**Advanced Hybrid Recommendation System Using Amazon Reviews Dataset**

**Table of Contents**



---

**Introduction**

In today's online shopping world, recommending the right products to users can make a big difference. This project is about building a smart recommendation system using Amazon Reviews data. Our goal is to create a system that suggests products users might like based on what they've reviewed before.

We will go through these main steps:

1. **Data Preprocessing:** Cleaning and organizing the data.
2. **Feature Engineering:** Turning text data into useful information.
3. **Model Selection & Development:** Building the recommendation model.
4. **Hyperparameter Tuning (Grid Search):** Finding the best settings for our model.
5. **Retrain and Evaluation:** Training the model again with the best settings and checking how well it works.
6. **Visualization of Results:** Making graphs to understand the results.
7. **Saving and Loading the Model:** Keeping the model ready for future use.
8. **Conclusion:** Summarizing what we did and what we learned.

---

**Project Structure**

Here's how the project is organized:

1. **Data Preprocessing:** Loading and cleaning the data.
2. **Feature Engineering:** Turning product titles and descriptions into numbers the model can understand.
3. **Model Selection & Development:** Creating the recommendation system.
4. **Hyperparameter Tuning (Grid Search):** Testing different settings to make the model better.
5. **Retrain Model with Best k:** Training the model with the best settings we found.
6. **Visualization of Results:** Making charts to show how well the model works.
7. **Saving and Loading the Model:** Saving the model so we can use it later without retraining.
8. **Conclusion:** Summing up the project.

You can find the code for this project in both .py (Python script) and .ipynb (Jupyter Notebook) formats. There's also a PDF version for easy reading.

---

**1. Data Preprocessing**

Cleaning and preparing the data is the first and very important step. Good data leads to better recommendations.

**What We Did:**

- **Loaded the Data:** Used the Amazon Reviews dataset.
- **Handled Missing Values:** Removed incomplete entries and filled in missing text.
- **Removed Duplicates:** Made sure each user has only one review per product.
- **Filtered Users and Products:** Kept only users and products with at least 5 reviews to make the data more reliable.
- **Encoded IDs:** Turned user and product IDs into numbers so the model can work with them.
- **Train-Test Split:** Divided the data into training and testing sets to train the model and then check how well it works.
- **Created Interaction Matrices:** Made tables showing which users reviewed which products.

---

**2. Feature Engineering**

Turning text into numbers is key for the model to understand product information. We used TF-IDF and reduced the size of our data to make it easier to work with.

**What We Did:**

- **Combined Text:** Merged product titles and descriptions into one text field.

- **TF-IDF Vectorization:** Converted text into numerical features that show how important each word is in a product's description.
- **Dimensionality Reduction:** Reduced the number of features to make computations faster and remove less important information.

**Code Implementation**

**Explanation:**

1. **Combining Textual Features:** We merged the product title and its detailed review text into one big text field called product_text. This gives the model more information about each product.
2. **TF-IDF Vectorization:**
    - **Purpose:** TF-IDF helps in understanding how important each word is in a product's description compared to all products.
    - **Configuration:** We removed common English words (like "the", "and") that don't add much meaning and limited the features to the top 5000 words to keep things manageable.
3. **Dimensionality Reduction with Truncated SVD:**
    - **Purpose:** Reducing the number of features from 5000 to 200 makes the data easier and faster for the model to handle without losing important information.
    - **Outcome:** We get a matrix (item_features) where each product is represented by 200 numbers instead of 5000, capturing the most important information.

---

**3. Model Selection & Development**

Now that our data is ready, we build the recommendation system. We combine two methods: Collaborative Filtering and Content-Based Filtering.

1. **Collaborative Filtering with SVD:**
    - **SVD (Singular Value Decomposition):** Breaks down the user-item interaction matrix into smaller matrices to find hidden patterns in user preferences.
    - **Predicted Ratings:** We use these patterns to predict how a user might rate products they haven't reviewed yet.
2. **Content-Based Filtering with Cosine Similarity:**
    - **Cosine Similarity:** Measures how similar two products are based on their features (from item_features).
    - **Item Similarity Matrix:** A table showing how similar each product is to every other product.
3. **Hybrid Recommendation Function:**

- **Combining Methods:** We take both the predicted ratings from collaborative filtering and the similarity scores from content-based filtering.
- **Normalization:** Both scores are scaled between 0 and 1 to ensure they contribute equally.
- **Final Score:** We add both scores together to get a final recommendation score.
- **Top Recommendations:** The products with the highest final scores are recommended to the user.

4. **Sample Recommendation:**
    - We picked the first user from the training set and generated a list of recommended products for them.

---

**4. Hyperparameter Tuning (Grid Search)**

To make our model as good as possible, we need to find the best settings. Here, we focus on finding the best number of hidden factors (k) in our SVD model.

1. **Testing Different k Values:** We tried values of k from 1 to 100 to see which number of hidden factors works best for our model.
2. **Calculating MSE:**
    - **Mean Squared Error (MSE):** Measures how close the model's predictions are to the actual ratings. Lower MSE means better accuracy.
    - **Procedure:** For each k, we predicted ratings and compared them to the actual ratings in the test set where users have provided feedback.
3. **Handling Errors:** If any k value causes an error (like shape mismatches), we skip it and move to the next one.
4. **Selecting the Best k:** After testing all values, we pick the k that resulted in the lowest MSE, meaning it was the most accurate.

---

**5. Retrain Model with Best k**

After finding the best k from our grid search, we train the model again using this optimal value and check how well it performs using another metric called NDCG.

1. **Retraining with Best k:** Using the best number of hidden factors found from our grid search, we perform SVD again to get a more accurate model.
2. **NDCG (Normalized Discounted Cumulative Gain):**
    - **Purpose:** Measures how good the ranking of recommended products is. It gives higher scores if relevant items are ranked higher.

- **Calculation:** For each user, we compare the top-10 recommended items with the actual items they liked. The NDCG score tells us how well the recommendations match the user's preferences.

3. **Result:** A higher NDCG score means our recommendations are better at ranking relevant products at the top.

---

**6. Visualization of Results**

Seeing our results in charts helps us understand how our model is performing and whether our recommendations make sense.

**Code Implementation**

1. **MSE vs. k Plot:**
   - **What It Shows:** How the Mean Squared Error changes as we try different numbers of hidden factors (k).
   - **Why It's Useful:** Helps us see if increasing k makes our predictions better or worse.
2. **Top-N Recommendations:**
   - **Table:** Lists the top 10 products recommended for a specific user, showing both their IDs and titles.
   - **Bar Chart:** Visualizes the recommended products, making it easier to see which products are being suggested.
3. **Insights:**
   - **Model Performance:** The MSE plot helps us confirm if our chosen k is indeed the best.
   - **Recommendation Quality:** By looking at the top recommendations, we can judge if the system is suggesting relevant and useful products to users.

---

**7. Saving and Loading the Model**

Once our model is ready and working well, we save it so we can use it later without having to build it again from scratch.

1. **Saving Components:**
   - **Encoders:** We save the user_encoder and item_encoder so we can use the same encoding for new data later.
   - **SVD Components:** Saving U_best, Sigma_best, and Vt_best allows us to predict ratings without doing SVD again.
   - **Item Similarity Matrix:** Saving item_similarity helps in quickly finding similar products when making recommendations.
2. **Loading Components:**

- **Purpose:** When we want to use our model in the future, we can load these saved parts and start making recommendations right away without retraining.

3. **Benefits:**
    - **Efficiency:** Saves time since we don't need to process the data and train the model again.
    - **Consistency:** Ensures that the model works the same way every time we use it.

---

**8. Conclusion**

We've successfully built a smart recommendation system that suggests products to users based on their past reviews and the content of the products. Here's what we achieved:

1. **Data Preprocessing:** Cleaned and organized the data to make it ready for modeling.
2. **Feature Engineering:** Turned product titles and descriptions into numbers that the model can understand.
3. **Model Development:** Created a hybrid system that uses both user ratings and product information to make recommendations.
4. **Hyperparameter Tuning:** Found the best number of hidden factors to make the model as accurate as possible.
5. **Evaluation:** Checked how well the model works using metrics like MSE and NDCG.
6. **Visualization:** Made charts to see how different settings affect the model and to show example recommendations.
7. **Model Persistence:** Saved the model so it can be used in the future without retraining.

**Future Improvements:**

- **Add More Features:** Include more information about users or products to make recommendations even better.
- **Real-Time Recommendations:** Make the system faster so it can recommend products instantly as users browse.
- **More Evaluation Metrics:** Use other ways to measure how good the recommendations are.
- **Build a User Interface:** Create an easy-to-use interface so people can interact with the recommendation system.