**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)。

1. **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。

1. **Matrix Factorization (MF)**
   1. **A basic matrix factorization model**

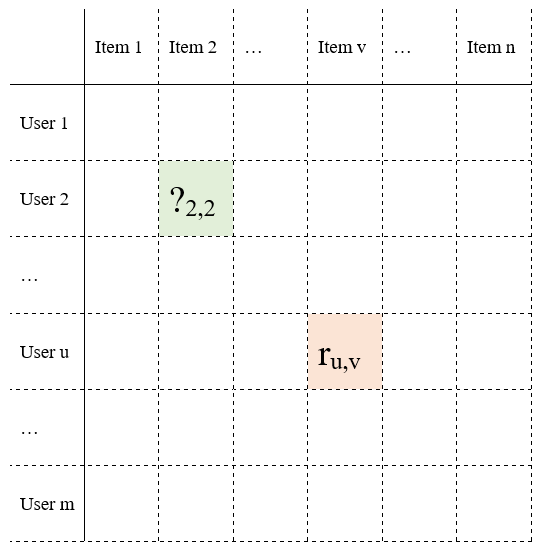


Figure 1. Rating user-item matrix R

圖一矩陣R中，m表user個數，n表item個數，ru,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。

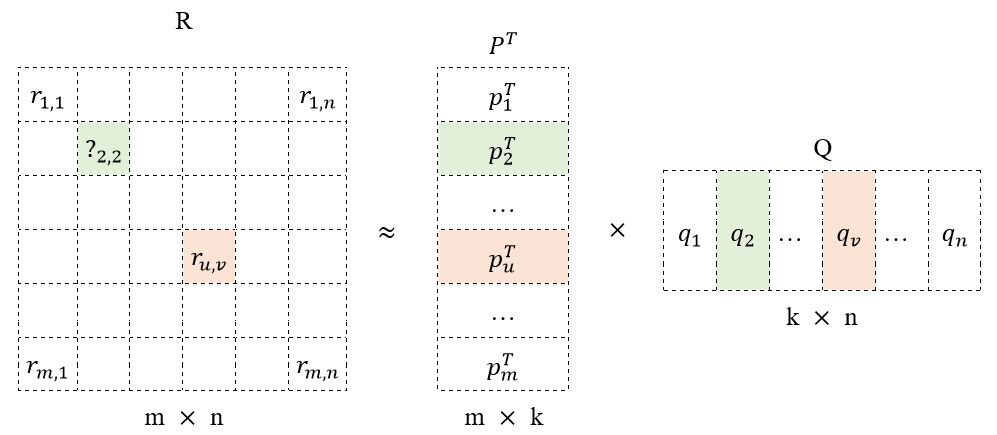


Figure 2. Matrix factorization for matrix R

將R矩陣分解後可以得到PT × Q，如圖二所示，PT 、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，為在latent factor space中用來代表user u的latent factor vector，中的k個elements可以想成是user u對k個latent factor的不同偏好；而為在latent factor space中用來代表item v的latent factor vector，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；、之dot product ()。

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

(1)

其中與為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。

* 1. **Learning algorithms**

1. **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：

其中prediction error 為：

1. **Alternating least squares (ALS)**

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

1. **Comparison between SGD and ALS**

更新矩陣P、Q 一次後ALS需花費*O*(|R|k2 + (m + n)k3)，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，因為在計算時可以不受user u以外的其他user factor的影響，同理在計算時也不受其他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)。

* 1. **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 為例：

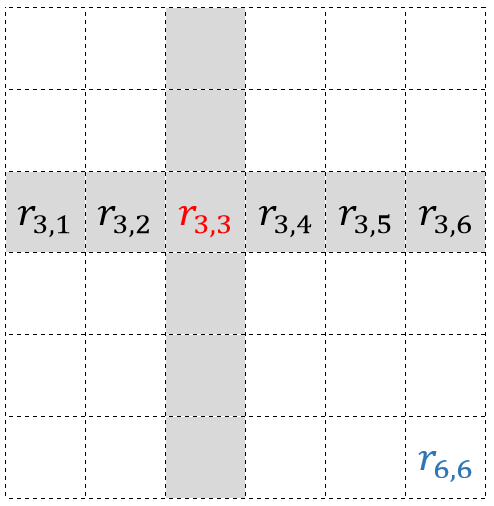


Figure 3. Partial rating matrix R

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

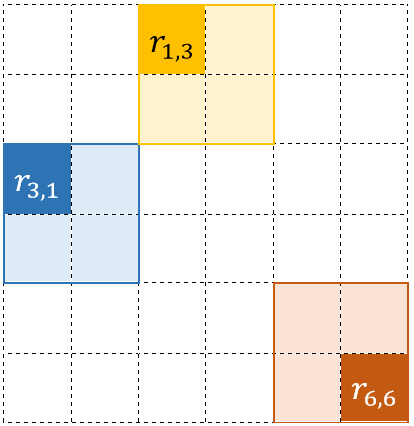


Figure 4. Splitting rating matrix R to blocks

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。

* 1. **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：

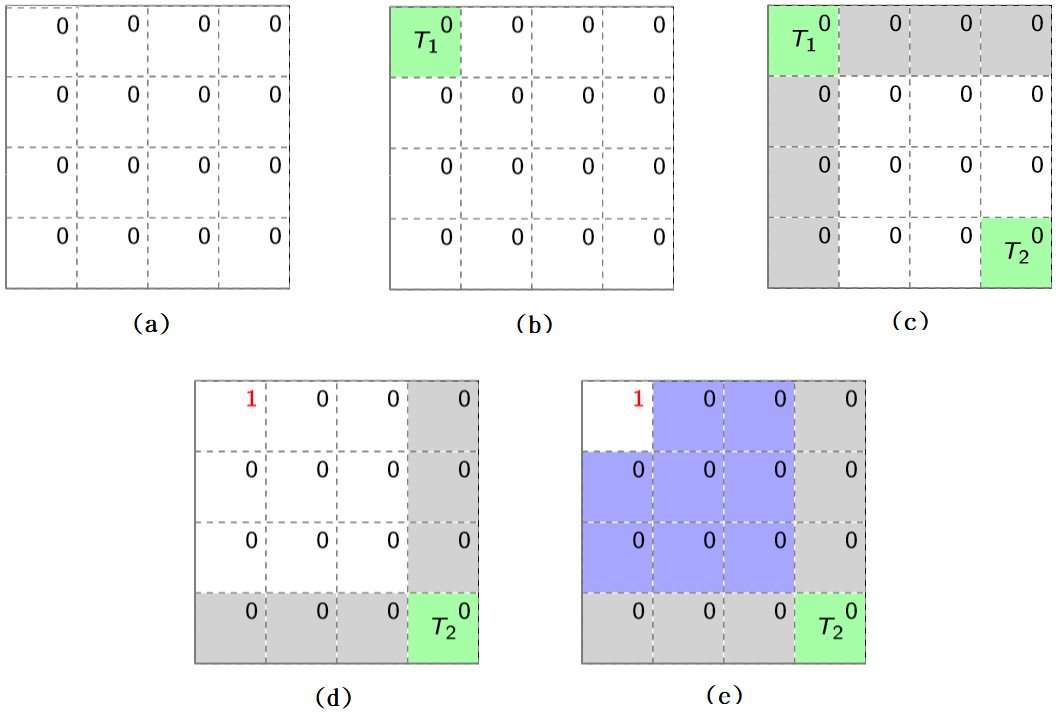


Figure 5. Process of Lock-Free scheduling

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 T1隨機選了某個block後(如圖5(b)的左上角綠色的block)，thread T2只能從thread T1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 T1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 T1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 T1與thread T2將反覆上述步驟直至收斂為止。

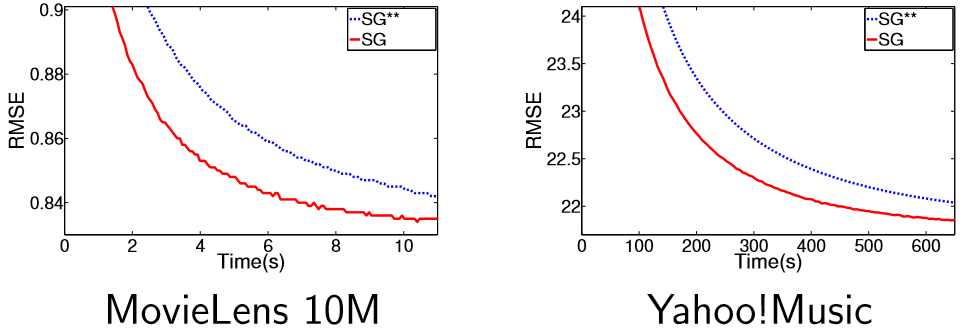


Figure 6. Lock-Free Scheduling (SG) and DSGD-like Scheduling (SG\*\*)

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。

* 1. **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’TQ’)：

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

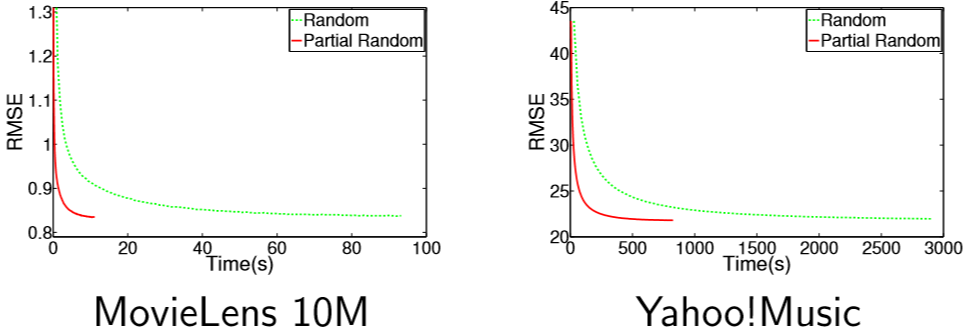


Figure 7. Partial random method and Random method (RMSE:方均跟差)

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)。

1. **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。

* 1. **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：

(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 或item v性質的latent vector 是學出來的，在MF中只有給予rating值!)：

∈ *RU*, Uuser的feature數

∈ *RV*, Vitem的feature數

則可以想成是每筆instance的feature vector為[ ]而target value為rating ，則解**regression problem**的目標函數為：

(3)

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

(4)

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

(5)

其中W ∈ RU×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 只有在第i個位置的elements值為1、其他皆為0，此時最佳解為：

此時將無法預測 (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WPTQ (P、Q為low-rank matrices)，此即變成MF：

(6)

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

* 1. **Field-aware Factorization Machines (FFM)**

在FM中，模型需maintain每對interaction (a pair of features)的weight：以interaction ⟨xj1,xj2⟩為例，每次看到不同instance x中出現interaction ⟨xj1,xj2⟩時就會分別update 模型weight wj1、wj2 (wj1、wj2皆為長度為k的vector，k為user-deﬁned parameter)。FM模型formulation可表示為：

(w,x) = (7)

其中C2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 ⟨xj1,xj2⟩時會分別update 模型weight wj1,**f2**、wj2,**f1**，其中f1與f2分別為feature j1與feature j2的field。FFM的optimization problem為(以Logistic Loss表示loss function)：

(8)

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

(w,x) = (9)

1. **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。

1. **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!