國立政治大學

112 學年度第 1 學期 網路搜索與探勘 期末報告

KKCompany Music Challenge: Next-5 Songcraft

授課教師：蔡銘峰　教授

學生：陳繹帆、陳品絜、沈欣柏、徐韶汶

目錄

圖目錄

表目錄

壹、業務問題

預測用戶在同一個聆聽 session 內聆聽一定數量歌曲後，接下來可能會聆聽哪些歌曲，同時需要避免過度集中於熱門音樂，以打造更個性化的音樂推薦服務，提高用戶滿意度。

貳、探索性資料分析

一、熱門歌曲特徵

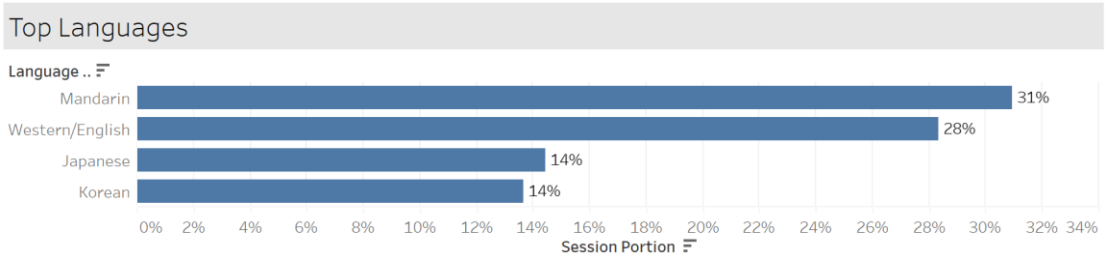

圖 1、Top Languages

有 31%的 Session 聽中文歌、28%聽西洋和英文歌，中英加總近 60%，各有 14%Session 聽日文、韓文歌，住四種語言總共佔將近九成 Session。

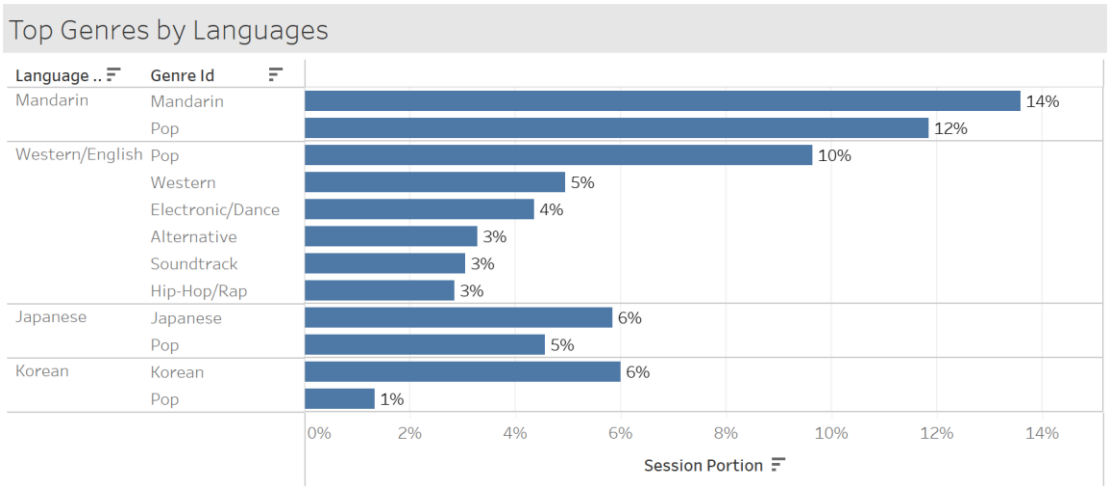

圖 2、 Top Genres by Languages

各種語言最熱門的曲風不外乎是自己的語言和流行樂（pop），26% Session 聽中文流行歌、15%西洋英文流行歌、11%日文流行歌、7%韓文流行歌，共佔近 60%，西洋和英文歌常聽的曲風比較多元。

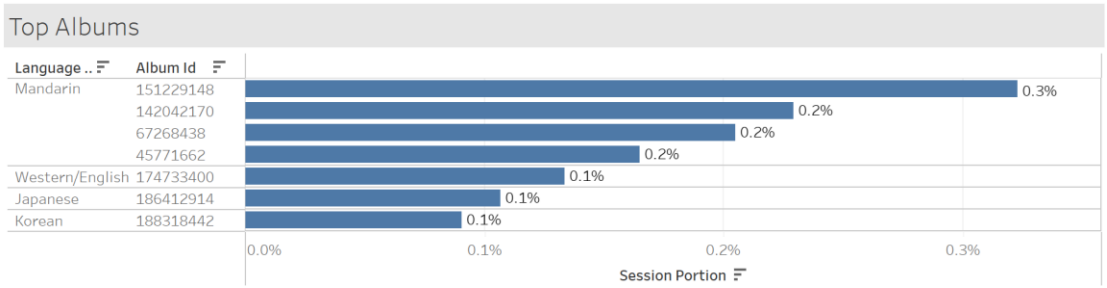

圖 3、 Top Albums by Languages

中文專輯中最熱門專輯的有 0.3%Session 在聽，前四名有 0.2%以上的 Session，英日韓最熱門專輯都不超過 0.1%Session。

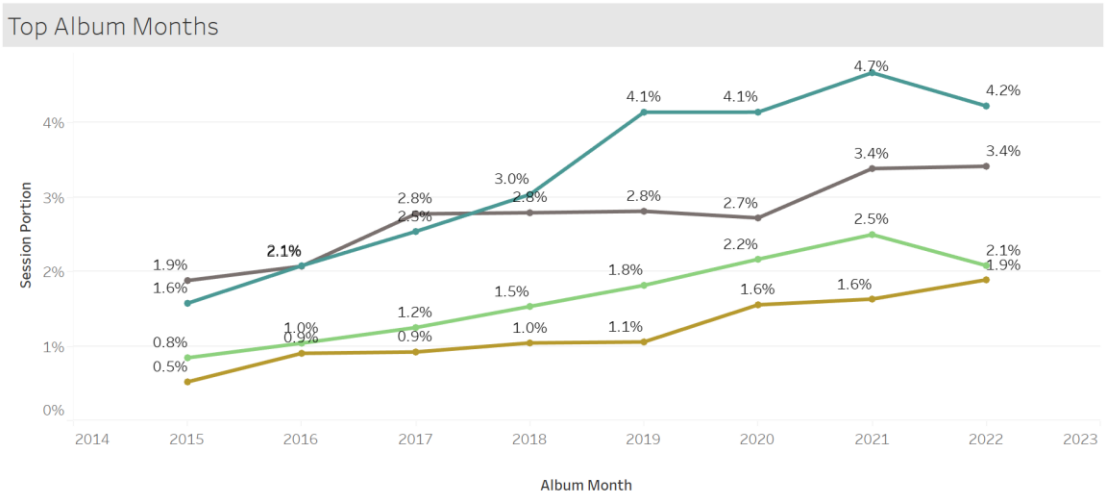

圖 4、 Top Songs by Years

6 成 session 聽 2015~2022 的歌，這 8 年的歌越新越多 session 聽，但是西洋、英文和韓文比較沒有像中日文歌這麼明顯 session 比例隨年份上升，英文在 2017~2020、韓 2017~2019 聽眾比例持平，沒有隨著歌曲越新越多聽眾。

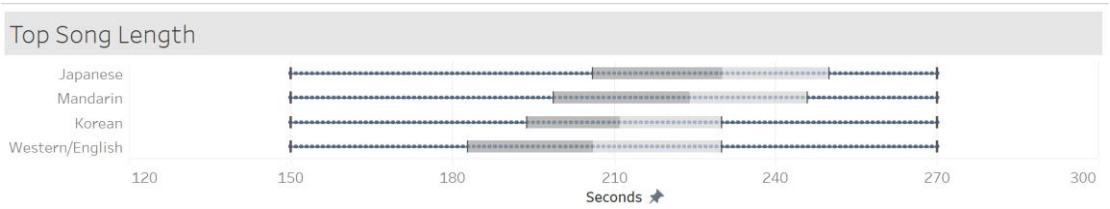

圖 5、 Top Song Length

70%session 150~270seconds (2'30"~4'30")，從這些 150~270seconds 的歌來看，中日歌曲長度比英韓長，中位數、Q1、Q3 都較大。

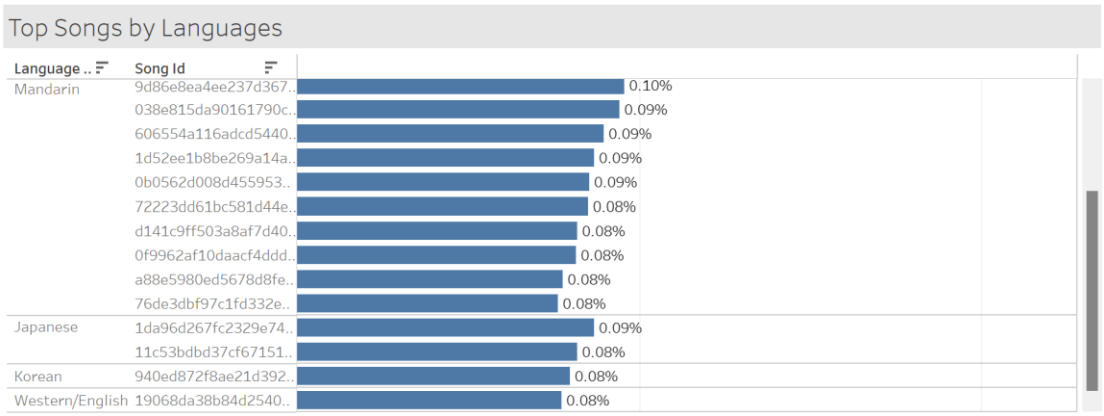

圖 6、 Top Songs by Languages

最熱門的歌集中在中文歌，最多 Session 聽的中文歌是 0.2%，其他英日韓文的 top songs 都不到 0.1%。（英文 0.08%、日 0.09%、韓 0.08%）

二、聽眾的聽歌習慣

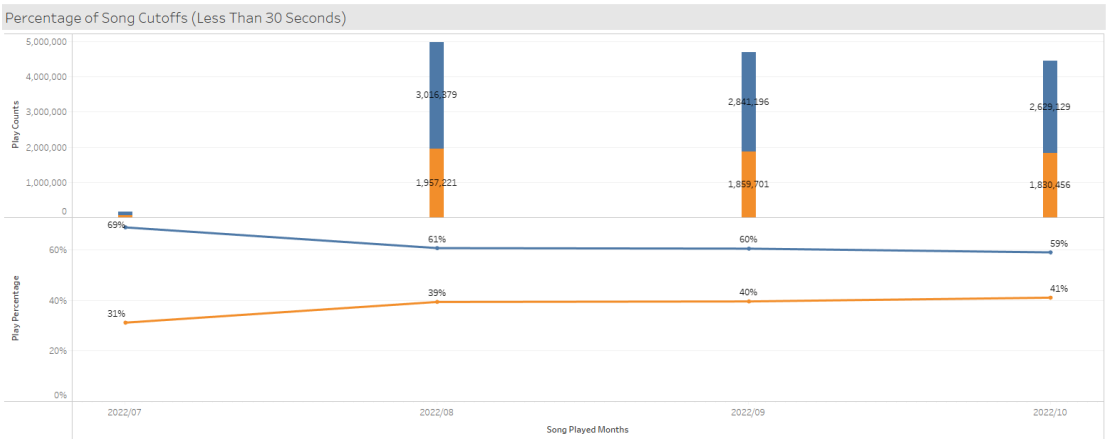

圖 7、 Percentage of Song Cutoffs

從播放次數來看，播放資料集中在 8.9.10 月，約 4 成左右的播放會切歌，8~10 月切歌比例從 39%~41%微幅增加。

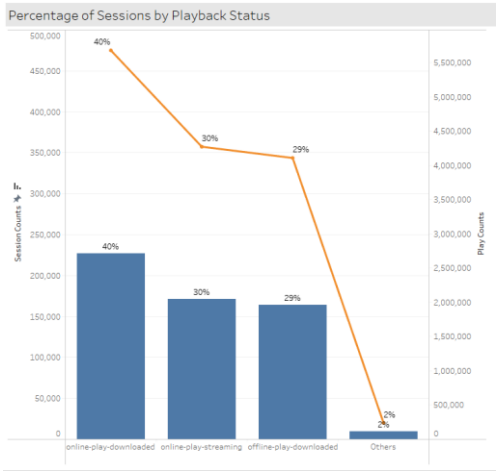

圖 8、 Percentage of Sessions by Playback Status

播放方式集中在 online play downloaded, streaming or offline play downloaded，40%sessiononline play downloaded，各約 30%session online streaming or offline play downloaded 。

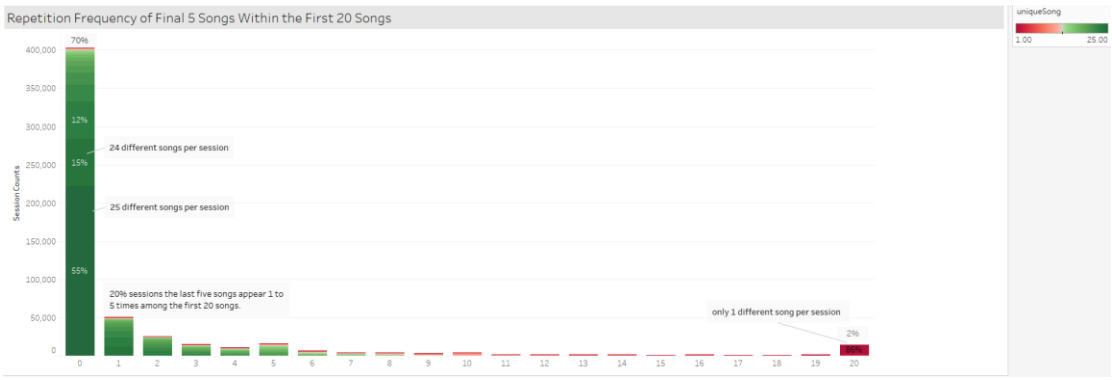

圖 9、 Repetition Frequency of Final 5 Songs Within the First 20 Songs

從後 5 首 (21~25)在 前 20 首(1~20)出現過幾次來看，70%的後 5 首在前 20 首沒出現過，可能比較喜歡聽新歌，20%的 session 後 5 首在前 20 首歌有出現過 1~5 次，2% 後 5 首在前 20 首歌出現 20 次。

從完整 25 首歌的 session 來看，70%的 session 是後 5 首在前 20 首沒出現過(喜歡聽新歌的人)，其中有 70%完整 session 有 24、25 首不同的歌，2%後 5 首在前 20 都有出現過的，其中 86%的人完整 session 只有一首歌，也就是整個 session 都在單曲循環。

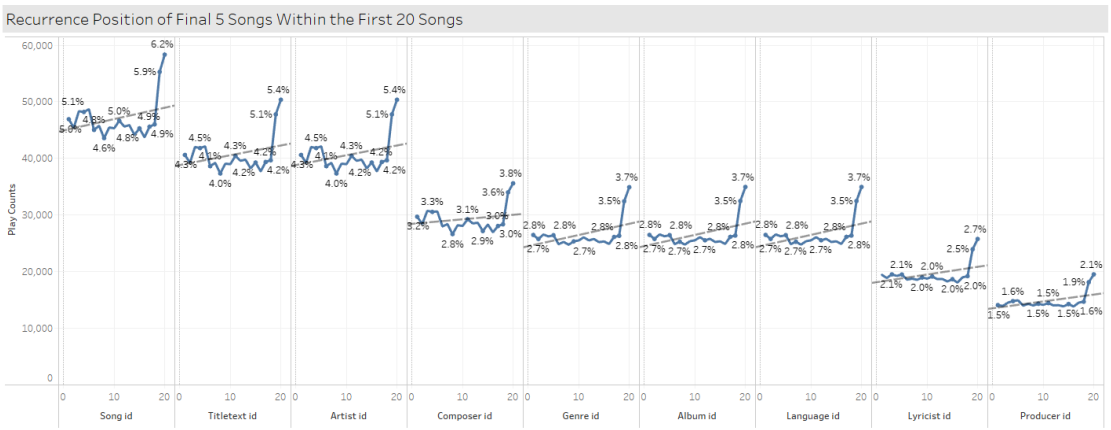

圖 10、Recurrence Position of Final 5 Songs Within the First 20 Songs

最後兩首歌(19~20)、歌的標籤、歌手、作詞、作曲、製作、專輯、語言、曲風比前 18 首有更高機率會出現在後 5 首歌。 6%播放次數後 5 首歌重複出現在第 19、20 首，5%播放次數後 5 首歌的歌手、標籤會重複出現在第 19、20 首。

參、 推薦系統 Ngram & Language Models with Jelinek-Mercer Smoothing

一、Ngram

主要問題是我們應該要怎麼有效去根據 25 首歌的聆聽順序去知道一首歌有多大的可能性出現在用戶的聆聽序列裡面?

大家都知道 n gram 會很好，但在冷門歌曲會怎麼樣?我在數據集中隨便找了('09afc5a29b686b0dfff1a3a5317398d4', 'ab87181893acfc80a465c39a443ea08b')這筆數據，如果採用單純的 n grams 的話，會得到

{'f223a183a28418c403c63955d79d9bda': 1,
'56cf3ecdd90729f015369c98151eb19b': 1,
'b26327d403a1101f32effb959d0128cd': 1}

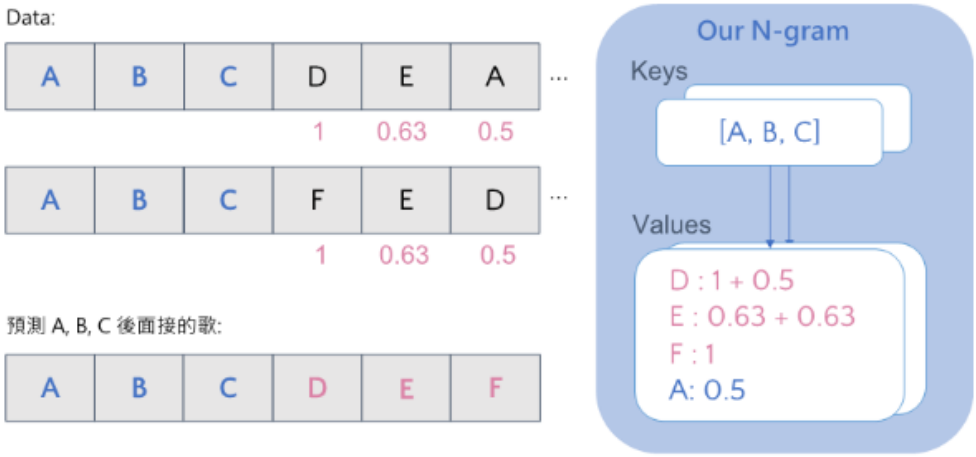

圖 11、N gram 示意圖

這樣做的好處是什麼，可以在我們找到的內容中看到。

[('56cf3ecdd90729f015369c98151eb19b', 2.13),
('f223a183a28418c403c63955d79d9bda', 1.63),
('b26327d403a1101f32effb959d0128cd', 1),
('7ba90df9bff5c864d5f89d6fdc36b664', 0.63),
('56ac6c4b6fb9b5a38dafdacaa718cb4e', 0.5),
('1858e0efb54fd28fd852355eef289288', 0.5),
('9b1683a0af39085ec55fe706506518eb', 0.43),
('5bec1696b6167510be10e389f3b2b2d7', 0.43),
('f1d9e6c5188a9b6c862704e4f4ef7e96', 0.38),
('9dcdc0dbd95d76bc98b938a1c4e2552f', 0.38)]

只要使用者聽過('09afc5a29b686b0dfff1a3a5317398d4', 'ab87181893acfc80a465c39a443ea08b')這個順序，56cf3ecdd90729f015369c98151eb19b 這首歌曲分別在他後面第一、二、三首被播放。

而 f223a183a28418c403c63955d79d9bda 在後面第一二首被播放，這說明了在清單之外，使用者聆聽的歌曲其實是有模糊的先後順序關係的，而我們的結構在這部分可以直觀的捕捉。

我們認為只要蒐集的歌曲樣本夠多，就可以簡易的拼湊出用戶可能會喜歡的歌曲和進行推薦，而在 n grams 中，這樣的模糊對應關係其實是會被吃掉的，所以這是我們的 n grams 在冷門歌曲上也會有一定成效的關係，跳脫猜測

歌曲清單的範疇，我們做的比較像是把機器學習要學的東西直接拿出來做估計。

二、Language Models with Jelinek-Mercer Smoothing

但是 n grams 雖然很好，可是只有一半的歌曲能用，剩下的歌曲應該怎麼辦?

```
artist_id_297472
album_id_4091453
language_id_62
genre_id_abdc0a50aff67c591737bc6f57a36e09
session_1 session_137043 session_324391 ...
```

圖 12、Language Models with Jelinek-Mercer Smoothing 示意圖

我們使用了 Language Models with Jelinek-Mercer Smoothing，將 100 萬首歌曲當成是 100 個 document，而歌曲的 meta data(歌手、專輯、曲風、語言)和聆聽這首歌的 session 組成 document 的內容來進行傳統的網路搜尋行為並做調參。

因為我們發現熱門的歌曲在推薦熱門歌序列的時候很有效，但在冷門歌曲推薦上，我們發現著重在全局統計上，調低熱門歌曲的影響力會效果更好，另外使用這個方法可以讓推薦系統在不重覆推薦用戶聽過的歌曲的情況下，達到比直接推薦後五首歌或 random 更好的效果，最終來對使用者進行有效推薦。

實務上的資料我們也對新歌曲做了很多考量，在簡報的時候我簡單提了一個每天把舊數據乘以 0.98 的行為，但好像沒人聽懂可以對歌曲產生多大的影響，這代表可以透過低成本的方式每天迭代資料集，原本的聆聽行為過了 30 天就會剩下一半的影響力，權重低於一定值也可以把歌曲踢出模型，讓模型不會無限變大。

我們認為這樣會比深度學習更好來做調控，用戶自己進行的點閱也可以透過放大權重的方式來快速讓模型轉變，但統計模型聽起來就沒有那麼潮，在解釋我們模型運作的時候，如果沒有一直摸，感覺大家也很難馬上理解有多厲害，所以就，好吧。

肆、推薦系統 Pyserini

一、Search song by pyserini

此方法是一種 Sparse retrieval 的相似文件搜尋應用，將每首歌以 meta-data 和出現在哪些 session 中當成文件， Session 內的數首歌則當成 Query，並各種 Ranking function 搜尋出最相似的五首歌。我們曾經嘗試過的 Ranking functions 包含了 Vector Space 的 BM25 和 language model 的幾種不同 smoothing 方法。

基於我們 EDA 分析的 1-20 首的哪些歌的 feature 最容易在 20-25 首出現及實驗結果，我們觀察到：同個 session 內越後面的歌越重要，因此在使用相似度搜尋時的 query 都以 session 內的第 16-20 首為主。

而在實作時，我們遇到 query 的 token 數太多的問題，我們試過直接截斷與增加上限，兩者的分數很相近。但更合理的做法是挑選最適合、有價值的 term，對於這個方法，我們實驗了兩個比較具代表性的方法，兩種方法都是設定相同 token 數量的上限，從第二十首開始加 term 直到上限，一種是從最熱門的 term 開始選，分數為 0.297 ，另一種則是從最冷門的 term 開始選，分數則為 0.20。這實驗告訴我們冷門與熱門的 term 有顯著的差異，因此也讓我們開始嘗試對 session 分群，希望能找出一些線索。

表 1、Query 處理方法與分數

Query 處理	Last N	Ranking function	score
直接截斷	2	BM25(k1=2, b=0)	0.26248
增加上限	3(*)	BM25(k1=2, b=0)	0.25802
直接截斷	2	MLE Dirchlet smoothing	0.27197
增加上限	2	MLE Dirchlet smoothing	0.27
挑選熱門的 term (選 term df 越大)	3	MLE Dirchlet smoothing	0.2978
挑選冷門的 term (term df 越小越優先，df = 1 不算)	3	BM25(k1=, b=0)	0.20

註：N 為採用最後幾首當 Query

二、使用 Pyserini 分群實驗

我們依照 EDA 分析專業組員的建議，將資料先用最簡單的方式將每個 session 用第 20 首的 play status 分成三組，第一組的 play-status 是 online-play-downloaded (40%)，第二組的 play-status 是 offline-play-downloaded (30%)，其他一組 (30%)，這一組最大宗為 streaming。接著，我們將資料分組，分別建立三個獨立的 corpus，再使用與先前相同的 sparse retrieval (pyserini) 方式，發現三個組完全搜尋不出任何結果的 session 數量有顯著的差異：

表 2、分組搜尋結果

組別	無任何搜尋結果的 session (無結果數量/該組總 session 數)
Offline	12.7%
Online	35.15%
Streaming	2.21%

對於這個結果，我們有以下的推測：offline 是每個使用者自己聽的歌曲，相對可能是這次競賽中最難的部分，而 Streaming 組可能會是很熱門、好預測的歌。這是一個很意思的結果，礙於時間因素沒有繼續研究下去。

三、Item-CF 與使用者偏好

ItemCF 的演算法使用物品之間的相似性來推薦用戶，我們嘗試了 ItemCF 演算法的 baseline 版本:所有聆聽的權重都相等(不考量聆聽時長等等的 Implicit feedback），就能取得 0.20 分。後來，考量到「聆聽時長」對於使用者來說一定程度的反應了他們對這首歌的偏好,於是我們將每首使用者所聆聽的歌曲以聆聽完成度（Complete ratio: 這首歌聽了多久/歌曲長度）計算相似度矩陣，最終得到的分數反而是更低的 0.1054。由此可推論，這次我們要預測的目標歌曲不一定是使用者喜歡的，反而是 KKbox 推薦但使用者不一定喜歡的歌，EDA 分析的切歌率也證實了本次資料集沒有忠實地反應使用者偏好。

表 3、方法與分數

Method	Score
Item-CF Baseline	0.2055
Item-CF By Complete Ratio	0.1053

伍、推薦系統 Deep Learning

本次比賽提供的各項資訊，皆缺乏自然語言訊息得以分析，在這點上，似乎不適合套用語言模型；然而，根據 train_source、train_target 與 test_source 提供的歌曲聆聽順序，可以將各個 session 聆聽的 25 首歌視為一句話，每首歌的 id 便是一個獨立字符，因此每個 session 的歌曲序列可以看成由 25 個「字」組成的「一句話」，得以用 NLP 的方法來處理──整個問題可以看成從「20 個字組成的一句話」來預測「後面 5 個字」。可見，本次比賽預測五首歌的任務，可以視為文本序列預測或分類預測的問題；以深度學習模型的任務來說，便是 multi classification、sequential learning、sequential generation。

就 multi-classification 來說，Bert 是較適合的模型。由於 self-attention 的機制，Bert 可以抓住語句的順序訊息，在分類問題上也表現優秀。在此，train_source 內每個 seesion 的 20 首歌變成了 features，而 Bert 透過這 20 個 features 來預測一個 label，也就是第 21 首歌；接下來，可以把預測後的共 21 首歌，再輸入 Bert 來預測下一個 label，也就是第 22 首歌，以此類推成 25 首歌。

若是處理 sequential learning，LSTM、GRU 會是首選。由於 LSTM 內部設計的 memory cell，LSTM 可以在較短的句子內學到其內的組成順序資訊，也就是 contextual structure；GRU 是輕量化的 LSTM，可以有效改善 LSTM 運行效率太長的問題，其效能也不太受影響。因此，將 20 首歌的句子送入 LSTM，請模型接續後面 5 個字（歌）。此次實驗結果約為 0.19，本次還另外實驗了 ItemCF 搭配 GRU 做了許多嘗試，最好的結果約為 0.27。

執行 sequentail generation 的任務，Bart 會是相對適合的模型。Bart 由於有 decoder layer，因此適合做 sequentail generation；再加上，Bart 的預訓練是將文件拆散，並重新組合成文件的方式來訓練模型，因此 Bart 比 Bert 更能抓住文字語句順序的訊息。然而，本次嘗試失敗，Bart 將 song_id 的字串當成一個獨立字符，因此輸出結果犧牲了歌曲 id 的順序資訊，等同 model 並沒有學習到此次訓練資料的資訊，此次 Bart 的嘗試仍需做其他調整。

總體而言，將本次問題套用深度學習模型，於 Kaggle 取得的分數並不好，相對於一般統計模型便可輕易達到 0.5 的成績，可見本次任務並不適合使用深度學習模型。

陸、結論與建議

我們認為我的模型才是最接近商用的，因為後五首歌曲會大量跟第 19、20 首歌重複，所以其他組基本上都會 ngrams 找不到就直接填。

但我們覺得這樣根本是不能實用的，透過 Language Models with Jelinek-Mercer Smoothing 做到不用推薦用戶聆聽過的歌曲還可以找到冷門用戶感興趣的歌曲造成效能提升，但評審好像不覺得最後五首歌直接填重複的推薦出去有什麼關係。

另外我們的 n grams 定義其實跟其他 n grams 定義不太一樣，所以才能天然比其他 n grams 找到更多東西，其他組的 n grams 其實只適合拿來推斷 NLP 模型，但我們認為沒有那麼適合拿來做歌曲推薦，我們的研究脈絡其實是因為 GNN 效果沒那麼好，乾脆把 GNN 學習的機率直接抽出來成為一個簡單的統計模型。

有些組的 n grams 目的是為了抄歌單所以一路選擇最大機率往後推，但是其實冷門歌曲他很大可能接下來的所有歌曲只有聽一到兩次，根本沒有在歌單裡面，要怎麼捕捉這種模糊行為其實才是我們魔改的核心，然後分數根本沒有優勢，感覺做的東西好像被覺得很容易。

還有其他人的相似度模型基本上是直接比較歌曲相似度，我們為了減少歌曲運算量和系統負荷，所以只取前面十個其他用戶聆聽紀錄就可以找到比直接比較更好的冷門歌曲搜尋結果。0.56-0.48~0.56-0.55 就是我們的模型能比別人多找到的東西。

然後為了能夠上真實系統集群運算，我們的 n grams 模型其實是手刻的，原本有預備算力不夠的時候直接把 n grams 拆成幾個獨立的字典來分別跑，Language Models with Jelinek-Mercer Smoothing 也為了節約算力只需要儲存最接近的 10 個 session 就好。

不同的計算也是使用不同筆記本，因為一天有二十次提交機會，所以我是拿筆電和桌機分別算不同的部份，同時迭代不同細節，才能一天出二十份結果和測不同的東西。

最後，實務上 NN 需要成本，Google search 也是 使用 Language model + 網頁之間互相 link 計算而來的，其實我不認為 NN 能學到什麼東西，因為這次沒有那麼多資料，要預測一百萬首歌，根據經驗至少需要每首歌被聆聽一百次以上的樣本才能比較好的做學習，然後樣本其實是不均勻的，我們有試過幾種 set、遮罩等等從統計學去模擬模型訓練的狀況來去探討為什麼機器學習訓練不起來，另外我不認為通用語言架構能夠適用於歌曲。

舉例來說，在通用語言模型中，「今天早上我要和朋友出去玩」和「朋友要和我今天早上出去玩」是同樣的意思，所以在訓練的時候，詞彙順序性不高。

但在聆聽行為上，聽完周杰倫再點去聽 K-pop 和聽完 K-pop 再點去周杰倫，用戶希望能得到的推薦清單其實是完全不同的，所以導致一些其他領域可以使用的機器學習模型會在這裡不準又消耗大量資源（雖然這裡感覺有點被很費資源拉走注意力的感覺）

這也是最後我們把架構砍到剩下兩個的原因，扣掉歌單，有出現過相似順序其實是可以根據分數去計算用戶偏好的，沒有出現過相似順序，也可以透過基礎語言模型去推斷哪些歌曲的用戶群是相近的，而不是透過集成模型的方式，我們覺得我們的模型最後簡化到可解釋性非常高。