دانشگاه اصفهان
دانشکده مهندسی کامپیوتر

مستند پروژه پیشنهاد تور گردشگری (FOL)

درس مبانی و کاربردهای هوش مصنوعی

دکتر کارشناس

اعضای گروه (۶):
امیرعلی لطفی (۴۰۰۳۶۱۳۰۵۳)
متین اعظمی (۴۰۰۳۶۲۳۰۰۳)
زهرا معصومی (۴۰۰۳۶۲۳۰۳۴)

زمستان ۱۴۰۲

فهرست مطالب

آماده‌سازی و خواندن دیتاست‌ها از فایل

در این بخش، فایل‌های مورد نیاز را با استفاده از کتابخانه Pandas می‌خوانیم و در دیتافریم‌ها قرار می‌دهیم.

آماده‌سازی دیتاست Destinations

```
dest_df = pd.read_csv('Destinations.csv')
dest_df.head(10)
```

	Destinations	country	region	Climate	Budget	Activity	Demographics	Duration	Cuisine	History	Natural Wonder	Accommodation	Language
0	Tokyo	Japan	East Asia	Temperate	High	Cultural	Solo	Long	Asian	Modern	Mountains	Luxury	Japanese
1	Ottawa	Canada	North America	Cold	Medium	Adventure	Family-friendly	Moderate	European	Modern	Forests	Mid-range	English
2	Mexico City	Mexico	North America	Temperate	Low	Cultural	Senior	Short	Latin American	Ancient	Mountains	Budget	Spanish
3	Rome	Italy	Southern Europe	Temperate	High	Cultural	Solo	Moderate	European	Ancient	Beaches	Luxury	Italian
4	Brasilia	Brazil	South America	Tropical	Low	Adventure	Family-friendly	Long	Latin American	Modern	Beaches	Budget	Portuguese
5	Canberra	Australia	Oceania	Temperate	High	Adventure	Family-friendly	Long	Western	Modern	Beaches	Luxury	English
6	New Delhi	India	South Asia	Tropical	Low	Cultural	Solo	Long	Asian	Ancient	Mountains	Budget	Hindi
7	Pretoria	South Africa	Africa	Temperate	Medium	Adventure	Solo	Moderate	African	Modern	Forests	Mid-range	English
8	Madrid	Spain	Southern Europe	Temperate	Medium	Cultural	Senior	Short	European	Ancient	Beaches	Mid-range	Spanish
9	Moscow	Russia	Eastern Europe	Cold	Low	Cultural	Senior	Long	Eastern European	Medieval	Mountains	Budget	Russian

در هنگام استفاده از اطلاعات این دیتاست، کلمات و حروف بزرگ را به حروف کوچک و فاصله‌ها را به "_" تبدیل می‌کنیم. همچنین بقیه کاراکترهای غیر مجاز را از رشته حذف می‌کنیم تا هنگام استفاده از Prolog به مشکل نخوریم.

ذخیره مقادیر دیتاست

در این قسمت از کد، هر کدام از مقاصد را به ویژگی‌های آن‌ها نگاشت می‌کنیم:

```
features = list(map(sanatize, dest_df.columns))
dest_features = {}
values = dest_df.values
for val in values:
```

```
    dest_features[sanatize(val[0])] = {}
    for i in range(1, len(features)):
        dest_features[sanatize(val[0])][features[i]] =
sanatize(val[i])
```

آماده‌سازی دیتاست مجاورت مقاصد

در data frame مربوط به این دیتاست، همسایه‌های هر شهر مشخص می‌شود:

```
map_df = pd.read_csv('Adjacency_matrix.csv',
index_col="Destinations")
map_df.head()
```

Destinations	Tokyo	Ottawa	Mexico City	Rome	Brasilia	Canberra	New Delhi	Pretoria	Madrid	Moscow	...	Bursa	Munich	Hamburg	Fra
Tokyo	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
Ottawa	0	0	0	0	0	0	0	0	0	0	...	1	1	0	
Mexico City	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
Rome	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
Brasilia	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

پردازش متن کاربر یا کلمات دیتافریم‌ها

در این بخش، کلمات و متن‌ها را بررسی کرده و در صورت وجود ویژگی‌ای که مانع ادامه الگوریتم یا باعث ایجاد مشکل در کار با Prolog می‌شود را از بین می‌بریم. همچنین برای کلمات هم‌معنی‌هایشان را پیدا و ذخیره می‌کنیم تا هنگام پردازش متن ورودی کاربر، کارایی و عملکرد بهتری را شاهد باشیم.

پیدا کردن کلمات هم‌معنی و مرتبط

در این تابع، با استفاده از کتاب‌خانه پردازش متن NLTK کلمات معادل و مرتبط با کلمه ورودی تابع را پیدا می‌کنیم.

```
def find_related_words(word):
    synonyms = set()
    synonyms.add(word)
    for syn in wordnet.synsets(word):
```

```
            for lemma in syn.lemmas():
                synonyms.add(lemma.name().lower())

    return synonyms
```

با استفاده از این تابع، کلمات هم‌معنی با مقادیر ستون‌های دیتافریم Destinations را پیدا کرده و ذخیره می‌کنیم:

```
categories = {}
for col in dest_df.columns:
    categories[sanatize(col)] = find_related_words(sanatize(col))
    categories[sanatize(col)].add(sanatize(col))

informations = {}
for col in dest_df.columns:
    informations[sanatize(col)] = {}
    for val in dest_df[col]:
        informations[sanatize(col)][sanatize(val)] = {}

for category in informations:
    for val in informations[category]:
        informations[category][val] = find_related_words(val)
```

پیدا کردن کلمات مهم متن

به کمک توابع زیر و با استفاده از کتاب‌خانه NLTK حروف اضافه و علامت‌های نگارشی و ... را حذف می‌کنیم تا متن ورودی صرفا شامل کلمات کلیدی باشد.

```
def delete_punctuation(text):
    # Remove punctuation using the string module
    translator = str.maketrans('', '', string.punctuation)
    text_without_punct = text.translate(translator)
    return text_without_punct

def delete_stopwords(text):
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(text)
    return [word for word in word_tokens if word.lower() not in
```

```
stop_words]

def keywords_text(text):
    return delete_stopwords(delete_punctuation(text))

def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(text)
    filtered_words = [word for word in words if word.lower() not in
stop_words]
    return ' '.join(filtered_words)
```

آماده‌سازی رشته ورودی

در این تابع، یک رشته دریافت کرده و آن را به فرم نرمال (حالتی که باعث ایجاد تداخل با قوانین Prolog نشود) تبدیل می‌کنیم:

```
def sanatize(string):
    return string.lower()
                 .replace(" ", "_")
                 .replace("-", "_")
                 .replace('\'', '')
                 .replace('.', '')
                 .split(',')
```

محاسبه تشابه ویژگی‌های استخراج شده با یک مقاصد

در این تابع کمکی، با استفاده از تشابه ویژگی‌های استخراج شده از متن و ویژگی‌های یک موقعیت، امتیازی برای تشابه آن با ویژگی‌های موجود در دیتاست محاسبه می‌کنیم:

```
def point_location(location, features):
    point = 0
    for feature in features:
        if dest_features[location][feature] in features[feature]:
            point += 1
    return point
```

ساخت پایگاه دانش

در این بخش، با استفاده از اطلاعات دیتاست‌ها پایگاه دانش را می‌سازیم.

ایجاد پایگاه دانش مربوط به مقاصد

با استفاده از روش یکپارچه، پایگاه دانشی برای ثبت ویژگی‌های هر شهر ایجاد می‌کنیم. ابتدا تمام مقادیر قبلی که احتمالاً وجود دارند را پاک کرده و سپس ویژگی‌های موجود در دیتافریم مربوط به مقاصد را به عنوان fact وارد پایگاه دانش می‌کنیم. همچنین برای جلوگیری از ایجاد خطا در دستورات Prolog، اطراف هر کلمه کوتیشن (' ') قرار می‌دهیم.

```
prolog.retractall("destination(_, _, _, _, _, _, _, _, _, _, _, _, _,
_)")
assertions = []
for data in dest_df.values:
    assert_str = ''
    for i in range(len(data)):
        if i != len(data) - 1:
            assert_str += '\'' + sanatize(data[i]) + '\', '
        else:
            languages = data[i].split(", ")
            for lang in languages:
                temp = assert_str
                temp += '\'' + sanatize(lang) + '\''
                assertions.append(temp)

for assert_str in assertions:
    prolog.assertz(f"destination({assert_str})")
```

در این بخش، زبان‌های مربوط به هر مقصد را پیدا کرده و به ازای هر کدام از آن‌ها یک رشته assertion می‌سازیم.

ایجاد پایگاه دانش مربوط به ارتباط مقاصد

با استفاده از اطلاعات موجود در دیتافریم مربوط به ارتباط شهرها، روابط موجود بین هر دو شهر را به عنوان fact وارد پایگاه دانش می‌کنیم. در این بخش دو نوع رابطه داریم: رابطه مستقیم

(directly_connected) و رابطه دو طرفه (connected) بین شهرها که با استفاده از دو قانون رابطه یک طرفه ایجاد و بررسی می‌شود:

```
prolog.retractall("directly_connected(_, _)")
prolog.retractall("connected(_, _)")

all_cities = map_df.index
visited = set()

for ct1 in all_cities:
    for ct2 in all_cities:
        if ct2 == ct1:
            continue

        if map_df[ct1][ct2] and (ct1, ct2) not in visited:
            prolog.assertz(f"directly_connected('{sanatize(ct1)}',
'{sanatize(ct2)}')")

            visited.add((ct1, ct2))
            visited.add((ct2, ct1))

prolog.assertz("connected(X, Y) :- directly_connected(X, Y)")
prolog.assertz("connected(X, Y) :- directly_connected(Y, X)")
```

وجود ارتباط بین شهرها

در تابع **check_connections** شهرهای دارای ویژگی‌های استخراج شده را از خروجی query به دست می‌آوریم.

```
def check_connections(self, results):
        locations = []
        for result in results:
            city = result["City"]
            locations.append(city)
        return locations
```

توابع کلاس App

توابعی که در این کلاس پیاده‌سازی شده‌اند، برای دریافت متن از کاربر، پردازش آن و استخراج ویژگی‌های مورد نظر او استفاده می‌شوند. در نهایت تابع ارزیابی را داریم که با توجه به ویژگی شهرها امتیازی برای ارزیابی یک مسیر محاسبه می‌کند.

تابع process_text

در این تابع، ابتدا متن ورودی کاربر را دریافت کرده و بعد، ویژگی‌های مورد نظر آن کاربر (آن‌هایی که در ستون‌های دیتاست ما نیز موجود است) را استخراج می‌کنیم. در تمام قسمت‌ها، با استفاده از تابع sanatize، کلمات را به فرم نرمال تبدیل می‌کنیم.
در صورت عدم وجود ویژگی (ویژگی‌های دیتاست)، یک "_" قرار می‌دهیم و با استفاده از آن‌ها، رشته کوئری را ایجاد می‌کنیم. سپس با prolog به پایگاه دانش کوئری می‌زنیم و شهرهای منطبق با آن ویژگی‌ها را دریافت می‌کنیم.
بعد، همه مسیرها را پیدا کرده و بهترین آن‌ها را انتخاب می‌کنیم.

```
  def process_text(self):
        """Extract locations from the text area and mark them on the
map."""
        text = self.text_area.get("1.0", "end-1c")  # Get text from
text area
        features = self.extract_features(text)  # Extract locations
(you may use a more complex method here)

        print(f"{features=}")

        properties = ""
        for i, col in enumerate(dest_df.columns):
            if i == 0:
                continue
            properties += f'{list(features[sanatize(col)])[0] if
sanatize(col) in features else "_"}'

            if i != len(dest_df.columns) - 1:
```

```
            properties += ", "
    query = f"destination(City, {properties})"

    print(f"{query=}")

    results = list(prolog.query(query))
    locations = self.check_connections(results)
    print(f"{location}")

    tours = get_tours(locations[0:4])
    best_path = self.evaluate(tours, features)
    print(best_path)
    self.mark_locations(best_path[0])
```

استخراج ویژگی‌ها

جمله حاصل از پردازش‌های پیدا کردن کلمات مهم متن به این تابع داده می‌شود تا با استفاده از دیکشنری‌های category و informations که در قبل توضیح داده شد کلمات مربوط به هر ویژگی را استخراج کنیم.

```
def extract_features(self, text):
    keywords = keywords_text(text)
    features = {}
    for category in categories:
        for value in categories[category]:
            if value in keywords:
                features[category] = set()

    for category in features:
        for value in informations[category]:
            for related in informations[category][value]:
                if related in keywords:
                    features[category].add(value)
```

تابع ارزیابی

در این تابع برای ارزیابی مسیرها و در نتیجه پیدا کردن بهترین مسیر تور، امتیاز هر شهر در یک مسیر ممکن را با توجه به ویژگی‌هایش تعیین می‌کنیم و با مقایسه امتیازها، بهترین مسیر را انتخاب و برمی‌گردانیم.

```
def evaluate(self, paths, features):
        best_path = ("", 0)
        for path in paths:
            point = 0
            print(f"path = {path}")
            arr_path = [x[1:-1] for x in path.split(', ')]
            for city in arr_path:
                point += self.point_location(city, features)
            if best_path[1] <= point:
                best_path = (arr_path, point)
        return best_path
```

تابع پیشنهاد تور

در این تابع، تعدادی شهر را دریافت کرده و تورهای مناسب را برمی‌گردانیم.
ابتدا شرط بیان شده برای تعداد شهرها را بررسی می‌کنیم که اگر بیشتر از ۵ است خطا برگردانیم که کاربر ویژگی‌های خاص‌تری را وارد کند و اگر ۰ بود خطای این‌که شهری با استفاده از ویژگی‌های استخراج شده پیدا نشده و اطلاعات بیشتری داده شود.

```
def get_tours(cities):
    if len(cities) == 0:
        raise Exception("No cities were found. Enter a more specific
input.")

    if len(cities) > 5:
        raise Exception("Too many cities were found. Enter a more
specific input.")

    paths = []
```

```
    if len(cities) == 5:
        for c in cities:
            paths.extend(trial([x for x in cities if c != x]))
    else:
        paths.extend(trial(cities))

    return replace_variables(paths)
```

این تابع عملکرد و هدف کلی برنامه را اجرا می‌کند. توابعی که در این تابع کلی استفاده شده‌اند، شامل موارد زیر می‌شود:

امتحان و آزمایش وجود مسیر بین شهرها

این تابع لیستی از شهرها را می‌گیرد و سعی می‌کند مسیری بین تعداد بیشتری از شهرهای ممکن پیدا کند. اگر مسیر پیدا نشود، هربار یک شهر را کنار می‌گذارد و سعی می‌کند بین شهرهای موجود مسیر پیدا کند.

زمانی‌که برای بار اول مسیر پیدا شد پترن‌های ممکن مسیر و مقادیر ممکن در آن‌ها را خروجی می‌دهیم.

```
def trial(cities):
    varc = 0
    removec = 0

    total_ok_paths = []

    while len(cities) - removec > 0:
        if removec == 0:
            ok_paths = check_cities(cities)

            if len(ok_paths) > 0:
                total_ok_paths.append(ok_paths)
        elif removec == 1:
            for c in cities:
                ok_paths = check_cities([city for city in cities if
city != c])
```

```
                if len(ok_paths) > 0:
                    total_ok_paths.append(ok_paths)
        elif removec == 2:
            for c1 in cities:
                for c2 in cities:
                    if c1 == ct2:
                        continue

                    ok_paths = check_cities([city for city in cities
if city != c1 and city != c2])

                    if len(ok_paths) > 0:
                        total_ok_paths.append(ok_paths)

        if len(total_ok_paths) > 0:
            return total_ok_paths

        removec += 1

    return []
```

بررسی امکان ایجاد مسیر

این تابع سعی می‌کند در ادامه تابع قبل بررسی کند برای هر تعداد شهر ممکن با چند متغیر بین آن‌ها می‌توانیم مسیری ایجاد کنیم.

```
def check_cities(cities):
    varc = 0
    total_ok_paths = []

    while varc + len(cities) <= 4:
        ok_paths = check_cities_varc(cities, varc)

        if len(ok_paths) > 0:
            total_ok_paths = ok_paths

        if len(total_ok_paths) > 0:
```

```
            return total_ok_paths

        varc += 1
    return []
```

بررسی جایگشت‌های ممکن

این تابع جایگشت‌های مختلف شهرها و متغیرهارا می‌سازد تا تمام حالت‌های ممکن برای مسیرها بررسی شود.

```
def get_permutation(cities, varc = 0):
    res = []
    if len(cities) == 1:
        return [cities]

    for i in range(len(cities)):
        for j in range(len(cities)):
            if i == j:
                continue

            src = cities[i]
            dest = cities[j]
            cnt = len(cities)

            rem = [x for xi, x in enumerate(cities) if xi != i and xi
!= j]

            if cnt == 2:
                if varc == 0:
                    res.append((src, dest))
                elif varc == 1:
                    res.append((src, 'X1', dest))
                elif varc == 2:
                    res.append((src, 'X1', 'X2', dest))
            if cnt == 3:
                if varc == 0:
                    res.append((src, dest))
                elif varc == 1:
```

```
                res.append((src, 'X1', rem[0], dest))
                res.append((src, rem[0], 'X1', dest))
        if cnt == 4:
            if varc == 0:
                res.append((src, rem[0], rem[1], dest))
                res.append((src, rem[1], rem[0], dest))


    return res
```

کوئری زدن در پایگاه دانش برای یافتن مقدار متغیرها

به کمک این تابع حالت‌های پترن مسیر که در توابع قبل ساخته شده بود را در پایگاه دانش query می‌زنیم تا مقادیر ممکن برای متغیرهای مسیر پیدا شود و خروجی داده شود.

```
def check_cities_varc(cities, varc):
    paths = get_permutation(cities, varc)

    ok_paths = []
    for p in paths:
        res = check_connection(p)
        if res is not None:
            ok_paths.append((p, res))


    return ok_paths


def check_connection(cities):
    if len(cities) <= 1:
        return [{}]

    prev = None
    query = ""
    for i, ct in enumerate(cities):
        if i == 0:
            prev = ct
```

```
            continue

        query += f"connected({prev}, {ct})"
        prev = ct
        if i != len(cities) - 1:
            query += ", "

    result = prolog.query(query)

    res_list = list(result)
    if result_exists(res_list):
        return res_list

    return None
```

کتابخانه‌های مورد استفاده

```
import sys
import tkinter
import tkinter.messagebox
from tkintermapview import TkinterMapView

import pandas as pd

from pyswip import Prolog

from nltk.corpus import wordnet
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk import download as nltkdl
import string
```

منابع

- https://www.swi-prolog.org/