Content based Restaurant Recommendation System

Keya Dobriyal
Amity University, Noida, INDIA
Author (e-mail: keya@ keyadobriyal.in).
This comprehensive analysis constitutes the project submitted for the bachelor's degree in computer science (Honors in Data Science) at Amity University, Noida, India.

"Smart Dining Decisions — A Content-Based Restaurant Recommendation System for Personalized, Data-Driven Choices."

Abstract

A Restaurant Recommendation System is proposed to address the challenge users face in choosing from the vast and growing number of restaurant options. Leveraging content-based filtering techniques, the system analyzes user preferences, locality, ratings, cuisine, and past experiences to provide personalized suggestions. The primary goal is to save user time and simplify the decision-making process by filtering a large dataset and providing the most relevant restaurant recommendations. Developed using Python libraries and machine learning (including data preprocessing, vectorization, and cosine similarity), the system increases user convenience and business retention rates. Future enhancements include incorporating collaborative filtering and real-time feedback to evolve into a more accurate hybrid recommendation system. This tool offers an efficient solution to the contemporary need for quick and easy decision support in the food industry.

Keywords

Recommender System, Preferences, multi-criteria methods, machine learning, Restaurant Recommendation, locations, Rating

Key Objectives of the Restaurant Recommendation System

The main goal of creating this restaurant recommendation system is to help users discover restaurants tailored to their individual preferences and past interactions. It is designed to offer a personalized, convenient, and efficient experience. This system achieves its objectives by:

- **Saving Time:** It suggests relevant restaurants, eliminating the need for users to manually browse through countless options.
- **Simplifying Decisions:** By filtering and narrowing down widespread choices, it allows users to make well-informed decisions. All key information and attributes are easily accessible for comparison.
- **Enhancing User Satisfaction:** It targets an interactive experience by recommending options most likely to be enjoyed based on the user's currently selected restaurant or cuisine. Ultimately, a successful recommendation system will significantly contribute to higher customer satisfaction and retention rates, addressing a major challenge for businesses in the food service industry.

How a recommendation system works?

The exponential growth of data collection has ushered in a new era driven by data-centric innovation. This vast data is leveraged to design smarter and more efficient systems, among which recommender systems play a key role. Recommender systems function as information-filtering tools that enhance search quality by presenting results most relevant to a user's query or aligned with their past interactions and preferences. There are basic five types of recommender system:

Content-Based Filtering System

A content-based filtering system recommends items similar to those already selected by a user by comparing their features using methods such as cosine similarity or Euclidean distance. In the restaurant recommendation system, attributes like location, rating, and cost are analyzed to suggest similar restaurants. It offers a personalized experience based solely on individual preferences without relying on data from other users.

Collaborative Filtering System

Collaborative filtering predicts user preferences by comparing their activity with others who have similar interests. It can be user–user (recommending what similar users liked) or item–item (suggesting similar items to those a user enjoyed). Though effective, it requires large datasets and struggles with new users or items due to lack of prior information.

Hybrid Recommendation System

A hybrid system integrates multiple techniques, typically combining content-based and collaborative filtering to balance their strengths and weaknesses. Common methods include weighted, switching, and mixed approaches. While more complex and data-heavy, hybrid systems deliver more accurate, diverse, and personalized results.

Knowledge-Based Recommendation System

Knowledge-based systems match user requirements with specific item attributes instead of using past behavior. Ideal for users with clear preferences, they rely on structured item data (e.g., cost, rating, or location) and use rule-based or constraint-based reasoning to generate precise suggestions.

Context-Aware Recommendation System

Context-aware systems consider external factors such as time, location, weather, or mood when generating recommendations. By integrating contextual data with user and item profiles, they provide adaptive and relevant suggestions. However, these systems can be costly to implement and raise privacy concerns due to the extensive data collection required.

Getting Started with Data and Strategy

To build our recommendation system, we first need a robust dataset with key features for analysis. We require details such as restaurant name, rating, cost, and user reviews to identify similarities and make accurate suggestions.

The Data

Dineout is a table booking platform helping customers to do table booking in their favourite restaurants for free and help them get great discounts. The Dineout dataset have been downloaded from Kaggle (https://www.kaggle.com/datasets/arnabchaki/indian-restaurants-2023) and includes thousands of restaurants with attributes such as location data, average rating, number of reviews, cuisine types, etc.

The dataset combines the restaurants from the main Indian cities and I shall be using this dataset for building restaurant recommendation system.

Strategy

My approach will employ Content-Based Filtering to generate personalized restaurant recommendations for each user. I use TF-IDF and Cosine Similarity to calculate similarity between user's description and restaurant reviews and then return a list of restaurants sorted by similarity.

Key Libraries Used in the System

Developing a machine learning project starts by importing Python libraries to provide ready-to-use functions for data processing, text vectorization, and similarity computation, reducing development complexity.

Pandas and NumPy

Pandas is used for data handling and preprocessing, enabling efficient cleaning, organization, and manipulation of datasets in tabular form. **NumPy** supports numerical computations using multi-dimensional arrays and matrices, aiding in feature transformation and similarity calculations.

Scikit-learn

Scikit-learn is central to feature extraction and modeling. It converts text data into numerical form through vectorization and computes restaurant similarity using cosine similarity measures.

Pickle

Pickle is used to serialize and store objects such as models, matrices, and dataframes locally. In this system, it saves the TF-IDF vectorizer, cosine similarity matrix, and preprocessed data, improving efficiency by avoiding repetitive loading.

Streamlit

Streamlit is an open-source Python library for creating interactive web applications. It enables users to search, select, and input preferences while allowing developers to build dynamic dashboards and interfaces with minimal effort.

Methodology

Data Preprocessing

Effective data preprocessing ensures accuracy and reliability in recommendation systems. The main steps followed are:

Removal of Null Values

Missing values were detected using isnull() and sum() functions and appropriately handled to maintain data integrity. Removing nulls ensures consistent feature combination and reliable similarity computation.

Deleting Duplicate Values

Duplicate entries, often caused by data merging or manual errors, were removed using Pandas' drop_duplicates() to prevent bias in similarity calculations and improve computational efficiency.

Selecting Relevant Columns

Only essential features — Name, Cuisine, Location, City, and Locality — were retained to focus on meaningful attributes affecting restaurant similarity and to reduce processing overhead.

Removal of Spaces

Irregular spacing in text fields was standardized using string replacement to ensure consistent categorical values (e.g., “North Indian” and “NorthIndian” treated uniformly).

Text Normalization (Lowercase Conversion)

All text entries were converted to lowercase using .str.lower() to eliminate case-based discrepancies, improving vectorization accuracy and reducing vocabulary redundancy.

Merging Columns

Key text attributes (Cuisine, Locality, City, Location) were concatenated into a single “combined_features” column, creating a comprehensive representation of each restaurant. This facilitates more efficient vectorization and enhances similarity detection during recommendations.

Vectorization: Converting Text to Numerical Data

Vectorization is the process of converting raw, unstructured textual data into numeric vectors that machine learning models can process, all while preserving the data’s original meaning. This conversion is a prerequisite for building computational models, as it facilitates faster computation and provides the necessary input format for algorithms. In this project, we employ the CountVectorizer function from the scikit-learn library.

- **Mechanism:** CountVectorizer converts a collection of text documents (in our case, the “combined_features” column) into a structured matrix where each dimension represents a word’s frequency within a restaurant’s feature set.
- **Application:** This technique encodes textual attributes like cuisine, location, ratings, and cost into vectors.
- **Result:** The resulting numeric vectors are then used to compute similarities between restaurants, forming the essential foundation for the content-based filtering system to identify and suggest comparable options.

Computing Cosine Similarity

After vectorization, cosine similarity is computed to measure the degree of resemblance between every restaurant in the dataset. — **Mechanism:** Cosine similarity calculates the angle between two non-zero vectors in a multi-dimensional space. A higher score (closer to 1) indicates a greater degree of closeness between two restaurants’ vector representations. — **Application:** The vectors are created from the combined textual attributes (cuisine, location, rating, cost). — **Result:** This pairwise computation forms a similarity matrix, where each entry is the similarity score between any two restaurants. This matrix is the central component for the recommendation process. When a user selects a restaurant, its vector is compared against all others in the matrix to find the most highly matched options, enabling personalized dining recommendations.

Frontend Development (Streamlit)

The frontend of the Restaurant Recommendation System is a user-friendly and visually engaging interface built using the Python **Streamlit library**. The frontend serves as the essential platform for user interaction.

- **User Experience:** The design prioritizes interactivity, visual appeal, and ease of use to maximize user engagement and minimize the effort required to find a restaurant.
- **Implementation:** We used standard Streamlit functions like st.title(), st.header(), and st.markdown(). Custom HTML designs were incorporated using st.markdown() to enhance visual attractiveness.
- **User Input:** Interactive widgets are used to collect user input, including:

1. Drop-down menus (st.selectbox()
2. Search bars (st.text_input()
3. Action buttons (st.button()

- **Output:** The interface is responsible for clearly displaying the processed recommendations using functions like st.write() and st.dataframe(). The frontend effectively acts as the bridge between the user and the backend's recommendation logic, making the system accessible and practical.

```
# IMPORTING LIBRARIES
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pickle

# --- DATA PREPROCESSING ---
# Reading dataset (Note: Ensure 'restaurants.csv' is in the same directory)
df = pd.read_csv('restaurants.csv')

# Show table info
print(df.info())

# Printing first 5 rows in dataset
print("\n--- First 5 rows (Original) ---")
print(df.head())

# Checking for null values
print("\n--- Null Value Count ---")
print(df.isnull().sum())

# Deleting any duplicate values
df.drop_duplicates(inplace=True)

# Updating relevant columns
df = df[['Name', 'Cuisine', 'Location', 'City', 'Locality']].copy()
```

```
# Deleting spaces in key columns
print("\n--- Removing Spaces ---")
df['Location'] = df['Location'].astype(str).str.replace(' ', '')
df['Cuisine'] = df['Cuisine'].astype(str).str.replace(' ', '')
df['Locality'] = df['Locality'].astype(str).str.replace(' ', '')

# Print first 5 lines after space removal
print(df.head())

# Converting strings to lowercase and removing leading/trailing spaces
print("\n--- Converting to Lowercase ---")
for col in ['Cuisine', 'Location', 'City', 'Locality']:
    # Ensure the column is treated as string before applying string methods
    df[col] = df[col].astype(str).str.lower().str.strip()

# Print first 5 lines after lowercasing
print(df.head())

# Merge all columns into a single 'combined_features' column
def combine_features(row):
    return f"{row['Cuisine']} {row['Location']} {row['City']} {row['Locality']}"

df['combined_features'] = df.apply(combine_features, axis=1)

# Print first 5 lines after merging
print("\n--- After Merging Features ---")
print(df.head())

# --- VECTORIZATION ---
# Vectorization of combined features using CountVectorizer
cv = CountVectorizer(max_features=5000, stop_words='english')
vectors = cv.fit_transform(df['combined_features']).toarray()

# --- COSINE SIMILARITY ---
# Compute cosine similarity between vectors
similarity = cosine_similarity(vectors)

# Print similarity values for the first restaurant
print("\n--- Similarity values for the first restaurant ---")
# Only print a snippet as the array is large
print(similarity[0][:10])

# --- RECOMMENDATION FUNCTION ---
def recommend(restaurant_name):
    # Find the index of the given restaurant name
    try:
        restaurant_index = df[df['Name'] == restaurant_name].index[0]
```

```
    except IndexError:
        print(f"Restaurant '{restaurant_name}' not found in the dataset.")
        return

    # Get similarity distances for that restaurant
    distance = similarity[restaurant_index]

    # Get top 5 most similar restaurants (excluding itself)
    # enumerate creates (index, distance) pairs
    restaurant_List = sorted(
        list(enumerate(distance)),
        key=lambda x: x[1],
        reverse=True
    )[1:6]

    print(f"\n--- Top 5 recommendations for '{restaurant_name}' ---")
    # Print the names of those restaurants
    for i in restaurant_List:
        print(df.iloc[i[0]]['Name'])

# Testing the function
recommend('Tamasha')
# Note: The raw print statement for index 0 was redundant after defining the function.

# --- SAVING OBJECTS WITH PICKLE ---
print("\n--- Saving objects with pickle ---")

# Save the DataFrame (converted to a dictionary for easier Streamlit use)
# Using 'df' directly is fine, but a dict might be marginally better for web apps
pickle.dump(df.to_dict(), open('restaurant_dict.pkl', 'wb'))

# Save the similarity matrix
pickle.dump(similarity, open('similarity.pkl', 'wb'))

print("Objects saved: 'restaurant_dict.pkl' and 'similarity.pkl'")
```

Frontend code for web application of restaurant recommendation system This code has been written using python in PyCharm

```
# ----------------- IMPORTS -----------------
import streamlit as st
import pickle
import pandas as pd

# ----------------- LOAD DATA -----------------
restaurant_dict = pickle.load(open('restaurant_dict.pkl', 'rb'))
restaurant_df = pd.DataFrame(restaurant_dict)
similarity = pickle.load(open('similarity.pkl', 'rb'))
```

```
# ---------- SESSION STATE ----------
if 'page' not in st.session_state:
    st.session_state.page = 'welcome'

# ---------- RECOMMENDATION FUNCTION ----------
def recommend(restaurant):
    if restaurant not in restaurant_df['Name'].values:
        return []

    restaurant_index = restaurant_df[restaurant_df['Name'] == restaurant].index[0]
    distance = similarity[restaurant_index]
    restaurant_list = sorted(
        list(enumerate(distance)),
        key=lambda x: x[1],
        reverse=True
    )[1:6]

    return [restaurant_df.iloc[i[0]].Name for i in restaurant_list]

# ---------- WELCOME PAGE ----------
if st.session_state.page == 'welcome':
    st.markdown("""
    <div style="text-align: center; padding-top: 80px;">
        <h1 style="font-size: 60px; font-weight: bold; color: #FF6347;">
            🍽 <span style="color:#FF6347;">Welcome to</span>
            <span style="color:#2E8B57;"> Delicious Bites</span>!
        </h1>
        <p style="font-size: 22px; color: #aaaaaa; margin-top: 10px;">
            Experience the taste that makes you smile 🍽
        </p>
    </div>
    """, unsafe_allow_html=True)

    st.markdown("<br><br>", unsafe_allow_html=True)
    col1, col2, col3 = st.columns([3, 1, 3])

    with col2:
        if st.button(" Enter App ", use_container_width=True):
            st.session_state.page = 'main'
            st.rerun()

# ---------- MAIN PAGE ----------
elif st.session_state.page == 'main':
    st.title('🍽 Restaurant Recommender System')

    selected_restaurant_name = st.selectbox(
        'Select a restaurant you like:',
```

```
    restaurant_df['Name'].values
)

if st.button('Recommend'):
    recommendations = recommend(selected_restaurant_name)

    if recommendations:
        st.subheader("You might also like:")
        for i in recommendations:
            st.write(f"🍽️ {i}")
    else:
        st.warning("No recommendations found for the selected restaurant.")

st.markdown("---")

# ---------- CUISINE DASHBOARD ----------
st.header("📊 Restaurant Dashboard by Cuisine")

# Load original dataset for cuisine analysis
restaurant_data = pd.read_csv("restaurants.csv")

# Extract unique cuisines
cuisine_list = set()
for c in restaurant_data['Cuisine'].dropna():
    for item in c.split(','):
        cuisine_list.add(item.strip())

cuisine_list = sorted(cuisine_list)

# Cuisine selector
selected_cuisine = st.selectbox("Choose a Cuisine", cuisine_list)

# Filter function
def filter_by_cuisine(cuisine):
    return restaurant_data[
        restaurant_data['Cuisine'].str.contains(cuisine, case=False, na=False)
    ]

# Apply filter
filtered_df = filter_by_cuisine(selected_cuisine)

# Format cost and rating
if 'Cost' in filtered_df.columns:
    filtered_df['Cost'] = filtered_df['Cost'].apply(
        lambda x: f"₹{x}" if pd.notnull(x) else 'N/A'
    )
if 'Rating' in filtered_df.columns:
    filtered_df['Rating'] = filtered_df['Rating'].apply(
```

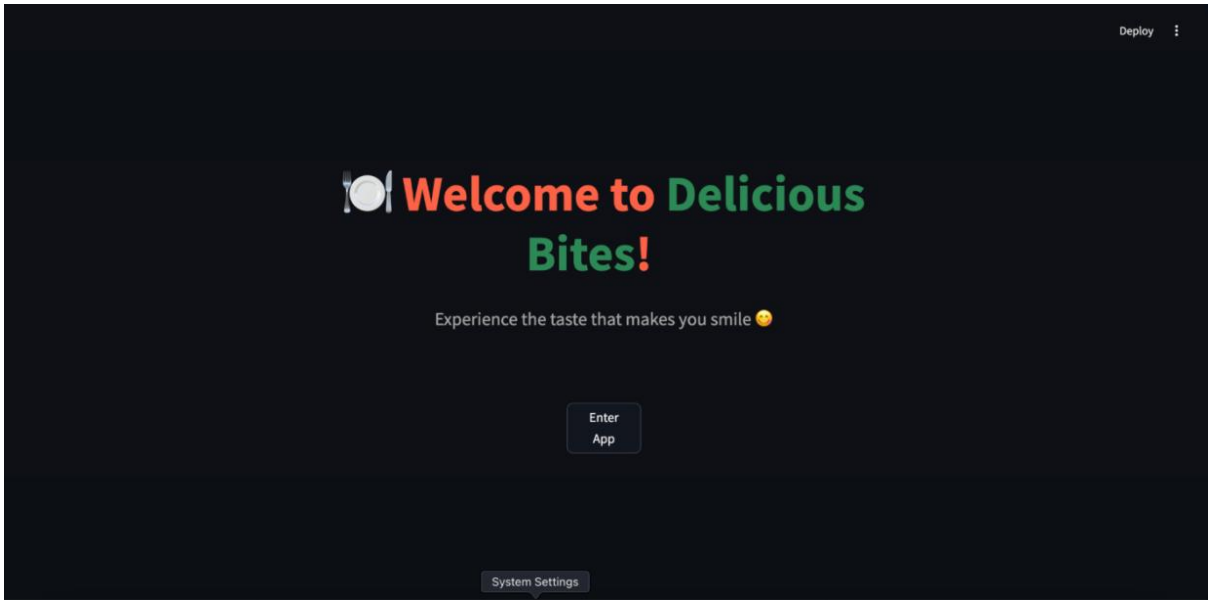
```
        lambda x: round(x, 1) if pd.notnull(x) else 'N/A'
    )

# Display results
st.write(f"### Restaurants offering *{selected_cuisine}* cuisine:")
st.dataframe(
    filtered_df[['Name', 'Cuisine', 'Location', 'City', 'Cost', 'Rating']].reset_index(drop=True)
)

# Exit Button
if st.button("❌ Exit App"):
    st.success("Thank you for visiting Delicious Bites! 🍽️")
    st.stop()
```

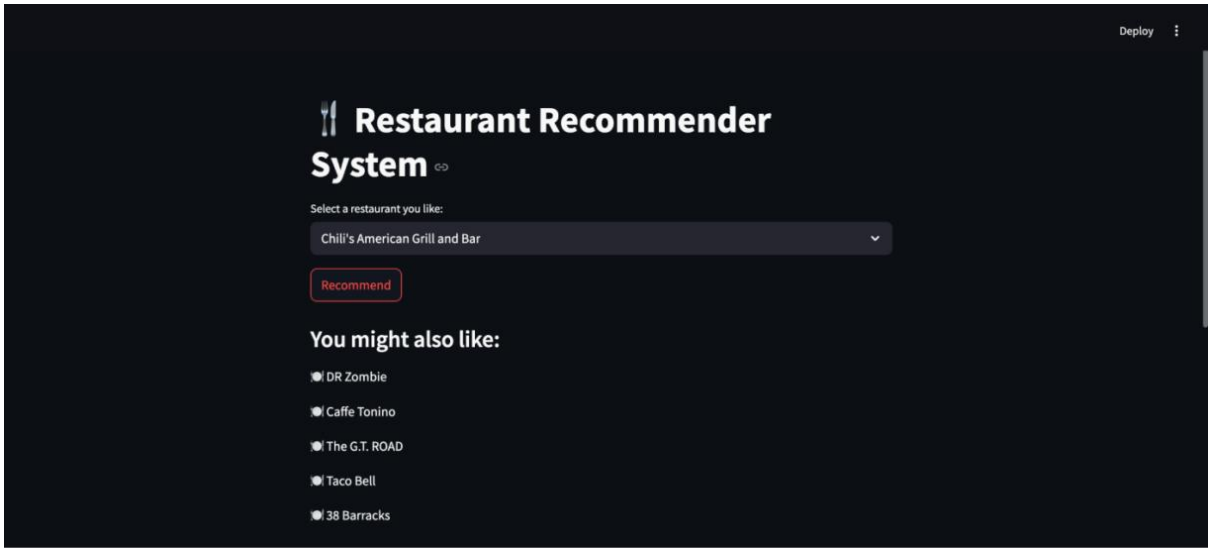
The Result

Press enter or click to view image in full size



Welcome page

Press enter or click to view image in full size



Selection page of the Application

Press enter or click to view image in full size

Recommendation Dashboard

Conclusion

This recommendation system successfully developed an accurate, scalable and user-friendly restaurant recommendations using content-based filtering. The system efficiently addresses challenges faced by both users (decision fatigue, time saving) and businesses (customer retention) by analyzing user preferences such as cuisine, location, and cost, ensuring a reliable and satisfying experience for every user.

Key Achievements:

- **Solves Cold-Start Problem:** Recommendations are provided immediately, even for new consumers with no prior interaction history.
- **Single-User Usability:** It functions effectively without requiring data from multiple users, respecting user privacy regarding personal locations and preferences.

- **Machine Learning Integration:** The platform integrates data preprocessing, vectorization, and cosine similarity computation, all presented through a user-friendly web interface built with Streamlit.
- **Business Impact:** By providing fast and reliable suggestions, it aims to increase customer return rates and boost business growth in the rapidly expanding food industry. The system is a strong foundation that can be scaled and improved in the future by incorporating more advanced techniques like collaborative filtering and real-time feedback to enhance personalization and accessibility, making it an ideal example of leveraging technology to solve everyday issues.

About Author:

Keya Dobriyal is currently pursuing the B.E. degree in Computer Science with Honors in Data Science from Amity University Noida. She has completed few projects in data science, machine learning, deep learning. Data-driven problem solver with a passion for turning raw data into valuable business intelligence, with hands-on academic experience in data mining, predictive modeling, and data visualization. Skilled in leveraging Python and R to extract actionable insights from complex datasets.