Travel Destination Recommendation System (TDRS)

Intelligent Reasoning Systems (IRS) Practice Module

Final Report

Institute of System Science (ISS)

National University of Singapore

Group 1 Member List:

LONG ZHEN A0297168L

ZHOU YUKANG A0296841R

An DONGQI A0296362W

TAO ZHEN A0296066U

Content

Introduction 3

Market Context 4

Market Research 5

Project Scope 6

Data Collection 7

System Design 8

Implementation 9

1. Data Wshing: 10
2. Chatbot: 10
3. Matching Algorithm: 11
4. Front-end Detail 12
5. Back-end Detail 13

Results 14

Challenges 15

Future Work 176

Appendix 177

Introduction

In today's digital age, travelers are constantly seeking personalized experiences to make their journeys memorable. The Travel Recommendation System (TRS) aims to address this demand by leveraging advanced algorithms and data analytic to provide customized travel suggestions. This system analyzes various data points from public dataset, including weather, cost of living index, and city type to offer tailored recommendations for destinations. By integrating existing evaluations and machine reasoning techniques, the TRS adapts to deliver more accurate and relevant suggestions over time. Besides to choosing from preset tags, our system supports natural language input by users. This function enable users to describe the expect destination to include more specific and personalized cases. This project not only enhances user satisfaction but also helps travel businesses optimize their offerings, ultimately contributing to a more engaging and efficient travel planning experience.

Market Context

The travel industry has been undergoing rapid digital transformation, especially with the growing demand for personalized experiences. Travelers today expect tailored suggestions for destinations, accommodations, activities, and restaurants based on their preferences, budgets, and past experiences. This shift has created a significant market for travel recommendation systems.

Enabled by the machine reasoning system and big data technology, travel recommendation systems can now analyze vast amounts of information from various sources (such as reviews, social media, and user behavior) to provide highly personalized recommendations. The travel industry is increasingly adopting such tools to attract and retain customers and propose the best fitted experiences for travelers with evolving needs.

Market Research

Existing TRSs could be summarized into two categories – booking oriented TRS and the mixed TRS. For booking oriented TRSs, we have *Airbnb* (Offer local experiences, tours, and recommendations for things to do in specific locations), *Skyscanner* (A flight comparison tool but also offers recommendations for hotels, car rentals, and travel itineraries), and *Booking.com* (Offers a wide variety of accommodations, from hotels to vacation rentals, along with recommendations for restaurants, activities, and landmarks). These systems provide specific options for transportation, hotel, and restaurants booking. Some of these, such as Airbnb, also provides local tour plan. However, these TRSs merely demonstrate the possible choices. Users still struggle to decide the destination and the plan to pick.

The other category is the mixed TRS. *TripAdvisor* (One of the largest and most popular platforms for travel recommendations. It provides reviews, travel-related content, and suggestions for hotels, activities, and restaurants based on user feedback and ratings) and *Google Travel* (offers trip planning, including recommendations for hotels, flights, attractions, and itineraries based on user data) are two leading systems in this category. These mixed TRSs provides the whole travel plan along with the transportation and hotel recommendations. On the other hand, large-scale platforms disturbed by their magnitude. The overloaded content confuses the users, the travelling plan might not be personalized enough (prioritize sponsored content), and the data shared protocol brings the privacy concerns to users.

To address these problems, we aim to develop a destination-focused recommendation system based on the existing tour evaluation dataset. Satisfaction and personalization are the main performance indicators for our systems. Only adopting existing dataset will not bring any privacy concerns to users. Our system provides:

1. Accurate filtering
2. Personalized recommendations
3. Privacy protection

Project Scope

We aim on providing the most accurate and personalized destination recommendations system according to users' preference and historical travel patterns (if any). Knowledge based reasoning system (knowledge graph/model trained by datasets), big data mining (recommending system), and cognitive techniques (supporting natural language input) are the main approaches to achieve the system. Due to the limited time and scale, we will not integrate the modules that requiring third-party authorization, such as the real-time transportation booking and hotel rental service.

Data Collection

a) Global Air Pollution Datasets

Air Pollution is contamination of the indoor or outdoor environment by any chemical, physical or biological agent that modifies the natural characteristics of the atmosphere. Household combustion devices, motor vehicles, industrial facilities and forest fires are common sources of air pollution. Pollutants of major public health concern include particulate matter, carbon monoxide, ozone, nitrogen dioxide and sulfur dioxide. Outdoor and indoor air pollution cause respiratory and other diseases and are important sources of morbidity and mortality.

b) World's Best Cities for People and the Planet

This dataset contains the index, from global design firm Arcadis and the Centre for Economics and Business Research, ranks cities' success based on social, environmental, and economic factors.

Arcadis used 32 indicators and a cross section of the world's urban areas, so not all capitals or large cities are necessarily represented. A city is scored on each of the three sustainability factors; its overall score is the average of those.

c) Cost of Living

The **Mercer Cost of Living City Ranking** is an annual survey that compares the cost of living in cities worldwide. The **Numbeo** is another comprehensive datasets that record the through out cost of living index from continuous years.

d) OpenWeather

OpenWeather is a platform that provides weather data and forecasts through Application Programming Interfaces (API) for developers, businesses, and organizations. It aggregates and delivers comprehensive weather information from multiple sources, including meteorological stations, radar, and satellite data, allowing users to access real-time weather updates, historical data, and forecasts.

System Design

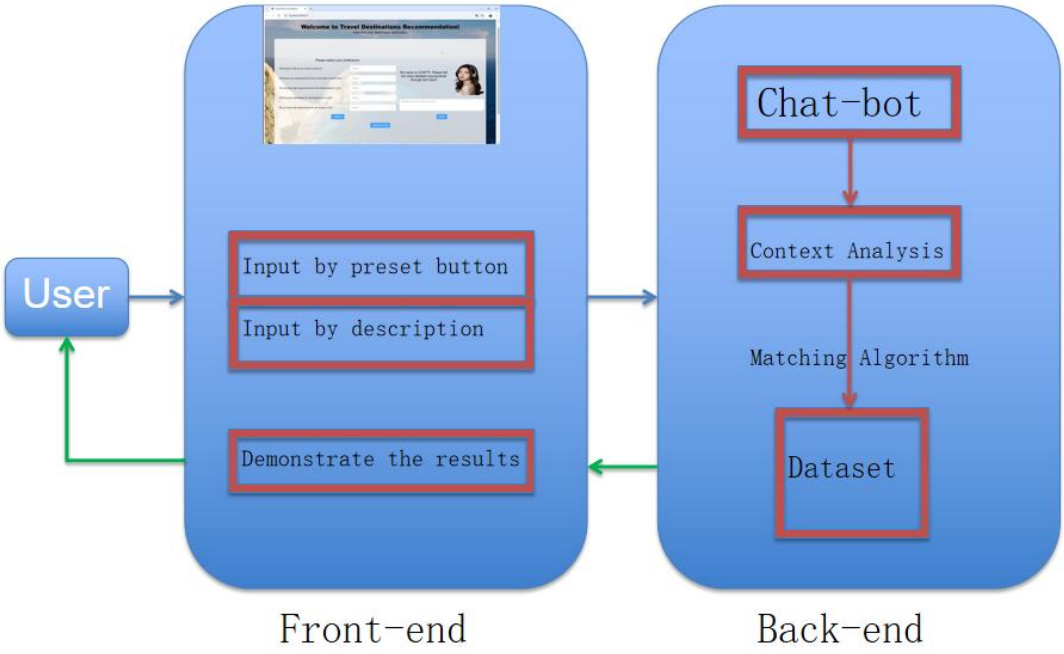

Figure 1

As shown in Figure 1, the front-end is responsible for user interaction, gathering input, and displaying results. The three key steps in this process are:

(1) Input by Preset Button: Users can input their preferences by selecting predefined options via buttons.

(2) Input by Description: Alternatively, users can provide a description of their preferences or needs through text input.

(3) Demonstrate the Results: After processing, the system presents the recommended travel destinations to the user.

The back-end processes user inputs and handles recommendation generation through the following modules:

(1) Chat-bot: This component facilitates interaction between the user and the system. It receives input from the front-end and communicates with the backend logic.

(2) Context Analysis: The input from the chat-bot is analyzed for relevant details about the user's preferences or requirements.
(3) Matching Algorithm: Based on the context analysis, a matching algorithm searches the dataset to find suitable travel destinations that align with the user’s input.
(4) Dataset: This is a repository of information about various travel destinations, which the matching algorithm uses to generate recommendations.

The user flow basically works as follows:

(1) The user interacts with the front-end by either selecting options or typing descriptions.
(2) The input is then passed to the back-end, where the chat-bot facilitates the interaction.
(3) The back-end performs context analysis on the user’s input, and the matching algorithm queries the dataset for suitable recommendations.
(4) The recommendations are sent back to the front-end, where they are displayed to the user.

This system provides a user-friendly interface for recommending travel destinations, with a focus on both structured and unstructured inputs from the user.

Implementation:

1. Data Washing:

(1) The Missing Value

We only extracted intended columns from one dataset and combine them together. It caused missing value when the row number of two datasets is different after merging. For example, the number of total cities is about 400 yet the cities in the cost of living index dataset are only about 200. To solve this problem, we try to use the data from other resources (government official report, Google Place API, e.t.c) to make our own evaluation according to the same evaluation criterion of the original dataset. We cannot yield exact same index as original dataset, still these index are reasonable and usable.

(2) The preprocessing

Due to the limited scale of the project, we decide to use as less dimensions as possible to reduce the running time of the matching algorithm. Therefore, we design some criterion to divide the multiple numerical variables into one categorical variable. For example, the information from Google API contains the number of the bus/subway stations, number of train/flight/ship that arriving and departing and some other related information about the local transportation. We combine these numerical information to decide whether a cities has a good/medium/bad transportation system, and finally record it in the dataset. This step significantly reduces the size of the dataset, and making the further processing much easier and straight forward.

2. Chatbot:

(1) Text Preprocessing

Standardize user input by performing lowercase conversion, abbreviation expansion, punctuation removal, lemmatization, and spell correction. These steps ensure consistency of user input for further processing. We do not remove all the stop words, because they are useful in later semantic analysis.

(2) Keyword Detection

Predefine keywords, including “city”, “cost”, “transport”, “temperature”, “climate”, and “air”. They are converted to word vectors using a spacy model.
The chatbot extracts nouns from the user’s input and compares them with the keywords based on semantic similarity. If the similarity exceeds a threshold, the word is considered relevant, and the chatbot attempts to extract adjectives that modify these nouns for more detailed information.

(3) Context Analysis

Use spacy to get the semantic structure of user input. We divide it into two types:

a) Adjective + noun: In this case, the relationship between key words and adjectives is “amod” or “compound”. Then we add those adjectives to the list of corrosponding key words.
b) Subject + linking verb + adjective: When a noun is identified as the subject, it checks the associated linking verb to find adjectival complements and negations. This helps capture descriptive information, such as “not” and “hot” from the sentence “The city is not hot”.

(4) Conversation Management

User can end the conversation by typing “Bye” or “Enough”. The chatbot will analyze the input to identify missing information and prompt the user to provide additional details until it has gathered sufficient data.

3. Matching Algorithm:

(1) Semantic Analysis:

The user input is processed through semantic analysis, where several key factors are extracted, including: City Type, Temperature, Cost, Transport, and Air Quality. Each of these features is sent for SBERT (Sentence-BERT) Encoding, which transforms the input into a numerical representation for comparison.

Once encoded, Cosine Similarity is calculated to determine how similar the input data is to predefined categories. This is used in the Category Matching process, where relevant city types or categories are identified.

(2) Vector Processing:

This stage involves transforming city features (e.g., temperature, cost, etc.) into vectors. The steps include: Vector Transformation, Encoding Categorical Features, and Scaling Numerical Features. These features are then concatenated into vectors for further similarity comparison.

Two similarity calculations are performed:

X1 Similarity: Based on the SBERT-encoded user input and its semantic analysis.

X2 Similarity: Based on the vectorized city data and user features. The two similarity scores are combined into a Final Similarity Score, which is used to sort and select the top cities most relevant to the user’s input.

(3) Specific Calculating Process:

For X1 Semantic Similarity Calculation:

a) User Input Text is processed using SBERT Encoding.

b) The encoded input is compared to predefined category encodings, and Cosine Similarity is calculated for each dimension (e.g., temperature, cost).

c) For each dimension, the maximum similarity is selected, and a normalized average is calculated. This represents the similarity between the user's preferences and the categories of interest.

For X2 Feature Vector Similarity Calculation:

a) User Features (derived from the user's input) and City Data are vectorized.

b) Once vectorized, Cosine Similarity is computed between the user's features and the city features.

c) The similarities are then normalized for consistency across different feature scales.

Finally, both X1 and X2 scores are combined in the Final Similarity Calculation, where a weighted average is applied to produce the overall similarity score for each city. This final score helps in ranking cities according to how well they match the user's preferences.

4. Front-end Detail (by Vue API):

(1) Collect User Operations:

a) The front-end gathers user operations or preferences from the input interface.

b) These collected options are then sent to the back-end server via an API call.

(2) Collect User Messages:

a) The front-end collects user messages or queries, which are also sent to the back-end through API requests.

(3) Receive Robot Reply:

a) The replies generated by the chatbot on the back-end are received by the front-end.

b) These replies are then displayed to the user in the chat interface.

(4) Receive Recommendation Options:

a) The front-end receives recommendation options from the back-end.
b) Similar to the robot replies, these recommendations are displayed to the user, assisting them in making informed decisions.

5. **Back-end Detail (by Django API):**

(1) Receive User Operations:

a) The back-end receives user operations sent from the front-end.
b) It updates the city information based on these options to tailor the chatbot's responses and recommendations.

(2) Receive User Messages:

a) User messages from the front-end are received and used to update city information.
b) This helps the back-end maintain an accurate context for generating meaningful responses.

(3) Generate Robot Reply:

a) The back-end processes the incoming data and generates a suitable reply from the chatbot.
b) This reply is sent back to the front-end to be displayed to the user.

(4) Recommendation Options:

a) Based on the processed user input, the back-end generates recommendation options.
b) These recommendations are sent to the front-end to guide the user in their decision-making process.

Results (illustrated in screenshot):

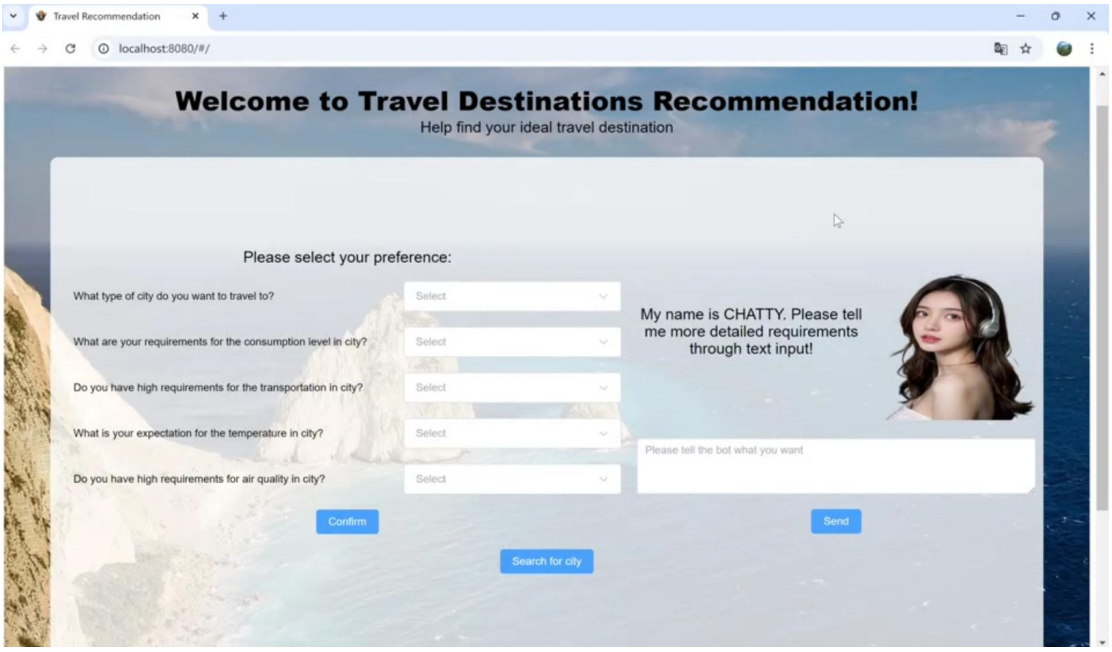

Figure 2 The main user interface

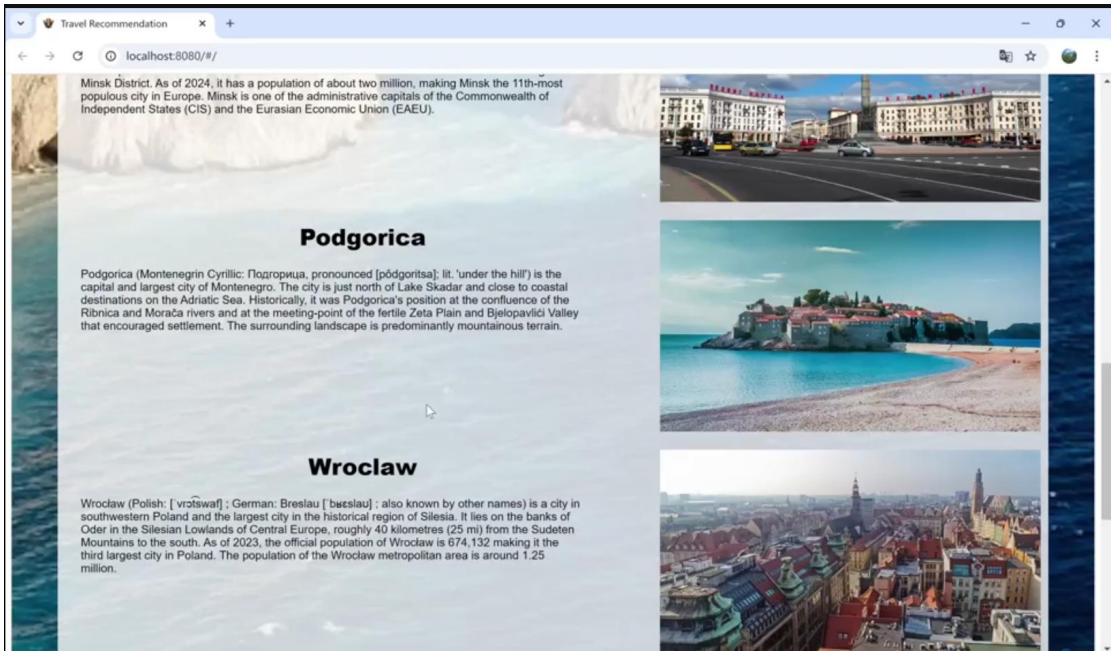

Figure 3 The recommendation results

Challenges:

1. The Limited Parameter: In this project, we design a system in a relatively small scale and use the vector match algorithm to find the most appropriate result. However, the vector match algorithm we adopted will be affected if the input parameter is too large. Therefore, we try to use the categorical data as much as possible to improve the computation speed. We adopt normalization for those numerical data first, and then convert them into 3 different categories. Finally, we yield a smaller dataset and keep the algorithm fast.

2. The Semantic Analysis: The main problem is about the inconsistency and difficulties in contextual understanding. The lack of sufficient contextual information in the data hindered the system's ability to interpret complex sentences or passages accurately. In such cases, the model often misinterpreted information, generating inaccurate semantic structures that affected subsequent processing and analysis. To address these challenges, we preprocessed the corpus, including steps like tokenization and stop-word removal, to reduce semantic noise. Finally, we enhanced the system's understanding of complex sentences. These solutions significantly improved processing efficiency, reduced semantic inconsistency issues, and increased the overall accuracy of results.

Future Work:

Admittedly, this system could be improve in some aspects due to the limit time and resources.

The first possible improvement is the scale of the dataset. The vector match algorithm takes long time to process the calculation if there are many classes to match. Therefore, we must scarify the number of classification of each column. For example, at first we plan to use at least 7 categories to indicate the climate, but we eventually only adopt 3 instead. This cut down undoubtedly reduces the accuracy of the algorithm. In the future, one can get more specific and personalized result by enlarging the categories in the dataset.

The second improvement could be found in the demonstrating stage. Due to the limited time, we now only offer the destinations alone. In the future, we can add on some related links to supply the recommendations, including different types of related travel blog, flight/train/ship/car booking information, and local hotel information to provide the one-stop services.

Appendix:

1. **Project Proposal**

Date of Proposal: 20 September, 2024
Project Title: Travel Destination Recommendation System
Background/Aims/Objective: The travel industry has been undergoing rapid digital transformation, especially with the growing demand for personalized experiences. Travelers today expect tailored suggestions for destinations, accommodations, activities, and restaurants based on their preferences, budgets, and past experiences. This shift has created a significant market for travel recommendation systems. Enabled by the machine reasoning system and big data technology, travel recommendation systems can now analyze vast amounts of information from various sources (such as reviews, social media, and user behavior) to provide highly personalized recommendations. The travel industry is increasingly adopting such tools to attract and retain customers and propose the best fitted experiences for travelers with evolving needs.
Requirement Overview: 1. Data collection and data washing 2. System integration 3. Making reasonable recommendation
Resource Requirement (Hardware, Software, and others): Hardware: Personal Laptop Software: 1. Front-end: Vue 2. Back-end: Django

Number of Member:

A team of 4 people

Long Zhen: Model training and Algorithm design

An Dongqi: Front-end development and system integration

Zhou Yukang: Back-end development and Chatbot implementation

Tao Zhen: Data preparing and reporting editing

2. Functionalities and Techniques Map

The Whole Recommendation System: Big data mining techniques, RS module

The chatbot that supports the natural language input by users: System designed with cognitive tools, CGS

The Vector match algorithm: Knowledge based techniques in decision automation, MR

3. User Guide

>git clone
https://github.com/ffffffklj/IRS-PM-2024-08-26-IS02PT-GRP-RushB-Travel-Destinations-Recommendation-System-TDRS-.git

> open files in the following path:

>IRS-PM-2024-08-26-IS02PT-GRP-RushB-Travel-Destinations-Recommendation-System-TDRS- --> SystemCode --> travelRecommendation

> install necessary libraries and RUN

> open files in the following path:

>IRS-PM-2024-08-26-IS02PT-GRP-RushB-Travel-Destinations-Recommendation-System-TDRS- --> SystemCode --> recommendationTravel

> npm install

> npm run serve

> Go to URL using web browser http://localhost:8080/