**A HIGH LEVEL DESIGN**

**On**

**PRUS: Product Recommendation System Based on User Specification and Customer Reviews**

**Submitted in the partial fulfilment of requirements to**

**CS - 454 - Project Work**

**By**

**Batch - 19**

**Oruganti Monik Paparao (Y22CS139)**

**Pendyala Skanda Bhagavan (Y22CS145)**

**Tulam Sai Sudheer (Y22CS184)**



**R.V.R. & J.C. COLLEGE OF ENGINEERING (Autonomous)**

**(NAAC ‘A+’ Grade)**

**Approved by AICTE:: Affiliated to Acharya Nagarjuna University**

**Chandramoulipuram::Chowdavaram**

**GUNTUR – 522 019, Andhra Pradesh, India.**

**Smt. N. Zareena Dr. S. J. R. K. Padmini Valli Dr. M. Sreelatha**

**Project Guide Project In-Charge Prof. & HOD**

**Contents:**

**1. Problem Statement**

**2. Functional Requirements**

**3. Basic Architecture**

**a. Major components**

**b. Diagram**

**c. Workflow**

**d. Pattern**

**4. Non Functional Requirements**

**5. Technology Stack**

**6. Summary**

1. **Problem Statement**

**Manual product recommendation systems often rely on overall product ratings or general customer reviews, which can overlook important details about specific product features. Most existing systems do not allow users to specify which features matter most to them, resulting in less personalized recommendations.**

Additionally, many approaches focus only on positive sentiments, ignoring valuable negative feedback that could influence buying decisions. PRUS addresses these challenges by applying aspect-level sentiment analysis to customer reviews, breaking them down into feature-specific opinions such as camera quality, battery life, or screen resolution. This system enables users to specify their preferences and ranks products using a novel scoring method called RANK-ify, which considers both positive and negative sentiments for each feature.

Key goals include:

* User-specified feature-based recommendations.
* Aspect-level sentiment analysis of customer reviews.
* Flexible ranking with emphasis on positive, negative, or balanced feedback.
* Enhanced personalization and decision support for buyers.

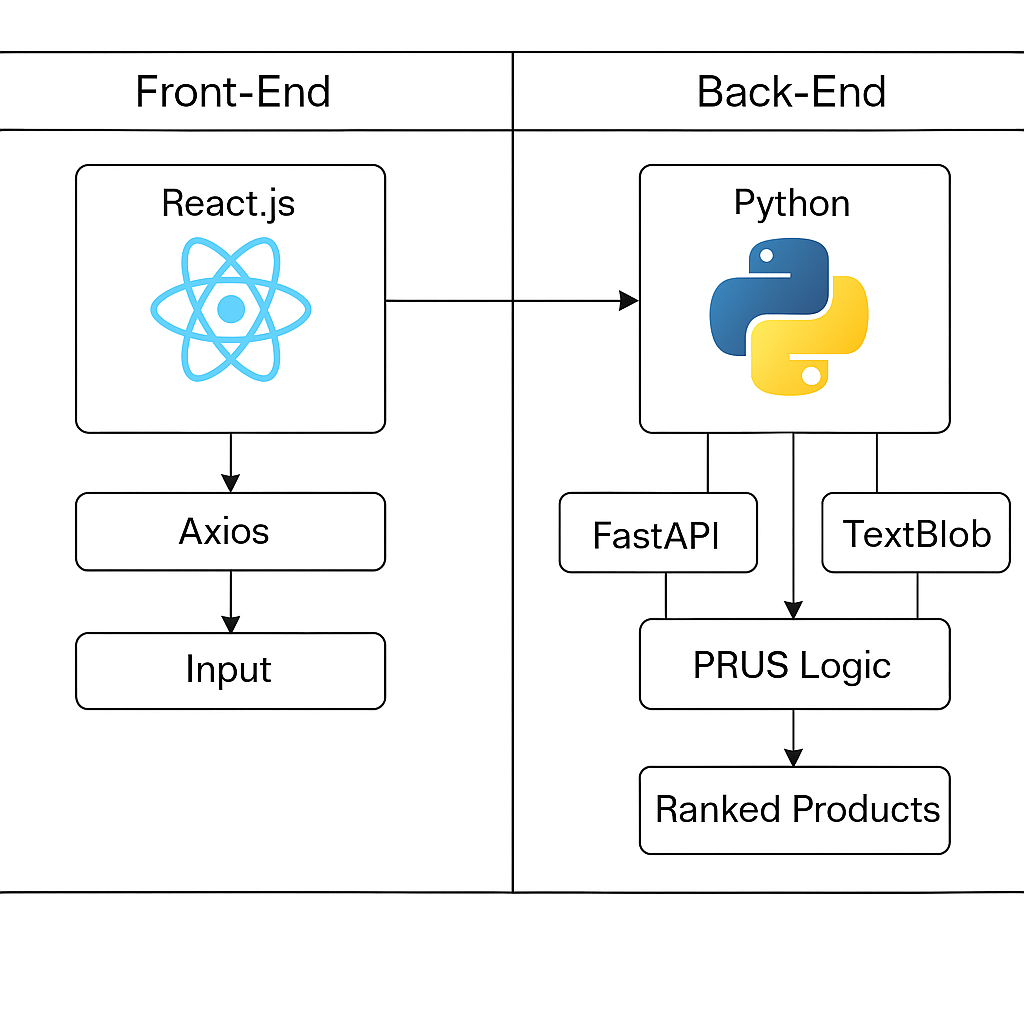
1. **Functional Requirements**

|  |  |
| --- | --- |
| Functionality | Description |
| User Input of Preferences | Allow users to enter specific product features (e.g., battery life, camera) via a web form. |
| Submit Query to Backend | Send user preferences and sentiment weights to backend API using Axios for processing. |
| Sentiment Analysis of Reviews | Backend performs sentence-level sentiment analysis and identifies feature-specific sentiments using NLP tools like TextBlob. |
| Feature Matching and Scoring | Match user-specified features with review data and calculate product scores using the RANK-ify algorithm. |
| Generate Ranked Product List | Produce a ranked list of products based on user input and sentiment analysis results. |
| Display Results on Frontend | Show the ranked product recommendations clearly using cards, lists, or tables on the frontend. |
| Handle Errors and Invalid Input | Validate user input and handle missing data, API failures, or incorrect queries gracefully |

1. **Basic Architecture**
   1. **Major Components:**

|  |  |  |
| --- | --- | --- |
| Component | Description | Tools/Libraries Used |
| 1. Data Collection | Collection of product reviews from online sources (e.g., Amazon mobile reviews). | Dataset |
| 2. Data Preprocessing | Cleansing and normalization of raw reviews — removing missing data, stop words, punctuation, short texts, and performing lemmatization. | Python, NLTK, WordNet Lemmatizer |
| 3. Sentence Segmentation | Breaking down reviews into individual sentences to capture feature-specific sentiments. | NLTK (sentence tokenizer) |
| 4. Feature Extraction & Sentiment Analysis | Identifying product features and analyzing sentiment polarity (positive, negative, neutral) at the sentence level. | TextBlob |
| 5. Sentiment Scoring | Calculating sentiment scores for each feature using:  Score = (Positive − Negative) / Total count | Python |
| 6. RANK-ify Algorithm | Custom algorithm to assign weights (w₁, w₂) to features, compute feature scores (FeaSco), and rank products based on user query. | Custom Python Algorithm |
| 7. Ranking Metrics | Evaluating recommendation accuracy using Rank Score (RS), DCG, and nDCG. | RS, DCG, nDCG formulas |
| 8. Refined Dataset | Filtered and cleaned dataset used for analysis and testing, including only useful and complete reviews. | Processed using Pandas, Numpy |
| 9. User Preferences Input | Accept user-specified feature priorities for personalized recommendations. | Web form / CLI input |
| 10. Output Visualization | Display of ranked products based on feature sentiments and query alignment. | Web UI / CLI / CSV Export |
| 11. User Interface | Interface for interacting with the system — product queries, preferences, and results. | React.js / Flask / FastAPI |
| 12. Data Storage | Store processed review data and feature | MongoDB |

* 1. **System Architecture Diagram**

****

* 1. **Work Flow**

Here's the step-by-step mechanism of how the PRUS system works detailed explanations for each step:

**Step 1: Collect Product Reviews:**

The PRUS system begins by gathering a large dataset of customer reviews from an e-commerce platform like Amazon. These reviews are focused on a specific category, such as mobile phones. Each review typically contains the product name, brand, and a written comment from a user. This raw data forms the foundation for understanding what customers think about different products.

**Step 2: Clean and Prepare the Reviews:**

Raw reviews often contain noise like emojis, short or meaningless sentences, punctuation, numbers, and common words (called stop words) that don’t add much value (e.g., “the”, “is”, “and”). In this step, the system removes all such noise to improve data quality.  
Additionally, the reviews are normalized using techniques like lemmatization, which simplifies words to their base forms (e.g., “running” becomes “run”).

**Step 3: Break Reviews into Sentences:**

Many product reviews contain multiple opinions within a single comment. For example, a user might say: “The camera is great, but the battery drains quickly.” To accurately understand sentiments about individual features, each review is split into separate sentences using tools like NLTK (Natural Language Toolkit). This allows the system to examine each thought independently.

**Step 4: Extract Features and Sentiments:**

After splitting the reviews into sentences, the system looks for product features mentioned (e.g., “battery”, “camera”, “design”) and the sentiment (positive, negative, or neutral) expressed toward each one.  
This is done using TextBlob, a Python library that performs sentiment analysis by assigning a polarity score to each sentence.  
*For example:*

*Sentence: “The display is stunning” → Feature: “display” → Sentiment: Positive*

*Sentence: “The phone overheats” → Feature: “overheating” → Sentiment: Negative*

**Step 5: Match Features with User Query:**

When a user visits the system, they enter a search query specifying what features they care about most (e.g., “good battery life”, “excellent camera”). The system compares this query with the features it found in the reviews. This allows it to filter out irrelevant features and focus only on those the user considers important.

**Step 6: Assign Scores to Features:**

Next, the system goes through all the reviews and counts how many times each feature is mentioned positively, negatively, or neutrally.  
A sentiment score is then calculated for each feature using this formula:

*Score = (Positive Count−Negative Count) / Total Count*

This scoring allows the system to measure how well each feature is perceived by users. For example, if "battery life" has 80 positive mentions and 20 negative mentions, it will get a high score, indicating good feedback.

The system also lets the user adjust weights (called w₁ for positives and w₂ for negatives), so they can decide if they care more about good reviews or avoiding bad ones.

**Step 7: Rank Products Using RANK-ify Algorithm:**

This is the heart of the system. The RANK-ify algorithm takes the calculated feature scores and generates an overall score for each product.  
If a product has multiple features matching the user query (e.g., “camera” and “battery”), the scores for those features are combined, and a Rank Score is assigned. The algorithm then sorts all products from highest to lowest based on how well they match the user’s preferences. It also considers how important each feature is to the user and applies positive/negative sentiment weights to generate a final list.

**Step 8: Display the Recommended Product List:**

Finally, the system presents a ranked list of products to the user. Each product in the list is selected based on how well it performed in the reviews for the features the user prioritized. Unlike traditional systems that rely on overall star ratings, PRUS offers a personalized list by Considering aspect-level feedback (specific features), Giving importance to both good and bad reviews, Letting users customize their priorities. This helps the user make smarter decisions by choosing a product that truly meets their expectations rather than one that is just generally popular.

**4. Non Functional Requirements**

|  |  |
| --- | --- |
| Category | Description |
| Performance | The system should return ranked results within a few seconds of receiving the user input. |
| Scalability | The system should send the user preferences and sentiment weights (positive/negative) to the backend API using Axios. |
| Security | The backend should be designed to handle large datasets (e.g., thousands of reviews) and many users simultaneously. |
| Usability | The frontend should be intuitive, responsive (mobile-friendly), and easy to use for non-technical users |
| Maintainability | The codebase should be modular and well-documented to allow future updates or improvements to the algorithm or UI. |
| Reliability | The system should remain available and correctly function even if one part (e.g., review sentiment) encounters errors. |
| Portability | The application should be deployable across various environments (local, cloud platforms like Vercel/Render). |

* 1. **Technology Stack**

### **Frontend:**

The frontend is developed using React.js, a JavaScript library for building user interfaces. It allows for seamless interaction with the system through features like navigation (using React Router) and HTTP communication with the backend via Axios or the Fetch API. Styling and responsiveness are handled using Tailwind CSS or Bootstrap. Tools like Vite or Create React App are used for managing and bootstrapping the React project efficiently. This setup enables a dynamic and responsive user experience.

### **Backend / Logic:**

The backend is built using Python with FastAPI, a modern and high-performance web framework for building RESTful APIs. Flask can also be used as an alternative. Text processing and NLP tasks such as sentiment analysis and feature extraction are performed using libraries like TextBlob, NLTK, or spaCy. Data handling and preprocessing are managed using Pandas and NumPy. Optionally, scikit-learn may be used for ML-based ranking tasks.

### **Data Storage:**

For storing application data, MongoDB is used when a NoSQL-style document-oriented database is preferred. This approach is particularly beneficial when dealing with flexible or evolving data schemas, commonly found in user-generated content or product reviews.

### **Development Tools:**

Development is supported by tools like Visual Studio Code (VS Code), recommended for both React and Python environments. Postman is used for testing backend APIs. Version control and collaboration are handled using Git and GitHub. Node.js and npm are required to run and manage React.js applications, while Python 3.9+ is necessary for backend development with FastAPI or Flask.

### **Deployment Tools:**

For deployment, the React frontend can be hosted on Vercel or Netlify, offering fast and reliable hosting services. The Python backend (FastAPI or Flask) can be deployed on platforms such as Render, Railway, or Heroku.

**6. Summary**

**This project develops a personalized product recommendation system where users can specify important product features like battery life or camera quality through a simple web form. Users can also choose how much weight to give positive or negative feedback, making the recommendations more tailored to their needs.**

The backend analyzes customer reviews using natural language processing to perform aspect-level sentiment analysis, breaking reviews into sentences and identifying sentiments related to specific features. This detailed approach helps the system understand how different product aspects are perceived.

Using the RANK-ify algorithm, the system matches user preferences with the sentiment data to score and rank products based on how well they meet the user’s criteria. This allows for flexible recommendations that consider both positive and negative opinions.

Finally, the ranked products are displayed clearly on the frontend, with input validation and error handling to ensure a smooth experience even if issues arise.