**Christopher Straka**

**Student ID:** 011551615

**Executive Summary:**

**Problem Statement:** Our streaming service is experiencing a decline in user engagement due to poor content discovery. There is no existing recommendation system, so our users struggle to find movies that match their preferences. This causes people to leave our platform and not recommend it as much, which leads to lower subscription counts. Finally, there is poor visualization and data insight into the ratings that users have left us. This leads to a limited understanding of customers’ preferences and hinders our ability to excel as a business.

**Customer Summary:** The primary customers for our recommender system are subscribers of the streaming service, ranging from casual viewers to cinephiles. These individuals seek a seamless and engaging viewing experience that offers content aligned with their tastes and preferences. The user-friendly interface of the proposed system is designed to cater to users of varying levels of technical proficiency. The application will run on their personal desktop and each movie will contain a link to imdb, a popular and user-friendly website for looking at movie reviews.

**Existing Systems Analysis:** There should be no substantial change to the company’s current infrastructure or systems. Prior to this initiative, the company’s streaming service lacked any recommendation process whatsoever. With the introduction of this new project, the company will now be able to make this desktop application available from their website. Users can then download the desktop application to their desktop and run it locally. This integration will fill the existing gap in automated, personalized movie suggestions, improving user experience without requiring significant changes to our existing infrastructure.

**Data:** The data for the proposed recommender system comes from two CSV files in a single Kaggle dataset. A links\_small.csv file of movie ids and a ratings\_small.csv file of ratings for those movies. There are 100,000 ratings, 164979 movies and 671 users from those two files. The links csv includes three unique identifiers: movie\_id serves to uniquely identify each movie in the dataset, imdb\_id which identifies the movie on imdb and tmdb\_id which identifies the movie on the movie database. The user will be given a list of imdb ids, and the user should be able to click on one to be brought to the imdb page for that movie. The ratings csv file is for user ratings. Each rating comes from a specific user id (1 – 671), and each movie\_id corresponds to a movie from the first csv file. Ratings are already normalized between 0 – 5. If a single user makes multiple ratings for multiple different movies, they will all show up here. Data will be collected from Kaggle and will not be preprocessed or cleaned as that work has already been done by its creator. The dataset from Kaggle will serve as a starting point that we can expand on later if desired.

**Project Methodology:** The development of the movie recommender desktop application will be guided by a modified version of the Waterfall Model, which follows a sequential design and development process. This method is particularly suitable for small-scale desktop applications with well-defined requirements.

Requirements: The waterfall model starts with gathering and listing requirements. For this project, the requirements are the following. The project must build a desktop application to offer movie recommendations for the company’s streaming service. A user should be able to provide any user id and the UI should respond back with all the movies that user has rated, as well as four movies the recommender system suggests they watch next. There should also be descriptive and non-descriptive data visualizations on the user interface. There should be three descriptive data visualizations: a bar chart that shows the top 10 most rated movies, a histogram that shows frequency rate of each rating, and a scatter plot that shows five groups of movies and their average ratings and their rating counts. There should also be one non-descriptive visualization, a heatmap that shows how likely the recommender system is to recommend 15 movies to 15 users.

Design: There are many popular diagrams that one can use to visualize software projects, such as UML, ERD, flowchart, and storyboard diagrams. A UML diagram is good for visualizing different components in a software system. An ERD is helpful for mapping out the data model in a system. A flowchart is good at mapping out the steps in a process and a storyboard is good for visualizing the user's journey when using an application. This application is quite simple in its design, so a diagram was not used in this case. The UI also is incredibly simple, hence a wireframe or a mock-up is not necessary there either.

Implementation: During this phase, the application will be coded directly. I will start by developing the machine learning model by downloading the relevant third-party libraries and then visualizing the data and training the algorithm. The dependencies will be stored in a requirements.txt file. After that, the UI for the frontend will be programmed using web technologies. Finally, the two will be connected using rust code so that the model’s output can be sent to the frontend. With these three things in place, the model, UI, and backend integration, the app will be successfully implemented.

Testing: Limited testing will be conducted to ensure the application's core functionality works as intended. The testing will involve running a set of predetermined inputs to verify that the system produces acceptable output. When training the model, a test set will also be separated from the original dataset so that the model can be tested later for its accuracy using metrics like RMSE and MAE. Unit tests, integration tests, system tests and acceptance tests could also round out the application, but these are outside of the scope for the current project. Unit tests verify individual components, integration tests check combined parts, system tests evaluate the complete application, and acceptance tests confirm end-user satisfaction.

Deployment: This phase is concerned with the deployment of the application. Since this is a standalone desktop app, the app will be packaged and made available for download on the company’s streaming website without any complex deployment process. The application even works offline, the only exception is when the model needs to be updated. Instructions will also be included to provide for a more seamless installation process. The application requires python 3.10.13 so that the user can run the model. The python distribution may be packaged in with the rest of the application or special instructions will be given to help users make it work.

Maintenance: Maintenance will be minimal for this application. There are already a lot of movies available in the Kaggle dataset. New movies and reviews can be added later, and the model can be retrained to account for that data, but that is outside the scope of the current project. The goal is to release an initial desktop application for the data on hand currently. There will also be logging in the application to make it easier to find and diagnose issues with the software if any pop up later.

**Project Outcomes:** The deliverables for this project include a model that has been trained on the Kaggle dataset using the SVD collaborative filtering algorithm. A model report in a Jupyter notebook used for analysis and visualization purposes. A desktop UI written in web technologies (HTML, CSS, TS) that feature the movie recommendations and data visuals. Documentation for the application to help install and understand the product. And a Tauri backend that ties the frontend and the model together into one cohesive desktop application. There are no deliverables for the design of this application concerning UML, ERD, flowcharts, storyboards, etc. because the application is simple enough to not need it.

**Implementation Plan:**

After the model, UI and desktop logic has been implemented, and tested following the waterfall methodology described earlier, the app can then enter the production implementation phase. The standalone application must be packaged before it can be made available on the company’s website. This will require careful consideration of the python dependencies like sklearn, and the desktop dependencies like Rust. Both are required for this app to work properly. There are also specific versions that must be installed like python 3.10.13 and pip 23.0.1, these can be packaged in the final application. Then there can be an initial release on a dedicated section of the website, followed by future updates based on new data and user feedback. Testing will have to be conducted to ensure that users can readily install the application on the website so that it can be made available for use. Key dependencies include website compatibility and backend support for hosting the application. Milestones include website integration, beta release, and final deployment. At the end of that process, there should be a tangible download that can be produced from the company’s website. And intangibly, there should be improved user engagement on the platform. Unit tests can be made to ensure the download feature works properly.

**Evaluation Criteria:** The model will primarily be assessed using its RMSE and MAE scores. An RMSE (Root Mean Squares Error) of >= 1 or an MAE (Mean Absolute Error) score >= 1 will be considered a bad score and hence, a failed recommendation system. The UI will be evaluated on whether it correctly lists the rated and recommended movies for a given user, as well as the data visualizations. It will also be evaluated for functionality, i.e., to see if the user id input works correctly and the UI dynamically updates with new data once it is fetched. Additionally, a predetermined set of users will be looked at to see if their recommended movies make sense from a high-level perspective.

**Environment:** The model, the program, the Jupyter notebook and the UI will all be coded in visual studio code with their related extensions (python, Jupyter, etc.) Tauri will be used for creating the desktop application interface, offering a lightweight solution that integrates well with various backend technologies. A M1 Macbook Pro running MacOS Sonoma 14.2.1 will also be used as the development machine for running this application. Please refer to the requirements.txt for all the packages and versions concerning the model and Jupyter notebooks. Please refer to the package.json for the Tauri dependencies. Python 3.10.13 is used with PIP 23.0.1.

**Costs:** The estimated development cost is $1,600, based on 32 hours of development at a rate of $50 per hour for a single developer (myself). The developer has skills in data science and can take care of the machine learning aspects of this project. This calculation covers the complete development lifecycle of the project. The tools used are open source, resulting in no additional licensing fees. The dataset is publicly available, eliminating data acquisition costs. There are no direct server costs as this is a standalone desktop application meant to be packaged and made available on the company’s existing website, although indirect costs are bound to go up depending on how many downloads are made. This cost evaluation is outside the scope for this project and speculative.

**Human Resources:** The main human resource is developer time. A developer skilled in handling the integration of the machine learning model with the application's backend and frontend is required. The project is planned for completion within a 32-hour timeframe, encompassing all stages from design to the final build. This approach ensures an efficient utilization of resources while leveraging cost-effective, open-source solutions for the development of the application.

**Projected Timeline and Milestones:**

**Total Duration: 32 hours | Dates are 2023**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Milestone** | **Start Date** | **End Date** | **Duration** | **Dependencies** | **Resources** |
| Planning | Dec 19th | Dec 19th | One day | Tauri/Scikit docs, pen and paper | Myself and my development environment. 8 hours. |
| Development | Dec 20th | Dec 21st | Two days | Vscode, python, rust, HTML, CSS, JS. Jupyter notebooks. | Myself and my development environment. 16 hours. |
| Documentation | Dec 22nd | Dec 22nd | One day | Vscode, README.md files, supporting documentation | Myself and my development environment. 8 hours. |

**Should start on Dec 19th and be finished at the end of Dec 22nd (or start of Dec 23rd).**