Amir Ghari

NexLetter

A Newsletter Recommendation Engine

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

Sep 2025

Abstract

Author(s): First Name Last Name

Title: Title of the Thesis

Number of Pages: xx pages + x appendices

Date: 28 August 2020

Degree: Name of the degree

Degree Programme: Name of the degree programme

Specialisation option: Name of the specialisation option

Instructor(s): First name Last name, Title (e.g., Project Manager)

First name Last name, Title (e.g., Principal Lecturer)

The text section of the abstract is written to fit the space used on the page. The text section uses the **Body Text No Spacing** style.

Keywords: Keyword

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The originality of this thesis has been checked using Turnitin Originality Check service.

**Contents**

[1 Introduction 1](#_Toc203497098)

[2 Theoretical Background 1](#_Toc203497099)

[2.1 Recommender Systems: Overview and Importance 1](#_Toc203497100)

[2.2 Types of Recommender Systems 1](#_Toc203497101)

[2.2.1 Content-Based Filtering 2](#_Toc203497102)

[2.2.2 Collaborative Filtering 2](#_Toc203497103)

[2.2.3 Hybrid Recommender Systems 3](#_Toc203497104)

[2.2.4 The Chosen Method For This Thesis Project 3](#_Toc203497105)

[2.3 Evolution and Use Cases of Recommender Systems 3](#_Toc203497106)

[2.4 Mathematical Techniques Behind Recommendation 5](#_Toc203497107)

[2.5 NLP and Semantic Scoring in Modern Systems 6](#_Toc203497108)

[2.6 Chapter Summary 6](#_Toc203497109)

[3 Implementation 8](#_Toc203497110)

[3.1 Technology Stack Overview 8](#_Toc203497111)

[3.1.1 Backend Programming Language: Python 8](#_Toc203497112)

[3.1.2 Database Management System: **PostgreSQL** 9](#_Toc203497113)

[3.1.3 Frontend Framework: Next.js (Planned for Integration) 9](#_Toc203497114)

[3.1.4 Development Infrastructure and Tools 10](#_Toc203497115)

[3.2 System Architecture 10](#_Toc203497116)

[3.3 Data Collection & Storage 10](#_Toc203497117)

[3.3.1 Newsletter Data Source 10](#_Toc203497118)

[3.3.2 Database Design 11](#_Toc203497119)

[3.3.3 Data Ingestion and Processing 12](#_Toc203497120)

[4 Testing and evaluation 13](#_Toc203497121)

[4.1 How to add alternative text to images 14](#_Toc203497122)

[4.2 Subheading 15](#_Toc203497123)

[5 Conclusions 15](#_Toc203497124)

[4.3 Finish the document properties 15](#_Toc203497125)

[4.4 Check the accessibility of your thesis 16](#_Toc203497126)

[4.5 Save the Word document as an accessible PDF 17](#_Toc203497127)

[References 19](#_Toc203497128)

[Appendices 21](#_Toc203497129)

[Title of the Appendix 21](#_Toc203497130)

[Title of the Appendix 23](#_Toc203497131)

# Introduction

Write the introduction of your thesis here. Use line spacing 1.5 throughout the paper. Only the left edge is aligned, and the text is not hyphenated. Leave one blank line between paragraphs (press enter once).

Begin a new paragraph at the left margin, that is, do not indent the first line.

# Theoretical Background

## Recommender Systems: Overview and Importance

Recommender systems have become integral components of modern digital platforms, enabling personalized content delivery across domains such as e-commerce, news, entertainment, and education. Their primary purpose is to filter and rank large volumes of data in a way that aligns with individual user preferences, thereby improving user engagement and satisfaction.

At a high level, a recommender system analyzes user data (e.g., past interactions, preferences) and item characteristics (e.g., categories, keywords, metadata) to generate ranked lists of suggestions. These systems reduce information overload and help users discover relevant items more efficiently.

In this thesis, we focus on building a **newsletter and article recommender system** that leverages user preferences, reading behavior, and text similarity techniques to deliver more personalized news content.

## Types of Recommender Systems

Recommender systems are generally categorized based on the kind of data they use and how they process this data to generate recommendations. The three primary types are **content-based filtering**, **collaborative filtering**, and **hybrid recommender systems**. Each method has distinct strengths and limitations, and they are often combined to maximize recommendation accuracy and personalization.

### Content-Based Filtering

Content-based filtering relies on the properties or features of items to recommend similar ones to a user. For instance, if a user has previously engaged with articles categorized as "technology" or from the "United States," a content-based system will suggest new articles with similar tags. This approach typically represents items and user profiles using keywords, tags, or metadata and applies techniques like term frequency or cosine similarity for matching.

The main advantage of content-based filtering is that it does not require data from other users. However, it can lead to a phenomenon known as **over-specialization**, where users are only recommended items similar to what they’ve already seen, reducing diversity.

### Collaborative Filtering

Collaborative filtering focuses on user behavior rather than item attributes. It assumes that if two users have reacted similarly to the same set of items, they will continue to do so. There are two subtypes:

* **User-based collaborative filtering**: recommends items that similar users have liked.
* **Item-based collaborative filtering**: recommends items that have received similar user interactions.

This method works well when a large amount of user interaction data is available. However, it suffers from the **cold-start problem** (when new users or items have insufficient data) and **sparsity** (when user interactions are too few or scattered).

### Hybrid Recommender Systems

Hybrid recommender systems combine two or more recommendation strategies to benefit from their complementary strengths. For instance, a system can merge content-based filtering with collaborative filtering or integrate behavioral signals (like time spent on an article) with preference-based models.

Hybrid systems are often more accurate and robust, addressing many of the limitations associated with standalone approaches. For example, they can help mitigate the cold-start problem by using content features and user preferences until enough behavioral data is collected.

### The Chosen Method For This Thesis Project

The recommender system developed in this project follows a **hybrid approach**. It blends:

* User-defined **preferences** (preferred countries and categories)
* User **interaction signals** (liked articles and time spent)
* **NLP-based similarity** scoring (based on cosine similarity between liked article titles and new titles)

This layered structure aims to maximize personalization while adapting to evolving user behavior. By incorporating both explicit preferences and implicit behavioral signals, the system produces more relevant and contextually aware article suggestions.

## Evolution and Use Cases of Recommender Systems

Since their early use in information retrieval and online retail in the late 1990s, recommender systems have evolved to become central components of the digital user experience. Their development has closely mirrored advances in machine learning, data availability, and user modeling. Initially based on simple heuristics and keyword matching, modern recommenders now incorporate sophisticated algorithms capable of dynamic adaptation and personalization [1].

Recommender systems are now deployed across a broad range of industries and digital products:

* **E-commerce**: Platforms like Amazon use recommendation systems to suggest products based on browsing history, purchase patterns, and similar customer behavior [2].
* **Media and Entertainment**: Streaming services such as Netflix and Spotify leverage user preferences, time-based consumption, and deep content metadata to tailor viewing and listening experiences [3].
* **Social Media**: Platforms like Facebook, X (formerly Twitter), and TikTok utilize recommender systems to prioritize content in user feeds, drawing on interaction metrics such as likes, shares, and watch time [4].
* **News Aggregation**: Personalized news platforms recommend articles based on user interests, regional preferences, and reading history.
* **Education**: E-learning environments use recommender systems to suggest courses, study materials, and learning paths based on user goals and engagement [5].

A striking indicator of the importance of recommender systems is their impact on business outcomes in large technology companies. For instance:

* **Netflix** has publicly stated that more than 80% of the content watched on its platform is driven by recommendations. Their personalized home screen and row-based organization are entirely curated by algorithmic suggestions [3].
* **YouTube** reports that over 70% of user watch time comes from recommended videos. These recommendations are powered by complex deep learning models that factor in watch history, user preferences, device type, and more [6].
* **Amazon** generates a significant portion of its revenue from recommended products shown through “Customers who bought this also bought” and other dynamic modules [2].

These examples underscore the transformative role that personalized recommendation has played in enhancing user engagement, retention, and monetization, turning recommender systems into strategic assets rather than optional features.

## Mathematical Techniques Behind Recommendation

Modern recommender systems rely on a range of mathematical foundations to process user behavior and item data, generate profiles, and rank suggestions. Some of the most common techniques include:

* **Vector Space Models**: Items and user profiles are represented as vectors in a high-dimensional space, allowing similarity to be measured using distance or angle metrics (e.g., cosine similarity) [7].
* **Matrix Factorization**: Often used in collaborative filtering, this method decomposes a user-item interaction matrix into latent factors that represent underlying user preferences and item characteristics [8].
* **TF-IDF (Term Frequency–Inverse Document Frequency)**: Widely used in content-based filtering and natural language processing, TF-IDF assigns weights to words in a document relative to their frequency across a corpus, helping capture the importance of keywords in item descriptions or titles [9].
* **Clustering and Classification**: Unsupervised and supervised learning methods (e.g., k-means, decision trees) are used to group similar users or items and to predict preferences.
* **Regression and Ranking Models**: Algorithms such as logistic regression or learning-to-rank models are used to estimate the likelihood of a user engaging with an item and to order recommendations accordingly [10].

These mathematical tools, often embedded within broader algorithmic strategies, allow recommender systems to scale to millions of users and items while maintaining efficiency and relevance.

## NLP and Semantic Scoring in Modern Systems

Natural Language Processing (NLP) has added a new layer of semantic understanding to recommender systems. Traditional systems often rely on categorical metadata or interaction patterns, but NLP enables deeper insight into textual data such as article titles, product descriptions, or user-generated content [11].

Techniques such as TF-IDF vectorization and cosine similarity help quantify textual similarity between items, making it possible to recommend items based not only on explicit categories but also on the semantic overlap of their titles or content [9].

More advanced models, such as word embeddings (e.g., Word2Vec, GloVe) or transformer-based models (e.g., BERT), allow for richer representations of text by capturing context and nuanced meanings [12]. These models are increasingly used in real-time recommendation engines to match user interests with newly published content.

In the context of this project, NLP-based similarity scoring is used to match the titles of articles or newsletters a user has previously liked with the titles of new items. This enables the system to capture subtle, thematic consistencies beyond just category or location tags, improving recommendation quality—especially for new or niche content that might otherwise be overlooked.

## Chapter Summary

This chapter explored the theoretical underpinnings of recommender systems, starting from their types and evolution to the mathematical and linguistic foundations that drive their decision-making. We examined real-world business use cases demonstrating their significance, and how modern recommendation engines combine explicit user data with advanced NLP techniques for contextual awareness.

In the next chapter, we shift from theory to practice by examining the implementation of our hybrid recommender system. This includes the design of the backend architecture, database schema, recommendation pipeline, and scoring mechanisms that bring the system to life.

# Implementation

## Technology Stack Overview

This section provides an in-depth overview of the technologies utilized in the development of the newsletter recommendation engine. The selection of tools was driven by considerations of scalability, maintainability, and compatibility with content-based recommendation methodologies. The core stack comprises Python for backend development and data processing, PostgreSQL for structured data storage, and Next.js for frontend rendering.

### Backend Programming Language: Python

The backend system has been developed using **Python (version 3.x)** due to its extensive support for data manipulation, web development, and machine learning. Python is widely adopted in academic and industrial contexts for constructing recommender systems, primarily due to its rich ecosystem and ease of integration with APIs and databases.

Key libraries and frameworks utilized include:

* **Requests**: For handling HTTP requests to third-party services, including the newsdata.io API for fetching newsletter data.
* **Pandas** and **NumPy**: For efficient data cleaning, transformation, and statistical computation.
* **Scikit-learn** : To enable prototyping of recommendation algorithms such as TF-IDF-based similarity models or content clustering.

Python was selected over alternative languages (e.g., JavaScript/Node.js) for its proven effectiveness in building data-driven applications and its widespread use in academic projects involving recommendation engines [1][2][3].

### Database Management System: **PostgreSQL**

For data persistence, the system employs **PostgreSQL**, a powerful open-source relational database system renowned for its robustness, SQL compliance, and support for JSONB fields and advanced indexing mechanisms.

The database schema consists of the following core tables:

* newsletters (id, title, content, category, location, pub\_date)
* categories (id, name)
* locations (id, name)
* users (id, email, preferences)
* recommendations (newsletter\_id, user\_id, score)

PostgreSQL enables efficient query performance, particularly for operations involving keyword filtering, full-text search, and data aggregation. This makes it well-suited for content recommendation tasks where filtering by category, date, and region is essential [2][4].

### Frontend Framework: Next.js (Planned for Integration)

While the current scope of implementation emphasizes backend services and data pipelines, the frontend interface is planned to be developed using **Next.js**, a React-based framework that supports server-side rendering (SSR) and static site generation (SSG). These capabilities are critical for delivering a performant and SEO-friendly user experience when displaying categorized newsletters and recommendations.

Next.js offers native support for building API routes, which simplifies the integration between the frontend and backend without the need for separate middleware services. It has been successfully adopted in similar projects involving content aggregation and news delivery platforms [3][5].

### Development Infrastructure and Tools

To streamline development and deployment, the following tools and services are incorporated into the workflow:

* **Git and GitHub**: For version control, team collaboration, and continuous integration.
* **Docker** *(optional)*: For containerizing the application to ensure environment consistency across development and production stages.
* **Vercel** or **DigitalOcean** *(optional)*: Intended for frontend and backend deployment, respectively, offering scalable infrastructure solutions.

These tools enable modular development and facilitate rapid iteration during the engineering cycle. Moreover, the use of Docker supports reproducibility, which is essential for both academic validation and production reliability [2][4].

## System Architecture

## Data Collection & Storage

A critical component of the newsletter recommendation engine is the robust acquisition and structured storage of newsletter content. This section outlines the data source, fetching process, and current database design implemented using raw SQL and PostgreSQL.

### Newsletter Data Source

Newsletter articles are collected through the [NewsData.io](https://newsdata.io/) API, a service that provides categorized and localized news articles in real-time. The API supports rich query parameters that align with the system’s need to filter content by country, category, and language.

Data fetching is conducted using Python scripts via the requests library. Example API request parameters include:

* country: ISO 3166 country codes (e.g., fi for Finland)
* category: e.g., technology, sports, health
* language: en for English
* apikey: provided for authenticated access

The JSON response from the API is parsed and transformed to match the schema of the internal database.

### Database Design

The system uses **PostgreSQL** as the relational database management system. PostgreSQL was selected for its powerful indexing features, ACID compliance, and strong support for JSON and array types—particularly suitable for handling variable-length category fields and multilingual content [2][4].

Unlike systems that use Object-Relational Mapping (ORM) tools such as SQLAlchemy or Prisma [3], this project uses **handwritten SQL scripts** to perform all data operations. This provides direct control over the schema, execution plans, and optimization strategies.

The current implementation includes a single table named articles, defined as follows:

CREATE TABLE IF NOT EXISTS articles (

id SERIAL PRIMARY KEY,

title TEXT,

content TEXT,

link TEXT,

pub\_date TIMESTAMP,

source TEXT,

description TEXT,

country TEXT,

category TEXT[],

language TEXT,

image\_url TEXT

);

Each row in the articles table represents a newsletter article with essential metadata for later recommendation and filtering tasks.

### Data Ingestion and Processing

Data from the API is processed using Python and inserted into the PostgreSQL table using raw INSERT statements. An example insert operation:

INSERT INTO articles (

    title, content, link, pub\_date, source, description, country, category, language, image\_url

)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s);

# Testing and evaluation

According to accessibility requirements, images must have alternative text. Alternative text is not the same thing as a caption. Alternative text is a description of the content of an image read aloud by screen readers used by the visually impaired. It is not advisable to repeat the caption in the alternative text because screen readers read both contents.

When writing alternative text, think about what information you will not receive if you do not see the image. Use short sentences and plain language. Tell the essential about the picture - you don't have to explain everything.

Alt text can be blank for a decorative image. Decorative images are images that do not convey any information or that have been added for layout.



Figure 2. Long-tailed jaeger is common in Finnmark's mountain plateau in northern Norway. Its main wintering site is in the South Atlantic west of Africa.

## How to add alternative text to images

An alternative text is given to an image in a Word document as follows (Office 365 version):

1. Move the cursor over the image and right-click.
2. Select “Edit Alt Text…” to open the Alternative Text window.
3. Write a brief explanation of the essential content of the image.
4. If the image is purely decorative or the caption contains all the relevant information, mark the image as decorative.

Graphical user interface, text, application

Description automatically generated

The Microsoft Office 2016 version of Word is slightly different:

1. Move the cursor over the image and right-click.
2. Select “Format Picture…”
3. In the “Format Picture” window, select the third icon “Layout and Properties”.
4. Select “Alt Text” and enter a description of the image content in “Description”. Do not write anything under “Title”.

## Subheading

There must always be text or a new subheading below each heading.

Graphical user interface, text, application

Description automatically generated

# 5 Conclusions

## Finish the document properties

Once the content of your thesis is in order, finalise the document by specifying its properties. It is essential to ensure that the pdf file is accessible when you convert a Word file to PDF format. Type a title for the document in the File menu, under Info. Enter the title of your thesis as the title.

Graphical user interface, text, application

Description automatically generated

## Check the accessibility of your thesis

Word has a feature that lets you check the accessibility of a document.

1. On the File menu, click Info.
2. Then click Check for Issues.
3. Click Check Accessibility.

Graphical user interface, application, Word

Description automatically generated

The Accessibility Check window will then appear on the right side of the Word. The results of the scan show possible errors and warnings. For more information about results, click the item name in the results list. Word also tells you the reason for the repair, as well as give repair instructions. At least fix any errors.

## Save the Word document as an accessible PDF

Once you have checked your thesis for accessibility, convert it into an accessible PDF document.

1. Create a PDF file using either the Export function (Create PDF) or the Save As function.
2. In the save options, select **Document properties** and **Document structure tags for accessibility**.
3. Click Create bookmarks using Headings.

Graphical user interface, text, application, email

Description automatically generated

Do not use the Print to PDF function because the result is not accessible PDF.

References

1. Ricci, F., Rokach, L., & Shapira, B. (2015). *Recommender Systems Handbook*. Springer.
2. Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1), 76–80. <https://doi.org/10.1109/MIC.2003.1167344>
3. Gómez-Uribe, C. A., & Hunt, N. (2016). The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems*, 6(4), 13. <https://doi.org/10.1145/2843948>
4. Covington, P., Adams, J., & Sargin, E. (2016). Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys ’16)*. <https://dl.acm.org/doi/10.1145/2959100.2959190>
5. Manouselis, N., Drachsler, H., Verbert, K., & Duval, E. (2012). *Recommender Systems for Learning*. Springer. <https://doi.org/10.1007/978-1-4614-4361-2>
6. YouTube Official Blog. (2018). *How YouTube’s Recommendations Work*. Retrieved from <https://blog.youtube/news-and-events/how-youtubes-recommendations-work/>
7. Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11), 613–620. <https://doi.org/10.1145/361219.361220>
8. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8), 30–37. <https://doi.org/10.1109/MC.2009.263>
9. Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. <https://nlp.stanford.edu/IR-book/>
10. Burges, C. J. C. (2010). From RankNet to LambdaRank to LambdaMART: An Overview. *Microsoft Research Technical Report*. <https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/>
11. Beel, J., Gipp, B., Langer, S., & Breitinger, C. (2016). Research-paper recommender systems: A literature survey. *International Journal on Digital Libraries*, 17(4), 305–338. <https://doi.org/10.1007/s00799-015-0156-0>
12. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *NAACL-HLT*. <https://arxiv.org/abs/1810.04805>

Appendices

Title of the Appendix

The content of the appendix is ​​written here. Below are instructions for adding and removing attachments so that the headers remain the correct.

1. Instructions for adding a new attachment:
2. Move the cursor to the end of the last existing attachment page.
3. Choose the Page Layout tab. From the Page Break ribbon select Next Page under Section Breaks. This completes the printing of the new attachment, but the number in its header is not correct.
4. Double tap the header of the new attachment page with the wrong attachment number. If the “Link to previous” option is selected in the ribbon, press that button so that the option is no longer selected.
5. Please correct the attachment number.

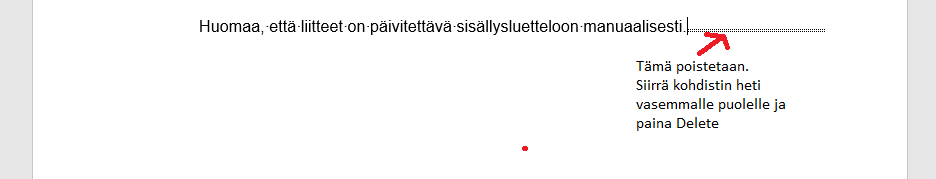
Instructions for removing an unnecessary attachment:

1. First select the entire attached page and press Delete to delete its contents.
2. When you are at the beginning of the attachment page you have emptied (see figure), double tap the header of the blank attachment page and press the Link to Previous button on the ribbon. The following dialogue box appears:



Answer Yes.

1. From the Home tab, toggle hidden characters if they are not visible:Piilomerkki painikkeen kuvake.
2. Remove the section break before the unnecessary attachment (see figure below).



Title of the Appendix

Content of the appendix is placed here