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 | | | |
| 1.初始化與設置   * 載入所需的模組和庫：   + json 用於讀取 JSON 文件。   + torch 和 torch.nn.functional 用於深度學習操作和計算餘弦相似度。   + ckiptagger.WS 用於中文分詞。   + transformers.BertTokenizer 和 transformers.BertModel 用於 BERT 中文模型的分詞與嵌入。   + re 用於正則表達式處理文本。 * 初始化 CKIP 的斷詞模型 ws。 * 從預訓練模型中載入 BERT 分詞器 tokenizer 和 BERT 模型 model。   2.定義函數 get\_bert\_embedding   * 輸入參數: text（文本字符串） * 使用 BERT 分詞器將文本轉換為張量格式。 * 通過 BERT 模型計算嵌入（embedding）。 * 返回文本的最後一層隱藏狀態（CLS token 的嵌入）。   3.定義函數 load\_json   * 輸入參數: file\_path（JSON 文件路徑） * 從指定路徑讀取 JSON 文件並返回解析結果。   4.定義函數 preprocess\_data   * 輸入參數: items（JSON 數據項列表） * 對於每個項目 item：   + 提取並拼接地址文本。   + 使用 CKIP 斷詞工具對地址文本進行斷詞。   + 使用 BERT 模型計算地址文本的嵌入。   + 將嵌入結果存儲在項目中。 * 返回處理過的數據 items。   5.定義函數 cosine\_similarity   * 輸入參數: tensor1, tensor2（兩個張量） * 計算並返回兩個張量之間的餘弦相似度。   6.定義函數 device\_similarity   * 輸入參數: devices1, devices2（兩組設備列表） * 計算兩組設備之間的匹配度，並返回是否匹配度超過 70%。   7.定義函數 get\_tokenized\_text   * 輸入參數: text（文本字符串） * 使用 CKIP 斷詞工具對文本進行斷詞並返回斷詞結果。   8.定義函數 compare\_patterns   * 輸入參數: pattern1, pattern2（兩個房屋模式字符串） * 將兩個模式進行斷詞並計算 BERT 嵌入。 * 計算兩個嵌入之間的餘弦相似度並返回是否相似度超過 90%。   9.定義函數 compare\_layers   * 輸入參數: layer1, layer2（兩個房屋樓層字符串） * 將樓層字符串分割為單層，對每對單層進行斷詞並計算 BERT 嵌入。 * 計算每對嵌入之間的餘弦相似度，返回最大相似度是否超過 90%。   10.定義函數 find\_similar\_items   * 輸入參數: data（已處理的完整數據） * 初始化 similar\_items 列表。 * 對於每對數據項目 data[i] 和 data[j]，檢查以下條件：   + 地址相似度是否超過 90%。   + 模式、大小和樓層的相似度是否均超過指定閾值。   + 設備列表是否匹配。   + 如果上述條件均滿足，將 data[i] 和 data[j] 的 hid 添加到 similar\_items 列表。 * 返回 similar\_items。   11.主程式入口 main   * 調用 load\_json 函數以讀取 JSON 數據。 * 調用 preprocess\_data 函數以預處理數據，獲取完整數據 complete\_data。 * 調用 find\_similar\_items 函數以查找類似項目。 * 打印找到的類似項目，最多顯示前三個。   12.主程式執行   * 調用 main 函數。 | | | |

▼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 | | | |
| 1.初始化與設置   * 載入所需的模組和庫：   + torch 用於深度學習相關操作。   + PIL.Image 用於圖像處理。   + transformers.CLIPProcessor 和 transformers.CLIPModel 用於 CLIP 模型的處理和推理。   + ultralytics.YOLO 用於 YOLO 物件偵測模型。   + collections.Counter 用於計算圖像的主要顏色。   + numpy 用於數據處理。   + sklearn.metrics.pairwise\_distances 用於計算顏色相似度。   + os 和 glob 用於文件系統操作。   2.初始化模型   * 初始化 YOLO 模型 yolo\_model。 * 從預訓練模型中載入 CLIP 模型 clip\_model 和 CLIP 處理器 clip\_processor。   3.定義函數 detect\_objects   * 輸入參數: image\_path（圖像路徑） * 使用 YOLO 模型進行物件偵測。 * 返回偵測到的物件座標和結果。   4.定義函數 get\_dominant\_color   * 輸入參數: image（PIL 圖像對象） * 將圖像調整為 50x50 大小，提取像素並計算主要顏色。 * 返回主要顏色。   5.定義函數 generate\_clip\_description   * 輸入參數: image（PIL 圖像對象），objects（偵測到的物件） * 對每個物件進行以下操作：   + 裁剪圖像中對應的物件部分。   + 計算裁剪圖像的主要顏色。   + 使用 CLIP 模型生成該物件的圖像嵌入。   + 定義一組包含顏色信息的文本描述。   + 使用 CLIP 模型生成文本嵌入。   + 計算圖像特徵和文本特徵之間的相似度，找出最匹配的文本描述。   + 將匹配的描述和顏色信息添加到描述列表中。 * 返回生成的描述列表。   6.定義函數 calculate\_similarity   * 輸入參數: desc1（描述列表1），desc2（描述列表2） * 初始化相似度列表 similarities。 * 對每對描述進行以下操作：   + 計算文本相似度（忽略顏色部分）。   + 計算顏色相似度（使用餘弦相似度）。   + 結合文本和顏色相似度，計算綜合相似度。   + 將綜合相似度添加到相似度列表中。 * 返回相似度列表。   7.定義函數 process\_folder   * 輸入參數: folder\_path（資料夾路徑） * 使用 glob 模組獲取資料夾中的所有圖像文件路徑。 * 初始化描述字典 all\_descriptions。 * 對每個圖像文件進行以下操作：   + 調用 detect\_objects 函數偵測物件。   + 使用 PIL 載入圖像。   + 調用 generate\_clip\_description 函數生成圖像描述。   + 將描述存儲到 all\_descriptions 字典中。   + 打印處理信息。 * 返回所有圖像的描述字典。   8.處理資料夾並比較相似度   * 定義兩個資料夾路徑 folder1 和 folder2。 * 調用 process\_folder 函數處理這兩個資料夾，獲取每個圖像的描述。 * 初始化相似度列表 all\_similarities。 * 對每對圖像描述進行以下操作：   + 如果描述列表不為空，調用 calculate\_similarity 函數計算相似度。   + 計算平均相似度並存儲結果。 * 根據設定的閾值判斷兩個資料夾內圖像的相似程度。 * 打印相似度結果和資料夾相似度比例。   9.主程式入口   * 執行資料夾處理和相似度比較邏輯，並打印最終結果。 | | | |

▼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 | | | |
| 1.初始化與設置   * 定義輸入文件路徑 input\_file\_path 和輸出文件路徑 output\_file\_path。 * 載入 JSON 文件中的數據。   2.定義自訂後綴與排除項   * 定義一組表示機構的自訂後綴 custom\_suffixes（如 "公園", "學校", "醫院" 等）。 * 定義一組需要排除的特定項目 exclusions。   3.定義函數 preprocess\_names   * 輸入參數: names（機構名稱列表） * 定義分割模式 split\_patterns，針對某些特殊情況對名稱進行分割和處理。 * 初始化 processed\_names 列表。 * 對於每個名稱 name：   + 檢查名稱是否符合 split\_patterns 中的任何模式。   + 如果符合，將其分割為多個名稱並添加到 processed\_names 列表中。   + 否則，將原始名稱添加到 processed\_names 列表中。 * 返回處理過的名稱列表 processed\_names。   4.定義函數 remove\_duplicates\_and\_manage\_empty   * 輸入參數: data（處理過的 JSON 數據） * 對於每個數據項 item：   + 將 store 列表轉換為集合，以移除重複項目。   + 如果集合中存在“醫院”，則確保集合中不會有“醫療機構”。   + 如果集合中沒有“醫院”，但存在“醫療機構”，則將“醫療機構”替換為“醫院”。   + 如果集合中存在“學校”，並且集合中包含特定的教育後綴（如“高中”、“小學”、“幼兒園”、“大學”、“學院”），則移除“學校”。   + 將集合轉換回列表，或者如果集合為空，則將 store 設置為 None。 * 返回處理過的數據 data。   5.保存處理後的數據   * 將處理過的數據保存到指定的輸出文件路徑 output\_file\_path 中，格式為 JSON。   6.打印完成訊息   * 打印消息，表明處理過的數據已保存到輸出文件路徑。 | | | |

▼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 | | | |
|  | | | |