第9章　程式

9-1 元件清單及其規格描述

▼表9-1-2 元件清單及其規格描述表(後端)

|  |  |  |
| --- | --- | --- |
| Python檔案(後端)－AI | | |
| 編號 | 檔案名稱 | 功能 |
| 2-6-1 | house\_similarity.py  house\_detail.py | 房屋的地址、坪數、設施等文字性相似度比較。 |
| 2-6-2 | Yolo.py  Yolo+clip.py | 房屋圖片的物件偵測與相似度比較。 |
| 2-6-3 | ner\_clean\_1.py  nearby\_ORG\_cleaned\_2.json | 提取房屋描述內有關附近生活機能的店家或設施。 |
| 2-6-4 | subway\_bus\_ckip.py  subway\_and\_bus.json | 提取房屋附近的捷運站及公車站，供neo4j資料庫使用。 |
| 2-6-5 |  |  |
| 2-6-7 |  |  |
| 2-6-8 |  |  |
| 2-6-9 |  |  |
| 2-6-10 |  |  |
| 2-6-11 |  |  |
| 2-6-12 |  |  |
| 2-6-13 |  |  |
| 2-6-14 |  |  |

9-2 其他附屬之各種元件

▼表9-2-1 部分程式碼－house\_detail.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 2-6-1 | 程式名稱 | house\_detail.py |
| 目的 | 房屋的地址、坪數、設施等文字性相似度比較。 | | |
| 部分程式碼 | | | |
| import json  import torch  import torch.nn.functional as F  from ckiptagger import WS #, POS, NER  from transformers import BertTokenizer, BertModel  import re  # 初始化 WS (中文斷詞工具)、BERT 模型 (中文預訓練語言模型)  ws = WS("D:\\AI\_data\\data")  tokenizer = BertTokenizer.from\_pretrained('bert-base-chinese')  model = BertModel.from\_pretrained('bert-base-chinese')  def取得文本的 BERT 向量(text):  # 將文本進行分詞並轉換為張量  inputs = 文本進行分詞並轉換為張量:  outputs = 文本的向量表示  return 回文本的向量表示  def載入 JSON 數據:  with 載入 JSON 文件:  return json.load(file)  def獲取地址的 BERT 向量:  for item in items:  address\_text =地址組合成字符串  address\_tokens = 進行分詞  item[分詞後的地址] = 轉換為 BERT 向量  def計算餘弦相似度(向量1, 向量2):  # 計算並返回兩個向量的  return 餘弦相似度(向量1.unsqueeze(0), 向量2.unsqueeze(0)).item()  def比較設備相似度(設備1, 設備2):  common\_devices =找出設備1和設備2中設備名稱的交集  matched =設備1中: 對於設備2 中:如果設備等於設備2且 皆可提供  : matched += 1  return if matched / min(len(設備1), len(設備2)) 超過 70%: return True  def比較房型的相似度(房型1, 房型2):  tokenized\_房型1 = 分詞([房型1])  tokenized\_房型2 = 分詞([房型2])  emb1 = 取得 BERT 向量(' '.join(tokenized\_房型1[0]))  emb2 = 取得 BERT 向量(' '.join(tokenized\_房型2[0]))  if相似度超過 0.9，返回 True  def比較樓層的相似度(樓層1, 樓層2):  layers1 = 樓層1.分段('/')  layers2 = 樓層2.分段('/')  similarities = []  for l1 in layers1:  for l2 in layers2:  tokenized\_樓層1 = 分詞([l1])  tokenized\_樓層2 = 分詞([l2])  emb1 = 取得 BERT 向量(' '.join(tokenized\_樓層1[0]))  emb2 = 取得 BERT 向量(' '.join(tokenized\_樓層2[0]))  similarities.append(cosine\_similarity(emb1, emb2))  return最高的相似度，並判斷是否超過 0.9  def查找相似的數據項目(data):  similar\_items = []  n = len(data)  # 遍歷數據中的每個項目進行兩兩比較  for i in range(n):  for j in range(i + 1, n):  address\_similarity =比較地址的相似度(data[i]['地址'], data[j]['地址']) > 0.9  if 地址相似:  房型相似 =compare\_patterns(data[i]['houseinfo'][房型], data[j]['houseinfo'][房型])  坪數相似=compare\_patterns(data[i]['houseinfo'][ 坪數], data[j]['houseinfo'][ 坪數])  樓層相似=compare\_layers(data[i]['houseinfo'][ 樓層], data[j]['houseinfo'][ 樓層])  設備相似= device\_similarity(data[i][ 設備], data[j][ 設備])  if 所有條件都匹配  return 此組項目視為相似  def載入數據、預處理並查找相似項目():  # 載入並預處理數據  json\_data = 載入("D:\\AI\_data\\detail.json")  complete\_data = 預處理數據(json\_data)  similar\_items = 查找相似的項目(complete\_data)  print("相似的項目:")  for hid1, hid2 in similar\_items:  print(f"HID1: {hid1}, HID2: {hid2}")  # 執行主函數  if \_\_name\_\_ == "\_\_main\_\_":  main() | | | |

▼表9-2-2 部分程式碼－Yolo+clip.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 2-6-2 | 程式名稱 | Yolo+clip.py |
| 目的 | 房屋圖片的物件偵測與相似度比較。 | | |
| 部分程式碼 | | | |
| import torch  from PIL import Image  from transformers import CLIPProcessor, CLIPModel  from ultralytics import YOLO  from collections import Counter  import numpy as np  from sklearn.metrics import pairwise\_distances  import os  import glob  # 初始化YOLO模型、CLIP模型  yolo\_model = YOLO("yolov8n.pt")  clip\_model = CLIPModel.from\_pretrained("openai/clip-vit-base-patch32")  clip\_processor = CLIPProcessor.from\_pretrained("openai/clip-vit-base-patch32")  def偵測物件(image\_path):  results =YOLO模型進行物件偵測(image\_path)  objects =提取物件位置  return返回物件位置和結果  def提取主要顏色(image):  image = 圖像縮小((50, 50))  pixels =將像素轉換為數組(-1, 3)  counter =統計每個像素顏色的出現次數  dominant\_color =找出出現最多的顏色  return主要顏色  def使用CLIP生成英文描述(image, objects):  初始化描述列表= []  for obj in objects:  x1, y1, x2, y2 =提取物件邊界  cropped\_image =剪裁圖像  dominant\_color =提取主要顏色  color\_name =生成顏色名稱    inputs =準備圖像輸入  with torch.no\_grad():  image\_features =計算圖像特徵  定義文本描述= [  f"一個 {color\_name} 房間", f"一個 {color\_name} 椅子", f"一個 {color\_name} 桌子", f"一個 {color\_name} 床", f"一個 {color\_name} 燈", f"一個 {color\_name} 窗戶", f"一個 {color\_name} 門", f"一個 {color\_name} 沙發", f"一個 {color\_name} 書架", f"一個 {color\_name} 書桌", f"一個 {color\_name} 櫥櫃", f"一個 {color\_name} 衣櫃", f"一個 {color\_name} 衣櫥", f"一個 {color\_name} 地毯", f"一個 {color\_name} 相框", f"一個 {color\_name} 電視", f"一個 {color\_name} 電腦", f"一個 {color\_name} 植物", f"一個 {color\_name} 鐘", f"一個 {color\_name} 鏡子" ]  text\_inputs = clip\_processor(text=texts, return\_tensors="pt", padding=True)  文本輸入 =計算文本特徵  similarities =計算相似度  best\_match =找出最佳匹配  添加描述到列表((texts[best\_match], dominant\_color))  return描述列表  def計算描述相似度(desc1, desc2):  初始化相似度列表= []  for d1, 顏色1 in desc1:  for d2, 顏色2 in desc2:  比較文本相似度= 1 if d1.split(" ")[1:] == d2.split(" ")[1:] else 0  計算顏色相似度= 1 - pairwise\_distances([顏色1], [顏色2], metric='cosine')[0][0]  結合文本和顏色相似度= 0.5 \* 文本相似度 + 0.5 \* 顏色相似度  添加相似度到列表  return相似度列表  def處理文件夾(folder\_path):  image\_paths =獲取文件夾中的圖像路徑  初始化描述字典= {}  for image\_path in image\_paths:  objects, yolo\_results =偵測物件  image =開啟圖像  descriptions =生成描述  all\_descriptions[image\_path] =儲存描述  print處理進度  return所有描述  # 定義文件夾路徑  路徑1= “file\_path\_1”  路徑2= “file\_path\_2”  處理文件夾中的圖像1 = process\_folder(folder1)  處理文件夾中的圖像2 = process\_folder(folder2)  計算相似度= []  for path1, desc1 in descriptions1.items():  for path2, desc2 in descriptions2.items():  if描述不為空  similarities =計算描述相似度  similarity\_score =計算平均相似度  儲存相似度結果  image\_閾值= 0.8  folder\_閾值= 0.6  similar\_count = 0  # 比較結果  if len(all\_similarities) > 0:  for path1, path2, score, desc1, desc2 in all\_similarities:  print比較的圖像  print CLIP描述  print CLIP描述  print相似度分數  if score > image\_threshold:  print圖像可能來自同一個房間  similar\_count += 1  else:  print圖像可能來自不同的房間  folder\_similarity\_ratio =計算文件夾相似度  print 輸出文件夾相似度  if folder\_similarity\_ratio > folder\_threshold:  print文件夾可能來自同一個房子  else:  print文件夾可能來自不同的房子  else:  print沒有找到文件夾間的相似度 | | | |

▼表9-2-3 部分程式碼－ner\_clean\_1.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 2-6-3 | 程式名稱 | ner\_clean\_1.py |
| 目的 | 提取房屋描述內有關附近生活機能的店家或設施。 | | |
| 部分程式碼 | | | |
| import json  import re  輸入文件路徑= input\_file\_path  輸出文件路徑= output\_file\_path  # 從輸入文件中載入JSON數據  with open(input\_file\_path, "r", encoding="utf-8") as file:  data = json.load(file)  定義自定義的組織後綴列表= [  "公園", "學校", "醫院", "機構", "市", "店", "超市", "站", "中心", "街",  "市場", "公司", "高中", "小學", "幼兒園", "大學", "學院", "診所", "館",  "局", "廣場", "院", "場所", "廟", "堂"  ]  定義特定排除項列表= ["不動產", "加盟店", "直營店", "元大花廣圓頂世紀館", "住都中心", "台北聯勝租賃部"]  def 處理名稱(names):  將複合名稱分割成多個組織名稱 = {  "台灣大學醫學院台北商業大學師範大學": ["台灣大學醫學院", "台北商業大學", "師範大學"],  "北商台大北科": ["北商", "台大", "北科"],  "東吳大學文化大學": ["東吳大學", "文化大學"],  "特力屋北市商銘傳大學": ["特力屋", "北市商", "銘傳大學"],  "台北商業大學醫療機構": ["台北商業大學","醫療機構"],  "全家超商醫療機構": ["全家超商", "醫療機構"],  "劍潭國小百齡高中": ["劍潭國小","百齡高中"],  "家樂福全聯蝦皮店": ["家樂福","全聯","蝦皮"],  "台灣銀行第一銀行花旗銀行合庫華南郵局": ["台灣銀行","第一銀行","花旗銀行","合庫","華南","郵局"],  "聯頂好家樂福銀行郵局": ["全聯","頂好","家樂福","郵局","銀行"],  "聯家樂福晴光商圈郵局": ["全聯","家樂福","晴光商圈","郵局"],  "家樂福郵局": ["家樂福","郵局"],  "南近郵局": ["郵局"]  }  初始化處理後的名稱列表= []  for 每個名稱:  for pattern, splits in檢查該名稱是否符合任何分割模式:  if 符合:  將名稱分割並添加到處理後的名稱列表中  break  else:  直接將名稱添加到處理後的名稱列表中  return 處理後的名稱列表  # 移除重複並處理空值的函數  def 移除重複並處理空值(data):  for item in data:  轉換列表為集合以去除重複  列表 = 集合(item.get('store', []))  if "醫院" in 列表:  列表移除("醫療機構")  else:  # 將所有“醫療機構”替換為“醫院”  if "醫療機構" in 列表:  列表移除("醫療機構")  列表新增("醫院")  if "學校" in列表and any(sub in列表for 包括 ["高中", "小學", "幼兒園", "大學", "學院"]):  列表移除("學校")  集合['store'] = 列表(unique\_stores) if unique\_stores else None  return data  with 處理後的數據保存到文件中  Print 打印完成訊息 | | | |

▼表9-2-4 部分程式碼－xxx.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 對應9-1編號 | 程式名稱 | xxx.py |
| 目的 |  | | |
| 部分程式碼 | | | |
|  | | | |

▼表9-2-5 部分程式碼－xxx.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 對應9-1編號 | 程式名稱 | xxx.py |
| 目的 |  | | |
| 部分程式碼 | | | |
|  | | | |

▼表9-2-6 部分程式碼－xxx.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 對應9-1編號 | 程式名稱 | xxx.py |
| 目的 |  | | |
| 部分程式碼 | | | |
|  | | | |

▼表9-2-7 部分程式碼－xxx.py

|  |  |  |  |
| --- | --- | --- | --- |
| 編號 | 對應9-1編號 | 程式名稱 | xxx.py |
| 目的 |  | | |
| 部分程式碼 | | | |
|  | | | |