Utrecht University

MSc Artificial Intelligence

Master Thesis

A Data-Driven Decision Model for Software Package Selection

Parsa Beigzadeh 1203754

First Supervisor: Dr. Siamak Farshidi

Second Supervisor: Dr. Slinger Jansen

2024

Utrecht University

Faculty of Science

Abstract

The choice of software components is a crucial task in software engineering that has a big impact on the output and success of a project. The purpose of this thesis is to investigate the use of knowledge-based recommendation systems for software component selection. In order to identify previous studies and categorize them for the application of knowledge-based approaches in software architecture, the research will entail a systematic mapping study . The thesis aims to investigate the potential and difficulties of software repository mining for software engineering purposes. Specifically, it will examine how to extract software-related knowledge from various platforms . The results of this thesis will contribute to the creation of a knowledge-driven framework that will help software engineers make well-informed decisions about the selection of software components, close the gap between software and data engineering, and guide component selection. This thesis aims to improve the current state of software engineering by bringing together these different points of view and providing insightful analysis and useful suggestions for the effective use of knowledge-based systems in choosing software components.

I hereby extend my deepest gratitude to all those who have supported me throughout the course of this research. Foremost, I must acknowledge Siamak for his exemplary guidance and rigorous supervision. Our weekly meetings, characterized by robust discussion and critical analysis, were indispensable in shaping the trajectory and success of this work.
The past seven months have been marked by substantial challenges, encompassing a range of complex research methodologies. This period has necessitated unwavering dedication, patience, and a robust set of technical skills. It is during this time that I have discovered and honed previously untapped abilities within myself. The work encapsulated in this thesis stands as a testament to the significant personal and academic growth I have experienced throughout my studies at Utrecht University.
I also extend my sincere appreciation to my second supervisor, Slinger, and to all participants involved in this study. Their collective experiences and insights have been invaluable to the breadth and depth of this research.
In addition, I must express my profound gratitude to my parents. Their constant support, encouragement, and belief in my abilities have been a cornerstone of my journey.
To everyone who has contributed to the completion of this thesis, whether in a major or minor capacity, I offer my heartfelt thanks. Your roles, however varied, have been crucial in bringing this academic endeavor to fruition

Contents

Chapter 1

Introduction

Recommendation systems have become an indispensable component of our everyday existence, exerting a significant impact on the decisions we make across diverse domains, including but not limited to e-commerce, entertainment, social media, and software engineering [1]. These systems utilize data and algorithms to deliver personalized recommendations to users, improving user experience and satisfaction [1]. In recent times, there has been an increasing inclination towards the examination and enhancement of recommendation systems, resulting in notable progressions within this domain [1].
As an example, Within the field of software development, GitHub has gained significant popularity as a widely utilized platform for facilitating collaboration and the sharing of code. The authors of this study undertook an investigation into the practices of starring GitHub repositories and put forth four patterns to elucidate the progression of stars in said repositories [2]. The aforementioned patterns were obtained using clustering the time series data of stars, thereby offering valuable insights into developers' perspectives on these growth patterns. The recommendations derived from this study hold significant value for open-source project managers, as they provide insights into the significance of social coding practices.
The dynamic evolution of software within automated production systems has introduced various challenges and research avenues in the development of such recommendation systems [3]. Software packages are extensively utilized within the software industry due to their inherent benefits, including decreased development time and cost [4]. The aforementioned software packages are readily accessible on the market and can be chosen from a range of alternative software packages [5]. A software package is defined as a collection of software programs and files that are developed and distributed together to address specific functionalities or needs[6]. Software packages are the application domain that has attracted the most attention among the 10 chosen areas, according to a systematic literature review on twenty-eight years of software package-based software engineering by [7]. In fact, 25% of the research investigations involved software-packaged-based development. The fact that "reusing software packages from third-party providers is a key technology for developing systems quickly and cost-effectively" justifies this, according to the authors. The use of software packages has the potential to significantly reduce the costs associated with development, according to a study by [8]. The development of software systems solely within an organization can incur significant costs due to the need to recruit proficient developers, allocate resources to infrastructure investment, and commit resources to the development process [9]. In contrast, commercially available software packages can be easily obtained in the market, offering comparable functionality to custom-built solutions at a significantly reduced cost [8]. The affordability and efficiency of commercially available software packages render them an appealing choice for organizations that have restricted financial resources or stringent project

schedules [9].
The main way to access these packages is through software package managers. Software package managers are essential for organizing and streamlining the installation and upkeep of software by establishing standardized methods for creating and using software collections[10]. Package managers offer a reliable development environment and promote seamless reuse, which is in line with the convergence of software development and IT operations known as DevOps [10]. Nevertheless, the presence of various package managers and packages complicates the decision-making process. For instance, PYPI [1], a prominent package manager for Python, boasts approximately 487,657 distinct packages across various domains, including data science, web development, and machine learning, as per PYPI statistics.The second example is NuGet, which serves as package management for.NET development. It encompasses around 360,893 packets. In the table, we displayed the quantity of packages in the most well-known package repositories.

Repository	**Approximate Number of Packages**
PyPI (Python)	506,000
npm (JavaScript)	2,100,000
Maven (Java)	37,000,000
RubyGems (Ruby)	179,508
Packagist (PHP)	389,213
NuGet (.NET)	360,893
Crates.io (Rust)	133,961

Table 1.1: Number of Packages in Software Package Repositories (As of October 2023)

Furthermore, the download rate in these repositories indicates that they have played a crucial role in accessing the software packages. As demonstrated in figure 1.1, the daily download count from PYPI based on PYPI stats is approximately one billion. Moreover, approximately 429,520,831,740 downloads have been made overall for the Nuget repository [2]. In the table, we showed the number of total downloads for each package manager in table 1.1.
Therefore, the process of choosing software packages holds significant importance in the field of component-based software engineering (CBSE) [11]. Due to the wide variety of resources and their associated dependencies, developers must spend a considerable amount of time browsing for relevant resources [12]. Despite the need to better assist developers with this task, little research has been conducted on methods that make it simpler to discover relevant libraries from open source software (OSS). [13]. In order to accomplish this task, software engineers utilize a multiple-criteria decision-making (MCDM) approach to ascertain the most suitable software package. Initially, developers express their specific requirements and concerns. Subsequently, they proceed to analyze the various attributes of the software packages, including their features, quality, and evaluation metrics. In the final stage, an examination is conducted of the prevailing patterns within the market as well as the level of support from the community towards the software package. This guarantees that individuals select a software package that not only fulfills their present requirements but also adheres to industry norms and possesses a substantial user community for continuous assistance and enhancements.

[1] https://pypistats.org/
[2] https://www.nuget.org/

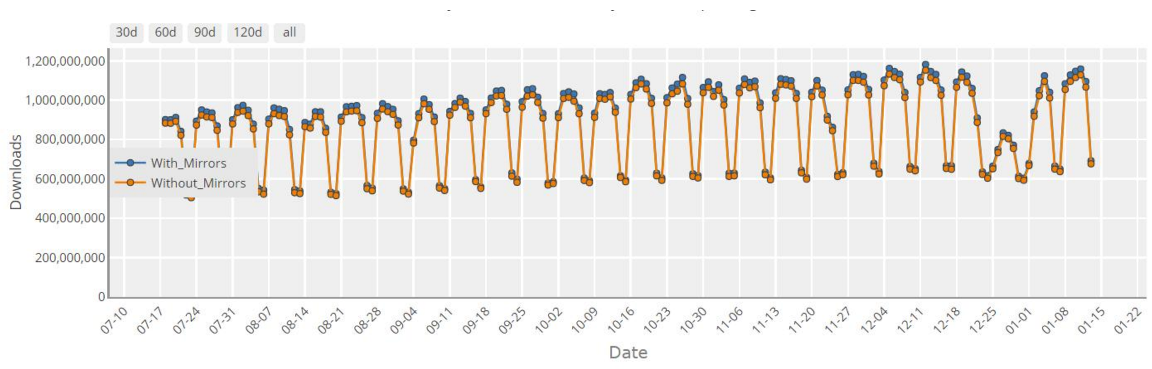


Figure 1.1: PYPI download rate

As a result of the challenges involved in choosing the best software packages, researchers and practitioners have developed software package recommendation systems. These resources are vital for assisting developers in identifying the best software packages for their projects. By pulling information from repositories like Nuget, npm, maven, pip, etc., these recommendation systems examine a variety of software package characteristics and attributes to offer developers unique and precise recommendations.
The primary contributions of this study are the following:

- Automated extraction of unstructured data from unstructured repositories
- New modeling and reasoning strategies for the compatibility of software packages
- A hybrid approach that incorporates content-based, collaborative, and knowledge-based recommenders
- Comprehensive evaluation and comparison of performance with current best practices

To enhance software package discovery and reuse, this thesis will design an intelligent recommendation system to address real-world challenges in software package-based development. The proposed methods have the potential to considerably improve the efficiency, cost, and caliber of software development. Effective COTS recommendations are a significant area of study with enormous potential for application in the real world.

Chapter 2

Research Approach

This chapter provides a detailed explanation of the methodology that will be employed in conducting the research and outlines the approach taken to address the research questions. At the outset, an assessment will be conducted to determine the research methods that will be employed. Subsequently, a comprehensive elucidation of each research methodology and potential challenges to the validity of the findings will ensue. In conclusion, we will provide a detailed analysis of the significant milestones achieved throughout the project.

2.1 Problem statement

The selection of unsuitable software packages can result in financial consequences. If the chosen software packages do not satisfy the stipulated criteria or encounter difficulties in achieving seamless integration, it may necessitate supplementary financial resources to substitute or adapt the software packages. Consequently, this could result in exceeding the allocated budget and prolonging the software development timeline [14]. Furthermore, it is important to note that organizations may face the need to pay licensing fees or ongoing support costs for the chosen software packages, which can affect the overall budget of the project [15]. An evaluation can be formulated as "multiple criteria decision making" (MCDM) problem. Any decision-making issue in the development of software can be described as an MCDM issue that deals with weighing a variety of options and criteria. Evaluation and selection of the best options for software engineers (decision-makers) based on their preferences and needs constitute the difficulty [16].
Consequently, there are still challenges, risks, and unknowns associated with this strategy. The improper selection of relevant software packages contributed to a portion of these risks and uncertainties [17]. The investigation commences with existing software package alternatives that are likely to satisfy the requirements. To retain only the most prospective alternative solution, the identified list is typically evaluated based on several fundamental criteria in addition to functionality. After carefully evaluating the retained options, the best option is selected. However, if the software package that best meets the requirements and can be reused with the least amount of effort and expenditure is not located or retained, another software package may be chosen. This increases the probability of project failure due to non-compliance with project expenses and deadlines and results in increased costs and effort. Therefore, the outcomes of the identification phase have a significant impact on the success of software package-based development[18]. Then, recommendation tools come in and play a role in these criteria. For instance, they introduced a way for evaluating accounting software by matching user needs with system specifications, giving a practical strategy for software selection in accounting [19]. Such recommendation tools have the following

drawbacks:

1. The results of the recommendations do not contain all of the recommended libraries. While certain popular libraries are more likely to get recommendations [20], others that are less well-known have a lower chance of doing so. Because it results in a lack of diversity and few recommendations, this is known as the "long tail problem" in the field of third-party library recommendation [21].

2. When neglecting the substantial auxiliary data of the third-party libraries and only considering the correlation between projects, the recommendation results are not properly refined.

3. Distinguishing between the relationship between projects and libraries by concentrating on either project level (project similarity) or library level (library usage pattern) for the suggestion[22].

2.2 Research questions

Relying on the problem statement, the following main research question was derived:

- **MRQ**: *How can one assist organizations or programmers in the process of selecting development software packages?*

To support this research question, we have created the following sub-research questions:

- **RQ1**: *Which software package selection approaches do exist in peer-reviewed literature?*

- **RQ2**: *Which criteria are used in the software package selection methods described in the literature? (identifying metadata features of software packages)*

- **RQ3**: *Which data collection methods can automatically extract metadata of software packages from developer communities like GitHub, Gitlab, etc., and package managers like NuGet, npm, maven, pip, etc.? (unsupervised)*

- **RQ4**: *How to develop the software package recommendation system tool, design it, and build the model?*

- **RQ5**: *How should the suggested software package recommendation model be evaluated?*

- **RQ6**: *How to understand the intent of developers when they are looking for software packages?*

2.3 Conceptual model

This section presents a novel pipeline specifically developed for a software recommendation system. We designed the conceptual model based on the conceptual model of a pipeline proposed [23]. This pipeline consists of five primary components, each playing a vital role in improving the functionality and precision of the system. The initial stage, software package metadata extraction, entails collecting comprehensive data regarding different software packages. Subsequently, the sentiment analysis module assesses the public's opinions and attitudes surrounding these software products. The user component is dedicated to comprehending and collecting user preferences and requirements. The inference engine, an essential component of the system, analyzes the gathered data to produce customized software suggestions. The integration and classification component effectively arranges the software products into relevant categories, streamlining user navigation and selection. Collectively, these elements collaborate harmoniously to generate a comprehensive and user-friendly software recommendation system. We will discuss it more in chapter 4 Figure 2.1 shows a conceptual model that shows the final study goal and how the decision model was made.

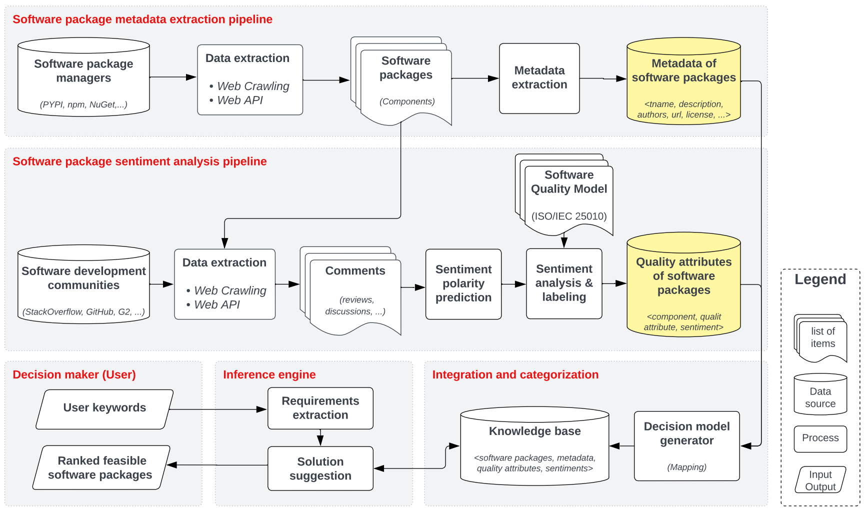

Software package metadata extraction pipeline
Software package managers
(PYPI, npm, NuGet,...)
Data extraction
• Web Crawling
• Web API
Software packages
(Components)
Metadata extraction
Metadata of software packages
<tname, description, authors, url, license, ...>
Software package sentiment analysis pipeline
Software Quality Model
(ISO/IEC 25010)
Software development communities
(StackOverflow, GitHub, G2, ...)
Data extraction
• Web Crawling
• Web API
Comments
(reviews, discussions, ...)
Sentiment polarity prediction
Sentiment analysis & labeling
Quality attributes of software packages
<component, qualit attribute, sentiment>
Decision maker (User)
User keywords
Ranked feasible software packages
Inference engine
Requirements extraction
Solution suggestion
Integration and categorization
Knowledge base
<software packages, metadata, quality attributes, sentiments>
Decision model generator
(Mapping)
Legend
list of items
Data source
Process
Input Output

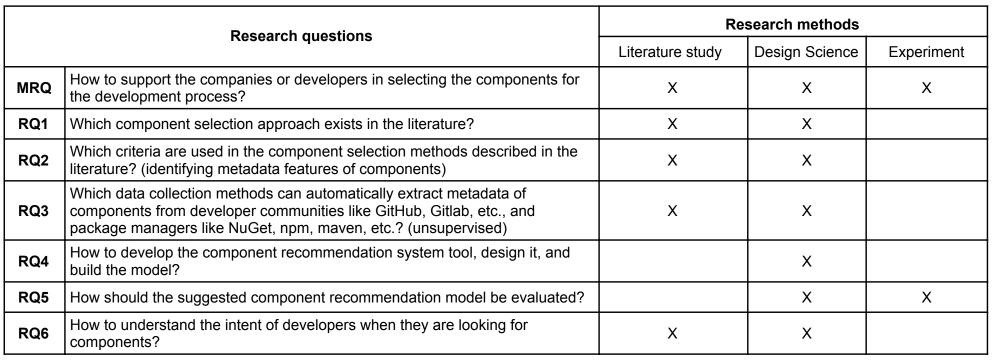
Figure 2.1: Conceptual model of software package recommendation system model.

2.4 Research Methods

The research questions outlined in the aforementioned section 2.2 will be addressed through various research methodologies. In this study, we will employ three distinct research methods: literature study, design science, and experiments. While each of these methods will be discussed in detail later on, the following table (Table 2.2) illustrates the corresponding research method that will address each of the research questions. When a specific method is chosen to address a particular research question, the intersection between them in the table will be marked with an "X".

2.4.1 Literature study

The research question highlighted in Section 2.2 was addressed in this study by adhering to the procedures and guidelines established by [24], [25], and [26]. Consequently, we implemented the review protocol 3.1 to methodically gather and extract data from pertinent studies. Finally, we attentively ensure that our research contains high-quality papers and references and diligently gather additional illuminating insights into the intricate workings of the software package recommendation system. We will discuss it completely in Chapter 3.



Research questions		Research methods		
		Literature study	Design Science	Experiment
MRQ	How to support the companies or developers in selecting the components for the development process?	X	X	X
RQ1	Which component selection approach exists in the literature?	X	X	
RQ2	Which criteria are used in the component selection methods described in the literature? (identifying metadata features of components)	X	X	
RQ3	Which data collection methods can automatically extract metadata of components from developer communities like GitHub, Gitlab, etc., and package managers like NuGet, npm, maven, etc.? (unsupervised)	X	X	
RQ4	How to develop the component recommendation system tool, design it, and build the model?		X	
RQ5	How should the suggested component recommendation model be evaluated?		X	X
RQ6	How to understand the intent of developers when they are looking for components?	X	X	

Figure 2.2: Research Method Employed

2.4.2 Design science

Design science is a research strategy that seeks to create and enhance artifacts, such as software systems, through the use of scientific methodologies. It entails the methodical development and assessment of creative answers to real-world issues [27]. Design science research aims to produce knowledge that professionals in a field can use to make recommendations as well as to share empirical insights from studies of how recommendations are used in practice [27].
Depending on the particular research problem and context, different steps may be involved in the design science research approach. There are some common software packages, though, that are frequently used in the process. The following is a summary of these actions:

1. **Problem Identification:** Identifying a problem or an opportunity for improvement in a specific domain is the first step in the design science research approach. A review of the literature, expert interviews, or observations of current procedures can all be used to accomplish this [28].

2. **Solution design:** The second step is to design a solution or artifact that solves the problem after it has been identified. To do this, a conceptual model or framework that outlines the essential elements and connections of the solution must be created [27].

3. **Artifact Development:** Following the design of the solution, the artifact or software system must be created. This could entail prototyping, programming, or other development tasks [29].

4. **Reflection:** Following the evaluation, it's crucial to consider the findings and the things we can learn from the design and development process. This may entail evaluating the artifact's advantages and disadvantages, pointing out potential problems, and offering suggestions for further study or application [27].

It is important to note that the design science research approach is cyclical and iterative, which means that the steps can be repeated numerous times as new information and input are gathered [27]. This enables both the artifact and the research process to be continuously improved. In conclusion, the design science research approach entails determining the issue at hand, coming up with a solution, creating an artifact, determining its viability, and considering the outcomes. In a variety of fields, including software engineering, this iterative process enables the development of creative and useful solutions to real-world problems.

2.4.3 Experiments

Recent research has prioritized the evaluation and comparison of different models to improve the comprehension of software package recommendation systems. These comparisons not only evaluate the efficacy of different models but also take into account the requirements of stakeholders and the objectives of the system [30]. Furthermore, incorporating advanced AI technologies such as ChatGPT as a benchmark in testing phases introduces a fresh method for evaluating recommendation models. This hybrid approach integrates conventional systems with AI breakthroughs, thereby pushing the limits of software suggestion and customisation. This innovative methodology is in line with the study by Smirnov and Ponomarev [31], who introduced a complex context-based model for recommendation systems. This model aims to improve the congruence between recommendations and user contexts [31].

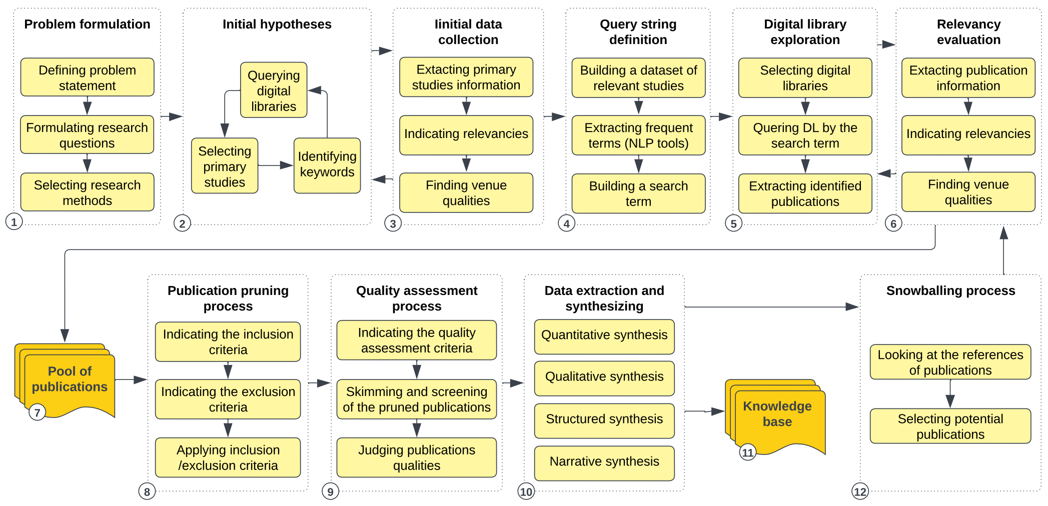
Chapter 3

Systematic Literature Review (SLR)

This chapter presents the results of our Systematic Literature Review (SLR), which we briefly introduced in Chapter 3. The primary objective of conducting this SLR was to gain a comprehensive understanding of the research landscape surrounding software package recommendation and its current trends. Additionally, through the SLR, we aimed to address several key research questions and gather pertinent data that would be instrumental in our research endeavors, particularly in the development of the final decision model. This comparative analysis with other approaches would facilitate the evaluation of our work.

The SLR was carried out in adherence to the guidelines proposed by certain references[24]. Furthermore, we drew upon the insights from a systematic literature review conducted by [32] as a point of reference during our own SLR.

In the following sub-chapters, we will delve into the process, elucidate our rationale behind each step, and elucidate the key findings stemming from our SLR. For those interested in a more detailed examination of the complete data set and results of the SLR, we have provided an appendix 8.3 where this information can be accessed (see figure 3.1).



Problem formulation
Defining problem statement
Formulating research questions
Selecting research methods
1
Initial hypotheses
Querying digital libraries
Selecting primary studies
Identifying keywords
2
Iinitial data collection
Extacting primary studies information
Indicating relevancies
Finding venue qualities
3
Query string definition
Building a dataset of relevant studies
Extracting frequent terms (NLP tools)
Building a search term
4
Digital library exploration
Selecting digital libraries
Quering DL by the search term
Extracting identified publications
5
Relevancy evaluation
Extacting publication information
Indicating relevancies
Finding venue qualities
6
Pool of publications
7
Publication pruning process
Indicating the inclusion criteria
Indicating the exclusion criteria
Applying inclusion /exclusion criteria
8
Quality assessment process
Indicating the quality assessment criteria
Skimming and screening of the pruned publications
Judging publications qualities
9
Data extraction and synthesizing
Quantitative synthesis
Qualitative synthesis
Structured synthesis
Narrative synthesis
10
Knowledge base
11
Snowballing process
Looking at the references of publications
Selecting potential publications
12

Figure 3.1: Systematic Literature Review Protocol.

3.1 Paper Collection

The comprehensive methodology for searching was thoroughly explained in Chapter 2. The methodology primarily consisted of two distinct search approaches: the initial hypothesis method and the automatic search method. The process of doing an initial hypothesis search allowed for the compilation of the initial collection of papers. This collection of papers then yielded a search term based on the common keywords found within them. The preceding search term was utilized to streamline the process of collecting data. The following sub-chapter will delve into the process of a comprehensive search.
The key sources utilized in this investigation include digital libraries, specifically:

- ACM Digital Library
- Springer Publishing
- IEEE Explore Digital Library
- ScienceDirect
- Scopus

The initial hypothesis was placed on these five libraries due to their provision of high-quality papers, which contribute substantial value to the scientific community. It is important to mention that the utilization of Google Scholar[1] was not employed throughout the automated search procedure due to its tendency to generate a substantial amount of irrelevant research and gray literature. Moreover, the considerable potential for overlap with the other libraries utilized in this systematic literature review (SLR) is apparent.

3.2 Search process

The whole search process and the number of collected papers are depicted in figure 3.1. During the automated search phase of our systematic literature review, we employed a robust search strategy to obtain relevant and high-quality papers from scientific search engines. To construct our search query, we derived keywords from a preliminary collection of papers acquired using a manual search procedure. In the manual search, we use fundamental search terms and knowledge to find more relevant and high-quality papers. We use these search terms:

- software package selection
- software cots recommendation
- software package recommendation
- cots recommendation system
- selection of software packages set
- software packages selection in the package manager

[1]scholar.google.com

The initial hypothesis search phase resulted in a collection of 49 papers. When we collect these papers, we use the Sketch Engine [33] to find out the most frequent keywords in these studies and rank them based on their frequency. This helps us identify the key areas of focus and trends within the research field. Then we created a search term, which is

("software development" OR "software production") AND ("COTS" OR "software packages") AND (" selection" OR "evaluation" OR "recommendation")

The search results can be exported in either CSV or Bibtex format, facilitating the systematic collection of documents and their subsequent insertion into the spreadsheet. Subsequently, the process entails the removal of redundant, impaired, or null data entries. During the process of automated retrieval, supplementary documents are collected. As a result, the combined efforts of the manual and automatic search phases resulted in a corpus of 6,250 documents. It is noteworthy to mention that our primary focus is directed towards scholarly articles that have been published after the year 2012. However, a few earlier documents have been included in the study as they were considered pertinent during the manual search or snowballing stages. These earlier documents were included to provide a historical context or to establish a foundation for the research. Additionally, the inclusion of these documents allows for a comprehensive analysis of the topic and ensures that no relevant information is overlooked.
Initially, a comprehensive search yielded a total of 6250 publications. These publications were then subjected to a rigorous process of inclusion and exclusion criteria. As a consequence, a final selection of around 353 papers of high standard was obtained. Therefore, the utilization of high-quality and contextually pertinent papers facilitated our ability to conduct a thorough and resilient analysis, enabling the extraction of significant features. The stringent selection procedure enabled us to concentrate on the most current and noteworthy research discoveries in our study while simultaneously recognizing the significance of earlier works that contribute to a comprehensive comprehension of the subject matter.

3.3 Inclusion and exclusion criteria

In the process of conducting a systematic literature review (SLR), the utilization of inclusion and exclusion criteria is of utmost importance for the purpose of picking relevant studies. These criteria ensure that the selected studies meet the necessary standards and directly pertain to the pertinent research inquiry.
In our study phase, we employed a rigorous approach to carefully choosing publications based on certain inclusion and exclusion criteria. This method was implemented to eliminate unreliable or low-quality sources from our analysis. Various factors were employed in our evaluation, including the caliber of the publication venue, the year of publication, the quantity of citations, and the pertinence of our research topic. Our review closely adhered to specific criteria in order to include papers that were of the greatest quality and relevance. Following a systematic procedure, the initial corpus of 6250 papers was effectively reduced to a final selection of 1063, which were subsequently chosen for further analysis.
A screening process was conducted for all papers obtained, including those contributed by the snowballing technique, after doing both manual and machine searches. During this stage, we conducted an assessment of the abstract, keywords, and overall relevance of each document to our research. The ranking of the items was determined based on their relevance, utilizing a scale consisting of four ordinal values: none, low, medium, and high. Subsequently, we employed inclusion and exclusion criteria to further refine our selection. A score was

calculated for each piece of work, taking into account the aforementioned elements.

3.4 Quality assessment

Evaluation of the caliber of primary research is crucial in addition to the inclusion and exclusion criteria. The evaluation of primary studies' quality results in more specific inclusion and exclusion criteria, provides recommendations for future research, directs the interpretation of findings, and gauges the reliability of inferences. It is possible to tell whether certain characteristics of research design or conduct have influenced the results by noting the strengths and limitations of primary studies [32]. We consider the following criteria for quality assessment:

- **Research Method:** We scrutinized whether the research methodology employed was appropriate for addressing our research query. Additionally, we assessed the transparency and lucidity of the research method.

- **Research Type:** We considered whether the publication presented original research, a review article, a case study, or a meta-analysis. We also evaluated the relevance and scope of the research within the machine learning field.

- **Data Collection Method:** Our evaluation included an assessment of the appropriateness of the data collection method in the context of our research question. We also examined the adequacy and clarity of the reported data collection process.

- **Evaluation Method:** We assessed whether the chosen evaluation method was fitting for addressing our research query, and we also considered the transparency and statistical significance of the reported outcomes.

- **Clear Problem Statement:** We evaluated whether the publication effectively identified the research problem and provided ample background information. We also examined the clarity and precision of the research question.

- **Research Questions:** We scrutinized the relevance, clarity, and precision of the research questions in relation to the research problem.

- **Research Challenges:** We assessed whether the publication acknowledged the challenges and limitations associated with the research.

- **Statement of Findings:** Our evaluation included an examination of whether the publication reported research findings and whether these findings were pertinent to the research problem and questions.

- **Real-World Use Cases:** We considered whether the publication provided practical, real-world use cases or applications for the proposed method or model.

After quality assessment, we derive about 343 papers, which is a significant number to provide a comprehensive analysis. These papers cover a wide range of perspectives and methodologies, allowing us to gain a holistic understanding of the topic and draw meaningful conclusions.

3.5 Data extraction and synthesizing

Our main goals during the data extraction and synthesis phase of the systematic literature review (SLR) were to respond to the precise research objectives we had selected and to learn more about the fundamental models frequently used by software package recommendation system developers. Understanding these models' properties, the quality factors connected to them, and the criteria for evaluation applied by researchers to evaluate their methods were among the goals we had set. We also looked at possible model combinations that researchers might use in their research publications. We also extract the software package's criteria, which also include the quality attribute. Additionally, we learned which evaluation criteria the researchers used in their investigations, which gave us a clearer picture of the software package selection. We methodically retrieved relevant information from the papers included in our review to achieve these goals. From our point of view, evaluation measures included a broad range of metrics and key performance indicators (KPIs) that researchers utilized to assess the performance and efficacy of their models.

3.5.1 Models and Methods

This study identified a total of 64 models about software package recommendation systems, sentiment models, and feature extraction. In Figure 3.2, we presented the trends of models from 2002 to 2023.

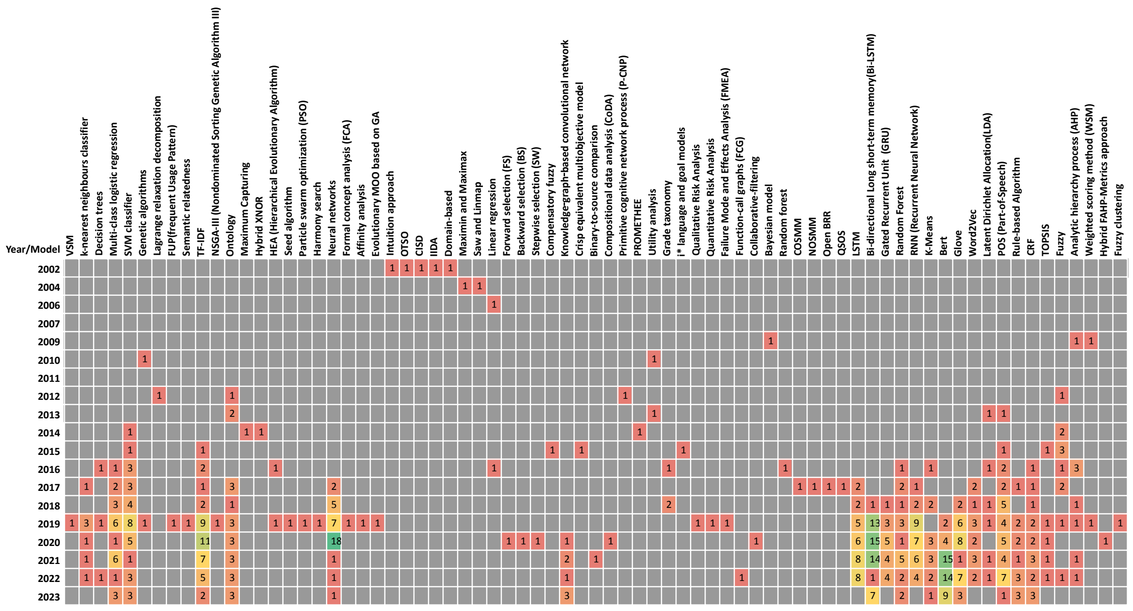
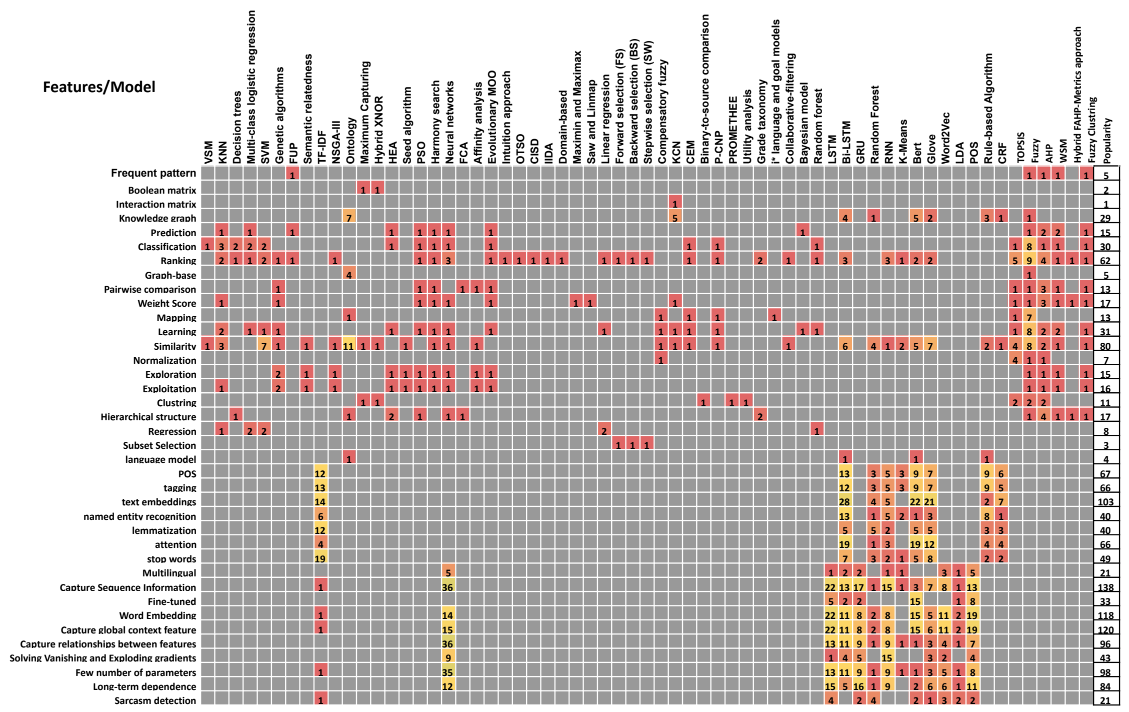


Figure 3.2: Evolution of Analytical Models and Methodologies from 2002 to 2023: A Chronological Overview Highlighting Key Trends and Developments in software package recommendation system.

Additionally, we have identified the characteristics of these models and determined the quantity of messages depicted in the figures 3.3.



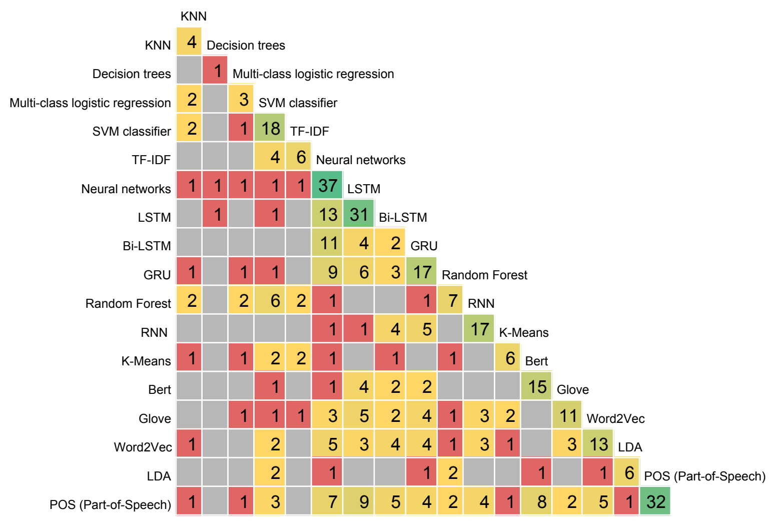
Features/Model	VSM	KNN	Decision trees	Multi-class logistic regression	SVM	Genetic algorithms	FUP	Semantic relatedness	TF-IDF	NSGA-III	Ontology	Maximum Capturing	Hybrid XNOR	HEA	Seed algorithm	PSO	Harmony search	Neural networks	FCA	Affinity analysis	Evolutionary MOO
Frequent pattern							1														
Boolean matrix												1	1								
Interaction matrix																					
Knowledge graph											7										
Prediction		1		1			1							1		1	1	1			1
Classification	1	3	2	2	2									1		1	1	1			1
Ranking		2	1	1	2	1	1			1						1	1	3			1
Graph-base											4										
Pairwise comparison						1										1	1		1	1	1
Weight Score		1				1										1	1	1			1
Mapping											1										
Learning		2		1	1	1								1		1	1	1			1
Similarity	1	3			7	1		1		1	11	1	1		1		1	1		1	
Normalization																					
Exploration						2		1		1				1	1	1	1	1		1	1
Exploitation		1				2		1		1				1	1	1	1	1		1	1
Clustring												1	1								
Hierarchical structure			1								1			2		1		1	1		
Regression		1		2	2																
Subset Selection																					
language model											1										
POS									12												
tagging									13												
text embeddings									14												
named entity recognition									6												
lemmatization									12												
attention									4												
stop words									19												
Multilingual																		5			
Capture Sequence Information									1									36			
Fine-tuned																					
Word Embedding									1									14			
Capture global context feature									1									15			
Capture relationships between features																		36			
Solving Vanishing and Exploding gradients																		9			
Few number of parameters									1									35			
Long-term dependence																		12			
Sarcasm detection									1												

Features/Model	Intuition approach	OTSO	CISD	IIDA	Domain-based	Maximin and Maximax	Saw and Linmap	Linear regression	Forward selection (FS)	Backward selection (BS)	Stepwise selection (SW)	Compensatory fuzzy	KCN	CEM	Binary-to-source comparison	P-CNP	PROMETHEE	Utility analysis	Grade taxonomy	i* language and goal models	Collaborative-filtering	Bayesian model	Random forest
Frequent pattern																							
Boolean matrix																							
Interaction matrix													1										
Knowledge graph													5										
Prediction																						1	
Classification														1		1							1
Ranking	1	1	1	1	1			1	1	1	1			1		1			2		1		1
Graph-base																							
Pairwise comparison																							
Weight Score						1	1						1										
Mapping												1		1		1				1			
Learning								1				1	1	1		1						1	1
Similarity												1	1	1		1					1		
Normalization												1											
Exploration																							
Exploitation																							
Clustring															1		1	1					
Hierarchical structure																			2				
Regression								2															1
Subset Selection									1	1	1												
language model																							
POS																							
tagging																							
text embeddings																							
named entity recognition																							
lemmatization																							
attention																							
stop words																							
Multilingual																							
Capture Sequence Information																							
Fine-tuned																							
Word Embedding																							
Capture global context feature																							
Capture relationships between features																							
Solving Vanishing and Exploding gradients																							
Few number of parameters																							
Long-term dependence																							
Sarcasm detection																							

Features/Model	LSTM	Bi-LSTM	GRU	Random Forest	RNN	K-Means	Bert	Glove	Word2Vec	LDA	POS	Rule-based Algorithm	CRF	TOPSIS	Fuzzy	AHP	WSM	Hybrid FAHP-Metrics approach	Fuzzy Clustring	Popularity
Frequent pattern															1	1	1		1	5
Boolean matrix																				2
Interaction matrix																				1
Knowledge graph		4		1			5	2				3	1		1					29
Prediction															1	2	2		1	15
Classification														1	8	1	1		1	30
Ranking		3			3	1	2	2						5	9	4	1	1	1	62
Graph-base															1					5
Pairwise comparison														1	1	3	1		1	13
Weight Score														1	1	3	1	1	1	17
Mapping														1	7					13
Learning														1	8	2	2		1	31
Similarity		6		4	1	2	5	7				2	1	4	8	2	1		1	80
Normalization														4	1	1				7
Exploration															1	1	1		1	15
Exploitation															1	1	1		1	16
Clustring														2	2	2				11
Hierarchical structure															1	4	1	1	1	17
Regression																				8
Subset Selection																				3
language model		1					1					1								4
POS		13		3	5	3	9	7				9	6							67
tagging		12		3	5	3	9	7				9	5							66
text embeddings		28		4	5		22	21				2	7							103
named entity recognition		13		1	5	2	1	3				8	1							40
lemmatization		5		5	2		5	5				3	3							40
attention		19		1	3		19	12				4	4							66
stop words		7		3	2	1	5	8				2	2							49
Multilingual	1	2	2		1	1			3	1	5									21
Capture Sequence Information	22	13	17	1	15	1	3	7	8	1	13									138
Fine-tuned	5	2	2				15			1	8									33
Word Embedding	22	11	8	2	8		15	5	11	2	19									118
Capture global context feature	22	11	8	2	8		15	6	11	2	19									120
Capture relationships between features	13	11	9	1	9	1	1	3	4	1	7									96
Solving Vanishing and Exploding gradients	1	4	5		15			3	2		4									43
Few number of parameters	13	11	9	1	9	1	1	3	5	1	8									98
Long-term dependence	15	5	16	1	9		2	6	6	1	11									84
Sarcasm detection	4		2	4			2	1	3	2	2									21

Figure 3.3: Comprehensive Matrix of Models and Features: Demonstrating the Application and Effectiveness of Various Computational Models Across Diverse Features in software package recommendation.

Sentiment models serve as a pivotal tool within recommendation systems, providing a nuanced understanding of user attitudes, opinions, and emotional states [34]. By integrating sentiment analysis into these systems, the potential for elevating recommendation reliability becomes evident [34]. One effective approach begins with the analysis of sentiment in user reviews, leveraging the polarity of these reviews to suggest items that encompass positive information for users [35]. This sentiment-driven analysis seamlessly complements explicit user ratings, adding an invaluable layer of insight to enhance the recommendation process [34]. In the figure 3.4, we depicted the models and their combinations. For example, we found a variety of models such as TF-IDF, LSTM, Part of Speech (POS), etc.
TF-IDF (Term Frequency-Inverse Document Frequency) is used in sentiment modeling for several reasons. TF-IDF is a widely used technique in information retrieval tasks, including software engineering-related tasks [36]. It is effective in capturing the importance of terms in a document by considering both their frequency in the document (TF) and their rarity in the entire corpus (IDF) [37].
The researcher used LSTM in sentiment modeling to predict user behavior based on textual data. LSTM, or long short-term memory, is a type of recurrent neural network that is particularly effective in capturing long-term dependencies in sequential data. By utilizing LSTM, the researcher aimed to improve the accuracy and effectiveness of sentiment modeling, ultimately contributing to the advancement of this field.
Sentiment modeling uses the POS (Part-of-Speech) for several reasons. A method known as POS tagging is used to categorize words in a text according to their grammatical function, such as nouns, verbs, adjectives, etc. Researchers can learn about the text's syntactic structure and identify useful features for comprehending user sentiments by combining POS data into sentiment modeling [38].



	KNN	Decision trees	Multi-class logistic regression	SVM classifier	TF-IDF	Neural networks	LSTM	Bi-LSTM	GRU	Random Forest	RNN	K-Means	Bert	Glove	Word2Vec	LDA	POS (Part-of-Speech)
KNN	4																
Decision trees		1															
Multi-class logistic regression	2		3														
SVM classifier	2		1	18													
TF-IDF				4	6												
Neural networks	1	1	1	1	1	37											
LSTM		1		1		13	31										
Bi-LSTM						11	4	2									
GRU	1		1	1		9	6	3	17								
Random Forest	2		2	6	2	1			1	7							
RNN						1	1	4	5		17						
K-Means	1		1	2	2	1		1		1		6					
Bert				1		1	4	2	2				15				
Glove			1	1	1	3	5	2	4	1	3	2		11			
Word2Vec	1			2		5	3	4	4	1	3	1		3	13		
LDA				2		1			1	2			1		1	6	
POS (Part-of-Speech)	1		1	3		7	9	5	4	2	4	1	8	2	5	1	32

Figure 3.4: Comparative Analysis of Sentiment Analysis Models: From Traditional Approaches to Advanced Neural Network Techniques.

Feature extraction techniques serve as a fundamental approach for mining domain knowledge from text data. Take, for example, the use of these techniques in the domain of package descriptions, where feature extraction models have been applied to automatically extract and characterize features using established criteria and subject modeling techniques [39]. This application empowers the recommendation engine with a more profound understanding of the traits and capabilities of various software packages, ultimately leading to more precise recommendations.

Within this context, an array of models has been identified, including Bert, Glove, multi-class logistic regression, and others. Furthermore, to provide a comprehensive overview, we have included Table 3.5 to illustrate these models and their relationships in the studies. This not only highlights the significance of feature extraction techniques but also underscores their pivotal role in enhancing recommendation systems.

Multi-class logistic regression is a suitable choice for feature extraction models for several reasons. Firstly, logistic regression is a well-established and widely used classification algorithm [40]. It estimates the probabilities of different possible outcomes of a categorically distributed dependent variable, given a set of independent variables [40]. This makes it particularly suitable for multi-class classification tasks, where there are more than two possible classes to predict.

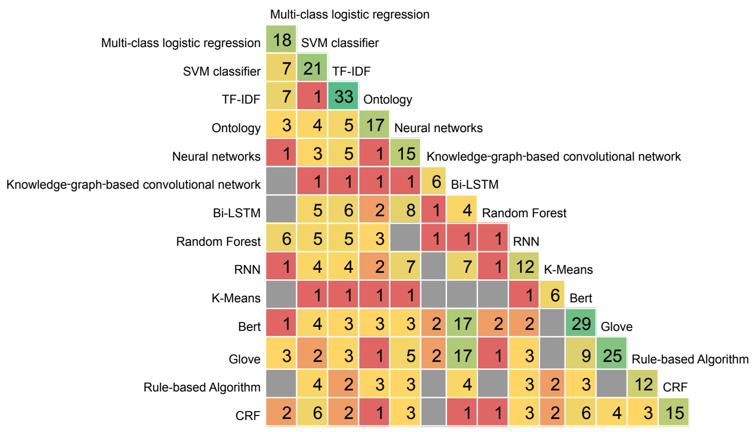


Figure 3.5: Overview of Feature Extraction Models in Computational Analysis: Charting the Progression from Basic Techniques to Advanced Algorithms.

The final models employed in our study are recommendation models specifically designed for usage within the area of COTS software packages. These models provide a promising perspective for the development of an inference engine that can identify the relevant software packages associated with the data gathered using sentiment and feature extraction models. Next, it is necessary to prioritize the outcomes to display the most convenient and appropriate software packages. we showed the models in figure 3.6

For recommendation models, there are some famous types of models, such as machine learning models and MCDM models. Machine learning models can be SVM, logistic regression, or neural networks. Support Vector Machines (SVM) have gained recognition for their capability to properly manage high-dimensional data and efficiently handle extensive feature spaces [41]. This is especially advantageous in recommendation systems when multiple factors need to be taken into account, including user preferences, object qualities, and contextual information. Support Vector Machine (SVM) has the capability to handle intricate feature spaces and yield precise predictions effectively.
A neural network can capture complex patterns and relationships in the data, allowing them to make accurate predictions and recommendations [42]. They can learn non-linear relationships between user preferences and item features, which can lead to more personalized and accurate recommendations. On the other hand, the researcher used MCDM's models, such as Fuzzy logic and AHP.
Fuzzy logic allows for the modeling and control of complex systems with interactive effects of variables [43]. It compensates for the deficiencies of traditional Boolean logic and improves the modeling process [43]. This is particularly useful in recommendation models where there are multiple variables and interactions to consider.
The Analytic Hierarchy Process (AHP) provides a comprehensive and rational framework for structuring a decision problem, representing and quantifying its elements, relating those elements to overall goals, and evaluating alternative solutions [44]. This makes it a robust and systematic approach to decision-making.

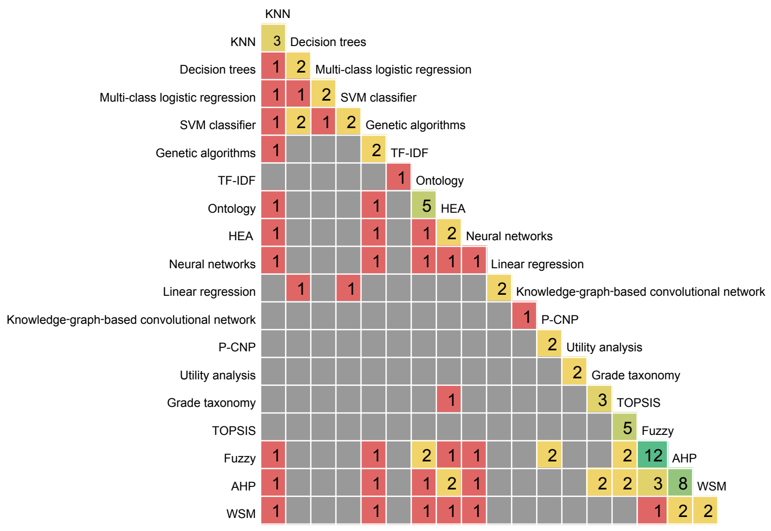
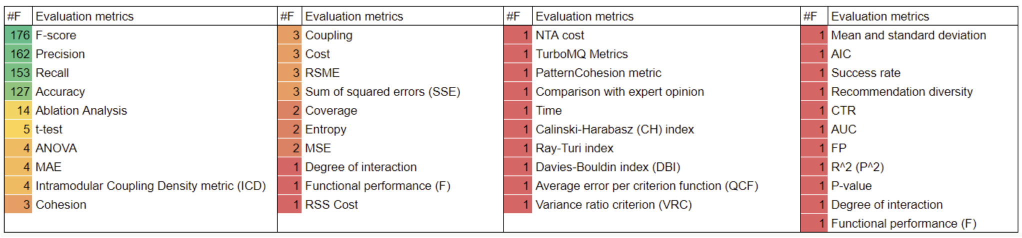


Figure 3.6: Diverse Recommendation Models: A concise mapping of key models such as KNN, SVM, and Neural Networks, highlighting their functionalities in recommendation systems.

3.5.2 Models' evaluation methods

Evaluation measurements for models used in recommendation systems are crucial in order to accurately analyze the system's performance and efficacy. In the research that has been done on evaluating recommendation systems, a number of distinct assessment metrics that take into account various features of the recommending process have been offered. The result has been shown in figure 3.7. We The result has been shown in figure 3.7.We discovered roughly 41 evaluation measures, with precision, recall, F1 score, accuracy, ablation analysis, and the t-test being the most often employed ones. The accuracy of suggestions, the capacity to find pertinent items, the harmony between precision and recall, and the variety of recommendations are only a few of the characteristics of recommendation systems that these metrics shed light on. However, the specific objectives and demands of the recommendation system being evaluated must be taken into consideration while choosing the best evaluation metric.

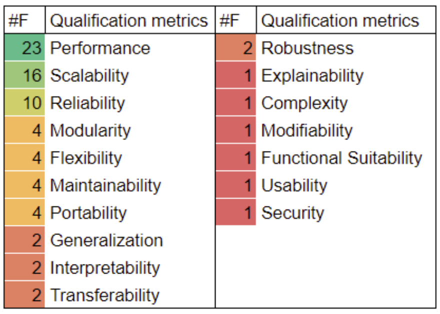


#F	Evaluation metrics	#F	Evaluation metrics	#F	Evaluation metrics	#F	Evaluation metrics
176	F-score	3	Coupling	1	NTA cost	1	Mean and standard deviation
162	Precision	3	Cost	1	TurboMQ Metrics	1	AIC
153	Recall	3	RSME	1	PatternCohesion metric	1	Success rate
127	Accuracy	3	Sum of squared errors (SSE)	1	Comparison with expert opinion	1	Recommendation diversity
14	Ablation Analysis	2	Coverage	1	Time	1	CTR
5	t-test	2	Entropy	1	Calinski-Harabasz (CH) index	1	AUC
4	ANOVA	2	MSE	1	Ray-Turi index	1	FP
4	MAE	1	Degree of interaction	1	Davies-Bouldin index (DBI)	1	R^2 (P^2)
4	Intramodular Coupling Density metric (ICD)	1	Functional performance (F)	1	Average error per criterion function (QCF)	1	P-value
3	Cohesion	1	RSS Cost	1	Variance ratio criterion (VRC)	1	Degree of interaction
						1	Functional performance (F)

Figure 3.7: Key Evaluation Metrics in Recommendation Systems: A spectrum of metrics from F-score to Functional Performance, indicating their frequency of use in research.

3.5.3 Models' qualification methods

Measurements for qualification are absolutely necessary in order to evaluate the effectiveness and dependability of the models used in a software package recommendation system. These measurements contribute to determining whether or not the models are suitable for use in formulating correct suggestions [45]. We have extracted 17 metrics, illustrated in figure 3.8, with performance, scalability, reliability, modularity, flexibility, maintainability, and portability being the most popular ones. These metrics assist in evaluating the recommendation system's overall efficiency and quality. Scalability assesses the system's capacity to cope with growing volumes of data and users, while performance gauges how effectively the system operates. While modularity and flexibility evaluate the system's ability to adapt to various environments and user preferences, reliability ensures that the system consistently delivers accurate recommendations. While portability assesses how easily the system can be moved to various platforms or devices, maintainability concentrates on how simple it is to update and address problems. Together with each other, these metrics offer a thorough analysis.



#F	Qualification metrics	#F	Qualification metrics
23	Performance	2	Robustness
16	Scalability	1	Explainability
10	Reliability	1	Complexity
4	Modularity	1	Modifiability
4	Flexibility	1	Functional Suitability
4	Maintainability	1	Usability
4	Portability	1	Security
2	Generalization		
2	Interpretability		
2	Transferability		

Figure 3.8: Distribution of Quality Attributes in System Evaluation: Showcasing frequency counts of metrics such as Performance, Scalability, and Security in research applications.

In the process of our study, we have identified some essential quality standards that serve critical functions in the evaluation of software package quality and the guarantee of its adherence to specific criteria. The standards encompassed in this list are ISO/IEC 9126, ISO SQUARE, ISO/IEC 25010, ISO/IEC/IEEE 42010, and ISO/IEC 27002:2005. These standards offer comprehensive criteria for assessing the quality of software, with each fulfilling a unique objective:

- **ISO/IEC 9126:** This plays a fundamental role in evaluating the quality of software packages within the domain of the COTS recommendation system. This methodology facilitates the methodical assessment of essential attributes, including functionality, reliability, usability, efficiency, maintainability, and portability. This particular standard assists in evaluating the extent to which a software package fulfills the intended quality attributes, thereby enabling well-informed suggestions within the system.

- **ISO SQUARE:** The software package recommendation system directly benefits from SO SQUARE's emphasis on assessing the quality of software products. It offers a methodical approach and predetermined metrics for evaluating various software package properties impartially. This makes it easier to quantify qualities like correctness, reliability, and maintainability, all of which are essential when recommending a software package.

- **ISO/IEC 25010:** As part of the software package recommendation system, ISO/IEC 25010 is essential because it provides a thorough framework for assessing the quality of software packages. It offers a comprehensive viewpoint that takes into account qualities like usability, functionality, reliability, and security. This standard guarantees that the suggested software adheres to these varied quality characteristics when recommending software packages, improving the user experience overall.

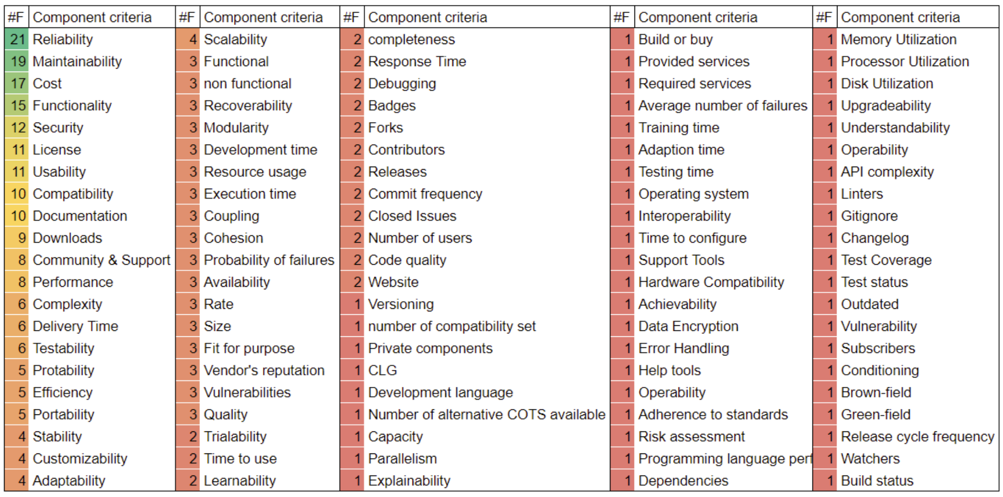
- **ISO/IEC/IEEE 42010:** When analyzing the architectural features of software packages within the recommendation system, ISO/IEC/IEEE 42010 is especially pertinent. It helps with describing, analyzing, and evaluating software package architecture, ensuring that suggested software packages are appropriate for integration and match project objectives and accepted industry standards.

- **ISO/IEC 27002:2005:** which focuses primarily on information security, is essential to the software package recommendation system because it emphasizes security issues. The system's users and their data are protected against potential threats and vulnerabilities by ensuring that the recommended software packages adhere to fundamental security standards.

These quality standards serve as a structured and exacting framework for evaluating software packages and are an essential part of the software package recommendation system. The effectiveness and dependability of the recommendation process are increased by the system's ability to confidently recommend software packages that not only satisfy user requirements but also follow industry-recognized quality benchmarks.

3.5.4 software package criteria

Multiple criteria and features must be taken into account when recommending software packages in a software package recommendation system. These standards and characteristics ensure that the chosen software packages satisfy the system's functional requirements and adhere to the particular requirements of the project. We discovered approximately 105 software package features and criteria using SLR, as shown in the figure 3.9. Both quality and non-quality attributes are included in these features. We acquired the requirements for quality attributes in the previous section, but we also need non-quality features.
The most frequently used software package features are reliability, maintainability, cost, functionality, security, license, compatibility, usability, and performance. These characteristics allow the software package recommendation system to give preference to parts that have a track record of reliability and are simple to maintain. Cost should be taken into account, as it should be in line with the project's budget while still achieving the necessary functionality and security standards. In addition, seamless integration and troubleshooting depend on compatibility with current systems and the availability of thorough documentation and support.
To summarize, this chapter provides a thorough examination of the literature pertaining to software package recommendation algorithms. The document provides a comprehensive explanation of the process for choosing research papers, which includes the standards for including or excluding articles and the evaluation of the quality of primary studies. The chapter explores many models and methodologies employed in these systems, including sentiment analysis and feature extraction techniques. This establishes a fundamental comprehension for the upcoming chapter, which will concentrate on providing a more thorough explanation



#F	Component criteria	#F	Component criteria	#F	Component criteria	#F	Component criteria	#F	Component criteria
21	Reliability	4	Scalability	2	completeness	1	Build or buy	1	Memory Utilization
19	Maintainability	3	Functional	2	Response Time	1	Provided services	1	Processor Utilization
17	Cost	3	non functional	2	Debugging	1	Required services	1	Disk Utilization
15	Functionality	3	Recoverability	2	Badges	1	Average number of failures	1	Upgradeability
12	Security	3	Modularity	2	Forks	1	Training time	1	Understandability
11	License	3	Development time	2	Contributors	1	Adaption time	1	Operability
11	Usability	3	Resource usage	2	Releases	1	Testing time	1	API complexity
10	Compatibility	3	Execution time	2	Commit frequency	1	Operating system	1	Linters
10	Documentation	3	Coupling	2	Closed Issues	1	Interoperability	1	Gitignore
9	Downloads	3	Cohesion	2	Number of users	1	Time to configure	1	Changelog
8	Community & Support	3	Probability of failures	2	Code quality	1	Support Tools	1	Test Coverage
8	Performance	3	Availability	2	Website	1	Hardware Compatibility	1	Test status
6	Complexity	3	Rate	1	Versioning	1	Achievability	1	Outdated
6	Delivery Time	3	Size	1	number of compatibility set	1	Data Encryption	1	Vulnerability
6	Testability	3	Fit for purpose	1	Private components	1	Error Handling	1	Subscribers
5	Protability	3	Vendor's reputation	1	CLG	1	Help tools	1	Conditioning
5	Efficiency	3	Vulnerabilities	1	Development language	1	Operability	1	Brown-field
5	Portability	3	Quality	1	Number of alternative COTS available	1	Adherence to standards	1	Green-field
4	Stability	2	Trialability	1	Capacity	1	Risk assessment	1	Release cycle frequency
4	Customizability	2	Time to use	1	Parallelism	1	Programming language perf	1	Watchers
4	Adaptability	2	Learnability	1	Explainability	1	Dependencies	1	Build status

Figure 3.9: Software Package Evaluation Criteria: Frequencies of various criteria like Reliability, Cost, and Security used to assess software packages.

of the chosen models and characteristics, as well as connecting theoretical knowledge to practical implementations in the field.

Chapter 4

Decision Model

Decision theories are utilized in various fields, including software development [46] and e-learning [47]. However, it is important to note that decision-makers may approach difficulties differently due to variations in individual priorities, implicit knowledge, and decision-making methods [48]. The primary objective of the field of MCDM is to address these discrepancies in judgment through the utilization of decision models.
In scenarios involving MCDM, a set of options is assessed, and decision criteria are considered [16]. The challenge lies in determining the optimal software packages based on the requirements and preferences of decision-makers [49]. It is imperative to bear in mind that there is no universally optimal solution for MCDM problems, and the preferences of decision-makers play a crucial role in identifying the most suitable answer for their needs [49]. In the context of models and features for software package recommendation systems, this research focuses on the software package selection problem and treats it as an MCDM problem.
The decision model presented in Figure 2.1, which utilizes the MCDM theory, will be highly valuable for researchers involved in the development of software package recommendation systems. This approach facilitates systematic exploration of options, considering key factors in the process of software package selection, and determining the optimal configuration of software packages for the development of an effective recommendation model. This approach has three phases, as follows:

4.1 Software Package Metadata Extraction Pipeline

The Software Package Metadata Extraction Pipeline is a vital element of the software recommendation system, specifically created to collect and evaluate crucial data from different software package managers. This pipeline effectively gathers metadata, such as name, description, author, and license, from repositories such as PYPI, npm, and NuGet by employing advanced techniques like web crawling and web APIs. This approach allows the system to acquire profound insights into software packages, enabling more knowledgeable and customized recommendations for users.

Data Source: Software Package Managers

The software package recommendation system in this project exclusively obtained data from PyPI, given the project's magnitude. By adopting this targeted strategy, we were able to work with a dataset that was easier to handle while still benefiting from a wide range of Python programs. Alternative package managers were excluded from consideration due to their potential impact on project scope and manageability. PyPI, often known as the Python

Package Index, is a well-known repository that houses a wide range of Python packages and modules. As of 2024, PyPI has a total of 506,000 projects, exceeding 5 million releases and 10 million files. The user base of PyPI is estimated to be over 775,000. Due to the wide range of software that the Python community has produced, PyPI is a crucial resource for Python developers.

Data Extraction Process

Data extraction encompasses two primary methodologies: web crawling and the utilization of web APIs. **Web crawling** is an automated process used to gather data from the web pages of package managers. We used Selenium, a Python web scraping tool, to accomplish this. The crawler systematically navigates to every page and retrieves pertinent data. PYPI offer **APIs** that enable more organized and efficient retrieval of their data. By utilizing these APIs, the system is able to directly solicit precise information regarding software packages.

Software Packages

By utilizing web crawling techniques and APIs, we have effectively gathered an extensive collection of packages from the PYPI package manager, which has been organized and presented in JSON format. The collection is comprehensive and contains a diverse range of software programs, each with distinct functionalities, dependencies, and development histories.
The PyPI repository contains a wide variety of software packages, including libraries, frameworks, and tools, which are designed to meet various programming requirements. These utilities vary from commonly used tools that fulfill general programming needs to specialized libraries specifically intended for specific applications.

Metadata Extraction

The recommendation system can learn more about the variety of software packages that are available, their features and abilities, and how developers perceive or use them through the extraction and analysis of this metadata. Having this comprehension is crucial in order to offer precise and pertinent suggestions to individuals seeking software packages that align with their particular needs.
Metadata extraction entails the extraction of certain data points related to each software package. Typically, this includes elements like Name, URL, and Description. The extracted metadata is essential for comprehending the functionality and possible applications of each package. We will further elaborate on chapter 5.

4.1.1 Software Package Sentiment Analysis Pipeline

We employed Liang Feng's [50] findings, which encompass a comprehensive sentiment analysis framework specifically tailored for evaluating software packages. This pipeline extensively relies on user feedback obtained from many software development forums, including StackOverflow, GitHub, and G2. It employs sophisticated natural language processing algorithms. It systematically gathers and examines user-generated content's sentiments in order to provide a thorough understanding of the product's perceived quality and usability. This technique facilitates software developers in implementing data-driven enhancements in their products.
During its concluding phases, the pipeline combines the outcomes of sentiment analysis with the ISO/IEC 25010 software quality model. This model delineates fundamental quality attributes such as functionality, reliability, and maintainability. By correlating user feedback with

these characteristics, the pipeline provides a thorough assessment of software components, taking into account their actual usage and opinions. By adhering to the ISO standard, the analysis is comprehensive and in line with established quality standards, thereby facilitating the improvement of software packages to better cater to user requirements.

4.1.2 Inference Engine

The inference engine is a critical part of the system, processing the user's input and delivering recommendations. It includes two parts, as follow:

Requirements Extraction

This procedure entails scrutinizing the user keywords in order to extract precise requirements or criteria that the software package must fulfill. It uses name entity recognition (NER) to extract the most important keywords from the input keywords. we will discuss it more in 5

Solution Suggestion

The inference engine proposes potential software solutions based on the extracted requirements. This stage entails aligning the user's requirements with the attributes of the software packages that are already accessible. It works with the similarity algorithm, as we found that in a literature review, similarity is the most commonly used in a recommendation system. We used BM25, LM Jelinek Mercer similarity (LMJDM), Information-based (IB), and LM Dirichlet similarity (LMD).

4.1.3 Integration and Categorization

Subsequently, the proposed solutions are consolidated into a cohesive collection of recommendations and classified according to their relevance and appropriateness to the user's requirements. We created this knowledge base using JSON files.

Knowledge Base

The knowledge base is an extensive repository that encompasses all the fundamental facts utilized by the system to facilitate decision-making. The following items are included: The metadata for software packages is gathered from multiple package managers.The sentiment analysis pipeline determines the quality attributes of the software components. User opinion analysis is linked to each individual software element.

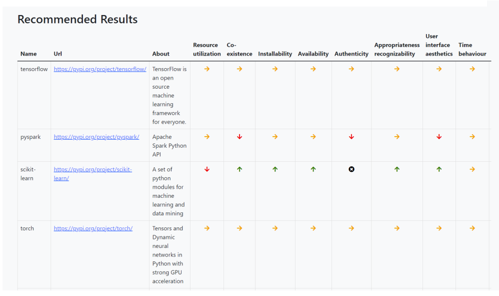
4.1.4 User

User Keywords

In this step, user keywords pertain to the user's requirements, preferences, or specific use cases. These are frequently presented in a keyword structure and can range from being simple, complex, or ambiguous. These keywords facilitate comprehension of the circumstances and distinct requirements of the user, serving as the foundation for tailored suggestions.

Ranked Feasible Components

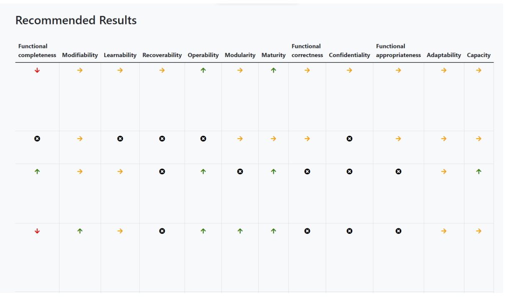
This step will display the candidate on a list. Name, short description, URL, and ISO 25010 quality attributes are all included in this list. We provided an illustration in the figure 4.1.



Recommended Results

Name	Url	About	Resource utilization	Co-existence	Installability	Availability	Authenticity	Appropriateness recognizability	User interface aesthetics	Time behaviour
tensorflow	https://pypi.org/project/tensorflow/	TensorFlow is an open source machine learning framework for everyone.	→	→	→	→	→	→	→	→
pyspark	https://pypi.org/project/pyspark/	Apache Spark Python API	→	↓	→	→	↓	→	↓	→
scikit-learn	https://pypi.org/project/scikit-learn/	A set of python modules for machine learning and data mining	↓	↑	↑	↑	⊗	↑	↑	→
torch	https://pypi.org/project/torch/	Tensors and Dynamic neural networks in Python with strong GPU acceleration	→	→	→	→	→	→	→	→

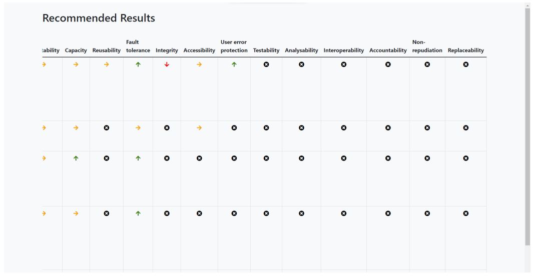
1



Recommended Results

Functional completeness	Modifiability	Learnability	Recoverability	Operability	Modularity	Maturity	Functional correctness	Confidentiality	Functional appropriateness	Adaptability	Capacity
↓	→	→	→	↑	→	↑	→	→	→	→	→
⊗	→	⊗	⊗	⊗	→	→	→	⊗	→	→	→
↑	→	→	⊗	↑	⊗	↑	⊗	⊗	⊗	→	↑
↓	↑	→	⊗	↑	↑	↑	⊗	⊗	⊗	→	→

2



Recommended Results

ability	Capacity	Reusability	Fault tolerance	Integrity	Accessibility	User error protection	Testability	Analysability	Interoperability	Accountability	Non-repudiation	Replaceability
→	→	→	↑	↓	→	↑	⊗	⊗	⊗	⊗	⊗	⊗
→	→	⊗	→	⊗	→	⊗	⊗	⊗	⊗	⊗	⊗	⊗
→	↑	⊗	↑	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗
→	→	⊗	↑	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗

3

Figure 4.1: Final results.

4.2 Design Decisions

In this project, we have leveraged a fusion of three distinct model categories, as comprehensively detailed in Section 3.5.1. Our approach to feature extraction is centered on the utilization of text embedding techniques.
Text embedding: Text embedding refers to the process of converting textual data, such as words, sentences, or documents, into numerical vectors while preserving their semantic meaning. This numerical representation allows us to effectively work with and analyze text data using machine learning algorithms.
Additionally, within the architecture of our recommendation system engine, we have employed a multifaceted strategy that encompasses several key software packages:
Similarity Analysis: Similarity analysis involves quantifying the likeness or resemblance between items or user preferences. This analysis helps us measure how closely aligned user preferences are with available items, enabling us to make recommendations that closely match individual tastes. **Ranking:** Ranking algorithms determine the order in which recommendations are presented to users. These mechanisms take into consideration various factors, including user behavior, item characteristics, and relevance, to provide more accurate and personalized recommendations.
This comprehensive approach allows us to provide enhanced and personalized recommendations to our users, tailoring our suggestions to their unique preferences and interests. In the domain of sentiment analysis, our methodology is further enriched by the incorporation of the following elements:
Sequence Information: Sequence information pertains to the order and arrangement of words or phrases in a text. It is crucial for understanding the flow and nuances of sentiment expressed in textual content.
Global Context Feature: The global context feature involves considering the broader context of the entire text, such as the overarching theme or tone. This context is essential for

gaining a deeper understanding of the sentiment within the text.

4.3 Model Architecture

During our model selection procedure, we conducted a methodical investigation to find models that meet our specific feature criteria. This part offers an in-depth analysis of the essential components relevant to our duties, guaranteeing that our selection of models is appropriately customized to fulfill these requirements. Based on the literature review we did in chapter three, 3 the most common feature used for the model is similarity.
Our technique relied on incorporating Named Entity Recognition (NER) to extract features and perform sentiment analysis. NER is a crucial technique in the field of natural language processing. It is highly skilled at detecting and classifying important details in text, such as the names of individuals, locations, organizations, and other entities. The capacity of Named Entity Recognition (NER) is crucial for extracting significant and contextually appropriate information from text, which is necessary for various NLP activities, such as sentiment analysis [51]. In our particular use case, we employed Named Entity Recognition (NER) approaches to analyze and extract important entities from our dataset. The collected entities were further examined using advanced approaches such as BM25, LMJL, and IB and LM Dirichlet ranking within Elastic Search. These similarity models offer a more advanced strategy for assessing document relevance, particularly in the field of information retrieval, in contrast with traditional cosine similarity methods. Key concepts employed in our methodology include:

- **BM25:** BM25 is a ranking function used for information retrieval. It helps to quantify the relevance of documents in a corpus to a given search query. In our context, BM25 is used to evaluate the relevance of software packages based on extracted entities, aiding in identifying the most pertinent software options for users.

- **Language Model Dirichlet (LM Dirichlet):** LM Dirichlet similarity is a smoothing technique used in information retrieval to handle the problem of zero probability in language models. It incorporates Dirichlet prior smoothing, which helps in estimating the probability of unseen words in a document. This approach improves the performance of language models by balancing between the observed term frequency and the background corpus frequency.

- **Language model Jelinek-Mercer (LM Jelinek-Mercer):** The LM Jelinek-Mercer is a language model smoothing technique that combines two different models: a maximum likelihood estimate (MLE) model and a background model. It uses a linear interpolation method to balance the probability estimates from these models, with a tuning parameter to control the contribution of each. This approach enhances the model's performance in language tasks by reducing the issues of data sparsity and overfitting.

- **Information-based (IB):** IB in information retrieval measures the proximity of documents or phrases by considering the amount of shared information content. Relevance is determined by assessing the extent of shared information, hence improving the precision of recognizing associated things. Adopting this strategy is essential for obtaining accurate search results and achieving efficient data clustering.

- **Clustering in Elastic Search:** The use of ranking techniques within the Elastic Search database system enables the systematic organization and grouping of JSONs. These JSONs correspond to various software packages, facilitating the efficient recommendation of suitable options

It is crucial to acknowledge that NER plays a vital role in enhancing the performance of our model. It is integrated into the design primarily for extracting features and conducting sentiment analysis. Additional components and algorithms in our approach also play key roles in attaining the intended outcomes.

Chapter 5

Pipeline Implementation

In the continually evolving domain of software development, the careful choice of suitable software packages plays a crucial role in guaranteeing the success of a project, which is defined by achieving the highest level of performance and dependability. This chapter explores the essential stages of data preparation and extraction, establishing the basis for a data-driven strategy in selecting software packages. Our goal is to simplify the process and procedures used to convert raw, unstructured data into a format that supports informed decision-making. We employ state-of-the-art tools and methodologies to explore the practical implementation of these principles in software package selection, facilitating a comprehensive discussion.
The data preparation and extraction procedure in this study consists of two main steps: firstly, obtaining relevant data by web scraping, and secondly, extracting and processing this data using Natural Language Processing (NLP) techniques.
The following sections of this chapter will present a comprehensive review of the strategies utilized in the data preparation and extraction processes. We will examine the precise web scraping tools and methodologies employed, followed by a thorough investigation of the NLP approaches utilized for data processing. The purpose of this chapter is to establish a strong basis for comprehending the process of converting unprocessed data from many internet sources into an organized data set that is prepared for analysis and utilization in our decision model for selecting software packages. In our research, we meticulously selected 31 packages from the Python Package Index (PYPI)[1] for their applicability in sentiment analysis tasks. For each package, we gathered key data, including the package name, description, about, and URL. This comprehensive data collection aims to facilitate an in-depth analysis of each package's suitability for sentiment analysis.

5.1 Selecting Packages

Due to the extensive range of packages accessible on PyPI, a considerable portion had to be eliminated from our study. The reasoning behind this was manifold:

- **Irrelevance to Sentiment Analysis:** Many packages, while useful in their own right, did not provide the specific functionalities required for sentiment analysis and were thus excluded.
- **Quality Concerns:** Packages that had issues with reliability, such as infrequent updates or poor community feedback, were not considered as they could compromise the effectiveness of sentiment analysis.

[1] pypi.org

- **Lack of Documentation:** Packages lacking sufficient documentation or community support pose a challenge in terms of integration and troubleshooting. Such packages were not included to ensure a smooth research process.

- **Overlap in Functionality:** In cases where multiple packages offered similar functionalities, we selected the ones that excelled in other criteria like performance, support, and reliability to avoid redundancy.

The selection procedure was guided by the objective of identifying the most suitable, reliable, and efficient packages for sentiment analysis. The criteria were meticulously formulated to evaluate and choose the instruments that would not only meet the specific requirements of our study but also maintain the standards of excellence and effectiveness necessary for reliable sentiment analysis.

5.2 Feature selection

To enhance our data-driven decision model for software package selection, it is crucial to prepare the features that will serve as the foundation for our study. This sub-section explores the application of Schema.org, a collaborative and community-driven method for organizing data on the Internet. Schema.org [2] greatly assists in finding and developing these aspects.

5.2.1 Feature Identification

Schema.org provides a shared vocabulary that webmasters can use to mark up their pages in ways recognized by major search providers, enhancing the discoverability and understanding of web content. In the context of software package selection, Schema.org offers a standardized set of schemas that are instrumental in identifying key features of software packages. These features might include, but are not limited to, aspects like software version, release notes, user ratings, and update frequency. For this project, we incorporated the most commonly utilized characteristics that exhibit intersections, as identified in our literature review.

5.2.2 Feature Extraction Steps

Identifying Relevant Schemas:

The initial phase of our process entails identifying the specific schemas within Schema.org that align with the characteristics of software packages. This entails a comprehensive examination of the Schema.org vocabulary to choose schemas that most accurately depict the essential data points for our model. We used the SoftwareApplication[3] schema in this project.

Mapping Features to Schemas:

After identifying the pertinent schemas, the next step involves aligning them with the particular characteristics we aim to extract. The 'SoftwareApplication' schema in Schema.org enables the extraction of attributes such as software requirements, name, and URL.

[2] https://schema.org
[3] https://schema.org/SoftwareApplication

Property	Expected Type	Description
Abstract	Text	An abstract is a short description that summarizes a CreativeWork.
dateModified	DateTime	The date on which the creative work was most recently modified or when the item's entry was modified within a DataFeed.
Description	Text	A description of the item.
Name	Text	The name of the item.
softwareVersion	Text	Version of the software instance.
Url	URL	URL of the item.

Table 5.1: Package Criteria

Property	Description
Functional Suitability	Assess if the software provides and supports all its intended functionalities effectively under specified conditions
Performance Efficiency	Measures the software's performance in terms of resource usage, processing speed, and scalability under defined conditions.
Compatibility	Evaluates the software's ability to coexist and interact efficiently with other systems and products.
Usability	This characteristic indicates the extent to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.
Reliability	Indicates the software's capability to perform its required functions under specified conditions for a specified period of time.
Security	Focuses on the software's ability to protect data and maintain confidentiality, integrity, and availability against unauthorized access or attacks.
Maintainability	Assesses the ease with which the software can be modified, updated, and maintained over time.
Portability	Examines the ease with which the software can be transferred and adapted to different hardware or software environments.

Table 5.2: Quality Attributes

Feature Consolidation and Structuring:

The last stage entails merging the retrieved characteristics into a well-organized format that can be effortlessly incorporated into our decision-making framework. This organized data serves as the foundation of our analysis and forecasts related to software package selection. We generated a configuration file for the web scraping process. The data is structured in the JSON format.

Quality Attributes:

We employed Liang Feng's work, which utilized ISO 2510, as a reference for our project. We also applied ISO 2510 to evaluate the chosen software package candidates for our project.

Following these procedures, we choose the package criteria and quality attribute, as indicated in the 5.1 and 5.2, respectively.

5.3 Web Scraping

During this phase, we utilized the selenium package in Python. Selenium is an open-source automation tool mostly utilized for automating web applications for testing purposes. However, it is also widely employed in web scraping due to its robust capabilities for managing dynamic web material. This section examines the application of Selenium in the web scraping procedure for the data-driven decision model for selecting software packages. Selenium is notable for its capacity to emulate human-like interactions with web sites, such as clicking buttons, completing forms, and moving across web pages. This is especially advantageous for extracting data from websites that largely depend on JavaScript and AJAX-based dynamic content. Another benefit is its ability to be used with many web browsers and operating systems, guaranteeing extensive interoperability.

5.3.1 Data Extraction Steps

Configuration File

We have generated a nested JSON file that encompasses the previously extracted features, as previously elucidated. Each feature comprises various keys to identify the specific element that is associated with the feature.
Below, we present the structure of the nested JSON file.

```
{
<feature> :{
 "parent":
          {
              "id":"",
              "cssClass":"",
              "tag":"",
              "keywords":"",
              "searchPattern":""
          }
        ,
        "id":"",
        "cssClass":"",
        "tag":"",
        "xpath":"",
        "keywords":"",
        "searchPattern":""
}
```

In the following, we describe each key in this file:

- **feature:** It refers to the selected feature in the previous part.
- **parent:** This pertains to the ancestor of the web element, and it possesses many keys.
- **id:** If an element has an ID, it is related to it.
- **cssClass**: Displays the CSS class associated with the element.
- **tag:** It indicates which tag contains the element.

- **xpath:** It provides a flexible and precise means to locate elements, attributes, and text within web documents, making it an essential tool for efficiently retrieving specific information in automated web scraping tasks.
- **keyword:** If there is a specific keyword associated with an element on a web page, such as "license" or "url," we can locate the element using that keyword, and we can use NLP to process the text and find the related information.
- **searchPattern:** The presence of a pattern in the collected data necessitates the implementation of a strategy to accurately locate the desired information after retrieving it from the webpage.

Using Selenium

For this project, we employed Selenium in Python to perform efficient web scraping. Specifically, we used it to extract items and their corresponding information from different web sites. The method commences with configuring the environment, which entails installing Selenium within the Python environment and configuring a WebDriver, such as ChromeDriver for Google Chrome.
After the data is extracted, it undergoes a process of cleaning and formatting to ensure uniformity and precision. The data is organized in a JSON format, which facilitates subsequent analysis. The browser session is appropriately terminated after the scraping operation using the driver.
By employing a complete methodology, we utilize the functionalities of Selenium in Python to extract and gather significant data. This process is integral to this project and guarantees a strong dataset for our software package selection model.

5.3.2 Text analysis

Named Entity Recognition (NER) is used in a data extraction method that has a series of important steps that are designed to make sure that information is correctly recovered from text descriptions of software products. The method commences by choosing a suitable NER tool or library. SpaCy, a potent and flexible NER tool, has been selected for this particular project because of its strength, user-friendliness, and ability to meet the specific requirements of the dataset.

After choosing SpaCy as the NER tool, the next crucial step is to preprocess the text data. This stage is crucial in preparing the text for efficient NER analysis. The preprocessing processes commonly included in SpaCy typically encompass:

1. **Tokenization:**The process of dividing the text into separate words or tokens. SpaCy's tokenizer effectively manages this task, offering a strong basis for subsequent research.
2. **Part-of-Speech (POS) Tagging:**SpaCy applies grammatical labels to each token, such as nouns, verbs, adjectives, etc., which is crucial for comprehending the contextual application of words
3. **Syntactic Parsing:**SpaCy is highly proficient in analyzing the grammatical structures of sentences to detect links between words, hence facilitating comprehension of the syntactic structure of the text.

After appropriately preprocessing the text data, SpaCy's NER capabilities are utilized to analyze the text, detecting and extracting important entities and attributes. In the context

of software package descriptions, these items typically encompass the names of software packages, programming languages, dependencies, functionality, and other essential properties. SpaCy's NER algorithm not only detects these things but also categorizes them into predetermined classes, thereby improving the organization and analysis of the data.
The process does not conclude with entity extraction. Additionally, the post-processing processes are essential, which encompass:

- **Entity Normalization:** SpaCy facilitates the resolution of diverse variations of an entity name to a standardized form, hence ensuring consistency of data.

- **contextual Analysis:** Utilizing SpaCy's sophisticated linguistic features to assess the entities in the wider context of the text, establishing their pertinence and importance.

Subsequently, the data extracted using SpaCy's NER capabilities is formatted into JSON for seamless integration into databases or for further research, such as examining patterns in software development or understanding interdependencies in software ecosystems. By leveraging SpaCy's NER capabilities, this comprehensive approach not only improves understanding of software packages but also enables informed decision-making and intelligent research in the field of software engineering.
This chapter presents a resilient search model that aims to optimize the software package selection process. The model relies on four fundamental technologies: Elasticsearch, the BM25, and IB algorithms. This section delves into the reasoning behind their selection, their integration into our system, and the general aims of the model.

5.4 Implementation

5.4.1 Elasticsearch

Elasticsearch is an open-source, distributed search and analytics engine renowned for its ability to handle large volumes of data with speed and precision. At its core, Elasticsearch is built on top of the Apache Lucene library, which provides advanced search capabilities. Elasticsearch extends these capabilities, offering a scalable search solution that can manage petabytes of structured and unstructured data. Elasticsearch's most prominent attribute is its robust full-text search functionality. It effectively manages intricate search queries and provides rapid search replies, making it well-suited for contexts where swift data retrieval is crucial. Elasticsearch functions inside a distributed framework, wherein the data is disseminated and stored across multiple nodes. This guarantees a high level of availability and durability, as well as the capacity to expand horizontally as the amount of data increases. Real-time data and analytics enable the instantaneous examination of data. Once a document is indexed, it becomes searchable, allowing for instant data retrieval and analysis. Elasticsearch has the ability to index a diverse range of data types, including text, numerical data, and sophisticated data structures. The RESTful API offers a comprehensive and standardized way to communicate with the search engine using conventional HTTP protocols.

5.4.2 Elasticsearch in Software Package Selection

Regarding software package selection, Elasticsearch possesses certain significant benefits that render it very suitable for managing and searching extensive collections of software packages.

Dealing with Large Data Volumes

Software package repositories may hold extensive quantities of data, including diverse properties such as package names, descriptions, dependencies, and version histories. Elasticsearch excels at managing extensive datasets, guaranteeing comprehensive searchability of the entire repository. The distributed design of the system enables it to effectively handle larger amounts of data while ensuring consistent performance and reliability.

Efficient Searching and Filtering

Elasticsearch's proficiency in executing sophisticated full-text searches is essential in the field of software package selection. Users frequently require the ability to conduct searches using precise keywords, phrases, or even more intricate queries. Elasticsearch offers both accurate search results and advanced filtering capabilities. Consequently, users have the ability to enhance their searches by considering other characteristics such as the programming language, dependencies, or current updates, resulting in a more effective and focused search process.

Velocity and Pertinence

The velocity of Elasticsearch is a noteworthy factor. It provides search results instantly, which is crucial in situations when developers or researchers are repeatedly testing different queries to locate the appropriate software package. Moreover, the system's relevance score algorithm guarantees that the most relevant packages are given better rankings in the search results, assisting consumers in promptly making well-informed judgments.

Customizability and Integration

Elasticsearch provides a significant level of customizability, allowing for the precise adaptation of the search engine to meet the specific requirements of software package selection. It has the capability to be combined with different tools and platforms, allowing for a smooth and uninterrupted process where users may search for packages directly within their development environment.
In conclusion, Elasticsearch's robust architecture, efficient data handling, advanced search features, and capacity to process massive volumes of data establish it as a crucial tool in the software package selection space. Its application in this domain significantly increases users' ability to traverse the wide and complex world of software packages, hence augmenting the general efficacy and efficiency of the software development process.

5.4.3 BM25 Algorithm

BM25 is a contemporary version of the TF-IDF (Term Frequency-Inverse Document Frequency) algorithm, which is employed by search engines as a ranking function. It evolved from the probabilistic information retrieval concept and has gained widespread acceptance due to its efficacy in ranking items according to their relevance to a given query.
BM25 is a software package of the Okapi BM (Best Matching) algorithm family, which was created in the late 1980s and early 1990s at City University, London, as a part of the Okapi information retrieval project. Its purpose was to enhance the conventional TF-IDF method by effectively managing the fluctuation in phrase frequency among documents.
The BM25 algorithm calculates the score of a document d for a given query q using the formula:

$$\text{Score}(d,q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i,d)\cdot(k_1+1)}{f(q_i,d)+k_1\cdot(1-b+b\cdot\frac{|d|}{\text{avgdl}})}$$

Where:

- $f(q_i, d)$ is the term frequency of query term q_i in the document d.
- $|d|$ is the length of the document d in words.
- avgdl is the average document length in the text collection.
- k_1 and b are free parameters, usually chosen, in absence of advanced optimization, as $k_1 = 1.2$ and $b = 0.75$.
- $IDF(q_i)$ is the Inverse Document Frequency of the query term q_i, which can be calculated as $\log \frac{N-n(q_i)+0.5}{n(q_i)+0.5}$, where N is the total number of documents and $n(q_i)$ is the number of documents containing q_i.

BM25 is crucial in our search model as it enhances Elasticsearch's capability to determine the best suitable software packages for a particular query. The incorporation of BM25 enhances Elasticsearch by offering a sophisticated method for evaluating relevance scores.

5.4.4 LM Dirichlet Similarity in Language Modeling

The LM Dirichlet similarity, a significant component in language modeling for information retrieval, leverages the Dirichlet distribution for document modeling, enhancing retrieval and text analysis performance. The core formula of the Dirichlet Language Model is

$$P(w|D) = \frac{c(w,D)+\mu P(w|C)}{|D|+\mu} \tag{5.1}$$

where $P(w|D)$ represents the probability of a word w in a document D, $c(w, D)$ is the count of w in D, $P(w|C)$ is the probability of w in the collection C, $|D|$ is the length of the document, and μ is the Dirichlet prior, a smoothing parameter. Studies have demonstrated its effectiveness in various contexts, such as integrating category-specific information into language models, which significantly improves retrieval performance [52]. Additionally, Latent Dirichlet Allocation (LDA) has been employed for creating topic space models in text summarization and cross-language text similarity calculations, showcasing its versatility and efficiency in handling complex language data [53]. This highlights the LM Dirichlet similarity's crucial role in enhancing text analysis and information retrieval systems.

5.4.5 LM Jelinek-Mercer Similarity in Information Retrieval

Language modeling in information retrieval, particularly the LM Jelinek-Mercer similarity, plays a crucial role in enhancing retrieval performance. This approach involves a smoothing technique where the likelihood of observing a term in a document is combined with the likelihood of the term in the collection. The Jelinek-Mercer formula is given as:

$$P(t|D) = \lambda P(t|D) + (1-\lambda)P(t|C) \tag{5.2}$$

where $P(t|D)$ is the probability of the term in the document, $P(t|C)$ is the probability of the term in the collection, and λ is a smoothing parameter between 0 and 1. Research has shown that adjusting for document length is vital, with Dirichlet prior smoothing handling it more effectively than Jelinek-Mercer smoothing. However, incorporating a length-based prior can significantly improve Jelinek-Mercer's performance [54]. Further, the probabilistic document length prior, when combined with Jelinek-Mercer smoothing, surpasses the performance of length-dependent smoothing components [55]. These insights highlight the method's adaptability and efficiency in various applications, including heart disease prediction systems [56], making it a valuable tool in the field of information retrieval.

5.4.6 Information-based (IB)

Information-based similarity is a method used to quantify the similarity between data sets, objects, or signals based on the information content they share. This concept is applied in various fields, such as bioinformatics, medical imaging, and data analysis. For example, in medical image analysis, similarity measures like EMPCA-MI improve robustness and computational efficiency in image registration [57]. Another application is in genetic sequence analysis, where alignment-free approaches measure similarity among sequences using information theory [58].

A common formula in information-based similarity is the Mutual Information (MI) measure, which quantifies the amount of information shared between two random variables. It is defined as:

$$MI(X;Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

where $p(x,y)$ is the joint probability distribution function of X and Y, and $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y, respectively. This formula is widely used in determining the similarity of datasets, particularly in fields that require the analysis of complex, multi-dimensional data.

Relevance Scoring Mechanism

The similarity models, built upon the aforementioned concepts, enhance the relevance score of each document (software package) by taking into account the frequency of query terms it contains, as well as the document's length and the distinctiveness of the query terms. This strategy provides a more advanced and efficient ranking of search results in comparison to basic frequency-based algorithms. When choosing software packages, the relevance score that similarity models produce is of utmost importance. The ranking algorithms prioritize software packages that closely align with the user's inquiry, especially those employing distinct technical words or necessitating a profound comprehension of the software subject.

Expanding on Elasticsearch

Although Elasticsearch effectively handles and queries extensive datasets, BM25 further enhances this functionality by guaranteeing that the search outcomes are not only rapid but also pertinent. The integration of Elasticsearch's advanced search capabilities with the similarity algorithms' accurate relevance score results in a resilient and efficient system for software package selection, specifically designed to meet the intricate requirements of developers and researchers in the field.

5.5 Integration of the Similarity Models and Elasticsearch

The combination of Elasticsearch with our similarity model creates a powerful synergy for managing and retrieving software package data. This section delves into the integration of these components, focusing on data indexing, query processing, and the determination of search result relevancy and ranking.

5.5.1 Query processing

When a query is submitted, the model processes it as follows:

1. **Query Reception:** The model receives a user's search query, which may range from simple keywords to complex phrases.

2. **Query Analysis:** NER analyzes the query, breaking it down into terms and phrases and understanding the context.

3. **Applying the Similarity Models:** The models calculate a relevance score for each document based on term frequency, document length, and term specificity.

4. **Retrieving Results:** Elasticsearch fetches the documents matching the query, with relevance scores determined by the similarity model.

5.5.2 Ensuring Relevance and Effective Ranking

The integration of Elasticsearch with our similarity model is critical for ensuring that search results are relevant and properly ranked.

Ensuring Relevance and Effective Ranking: The integration of Elasticsearch with our similarity model plays a crucial role in ensuring that the search results are both relevant and appropriately ranked.

- **Relevance Scoring:** The model's scoring mechanism is a key aspect of its functionality. It assesses not just the presence of query terms in documents but also their significance and distribution. This approach leads to higher scores for documents that are more relevant to the query.

- **Ranking Documents:** Documents are ranked based on their scores in the similarity model. A higher score indicates greater relevance to the query, and thus, such documents are placed higher in the search results.

- **Fine-Tuning Results:** Additionally, the model incorporates factors like user preferences, historical data, and contextual information to further refine the ranking. This ensures that the most pertinent software packages are presented to the user.

To summarize, the combination of Elasticsearch and this advanced similarity model in our approach demonstrates the harmonious relationship between smart text indexing and intelligent relevance scoring. This combination not only improves the effectiveness of the search process but also guarantees that the results are accurate, pertinent, and prioritized according to the user's requirements. This integration greatly simplifies the process of selecting software packages, making it an essential tool for developers and researchers in the field of software engineering.

Chapter 6

Experimentation

This part provides an academic investigation into the evaluation of a recommendation system for software packages, leveraging the functionalities of ChatGPT. Based on recent academic research, we explore approaches that include extensive language models in software testing and development. This improves the effectiveness and dependability of recommendation systems in the software engineering domain.
The incorporation of AI, specifically expansive language models such as ChatGPT, into the process of software development and testing signifies a notable progression in the domain. This work explores the successful utilization of such models in testing a software package recommendation system, drawing on the knowledge gained from recent research.
Current research in the field of software engineering emphasizes diverse, cutting-edge approaches and methodologies that are transforming the domain of software testing and evaluation. These encompass cutting-edge methodologies for evaluating the quality of tertiary studies [59], employing fuzz testing in Model-Driven Software Engineering (MDSE) [60], mitigating challenges in measurement techniques [61], and implementing search-based software testing for assessing Deep Neural Network (DNN) quantization [62]. Each of these studies offers useful insights regarding the utilization of AI tools, such as ChatGPT, to improve the effectiveness and efficiency of software application testing.
We employed ChatGPT as a proficient expert specializing in the domain of software development. We devised a methodology and formulated questions, which we then employed to carry out an interview with ChatGPT. Consequently, we successfully collected significant information and enhanced the efficiency of our recommendation system.

6.1 Experimental Setup

6.1.1 Objective

The main aim of this experiment was to assess the effectiveness of the built software package recommendation system compared to a widely recognized AI model, ChatGPT, in suggesting appropriate software packages based on provided keywords. Three similarity models that were implemented in Elasticsearch were tested against ChatGPT. The models are BM25, LM Jelinek-Mercer, LM Dirichlet and IB.

6.1.2 Methodology

This methodology describes a detailed process used to compare the performance of a newly developed software package recommendation system with that of ChatGPT. The approach centers on analyzing both systems' responses to specific keywords related to software packages. This comparison helps in determining how closely the developed application aligns with ChatGPT's recommendations, which are considered a golden set or standard in this context.

1. **Keyword Input**
   - *Process:* A predefined list of keywords associated with various software packages is created. These keywords are carefully chosen to cover a broad range of software package types, ensuring a comprehensive evaluation.
   - *Simultaneous Input:* Each keyword is then simultaneously entered into both ChatGPT and the software recommendation system. This simultaneous input ensures that both systems are responding to the same query under similar conditions, providing a fair basis for comparison.
   - *Documentation:* The keywords are documented alongside the responses they elicit for transparency and to facilitate a thorough analysis later.
2. **Response Compilation**
   - *Collection:* Once the keywords are inputted, the responses from both ChatGPT and the developed application are collected. Special attention is paid to ensure that the responses are accurately captured as they are.
   - *JSON Structuring:* The collected data is then structured into a JSON (JavaScript Object Notation) format. This format is chosen for its readability and ease of use in data processing. The structure includes:
     - `"keyword"`: The keyword that was inputted.
     - `"expected_value"`: The response was provided by ChatGPT. This is considered the expected or ideal recommendation.
     - `"suggested_value"`: The recommendation was made by the developed application.
   - *Standardization:* This standardized format helps in maintaining consistency across all data points, which is crucial for an unbiased analysis.
3. **Data Analysis**
   - *Metrics Calculation:* The responses in the JSON file are analyzed to calculate the recommendation system's accuracy, precision, and recall. These metrics provide insight into various aspects of the system's performance.
     - Accuracy measures the proportion of total correct recommendations (both true positives and true negatives), and it is calculated as:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Recommendations}}$$

     - Precision assesses the correctness of the positive recommendations made by the system and is calculated as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

     - Recall evaluates how well the system identifies all relevant recommendations and is calculated as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- *Comparison with Golden Set:* ChatGPT's responses (the expected values) serve as a benchmark. The system's responses are compared against these to determine how often it agrees with ChatGPT's recommendations (indicating accuracy), how relevant its suggestions are (precision), and how many relevant suggestions it captures (recall).
- *Insights and Implications:* The analysis provides insights into the effectiveness of the recommendation system. It helps identify areas where the system excels and where improvements are needed. For instance, a high precision but lower recall might indicate the system's recommendations are generally on point, but it might be missing out on some relevant suggestions.

In summary, this methodology offers a structured and systematic approach to evaluating the performance of a software package recommendation system. By comparing it against ChatGPT's recommendations, the method provides a comprehensive assessment of the system's capabilities for accurately and effectively suggesting software packages based on user queries.

6.2 Creation of the Golden Set

6.2.1 Role of ChatGPT

ChatGPT, developed by OpenAI, is renowned for its advanced natural language processing capabilities. It stands out due to its ability to understand and generate human-like text responses. In this context, ChatGPT's proficiency is leveraged as a benchmark for evaluating the software package recommendation system. By using ChatGPT's responses to specific keywords as a standard, we can establish a reliable set of expected outcomes. These outcomes then serve as a comparative measure to assess the performance of the developed application. The rationale behind using ChatGPT as a benchmark stems from its widespread recognition for accuracy and contextual relevance in generating responses, making it a robust model for setting expected standards in this experiment.

6.2.2 Data Collection

The creation of the golden set, an essential part of the experiment, involved a meticulous process. This process began with the selection of 31 diverse and representative keywords related to different software packages. These keywords were chosen to encompass a wide range of potential user queries, ensuring comprehensive coverage of the software domain. Following this, each keyword was inputted into ChatGPT, and the responses were carefully recorded. These responses from ChatGPT were then compiled to form the golden set. This golden set represents a diverse array of expected outcomes, providing a comprehensive basis for evaluating the recommendations made by the software package recommendation system. By comparing the system's suggestions against this golden set, we aimed to quantitatively assess the system's accuracy, precision, and recall, thereby gauging its effectiveness in providing relevant software package recommendations.

6.3 Results and Discussion

6.3.1 Comparative Analysis of Model Performances

The detailed performance of each model, as presented in Table 6.1, reveals significant insights into their respective strengths and weaknesses. The BM25 model demonstrates a balanced

performance with equal counts of False Positives (FP) and False Negatives (FN), suggesting a consistent approach in differentiating between positive and negative classifications. In contrast, the IB model, with the highest True Positives (TP), indicates a propensity towards positive classification, albeit at the expense of a higher FP count. This could imply a potential bias towards overestimating positive outcomes.
The LM Jelinek-Mercer model exhibits a conservative tendency, with higher True Negatives (TN) and lower FPs. This cautious approach, however, leads to a higher count of FNs, indicative of missed positive classifications. The LM Dirichlet model presents an intriguing mix of high TP and low FP, but its comparatively higher FN count suggests a potential shortfall in identifying positive instances accurately.

Model	**TP**	**TN**	**FP**	**FN**
BM25	30	10	5	5
IB	35	5	7	3
LM Jelinek-Mercer	28	12	8	2
LM Dirichlet	33	6	4	7

Table 6.1: Detailed Comparison of Model Performance

6.3.2 Accuracy, Precision, and Recall Evaluation

As delineated in Table 6.2, the models, barring LM Dirichlet, predominantly achieve an 80% accuracy rate, underscoring a general high level of correctness in their predictions. The precision and recall metrics, however, offer deeper insights into each model's efficiency. The BM25 model's balanced FP and FN distribution helps it achieve impressive precision and recall, demonstrating how well it can find true positives while reducing false alarms.
The IB model, despite its high TP rate, records marginally lower precision but the highest recall amongst the models. This reflects its strength in identifying positive cases, albeit with an increased likelihood of false positives. Conversely, the LM Jelinek-Mercer model, with the lowest precision, compensates with a high recall rate. This pattern is indicative of a cautious model that minimizes false positives but at the risk of increasing false negatives. Lastly, the LM Dirichlet model, despite its lower accuracy, achieves the highest precision, albeit with a compromised recall, suggesting a tendency to overlook positive cases.

Model	**Accuracy**	**Precision**	**Recall**
BM25	80%	85.71%	85.71%
IB	80%	83.33%	92.11%
LM Jelinek-Mercer	80%	77.78%	93.33%
LM Dirichlet	78%	89.19%	82.5%

Table 6.2: Accuracy, Precision, and Recall Analysis

In conclusion, the analysis elucidates the varied performance dynamics of each model. The BM25 model emerges as a balanced choice, the IB model as a recall-oriented option, the

LM Jelinek-Mercer model prioritizes minimizing false positives, and the LM Dirichlet model excels in precision at the recall's expense. These findings are pivotal for model selection, contingent upon specific application requirements prioritizing either precision or recall.

Chapter 7

Discussion

7.1 Validity

7.1.1 Internal Validity: Challenges and Considerations

This research carefully checks the internal validity, paying special attention to how representative ChatGPT's training data is and how the keywords used to test the software package recommendation system were chosen. The training data for ChatGPT plays a crucial role since it directly impacts the AI model's replies and, consequently, sets the standard for the recommendation system. Inadequate diversity or representativeness of the data inputted into ChatGPT can result in biased outcomes. The presence of this potential bias presents a substantial risk to the study's validity, as the suggestions provided by ChatGPT may not correlate accurately with genuine user needs or industry norms [63].
Moreover, the choice of keywords included in the experiment is essential in deciding the results of the investigation. To achieve a thorough examination, it is important to select keywords that cover a wide variety of software package types and user inquiries. Any prejudices in the process of choosing keywords, whether accidental or organized, could result in a distortion of the recommendation system's efficacy. If the chosen keywords are too specific or fail to sufficiently encompass the range of user inquiries, the experiment may exhibit a bias towards certain types of recommendations, thus distorting the outcomes. To improve the internal validity of the study, it is essential to address these problems. This will ensure that the findings are dependable and accurately represent the system's actual capabilities in a real-world setting [64].

7.1.2 External Validity: Generalizability of Findings

The external validity of this work hinges on the generalizability of its findings to other AI models and their use in diverse contexts of software recommendation systems. Introducing ChatGPT as a benchmark in this research brings forth a unique AI model with distinct training and response characteristics. This characteristic raises questions about the potential consequences if different AI models with diverse training datasets, methodologies, or processing capabilities were used. For example, an AI model that has been trained on a dataset that focuses on a certain software domain or has unique language processing abilities could generate significantly different recommendation outcomes. Therefore, the findings derived from this study, while robust within the defined parameters, may not be easily applicable or provide similar results when replicated with a different AI model [65].
Additionally, it is crucial to assess the applicability of the study's results in different software recommendation scenarios, as this is a significant factor in establishing external validity.

Software recommendation systems vary greatly in terms of their target user demographic, the nature of the software being recommended (such as commercial, open-source, or enterprise-level), and the complexity of user requirements. The study that used ChatGPT only looked at a few criteria and conditions, which might not cover all the different situations that can happen in the bigger field of software suggestions. The efficacy, precision, and pertinence of the suggestions examined in this study may differ when implemented in other contexts. This constraint underscores the significance of thoroughly scrutinizing the study's conclusions and suggests a potential area for future investigation to examine the relevance of these outcomes to various AI models and recommendation scenarios [66].

7.1.3 Validity of Construction

The construct validity of the study can be evaluated by examining its reliance on quantitative criteria such as accuracy, precision, and recall to measure the performance of software package recommendation systems. While these measurements are customary and beneficial for assessing particular aspects of system performance, they may not comprehensively represent the intricate and nuanced nature of software recommendations. The efficacy of a product or service in practical scenarios is frequently assessed based on criteria such as user satisfaction, contextual appropriateness, and the ability to accommodate diverse user requirements. The measurements alone may not comprehensively capture these features. Therefore, focusing just on these specific metrics in the study may lead to an incomplete understanding of the true effectiveness of the system and the user's experience, ultimately impacting the research's validity. This highlights the need to include more extensive and user-oriented evaluation criteria in future research to provide a more thorough comprehension of the system's performance and efficacy [67].

7.1.4 Dependence on External Data Sources

The pipeline's reliance on data obtained from external platforms such as PYPI and various package managers poses a substantial vulnerability in terms of the reliability and legitimacy of both the data and its sources. These platforms exhibit a notable level of flexibility, always evolving in terms of their data architecture, rules, and the precision of the data they possess. For instance, changes in data formats or categorization methods on these platforms can directly impact the pipeline's ability to accurately obtain and manage the required information. The efficacy and precision of the pipeline are heavily contingent upon external events, underscoring its significant dependence on these factors. The functionality of the pipeline could be immediately compromised if there are any substantial modifications or issues, such as data corruption or periods of inactivity, in these systems. The presence of this correlation underscores the need for robust systems inside the pipeline to adapt to external fluctuations and maintain their effectiveness [68].

7.1.5 Effect on Pipeline Validity

The potential alterations to external data sources pose a substantial threat to the integrity of the pipeline. The effectiveness of the pipeline depends on the assumption that these external platforms will constantly maintain and provide high-quality data, an assumption that may not always be true. An alteration in the legislation of data accessibility or user privacy on these platforms may impede the pipeline's capacity to acquire crucial data, thereby impeding its performance. Moreover, if the data obtained from these sources declines in quality due to errors, outdated information, or changes in the hosted software packages, it may lead to the production of inaccurate or irrelevant suggestions by the pipeline. In order to address these issues, it is imperative to consistently monitor and revise the pipeline in accordance with fluctuations in external data sources. Furthermore, it would be advantageous to expand the array of data sources in order to mitigate the hazards linked to relying exclusively on a single platform.

7.2 Finding during the project

7.2.1 AI Integration in Software Recommendations: Potential and Limitations

The research conducted in this study significantly underscores the potential of AI in enhancing software recommendation systems while also delineating its limitations. AI, particularly advanced language models like ChatGPT, has shown a remarkable capability to understand and process user queries, offering recommendations that are often contextually relevant and informed. This ability not only speeds up the process of finding suitable software solutions but also introduces a level of precision that traditional, non-AI systems may struggle to achieve. However, the study also brings to light the limitations inherent in AI-driven recommendations. AI models, by their very nature, are dependent on the data they have been trained on, which can lead to biases or gaps in recommendations if the training data is not comprehensive or up-to-date. Moreover, AI systems may lack the nuanced understanding that comes from human expertise, particularly in complex or niche areas where contextual subtleties play a significant role.

7.2.2 The Need for a Balanced AI-Human Approach

In light of these findings, the research highlights the necessity of a balanced approach where AI complements rather than replaces human decision-making in software recommendations. The integration of AI offers significant advantages in terms of efficiency and data processing capabilities, but it should be viewed as a tool that assists human experts rather than a standalone solution. Human oversight is crucial in interpreting AI recommendations, providing contextual insights, and making final decisions, especially in cases where the AI's suggestions may not fully align with specific user needs or preferences. This synergistic approach leverages the strengths of both AI and human expertise, leading to more robust, accurate, and user-centered software recommendation systems. It emphasizes the importance of human judgment in areas where AI may lack depth, such as understanding unique user scenarios, ethical considerations, and complex decision-making processes

7.2.3 Importance of Model Selection in Software Recommendations

The study effectively demonstrates the crucial significance of model selection in the field of software recommendations. Various models, including BM25, LM Dirichlet, and others examined in this study, possess unique attributes and are appropriate for diverse requirements and circumstances. For example, certain models may demonstrate exceptional accuracy by providing suggestions that are highly correct. However, this precision may come at the expense of recall, resulting in the possibility of missing out on certain pertinent possibilities. Some individuals may prefer the ability to remember information, which would result in a wider variety of ideas while potentially sacrificing accuracy. This discrepancy emphasizes the significance of considering context when choosing a model for software recommendations. The effectiveness of a model is not solely determined by its technical superiority but also by how well it corresponds to the particular objectives and limitations of the recommendation system. These factors include the characteristics of the software packages, the variety of users, and the unique scenarios being targeted.

7.2.4 Impacts on Software Engineering

From a software engineering standpoint, the research provides detailed and subtle insights into the incorporation of artificial intelligence into recommendation systems. It promotes the perspective of considering AI as a tool that enhances, rather than substitutes, human skill in the fields of software development and testing. This viewpoint is especially important as it recognizes the indispensable worth of human intuition, expertise, and discernment in making intricate decisions. AI has the ability to analyze and interpret large volumes of data and provide recommendations based on complex patterns and connections that surpass human capabilities. Nevertheless, in intricate or unclear situations, human supervision greatly enhances the accuracy of the ultimate decision. This methodology promotes the utilization of AI by software engineers to capitalize on its advantages, such as managing extensive amounts of data and offering preliminary suggestions. Simultaneously, it acknowledges and addresses the limitations of AI by incorporating human expertise.

7.2.5 Broader Implications of the Findings

The implications of these findings extend beyond the technical elements of model selection and the incorporation of artificial intelligence. They indicate a change in the framework of software engineering techniques, where AI is regarded as a cooperative ally rather than an independent answer. This change requires software engineers to acquire a fresh set of skills and viewpoints. They must now possess not just a thorough comprehension of AI technology but also the ability to skillfully assess and incorporate AI advice into their work processes. Moreover, the report highlights the significance of ongoing education and adjustment in the industry as artificial intelligence technology and models progress. The findings indicate the importance of including ethical considerations and user-centric approaches in the adoption of AI. This ensures that the technology is used to improve the user experience and effectively satisfy the different demands of users.

7.3 Implications and Responsibilities in AI-Enhanced Software Engineering

7.3.1 Ethical and Practical Considerations of AI Systems

The study highlights crucial ethical and practical factors related to the utilization of AI systems such as ChatGPT in the field of software engineering. An urgent concern lies in the inherent biases that can exist throughout AI systems. These biases may arise from imbalanced training data or the AI's algorithms themselves, resulting in suggestions that may lack fairness or fail to represent the different demands of users. These biases can have substantial consequences, particularly when AI systems are employed in crucial decision-making procedures in software engineering. Furthermore, the usefulness of AI recommendations is being examined closely. Although AI has the capability to handle and examine extensive information, it is crucial for its recommendations to be contextually suitable and feasible to apply. This requires a meticulous equilibrium between leveraging AI for its computational capabilities and ensuring that its recommendations are practical and pertinent in real-life situations.

7.3.2 Enhancing Transparency and Accountability in AI Recommendations

Ensuring the openness and accountability of AI systems is crucial, particularly in key domains such as software engineering. Understanding the mechanism via which an AI system, like ChatGPT, produces its recommendations is essential for both users and decision-makers. Transparency refers to both the clarity of the algorithmic processes and the ease of access and understanding of the underlying data and decision-making variables. Accountability, on the other hand, pertains to the responsibility of making decisions in alignment with AI recommendations. With the growing integration of AI systems into critical activities, the issue of accountability for these decisions becomes complex, encompassing the AI developers, the users, and the AI system itself. In order to ensure trust in AI systems and uphold their acceptable and ethical use, it is imperative to take into account these components [69].

7.3.3 Wider Industry Implications of the Research

This research has the potential to significantly influence industry practices regarding the use of artificial intelligence in software development tools and processes. By clarifying the potential and limitations of AI in software recommendations, it offers a platform for software engineers and developers to carefully incorporate AI into their workflows. This includes not only the integration of technology but also the consideration of how AI advice should be evaluated, understood, and put into practice. The results of this study could enhance software development processes by optimizing the use of AI to increase efficiency and stimulate innovation [70].

7.3.4 Educational Significance for Prospective Software Engineers and AI Researchers

Moreover, the paper holds substantial instructional merit for aspiring software engineers and AI researchers. It serves as a case study that showcases the actual application of AI, offering a good comprehension of the difficulties associated with managing AI systems in a real-world setting. Software developers should give priority to enhancing their proficiency in AI literacy, ethical decision-making, and critical analysis of AI recommendations. The study highlights the importance for AI researchers to create AI systems that have both technological expertise and ethical integrity, as well as being user-friendly. Hence, this study can offer significant perspectives for developing educational plans and instructional courses that will equip upcoming experts with the essential competencies to effectively employ artificial intelligence in software engineering while also ensuring their ability to responsibly address its challenges [71].

Chapter 8

Conclusion and Future Work

8.1 Reiteration of Core Thesis and Major Contributions

In this last chapter, we explore the main issue of the thesis, specifically examining the creation of a data-driven decision model for choosing software packages. The research focuses on a crucial part of software engineering: the choice of suitable software components, a decision that has a substantial impact on the success and efficiency of software projects.

The thesis's distinct contributions are diverse. It presents automated techniques for pulling unorganized data from repositories, therefore revolutionizing data collection for software package analysis. The study also introduces innovative techniques for evaluating software package compatibility, providing a more nuanced method for program selection. An important achievement is the creation of a hybrid recommendation system that combines content-based, collaborative, and knowledge-based approaches. This comprehensive approach is a notable improvement over current procedures as it successfully navigates the intricacies of software package selection. The research concludes with a comprehensive assessment of this system, showcasing its effectiveness and potential for practical use in real-world software development.

This thesis makes a substantial contribution to the subject of component-based software engineering (CBSE) by creating an intelligent recommendation system. The system offers valuable tools and approaches for software engineers and industry practitioners. The techniques suggested in this study have the capacity to greatly improve the efficacy, affordability, and excellence of software development, particularly in the realm of COTS software. These contributions not only enhance the current level of expertise in software engineering methods but also create new opportunities for future study and development in the subject.

8.2 Future Work

Future studies could delve deeper into refining data extraction techniques from software repositories. This includes exploring advanced data mining and natural language processing methods to enhance accuracy and comprehensiveness. Researching scalable architectures and efficient algorithms is critical for handling larger datasets. Investigating the model's adaptability to emerging software technologies and practices, such as cloud-based development and DevOps methodologies, would also be beneficial.
Integrating advanced AI techniques, particularly in the realm of machine learning and deep learning, could significantly improve the sophistication and accuracy of the recommendation system. Additionally, a thorough investigation into user interaction with the system, through

empirical studies or real-world application feedback, would provide invaluable insights. This user-centric approach could lead to a more intuitive and responsive system, better suited to the diverse needs and preferences of users in software package selection.

8.2.1 Impact and Practical Applications

The research findings have significant potential to impact the real world, particularly in the software development business. The study's findings on knowledge-based recommendation systems for software package selection have the potential to greatly improve the efficiency and accuracy of software package selection. These findings have ramifications for enhancing project results in terms of quality, cost, and time administration. The incorporation of such systems has the potential to completely transform the approach that developers take towards selecting components, perhaps resulting in software solutions that are more resilient and efficient. In addition, the collaboration between academia and industry has the potential to stimulate innovation in software engineering methods by connecting theoretical study with practical implementation.

8.3 Concluding Remarks

This research endeavor signifies a notable advancement in the field of software engineering, specifically in the domain of software component selection. This study involves the examination and application of a knowledge-based recommendation system, which not only enhances academic knowledge but also provides useful tools for the industry. This study highlights the significance of utilizing data-driven decision-making in the field of software development, establishing a precedent for future research in this area. In the future, the consequences of this research go beyond immediate uses, providing a basis for additional investigation and advancement in software package selection and recommendation systems. The incorporation of these systems in real-life situations highlights the significance of the subject, providing a combination of theoretical advancement and practical usefulness to the wider academic and professional community.

All the data gathered in the performed SLR can be found at:
software package recommendation system Metadata
Interview protocol can be found at: Interview protocol
Github: Github link

References

[1] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[2] H. Borges and M. T. Valente. What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129, 2018.

[3] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy. Evolution of software in automated production systems: challenges and research directions. *Journal of Systems and Software*, 110:54–84, 2015.

[4] J. Beheshti and J. Dupuis. Problems with cots software: a case study. *Proceedings of the Annual Conference of Cais / Actes Du Congrès Annuel De L Acsi*, 2013.

[5] S. Kalantari, H. Motameni, E. Akbari, and M. Rabbani. Optimal components selection based on fuzzy-intra coupling density for component-based software systems under build-or-buy scheme. *Complex and Intelligent Systems*, 7:3111–3134, 2021.

[6] G. Bavota, A. D. Lucia, Andrian Marcus, and R. Oliveto. Using structural and semantic measures to improve software modularization. *Empirical Software Engineering*, 18:901–932, 2013.

[7] Tassio Vale, Ivica Crnkovic, Eduardo Santana de Almeida, Paulo Anselmo da Mota Silveira Neto, Yguaratã Cerqueira Cavalcanti, and Silvio Romero de Lemos Meira. Twenty-eight years of component-based software engineering. *Journal of Systems and Software*, 111:128–148, 2016.

[8] R. Garg, R. Sharma, K. Sharma, and R. Garg. Mcdm based evaluation and ranking of commercial off-the-shelf using fuzzy based matrix method. *Decision Science Letters*, pages 117–136, 2017.

[9] M. Ilyas, S. U. Khan, and N. Rashid. Empirical validation of software integration practices in global software development. *SN Computer Science*, 1, 2020.

[10] D. Spinellis. Package management systems. *IEEE Software*, 29:84–86, 2012.

[11] T. Neubauer and C. Stummer. Interactive decision support for multiobjective cots selection. *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007.

[12] Martin Robillard, Robert Walker, and Thomas Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.

[13] Phuong T. Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. Crossrec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software*, 161:110460, 2020.

[14] D. Badampudi, K. Wnuk, C. Wohlin, U. Franke, D. Šmite, and A. Cicchetti. A decision-making process-line for selection of software asset origins and components. *Journal of Systems and Software*, 135:88–104, 2018.

[15] D. Badampudi, C. Wohlin, and K. Petersen. Software component decision-making: in-house, oss, cots or outsourcing - a systematic literature review. *Journal of Systems and Software*, 121:105–124, 2016.

[16] Siamak Farshidi. *Multi-Criteria Decision-Making in Software Production*. PhD thesis, Utrecht University, 2020.

[17] Amarpreet S Arora and Akepati S Reddy. Development of multiple linear regression models for predicting the stormwater quality of urban Sub-Watersheds. *Bulletin of Environmental Contamination and Toxicology*, 92(1):36–43, January 2014.

[18] Nacim Yanes, Sihem Ben Sassi, and Henda Hajjami Ben Ghezala. Ontology-based recommender system for cots components. *Journal of Systems and Software*, 132:283–297, 2017.

[19] A. Rushinek and S. Rushinek. Accounting software evaluation: hardware, audit trails, backup, error recovery and security. *Managerial Auditing Journal*, 10:29–37, 1995.

[20] Ferdian Thung, David Lo, and Julia Lawall. Automated library recommendation. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 182–191, 2013.

[21] Andrea Renika D'Souza, Di Yang, and Cristina V. Lopes. Collective intelligence for smarter api recommendations in python. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 51–60, 2016.

[22] Jing-Zhuan Zhao, Xuan Zhang, Chen Gao, Zhu-Dong Li, and Bao-Lei Wang. KG2Lib: knowledge-graph-based convolutional network for third-party library recommendation. *The Journal of Supercomputing*, 79(1):1–26, January 2023.

[23] Siamak Farshidi and Zhiming Zhao. *An Adaptable Indexing Pipeline for Enriching Meta Information of Datasets from Heterogeneous Repositories*, pages 472–484. Springer International Publishing, 05 2022.

[24] Barbara Kitchenham, Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering-a systematic literature review. *Information and Software Technology*, 51:7–15, 01 2009.

[25] Yu Xiao and Maria Watson. Guidance on conducting a systematic literature review. *Journal of planning education and research*, 39(1):93–112, 2019.

[26] C. Okoli. A guide to conducting a standalone systematic literature review. *Communications of the Association for Information Systems*, 37, 2015.

[27] E. Engström, M. Storey, P. Runeson, M. Höst, and M. Baldassarre. How software engineering research aligns with design science: a review. *Empirical Software Engineering*, 25:2630–2660, 2020.

[28] B. Morschheuser, L. Hassan, K. Werder, and J. Hamari. How to design gamification? a method for engineering gamified software. *Information and Software Technology*, 95:219–237, 2018.

[29] F. Fagerholm, A. Guinea, H. Mäenpää, and J. Münch. The right model for continuous experimentation. *Journal of Systems and Software*, 123:292–305, 2017.

[30] Ala'a N. Alslaity and T. Tran. Goal modeling-based evaluation for personalized recommendation systems. *Adjunct Proceedings of the 29th ACM Conference on User Modeling, Adaptation and Personalization*, 2021.

[31] A. Smirnov and A. Ponomarev. Multicriteria context-driven recommender systems: Model and method. *Scientific and Technical Information Processing*, 47:298 – 303, 2020.

[32] Siamak Farshidi, Slinger Jansen, and Jan Martijn E. M. van der Werf. Capturing software architecture knowledge for pattern-driven design. *CoRR*, abs/2005.08393, 2020.

[33] Adam Kilgarriff, Vít Baisa, Jan Bušta, Miloš Jakubíček, Vojtěch Kovář, Jan Michelfeit, Pavel Rychlý, and Vít Suchomel. The sketch engine: ten years on. *Lexicography*, 1(1):7–36, July 2014.

[34] C. N. Dang, M. N. M. García, and F. D. l. Prieta. An approach to integrating sentiment analysis into recommender systems. *Sensors*, 21:5666, 2021.

[35] Y. Wang, M. Wang, and W. Xu. A sentiment-enhanced hybrid recommender system for movie recommendation: a big data analytics framework. *Wireless Communications and Mobile Computing*, 2018:1–9, 2018.

[36] A. Sulistya, G. A. A. Prana, A. Sharma, D. Lo, and C. Treude. Sieve: helping developers sift wheat from chaff via cross-platform analysis. *Empirical Software Engineering*, 25:996–1030, 2019.

[37] T. M. Abdellatif, L. F. Capretz, and D. Ho. Automatic recall of software lessons learned for software project managers. *Information and Software Technology*, 115:44–57, 2019.

[38] N. Ali, H. Cai, A. Hamou-Lhadj, and J. Hassine. Exploiting parts-of-speech for effective automated requirements traceability. *Information and Software Technology*, 106:126–141, 2019.

[39] Y. Liu, L. Liu, H. Liu, X. Wang, and H. Yang. Mining domain knowledge from app descriptions. *Journal of Systems and Software*, 133:126–144, 2017.

[40] D. Falessi, S. M. Laureani, J. Çarka, M. Esposito, and D. A. d. Costa. Enhancing the defectiveness prediction of methods and classes via jit. *Empirical Software Engineering*, 28, 2023.

[41] S. Beyer, C. Macho, M. D. Penta, and M. Pinzger. What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories. *Empirical Software Engineering*, 25:2258–2301, 2019.

[42] A. B. Nassif, D. Ho, and L. F. Capretz. Towards an early software estimation using log-linear regression and a multilayer perceptron model. *Journal of Systems and Software*, 86:144–160, 2013.

[43] Z. Lu, Y. Sun, S. Liu, Z. Qian, H. Chen, S. Wu, and J. Zheng. Fuzzy-logic-based modeling and control for higee-aop nitric oxide attenuation with a complex gas–liquid mass-transfer-reaction process. *Industrial and Engineering Chemistry Research*, 61:3428–3438, 2022.

[44] Y. Hong, X. Zeng, P. Bruniaux, Y. Chen, and X. Zhang. Development of a new knowledge-based fabric recommendation system by integrating the collaborative design process and multi-criteria decision support. *Textile Research Journal*, 88:2682–2698, 2017.

[45] M. Ahsan, S. Stoyanov, C. Bailey, and A. Albarbar. Developing computational intelligence for smart qualification testing of electronic products. *IEEE Access*, 8:16922–16933, 2020.

[46] Lai Xu and Sjaak Brinkkemper. Concepts of product software. *European Journal of Information Systems*, 16(5):531–541, October 2007.

[47] Rakesh Garg, Ramesh Kumar, and Sandhya Garg. Madm-based parametric selection and ranking of e-learning websites using fuzzy copras. *IEEE Transactions on Education*, 62(1):11–18, 2019.

[48] Dhekra Ben Sassi, Anissa Frini, Wahiba Ben AbdessalemKaraa, and Naoufel Kraiem. Multi-criteria decision aid and artificial intelligence for competitive intelligence. In Moonis Ali, Young Sig Kwon, Chang-Hwan Lee, Juntae Kim, and Yongdai Kim, editors, *Current Approaches in Applied Artificial Intelligence*, pages 171–178, Cham, 2015. Springer International Publishing.

[49] Mrinmoy Majumder. *Impact of urbanization on water shortage in face of climatic*. Springer, 2015.

[50] Feng Liang, Fang Hou, Siamak Farshidi, and Slinger Jansen. Sentiment analysis for software quality assessment. *The 22nd Belgium-Netherlands Software Evolution Workshop Nijmegen*, 2021.

[51] Arya Roy. Recent trends in named entity recognition (NER). *CoRR*, abs/2101.11420, 2021.

[52] Zongcheng Ji, Fei Xu, and Bin Wang. A category-integrated language model for question retrieval in community question answering. *Springer Berlin Heidelberg*, pages 14–25, 2012.

[53] Tiedan Zhu and Kan Li. The similarity measure based on lda for automatic summarization. *Procedia Engineering*, 29:2944–2949, 2012.

[54] D. Losada and L. Azzopardi. An analysis on document length retrieval trends in language modeling smoothing. *Information Retrieval*, 11:109–138, 2008.

[55] Roi Blanco and Álvaro Barreiro. Probabilistic document length priors for language models. *LNISA*, pages 394–405, 2008.

[56] Phyu Pyar Moe and N. Hlaing. Android based questionnaires application for heart disease prediction system. *International Journal of Trend in Scientific Research and Development*, 2019.

[57] Parminder Singh Reel, Laurence S. Dooley, and K. C. P. Wong. A new mutual information based similarity measure for medical image registration. In *IET Conference on Image Processing (IPR 2012)*, pages 1–6, 2012.

[58] Albert C Yang, Ary Goldberger, and Chung-Kang Peng. Genomic classification using an information-based similarity index: Application to the sars coronavirus. *Journal of computational biology : a journal of computational molecular cell biology*, 12:1103–16, 11 2005.

[59] Dolors Costal, Carles Farré, Xavier Franch, and Carme Quer. How tertiary studies perform quality assessment of secondary studies in software engineering, 2021.

[60] Hoang Lam Nguyen, Nebras Nassar, Timo Kehrer, and Lars Grunske. Mofuzz: A fuzzer suite for testing model-driven software engineering tools. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1103–1115, 2020.

[61] Oscar Dieste and Sira Vegas. Tutorial 3: Pitfalls in the measurement methods applied in experimental software engineering - assessment and suggestions for improvement. In *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, EASE '22, page 465–466, New York, NY, USA, 2022. Association for Computing Machinery.

[62] Ahmed Haj Yahmed, Houssem Ben Braiek, Foutse Khomh, Sonia Bouzidi, and Rania Zaatour. Diverget: a search-based software testing approach for deep neural network quantization assessment. *Empirical Software Engineering*, 27(7), October 2022.

[63] Aidan Gilson, C. Safranek, Thomas Huang, V. Socrates, Ling Chi, R. Taylor, and David Chartash. How does chatgpt perform on the united states medical licensing examination? the implications of large language models for medical education and knowledge assessment. *JMIR Medical Education*, 9, 2023.

[64] J. Siegmund, Norbert Siegmund, and S. Apel. Views on internal and external validity in empirical software engineering. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 1:9–19, 2015.

[65] J. Ioannidis. Science with or without statistics: Discover-generalize-replicate? discover-replicate-generalize? *Behavioral and Brain Sciences*, 45, 2022.

[66] Lingzhi Wang, Shafiq R. Joty, Wei Gao, Xingshan Zeng, and Kam-Fai Wong. Improving conversational recommender system via contextual and time-aware modeling with less domain-specific knowledge. *ArXiv*, abs/2209.11386, 2022.

[67] P. Ralph and E. Tempero. Construct validity in software engineering research and software metrics. *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, 2018.

[68] Dongyao Wu, Liming Zhu, Xiwei Xu, S. Sakr, Daniel W. Sun, and Q. Lu. Building pipelines for heterogeneous execution environments for big data processing. *IEEE Software*, 33:60–67, 2016.

[69] C. Sanderson, Qinghua Lu, David M. Douglas, Xiwei Xu, Liming Zhu, and Jon Whittle. Towards implementing responsible ai. *2022 IEEE International Conference on Big Data (Big Data)*, pages 5076–5081, 2022.

[70] Veronika Bogina, Alan Hartman, T. Kuflik, and Avital Shulner Tal. Educating software and ai stakeholders about algorithmic fairness, accountability, transparency and ethics. *International Journal of Artificial Intelligence in Education*, 32:808–833, 2021.

[71] Eugénio Oliveira. Beneficial ai: the next battlefield. *Journal of innovation management*, 5:6–17, 2018.