JOHANNES KEPLER
UNIVERSITY LINZ

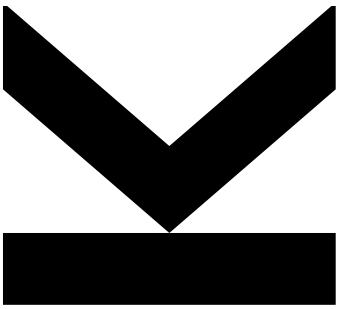
Submitted by
Christian Schießl

Submitted at
Institute for Business Informatics - Data & Knowledge Engineering

Supervisor
Assoz.-Prof. Mag. Dr. Christoph Schütz

December 2024

Design and Implementation of a Data Warehouse System for Online Analytical Processing of Reviews on University Evaluation Platforms



Master Thesis

to obtain the academic degree of

Master of Science

in the Master's Program

Business Informatics

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Dr. Christoph Schütz, for his assistance and guidance throughout the course of this thesis. I also extend my thanks to Professor Dr. Christoph Teller for his inspiration in helping me find the topic for this research. Finally, I would like to thank my family and friends for their unwavering support and encouragement during the entire thesis process.

Abstract

This thesis presents the design and implementation of a data warehouse system aimed at analyzing student sentiment in online reviews from university evaluation platforms. By applying Aspect-Based Sentiment Analysis (ABSA) using BERT, the system transforms unstructured textual data into a structured format suitable for analysis through Online Analytical Processing (OLAP). The system offers Higher Education Institutions (HEIs) a multidimensional view of student satisfaction, focusing on aspects such as study programs, time periods, and specific attributes of the educational experience.

A prototype system was developed to collect, preprocess, and analyze German-language online reviews related to Austrian universities. It demonstrated how ABSA can provide actionable insights into student feedback, which can assist HEI administrators in improving student retention and recruitment efforts. The system's ability to analyze large volumes of unstructured data opens up new possibilities for data-driven decision-making in academia.

Several limitations were encountered during the project, such as the restricted coverage of aspect categories and the difficulty in addressing complex linguistic features, i.e., irony and sarcasm. Future work could focus on increasing the system's reliability by expanding data sources, integrating topic modeling techniques, i.e., Latent Dirichlet Allocation (LDA), and improving the handling of contextual nuances. In addition, adapting the aspect categories to specific courses or programs could further enhance the accuracy of the analysis.

This thesis provides a foundation for automating student feedback analysis at HEIs and sets the stage for future research to refine and expand the system's capabilities.

Kurzfassung

Diese Arbeit behandelt das Konzept und die Implementierung eines Data-Warehouse-Systems, das Bewertungen aus Bewertungsportalen im Hochschulbereich hinsichtlich der Zufriedenheit der Studierenden mit spezifischen Aspekten der Hochschul-Services auswerten soll.

Mithilfe von BERT und Aspect-Based Sentiment Analysis (ABSA) werden unstrukturierte Texte aus den Bewertungen so aufbereitet, dass sie im Rahmen des Online Analytical Processing (OLAP) analysiert werden können. Das System bietet Hochschuleinrichtungen eine mehrdimensionale Sicht auf die Zufriedenheit der Studierenden, die verschiedene Aspekte wie Studiengänge, Zeiträume und spezifische Hochschul-Services umfasst.

Ein Prototyp des Systems wurde entwickelt, um deutschsprachige Online-Bewertungen von österreichischen Hochschulen zu sammeln, zu verarbeiten und zu analysieren. Es wurde gezeigt, wie ABSA wertvolle Einblicke in die Meinungen der Studierenden liefern kann, die der Hochschulverwaltung helfen können, die Bindung und Rekrutierung von Studierenden zu verbessern. Die Fähigkeit des Systems, große Mengen unstrukturierter Daten zu sammeln und zu analysieren, eröffnet neue Möglichkeiten für eine datengestützte Entscheidungsfindung im Hochschulbereich.

Während des Projekts traten mehrere Einschränkungen auf, wie die begrenzte Abdeckung von Aspekt-Kategorien und die Schwierigkeit, komplexe sprachliche Merkmale wie Ironie und Sarkasmus zu berücksichtigen. Zukünftige Arbeiten könnten sich darauf konzentrieren, die Zuverlässigkeit des Systems durch die Erweiterung der Datenquellen, die Integration von Topic-Modeling-Techniken wie Latent Dirichlet Allocation (LDA) und die Verbesserung des Umgangs mit kontextuellen Nuancen zu erhöhen. Darüber hinaus könnte eine Anpassung der Aspekt-Kategorien an spezifische Kurse oder Studiengänge die Genauigkeit der Analyse weiter steigern.

Diese Arbeit liefert eine Grundlage für die Automatisierung der Analyse von Studenten-Feedback und schafft die Voraussetzungen für zukünftige Forschungen zur Verfeinerung und Erweiterung der Fähigkeiten des Systems.

Contents

1

Introduction

Higher education institutions (HEIs) face the challenge of positioning themselves in an increasingly competitive environment (Hemsley-Brown and Oplatka, 2006).

Retention and student recruitment are important aspects for HEIs (Han, 2014). University rankings may provide a first indication for managers of how well a university is positioned in comparison to others, especially in terms of academic and employer reputation. However, they have many downsides and do not reflect student satisfaction (Johnes, 2018; Moustafa, 2024).

This is linked to the fact that, even though marketing processes are generally well-established in HEIs, there are often disagreements about who actually represents the customer of HEIs (Guilbault, 2016). Research on marketing and customer satisfaction in higher education shows that there has been a long-standing debate over whether students should be considered customers (Alford, 2002, Olshavsky and Spreng, 1995, Pitman, 2000). Even though it is clear that, as in other industries, HEIs can have a variety of customers, including employers and other stakeholders, failing to recognize students as customers could negatively impact the orientation of the services offered and thus jeopardize student satisfaction (Guilbault, 2016). As a market-driven orientation aims to satisfy customer needs (Kotler, 1977), it is important to address the customer question.

Student feedback can be found in a variety of sources. A course evaluation conducted by the university each semester is just one example. Students' opinions about their study experience can also be found on various internet platforms, such as forums, blogs, Facebook, Twitter, and rating platforms. The data is mostly available in a highly unstructured form. This means that the required data is not always available in a structured format, unlike data from Excel files or databases. One popular way to express an opinion about a product or service is through online rating platforms. Nowadays, anyone using the internet can easily write a review, provide feedback, or rate any product or service. For instance, with products offered on Amazon, customers can rate the product they purchased. They can generally rate the product by selecting a number between 1 and 5, which constitutes the so-called star rating. In addition, they can write a review in the form of an open text comment. In the comment, customers can share their experiences using the product or service. Reja et al. (2003) argues that, in contrast to closed-ended questions, which limit customers to a set of predefined answer options, open-ended questions allow them to express their opinions freely without being influenced by predefined response alternatives. They encourage customers to provide spontaneous responses, giving product and service providers the opportunity to uncover hidden insights about customer satisfaction (Reja et al., 2003).

In higher education, there are numerous platforms where current and former students can write reviews about their experiences in study programs. Whether a review consists of a single text field or multiple fields with various open-ended questions, the result is ultimately unstructured user-generated content (UGC). While these reviews are a useful source for those interested in studying and deciding where to apply, they remain an underused resource for HEIs themselves. This is because it is not feasible for universities to manually read all reviews; thus, computational processing is needed.

Automatic processing and analysis of student feedback (i.e., online reviews) can be achieved by introducing a data warehouse system, which, in contrast to operational database systems, is used for analytical purposes in organizations (Kimball and Ross, 2011). However, reviews sourced from university evaluation platforms (UEPs) represent semi-structured data, containing both structured elements like star ratings and unstructured data like free-text comments. Therefore, in order to analyze the student feedback, the unstructured part of the reviews—the comments themselves—must be transformed into a structured format appropriate for storage and analysis in a data warehouse.

Sentiment analysis, a Natural Language Processing (NLP) technique, is a popular method to achieve this transformation. Aspect-based sentiment analysis (ABSA) allows for a more fine-grained analysis (B. Liu, 2022). Once sentiments expressed over a certain aspect are extracted, they can be integrated into a data warehouse, representing structured data in a multidimensional view. This data structure enables Online Analytical Processing (OLAP), a mechanism used in data warehouses for data analysis (Vaisman and Zimányi, 2014). This process allows HEIs to analyze the latent sentiments found in online reviews across multiple dimensions—such as aspects, study programs, and time. As a result, administrators at HEIs can gain insights into student satisfaction, identifying strengths and weaknesses in their services compared to other HEIs. Ultimately, this enables them to make data-driven decisions that enhance both retention and student recruitment.

One language model developed by Google that has gained recognition recently is BERT. BERT can be applied in ABSA, as it was trained on large amounts of data and has achieved state-of-the-art performance on various NLP tasks (Devlin et al., 2018).

Research and practical applications on how HEIs can design and implement a data warehouse system applying ABSA to student reviews remain scarce (Melba Rosalind and Suguna, 2022). This thesis aims to address this gap by examining how a data warehouse system can be designed and implemented to analyze student sentiment in online reviews from university evaluation platforms. This will enable both instructors and administrators at HEIs to systematically analyze students' sentiments from online reviews on university evaluation platforms.

Following the design science approach, this thesis aims to develop a prototype of a data warehouse system representing the artifact, starting from collecting online reviews from a data source representing student feedback on studying in Austrian HEIs, until loading sentiment analysis results into a data destination. The system is intended to be used again, whether for further development or reconstructive research.

Different techniques and methods available for analyzing sentiments in textual data are examined. In this context, aspect-level approaches are focused on, as they have a greater impact on valuable insights. Furthermore, to address the context of Austrian HEIs, data in the form of German-

language online reviews are collected from a university evaluation platform on which students can write about their study experiences at a certain HEI in Austria.

After collecting and preprocessing the reviews, a training dataset is created for the customized fine-tuning of a BERT-based language model. This is done using *PyABSA*, a Python framework for ABSA. The adapted model is then used to perform ABSA on the underlying student feedback. The resulting predictions, including aspects and corresponding sentiments, are then loaded into a data warehouse for final analysis purposes.

The data warehouse system architecture designed in this thesis is based on the reference architecture from Vaisman and Zimányi (2014), presented in Chapter 2. As a matter of preference, the system's logical implementation follows the relational OLAP (ROLAP) approach, which means that the data warehouse is implemented in a relational database (Vaisman and Zimányi, 2014). Consequently, no physical cube was generated, as would be the case with the multidimensional OLAP (MOLAP) approach, where data is physically stored in multidimensional structures (Kimball and Ross, 2011). However, logically, the data warehouse is treated as a multidimensional cube in this thesis.

The designed architecture consists of four components: the data source, which represents the university evaluation platform; a back-end tier for data collection and sentiment analysis; a data warehouse tier representing the defined sentiment cube; and a front-end tier consisting of SQL queries for analyzing the cube.

The back-end tier consists of two modules: data collection and sentiment analysis. The data collection process, which is performed through web scraping, is detailed in Chapter 4. Chapter 5 focuses on sentiment analysis as a central function, including the fine-tuning of a custom ABSA model and the development of a classifier for aspect categories. Chapter 6 then covers the data warehouse tier, which includes the creation of a sentiment cube to enable stakeholders at higher education institutions (HEIs) to analyze sentiments associated with specific aspects. Finally, examples of SQL queries, which represent a rudimentary form of the front-end tier, are discussed in Section 6.4.

The thesis proceeds with Chapter 2, which provides the theoretical background necessary for further understanding. Theory about data warehousing and OLAP is covered in Section 2.1, while aspect-based sentiment analysis is addressed in Section 2.2. Lastly, related works are discussed in Section 2.3. Chapter 3 serves as an introduction to the main part of the thesis, providing an overview of the training process for developing the ABSA model and also describing the architecture of the designed data warehouse system. The final chapter concludes the thesis with a critical discussion of the undertaken approach, illustrating its limitations and suggesting directions for future research.

2

Background

This chapter provides a theoretical overview of the underlying concepts that are crucial to understanding this research.

It starts by explaining the concepts of data warehousing and OLAP in Section 2.1. Next, the topic of Aspect-Based Sentiment Analysis (ABSA) is covered in Section 2.2. Section 2.2.1 provides a brief introduction to aspect-based sentiment analysis, and Section 2.2.2 clarifies how researchers have addressed ABSA tasks by training large language models (LLMs), showcasing potential challenges and strategies for overcoming them. Finally, Section 2.3 outlines related work. Section 2.3.1 covers related studies focusing on ABSA applications designed for the analysis of university reviews, while Section 2.3.2 covers related works addressing data warehouses and OLAP systems in terms of social business intelligence.

2.1 Data Warehousing and OLAP

Vaisman and Zimányi (2014) explain that operational and transactional databases were designed to address day-to-day tasks, such as processing order income. However, they are insufficient for data analysis and, consequently, decision-making processes. Therefore, to address analytical requirements, data warehousing and Online Analytical Processing (OLAP) were introduced in the 1990s. They are considered database technologies and encompass architectures, tools, techniques, and algorithms to integrate various data sources into a centralized location for later analysis.

Vaisman and Zimányi (2014) describe the term data warehouse as a location where data is stored, while a data warehouse system additionally includes back-end tools for data collection and front-end tools for presenting data. OLAP, on the other hand, merely represents the mechanism used to access and analyze the data, whereas an OLAP system provides users with the ability to query and aggregate the data stored in the data warehouse (Vaisman and Zimányi, 2014). OLAP itself stands for Online Analytical Processing, which Chaudhuri and Dayal (1997) describes as "functional and performance requirements" that are to be supported by a data warehouse (Chaudhuri and Dayal, 1997, p. 1).

Those requirements call for data structures that are different from those of traditional databases. According to Kimball and Ross (2011), data in a data warehouse is organized in alignment with the multidimensional model. The multidimensional model consists of one or more facts representing the business interests for the decision-making process and dimensions representing

the parameters to analyze the facts (Kimball and Ross, 2011). This n-dimensional view on data is often referred to as data cubes (Vaisman and Zimányi, 2014). Therefore, for the sake of simplicity, the multidimensional data structure developed in this thesis may be referred to as a sentiment cube.

The architecture of the data warehouse system to be designed in this thesis is based on the reference architecture for data warehouse systems proposed by Vaisman and Zimányi (2014). It is illustrated in Figure 2.1 and outlined briefly below.

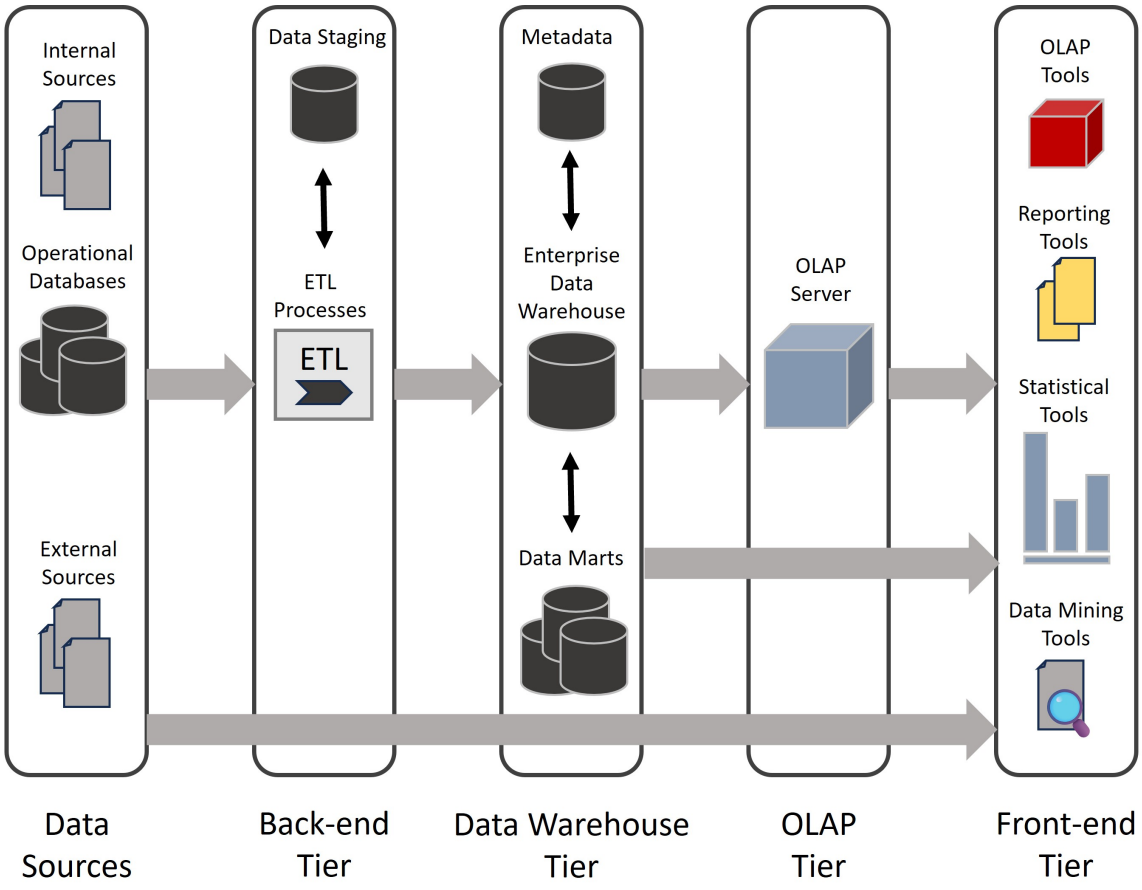


Figure 2.1: Architecture of a Data Warehouse System. Own illustration, adapted from Vaisman and Zimányi (2014).

This architecture consists of five key components: data sources, back-end tier, data warehouse tier, OLAP tier, and front-end tier. The data can originate from both internal and external sources. In many cases, they represent the operational database systems of an organization. The back-end tier covers ETL processes in the data staging area, which stands for extraction, transformation, and loading. The data is extracted from the sources into the data staging area, transformed to fit the data warehouse model, and then loaded into the next tier: the data warehouse tier. This stage is represented by an enterprise data warehouse (EDW) and/or several data marts specialized for functional departments. In addition, metadata is typically documented to describe the underlying data, such as the DW model. The next tier is the OLAP tier, which consists of an OLAP server for data presentation purposes. Often, it is an extension of a database product used for constructing and querying OLAP cubes, regardless of how the data is stored in the back-end. Finally, the front-end tier is used for data analysis and visualization. It can consist of many client tools, such as statistical or data mining tools, for analyzing the data (Vaisman and Zimányi, 2014).

Vaisman and Zimányi (2014) explain that in practice, some of the architecture components may not be used. For instance, in some cases, a data warehouse may not exist and must be created by integrating data marts. In another scenario, client tools operate on the data warehouse without an OLAP server in between. In an extreme case, there is only a virtual data warehouse represented by predefined views over operational databases (Vaisman and Zimányi, 2014).

2.2 Aspect-Based Sentiment Analysis

This section discusses the topic of sentiment analysis, which serves as the core functionality in this thesis for transforming the underlying data into valuable insights. It begins with Section 2.2.1, which introduces aspect-based sentiment analysis, and concludes with Section 2.2.2, illustrating how LLMs might be applied to ABSA tasks.

2.2.1 Introduction to Aspect-Based Sentiment Analysis (ABSA)

To understand the sub-field of aspect-based sentiment analysis (ABSA), it is necessary to first explain sentiment analysis (SA) in general. The terms sentiment analysis and opinion are often used synonymously, as they are commonly exchanged in practice. In this research, however, only the term sentiment analysis is used. According to B. Liu (2017), it is defined as the study and computational analysis of human sentiments, which can cover opinions, attitudes, evaluations, and moods of humans. It is a popular research area in natural language processing (NLP). Moreover, it is applied in information retrieval, data mining, and web mining. Since human opinions are of great interest to business and society, sentiment analysis and its applications have also gained importance in management sciences and social sciences (B. Liu, 2017). Sentiment analysis is a natural language processing task; therefore, it is important to clarify what NLP means. Liddy (2001) defines NLP as *"a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications"* (Liddy, 2001, p. 3).

B. Liu (2022) explains that while document-level analysis results in determining a single overall polarity for an entire text, sentence-level analysis aims to determine one polarity for a single sentence. Depending on the use case, this may suffice. However, in practice, business stakeholders often seek to gain more detailed knowledge on how customers perceive different aspects of certain entities (Siegel and Alexa, 2020).

At this point, aspect-level analysis, or ABSA, becomes relevant. It is *"a natural language processing technique that seeks to recognize and extract the sentiment connected to various qualities or aspects of a specific good, service, or entity"* (Kandhro et al., 2024, p. 1). According to W. Zhang et al. (2023), ABSA can also be seen as the extraction and determination of aspect-level sentiment elements. These elements include aspect term, aspect category (entity), opinion term, and sentiment polarity. For instance, in the sentence *The course material provided was relevant and interesting*, the sentiment elements would be *course material* (aspect term), *courses* (aspect category), *relevant and interesting* (opinion term), and *positive* (sentiment polarity) respectively. While the aspect term *course material* and the opinion term *relevant and interesting* are explicitly

present in the text, the aspect category and sentiment polarity are part of predefined sets of aspect categories and sentiments, and therefore implicit.

Moreover, W. Zhang et al. (2023) distinguishes between single ABSA tasks and compound ABSA tasks. Single tasks aim to predict only one of the sentiment elements. The task Aspect Term Extraction (ATE) focuses on extracting the appropriate aspect terms that are explicitly present in the sentence. Aspect Sentiment Classification (ASC), on the other hand, is responsible for determining the sentiment polarities expressed towards the previously extracted aspect terms. Moreover, Aspect Category Detection (ACD) is the task of predicting one or more aspect categories or entities within the sentence. Last but not least, W. Zhang et al. (2023) displays a fourth single task, called Opinion Term Extraction (OTE). This task predicts terms in the sentence that express either a positive or negative sentiment towards an aspect term. In contrast to single tasks, compound tasks aim to predict multiple sentiment elements simultaneously.

It is important to note that within ABSA literature, many different definitions are used. For instance, the terms *opinion target*, *target*, *aspect*, and *entity* are often used interchangeably and refer to the target within the sentence over which the opinion is expressed. In this study, the sentiment elements described in the previous paragraph are used, as they are the most common in research.

Ligthart et al. (2021) uses different perspectives to differentiate sentiment analysis: a task perspective, an approach perspective, and a level of analysis perspective. From the task perspective, the authors divide sentiment analysis into subjectivity classification, sentiment classification, opinion spam detection, implicit language detection, and aspect extraction. From the approach perspective, SA is classified into machine learning, hybrid, and lexicon-based approaches. From the level of analysis perspective, a distinction can be made between aspect-, sentence-, and document-level analysis.

As described earlier, due to the higher relevance of deeper aspect-level analysis, the latter two analysis levels are neglected in this thesis. The underlying tasks within the level of interest, the aspect level, are those mentioned earlier, outlined in detail by W. Zhang et al. (2023), and are addressed in this study as ATE (Aspect Term Extraction), ASC (Aspect Sentiment Classification), and ACD (Aspect Category Detection). Another existing task, known as OTE (Opinion Term Extraction), exists but is not considered further. This is because it is not a core task in ABSA and detects merely the information in the reviews responsible for the sentiment polarity (terms such as *good* or *bad*). Since sentiment polarity itself is sufficient for the use case of this thesis, the opinion term extraction task is neglected.

Lexicon-based approaches are the traditional methods in sentiment analysis (Hemmatian and Sohrabi, 2019). According to B. Liu (2022), the core of these approaches is the focus on opinion terms, also referred to as sentiment words, opinion words, or polar words. The author explains that these are the terms that reflect the sentiment expressed on a given aspect. While positive sentiment terms can be *good*, *fantastic*, *superb*, *great*, *wonderful*, or *nice*, negative sentiment terms include *bad*, *terrible*, *disappointing*, *awful*, or *disgusting*. Conjointly, they can be summarized as a sentiment lexicon, which is a collection of opinion terms. Moreover, B. Liu (2022) divides lexicon-based approaches into dictionary-based and corpus-based approaches. In the dictionary-based approach, freely available dictionaries (e.g., WordNet) are used to generate the sentiment

lexicon. This approach is especially used for general domain use cases. However, sometimes one word implies a different sentiment depending on the domain. Even within one domain, this can be true for different products. For instance, when comparing a regular car to a sports car, the word *loud* can have different sentiments depending on the context. Commonly, good sports cars are associated with being loud, while regular cars are often expected to be quieter due to improved insulating materials. The corpus-based approach addresses this issue. In this approach, the more general sentiment lexicon is expanded with sentiment words from a domain-specific corpus (B. Liu, 2022).

In contrast to the traditional lexicon-based approaches, there are machine learning approaches. L. Zhang et al. (2018) explains that statistical and probabilistic methods of machine learning were primarily used for NLP applications, such as Naïve Bayes, hidden Markov models, and k-nearest neighbors. However, the field of NLP has experienced a breakthrough thanks to deep learning within the last decade. Sentiment analysis tasks have recently proven to be popular for applying deep learning techniques. According to Otter et al., 2020, the transformer architecture has become a paradigmatic element in many artificial neural networks for the application of NLP. Devlin et al. (2018) introduces BERT, a model based on the transformer architecture and used in this thesis. It stands for Bidirectional Encoder Representations from Transformers and represents a Large Language Model (LLM) developed in 2018 by Google. As the name suggests, the model uses only the encoder part of the transformer architecture.

2.2.2 Usage of LLMs for ABSA

This section illustrates how large language models (LLMs) can be applied to address aspect-based sentiment analysis (ABSA) tasks and highlights key challenges, such as the risk of overfitting, that need to be overcome.

Fine-Tuning of Pretrained Large Language Models

The BERT model from Google (Devlin et al., 2018) and the GPT model from OpenAI (Radford et al., 2019) have gained significant attention for their effective performance in natural language processing (NLP) tasks. Pretraining involves feeding the model large amounts of unlabeled data, as outlined by Devlin et al. (2018). This data can span multiple domains and languages. Pretraining is conducted in an unsupervised manner and aims to teach the machine learning (ML) model the context of language. During this process, the model is trained on general-purpose tasks. Afterward, the pretrained language model (PLLM) can be adapted for specific NLP tasks in targeted domains. In contrast, Houlsby et al. (2019) explains that fine-tuning, a crucial transfer mechanism in NLP, typically requires smaller datasets tailored to the task of interest. These specific tasks are often referred to as downstream tasks in the NLP literature (Min et al., 2023). For example, in ABSA, a pretrained BERT model can be fine-tuned for tasks like Aspect Term Extraction or Aspect Sentiment Classification. Min et al. (2023) argues that adapting the model to a specific task often requires adding task-specific architecture on top of the pretrained language model.

When considering fine-tuning, it is important to understand the various strategies involved. Sun et al. (2019) proposes different fine-tuning approaches for BERT, as illustrated in Figure 3.1. The figure shows alternative fine-tuning scenarios, with different colors representing various options.

After BERT is pretrained on a general-domain corpus, additional pretraining on unlabeled target-domain data can improve performance. BERT can then be fine-tuned as described earlier, either for a single task or multiple tasks simultaneously. Furthermore, Sun et al. (2019) found that single-task fine-tuning can benefit from an initial multitask fine-tuning phase.

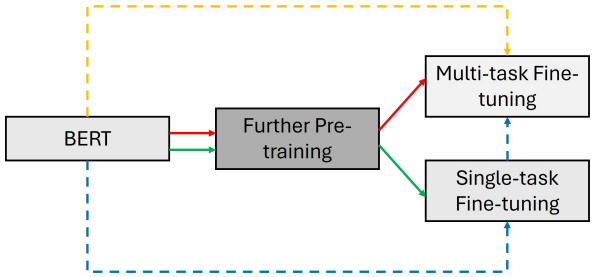


Figure 2.2: Different Strategies for Fine-Tuning BERT. Own illustration, adapted from Sun et al. (2019).

Sun et al. (2019) argues that further pretraining is not mandatory during fine-tuning, but it is a viable option when additional data is available. This further pretraining can be categorized into three types: within-task, in-domain, and cross-domain pretraining. Within-task pretraining involves further training BERT specifically on the dataset of the target task, enhancing its adaptation to that task. In-domain pretraining uses data from the same domain as the target task to extend the model's learning. For example, in sentiment analysis, there are multiple sentiment classification tasks with similar data distributions, and the combination of data from these tasks can be used to further train BERT. Finally, cross-domain pretraining incorporates data from both the target domain and other domains, expanding the model's generalization capabilities (Sun et al., 2019).

Cross-Domain and Cross-Lingual ABSA

W. Zhang et al. (2023) discusses a key challenge in developing ABSA applications: the lack of machine learning (ML) models trained on the specific domain of interest. The authors note that recent ABSA models with good performance typically share one common feature: the training and testing data come from the same distribution. This means that the data used to train the model closely matches the data used for testing and evaluation, with "matching" referring to the same domain or language. W. Zhang et al. (2023) present two approaches to address this challenge: training an ML model from scratch for the target domain or adapting/fine-tuning a pretrained ML model with domain-specific data. The first method, while effective, requires a large amount of domain-specific data, particularly annotated data at the aspect level, which is often scarce and essential for ABSA training. This leads to the need for custom annotation resources and building a tailored training dataset. The second method, which also requires domain-specific data, typically needs fewer examples compared to training from scratch (W. Zhang et al., 2023). Quantifying the exact amount of data needed for ML training is challenging, but according to Goodfellow et al. (2016), tens of thousands to millions of examples may suffice for complex tasks.

To build ABSA systems without the extensive and time-consuming task of collecting data and implementing ML models from the ground up, transferable ABSA presents a viable alternative. In the context of this thesis, two types of transferable ABSA, as outlined by W. Zhang et al. (2023), are particularly relevant: cross-lingual and cross-domain ABSA. For language-related challenges, cross-lingual ABSA may be a suitable approach, especially due to the lack of resources and tools for languages other than English. B. Liu (2022) highlights that much research has focused on

cross-language sentiment classification at the document or sentence level, but there is limited literature at the aspect level. On the other hand, cross-domain ABSA focuses on adapting ML models to unseen domains, such as the higher education domain addressed in this thesis.

W. Zhang et al. (2023) argue that an effective approach for implementing an ABSA system and adapting it to new domains and languages is to use pretrained large language models (PLLMs) and fine-tune them with custom training data, leveraging the recent advancements in PLLMs (W. Zhang et al., 2023).

Overfitting, Data Augmentation, and Class Imbalance

A prominent issue in machine learning (ML) when training data is scarce is overfitting. Shorten et al. (2021) identifies data augmentation as a solution to address this problem, justifying its use in this thesis to extend the annotated training data and prevent overfitting.

Ying (2019) explains that overfitting occurs when a ML model performs well on the training data but poorly on the test data, due to its inability to generalize to unseen data. In natural language processing (NLP), overfitting can manifest as the model memorizing high-frequency numeric patterns in token embeddings or retaining specific language forms that do not generalize well. Ying (2019) outlines several causes of overfitting, including noise, limited training data, and unnecessary classifier complexity. Noise refers to random data points that distort general patterns, while unnecessary complexity occurs when a model includes more predictors than needed. For example, Hawkins (2004) argue that if a regression task only requires two variables to predict an outcome, using more predictors may lead to overfitting. In this thesis, the primary concern is limited training data, as discussed earlier, while noise and unnecessary complexity are not significant issues. Noise is less relevant since the target data consists primarily of subjective student opinions, and the feature set (sentence, aspect, sentiment polarity) is clearly defined, minimizing the risk of excessive model complexity.

Shorten et al. (2021) emphasizes that data augmentation is particularly useful when training data is limited or annotation resources are scarce. It is commonly employed to prevent overfitting. They further explain that other regularization methods, such as dropout and weight penalties, also aim to reduce overfitting. Dropout introduces noise to intermediate layers of the model, while weight penalties add costs to the loss function based on the magnitude of model weights, promoting simpler, more generalizable models. However, the authors state that these techniques do not fully capture semantic invariance. Data augmentation, on the other hand, creates synthetic data by applying minor transformations to existing data, preserving the original meaning. Neural augmentation methods, such as back-translation, generative data augmentation, and style augmentation, are often used for this purpose (Shorten et al., 2021).

Back-translation, a popular neural network technique, translates a sequence of words into another language and then back to the original language, as explained by Sennrich et al. (2015). In this thesis, generative data augmentation was employed, using pretrained large language models (LLMs) based on the transformer architecture, as discussed by Y. Yang et al. (2020). While back-translation could have been used, it typically incurs service fees, making generative data augmentation the more practical choice.

Besides overfitting, Japkowicz and Stephen (2002) outline class imbalance as another challenge in machine learning. It refers to the *"problem encountered by inductive learning systems on domains for which one class is represented by a large number of examples while the other is represented by only a few"* (Japkowicz and Stephen, 2002, p. 429). This imbalance can severely impact model performance, especially in supervised learning settings, where one label class occurs much more frequently than others. This is common in real-world applications, where data is often skewed. For example, in a binary classification task, class imbalance arises when there are many more positive samples than negative ones.

In sentiment analysis, the common classes are positive, neutral, and negative. While misclassification of the minority class is particularly problematic in high-stakes fields like medical diagnosis, in sentiment analysis, the consequences are typically less severe. However, misclassifying negative feedback in the context of higher education could be more problematic than misclassifying positive feedback, as failing to address issues raised by students can have more harmful effects than overlooking praise.

Krawczyk (2016) identifies three main strategies for dealing with class imbalance in ML: data-level techniques, algorithm-level techniques, and hybrid approaches. Data-level techniques aim to balance the class distribution by generating more samples of the minority class (oversampling) or removing samples from the majority class (undersampling). Early methods involved random sampling, which often led to irrelevant data being added or meaningful data being removed. More recent techniques have been developed to better preserve the context of the text. Algorithm-level methods adjust the learning algorithm itself to address the imbalance, while hybrid approaches combine both data-level and algorithm-level techniques. An example of a hybrid approach is found in Wang et al. (2012), where they combine sampling methods with cost-sensitive learning.

Tanaka and Aranha (2019) notes that data augmentation can also help alleviate class imbalance by oversampling the minority class to create a more balanced label distribution. Therefore, data augmentation can address both overfitting and class imbalance.

In this thesis, due to the limited amount of training data and the highly imbalanced label distribution, data augmentation techniques were employed to mitigate overfitting and improve generalization.

2.3 Related Work

Section 2.3.1 explores ABSA research within the domain of higher education, with a particular focus on student feedback regarding their university study experiences. This section is divided into two parts: a technical view, which examines the methods employed in ABSA research, and a business view, which considers the intended outcomes of ABSA applications in a business context. Finally, Section 2.3.2 reviews related works that integrate sentiment analysis results into OLAP systems.

2.3.1 ABSA of University Reviews

The following section reviews studies that apply ABSA to university reviews, with a focus on the technical perspective.

Technical Perspective

According to Pan and Yang, 2009, most machine learning algorithms assume that the training data and unseen target data come from the same domain with a similar data distribution. However, this assumption may not always hold in practice, as ML models may be trained on data from a different domain or language than the one of interest. In such cases, knowledge transfer can be leveraged to improve model performance. Ramponi and Plank (2020) argue that the performance of a model often suffers when applied to a new, unseen domain because the aspects in the trained domain may differ significantly from those in the target domain. For example, a large language model (LLM) trained on financial data might be familiar with aspects like *revenue, expenses, investments*, whereas a model applied to the university domain might encounter terms such as *lectures, books, professors, study contents*. As a result, the model trained on financial data would likely perform worse on university-related data.

Research on ABSA in higher education is limited (Melba Rosalind and Suguna, 2022). Nevertheless, the remainder of this section provides an overview of recent studies focused on ABSA for educational domain texts.

Nikolić et al. (2020) developed an ABSA system for the Serbian language using a hybrid approach that combined dictionary-based methods with machine learning, citing insufficient annotated training data for fully relying on deep learning models. They collected student surveys from a single university in Serbia as well as publicly available student reviews from the 'Rate my Professors' website. The reviews were split into sentences and sentence segments, with aspect categories defined under three main categories: professors, courses, and other. Sub-categories included lectures and helpfulness under professors, and materials and organization under courses. Annotators were asked to identify both an aspect category and a sentiment for each sentence segment.

Melba Rosalind and Suguna (2022) built an ABSA system for online student reviews in English. They first preprocessed the reviews by removing irrelevant information such as stopwords, then split the reviews into sentences. Using the Latent Dirichlet Allocation (LDA) algorithm for topic modeling, they identified aspect categories within the sentences. Lastly, they applied a machine learning algorithm to determine the sentiment associated with each aspect category. Figure 2.3 illustrates their proposed ABSA system.

Edalati et al. (2022) conducted ABSA using data collected from the Coursera platform, which consisted of student feedback rating their experiences with online courses. The authors employed several algorithms, including Random Forest, Support Vector Machine, and Decision Trees. They also used deep learning models and extracted teaching-related aspects in combination with predicted student opinions on those aspects.

Similarly, Kastrati et al. (2020) applied ABSA to Coursera reviews, proposing a framework for the automatic extraction of student opinions related to specific aspects. They annotated domain-related aspects in a weakly supervised manner, allowing them to avoid the labor-intensive process of manual annotation and to identify aspect categories reflected in the unlabeled reviews. The researchers used convolutional neural networks (CNNs) to identify aspects and classify sentiment polarities.

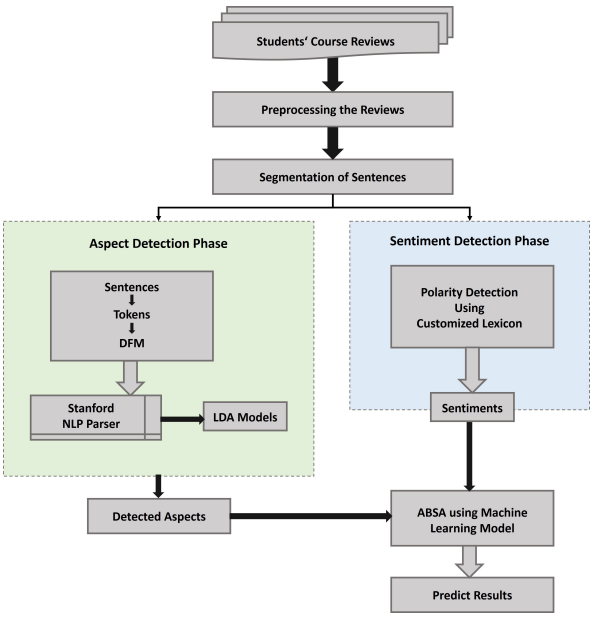


Figure 2.3: Concept of a proposed ABSA system. Own illustration, adapted from Melba Rosalind and Suguna (2022).

Dehbozorgi and Mohandoss (2021) analyzed students' speech and performance by using the Python package Text2Emotion to classify emotions such as anger, happiness, surprise, fear, and sadness. They applied rule-based POS tagging to extract aspects and used the k-nearest neighbors algorithm to predict students' performance, linking aspects to their emotions.

Schurig et al. (2022) applied ABSA to feedback from surveys that rated online teaching during the COVID-19 pandemic. As many reviews were written in a mix of English and Danish, they used the Google Cloud API to translate everything into English. Additionally, they employed dependency parsing rules to extract aspects, specifically the aspect terms present in the sentences. Sentiments were predicted using the libraries TextBlob, NLTK VADER, and Flair, with Flair leveraging a pretrained transformers model to predict positive and negative sentiment polarities. Finally, the authors manually defined aspect categories and evaluated the aspect terms detected earlier.

Bhowmik et al. (2023) collected student comments that had already been labeled with teaching aspects. After preprocessing the data to remove irrelevant characters, they developed a deep learning model based on a bidirectional LSTM network to predict the sentiments associated with the identified aspects.

Sindhu et al. (2019) collected student reviews from a single university, defining 12 aspect categories through interviews with domain experts. After preprocessing and splitting the comments, they manually annotated around 5,000 sentences with an aspect category and sentiment. Using word embeddings with the word2vec framework and a two-layer LSTM, they performed the ABSA modeling procedure.

The following section outlines studies that apply ABSA to university reviews from a business perspective.

Business Perspective

As a reminder, the objective of this thesis is to design and implement a data warehouse system that supports higher education institutions (HEIs) in the systematic analysis of sentiments expressed in students' ratings. The goal is to ultimately gain insights into why students have positive or negative feelings about studying at a particular university. With this in mind, it is essential to consider the literature that addresses the business requirements of an ABSA system, rather than focusing solely on the technical details, as discussed in the previous section. Therefore, this section highlights how ABSA can serve as a valuable decision-making tool for higher education institutions.

Dolianiti et al. (2019) identified five distinct use cases or task types for sentiment analysis within the educational domain:

1. Instruction evaluation
2. Institutional decision/policymaking
3. Intelligent information/learning systems enhancement
4. Assignment evaluation and feedback improvement
5. New research insights

Instruction evaluation has proven to be a popular data source for applying sentiment analysis. Many studies have focused on analyzing student feedback from open-ended questions in course evaluations (Koufakou et al., 2016; Dhanalakshmi et al., 2016; Gottipati et al., 2017). Some studies have used publicly available reviews from websites like "Rate My Professors" or Massive Open Online Course (MOOC) platforms such as Coursera (Azab et al., 2016; Onan, 2021; Dalal et al., 2014; Z. Liu et al., 2016). These course evaluations aim to improve courses in the upcoming semesters. However, other studies have sought to analyze student feedback more quickly in order to enhance course quality within the current semester. For example, Colace et al. (2014) applied sentiment analysis to student discussions on the Moodle forum for a given course. They found that students' sentiments improved during the course, as the teacher adapted their teaching based on the feedback. Similarly, Altrabsheh et al. (2013) developed a sentiment analysis system for real-time feedback during lectures. Students provided feedback via Twitter, enabling instructors to quickly assess whether the pace of the lecture was suitable or if students needed assistance. Onan (2021) analyzed 66,000 MOOC reviews, comparing machine learning, ensemble learning, and deep learning approaches. They found that Long Short-Term Memory (LSTM) networks combined with GloVe word embeddings delivered the best results. Pramod et al. (2022) proposed a faculty evaluation system using sentiment analysis. They conducted a structured questionnaire to collect student feedback on faculty performance and analyzed qualitative responses to detect sentiment polarities, which varied across faculty and over time. These results were combined with quantitative data to predict faculty popularity. Similarly, Roaring et al. (2022) evaluated faculty performance at a university by comparing numerical ratings with textual comments. They found that low ratings (1) correlated with negative sentiment, neutral ratings (2) were linked to neutral words, and higher ratings (3-5) were associated with positive sentiment.

Since the task type *institutional decision/policymaking* best aligns with the data and objectives of this thesis, it will be discussed in more detail at the end of this section.

Intelligent information/learning systems enhancement is defined by Dolianiti et al., 2019 as the development of learning systems focused on helping individual students rather than analyzing general opinions from the student body. Sentiment analysis can aid in understanding individual student characteristics, thereby creating a more personalized learning experience. For example, the study by Scaffidi (2016) analyzed forum messages to determine whether replies provided solutions to user-specific problems. Sentiment analysis was applied to the responses to assess whether the subsequent comments addressed the issues raised in the initial message.

Assignment evaluation and feedback improvement is another task type (Dolianiti et al., 2019). According to Dolianiti et al. (2019), sentiment analysis (SA) can be used to help automate the evaluation of assignments written in the form of essays by students. Furthermore, feedback from teachers can be augmented with additional information based on the sentiments expressed.

As the task type *new research insights*, Dolianiti et al. (2019) summarizes all miscellaneous use cases, such as detecting the relationship between students' sentiments and their performance.

The ratings used in this study reflect opinions on not only courses or professors, but also on several other aspects, such as equipment, library, study content, and organization. In literature, however, most works analyze comments that evaluate only one specific course (Dolianiti et al., 2019). Yet, the use of sentiment analysis for *institutional decision/policymaking* must not be neglected, as it can help increase the university's attractiveness and facilitate the process of student recruitment. The analysis of broader aspects, beyond those found in course evaluations, can be beneficial for this purpose. This area is related to the broader field of educational data mining, which, according to Shaik et al. (2023), helps educational institutions in both the student recruitment process and improving policies concerning student retention. The remainder of this section discusses recent studies related to this task type.

With the research goal of assisting policymakers in higher education institutions, Hussain et al. (2022) developed a three-layer ABSA system using topic modeling and hybrid machine learning. The first layer was used to extract general high-level aspects from the data using the Latent Dirichlet Allocation (LDA) algorithm as the topic modeling technique. Using similar techniques, the second layer then extracted low-level aspects associated with the high-level aspects. As a reminder, in this thesis, high-level aspects are referred to as aspect terms present in the sentence, whereas low-level aspects are the aspect categories. Figure 2.4 illustrates their ABSA workflow.

The authors then detected sentiment polarities associated with the aspects in a third layer using various machine learning methods. They argue that their results may enhance decision-making processes at higher education institutions.

In another study, social media comments from Twitter were collected to rate selected German universities. In the first step, the comments were classified as either positive or negative. In the second step, they were analyzed in greater depth with regard to underlying topics. The authors justified their approach by arguing that the insights provided could complement university rankings, which, according to Abdelrazeq et al. (2016), do not always prove to be reliable when measuring critical indicators.

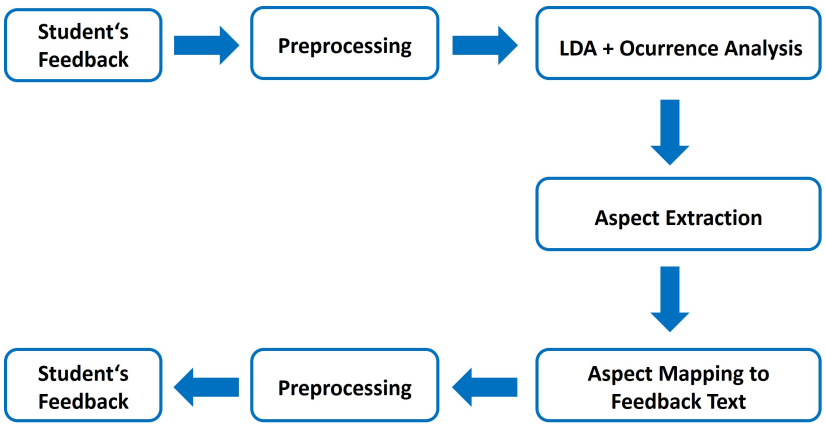


Figure 2.4: ABSA Workflow. Own illustration, adapted from Hussain et al. (2022).

Cirqueira et al. (2017) sought to improve relationship management processes at a Brazilian university by applying topic modeling and sentiment analysis to data from the university's Facebook page. They classified the overall sentiment polarities (positive, negative, neutral) expressed not only by students, but also by staff. They then detected the main aspects addressed using Latent Dirichlet Allocation (LDA) as a topic modeling technique. The findings provided valuable insights that helped university administrators enhance communication, address problem areas, and improve relationships with stakeholders. In a similar study, Santos et al. (2018) analyzed online reviews from international students rating their host universities to identify key drivers of university attractiveness. By using LDA for topic modeling and sentiment analysis, they uncovered hidden aspects and how these were rated positively or negatively. These insights could assist university administrators in adjusting their communication strategies to promote positively perceived aspects or address problematic areas.

A slightly different approach was pursued by Jena (2019), who analyzed data from Twitter, Facebook, and Moodle. In the first step, they used machine learning methods to detect sentiment polarities (positive, negative, neutral) from students. In the second step, employing big data frameworks, they predicted student emotions such as Amusement, Anxiety, Boredom, Confusion, Enthusiasm, Excitement, and Frustration. The results of this study could help universities take student feedback into account when planning specific activities.

2.3.2 Social Business Intelligence

In a study on Social Business Intelligence (SBI), which combines user-generated content (UGC) with corporate data, Golfarelli, 2014 addresses OLAP analysis of textual UGC and highlights its deviations from traditional OLAP systems within the context of business intelligence (BI) applications. The author argues that, unlike standard BI projects, OLAP systems for UGC analysis require additional modules, such as the semantic enrichment of unstructured data.

At a high level, the study proposes an architecture consisting of three components: the crawling component, the semantic enrichment component, and the ETL component. The crawling component extracts UGC from the web, transforms it into a structured format, and loads it into the data staging area, also referred to as the operational data store (ODS). Optionally, the ODS can be linked to a document database. The semantic enrichment component extracts semantic information from the data present in the ODS. Finally, the ETL component is responsible for

regularly extracting the data from the ODS and integrating it with corporate data sourced from an Enterprise Data Warehouse (EDW). Afterward, the data is loaded into multidimensional cubes Golfarelli, 2014.

In related research, Gallinucci et al., 2015 proposes the meta-star approach for modeling flexible topic hierarchies for UGC. Since topics found in social media data are dynamic and change over time, the authors argue that a traditional static star schema is insufficient. In this thesis, a topic modeling approach was considered but ultimately discarded. This is because the topics identified were equivalent to those already presented as aspect categories on the evaluation platform. Consequently, the topic hierarchy modeled in this thesis is static and low in complexity, making the traditional star schema sufficient.

In comparison to these studies (Golfarelli, 2014; Gallinucci et al., 2015), in this thesis, the crawling component is equivalent to the collection of online reviews from a university evaluation platform. However, no document database is used; instead, the reviews are stored locally as Excel files. The semantic enrichment component reflects the application of the ABSA model, which detects aspects and corresponding student sentiments. The ETL component is only partially addressed in this thesis, as there is no underlying business data warehouse, and therefore no corporate data is integrated. However, the prediction results of the ABSA are stored in a multidimensional cube.

Another study conducted by Cuzzocrea et al., 2016 analyzes Twitter tweets by combining Formal Concept Theory (FCT) with OLAP techniques. They collect the data using Twitter's public APIs and detect topics using the FCT approach. The data is then modeled in a multidimensional format and analyzed using standard OLAP operations. In similar research, Kraiem et al., 2015 aim to collect and analyze social media data in an OLAP fashion, focusing on Twitter tweets. Besides the standard ETL operations, they design a generic multidimensional schema based on the data structure of the tweets.

With the objective of designing a data warehouse for UGC, Moalla et al., 2022 extract data from various social media accounts (Facebook, Twitter, and YouTube) and model three data marts, one for each account. To build the complete data warehouse, they combine the different schemas from the data marts into a single schema. They then use machine learning techniques, such as Support Vector Machines, for sentiment analysis before loading the data into a NoSQL database. For data collection, public APIs are utilized. In contrast, this thesis focuses on a single data source and, since no API is available, uses web scraping techniques. In addition, a relational database was used in this thesis to store the obtained multidimensional data.

3

System Design

This chapter serves as an introduction to the main part of the thesis. Its purpose is to highlight the clear distinction between the ABSA model training process and its application. On the one hand, it provides an overview of the fine-tuning process undertaken in this thesis, which is detailed in Chapters 4–5. On the other hand, it outlines the architecture of the data warehouse system designed in this thesis, with its implementation covered in Chapter 6.

3.1 Training Process

The training process in this thesis comprises two distinct components: the training of an ABSA model (Fig. 3.1) and the training of an aspect category classifier (Fig. 3.2). The ABSA model is responsible for extracting one or more aspect terms implicitly present in a sentence (e.g., *classes*) and determining their associated sentiment polarities (*positive*, *negative*, or *neutral*).

The aspect classifier, on the other hand, is tasked with categorizing the aspect term(s) into an aspect category from a predefined set (e.g., *lectures*). As discussed later, the ABSA training implementation from the Python framework *Python*, used in this thesis, initially lacked support for classifying an aspect term into an explicit aspect category that is not directly mentioned in the sentence. As a result, a separate classifier was developed to address this limitation, leading to two distinct training processes.

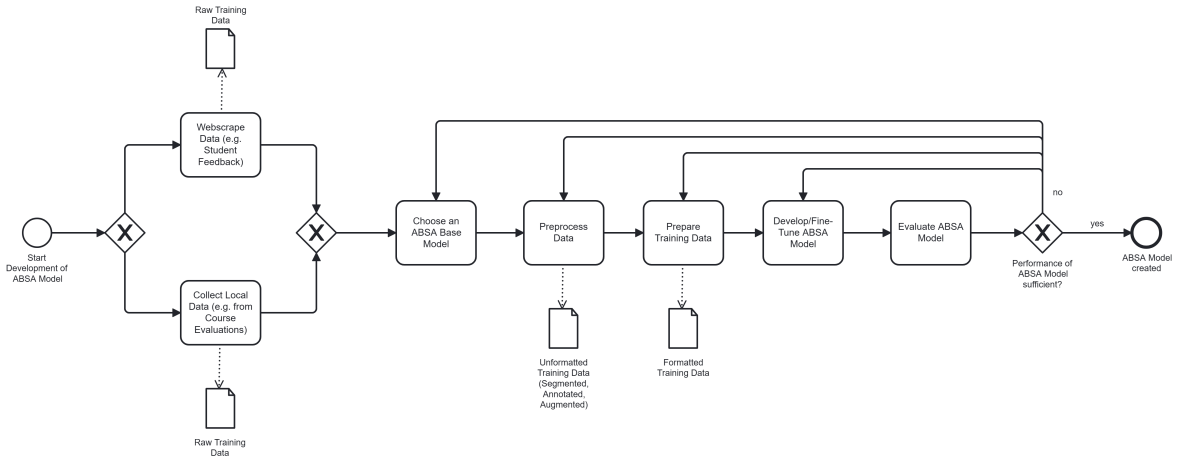


Figure 3.1: Fine-tuning the ABSA Model

The process of fine-tuning the ABSA model is illustrated in Figure 3.1. The process begins with the collection of suitable data for training or fine-tuning the model. In this case, since no pre-existing training data was available, student feedback was crawled from a university evaluation platform using web scraping techniques. This feedback serves two purposes: as training data for the ABSA model and later as input for populating the sentiment cube. However, alternative data sources could also be considered for this step, such as course evaluations conducted each semester by the university.

The next step involves selecting a language model to serve as the base model for ABSA training, as developing an ABSA model from scratch was deemed unfeasible. Once a base model is chosen, the collected data must be preprocessed. This step includes several data transformation procedures to ensure that the data is ready for training. In this thesis, preprocessing involved data segmentation, data annotation, and data augmentation. Data segmentation refers to splitting each student review comment into individual sentences, as the ABSA model requires one sentence per input, and each record in the training dataset corresponds to a single sentence. Data annotation involves manually labeling each sentence with one or more aspect terms and their corresponding sentiment polarities, as no labeled training data was initially available. Data augmentation aims to artificially extend the annotated dataset to address issues such as class imbalance and overfitting.

In the subsequent step, the prepared dataset must be adjusted to meet the format requirements of *pyabsa*'s training implementation. This involves transforming the data into a specific format compatible with the Python library. Depending on the training framework used, this step may not always be necessary.

The next stage is the modeling or training phase, which involves fine-tuning the base model using the prepared training data. This stage leverages the completed dataset as input to the base model to develop the ABSA model.

Finally, the model is evaluated. If the developed model does not meet performance expectations, the process can be revisited from earlier steps to identify and address any issues.

The process of developing the aspect category classifier is illustrated in Figure 3.2. In this thesis, a predefined set of aspect categories was selected, which later represent the final entity of the aspect dimension within the sentiment cube. To achieve this, a custom training dataset was created by labeling each previously annotated sentence with the appropriate aspect category corresponding to each aspect term.

Next, the training data is prepared by replacing each aspect category in the dataset with its one-hot encoding representation. The prepared training data is then fed into a base model suitable for multi-label text classification. In this thesis, an implementation of the BERT model was chosen; however, other language models could also be used for the aspect category classifier. Finally, the classifier is evaluated in the testing phase.

As mentioned earlier, both processes are detailed in Chapters 4–5. Chapter 4 focuses on the collection of student feedback as raw training data, while Chapter 5 discusses the training of the ABSA model and the development of the aspect category classifier.

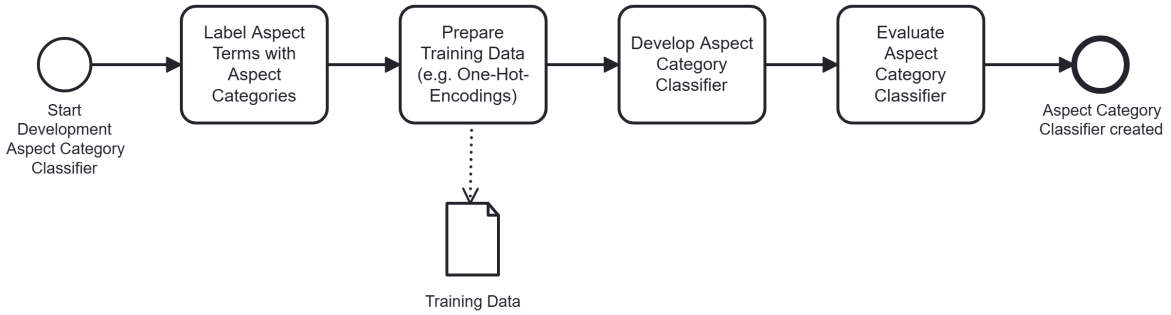


Figure 3.2: Developing the Aspect Category Classifier

3.2 Data Warehouse System Architecture

The architecture of the data warehouse system designed in this thesis is shown in Figure 3.3 and is inspired by the reference architecture from Vaisman and Zimányi (2014). The foundation of the system is the data source, which represents a real-world example of a university evaluation platform.

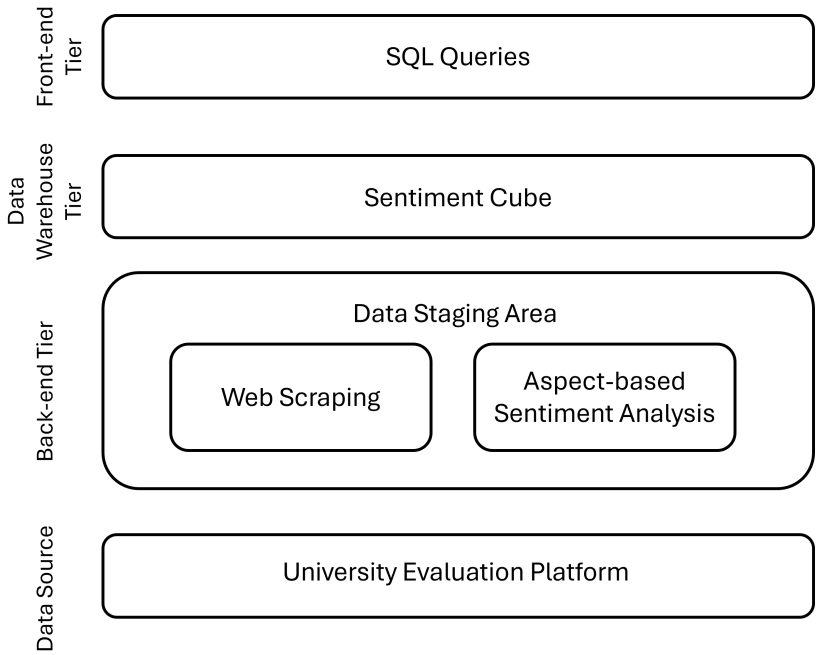


Figure 3.3: Data Warehouse System Architecture

The next layer is the back-end layer, which serves as the data staging area and is designed as a two-module component: a web scraping module and an aspect-based sentiment analysis module. This layer is responsible for the ETL process: data extraction, transformation, and loading. First, student feedback to be analyzed is extracted from the data source using the same web scraping function employed for collecting the training data during the ABSA model training. In the next step, the data is enriched with aspect terms, sentiment polarities, and aspect categories by applying the developed ABSA model and aspect category classifier. Finally, the results are loaded into a data staging area, represented in this thesis by a relational staging database.

The subsequent layer is the data warehouse tier, which corresponds to the data warehouse itself—specifically, the sentiment cube. Its multidimensional structure is designed to support

querying and is populated with data from the staging area. The final layer is the front-end tier, implemented in a rudimentary form in this thesis. Here, it is represented by SQL queries provided as examples for analyzing the sentiment cube.

As mentioned earlier, the implementation of the data warehouse system is discussed in Chapter 6. This includes the definition of the sentiment cube, the data integration processes using web scraping and the developed models, and analysis examples presented as SQL queries.

4

Web Scraping of Student Feedback

This chapter addresses the collection of student feedback as raw training data for the fine-tuning process of the ABSA model. It focuses on collecting the data needed for two purposes: creating a training dataset, as discussed in Chapter 5, and collecting the raw data to be analyzed.

To understand the required data, Section 4.1 provides a brief overview of the business perspective of the use case. It emphasizes the importance of analyzing students' sentiments in detail, as previously discussed, and outlines the business requirements.

Section 4.2 describes the characteristics of typical reviews on university evaluation platforms. Section 4.3 details the web scraping process used to collect 10,810 student ratings from a real-world university evaluation platform.

4.1 Business Requirements

Information found in university rankings alone does not suffice to capture students' satisfaction (Johnes, 2018; Moustafa, 2024). Hence, it is assumed that analyzing student feedback is necessary. Furthermore, based on the findings of Siegel and Alexa (2020), analysis at the document or sentence level may not be sufficient to draw meaningful conclusions about how students perceive various aspects of specific university offerings. Therefore, the general assumption is that HEIs need to employ ABSA to gain insights into their students' satisfaction with different services provided.

Additionally, it is assumed that understanding the satisfaction levels of not only an institution's own students but also those of other HEIs is of interest, as it enables meaningful comparisons.

The findings from research and the assumptions drawn were confirmed through a discussion with a marketing professor at Johannes Kepler University (JKU) Linz. For JKU Linz, it was particularly important to uncover insights from their students' sentiments, hidden in various textual data sources. Currently, JKU does not have a system capable of fully leveraging the potential of textual feedback provided by their students. Moreover, apart from the annual lecture evaluations, no existing data is readily available for this use case. Consequently, one requirement for a potential system would be the ability to collect reviews written by students before analyzing hidden sentiments. Additionally, there was a need to compare the sentiments expressed by JKU students with those of students enrolled at other Austrian HEIs. This would enable valuable conclusions to be drawn from both internal and external perspectives. The system would need to make it possible to compare student satisfaction from different perspectives, such as time and

study course. For instance, it should allow for comparisons of satisfaction with lectures in study course A at university A to satisfaction with lectures in study course A at university B, both during the first half of 2023.

Following the discussion, it was assumed that other HEIs face similar requirements to those of JKU Linz. Furthermore, it was assumed that, for this use case, collecting reviews from Austrian HEIs would suffice, eliminating the need to gather data from HEIs in other German-speaking countries. Therefore, the system needed to be capable of crawling reviews from students enrolled at Austrian HEIs. Collecting reviews on HEIs from different countries, however, could also be considered.

One of the goals of the analysis is to identify which aspects of studying at an Austrian HEI are perceived positively by students and which aspects are subject to complaints. With equivalent insights regarding other Austrian HEIs, it may become possible to compare the services offered by one institution with those of its competitors.

As a result, the system's requirements include collecting textual student feedback on various Austrian HEIs from the web and analyzing the hidden sentiments embedded within this feedback, with a focus on different HEI offerings as well as other factors, such as time and study course. This enables comparisons of university services provided by different Austrian HEIs. It was assumed that OLAP techniques might be suitable for this analytical purpose. Consequently, OLAP was utilized in this thesis.

Ultimately, the insights revealed by the system could assist HEIs in highlighting their strengths, addressing weaknesses, improving student retention, and, most importantly, enhancing student recruitment.

4.2 Data Selection

The data source for populating the data warehouse system must consist of student feedback, as this is the data to be analyzed later. While student feedback can be found across various platforms, such as Facebook, blogs, or rating sites, university evaluation platforms are likely the most reliable source for gathering students' opinions on specific services offered at higher education institutions (HEIs).

Figure 4.1 presents a fictional example of a German-language review from a generic university evaluation platform. This example is used in this thesis to illustrate the development of a web scraping component for collecting student feedback, which serves as training data for the ABSA model training pipeline (see Figure 3.1).

The example review provides student feedback on experiences with the Bachelor's program in Biological Chemistry at a specific HEI. As illustrated, a typical review includes a title (Unorganisiert), a course of study (Biological Chemistry (B.Sc.)), a comment (upper text), an overall star rating (3.6), and a publication date (07.10.2023). Occasionally, reviews may also include feedback on the university's digitalization efforts (lower text) and whether the author recommends the program or university (smiley icon at the top left).

The data collected in this thesis is sourced from a prominent university evaluation platform (UEP) based in Germany, which served as the basis for the example review in Figure 4.1. To the best

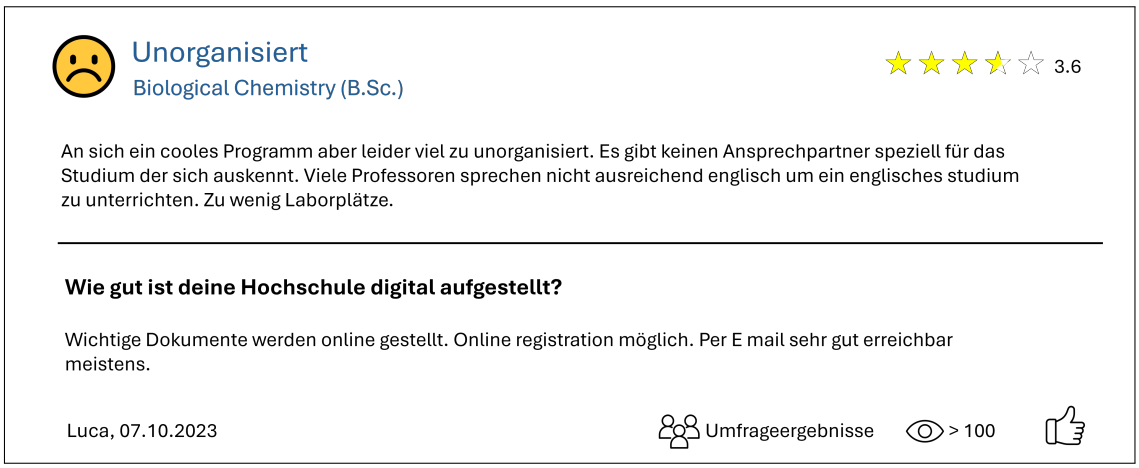


Figure 4.1: Example of a Review from a Generic UEP (Own Illustration)

of my knowledge, no other publicly available data source is better suited for the use case of this thesis. Other potential data sources for students' opinions may include social media channels such as Facebook, Twitter, or Instagram. However, the platform used in this study provides by far the most complete, specialized, and comprehensive source of ratings for Austrian higher education institutions (HEIs). It contains ratings for all HEIs based in both Austria and Germany and allows current students to rate their study experience.

A total of 74 Austrian-based HEIs were considered, resulting in the collection of 10,810 reviews. The data is briefly described below.

Each review consists of two text fields. In the upper field, students can freely compose a general comment, sharing their overall study experience at the respective HEI. In the optional lower field, students are prompted to respond to the question "Wie gut ist deine Hochschule digital aufgestellt?" (ENG: "How well is your university set up digitally?"). Additionally, each review must include a title and specify the study degree pursued by the student. A smiley icon located in the top left corner indicates whether the reviewer recommends the study degree or the respective HEI. The mandatory star rating in the top right corner reflects an overall rating, ranging from 1 to 5. Each review also includes the reviewer's first name and the date the rating was created. Apart from the first name, no personal data is captured by the platform. Users also have the option to write anonymously. Since this study does not aim to collect personal data, this is not a concern.

The majority of the reviews are written in German, with only a few exceptional cases in English. The English reviews were also collected, as it was assumed that including English data would not pose a problem, given that the pretrained language model BERT was trained on English corpora.

Regarding the selection of data features, two options were considered. The first option involved initially web scraping all the data features required to build the sentiment cube, while the second option involved collecting only those necessary for the training procedure and scraping the rest after the model training. It was decided to proceed with the first option, so all attributes of the data required for the sentiment cube were web scraped, as detailed in Section 4.3.

The collected features comprise nine attributes: university name, university type, review title, study degree, overall rating, date, comment, COVID comment, and recommendation. The COVID comment refers to the second text field describing the digital setup of the higher education

institution, often addressing the period during the pandemic.

These fields were considered relevant for subsequent querying of the analysis results. Among them, university name, university type, study degree, and date were identified as essential features for OLAP querying. The inclusion of the date is particularly crucial for understanding how student sentiments evolve over time.

The fields for overall rating and recommendation were considered supplementary. While the review title often contains sentiment-related information, it was excluded from the analysis, as it typically only summarizes the rating in a few words, which duplicates sentiment information already present in the text fields.

Finally, the comment and COVID comment fields are necessary for training the ABSA model, as they contain the relevant information about aspects and sentiments.

The subsequent section explains how the reviews can be collected from the university evaluation platform (UEP) using web scraping techniques.

4.3 Web Scraping Process

The student ratings on the used platform are public, hence, everyone can read them. For this purpose, one does not even need to be registered on the platform. However, crawling the data was necessary, as there was no option to download the reviews. To this end, web scraping was applied, as it serves as an efficient technique for automatically crawling data from websites (Khder, 2021).

As a web scraping tool, Beautiful Soup[1] was chosen. It is a Python framework designed to extract data from HTML and XML files. It works with various parsers to provide intuitive methods for navigating, searching, and modifying the parse tree.

The following Subsection 4.3.1 presents example code that can be used for web scraping reviews from UEPs. The code is described in detail.

It is worth noting that the code is provided in a generic format (see the uppercase HTML elements/tags and class attribute values). Therefore, it must be adapted accordingly. For instance, the generic HTML element "CONTAINER_ELEMENT" may be replaced with "div," whereas the generic class attribute "REVIEW" may be replaced with the respective class attribute value representing the review (refer to line 37 in Listing 4.2). The values for both the HTML element and the class attribute may vary between different UEPs.

After describing the generic code, Subsection 4.3.2 concludes the chapter by presenting the outcomes of the data collection process, obtained by applying the code to a real-world UEP.

4.3.1 Web Scraping Implementation

Listing 4.1: Libraries needed for Web Scraping

```
1 from bs4 import BeautifulSoup
2 import requests
3 import pandas as pd
```

[1] https://beautiful-soup-4.readthedocs.io/en/latest/

```
import numpy as np
from time import sleep
from random import randint
```

The scraping process was carried out using Python and Jupyter Notebook. In Listing 4.1, typical libraries used for the web scraping process are depicted. Beautiful Soup facilitates web crawling, the requests library can be utilized for making HTTP requests to the platform's web server, pandas handles data manipulation tasks, and numpy supports basic numerical computations.

In total, three functions were implemented for web scraping the data: one main function containing the core logic and two helper functions utilized within the main function.

The main function get_uni_reviews ultimately crawls all reviews from one HEI at a time. It is divided into three parts, as illustrated in the following three listings. Listing 4.2 depicts the first part. This function expects three parameters: the URL leading to the ratings of one HEI, the name of the university, and the type of the university. Since both the name and type of the university were not intuitively found on the same URL containing the ratings, it was decided to manually pass them to each scraping iteration. Alternatively, they could have been extracted through scraping if desired. As illustrated by the example review in Figure 4.1, the features to be collected from each review via web scraping include the review title, study course/degree, overall rating, recommendation, date, comment, and COVID-related comment. This data is gathered by accessing their corresponding HTML elements and class attributes. The web scraping process is described in more detail below.

Listing 4.2: Main function get_uni_reviews (1/3)

```
def get_uni_reviews(url, uni, uni_type):
    # creating a dictionary that will be passed to modify headers in
       http request
    headers = {
        'User-Agent': YOUR_USER_AGENT
    }

    global review_titles
    global study_courses
    global rating_values
    global recommendation
    global comment_dates
    global all_comments
    global covid_comment

    review_titles = []
    study_courses = []
    rating_values = []
    recommendation = []
    comment_dates = []
    all_comments = []
    covid_comment = []

    pages = np.arange(1,1000,1)
```

```
24
25      try:
26          for page in pages:
27              page = requests.get(url+"seite-"+str(page), headers=headers)
28              soup = BeautifulSoup(page.text, 'html.parser')
29              soup.prettify()
30
31              if next(page) is False:
32                  print("No more pages left.")
33                  break
34
35              print(page.status_code) #should be 200 (status -> OK)
36              print(page.raise_for_status()) #should be None (no error
                    occurred)
37              reviews_from_one_uni = soup.find_all("CONTAINER_ELEMENT",
                    class_="REVIEW")
38              sleep(randint(1,3))
```

In the beginning of the function, the computer's user agent is passed to the variable *headers* which is later needed by the Beautiful Soup object.

In lines 7-13, all data features, except university name and university type, are defined as global variable. In lines 15-21, they are assigned an empty array, which later would be filled with data from the student reviews. In line 26, a for-loop is executed which iterates through each page containing reviews, starting with the URL passed to the function. Throughout the library `requests` an HTTP request is sent to the web server, accessing the URL. A key point here is the construction of the substring `"seite-"+str(page)` in line 27, which facilitates the transition to the next URL hosting the subsequent ratings. The content of each page is captured in the variable "soup", denoting a Beautiful Soup (BS) object. Another crucial aspect is depicted in line 37. The `find_all` function from Beautiful Soup retrieves all HTML elements, in this example tagged as "CONTAINER_ELEMENT" with the class attribute "REVIEW", resulting in the assembly of all reviews as containers, displayed in the current page. Line 31 calls the first helper function `next`, which is described in more detail later. The for-loop defined in line 26 terminates when the value of `next(page)` is false (line 31), indicating that there are no more pages left to crawl reviews from. Finally, a noteworthy point in Listing 4.2 is in line 38. The function `sleep` is used to make the program wait briefly before continuing with the subsequent code. This turned out to be helpful because, in the second part of the main function, another HTTP request is made. Without this waiting time, issues such as server overload or request throttling occurred.

Listing 4.3 depicts the second part of the main function. The core component here is the nested for-loop, which iterates through all reviews from the current page, which were collected in the first part of the function, as shown in Listing 4.2. This ensures that the necessary data features within each individual review are processed. By applying the BS function `find` to each review, it was possible to extract all elements of the review by accessing their corresponding HTML elements and classes. In this way, all data features could be populated. For example, if a review included the tag "ICON_ELEMENT" with the class attribute "SMILEY_POSITIVE" this indicated that the HEI was recommended (see line 15). Consequently, the recommendation array was updated with

the value "Ja" or "Nein," depending on whether the class attribute was "SMILEY_POSITIVE" or "SMILEY_NEGATIVE" respectively.

For obtaining the review title, study degree, rating total, date, comment, and COVID comment, the function `get_text(strip=True)` was applied. For the comment, it was necessary to distinguish between a long and short comment. If a student wrote a comment exceeding a certain number of characters, the full comment had to be opened by pressing a "Read More" button. This distinction is realized by calling the second helper function `getnexturl` (line 26), which is described in more detail later. The COVID comment is optional for the student to fill out. Therefore, in case of absence, the value "n.a." was added to the respective array. At the end of the second part of the `get_uni_reviews` function, the outer for-loop terminates, indicating that all pages containing the ratings of the respective university have been iterated through. Consequently, all required data is saved in the previously defined arrays.

Listing 4.3: Main function get_uni_reviews (2/3)

```
1         for review in reviews_from_one_uni:
2             # Review titles - variable review_titles
3             title = review.find("TITLE_ELEMENT",
                  class_="TITLE_OF_A_REVIEW").get_text(strip = True)
4             review_titles.append(title)
5
6             # Study courses - variable study_courses
7             study = review.find("PARAGRAPH_ELEMENT",
                  class_="STUDY_COURSE_OF_A_REVIEW").get_text(strip =
                  True).rstrip('\n')
8             study_courses.append(study)
9
10            # Total rating values of the review - variable rating_values
11            rating = review.find("CONTAINER_ELEMENT",
                  class_="RATING_VALUE_OF_A_REVIEW").get_text(strip = True)
12            rating_values.append(rating)
13
14            # Recommendations ("Ja" or "Nein") - variable recommendations
15            if(review.find("ICON_ELEMENT", class_="SMILEY_POSITIVE") is
                  not None):
16                recommendation.append("Ja")
17            elif(review.find("ICON_ELEMENT", class_="SMILEY_NEGATIVE")
                  is not None):
18                recommendation.append("Nein")
19
20            # Publication dates - variable comment_dates
21            if(review.find("INLINE_ELEMENT", class_="DATE_OF_A_REVIEW")
                  is not None):
22                date = review.find("INLINE_ELEMENT",
                      class_="DATE_OF_A_REVIEW").get_text(strip = True)
23                comment_dates.append(date)
24
25            # Comments (text) - variable all_comments -differentiate
                  between short and long comments!
```

```
            newurl = getnexturl(review)
            if (newurl is not None):
                try:
                    page = requests.get(newurl, headers=headers)
                    soup = BeautifulSoup(page.text, 'html.parser')
                    soup.prettify()
                    longcomment = soup.find("CONTAINER_ELEMENT",
                        class_="COMMENT_OF_A_REVIEW").get_text(strip =
                        True)
                    all_comments.append(longcomment)
                except Exception as e:
                    time.sleep(5)
                    print(e)
            else:
                comment = review.find("PARAGRAPH_ELEMENT", class_=
                    "COMMENT_OF_A_REVIEW").get_text(strip = True)
                all_comments.append(comment)

            # Covid comments - variable covid_comment
            if(review.find("CONTAINER_ELEMENT",
                class_="COVID_COMMENT_OF_A_REVIEW") is not None):
                cov_text = review.find("CONTAINER_ELEMENT",
                    class_="COVID_COMMENT_OF_A_REVIEW")\
                                .get_text(strip=True).strip('Wie reagiert
                                    deine Hochschule auf die
                                    Corona-Krise?')
                covid_comment.append(cov_text)
            else:
                covid_comment.append("n.a.")

except Exception as e:
    print(e)
```

The third and final part of the main function is depicted in Listing 4.4 and involves saving the data. First, a pandas DataFrame uni_reviews is created and populated with the collected data by attaching dictionaries that include the previously created arrays. The DataFrame is then saved as both Excel and JSON files to the local file system. Finally, the function returns the DataFrame, representing all reviews from one HEI.

Listing 4.4: Main function get_uni_reviews (3/3)

```
# Create Dataframes for each Uni
uni_reviews = pd.DataFrame({"Hochschule": uni, "Typ": uni_type, "Titel":
    review_titles, "Studiengang": study_courses, "Bewertung":
    rating_values, "Datum": comment_dates, "Kommentar": all_comments,
    "Umgang mit Corona": covid_comment, "Weiterempfehlung":
    recommendation})
print(uni_reviews)

# Path to save data -> Create Excel
path_to_save = 'C:/Users/YOUR_USER/Documents/Data/Excel/%s.xlsx' % (uni)
```

```
uni_reviews.to_excel(path_to_save, encoding='utf-8', index=False)

# Path to save data -> Create JSON
path_to_save_json = 'C:/Users/YOUR_USER/Documents/Data/JSON/%s.json' %
    (uni)
uni_reviews.to_json(path_to_save_json, force_ascii=False)

return uni_reviews
```

As discussed earlier, the helper function next is used by the main function. It determines whether the current page is the last page. The function requires the HTTP response object of the current page, which is then used to create a BeautifulSoup (BS) object, likewise as in the main function. By searching within the tag "NAVIGATION_LIST_ELEMENT" for the class "PAGINATION" or similar, the function checks if there is a subsequent page.

Listing 4.5: Helper Function next

```
def next(next_page):
    soup = BeautifulSoup(next_page.text, 'html.parser')
    next_ = soup.find('NAVIGATION_LIST_ELEMENT', {'class':'PAGINATION'})
    if next_ is None:
        return False
    else:
        return True
```

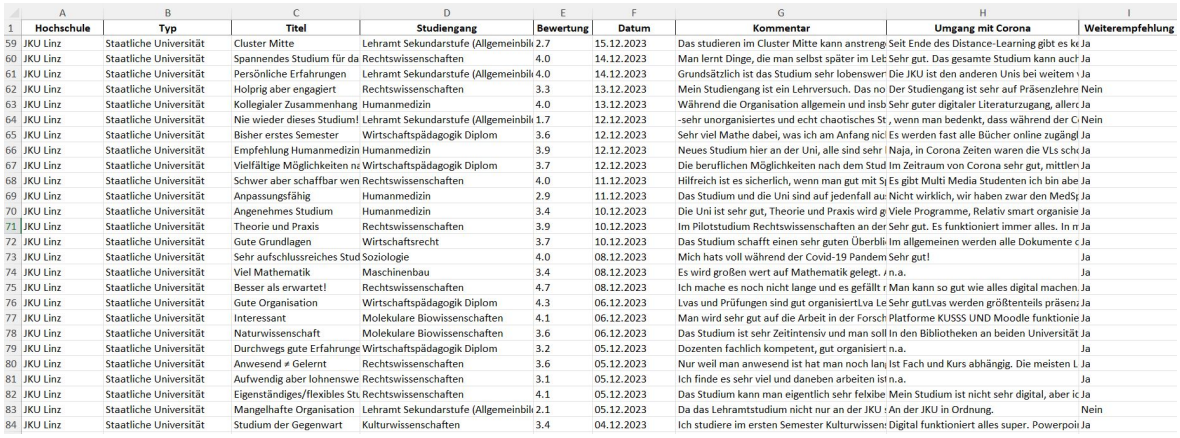
The helper function getnexturl is also called by the main function. Its purpose is to distinguish between reviews that have a "Read More" button and those that do not. It takes the current review, represented by a BeautifulSoup (BS) object, as input. If the review contains a class similar to "READMORE" within a "LINK_ELEMENT" tag, the URL linked to by the "Read More" button is created and returned. Otherwise, the function returns None, indicating that the review contains a short comment without a "Read More" button. If the URL is not None, the main function proceeds to crawl the long comment from the new URL. Otherwise, it retains the original URL.

Listing 4.6: Helper Function getnexturl

```
def getnexturl(soup):
    if soup.find('LINK_ELEMENT', class_='READMORE'):
        newurl = str(soup.find('LINK_ELEMENT')['URL_ATTRIBUTE'])
        return newurl
    else:
        return None
```

4.3.2 Results of Web Scraping

This section presents the results of the web scraping process. The generic code presented above was used on a UEP to collect reviews commenting on in total 74 different Austrian-based HEIs. Due to personal preferences, the Excel data was prioritized for further work over the JSON data. Figure 4.2 provides an example of collected ratings from JKU in the form of an Excel sheet snippet. In total, 10,810 ratings from 74 different HEIs were collected, resulting in 74 Excel files.



	A	B	C	D	E	F	G	H	I
1	Hochschule	Typ	Titel	Studiengang	Bewertung	Datum	Kommentar	Umgang mit Corona	Weiterempfehlung
59	JKU Linz	Staatliche Universität	Cluster Mitte	Lehramt Sekundarstufe (Allgemeinbild	2.7	15.12.2023	Das studieren im Cluster Mitte kann anstreng	Seit Ende des Distance-Learning gibt es ke	Ja
60	JKU Linz	Staatliche Universität	Spannendes Studium für da	Rechtswissenschaften	4.0	14.12.2023	Man lernt Dinge, die man selbst später im Leb	Sehr gut. Das gesamte Studium kann auch	Ja
61	JKU Linz	Staatliche Universität	Persönliche Erfahrungen	Lehramt Sekundarstufe (Allgemeinbild	4.0	14.12.2023	Grundsätzlich ist das Studium sehr lobenswer	Die JKU ist den anderen Unis bei weitem v	Ja
62	JKU Linz	Staatliche Universität	Holprig aber engagiert	Rechtswissenschaften	3.3	13.12.2023	Mein Studiengang ist ein Lehrversuch. Das no	Der Studiengang ist sehr auf Präsenzlehre	Nein
63	JKU Linz	Staatliche Universität	Kollegialer Zusammenhang	Humanmedizin	4.0	13.12.2023	Während die Organisation allgemein und insb	Sehr guter digitaler Literaturzugang, aller	Ja
64	JKU Linz	Staatliche Universität	Nie wieder dieses Studium!	Lehramt Sekundarstufe (Allgemeinbild	1.7	12.12.2023	-sehr unorganisiertes und echt chaotisches St	, wenn man bedenkt, dass während der C	Nein
65	JKU Linz	Staatliche Universität	Bisher erstes Semester	Wirtschaftspädagogik Diplom	3.6	12.12.2023	Sehr viel Mathe dabei, was ich am Anfang nic	Es werden fast alle Bücher online zugängl	Ja
66	JKU Linz	Staatliche Universität	Empfehlung Humanmedizin	Humanmedizin	3.9	12.12.2023	Neues Studium hier an der Uni, alle sind sehr	Naja, in Corona Zeiten waren die VLs sch	Ja
67	JKU Linz	Staatliche Universität	Vielfältige Möglichkeiten na	Wirtschaftspädagogik Diplom	3.7	12.12.2023	Die beruflichen Möglichkeiten nach dem Stud	Im Zeitraum von Corona sehr gut, mittler	Ja
68	JKU Linz	Staatliche Universität	Schwer aber schaffbar wen	Rechtswissenschaften	4.0	11.12.2023	Hilfreich ist es sicherlich, wenn man gut mit S	Es gibt Multi Media Studenten ich bin abe	Ja
69	JKU Linz	Staatliche Universität	Anpassungsfähig	Humanmedizin	2.9	11.12.2023	Das Studium und die Uni sind auf jedenfall au	Nicht wirklich, wir haben zwar den MedSp	Ja
70	JKU Linz	Staatliche Universität	Angenehmes Studium	Humanmedizin	3.4	10.12.2023	Die Uni ist sehr gut, Theorie und Praxis wird g	Viele Programme, Relativ smart organisie	Ja
71	JKU Linz	Staatliche Universität	Theorie und Praxis	Rechtswissenschaften	3.9	10.12.2023	Im Pilotstudium Rechtswissenschaften an der	Sehr gut. Es funktioniert immer alles. In n	Ja
72	JKU Linz	Staatliche Universität	Gute Grundlagen	Wirtschaftsrecht	3.7	10.12.2023	Das Studium schafft einen sehr guten Überbli	Im allgemeinen werden alle Dokumente c	Ja
73	JKU Linz	Staatliche Universität	Sehr aufschlussreiches Stud	Soziologie	4.0	08.12.2023	Mich hats voll während der Covid-19 Pandem	Sehr gut!	Ja
74	JKU Linz	Staatliche Universität	Viel Mathematik	Maschinenbau	3.4	08.12.2023	Es wird großen wert auf Mathematik gelegt. A	n.a.	Ja
75	JKU Linz	Staatliche Universität	Besser als erwartet!	Rechtswissenschaften	4.7	08.12.2023	Ich mache es noch nicht lange und es gefällt r	Man kann so gut wie alles digital machen.	Ja
76	JKU Linz	Staatliche Universität	Gute Organisation	Wirtschaftspädagogik Diplom	4.3	06.12.2023	Lvas und Prüfungen sind gut organisiert	Lva Le Sehr gut	Lvas werden größtenteils präsenz	Ja
77	JKU Linz	Staatliche Universität	Interessant	Molekulare Biowissenschaften	4.1	06.12.2023	Man wird sehr gut auf die Arbeit in der Forsch	Platforme KUSSS UND Moodle funktionie	Ja
78	JKU Linz	Staatliche Universität	Naturwissenschaft	Molekulare Biowissenschaften	3.6	06.12.2023	Das Studium ist sehr Zeitintensiv und man soll	In den Bibliotheken an beiden Universität	Ja
79	JKU Linz	Staatliche Universität	Durchwegs gute Erfahrunge	Wirtschaftspädagogik Diplom	3.2	05.12.2023	Dozenten fachlich kompetent, gut organisiert	n.a.	Ja
80	JKU Linz	Staatliche Universität	Anwesend ≠ Gelernt	Rechtswissenschaften	3.6	05.12.2023	Nur weil man anwesend ist hat man noch lan	Ist Fach und Kurs abhängig. Die meisten L	Ja
81	JKU Linz	Staatliche Universität	Aufwendig aber lohnenswe	Rechtswissenschaften	3.1	05.12.2023	Ich finde es sehr viel und daneben arbeiten ist	n.a.	Ja
82	JKU Linz	Staatliche Universität	Eigenständiges/flexibles St	Rechtswissenschaften	4.1	05.12.2023	Das Studium kann man eigentlich sehr felxibe	Mein Studium ist nicht sehr digital, aber ic	Ja
83	JKU Linz	Staatliche Universität	Mangelhafte Organisation	Lehramt Sekundarstufe (Allgemeinbild	2.1	05.12.2023	Da das Lehramtstudium nicht nur an der JKU s	An der JKU in Ordnung.	Nein
84	JKU Linz	Staatliche Universität	Studium der Gegenwart	Kulturwissenschaften	3.4	04.12.2023	Ich studiere im ersten Semester Kulturwissen	Digital funktioniert alles super. Powerpoi	Ja

Figure 4.2: Excel Sheet Snippet of Collected JKU Ratings

The number of ratings varies significantly across universities. Figure 4.3 presents an overview of the collected ratings, with the y-axis indicating the number of ratings available for each university. The University of Vienna stands out with the highest number of ratings, totaling 1,584. Following closely is the FOM University, with 1,572 ratings. Other universities with significant numbers of ratings are the University of Innsbruck, the TU Vienna, and the University of Graz, with 661, 575, and 556 reviews respectively. The JKU Linz occupies the 10th position with a total of 385 ratings. It trails behind the TU Graz, the University of Salzburg, and the WU Vienna, which have 514, 426, and 412 ratings respectively. In total, 17 Austrian HEIs had less than 10 ratings, as can be seen in 4.3. Still, all collected reviews were considered valuable for later data annotation and training, as discussed in Chapter 5. The assumption was that the majority of universities contain ratings addressing similar aspects or aspect categories. In other words, for simplicity's sake, all ratings were associated with only one domain: higher education.

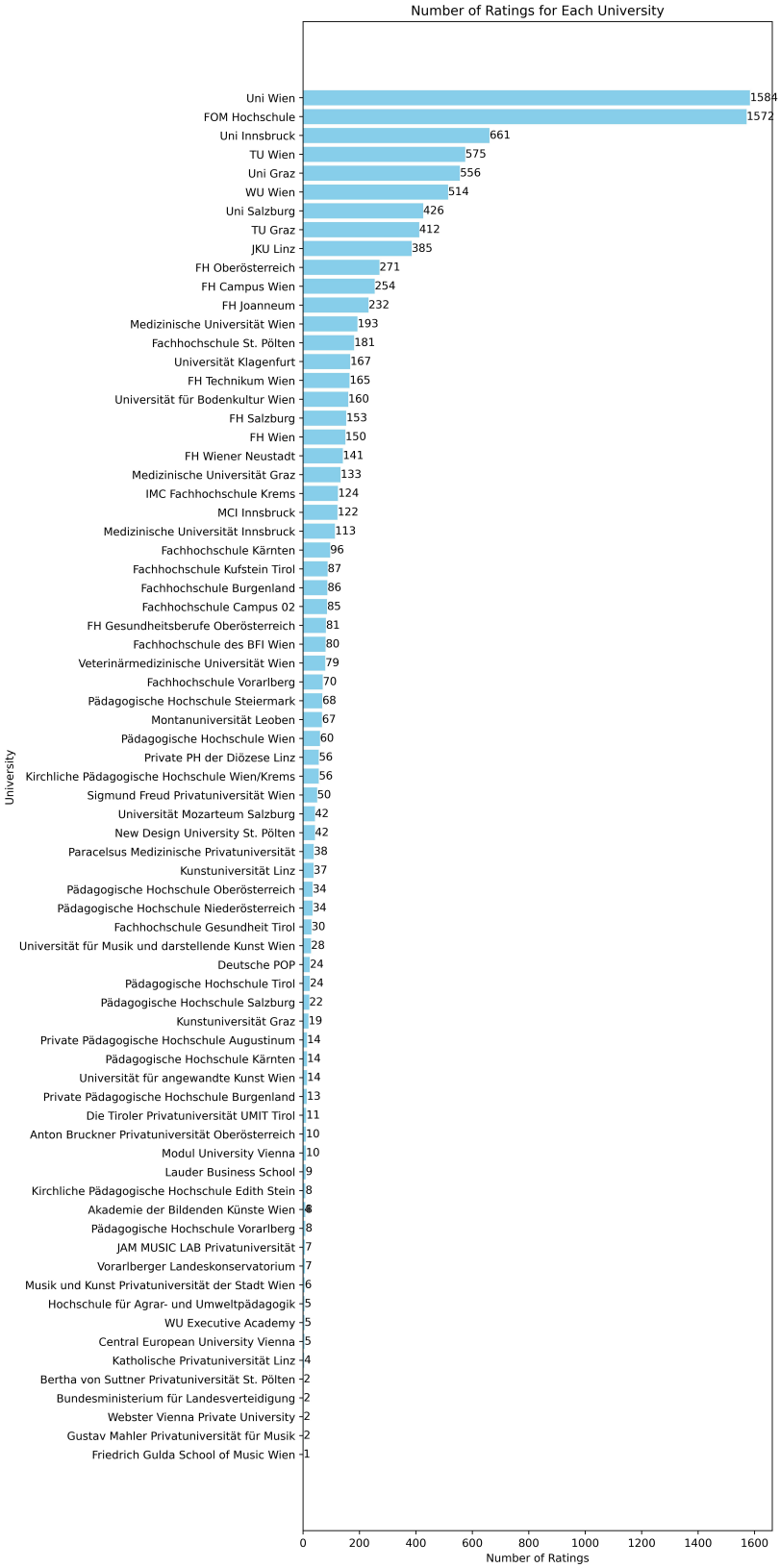


Figure 4.3: Overview of Collected Ratings - HEI Name and Number of Ratings

5

Development of a Model for Aspect-based Sentiment Analysis

This chapter discusses the training of the ABSA model and the development of the aspect category classifier, utilizing the data collected through web scraping as detailed in Chapter 4. It involves various steps including aspect category selection, data preprocessing, model training, model evaluation, and lastly, the development of a classifier for predicting aspect categories. The aim of this module is to be able to detect student sentiments expressed on certain aspects of a HEI. Therefore, it is necessary to develop a model, that can both extract aspect terms from a sentence on which an opinion is expressed, and predict the corresponding sentiment. Moreover, the aspects eventually need to be in a categorical format to later serve for OLAP querying. Since the ABSA model trained during this thesis only extracts aspect terms, explicitly occurring in the sentence, a second ML model respectively classifier had to be developed. The second model is responsible for classifying an aspect term into an aspect category. Alternatively, a model could have been developed which is able to extract aspect terms, detect sentiments, and classify aspect terms. In this scenario, a second ML model would not be needed.

The overall design of the ABSA workflow follows a similar approach to those found in the literature, as covered in Section 2.3.1. In contrast to traditional ABSA approaches, extensive preprocessing of the data is not necessary. This is because the sentiment analysis was performed using a BERT-based model. However, several processing steps to prepare the training data were still required. After collecting the student feedback, the data is prepared for the modeling phase.

Following the CRISP-DM cycle proposed by Wirth and Hipp (2000), a standard process model for data mining projects, this central module of the data warehouse system's back-end tier was conceptualized as an iterative flow, depicted in Figure 5.1.

The first step (Aspect Category Selection) details the process of defining a set of aspect categories to be later classified by the second ML model. This step involves understanding both business and data understanding and is addressed in Section 5.2. The subsequent step in the cycle (Data Preprocessing) reflects necessary data preprocessing before training both the ABSA model and the category classifier. This step includes data segmentation, data annotation, data augmentation, and training data preparation and is covered in Section 5.3. The next step after data preprocessing represents the modeling process (Modeling), which aims at training the two models with the required training data. After completing training, the models can be evaluated in a next step (Model Evaluation). To reduce complexity, the development and evaluation of the two models was

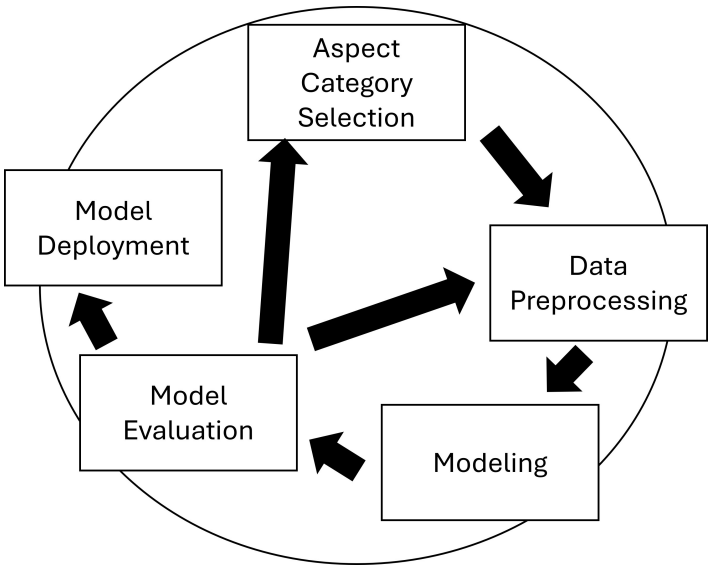


Figure 5.1: Sentiment Analysis Module (adapted from Wirth and Hipp, 2000).

divided: Section 5.4 covers PyABSA training of the ABSA model, while its evaluation is addressed in Section 5.5. Section 5.6 showcases both the development and evaluation of the second model needed for aspect category classification. Applying the trained models to the online reviews forms the last step of the cycle (Model Deployment). This step, however, is covered in Chapter 6, where both the data warehouse and front-end tiers of the designed system are discussed. Argued by Wirth and Hipp (2000), the cycle can be terminated or restarted, depending on the performance of the developed models. For instance, if the performance of the trained ABSA model is not sufficient, the data might be augmented, which is part of the data preprocessing step. Another scenario might be that after final data analysis, business stakeholders request changes concerning the selection of the aspect categories. In this case, it can be proceeded with the first step of the cycle (Wirth and Hipp, 2000).

Before starting with the first step of the cycle, the following section briefly introduces the Python library PyABSA, a modularized ABSA framework chosen for this work.

5.1 PyABSA – a Framework for Reproducible ABSA

Given the challenges ABSA poses for beginners, a user-friendly framework was needed that offers straightforward functionalities for ABSA and facilitates the training of a custom model for this specific use case. PyABSA is a library that proved helpful for this purpose. It is a modularized Python framework for reproducible ABSA built on PyTorch. Leveraging BERT-based models, it can be used for both zero-shot ABSA via its API and for further training by feeding custom training data. In this research, the latter approach was utilized to adapt to the domain of higher education involving German ratings. The framework includes task-specific logic, adding layers to the BERT architecture for fine-tuning a pretrained language model, as outlined earlier in Section 2.2.2. It supports several ABSA tasks, with the core tasks being Aspect Term Extraction (ATE), Aspect Polarity Classification (APC), and Aspect Term Extraction and Aspect Polarity Classification (ATEPC). The latter is a compound task combining ATE and APC (H. Yang et al., 2023). It's important to note that different terms are sometimes used within PyABSA sources, with APC and Aspect Sentiment Classification (ASC) referring to the same task, and ATEPC and

End-to-End-ABSA (E2EABSA) also being synonymous. Since both ATE and APC are relevant for this thesis, the focus was on the compound task ATEPC. To this end, a custom dataset for training is created using the collected data (refer to 4), and a custom ABSA model is developed by passing the training dataset to PyABSA's trainer.

Having outlined the framework used in this thesis, the subsequent section discusses the first step of the cycle: aspect category selection, detailing the process of defining a set of categories to be later classified by the second ML model.

5.2 Aspect Category Selection

Before proceeding with preprocessing the collected data, it was essential to consider how aspect categories would be later detected by the ABSA model. As outlined in the theoretical part of this thesis, there are alternative methods to consider. Aspect categories can either be directly predicted, referred to here as implicit aspect terms, or they can be predicted after first extracting explicit aspect terms. However, given PyABSA's logic of detecting aspect terms occurring explicitly within a sentence (task ATE), it was evident to opt for the same approach.

During this period, PyABSA lacked a subtask specifically aimed at aspect category detection. Therefore, it became necessary to develop a model capable of predicting a category given an aspect term. This was crucial for the final development of the sentiment cube, as it requires a fixed set of categories for querying, rather than hundreds of different aspect terms. As mentioned earlier, this step will be discussed further in Section 5.6. However, at this stage, it was necessary to decide which categories the classifier would later predict.

Figure 5.2 presents the categories of a HEI defined by the UEP which were considered highly relevant as a predefined set of aspect categories. Students may rate each category with stars ranging from 1 to 5. The overall rating is calculated by averaging the individual aspect ratings.

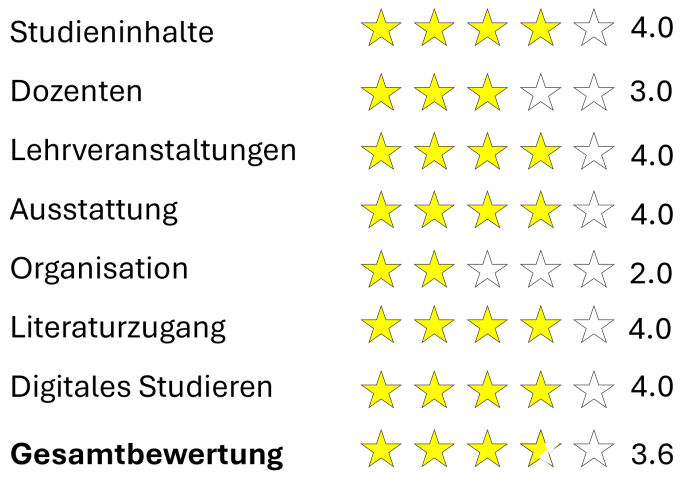


Figure 5.2: HEI Categories from the UEP (Own Illustration)

To access this view, one must click on the button *Umfrageergebnisse* (ENG: survey results), located at the bottom center of a review from the chosen UEP. Note that participation in the survey is optional on this UEP, and therefore both this button and view may not always be present. In the case of survey participation, the platform gathers more detailed data, such as student age, gender,

duration of study, start of studies, and more. However, this information is too detailed for the scope of the final sentiment cube and was therefore not collected. The categories defined by the UEP are in total seven.

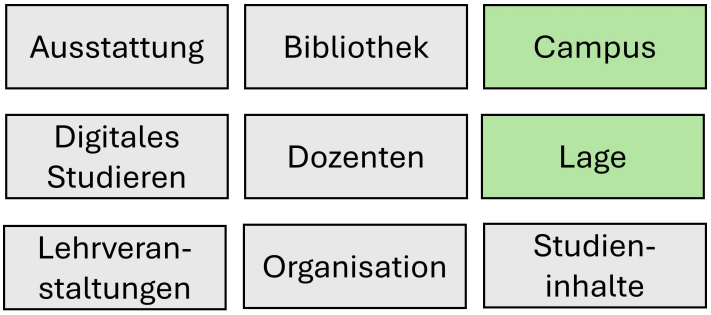


Figure 5.3: HEI Categories Extended by the Addition of "Lage" (Location) and "Campus"

It appears that the platform has slightly modified the name of the category *"Bibliothek"* (ENG: library) to *"Literaturzugang"* (ENG: access to literature). Before the process of data annotation started, the seven categories shown in Figure 5.2 were considered. The English equivalents of these categories are *Facilities, Library, Digital Studying, Lecturers, Courses, Organization, Curriculum/Study Contents*. Because students rated their study experience based on these aspects provided by the UEP, they were chosen as the basis for the set of aspect categories.

During the data annotation process, a considerable number of students expressed opinions regarding both the location of the HEI and its campus. Consequently, it was decided to incorporate the aspects of "Campus" and "Lage" (ENG: location) into the existing set of aspect categories. The updated set is illustrated in Figure 5.3.

Before commencing data annotation, there was an opportunity to explore additional aspects mentioned in the reviews. The objective was to identify other aspects rated within the UEP's predefined set of categories. Methods such as topic modeling, which can uncover latent themes within texts, were considered. One effective technique for topic modeling is Latent Dirichlet Allocation (LDA) (Jelodar et al., 2019), which was utilized to identify additional topics that could potentially serve as aspect categories.

To apply the theory of LDA, the Python library `gensim` was used to discover the most used terms within the ratings. To this end, the text data from the columns "Titel", "Kommentar", and "Umgang mit Corona" (refer to 4.2) was used and processed. After reading the data from the Excel files, it was transformed to a Pandas DataFrame and eventually concatenated to one big DataFrame. Several processing steps such as lemmatizing and tokenizing the data lead to a list of tokenized documents. In other words, the text collected from the ratings was each by each transformed into a list of tokens. Each document is represented by a list of tokens and reflects one single rating. In Listing 5.1, the list of tokenized documents (`tokenized_docs`) is passed to the function `Dictionary`, which is provided by gensim. The variable `id2word` represents a dictionary returned by the function. It includes key-value-pairs for each unique token/term within `tokenized_docs`, joined with a unique integer ID. This dictionary is then used by the LDA algorithm to discover the most frequent terms and clusters them into topics.

Listing 5.1: Creation of a Dictionary including unique terms

```
# Mapping from word IDs to words
id2word = corpora.Dictionary(tokenized_docs)
```

After obtaining the dictionary, the next step was to create bag-of-word representations (BOW) for each document (rating). To achieve this, LDA necessitates the generation of one document-term matrix for each document. Listing 5.2 illustrates this process. For each document from tokenized_docs, a BOW or a document-term matrix is generated using the function doc2bow on the previously created dictionary id2word.

Ultimately, the variable corpus contains a list of document-term matrices for each document or rating, representing the frequency of each term in each document.

Listing 5.2: Creation of the Document-Term-Matrix for LDA

```
# Prepare Document-Term Matrix
corpus = []
for doc in tokenized_docs:
    corpus.append(id2word.doc2bow(doc))
```

Thereafter, an LDA model was trained using gensim's LdaModel function. This model was initialized with the dictionary id2word and the document-term matrices corpus, as demonstrated in Listing 5.3. Finally, the parameter num_topics was set to 5, indicating the number of latent topics the algorithm should extract.

Listing 5.3: LDA Topic Modeling with gensim

```
topic_model = gensim.models.ldamodel.LdaModel(
    corpus = corpus,       # Document-Term Matrix
    id2word = id2word,     # Map word IDs to words
    num_topics = 5,       # Number of latent topics to extract
    random_state = 100,
    passes = 100,          # Number of passes through the corpus during
        training
    )
```

The following figures visualize the results of the topic modeling approach using pyLDAvis, depicting the most frequent words within each of the five detected latent topics. For this visualization, the relevance metric λ was set to 0. The relevance metric ranges between 0 and 1 and ranks the terms based on two factors: term frequency and exclusivity. With λ set to 0, exclusivity is prioritized, meaning terms unique to the regarded topic and not appearing frequently in other topics are ranked higher. Conversely, with λ set to 1, terms that appear frequently within the regarded topic are ranked higher, regardless of their frequency in other topics (Sievert and Shirley, 2014). With λ set to 0, the red color of the bars represents the estimated term frequency within the topic, while the blue color indicates the overall term frequency. The blue color is barely recognizable because λ being set to 0 ranks terms that appear infrequently in other topics.

Figure 5.4 shows the result for Topic 1. The model associates a total of 34.8% of all tokens with the topic. Given the frequent terms "Praxis" and "Theorie," this topic could potentially be linked

to a category such as "Praxisanteil" (ENG: practical component) or "Praxis-Theorie-Balance" (ENG: practice-theory balance). However, it was decided not to define this as an additional aspect category. This is because the aspect of practice-theory balance can be reflected by the predefined category "Studieninhalte" (ENG: curriculum/study contents), which already exists in the predefined set. Similarly, frequent terms detected in the rest of the topics were already covered by the base set of categories. The result of all five topics, including the top three occurring terms, is summarized below:

- **Topic 1**: Praxis, Theorie, praktisch (34.8% of all tokens)
- **Topic 2**: Literatur, Kultur, JUS (5.6% of all tokens)
- **Topic 3**: Fakultät, Abschnitt, Journalismus (5% of all tokens)
- **Topic 4**: Organisation, online, digital (34.8% of all tokens)
- **Topic 5**: Chemie, TU, Biologie (19.8% of all tokens)

Topic 2 seems to correlate with the category "Bibliothek" (library). Topic 3 exhibits associations with multiple categories. Topic 4 appears to align with the existing aspect categories of "Organisation" or "Digitales Studieren." Meanwhile, Topic 5 predominantly revolves around study degrees, suggesting a possible connection to the existing aspect categories "Studieninhalte" (curriculum/study contents) or "Lehrveranstaltungen" (lectures). However, due to the subjective nature of category definition and the somewhat arbitrary nature of linking frequent words to aspect categories, it was deemed unnecessary to introduce any additional aspect categories to the base set. The observed lack of relevance in the results may stem from the high correlation or overlap among the topics present in the data. Additionally, the identified topics can be mapped to the existing base set of categories.

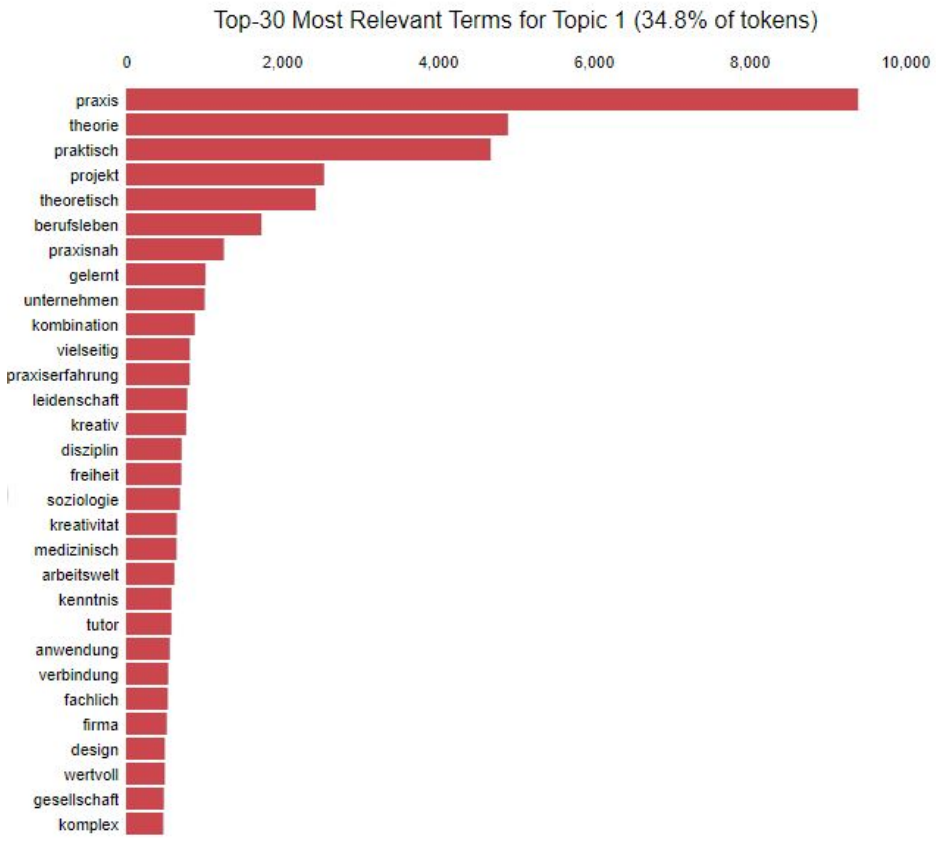


Figure 5.4: LDA Algorithm - Topic 1 Including Frequent Terms

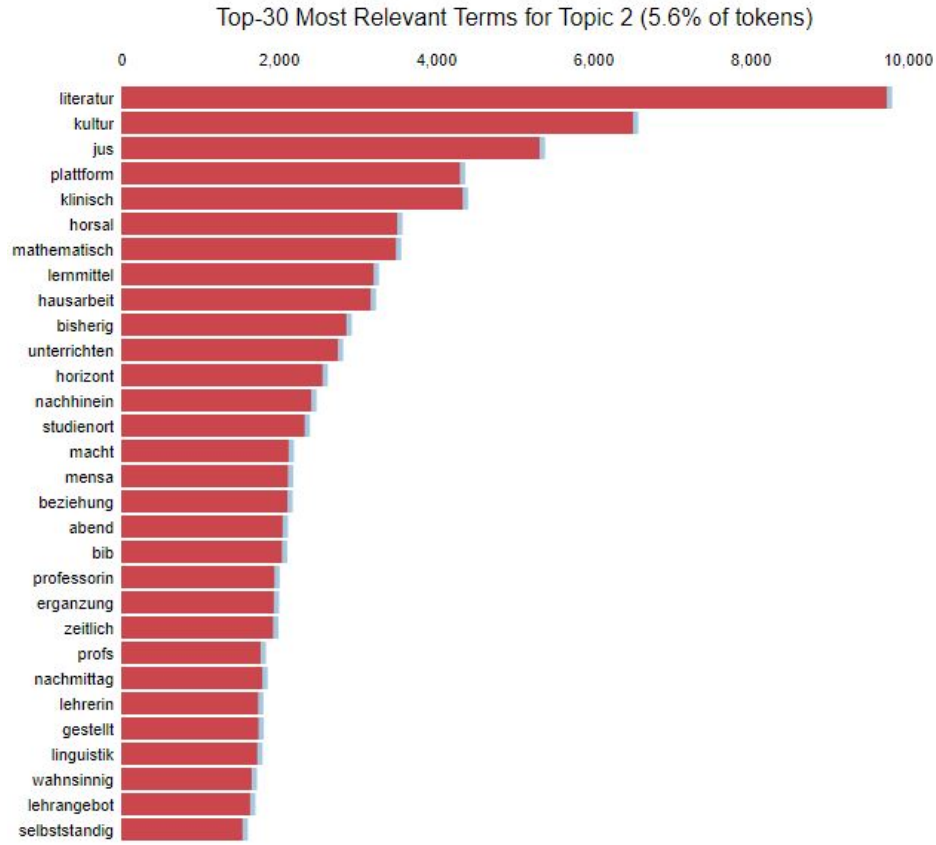


Figure 5.5: LDA Algorithm - Topic 2 Including Frequent Terms

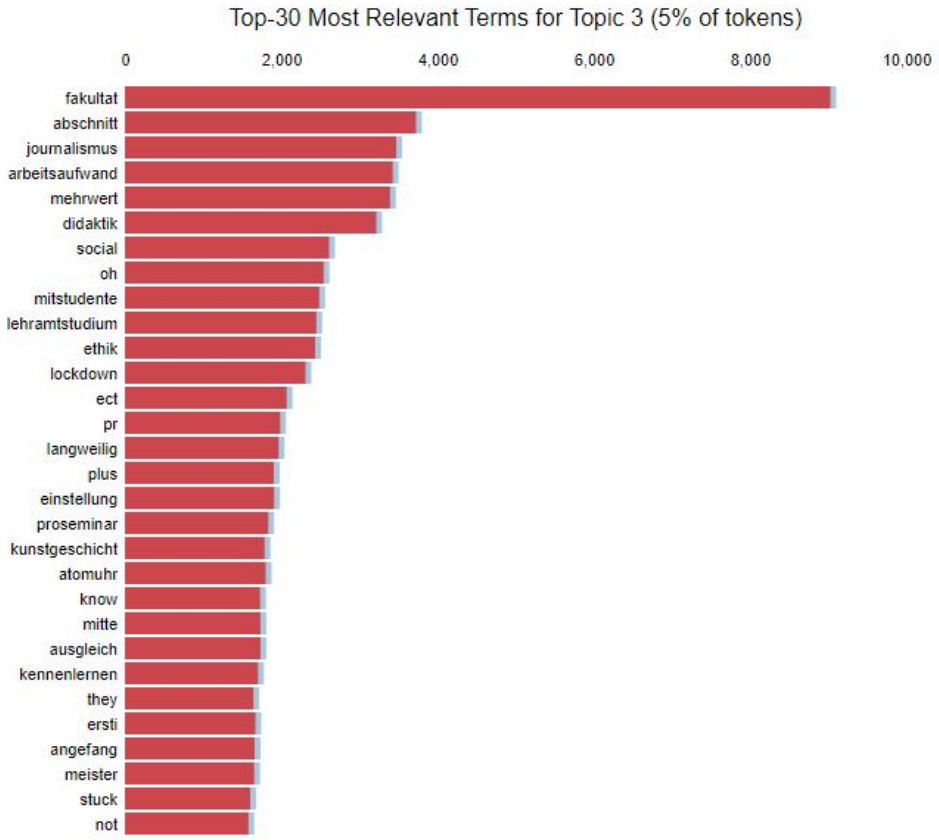


Figure 5.6: LDA Algorithm - Topic 3 Including Frequent Terms

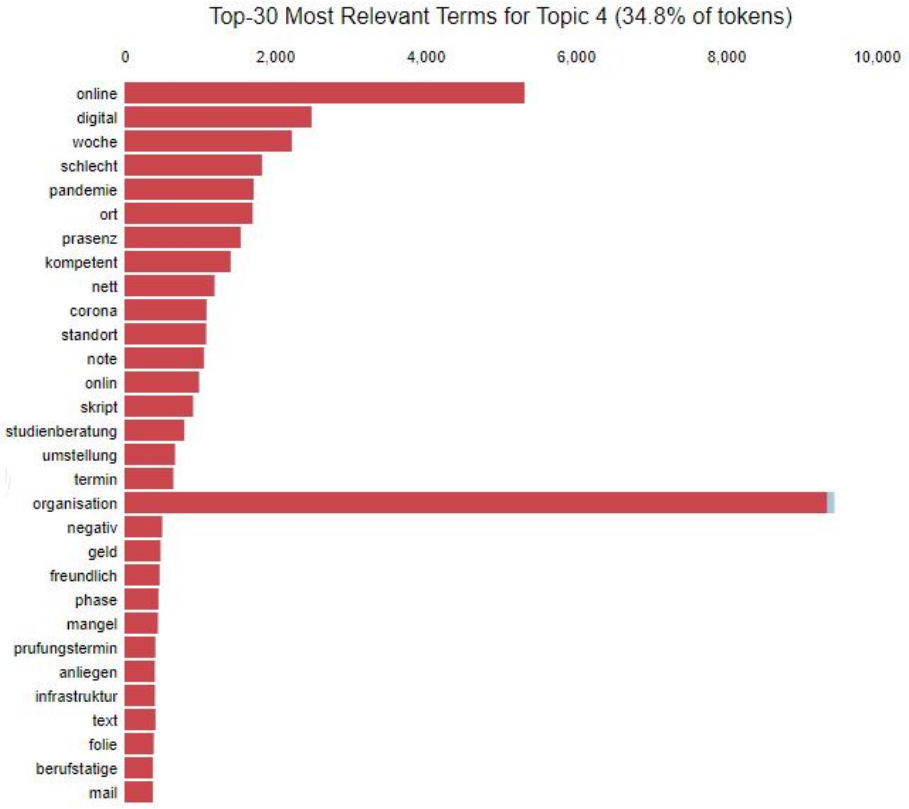


Figure 5.7: LDA Algorithm - Topic 4 Including Frequent Terms

It turned out that the results of the topic modeling approach did not yield promising results for this use case. As shown in the list above, there are no frequent words occurring which could be

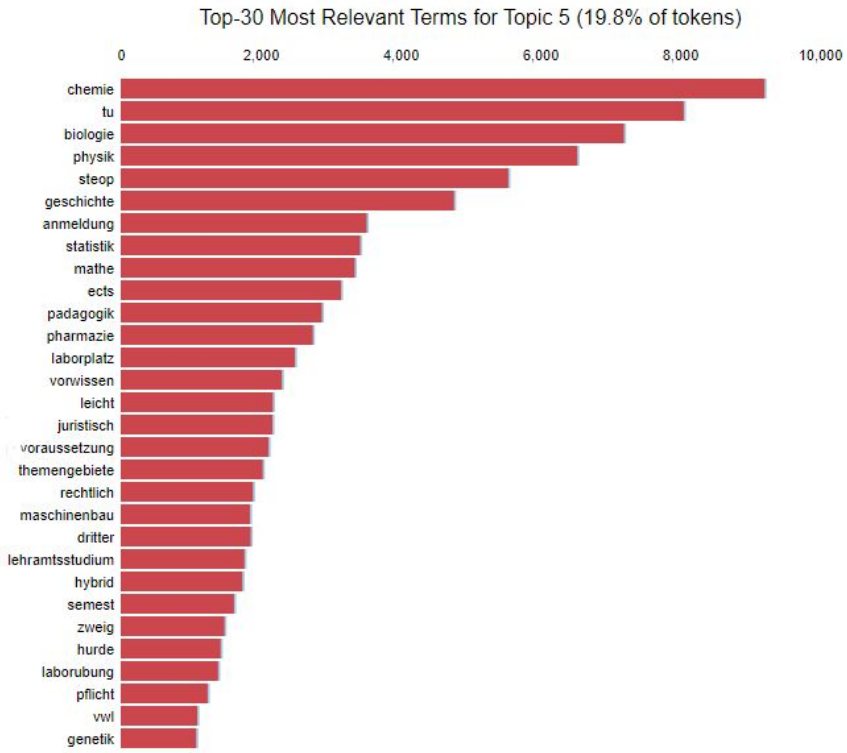


Figure 5.8: LDA Algorithm - Topic 5 Including Frequent Terms

associated with a category other than the already defined ones on the UEP. Therefore, no further categories were included.

If the ABSA model fails to detect any aspect term within a sentence or the detected aspect term cannot be associated with any of the nine predefined categories, a general category, "Allgemein", is included in the set of predefined aspect categories. As a result, there are a total of 10 defined aspect categories.

5.3 Data Preprocessing

After deciding on which aspect categories to select, the subsequent step involved preparing a dataset essential for training the ABSA model. This necessitated researching the specific requirements set by PyABSA for training data. As outlined in Chapter 2, the BERT's context-aware nature renders typical NLP preprocessing steps such as stemming, lemmatization, and the removal of stop words and punctuation unnecessary. However, the preparation of the collected data to fit for the model involved other data preprocessing steps.

Initially, data segmentation had to be performed, as models trained with PyABSA expect a sentence as input, not a whole comment. Therefore, the reviews had to be split into sentences. This step is described in Section 5.3.1. The next step was to annotate the data to create the required dataset for training the model. This included labeling each sentence with both an aspect term and its corresponding sentiment (positive, neutral, or negative). The data annotation process is outlined in Section 5.3.2. After annotation, the training data was augmented due to two reasons: a high imbalance found in the distribution of sentiments, and a low quantity of training examples. The process of data augmentation is detailed in Section 5.3.3. The last preprocessing step was to first

split the training data into three subsets, before transforming them into the format required by PyABSA. This process is covered in Section 5.3.4.

5.3.1 Data Segmentation

The collected data comprises 74 Excel files, totaling 10,810 ratings. Assuming an average of seven sentences per comment and COVID-related comment, there are approximately 75,670 sentences to annotate. To manage the workload, only 10 percent of the reviews were selected for annotation, resulting in 8,312 sentences. Moreover, the sample was generated in a randomized manner to ensure a representative and unbiased sample of the data (refer to 5.4). As discussed earlier in Chapter 2, it was assumed that several thousand examples typically suffice for further post-training of a pretrained model.

As previously mentioned, it was essential to verify the requirements for training data. Provided by the author of PyABSA, Figure 5.9 depicts two data examples in the APC format. This format is necessary in PyABSA for predicting sentiments and addresses how to handle examples with multiple aspect terms: each aspect term requires a separate tuple, necessitating duplication of the sentence. Additionally, each example corresponds to a single sentence rather than an entire paragraph. Therefore, both comments and COVID comments needed to be segmented into individual sentences.

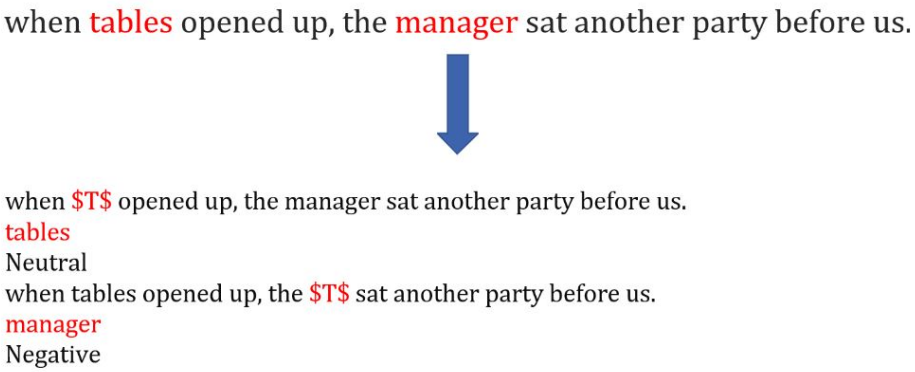


Figure 5.9: Training Data Example - APC Format in PyABSA (H. Yang, 2022)

The APC format illustrated in Figure 5.9 will be pertinent in Section 5.3.4. Subsequently, the process of splitting reviews into sentences is detailed.

To streamline the data segmentation, a Python function, illustrated in Listing 5.4, was developed. The function was utilized to write sentences to be annotated into an Excel file. For splitting text into sentences, the Python library SoMaJo was used. It is a library containing tokenizing and splitting functionalities. It was selected due to its specialization in German and English social media texts (Proisl and Uhrig, 2016). However, other tools could have been used as well. The function requires a path to an Excel file containing all crawled ratings from a single HEI. Therefore, each of the 74 Excel files was processed individually by passing their respective file paths to the function. In essence, the function was called 74 times, resulting in the creation of one Excel file containing all sentences. The following paragraphs detail the content of the function.

After passing the Excel file path to the function, it was transformed into a pandas `DataFrame`. Subsequently, using the `sample` function from pandas, a randomized sample of 10 percent was

taken in line 11. Since only the comment and COVID comment contain the data relevant for training, the two columns "Kommentar" and "Umgang mit Corona" were simultaneously retrieved using pandas' column selection `[[]]`. Then, both the comments and COVID comments were saved into individual variables `all_comments` and `all_covid_comments`, respectively. These variables represent pandas `Series` whose contents were then copied into a list named `paragraphs`, where each element corresponds to a complete comment or rating.

The next step involved using SoMaJo's tokenizer to segment each sentence, resulting in tokenized sentences stored in the variable `sentences`. To facilitate further processing, these sentences were converted into a list format. To achieve this, a for loop (defined in line 49) iterated through each sentence, appending the text of each token to an initially empty Python list `result_sentences` (created in line 46).

Additionally, the sentences were cleaned of extra blank spaces added by SoMaJo's tokenizer before each end punctuation mark of a sentence, a task accomplished in lines 53-55. Subsequently, the finalized list of sentences was copied into a pandas `DataFrame` `df_sen` and returned by the function. In addition to the "sentence" column, the DataFrame was added three additional columns: "aspect", "sentiment", and "aspect category."

Listing 5.4: Function create_sentences - Data Segmentation for Annotation

```
1  import pandas as pd
2  from somajo import SoMaJo
3
4  def create_sentences(reviews):
5      # reviews: Excel file containing all ratings from one single
           university
6
7      # Transform to pandas DataFrame
8      reviews_df = pd.read_excel(reviews)
9
10     # Take random 10 percent of the reviews
11     reviews_random = reviews_df.sample(frac=0.1)[["Hochschule",
           "Kommentar", "Umgang mit Corona"]]
12
13     # Get comments respectively covid comments
14     all_comments = reviews_random["Kommentar"]
15     all_covid_comments = reviews_random["Umgang mit Corona"]
16
17     # To be tokenized
18     paragraphs = []
19
20     # Add normal comment
21     for review in all_comments:
22         if isinstance(review, str):  # Check if the comment is already a
               string
23             paragraphs.append(review)
24         elif isinstance(review, float) and pd.isna(review):  # Handle
               NaNs if present
25             paragraphs.append("")
```

```
        else:  # Convert to string if it's not already
            paragraphs.append(str(review))

    # Add covid comment
    for review in all_covid_comments:
        if isinstance(review, str):  # Check if the comment is already a
            string
            paragraphs.append(review)
        elif isinstance(review, float) and pd.isna(review):  # Handle
            NaNs if present
            paragraphs.append("")
        else:  # Convert to string if it's not already
            paragraphs.append(str(review))


    # SoMaJo - library for splitting text in grammatically correct
        sentences (effective for German texts)
    tokenizer = SoMaJo("de_CMC", split_camel_case=True)

    # Tokenize paragraphs
    sentences = tokenizer.tokenize_text(paragraphs)

    # List used for resulting sentences
    result_sentences = []

    # Get the text for each token and append to a list
    for sentence in sentences:
        result_sentences.append(" ".join([token.text for token in
            sentence]))

    # Removing blank spaces before punctuation
    for i in range(len(result_sentences)):
        result_sentences[i] = result_sentences[i].replace(" ,",
            ",").replace(" .", ".").replace(" !", "!").replace(" ?", "?")

    # Transform list into pandas DataFrame
    df_sen = pd.DataFrame({'sentence': result_sentences, 'aspect': None,
        'sentiment': None, 'aspect category': None})

    return df_sen
```

Listing 5.5 demonstrates the function get_file_paths, designed to retrieve a list of file paths containing all 74 Excel files comprising the scraped reviews. It takes as a parameter the directory path where these Excel files are located.

Listing 5.5: Function get_file_paths - Data Preparation for Annotation

```
import os

def get_file_paths(path_to_raw_reviews):
```

```
    # path_to_raw_reviews: path including all 74 Excel files containing
        student reviews

    # Verify that path_to_raw_reviews is a valid directory
    if not os.path.isdir(path_to_raw_reviews):
        raise ValueError(f"The provided path '{path_to_raw_reviews}' is
            not a valid directory.")

    # Initialize an empty list to store file paths for each of the 74
        Excel files
    file_paths = []

    # Iterate through all items (files and directories) in the directory
    for item in os.listdir(path_to_raw_reviews):
        item_path = os.path.join(path_to_raw_reviews, item)

        # Check if the item is a regular file (not a directory)
        if os.path.isfile(item_path):
            file_paths.append(item_path)

    return file_paths
```

Listing 5.6 presents a function designed to generate a pandas DataFrame that consolidates all sentences retrieved by the create_sentences function. It utilizes both of the previously described functions. Subsequently, this DataFrame is saved into an Excel file intended for annotation purposes. As discussed earlier in Chapter 2, BERT learns bidirectional context simultaneously. Therefore, it was assumed that no further preprocessing, such as the removal of stop words, was necessary, as BERT benefits from this enriched context.

Listing 5.6: Function prepare_annotation - Data Preparation for Annotation

```
def prepare_annotation(path_to_raw_reviews):

    # Create an empty DataFrame for Training Data
    df = pd.DataFrame(columns=['sentence', 'aspect', 'sentiment',
        'aspect category'])


    # Get file paths containing all 74 Excel files
    file_paths = get_file_paths(path_to_raw_reviews)

    # Write sentences of each Excel file to the result Excel for
        Annotation
    for file_path in file_paths:
        df_sen = create_sentences(file_path)

        # Append df_sen to df
        df = pd.concat([df, df_sen], ignore_index=True)

    return df
```

5.3.2 Data Annotation

Figure 5.10 illustrates a snippet of the Excel file prepared for annotation. Before eventually transforming the training data into the format required by PyABSA, the data was first labeled within the Excel file.

The annotation of the data was performed entirely by myself. As part of this thesis, the data was manually labeled to ensure the quality and consistency of the annotations. This involved assigning them to specific aspects and sentiments. For the annotation, a systematic approach was followed, where each sentence was reviewed carefully, its relevant aspect term was extracted, and a label (positive, neutral, negative) was assigned that corresponds with the selected aspect term. Moreover, at this time, it was not clear how the corresponding aspect categories would be predicted later. Therefore, the decision was made to additionally label each sentence with an aspect category. This ensured that there would be training data available to model a classifier for category prediction.

Figures 5.11shows a snippet of the annotated sentences. In summary, the following attributes were considered:

- Sentence: The sentence to be annotated.
- Aspect: The relevant aspect of the sentence, if any occurred, such as "Professoren," (ENG: professors) "Materialien," (ENG: materials) or "organisiert" (ENG: organized).
- Sentiment: The corresponding polarity of the selected aspect term, categorized as "positive," "negative," or "neutral."
- Aspect Category: One of the 10 predefined aspect categories, serving as a higher-level class to group similar aspects together.

	A	B	C	D
1	**sentence**	**aspect**	**sentiment**	**aspect category**
2	Der Instrumentalunterricht und somit das wichtigste in meinem Studium ist unvergleichlich großartig.			
3	Ansonsten kann ich mit den Lehrinhalten und dem verzweifelten Versuch einem Instrumentalstudium akademischen Charakter zu geben wenig bis gar nichts anfangen.			
4	Online Vorlesungen und PrüfungenInstrumentalunterricht online leider unmöglich.			
5	Ich konnte viel mitnehmen aus den meisten Kursen für die richtige Berufswelt inbezug auf Recording, Mix & Master sowie auch Sound für Film.			
6	Einen besonderen Dozenten gab es für mich der einem die Augen geöffnet hat und alles getan hat um einem weiter zu helfen jedoch meist zum selbst denken angeregt hat.			
7	Dennoch war nicht jeder Dozent so.			
8	Natürlich wird man nicht direkt auf alles vorbereitet was danach passiert aber das wissen wo man sich aneignen kann und weitergegeben wird ist definitiv hilfreich für den Start und die ersten Schritte.			
9	Man merkt auch durch das Studium das man nichts erreicht wenn man es nicht selbst anpackt denn geschenkt wird einem hier nichts.			
10	Leider war alles Online und das Face to Face Kontakte knüpfen hat gefehlt was eventuell hinterher immer positiv für einen selbst sein kann bezüglich auf Karriere.			
11	Ich habe den Bachelor in Music Technology abgeschlossen und kann leider fast nur schlechtes Berichten.			
12	Der Grund warum ich durchgezogen habe den BA zu beenden, war weil ich bereits alles im Vorhinein gezahlt hatte.			
13	Das war auch schon wirklich der einzige Grund.			
14	Die Deutsche POP ist leider inhaltlich und organisatorisch ein absoluter Reinfall und ich rate jeder Person die ernsthaft studieren will absolut davon ab dies dort zu machen.			
15	Für unsere Klasse war es so schlecht dass viele von uns eine beschwerde abgegeben haben....			
16	Auch lockt sie mit ihrem " International anerkannten Bachelor " den man ganz Sketchy aber nicht aus Deutschland oder von der POP sondern von der University of West London aus England... welche man nie zu Gesicht bekommt und nichts damit zu tun hat.			
17	In der Realität sieht es dann auch so aus dass man auf so gut wie keiner Universität mit dem Abschluss akzeptiert wird, da Allgemein bekannt ist dass das Niveau so niedrig ist und der BA Titel eben anscheinend mit Tricks umgangen wird.			
18	(Sonst könnte man ja auch einfach nen BA Titel aus Deutsvhland oder Österreich kriegen... und müsste nicht ständig betonen dass dieser anerkannt wird schließlich sollte das ja eig logisch sein?)			
	Ich kenne nun sehr viele Leute - mich inkludiert - die an verschiedensten Unis abgelehnt wurden da sie die Deutsche POP und deren			

Allgemein
Ausstattung
Bibliothek
Campus
Digitales Studieren
Dozenten
Lage
Lehrveranstaltungen
Organisation
Studieninhalte

Figure 5.10: Snippet of Data Prepared for Annotation

As discussed earlier, if more than one aspect term was identified within a sentence—meaning if multiple terms were subjectively deemed relevant—the sentence was duplicated to ensure that each tuple contained only one aspect term.

For example, rows 35 to 37 illustrate a duplicated sentence due to the identification of multiple aspect terms. The English translation of the sentence is: "The 2 lecturers who run the place really

have a passion for the content, which is also noticeable in the lectures, only the teaching style and the expectations of the students still need work." In this instance, the phrase "really have a passion for the content" expresses a positive sentiment towards the aspect term "lecturers", while "which is also noticeable in the lectures" conveys another positive opinion about the aspect term "lectures". Conversely, "still need work" expresses a negative sentiment towards the aspect term "teaching style". In total, three aspect terms were identified within the same sentence, consisting of two positive sentiments and one negative sentiment. The aspect categories assigned to the extracted aspect terms are "Dozenten" (ENG: lecturers), "Lehrveranstaltungen" (lectures), and again "Lehrveranstaltungen". Figure 5.12 summarizes the annotations for this sentence.

As previously discussed, the aspect terms were required to match exactly those present in the sentence, including any typos. If no aspect terms were identified, a NULL value was inserted, indicating the generic aspect category "Allgemein". Additionally, PyABSA considers neutral sentiments alongside positive and negative ones. Therefore, if an aspect term was identified towards an objective expression, it was associated with neutral sentiment. Moreover, in some sentences, there can be both a positive and negative phrase associated with the same aspect term. In such cases, the neutral label was also used. For instance, the following sentence contains both a positive and negative sentiment expressed towards the aspect term "Lehrveranstaltungen": "Zum Teil richtig gute Lehrveranstaltungen und zum Teil wieder welche, bei denen ich mir mehr erhofft habe." (ENG: "Some of the courses are really good and some of them I had hoped for more.") Consequently, in situations like this, the neutral label was chosen as the sentiment, justified by balancing both positive and negative opinions.

	sentence	aspect	sentiment	aspect category
30	Jedoch sind mittlerweile die Kursorganisation und die Systeme an Corona angepasst und sehr ordentlich verarbeitet.	Systeme	positive	Organisation
31	Das Studium ist ganz neu, klar, dass es noch nicht so ausgereift ist.	Studium	neutral	Allgemein
32	Das Studium ist ganz neu, klar, dass es noch nicht so ausgereift ist.	NULL	negative	Allgemein
33	Der Studiengang bilded sich quasi wärend es läuft, und als Studi muss man sich halt immer aufs nächste Semester überraschen lassen.	Studiengang	neutral	Allgemein
34	Der Studiengang bilded sich quasi wärend es läuft, und als Studi muss man sich halt immer aufs nächste Semester überraschen lassen.	NULL	negative	Allgemein
35	Die 2 Dozentinnen, die den Laden schmeißen haben dafür wirklich eine Leidenschaft für die Inhalte, das merkt man auch in den Vorlesungen, nur am Lehrstil und den Erwartungen an die Studierenden müsste noch gearbeitet werden.	Dozentinnen	positive	Dozenten
36	Die 2 Dozentinnen, die den Laden schmeißen haben dafür wirklich eine Leidenschaft für die Inhalte, das merkt man auch in den Vorlesungen, nur am Lehrstil und den Erwartungen an die Studierenden müsste noch gearbeitet werden.	Vorlesungen	positive	Lehrveranstaltungen
37	Die 2 Dozentinnen, die den Laden schmeißen haben dafür wirklich eine Leidenschaft für die Inhalte, das merkt man auch in den Vorlesungen, nur am Lehrstil und den Erwartungen an die Studierenden müsste noch gearbeitet werden.	Lehrstil	negative	Lehrveranstaltungen
38	Super schnell, eigentlich.	NULL	positive	Allgemein
39	Und das, obwohl es bei den meisten Studiengängen eher schwierig ist, das ganze auf Online Unterricht umzuverlagern.	Online Unterricht	positive	Digitales Studieren
40	Da es sich um einen neuen Studiengang handelt, bedarf es noch an besserer Organisation was Studueninhalte betrifft.	Studiengang	neutral	Allgemein
41	Da es sich um einen neuen Studiengang handelt, bedarf es noch an besserer Organisation was Studueninhalte betrifft.	Studueninhalte	negative	Studieninhalte
42	Außerdem mangelt es leider stark an technischer Ausstattung am Medienstandort.	Ausstattung	negative	Ausstattung
43	Jedoch ist es insgesamt bis jetzt ein interessantes Studium mit guten Grundlagen und einem sehr breiten Spektrum an verschiedenen Themengebieten.	Studium	positive	Allgemein
44	Jedoch ist es insgesamt bis jetzt ein interessantes Studium mit guten Grundlagen und einem sehr breiten Spektrum an verschiedenen Themengebieten.	Grundlagen	positive	Allgemein
45	Jedoch ist es insgesamt bis jetzt ein interessantes Studium mit guten Grundlagen und einem sehr breiten Spektrum an verschiedenen Themengebieten.	Themengebiete	positive	Studieninhalte
46	Prinzipiell gut ausgestattet mit Kameras für hybrid LVs, doch die Umsetzung hängt von den ProfessorInnnen ab.	ProfessorInnnen	negative	Dozenten
47	Prinzipiell gut ausgestattet mit Kameras für hybrid LVs, doch die Umsetzung hängt von den ProfessorInnnen ab.	ausgestattet	positive	Ausstattung
48	Die meisten haben distance learning ganz gut umgesetzt.	distance learning	positive	Digitales Studieren
49	Die Lehrveranstaltungen sind teils anspruchsvoll teils sehr einfach bzw. nicht arg fordernd.	Lehrveranstaltungen	neutral	Lehrveranstaltungen
50	Der Studiengang ist noch sehr frisch und somit im Aufbau und ständiger Verbesserung somit fehlt es noch an Wissen über passende Lehrinhalte und Dozenten.	Lehrinhalte	negative	Studieninhalte

Figure 5.11: Snippet of Data Annotated

As a result of data annotation, a dataset containing 8,312 tuples was created. However, since PyABSA does not support NULL aspect terms, tuples without a detected aspect term (i.e., those with the value "NULL") were removed. This lead to a dataset with a total of 6,499 examples. As mentioned earlier, PyABSA requires custom training datasets to include only the attributes "sentence," "aspect term," and "sentiment. Thus, to further prepare the data for model training, a copy of the file was created, omitting the "aspect category" column. A file containing the attributes "aspect" and "aspect category" was saved for the potential development of a separate category classifier, as discussed earlier.

The next step involved analyzing the sentiment label distribution in the annotated data. Figure

Sentence	Aspect	Sentiment	Aspect Category
The 2 **lecturers** who run the place really have a passion for the content, which is also noticeable in the **lectures**, only the **teaching style** and the expectations of the students still need work.	lecturers	Positive	lecturers
The 2 **lecturers** who run the place really have a passion for the content, which is also noticeable in the **lectures**, only the **teaching style** and the expectations of the students still need work.	lectures	Positive	lectures
The 2 **lecturers** who run the place really have a passion for the content, which is also noticeable in the **lectures**, only the **teaching style** and the expectations of the students still need work.	teaching style	Negative	lectures

Figure 5.12: Example of an Annotated Sentence (Translated from German)

5.13 illustrates a significant class imbalance, particularly with a scarcity of negative labels compared to positive ones. Given the higher significance of negative feedback over neutral, addressing this class imbalance became crucial. Therefore, the subsequent Section 5.3.3 outlines an oversampling process aimed at balancing the class distribution. Simultaneously, the dataset was extended to obtain more examples and further prevent overfitting.

Sentiment	Count
Positive	3.079
Neutral	2.084
Negative	1.336

Figure 5.13: Class Imbalance before Data Augmentation

5.3.3 Data Augmentation

As discussed earlier in Chapter 2, various data augmentation and oversampling techniques are available to address both data scarcity and class imbalance issues. It was decided to utilize the Python library NLPAug for data augmentation in NLP tasks. This framework offers a range of methods for augmenting textual data, including random token insertions, word swapping, deletion, and substitution. Moreover, NLPAug provides advanced techniques leveraging pretrained language models such as BERT Ma, 2019. For the purpose of this thesis, contextual word embeddings using a BERT-based model were used. The used model is called "dbmdz/bert-base-german-cased" (Hugging Face Inc. and deepset, 2023), which was specifically designed for German text. The selection of this technique is justified by PLLMs outperforming traditional methods, as discussed earlier in Chapter 2.

To achieve a fully balanced dataset across all three sentiment classes, the approach involved initially doubling the number of tuples in the neutral class, increasing from 2,084 to 4,168 examples. Subsequently, the positive and negative sentiment examples, initially 3,079 and 1,336 respectively, were augmented to match the size of the neutral class at 4,168 examples each. This resulted in a dataset totaling 12,504 examples with an equal distribution of sentiments.

To facilitate augmentation, a Python function was developed requiring a pandas `DataFrame` with columns "sentence", "aspect", and "sentiment". The DataFrame is filtered based on the sentiment class targeted for oversampling. The function returns a new pandas `DataFrame` featuring augmented sentences. This augmentation involves randomly inserting tokens into each sentence while preserving the original tuple count. The implementation of this function can be found in Listing 5.7.

To utilize NLPAug's contextual word embeddings, the `nlpaug.augmenter.word` module was imported. In lines 6-7, an instance of `ContextualWordEmbsAug` was created, specifying the model path for embedding construction. Moreover, the action parameter was set to determine the augmentation technique. Besides "insert", "substitute", "swap", or "delete" could be selected as parameter value. After experimenting with `"substitute"` for several attempts, it was concluded to proceed with `"insert"` instead of substitutions. This is because using "substitute" would replace extracted aspect terms with synonyms, necessitating the labeling of augmented sentences. The approach pursued was to assign the same aspect term and sentiment label to the resulting augmented sentences as the original sentences, thereby avoiding additional manual annotation. Moreover, substitutions have shown to potentially alter the sentence's semantics by changing phrases relevant to the polarity of the aspects. Deletions were disregarded for similar reasons, and while "swap" was considered, it was found not to significantly alter the sentence structure. Thus, random insertions were chosen as they might effectively augment the sentence content.

In line 10, the DataFrame was trimmed to the "sentence" column, as augmentation was solely regarding sentences. In line 13, a pandas `Series` of booleans was saved as `dupl`, indicating whether a sentence has already occurred or if it is the first occurrence. This was achieved by setting the keep parameter to `"first"`. The Series `dupl` was then converted to a Python list in line 16. This list was necessary later to ensure that duplicated sentences are augmented equally.

In line 19, the original sentences were copied into a list. An empty list `aug_list` was created in line 22 to store the augmentation results.

Then, a for loop (lines 27-32) iterates through a combination of both lists: `sen_list` and `dupl_list`. If a sentence represents its first occurrence, it is augmented using NLPAug's `augment` function, and the result is appended to `aug_list`. Otherwise, if the sentence is a duplicate, the previously augmented sentence is appended again to `aug_list`.

Finally, a DataFrame containing the augmented sentences is created, comprising the same number of tuples as the input DataFrame.

Listing 5.7: Function augment_context_words - Data Augmentation

```
1  import nlpaug.augmenter.word as naw
2  import pandas as pd
3
4  # Function that takes a df and returns a df with augmented sentences
5  def augment_context_words(df):
6      aug = naw.ContextualWordEmbsAug(
7      model_path='dbmdz/bert-base-german-cased', action="insert")
8
9      # Taking just the sentence column drom df
10     df_s = df.drop(columns=["aspect", "sentiment"])
```

```
11
12      # Create a Series of booleans (dupl) saying whether a sentence is a
            duplicate or not
13      dupl = df_s["sentence"].duplicated(keep='first')
14      dupl.name = 'Duplicates'
15      # Using Series.values.tolist()
16      dupl_list = dupl.values.tolist()
17
18      # Using Series.values.tolist()
19      sen_list = df_s["sentence"].values.tolist()
20
21      # New empty df for the augmented sentences
22      aug_list = []
23
24      # A for loop that iterates over the 2 lists of sentences and
            duplicates
25      # A df of the augmented sentences is returned
26      # If a sentence is a duplicate --> same augmented sentence is
            created for this sentence
27      for x, y in zip(sen_list, dupl_list):
28          if y == False:
29              aug_sen = aug.augment(x)
30              aug_list.append(aug_sen)
31          else:
32              aug_list.append(aug_sen)
33
34      return pd.DataFrame(aug_list)
```

Figures 5.14 and 5.15 illustrate the augmentation of a single sentence using different actions with the `ContextualWordEmbsAug` module: once with the action parameter set to "substitute", and once with it set to "insert".

```
Original:
Doch bis jetzt ist die technische Ausstattung nicht geboten und auch keine Lernplätze oder ausreichend große Aufenthaltsräume.
Augmented Text:
['Doch nach 2004 wurde die technische Modernisierung nicht geboten und oft keine Decken oder ausreichend große Aufenthaltsräum
e.']
```

Figure 5.14: Example of NLPAug's Contextual Word Embeddings - Substitutions

```
Original:
Doch bis jetzt ist die technische Ausstattung nicht geboten und auch keine Lernplätze oder ausreichend große Aufenthaltsräume.
Augmented Text:
['Doch spätestens bis Weihnachten jetzt ist die technische Ausstattung nicht so geboten belegt und auch keine geeigneten Lernpl
ätze oder ausreichend große helle Aufenthaltsräume.']
```

Figure 5.15: Example of NLPAug's Contextual Word Embeddings - Insertions

The function augment_context_words was applied to each of the three sentiment classes. Following this, the resulting augmented DataFrames had the corresponding values for the attributes "aspect" and "sentiment added from the original data. Subsequently, the augmented DataFrames were merged with the DataFrames containing the original sentences. Finally, as done previously before oversampling the data, one large DataFrame merging all three sentiment classes was created, resulting in a total of 12,504 examples.

As mentioned earlier, the data was not lemmatized or stemmed, nor were stop words removed. However, at this stage, it was considered good practice to remove symbols to prevent potential errors during model training. Listing 5.8 illustrates a function that expects a pandas `DataFrame` and returns it cleaned of any characters that do not match the regular expression "a-zA-Z0-9äöüÄÖÜß!?,: \n.".

Listing 5.8: Function to Remove Symbols from DataFrame

```
1 import re
2
3 def clean_dataframe_from_symbols(df):
4     df_updated = df.replace(to_replace ='[^a-zA-Z0-9äöüÄÖÜß!?,: \n\.]',
          value = '', regex = True)
5
6     return df_updated
```

Having created a new training dataset, the next step was to further transform it into the format required by PyABSA's `Trainer` function. Additionally, the classic train-test split of the dataset was performed, which is essential for the training process. These steps are covered in detail in the next section.

5.3.4 Training Data Preparation

As mentioned earlier in Section 5.3, the training data needed to be converted into a format specifically required by PyABSA. First, the augmented training data is split into three DataFrames: one for training, one for validation, and one for testing, all of which will be necessary for PyABSA's `Trainer` function. Subsequently, each dataset is converted into the APC (Aspect Polarity Classification) format, before finally transforming the data from APC format to the ATEPC (Aspect Term Extraction Polarity Classification) format. The latter transformation could be easily accomplished using a function provided by PyABSA. However, the initial conversion to APC format needed to be developed by the user. It was decided to split the data first and then transform it into the APC format.

In the following, the procedure of the train-test split is explained, before outlining the conversion of the data format.

Listing 5.9 presents the function `split_stratified_into_train_val_test`. It is adapted from StackOverflow (2016a) and used to divide the dataset into three subsets. It takes six parameters: `df_input` for the DataFrame to be split, `stratify_colname` for the column name used for stratification, `frac_train`, `frac_val`, and `frac_test` for the split proportions, and `random_state` to control the randomness of the split.

To perform the splitting, the function `train_test_split` from the `sklearn` library is used. Since it splits data into only two subsets, it is executed twice: once to split the DataFrame into a training set and a temporary set (lines 18-23), and again to split the temporary set into a validation set and a test set (27-32). Although the dataset was already balanced after data augmentation, it was decided to use stratification as best practice in classification tasks. The `stratify` parameter in `sklearn`'s `train_test_split` function ensures a balanced label distribution in all three subsets.

To prevent potential errors due to incorrect inputs, two `if` statements were added. The first, in line 7, ensures that the fractions representing the train, validation, and test splits sum up to 1.0 (100%). The second, in line 11, verifies that the column specified for stratification is indeed present in the DataFrame. Furthermore, line 34 checks that the length of the resulting three subsets sum up to the length of the input DataFrame. Eventually, the function returns the three subsets. Lines 39-41 provide an example of how the function is called, using an 80-10-10 split: 80 percent for the training subset, and 10 percent each for the validation and test subsets. The input `df_clean` reflects the dataset resulting from data annotation and augmentation (12,504 examples). The argument `"sentiment"` indicates the label column.

Ultimately, the three subsets were saved locally as Excel files. In addition to the 80-10-10 split, the training data was also prepared in 70-15-15 and 60-20-20 splits.

Listing 5.9: Function Used for the Train-Test Split. Adapted from StackOverflow (2016a)

```
1  from sklearn.model_selection import train_t56est_split
2
3  def split_stratified_into_train_val_test(df_input, stratify_colname='y',
4                                           frac_train=0.6, frac_val=0.15,
                                                frac_test=0.25,
5                                           random_state=None):
6
7      if frac_train + frac_val + frac_test != 1.0:
8          raise ValueError('fractions %f, %f, %f do not add up to 1.0' % \
9                           (frac_train, frac_val, frac_test))
10
11     if stratify_colname not in df_input.columns:
12         raise ValueError('%s is not a column in the dataframe' %
               (stratify_colname))
13
14     X = df_input # Contains all columns.
15     y = df_input[[stratify_colname]] # Dataframe of just the column on
           which to stratify.
16
17     # Split original dataframe into train and temp dataframes.
18     df_train, df_temp,
19     y_train, y_temp = train_test_split(X,
20                                        y,
21                                        stratify=y,
22                                        test_size=(1.0 - frac_train),
23                                        random_state=random_state)
24
25     # Split the temp dataframe into val and test dataframes.
26     relative_frac_test = frac_test / (frac_val + frac_test)
27     df_val, df_test,
28     y_val, y_test = train_test_split(df_temp,
29                                        y_temp,
30                                        stratify=y_temp,
31                                        test_size=relative_frac_test,
32                                        random_state=random_state)
```

```
33
34      assert len(df_input) == len(df_train) + len(df_val) + len(df_test)
35
36      return df_train, df_val, df_test
37
38  # Splitting df with ratio 80/10/10
39  df_train, df_val, df_test =
        split_stratified_into_train_val_test(df_clean, "sentiment", 0.8,
        0.10, 0.10)
```

As a next step, the three datasets needed to be converted into the APC format. To achieve this, the function `transform_to_apc_format`, illustrated in Listing 5.10 was implemented. It takes a `string` value, representing the Excel file path of one of the three subsets. Eventually, it returns a `string` value representing each row of the Excel file in the required APC format. This is done by applying pandas' `iterrows` function.

Listing 5.10: Function Used for Conversion to APC Format

```
1  def transform_to_apc_format(path: str):
2      df = pd.read_excel(path)
3      #df = df.drop(columns=['Unnamed: 0', 'Unnamed: 1', 'Unnamed: 2'])
4
5      df = df.reset_index()  # make sure indexes pair with number of rows
6
7      # define a string variable that we will feed the result data
8      res=''
9
10     # looping through each row of the df and create the needed format
11     for index, row in df.iterrows():
12         asp = str(row['aspect'])
13         sen = str(row['sentence']).replace(asp, '$T$')
14         res = res + sen + '\n'
15         res = res + asp + '\n'
16         sent = str(row['sentiment'])
17         res = res + sent + '\n'
18
19     return res
```

Listing 5.11 depicts two steps: the conversion of the three splits into APC format and the saving of each of the three results into text files. The latter reflects a requirement of PyABSA's function `convert_apc_set_to_atepc_set`.

Listing 5.11: Conversion of the three Splits into APC Format and Saving as Text File

```
1  ####### Transform the 3 Excel Splits each into APC-Format
2  path_train = 'C:/Users/YOUR_USER/Documents/Prepare Train
       Dataset/Excel_Dataframes_Unis/Clean Split
       Data/80_10_10/Train_Set.xlsx'
3  path_val = 'C:/Users/YOUR_USER/Documents/Prepare Train
       Dataset/Excel_Dataframes_Unis/Clean Split Data/80_10_10/Val_Set.xlsx'
```

```
path_test = 'C:/Users/YOUR_USER/Documents/Prepare Train
    Dataset/Excel_Dataframes_Unis/Clean Split Data/80_10_10/Test_Set.xlsx'

train_apc = transform_to_apc_format(path_train)
val_apc = transform_to_apc_format(path_val)
test_apc = transform_to_apc_format(path_test)

####### write the result to a new text file:
#open text file
text_file = open("C:/Users/YOUR_USER/Documents/Prepare Train
    Dataset/Excel_Dataframes_Unis/Clean Split
    Data/80_10_10/datasets/apc_datasets/train.apc.dataset.txt", "w")
#write string to file
text_file.write(train_apc)
#close file
text_file.close()

text_file = open("C:/Users/YOUR_USER/Documents/Prepare Train
    Dataset/Excel_Dataframes_Unis/Clean Split
    Data/80_10_10/datasets/apc_datasets/val.apc.dataset.txt", "w")
#write string to file
text_file.write(val_apc)
#close file
text_file.close()

text_file = open("C:/Users/YOUR_USER/Documents/Prepare Train
    Dataset/Excel_Dataframes_Unis/Clean Split
    Data/80_10_10/datasets/apc_datasets/test.apc.dataset.txt", "w")
#write string to file
text_file.write(test_apc)
#close file
text_file.close()
```

Figure 5.16 illustrates examples of the resulting APC format. For instance, the first sentence, "Die Professoren hingegen sind alle mit Herz bei der Sache." (ENG: "The professors, on the other hand, are all passionate about their work.") includes the annotated aspect term "Professoren" with a sentiment label "positive". In APC format, the first line shows the sentence where the aspect term is replaced with "T". The second and third lines denote the aspect term and its sentiment, respectively.

Subsequently, the three text files were transformed by using PyABSA's function convert_apc_set_to_atepc_set. In Listing 5.12, the function is used from Zheng (2024). PyABSA requires the text files to be named in the following format: train.apc.dataset.txt, val.apc.dataset.txt, and test.apc.dataset.txt.

Listing 5.12: Conversion of the three Text Files from APC Format to ATEPC Format. Adapted from Zheng (2024)

```
from pyabsa import convert_apc_set_to_atepc_set

```

```
train_atepc =
    convert_apc_set_to_atepc_set('C:/Users/YOUR_USER/Documents/Prepare
    Train Dataset/Excel_Dataframes_Unis/Clean Split
    Data/80_10_10/datasets/apc_datasets/train.apc.dataset.txt')
val_atepc =
    convert_apc_set_to_atepc_set('C:/Users/YOUR_USER/Documents/Prepare
    Train Dataset/Excel_Dataframes_Unis/Clean Split
    Data/80_10_10/datasets/apc_datasets/val.apc.dataset.txt')
test_atepc =
    convert_apc_set_to_atepc_set('C:/Users/YOUR_USER/Documents/Prepare
    Train Dataset/Excel_Dataframes_Unis/Clean Split
    Data/80_10_10/datasets/apc_datasets/test.apc.dataset.txt')
```

As a result, three ATEPC files were saved locally. Figure 5.17 illustrates the first sentence from 5.16 in ATEPC format. During the conversion of the text files, the function appends ".atepc" to their names (e.g., train.apc.dataset.txt becomes train.apc.dataset.txt.atepc).

```
Die $T$ hingegen sind alle mit Herz oben bei der Sache.
Professoren
positive
Außerdem musste durch $T$ fast alles so gut wie selbst beigebracht werden.
digital Lehre
negative
Allerdings hätte ich mir zumindest für zu Beginn gern mehr politisch sinnvolle eigene $T$ gewünscht.
Inhalte
negative
Google Distance Corporate Learning, ferner Online Business Prüfungen, also Desinfektionsspender, Drucker usw.
Online Prüfungen
neutral
Vorlesungen waren meistens nicht von der unterschiedlich selben Qualität  $T$ fielen aber relativ oft untereinander gleich schwer.
Prüfungen
neutral
Spannende $T$.
Praxisprojekte
positive
Es gab sowohl positiv als auch negativ gesehen $T$, die ich nie vergessen werde.
Vorlesungen
neutral
Man kann sich also zb immer eine $T$ ausborgen.
Kamera
positive
Meisten der $T$ wurde auch online digital angeboten.
Vorlesung
neutral
Die TU Wien bietet uns eine Möglichkeit $T$ aus am Unterricht teilzunehmen.
von zuhause
positive
$T$ auch gut umgesetzt, sehr fair.
Prüfungen
positive
Zudem ist die $T$ mitten in der Stadt und man hat auch viele Möglichkeiten in den Pausen die Stadt zu erkunden.
Hochschule
positive
An der MdW mussten wir stets getestet sein und zusätzlich Masken tragen außer im künstlerischen $T$.
Einzelunterricht
neutral
Indem alles noch $T$ für Studierende wird.
unorganisierter
negative
```

Figure 5.16: Example Sentences in APC Format

Having prepared the training data needed to train a custom model in PyABSA, the next section delves into the training process of the ABSA model, using the three files in ATEPC format.

5.4 Modeling

As an initial step in the modeling process, a configuration setup had to be imported from PyABSA. The configuration settings cover various aspects essential for training and evaluation. For both usage and training of ABSA models in PyABSA, the module `AspectTermExtraction` needs to be imported. Listing 5.13 depicts the code for preparing the configuration. By using PyABSA's

```
Die O -100
Professoren B-ASP positive
hingegen O -100
sind O -100
alle O -100
mit O -100
Herz O -100
bei O -100
der O -100
Sache O -100
. O -100
```

Figure 5.17: Example of one Sentence in ATEPC Format

`ATEPCConfigManager`, different configuration setups can be initialized. For instance, Listing 5.13 presents code adapted from Zheng (2024), showcasing a preconfigured set of parameters for training an ABSA model that supports multiple languages. Furthermore, in line 6 the base model is defined which is to be further trained with the created training data. In this case, the PyABSA model "FAST_LCF_ATEPC" is used, which is one of the library's models fine-tuned on specific ABSA tasks. It can be initialized by calling the class `ATEPCModelList`.

Listing 5.13: PyABSA Configuration for Training. Adapted from Zheng (2024)

```
1 from pyabsa import AspectTermExtraction as ATEPC
2
3 config = (
4     ATEPC.ATEPCConfigManager.get_atepc_config_multilingual()
5 )  # this config contains 'pretrained_bert', it is based on pretrained
    models
6 config.model = ATEPC.ATEPCModelList.FAST_LCF_ATEPC # improved version of
    LCF-ATEPC
```

Listing 5.14 provides an example illustrating the customization of predefined configurations. It demonstrates the configuration employed for the initial model training in this study, where the only parameter varied across different model trainings was "pretrained_bert". For the initial training attempt, the pretrained BERT model "yangheng/deberta-v3-base-absa" was utilized, which is a PyABSA model based on BERT and fine-tuned on ABSA. In concrete, "deberta-v3-base" refers to the BERT variant "DeBERTa", which was used by PyABSA's author for further fine-tuning. DeBERTa stands for Decoding-enhanced BERT for Disentangled Attention, a version of BERT improved by innovations in both decoding and attention mechansim (He et al., 2021). Unlike the `pretrained_bert` parameter, which specifies the base BERT model used as the backbone, the `model` parameter—in this case set to FAST_LCF_ATEPC—represents the specific architecture fine-tuned for the task of ATEPC (Aspect Term Extraction and Polarity Classification).

In the following, the rest of the configuration parameters depicted in Listing 5.14 are explained. The parameter `config.evaluate_begin = 0` specifies that evaluation should start from the first epoch, ensuring early monitoring of performance. The command `max_seq_len = 128` sets the maximum length for input sequences, standardizing them to 128 tokens, while `batch_size = 16` determines that during training, 16 samples will be processed in each batch. Logging frequency is specified by `log_step = -1`, where a negative value usually implies default settings or no logging.

L2 regularization is applied with `l2reg = 1e-8` to prevent overfitting. The model is trained over `num_epoch = 20` epochs, referring to 20 passes through the training data. For reproducibility, `seed = 42` is set to initialize the random number generator in a specific state. Automatic Mixed Precision (AMP) is disabled with `use_amp = False`, meaning only 32-bit floating-point types are used. To improve efficiency, `cache_dataset = True` allows caching of the dataset for quicker subsequent runs. Finally, `cross_validate_fold = -1` indicates that cross-validation is not used, focusing on a single train-test split for model training and evaluation (H. Yang et al., 2023). Since the amount of generated training data reflects thousands of examples, it was assumed that it is large enough. As a result, cross-validation was not applied for training the model.

Listing 5.14: Futher Configuration Parameters for Training in PyABSA. Adapted from Zheng (2024)

```
config.evaluate_begin = 0
config.max_seq_len = 128
config.batch_size = 16
config.pretrained_bert = 'yangheng/deberta-v3-base-absa'
config.log_step = -1
config.l2reg = 1e-8
config.num_epoch = 20
config.seed = 42
config.use_bert_spc = True
config.use_amp = False
config.cache_dataset = True
config.cross_validate_fold = -1
```

Finally, Listing 5.15 illustrates the code used for training the ABSA model. In line 1, the variable `my_dataset` is defined to represent the directory path where the three training data files are stored. To train a model for the ATEPC task, an instance of the PyABSA class `ATEPCTrainer` must be instantiated, as shown in line 3. Several parameters are required to configure the trainer: `config` specifies the earlier described configuration, `dataset` expects a directory path containing the three subsets for training or a PyABSA `DataSetObject`, `from_checkpoint` is optional and takes the name of a pretrained checkpoint, `auto_device` determines whether to use CUDA or CPU, `checkpoint_save_mode` controls how the model is saved, and `path_to_save` specifies the directory where the model should be saved. Four files are automatically saved upon training completion: an .args file storing the command-line arguments or training parameters, a file containing the tokenizer information used during training, a configuration file detailing the settings used for the model and training process, and a state dictionary file containing the learned parameters of the model.

For the initial model trained, the checkpoint "multilingual" was utilized, as indicated in line 6 of Listing 5.15. However, subsequent models created during this study were trained without relying on a checkpoint. This decision stemmed from a reported bug in PyABSA's multilingual checkpoint, as acknowledged by its author. Given that other checkpoints were not suitable for the German language specific to this use case, training proceeded without utilizing the optional checkpoint parameter. After consulting with the author of PyABSA, this approach was recommended. As an argument for `dataset`, using a directory path was preferred over defining the PyABSA `DataSetObject`, and for saving the model, the State Dict mode was employed.

Listing 5.15: Training Procedure in PyABSA. Adapted from Zheng (2024)

```
my_dataset = 'C:/Users/YOUR_USER/Documents/Prepare Train
    Dataset/Modeling/integrated_datasets/atepc_datasets/177.University'

trainer = ATEPC.ATEPCTrainer(
    config=config,
    dataset=my_dataset,
    from_checkpoint="multilingual",  # optional - in case of training
        resuming from a pretrained checkpoint, pass the name of the
        checkpoint
    auto_device=DeviceTypeOption.AUTO,  # use cuda if available
    checkpoint_save_mode=ModelSaveOption.SAVE_MODEL_STATE_DICT,  # save
        only state dict, instead of the model as a whole
    path_to_save='C:/Users/YOUR_USER/Documents/Prepare Train
        Dataset/Modeling/saved_trained_models'
    )
```

Eventually, several models were trained using various splits of training data and different pretrained BERT models in the configuration.

Figure 5.18 summarizes the models trained, excluding the initial model trained with a checkpoint. The first models utilized the "bert-base-multilingual-uncased" pretrained BERT model, trained on three different data splits. Additionally, experiments were conducted using varying amounts of training data: some models were trained on datasets before augmentation, while others utilized the fully augmented dataset.

Following consultation with PyABSA's author, additional modeling attempts involved different pretrained BERT models. Subsequently, two more models were trained using an 80-10-10 split: one employing the "dbmdz/bert-base-german-cased" model, specifically designed for the German language and sourced from the Bavarian State Library; the other using the "microsoft/mdeberta-v3-base" model, recommended by PyABSA's author. Moreover, due to superior performance observed with the 80-10-10 split, the other two splits were excluded when training the last two models.

Pretrained BERT	Training Data Split
bert-base-multilingual-uncased	80-10-10
bert-base-multilingual-uncased	70-15-15
bert-base-multilingual-uncased	60-20-20
dbmdz/bert-base-german-cased	80-10-10
microsoft/mdeberta-v3-base	80-10-10

Figure 5.18: ABSA Models Trained in PyABSA

5.5 Model Evaluation

This section is divided into two parts: the first part describes metrics evaluated within PyABSA training, while the second part presents an alternative evaluation approach.

5.5.1 PyABSA Evaluation - Metric Visualizer

PyABSA has its own evaluation functionality during the training process, referred to as "metric_visualizer". Once the training completes, it generates a visualization displaying three key metrics: APC_ACC, APC_F1, and ATE_F1. These metrics assess the accuracy and F1 scores for both PyABSA tasks: APC (Aspect Polarity Classification) and ATE (Aspect Term Extraction).

Figures 5.19 and 5.20 depict the visualizations generated by PyABSA's evaluation after training, both before and after data augmentation. In both cases, the pretrained BERT parameter was set to "bert-base-multilingual-uncased" with a training split of 80-10-10.

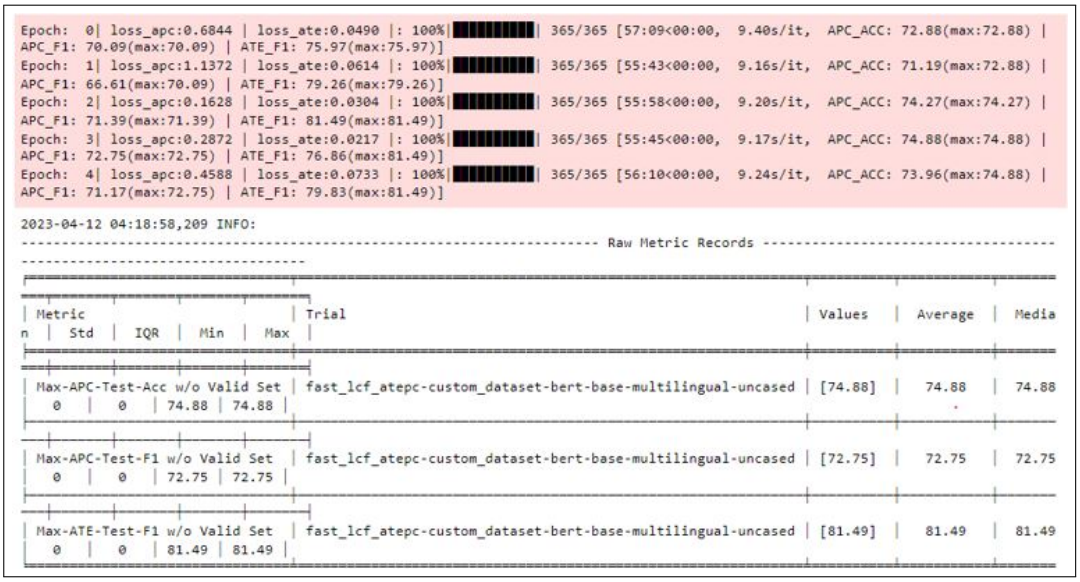


Figure 5.19: PyABSA's Metric Visualizer - Before Data Augmentation

According to PyABSA metrics, the model trained with data before augmentation (5.19) achieved an accuracy of 74.88% for the APC task during the best epoch, with respective scores of 72.75% for APC_F1 and 81.49% for ATE_F1. In contrast, the model trained with augmented data (5.20) reached a peak accuracy of 85.8% for the APC task, along with 85.73% for APC_F1 and 81.92% for ATE_F1.

These figures demonstrate improvements in both metrics following data augmentation. Therefore, two models using "dbmdz/bert-base-german-cased" and "microsoft/mdeberta-v3-base" as pretrained BERT were also trained with the augmented datasets. The PyABSA metrics calculated during the training of both models are depicted in Figure 5.21. The model based on Microsoft's DeBERTa demonstrated better metrics compared to the model based on "dbmdz/bert-base-german-cased".

However, during tests with these models, the observed metrics did not align with those reported by PyABSA's metric_visualizer. This discrepancy indicated a poorer perception of the models' performance based on the test results. An example of the test results is illustrated in Figure

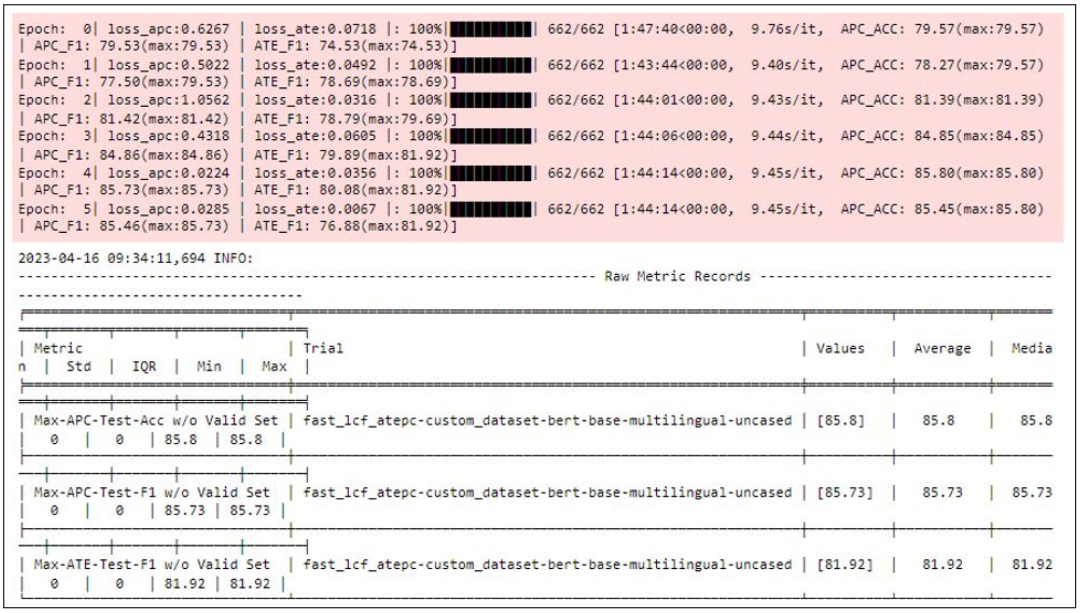


Figure 5.20: PyABSA's Metric Visualizer - After Data Augmentation

Pretrained BERT	APC ACC Max	APC F1 Max	ATE F1 Max
dbmdz/bert-base-german-cased	71.43%	68.28%	41.82%
microsoft/mdeberta-v3-base	86.67%	86.69%	64.86%

Figure 5.21: PyABSA Metrics

5.22, showcasing the first 35 sentences, sourced from PH Steiermark ratings, along with their detected aspects and sentiments. Listing 5.16 provides the code for how testing was conducted. First, in line 1, the model trained with "microsoft/mdeberta-v3-base" as the pretrained BERT is initialized. This was achieved by calling PyABSA's `AspectExtractor` object, which originates from the PyABSA module `AspectTermExtraction`. The string passed to `AspectExtractor` represents the name of the directory path that contains the four model files generated immediately after training in PyABSA. Then, sentences were extracted from reviews collected in Chapter 4, sourced from PH Steiermark. This was done using the previously described `create_sentences` function from Listing 5.4. Line 9 takes only the sentences retrieved from `df_ph_steiermark` and converts their values into a Python list. By applying the `predict` function inherited by the model to each sentence, the model detects aspects and their corresponding sentiments. In line 22, the results for each sentence were collected in a list named `results` and subsequently stored in both a JSON file and an Excel file.

Listing 5.16: Testing the Fine-Tuned DeBERTa Model with sentences rating PH Steiermark

```
1 aspect_extractor_mdeberta =
      ATEPC.AspectExtractor('DIRECTORY\_PATH\_TRAINED\_MODEL',
2                                         auto_device=True,  # False
                                              means load model on CPU
3                                         cal_perplexity=True,
4                                         )
5
```

	sentence	aspect	sentiment
2	Zu Beginn des Semesters waren der Stundenplan sehr voll .	['Stundenplan']	['neutral']
3	Einige Nachtschichten mussten eingelegt werden und es war auch sehr überfordernd , da es auch das erste Semester war .	['Nachtschichten']	['neutral']
4	Je länger das Semester jedoch andauerte , desto eingespielter lief alles ab und einige LVs waren schon früher fertig .	['LVs']	['neutral']
5	In vielen Lehrveranstaltungen werden Inhalte gelehrt , die einem f ü r den zuk ü nftigen Job nicht viel bringen ...	['Inhalte']	['negative']
6	Ich habe diese Rückmeldung auch schon von vielen anderen Studierenden bekommen .	['Rückmeldung']	['positive']
7	Ebenso bekommen wir teils sehr oft Arbeitsaufträge , die reine Beschäftigungstherapie sind und kaum einen Nährwert für einen haben .	['Arbeitsaufträge']	['negative']
8	Zu wenig materialien und räume zum kreativen Ausleben zur Verfügung .	[]	[]
9	Die Lehrveranstaltungen sind besser geworden vom Inhalt her leider ist es aber noch ausbaufähig bezüglich der Materialen .	['Materialen']	['negative']
10	Die Professoren / innen sind meist sehr nett .	['Professoren']	['positive']
11	Jedoch lernt man wenig Grundlagen über verschiedene Techniken .	['Grundlagen']	['negative']
12	Die DozentInnen und das Team der Organisation gibt sich wirklich sehr viel Mühe , sodass man eine schöne Zeit hat .	['Organisation']	['positive']
13	Es gibt neben den LVs viele sehr interessante Wahlfächer , indenen man über sich hinauswachsen kann und sich mit anderen Studierenden vernetzen kann !	['LVs']	['positive']
14	Es gibt mehrere Sitz - und Arbeitsmöglichkeiten , die individuell ausgestattet sind .	['Sitz']	['positive']
15	Egal ob du alleine , zu zweit oder in der Gruppe arbeiten möchtest .	['Gruppe']	['neutral']
16	Es ist für jeden etwas dabei .	[]	[]
17	Die Lehrveranstaltungen reichen von sehr anspruchsvoll bis weniger anspruchsvoll .	['Lehrveranstaltungen']	['neutral']
18	Man muss , jedoch sagen , dass wirklich jede / r DozentIn stets bemüht ist und auf unsre eigenen Bedürfnisse immer eingeht .	[]	[]
19	Die Praktika in den Schulen machen sehr viel Spaß und man kann vieles anwenden , was man in der Hochschule gelernt hat .	['Praktika']	['positive']
20	Die Prüfungen und Arbeitsaufträge haben einen passenden Zeitaufwand nötig .	[]	[]
21	Die Anwesenheitspflicht in den meisten Fächern finde ich gut , weil ich so hin gehen muss .	[]	[]
22	Ich finde das Studium Primarstufe ist sehr gut , jedoch kommen einige Inhalte vor die überflüssig sind und auch etwas unnötig , da man diese nicht in der Zukunft brauchen wird .	['Inhalte']	['negative']
23	Das weiß ich daher weil ich bereits in einer Volksschule arbeite und mit den Inhalten aus den LVs bichts anfangen kann .	['LVs']	['neutral']
24	Trotzdem würde ich es aber weiterempfehlen , da es ein Studium für einen sehr wichtigen Beruf ist !	[]	[]
25	Bis jetzt ist das Studium super .	[]	[]
26	Tolle Lehrveranstaltungen , super Lehrpersonen alles klasse .	['Lehrveranstaltungen']	['positive']
27	Würde mein Studium sofort weiterempfehlen , vor allem an der Ph Steiermark .	[]	[]
28	Auch die Verknüpfung mit den anderen Universitäten und Hochschulen läuft hervorragend .	['Verknüpfung']	['positive']
29	Das Studium ist für einen Neustudierenden sehr umfangreich da es einige Fächer beinhaltet , was oft sehr überwältigen ist .	['Fächer']	['neutral']
30	Es sind jedoch super spannende Lehrveranstaltungen dabei .	['Lehrveranstaltungen']	['positive']
31	Die Fächer werden jedoch ab dem 3 . Semester reduziert , was einem Hoffnung gibt .	['Fächer']	['neutral']
32	Die PH ist super schön , großflächig und hat eine eigene Grundschule dort bereits inbegriffen , was super spannend ist , da man täglich Kinder sieht und hört , was natürlich einen Einblick in das zukünftige Berufsleben gibt .	['PH']	['positive']
33	Ich habe im Bachelorstudium schon gearbeitet , nachdem aber der Master verpflichtend ist , habe ich diesen direkt begonnen .	[]	[]
34	Daneben habe ich mit halber Lehrverpflichtung unterrichtet .	['Lehrverpflichtung']	['neutral']
35	Mir kamen die Kurse meist nicht relevant vor , lieber würde ich mich auf konkrete Didaktik fokussieren und Inhalte aus dem Bachelor vertiefen , damit ich das Gefühl habe , in den Fächern tatsächlich didaktische Expertise zu haben .	['Kurse', 'Inhalte']	['negative', 'neutral']

Figure 5.22: PH Steiermark Test Results Using DeBERTa - First 35 Sentences

```
ph_steiermark =
    pd.read_excel('C:/Users/YOUR_USER/Documents/Scraping/Data/
Excel/PH_Steiermark.xlsx')
df_ph_steiermark = create_sentences(ph_steiermark)
ph_steiermark_sentences = df_ph_steiermark['sentence'].tolist()

import json

# Initialize a list to store results
results = []

for sentence in ph_steiermark_sentences:
    result = aspect_extractor_mdeberta.predict(sentence,
                            save_result=True,
                            print_result=True,  # print the result
                            ignore_error=True,  # ignore the error when the
                                model cannot predict the input
                            )
    results.append(result)
    #print(result)  # Print the result to the console

# Save all results to a JSON file
with open('test_results.json', 'w') as f:
    json.dump(results, f, indent=4)

# Convert results to a pandas DataFrame
df_results = pd.DataFrame(results)

# Save results to an Excel file
excel_file = "ABSA_Test_results_PH_Steiermark.xlsx"
df_results.to_excel(excel_file, index=False)
```

Although acceptable PyABSA metrics were achieved during training, the perceived performance of the models prompted additional evaluations beyond PyABSA's `metric_visualizer`, as described in the next section.

5.5.2 Further Evaluation

To conduct an independent evaluation, several Python functions were developed. This evaluation utilized the model based on "microsoft/mdeberta-v3-base", chosen for its superior performance observed during testing. Listing 5.17 illustrates the function `my_own_evaluate`, which requires an initialized ABSA model stored in the variable `aspect_extractor`, a pandas DataFrame containing labeled test sentences from the training data, and an Excel file path for saving the resulting predictions. The function outputs both the test DataFrame `test_df` and predictions stored in a DataFrame `predictions_df`.

In line 7, the function `extract_aspect_polarity` is used to capture predictions for each sentence from `test_df` into lists. The specific functionality of this function will be further explained in Chapter 6. If the model fails to identify any aspect, an empty string is recorded for both aspect and sentiment in `predictions_df` (first if statement). Conversely, if the model successfully detects an aspect term, all predicted values are appended to `predictions_df` (second if statement).

Listing 5.17: Function to create a DataFrame with predictions

```
1  def my_own_evaluate(aspect_extractor, test_df, path_to_save_pred_df):
2      predictions_df = pd.DataFrame(columns=['sentence', 'aspect',
           'sentiment'])
3
4      for index, row in test_df.iterrows():
5          sample = row['sentence']
6
7          aspects, sentiments, confidences =
               extract_aspect_polarity(sample, aspect_extractor)
8
9          if len(aspects) == 0:
10             predictions_df = predictions_df.append({
11                 'sentence': sample,
12                 'aspect': '',
13                 'sentiment': ''
14             }, ignore_index=True)
15
16         else:
17             for i in range(len(aspects)):
18                 # save predictions in 2nd df
19                 predictions_df = predictions_df.append({
20                     'sentence': sample,
21                     'aspect': aspects[i],
22                     'sentiment': sentiments[i]
23                 }, ignore_index=True)
24
25     predictions_df.to_excel(path_to_save_pred_df)
26
```

```
27      return predictions_df, test_df
```

A second function (see Listing 5.18) was utilized to generate merged DataFrames essential for computing various metrics, including accuracy and recall. This function takes both `test_df` and `predictions_df` from the previous function and returns three distinct DataFrames merged based on them: one merged using an inner join solely on the sentences, another merged on sentences and aspects, and a third merged on sentences, aspects, and sentiments.

The first merged DataFrame is crucial for calculating recall (see Listing 5.19), the second is used to compute the accuracy of the Aspect Term Extraction (ATE) task, and the third is utilized to calculate the accuracy of both ATE and Aspect Polarity Classification (APC) tasks simultaneously.

Listing 5.18: Obtain Merged DataFrames from Test Data and Predictions

```
1  def get_merged_dfs(test_df, pred_df):
2      import pandas as pd
3
4      merged_df_on_sentence_only = pd.merge(test_df, pred_df,
           on=['sentence'], how='inner')
5      merged_df_for_ATE_Task = pd.merge(test_df, pred_df, on=['sentence',
           'aspect'], how='inner')
6      merged_df_for_ATE_APC_Task = pd.merge(test_df, pred_df,
           on=['sentence', 'aspect', 'sentiment'], how='inner')
7
8      if merged_df_for_ATE_Task.empty:
9          print('Keine Uebereinstimmungen gefunden.')
10     else:
11         print('Uebereinstimmungen gefunden.')
12
13     acc_ate = (len(merged_df_for_ATE_Task) / len(test_df)) * 100
14     print(f"Das ist die ATE-Accuracy: {acc_ate} %")
15
16     acc_ate_apc = (len(merged_df_for_ATE_APC_Task) / len(test_df)) * 100
17     print(f"Das ist die ATE_APC-Accuracy: {acc_ate_apc} %")
18
19
20     return merged_df_on_sentence_only, merged_df_for_ATE_Task,
           merged_df_for_ATE_APC_Task
```

Listing 5.19 takes as inputs the test data `test_df` and three merged DataFrames obtained from the previous function. It computes three key metrics: accuracy for the single task ATE, accuracy for the combined task ATE_APC, and recall for the combined task ATE_APC. The metric values obtained are as follows: 40.0% for ATE accuracy, 37.38% for ATE_APC accuracy, and 55.67% for ATE_APC recall.

Listing 5.19: Obtain Metrics: Accuracy for ATE, Accuracy for ATE_APC, and Recall for ATE_APC

```
1  def get_metrics(test_df, merged_df_sen, merged_df_sen_asp,
       merged_df_sen_asp_senti):
2
3      true_pos = len(merged_df_sen_asp)
```

```
    acc_ate = (true_pos / len(test_df)) * 100
    print(f"Das ist die ATE-Accuracy: {acc_ate} %")

    acc_ate_apc = (len(merged_df_sen_asp_senti) / len(test_df)) * 100
    print(f"Das ist die ATE_APC-Accuracy: {acc_ate_apc} %")

    # now for metric recall:
    false_neg = 0

    # Count rows where aspect_y and sentiment_y are NaN
    false_negatives = merged_df_sen[merged_df_sen['aspect_y'].isnull() &
        merged_df_sen['sentiment_y'].isnull()]

    # Number of false negatives
    false_neg = len(false_negatives)

    recall_ate_apc = (true_pos / true_pos + false_neg) * 100
    print(f"Das ist der ATE_APC-Recall: {recall_ate_apc} %")

    return acc_ate, acc_ate_apc, recall_ate_apc
```

Metric	Result
ATE Accuracy	40.0%
ATE_APC Accuracy	37.38%
ATE_APC Recall	55.67%

Figure 5.23: Evaluation Metrics Derived from Own Analysis

Doing an overall comparison between the metrics obtained by PyABSA (refer to 5.20) and those obtained independently (refer to 5.23, the results indicate a significant performance drop when considering the custom metrics, which reflect the perceived performance of the model.

The final section of this chapter outlines the development of a BERT-based classifier essential for categorizing aspect terms into aspect categories. This step was crucial before proceeding to load all ABSA results into a database for comprehensive analysis. In this way, numerous and varied aspect terms can be summarized and ultimately replaced by a fixed number of high-level aspect categories for querying.

5.6 Building an Aspect Category Classifier

To perform aspect category classification, the pretrained BERT model "dbmdz/bert-base-german-cased," originating from the Bavarian State Library, was utilized. While the pretrained model generally understands language and context, it is not aware of the predefined 10 aspect categories and cannot classify the aspect terms without further training. Therefore, it needed to be fine-tuned

for the specific task of multi-label text classification. This was accomplished by adapting a solution from The Artificial Guy (2024). To facilitate the recreation of the classifier development, it was decided to include the adapted code.

A custom training dataset was created using the aspects and categories from the training data, comprising 12,504 examples obtained through data augmentation (see Section 5.3.3). This was achieved by applying one-hot encoding to the target labels for easier processing. Additionally, a BERT model was defined, incorporating a dropout layer and a linear layer. The following listings provide the details of the classifier's development. Section 5.6.1 discusses the preparation of the training data, while Section 5.6.2 focuses on the modeling process.

5.6.1 Training Data Preparation

Listing 5.20 presents code adapted from StackOverflow. Specifically, StackOverflow (2015) was referenced for splitting the training data, while StackOverflow (2016b) was utilized to create one-hot encodings for the aspect categories. Moreover, similar to the training preparation for the ABSA model, the function `train_test_split` from sklearn was used to split the data. The variable `data` represents the training data containing the two attributes aspect and aspect category, while the variable `y` is the target variable, consisting of only the aspect category column. Since the distribution of aspect categories is not balanced, the function is set to split the data by stratifying on the target variable. Additionally, the split is performed to allocate 70% of the data for training and 30% for validation.

In lines 6 and 7, one-hot encodings are generated for the target variable from both the training and validation data, and saved to `X_train_dummies` and `X_test_dummies`, respectively. Then, in lines 10 and 11, both training and validation data are trimmed to include only the aspect terms, before being concatenated with the one-hot encodings in lines 14-15. This procedure replaces each unique aspect category with a binary column indicating the presence (1) or absence (0) of that category. This transforms the multi-label classification problem into a binary classification problem. Furthermore, since the targets were embedded in `X_train` and `X_test`, there was no further usage of the variables `y_train` and `y_test` resulting from the split. Lastly, the hyperparameters used for training are set in lines 22-26.

Listing 5.20: Train-Test Split and One-Hot Encodings. Adapted from StackOverflow (2015) and StackOverflow (2016b).

```
1  # Train Test Split
2  from sklearn.model_selection import train_test_split
3  X_train, X_test, y_train, y_test = train_test_split(data, y, stratify=y,
       test_size=0.3)
4
5  # One-Hot Encoding the Labels
6  X_train_dummies = pd.get_dummies(X_train['aspect category'])
7  X_test_dummies = pd.get_dummies(X_test['aspect category'])
8
9  # Filtering to Aspect Terms
10 X_train = X_train.drop(columns=['aspect category'])
11 X_test = X_test.drop(columns=['aspect category'])
12
```

```
# Joining the Aspect Terms with the One-Hot-Encodings (Aspect Categories)
X_train = pd.concat([X_train, X_train_dummies], axis=1)
X_test = pd.concat([X_test, X_test_dummies], axis=1)

# Changing Names and reset the index
train_df = X_train.reset_index(drop=True)
val_df = X_test.reset_index(drop=True)

# Hyperparameters Used for Data Preparation and Training
MAX_LEN = 256
TRAIN_BATCH_SIZE = 32
VALID_BATCH_SIZE = 32
EPOCHS = 2
LEARNING_RATE = 1e-05
```

Figure 5.24 illustrates one-hot encodings for the target variable before the train-test split.

	Allgemein	Ausstattung	Bibliothek	Campus	Digitales Studieren	Dozenten	Lage	Lehrveranstaltungen	Organisation	Studieninhalte
0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	1	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
12499	0	0	0	0	0	1	0	0	0	0
12500	0	0	0	0	1	0	0	0	0	0
12501	0	0	0	0	0	0	0	0	0	1
12502	0	0	0	0	0	0	0	1	0	0
12503	0	0	0	0	1	0	0	0	0	0

12504 rows × 10 columns

Figure 5.24: One-Hot-Encodings for Aspect Categories

The following Listings 5.21 to 5.27 are adapted from The Artificial Guy (2024), which provides guidance on performing multi-label text classification using BERT and PyTorch. Listing 5.21 represents the class `CustomDataset`, which is essential for handling and preparing data for training and evaluation with the BERT model. The data must be transformed in several steps to match the required input format for the model.

Before the class declaration, both the BERT model and tokenizer are imported from the transformers library. The BERT model is needed later in Section 5.6.2, while the BERT tokenizer is required in this step for data preparation. It is defined using the "dbmdz/bert-base-german-cased" model. Additionally, the variable `target_list` contains the 10 aspect categories, which are necessary within the `CustomDataset` class. Inside the class, the value of an aspect term is saved in the variable `self.title`, representing the features or input data, while the possible values for aspect categories are saved in `self.targets`, representing the labels or target data.

Listing 5.21: Pytorch Class for Transforming a pandas DataFrame for BERT Training. Adapted from The Artificial Guy (2024)

```
from transformers import BertTokenizer, BertModel

```

```
tokenizer = BertTokenizer.from_pretrained('dbmdz/bert-base-german-cased')

target_list = ['Allgemein', 'Ausstattung', 'Bibliothek', 'Campus',
    'Digitales Studieren', 'Dozenten', 'Lage',
                'Lehrveranstaltungen', 'Organisation', 'Studieninhalte']

class CustomDataset(torch.utils.data.Dataset):

    def __init__(self, df, tokenizer, max_len):
        self.tokenizer = tokenizer
        self.df = df
        self.title = df['aspect']
        self.targets = self.df[target_list].values
        self.max_len = max_len

    def __len__(self):
        return len(self.title)

    def __getitem__(self, index):
        title = str(self.title[index])
        title = " ".join(title.split())

        inputs = self.tokenizer.encode_plus(
            title,
            None,
            add_special_tokens=True,
            max_length=self.max_len,
            padding='max_length',
            return_token_type_ids=True,
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt'
        )

        return {
            'input_ids': inputs['input_ids'].flatten(),
            'attention_mask': inputs['attention_mask'].flatten(),
            'token_type_ids': inputs["token_type_ids"].flatten(),
            'targets': torch.FloatTensor(self.targets[index])
        }
```

In Listing 5.22, the two training subsets resulting from Listing 5.20 are converted into two CustomDataset objects. This process requires the training DataFrame, a tokenizer, and the maximum input length, as specified by the __init__ function in the class from Listing 5.21.

Subsequently, both the training and validation datasets undergo further conversion using the DataLoader class from PyTorch. This step ensures that both datasets are managed with specified batch sizes and that the training data is shuffled for each epoch to prevent overfitting. However, to maintain consistent evaluation metrics, the validation data was not shuffled.

Listing 5.22: Conversion of the Training Data. Adapted from The Artificial Guy (2024)

```
train_dataset = CustomDataset(train_df, tokenizer, MAX_LEN)
valid_dataset = CustomDataset(val_df, tokenizer, MAX_LEN)

train_data_loader = torch.utils.data.DataLoader(train_dataset,
    batch_size=TRAIN_BATCH_SIZE, shuffle=True, num_workers=0
)

val_data_loader = torch.utils.data.DataLoader(valid_dataset,
    batch_size=VALID_BATCH_SIZE, shuffle=False, num_workers=0
)
```

5.6.2 Modeling

Having established the prerequisites for the training data, this section focuses on modeling the classifier. Listing 5.23 introduces two essential functions for managing checkpoints during the training process. The load_ckp function is used after training to load a model. It expects a path to a saved checkpoint file and loads its parameters to a model passed to the function. Alongside the model's state dictionary, the function returns the optimizer's state, the epoch of the checkpoint, and the minimum validation loss encountered during training.

Conversely, the save_ckp function plays a crucial role during training. Using `torch.save`, it stores a checkpoint of the model (`state`) at the specified path `checkpoint_path`. If the boolean variable `is_best` is set to True during training, indicating the model has achieved the best performance at that specific epoch, the checkpoint is also copied to the path `best_model_path`, as demonstrated in Listing 5.25.

Listing 5.23: Loading and Saving of Checkpoints. Adapted from The Artificial Guy (2024)

```
def load_ckp(checkpoint_fpath, model, optimizer):
    """
    checkpoint_path: checkpoint path
    model: model that we want to load checkpoint parameters into
    optimizer: optimizer we defined in previous training
    """
    # load check point
    checkpoint = torch.load(checkpoint_fpath)
    # initialize state_dict from checkpoint to model
    model.load_state_dict(checkpoint['state_dict'])
    # initialize optimizer from checkpoint to optimizer
    optimizer.load_state_dict(checkpoint['optimizer'])
    # initialize valid_loss_min from checkpoint to valid_loss_min
    valid_loss_min = checkpoint['valid_loss_min']
    # return model, optimizer, epoch value, min validation loss
    return model, optimizer, checkpoint['epoch'], valid_loss_min

def save_ckp(state, is_best, checkpoint_path, best_model_path):
    import shutil
    """
    state: checkpoint we want to save
```

```
    is_best: is this the best checkpoint; min validation loss
    checkpoint_path: path to save checkpoint
    best_model_path: path to save best model
    """
    f_path = checkpoint_path
    # save checkpoint data to the path given, checkpoint_path
    torch.save(state, f_path)
    # if it is a best model, min validation loss
    if is_best:
        best_fpath = best_model_path
        # copy that checkpoint file to best path given, best_model_path
        shutil.copyfile(f_path, best_fpath)
```

Listing 5.24 introduces the setup of a BERT-based neural network model designed for multi-label text classification. In line 1, the device the model will be loaded on is defined. Notably, within `BERTClass`, lines 6 and 10 are key: `self.bert_model` is initialized using the pretrained BERT model "dbmdz/bert-base-german-cased" via the `BertModel` module (imported in 5.21), while `self.linear` specifies the linear layer `torch.nn.Linear` used for classification. In addition, 10 is set as number of output classes, corresponding to the predefined aspect categories.

The conclusion of Listing 5.24 introduces the definition of the loss function, crucial during the training phase. As the task is transformed into a binary classification problem using one-hot encodings, the binary cross-entropy loss (`BCEWithLogitsLoss`) is employed.

Listing 5.24: Definition of the BERT Class and Loss Function. Adapted from The Artificial Guy (2024)

```
1  device = torch.device('cuda') if torch.cuda.is_available() else
       torch.device('cpu')
2
3  class BERTClass(torch.nn.Module):
4      def __init__(self):
5          super(BERTClass, self).__init__()
6          self.bert_model =
               BertModel.from_pretrained('dbmdz/bert-base-german-cased',
               return_dict=True)
7          self.dropout = torch.nn.Dropout(0.3)
8          # Setting the Linear Layer comprising a 10-dimensional output
               layer
9          # Parameter 10 Refers to 10 aspect categories (targets)
10         self.linear = torch.nn.Linear(768, 10)
11
12     def forward(self, input_ids, attn_mask, token_type_ids):
13         output = self.bert_model(
14             input_ids,
15             attention_mask=attn_mask,
16             token_type_ids=token_type_ids
17         )
18         output_dropout = self.dropout(output.pooler_output)
19         output = self.linear(output_dropout)
20         return output
```

```
21
22 def loss_fn(outputs, targets):
23     return torch.nn.BCEWithLogitsLoss()(outputs, targets)
```

The core logic of training is implemented in the `train_model` function, illustrated in Listing 5.25. It was needed for training and validating the model for several epochs. Furthermore, it tracks the validation loss (using the loss function) to monitor model performance and automatically saves the best model using the save_ckp function from Listing 5.23.

The parameters expected by the `train_model` function include the number of epochs to train the model (n_epochs), DataLoaders for the training and validation datasets (`training_loader`, `validation_loader`), the BERT-based model to be trained (`model`), and the optimizer used for training (`optimizer`). Lastly, checkpoint_path is the path to save checkpoint files, while `best_model_path` expects the path to save the best model checkpoint.

The function begins by initializing a tracker, `valid_loss_min`, which keeps track of the minimum validation loss encountered during training. In line 11, a for loop is defined with n_epochs as number of iterations. This loop lasts until the end of the function, before `train_model` ultimately returns the trained model. Inside this loop, there are two distinct blocks: one for training and one for validation.

During the training block, the model parameters are adjusted using backpropagation, where gradients are computed and weights are updated to minimize the training loss. This process is crucial for improving the model's ability to make predictions on the training data.

Conversely, during the validation block, the model's performance is evaluated on unseen validation data. Here, no gradients are computed, and weights are not updated, as indicated by the use of `torch.no_grad()`. This ensures that the model's evaluation on the validation set remains independent of its training, thereby providing an unbiased assessment of its generalization capabilities and guarding against overfitting.

Throughout training, checkpoints of the model are saved to the checkpoint_path, and the best model, based on the validation loss, is also saved to the `best_model_path`. This ensures that the best performing model observed during training is preserved for future use or further evaluation.

The two lists defined before the function, `val_targets` and `val_outputs`, are used to store the true labels and predicted outputs for the validation dataset. While this thesis does not include a detailed evaluation of the developed classifier, and thus these lists are filled but not used, they could be used for calculating evaluation metrics of the trained model.

Listing 5.25: Training Implementation. Adapted from The Artificial Guy (2024)

```
1 val_targets=[]
2 val_outputs=[]
3
4 def train_model(n_epochs, training_loader, validation_loader, model,
5                 optimizer, checkpoint_path, best_model_path):
6
7   # initialize tracker for minimum validation loss
8   valid_loss_min = np.Inf
```

```
9
10
11  for epoch in range(1, n_epochs+1):
12    train_loss = 0
13    valid_loss = 0
14
15    model.train()
16    print('##### Epoch {}: Training Start   #####'.format(epoch))
17    for batch_idx, data in enumerate(training_loader):
18        #print('yyy epoch', batch_idx)
19        ids = data['input_ids'].to(device, dtype = torch.long)
20        mask = data['attention_mask'].to(device, dtype = torch.long)
21        token_type_ids = data['token_type_ids'].to(device, dtype =
              torch.long)
22        targets = data['targets'].to(device, dtype = torch.float)
23
24        outputs = model(ids, mask, token_type_ids)
25
26        optimizer.zero_grad()
27        loss = loss_fn(outputs, targets)
28        #if batch_idx%5000==0:
29         #   print(f'Epoch: {epoch}, Training Loss:  {loss.item()}')
30
31        optimizer.zero_grad()
32        loss.backward()
33        optimizer.step()
34        #print('before loss data in training', loss.item(), train_loss)
35        train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.item()
              - train_loss))
36        #print('after loss data in training', loss.item(), train_loss)
37
38    print('##### Epoch {}: Training End     #####'.format(epoch))
39
40    print('##### Epoch {}: Validation Start  #####'.format(epoch))
41    ######################
42    # validate the model #
43    ######################
44
45    model.eval()
46
47    with torch.no_grad():
48      for batch_idx, data in enumerate(validation_loader, 0):
49            ids = data['input_ids'].to(device, dtype = torch.long)
50            mask = data['attention_mask'].to(device, dtype = torch.long)
51            token_type_ids = data['token_type_ids'].to(device, dtype =
                  torch.long)
52            targets = data['targets'].to(device, dtype = torch.float)
53            outputs = model(ids, mask, token_type_ids)
54
55            loss = loss_fn(outputs, targets)
```

```
            valid_loss = valid_loss + ((1 / (batch_idx + 1)) *
                (loss.item() - valid_loss))
            val_targets.extend(targets.cpu().detach().numpy().tolist())
            val_outputs.extend(torch.sigmoid(outputs).cpu().detach().
            numpy().tolist())

      print('##### Epoch {}: Validation End     #####'.format(epoch))
      # calculate average losses
      #print('before cal avg train loss', train_loss)
      train_loss = train_loss/len(training_loader)
      valid_loss = valid_loss/len(validation_loader)
      # print training/validation statistics
      print('Epoch: {} \tAvgerage Training Loss: {:.6f} \tAverage
          Validation Loss: {:.6f}'.format(
            epoch,
            train_loss,
            valid_loss
            ))

      # create checkpoint variable and add important data
      checkpoint = {
            'epoch': epoch + 1,
            'valid_loss_min': valid_loss,
            'state_dict': model.state_dict(),
            'optimizer': optimizer.state_dict()
      }

        # save checkpoint
      save_ckp(checkpoint, False, checkpoint_path, best_model_path)

      ## TODO: save the model if validation loss has decreased
      if valid_loss <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving
            model ...'.format(valid_loss_min,valid_loss))
        # save checkpoint as best model
        save_ckp(checkpoint, True, checkpoint_path, best_model_path)
        valid_loss_min = valid_loss

    print('############# Epoch {}  Done   #############\n'.format(epoch))

  return model
```

In Listing 5.26, the BERT-based model is initialized and moved to the device for computation. Moreover, an optimizer is instantiated, which plays a crucial role in adjusting the model's parameters during training when passed to the `train_model` function. Lastly, two file paths are defined for saving the checkpoints of the trained model.

Listing 5.26: Parameter Initialization Prior to Training. Adapted from The Artificial Guy (2024)

```
model = BERTClass()
model.to(device)
```

```
3
4 optimizer = torch.optim.Adam(params =  model.parameters(),
      lr=LEARNING_RATE)
5
6 ckpt_path = r"C:\Users\YOUR_USER\Documents\Prepare Train
      Dataset\Building Classifier for Aspect Categories\curr_ckpt\model.pth"
7 best_model_path = r"C:\Users\YOUR_USER\Documents\Prepare Train
      Dataset\Building Classifier for Aspect
      Categories\best_model\best_model.pth"
```

Figure 5.25 depicts the invocation of the training function and displays the subsequent training results outputted in the console.

```
trained_model = train_model(EPOCHS, train_data_loader, val_data_loader, model, optimizer, ckpt_path, best_model_path)

############# Epoch 1: Training Start   #############
############# Epoch 1: Training End     #############
############# Epoch 1: Validation Start   #############
############# Epoch 1: Validation End     #############
Epoch: 1        Avgerage Training Loss: 0.000808        Average Validation Loss: 0.000919
Validation loss decreased (inf --> 0.000919).  Saving model ...
############# Epoch 1  Done   #############

############# Epoch 2: Training Start   #############
############# Epoch 2: Training End     #############
############# Epoch 2: Validation Start   #############
############# Epoch 2: Validation End     #############
Epoch: 2        Avgerage Training Loss: 0.000320        Average Validation Loss: 0.000536
Validation loss decreased (0.000919 --> 0.000536).  Saving model ...
############# Epoch 2  Done   #############
```

Figure 5.25: Training the Aspect Category Classifier over 2 Epochs

Ultimately, the trained model was saved to the variable `trained_model` (in 5.25). However, this variable represents the model including all parameters after the end of all epochs. It is not necessarily the best model achieved. Therefore, for further evaluation and testing of the trained model, it is loaded by applying the earlier described `load_ckp` function from Listing 5.23.

5.6.3 Evaluation

Testing of the trained classifier is exemplified in Listing 5.27. In lines 23-33, the example aspect "Lernräume" is tokenized using the BERT tokenizer, generating input encodings suitable for the model. Further key aspects to note are in lines 41 and 42, where the model's output is transformed into a predicted category label. This transformation involves determining the category with the highest predicted probability for each target.

Listing 5.27: Testing the Trained Classifier. Adapted from The Artificial Guy (2024)

```
1 # Path containing the checkpoint of the best model
2 best_model_path = r"C:\Users\YOUR_USER\Documents\Prepare Train
      Dataset\Building Classifier for Aspect
      Categories\best_model\best_model.pth"
3
4 # Initailizing the BERT Model and Optimizer
5 model = BERTClass()
6 optimizer = torch.optim.Adam(params =  model.parameters(),
      lr=LEARNING_RATE)
```

```
7
8  model, optimizer, epoch, valid_loss_min = load_ckp(best_model_path,
       model, optimizer)
9
10 ######## Test the Classifier
11
12 # Set model to evaluation/validation mode:
13 model.eval()
14
15 # Defining values for my 10 categories and save to variable "categories"
16 categories = ['Allgemein', 'Ausstattung', 'Bibliothek', 'Campus',
       'Digitales Studieren', 'Dozenten', 'Lage', 'Lehrveranstaltungen',
       'Organisation', 'Studieninhalte']
17
18 # Example of aspect to be classified as string
19 aspect = "Lernräume"
20
21 # Create encodings representing the aspect (using BertTokenizer)
22 # Encodings whose input_ids, attention_masks and token_type_ids are
       later used and passed to the model
23 encodings = tokenizer.encode_plus(
24     aspect, # aspect to be classified
25     None,
26     add_special_tokens=True,
27     max_length=MAX_LEN,
28     padding='max_length',
29     return_token_type_ids=True,
30     truncation=True,
31     return_attention_mask=True,
32     return_tensors='pt'
33 )
34
35 with torch.no_grad():
36     input_ids = encodings['input_ids'].to(device, dtype=torch.long)
37     attention_mask = encodings['attention_mask'].to(device,
           dtype=torch.long)
38     token_type_ids = encodings['token_type_ids'].to(device,
           dtype=torch.long)
39     output = model(input_ids, attention_mask, token_type_ids)
40     final_output = torch.sigmoid(output).cpu().detach().numpy().tolist()
41     y = categories[int(np.argmax(final_output, axis=1))]
42     print(y)
```

Figure 5.26 displays the prediction results of the trained classifier using 20 example aspects. Each aspect is shown on the left side, accompanied by its predicted category.

Having developed both the ABSA model and the aspect category classifier, the following chapter covers the data integration process, specifically the implementation of the data warehouse system. It explains how the developed web scraping functionality and the two models for ABSA are integrated to form the back-end tier of the data warehouse system. Furthermore, it covers the

Aspect Term	Aspect Category
Lernräume	Ausstattung
organisiert	Organisation
Professoren	Dozenten
Tische	Ausstattung
Materialien	Lehrveranstaltungen
Essen	Ausstattung
Kurse	Lehrveranstaltungen
LVAs	Lehrveranstaltungen
Inhalte	Studieninhalte
online	Digitales Studieren

Figure 5.26: Inference Testing of the Trained Classifier

development of a sentiment cube, which forms the data warehouse tier, and concludes with examples of how to analyze the created cube.

6

Sentiment Cube

This chapter begins with Section 6.1, which describes the conceptual model of the data warehouse using the Dimensional Fact Model (DFM) notation. Furthermore, the model's logical implementation in a database is covered in Section 6.2. Section 6.3 addresses the integration of the ABSA results with the created sentiment cube. On the one hand, it includes a presentation of how the two core functionalities from the previous chapters were integrated, covering data collection, model deployment, and database population in one modularized Python function. On the other hand, it demonstrates how a relational staging database is populated and then used as the data source for the previously defined data warehouse. Lastly, Section 6.4 provides examples of how the resulting sentiment cube may be applied by HEIs using SQL queries.

6.1 Conceptual Model

To create the data warehouse, a conceptual model was initially designed to ease further implementation. To this extent, the Dimensional Fact Model (DFM) notation was used, introduced by Golfarelli et al. (1998). In contrast to entity relationship models, which address modeling relational data for building a transactional database, the dimensional fact model is a way of representing multidimensional data. It aims at providing an easy and intuitive understanding of the dimensional data at a conceptual level, before the model is being transferred to the logical implementation of the data warehouse (Golfarelli et al., 1998).

At this stage, it was important to reflect on two questions: what data serves as input for the two developed ML models, and what is the business outcome of the models' predictions. For the ABSA model, one single sentence serves as input, obtained by splitting each review comment by sentences. For the aspect category classifier, one single aspect term serves as input, which is to be previously extracted by the ABSA model. The business outcome or output of both models' predictions is a sentiment score associated with an aspect term and sentence.

To create the DFM, one must initially define a fact or facts, before deciding on which attributes serve as the fact's measures, and which serve as dimensions and hierarchies (Golfarelli et al., 1998).

Figure 6.1 depicts the DFM of the data warehouse to be implemented later. A sentiment table was modeled as the only fact, representing the business interest. As the ABSA model predicts both a sentiment score and the corresponding confidence of predicting it, both a score and confidence were added as the fact's measures. The lowest levels of each dimension are defined by the lowest

granularity of sentiment, determining the fact. Since each sentiment's prediction is expecting a sentence as input and results in its linkage with an aspect term, two dimensions were created: one rating dimension and one aspect dimension. The rating dimension has the sentence as its lowest level. The sentence derives from a rating, which can be linked with the study program and university the student refers to. Moreover, each rating can be associated with a date. To be able to aggregate prediction results over time, the dimensional attributes "day", "month", and "year" were added. These were derived from the earlier fetched date attribute from the rating website, representing the day the rating was published. The rating total and recommendation, linked to each rating, and the university type, linked to a university, were added as non-dimensional attributes. This is because they merely serve as additional descriptive information. The aspect dimension has an aspect at its lowest level, followed by its predicted aspect category. This design allows for later roll-up queries, for instance when seeking to obtain sentiment scores by aspect category and a university's study program.

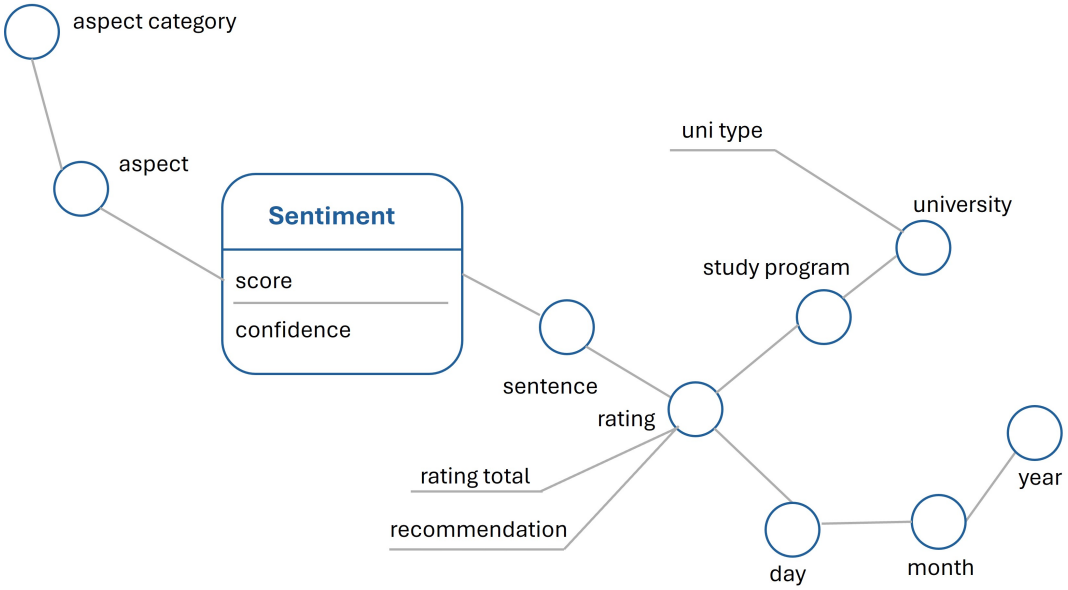


Figure 6.1: Dimensional Fact Model for the Sentiment Cube

6.2 Logical Implementation

This section covers how the previously modeled data was stored in a database. According to Vaisman and Zimányi (2014), there are several approaches to implement a multidimensional model. Mostly, the data is implemented by using relational data models, referring to relational online analytical processing (ROLAP). This approach was also used in this thesis, as it is well established and simple to adapt (Vaisman and Zimányi, 2014). However, other approaches such as multidimensional OLAP (MOLAP) or hybrid OLAP (HOLAP) could have been used.

Moreover, there are different ways of structuring the logical model. Following the star schema, one central fact table is connected with one dimensional table per dimension. The dimension tables in a star schema are typically not normalized. In contrast, applying the snowflake schema, each dimension is split into several dimension tables, normalizing its attributes (Vaisman and Zimányi, 2014). Since the data model of this use case is of low complexity and the amount of data is manageable, the star schema was selected for logical implementation, illustrated in Figure 6.2.

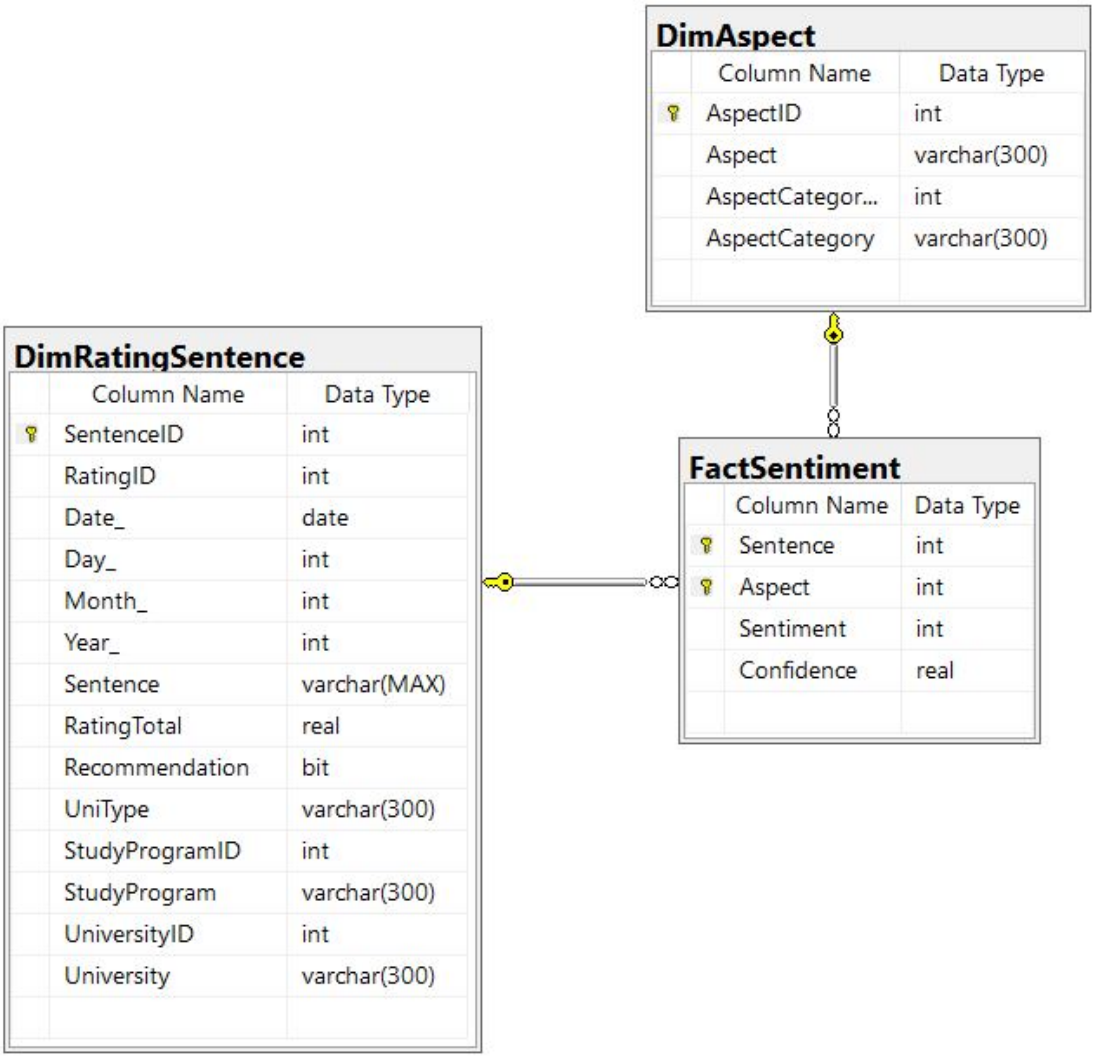


Figure 6.2: Star Schema of the Sentiment Cube

To implement the designed star schema, Microsoft SQL Server Management Studio (SSMS) was used as a relational database management system, with Microsoft SQL Server as the database engine. Listing 6.1 depicts the logical definition of the data warehouse in SSMS.

Listing 6.1: Logical Implementation of the Sentiment Cube in SSMS

```
CREATE DATABASE student_ratings_DW;
USE student_ratings_DW;
GO

CREATE TABLE DimRatingSentence (
    SentenceID INT,
    RatingID INT NOT NULL,
    Date_ DATE NOT NULL,
    Day_ INT NOT NULL,
    Month_ INT NOT NULL, -- 1:12
    Year_ INT NOT NULL,
    Sentence VARCHAR(max) NOT NULL,
    RatingTotal FLOAT(24) NOT NULL,
```

```
    Recommendation BIT,
    UniType VARCHAR(300) NOT NULL,
    StudyProgramID INT NOT NULL,
    StudyProgram VARCHAR(300) NOT NULL,
    UniversityID INT NOT NULL,
    University VARCHAR(300) NOT NULL,
    CONSTRAINT PK_DimRatingSentence PRIMARY KEY (SentenceID)
);

CREATE TABLE DimAspect (
    AspectID INT,
    Aspect VARCHAR(300) NOT NULL,
    AspectCategoryID INT NOT NULL,
    AspectCategory VARCHAR(300) NOT NULL,
    CONSTRAINT PK_DimAspect PRIMARY KEY (AspectID)
);

CREATE TABLE FactSentiment (
    Sentence INT NOT NULL FOREIGN KEY REFERENCES
       DimRatingSentence(SentenceID),
    Aspect INT NOT NULL FOREIGN KEY REFERENCES DimAspect(AspectID),
    Sentiment INT NOT NULL,
    Confidence FLOAT(24) NOT NULL,
    CONSTRAINT PK_FactSentiment PRIMARY KEY (Sentence, Aspect)
);
```

6.3 Data Integration

Having set the foundation for populating the sentiment cube, it was time to integrate the previously implemented modules addressed in Chapters 4 and 5: data collection and sentiment analysis. This integration involves the web crawling of all online reviews of a HEI and their subsequent processing using the two developed ML models. Once the aspects and sentiments are detected, they are inserted into a relational staging database. This entails populating the database with all reviews, including every sentence and its associated prediction results. Finally, the data is loaded into the data warehouse to enable OLAP querying on a sentiment cube.

To provide an end-to-end integration, starting with data collection and ending with database population, several Python functions were developed and embedded in one main function `absa_pipeline`, illustrated in Listing 6.2. This modular setup simplifies further usage because individual functionalities can be easily exchanged. For instance, if another library is used for data crawling online reviews, the function `get_uni_reviews` can be replaced. After calling the main function, all reviews from one HEI are crawled and processed.

Listing 6.2: Integration of Data Collection, Data Processing, and Data Population

```
def absa_pipeline(url, uni, uni_type, excel_path, json_path):
    # import pyabsa
    from pyabsa import AspectTermExtraction as ATEPC

```

```
5       # url: URL containing ratings for a single uni
6       # uni: String representing the desired name for the uni
7       # uni_type: String representing the desired name for the uni type
8       # excel_path: path where the reviews of the uni will be saved as
            excel
9
10      # scrape reviews from ONE HEI and save to df
11      df = get_uni_reviews(url, uni, uni_type, excel_path, json_path)
12
13      if url is None:
14          print("The URL provided is not valid.")
15
16      # if dataframe is emtpy then it was not possible to scrape it, so
            just ignore it
17      if df.empty:
18          print("Data Collection failed.")
19          return
20
21      # clean df (5 steps...)
22      df = clean_df(df)
23
24      # save df as excel to local (Overwriting to same path by the cleaned
            dataframe!)
25      # Create Excel without index
26      excel_path = excel_path + '/%s.xlsx' % (uni)
27      df.to_excel(excel_path, encoding='utf-8', index=False)
28
29      # generate a list of the pandas dataframe
30      df_list = df.values.tolist()
31
32      # insert Rating and inherent sentences into DB and save df_sentences
            of all reviews
33      df_sentences = INSERT_DB(df_list, None)
34
35      # save df_sentences of single HEI locally
36      df_sentences.to_excel(r'C:\Users\YOUR_USER\Documents\
37      Pipeline\DF_Sentences_DB-Insertions\%s_sentences_inserted.xlsx' %
            (uni), encoding='utf-8', index=False)
```

In line 22, a function clean_df was added to ensure that the data type of each crawled attribute aligns with the SQL data type defined for the database. This involves several steps, such as changing the date format from "dd-mm-YYYY" to "YYYY-mm-dd".

The function INSERT_DB, called in line 33, contains the core logic. It takes two arguments: all crawled reviews from a HEI as a list and, if desired, a path to an alternative ABSA model. Listing 6.3 shows a snippet of the INSERT_DB function. As the ABSA model using mdeberta as the pretrained BERT proved to yield the best results, it was defined as the default.

Listing 6.3: Initialization of the ABSA Model

```
1 def INSERT_DB(df_list, path_to_model):
```

```
    # df_list: list containing all crawled reviews from a HEI
    # path_to_model: path that represents a folder containing a
        checkpoint of ABSA Model (state_dict, tokenizer, config, args)

    # initialize ABSA Model!
    if path_to_model is not None:
        aspect_extractor = ATEPC.AspectExtractor(path_to_model,
                                                 auto_device=True,  # False
                                                    means load model on CPU
                                                 cal_perplexity=True,
                                                 )

    else: # use mdeberta Model as default ABSA Model
        aspect_extractor = ATEPC.AspectExtractor('mdeberta Model -
            fast_lcf_atepc_custom_dataset_cdw_apcacc_86.67_
        apcf1_86.69_atef1_64.86',
                                                 auto_device=True,  # False
                                                    means load model on CPU
                                                 cal_perplexity=True,
                                                 )
```

The resulting variable aspect_extractor represents the ABSA model, which was later applied on each single sentence.

Listing 6.4 depicts the function extract_aspect_polarity which was used for this purpose. It expects a sentence to be predicted and a PyABSA AspectExtractor object, representing the developed ABSA model. Since PyABSA's predict function saves the predictions in a JSON file, they have to be read accordingly. Eventually, the function returns all detected aspect terms of a single sentence as input, their corresponding sentiment polarities, and their prediction confidences.

Listing 6.4: Application of the ABSA Model

```
def extract_aspect_polarity(sentence, aspect_extractor):
    # uses the ML-model API to extract aspects from a sentence and
        determines its polarities
    # sentence: sentence to be analyzed (string)
    # aspect_extractor: PyABSA AspectExtractor object representing the
        ABSA model

    aspect_extractor.predict(sentence,
                             save_result=True,
                             print_result=True,
                             ignore_error=True
                             )

    import json
    # load ABSA results from JSON
    with open('Aspect Term Extraction and Polarity
        Classification.FAST_LCF_ATEPC.result.json', 'r',
        encoding='utf-8') as file:
        data = json.load(file)
```

```
16
17      # Saving results before they will be overwritten by next method call
18      aspects = data[0]['aspect']
19      sentiments = data[0]['sentiment']
20      confidences = data[0]['confidence']
21
22      return aspects, sentiments, confidences
```

As mentioned earlier in Chapter 5, the developed ABSA model seems to lack German language understanding. Therefore, its performance in aspect term detection is insufficient for quantitative analysis purposes. Thus, if the function above returns empty values, the corresponding sentence is ignored and not inserted into the database. This ensures that the database represents only sentences with actual predictions, enabling more useful OLAP querying.

Finally, after applying the function absa_pipeline for each URL containing all reviews of an Austrian HEI on the UEP, the database is populated. Figure 6.3 shows a snippet of the database entries. For clarity reasons, the SentenceCount column represents a counter for all sentences within one rating.

ID	SentenceCount	Sentence	Aspect	Aspect_Category	Sentiment	Confidence	RatingID
67	15	Es gibt Berufe für die man weniger lange studiert , m...	Berufe	Studieninhalte	neutral	0,97810000...	4
68	1	Das Studium zeichnet sich durch die hohe Flexibilät ...	Flexibilät	Studieninhalte	positive	0,99330002...	5
69	2	Es steht eine hohe Anzahl an Vertiefungen zur Verfü...	Vertiefungen	Studieninhalte	positive	0,99610000...	5
70	1	Insgesamt ist das Studium ganu gut aufgestellt , abe...	aufgestellt	Organisation	positive	0,84390002...	6
71	2	Die Organisation ist bei größeren Jahrgängen z. B. i...	Organisation	Organisation	positive	0,97479999...	6
72	3	Außerdem sind wir fast immer in den letztem A. Räu...	Räumen	Ausstattung	positive	0,47040000...	6
73	4	Auch die Dozent*innen sind teilweise nicht wirklich a...	Dozent	Dozenten	negative	0,97560000...	6

Figure 6.3: Snippet of the Sentences Table from the Database

To fill the data warehouse with the populated database as the source, Microsoft SQL Server Integration Services (SSIS) was used. Figure 6.4 depicts the data integration process designed in SSIS. Minor data manipulation tasks were fulfilled during the process, such as mapping the sentiment values from "positive" to 1, "neutral" to 0, and "negative" to -1. Equivalently, the data could have been loaded using solely SQL queries or other integration tools.

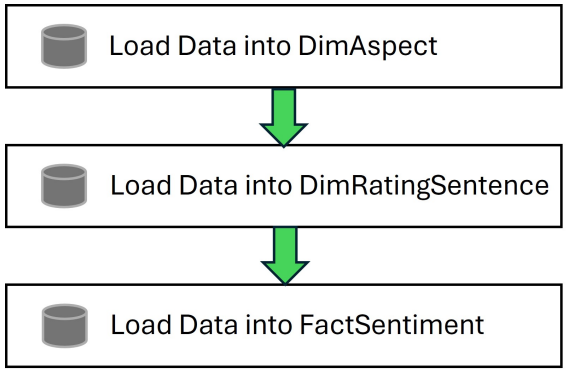


Figure 6.4: Data Integration Process using Microsoft SSIS

6.4 Querying the Sentiment Cube

To gain useful insights, the developed sentiment cube can be analyzed using a variety of tools and techniques, such as Power BI or Microsoft SQL Server Reporting Services (SSRS). In this thesis, SQL queries were used as the analysis method.

For instance, the query in Listing 6.5 calculates the average sentiment score and the average ABSA model's confidence of each sentence, considering only ratings commenting on business informatics related study programs and concerning the aspect category "study contents".

Listing 6.5: SQL Query 1: Average Sentiment Score on Study Contents in Business Informatics Programs

```
SELECT
   R.University,
   R.StudyProgram,
   A.AspectCategory,
  ROUND(SUM(CAST(F.Sentiment AS FLOAT))/count(*), 2) AS "Average
     Sentiment Score",
  count(*) as "Total Sentences",
  SUM(F.Sentiment) AS "Total Sentiment Score",
  ROUND(SUM(F.Confidence)/count(*), 2) AS "Average
  Confidence"
FROM FactSentiment_ F
  JOIN DimAspect_ A
    ON F.Aspect = A.AspectID
  JOIN DimRatingSentence_ R
    ON F.Sentence = R.SentenceID
WHERE R.StudyProgram LIKE '%Wirtschaftsinformatik%' AND A.AspectCategory
   = 'Studieninhalte'
GROUP BY
  R.University,
  R.StudyProgram,
  A.AspectCategory
ORDER BY
  "Average Sentiment Score" DESC
```

	University	StudyProgram	AspectCategory	Average Sentiment Score	Total Sentences	Total Sentiment Score	Confidence Average
1	Fachhochschule Campus 02	Informationstechnologien & Wirtschaftsinformatik(...	Studieninhalte	1	1	1	1
2	Fachhochschule Vorarlberg	Wirtschaftsinformatik - Digital Transformation(M.S...	Studieninhalte	1	1	1	0,99
3	Universität Klagenfurt	Wirtschaftsinformatik(B.Sc.)	Studieninhalte	1	3	3	0,99
4	JKU Linz	Wirtschaftsinformatik(B.Sc.)	Studieninhalte	0,74	35	26	0,92
5	FH Technikum Wien	Wirtschaftsinformatik(Bachelor)	Studieninhalte	0,71	31	22	0,94
6	FH Salzburg	Wirtschaftsinformatik & Digitale Transformation(B....	Studieninhalte	0,6	5	3	0,99
7	FOM Hochschule	Wirtschaftsinformatik(B.Sc.)	Studieninhalte	0,51	92	47	0,91
8	Fachhochschule Campus 02	Wirtschaftsinformatik(Bachelor)	Studieninhalte	0,5	2	1	0,87
9	Uni Innsbruck	Wirtschaftsinformatik(M.Sc.)	Studieninhalte	0,5	4	2	0,95
10	FOM Hochschule	Wirtschaftsinformatik - Business Information Syste...	Studieninhalte	0,47	43	20	0,93
11	TU Wien	Wirtschaftsinformatik(B.Sc.)	Studieninhalte	0,31	26	8	0,92
12	Uni Wien	Wirtschaftsinformatik(B.Sc.)	Studieninhalte	0,22	9	2	0,93
13	Uni Wien	Wirtschaftsinformatik(M.Sc.)	Studieninhalte	0	5	0	0,86
14	JKU Linz	Wirtschaftsinformatik(M.Sc.)	Studieninhalte	-0,2	5	-1	0,99
15	FOM Hochschule	Wirtschaftsinformatik - kommunal(B.Sc.)	Studieninhalte	-0,2	10	-2	0,89

Figure 6.5: Result of SQL Query 1

The result of the query is depicted in Figure 6.5. For instance, the sentiment score of a sentence commenting on the study contents of the Business Informatics Bachelor study program at JKU Linz is on average 0.74. In comparison, the equivalent metric concerning the Austrian HEIs "TU

Wien" and "Uni Wien" are 0.31 and 0.22 respectively. This might indicate that the Business Informatics Bachelor program at JKU Linz might outperform the same program offered by the other two HEIs in terms of its study contents.

Listing 6.6 provides another example of an SQL query on the sentiment cube. It calculates the average sentiment score and the average ABSA model's confidence of each sentence written in 2023, considering only ratings commenting on business administration related study programs and concerning the aspect category "lectures".

Listing 6.6: SQL Query 2: Average Sentiment Score on Lectures in Business Administration Programs

```
SELECT
   R.University,
   R.StudyProgram,
   A.AspectCategory,
      ROUND(SUM(CAST(F.Sentiment AS FLOAT))/count(*), 2) AS "Average
         Sentiment Score",
      count(*) as "Total Sentences",
      SUM(F.Sentiment) AS "Total Sentiment Score",
      ROUND(SUM(F.Confidence)/count(*), 2) AS "Confidence Average",
      R.Year_
FROM FactSentiment_ F
  JOIN DimAspect_ A
    ON F.Aspect = A.AspectID
  JOIN DimRatingSentence_ R
    ON F.Sentence = R.SentenceID
WHERE
    (R.StudyProgram LIKE '%Wirtschaft%' AND R.StudyProgram LIKE
       '%Betrieb%')
    AND A.AspectCategory = 'Lehrveranstaltungen'
    AND R.Year_ = 2023
GROUP BY
  R.University,
  R.StudyProgram,
  A.AspectCategory,
  R.Year_
ORDER BY
  "Average Sentiment Score" DESC
```

	University	StudyProgram	AspectCategory	Average Sentiment Score	Total Sentences	Total Sentiment Score	Confidence Average	Year_
1	Uni Graz	Betriebswirtschaft(M.Sc.)	Lehrveranstaltungen	1	1	1	0,94	2023
2	JKU Linz	Internationale Betriebswirtschaft (IBWL)(B.Sc.)	Lehrveranstaltungen	1	1	1	1	2023
3	Uni Wien	Internationale Betriebswirtschaft(B.Sc.)	Lehrveranstaltungen	0,62	13	8	0,89	2023
4	IMC Fachhochschule Krems	Betriebswirtschaft für das Gesundheitswesen(Bach...	Lehrveranstaltungen	0,5	2	1	0,95	2023
5	FOM Hochschule	Betriebswirtschaft & Wirtschaftspsychologie(B.Sc.)	Lehrveranstaltungen	0,43	46	20	0,94	2023
6	Fachhochschule Vorarlberg	Internationale Betriebswirtschaft(B.A.)	Lehrveranstaltungen	0,4	5	2	0,91	2023
7	JKU Linz	Betriebswirtschaftslehre (BWL)(B.Sc.)	Lehrveranstaltungen	0,27	11	3	0,95	2023
8	Uni Graz	Betriebswirtschaft(B.Sc.)	Lehrveranstaltungen	0,13	23	3	0,91	2023
9	FH Salzburg	Betriebswirtschaft(B.A.)	Lehrveranstaltungen	0	7	0	0,94	2023
10	Universität Klagenfurt	Betriebswirtschaft(B.A.)	Lehrveranstaltungen	0	4	0	0,88	2023
11	FH Salzburg	Betriebswirtschaft(M.A.)	Lehrveranstaltungen	0	1	0	0,91	2023
12	WU Wien	Internationale Betriebswirtschaft(B.Sc.)	Lehrveranstaltungen	0	1	0	0,99	2023
13	Uni Wien	Betriebswirtschaft(M.Sc.)	Lehrveranstaltungen	-0,23	13	-3	0,84	2023
14	Uni Wien	Betriebswirtschaft(B.Sc.)	Lehrveranstaltungen	-0,24	21	-5	0,9	2023
15	Uni Wien	Internationale Betriebswirtschaft(M.Sc.)	Lehrveranstaltungen	-0,25	4	-1	0,81	2023
16	MCI Innsbruck	Betriebswirtschaft / Business Administration Online...	Lehrveranstaltungen	-1	1	-1	0,54	2023

Figure 6.6: Result of SQL Query 2

The output of this query is illustrated in Figure 6.6. For instance, the sentiment score of a sentence commenting on the lectures of the Business Administration Bachelor study program at "Uni Wien" is, on average, -0.24. In comparison, the equivalent metric concerning the lectures of the International Business Administration Bachelor program at the same university is 0.62. This might indicate that the lectures from the latter study program are perceived as more positive by the students.

7

Conclusion

This thesis explores the design and implementation of a data warehouse system for analyzing student sentiment in online reviews from university evaluation platforms, with a focus on Austrian Higher Education Institutions (HEIs). Large Language Models (LLMs), such as BERT, were leveraged to develop a system capable of transforming unstructured text from online reviews into structured data suitable for OLAP analysis. The resulting sentiment cube enables HEI administrators to extract valuable insights into student satisfaction across multiple dimensions, including universities, study programs, time periods, and aspect categories, which refer to specific aspects of the educational experience.

The prototype system successfully demonstrates the feasibility of automating student feedback analysis, allowing for data-driven decisions regarding student retention, recruitment, and the improvement of services offered by HEIs. This approach is particularly beneficial for managing the vast amounts of unstructured textual data generated by students in online forums, reviews, and evaluations, which are otherwise difficult to process manually.

However, several limitations were identified. One significant challenge is the limited coverage of aspect categories. Topic modeling was performed, but the resulting topics were deemed unhelpful, as they did not extend the predefined set of aspect categories. Another limitation is the difficulty in capturing nuanced contextual meanings in student reviews. For example, the model struggles with complex sentiments such as irony or sarcasm, which are common challenges in NLP tasks. Additionally, the ABSA model was not always able to extract an aspect term from every review. In some cases, no aspect term was identified, meaning that not all data examples were included in the final analysis. Furthermore, the aspect categories defined for university-wide feedback may not perfectly align with the specific context of individual courses or programs. Future research could involve expanding aspect categories or defining custom categories for specific courses, which would improve the precision of the analysis.

Moreover, future work may focus on enhancing the reliability and accuracy of the system by incorporating additional data sources, such as course evaluations conducted each semester. Integrating ABSA with topic modeling techniques like Latent Dirichlet Allocation (LDA) could provide a more nuanced understanding of the topics discussed in student reviews, potentially leading to better categorization of aspects. Additionally, addressing classical NLP challenges such as negation handling and sarcasm detection will be crucial for refining the sentiment analysis process.

Overcoming these challenges will allow the system to be further adapted and scaled for use across different institutions and contexts, providing more tailored insights into student satisfaction.

In summary, this thesis lays the groundwork for automating the analysis of student feedback at HEIs using a data warehouse system combined with ABSA. While the results are promising, several areas for improvement were identified, ensuring that future iterations of the system can offer more accurate and context-aware insights to support decision-making at universities.

Bibliography

Abdelrazeq, A., Janßen, D., Tummel, C., Jeschke, S., & Richert, A. (2016). Sentiment analysis of social media for evaluating universities. *Automation, Communication and Cybernetics in Science and Engineering 2015/2016*, 233–251.

Alford, J. (2002). Defining the client in the public sector: A social-exchange perspective. *Public administration review*, *62*(3), 337–346.

Altrabsheh, N., Gaber, M., & Cocea, M. Sa-e: Sentiment analysis for education [Additional Information: Frontiers of Artificial Intelligence and Applications (FAIA) series, IOS Press.; 5th KES International Conference on Intelligent Decision Technologies ; Conference date: 26-06-2013 Through 28-06-2013]. English. In: Additional Information: Frontiers of Artificial Intelligence and Applications (FAIA) series, IOS Press.; 5th KES International Conference on Intelligent Decision Technologies ; Conference date: 26-06-2013 Through 28-06-2013. 2013.

Azab, M., Mihalcea, R., & Abernethy, J. (2016). Analysing ratemyprofessors evaluations across institutions, disciplines, and cultures: The tell-tale signs of a good professor. *Social Informatics: 8th International Conference, SocInfo 2016, Bellevue, WA, USA, November 11-14, 2016, Proceedings, Part I 8*, 438–453.

Bhowmik, A., Noor, N. M., Miah, M. S. U., Mazid-Ul-Haque, M., & Karmaker, D. (2023). A comprehensive dataset for aspect-based sentiment analysis in evaluating teacher performance. *AIUB Journal of Science and Engineering (AJSE)*, *22*(2), 200–213.

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and olap technology. *ACM Sigmod record*, *26*(1), 65–74.

Cirqueira, D., Pinheiro, M., Braga, T., Jacob Jr, A., Reinhold, O., Alt, R., & Santana, Á. (2017). Improving relationship management in universities with sentiment analysis and topic modeling of social media channels: Learnings from ufpa. *Proceedings of the International Conference on Web Intelligence*, 998–1005.

Colace, F., Santo, M. D., & Greco, L. (2014). Safe: A sentiment analysis framework for e-learning. *Int. J. Emerg. Technol. Learn.*, *9*, 37–41. https://api.semanticscholar.org/CorpusID:44914999

Cuzzocrea, A., De Maio, C., Fenza, G., Loia, V., & Parente, M. (2016). Olap analysis of multidimensional tweet streams for supporting advanced analytics. *Proceedings of the 31st annual ACM symposium on applied computing*, 992–999.

Dalal, R., Safhath, I., Piryani, R., Kappara, D., & Singh, V. K. (2014). A lexicon pooled machine learning classifier for opinion mining from course feedbacks. *Intelligence and Security Informatics*. https://api.semanticscholar.org/CorpusID:42719205

Dehbozorgi, N., & Mohandoss, D. P. (2021). Aspect-based emotion analysis on speech for predicting performance in collaborative learning. *2021 IEEE frontiers in education conference (FIE)*, 1–7.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dhanalakshmi, V., Bino, D., & Saravanan, A. M. (2016). Opinion mining from student feedback data using supervised learning algorithms. *2016 3rd MEC international conference on big data and smart city (ICBDSC)*, 1–5.

Dolianiti, F. S., Iakovakis, D., Dias, S. B., Hadjileontiadou, S. J., Diniz, J. A., Natsiou, G., Tsitouridou, M., Bamidis, P. D., & Hadjileontiadis, L. J. (2019). Sentiment analysis on educational datasets: A comparative evaluation of commercial tools. *Educational Journal of the University of Patras UNESCO Chair*.

Edalati, M., Imran, A. S., Kastrati, Z., & Daudpota, S. M. (2022). The potential of machine learning algorithms for sentiment classification of students' feedback on mooc. *Intelligent Systems and Applications: Proceedings of the 2021 Intelligent Systems Conference (IntelliSys) Volume 3*, 11–22.

Gallinucci, E., Golfarelli, M., & Rizzi, S. (2015). Advanced topic modeling for social business intelligence. *Information Systems*, *53*, 87–106.

Golfarelli, M. (2014). Social business intelligence: Olap applied to user generated contents. *2014 11th International Conference on e-Business (ICE-B)*, IS–11.

Golfarelli, M., Maio, D., & Rizzi, S. (1998). The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, *7*(02n03), 215–247.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Gottipati, S., Shankararaman, V., & Gan, S. (2017). A conceptual framework for analyzing students' feedback. *2017 IEEE frontiers in education conference (FIE)*, 1–8.

Guilbault, M. (2016). Students as customers in higher education: Reframing the debate. *Journal of Marketing for Higher Education*, *26*(2), 132–142.

Han, P. (2014). A literature review on college choice and marketing strategies for recruitment. *Family and Consumer Sciences Research Journal*, *43*, 120–130. https://doi.org/10.1111/FCSR.12091

Hawkins, D. M. (2004). The problem of overfitting. *Journal of chemical information and computer sciences*, *44*(1), 1–12.

He, P., Liu, X., Gao, J., & Chen, W. (2021). Deberta: Decoding-enhanced bert with disentangled attention. *International Conference on Learning Representations*. https://openreview.net/forum?id=XPZIaotutsD

Hemmatian, F., & Sohrabi, M. K. (2019). A survey on classification techniques for opinion mining and sentiment analysis. *Artificial Intelligence Review*, *52*(3), 1495–1545. https://doi.org/10.1007/s10462-017-9599-6

Hemsley-Brown, J., & Oplatka, I. (2006). Universities in a competitive global marketplace: A systematic review of the literature on higher education marketing. *International Journal of public sector management*, *19*(4), 316–338.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., & Gelly, S. (2019). Parameter-efficient transfer learning for nlp. *International conference on machine learning*, 2790–2799.

Hugging Face Inc. & deepset. (2023). Dbmdz/bert-base-german-cased: German bert model [Accessed: 2024-06-15].

Hussain, S., Ayoub, M., Jilani, G., Yu, Y., Khan, A., Wahid, J. A., Butt, M. F. A., Yang, G., Moller, D. P., & Weiyan, H. (2022). Aspect2labels: A novelistic decision support system for higher educational institutions by using multi-layer topic modelling approach. *Expert Systems with Applications*, *209*, 118119.

Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent data analysis*, *6*(5), 429–449.

Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y., & Zhao, L. (2019). Latent dirichlet allocation (lda) and topic modeling: Models, applications, a survey. *Multimedia tools and applications*, *78*, 15169–15211.

Jena, R. (2019). Sentiment mining in a collaborative learning environment: Capitalising on big data. *Behaviour & Information Technology*, *38*(9), 986–1001.

Johnes, J. (2018). University rankings: What do they really show? *Scientometrics*, *115*(1), 585–606.

Kandhro, I. A., Ali, F., Uddin, M., Kehar, A., & Manickam, S. (2024). Exploring aspect-based sentiment analysis: An in-depth review of current methods and prospects for advancement. *Knowledge and Information Systems*. https://doi.org/10.1007/s10115-024-02104-8

Kastrati, Z., Imran, A. S., & Kurti, A. (2020). Weakly supervised framework for aspect-based sentiment analysis on students' reviews of moocs. *IEEE Access*, *8*, 106799–106810.

Khder, M. A. (2021). Web scraping or web crawling: State of art, techniques, approaches and application. *International Journal of Advances in Soft Computing & Its Applications*, *13*(3).

Kimball, R., & Ross, M. (2011). *The data warehouse toolkit: The complete guide to dimensional modeling*. John Wiley & Sons.

Kotler, P. (1977). From sales obsession to marketing effectiveness. *Harvard business review*, *55*, 67–75.

Koufakou, A., Gosselin, J., & Guo, D. (2016). Using data mining to extract knowledge from student evaluation comments in undergraduate courses. *2016 International Joint Conference on Neural Networks (IJCNN)*, 3138–3142.

Kraiem, M. B., Feki, J., Khrouf, K., Ravat, F., & Teste, O. (2015). Modeling and olaping social media: The case of twitter. *Social Network Analysis and Mining*, *5*, 1–15.

Krawczyk, B. (2016). Learning from imbalanced data: Open challenges and future directions. *Progress in Artificial Intelligence*, *5*(4), 221–232.

Liddy, E. D. (2001). Natural language processing.

Ligthart, A., Catal, C., & Tekinerdogan, B. (2021). Systematic reviews in sentiment analysis: A tertiary study. *Artificial Intelligence Review*, 1–57.

Liu, B. (2017). Many Facets of Sentiment Analysis. In E. Cambria, D. Das, S. Bandyopadhyay & A. Feraco (Eds.), *A Practical Guide to Sentiment Analysis* (pp. 11–39). Springer International Publishing. https://doi.org/10.1007/978-3-319-55394-8_2

Liu, B. (2022, May). *Sentiment Analysis and Opinion Mining* [Google-Books-ID: xYhyEAAAQBAJ]. Springer Nature.

Liu, Z., Liu, S., Liu, L., Sun, J., Peng, X., & Wang, T. (2016, March). *2016 sentiment recognition of online course reviews using multi-swarm optimization-based selected features*.

Ma, E. (2019). Nlp augmentation.

Melba Rosalind, J., & Suguna, S. (2022). Predicting students' satisfaction towards online courses using aspect-based sentiment analysis. *International Conference on Computer, Communication, and Signal Processing*, 20–35.

Min, B., Ross, H., Sulem, E., Veyseh, A. P. B., Nguyen, T. H., Sainz, O., Agirre, E., Heintz, I., & Roth, D. (2023). Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, *56*(2), 1–40.

Moalla, I., Nabli, A., & Hammami, M. (2022). Data warehouse building to support opinion analysis in social media. *Social Network Analysis and Mining*, *12*(1), 123.

Moustafa, K. (2024). University rankings: Time to reconsider. *BIOIMPACTS*.

Nikolić, N., Grljević, O., & Kovačević, A. (2020). Aspect-based sentiment analysis of reviews in the domain of higher education. *The Electronic Library*, *38*(1), 44–64.

Olshavsky, R. W., & Spreng, R. A. (1995). Consumer satisfaction and students: Some pitfalls of being customer driven. *Journal of Consumer Satisfaction, Dissatisfaction and Complaining Behavior*, *8*, 69–77.

Onan, A. (2021). Sentiment analysis on massive open online course evaluations: A text mining and deep learning approach. *Computer Applications in Engineering Education*, *29*(3), 572–589.

Otter, D. W., Medina, J. R., & Kalita, J. K. (2020). A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, *32*(2), 604–624.

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345–1359.

Pitman, T. (2000). Perceptions of academics and students as customers: A survey of administrative staff in higher education. *Journal of Higher Education policy and management*, *22*(2), 165–175.

Pramod, D., Bharathi, S. V., & Raman, R. (2022). Faculty effectiveness prediction using machine learning and text analytics. *2022 IEEE Technology and Engineering Management Conference (TEMSCON EUROPE)*, 40–47.

Proisl, T., & Uhrig, P. (2016). SoMaJo: State-of-the-art tokenization for German web and social media texts. In P. Cook, S. Evert, R. Schäfer & E. Stemle (Eds.), *Proceedings of the 10th Web as Corpus workshop (WAC-X) and the EmpiriST shared task* (pp. 57–62). Association for Computational Linguistics. https://doi.org/10.18653/v1/W16-2607

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.

Ramponi, A., & Plank, B. (2020). Neural unsupervised domain adaptation in nlp—a survey. *arXiv preprint arXiv:2006.00632*.

Reja, U., Manfreda, K. L., Hlebec, V., & Vehovar, V. (2003). Open-ended vs. close-ended questions in web questionnaires. *Developments in applied statistics*, *19*(1), 159–177.

Roaring, B. F., Patacsil, F. F., & Parrone, J. M. (2022). Analyzing pangasinan state university student's faculty teaching performance rating using text mining technique. *WSEAS Transactions on Information Science and Applications*, *19*, 161–170.

Santos, C. L., Rita, P., & Guerreiro, J. (2018). Improving international attractiveness of higher education institutions based on text mining and sentiment analysis. *International Journal of Educational Management*, *32*(3), 431–447.

Scaffidi, C. (2016). Mining online forums for valuable contributions. *2016 11th Iberian Conference on Information Systems and Technologies (CISTI)*, 1–6.

Schurig, T., Zambach, S., Mukkamala, R. R., & Petry, M. (2022). Aspect-based sentiment analysis for university teaching analytics.

Sennrich, R., Haddow, B., & Birch, A. (2015). Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

Shaik, T., Tao, X., Dann, C., Xie, H., Li, Y., & Galligan, L. (2023). Sentiment analysis and opinion mining on educational data: A survey. *Natural Language Processing Journal*, *2*, 100003.

Shorten, C., Khoshgoftaar, T. M., & Furht, B. (2021). Text data augmentation for deep learning. *Journal of big Data*, *8*(1), 101.

Siegel, M., & Alexa, M. (2020). *Sentiment-Analyse deutschsprachiger Meinungsäußerungen: Grundlagen, Methoden und praktische Umsetzung*. Springer Fachmedien. https://doi.org/10.1007/978-3-658-29699-5

Sievert, C., & Shirley, K. (2014). Ldavis: A method for visualizing and interpreting topics. *Proceedings of the workshop on interactive language learning, visualization, and interfaces*, 63–70.

Sindhu, I., Daudpota, S. M., Badar, K., Bakhtyar, M., Baber, J., & Nurunnabi, M. (2019). Aspect-based opinion mining on student's feedback for faculty teaching performance evaluation. *IEEE Access*, *7*, 108729–108741.

StackOverflow. (2015). Stratified train/test split in scikit-learn [Accessed: 29.04.2024]. https://stackoverflow.com/questions/29438265/stratified-train-test-split-in-scikit-learn

StackOverflow. (2016a). How to split data into 3 sets (train, validation, and test)? [Accessed: 15.04.2024]. https://stackoverflow.com/questions/38250710/how-to-split-data-into-3-sets-train-validation-and-test

StackOverflow. (2016b). Pandas get_dummies [Accessed: 29.04.2024]. https://stackoverflow.com/questions/36285155/pandas-get-dummies

Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to fine-tune bert for text classification? *Chinese computational linguistics: 18th China national conference, CCL 2019, Kunming, China, October 18–20, 2019, proceedings 18*, 194–206.

Tanaka, F. H. K. D. S., & Aranha, C. (2019). Data augmentation using gans. *arXiv preprint arXiv:1904.09135*.

The Artificial Guy. (2024). Multi-label text classification using bert (pytorch) [Accessed: 10.05.2024]. https://github.com/theartificialguy/NLP-with-Deep-Learning/blob/master/BERT/Multi%20Label%20Text%20Classification%20using%20BERT%20PyTorch/bert_multilabel_pytorch_standard.ipynb

Vaisman, A., & Zimányi, E. (2014). Data warehouse systems. *Data-Centric Systems and Applications*, *9*.

Wang, S., Li, Z., Chao, W., & Cao, Q. (2012). Applying adaptive over-sampling technique based on data density and cost-sensitive svm to imbalanced learning. *The 2012 international joint conference on neural networks (IJCNN)*, 1–8.

Wirth, R., & Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, *1*, 29–39.

Yang, H. (2022). ABSADatasets [Accessed: 2024-06-13].

Yang, H., Zhang, C., & Li, K. (2023). Pyabsa: A modularized framework for reproducible aspect-based sentiment analysis. *Proceedings of the 32nd ACM international conference on information and knowledge management*, 5117–5122.

Yang, Y., Malaviya, C., Fernandez, J., Swayamdipta, S., Bras, R. L., Wang, J.-P., Bhagavatula, C., Choi, Y., & Downey, D. (2020). Generative data augmentation for commonsense reasoning. *arXiv preprint arXiv:2004.11546*.

Ying, X. (2019). An overview of overfitting and its solutions. *Journal of physics: Conference series*, *1168*, 022022.

Zhang, L., Wang, S., & Liu, B. (2018). Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *8*(4), e1253.

Zhang, W., Li, X., Deng, Y., Bing, L., & Lam, W. (2023). A survey on aspect-based sentiment analysis: Tasks, methods, and challenges. *IEEE Transactions on Knowledge and Data Engineering*, *35*(11), 11019–11038. https://doi.org/10.1109/TKDE.2022.3230975

Zheng, Y. (2024). Pyabsa: Open framework for aspect-based sentiment analysis [Accessed: 22.04.2024]. https://github.com/yangheng95/PyABSA/tree/v2/pyabsa