# Matrix factorization

## 1. Introduction

Matrix Factorization (MF)是一種目前應用在推薦系統(recommended system)問題上成功的方法之一[1]，MF 能將龐大的稀疏矩陣降維成兩個維度較小的矩陣相乘，使所需的模型參數數量大幅降低以提升效率，並間接訓練出含有對 rating 有豐富訊息的 latent features。MF 的訓練速度是目前仍可以再改進的問題，本報告將會介紹其中一種解決方法—在技術層面上以平行化 shared-memory 的方式改進其運算速度[2]。2010 年 Rendle 提出了一個應用更廣泛的模型—Factorization Machines (FM)[3]：MF 可視為 FM 的一個 special case，FM 不僅可用於推薦系統，對於解稀疏特徵的分類問題 (classification problem with sparse feature)也十分有效。

本報告將依序介紹推薦系統(2nd section)、Matrix Factorization 與其平行化的應用(3rd section)，以及 Matrix Factorization 與 Factorization Machines 的關係(4th section)。

## 2. Recommended System

推薦系統(recommended system)的解法主要可以分為兩種策略： Content-based approach 與 Collaborative filtering[1]。Content-based approach 主要是分析使用者與商品個別的特徵屬性組合(user profile and product profile)，將雙方的特徵比對計算關聯性後再推薦較高關聯性的商品給使用者。Collaborative filtering 主要是藉由使用者對不同項目的評價找出一群與目標使用者喜好最相近的一群使用者(Nearest Neighbors)，依據同一群的其他使用者與目標使用者的相似度、以及這些同一群內其他使用者對項目的評價，預測目標使用者對尚未見過的商品之喜好程度，通常以數值表示、或列出目標使用者可能潛在喜愛的項目清單推薦給目標使用者。

Collaborative filtering 可依算法分為兩種：Neighborhood method 與 Latent factor models。Neighborhood method 藉由計算使用者彼此之間(User-oriented approach)、或商品彼此之間(Item-oriented approach)的關聯性來產生推薦，以 Item-oriented approach 為例：此方法在評估一位使用者對某項商品的喜好時，會根據該使用者評價過的項目找出類似該使用者其他未見過的商品再排序推薦給該使用者。Latent factor models 簡單來說就是一種能夠基於使用者的行為對項目進行自動聚類、也就是把項目劃分到多種不同潛在的類別或主題，而

這些類別或主題可以理解為使用者的興趣指標，本報告所介紹的 Matrix Factorization 即是一種最成功的 Latent factor models 之一。

# 3. Matrix Factorization (MF)

## A. A basic matrix factorization model

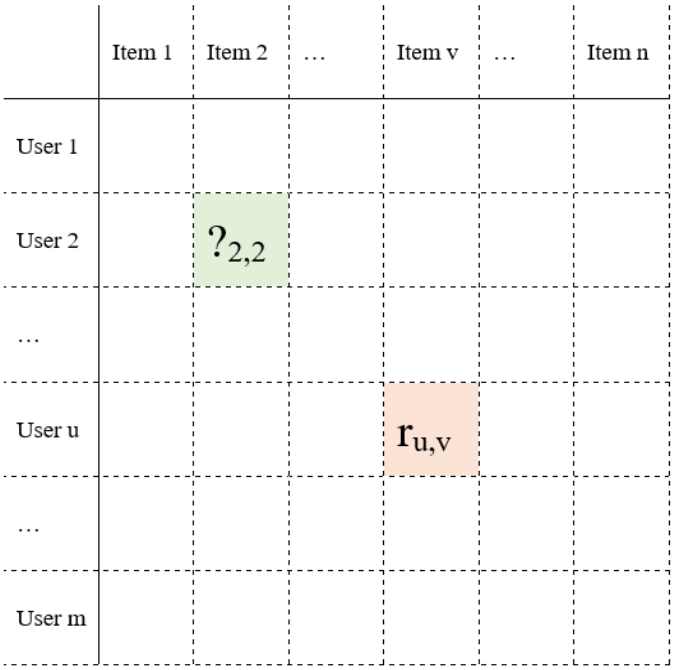


Figure 1. Rating user-item matrix R

圖一矩陣 R 中，m 表 user 個數，n 表 item 個數，$r_{u,v}$ 為 user u 對 item v 之 rating 值。在推薦系統中通常一個 user 評價過的 item 數量很少，因此矩陣 R 通常會是一個稀疏矩陣。

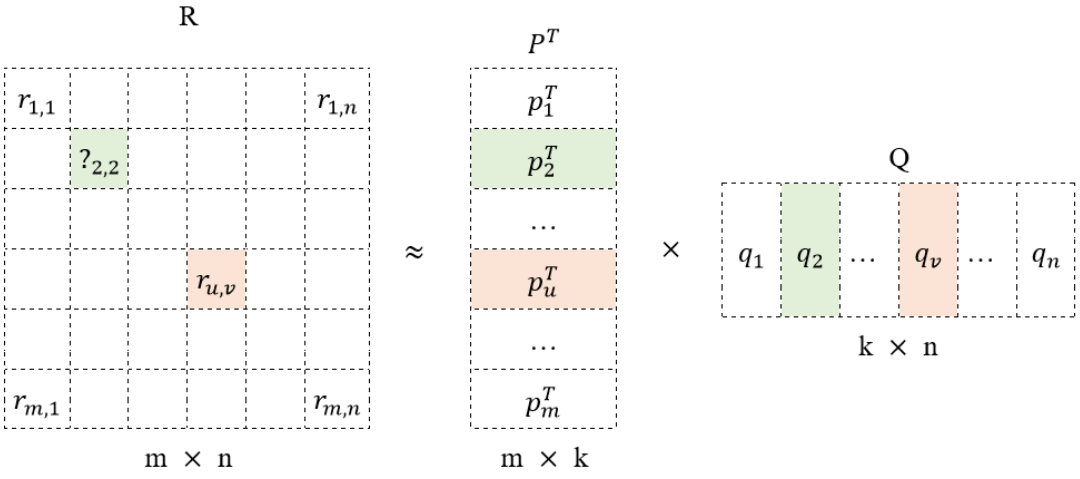

Figure 2. Matrix factorization for matrix R

將 R 矩陣分解後可以得到 $P^T \times Q$，如圖二所示，$P^T$ 、Q 分別為 m × k、k × n 的矩陣。其中 k 為 user 與 item 投影到的 latent factor space 的維度大小，$p_u$ 為在 latent factor space 中用來代表 user u 的 latent factor vector，$p_u$ 中的 k 個 elements 可以想成是 user u 對 k 個 latent factor 的不同偏好；而$q_v$ 為在 latent factor space 中用來代表 item v 的 latent factor vector，$q_v$ 的 k 個 elements 則可以想成是 item v 在 k 個 latent factor 具有不同程度的屬性；$r_{u,v}$為$p_u^T$、$q_v$ 之 dot product ($p_u^T q_v$ )。

Matrix factorization model 解的 non-convex optimization problem 可表示為[1]：

$$\min_{P,Q} \sum_{(u,v)\in R} ( (r_{u,v} - p_u^T q_v )^2 + \lambda_P \| p_u \|_F^2 + \lambda_Q \| q_v \|_F^2) \quad (1)$$

其中$\lambda_P$與$\lambda_Q$為 regularization parameters。若限制目標函數(1)式使矩陣 P、Q 的所有 elements 皆為正數或為 0，則此目標函數可延伸為解 Non-negative Matrix Factorization (NMF)。在 MF 中解(1)式可分為兩種方法：Stochastic gradient descent (SGD)與 Alternating least squares (ALS)[1]，在下一個 section 會逐一介紹。

## B. Learning algorithms

### a. Stochastic gradient descent (SGD)

典型的 gradient descent 是看過所有的 training data 後再決定更新的 gradient 方向，而 SGD 則是每看過一個 training data(也就是矩陣 R 中每個元素 rating)就調整更新一次模型參數(矩陣 P、Q 的元素)，在實作上常可見 SGD 的收斂速度會比 gradient descent 快很多。SGD 的 update rule 可表示為：

$$p_u \leftarrow p_u + \gamma(e_{u,v} q_v - \lambda_P p_u )$$

$$q_v \leftarrow q_v + \gamma(e_{u,v} p_u - \lambda_Q q_v )$$

其中 prediction error $e_{u,v}$為：

$$e_{u,v} \equiv r_{u,v} - p_u^T q_v .$$

### b. Alternating least squares (ALS)

因為$p_u$、$q_v$皆為未知，因此目標函數(1)式為 non-convex。ALS 的觀念為：在解$p_u$時若將$q_v$的參數固定住，則解$p_u$的最佳化問題可視為 quadratic，因而能找到最佳解。而每更新一次$p_u$後，在用同樣的作法解$q_v$，也就是說換成將$p_u$的參數固定住、解$q_v$的最佳化問題，如此反覆依序下去直到問題收斂為止。

### c. Comparison between SGD and ALS

更新矩陣 P、Q 一次後 ALS 需花費 $O(|R|k^2+(m+n)k^3)$，而在看過每一個矩陣 R 的 elements 後 SGD 只需花費 $O(|R|k)$，且因 SGD 更容易實作，因此 SGD 多為解 MF 的首選。但在一些情況下，ALS 能比 SGD 發揮更好的效果，例如：當系統可以平行化時，因為在計算$p_u$時可以不受 user u 以外的其他 user factor 的影響，同理在計算$q_v$時也不受其他 item factor 的影響，因此 ALS 用於平行化系統將能大幅提升訓練速度(註：SGD 也可用於平行化系統[5]，但是會牽涉到更複雜的演算法設計，這部分將於下一個 section 介紹)。

## C. Parallel matrix factorization

改善 MF 的訓練速度一直是一個重要的議題，本報告在此 section 中會探討本校林智仁教授的 LIBMF 套件[2]實作方法(在本報告中只介紹在 shared memory 系統下優化 SGD 訓練速度的部分[5])。

LIBMF 在 SGD 方法上使用平行化運算的核心概念是將 rating matrix R 切割成彼此獨立的小區塊(block)，在平行化技術中首先要先能區分在計算到的每個步驟時哪些資訊不能被更動，下列先以在 rating matrix R 中計算到 rating $r_{3,3}$為例：

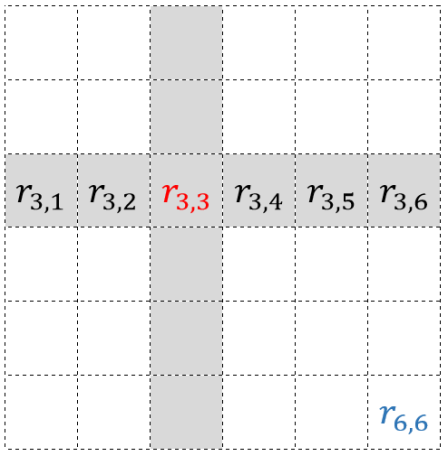

Figure 3. Partial rating matrix R

如圖 3 所示，在 optimization problem 中當$r_{3,3}$被選到時，圖中灰色部分的 ratings 不能被 update，而其他 ratings (例如：$r_{6,6}$)則允許同時被選出來 update。

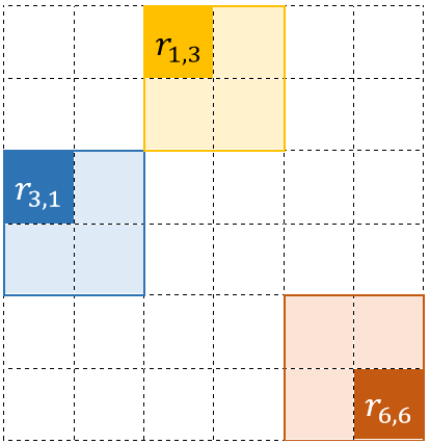


Figure 4. Splitting rating matrix R to blocks

將 rating matrix R 切為區塊後(圖 4)，因為不同區塊中的 ratings 不會共用 p 或 q，因此可將不同區塊餵給不同 thread 達到平行計算的目的。

在 LIBMF 實作中，針對平行化的演算法設計可區分為兩個部分：Block splitting 與 Partial random method，Block splitting 重點在於如何有效率地切出區塊以避免 synchronization time，而 Partial random method 則著重在 SGD 更新 rating 的順序選擇，下面會分別介紹這兩種設計方法。

**a. Block splitting：Lock-Free scheduling**

平行運算的設計在不同系統下需要著重的部分不同，以分散式系統為例，communication cost 是其首要考量的議題，因此在分散式系統中若想使用 SGD 且同時達到平行化的目的，可直接使用 naive 的方法 (DSGD-like Scheduling)：若有 T 個 nodes (機器)則直接將 rating matrix R 切成 T × T 個 blocks 即可。

但在 shared memory 系統中，synchronization 才是需要優先考慮的問題，因此若用這種 naive 的方法將會造成問題：舉例來說，若系統有 3 個 threads 且 rating matrix R 切成 3 × 3 個 blocks，當每

個 thread 皆在個別處理不同 block 時，若其中兩個 block 各需 20s 而第三個只需 10s 就能處理完，則處理第三個 block 的 thread 在另兩個 threads 仍在 busy 時就會呈現 idle。LIBMF 的 Lock-Free scheduling 可以避免這種 synchronization 問題：

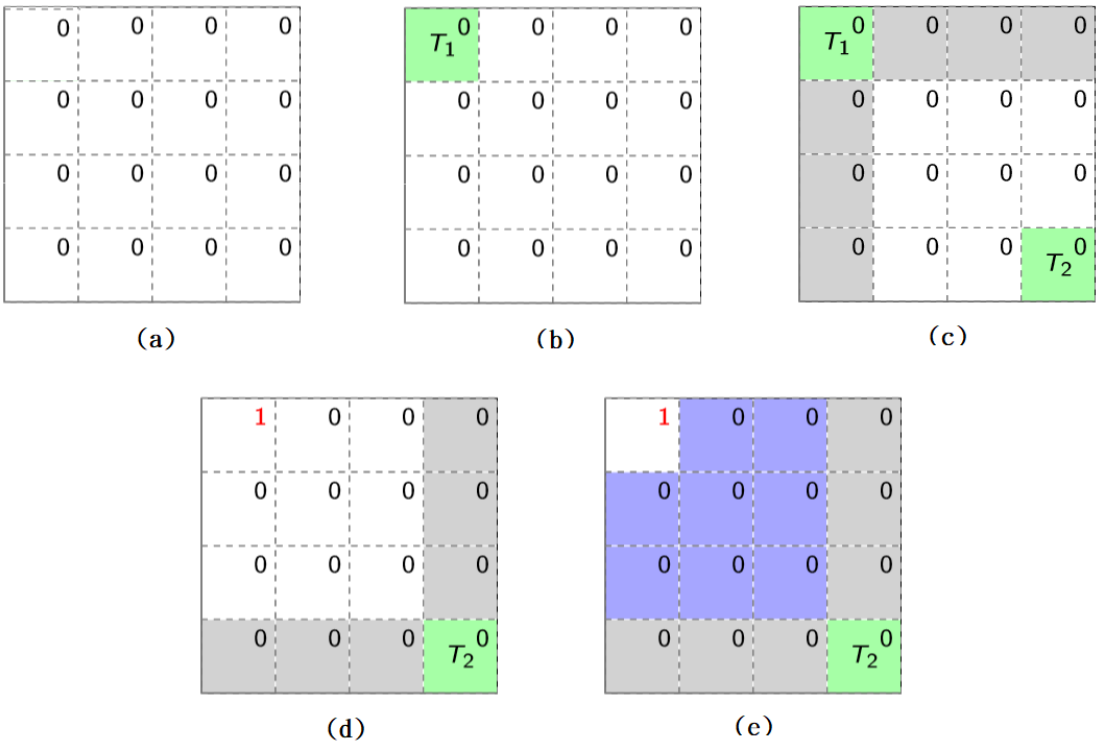


Figure 5. Process of Lock-Free scheduling

首先須先將 rating matrix 切出足夠多的 block 數量，如圖 5 將 rating matrix R 切為 4 × 4 個 blocks 可用於只有 2 個 threads 可用的系統中。每個 block 皆會記錄 update 次數，例如：圖 5(a)中每個 block 的 update 次數為 0 表 block 中每個 rating 皆未被 update 過。

- 當 thread $T_1$ 隨機選了某個 block 後(如圖 5(b)的左上角綠色的 block)，thread $T_2$ 只能從 thread $T_1$ 選中的 block 與該 block 同列或同欄的 blocks(如圖 5(c)綠色與灰色的 blocks)之外的其他 blocks 中隨機挑選 block。
- 當 thread $T_1$ 結束在該 block 的運算後，該 block 會更新 update 次數為 1，如圖 5(d)所示。
- 接著 thread $T_1$ 可從 update 次數最少的 blocks 中(如圖 5(e)紫色的 blocks)隨機選擇下一個 block。
- thread $T_1$ 與 thread $T_2$ 將反覆上述步驟直至收斂為止。

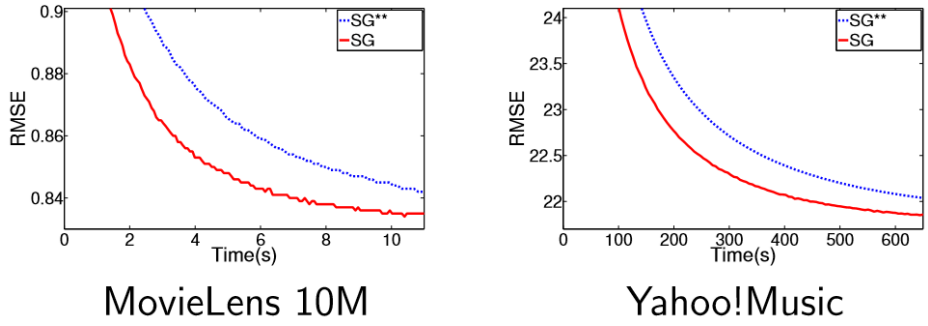


Figure 6. Lock-Free Scheduling (SG) and DSGD-like Scheduling (SG**)

由[5]中的實驗可知(Figure 6)，無論是在 MovieLens 或 Yahoo!Music 的 dataset 上，Lock-Free Scheduling 的效果均比 DSGD-like Scheduling (前面提到的在分散式系統中在使用 SGD 的情況下達到平行化所使用 naive 的方法)好。

**b. Partial random method**

在 SGD 中，ratings 的 update 順序可以有兩種選擇：Random 與 Sequential：Random 的優點為快速且穩定，缺點是會造成記憶體不連續，且隨著 training time 增長問題也會越大；而 Sequential 的優缺點則與 Random 相反。LIBSVM 在 SGD 的實作方法中結合了 Random 與 Sequential 兩種方法—對 rating matrix R 中每個 block R' 而言(block $R'=P'^{T}Q'$)：

- Random：Lock-Free scheduling 隨機選取 block R'
- Sequential：依序存取 R'(與 P'或 Q')的元素

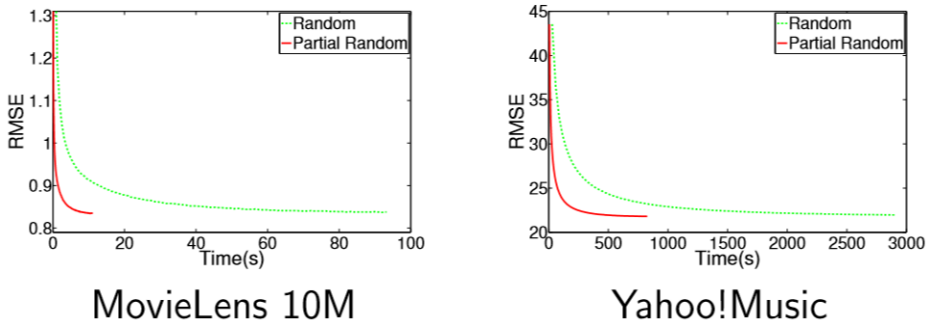


Figure 7. Partial random method and Random method (RMSE:方均跟差)

由[5]中的實驗可知(Figure 7)，無論是在 MovieLens 或是 Yahoo!Music 的 dataset 上，Partial random method 的效果均優於 Random method (隨機選取 block 並隨機存取 block 中的 rating)。

# 4. Factorization Machines (FM)

Rendle 在 2010 年提出了一個比 MF 模型應用更為廣泛的模型—Factorization Machines (FM) [3] (MF 為 FM 的一個 special case)，FM 不僅可應用於解 Ranking problem，也在解非常稀疏(sparse features)的 classification problems (或 regression problem)有非常好的效果。往後幾年隨著 FM 的發展與 KDD、kaggle 等比賽的興起，一種特別的 FM 模型—Field-aware Factorization Machines (FFM) [4] 從眾多機器學習模型中脫穎而出 (2012 年本校團隊使用 FFM 贏得 KDD 亞軍後，之後幾年的 KDD 比賽的前幾名隊伍幾乎都使用 FFM [6])。在這個 section 中將依序簡單介紹 FM 與 MF 的關聯，以及 FFM 與 FM 的關聯。

## A. From matrix factorization to factorization machines

為了方便介紹 FM 的概念，首先在這裡先將 MF 的 optimization problem (1)式省略 regularization term 簡化成：

$$\min_{P,Q} \sum_{(u,v)\in R} (r_{u,v} - p_u^T q_v)^2 \tag{2}$$

假設給予 user u 與 item v 的 feature vectors (註：在 MF 中可以用來表示 user u 偏好的 latent vector $p_u$ 或 item v 性質的 latent vector $q_v$ 是學出來的，在 MF 中只有給予 rating 值!)：

$f_u \in R^U$, U≡user 的 feature 數

$g_v \in R^V$, V≡item 的 feature 數

則可以想成是每筆 instance 的 feature vector 為$[f_u^T \ g_v]$而 target value 為 rating $r_{u,v}$，則解 **regression problem** 的目標函數為：

$$\min_{w} \sum_{(u,v)\in R} (R_{u,v} - w^T \begin{bmatrix} f_u \\ g_v \end{bmatrix})^2 \tag{3}$$

若考慮 user 與 item 間的 **interaction**，則可以借用 SVM 中 degree-2 polynomial mappings (poly-2)的概念直接將每個$f_u$ 中的元素與$g_v$ 中的元素相乘解：

$$\min_{w_{t,s},\forall t,s} \sum_{(u,v)\in R} (r_{u,v} - \sum_{t=1}^{U} \sum_{s=1}^{V} w_{t,s} (\boldsymbol{f_u})_t (\boldsymbol{g_v})_s)^2 \tag{4}$$

其中 t = 1,...,U,s = 1,...V。將(4)中加總 weight $w_{t,s}$運算部分改寫成矩陣表示形式後為：

$$\min_{W} \sum_{(u,v)\in R} (r_{u,v} - f_u^T W g_v)^2 \tag{5}$$

其中 $W \in R^{U\times V}$。然而此模型無法解 extremely sparse features 的情況：考慮最極端的情況，假設只給予 user ID 與 item ID 兩種 feature，則 user

vector $f_i$ 只有在第 i 個位置的 elements 值為 1、其他皆為 0，此時最佳解為：

$$W_{u,v} = \begin{cases} r_{u,v}, & \text{u,v} \in \text{R} \\ 0, & \text{u,v} \notin \text{R} \end{cases}$$

此時將無法預測u,v $\notin$ R 之 rating $r_{u,v}$ (在 optimization problem 中會因為 variables 數量(=mn) ≫ instances 數量(=|R|)因此無法預測沒看過的 data)。若要避免此 overfitting problem，可令上式中 W≈$P^TQ$ (P、Q 為 low-rank matrices)，此即變成 MF：

$$\min_{W} \sum_{(u,v)\in R} (r_{u,v} - f_u^T P^T Q g_v \ )^2 \quad (6)$$

若將此模型的應用情況 generalize 成應用於更 sparse 的 user 或 item 的 features，則可將 $Pf_u$ 與 $Qg_v$ 分別視為 user u 與 item v 的 latent representation，此即 FM。

### B. Field-aware Factorization Machines (FFM)

在 FM 中，模型需 maintain 每對 interaction (a pair of features)的 weight：以 interaction $\langle x_{j1}, x_{j2} \rangle$為例，每次看到不同 instance x 中出現 interaction $\langle x_{j1}, x_{j2} \rangle$時就會分別 update 模型 weight $w_{j1}$、$w_{j2}$ ($w_{j1}$、$w_{j2}$ 皆為長度為 k 的 vector，k 為 user-defined parameter)。FM 模型 formulation 可表示為：

$$\phi(\text{w,x}) = \sum_{j_1, j_2 \in C_2} \langle w_{j1}, w_{j2} \rangle x_{j1} x_{j2} \quad (7)$$

其中 $C_2$ 為 instance x 中所有非零的 elements feature 值的兩兩組合 (interaction)。

而在 FFM 中，一開始就會給予每種 feature 一個 field(假設每筆 instance 的 feature 個數為 n，field 個數為 f，f 必須小於 n)，在 maintain FFM 模型參數的過程中，若看到 interaction $\langle x_{j1}, x_{j2} \rangle$時會分別 update 模型 weight $w_{j1,f2}$、$w_{j2,f1}$，其中 $f_1$ 與 $f_2$ 分別為 feature $j_1$ 與 feature $j_2$ 的 field。FFM 的 optimization problem 為(以 Logistic Loss 表示 loss function)：

$$\min_{w} \sum_{i=1}^{L} (\log(1 + \exp(-y_i \phi(\text{w}, \text{x}_i))) + \frac{\lambda}{2} \|w\|^2) \quad (8)$$

其中 L 為 instance 數，λ 為 regularization parameter，FFM 模型 formulation $\phi$(w,x)為：

$$\phi(\text{w,x}) = \sum_{j_1, j_2 \in C_2} \langle w_{j1,f2}, w_{j2,f1} \rangle x_{j1} x_{j2} \quad (9)$$

## 5. Discussion and Conclusion

MF 至今已經有成熟的研究發展，本報告探討了在實際應用上可以如何

利用 MF 的特性加快運算速度，以 shared-memory 系統的平行化為例，切 rating matrix R 為 blocks 時，以 Lock-Free scheduling 方法安排 thread 選擇 block 時避免發生 synchronization problem，而在 SGD 更新 rating 的順序選擇上則同時採取 random 與 sequential 的方法以減緩 memory 不連續的問題。

MF、FM 與 FFM 皆為 optimization problems，但應用的情境有所不同。MF 為一種特別的 FM，MF 只適用於推薦系統 (當 rating 是唯一給予的資訊時)，而 FM 不僅適用於推薦系統，對於分類問題(除了 rating 資訊，亦提供 user 與 item 相關的 features)—尤其是當 feature 非常稀疏時特別有顯著的效果。本報告也介紹了另一種 FM 模型—FFM，FFM 的 data format 與 FM 或 MF 不同，FFM 需要額外的 feature field 資訊，其在 KDD 等比賽有非常亮眼的表現。FM(與 FFM)目前最需克服的問題是運算速度(MF 在運算速度改善方面的研究通常無法拓展用於 FM)，另一個可探討的議題則為如何開拓應用的情境或問題。

## 6. References

[1] Koren, Y., Bell, R.M., Volinsky, C. Matrix factorization techniques for recommender systems. *Computer* 42(8), 30–37, 2009.

[2] W.-S. Chin, B.-W. Yuan, M.-Y. Yang, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. LIBMF: A library for parallel matrix factorization in shared-memory systems. *JMLR*, 17(86):1–5, 2016.

[3] S. Rendle, “Factorization machines,” in *Proceedings of IEEE International Conference on Data Mining (ICDM)*, pp. 995–1000, 2010.

[4] Y.-C. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin. Field-aware factorization machines for CTR prediction. In *Proceedings of the 9th ACM Conference on Recommender Systems*. 2016.

[5] W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin. A fast parallel stochastic gradient method for matrix factorization in shared memory systems. *ACM Transactions on Intelligent Systems and Technology*, 6:2:1–2:24, 2015a.

[6] C.-J. Lin. Invited talk at SDM workshop on Machine Learning Methods on Recommender Systems, May 2, 2015.

## Note

藉由完成 final project 的過程中，我探討了 MF 的相關技術與應用，即使 MF 的歷史發展久遠，但是其應用一直在機器學習領域有著推陳出新的變化，尤其當我知道與 MF 有著這麼密切關係的 FFM 居然在近年 KDD 比賽中這麼有名，真的讓我感到非常訝異與有興趣!