Travel Recommendation Engine

Team Infinity

Introduction

Tourism is a rapidly evolving industry, and providing personalised recommendations for travellers can significantly enhance their experience. In a digital age where tourists are presented with overwhelming choices, an efficient recommendation system can help users discover destinations that match their preferences, such as activities they enjoy and their dream destinations. Our Recommendation System is designed to solve this challenge by recommending the top five places to visit based on users' preferences and public ratings.

The primary goal of this recommendation system is to suggest tourist destinations in Sri Lanka based on a user's preferred activities and their bucket list destination. By analysing a dataset that contains information about tourist destinations, such as average ratings, total number of ratings, and activities offered at each location, the system computes a similarity score between the user's preferences and the available destinations. It then combines this score with other key factors, like user ratings and the total number of ratings, to deliver personalised suggestions.

Data Collection and Preprocessing

The **Travel Recommendation Engine** is built using two datasets that contains rich information about various tourist destinations in Sri Lanka. The data sets were provided by rootcode This chapter details the dataset sources, the features we extracted, and how these features are used to generate recommendations.

Data Sources

The data used to enrich the 2nd dataset was obtained from publicly available tourism platforms and open datasets. These sources include:

- **Open datasets**: We leveraged publicly available datasets on tourism in Sri Lanka, which included structured data on locations, activities, and tourist feedback.

The collected data was preprocessed to remove any missing or inconsistent entries, ensuring a clean and usable dataset for our recommendation model. We filled in the missing values of the 2nd data set by using these data sources.

Extracting Data

We extracted users' preferred activities from 'Visitors Preference Dataset' using a python code. Then we extracted activities related to certain places from the latest reviews column using NLP. We used these extracted data to enrich our 2nd dataset(places).

Data Preprocessing

Once the raw data was collected, we undertook several steps to clean and prepare it for use in the recommendation system:

1. **Handling Missing Data**: Any rows with missing critical values such as ratings or activities were either filled or removed, depending on the severity of the missing information.

2. **Cleaning the latest reviews**: We cleaned the last reviews using a python code. There were some garbage values in this column.

```python
def clean_text_with_ftfy(text):
    fixed_text = ftfy.fix_text(text)
    return unidecode(fixed_text)


# Clean all reviews in the 'latest_reviews' column
df2['cleaned_reviews'] = df2['latest_reviews'].apply(clean_text_with_ftfy)
```

3. **Normalisation**: The ratings and total ratings columns were normalised to ensure that they were on the same scale. This allows the system to combine them in a meaningful way without one factor overpowering the other.

```python
# Normalize ratings and total ratings to bring them to the same scale
scaler = MinMaxScaler()
places_df[['Normalized_Average_Rating', 'Normalized_Total_Ratings']] = scaler.fit_transform(places_df[['rating', 'user_ratings_total']])
```

4. **Activity Vectorization**: We used the **TF-IDF (Term Frequency-Inverse Document Frequency)** method to vectorize the activities column. This converts the activities into a numerical format that allows us to compute similarity between user preferences and the activities offered by each destination.

```python
# TF-IDF Vectorizer for Activities
tfidf = TfidfVectorizer(stop_words='english')
activities_matrix = tfidf.fit_transform(places_df['activities'])
```

Conclusion

The data collection process focused on gathering relevant and diverse information about tourist destinations in Sri Lanka. This data serves as the backbone of our recommendation system, ensuring that the suggestions provided are based on rich and meaningful insights. Through careful preprocessing and feature engineering, we transformed the raw data into a format that can be easily integrated into the model for generating recommendations.

Model Explanation

The Travel Recommendation Engine is designed to provide personalised suggestions for tourists based on their preferences for activities and bucket list destinations in Sri Lanka. This chapter outlines the core methodology behind the recommendation model, including the design choices, feature handling, and the final scoring mechanism used to generate the recommended places.

1. Overview of the Model

The model used in the Travel Recommendation Engine is a content-based recommendation system. Unlike collaborative filtering models that rely on user interactions and preferences of similar users, content-based models focus on the attributes of the items (in this case, tourist preferred destinations and preferred activities) and the features that describe them.

The recommendation process involves:

- Matching the user's preferences (e.g., preferred activities) to the available destinations based on the features extracted from the dataset.
- Ranking the destinations by calculating a final score, which combines various factors such as activity similarity, ratings, popularity, and bucket list relevance.
- Returning the top 5 places as recommendations.

2. Model Inputs

The system takes two key inputs from the user:

- **Preferred Activities**: A list of activities the user enjoys (e.g., hiking, sightseeing, swimming). This is the primary factor in matching the user's preferences to the destinations.
- **Bucket List Destination**: A place in Sri Lanka that the user is specifically interested in visiting. This input provides a "boost" to the score of destinations that match the user's bucket list.

```
# Example user inputs
preferred_activities = 'cycling, historical monuments, village homestays'
bucket_list_destination = 'Polonnaruwa, Hatton, Anuradhapura, Ella, Haputale'
```

3. Feature Vectorization

A key component of the model is the transformation of textual data (activities) into numerical form. For this, the system uses **TF-IDF Vectorization (Term**

Frequency-Inverse Document Frequency). TF-IDF is widely used in information retrieval and text mining to weigh the importance of words (in this case, activities) based on their frequency across the dataset.

TF-IDF for Activities

- **Term Frequency (TF)**: Measures how frequently a word appears in a specific document (or place in this case) relative to the total number of words in that document. Frequent words (activities) have higher weights.

$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

- **Inverse Document Frequency (IDF)**: Reduces the weight of common words (activities) that appear in many documents. Activities that are common across multiple destinations (e.g., "sightseeing") will have lower weights, while rarer activities will be given higher importance.

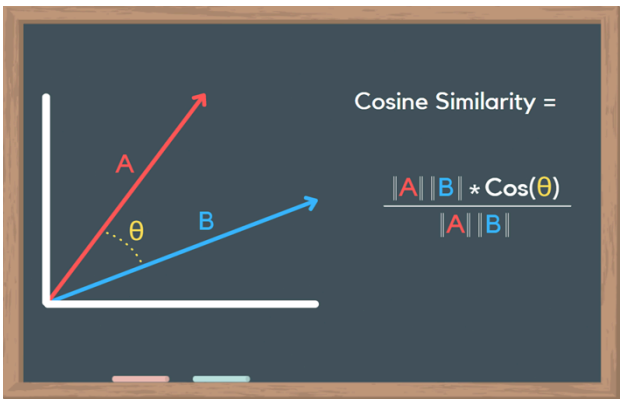
$$TF(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

This vectorization process transforms the **activities** column in the dataset into a matrix of numerical values, where each row represents a place and each column represents an activity. The similarity between user preferences and destination activities is calculated using this vectorized matrix.

4. Similarity Calculation

To determine how well the user's preferred activities match with the available destinations, we calculate the **cosine similarity** between the user's activity vector and the activity matrix of all destinations. Cosine similarity measures the angle between two vectors in multi-dimensional space, with higher similarity scores indicating a better match between user preferences and destination activities.

```
# Calculate the similarity between user preferences and place activities
cosine_sim = cosine_similarity(user_activities_vector, activities_matrix)
```



5. Score Calculation

The model does not rely solely on activity similarity to recommend places. It incorporates several other important features to calculate a **final score** for each destination. This final score is used to rank and recommend the top 5 destinations. The following components are used to calculate the final score:

Similarity Score

This score represents how closely the destination's activities match the user's preferred activities. It is weighted as the most significant factor in the final score.

Similarity Score=Cosine Similarity between the user's activity vector and place's activity

Normalized Average Rating

Destinations with higher average ratings are preferred by more users, so this score rewards places that are highly rated. The rating is normalized to a scale of 0 to 1 to ensure it is comparable with other features.

$$\text{Normalized Average Rating} = \frac{\text{Place Rating} - \text{Min Rating}}{\text{Max Rating} - \text{Min Rating}}$$

Normalized Total Ratings

Destinations with more ratings (popularity) are given a boost to account for their popularity, as places with higher user interactions tend to be more well-known and reliable.

$$\text{Normalized Total Ratings} = \frac{\text{Number of Ratings} - \text{Min Total Ratings}}{\text{Max Total Ratings} - \text{Min Total Ratings}}$$

Bucket List Boost

If the user has specified a destination as their bucket list destination, it receives an additional boost in the final score. This feature ensures that places the user is particularly interested in are prioritized.

```
# Add a boost to places that match the bucket list destination
places_df['Bucket_List_Boost'] = places_df['name'].apply(lambda x: 10 if x == bucket_list_destination else 0)
```

Final Score Calculation

The final score for each place is a weighted combination of the components mentioned above:

Final Score=(Similarity Score×0.5)+(Normalized Average Rating×0.2)+(Bucket List Boost×0.1)+(Normalized Total Ratings×0.2)

We can adjust weights and optimize the Recommendation engine for achieve best performances.

6. Recommendation Generation

Once the final score for each place is calculated, the destinations are sorted in descending order by their final score. The top 5 destinations are then selected and returned as recommendations to the user.

7. Model Design Choices

Several design choices were made to ensure that the recommendation system provides meaningful and personalized results:

- **Content-based approach**: We chose a content-based recommendation system because it allows the model to tailor recommendations directly to the user's input rather than relying on the preferences of other users.
- **TF-IDF Vectorization**: Using TF-IDF to vectorize the activities column ensures that the model can distinguish between common and unique activities, providing more relevant suggestions.
- **Cosine Similarity**: This metric was chosen for its simplicity and effectiveness in measuring the similarity between user preferences and place activities.

- **Feature Weighting**: The final score calculation places greater emphasis on activity similarity but still incorporates other factors like ratings and popularity to ensure a well-rounded recommendation.

8. Limitations of the Model

While the model is effective in generating recommendations based on user preferences, there are some limitations:

- **Cold Start Problem**: New destinations with few or no ratings may not perform as well in the recommendation system.
- **Static User Inputs**: The model does not learn from the user's interactions over time. Future improvements could incorporate feedback loops where user selections can further refine the recommendations.

9. Future Improvements

To further enhance the recommendation system, the following improvements could be considered:

- **Incorporate Collaborative Filtering**: Adding a collaborative filtering component could help the system learn from the preferences of other similar users.
- **Dynamic Recommendations**: Implementing a feedback loop where user interactions are logged could allow the model to adapt and improve recommendations over time.
- **Contextual Recommendations**: Incorporating additional contextual information, such as weather or time of year, could provide even more personalized suggestions.
- **Location Based Recommendation**: Add extra score to places near to bucket list places.To achieve this we can use longitude and latitude.
- **Sentimental Analysis of Reviews**: Use sentimental analysis to identify the user opinions about the places.

5. Evaluation Metrics

Evaluation Overview

To assess the performance of our tourist destination recommendation system, we employed precision@K as the primary evaluation metric. Precision@K measures how well the model's top-K recommendations align with user preferences based on activities and destinations. This approach helps gauge the relevance and quality of the recommendations made by the model.

Precision@K Metric

Precision@K focuses on the proportion of relevant recommendations within the top-K results. For example, if a user prefers certain activities or has a destination on their bucket list, we assess whether the system recommends places that align with these preferences.

The formula for precision@K is:

$$\text{Precision@K} = \frac{\text{Number of Relevant Recommendations in Top K}}{K}$$

Where:

- **K** is the number of recommendations (typically 5).
- **Relevant Recommendations** refer to places that match user preferences (activities or destinations).

Dataset

The evaluation was conducted using a dataset(Visitor Preference Dataset) containing 10,000 user inputs, which include:

- Name
- Preferred Activities
- Bucket List Destinations

For evaluation purposes, the dataset was split into subsets for testing, where the system's recommendations were compared against actual user preferences (activities and destinations).

Evaluation Process

1. **Activities-Based Evaluation**:

    - For each user, the preferred activities were compared with the recommended places' activities.
    - Precision@K was calculated to determine how accurately the system recommended places with relevant activities.
2. **Destination-Based Evaluation**:
    - The system also evaluated recommendations based on the user's bucket list destination.
    - Precision@K was used here to check if the top-K recommendations included the desired destination.
3. **Combined Evaluation**:
    - We also performed a combined evaluation, where both activities and destinations were considered together to assess the overall accuracy of recommendations.

Results

- **Precision@K for Activities**: The model achieved a precision score of , indicating that the majority of the top-K recommendations contained relevant activities.
- **Precision@K for Destinations**: The precision score for destination-based recommendations was , showing that the model effectively suggests destinations aligned with user preferences.

Conclusion

The evaluation demonstrated that the recommendation system provides high-quality suggestions based on user activities and destinations. The model performed well across both activities and destination metrics, validating its effectiveness in assisting tourists with relevant place suggestions.

```python
def precision_at_k(recommended, relevant, k):
    recommended_at_k = recommended[:k]
    hits = len(set(recommended_at_k) & set(relevant))
    return hits / k

def evaluate_precision_at_k(test_data, model, k=5):
    activity_precisions = []
    destination_precisions = []

    for i, row in test_data.iterrows():
        preferred_activities = row['Preferred Activities']
        bucket_list = row['Bucket list destinations Sri Lanka']

        # Get the top K recommended places
        recommended_places = model(preferred_activities, bucket_list)['name'].tolist()

        # Check for activity matches
        relevant_activities = test_data[test_data['Preferred Activities'] == preferred_activities]['Preferred Activities'].tolist()
        activity_precision = precision_at_k(recommended_places, relevant_activities, k)
        activity_precisions.append(activity_precision)

        # Check for destination matches
        relevant_destinations = test_data[test_data['Bucket list destinations Sri Lanka'] == bucket_list]['Bucket list destinations Sri Lanka'].tol
        destination_precision = precision_at_k(recommended_places, relevant_destinations, k)
        destination_precisions.append(destination_precision)
```

```python
# Calculate average precision across all users
avg_activity_precision = np.mean(activity_precisions)
avg_destination_precision = np.mean(destination_precisions)

print(f'Average Precision@{k} for Activities: {avg_activity_precision:.2f}')
print(f'Average Precision@{k} for Destinations: {avg_destination_precision:.2f}')
```

Thank you