****

**HARAMAYA UNIVERSITY**

**COLLEGE OF COMPUTING AND INFORMATICS**

**DEPARTMENT OF SOFTWARE ENGINEERING**

**FUNDAMENTALS OF ARTIFICIAL INTELLEGENCE**

**RECOMENDATION SYSTEM FOR APPROPIRATE TEACHER FOR APPROPIRATE COURSE Documentation**

**Prepared by Group 7**

Name of members Id No

1. Atinaf Bedasa ---------------------- 0176/15
2. Kenean Dita ------------------------ 0629/15
3. Mesfin Ayele ----------------------- 0732/15
4. Michael T/Giorgis ---------------- 0739/15
5. Kidist Teshome -------------------- 0620/15
6. Eden Nigatu ----------------------- 0348/15

Table of Contents

[1. Overview 3](#_Toc200614203)

[2. Objectives 4](#_Toc200614204)

[3. System Architecture 5](#_Toc200614205)

[4. Data Flow and Processing Pipeline 6](#_Toc200614206)

[5. Recommendation Algorithm (Detailed) 7](#_Toc200614207)

[6. Model Training and Evaluation 8](#_Toc200614208)

[7. Performance and Evaluation Metrics 9](#_Toc200614209)

# 1. Overview

The Recommendation System for Appropriate Teacher for Appropriate Course is a comprehensive AI-powered platform developed to revolutionize the academic staff assignment process within educational institutions. Traditionally, assigning teachers to courses has been a manual, time-consuming task that often fails to take into account all the variables that can contribute to successful educational outcomes. This system addresses that problem by using data-driven methodologies to automate, refine, and optimize the matching process between teachers and courses. It is designed with the primary aim of enhancing educational delivery, improving student outcomes, and reducing administrative workload.

Built by students of Haramaya University's Software Engineering Department, the system embodies a real-world application of artificial intelligence concepts such as machine learning, natural language processing, and recommendation algorithms. At its core, the system analyzes various teacher attributes including primary and secondary subject expertise, education level, teaching style, certifications, availability, gender, country, years of experience, and student ratings to recommend the most suitable teachers for specific course needs. The project is implemented using **Streamlit**, a Python-based web framework that allows for quick and interactive development of user interfaces. The front-end facilitates input from administrative users while presenting the top recommended teachers in a clean, user-friendly table format.

Technically, the system uses a **hybrid recommendation model** that blends **content-based filtering** and **collaborative filtering**. Content-based filtering considers the characteristics of the teacher and the course, such as subject area, teaching approach, and other metadata. On the other hand, collaborative filtering utilizes feedback data, such as student satisfaction and historical matches, to improve future recommendations. This dual approach ensures a more holistic matching system, combining both statistical similarity and human experience.

The data pipeline begins with input collection, where teachers provide their professional profiles. This data is then preprocessed—normalized and encoded—before being used for similarity computations and model training. A supervised learning model, trained on historical data including student feedback and academic outcomes, ranks the teachers based on their probability of a successful match. Cosine similarity is applied to vectorized profiles to compute initial recommendations, which are further refined using model predictions.

The final output is presented as a ranked list of teachers, including key details such as full name, email, primary subject, rating, years of experience, and number of courses taught. The system also includes filtering tools for attributes like certification type, research activity status, and language fluency, giving administrators full control over the selection process.

# ****2. Objectives****

The primary objective of the Recommendation System for Appropriate Teacher for Appropriate Course is to intelligently match teachers to courses based on a variety of qualitative and quantitative factors, ultimately aiming to improve educational quality and administrative efficiency. At its core, the system seeks to transform traditional, manual teacher assignment into a data-driven and dynamic process. Educational institutions frequently face challenges in aligning available faculty resources with specific course requirements due to factors such as scheduling conflicts, subjective selection criteria, and limited visibility into teacher capabilities. This system addresses these issues by leveraging machine learning to create a scalable, fair, and efficient recommendation engine.

One of the major goals is to deliver personalized teacher recommendations tailored to course characteristics and student learning needs. This personalization is based on the comprehensive analysis of teacher profiles, which include metrics such as teaching style, experience level, certifications, availability, academic background, and historical performance. By aligning these features with course demands, the system enhances the likelihood of academic success and student satisfaction.

Another important objective is automation. The tool is designed to reduce the administrative burden on academic planners and decision-makers by streamlining the matchmaking process. This leads to quicker decision cycles and more consistent, objective assignments. Additionally, the system incorporates continuous learning from historical match results and user feedback to improve its recommendation quality over time. Through iterative model training and data collection, it becomes increasingly intelligent and adaptable to evolving institutional needs.

The project also aims to demonstrate the practical application of artificial intelligence in the education domain. It integrates natural language processing for analyzing teacher feedback and machine learning algorithms for ranking and recommendation. These technologies come together to support an intelligent system that not only makes accurate decisions but also adapts based on usage patterns and user input. The system is an embodiment of intelligent decision support tailored for the education sector, promoting fairness, efficiency, and scalability in teacher-course allocation.

# ****3. System Architecture****

The architecture of the recommendation system is modular and robust, designed to handle different functional aspects such as data collection, preprocessing, feature engineering, recommendation logic, and user interaction in a seamless manner. It consists of three main subsystems: the front-end interface, the recommendation engine, and the data handling infrastructure. Each of these components works in coordination to ensure real-time processing, accurate predictions, and intuitive user experience.

The **front-end interface** is developed using Streamlit, an open-source Python framework for creating interactive web applications. This interface allows administrators to input course-related preferences, such as subject requirements, education level, teaching style, and availability. Streamlit’s ease of use and fast refresh cycles make it an ideal choice for rapidly building and testing user-facing applications. Users are also able to apply filters on variables such as years of experience, gender, language, certification, and research involvement, allowing for dynamic control over the recommendations.

At the core of the system lies the **recommendation engine**, a machine learning-driven module responsible for feature extraction, data normalization, similarity calculations, and model inference. Feature vectors are generated based on teacher profiles and course demands. These vectors are scaled using a pre-trained StandardScaler, and similarity is computed using cosine similarity. To further enhance accuracy, a supervised learning model is employed to predict match quality, based on labeled training data.

The **data infrastructure** involves datasets stored as CSV files and preprocessed using Pandas and NumPy. These include encoded feature matrices, saved teacher vectors, and model parameters serialized with joblib. This design ensures portability and scalability of the system without requiring complex database setups. The architecture supports offline training and real-time recommendation, enabling both periodic model updates and instant feedback generation.

Altogether, the architecture ensures a clean separation of concerns, allowing for independent updates to the UI, model logic, and data handling without breaking system functionality. This modularity supports continuous development, testing, and integration of new features, making the system adaptable for future use cases in other institutions or departments.

# ****4. Data Flow and Processing Pipeline****

The recommendation system’s functionality is anchored in a well-structured data flow and processing pipeline, designed to handle complex, heterogeneous data while ensuring accuracy and scalability. The pipeline consists of sequential stages including input collection, data cleaning and transformation, feature encoding, similarity computation, and final recommendation ranking. Each stage is designed to maximize data integrity, processing speed, and the relevance of output results.

The **input collection stage** gathers detailed teacher profile data, including primary and secondary subject expertise, education level, teaching style, certifications, availability, gender, nationality, language fluency, years of experience, rating, and number of courses taught. These inputs are collected manually through administrative forms or extracted from institutional databases, and stored in structured formats such as CSV or Pandas DataFrames.

The **data preprocessing stage** ensures consistency and prepares the data for machine learning. Numeric values are scaled using a trained StandardScaler, while categorical features are one-hot encoded. For example, teaching styles like “Lecture” or “Interactive” are converted into binary indicators, ensuring uniformity in the feature space. Preprocessed feature vectors are saved for real-time access during inference. Missing data is handled using default values or imputation strategies to avoid sparsity in the feature matrix.

In the **feature transformation stage**, a course profile is dynamically generated based on user inputs. This profile is converted into a feature vector matching the format used for teacher vectors. Using features.pkl, the system ensures consistent feature ordering and indexing between training and inference phases.

**Cosine similarity** is then computed between the course profile vector and the teacher feature vectors. This step generates a preliminary list of teachers ranked by similarity score. These results are further passed through a trained supervised model that predicts match probability, considering past outcomes like feedback and grade improvement.

Finally, the **recommendation stage** filters the top-ranked teachers and displays them using Streamlit’s dataframe visualization. The system includes display features such as formatting and rounding for clarity, and users can further interact with the recommendations using side-panel filters.

This pipeline not only ensures accurate predictions but is also modular and extensible, allowing the integration of new data sources or preprocessing techniques with minimal changes to the existing system.

# ****5. Recommendation Algorithm (Detailed)****

The core intelligence of the recommendation system lies in its multi-layered, hybrid recommendation algorithm. Designed to maximize match quality and decision confidence, the algorithm merges content-based filtering, collaborative filtering, and supervised learning to generate high-quality teacher recommendations. This hybrid approach allows the system to leverage both static profile data and dynamic feedback, ensuring recommendations are personalized, data-informed, and constantly evolving.

The **content-based filtering** component uses explicit teacher attributes such as subject expertise, teaching style, education level, and availability. A course profile is created by encoding these preferences into a vector using one-hot encoding and numeric scaling. Each teacher profile is similarly represented as a feature vector. Using **cosine similarity**, the system compares the course profile with every teacher profile, producing a ranked list based on vector closeness. This ensures that the recommended teachers are similar in attributes to what the course requires.

The **collaborative filtering** component enriches the algorithm by learning from past interactions between courses and teachers. Specifically, it incorporates feedback data such as student satisfaction ratings, match success history, and academic performance outcomes. Although not explicitly visible in the front-end, this component helps the system identify hidden patterns, such as teachers who are generally more effective in particular types of courses or student demographics.

To fuse both methodologies, a **hybrid model** is employed. The model takes in both the similarity score from content-based filtering and features extracted from collaborative data, then passes them through a **supervised machine learning classifier** (e.g., logistic regression or random forest). This classifier, trained on historical match results, outputs a probability score that indicates the likelihood of a successful match. The final ranking is done by sorting teachers based on this score, not just raw similarity.

This layered architecture allows the system to be robust against data sparsity, biases, and cold-start problems. Teachers with limited feedback history can still be recommended based on their content features, while frequently matched teachers benefit from collaborative insights. The algorithm also accommodates future enhancements, such as incorporating NLP for analyzing open-text feedback or deploying deep learning models for embedding generation.

# ****6. Model Training and Evaluation****

The model training process is essential to the system’s ability to provide accurate and personalized recommendations. The system relies on supervised learning to train models that can predict the success of a course-teacher match, based on historical data. This process includes multiple stages: dataset preparation, feature engineering, model selection, training, validation, and evaluation.

The **training dataset** consists of past course-teacher pairings enriched with feedback metrics such as student ratings, grades before and after the match, and written comments. Text feedback is analyzed using sentiment analysis to convert qualitative insights into quantitative values. These features are combined with teacher and course metadata to form a structured training dataset suitable for classification.

During **feature engineering**, data is cleaned, missing values are handled, and inputs are normalized using a standard scaler. One-hot encoding is used for categorical features to ensure that the model can interpret teacher and course attributes correctly. The final feature vectors include over 50 dimensions, capturing both explicit and derived features.

A **supervised learning algorithm** such as logistic regression or gradient boosting—is trained to learn from these features. The target variable is typically a binary indicator (e.g., successful match vs. unsuccessful match) or a numerical score representing match quality. Model training is done in batches, and hyperparameters are optimized using cross-validation techniques to avoid overfitting.

The **evaluation metrics** include Mean Absolute Error (MAE), accuracy, precision, recall, and F1-score. MAE is particularly important for measuring prediction consistency, while classification metrics are used when the target is categorical. Evaluation is performed on a separate validation set to ensure the model generalizes well to unseen data.

This training pipeline is designed to run periodically, allowing the system to incorporate newly collected data. The retrained model is serialized using joblib and integrated into the inference pipeline without service downtime. This continuous learning loop enables the system to adapt and improve over time.

# ****7. Performance and Evaluation Metrics****

To ensure that the recommendation system delivers practical value and maintains a high level of reliability, a set of rigorous performance and evaluation metrics has been defined. These metrics assess the system’s efficiency, accuracy, responsiveness, and overall user satisfaction. Monitoring these metrics allows developers and administrators to continually evaluate and refine the system for improved outcomes.

One of the key performance indicators is **recommendation accuracy**, which measures how often the top-ranked teachers match with administrator expectations or receive positive feedback from students. This is often validated through satisfaction surveys or by tracking whether the recommendations were accepted and implemented. A high acceptance rate is an indicator of the system’s alignment with human judgment.

Another critical metric is **learning improvement**, measured by comparing student performance before and after being taught by the recommended teacher. This quantifies the academic impact of the recommendations and supports the system’s core objective of improving educational outcomes. Additional metrics such as **student retention** and **exam pass rates** can also be tracked for long-term impact evaluation.

Operationally, the system is evaluated on **response time** and **system latency**, ensuring that recommendations are generated in real-time or near-real-time to support administrative decision-making. The current goal is to maintain average response times below 2 seconds and system uptime at 99.9%, adhering to a standard SLA (Service-Level Agreement).

**System reliability** is measured using uptime logs and error reports. Crash recovery mechanisms and model fallback options (e.g., default recommendations in case of failure) are in place to guarantee consistent user experience.

From a machine learning perspective, **evaluation metrics** such as MAE (Mean Absolute Error) are used to assess model prediction quality, while metrics like precision, recall, and AUC-ROC are applied when using classification-based match scoring.

By combining user-centric and system-centric metrics, the system ensures robust monitoring and continuous feedback, facilitating an iterative development process that keeps the platform reliable, accurate, and responsive to institutional needs.