Project: End-to-End Recommender System with Hybrid Models

**Goal:** To build and deploy a comprehensive recommendation system that combines collaborative filtering, advanced NLP (BERT), and a machine learning ranker. The final output will be an interactive web application built with Streamlit.

WEEK 1: Foundation — EDA & Classic Recommenders

✅ Day 1: Project Setup & Exploratory Data Analysis (EDA)

* **Objective:** Understand the dataset's structure, clean it, and perform initial filtering to create a manageable and dense dataset for modeling.
* **Key Tasks:**

1. **Environment:** Set up a Python environment (conda or venv) with core libraries: pandas, numpy, matplotlib, seaborn.
2. **Data Loading:** Load the Amazon reviews and metadata for a chosen category (e.g., "Books").
3. **Cleaning & Filtering:** Handle missing values. Critically, filter the dataset to keep only active users (e.g., >20 reviews) and popular items (e.g., >50 reviews) to reduce data sparsity.
4. **Visualization:** Generate plots to understand the data:

* Distribution of ratings (overall).
* Top 20 most-reviewed products.
* Histogram of review text length.
* **Deliverables:** data/cleaned\_reviews.csv, notebooks/eda.ipynb

✅ Day 2: Baseline Collaborative Filtering Model

* **Objective:** Implement a classic collaborative filtering model to serve as a performance baseline.
* **Key Tasks:**

1. **Library:** Use the Surprise library, which is excellent for building and evaluating rating prediction models.
2. **Model:** Implement SVD (Singular Value Decomposition).
3. **Evaluation:** Split the data into training and testing sets. Train the SVD model and evaluate its performance using:

* **RMSE** (Root Mean Squared Error) for rating prediction accuracy.
* **Precision@K & Recall@K** for the quality of the top-K recommendations.
* **Deliverables:** models/svd\_model.pkl, outputs/baseline\_results.csv

✅ Day 3: Text Preprocessing & Sentiment Analysis

* **Objective:** Extract a sentiment signal from the review text to use as a feature in later models.
* **Key Tasks:**

1. **Text Cleaning:** Apply standard text preprocessing: convert to lowercase, remove punctuation/stopwords.
2. **Sentiment Scoring:** Use a pre-trained model like **VADER** to calculate a compound sentiment score for each review. VADER is effective for review-style text.
3. **Data Augmentation:** Add the sentiment\_score as a new column to your dataset.

* **Deliverables:** data/reviews\_with\_sentiment.csv, notebooks/sentiment\_analysis.ipynb

✅ Day 4: Content-Based Recommender (TF-IDF)

* **Objective:** Build a recommender that suggests items based on the similarity of their review text.
* **Key Tasks:**

1. **Vectorization:** Use TfidfVectorizer from scikit-learn to convert the cleaned review text into numerical vectors.
2. **Similarity Calculation:** Compute the cosine similarity between all item vectors to find items with similar textual content.
3. **Recommendation Function:** Create a function that takes an item ID and returns the top N most similar items.

* **Deliverables:** models/tfidf\_model.pkl, recommenders/content\_recommender.py

✅ Day 5-6: Streamlit App v1 & Merging Outputs

* **Objective:** Create the first version of the user-facing dashboard and consolidate model outputs.
* **Key Tasks:**

1. **Streamlit UI:** Build a simple interface where a user can input their ID and get a list of top 5 recommendations from the SVD model.
2. **Display Info:** Show the recommended item's title, predicted rating, and average sentiment.
3. **Merge Outputs:** Create a unified CSV file containing recommendations from both SVD and TF-IDF models for comparison.

* **Deliverables:** app/app\_v1.py, outputs/initial\_recommendations.csv

WEEK 2: Semantic NLP & Hybrid Models

✅ Day 7-8: BERT Embeddings & Semantic Recommender

* **Objective:** Move beyond keyword matching (TF-IDF) to semantic understanding using BERT.
* **Key Tasks:**

1. **Embeddings:** Use the sentence-transformers library (e.g., with the all-MiniLM-L6-v2 model) to generate dense vector embeddings for product titles or reviews.
2. **Semantic Similarity:** Calculate cosine similarity on these BERT embeddings.
3. **Comparison:** Create a recommender based on BERT similarity and compare its results qualitatively against the TF-IDF model. Note the differences in recommendations.

* **Deliverables:** models/bert\_embeddings.npy, recommenders/bert\_recommender.py, notebooks/bert\_vs\_tfidf.ipynb

✅ Day 9: Hybrid Model v1 (Weighted Average)

* **Objective:** Create the first hybrid model by combining the outputs of previous models.
* **Key Tasks:**

1. **Feature Combination:** For a given user-item pair, you have multiple signals: SVD predicted rating, BERT similarity score, and average sentiment score.
2. **Weighted Formula:** Combine these signals using a simple weighted average to calculate a final hybrid score. For example: Hybrid Score = (0.5 \* SVD) + (0.3 \* BERT) + (0.2 \* Sentiment).
3. **Evaluation:** Re-rank recommendations based on this new hybrid score and evaluate.

* **Deliverables:** recommenders/hybrid\_model\_v1.py, outputs/hybrid\_v1\_results.csv

WEEK 3: Advanced ML & Deep Learning

✅ Day 10: XGBoost Hybrid Ranker

* **Objective:** Use a machine learning model to learn the optimal way to combine features, moving from a simple weighted average to a "learning-to-rank" approach.
* **Key Tasks:**

1. **Feature Engineering:** Create a training dataset where each row is a (user, item) pair. Features include: svd\_score, bert\_similarity, sentiment\_score, user\_average\_rating, item\_average\_rating. The target is the actual rating.
2. **Train Ranker:** Train an XGBoost Regressor to predict the rating based on these features.
3. **Evaluation:** Evaluate the ranker using metrics like **AUC** (if framed as classification) and **Precision@K**.

* **Deliverables:** models/xgb\_ranker.pkl, data/xgb\_features.csv

✅ Day 11: Neural Collaborative Filtering (NCF)

* **Objective:** Implement a deep learning-based collaborative filtering model.
* **Key Tasks:**

1. **Framework:** Use PyTorch or TensorFlow/Keras.
2. **Architecture:** Build a model that takes user\_id and item\_id as input, passes them through separate embedding layers, concatenates the embeddings, and feeds them through dense layers to predict an interaction score.
3. **Training:** Train the model using Binary Cross-Entropy (BCE) loss, treating all viewed items as positive examples and sampling some unseen items as negative examples.

* **Deliverables:** models/ncf\_model.pt, outputs/ncf\_eval.csv

✅ Day 12: Final Hybrid Scoring Engine

* **Objective:** Combine all models into a final, robust scoring engine.
* **Key Tasks:**

1. **Ensemble Predictions:** For a candidate list of items for a user, generate prediction scores from all models: SVD, BERT, XGBoost, and NCF.
2. **Meta-Model/Final Logic:** Use a final weighted logic or a simple meta-model to combine these scores into a final ranking. The XGBoost model often serves this purpose well.

* **Deliverables:** recommenders/final\_recommender.py, outputs/final\_hybrid\_recommendations.csv

WEEK 4: App Polish, Deployment & Portfolio

✅ Day 13: Streamlit App v2

* **Objective:** Enhance the Streamlit app to be more interactive and feature-rich.
* **Key Tasks:**

1. **Tabs:** Add separate tabs for "User-Based Recommendations" and "Item-Based Similarities".
2. **Model Selection:** Add a dropdown to allow switching between different recommendation models (e.g., "SVD", "BERT", "XGBoost Hybrid").
3. **Visuals:** Include more visualizations, like a word cloud of reviews for a selected item or a bar plot of predicted ratings.

* **Deliverables:** app/app.py (updated)

✅ Day 14: Deployment

* **Objective:** Make your application publicly accessible online.
* **Key Tasks:**

1. **Platform Choice:**

* **Streamlit Cloud:** The easiest and most direct way.
* **HuggingFace Spaces:** Also very easy and great for ML/AI projects.

1. **Dependencies:** Create a requirements.txt file.
2. **Deploy:** Follow the platform's instructions to link your GitHub repo and deploy the app.

* **Deliverables:** A live URL to your deployed application.

✅ Day 15-17: Final Polish & Portfolio Update

* **Objective:** Finalize the project for your portfolio.
* **Key Tasks:**

1. **GitHub Repo:** Write a detailed README.md explaining the project's goal, structure, how to run it, and key results. Include screenshots of the app.
2. **Code Cleanup:** Add comments, refactor code for clarity, and add basic unit tests for key functions.
3. **Demo & Resume:** Record a short (1-2 minute) video demonstrating the app. Update your LinkedIn and resume with a strong bullet point describing this project, highlighting the technologies used (Python, BERT, XGBoost, PyTorch, Streamlit) and the skills demonstrated (hybrid modeling, deployment).

* **Deliverables:** Final GitHub repo, demo video, updated resume/LinkedIn.