基於 NoSQL 資料庫的社群網路推薦系統設計與實現

組員姓名	負責工作
徐崇恆	架構設計，社群模擬模塊
盛正璿	演算法模塊，資料視覺化
梁立昀	資料交互模式與資料庫模塊
葉俊廷	資料交互模式與資料庫模塊

1 摘要

本專題探討一套模擬社群網路的系統，旨在研究使用類 Redis 資料庫設計更符合使用者喜好的內容推薦機制。我們使用了我們在課堂作業開發的類 Redis 資料庫程式作為資料庫。該系統模擬了一批虛擬使用者，為每位使用者分配一組隱藏的興趣標籤。透過演算法推測使用者的潛在興趣，系統試圖提升內容推薦的精確性與吸引力。

此外，系統設計了一套按讚機制，模擬使用者的互動行為。當貼文內容與使用者的興趣標籤相符時，按讚的可能性將提高。為貼近真實情境，模擬過程中加入了隨機性，以反映人類行為的多樣性。最後，系統透過多回合模擬評估其效能，檢驗演算法對使用者興趣預測的準確性與推薦效果。

2 需求

本專題的需求包括系統功能設計及使用類 Redis 資料庫的具體應用。首先，系統需能生成並管理虛擬使用者，為每位使用者分配並更新隱藏的興趣標籤。利用記憶體資料庫的高效資料存取特性，實現興趣推測演算法，根據使用者的行為數據動態調整內容推薦，提升推薦的精確性和個性化程度。

此外，系統需設計按讚互動機制，模擬使用者對貼文的按讚行為，並引入隨機性以反映真實行為的多樣性。記憶體資料庫可用於快速存取和更新使用者的互動記錄，支持即時的行為模擬與反饋。系統還需具備多回合模擬運行的能力，通過記憶體資料庫的快速讀寫優勢，實現大規模數據的高效處理與模擬運行。

在評估方面，系統應能對演算法的效能進行準確性和推薦效果的分析，利用資料庫儲存和處理大量模擬數據，並支持即時的數據分析與視覺化展示。最後，系統需具備良好的可擴展性和可靠性，確保在使用者數量和內容量增加時依然能保持高效運行，並保障數據的安全性與完整性。

總結而言，程式應具備虛擬使用者生成與管理、興趣推測與內容推薦、按讚行為模擬、隨機性引入、多回合模擬運行及效能評估等核心功能，並充分利用資料庫的高效存取和處理能力，以實現系統的穩定運行與高效性能。

3 設計

資料模型

本系統專注於模擬社群網路，因此無需生成具體的貼文內容或使用者詳細資訊。我們的物件類型包括 User、Post 和 Tag，物件實際上是抽象物件，它們的屬性儲存在資料庫中，僅在查詢或更新屬性時進行讀寫操作。開發工作聚焦於基於資料庫的上層應用，且不對外提供直接操作資料庫的接口。

Tag

- name: string

Post

- tags: Tag[] // 儲存 Tag IDs 的字串 List，無權重

User

- name: string
- p_tags: TagWithWeight[] // 儲存 Tag IDs 的字串 List，每個字串格式為 "<id>:<weight>"

a_tags: TagWithWeight[] // 同 p_tags

資料庫

建置於資料庫核心程式的上層應用。

主要功能

- **統一介面：**提供與資料庫互動的標準化方式，簡化操作並提升靈活性。
- **功能包裝：**將資料庫功能整合為高階介面，支援進階查詢場景。
- **抽象物件處理：**將物件的屬性 (fields) 儲存於不同的鍵值對中，適配 Redis 類資料庫架構。

為什麼需要統一介面？

- 隱藏底層複雜度，讓開發者專注於應用邏輯。
- 減少重複程式碼，提升維護性與開發效率。

社群網路

利用資料庫模塊並與演算法模塊配合。

主要功能

- **初始化：**清除原有資料，重置環境，確保模擬運行的一致性。
- **建立假資料：**隨機生成大量的使用者 (user) 和貼文 (post)，模擬真實的社群平台數據。
- **模擬社群使用行為：**推送貼文給使用者，模擬使用者對貼文的互動行為（如按讚）。
- **結合反饋進行評估更新：**根據使用者的互動結果，動態更新對使用者偏好的評估，提升模擬準確性。
- **輸出結果：**輸出模擬結果，以便後續分析及生成可視化圖表。

按贊機制

根據使用者 atag 的權重進行按贊，例如使用者對 tagX 有 0.7 的權重，當他瀏覽一篇包含 tagX 的帖文時，他對該帖文有 70% 的概率按贊。

與演算法函式配合

- 對於推薦演算法，函數輸出的 y 即根據使用者當前 ptags 進行推送的比例，x 是迭代的完成度。例如 y = x^2 時，根據使用者當前的 ptags 推送的比例會逐漸上升。
- 對於擬合演算法，函數輸出的 y 是 ptags 偏移量所佔的權重。例如 y = x^2 時，根據 ptags 的偏移量在前期其較大的作用，在後期則對其較小作用。這樣設計是為了在前期迭代時對使用者 ptags 的更大程度地更新。

演算法

主要功能

- 決定隨機帖文的佔比以及 ptags 擬合的方式

函數選擇

- 機器學習常用的激勵函數（s_sigmoid、s_selu）、自己設計的函數（s_square、s_create、s_075_025、s_cube)
- 優勢：擺脫線性關係
- 選用原因：好奇這些函數，如果對 AI 思考有用，在對此專題時是否有用。

3 實作

資料庫 Object ID

本系統採用 16 進制、長度為 16 的字串作為資料庫物件的唯一識別碼（Object ID, 簡稱 OID），用於高效查詢。為避免碰撞，OID 的生成參考了主流的 NoSQL 包含了時間戳信息。

模塊

資料庫模塊

OID

- **generate_oid**
  功能：利用毫秒級別的時間戳來創造oid，實現oid唯一性
  實踐：利用gettimeofday 獲取當前時間戳，再計算毫秒級別的時間戳來創造oid，接著更新時間戳與序列（檢查是否跨毫秒、序列是否超過最大值，以確保oid唯一性），最後格式化oid（OID結構： 由 12 位時間戳 + 4 位序列號）。
- **parse_oid**
  功能：提取 key 的 OID
  邏輯：從格式為 namespace:oid:field 的鍵字串中提取出中間的 oid，並返回一個動態分配的字串表示。
- **convert_to_ids**
  功能：將輸入的鍵列表（keys）轉換為對應的 ID 列表。
  邏輯：遍歷列表節點。，提取鍵中的 ID， 替換鍵值為 ID，返回處理後的列表。
- **create_query_key**
  功能：生成一個查詢鍵，格式為 namespace:id:field，並返回動態分配的字串。
  實踐：動態分配記憶體給符合格式化的鍵字串。
- **db_set_list**
  功能：將一個列表（DBList）的所有值存儲到數據庫中，與給定的鍵（key）相關聯。
  實踐：調用 dbapi_del 刪除鍵 key 在數據庫中對應的舊數據。 遍歷列表節點後， 將每個節點的值（字符串）追加到鍵 key 對應的數據庫列表中。

User

- **get_user_ids**
  列出所有使用者的 id
- **create_user_with_id_returned**
  功能：在資料庫中創建一個新的用戶，並將用戶的name和a_tags存儲到對應的鍵中。
  實踐：調用 generate_oid 生成唯一的用戶 ID。 使用 create_query_key 構造數據庫鍵，將用戶名稱存儲到數據庫中。 再次使用 create_query_key 構造數據庫鍵，將用戶標籤存儲到數據庫中。 返回生成的用戶唯一 ID。
- **create_user**
  執行：free(create_user_with_id_returned(name, atags))
- **get_user_id_by_name**
  功能：通過用戶名稱（name）查找對應的用戶 ID，並返回該用戶的唯一 ID（user_id）。
  邏輯：獲取所有用戶 ID 列表。 遍歷每個用戶 ID，查詢名稱並比較。 返回匹配的用戶 ID 或 NULL（未找到）。
- **delete_user**
  功能：根據給定的用戶唯一 ID (oid)，從數據庫中刪除該用戶的所有相關數據（名稱、標籤等）。
  邏輯：檢查 oid 是否有效，分別刪除用戶名稱和標籤的數據，釋放分配的鍵內存。

Post

- **get_post_ids**
  列出所有貼文的 id
- **create_post_with_id_returned**
  功能：創建一個新的post，將與該帖子相關聯tags存儲到數據庫中，最後返回該post的oid。
  邏輯：調用 generate_oid 生成唯一的帖子 ID。 使用 create_query_key 為帖子生成存儲標籤的數據庫鍵。 調用 db_set_list 將標籤數據存入數據庫。 釋放分配的鍵內存。 返回生成的帖子 ID 給調用方。
- **create_post**
  執行：free(create_post_with_id_returned(tags))
- **create_post_indexes**
  功能：為數據庫中的每個tag_id創建索引，將與該標籤相關聯的post_id列表存儲到索引命名空間INDEX_NS中。
  邏輯：獲取數據庫中所有標籤的 ID 列表，並逐一處理每個標籤。 對於每個標籤，查詢其相關聯的帖子，並將帖子列表存入索引命名空間中。 釋放臨時分配的內存，確保內存使用安全，最後完成索引重建。
- **delete_posts**
  功能：刪除數據庫中貼文與索引。
  邏輯：構造匹配所有貼文、索引鍵的模式鍵。查詢並刪除所有與有貼文、索引相關的鍵。釋放資源：確保所有臨時分配的內存均被正確釋放。
- **get_post_tags**
  功能：查詢與貼文有關聯的標籤列表。
  實踐：生成查詢鍵，利用dbapi_lrange 查詢數據庫中該鍵對應的列表型數據，返回查詢結果，並釋放臨時資源。
- **get_posts_by_tag**
  功能：根據指定的tag_id，查詢數據庫中與該標籤相關的post_id，並返回最多 limit 條結果。
  實踐：如果 by_index 為 true，則通過索引鍵快速查詢與tag_id 相關的帖子，並返回最多 limit 條結果。 如果不使用索引，則遍歷所有帖子 ID，逐一檢查每個帖子的標籤是否匹配 tag_id，將符合條件的帖子加入結果列表，直到達到 limit。 確保過程中分配的臨時資源（如鍵、帖子列表、標籤列表）均被正確釋放，最後返回過濾後的帖子 ID 列表。

Tag

- **get_tag_ids**
  功能：獲取所有標籤的 ID 列表
  邏輯：根據鍵的命名規範，生成匹配所有標籤鍵的模式。 使用模式查找數據庫中所有符合條件的鍵。 從查找到的鍵中提取標籤的唯一 ID，並返回這些 ID 的列表。
- **create_tag_with_id_returned**
  功能：給定名稱創建一個唯一標識的標籤（Tag），並將標籤信息存儲到數據庫中。（生成一個唯一的 ID，將名稱與該 ID 關聯，然後返回該 ID）
  實踐：調用 generate_oid() 生成唯一標識符。 使用 create_query_key() 組裝存儲鍵，結合命名空間、ID 和字段名。 調用 dbapi_set() 將鍵值對存入數據庫。 釋放臨時鍵內存。 返回生成的唯一標識符。
- **create_tag**
  功能：執行free(create_tag_with_id_returned(name))

- **get_user_atags**
  功能：根據user_id查詢與該用戶相關的所有atags。
  實踐：生成包含user_id 和atags 的查詢鍵，並用該鍵從資料庫中提取對應的列表型數據。 釋放生成的查詢鍵後，返回查詢到的標籤列表。
- **set_user_atags**
  功能：根據user_id更新用戶的atags 列表為提供的標籤列表tags。
  實踐：生成包含user_id 和atags 的查詢鍵，並使用該鍵將新的標籤列表存入資料庫，替換舊值。 釋放生成的查詢鍵後，返回 true 表示操作成功。
- **get_user_ptags**
  功能：根據user_id查詢與該用戶相關的所有ptags。
  實踐：生成包含user_id 和ptags 的查詢鍵，並用該鍵從資料庫中提取對應的列表型數據。 釋放生成的查詢鍵後，返回查詢到的標籤列表。
- **set_user_ptags**
  功能：根據user_id更新用戶的ptags 列表為提供的標籤列表tags。
  實踐：生成包含user_id 和ptags 的查詢鍵，並使用該鍵將新的標籤列表存入資料庫，替換舊值。 釋放生成的查詢鍵後，返回 true 表示操作成功。

Database 相關

- **start_db** //引用dbapi_start_server
  功能：這個函數的作用是啟動數據庫服務器，並確保操作過程是線程安全的。
  邏輯：
  1. core_unlock() 鎖定核心資源，保證數據庫啟動的操作是線程安全的。 確保在執行 db_start() 時，其他線程無法同時訪問核心資源，避免競態條件。
  2. db_start() 啟動數據庫服務器的核心函數，執行初始化操作。 包括加載配置文件、初始化內部結構、分配內存、打開網絡端口等。
  3. core_unlock() 解鎖核心資源，允許其他線程訪問。 確保 db_start() 完成後，核心資源不再被獨占。

  （互斥鎖是一種用於多線程編程中的同步工具，主要用來保護共享資源，防止多個線程同時訪問造成競態條件。它的核心作用是確保同一時間只有一個線程能進入被保護的臨界區）

- **save_db** //引用dbapi_save
  功能：執行數據庫的同步保存操作，並返回操作是否成功的結果。
  邏輯：創建並發送一個 DB_SAVE 請求。 獲得數據庫響應，檢查是否有錯誤。 如果響應不是錯誤，進一步檢查響應是否為字符串 "OK"。 最後，返回操作是否成功的結果。
- **flush_all** // 直接引用 free_dblist(list)
  清空資料庫
  功能：數據庫的清空操作，並將變更保存到持久化存儲。

社群網路模塊

此模塊是將被主程式引用的重要模塊。在這個模塊中，我們研究了適合推薦帖子的演算法，以及思考如何盡可能模擬真實使用者的行為。

- **init_social_network**
  功能：初始化社交網路的使用者和 POST。
  邏輯：
  1. 清空資料庫。

2. 建立標籤和初始機率（tag_prob_dict 保存標籤機率，tag_id_dict 保存標籤與其 ID 的映射）。
3. 根據標籤創建 TOTAL_USERS 的使用者，隨機分配標籤和權重。
4. 創建 TOTAL_POSTS 條 POST，每個 POST 包含隨機標籤。
5. 釋放相關記憶體（如標籤的雜湊表、清單等）。

- **create_tag_w**
  功能：創建一個帶有權重的標籤結構。
  邏輯：
  1. 檢查 tag_id 是否為空。
  2. 分配記憶體給 TagWithWeight 結構。
  3. 初始化標籤 ID 和權重。
  4. 返回創建的結構。
- **free_tag_w**
  功能：釋放 TagWithWeight 結構的記憶體。
  邏輯：
  1. 檢查指標是否為空。
  2. 釋放標籤 ID 和結構本身的記憶體。
- **serialize_tag_w**
  功能：將 TagWithWeight 結構序列化為字串。
  邏輯：
  1. 檢查結構是否有效。
  2. 計算緩衝區大小。
  3. 使用 snprintf 格式化標籤 ID 和權重，並返回序列化的字串。
- **parse_tag_w**
  功能：解析標籤與權重的字串，返回 TagWithWeight 結構。
  邏輯：
  1. 確認輸入字串非空。
  2. 找到 : 分隔符，提取標籤 ID 和權重。
  3. 使用 create_tag_w 創建結構並返回。
- **init_users_ptags**
  功能：初始化使用者的 PTAG（喜好標籤）。
  邏輯：
  1. 為所有使用者創建空的 PTAG 清單。
  2. 遍歷所有使用者，將空 PTAG 或傳入的 PTAG 設置為該使用者的 PTAG。
  3. 釋放記憶體。
- **likes_dict_to_ptags**
  功能：根據使用者喜歡的 POST，生成對應的 PTAG（標籤及其權重）。
  邏輯：
  1. 收集所有 POST 的 ID。
  2. 建立 tag_likes_dict 和 tag_total_dict：
     - tag_likes_dict 記錄每個標籤的喜歡數。
     - tag_total_dict 記錄每個標籤的總出現次數。
  3. 計算每個標籤的權重（喜歡數/總出現次數），並轉換為 PTAG。
  4. 返回生成的 PTAG 清單。
- **create_user_feedback**
  功能：創建一個使用者回饋結構。
  邏輯：
  1. 分配記憶體。

2. 初始化 likes_dict（喜歡的 POST）、ptags（標籤權重）、users_count（使用者數量）、likes_count（喜歡數）、posts_count（POST 數）。
3. 計算喜歡率（喜歡數 / POST 數）。
4. 返回創建的結構。

- **free_user_feedback**
  功能：釋放使用者回饋結構的記憶體。
  邏輯：
  1. 釋放 likes_dict 和 ptags。
  2. 釋放回饋結構。
- **calculate_post_like_probability**
  功能：計算某個 POST 被某個使用者喜歡的概率。
  邏輯：
  1. 獲取 POST 的標籤清單。
  2. 遍歷 POST 標籤和使用者的 ATAG（偏好標籤），計算匹配的權重和概率。
  3. 返回最終計算的喜歡概率（不超過 1）。
- **simulate_user_feedback**
  功能：模擬單一使用者對多個 POST 的回饋。
  邏輯：
  1. 獲取使用者的 ATAG（偏好標籤）。
  2. 模擬每個 POST 的喜歡或不喜歡，記錄到 likes_dict。
  3. 返回模擬的回饋結構。
- **collect_popular_feedback**
  功能：收集所有使用者對 POST 的回饋數據。
  邏輯：
  1. 模擬每個使用者對所有 POST 的回饋。
  2. 累積所有 POST 被喜歡的次數。
  3. 返回包含總體喜歡數據的回饋結構。
- **get_posts_by_ptags**
  功能：根據 PTAG（偏好標籤）推薦 POST。
  邏輯：
  1. 標準化 PTAG 權重，使其總和為 1。
  2. 根據權重比例分配推薦的 POST 數量。
  3. 返回推薦的 POST 清單。
- **recommand_posts**
  功能：推薦 POST，基於使用者 PTAG 和熱門 PTAG。
  邏輯：
  1. 根據算法計算推薦比例（使用者偏好與熱門標籤的權重）。
  2. 獲取基於 PTAG 和熱門 PTAG 的推薦 POST。
  3. 合併兩部分推薦 POST，去重後返回。
- **aggregate_ptags**
  功能：聚合使用者的 PTAG，基於回饋數據。
  邏輯：
  1. 計算新的 PTAG 權重（舊權重與回饋權重的加權平均）。
  2. 更新已有的 PTAG 或新增新的 PTAG。
  3. 返回更新後的 PTAG。
- **run_simulations**
  功能：模擬多輪推薦和回饋過程。
  邏輯：

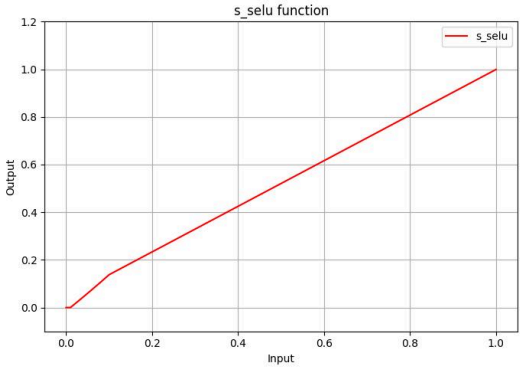
1. 初始化熱門 PTAG。
2. 模擬多輪推薦：
    a. 為每個使用者推薦 POST，記錄回饋。
    b. 基於回饋更新使用者的 PTAG。
3. 清理低權重的 PTAG，保存結果。

演算法模塊

所有的函數都經過不斷調整，保證輸出值在0~1之間，提供不一樣的new_p_tag權重調整讓社群網路模塊選讓使用哪一種函數，並且在膜快最上方提供調整參數的方式

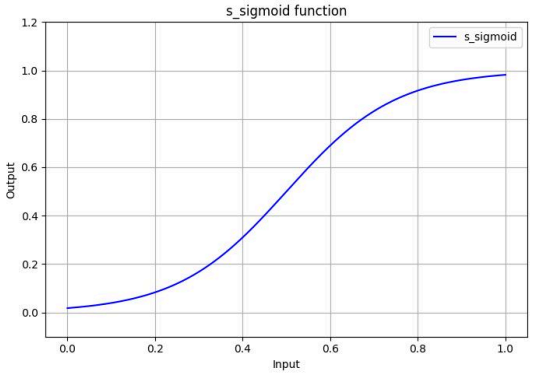
Selu

- 平移傳統SeLU函數，並調整lamda值讓最大值接近1，ReLU讓過小值為0
- 其他地區線性相對放大，在接近0時改用曲線逼近0



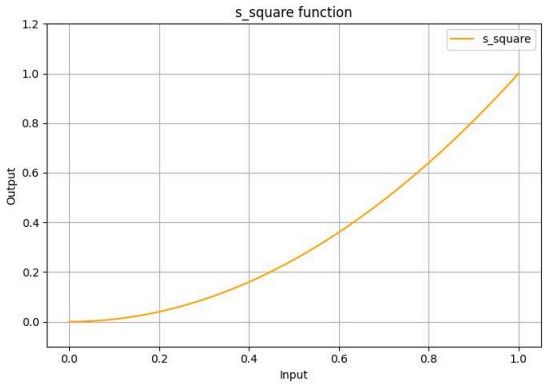
Sigmoid

- 更改傳統羅吉斯回歸函數，調整k值讓最大值接近1，讓最小值接近0
- 小於一半相對變小，大於一半相對變大



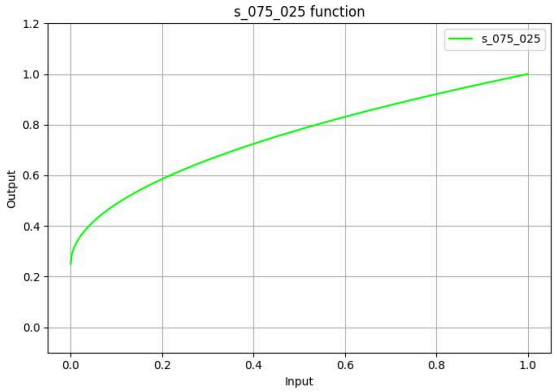
Square

- 回傳平方
- 小值變小的比例放大



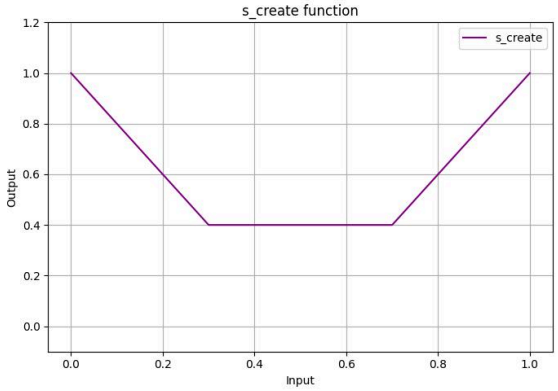
075_025

- y = 0.75*x^0.5 + 0.25
- 自己設計的函數。



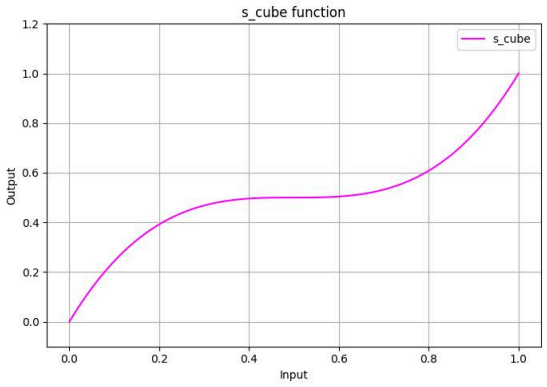
Create

- 自己設計的函數。



Cube

- y = 4 (x-0.5)^(3) + 0.5
- 自己設計的函數。



5 測試與結果

各種演算法組合的預測準確率結果

每回合帖文：20 篇 (±5篇)
模擬回合次數：20 回合

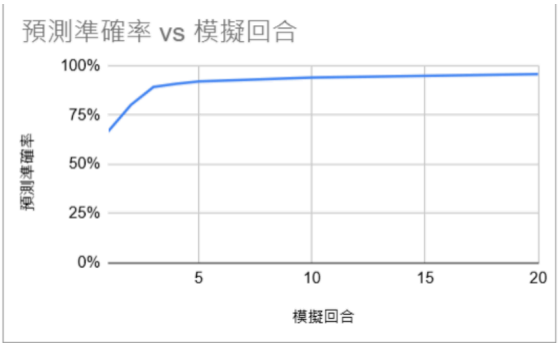
擬合函數 ⇓

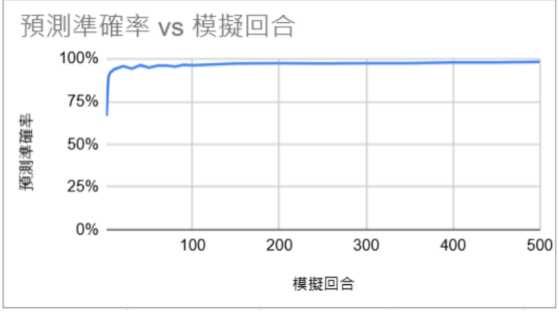
推薦函數 ⇒

	s_075_025	s_cube	s_selu	s_sigmoid	s_square	s_create
s_075_025	95.23%	92.57%	95.10%	95.65%	94.31%	94.29%
s_cube	95.86%	93.33%	95.92%	96.42%	93.02%	92.71%
s_selu	95.88%	93.00%	94.09%	96.40%	95.10%	92.54%
s_sigmoid	95.71%	94.15%	94.36%	96.30%	94.04%	92.30%
s_square	94.55%	94.66%	94.78%	95.33%	94.09%	93.89%
s_create	95.22%	92.80%	95.52%	94.56%	94.33%	93.89%

註：表格內的數據皆是經過多輪測試的平均

模擬回合(總數)	預測準確率
1	67%
2	80%
3	89%
4	91%
5	92%
10	94%
20	96%
30	94%
40	96%
50	95%
60	96%
70	96%
80	96%
90	97%
100	96%
150	97%
200	98%
250	97%
300	98%
350	98%
400	98%
450	98%
500	98%





每回合帖子：
15 篇 (±5篇)

推薦函數：
s_cube

擬合函數：
s_075_025

沒有 Popular ptags			有 Popular ptags	
第n回	預測準確率		第n回	預測準確率
1	80%		1	93%
2	86%		2	85%
3	88%		3	88%
4	90%		4	89%
5	90%		5	93%
6	89%		6	92%
7	90%		7	92%
8	92%		8	93%
9	93%		9	94%
10	93%		10	94%

每回合帖子：
15 篇 (±5篇)

推薦函數：
s_cube

擬合函數：
s_075_025

6 結論

我們實作了利用**資料庫程式**開發程式，並研究了**推薦演算法**和用來符合使用者喜好的**標籤演算法**。在這個過程中，我們嘗試使用了幾種不同的函數來推薦內容和分析使用者的偏好。經過測試，我們發現這些方法都能很好地推斷出使用者的喜好。這是因為我們假設帖文的標籤都是正確的，並且使用者按讚的行為符合一定的規律。然而，這樣的假設也讓我們沒能找到一個特別突出的函數。如果我們拿這些結果和現實生活中的情況相比，會發現實際影響使用者喜好的因素有很多。例如，不是每個人都會按讚，所以對帖文在熒幕的停留時間以及分享次數的考慮也很重要。這些行為也能反映他們的喜好，但因為這些功能現階段對我們來說比較複雜，目前我們還沒有把它們加進系統中。

我們也能由此推論，只要社群平台可以對帖文做好 **tagging**，不管任何演算法，使用者的喜好輕易的被掌控。因此像是利用機器學習去對一個內容做 tagging 在現實情況中也是很

重要的一個環節。因為只要 tagging 做好，哪怕使用者只做了很少的互動，平台就足以推斷出使用者的性格。 而我們的社群網路的標籤數比較少，我們把每個標籤都當作獨立的來看，這在小規模上沒有問題。但是現實生活中可能有上萬個標籤，在這樣的情況下，效能就有問題了。我們需要能對標籤進行**分級與關聯性**的建立，才能更有效地探索使用者的喜好。

7 參考

1. Redis Docs
   https://redis.io/docs/latest/

8 附錄

1. cJSON 函式庫
   https://github.com/DaveGamble/cJSON
2. 資料結構 Final Project 完整 DEMO
   https://youtu.be/FJtAljRq83c