Department of Information
Engineering and Computer Science

Master's Degree in
Computer Science

Final Dissertation

Extracting Personal Attributes from Customer Reviews

Supervisor University

Alberto Montresor

Supervisor Company

Mouna Kacimi

Student

Nina Verbeeke

Academic year 2023/2024

Contents

Abstract

In today's highly competitive market, understanding customers is crucial for product development and decision-making. Traditional methods like surveys and focus groups often fall short in capturing the dynamic nature of consumers. This thesis focuses on leveraging customer-generated content, specifically product reviews, to extract personal attributes that provide deeper insights into customer characteristics. Personal attributes refer to unique traits that describe an individual, such as physical appearance, profession, lifestyle, and circumstances.

The main goal of this research was to develop a flexible, automated approach for extracting personal attributes from product reviews using large language models (LLMs). Unlike previous studies that relied on predefined lists of attributes, this thesis introduces an end-to-end pipeline capable of extracting a broad range of personal attributes based on the context and input data. The proposed pipeline includes key steps such as prefixing, coreference resolution, text segmentation, proposition filtering, triplet extraction, entity decomposition, entity deduplication, and graph construction. Each step contributes to refining the extraction process and generating structured knowledge from unstructured text.

To evaluate the pipeline, a case study was conducted using product reviews for a day cream. The focus was on extracting skin conditions mentioned by reviewers, such as acne, eczema, and dry skin. The results demonstrated that the pipeline could effectively extract relevant personal attributes without requiring fine-tuning, highlighting the potential for future applications in different contexts, such as hair type in reviews for hair dryers or pet ownership in reviews for vacuum cleaners.

The main contributions of this research include the development of a flexible framework for personal attribute extraction, a comparison with traditional triplet extraction methods, and the successful implementation of a case study. This thesis presents a pipeline demonstrating that LLMs provide a solution for extracting personal attributes from product reviews, providing valuable insights for businesses seeking to better understand their customers.

Finally, this work opens up opportunities for future research in extending the application of this framework and refining the techniques used for knowledge graph construction. The flexibility of LLMs offers a promising direction for further improving the efficiency and adaptability of customer analysis methods.

Chapter 1

Introduction

1.1 Background

Truly understanding customers is key to a successful business. Companies increasingly leverage data analytics to uncover insights into their customers' needs, preferences, and socio-demographic profiles. This information is essential for enhancing user experiences, differentiating from competitors, and creating products that satisfy market demands. Moreover, understanding customers improves strategic planning by facilitating informed decisions about market positioning, pricing, and promotional strategies. By aligning marketing strategies with consumer needs, businesses can achieve more successful and impactful outcomes.

Traditional approaches to understanding customers, such as surveys, focus groups, and market research reports, have long been relied upon. While these methods can provide useful insights, they have significant limitations. Surveys, for instance, often suffer from low response rates and biased results. Focus groups may not accurately represent the entire customer base, leading to incomplete or skewed findings. Market research reports, while detailed, are frequently expensive and time-consuming to produce. Additionally, these traditional methods typically provide only a static view of customer preferences, failing to capture the dynamic changes in consumer behavior over time.

To address these limitations, businesses are increasingly turning to more innovative approaches. One such approach is the analysis of customer-generated content, such as product reviews. Product reviews offer valuable insights into customer preferences and sentiments. Various techniques, including sentiment analysis, topic modeling, and natural language processing, can be employed to extract meaningful information from this user-generated content.

1.2 Research objectives

Traditionally, product reviews have been analyzed primarily on feedback about the product, such as customer opinions, satisfaction levels, and sentiments on specific features. However, insights into customers' personal attributes, behaviors, and usage contexts have not been thoroughly explored. Addressing this gap presents a valuable opportunity to leverage user-generated content for a deeper understanding of customers.

The extraction of personal attributes from product reviews is the focus of this research. Personal attributes are specific characteristics that uniquely describe an individual's traits, lifestyle, and circumstances, such as physical appearance, profession, household composition, and living situation. These attributes could include details such as hair type, skin condition, profession, number of family members or pets, and whether they live in an apartment, townhouse or single-family home. By extracting personal attributes from product reviews, we aim to gain a deeper understanding of the characteristics of customers.

1.3 Approach

Recent research has explored methods for extracting personal attributes from unstructured text. A significant portion of this work has been specifically focused on dialogues. The related work will be discussed in more detail in Chapter 2. In line with previous research on extracting personal attributes, this study will also use the triplet format to store these attributes. However, our approach diverges by addressing some of the key limitations of earlier studies, including that previous research often relied on predefined lists of attributes. Inspired by recent advancements in pre-trained large language models (LLMs), this work introduces a novel approach for extracting personal attributes using these models. Our method aims to extract triplets from product reviews without being limited by predefined attribute categories, thus offering a more comprehensive and adaptable solution to capture the various personal attributes revealed in customer feedback.

We present an end-to-end pipeline, starting from raw text all the way through to a constructed knowledge graph. The approach , text pre-processing, triplet extraction post-processing.

1.4 Main contributions

The main contributions of this thesis are as follows:

- **End-to-end pipeline**: We developed an end-to-end pipeline that leverages LLMs to automate the construction of Knowledge Graphs (KGs) from unstructured data. This framework is designed to be adaptable and scalable, reducing the need for extensive human intervention and expertise.
- **Comparison with Traditional Method**: We provided a comparison between our LLM-based approach and a traditional triplet extraction method.
- **Case Study Implementation**: We conducted a detailed case study to evaluate the performance of our framework. This case study demonstrates the framework's ability to accurately and efficiently construct KGs, showcasing its potential to outperform existing methods in various contexts.
- **Future Directions**: We identified potential areas for future research and development, offering insights into how the framework can be further improved and extended to different domains and applications.

These contributions collectively demonstrate the potential of LLMs to revolutionize the process of triplet extraction, offering a more automated, scalable, and efficient approach compared to traditional methods. This research introduces a novel approach by focusing on extracting personal attributes from product reviews, an aspect that has not been extensively explored in prior research. Unlike previous studies that concentrated on analyzing what people say about products, our approach emphasizes understanding what people reveal about themselves through their reviews. Additionally, our method leverages LLMs to extract triplets, offering enhanced flexibility and adaptability compared to traditional methods constrained by predefined attribute categories.

1.5 Thesis outline

The rest of this thesis is organized as follows: In Section 2, we establish the background by presenting a literature review that highlights findings from relevant studies. This review emphasizes various settings and methodologies employed in prior research. Although these works differ in context, they provide valuable insights that inform our research. Additionally, Section 2 introduces foundational definitions, including knowledge graphs and

triplets, and outlines the problem formulation. Following this, Section 3 offers a high-level overview of our approach, which consists of an end-to-end pipeline. Each step of this pipeline is explained in detail. Section 4 delves into the implementation specifics of the pipeline, providing a sequential breakdown of each step, including the tools and packages utilized, along with relevant code snippets. In Chapter 5, we present the design of two experiments. The first experiment serves as a baseline study, comparing our method with a traditional approach to personal attribute extraction. The second experiment is a case study focused on extracting skin conditions from a set of customer reviews. Finally, in Section 6, we summarize the contributions of this work, discuss the implications of our findings, and suggest possible directions for future research.

Chapter 2

Background

This chapter provides the background of this research. It begins with a literature review of related work, which contextualizes the study and highlights gaps that this research aims to address. The chapter then introduces foundational definitions, including knowledge graphs and triplets, and outlines the problem formulation.

2.1 Related Work

This section outlines findings from the relevant literature, highlighting various settings and methodologies considered in previous studies. Although differing in context, these works provide valuable insights for our research.

2.1.1 UX extraction

Yang et al. [1] demonstrate the potential of leveraging machine learning and text mining techniques to extract and categorize user experience (UX) information from product reviews. They propose a faceted conceptual model for UX modeling that comprises four components: product, situation, UX state, and user aspects. The *Product* aspect encompasses product characteristics, including features, services, and functions. The *Situation* aspect describes the contextual factors of product use, which can be categorized into time, activities, and locations. The *UX state* pertains to users' feelings about the experience. The *User* involves user-related information, categorized into user groups (e.g., experienced users and novices) and user backgrounds. Their approach uses a BERT-based method to extract UX-related data from review sentences and classify it into these semantic groups: *Product*, *Situation*, *UX state*, or *User*. Of particular interest to our research is the *User* category, which includes attributes such as age and occupation. This aligns with our goal of extracting personal attributes from product reviews. However, while their research addresses all four categories (*Product*, *Situation*, *UX state*, and *User*) and aims for a comprehensive UX extraction, our approach is more focused specifically on *User* attributes.

Tong et al. [2] also explore UX extraction from product reviews using the BERT model, similar to Yang et al. [1]. However, Tong et al. [2] specifically focus on extracting information related to the *context of use*. They categorize the context of use into three main components: Users, Tasks, and Environments. *Users* refer to the individuals using the product. *Tasks* denote the purposes or activities for which the product is used. *Environments* describe the settings or contexts in which the product is used.

Similarly, Yi Han et al. [3] also extract UX information from product reviews. However, unlike the aforementioned studies, Yi Han et al. [3] specifically aim to identify *the needs of individual users* of products. They start by scraping product descriptions to create a lexicon of product attributes, such as color, permeability, weight, stability, and durability.

In addition, they scrape product reviews and apply Named Entity Recognition (NER) to annotate and label these reviews, thereby extracting named entities. The next step involves integrating the annotated entities from the product reviews with the product attributes derived from the product descriptions. This approach enables them to effectively identify user needs on a large scale from online reviews.

In line with studies on UX extraction, Li et al. [4] also aim to extract UX information from textual data, specifically focusing on identifying *user problems*. Two key differences in their approach are the use of customer service chat logs as their data source instead of product reviews, and the employment of a knowledge graph as the output for storing and organizing the extracted data. They propose AliMe KG, a knowledge graph that captures user problems, item information, and more. Their ontology includes a concept where a "User" is associated with a "Problem", reflecting problematic states experienced by users. They use heuristic rules and phrase mining to extract these problems, followed by classification with a BERT classifier. Although both studies involve building a knowledge graph from free text, their approach retrieves data from customer service chat logs, whereas our approach focuses on product reviews. In addition, they focus on user problems, while our study aims to extract personal attributes. In their paper, they provide examples of user problems such as "pimple" or "dry skin", which could also be considered personal attributes in our study.

2.1.2 Personal Knowledge Graphs

Li et al. [5] proposes a pipeline-model for personal knowledge graph population from user utterances. The first step is a personal assertion classifier(SVM-model), this model predicts if a given statement contains relevant personal information. The second step is relation detection, assigning the assertion in to one of the relationship classes (SVMmodel). The final step is slot filling (CRFs).

Additionally, Chakraborty et al. [6] propose the idea of modeling a Personal Research Knowledge Graph as a labeled property graph using Neo4j, demonstrating a method for organizing personal data in a graph database. Neo4j enables flexible data modeling without a fixed schema, allowing new data to be added as new information is discovered.

Balog and Kenter [7] define a personal knowledge graph as a structured representation of entities related to an individual, including their attributes and the relationships between them. Similarly, Skjæveland et al. [8] discuss personal knowledge graphs, highlighting their role in integrating personal data. Personal knowledge graphs differ from our approach, as their application and purpose are broader. In addition, they include a wider range of private and public data, while we focus solely on public data.

Despite these differences, the methodologies of personal knowledge graphs and our research share a common goal. Both aim to structure and analyze the data to improve the understanding of individual characteristics. Personal knowledge graphs are relevant to our research because they illustrate the use of organizing and leveraging data to create comprehensive profiles. This aligns with our objective of using structured information from product reviews to gain insights into customer attributes. In particular, their use of public data sources in constructing personal knowledge graphs is especially relevant to us, as we rely on public data sources.

2.1.3 Product Recommendations

McAuley et al. [9] looked at two types of product recommendations that can be made which are referred to as substitutes and complements: *substitutes* are products that can be purchased instead of each other, while *complements* are products that can be purchased in addition to each other. They developed methods to automatically extract these relationships from product reviews. Although their focus is on product recommendations and our research centers on extracting personal attributes from reviewers, their approach is promising because the techniques for identifying relationships and patterns in textual data can be adapted to the extraction of personal attributes. Their methods provide a foundation for understanding how to derive structured information from unstructured text.

Da Xu et al. [10] developed a product knowledge graph (PKG) for e-commerce, focusing on various product relations such as complement (co-buy), co-view, and substitute. For instance, their PKG identifies a Soundbar as having a complement relationship with a Television, suggesting that these items are frequently purchased together. In contrast, two televisions might be linked by a substitute relationship. The dataset used to construct this PKG includes product catalog information (e.g., descriptions, categories) and raw user-product interaction records. A notable conclusion from their study is the potential for future research to incorporate customer knowledge, such as demographic information, into the PKG, thus creating a customer-product knowledge graph. Although this integration has not yet been implemented by the researchers, the concept of a customer knowledge graph is relevant to our research.

2.1.4 Other Data Sources

Construction-related Regulatory Documents

Zhang et al. [11] propose a novel approach to the extraction of semantic information that integrates domain-specific knowledge with syntax-related features to automatically extract and structure information from unstructured data. Their particular focus is on construction-related regulatory documents and they use ontologies to represent domain knowledge, enabling the extraction of information by understanding the relationships between concepts.

Research articles

Sateli et al. [12] applied text mining techniques to extract competences from full-text research articles. They utilized DBpedia Spotlight, a NER tool, to identify competences as named entities within the text. These identified entities were then represented as semantic triples in RDF format, enabling structured knowledge representation. The next step in their approach involves using the extracted competences to build user profiles for researchers. Typically, constructing user profiles requires gathering user information over an extended period, a process known as user profiling. Although the context of Sateli et al.'s research differs from ours, it is noteworthy that they apply text mining techniques to construct user profiles. This approach could potentially be adapted to other settings, such as product reviews, thereby facilitating the creation of user profiles for product reviewers.

User Profiling

With the increasing volume of product reviews and advancements in text mining, it is becoming more feasible to understand and profile users based on the content they have written in these reviews. Such profiling can encompass both demographic and personal characteristics, including age, name, knowledge, or expertise, as well as deeper traits like preferences, interests, and behavior. This has emerged as a significant research direction in the field of business intelligence. While this is beyond the scope of our research, our findings could contribute to this evolving area of study.

2.1.5 Personal Attributes from Conversation

Mohanaraj et al. [13] propose a framework for extracting personal information from conversational data, focusing on two primary tasks: Triple Extraction and Entity Linking. Triple extraction organizes personal information into triples, where each triple consists of two entities and a relationship, such as *HasName*, *HasAge*, *HasGender*, *SchoolStatus*, or *JobStatus*. Entity linking enhances triples by applying various techniques, such as coreference resolution and string similarity, to link the entities with external knowledge graphs. In this step, the framework connects extracted entities to either a personal knowledge graph, which is incrementally built during the conversation and includes previously identified personal entities. Alternatively, it links entities to general-purpose knowledge graphs, such as ConceptNet, which provide additional contextual information and enrich the data.
Tigunova et al. [14] present the CHARM model, an approach developed to extract personal attributes from conversational dialogues. This model is particularly focused on extracting specific attributes such as "profession" and "hobby". CHARM works by extracting phrases from text that potentially contain personal attributes. These extracted phrases are then aligned with structured data from Wikipedia lists to infer personal attributes. CHARM has been tested in various applications, including interactions with chatbots and the analysis of social media posts. It has also been applied to author profiling, which involves determining personal attributes and characteristics of authors based on their written content.

The Persona-Chat dataset, introduced by Zhang et al. [15], comprises dialogues where each speaker is given a predefined persona or profile. Each pair of speakers engages in conversations conditioned on their respective profiles. The Persona-Chat dataset includes both dialogue exchanges and detailed persona profiles for each speaker. The dialogues consist of actual conversations between speakers, while the persona profiles provide predefined descriptions of each speaker's background, preferences, and personality traits. This combination facilitates the examination of how persona attributes influence conversational content and interaction patterns.

Wu et al. [16] use the Persona-Chat dataset to develop an approach for automatically extracting user attributes from dialogues. They define user attributes as explicit representations of an individual's identity and characteristics in a structured format. The goal of their approach is to predict user information from a given utterance, representing it in a (Subject, Predicate, Object) triplet format. The study leverages conversational data to extract these attributes, aiming to enhance user understanding. To accomplish this, the authors fine-tune the BERT model.

Wang et al. [17] focus on the task of extracting and inferring personal attributes from dialogue. They define personal attributes as structured information about a person, such as their hobbies, pets, and family. They propose representing these attributes as knowledge graph triples. The task of persona extraction is divided into two subtasks: extraction and inference. The extraction subtask involves identifying phrases in an utterance that indicate personal attributes, while the inference subtask requires models to predict personal attributes that are not explicitly stated verbatim in the utterance. Wang et al. trained their model using an autoregressive language model on the persona extraction dataset DialogNLI. They demonstrate that their model outperforms baselines in the extraction of personal attributes from dialogues. A dataset for persona extraction is also provided by Wang et al. [17].

Leveraging this dataset, Zhu et al. [18] employed a variational auto-encoder to generate new synthetic examples for model training. They introduced a BART-based zero-shot model, named PAED, and also released dataset, referred to as PersonaExt.

2.1.6 Personal Attributes Extraction from Product Reviews

The study called "*Extracting Data-Driven User Segments and Knowledge by Using Online Product Reviews*" [19] by Güneş closely aligns with our research, as both aim to extract personal attributes from product reviews.

Güneş's research closely aligns with our study, as both aim to extract personal attributes from product reviews. Although Güneş employs a fixed list of personal attributes and different extraction methods compared to ours, the insights gained are highly relevant and provide valuable context for our work.

Although the methods employed differ from ours, the insights gained are highly relevant and provide valuable context for our work. Güneş investigates methods for creating user profiles from product reviews scraped from Amazon in his study. Notably, the approach involves scraping all reviews from a public profile to gain a comprehensive view of the reviewer's attributes. The author notes that analyzing a single review often provides limited information about personal attributes. To overcome this limitation, all reviews from a public profile are aggregated. In other words, the complete list of reviews for a given profile is used to construct a more accurate picture of the reviewer's personal attributes.

Most of the personal attributes Güneş seeks to extract from product reviews are highly relevant to our research. In particular, he focuses on the following attributes: gender, age, occupation, marital status, residence, pet ownership, and details about pets and children. Various techniques are used for extraction. Age is determined using the Allen NLP NER model. Gender is identified with Damegender software. Marital status is extracted through keyword-based pattern matching. Occupation and residence are extracted using the Meaning Cloud Topic Extraction API and classification. Pet ownership and child-related information are analyzed with Topic Extraction and keyword-based algorithms. Subsequently, the extracted attributes are used to cluster similar reviewers. A clustering tool identifies natural groupings within the data, creating demographic clusters that aid in identifying customer groups based on personal attributes. In the final analysis, clusters are examined by aggregating opinions and evaluating both positive and negative sentiments to derive insights into overall sentiment trends.

While Güneş's research is relevant to our study, it primarily focuses on a fixed list of personal attributes. In contrast, our work is different in that it explores the use of LLMs to extract a broader range of personal attributes, such as skin condition and hair type, offering greater flexibility and depth in attribute extraction.

2.2 Problem Formulation

Definition of Knowledge Graph (KG). A knowledge graph (KG) is a directed, labeled graph data structure designed to represent structured and related information. It is formally defined as a graph $G = (E, R, L)$, where:

- E is the set of entities,
- R is the set of directed relations between entities,
- L is the set of labels, with each label l mapping pairs of entities and relations $(E \times R)$ to a label in L.

Entities in the knowledge graph can be either concrete objects or abstract concepts. Concrete entities include tangible items such as persons or objects, while abstract entities represent non-physical concepts. For simplicity, both types are collectively referred to as entities in this research. For instance, although "Age", "Skin Condition", and "Hair Type" are abstract concepts, they are treated as entities within the knowledge graph for the purposes of this study.

Definition of Triplets. A triplet in a knowledge graph consists of three elements:

- **Source Entity:** The starting point of the relationship, characterized by a unique identifier and an associated entity type.
- **Relation:** The type of relationship or connection between the head and tail entities.
- **Target Entity:** The endpoint of the relationship, characterized by a unique identifier and an associated entity type.

A triplet is denoted as (source entity, relation, target entity). For example, in the triplet (John, has_hair_type, Curly), "John" is the source entity with the type Person, "has_hair_type" represents the relation, and "Curly" is the target entity with the type Hair Type.

Problem Definition 1: Extraction of Triplets. Given a sentence S, the goal is to extract triplets in the form of (head entity, relation, tail entity) that capture personal attribute information. For example, from the sentence "John is 30 years old," the extracted triplet would be (John, has_Age, 30).

Problem Definition 2: Constructing a Knowledge Graph. Given a set of extracted triplets, the objective is to construct a knowledge graph where the entities represent product reviewers, products, and reviewer personal attributes. This involves organizing the triplets into a structured graph where relationships between entities are clearly defined. For instance, the knowledge graph would link a reviewer to a product they reviewed and include attributes such as their age and gender.

Chapter 3

Approach

This research aims to extract personal attributes from customer reviews. To achieve this, we have developed an end-to-end pipeline that applies various NLP techniques. The unstructured reviews are transformed into a graph-based representation, enabling further analysis and data exploration. This chapter begins with an overview of the pipeline, followed by a detailed, step-by-step description of each component.

3.1 Pipeline

In this study, we present an end-to-end information extraction pipeline. The goal of information extraction is to transform unstructured text data into structured data that can be easily analyzed, searched, and visualized. The main focus of this study is to perform information extraction on customer reviews. More specifically, our aim is to extract personal attributes of the reviewers, allowing for a deeper understanding of their characteristics and facilitating market segmentation.

3.1.1 Graph Representation

The pipeline processes customer reviews, transforming them from unstructured text into structured data. The information extraction pipeline results in a graph representation of the information. The nodes represent entities, and the connecting lines denote the relationships between these entities. An example is shown in Figure 3.1, where the input is a review, and the output is a graph representation showing structured information.

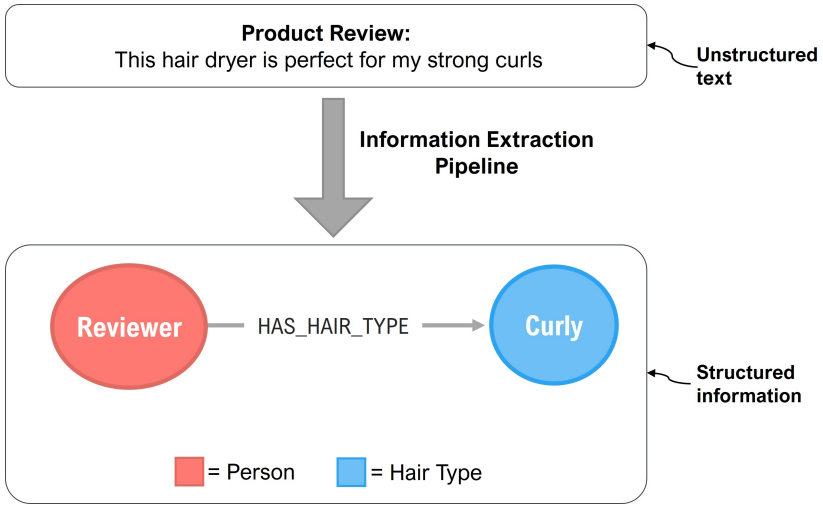

Figure 3.1: The goal of the information extraction pipeline is to extract structured information from unstructured text

By running the pipeline on a large number of reviews, a comprehensive graph representation is generated that visually captures the information extracted from those reviews. This detailed graph organizes and illustrates the data, highlighting relationships and entities identified in the reviews. Specifically, the graph aims to showcase the personal attributes of the product reviewers. This graph facilitates the analysis of patterns and relationships, offering valuable insights into different types of reviewers and providing a clearer understanding of reviewer profiles.

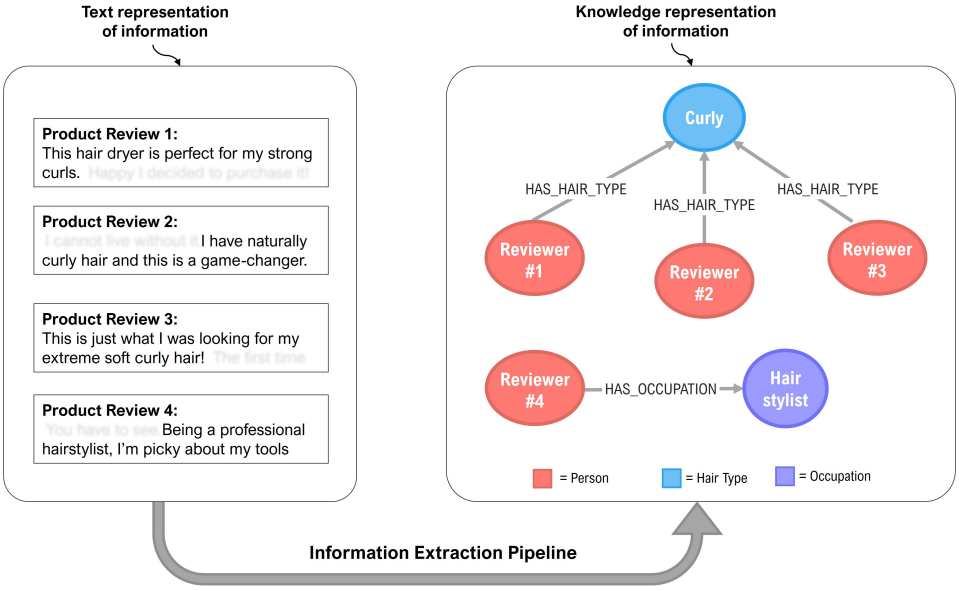

Figure 3.2: Transformation of multiple product reviews into a structured graph representation.

The information extraction process can be broken down into smaller processing stages. Therefore, we are designing it as an end-to-end pipeline, allowing us to incorporate modular components that focus on individual tasks. Designing the process as a modular pipeline simplifies future updates, as each component can be independently modified or replaced without impacting the others. This also offers the option to add new processing steps in the future, since new steps can be integrated without having to manage complex dependencies.

3.1.2 Overview of the pipeline

Figure 3.3 provides a visual representation of the pipeline steps. This diagram illustrates the sequential process used to extract structured information from unstructured customer reviews. We will now provide a brief overview of each step in the pipeline.

- **Step 1: Prepend Prefix**
  - **Problem**: As part of our pipeline, we will perform coreference resolution to identify and replace expressions referring to specific entities. However, an issue must be addressed before we can execute this step. Many product reviews use pronouns without explicitly mentioning corresponding entities, making it difficult for the model to resolve coreferences. Without a clear reference, the model lacks the context to replace pronouns.
  - **Solution**: To address this, we start with Prefixing, where we prepend the prefix "`Jane says:`" to each review. This lexical addition informs the model that the reviewer, referenced by the pseudonym "Jane" is the one providing the review. The name "Jane" serves as a universal pseudonym for all reviewers. This prefix

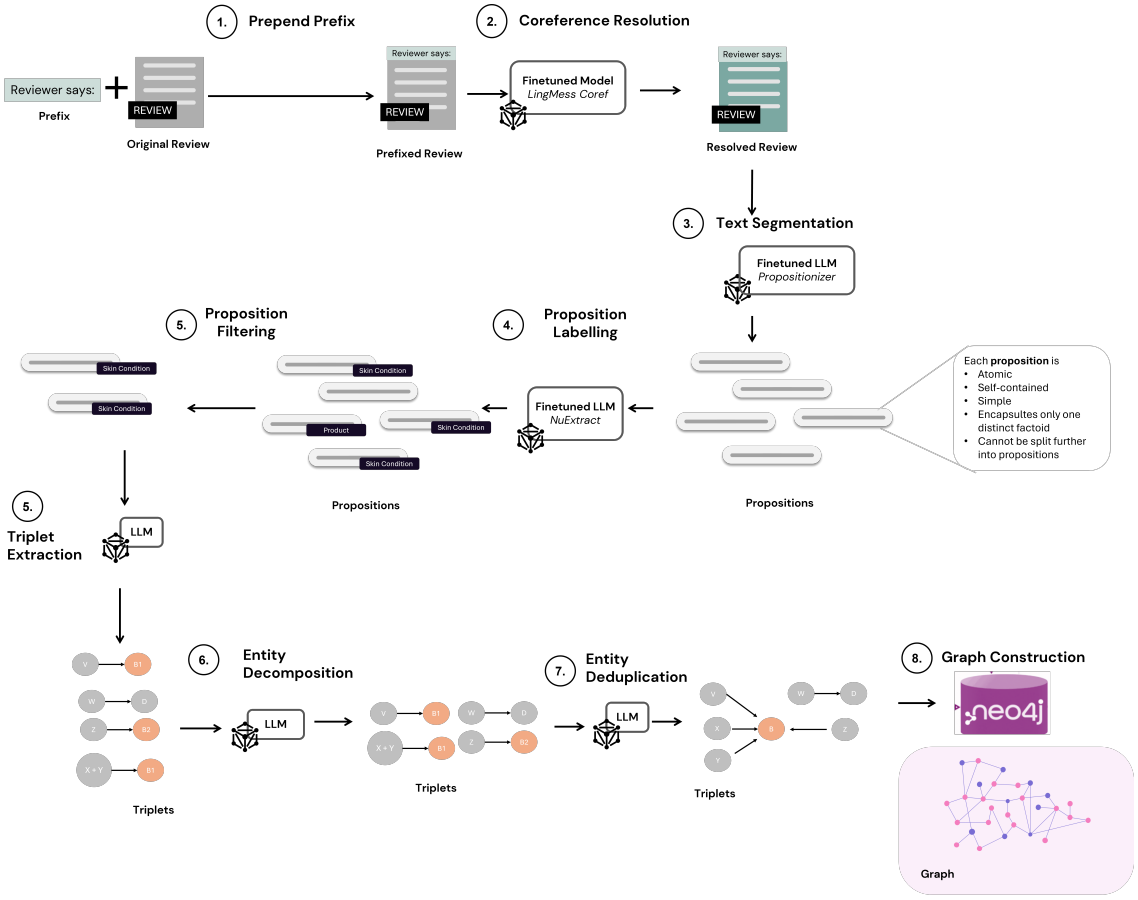

Figure 3.3: Graphical representation illustrating all steps in the pipeline.

provides an explicit reference to the reviewer, which is important for the next coreference resolution step, where pronouns will be replaced.

- **Step 2: Coreference Resolution**
  - **Problem**: In text, different words or phrases may refer to the same entity. In the graph produced by our pipeline, all these references need to be recognized as pointing to a single entity, ensuring that they are treated as one single entity instead of separate ones.
  - **Solution**: Therefore, in this step, the prefixed review is processed through a coreference resolution model, which identifies and replaces referring expressions, such as pronouns, with their corresponding entities. Since we have added "Jane" in the prefix, the coreference resolution model will replace instances of "I" and "my" with "Jane".
- **Step 3: Text Segmentation**
  - **Problem**: Customer reviews often contain long, complex sentences that intertwine multiple pieces of information, making triplet extraction using LLMs challenging. When a sentence includes various facts, the LLM may only extract some as triplets while missing others. Consequently, not all relevant information is captured in the extracted triplets, resulting in a loss of valuable data from the review. To address this issue, we first perform text splitting, producing shorter text segments that can then be provided to the LLM for triplet extraction.
  - **Solution**: We provide the coreference-resolved product review to the Propositionizer model [20]. The output consists of a set of propositions, each representing a short, independent sentence. We expect each proposition to describe exactly

one contextualized atomic fact. Necessary context is incorporated within each proposition, allowing its meaning to be interpreted independently of the original text.

- **Step 4: Proposition Classification**
  - **Problem**: In the previous step, we generated a list of propositions for each review. In Step 5, we will focus on proposition filtering. However, to effectively carry out this filtering, we first need to classify the propositions into predefined categories. The challenge lies in effectively performing multi-class classification to accurately assign each proposition to one or more of the predefined classes.
  - **Solution**: We will address this challenge by employing a fine-tuned large language model called NuExtract to perform named entity recognition (NER). The extracted entities will then be utilized to carry out multi-class classification. Each proposition will be assigned to one or more predefined classes, or it may be classified as having no class if it does not fit any category.
- **Step 5: Proposition Filtering**
  - **Problem**: After Step 3, Text Segmentation, we obtain a list of propositions for each product review. While it is possible to input this list directly into the triplet extraction process, doing so is impractical and unscalable due to the significant number of propositions associated with each review.
  - **Solution**: To address this issue, we perform proposition filtering to decrease the number of propositions that need to be processed in the Triplet Extraction step. In the previous step, we classified each proposition and will now filter and retain only those propositions that have at least one assigned class.
- **Step 6: Triplet Extraction**
  - **Problem**: After pre-processing steps, including coreference resolution and text splitting, we have a set of propositions for each product review. However, these propositions are still unstructured text, making it challenging to extract meaningful insights and identify relationships between entities. Thus, we want to extract structured information from this unstructured text.
  - **Solution**: Triplet Extraction is performed to extract structured triplets from the propositions, capturing key relationships and entities. A LLM will be used to perform this triplet extraction.
- **Step 7: Entity Decomposition**
  - **Problem**: Composite entities combine various concepts into a single entity. For example, "long curly" is a composite entity with the type `Hair Type`, as it merges multiple hair types into one entity.
  - **Solution**: We perform to Entity Decomposition, which involves breaking down composite entities into their separate and distinct components. During this process, each composite entity is deconstructed to isolate and identify each individual concept. In the example, this would result in separate entities labeled as "Long" and "Curly". This step ensures that each entity represents a single, distinct concept.
- **Step 8: Entity Deduplication**
  - **Problem**: In contrast to the problem addressed in the previous step, Entity Deduplication deals with cases where separate entities actually represent the same concept. For example, there may be an entity labeled "Curly" and another

labeled “strong curls”, both classified under the type `Hair Type`. Although these are distinct entities, they actually represent the same hair type and should be merged into a single entity with a label that encompasses both.

  - **Solution**: The goal of this step is to merge redundant entities that represent the same concept into a unified entity. This eliminates duplicates and ensures a more accurate and cohesive representation of concepts in the final graph, simplifying the graph structure and making analysis easier. For example, the two entities “Curly” and another labeled “strong curls” would be merged into one single entity, such as “Curly”.

- **Step 9: Graph Construction**

  - **Problem**: The previous steps result in a large number of triplets. Each triplet includes a source entity (or source node), a target entity (or target node), and a relationship connecting these two nodes. For downstream tasks such as analysis, we want to aggregate these triplets to reveal broader patterns and connections within the data. Thus, we need a structured framework that organizes and aggregates the triplets effectively.
  - **Solution**: In this final step, Graph Construction, we use the extracted triplets to build a graph. This graph provides a structured representation of entities and their relationships, creating nodes for each unique entity and edges to illustrate the relationships defined by the triplets. By organizing the information in this graph, we facilitate downstream tasks such as querying and analysis.

We have now presented an overview of the various steps involved in the pipeline. In the following sections, we will provide a comprehensive and detailed explanation of each component, delving into the methodologies used at every step.

3.2 Prefixing and Coreference Resolution

The pipeline begins with Step 1, Prefixing, followed by Step 2, Coreference Resolution. This section aims to explain the motivation behind implementing these steps, along with a detailed explanation of each. Figure 3.4 provides an example that illustrates these steps.

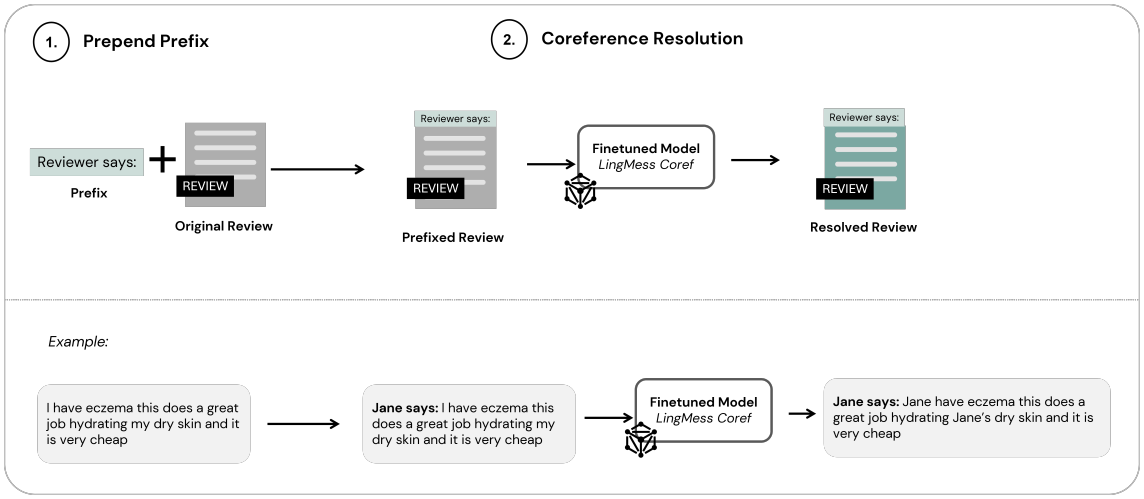

Figure 3.4: Graphical explanation of the steps: Prepend Prefix and Coreference Resolution, along with an example.

Motivation

Coreferences occur when two or more expressions within a text refer to the same entity. Coreference resolution is the task of determining whether two or more referring expressions are used to refer to the same entity. It is essential for improving the accuracy of tasks that rely on understanding relationships within text, such as triplet extraction. When converting reviews into triplets, our goal is to create triplets that are both meaningful and independent. To achieve this, each element of the triplets must be distinct and self-contained. This means that each component should be clearly defined and not depend on contextual information for interpretation. For example, instead of using ambiguous expressions or pronouns that refer back to other parts of the text, each triplet should explicitly name the entities. This ensures that the triplets are straightforward and can be understood on their own. Additionally, the same name should be used consistently for a component if it refers to the same entity. This consistency ensures that each entity is represented uniformly throughout the extracted triplets.

Problem

A referring expression in a sentence may refer to entities that are *explicitly mentioned* elsewhere in the text. To address this, we perform coreference resolution, which involves identifying and linking all expressions in the text that refer to the same entity. This process ensures that referring expressions, such as pronouns, are correctly replaced with their corresponding entities.

However, expressions can also refer to entities that are *not explicitly mentioned* but are implied through context. In many product reviews, pronouns referring to the reviewer can be found, but there is often a lack of explicit mentions of the reviewer. In such cases, coreference resolution alone is insufficient because the entities are not clearly stated. In other words, coreference resolution is not possible in these situations because referring expressions, such as pronouns, cannot be replaced with the entities to which they refer if those entities are not explicitly mentioned.

Solution

To resolve this, we introduce an additional step: **adding a contextual prefix** to each review. Specifically, we prepend the phrase "`Jane says:`" to the beginning of each review. This approach establishes a clear reference to the reviewer. By using the pseudonym "`Jane`", we provide a distinct reference to the reviewer while maintaining anonymity. Adding a prefix to our product review ensures that there is always a clear mention of both the product reviewer and the product itself. Consequently, coreference resolution becomes possible because referring expressions, such as pronouns, can now be replaced with the specific mentions of the reviewer.

3.3 Text Segmentation

The overall goal of the pipeline is to convert product reviews into a structured format, specifically by storing them in a graph. To achieve this, product reviews will be converted into triplets using a LLM, as detailed in Section 3.5. While it is technically possible to provide the entire product review to the LLM and then use it to extract triplets, the results of triplet extraction can be improved by first splitting the input text. The following paragraph will explain why splitting is crucial before performing triplet extraction. After that, a more detailed explanation of the approach will be provided.

Problem

It is important to note that product reviews often contain long, complex sentences with multiple pieces of information intertwined. Providing the entire review to the LLM at once can lead to difficulties in parsing and accurately extracting all triplets due to the

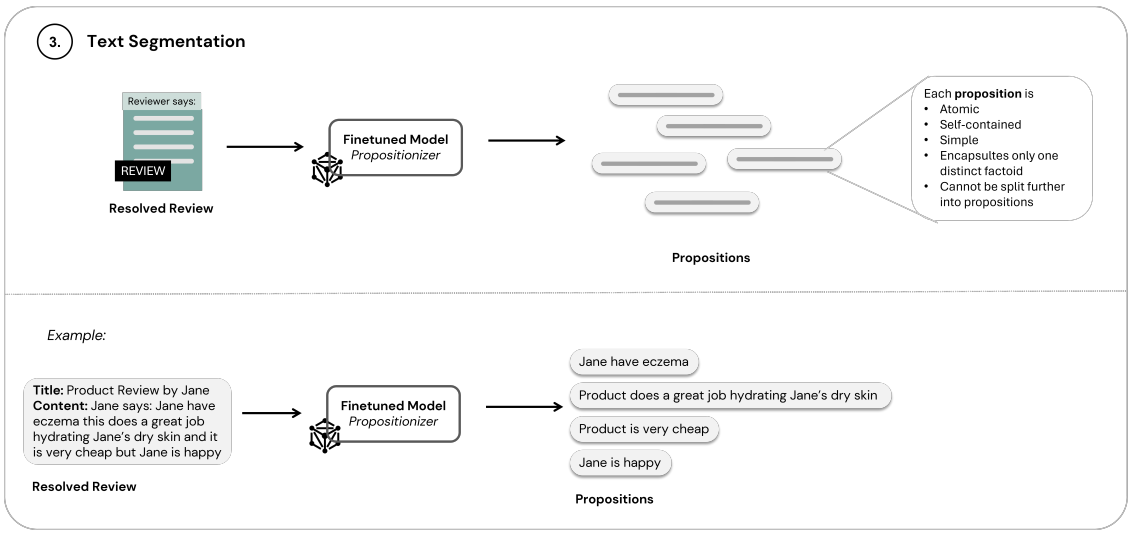

Figure 3.5: Graphical explanation of the step: Text Segmentation, along with an example.

complexity and length of the text. Triplet extraction depends on clear, concise sentence structures to correctly identify source-relation-target relationships. Consequently, complex sentences may result in incorrect or incomplete triplet extraction. If a sentence contains multiple pieces of information, the LLM might only extract some of the facts as triplets while missing others. This can result in incomplete extraction where not all relevant information is captured, leading to a loss of valuable data from the review. To address this problem, text segmentation will be performed, as explained in the next section.

Solution

To ensure that as much information as possible from a product review is converted into triplets and thus stored in the graph, we need to break the review into smaller parts. This process is called **text segmentation**, where we split the review into short, independent sentences. Segmenting the text ensures that each triplet can be more easily identified and extracted.

Segmentation is a complex problem. A straightforward method could be to split the product review into sentences. However, sentences can still be complex with multiple pieces of information intertwined. Thus, we need to further split the sentences. but this brings up two challenges.

- The first challenge is dealing with the retrieval granularity, which involves deciding how small to make the parts when we split the review. If the parts are too big, we might miss important details. If they are too small, it might be harder to understand the information. Finding the right balance is important for accurately splitting the review.

- The second challenge is that after splitting a review into sentences, these sentences are often not self-contained and lack necessary context. This means that if we break the review down into smaller parts, those parts might still not make complete sense on their own, making it difficult to convert them into useful triplets for the knowledge graph.

To address these challenges, we use the model **Propositionizer** presented in [20]. The input text is split into so-called propositions, which are defined as atomic expressions within the text, each encapsulating a distinct factoid and presented in a concise, self-contained natural language format. Propositions represent atomic expressions of meanings in text and are defined by three principles:

- Each proposition should correspond to a distinct piece of meaning in the text, where the composition of all propositions represents the semantics of the entire text.
- A proposition should be minimal, meaning it cannot be further split into separate propositions.
- A proposition should be contextualized and self-contained, including all necessary context from the text (e.g., coreference) to interpret its meaning.

Each product review will be processed by the Propositionizer, generating a list of propositions for every review. This list will then be used as the input for the next step, which involves extracting triplets from the propositions.

3.4 Proposition Classification and Filtering

In the previous step, a list of propositions is generated for each review. While we could proceed directly to the Triplet Extraction step, this approach would be impractical due to the large number of propositions. When a large number of triplets need to be generated for each product review, it results in a substantial amount of data that takes a long time to process, making it unscalable. To address this issue, the following steps involve first classifying each proposition into one of the predefined classes, or no class. We then keep only the propositions for which a class was assigned.

3.4.1 NER

The method employed for text classification in this study is based on NER. More specifically, we classify text using the entities extracted by NER. Specifically, we classify text using the entities extracted by NER. While our approach will be explained in the next section, let us first clarify what constitutes a named entity.

Several studies restrict the definition of a named entity to a "real-world object" with a distinct name, such as a person, a country, a product, or a book title. However, this study adopts a broader definition. For our purposes, a ***named entity*** is any phrase that distinctly identifies an item within a set of similar items. This includes personal attributes such as age, skin condition, and hair type, which are considered entities. Although these may not be "real-world objects", they are considered entities according to our definition because they clearly identify distinct items within their respective categories.

Traditional NER models are limited to predefined entity types, and expanding these types often requires costly and time-consuming data labeling and model retraining. LLMs have revolutionized NER by enabling the extraction of various entity types using natural language instructions. However, LLMs demand significant computing resources, and scaling their use can be expensive. Recently, task-specific foundation models have emerged. These models are tailored for specific tasks, such as entity recognition, and often outperform larger, more generic models in the tasks for which they are specialized. In this study,

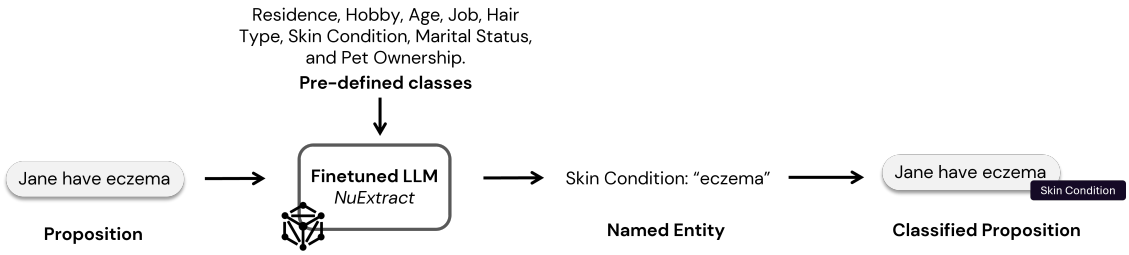

Figure 3.6: Example illustrating how the NuExtract model is first used to extract a named entity, which is then used to classify the proposition.

we will use a task-specific foundation model, specifically NuExtract, which is fine-tuned for entity recognition.

3.4.2 Proposition Classification

Text classification involves categorizing text into predefined classes. In this study, we use a fine-tuned large language model called NuExtract to perform text classification. The process consists of two steps.

- In the first step, a proposition and a list of predefined classes are provided to NuExtract. Given that NuExtract is fine-tuned for NER, it returns named entities, each consisting of a type and a value.
- Then, in the second step, the extracted entities from NuExtract will be used to perform classification. Specifically, if NuExtract returns a value for a specific type, the proposition is classified according to that type.

Given that this study focuses on extracting personal attributes, the list of predefined classes might include, for example, Residence, Hobby, Age, Job, Hair Type, Skin Condition, Marital Status, and Pet Ownership. To demonstrate the process of text classification, we will consider an example. Given the proposition "Jane has eczema", NuExtract returns the entity with the value "Eczema" and the type "Skin Condition". Therefore, the proposition "Jane has eczema" will be classified as "Skin Condition". This example is visually shown in Figure 3.6.

It is important to note that we perform multi-label classification. If NuExtract returns multiple entities with different types for a proposition, it indicates that multiple classes will be assigned to that proposition. For example, given the proposition "Jane is a 19-year-old girl with acne", NuExtract returns two entities: one with the value "19" and type "Age", and another with the value "acne" and type "Skin Condition". As a result, the proposition will be classified with the classes "Age" and "Skin Condition".

3.4.3 Proposition Filtering

Following the proposition classification step, where propositions have been categorized, we now proceed with Proposition Filtering. This process involves retaining only those propositions that are classified into at least one of the predefined classes. By filtering out sentences without classification, we streamline the subsequent Triplet Extraction step, enhancing the overall efficiency and scalability of the pipeline. By focusing on propositions with identified classes, we improve the quality of the extracted triplets, as these propositions are more likely to provide meaningful and relevant information for our primary objective: extracting personal attributes.

3.5 Triplet Extraction

The goal of this step is to extract triplets from unstructured text. To achieve this, we will leverage recent advancements in pre-trained LLMs. In the previous steps, the product review was segmented into a list of propositions. Following the proposition filtering step, triplet extraction will be performed only on those propositions containing at least one class. For each proposition in this filtered list, we will execute the triplet extraction process.

By processing each proposition separately, we ensure that every proposition is accurately converted into a triplet. While it is possible to provide larger text chunks to the LLM for triplet extraction, this method may result in incomplete extraction, as not all information from the larger text chunk might be converted into triplets. Therefore, we perform the triplet extraction process on each proposition separately. This involves generating and processing a distinct LLM prompt for each proposition.

Figure 3.7: Example illustrating how an input text, which is a proposition, is processed to extract a triplet using an LLM.

The LLM prompt, as shown below, will be used. In this prompt, the placeholder `input_sentence` will be replaced with the actual sentence that we are running the triplet extraction for. In a previous step, we performed NER on this proposition, which provided named entities along with their IDs and types. We incorporate the entity type provided by NER into our LLM prompt by replacing the `target_type` placeholder with the information obtained from the NER step.

Listing 3.1: LLM Prompt for Triplet Extraction

```
You are a top-tier algorithm designed for extracting information in
structured formats to build a knowledge graph. Your task is to identify the
entities and relations requested with the user prompt from a given text.
You must generate the output in a JSON format containing a list with JSON
objects. Each object should have the keys: 'head', 'head_type', 'relation',
'tail', and 'tail_type'.

For the following text, extract entities and relations.
Text: {input_sentence}

Target Entity:
- Target ID: {target_id}
- Target Type: {target_type}
```

The prompt specifies the desired format for the LLM's response. However, the LLM is not guaranteed to give back the results in this format. LLMs can be unreliable in returning well-formed JSON data, occasionally omitting parentheses or including extra words. Therefore, we will use the Python package json_repair to correct any invalid JSON strings. We will iterate over each output in the list of all reponses, and each response is properly formatted to list of entities and a list of triplets using json_repair. The final output of this step will be a complete set of extracted triplets for all the sentences provided as input.

3.6 Entity Decomposition

The previous step, Triplet Extraction, produces triplets with a source node, a target node, and their relationship. If a node contains multiple concepts, we want to split it into distinct entities. This step, called Entity Decomposition, is important for ensuring each node accurately represents a single concept.

Figure 3.8: Example illustrating how the composite node "long naturally curly hair" is split into two separate nodes: "long" and "naturally curly hair."

Problem

A challenge with triplet extraction using the LLM is the presence of composite entities in the extracted tripltets. Composite entities are defined as entities that encapsulate multiple concepts within a single entities. For example, an entity with the label "long curly hair" is a composite entity. In a well-structured triplet extraction system, each entity should ideally represent a single, well-defined concept to maintain clarity in data representation. So, we would expect to have two separate entities, for example, "Long" and "Curly", because these two represent different hair types. A major problem with composite entities is that they do not align with the principle that each entity should represent one single concept, and thus this might be a problem for downstream tasks such as analyzing the data.

Another issue with composite entities is related to the next post-processing step, "Entity Deduplication", which will be discussed in Section 3.7. The goal of this step is to merge entities that represent the same concept. When entities represent multiple concepts, it becomes harder to determine if two entities are truly the same. For instance, if one entity represents "long curly hair" and another represents "strong curls," it is not straightforward to identify the similarity. Composite entities have multiple concepts that need to be reconciled during the merge. This adds complexity to determine if two entities are indeed representing the same concept and thus should be merged. Merging composite entities effectively requires separating the multiple concepts they represent first, which is the goal of this step.

Solution

To handle composite entities that encapsulate multiple concepts, we will perform Entity Decomposition, which involves breaking down the composite entity into distinct, individual entities. To achieve this, we will use an LLM. The prompt, as shown in 3.3, will be provided to the LLM, where the placeholder `{text}` is replaced with the actual input text to be split.

Listing 3.2: LLM Prompt for Entity Decomposition

```
Extract the entities mentioned in the following sentence and return them as
a comma-separated list.

Input: {text}
```

This approach leverages the LLM's ability to break down entity labels containing multiple concepts into a comma-separated list of distinct entity labels. For instance, if the input text is `"thick high porosity hair"`, the output will be `["thick", "high porosity hair"]`.

The outcome of the LLM determines our next steps. If the LLM response contains only a single item, the original triplet remains unchanged. However, if the LLM response contains multiple items in a comma-separated list, we proceed with decomposition. In this case, we create a separate entity for each item in the list. For each newly created entity, we generate a new triplet. These new triplets maintain the same relationship and head entity as the original triplet, but with the tail entity replaced by one of the decomposed entities. This decomposition process ensures that composite entities are decomposed, while preserving the original relationships within our triplets.

3.7 Entity Deduplication

In the Triplet Extraction step, we obtain triplets containing nodes. A challenge arises when multiple distinct nodes represent the same concept. In this step, known as Entity Deduplication, we aim to consolidate these nodes into a single entity. Mohanaraj et al. [13] propose a framework for extracting personal information from conversational data,

focusing on two primary tasks: Triple Extraction and Entity Linking. Triple extraction organizes personal information into triples, where each triple consists of two entities and a relationship, such as *HasName*, *HasAge*, *HasGender*, *SchoolStatus*, or *JobStatus*. Entity linking enhances triples by applying various techniques, such as coreference resolution and string similarity, to link the entities with external knowledge graphs. In this step, the framework connects extracted entities to either a personal knowledge graph, which is incrementally built during the conversation and includes previously identified personal entities. Alternatively, it links entities to general-purpose knowledge graphs, such as ConceptNet, which provide additional contextual information and enrich the data.

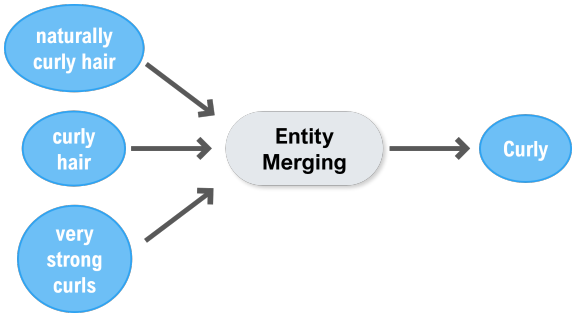

Figure 3.9: Example illustrating how three separate nodes, all representing the same hair type, are merged into a single node.

Problem

In the previous step, Entity Decomposition, we addressed entities that represented multiple distinct concepts by splitting them accordingly. Conversely, another issue with the extracted triplets is that some separate entities actually represent the same concept. This is because the LLM uses synonymous keywords for the same concept in the triplets. Having more distinct entities representing the same concept will lead to a more complex graph structure. This will complicate downstream tasks such as querying or reasoning. To address this, we need to merge these redundant entities into a single, unified entity.

Solution

To address the challenge of multiple entities representing the same concept, we implement a pairwise merging process using a LLM. This approach is applied independently for each entity type to ensure accurate merging. The process begins by collecting all unique entity labels for a given entity type. The next step is the creation of keyword embeddings. For each unique entity label, we generate an embedding using the SentenceTransformer model. These embeddings provide a dense vector representation of the semantic content of each label. We then conduct pairwise comparisons, calculating the cosine similarity score for each pair of embedded labels. This allows us to calculate the semantic similarity between different entity labels. We establish a threshold of 0.9 for the cosine similarity score, with pairs exceeding this threshold considered candidates for merging. We take the top 10 pairs that surpass the 0.9 threshold for the current iteration. It is important to note that this is an iterative process, allowing for unmerged pairs to be reconsidered in subsequent rounds.

For the selected pairs, we employ an LLM to perform the actual merging task. The prompt, shown below, is used to generate a concise, merged keyword or keyphrase that accurately represents both entity labels in the pair being processed.

This solution leverages the LLM's capabilitiaes to merge high-similarity pairs into consolidated keywords, effectively reducing redundancy in our entity set. This is an iterative process, and we keep doing this until there are no more keyword pairs left with a similarity score higher than the threshold.

Listing 3.3: LLM Prompt for Merging

```
Given two phrases describing the same concept, provide a concise, merged
keyword or keyphrase that accurately represents both.
Input: {keyword_pair}
```

3.8 Graph Construction

In this step, the extracted triplets will be used to construct a graph, a visual representation that organizes and illustrates the relationships and entities derived from the reviews. The preceding steps have enabled us to extract structured information from text, specifically by transforming unstructured customer reviews into triplets that represent relationships and attributes. The final step involves storing this structured information in a graph database, which will facilitate subsequent analytical and retrieval tasks. For this study, we will use Neo4j as our graph database. Neo4j offers robust graph storage capabilities and support for complex queries on interconnected data. By storing the triplets in Neo4j, we can efficiently manage and query the graph, enabling effective downstream applications.

Chapter 4

Implementation

While the previous chapter outlined the approach and described each step, this chapter will provide a more detailed account of the implementation. It will introduce the tools and packages used, along with code snippets to illustrate the practical execution of the approach. The chapter is structured step by step, meaning that each stage of the process will be covered in sequence.

4.1 Code

The implementation has been written in Python, and the code is available upon request. The code allows to run as batch functions in Amazon Web Services (AWS).[1] Batch processing which offers several advantages for large-scale processing. AWS Batch allows for the execution of jobs in parallel, optimizing compute resources based on job requirements. This is particularly beneficial when working with extensive datasets, as it enables the system to automatically allocate the necessary resources without manual intervention. To handle data processing, Polars[2] is used due to its high performance and ability to handle large datasets efficiently. Polars offers faster operations compared to traditional data manipulation libraries, making it particularly suitable for large-scale data pipelines.

4.2 Implementation Details

This section will discuss each step of the implementation process, providing detailed descriptions of the methodologies employed. The implementation of each step will be explained, the tools used will be mentioned, and relevant code snippets will be included for clarity.

4.2.1 Coreference Resolution

We use the fastcoref[3] Python package for fast, accurate, and easy-to-use coreference resolution in English. Introduced by Otmazgin et al. [21], the fastcoref package is designed for scalability, making it an excellent choice for large-scale applications requiring coreference resolution. The LINGMESS model[4] enhances accuracy by incorporating linguistic expertise into the coreference resolution process. LINGMESS stands for Linguistically Informed Multi Expert Scorers for Coreference Resolution, and it was also introduced by Otmazgin et al. in a different paper [22]. Finally, we also use SpaCy[5], a natural language processing library known for its fast, efficient, and easy-to-use tools for processing text. We can use

[1] https://aws.amazon.com/batch/
[2] https://docs.pola.rs/
[3] https://github.com/shon-otmazgin/fastcoref
[4] https://huggingface.co/biu-nlp/lingmess-coref
[5] https://spacy.io/

fastcoref to obtain the coreference clusters for each text input. However, this does not replace each coreference mention with the main mention in the associated cluster, as this functionality is not natively supported by fastcoref. To replace referring entities in our text with their corresponding referring expressions, we utilize SpaCy. While fastcoref handles coreference clustering, it requires SpaCy in combination to effectively resolve the references in the text. The fastcoref package integrates seamlessly with SpaCy by offering a custom component that can be directly plugged into a SpaCy pipeline.

The following code snippet shows how we integrate the fastcoref component into a SpaCy pipeline to resolve coreferences in a list of text reviews using the LingMess model.

```
from fastcoref import spacy_component
import spacy

nlp = spacy.load("en_core_web_lg", enable=[])

nlp.add_pipe(
    "fastcoref",
    config={
        "model_architecture": "LingMessCoref",
        "model_path": "biu-nlp/lingmess-coref",
        "device": "cuda",
    },
)

resolved_reviews = []
for doc in nlp.pipe(
    reviews,
    component_cfg={"fastcoref": {"resolve_text": True}}
):
    resolved_reviews.append(doc._.resolved_text)
```

4.2.2 Proposition Extraction

The next step involves extracting propositions from each product review. For proposition extraction, we employ the Propositionizer[6], which is a fine-tuned Flan-T5-large model as introduced in [20]. To enhance the speed of this process, we use the Hugging Face Text Generation Inference (TGI)[7].

4.2.3 Proposition Classification

For proposition classification, we use NuExtract[8], which is a version of phi-3-mini fine-tuned on a private high-quality synthetic dataset for information extraction.

For LLM inference, we employ vLLM[9]. vLLM was introduced by Kwon et al. [23] and it is a library designed for LLM inference and serving. vLLM is particularly advantageous due to its state-of-the-art serving throughput, enabling fast processing.

```
from vllm import LLM, SamplingParams

base_model_name = "microsoft/Phi-3-mini-4k-instruct"
merged_peft_model_name = "numind/NuExtract"

llm = LLM(model=merged_peft_model_name, tokenizer=base_model_name,
              enforce_eager=True)
sampling_params = SamplingParams(temperature=0, top_p=0.95, max_tokens=40980)
```

[6] https://huggingface.co/chentong00/propositionizer-wiki-flan-t5-large
[7] https://huggingface.co/docs/text-generation-inference/index
[8] https://huggingface.co/numind/NuExtract
[9] https://github.com/vllm-project/vllm

To use the NuExtract model, we provide an input text—specifically, the proposition—along with a template that describes the information to be extracted. The preparation of the input message for NuExtract is demonstrated in the code snippet below:

```
def get_single_chat_message(text):
    schema = """{
            "Person": [
                {
                    "Residence": [],
                    "Has_Job": [],
                    "Age": "",
                    "Has_Pet": [],
                    "Has_Hobby": [],
                    "Has_Skin_Condition": []
                }
            ]
    }"""

    schema = json.dumps(json.loads(schema), indent=4)
    input_llm = "<|input|>\n### Template:\n" + schema + "\n"
    input_llm += "### Text:\n" + text + "\n<|output|>\n"
    return input_llm
```

4.2.4 Triplet Extraction

The next step is performing triplet extraction, where we identify subject-predicate-object relations from the propositions. This step is crucial for transforming unstructured text into structured data that can be used for further analysis. To achieve this, we leverage a LLM to extract triplets efficiently. Once again, we use vLLM to accelerate the inference process, taking advantage of its optimized serving capabilities that ensure high throughput without sacrificing accuracy.

The specific prompt used for triplet extraction can be found in Appendix A. This prompt was adapted from Langchain[10], a framework designed for building applications powered by LLMs. While the framework itself is not used in our work, it is important to note that both the prompt and the inspiration for this step were drawn from Langchain.

Below is the code snippet demonstrating the configuration of the LLM and the parameters used for inference. However, please note that we are not limited to a specific LLM and it can be executed with other LLMs as well.

```
sampling_params = SamplingParams(temperature=0, top_p=0.99, max_tokens=4096)

model_id = "meta-llama/Meta-Llama-3.1-8B-Instruct"
llm = LLM(
    model=model_id,
    trust_remote_code=True,
    enforce_eager=True,
    max_model_len=4096,
    gpu_memory_utilization=0.95
)
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

The LLM is asked to return triplets in JSON format, but there is no guarantee that the output will be valid. LLMs can occasionally produce malformed JSON, such as missing parentheses or extra words. To address this, we use the json_repair[11] package, which reliably corrects invalid JSON without altering the content.

[10] https://www.langchain.com/
[11] https://github.com/mangiucugna/json_repair/

```
import json_repair

nodes_set = set()
relationships = []

parsed_json = json_repair.loads(raw_schema)

for rel in parsed_json:
    # Nodes need to be deduplicated using a set
    nodes_set.add((rel["head"], rel["head_type"]))
    nodes_set.add((rel["tail"], rel["tail_type"]))

    source_node = Node(id=rel["head"], type=rel["head_type"])
    target_node = Node(id=rel["tail"], type=rel["tail_type"])
    relationships.append(Relationship(source=source_node, target=target_node,
    type=rel["relation"]))
    # Create nodes list
    nodes = [Node(id=el[0], type=el[1]) for el in list(nodes_set)]
    all_nodes.append(GraphDocument(nodes=nodes, relationships=relationships))
```

4.2.5 Entity Decomposition

Next, entity decomposition is performed, where composite entities are split into their separate components. For example, the entity "long curly" is decomposed into two distinct entities: "long" and "curly". This step ensures that each entity represents a single, well-defined concept.

Since this task is relatively straightforward, a smaller LLM suffices, providing faster inference. We use gemma-2-9b-it for this purpose, as it offers the necessary performance with reduced computational overhead. To further optimize speed, we again utilize vLLM for efficient inference.

```
sampling_params = SamplingParams(temperature=0, top_p=0.9, max_tokens=4098)

llm = LLM(
    model="google/gemma-2-9b-it",
    trust_remote_code=True,
    enforce_eager=True,
    max_model_len=2400,
    gpu_memory_utilization=0.95,
    dtype=torch.bfloat16
)
tokenizer = AutoTokenizer.from_pretrained("google/gemma-2-9b-it")
```

The prompt used for entity decomposition may vary depending on the type of entities being split. The following code snippet shows an example of a prompt used for decomposing entities of the type "Skin Condition":

```
def get_single_chat_message(text):
    user_content = f"""
                    Extract the skin conditions mentioned in the following
                    sentence and return them as a comma-separated list.
                    It is very important that you only provide the
                    comma-seperated list without any additional comments or
                    remarks. If no skin conditions are mentioned, return an
                    empty string.

                    **Input:** {text}
                    """

    return [{"role": "user", "content": user_content}]
```

4.2.6 Entity Deduplication

The goal of Entity Deduplication is to merge redundant entities that represent the same concept into a single entity. For example, entities such as "dry" and "dryness" could be merged into a single entity, "Dry". The approach used for Entity Deduplication was inspired by the work of Cao et al. [24]. The deduplication process consists of several substeps, generally following two key phases:

- **Similarity Calculation:** Cosine similarity scores are computed for all embeddings of the entity pairs, and the pairs are ranked based on their similarity. Entities with a similarity score above a predefined threshold are flagged for merging.
- **LLM-Based Merging:** A LLM is used to merge high-similarity entity pairs into unified keywords.

1) Generate embeddings for entities.
The first step involves generating embeddings for all entity IDs. For instance, when the task pertains to skin conditions, the following code snippet illustrates the process. It is important to note the inclusion of a prefix "Skin condition is {text}" to provide context for the embedding:

```
st_model = SentenceTransformer("paraphrase-MiniLM-L6-v2")
text = f"Skin condition is {text}"
embeddings = st_model.encode(text)
```

2) Calculate cosine similarity between entity pairs. After calculating the embeddings for each entity ID in the previous step, the next task is to compute the cosine similarity between all pairs of entities, as demonstrated in the following code snippet:

```
embeddings = np.array(list(keyword_embedding.values()))
norms = np.linalg.norm(embeddings, axis=1, keepdims=True)
normalized_embeddings = embeddings / norms
cosine_similarity_matrix = np.dot(normalized_embeddings, normalized_embeddings.T)

pairs_with_similarity = []

keywords = list(keyword_embedding.keys())

for i in range(len(keywords)):
    for j in range(i + 1, len(keywords)):
        similarity = cosine_similarity_matrix[i, j]
        pairs_with_similarity.append((keywords[i], keywords[j], similarity))

pairs_with_similarity.sort(key=lambda x: x[2], reverse=True)
```

3) Identifies pairs of keywords to merge based on a similarity threshold. The previous step yields a similarity score for each pair of keywords. In this phase, we retrieve all pairs with a similarity score exceeding a specified threshold, such as 0.9. The following code snippet illustrates this process:

```
similarity_threshold=0.9
pairs_to_merge = []
merged = []

for keyword_pair in pairs:
    if (
        (keyword_pair[2] > similarity_threshold)
        and (keyword_pair[0] not in merged)
        and (keyword_pair[1] not in merged)
    ):
        pairs_to_merge.append(keyword_pair)
        merged.append(keyword_pair[0])
        merged.append(keyword_pair[1])
```

We will select only the top 10 pairs, ranked by the highest similarity scores, to be passed to the LLM for merging in the next step. It is important to note that this process is iterative. Thus, pairs not merged in the current iteration will be merged in next iterations.

4) LLM prompting to merge pairs

The final step involves using an LLM to merge the keyword pairs identified in the previous phase. For this task, we employ vLLM once again to ensure faster inference.

```
sampling_params = SamplingParams(temperature=0, top_p=0.99, max_tokens=4096)

model_id = "meta-llama/Meta-Llama-3.1-8B-Instruct"
llm = LLM(
    model=model_id,
    trust_remote_code=True,
    enforce_eager=True,
    max_model_len=4096,
    gpu_memory_utilization=0.95
)
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

For instance, if the task is specifically related to entities of the type "Skin Condition" the LLM prompt could be as follows:

```
def get_single_chat_message(text):

    user_content = f"""
                    Given two phrases describing the same skin condition,
                    provide a concise, merged keyword or keyphrase that
                    accurately represents both.
                    It is very important that you only provide the final skin
                    condition without any additional comments or remarks.

                    Example:
                    * Input: ["slightly dry skin", "Dry skin"]
                    * Output: Dry

                    Your Turn:
                    * Input: {text}
                    """
    return [{"role": "user", "content": user_content}]
```

The entire Entity Deduplication process, as described above, is iterative. This means the process is repeated until no pairs remain with a similarity score above the threshold (set to 0.9 in our experiments).

Chapter 5

Experimental Design

This chapter will outline the experimental design that will be used to evaluate the proposed end-to-end pipeline. Two experiments, both with a different dataset, will be presented. The main purpose is to evaluate the performance of our approach at extracting personal attributes from text.

5.1 Experiment 1: Baseline Study

The goal of this first experiment is to establish a baseline for performance evaluation. This involves determining whether our approach can extract personal attributes with at least the same effectiveness as existing methods. The main advantage of our approach is that it is not limited by predefined attribute categories, thus offering a more comprehensive and adaptable solution. Despite this advantage, it is essential to verify that our approach meets the baseline performance level established by methods using predefined categories.

5.1.1 Extraction from conversational texts

The approach of this study was developed with the main goal of extracting personal attributes from product reviews. Studies in this field are very limited. The closest to our research are studies that present methods to extract personal attributes from conversational texts, such as chat messages sent to a chatbot. Given the limited number of studies focusing on extracting personal attributes from product reviews, this first experiment will instead focus on conversational texts. This switch is justified because both product reviews and chat messages are forms of unstructured text where users may share personal attributes. Although the contexts differ, the underlying goal of extracting personal attributes remains the same. By using conversational texts, which have more available research and methods, this experiment can provide valuable insights and evaluate the approach.

5.1.2 Dataset

The dataset PersonaExt from the study PAED: Zero-Shot Persona Attribute Extraction in Dialogues will be a key asset in this first experiment. This dataset contains text messages and, for each text message, a triplet consisting of a `head`, `relation_type`, and `tail`.

PersonaExt includes 105 relation types. For this experiment, we reduced the number of relation types. We removed several relation types that were either irrelevant or too broad in scope, such as `do_not_do`, `have_no`, and `pre_employed_by_company`. We decided to keep only 4 relation types, which we think are most relevant to our study, in order to have a more focused and manageable experimental setup. Specifically, we will be looking at the following 4 relation types: `live_in_citystatecountry`, `has_profession`, `has_age`, `have_pet`. Reducing the number of relation types allows us to focus on the most relevant personal attributes for our study. An overview of the number of pairs for

each relation type can be found in Table 5.1. For the pairs retrieved from the relation types `live_in_citystatecountry` and `has_age`, all available pairs in the PersonaExt dataset are used. However, for the pairs from the relation types `has_profession` and `have_pet`, only the first 500 pairs are considered for this experiment due to time constraints.

Personal Attribute	**Relation Type in PersonaExt**	**#Utterance-triplet pairs**
Residence	`live_in_citystatecountry`	595
Age	`has_age`	484
Profession	`has_profession`	500
Pet	`have_pet`	500

Table 5.1: Personal attributes with the relation types and the total number of utterance-triplet pairs retrieved from the PersonaExt dataset.

5.1.3 Approach

We will collect all utterances for each relation type listed in Table 5.1 and process them through our pipeline, the details of which are described in Chapter 3. We define a predefined list of personal attributes for extraction to be used in the Proposition Classification step: `Residence`, `Job`, `Age`, `Has_Pet`.

The attributes in this list are in line with the personal attributes given in Table 5.1. This list helps classify the propositions, allowing us to filter and retain only those that are relevant. Consequently, this approach becomes more scalable by reducing the amount of data processed in the triplet extraction step.

For each utterance, we have the relation type and thus know which personal attribute we want to extract. For example, if we have the relation type from our PersonaExt dataset which is `has_age`, then we already know that we want to extract `Age` from our utterance. One could argue why not simply provide the schema `{"Person":[{"Has_Age":[]}]}` to all utterances with the relation type `has_age` so we can easily extract only `Age` using our approach and compare it to the extracted `Age` in the PersonaExt dataset. However, it was deliberately chosen not to do this for multiple reasons. First and foremost, when we run our approach on product reviews, we also do not know beforehand which personal attributes this product review will contain. In this case, we know it because it is provided in the PersonaExt dataset, but we want to resemble a real case.

Triplet Filtering

Each relation type has its own set of utterances in which the associated personal attribute is mentioned. For example, most utterances associated with the relation `live_in_citystatecountry` will mention a Residence. However, these utterances may also contain additional personal attributes. For instance, consider an utterance from the set `live_in_citystatecountry`, which has the text: "*i am 29 and live in the suburb of boston*," and in the PersonaExt dataset, `<i, live_in_citystatecountry, boston>` is the extracted triplet. If we give this utterance to our pipeline, it extracts two personal attributes: namely, that the person is 29 years old, and that the person lives in Boston. We will be filtering and only looking at the extracted personal attribute associated with the original relation type of this utterance. In this example, since this utterance has the relation type `live_in_citystatecountry` in the PersonaExt dataset, we will only consider Residence (boston) as the extracted personal attribute, and the other extracted personal attributes (has age: 29) will be filtered out. Even though the other extracted personal attributes for this utterance might also be interesting to evaluate, in this experiment, they are filtered out. The main reason for filtering is that we only have the extracted personal attribute in the PersonaExt dataset depending on the relation type.

Thus, since we want to have an extracted value for a personal attribute to compare it to, we will only be looking at one personal attribute for each relation type. We give an utterance to our pipeline, which then returns the extracted triplets. However, we want to filter these triplets based on the relationship type from the PersonaExt dataset. To achieve this, specific filtering rules will be defined and applied. For instance, when examining the utterances related to the relationship `has_age` in PersonaExt, we may decide to retain only those extracted triplets where the source node is of type `Person` and the target node is of type `Age`. Subsequently, we will evaluate whether the extracted value, in this case, the age, is correct. The precise rules for this triplet filtering will be detailed in the Section 6.1, as these rules depend on the extracted triplets.

5.1.4 Evaluation

We will compare the personal attributes extracted by our approach with those from PersonaExt through a two-step evaluation process. Firstly, an automatic evaluation will be performed. In the second step, any utterances that were not matched automatically will be reviewed manually.

Automatic evaluation

The first step is to automatically search for matches between the two sets of extracted personal attributes. Specifically, we will use two columns of data: one containing the personal attributes from the PersonaExt dataset and the other containing the personal attributes extracted by our approach. For each utterance, we will compare the personal attribute from the PersonaExt column with the extracted personal attribute from our approach to determine if there is a match.

A match is defined as the extracted personal attribute from our approach containing *at least* the personal attribute available in PersonaExt. If both values are identical, it is considered correct and labeled as `Correct Extraction`. Similarly, similar extractions that are not fully identical but still relevant may also be labeled as `Correct Extraction`. For example, if PersonaExt lists the personal attribute as "*teacher*" and our approach extracts "*teacher at school*", this will still be regarded as a match and labeled as `Correct Extraction`. Even though our extracted attribute includes additional context, it will still be deemed correct and labeled as such. This method ensures that attributes are evaluated accurately, even when the extracted values include extra details or context.

By using a lenient condition, more personal attributes will be automatically labeled as a match. The assumption is made that, in most cases, additional details or context in the extracted personal attributes do not impact the accuracy of the extracted personal attribute. In other words, even if the extracted attribute includes extra information or context, it is still considered correct as long as it includes the intended personal attribute. This is based on the idea that such additional details usually do not change the fundamental correctness of the attribute.

One might question why we are not using a stricter condition that requires an exact match when comparing the extracted personal attributes. However, by adopting a more relaxed condition, we can increase the number of attributes labeled automatically, thus reducing the need for manual checks. Since many personal attributes are extracted and manual checking is time-consuming, we aim to minimize the number of attributes that require manual review. It might be that the extracted values include extra details or context; however, we assume that this does not make the extraction incorrect.

Manual Check

After the initial automatic evaluation, some utterances will be labeled as `Correct Extraction`, indicating that the extracted personal attributes align with the expected values. However, there will also be utterances that do not receive this label, meaning no

match was found automatically. The utterances that remain after the automatic check will be manually reviewed. One possible outcome of the manual check might be that our approach extracted a different value for the personal attribute, but upon inspection, we may find that the extracted attribute is still correct. In this case, we apply the label "Correct Extraction". Alternatively, the manual check might show that our approach has extracted an incorrect value for the personal attribute or did not extract any value for it, then we give the label "Failed Extraction". Lastly, it is possible that, upon manual inspection, the utterance from the PersonaExt dataset is disregarded because it does not explicitly mention a personal attribute, it will be disregarded and labeled as "Remove".

Evaluation Outcomes

Following the automatic and manual checks, each extracted personal attribute is assigned one of three labels: "Correct Extraction", "Failed Extraction", "Remove". An overview of these labels, along with their definitions and the extraction methods used, can be found in Table 5.2. The label "Correct Extraction" can be assigned through both the automatic and manual checks. In the automatic check, an extracted personal attribute is labeled as "Correct Extraction" if it contains the corresponding value from the PersonaExt dataset, either as an exact match or with additional contextual information. For all attributes not labeled as "Correct Extraction" in the automatic check, a manual inspection is performed. During this manual review, the labels "Correct Extraction", "Failed Extraction", and "Remove" are used to categorize the extracted personal attributes.

Label	**Definition**	**Method**
Correct Extraction	The extracted personal attribute **contains** the PersonaExt value, either as an exact match or with additional information	Automatic check
Correct Extraction	The extracted personal attribute is different from the PersonaExt value, but it is still deemed correct after manual inspection	Manual check
Failed Extraction	The extracted personal attribute is either incorrect or no value was extracted	Manual check
Remove	Upon manual inspection, the utterance from the PersonaExt dataset is disregarded because it does not explicitly mention a personal attribute.	Manual check

Table 5.2: Different labels after evaluation, including the definition of each label and the method used to determine the label

We can now assess the total number of utterances for which our model correctly extracts personal attributes, as well as the number it fails to extract. These amounts of correct and failed extractions can be determined for each personal attribute. After running the evaluation on the dataset, we have numerical data that we can use to assess the performance of our approach in extracting personal attributes from the given utterances in the PersonaExt dataset. Additionally, it allows us to compare if there is any difference in extraction depending on the personal attributes.

5.2 Experiment 2: Case Study

The first experiment serves as a baseline study aimed at determining whether our approach can extract personal attributes with at least the same effectiveness as an existing method. However, the key advantage of our approach lies in its adaptability; it is not limited by

predefined attribute categories, allowing us to easily add additional personal attributes for extraction.

In this case study, we will retrieve a set of product reviews for day creams, with the goal of extracting personal attributes from these reviews. While we remain interested in some of the same personal attributes as before (such as Age, Residence, and Profession), this study also emphasizes the importance of the Skin Condition attribute, relevant to the context of day creams. The input dataset comprises 500 product reviews, and every step of the pipeline will be executed. Finally, a manual analysis of the results will be performed, and the findings will be presented and evaluated.

The main focus of this experiment is to evaluate the accuracy of extracting skin conditions from product reviews. Additionally, we will examine the results of the Entity Merging step, where we expect that similar skin conditions, though phrased differently, will be consolidated into a single node (e.g., we expect that "dry" and "dryness" will be merged into one node).

Chapter 6

Experimental Results

In the previous chapter, the design of the two experiments conducted in this thesis was explained and discussed. This chapter presents and analyzes the results of these experiments. The first section covers the findings from the baseline study, while the second section focuses on the case study results, which was performed on a set of customer reviews.

6.1 Experiment 1: Baseline Study

As discussed in Section 5.1, this experiment utilizes the PersonaExt dataset, which consists of utterances. These utterances are processed by our pipeline to extract triplets. However, to focus on relevant information, we apply the following filtering rules to the extracted triplets:

- **Residence**: We keep only triplets where the relation includes “LIVE” or “RESIDE”, or if the relation is exactly “FROM”.
- **Age**: We keep only LLM triplets where the Target Type is equal to “Age”
- **Profession**: We keep only LLM triplets where the Target Type contains “Job”, "Profession", "Occupation" or "Work"
- **Pet**: We keep only LLM triplets where the Target Type contains “Pet” or “Animal”

The extracted values are then evaluated. As described in Section 5.1, the evaluation process begins with an automatic check, where the personal attribute values extracted by our approach are compared to those in the PersonaExt dataset. If the values match, the extraction is considered correct. For utterances that do not pass the automatic check, manual inspection is conducted. In some cases, the utterance may be disregarded after manual inspection if it is determined that there was no clear mention of the personal attribute. The remaining utterances are either labeled as a Correct Extraction or a Failed Extraction. The evaluation is performed across four categories: Age, Residence, Profession, and Pet Possession. A visual representation of the results is provided in Figure 6.1. On the left, the input utterances are shown, and moving toward the right, the figure illustrates how the utterances are labeled; first by the automatic check and then by the results of the manual inspection. The following paragraphs will describe the outcomes in more detail for each category. A sheet containing all the data and results is available upon request.

Results by Category

For the **Age** category, a total of 484 utterances were used as input. We performed the automatic check, and 439 utterances passed the automatic check, meaning that the personal attribute extracted by our approach aligned with the value in the PersonaExt dataset. There were 45 utterances that did not pass the automatic check and required manual

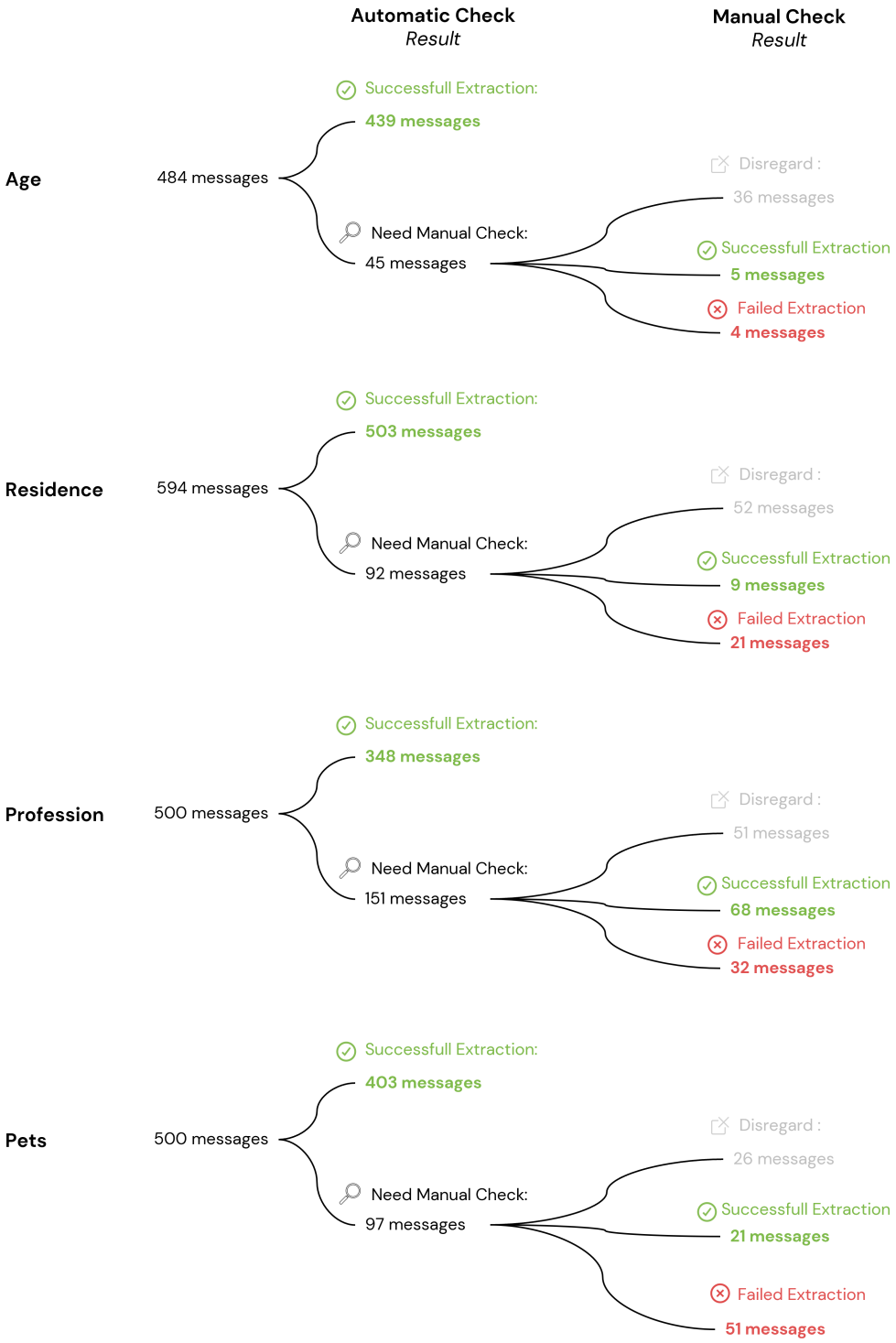

Figure 6.1: Overview of the results of the first experiment, presented in four categories, highlighting the number of successful extractions and failed extractions.

inspection. Out of these 45, 36 were disregarded during manual inspection because the personal attribute was not clearly mentioned. In 5 utterances, although the extracted value differed from the value in PersonaExt, it was still deemed correct. The extraction failed in 4 utterances, either due to no value being extracted for age or because the wrong value was extracted. Therefore, after disregarding 36 utterances, we have a total of 448 utterances, with the age correctly extracted in 444 of them.

For the **Residence** category, there were 594 utterances. A total of 503 utterances passed the automatic check, meaning the residence was correctly extracted using our approach. The remaining 92 utterances required manual inspection. After manual review, 52 utterances were disregarded because the mention of residence was unclear. For 9 utterances, the residence extraction was considered correct, even though it differed from the PersonaExt value. The extraction failed in 21 utterances. After disregarding the 52 unclear utterances, we had 542 utterances in total, with the residence correctly extracted in 512 of them.

For the **Profession** category, we analyzed the first 500 utterances from the PersonaExt dataset. Out of these, 348 utterances passed the automatic check. Manual inspection was performed for the remaining 151 utterances. Of these, 51 were disregarded due to unclear mentions of profession. The profession extraction was deemed correct for 68 utterances, even though it differed from the PersonaExt value. The extraction failed in 32 utterances. After disregarding the 51 unclear utterances, we had a total of 449 utterances, with the profession correctly extracted in 416 of them.

Finally, for the **Pets** category, we also selected the first 500 utterances from PersonaExt. A total of 403 utterances passed the automatic check and were considered correct extractions of pet ownership. The remaining 97 utterances underwent manual inspection. Of these, 26 were disregarded because there was no clear mention of pet ownership. In 21 utterances, the pet extraction was manually labeled as correct. However, the extraction failed in 51 utterances, which is a higher failure rate compared to the other categories. In many of these failed cases, our approach extracted the name of the pet but did not correctly extract the type of pet, leading to the failed extraction. This could be improved in future implementations by modifying the approach to extract both the name and the type of pet. After disregarding 26 utterances, we had a total of 474 utterances, with the pet correctly extracted in 424 of them.

Conclusion

In conclusion, this experiment demonstrates that for the majority of utterances across all four categories, our approach successfully extracts the correct value for the personal attribute. An overview of the results for each category is presented in Table **??**, where we observe a high number of correct extractions.

	Correct Extraction	Failed Extraction	Correct Extraction Rate
Age Extraction	444	4	**99.1%**
Residence Extraction	512	21	**96%**
Profession Extraction	416	32	**92.9%**
Pet Extraction	424	51	**89.2%**
			Average: **94.3%**

The primary goal of this first experiment was to establish a performance baseline, determining whether our method can extract personal attributes with effectiveness comparable to existing approaches. While there were some instances where the extraction was inaccurate, the majority of cases were successful, indicating that our approach is generally effective. Although one might argue that PersonaExt achieves higher accuracy in some cases—since our approach occasionally fails to extract the correct value—it is important to highlight the flexibility of our method. The PersonaExt extraction approach is fine-tuned

for a limited set of personal attributes, whereas our approach can be extended to extract any personal attribute of interest. For example, the next section will demonstrate the extraction of Skin Condition as a personal attribute. This is something that the approach used for generating PersonaExt cannot achieve. This flexibility is a significant advantage of our approach and underscores its potential.

6.2 Experiment 2: Case Study

The first experiment established a baseline, demonstrating that our approach could extract personal attributes with effectiveness comparable to existing methods. However, the true strength of our approach lies in its flexibility. Our approach can be extended to include additional personal attributes as needed. In this case study, we focus on extracting personal attributes from a set of product reviews for day creams. While some attributes from the previous experiment, such as Age, Residence, and Profession, remain relevant, this study introduces a new emphasis on the Skin Condition attribute, which is highly relevant in the context of skincare products. The dataset consists of 500 product reviews, and our full extraction pipeline will be applied. The primary objective of this experiment is to evaluate the accuracy of skin condition extraction from these reviews. Additionally, we will assess the performance of the Entity Deduplication step, where we expect similar skin conditions expressed in different forms (e.g., “dry” and “dryness”) to be merged into unified entities. Finally, a manual data analysis of the pipeline results will be conducted to evaluate the findings.

6.2.1 Results of Skin Condition Extraction

The initial dataset consisted of 500 customer reviews. One review was empty and subsequently disregarded, leaving 499 reviews. All reviews were manually inspected and annotated, the results are available upon request. Out of these, 149 reviews did not provide a clear mention of the person’s skin condition, reducing the relevant reviews to 350.

- **Correct Extraction of Skin Condition:** In 328 out of 350 reviews, our approach successfully extracted the correct skin condition.
- **Failed Extraction of Skin Condition:** In 22 out of 350 reviews, the extraction of the skin condition failed, either due to no extraction taking place or because an incorrect value was extracted.

6.2.2 Results Entity Deduplication

To handle variations in the extracted skin conditions, a merging process was applied to consolidate similar terms. The results of this merging can be found in the list below. For example, there were numerous variations of "Dry," such as “dry”, “dry skin”, “dryness”, “very dry skin”, “extremely dry skin”, “super dry skin”, “extra dry skin”, and “severely dry skin”. Initially, these were treated as separate entities, but after the entity deduplication step, they were merged into a single entity labeled “Dry”. Similar merges were performed for other skin conditions.

The following list presents the merged skin condition names along with their corresponding variations that have been consolidated into the entity:

- **Dry**: dry, dry skin, dryness, very dry skin, extremely dry skin, super dry skin, extra dry skin, severely dry skin
- **Sensitive**: sensitive skin, sensitive, super sensitive skin, supersensitive skin
- **Acne**: acne, hormonal acne, acne prone skin, acne-prone, acne-prone skin, adult hormonal acne, acne scars, inflammatory acne

- **Eczema**: eczema, eczema-prone skin
- **Oily**: oily, oily skin, oily face, oily skin type
- **Redness**: redness, red skin, red
- **Breakout**: breakouts, break outs, breakout, break out, abnormal breakout, skin breakout
- **Irritation**: irritated skin, irritation, skin irritation, irritating skin, irritating reactions, irritated, skin irritations
- **Combination**: combination, combination skin
- **Flaky**: flaky skin, flaky, flakey skin, flakey spots, flaky patches
- **Burn**: burn, burns, burned, burned skin, burning
- **Rosacea**: rosacea
- **Itchy**: itchy skin, itchy

6.2.3 Frequency Analysis of Extracted Skin Conditions

Once the skin conditions were clustered, their frequencies were analyzed. The frequency distribution of the clustered skin conditions is shown visually in Figure 6.2, where it is clear that "Dry" was the most frequently extracted condition, namely 259 times. It is important to note that a single product review may contain multiple different skin conditions, all of which can be extracted. To visualize the frequencies, a word cloud was generated, as shown in Figure 6.4, where the size of each word reflects its extraction frequency, with larger words indicating higher occurrences.

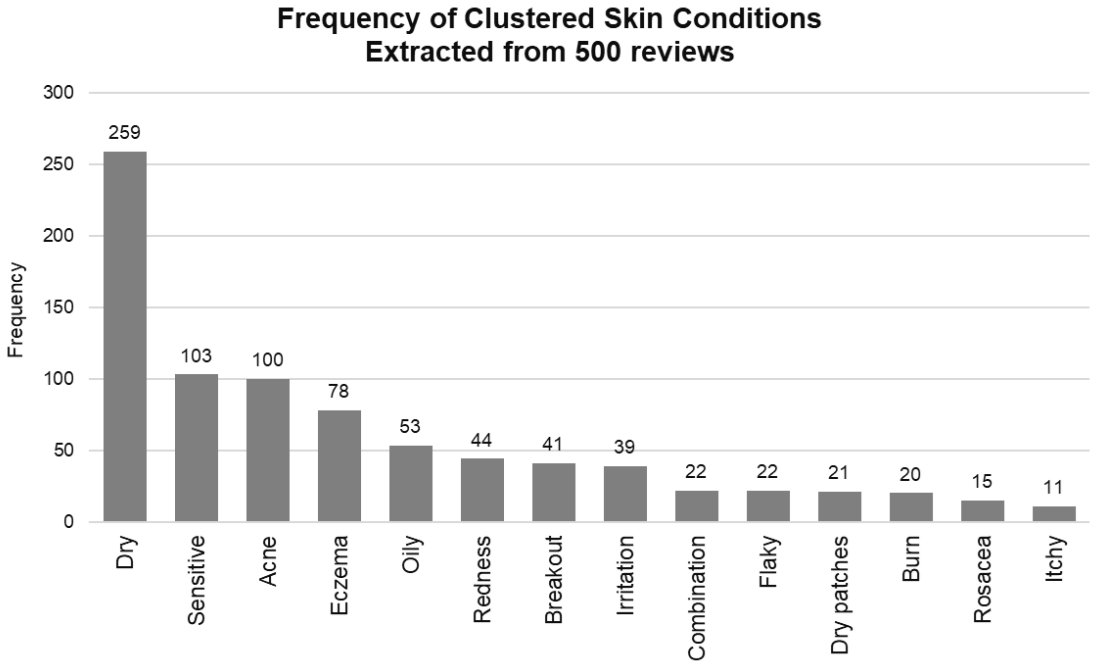

Figure 6.2: Plot showing the frequency of clustered skin conditions extracted from product reviews, displaying only those skin conditions with a frequency of 11 or higher.

Figure 6.3: Word cloud illustrating skin conditions, where larger text indicates a higher frequency of extraction for those conditions.

Review: I have **dry, flaky** skin. This moisturizer has been my go-to for years. It even got me through my adjustment to retinol with minimal flaking.

Source Value	Source Type	Relation	Target Value	Target Type
Reviewer	Person	HAS_CONDITION	Flaky	Skin Condition
Reviewer	Person	HAS_CONDITION	Dry	Skin Condition

Review: I was not impressed with this. I have very **dry** skin and got it to use on my face, and it was just OKAY, but didn't seem to do much for my **dry patches**.

Source Value	Source Type	Relation	Target Value	Target Type
Reviewer	Person	HAS_CONDITION	Dry	Skin Condition
Reviewer	Person	HAS_CONDITION	Dry patches	Skin Condition

Review: My holy grail and the only thing to help my **eczema**!!

Source Value	Source Type	Relation	Target Value	Target Type
Reviewer	Person	HAS_CONDITION	Eczema	Skin Condition

Figure 6.4: Three examples of input reviews along with some corresponding extracted triplets.

Chapter 7

Conclusion & Future Work

This chapter presents the key findings and contributions of this thesis, highlighting the main outcomes of the research. It also discusses the limitations encountered throughout the study and provides recommendations for future work to address these challenges and explore new directions.

7.1 Conclusion

Data analytics is essential for uncovering customer needs, preferences, and profiles, which drive product development and strategic decision-making. This research focused on extracting personal attributes from product reviews. Personal attributes are distinct characteristics that represent an individual's traits, such as physical appearance, age, and profession. While previous work has explored extracting personal attributes from text using established models, these approaches often rely on predefined lists of attributes. This limitation necessitates model retraining whenever there is a need to extract additional attributes. Therefore, the primary goal of this research was to develop a more flexible approach that allows for the extraction of any relevant personal attribute based on the context and data source.

To achieve this, we aimed to leverage LLMs to effectively extract personal attributes from product reviews. This research presents an end-to-end pipeline for extracting personal attributes, providing valuable insights into customer characteristics. The pipeline consists of the following key steps:

- **Prefixing**: The prefix "Jane says:" is added before each review, with "Jane" serving as a universal pseudonym for all reviewers. This lexical addition creates a clear reference to the reviewer in each review, which is helpful for coreference resolution.
- **Coreference Resolution**: This step replaces pronouns and other referring expressions with the appropriate entity, ensuring that all references are unified.
- **Text Segmentation**: Long, complex sentences in reviews are split into shorter, more manageable segments, called propositions, to facilitate effective triplet extraction.
- **Proposition Filtering**: We filter the segmented propositions to retain only those that contain relevant information, reducing the input for the triplet extraction process.
- **Triplet Extraction**: Structured triplets are extracted from the filtered propositions, capturing key relationships and entities.
- **Entity Decomposition**: Composite entities are broken down into distinct components, ensuring that each entity represents a single concept.

- **Entity Deduplication**: Redundant entities representing the same concept are merged in a single node.
- **Graph Construction**: Finally, the extracted triplets are organized into a graph, providing a structured representation of entities and their relationships for easier analysis.

Through this end-to-end pipeline, we have expanded existing methodologies and demonstrated a flexible approach to extracting personal attributes. This flexibility allows us to easily select and extract specific personal attributes of interest based on the context and input data. In the case study conducted in this thesis, the personal attribute of interest was skin condition, as the dataset comprised product reviews for a day cream in which reviewers mentioned their skin conditions, such as acne, eczema, and dry skin. The results show promising outcomes; with only a few exceptions, our pipeline was able to extract skin condition effectively without requiring any fine-tuning. This is encouraging for future applications, where the list of personal attributes can be adapted, such as interest in hair type for reviews of hair dryers or interest in pet ownership in reviews for vacuum cleaners.

7.2 Limitations

This section outlines the limitations that must be considered with our approach. Firstly, the segmentation of reviews into numerous propositions can lead to a loss of context, making it difficult to capture the full meaning and nuance of the original text. Although this step is intended to enhance the extraction process, it often results in propositions that may contain incomplete or fragmented information, which can hinder personal attributes extraction.

Furthermore, the high volume of propositions complicates the extraction process. The pipeline includes a filtering step where we retain only the propositions relevant for triplet extraction. Despite this, we are still left with a large number of propositions. Additionally, the filtering process relies heavily on the classification of these propositions. If the classification is incorrect, important propositions may be filtered out, resulting in missed information and potentially preventing the extraction of personal attributes.

Moreover, while we leverage open-source LLMs, which effectively eliminate licensing and inference costs, we must still account for the infrastructure expenses associated with processing large datasets. These costs can include expenses for computing resources, storage, and cloud services.

In addition, the pipeline components have not been fully optimized, resulting in slower processing times than desired. This inefficiency pertains to both the overall structure of the pipeline and its implementation. This highlights the need for improvements in both the structural design and the underlying code to enhance performance and achieve faster results.

Lastly, despite our efforts to minimize information loss during triplet extraction and capture all relevant personal attributes, our results indicate instances where our approach fails to extract specific attributes from the text. This limitation shows an opportunity for future research to refine and enhance the extraction pipeline, aiming for improved accuracy and reliability in extracting personal attributes from text.

7.3 Future Work

To address the efficiency limitations outlined earlier, there is considerable potential for optimization within our pipeline. Currently, the pipeline consists of numerous separate steps, and future research could focus on removing or combining certain steps to assess their necessity and effectiveness.

While general-purpose LLMs yield promising results, their high resource demands pose challenges for efficiently handling large datasets. Future work could explore the application of task-specific LLMs tailored to individual steps within data processing pipelines. By using models that are fine-tuned for specific tasks, it may be possible to achieve high performance while reducing computational costs. Therefore, a valuable direction for future work is to investigate the use of task-specific LLMs. For instance, we employed NuExtract for the Proposition Classification step, as it is fine-tuned for this particular task. Future research should similarly explore the potential for implementing task-specific LLMs in other steps of the pipeline.

Furthermore, as part of our pipeline, we perform entity deduplication through pairwise merging. This task is inherently complex, and while pairwise merging has shown promising results, more experimentation is needed to refine and optimize this process. Given the significance of entity deduplication within the pipeline, enhancing the pairwise merging step is crucial for improving the final graph that is constructed.

The primary focus has been on the steps from processing unstructured text to extracting triplets and using these triplets to build a graph. However, more attention should be given to the graph construction process itself. Future research could also explore graph data science, investigating how the graph can be queried and how the data within it can be analyzed to extract meaningful insights.

Moreover, additional experimentation is necessary to evaluate the effectiveness of each step in our pipeline. By conducting more extensive experiments, we can better assess the quality of individual steps and make improvements where necessary. This would also provide stronger justification for the decisions we have made in designing the pipeline. While smaller experiments have guided our current choices, larger-scale experimentation in future work would offer more comprehensive insights and help validate the pipeline's structure.

Finally, engaging with businesses to understand their specific data extraction needs would be advantageous. This collaboration could help tailor the pipeline to align with business objectives, facilitating applications such as the development of data-driven buyer personas or performing market segmentation

Bibliography

[1] Bai Yang, Ying Liu, and Wei Chen. A twin data-driven approach for user-experience based design innovation. 68:102595.

[2] Yanzhang Tong, Yan Liang, Ying Liu, Irena Spasić, and Yulia Hicks. Understanding context of use from online customer reviews using BERT. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 1820–1825. ISSN: 2161-8089.

[3] Yi Han and Mohsen Moghaddam. Eliciting attribute-level user needs from online reviews with deep language models and information extraction. 143(61403).

[4] Feng-Lin Li, Hehong Chen, Guohai Xu, Tian Qiu, Feng Ji, Ji Zhang, and Haiqing Chen. AliMeKG: Domain knowledge graph construction and application in e-commerce. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2581–2588. ACM.

[5] Xiang Li, Gokhan Tur, Dilek Hakkani-Tur, and Qi Li. Personal knowledge graph population from user utterances in conversational understanding. *2014 IEEE Workshop on Spoken Language Technology, SLT 2014 - Proceedings*, pages 224–229, 04 2015.

[6] Prantika Chakraborty and Debarshi Kumar Sanyal. A personal knowledge graph for researchers. In *Proceedings of the 15th Annual Meeting of the Forum for Information Retrieval Evaluation*, FIRE '23, page 129–132, New York, NY, USA, 2024. Association for Computing Machinery.

[7] Krisztian Balog and Tom Kenter. Personal knowledge graphs: A research agenda. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '19, page 217–220, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Martin G. Skjæveland, Krisztian Balog, Nolwenn Bernard, Weronika Lajewska, and Trond Linjordet. An ecosystem for personal knowledge graphs: A survey and research roadmap.

[9] Julian McAuley, Rahul Pandey, and Jure Leskovec. Inferring networks of substitutable and complementary products.

[10] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Product knowledge graph embedding for e-commerce. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, pages 672–680. Association for Computing Machinery.

[11] Jiansong Zhang and Nora El-Gohary. Automated information extraction from construction-related regulatory documents for automated compliance checking.

[12] Bahar Sateli, Felicitas Löffler, Birgitta König-Ries, and René Witte. ScholarLens: extracting competences from research publications for the automatic generation of semantic user profiles. 3:e121.

[13] A. Mohanaraj and E. Niemeyer Laursen. Entity linking to dynamically-evolving personal knowledge graphs in conversations. Master's thesis, Aalborg University, Aalborg, Denmark, 2022.

[14] Anna Tigunova, Paramita Mirza, Andrew Yates, and Gerhard Weikum. Exploring personal knowledge extraction from conversations with CHARM. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, pages 1077–1080. Association for Computing Machinery.

[15] Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing dialogue agents: I have a dog, do you have pets too? Publisher: arXiv Version Number: 5.

[16] Chien-Sheng Wu, Andrea Madotto, Zhaojiang Lin, Peng Xu, and Pascale Fung. Getting to know you: User attribute extraction from dialogues. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 581–589. European Language Resources Association.

[17] Zhilin Wang, Xuhui Zhou, Rik Koncel-Kedziorski, Alex Marin, and Fei Xia. Extracting and inferring personal attributes from dialogue.

[18] Luyao Zhu, Wei Li, Rui Mao, Vlad Pandelea, and Erik Cambria. PAED: Zero-shot persona attribute extraction in dialogues. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9771–9787. Association for Computational Linguistics.

[19] Serkan GÜNEŞ. Extracting data-driven user segments and knowledge by using online product reviews. 11(1):139–152. Publisher: Gazi University.

[20] Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense x retrieval: What retrieval granularity should we use? *arXiv preprint arXiv:2312.06648*, 2023.

[21] Shon Otmazgin, Arie Cattan, and Yoav Goldberg. F-coref: Fast, accurate and easy to use coreference resolution. In *AACL*, 2022.

[22] Shon Otmazgin, Arie Cattan, and Yoav Goldberg. Lingmess: Linguistically informed multi expert scorers for coreference resolution, 2023.

[23] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[24] Lele Cao, Vilhelm von Ehrenheim, Mark Granroth-Wilding, Richard Anselmo Stahl, Andrew McCornack, Armin Catovic, and Dhiana Deva Cavalcanti Rocha. Companykg: A large-scale heterogeneous graph for company similarity quantification. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 4816–4827, New York, NY, USA, 2024. Association for Computing Machinery.

A Prompt for Triplet Extraction

```
def get_single_chat_message(self, proposition: str, labels=[]):

    user_content = f"""
        You are a top-tier algorithm designed for extracting information in
        structured formats to build a knowledge graph. Your task is to identify
        the entities and relations requested with the user prompt from a given
        text. You must generate the output in a JSON format containing a list
        with JSON objects. Each object should have the keys: "head",
        "head_type", "relation", "tail", and "tail_type". The "head" key must
        contain the text of the extracted entity with one of the types from the
        provided list in the user prompt.

        Attempt to extract as many entities and relations as you can. Maintain
        Entity Consistency: When extracting entities, it's vital to ensure
        consistency. If an entity, such as "John Doe", is mentioned multiple
        times in the text but is referred to by different names or pronouns
        (e.g., "Joe", "he"), always use the most complete identifier for that
        entity. The knowledge graph should be coherent and easily
        understandable, so maintaining consistency in entity references is
        crucial.

        IMPORTANT NOTES:
        - Don't add any explanation, text or code. Only return JSON.
        {'- If possible, try to use the following entity types: Person,
        Product, ' + ', '.join(labels) if labels else ''}

        Based on the following example, extract entities and relations from the
        provided text.

        Below are a number of examples of text and their extracted entities and
        relationships.

        Example:
        Text: Adam is a software engineer in Microsoft since 2009, and last
        year he got an award as the Best Talent
        Extracted Entities and Relations:
        [
        {{"head": "Adam", "head_type": "Person", "relation": "WORKS_FOR",
        "tail": "Microsoft", "tail_type": "Company"}}
        {{"head": "Adam", "head_type": "Person", "relation": "HAS_AWARD",
        "tail": "Best Talent", "tail_type": "Award"}}
        ]

        Example:
        Text: Microsoft is a tech company that provide several products such as
        Microsoft Word
        Extracted Entities and Relations:
        [
        {{"head": "Microsoft Word", "head_type": "Product", "relation":
        "PRODUCED_BY", "tail": "Microsoft", "tail_type": "Company"}}
        ]
```

```
        Example:
        Text: Microsoft is a tech company that provide several products such as
        Microsoft Word
        Extracted Entities and Relations:
        [
        {{"head": "Microsoft Word", "head_type": "Product", "relation":
        "PRODUCED_BY", "tail": "Microsoft", "tail_type": "Company"}}
        ]

        Example:
        Text: Microsoft Word is a lightweight app that accessible offline
        Extracted Entities and Relations:
        [
        {{"head": "Microsoft Word", "head_type": "Product", "relation":
        "HAS_CHARACTERISTIC", "tail": "lightweight app", "tail_type":
        "Characteristic"}},
        {{"head": "Microsoft Word", "head_type": "Product", "relation":
        "HAS_CHARACTERISTIC", "tail": "accessible offline", "tail_type":
        "Characteristic"}}
        ]

        The output should be formatted as a JSON instance that conforms to the
        JSON schema below.

        As an example, for the schema {{"properties": {{"foo": {{"title":
        "Foo", "description": "a list of strings", "type": "array", "items":
        {{"type": "string"}}}}}}, "required": ["foo"]}}
        the object {{"foo": ["bar", "baz"]}} is a well-formatted instance of
        the schema. The object {{"properties": {{"foo": ["bar", "baz"]}}}} is
        not well-formatted.

        Here is the output schema:

        {{"properties": {{"head": {{"title": "Head", "description": "extracted
        head entity like Microsoft, Apple, John. Must use human-readable unique
        identifier.", "type": "string"}}, "head_type": {{"title": "Head Type",
        "description": "type of the extracted head entity like Person, Company,
        etc", "type": "string"}}, "relation": {{"title": "Relation",
        "description": "relation between the head and the tail entities",
        "type": "string"}}, "tail": {{"title": "Tail", "description":
        "extracted tail entity like Microsoft, Apple, John. Must use
        human-readable unique identifier.", "type": "string"}}, "tail_type":
        {{"title": "Tail Type", "description": "type of the extracted tail
        entity like Person, Company, etc", "type": "string"}}}}, "required":
        ["head", "head_type", "relation", "tail", "tail_type"]}}


        For the following text, extract entities and relations as in the
        provided example.
        Text: {proposition}"""

    return [{"role": "user", "content": user_content}]
```