Research Report

Recommender System

推薦方法依照考慮的資料來源分為兩種, 主要考慮物品屬性或使用者資料進行的推薦方法是 **Content-based/filtering**, 考慮物品和使用者或使用者之間”關連”(link)的方法是現在流行的 **Collaborative filtering**，Content-based 的做法直接考量描述

物品或使用者各自的內容，利用 Cosine similarity 類似的計算相似度方法找出相似的物品推薦，這種作法相對簡單，所需要的資料也較少，是早期推薦系統流行的作法。但隨著計算能力和演算法的提升，為了追求更高的推薦精準度，**Collaborative filtering** 已經是現在推薦系統主要的研究趨勢。

Collaborative filtering 有兩個主要的領域，一是 *Nearest Neighborhood* based method(在部分 paper 以 memory based 稱呼) ，另一個則是現在主流的 model based 方法，也就是以 *Latent factor models* 為基本 model 架構[5]，再加入其他元素作為延伸。

Nearest Neighborhood 藉由歷史紀錄找出與被推薦人有相似行為的其他使用者 i.e., 利用 $Jaccard\ similarity_{CJ} = {R_C \cap R_J}/{R_C \cup R_J}$，$R_C$代表使用者$C$的歷史紀錄，計算出其他使用者$J$和被推薦人的相似度，然後從最相似的"鄰居"們的歷史紀錄挑出物品推薦給使用者，假設現在要計算一個物品 P 給使用者 C 的分數$C_{P_{pred}}$。

$$C_{P_{pred}} = \bar{C} + {\sum_{J \in rates}(J_P - \bar{J}) \times r_{CJ}}/{\sum_J |r_{CJ}|}$$

r_{CJ}代表使用者 C 和鄰居 J 的相似度，J_P是鄰居 J 對物品 P 的評分。

以圖 2 [4]為範例：

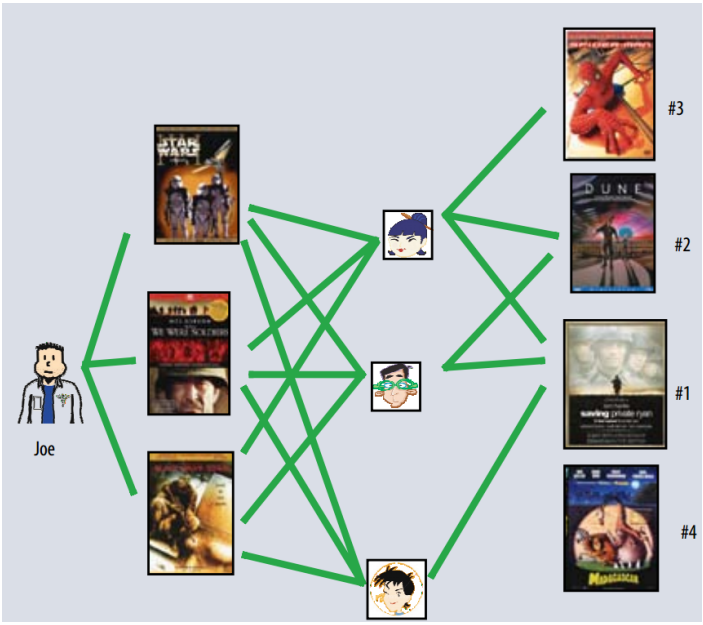


Figure 2.　　Nearest Neighborhood 方法示意圖

Joe 喜歡左邊三部電影，那 similarity 方法應該會找出也喜歡這三部電影的使用者(因為交集高)，中間三位使用者其他喜歡的電影如右，然後再找出這些電影的加權平均(權重是和 Joe 的 similarity 分數)作為推薦的分數，這裡應該會推薦出#1 的 Saving Private Ryan。

但是 Nearest Neighborhood based 最大的問題就是 data sparsity，當網站提供的商品種類和使用者越多，0 就會充滿整個矩陣，因為就算是活躍的使用者最多也不會留下超過 1%物品數的紀錄，這樣會使 similarity 很難計算因為交集很少，從而導致 Neighborhood 的效用降低(原意是找出相同興趣的人，但可能因為交集太少變成只偶然有相同的紀錄就被解讀為相同興趣)甚至是難產。另外就是 Nearest Neighborhood based 的方式難以規模化，因為光是尋找出鄰居就需要 O(user× item)的計算複雜度，一旦系統內的使用者和物品數增加就會導致推薦計算過於複雜。最後則是 Synonymy 的問題，在 IR 領域，這是一個被重視的問題，相同意涵的字詞照理該被自動歸屬到相同的詞或分類，而 IR 領域就是使用"latent"來解決此問題，在 Nearest Neighborhood based 的方法下沒有導入 latent 的概念所以成效也會打折。

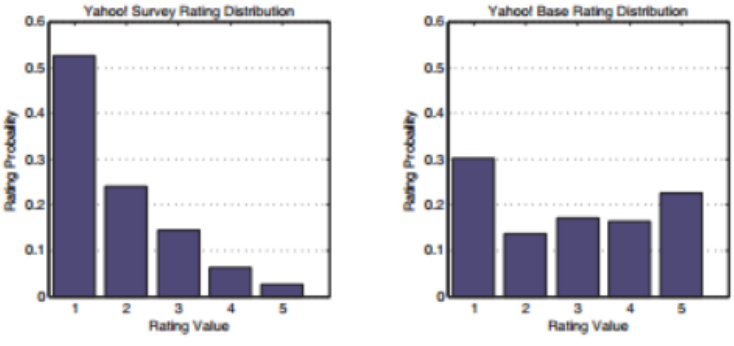
Missing Not At Random

推薦系統的概念就是要自動地去推薦每個使用者他可能感興趣的東西。那通常在正式上線給使用者用之前，都會需要做一些 offline testing 並用一些 measure 來評估 RS 的效用，那 recommendation accuracy 就是其中一個方式。推薦系統的文獻分兩種方式來評估推薦的準確度。然後分別用 Root Mean Square Error (RMSE) 或是 precision 和 recall。

那過去推薦系統的文獻在做的通常是去推測「使用者自己選擇要評分的那些 item」。也就是使用者評過分的那些物品取一部份出來當 training set，剩下的一些 pairs 拿來做 testing set，再來看這個推薦系統準不準。因為是使用者有實際評過分的資料，所以很容易在網路上取得，且很適合用 RMSE 來評估預測分數跟實際分數的差異。也因此 RMSE 常被用來當 rating prediction 的目標函式，就是要去最小化預測的評分跟實際使用者給的評分之間的差異。

但其實這些收集來的評分資料常會有 bias，因為使用者會自己選擇要評那些物品，而不是真正的隨機挑物品來做評分。也就是所謂的 missing not at random(MNAR)。

但上述這種「最佳化預測使用者評過分的物品的評分」方法所做出來的推薦系統，跟現實生活的



(a) Yahoo! Random　　(b) Yahoo! User

推薦系統的應用有差距。像是 Netflix 就是要給所有 video 一個分數，不管使用者有沒有評過分。像是下圖就是 Yahoo! Music 的 dataset，圖左是 Yahoo! 叫使用者隨機給物品評分，而這些分數可以當作使用者"沒有"評過分的資料，因為沒叫使用者填的話，他們實際上就不會去評過這些東西；而右邊則是一般推薦系統的情境，使用者自己選擇要評分的物品。可以看出兩個分數有所差距，且因為 data sparsity 的原因，通常實際的分布會比較接近左邊的分布，也因此在[8]裏頭提出類似 smoothing unseen rating 的方法來修正 MNAR 的問題。其中 $R_{i,u}$ 是 Model 算出來的預測值，前面則是實際觀測值。如果使用者 u 真的有給 item i 評過分的話就是那個值，否則為一個 predefined 的 smoothing 值(R_0)。w_0 不為 0，而且 R_0 設定成比平均觀測評分的平均還低。這個設定也有確實地反應了未知的機率分布大部分的值都偏低(可參考前面 yahoo 的圖)，相較於使用者自己選擇物品評分的情況。而傳統的 MF model 這邊的 w_0 就會設成 0。

$$\sum_{\text{all } u}\sum_{\text{all } i} W_{i,u}\cdot\left\{\left(R_{i,u}^{\text{o\&i}}-\hat{R}_{i,u}\right)^2+\lambda\left(\sum_{j=1}^{j_0}P_{i,j}^2+Q_{u,j}^2\right)\right\}, \tag{12}$$

$$\hat{W}_{i,u}=\begin{cases}1 & \text{if } R_{i,u}^{\text{obs}} \text{ observed}\\ w_0 & \text{otherwise}\end{cases}.$$

Matrix Factorization

自從 Netflix Prize 2006 年的競賽後，Matrix Factorization(後簡稱 MF)的方法已經普遍被認為是 state of the art 的 Collaborative filtering-Latent factor model 的實現方法。

MF 的優點在於：

1. 容易加入額外的資訊像是 implicit feedback，增加預測的準確度。
2. 可以找出隱藏在我們未知 dimensions 的關聯(latent) 。

最基本的 MF 形式是從使用者對物品的**評分模式**歸類出**使用者和物品**各自的 vectors of factors，MF models 將使用者和物品映射(mapping)到一個 ***joint latent factor space***，使用者和物品之間的互動(評分)就可以用**內積**來模擬，在一個多維的空間中找到相近的 user vector 和 item vector。

如下圖，我們可以看到使用者和物品在一個二維空間中以向量模擬的空間關係。

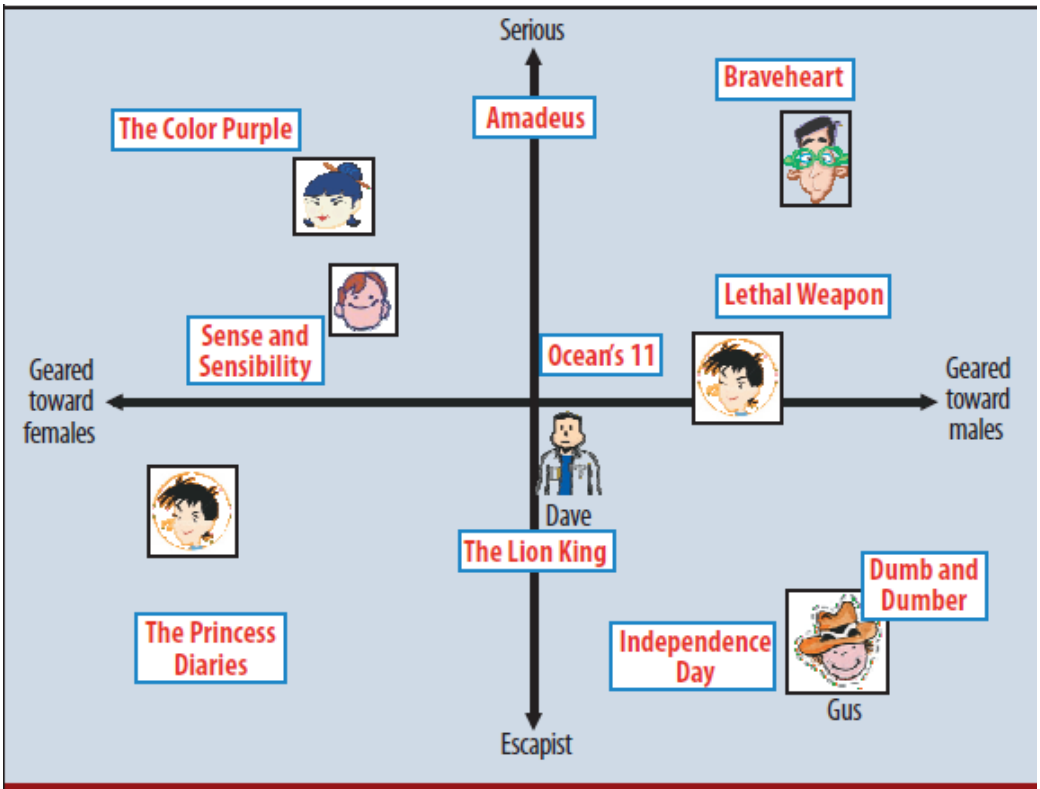


Figure 2. A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.

從上圖範例可以看出，我們要找出和使用者興趣相符的物品就是要找到在多維空間中和使用者相近的物品，利用內積我們可以估計使用者對物品的評分:
$\hat{r}_{ui} = q_i^T p_u$，從中找到分數最高即最符合使用者喜好的物品。
在 MF 裡最主要的挑戰及工作就是要計算出使用者和物品映射到一個 latent factor space R^f的向量，在 IR 領域中已經很常使用的 SVD 和這個問題很相似，但是在推薦系統這部分會遇到歷史資料 sparse 的問題，傳統的 SVD 是不能解一個不完整矩陣的問題。以前解決這個問題的方法是想辦法塞滿 input 的矩陣，不過這是一個高成本且容易導致結果不精準的作法，最近的研究方法都是直接 model 現有的觀測值(ratings)，利用 Machine Learning 的方法學習q_i ，p_u。

- Basic objective function minimizes RMSE

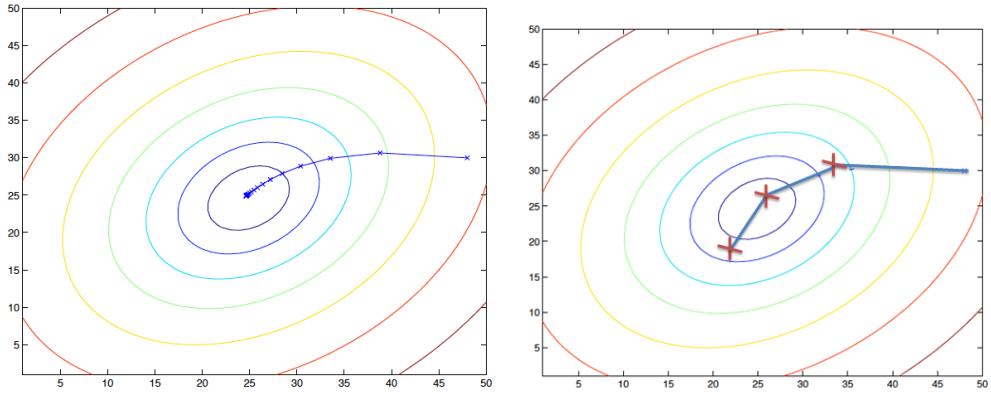
$$min_{q,p} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

MF 與其說是分解矩陣，不如說是在找出q, p factors 使得它們內積能夠近似於觀測值(input 的矩陣)，我們要把每一個存在於 K(training set)的 user-item pair 做出來的預測值越接近觀測值越好，後項則是為了讓學出來的 factor 能夠具有一般性，能繼續用於測試其他 set 的資料，避免過於 over-fitting training set，那在這個 ML 的部分最常使用的方法是 SGD 和 ALS。

Machine Learning in MF

Stochastic gradient descent

傳統 ML 用在線性回歸求解的 least mean squares 演算法，假設J(θ)是**誤差項**，θ是變數(factors)集合，則我們每次學習就是要更新θ： $\theta_j := \theta_j - \alpha \partial/\partial\theta_j J(\theta)$ ，α是 learning rate，α的意義是每一個**移動方向**的前進幅度，如下圖範例，左邊是較小的α，一次移動較小的距離，雖然需要比較多次的運算但是比較大的機率收斂，較大的α可以減少運算次數但可能因為移動幅度過大導致不會收斂。



而在 training set 做這個學習的演算法的方法會是：

Until 收斂{

For every factor j {

$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} \partial/\partial\theta_j J(\theta) = \theta_j - \alpha \sum_{i=1}^{m} (觀測值^{i} - 預測值^{i}) input_j$

}//$input_j$ 代表這筆 input 資料的第 j 個元素，在 MF 就代表第 j 個 vector

}

這種方式是 batch gradient descent，在算每一筆 training data 時都需要考量到整體的資料，所以會耗費非常大量的時間，所以現在大部分常見的都是使用 stochastic gradient descent，因為 SGD 實作上比較簡單且較快，在 SGD 演算法裡每一筆 training data 都只會讓**現在**的q, p factor **局部性**的進化一點。

Until 收斂{

For every training data i {

For every factor j {

$\theta_j := \theta_j - \alpha(\partial/\partial\theta_j J(\theta)) = \theta_j - \alpha(觀測值^{i} - 預測值^{i}) input_j$

}//$input_j$ 代表這筆 input 資料的第 j 個元素，在 MF 就代表第 j 個 vector

}

}

SGD 沿用 ML 的求解方法，首先初始化(猜) q, p factor，然後依照現在的預測值和觀測值的差距去不斷地更新現在的q, p factor。每一筆資料 rating 進來，一開始都會先計算現在的q, p factor 做出來的預測誤差：$e_{ui} = r_{ui} - q_i^T p_u$，然後同時調整q, p factor。

$$q_i \leftarrow q_i + \gamma \cdot \left(e_{ui} \cdot p_u - \lambda \cdot q_i\right) \quad p_u \leftarrow p_u + \gamma \cdot \left(e_{ui} \cdot q_i - \lambda \cdot p_u\right)$$

Alternating least squares

ALS 演算法優點在於目標函式可能不是一個凸函式，所以找最小值可能很難，但是只要將變數限制為一個，那就可以確定目標函式會是一個凸函式，所以 ALS 演算法在一個 iteration 內固定其他變數**只解一個變數**，ALS 演算法由於運算方式是獨立各個變數作微分學習，所以大部分用在可以平行化運算以及 dataset 比較集中(不 sparse) 的情境。

Until 收斂{
    For every factor j {
        Fix other factors, solved factor j, $\theta_j := \theta_j - \alpha(\partial / \partial \theta_j J(\theta))$
    }// 一元多次項的微分比起偏微分好運算
}

Matrix Factorization Extension

MF 由於其易擴充、延伸的特性，所以這邊將幾個常見的變化整理出來。

Bias models

大部分 dataset 都會具有 biases，像是評分網站的紀錄大部分都是使用者給自己喜歡的項目評分，所以紀錄都會有偏向高分的 bias，在這種情況下將$q_i^T p_u$視作分數項目裡的變異數項會比較有辨別力，預測值$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u$ (μ是所有 ratings 的平均，b_i是 item i 的評分平均和μ的差項，b_u是 user u 評分平均和μ的差項) ，目標函式也被改寫為

$$min_{q,p} \sum_{(u,i) \in K} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2 + {b_i}^2 + {b_u}^2)$$

Dynamic temporal models

在 dataset 具有時間性的資料，能夠表現出時間差異性質的情況下，MF 也可以藉

由添加 temporal bias 項將這種時間變化的效果也加入 model 裡，定義一個時間變化的(t) function，並將可以被(t) function model 的因子帶入，這篇的舉例是認為 item 和 user 的 bias(item 可能在某些時間會特別流行 , user 可能某段時間特別偏好給高分)都是會受時間影響的，以及使用者喜好會隨時間改變(這裡則認為物品的屬性不會隨時間變化) ，預測值則以下式表示：

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T p_u(t)$$

Learning to Rank (LTR)

一般 Ranking 的問題，evaluation 主要是透過能顯示 ranking 品質的 evaluation metrics , 如 Mean Reciprocal Rank(MRR), Mean Average Precision(MAP), Normalized Discount Cumulative Gain(NDCG)或是 Rank Correlation(RC)。

LTR 為機器學習(machine learning)中一種監督式學習(所以需要 training set)， 有別於傳統的 IR ranking model(如 BM25, LMIR)，不會因為要讓上述的 evaluation metrics 變好看，而一直拿 validation set 來 tune 參數，參數是固定的，而是想辦法去 learn 出怎樣組合 feature 最好(feature 後述)；LTR 也可以避免 overfitting 的問題，目前不只在 IR 領域，在其他領域如推薦系統都有應用 LTR 的例子。

因為 LTR 本身從機器學習來 , 簡單介紹機器學習的 framework , 有(1)Input Space, (2)Output Space, (3)Hypothesis function h, (4)Loss function。Input Space 多為 feature vector；Output Space 為我們想得到的結果，如果是做回歸則 Output Space 的範圍應為實數 R；Hypothesis function 則將 Input Space 的 feature vectors 做一些運算後來對應到 Output Space；而最後的 Loss function 很重要，因為它定義了什麼預測是對的，什麼不是對的。通常最好的 hypothesis h() 就是透過最小化 loss function 的值而得到的。

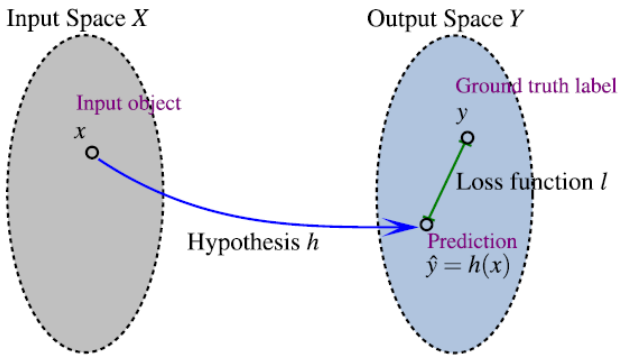


Fig. 1.5 Machine learning framework

廣義地來說，只要使用機器學習的 framework 來解決 ranking 問題的方法都叫做 "Learning to Rank”, 像是 relevance feedback; 但狹義地來說, 符合(1)Feature-based, (2)Discriminative Training 兩特性的才稱為 LTR method。

Feature-based 代表一文件或是推薦的物品要以 feature vector 的形式來表達，而這一些 features 一定程度上就分別表示文件跟 query 的關係、item 跟使用者的關係(使用 LTR 的好處是可以擴充 feature vector, 但如果我們使用 MF 的 latent factor 可能沒那麼容易增加新的 feature)。

Discriminative Training 代表此 LTR approach 有自己的 Input space, output space, hypothesis function, loss function，不需要額外定義一個機率模型來描述 object 的產生機率、預測的準確度。

因為 learning to rank 也是監督式學習，所以也需要 training 資料，以 IR 為例，其中包含了很多組 training query，每個 query 都有對應的關聯(associated)文章，還有解答(relevance judgment)。然後使用特定一個 learning 演算法去想辦法 train 出一個 ranking function，讓這個 function 可以有效地把所有 feature 以某種形式組合起來, 拿來預測 training data 的 ground true label, 至於預測的結果跟 ground true 差距越小越好(差距以 loss function 衡量)。接著 train 到最好，再去預測 test data 並排序文件給使用者 ranking list。

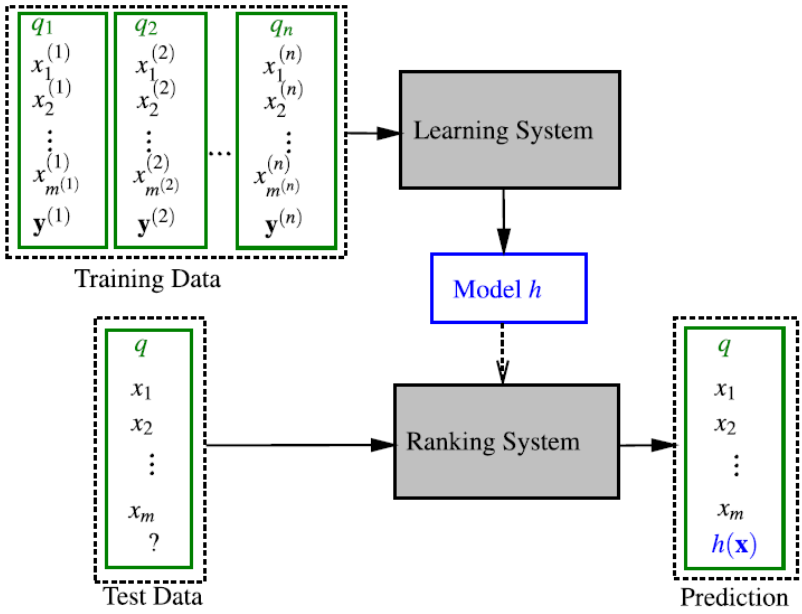


Fig. 1.6 Learning-to-rank framework

Pointwise Approach 就是拿一篇文章的 feature vector 當 Input Space，

Output space 則為此文章的相關程度(分數值),這樣的缺點是沒有考慮 Query level 跟 Position 的問題，可能造成對 loss function 不必要的額外 loss(像是相關文章多的 query 結果還有表現特別不好的 query 結果可能會 dominate 整個 loss function 的結果)。

Pairwise 的 hypothesis space h()理論上有兩變數(兩篇文章)，output space 則為兩者的大小關係(1 or -1)。但有些演算法還是會先用一個 scoring function f()來算出針對一個 query, 一篇文章的相關分數,再以 f()出來的分數為基準,比較兩文件，定義 h()。 $h(xu, xv) = 2 \cdot I\{f(xu) > f(xv)\} - 1$。

Listwise Approach 理論上 Input Space 跟 Output Space 都是 list，但實際上 Hypothesis function h() 通常也是透過一個 score function f()來為每個物品算出一個分數,最後 h() 再利用此分數排序出一個列表,像是我們在[7]看到的「Top one probability」就是一個 score function，它計算一個 ranking list 裡某一個物品被排在 top one 的機率，而 Hypothesis funcction 以此來當作排序物品的依據，最後產生一個給使用者的推薦列表。

Table 1.2 Summary of approaches to learning to rank

Category	Pointwise		
	Regression	Classification	Ordinal regression
Input space	Single document x_j		
Output space	Real value y_j	Non-ordered category y_j	Ordered category y_j
Hypothesis space	$f(x_j)$	Classifier on $f(x_j)$	$f(x_j)$ + thresholding
Loss function	$L(f; x_j, y_j)$		

Category	Pairwise	Listwise	
	–	Non-measure-specific	Measure-specific
Input space	Document pair (x_u, x_v)	Set of documents $\mathbf{x} = \{x_j\}_{j=1}^m$	
Output space	Preference $y_{u,v}$	Ranked list π_y	
Hypothesis space	$2 \cdot I_{\{f(x_u) > f(x_v)\}} - 1$	$sort \circ f(\mathbf{x})$	
Loss function	$L(f; x_u, x_v, y_{u,v})$	$L(f; \mathbf{x}, \pi_y)$	

待補充: 近期運用 LTR 的 & 第一篇用 LTR 在 MF 上的

Reference:

[1]Matrix factorization techniques for recommender systems (IEEE 2009)
[2]Learning to Rank Social Update Streams (SIGIR 2012)
[3]Diffusion-aware Personalized Social Update Recommendation(RecSys' 2013)
[4]On top k recommendation using social networks(RecSys 2012)
[5]Sorec: Social recommendation using probabilistic matrix factorization (CIKM 2008)
[6]Learning to recommend with social trust ensemble (SIGIR 2009)
[7]List-wise Learning to Rank with Marix Factorization for Collaborative Filtering(RecSys' 2010)
[8]Evaluation of Recommendations: Rating-Prediction and Ranking(RecSys' 2013)
[9]Informational Friend Recommendation in Social Media (SIGIR 2013)
[10] Collaborative Personalized Tweet Recommendation (SIGIR 2012)
Cousera Standford Machine learning https://class.coursera.org/ml-006
Standford Machine learning 學習筆記
http://www.gtwang.org/2013/07/standford-machine-learning-1.html