UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Estimating Context-Specific Subjective Content Descriptions using Transformer Language Models

Bestimmung kontextspezifischer subjektiver Inhaltsbeschreibungen mit Transformer-Sprachmodellen

Masterarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Magnus Bender

ausgegeben und betreut von
Prof. Dr. Ralf Möller

mit Unterstützung von
Felix Kuhr und PD Dr. Amir Madany Mamlouk

Lübeck, den 1. September 2021

Kurzfassung Eine wichtige Aufgabe im Bereich der natürlichen Sprachverarbeitung ist es, natürlichsprachliche Texte zu verstehen. Beim Lesen eines Textes nutzen Menschen Erfahrungen über die Welt, in der sie leben. Daher benötigt ein Agent, der einen Text verarbeitet, ähnliche Erfahrungen. Beispielsweise würden Menschen, wenn sie das Wort *Bank* in einem Text über das Thema Geld lesen, davon ausgehen, dass es sich bei der *Bank* um ein Finanzinstitut handelt und nicht um etwas, auf dem man sitzen kann. Das Ziel von Subjective Content Descriptions (SCDs) ist es, Daten als Annotationen zu natürlichsprachlichen Texten hinzuzufügen. In unserem Beispiel könnte die SCD ein weiterer Satz sein, der die *Bank* als Finanzinstitut definiert, oder ein Link zu einer Entität *Finanzinstitut* in einer externen Quelle.

In dieser Arbeit skizzieren wir die Bedeutung von SCDs und vergleichen Techniken zur Bestimmung von textuellen SCDs. Wir verwenden das bekannte Sprachmodell Bidirectional Encoder Representations from Transformers (BERT) [DCLT19] und vergleichen BERTs Leistung mit Algorithmen für das Most Probably suited SCD [KBBM19] und inline SCD [BBG^{+}21b] Problem.

Abstract An important task in the area of Natural Language Processing is language understanding. While reading a text, humans rely on experiences of the world they live in. Thus, an agent processing a text needs similar experiences, e.g., when humans read the word *bank* in a text dealing about money, they would assume the *bank* to be a financial institution and not something to sit on. The goal of Subjective Content Descriptions (SCDs) is to add data in form of annotations to natural language texts. In our example, the SCD could be another sentence, defining the *bank* to be a financial institution, or a link to an entity *financial institution* in an external source.

In this thesis, we outline the importance of SCDs and compare techniques for estimating textual SCDs. We use the well-known Bidirectional Encoder Representations from Transformers (BERT) [DCLT19] language model and compare BERT's performance against algorithms solving the Most Probably suited SCD [KBBM19] and inline SCD [BBG$^+$21b] problem.

List of Variables and Notations

This list provides an overview of the variables and notations used in this thesis. All variables will be introduced and explained in the chapters of the thesis, too.

- Chapter 1 – General notations

w_i:	Word from a given vocabulary $\mathcal{V} = \{w_1, \ldots, w_L\}$, $L \in \mathbb{N}$
d:	Document, a sequence of words $(w_1^d, \ldots, w_N^d)$, $N \in \mathbb{N}$, where each word $w_i^d \in \mathcal{V}$
$w_{i,j}^d$:	Subsequence of words $w_{i,j}^d = (w_i^d, ..., w_j^d)$ from document d where $1 \leq i < j \leq N$
$\mathcal{D}$:	Corpus of documents $\{d_1, \ldots, d_{\lvert\mathcal{D}\rvert}\}$, $\lvert\mathcal{D}\rvert \in \mathbb{N}$
t:	SCD, a sequence of words $t = (w_1', ..., w_l')$, $l \in \mathbb{N}$
(t, ρ):	Located SCD t, where ρ defines a position, $\rho \in [i, j]$, in d
$g(d)$:	Set containing located SCDs $\{(t_j, \rho_j)\}_{j=1}^M$, $M \in \mathbb{N}$, for d
$win_{d,\rho}$:	SCD window $win_{d,\rho} \subseteq w_{1,N}^d$, a sequence of words in d surrounding the word w_ρ^d in d
$I(w^d, win_{d,\rho})$:	Influence value for each word $w^d \in win_{d,\rho}$ representing distance of w^d and position ρ

- Chapter 2 – Subjective Content Descriptions

$\delta(\mathcal{D})$:	SCD matrix shaped $M \times L$
$\mathcal{W}$:	Set of similarity values $sim \in \mathcal{W}$ yielded by the MPS²CD algorithm
th:	Threshold for the threshold-based algorithm solving the iSCD problem

- Chapter 3 – Transformer Language Models

x_i:	Inputs of a unit in a neural network, $i = 1, ..., n$
y_i:	Outputs of a unit in a neural network, $i = 1, ..., n$
α_i:	Weights of a unit in a neural network, $i = 1, ..., n$
φ:	Activation function of a perceptron
q, k, v:	Query, key, and value vector, used as input for scaled dot-product attention

Q, K, V:	Matrices of multiple q, k, v each
e_w:	Embedded representation of word w
W_q, W_k, W_v:	Matrices of TLM to derive query, key and value
h:	Number of scaled dot-product attentions used by the multi-head attention
$E_1, ..., E_N$:	Embedded input vectors for e.g. BERT
$T_1, ..., T_N$:	Encoded output vectors of e.g. BERT

- Chapter 4 – Transformer Language Models for Subjective Content Descriptions

c_1, c_2:	Two different contexts of corpora
$\mathcal{D}_{c_1}$:	Corpus containing documents from c_1
$g(\mathcal{D}_{c_1})$:	Set of SCDs matching corpus' context

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Quellen und Hilfsmittel angefertigt zu haben.

(Magnus Bender)
Lübeck, den 1. September 2021

Contents

1. Introduction

The most intuitive way of human communication is natural language. In a modern world, humans do not only interact with other humans. Much interaction also takes place against software, so-called agents. We follow the idea that an agent is a program assigned a task to fulfil. From an agent-theoretic perspective, an agent is a rational, autonomous unit acting in a world, fulfilling its task. For example, the task of an agent is to provide a document retrieval service, i.e., given a document from a user the agent returns a ranked list of similar documents. We assume the agent maintains a corpus of text documents and then returns documents from its corpus.

To concretize the example, the agent provides a search engine for browsing publications of the category *computer science*. Such an agent has to maintain a corpus of text documents consisting of relevant publications for its users. As continuously new publications are made available on web sites in the internet, the agent has to update its corpus with new documents. Therefore, the agent might consist of a crawler scanning the internet for relevant web sites. For simplicity, we may interpret a web site found by the crawler as a document of plain natural language texts. Now, given such document from a web site, the agent has to decide what to do with the document. The agent may add the document to its corpus or ignore it, because it does not match the defined category *computer science*. However, there are more possibilities, a publication found by the agent may be already contained in the corpus or it may be a revised or extended version of an already contained publication. Besides maintaining the documents in the corpus, the agent has to answer queries. A query send by a user to the agent is also a natural language text to which the agent has to return similar documents as a result.

One of the key competences of an agent providing a document retrieval service is to process and understand natural language. Therefore, the field of Natural Language Processing (NLP) became more and more important in the last decades. Current NLP approaches mostly use neural networks, for tasks like calculating vector representations of texts, so-called embeddings. Furthermore, neural networks are used to translate, summarize, and even generate text.

A remarkable improvement in the field of NLP was reached by Transformer Language Models (TLMs) [VSP+17]. TLMs use a special architecture of neural networks and can, in contrast to the approaches used before, handle much larger amounts of data. Thus, TLMs allow to build larger and more powerful models, while they can be used for the same tasks as neural networks before, e.g., automatically translate text documents and generate vector representations of text documents. In this thesis, we use the TLM Bidirectional Encoder Representations from Transformers (BERT) [DCLT19], which is an encoder and calculates vector representations for text documents.

However, a vector representation gained from BERT does not directly help to understand a text document, as it is only a different representation of the same document. Humans reading a text document rely on experiences of the world they live in. Though, an agent processing a text document does not have such experiences. Thus, an agent needs further data associated with a text document it processes. In this thesis, we use the vector representations gained from BERT to associate Subjective Content Descriptions (SCDs) [KBBM19] with text documents. SCDs are location-specific data making the content near the SCD's location explicit by providing descriptions, references, or explanations. For example, using SCDs the previously described agent providing a search engine can automatically take the decision if a found publication is an related, unrelated, revised, or extended version of the publications already contained in its corpus [KBBM19].

In this thesis, we are dedicated to follow problems in the field of SCDs: (i) An agent may be faced with documents which lack associated SCDs needed by the agent. Thus, the agent first has to associate SCDs with the given document only given the document itself. The Most Probably suited SCD (MPS^{2}CD) problem [KBBM19] formalizes associating *suitable* SCDs to documents. (ii) An agent may be faced with documents where *content* and SCDs are interleaved. To solve the agent's task, it first has to separate *content* and SCDs. The inline SCD (iSCD) problem [BBG$^+$21b] formalizes separating *content* from SCDs in a document.

Both problems were introduced in [KBBM19, BBG$^+$21b]. Also, the authors propose a solution for both problems using an SCD-word distribution matrix as basic model. In this thesis, we describe how to use BERT for both problems and compare the performance using BERT and the SCD-word distribution matrix.

We divide this thesis into four chapters:

Subjective Content Descriptions We introduce SCDs in general and describe in detail the MPS^{2}CD and iSCD problem and the solutions introduced by Kuhr et al. [KBBM19, BBG$^+$21b]. We also outline differences in our usage to make the solutions comparable with TLM-based solutions in the evaluation.

Transformer Language Models We introduce TLMs and especially the attention mechanism as well as the architecture of Transformers. At the end of the chapter we give an introduction to BERT.

Transformer Language Models for Subjective Content Descriptions We show multiple approaches to solve the iSCD problem using BERT in Section 4.1 and estimate MPS²CDs for new documents with BERT in Section 4.2. At the end of the chapter we describe how to use BERT for detecting the context of a document and associating a corpus with context-specific SCDs.

Evaluation For the evaluation of our approaches we mainly use the well-known 20 newsgroups data set[1] and generate SCDs using definitions from the online dictionary Wiktionary[2]. We compare the performance of the approaches using the SCD-word distribution matrix against the approaches using BERT for the MPS²CD and iSCD problem.

Finally, we also show the performance of all approaches on a corpus containing documents from different contexts.

1.1. Related Work

Adding data to corpora of text documents has been investigated for a long time. Often the data associated with a corpus is denoted an annotation. Thus, an SCD is an annotation subjectively describing the content depending on the corpus' context. Commonly, we assume all systems adding annotations provide a value for an agent solving a task using the annotated corpus.

The Brown Corpus [Mav69] is one of the first corpora used to analyze natural language. First, the distribution of words among different categories and contexts of natural language was analyzed. Later, *part-of-speech* tags were added, these tags can already be interpreted as annotations assigning a class to each word.

In the beginning of natural language annotation, most annotations had to be manually added to the corpora. Even today, crowdsourcing can be used to manually annotate text documents [SBDS14]. However, manually adding annotations is a time consuming task. Thus, semi-automatic and automatic annotation systems were developed, e.g, DBpedia[3] and OpenCalais[4].

Since 2017, TLMs have shown that they are a powerful technique processing natural language. TLMs translate text documents [VSP$^+$17], generate sentences [RWC$^+$19] and automatically generate titles for papers based on the abstracts [MDSS21].

[1]http://qwone.com/~jason/20Newsgroups/
[2]https://en.wiktionary.org/
[3]https://www.dbpedia.org/
[4]http://www.opencalais.com/

In this thesis, we show how to apply TLMs as annotation systems for SCDs. We focus on SCDs containing additional textual data. However, an agent may gain larger benefits from SCDs containing structured data representing underlying structures and relations in a text document.

Beltagy et al. [BRC$^+$16] describe a system for textual entailment. The system transforms sentences in logical representations, builds a knowledge base, and performs probabilistic inference to predict whether a sentence entails another sentence. For example, an SCD could be an entailed sentence in its textual representation, again. However, the SCD could also be the logical representation of the sentence. Such logical representation would allow an agent to check text documents for coherence and detect mismatching sentences.

In contrast, Shanahan [SCBC20] detaches from language. He defines *Common Sense* as capturing the deep structures of the world. *Common Sense* is not about detecting the entailment of two sentences, but about understanding what each sentence means in its full depth.

1.2. Notations

Before we introduce SCDs and TLMs, we formalize our setting of a corpus.

- A word w_i, $i = 1, ..., L$, is a basic unit of discrete data from a vocabulary $\mathcal{V} = \{w_1, \ldots, w_L\}$, $L \in \mathbb{N}$.
- A document d is defined as a sequence of words $(w_1^d, \ldots, w_N^d)$, $N \in \mathbb{N}$, where each word $w_i^d \in d$ is an element of vocabulary $\mathcal{V}$.
- A subsequence of words from d can be represented as $w_{i,j}^d = (w_i^d, ..., w_j^d)$ where $1 \leq i < j \leq N$. Commonly used subsequences are sentences, they are defined as a sequence of words terminated by punctuation symbols like ".", "!", or "?".
- A corpus $\mathcal{D}$ represents a set of documents $\{d_1, \ldots, d_{|\mathcal{D}|}\}$, $|\mathcal{D}| \in \mathbb{N}$.
- Variables marked with an apostrophe describe another object of the same type, e.g., d and d' are documents containing a different sequence of words.
- Sequences of words or single words not assigned to a document may be written as $w_{i,j}$ instead of $w_{i,j}^d$.
- An SCD $t = (w'_1, ..., w'_l)$, $l \in \mathbb{N}$, is a sequence of words. The SCD t can be associated with a position $\rho \in [i, j]$ in a document d. We use the term located SCD interchangeably for associated SCD and represent a located SCD t by the tuple (t, ρ).

- For each document $d \in \mathcal{D}$ there exists a set g denoted as SCD set containing a set of M located SCDs $\{(t_j, \rho_j)\}_{j=1}^{M}$. Given a document d or a set g, the terms $g(d)$ and $d(g)$ refer to the set of located SCDs associated with document d and the corresponding document d, respectively. The set of all located SCD tuples in $\mathcal{D}$ is given by $g(\mathcal{D}) = \bigcup_{d \in \mathcal{D}} g(d)$.
- For each located SCD $(t, \rho) \in g(d)$ there exists an SCD window $win_{d,\rho} \subseteq w_{1,N}^{d}$ that refers to a sequence of words in d surrounding the word w_{ρ}^{d}. In our case the SCD window is represented by the sentence w_{ρ}^{d} belongs to.
- Each word $w^d \in win_{d,\rho}$ is associated with an influence value $I(w^d, win_{d,\rho})$ representing the distance in the text between w^d and position ρ. The closer w^d is positioned to ρ in $win_{d,\rho}$, the higher its corresponding influence value $I(w^d, win_{d,\rho})$. The influence value is chosen according to the task and might be distributed binomial, linear, or constant.

2. Subjective Content Descriptions

Kuhr et al. have introduced SCDs in [KBBM19]. SCDs provide additional location-specific data for documents. The data provided by SCDs may be of various types, like additional definitions or links to knowledge graphs. The theory of SCDs is not restricted to text documents annotated with additional text. However, in our evaluation and use-cases we use text documents annotated with additional textual definitions.

Aforementioned, Kuhr et al. use an SCD-word distribution represented by a matrix when working with SCDs. The SCD-word distribution matrix, in short SCD matrix, can be interpreted as a generative model. A generative model for SCDs is characterized by the assumption that the SCDs generate the words of the documents. We assume that each SCD shows a specific distribution of words near the SCD's location in the document.

The SCD matrix $\delta(\mathcal{D})$ models the distributions of words for all SCDs $g(\mathcal{D})$ of a corpus $\mathcal{D}$ and is structured as follows:

$$\delta(\mathcal{D}) = \begin{array}{c} \\ t_1 \\ t_2 \\ \vdots \\ t_M \end{array} \begin{array}{c} \begin{array}{ccccc} w_1 & w_2 & w_3 & \cdots & w_L \end{array} \\ \begin{pmatrix} \nu_{1,1} & \nu_{1,2} & \nu_{1,3} & \cdots & \nu_{1,L} \\ \nu_{2,1} & \nu_{2,2} & \nu_{2,3} & \cdots & \nu_{2,L} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \nu_{M,1} & \nu_{M,2} & \nu_{M,3} & \cdots & \nu_{M,L} \end{pmatrix} \end{array}$$

The SCD matrix consists of M rows, one for each SCD in $g(\mathcal{D})$, and each row contains the word probability distribution for the SCD. Therefore, the SCD matrix has L columns, one for each word in the vocabulary of the corresponding corpus.

The supervised training of an SCD matrix is described in Algorithm 1. Given a corpus $\mathcal{D}$, the algorithm iterates over each document in the corpus and the document's located SCDs. For each located SCD given by a tuple (t, ρ), the SCD matrix is updated. First, the sentence in d at position ρ is reconstructed. Next, the row of the matrix representing SCD t gets incremented for each word in the reconstructed sentence.

Algorithm 1 Training the SCD-word distribution matrix $\delta(\mathcal{D})$

```
1: function buildMatrix($\mathcal{D}$, $g(\mathcal{D})$)
2:     Input: Corpus $\mathcal{D}$, Set of SCDs $g(\mathcal{D})$
3:     Output: SCD-word distribution matrix $\delta(\mathcal{D})$
4:     Initialize an $M \times L$ matrix $\delta(\mathcal{D})$ with zeros
5:     for each $d \in \mathcal{D}$ do
6:         for each $(t, \rho) \in g(d)$ do
7:             for each $w^d \in win_{d,\rho}$ do
8:                 $\delta(\mathcal{D})[t][w^d]$ += $I(w^d, win_{d,\rho})$
9:     return $\delta(\mathcal{D})$
```

Kuhr et al. use a sliding window instead of our previously described sentence-based approach. The authors assume an SCD generates the words in a certain radius around the SCD's location, while we assume an SCD generates the words of the sentence at the SCD's location. The sentence-wise approach is required in this thesis due to the comparability to BERT working on whole sentences. Furthermore, a sliding window results in more computations and as we use larger corpora as Kuhr et al. sentence-wise iteration allows us to keep the computations sufficiently low.

After Algorithm 1 has finished, the SCD matrix needs to be normalized row-wise to meet the requirements of a probability distribution. However, we skip the normalization because multiple calculations on small decimal values on a computer reduce the accuracy. Later, we use the cosine similarity with the rows of the matrix and the cosine similarity does a normalization by definition. Thus, by skipping the normalization we save computational resources and get slightly more accurate results.

2.1. Most Probably Suited Subjective Content Descriptions

The previously described and trained SCD matrix can be used to estimate SCDs for a document without associated SCDs. First we formalize the MPS²CD problem and afterwards solve the problem by Algorithm 2 using the SCD matrix [KBBM19].

The MPS²CD problem asks for the M most probably suited SCDs $t_1, ..., t_M$ for a document d' given the SCD matrix $\delta(\mathcal{D})$:

$$g(d') = \operatorname*{arg\,max}_{t_1,...,t_M \in g(\mathcal{D})} P(t_1, ..., t_M \mid d', \delta(\mathcal{D}))$$

The definition of the MPS²CD problem does not consider the sentence-wise iteration used while training the SCD matrix. We can reformulate the MPS²CD problem to

Algorithm 2 Estimating MPS²CD sequence and similarity values

```
1: function ESTIMATEMPS²CD($d'$, $\delta(\mathcal{D})$)
2:     Input: Document $d'$, SCD matrix $\delta(\mathcal{D})$
3:     Output: SCDs $g(d')$ with similarity values $\mathcal{W}$
4:     $\mathcal{W} \leftarrow \emptyset$
5:     for each sentence $w_{i,j} \in d'$ do
6:         $win_{d',\rho} \leftarrow w_{i,j}$
7:         $\delta(win_{d',\rho}) \leftarrow$ new zero-vector of length $K$
8:         for each word $w \in win_{d',\rho}$ do
9:             $\delta(win_{d',\rho})[w] \mathrel{+}= I(w, win_{d',\rho})$
10:
11:        $t' \leftarrow \underset{t \in g(\mathcal{D})}{\arg\max} \frac{\delta(\mathcal{D})[t] \cdot \delta(win_{d',\rho})}{\|\delta(\mathcal{D})[t]\|_2 \cdot \|\delta(win_{d',\rho})\|_2}$
12:
13:        $sim \leftarrow \max_{t \in g(\mathcal{D})} \frac{\delta(\mathcal{D})[t] \cdot \delta(win_{d',\rho})}{\|\delta(\mathcal{D})[t]\|_2 \cdot \|\delta(win_{d',\rho})\|_2}$
14:
15:        $g(d') \leftarrow g(d') \cup \{(t', \rho)\}$
16:        $\mathcal{W} \leftarrow \mathcal{W} \cup \{(t', sim)\}$
17:    return $g(d')$, $\mathcal{W}$
```

consider the sentence-wise iteration:

$$g(d') = \bigcup_{\substack{\text{sentences} \\ w_{i,j} \in d'}} \underset{t \in g(\mathcal{D})}{\arg\max}\, P(t \mid w_{i,j}, \delta(\mathcal{D}))$$

Analogous to the second definition of the MPS²CD problem, Algorithm 2 iterates over each sentence of d'. For each sentence the algorithm creates a vector representing the words of the sentences $\delta(win_{d',\rho})$. The vector is created using the approach that was used for the rows of the matrix in Algorithm 1. We use the cosine similarity to compare $\delta(win_{d',\rho})$ with a row of the SCD matrix $\delta(\mathcal{D})[t]$ representing SCD t:

$$\frac{\delta(\mathcal{D})[t] \cdot \delta(win_{d',\rho})}{\|\delta(\mathcal{D})[t]\|_2 \cdot \|\delta(win_{d',\rho})\|_2}$$

The most probably suited SCD t is defined as the SCD belonging to the row resulting in the highest cosine similarity value.

The MPS²CD algorithm allows us to estimate the most probably suited SCDs for any sentence given the words of the sentence and the SCD matrix. Next, we use the MPS²CD algorithm to separate SCDs from *content* occurring interleaved in a document.

2.2. Inline Subjective Content Descriptions

In the last section we have described the MPS^{2}CD algorithm. The algorithm does not only return the sequence of MPS^{2}CDs, it also returns a sequence of similarity values. The MPS^{2}CD similarity values give further insights about the corpus $\mathcal{D}$ and the new document d'. We assume a similar context of d' and $\mathcal{D}$ if the values are overall high and a less similar context in the case of low values. This assumption is based on the fact that documents of similar contexts share similar words and thus, similar word distributions. When looking at the MPS^{2}CD similarity values for each sentence or window, it is also possible to detect similar and less similar parts of d'.

The MPS^{2}CD similarity values are used to classify documents as related, extended, revised, or unrelated in [KBBM20]. In this thesis, we focus on the iSCD problem and describe the solution provided by Bender et al. [BBG$^+$21b] using MPS^{2}CD similarity values, too.

In the scenario of the iSCD problem, documents are already annotated with SCDs, but the SCDs are not separated from the *content* of the documents. For each word of each document, the agents has then to decide whether the word is part of an SCD or belongs to the *content*. The iSCD problem asks to separate SCDs and *content* given interleaved in a document d'. Formalized, the iSCD problem's input is a document $d' = (w_1^{d'}, ..., w_N^{d'})$ and the output is the *content* d as sequence of words and a set of SCDs $g(d)$.

Example 2.1. Inline SCD Example
Assume a new document d' contains the following sentence with two SCDs interleaved. The underlined words represent the SCDs, while other the words form the content*.*

> "We visited the bisons large animals in the zoo a place where non-domestic animals are exhibited."

Document d' can be represented as a sequence of words. The words belonging to the SCDs are only underlined for readability.

$$d' = (w_1^{d'}, w_2^{d'}, w_3^{d'}, w_4^{d'}, \underline{w_5^{d'}, w_6^{d'}}, w_7^{d'}, w_8^{d'}, w_9^{d'}, \underline{w_{10}^{d'}, w_{11}^{d'}, w_{12}^{d'}, w_{13}^{d'}, w_{14}^{d'}, w_{15}^{d'}, w_{16}^{d'}})$$

After solving the iSCD problem, the result would be:

$$\begin{aligned} d &= (w_1^{d'}, w_2^{d'}, w_3^{d'}, w_4^{d'}, w_7^{d'}, w_8^{d'}, w_9^{d'}) \\ g(d) &= \{ \quad \{(w_5^{d'}, w_6^{d'}), 4\}, \quad \{(w_{10}^{d'}, w_{11}^{d'}, w_{12}^{d'}, w_{13}^{d'}, w_{14}^{d'}, w_{15}^{d'}, w_{16}^{d'}), 9\} \quad \} \end{aligned}$$

Bender et al. propose the following three approaches solving the iSCD problem: (i) the word-based approach, (ii) the threshold-based approach, and (iii) the Hidden Markov Model (HMM)-based approach.

The word-based approach does not use an SCD matrix, it only uses the frequencies of the words in the set of SCDs and the *content*.

The threshold-based approach works as follows:

(i) Choose a threshold $th \in (0, 1)$.

(ii) Train an SCD matrix.

(iii) Run the MPS²CD algorithm for the new document d' and collect the similarity values for each word $w^{d'}$.

(iv) Classify each word $w^{d'}$ as SCD or *content* based on its similarity value *sim*.

$$class(sim) = \begin{cases} \text{"SCD"} & sim < th, \\ \text{"}content\text{"} & sim \geq th \end{cases}$$

Intuitively, a subsequence containing an SCD yields lower MPS²CD similarity values than a subsequence containing *content*. The similarities are lower because the SCD matrix is trained only on *content* and not on SCDs, for which a different vocabulary is assumed. For a vector representing *content*, Algorithm 2 may estimate more similar rows in the SCD matrix than for a vector representing an SCD.

The first three steps of the HMM-based approach are identical to the threshold-based approach, but for the classification the authors train an HMM with two states on the discretized sequence of similarity values. One state of the HMM represents the SCDs and the other state represents the *content*. The classification task of a new document then uses the Viterbi algorithm [Vit67]. The Viterbi algorithm returns the most likely sequence of states in an HMM given a sequence of observation symbols. The authors use the Viterbi algorithm to calculate the HMM's state for each word and use the discretized similarity values of the new document as observation symbols.

We decide to use the threshold-based approach for our evaluation. Even though the performance of the HMM-based approach is slightly better, choosing a good threshold *th* results in a nearly equal performance and the threshold-based approach simplifies the implementation significantly.

In this thesis, we again deviate from Bender et al. as we decide for each sentence and not for each word of a document, whether the sentence belongs to an SCD or to the *content*.

3. Transformer Language Models

In the previous chapter we have introduced SCDs. In this chapter, we describe Transformer Language Models (TLMs) [VSP$^+$17] and focus on the Bidirectional Encoder Representations from Transformers (BERT) [DCLT19]. We use BERT to solve the MPS^{2}CD and iSCD problem introduced in Section 2.1 and Section 2.2.

3.1. Neural Networks

Neural networks are a commonly used technique in the area of machine learning to classify an input value by calculating the network's output value. The structure of a neural network consists of multiple layers, where each layer consists of multiple units. An input value is fed into the units of the first layer. Classically only booleans act as input values, but today also numeric values or vectors are used. Each unit then computes an output value and forwards the value to one or multiple units of the next layer. The values are passed along the layers until the last layer is reached. The outputs of the units of the last layer than form the output of the entire network.

The network is described by its architecture, the number of layers, units and connections between the units, as well as the units. A classical unit is the perceptron, it consists of weights $\alpha_1, ..., \alpha_n$ and an activation function φ. A perceptron getting the inputs $x_1, ..., x_n$ and calculating the output y may be described by the formula:

$$y = \varphi\left(\sum_{i=1}^{n} \alpha_i \cdot x_i\right)$$

More complex units are, e.g., Long Short-Term Memory (LSTM) [HS97] units used to process sequential data. LSTMs can store information in their units.

Usually the architecture of a network and the units, including their functions, have to be defined by the user before the network can be trained. During the training the weights and further parameters of each unit are estimated based on labeled data, i.e., pairs of input and expected output. The training pursues the objective to minimize the error of the network, whereas the error is often defined as difference of the output gained from the network and the expected output.

The architecture of neural networks can be divided in many types. Feed-Forward networks [RN10] allow only connections between units in one direction, so the values are passed from layer to layer until they reach the last layer. In contrast, Recurrent Neural Networks (RNNs) [RHW86] allow feedback loops. Feedback loops pass the output of a unit back into the same unit or another unit on the same layer or on a previous layer. RNNs allow the processing of sequential data, as the feedback loops allow to store previous values.

A commonly used function to normalize vectors gained by neural networks is the softmax, defined by:

$$\text{softmax}(x_1, ..., x_n)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

The softmax does not only yield a vector whose sum is 1, but also takes into account the size of the distance of each numerical value, e.g., $\text{softmax}(1, 2) = (0.27, 0.73)$ while $\text{softmax}(5, 10) = (0.006, 0.994)$.

3.2. Transformer Architecture

Vaswani et al. [VSP$^+$17] have introduced Transformer Language Models (TLMs). Over the last decades, commonly used techniques in the area of NLP were RNNs and neural network with LSTM units. For both techniques, the inputs are fed into the model sequentially one after the other. Inside the network, the previous inputs are then stored by the feedback loops of the RNN or in the LSTM units. Furthermore, RNNs suffer from a *vanishing gradient*, i.e., when adding a new input the influence of each previous input becomes a bit smaller and vanishes for long input sequences.

In contrast, TLMs have the major advantage that all inputs are fed simultaneously into the model. This advantage does not only solve the problem of a *vanishing gradient*, it also allows to train TLMs in parallel. Thus, the introduction of TLMs initiated the development of a large number of new models in the area of NLP.

In the following, we first describe the attention mechanism used by TLMs and the architecture of a Transformer, the key part of TLMs. Afterwards, we introduce BERT including the training of BERT and BERT's use-cases.

3.2.1. Attention

The main mechanism behind TLMs is the so-called attention. The idea behind attention is to put attention on the relevant parts of input data letting less relevant parts fade out. For TLMs in particular the scaled dot-product attention is used.

Given a triple of a query vector q, a key vector k, and a value vector v, the scaled dot-product attention yields an output vector. For faster simultaneous computations multiple vectors q, k, and v are combined in matrices Q, K, and V. The formula of the scaled dot-product attention, combining the output vectors in a matrix, too, is defined by:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{dimension_k}}\right) V$$

Given the output vectors of two different triples containing different q, k, and v, the similarity between both input triples can be estimated using the dot-product, again. A high value of the dot-product between the output vectors induces a relation of the two input triples and a low value induces no relation.

The scaled dot-product attention needs an input of three vectors, however documents consist of sequences of words. Thus, the words have to be encoded as (input) vectors. A common approach encoding words as vectors are embeddings, and depending on the TLM, different embedding techniques are used. For now, we only assume an embedding technique as a function transforming a word w into a vector e_w representing w.

Before we can apply the scaled dot-product attention on an input e_w, we have to derive three input vectors q, k, and v from e_w. The derivation of q, k, and v only uses e_w as input and the resulting scaled dot-product attention is therefore also called self-attention. The vectors q, k, and v all have slightly different duties in the scaled dot-product attention and the TLMs train three matrices W_q, W_k, and W_v for deriving q, k, and v from e_w:

$$q = W_q\, e_w, \quad k = W_k\, e_w, \quad v = W_v\, e_w$$

The scaled dot-product attention allows to identify pairs of related words in a sentence by calculating the scaled dot-product attention of each word and comparing the outputs vectors. However, one word in a sentence often has more than one relation to multiple other words, e.g., a predicate has a relation to the subject and a different relation the object. For a single scaled dot-product attention it would be difficult to represent both relations simultaneously.

Using multiple scaled dot-product attentions the problem representing different relations to other words can be solved. A multi-head attention uses h scaled dot-product attentions and provides a scaled dot-product attention for each type of relation. The entire setup of a multi-head attention in combination with scaled dot-product attentions is illustrated in Figure 3.1.

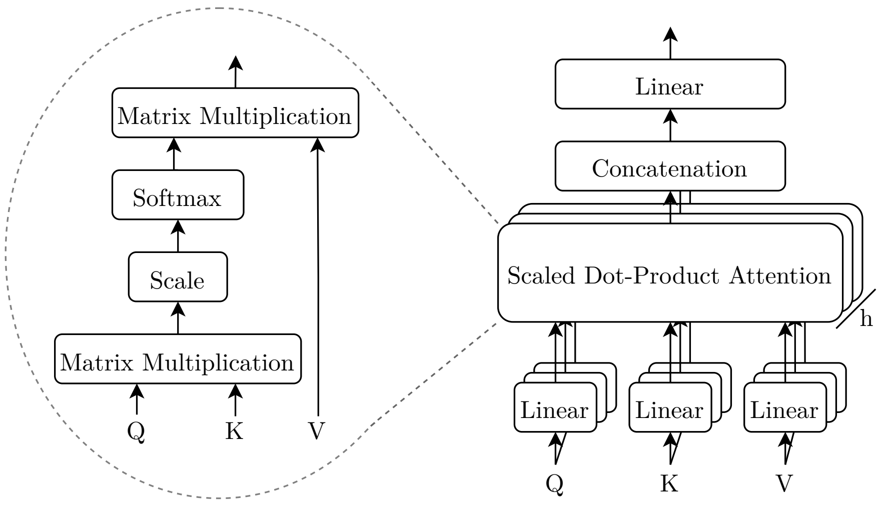

Figure 3.1.: The multi-head attention (right) consists of multiple layers and features multiple scaled dot-product attentions (left) at their center. Visualization inspired by Figure 2 in [VSP$^+$17].

Technically, multi-head attentions partition an input vector e_w into h chunks. Each chunk uses a dedicated scaled dot-product attention, with its own matrices W_q, W_k, and W_v. In the end, the h resulting output vectors are concatenated again to a single vector.

3.2.2. Transformer

The key part of an TLM is the Transformer and the Transformer is divided into an encoder and a decoder. The encoder takes a text document as input, calculates the embedding first and afterwards applies the multi-head attention multiple times to create an encoded representation of the input. The decoder then takes the encoded representation and is able to decode it back into a textual representation.

TLMs are typically used to translate text documents from one language to another language. However, the concept of TLMs can be used for many other tasks.

In this thesis, we use the encoded representations generated by BERT. Therefore, we focus on the encoder of TLMs and especially BERT. Figure 3.2 represents the detailed structure of a full encoder unit. First the embedding of the input is calculated. Afterwards, a positional embedding is added. The multi-head attention does not recognize the order of words in an input sequence, therefore the order is fed into the model by adding vectors encoding the position of the words.

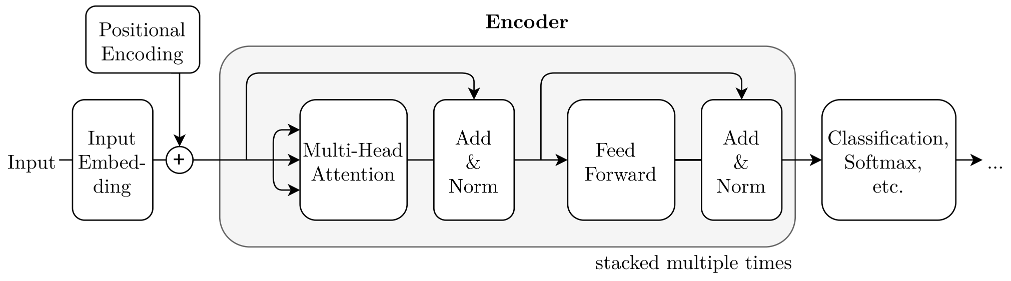

Figure 3.2.: The encoder used by TLMs. The actual encoding takes place in the layer highlighted gray. Visualization inspired by Figure 1 in [VSP+17].

Next, the part highlighted gray represents the main part of the encoder. This main part may be stacked multiple times to increase the performance. The main part consists of a multi-head attention, one may see the derivation of query, key, and value. Afterwards, the output is summed to the input and normalized before it is passed into a fully connected feed-forward network.

It is important to note, that TLMs encode each input vector separately. Encoding a document of N words results in N vectors, each representing a word. Later, the bidirectional encoding of BERT will soften the separate encodings.

Well-known TLMs are BERT [DCLT19] and Generative Pre-trained Transformer (GPT) [RN18, RWC+19, BMR+20]. ELMo [PABP17, JPH18] is another well-known encoder, but it uses LSTMs instead of Transformers. We focus on BERT in this thesis, however we will also have a short comparison of BERT, ELMo, and GPT.

3.3. Bidirectional Encoder Representations from Transformers

A well-known TLM is the Bidirectional Encoder Representations from Transformers (BERT). BERT was introduced by Devlin et al. [DCLT19]. As stated before, BERT is only an encoder and encodes a sequence of inputs, i.e. words, into a sequence of vector representations. In addition to the encoded vector for each word, BERT features a *class* output representing the entire input sequence in one vector.

BERT uses a special architecture of the encoders from TLMs, described in the previous section. The architecture of BERT is shown in Figure 3.3. In Figure 3.3, $E_1, ..., E_N$ represent the sequence of input vectors, already transformed to the embeddings, and $T_1, ..., T_N$ represent the sequence of outputs, the encoded vectors. Between input and output, the layers and connections of the network are depicted and each encoder unit from TLMs is marked as TR.

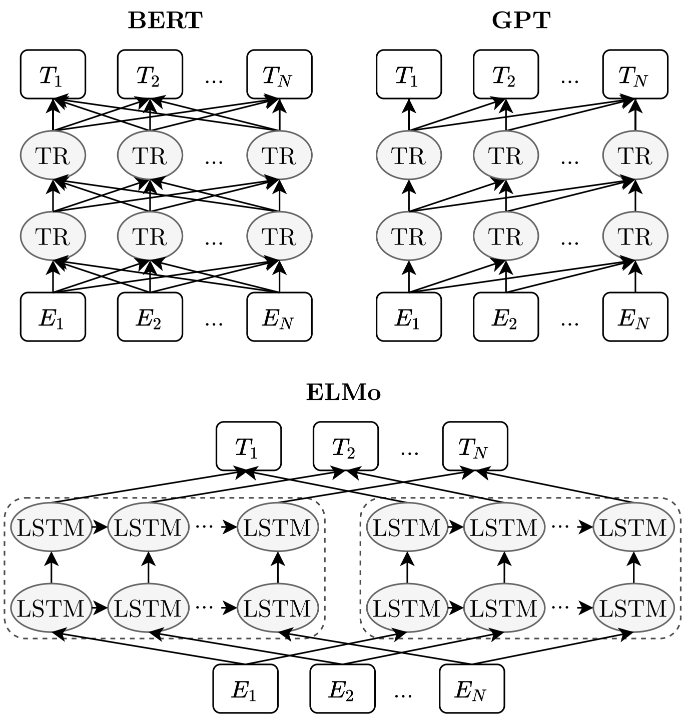

Figure 3.3.: Comparison of the architecture for the three well-known models BERT, GPT, and ELMo. Each encoder unit TR in each layer of BERT is bidirectionally connected to all units in the next layer. In GPT each TR is only connected to the units representing the subsequent elements in the input sequence. In contrast to BERT and GPT, ELMo uses a different approach, it independently trains LSTMs left-to-right and right-to-left per input and concatenates both outputs. Visualization inspired by Figure 3 in [DCLT19].

As the full name of BERT states, BERT uses a bidirectional encoding. The bidirectionality of BERT is based on the fact that each TR is connected to each TR of the next layer. By the bidirectional connections, each input of BERT does not only influence its own output, but may also influence all other outputs of the entire sequence. Therefore, BERT may encode entire sentences as well as the context of words and phrases.

BERT is available in different sizes. The default size consists of 12 layers and the maximum length for the input sequence, which is also the number of TR units per layer, is 512. Furthermore, the used vectors consist of 768 elements each and each multi-head attention combines $h = 12$ dot-product attentions. In total, BERT features 110 million parameters.

In comparison to BERT, GPT does not use bidirectional connections between each layer. The architecture of GPT features only connections of each TR to the sub-

Text		Bisons are large animals.							They run fast.						
Tokenized Text	[CLS]	bison	##s	are	large	animals	.	[SEP]	they	run	fast	.	[SEP]		
Token IDs	101	22285	2015	2024	2312	4176	1012	102	2027	2448	3435	1012	102	0	...
Segment IDs	0	0	0	0	0	0	0	0	1	1	1	1	1	0	...
Position	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	...
Sequence Mask	1	1	1	1	1	1	1	1	1	1	1	1	1	0	...

Figure 3.4.: Generation of the input vectors of BERT and the different parts combined in each vector. The token IDs, segment IDs, and positions are each represented as a vector and added up for each token to a vector forming the input of BERT. The sequence mask activates the correct number of inputs from the 512 inputs BERT has.

sequent TRs of the next layer (see Figure 3.3). Therefore, in GPT the first element in the sequence of inputs can influence all outputs, while the last element can only influence the last output. ELMo does not use TR units and instead uses LSTM units, but ELMo also models bidirectional influences. ELMo trains one set of LSTMs left-to-right and another set right-to-left, and in the end, the resulting vectors are concatenated.

In the previous paragraphs we always assume to get a sequence of vectors as input for BERT. These vectors are an embedded representation of a sequence of words, e.g., one or multiple sentences. BERT uses the WordPiece embedding technique [WSC+16] with a vocabulary of 30 000 words. The embedding is trained along with BERT.

An example of BERT's input embedding with two sentences is shown in Figure 3.4. The figure shows the different parts combined in each input vector. First, two special tokens `[CLS]` and `[SEP]` are added. The token `[CLS]` is always the first token for any input sequence, it marks the *class* output. Therefore, the vector at the first position of an output from BERT always represents the entire input sequence and can be used to classify the sequence. The token `[SEP]` marks the end of the input sequence and also separates multiple sentences. Second, both sentences are tokenized into their words and punctuation. The word `bisons` creates two tokens `bison` and `##s`, together both tokens represent the plural of `bison`. The representation as `bison` and `##s` allows to represent words that are not part of the vocabulary, i.e., `bisons` is not contained in the vocabulary, but `bison` is. Third, all tokens are transformed into their numeric representation, which can be converted to a vector later.

Before we can pass the vectors to BERT a positional encoding, which was already used before in TLMs, is added. BERT may get two sentences simultaneously, the segment IDs mark for each token to which sentence the token belongs. For each token, the added up token ID, segment ID, and position form a vector of the input sequence.

The maximum length for the input sequence of BERT is 512, however, not all input sequences may contain 512 vectors. The sequence mask makes sure to use only the available vectors.

We have now described the basic architecture of BERT, next we describe how BERT is trained and how BERT can be used.

3.3.1. Pre-Training and Fine-Tuning

Training BERT requires two stages, the pre-training and fine-tuning. The pre-training is done on a huge corpus and pursues the objective to understand natural language. A pre-trained model provides a general language understanding and context dependent encoding of input tokens. Normally, such pre-trained models do not facilitate a special task and are available for public download. Being available for download and usable for many different tasks is required, since the pre-training takes a long time and requires large computational resources.

Afterwards, the fine-tuning will be done on a previously pre-trained model and will focus the model on a specific task. During the fine-tuning the model is trained by labeled inputs and “learns” how to solve its special task. As the model already provides an understanding of natural language, the fine-tuning may be done on less data and runs much faster.

One may understand the pre-training as learning the basics of a language and human communication with no focus on special use-cases. Whereas fine-tuning means focussing on a use-case and gaining specific “knowledge” about the domain and given tasks.

The pre-training of BERT is done on two objectives. Both objectives use only corpora of text documents and do not need further labels or annotations for the documents.

Masked Word Prediction A single sentence is used as input. In the sentence randomly one word is replaced by the special token `[MASK]` 80% of the time. In each 10% of the time, the randomly chosen word is left unchanged or the randomly chosen word is replaced by a different randomly selected word. The task of the model is to predict the correct word at the position of the randomly chosen word. This prediction is realized by a linear classification of the vector returned by BERT at the position of the randomly chosen word.

The masked word prediction forces BERT to learn a representation of each word based on the word's surroundings and context. The token `[MASK]` is only used 80% of the time to make sure every word gets known to BERT and to make it not always possible for BERT to know which word needs to be predicted.

Next Sentence Prediction A pair of two sentences is used as input. Either both sentences occur consecutively in a document or both sentences are randomly put together. The task of the model is to classify if both sentences occur consecutively or not.

The next sentence prediction forces BERT to learn the relation and context between two sentences occurring consecutively.

It is possible to train the model for both objectives simultaneously by using two sentences and additionally mask words in the sentences.

Next, we present the different use-cases for BERT. A pre-trained model is fine-tuned specifically for one of the use-cases.

3.3.2. Use-Cases for BERT

BERT can be used in a variety of different use-cases. For every use-case the same pre-trained model can be used and fine-tuned.

Figure 3.5 shows the pre-training and the typical use-cases for BERT. We now imagine BERT as a black box model getting at maximum 512 tokens as input and therefore generating 512 output vectors. Of course, the special tokens `[CLS]` and `[SEP]` are always added to the inputs. The black box model is depicted as a box getting the inputs at the bottom and yielding the outputs at the top.

On the left side of Figure 3.5 the objectives during the pre-training are shown. In case of the masked word prediction, a linear classifier predicts the masked word based on the output at the position of the token `[MASK]`. For the next sentence prediction, two sentences separated by `[SEP]` are fed into the model and the *class* output is used to decide if both sentences occur consecutively. In the literature, the different classifiers placed on top of BERT are called *heads*.

As the pre-trained BERT does not provide a wide range of functionality—mostly it only creates encoded representations—the following *heads* use the encoded representations. However, the encoded representations contain the required data for the use-cases. Thus, the use-cases can be realized using simple methods like linear classifiers, dot-products and the softmax function. The linear classifiers are learned from scratch during fine-tuning.

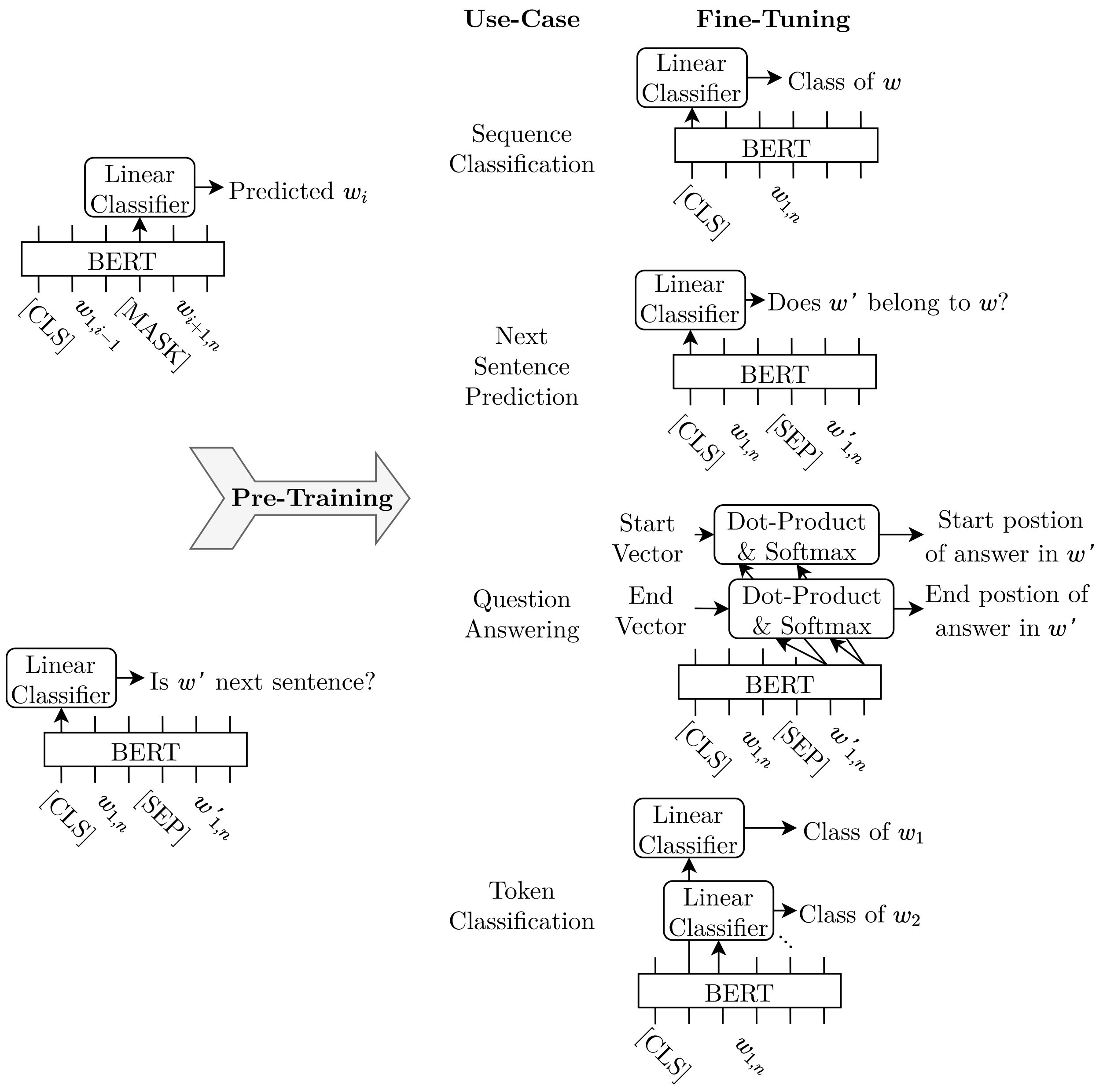

Figure 3.5.: The different *heads* put on top of BERT during pre-training and fine-tuning for multiple use-cases. BERT is depicted as a box getting the inputs at the bottom and yielding the outputs at the top. The different *heads* are then placed on top of BERT's box yielding the actual results for each use-case.

Sequence Classification A single sentence is used as input. The task of the model is to assign a class to the sentence. The classes assigned are taken from the class labels in the training data used for fine-tuning.

The classification uses a linear classifier at BERT's *class* output.

Next Sentence Prediction A pair of two sentences is used as input. The task of the model is similar to the next sentence prediction objective already used for pre-training. However, here the model may learn any type of relation between both sentences given as input. The relation between both sentences is given by boolean labels in the training data used for fine-tuning.

Again, a linear classifier at BERT's *class* output is used.

Multiple Choice A single sentence, the query, together with a set of sentences, the possible answers, is used as input. The task of the model is to select the best answer from the set of answers for the query.

Multiple choice is the only use-case not shown in Figure 3.5 because it uses next sentence prediction multiple times. For each possible answer to the query, a pair of query and answer is passed to the next sentence prediction. Then for each pair, the next sentence prediction returns a score and the answer in the pair reaching the best score is returned as solution for the multiple choice use-case.

Question Answering A single sentence, the question, and a short document containing the answer are used as input. The task of the model is to select the answer to the question in the short document. The selection is done by returning a start and end position and thus an interval. Then, the answer generated by the model is formed by the words of the short document contained in the interval.

During the fine-tuning two vectors are trained. The *start* vector is used to calculate the start position and the *end* vector to calculate the end position, respectively. The data used during fine-tuning contains start and end positions as labels. To calculate the start position, each output vector resulting from a token of the short document is dot-multiplied by the *start* vector. The softmax across all dot-products gives the start position. The end position is calculated the same way using the *end* vector.

Token Classification A single sentence is used as input. The task of the model is to assign a class to each token of the input. The classes are taken from the class labels in the training data used for fine-tuning.

The classification uses a linear classifier at each of BERT's outputs.

We have now presented SCDs, the MPS^{2}CD algorithm, and BERT. In the next chapter, we show how to solve the iSCD and MPS^{2}CD problem with BERT.

4. Transformer Language Models for Subjective Content Descriptions

Applying the theoretical foundations of SCDs and TLMs introduced in the previous chapters, the contribution of this thesis is to combine the framework of SCDs with TLMs and use BERT to solve problems Kuhr et al. have introduced along with SCDs.

In this chapter, we describe how to apply the use-cases of BERT introduced in Subsection 3.3.2 to SCDs. All use-cases have in common that they are based on a pre-trained model of BERT which is fine-tuned on the particular use-case and problem to solve. Hence, we apply BERT to solve the MPS^{2}CD and iSCD problem. For both problems, we outline the suitable use-cases and describe how to arrange the documents and SCDs as input sequences for BERT.

BERT gets a sequence of words as input. We decide to always use the words of one sentence as one input sequence. This sentence-wise approach differs from Kuhr et al. as they use a sliding window over the words of the documents in a corpus. Using a sliding window results in more input sequences to consider and less variation among the sequences. Less variation may lead to overfitting during the fine-tuning of BERT. In this thesis, we use larger corpora than Kuhr et al. because the fine-tuning of BERT needs more samples than required to train the SCD matrix. We use the sentence-wise approach to prevent overfitting and to keep the number of input sequences moderate.

We argue that the sentence-wise approach does not result in a bad partitioning as in natural language a sentence is a logical unit. Thus, the most influential words of a word belong to the same sentence the word itself belongs to. The sentence-wise approach maintains the logical structure of the documents.

Theoretically, it would be possible to apply the use-cases described below using sentences in the same way to sequences obtained from a sliding window.

4.1. Identifying Subjective Content Descriptions

The iSCD problem asks to separate SCDs and *content* given interleaved as sequence of words in a document. Applying the sentence-wise approach, the input for the iSCD problem is a sequence of sentences to be distinguished into the sentences being SCDs and the sentences being *content*. Thus, the iSCD problem is a classification problem with two classes, namely SCD and *content*.

The use-cases of BERT solving classification problems are sequence classification, next sentence prediction, and token classification (see Figure 3.5 on page 30 for a visualization). In the following paragraphs we consider each of the three use-cases.

4.1.1. Sequence Classification

Using sequence classification to solve the iSCD problem by BERT is straightforward. For each sentence as input sequence the encoded representation is calculated. From the encoded representation, only the vector at the *class* output is needed to classify the sentence as SCD or *content*. We train a linear classifier for the vectors at the *class* output.

We fine-tune the model with $\mathcal{D}$ and measure the model's performance on a different $\mathcal{D}'$. Especially, the sets of SCDs are disjoint, i.e., $g(\mathcal{D}) \cap g(\mathcal{D}') = \emptyset$. The disjoint sets of SCDs are important to prevent the model from simply memorizing all SCDs. To prevent the model from remembering undesired relations, we also randomly shuffle the sentences such that no pattern of occurrence between SCD and *content* exists.

A disadvantage when using sequence classification is that the relation between sentences and their associated SCDs is not modeled. Each sentence, whether SCD or *content*, gets classified independently.

4.1.2. Next Sentence Prediction

In the last paragraph, we noticed that sequence classification does not model the relations between sentences and their SCDs. Though, next sentence prediction models the relation between two sentences and classifies if two sentences are in a relation. We specify to model the relation between a sentence from the *content* and its associated SCD. The model is fine-tuned on tuples of two sentences, meaning the first sentence is always a sentence from the *content* and the second sentence may be a related SCD or the subsequent sentence from the *content*. Thus, the model classifies the second sentence as (related) SCD or no SCD.

As with sequence classification, we use different corpora and disjoint sets of SCDs, as well as shuffle the tuples of sentences.

4.1.3. Token Classification

In combination with the sentence-wise approach, we are only interested in the results for the entire input sequence. Though, token classification calculates classes for each token. Thus, we do not use token classification in this thesis.

However, it would be possible fine-tuning BERT to classify each word as SCD or *content*. When the MPS^{2}CD algorithm is used with a sliding window over the words of a document, it also yields similarity values for each word, which can then be converted to classes by a threshold. In this way, our sentences-based approach can be used in combination with a sliding window.

The token classification is the only use-case that allows using such a mixture of the sentence-based approach for BERT and a sliding window for the MPS^{2}CD algorithm. Hence, we do not investigate the idea further.

4.2. Estimating Most Probably Suited Subjective Content Descriptions

The MPS^{2}CD problem asks for the most probably suited SCDs associated with a given non-annotated sentence. Solving the MPS^{2}CD problem allows us to estimate SCDs from a set of known SCDs for each sentence in a document. Thus, documents featuring a similar context as the set of known SCDs can be annotated.

Multiple choice and question answering are the remaining use-cases of BERT (see Figure 3.5 on page 30 for a visualization) and both can select best matches from a set of possibilities. In the following paragraphs we apply each of the two use-cases.

4.2.1. Multiple Choice

Using the multiple choice use-case to solve the MPS^{2}CD problem by BERT is straightforward. For each sentence a set of four SCDs is given. We offer four SCDs to BERT because the duration running BERT four times is acceptable and well-known tasks like SWAG [ZBSC18] also provide four options to choose from. In the set of four SCDs, the solution always has to be unique, i.e., one SCD may be associated with the sentence while the other three must not be associated. Internally, the model does a next sentence prediction four times and returns the SCD with the highest probability.

We fine-tune and measure the performance of the model again on different corpora and disjoint sets of SCDs. However, we do not shuffle the ordering of the sentences in the documents and only randomly choose at which of the four positions the correct SCD is presented to the model.

4.2.2. Question Answering

Analogous to multiple choice, the question answering use-case gets a sentence and four SCDs to select one SCD from. The four SCDs are randomly shuffled and concatenated to a short document. The sentence and the short document are then fed into BERT. BERT returns an interval and selects the words in this interval of the short document as SCD for the sentence.

The input sequence must not exceed BERT's size of 512 tokens, concatenating four SCDs might result in too long documents. If a document gets too long, we first try to omit one or two SCDs and ignore the entire sample in the end.

We fine-tune and measure the performance of the model again on different corpora. Once more, we use disjoint sets of SCDs as well as, in addition, the same set of possible SCDs for both corpora.

4.2.3. Notable Difference

The MPS^{2}CD algorithm provided by Kuhr et al. returns an SCD known by the model and does not select a best SCD from a given set. To validate the solution of the MPS^{2}CD algorithm we have to compare two SCDs, the SCD labeled as correct in the given set and the SCD returned by the MPS^{2}CD algorithm. Especially, when the sets of SCDs used for training and testing are disjoint, the two SCDs will never be equal but might be both correct.

In our evaluation, the SCDs are gained from an agent. This agent allows inverse queries, i.e., the agent can tell if it would annotate a sentence with both annotations. If the agent would do so, we assume the MPS^{2}CD algorithm selected the correct SCD.

Another possibility is to compare the SCD returned by the MPS^{2}CD algorithm to all four SCDs given. We use the embedding technique Doc2Vec [LM14] to encode the SCDs as vectors. Afterwards, the cosine similarity is calculated between all four SCDs and the SCD returned by the MPS^{2}CD algorithm in their Doc2Vec representation. Finally, the SCD resulting in the highest similarity to the SCD returned by the MPS^{2}CD algorithm is selected from all four SCDs.

4.3. Context-Specific Subjective Content Descriptions

The title of this thesis states to estimate context-specific SCDs. The estimation is primarily situated in the MPS^{2}CD approach, but we did not mention the specific context until now.

In the previous parts, we have always assumed context-specific SCDs because the collection of documents in a corpus always represents a specific context for us. The corpora and the SCDs represent a context and the model silently learns that context. However, in a more realistic scenario, a model also has to disambiguate between multiple contexts. For example, a sentence in a scientific paper should be annotated with different SCDs than a sentence in a children's book.

Determining a context and estimating MPS^{2}CDs can be realized with two distinct models. One model selects the contextually most similar corpus from a set of known corpora, and depending on the selected corpus, a different model trained on the selected corpus then estimates the MPS^{2}CDs. However, we propose a context-sensitive model combining both steps. We use the same approaches and problems introduced in Section 4.1, 4.2, and Chapter 2 and only change the corpora and their SCDs.

Given are two corpora $\mathcal{D}_{c_1}, \mathcal{D}_{c_2}$ representing two different contexts c_1, c_2 with their context dependent SCDs $g(\mathcal{D}_{c_1}), g(\mathcal{D}_{c_2})$. We create a combined corpus $\mathcal{D}$ and its SCDs $g(\mathcal{D})$ to train the context-sensitive model on:

$$\mathcal{D} = \mathcal{D}_{c_1} \cup \mathcal{D}_{c_2}, \quad g(\mathcal{D}) = g(\mathcal{D}_{c_1}) \cup g(\mathcal{D}_{c_2})$$

$\mathcal{D}$ and $g(\mathcal{D})$ are formed by the union while taking care to update the positions of the located SCDs.

The MPS^{2}CDs are estimated based on the words in a sentence. In the combined corpus the model now has to represent the originating corpus and the MPS^{2}CD together.

For the context-sensitive model we assume the performance stays similar if the solution does not model the relation between sentence and associated SCD, e.g., solving the iSCD problem using the SCD matrix or using BERT's sequence classification use-case. Further we assume, the performance slightly decreases if the solution models the relation between sentence and associated SCD, e.g., solving the MPS^{2}CD problem using the SCD matrix or using BERT's multiple choice use-case.

5. Evaluation

After we have introduced two approaches each solving the iSCD and MPS^{2}CD problem with BERT, we present an evaluation of the previously introduced approaches. We compare the performance of the together four approaches using BERT to the performance of the two approaches using an SCD matrix. Especially, we demonstrate that both, BERT and the SCD matrix, are capable techniques to model the relations of SCDs and sentences.

5.1. Datasets

In this evaluation we use the 20 newsgroups[1] dataset. 20 newsgroups is a well-known corpus consisting of e-mails from 20 e-mail newsgroups. Thematically, the 20 newsgroups can be divided into six topics, *computer*, *sport*, *science*, *politics*, *religion* and *for sale*. The entire corpus consists of 18 828 text documents. The documents have between 1 and 39 682 words with a median of 160 words.

For training the SCD matrix and fine-tuning BERT on the 20 newsgroups dataset, we need not only the documents but also SCDs associated with each sentence in the documents. However, documents in the 20 newsgroups dataset are not associated with SCDs. Therefore, we use definitions from the online dictionary Wiktionary[2] and annotate each sentence in the documents of the 20 newsgroups dataset with a definition from Wiktionary acting as SCD. The set of definitions from Wiktionary contains in total 293 296 definitions for 201 688 different words and phrases.

The Wiktionary annotation agent allows us to automate the annotation of documents and generates $g(\mathcal{D})$ for any corpus $\mathcal{D}$. Thereby, a number of SCDs generated for each sentence can be freely chosen. For the same sentence, the agent always returns the same SCD, while trying to maximize the variance of SCDs for similar but different sentences.

The Wiktionary annotation agent also allows to generate non-matching SCDs for a sentence and thus gives negative samples. Furthermore, it is possible to query the agent inversely and check if an SCD describes a sentence.

[1]http://qwone.com/~jason/20Newsgroups/
[2]https://en.wiktionary.org/

To evaluate the context-sensitive model proposed in Section 4.3, a second annotation agent and a second dataset is needed. The second annotation agent works similar to the Wiktionary annotation agent, but uses annotations from the 500 000 quotes [GMG18] dataset[3]. As second dataset we use Manuscript cultures[4], an openly accessible journal publishing exhibition catalogues and articles from the field of written artefacts. The two datasets provide different contexts, namely *written artefacts* and *computer and science*.

5.2. Implementation

The entire evaluation is implemented in Python. Python is a suitable choice because it provides a large number of libraries for machine learning tasks and neural networks. We use the popular and well documented Huggingface Transformers[5] implementation of BERT. The implementation of Algorithm 1 and Algorithm 2 is based on Gensim[6], NumPy[7] and SciPy[8]. Additionally, for the preparation of the corpora the Natural Language Toolkit (NLTK)[9] is used.

The entire implementation of our evaluation is designed object oriented and with maintainability in mind. For future work, it is easily possible to add more corpora or evaluate on other TLMs than BERT. The implementation of BERT we use is backed by PyTorch and therefore requires a graphics card to run fast. However, the implementation of Algorithm 1 and Algorithm 2 benefits from running on multiple processor cores. To be able to easily deploy the implementation on multiple platforms, the entire implementation is bundled as a Python package running in a Docker container. For further details about running the evaluation, we refer to the source code and its documentation included on the CD of this thesis[10].

5.3. Workflow

In the evaluation workflow, the experiments run on two different platforms. All experiments using the SCD matrix run in a Docker container on a machine featuring 8 Intel 6248 cores at 2.50GHz (up to 3.90GHz) and 16GB RAM. However, the virtual

[3]`https://github.com/ShivaliGoel/Quotes-500K`
[4]`https://www.csmc.uni-hamburg.de/publications/mc.html`
[5]`https://huggingface.co/transformers/`
[6]`https://radimrehurek.com/gensim/`
[7]`https://numpy.org/`
[8]`https://www.scipy.org/`
[9]`https://www.nltk.org/`
[10]Also available online at `https://www.ifis.uni-luebeck.de/~bender/ma/docs/`

machine does not provide a graphics card for the implementation of BERT. Thus, all experiments using BERT run on a single NVIDIA A100 40GB graphics card of an NVIDIA DGX A100 320GB. Beneath, the NVIDIA Container Toolkit is used to run our Docker container with NVIDIA CUDA support.

We run all experiments five times and take the means of the resulting scores to increase the statistical correctness. Each experiment follows a similar procedure:

(i) Download the corpus and the set of SCDs for the annotation agent.

(ii) Lowercase all characters, stem the words, tokenize the sentences and eliminate stop words from a wordlist containing 179 words. These four tasks are called preprocessing tasks. We perform them on the corpus and the set of SCDs for the annotation agent. Preprocessing a text of a document transforms the text in a more digestible form for machine learning algorithms and increases their performance [VIN15].

The four preprocessing tasks are combined with the tokenizing mechanism for BERT described in Figure 3.4. Thus, all preprocessed documents are preprocessed for BERT again. Theoretically, it would be enough to preprocess the documents only once as described in Figure 3.4, but as we compare the performance of multiple approaches, we use a common preprocessing across all experiments. In Section A.1 of the appendix we provide the results, when skipping the four preprocessing tasks.

(iii) Split the corpus randomly into a training set containing 80% of the documents and a test set containing the remaining 20%. If a disjoint set of SCDs is used in the current experiment, the set of SCDs for the annotation agent is also split into 80% and 20% of the definitions.

(iv) Generate the SCDs for the training set and test set by the annotation agent. If the iSCD problem is evaluated in the current experiment, also generate documents containing randomly interleaved SCDs and *content*.

(v) On the training set train the SCD matrix or fine-tune BERT, depending on the current experiment.

We use the pre-trained `bert-base-uncased`[11] version of BERT. This version of BERT is case insensitive and a standard model to fine-tune for downstream tasks.

(vi) Evaluate the performance of the trained model on the test set. We always calculate the accuracy of the model and other measures depending on the evaluated problem.

[11] `https://huggingface.co/bert-base-uncased`

5.4. Performance Measures

We measure the performance of each experiment with different measures. The measure depends on the problem and model. Generally, for each sample in the test set a prediction is generated using the trained model and the predicted value is compared against the sample's label. In the following, the frequency of an outcome o is denoted by $\#o$.

For all problems and models, given the number of samples predicted correct and wrong, the accuracy is defined by

$$accuracy = \frac{\#correct}{\#correct + \#wrong}.$$

Further, for the iSCD problem we define:

True positive tp
The sentence is classified as an iSCD by the model while being an iSCD.

False positive fp
The sentence is classified as an iSCD by the model while being *content*.

False negative fn
The sentence is classified as *content* by the model while being an iSCD.

True negative tn
The sentence is classified as *content* by the model while being *content*.

Precision is a measure to evaluate how many of the detected iSCDs are actually iSCDs.

$$precision = \frac{\#tp}{\#tp + \#fp}$$

Recall is a measure to evaluate how many of all iSCDs are actually detected.

$$recall = \frac{\#tp}{\#tp + \#fn}$$

F1-score is a combination of the precision and the recall, resulting in a single score.

$$F1{-}score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

BERT's question answering use-case returns an interval as prediction. It is possible to calculate the equality between two intervals by checking if the lower and upper bounds are equal. However, intervals overlapping to a large extent can not be counted as equal and also should not be counted as completely different. The interval similarity introduced by Kabir et al. [KWH$^+$17] takes the overlapping ratio of intervals into account and yields a score between 0 and 1. We use the mean interval similarity across all samples in the test set.

Allen's interval algebra [All83] provides the relations *during* and *contains* between two intervals. The predicted interval may *contain* the SCD or the predicted interval may be located *during* the SCD. We count across all samples in the test set, how often the predicted and labeled intervals are equal or in each of the two relations. Using this counts, we can calculate accuracies for *during* and *contains*, too.

5.5. Results

In this section, we present results for the overall six approaches solving either the iSCD or the MPS²CD problem. The results are gained using the previously described implementation and workflow for experiments. First, we define for each problem and each approach a scenario to measure the performance in an experiment:

`Matrix iSCD` This scenario uses an SCD matrix to solve the iSCD problem (Section 2.2). We differentiate between two values for the threshold th, 0.55 is chosen manually and 0.49 automatically. For the first threshold, we manually test multiple values and select the best. For the second threshold, we use the 0.7 percentile of all similarity values $sim \in \mathcal{W}$ yielded by the SCDs in the training data. We only have a manually chosen threshold for the 20 newsgroups dataset and use the 0.7 percentile for all other datasets.

`BERT Classify` This scenario uses the sequence classification use-case of BERT to solve the iSCD problem (Subsection 4.1.1).

`BERT Next` This scenario uses the next sentence prediction use-case of BERT to solve the iSCD problem (Subsection 4.1.2).

`Matrix MPS²CD` This scenario uses an SCD matrix to solve the MPS²CD problem (Section 2.1). We further differentiate between the usage with a Doc2Vec `d2v` embedding or an inverse annotation `ia` queried from the annotation agent as introduced in Subsection 4.2.3.

`BERT Choose` This scenario uses the multiple choice use-case of BERT to solve the MPS²CD problem (Subsection 4.2.1).

α	β_1	β_2	ε	λ
$5 \cdot 10^{-5}$	0.9	0.999	10^{-8}	0.01
Learning rate				Weight decay

Table 5.1.: Hyperparameters used with AdamW during the fine-tuning of BERT.

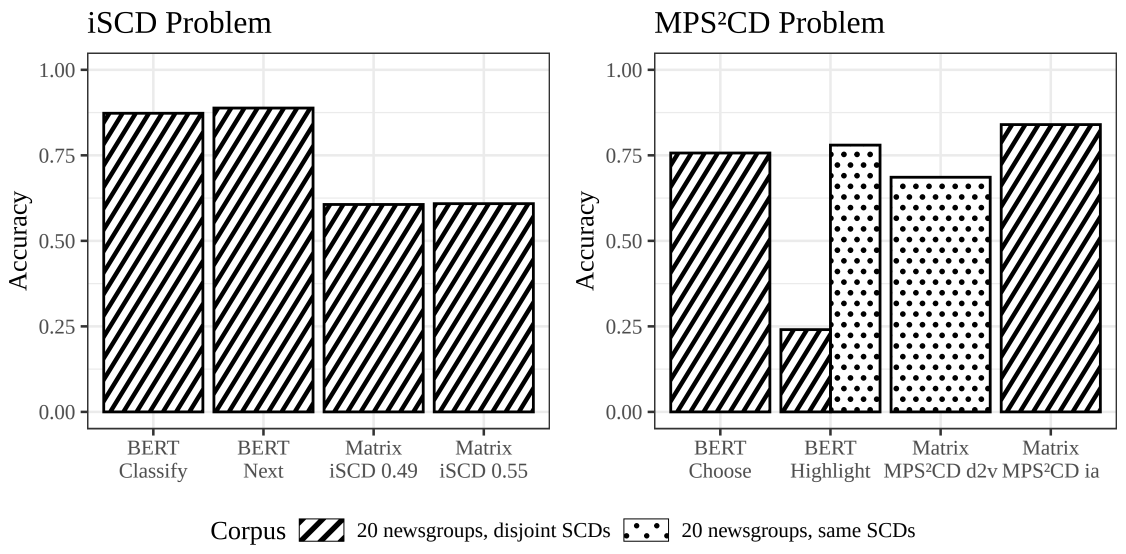

Figure 5.1.: Accuracies gained for all scenarios using the 20 newsgroups dataset and the Wiktionary annotation agent.

`BERT Highlight` This scenario uses the question answering use-case of BERT to solve the MPS²CD problem (Subsection 4.2.2). The interval returned by BERT highlights the MPS²CD for a given sentence.

We do not need to specify more hyperparameters for the scenarios using the SCD matrix. In contrast, there are multiple hyperparameters to specify for BERT. We use a batch size of 40 during fine-tuning (10 for `BERT Choose`), because 40 samples fit into the 40GB of memory of the graphics card. The pre-trained model `bert-base-uncased` uses a dropout of 0.1 and cross-entropy loss to determine the model's error.

We run the fine-tuning for 3 epochs and use AdamW [LH19]. AdamW specifies how to change the weights of the pre-trained BERT to minimize the error during fine-tuning. We test multiple values for the hyperparameters of AdamW and select the best. Table 5.1 contains the used hyperparameters. The learning rate rises linear from 0 to α in the first 500 steps of the fine-tuning.

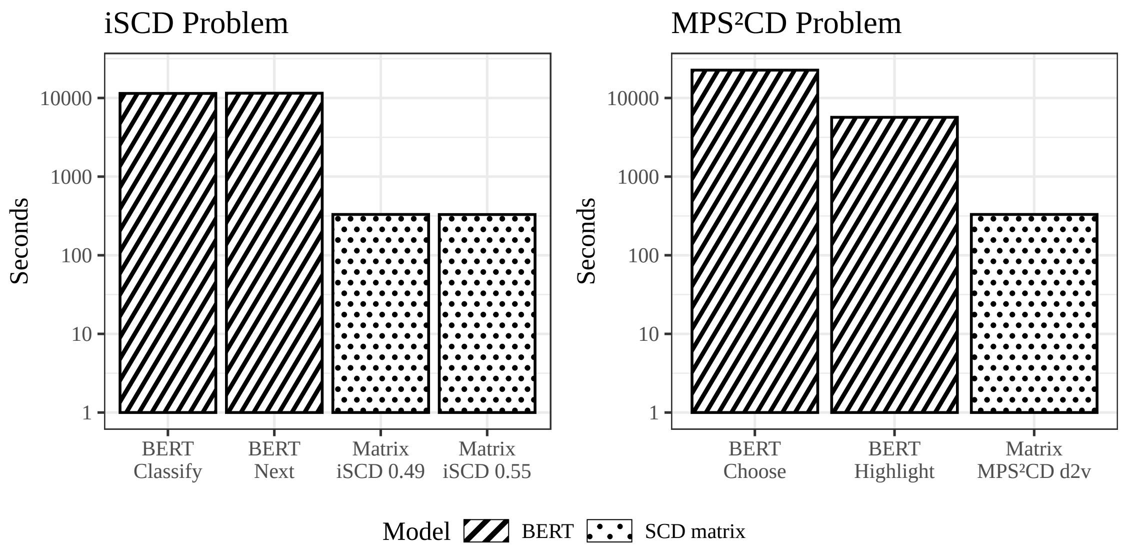

Figure 5.2.: Time needed training the models for all scenarios using the 20 newsgroups dataset and the Wiktionary annotation agent. There is no difference training the SCD matrix for `Matrix MPS²CD ia` or `Matrix MPS²CD d2v`.

The accuracies in Figure 5.1 demonstrate that BERT is good at solving the iSCD problem. There is only a very small difference between `BERT Classify` and `BERT Next`. The small difference indicates that BERT does not benefit much when getting a pair of sentence and associated SCD simultaneously. Both scenarios using the SCD matrix show nearly the same accuracy of around 0.61 and thus `Matrix iSCD` is clearly worse than `BERT Classify` and `BERT Next`.

For the MPS²CD problem, the scenarios using BERT and the SCD matrix result in similar values. Only `BERT Highlight` with a disjoint set of SCDs achieves a very low accuracy. As `BERT Highlight` asks to highlight the matching SCD out of four SCDs, the accuracy of 0.25 is as worse as randomly highlighting an SCD. We simplify the problem for `BERT Highlight` and do not split the set of SCDs. Using `BERT Highlight` with the same set of SCD, then, shows a similar performance as `BERT Choose` and `Matrix MPS²CD`. However, for `Matrix MPS²CD d2v` we also have to use the same set of SCDs.

The best accuracy for the iSCD problem is yielded by `BERT Next` and for the MPS²CD problem by `Matrix MPS²CD ia`.

Besides the performance of all scenarios, also the runtime and the computational resources needed for training are relevant. In Figure 5.2, the duration for training each of the models is shown with a logarithmic scale. The training time of an SCD

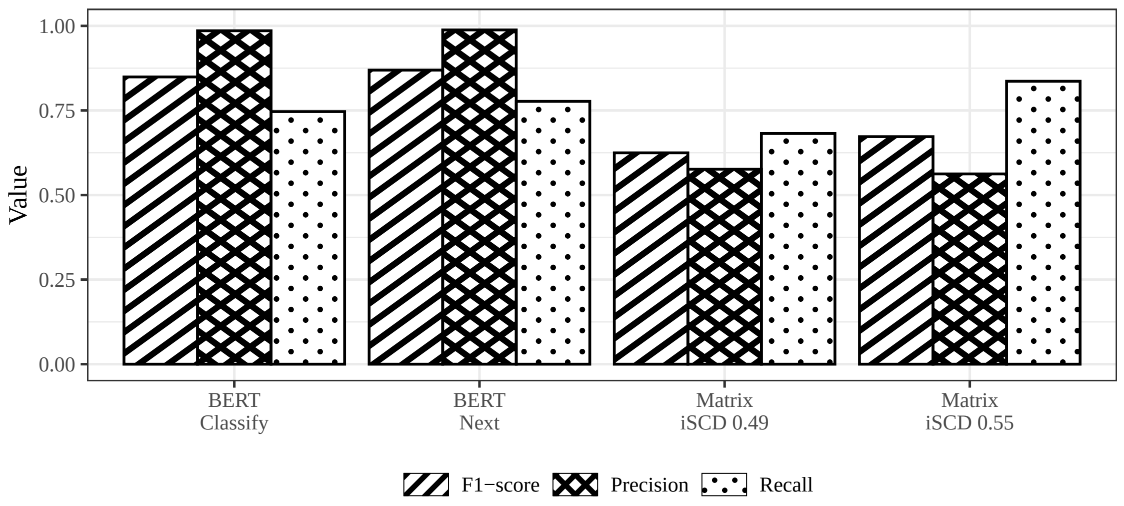

Figure 5.3.: Precison, recall and F1-score gained solving the iSCD problem on the 20 newsgroups dataset and the Wiktionary annotation agent.

matrix is always similar and very fast in contrast to the fine-tuning of BERT. Also, the training of the SCD matrix runs on a default CPU while BERT is fine-tuned on a graphics card.

5.5.1. Identifying Subjective Content Descriptions

In addition to Figure 5.1 showing the accuracies, we present in Figure 5.3 the precison, recall and F1-score for the scenarios solving the iSCD problem. Looking at the three additional measures, we can confirm that BERT is good at solving the iSCD problem.

The precision of `BERT Classify` and `BERT Next` nearly reaches 1, what means that all sentences classified as an SCD by BERT are actually an SCD. However, a precision of nearly 1 can be an indication of overfitting, i.e., the model memorizes all SCDs. In both scenarios we use a disjoint set of SCDs, thus we test the model on sentences and SCDs it has never seen before. It is unlikely that the model is overfitted, rather the iSCD problem might be a bit too simple. Nevertheless, it is a prospective task to investigate a possible overfitting further.

In our scenario of the iSCD problem, the chance for a sentence being an SCD is 50%. Thus, an accuracy of 0.61 gained by `Matrix iSCD` for both thresholds comes close to random guessing. Even though Bender et al. [BBG$^+$21b] got partially similar

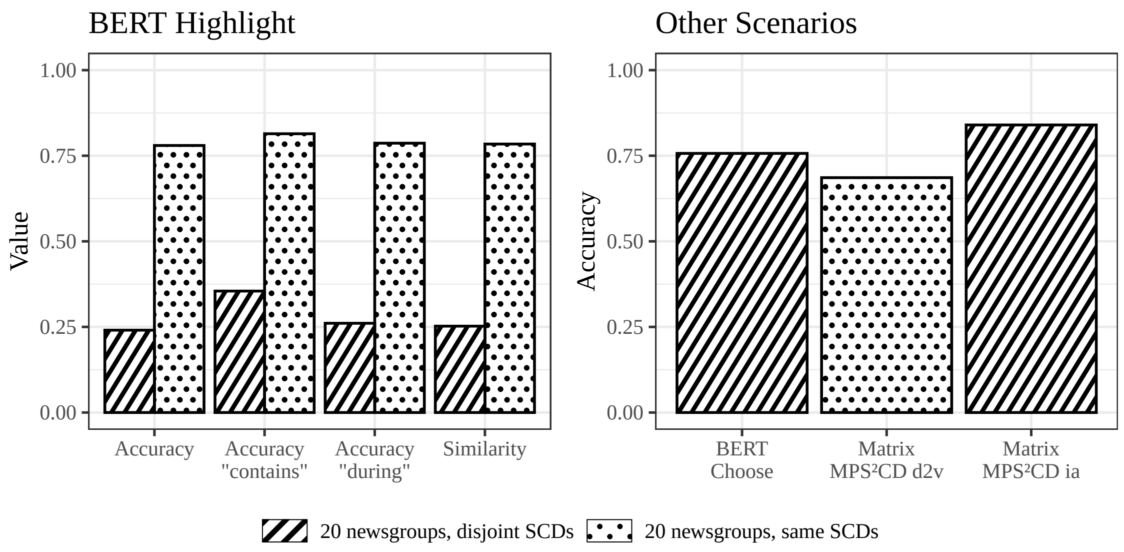

Figure 5.4.: Accuracies, interval accuracies, and interval similarities gained solving the MPS^{2}CD problem on the 20 newsgroups dataset and the Wiktionary annotation agent.

results, they identify windows of words containing the SCDs. Thus, the authors solve a slightly different and a more difficult problem, meaning they can not randomly guess to reach an accuracy of 0.5.

5.5.2. Estimating Most Probably Suited Subjective Content Descriptions

Along with Figure 5.1 we only have considered the accuracy for `BERT Highlight`, but `BERT Highlight` returns intervals and we have already defined an interval similarity and two more accuracies based on Allen's interval algebra. In Figure 5.4 all four measures for `BERT Highlight` are shown, again distinguished by using the same set or a disjoint set of SCDs.

The accuracy, accuracy *during* and interval similarity yield similar values. The accuracy *contains* yields slightly higher values than the three other measures, i.e, `BERT Highlight` returns occasionally a larger interval containing the correct SCD.

Overall `BERT Choose` and `Matrix MPS²CD` show the best performance solving the MPS^{2}CD problem.

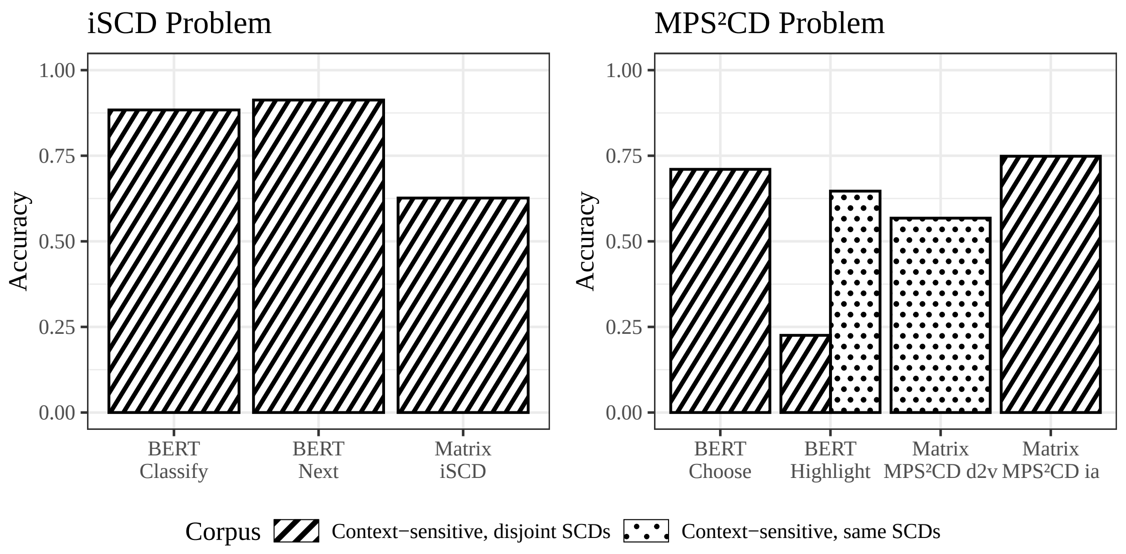

Figure 5.5.: Accuracies of the context-sensitive model using the 20 newsgroups dataset with the Wiktionary annotation agent as first context and the Manuscript cultures dataset with the quotes annotation agent as second context.

Similar to Figure 5.1, Figure 5.3, and Figure 5.4, in Section A.2 of the appendix the results using the Manuscript cultures dataset are shown.

5.6. Context-Specific Subjective Content Descriptions

In the previous sections of this chapter we have only a single context, the 20 newsgroups dataset and the Wiktionary annotation agent. However, as proposed in Section 4.3, we can use the same scenarios from the previous section to create a context-sensitive model featuring two contexts. We use a combined corpus of two different corpora with two different annotation agents, where both pairs of corpus and agent represent a different context.

The accuracies of the context-sensitive model in Figure 5.5 are very similar to the accuracies of the single-context model in Figure 5.1. The context-sensitive model reaches slightly smaller values solving the MPS²CD problem, while the relations between the accuracies of the scenarios remain the same. Surprisingly, solving the iSCD problem, the values reached by the context-sensitive model are slightly better.

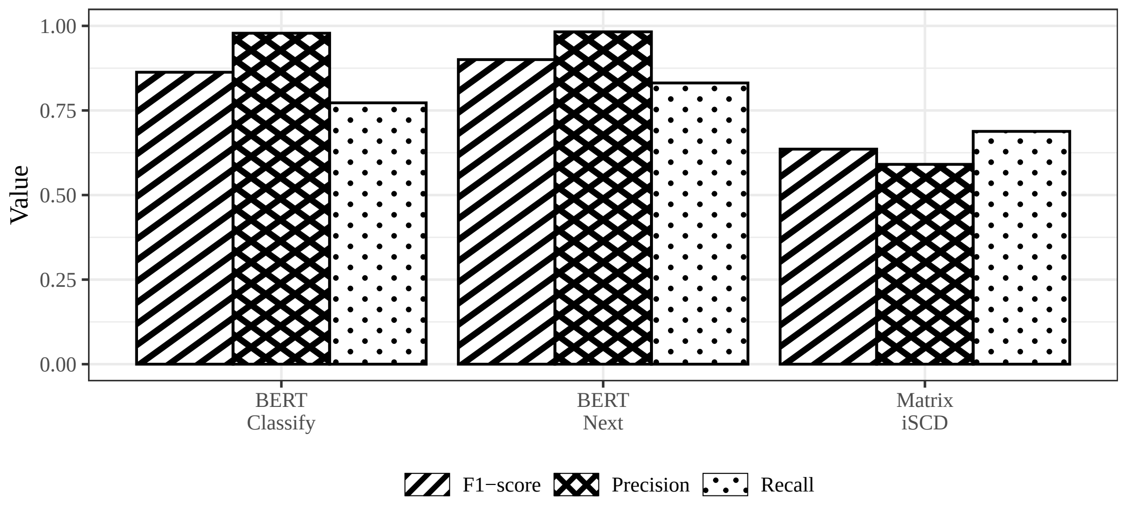

Figure 5.6.: Precison, recall and F1-score of the context-sensitive model, again using the 20 newsgroups and Manuscript cultures datasets as well as the Wiktionary and quotes annotation agents.

Considering only the iSCD problem in Figure 5.6, the results are again similar to the results of the single-context model in Figure 5.3. We use only the 0.7 percentile as threshold for `Matrix iSCD`. The precision of `BERT Classify` and `BERT Next` is slightly lower, making an overfitting less likely.

The context-sensitive model shows the potential of BERT and the SCD matrix for estimating SCDs. Although, the model takes a second objective, i.e., detecting the context, the overall performance of the model is minimally reduced at most. In [BBG+21a] the authors present an approach selecting context-dependent dictionaries for documents and afterwards solve the iSCD problem for the documents. Using the context-sensitive model, we do not need a second approach to detect contexts.

6. Conclusion and Outlook

In this thesis, we first introduce SCDs and TLMs. Thereby, we focus on the SCD matrix to solve the iSCD and MPS^{2}CD problem. Also, we describe the attention mechanism used by TLMs and the architecture of BERT.

In the main part, we introduce how to apply TLMs to SCDs. We enumerate the different use-cases of BERT and describe for each use-case how to solve the iSCD or MPS^{2}CD problem. We also propose a context-sensitive model, capturing the context of a text document yielding context-specific SCDs.

At the end of this thesis, we provide an extensive evaluation applying BERT to SCDs. Summarized, the evaluation shows that BERT can be fined-tuned well to represent SCDs for text documents. On the iSCD problem, BERT performs better than the approaches using the SCD matrix. On the MPS^{2}CD problem, the approaches using the SCD matrix perform slightly better, however, BERT performs similar.

We recommend to use BERT for solving the iSCD problem and `Matrix MPS²CD` for solving the MPS^{2}CD problem. `Matrix MPS²CD` and `BERT Choose` reach nearly the same accuracy, while `Matrix MPS²CD` needs less computational resources.

We conclude that TLMs are able to grasp the concept of SCDs, in a way that TLMs can be trained to solve SCD-related tasks. Also, the SCD matrix and BERT provide enough capacity to additionally learn the context of corpora and yield context-specific results.

In the field of this thesis, it would be interesting to use the token classification use-case of BERT and compare the results with the window-based approach solving the iSCD problem introduced by Bender et al. Another possibility is to use other TLMs similar to BERT and rerun the evaluation of this thesis.

DistilBERT [SDCW19] is a distilled version of BERT. Distillation is a mechanism to reduce the size of a model by trying to imitate a larger model focusing on the model's task. Distilled TLMs often provide better results and use less computational resources. Longformer [BPC20] is a TLM with no limit on the length of the input sequence. Longer input sequences would allow us to run `BERT Highlight` with more than four SCDs to select from.

A. Appendix

A.1. Less Preprocessing

The accuracies in Figure A.1 are gained for the same scenarios as in Figure 5.1, but we do not preprocess the inputs. We run only the tokenizing mechanism for BERT and neither stemming nor stop word elimination. We use only the 0.7 percentile to estimate the threshold for `Matrix iSCD`.

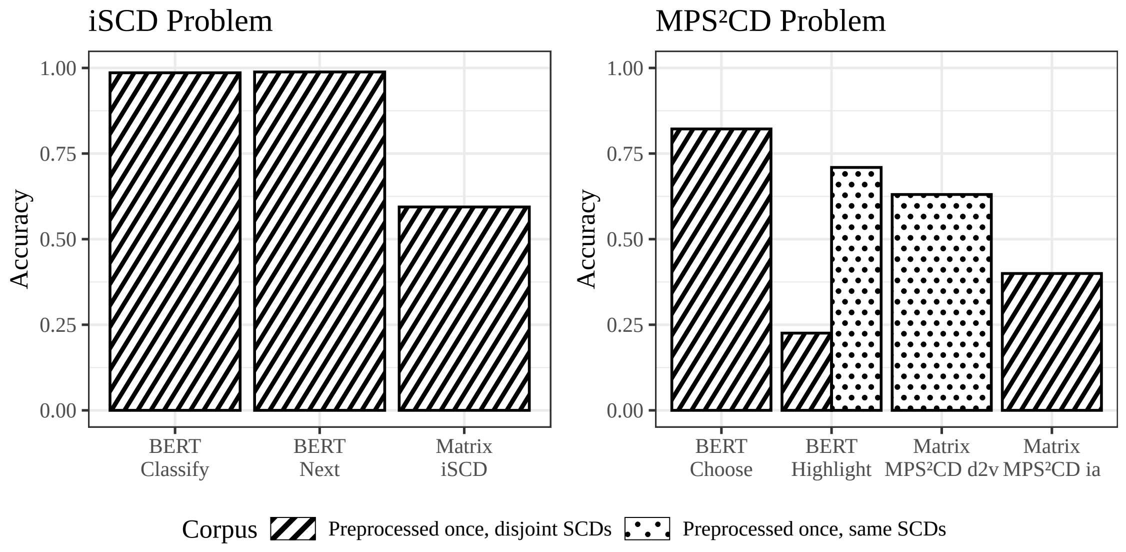

Figure A.1.: The accuracies when preprocessing the inputs for BERT only once and skipping (ii) from Section 5.3.

A.2. Additional Corpus

The same plots as in Section 5.5, but running the experiments on the Manuscript cultures[1] dataset. We use only the 0.7 percentile to estimate the threshold for `Matrix iSCD`.

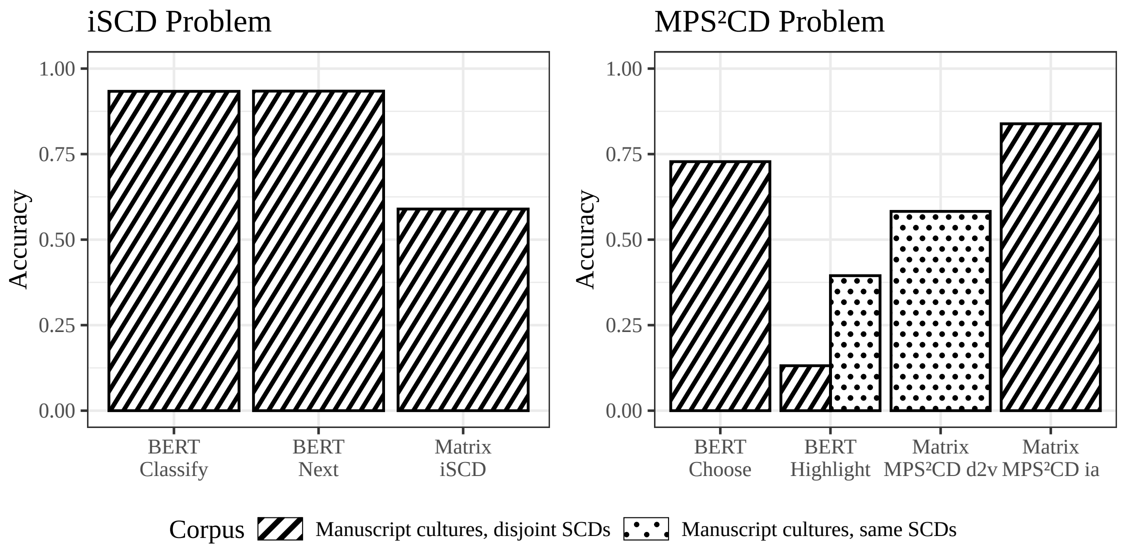

Figure A.2.: Accuracies gained for all scenarios using the Manuscript cultures dataset and the Wiktionary annotation agent.

[1]`https://www.csmc.uni-hamburg.de/publications/mc.html`

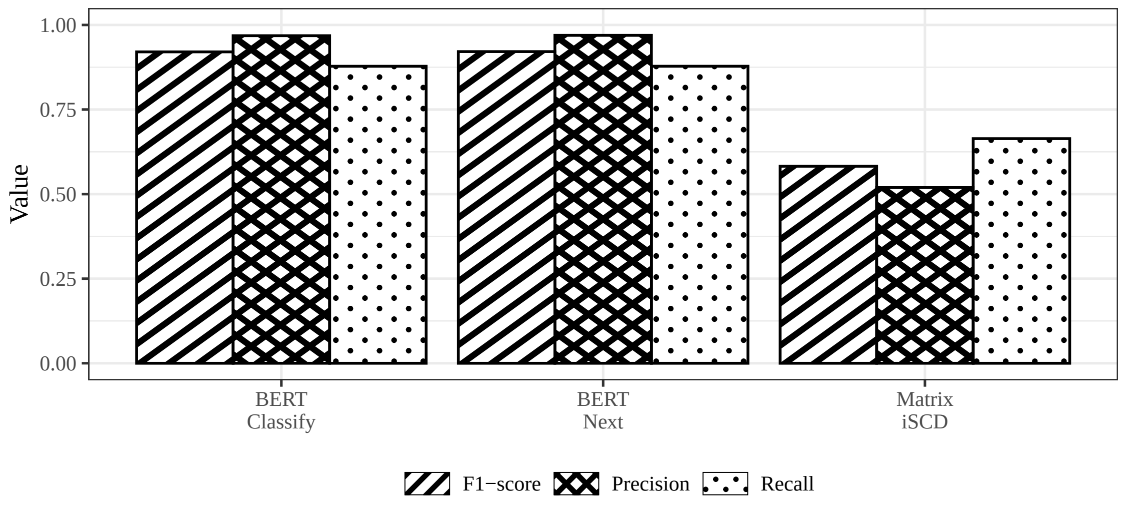

Figure A.3.: Performance solving the iSCD problem on the Manuscript cultures dataset and the Wiktionary annotation agent.

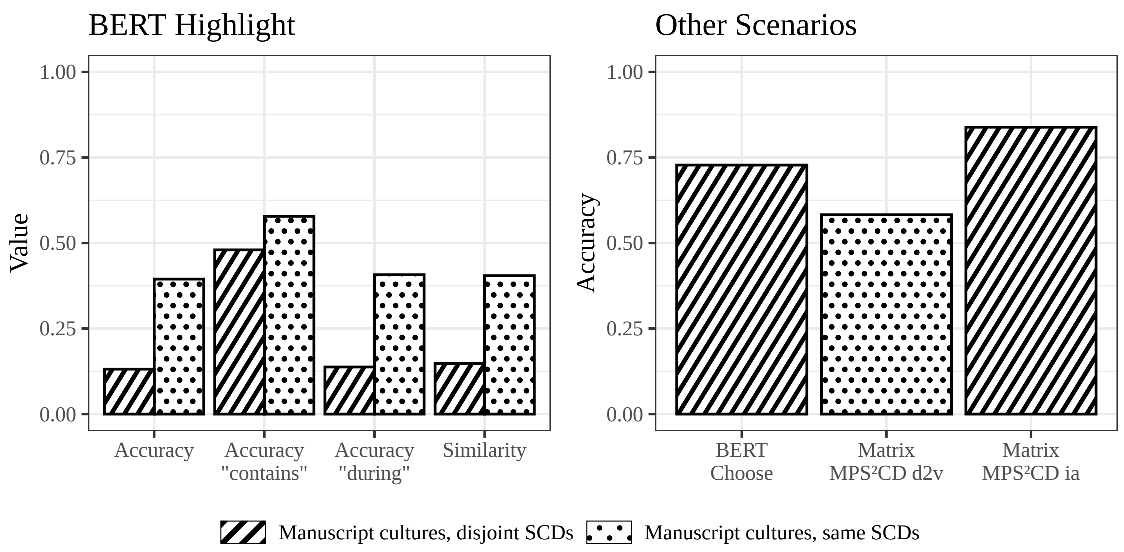

Figure A.4.: Performance solving the MPS^{2}CD problem on the Manuscript cultures dataset and the Wiktionary annotation agent.

List of Figures

Bibliography

[All83] Allen, James F.: Maintaining Knowledge about Temporal Intervals. In: *Commun. ACM* 26 (1983), November, Nr. 11, 832–843. https://doi.org/10.1145/182.358434

[BBG+21a] Bender, Magnus ; Braun, Tanya ; Gehrke, Marcel ; Kuhr, Felix ; Möller, Ralf ; Schiff, Simon: Identifying and Translating Subjective Content Descriptions Among Texts. In: *Int. J. Semantic Computing* 15 (2021). – Accepted for publication

[BBG+21b] Bender, Magnus ; Braun, Tanya ; Gehrke, Marcel ; Kuhr, Felix ; Möller, Ralf ; Schiff, Simon: Identifying Subjective Content Descriptions among Text. In: *Proceedings of the 15th IEEE International Conference on Semantic Computing (ICSC-21)* (2021). https://doi.org/10.1109/ICSC50631.2021.00008

[BMR+20] Brown, Tom B. ; Mann, Benjamin ; Ryder, Nick ; Subbiah, Melanie ; Kaplan, Jared ; Dhariwal, Prafulla ; Neelakantan, Arvind ; Shyam, Pranav ; Sastry, Girish ; Askell, Amanda ; Agarwal, Sandhini ; Herbert-Voss, Ariel ; Krueger, Gretchen ; Henighan, Tom ; Child, Rewon ; Ramesh, Aditya ; Ziegler, Daniel M. ; Wu, Jeffrey ; Winter, Clemens ; Hesse, Christopher ; Chen, Mark ; Sigler, Eric ; Litwin, Mateusz ; Gray, Scott ; Chess, Benjamin ; Clark, Jack ; Berner, Christopher ; McCandlish, Sam ; Radford, Alec ; Sutskever, Ilya ; Amodei, Dario: Language Models are Few-Shot Learners. (2020). https://arxiv.org/abs/2005.14165

[BPC20] Beltagy, Iz ; Peters, Matthew E. ; Cohan, Arman: Longformer: The Long-Document Transformer. In: *CoRR* (2020). https://arxiv.org/abs/2004.05150

[BRC+16] Beltagy, I. ; Roller, Stephen ; Cheng, Pengxiang ; Erk, Katrin ; Mooney, Raymond J.: Representing Meaning with a Combination of Logical and Distributional Models. In: *Computational Linguistics* 42 (2016), Dezember, Nr. 4, 763–808. https://doi.org/10.1162/COLI_a_00266

[DCLT19] Devlin, Jacob ; Chang, Ming-Wei ; Lee, Kenton ; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (2019). https://arxiv.org/abs/1810.04805

[GMG18] Goel, Shivali ; Madhok, Rishi ; Garg, Shweta: Proposing Contextually Relevant Quotes for Images. In: *Advances in Information Retrieval* (2018). https://doi.org/10.1007/978-3-319-76941-7_49

[HS97] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-Term Memory. In: *Neural Comput.* 9 (1997), November, Nr. 8, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735. – ISSN 0899–7667

[JPH18] Joshi, Vidur ; Peters, Matthew ; Hopkins, Mark: Extending a Parser to Distant Domains Using a Few Dozen Partially Annotated Examples. (2018). https://arxiv.org/abs/1805.06556

[KBBM19] Kuhr, Felix ; Braun, Tanya ; Bender, Magnus ; Möller, Ralf: To Extend or not to Extend? Context-specific Corpus Enrichment. In: *Proceedings of AI 2019: Advances in Artificial Intelligence* (2019), 357–368. https://doi.org/10.1007/978-3-030-35288-2_29. ISBN 978–3–030–35288–2

[KBBM20] Kuhr, Felix ; Bender, Magnus ; Braun, Tanya ; Möller, Ralf: Augmenting and Automating Corpus Enrichment. In: *Int. J. Semantic Computing* 14 (2020), Nr. 2, 173–197. https://doi.org/10.1142/S1793351X20400061

[KWH+17] Kabir, Shaily ; Wagner, Christian ; Havens, Timothy C. ; Anderson, Derek T. ; Aickelin, Uwe: Novel similarity measure for interval-valued data based on overlapping ratio. In: *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (2017), 1-6. https://doi.org/10.1109/FUZZ-IEEE.2017.8015623

[LH19] Loshchilov, Ilya ; Hutter, Frank: Decoupled Weight Decay Regularization. (2019). https://arxiv.org/abs/1711.05101

[LM14] Le, Quoc V. ; Mikolov, Tomas: Distributed Representations of Sentences and Documents. (2014). https://arxiv.org/abs/1405.4053

[Mav69] Maverick, George V.: Computational Analysis of Present-Day American English. Henry Kučera, W. Nelson Francis. In: *International Journal of American Linguistics* 35 (1969), Nr. 1, 71-75. https://doi.org/10.1086/465045

[MDSS21] Mishra, Prakhar ; Diwan, Chaitali ; Srinivasa, Srinath ; Srinivasaraghavan, G.: Automatic Title Generation for Text with Pre-trained Transformer Language Model. (2021), 17-24. https://doi.org/10.1109/ICSC50631.2021.00009

[PABP17] Peters, Matthew E. ; Ammar, Waleed ; Bhagavatula, Chandra ; Power, Russell: Semi-supervised sequence tagging with bidirectional language models. (2017). https://arxiv.org/abs/1705.00108

[RHW86] Rumelhart, D. ; Hinton, Geoffrey E. ; Williams, R. J.: Learning representations by back-propagating errors. In: *Nature* 323 (1986), 533-536. https://doi.org/10.1038/323533a0

[RN10] Russel, Stuart ; Norvig, Peter: *Artificial Intelligence - A Modern Approach*. Third Edit. Pearson Education, Inc., 2010 http://aima.cs.berkeley.edu/index.html

[RN18] Radford, Alec ; Narasimhan, Karthik: Improving Language Understanding by Generative Pre-Training. (2018). https://api.semanticscholar.org/CorpusID:49313245

[RWC$^+$19] Radford, Alec ; Wu, Jeff ; Child, Rewon ; Luan, David ; Amodei, Dario ; Sutskever, Ilya: Language Models are Unsupervised Multitask Learners. (2019). https://api.semanticscholar.org/CorpusID:160025533

[SBDS14] Sabou, Marta ; Bontcheva, Kalina ; Derczynski, Leon ; Scharl, Arno: Corpus Annotation through Crowdsourcing: Towards Best Practice Guidelines. In: *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC'14)* (2014), 01. http://www.lrec-conf.org/proceedings/lrec2014/pdf/497_Paper.pdf

[SCBC20] Shanahan, Murray ; Crosby, Matthew ; Beyret, Benjamin ; Cheke, Lucy: Artificial Intelligence and the Common Sense of Animals. In: *Trends in cognitive sciences* 24 (2020), 11, 862-872. https://doi.org/10.1016/j.tics.2020.09.002

[SDCW19] Sanh, Victor ; Debut, Lysandre ; Chaumond, Julien ; Wolf, Thomas: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In: *CoRR* abs/1910.01108 (2019). http://arxiv.org/abs/1910.01108

[VIN15] Vijayarani, S. ; Ilamathi, J. ; Nithya, S.: Preprocessing Techniques for Text Mining - An Overview. In: *International Journal of Computer Science & Communication Networks* 5 (2015), S. 7–16

[Vit67] Viterbi, A.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In: *IEEE Transactions on Information Theory* 13 (1967), Nr. 2, 260-269. https://doi.org/10.1109/TIT.1967.1054010

[VSP+17] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Lukasz ; Polosukhin, Illia: Attention Is All You Need. (2017). https://arxiv.org/abs/1706.03762

[WSC+16] Wu, Yonghui ; Schuster, Mike ; Chen, Zhifeng ; Le, Quoc V. ; Norouzi, Mohammad ; Macherey, Wolfgang ; Krikun, Maxim ; Cao, Yuan ; Gao, Qin ; Macherey, Klaus ; Klingner, Jeff ; Shah, Apurva ; Johnson, Melvin ; Liu, Xiaobing ; Kaiser Łukasz ; Gouws, Stephan ; Kato, Yoshikiyo ; Kudo, Taku ; Kazawa, Hideto ; Stevens, Keith ; Kurian, George ; Patil, Nishant ; Wang, Wei ; Young, Cliff ; Smith, Jason ; Riesa, Jason ; Rudnick, Alex ; Vinyals, Oriol ; Corrado, Greg ; Hughes, Macduff ; Dean, Jeffrey: Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. (2016). https://arxiv.org/abs/1609.08144

[ZBSC18] Zellers, Rowan ; Bisk, Yonatan ; Schwartz, Roy ; Choi, Yejin: SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference. (2018). https://arxiv.org/abs/1808.05326