by amount	0.701
Mode precision by amount	0.829
Mode parent accuracy by amount	0.749
Categorization level	0.909
Weighted mean KPI score	0.785

Table A.14: The performance metrics when we use the NER model is trained on 3200 transactions instead of the one trained on 1700 transactions.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-03-17

EXAMENSARBETE Named Entity Recognition on Transaction Descriptions
STUDENT Nik Johansson
HANDLEDARE Pierre Nugues (LTH), Trent Woodbury (Tink AB)
EXAMINATOR Marcus Klang (LTH)

Klassificering av beståndsdelarna i transaktionsbeskrivningar

POPULÄRVETENSKAPLIG SAMMANFATTNING **Nik Johansson**

Med en stor ökning av applikationer som använder sig av finansiell data ökar också behovet av att på ett strukturerat sätt kunna extrahera viktig information från transaktionsbeskrivningar. Detta arbete introducerar vi en modell som i transaktionsbeskrivningar kan identifiera organisationer, geografiska platser, personer, med mera.

I slutet av 2015 röstades EU direktivet *PSD2* igenom som innebar att konsumenterna nu har möjlighet att själva bestämma vilka som får tillgång till deras finansiella data. Detta har öppnat upp för produkter och applikationer som på ett innovativt sätt använder sig av denna data, vilket brukar benämnas som *open banking*.

För att kunna möjliggöra fler och bättre produkter och applikationer som baseras på transaktionsdata krävs system som på ett strukturerat sätt kan identifiera och klassificera viktiga komponenter i transaktionsbeskrivningar, så som organisationer, geografiska platser, personer, betalningsleverantörer, produkter, och appar. I detta examensarbete introducerar vi, så vitt vi vet, den första publicerade rapporten som tar fram en modell som kan identifiera och klassificera dessa viktiga komponenter i transaktionsbeskrivningar.

Modellen är baserad på maskinlärning: en metod där systemet lär sig själv att lösa problemet genom att träna på sammanställd data. I arbetet lyckas vi utveckla en modell som kan identifiera och klassificera dessa komponenter med en träffsäkerhet som inte alls är långt ifrån de bästa modellernas prestation på enklare data (så som nyhetsartiklar). Genom smart användning av datan lyckas vi få bra resultat, trots att tillgången på data var mycket begränsad, samt att transaktionsbeskrivningar är betydligt stökigare än nyhetsartiklar och dylikt.

Transaktionsbeskrivningar

Hemköp$_{\text{ORG}}$ Stockholm$_{\text{LOC}}$ Hor$_{\text{LOC}}$

Atm Kontanten1073 Helsingborg$_{\text{LOC}}$

Swish$_{\text{MISC}}$ Svenska Spel$_{\text{ORG}}$

Överf N Johansson$_{\text{PER}}$

Betalning Bg 123-4567 Trängselskatt

Betalning Bg 1234-5678 If Skadeförs$_{\text{ORG}}$

Klarna$_{\text{MISC}}$ Apotea$_{\text{ORG}}$

7 Eleven$_{\text{ORG}}$

Ica Kvantum$_{\text{ORG}}$ Taby C$_{\text{LOC}}$ Taby$_{\text{LOC}}$ Se$_{\text{LOC}}$

Swish$_{\text{MISC}}$ Från Nik Johansson$_{\text{PER}}$

Det visar sig även att ett sådant här system kan användas för att förbättra system för kategorisering av transaktioner. Vår modell tillför information till systemet för kategorisering av transaktioner, så att det blir lättare att avgöra ifall en transaktion är livsmedel, ett restuarangbesök, eller dylikt.