6 Appendix

6.1 Proofs of Theorems 2 and 1

6.1.1 Negative Associativity of Bloom Filters

First, let us go over the definition of negative associativity. Random variables, $\{q_i\}_{i=1}^c$, are negatively associative (NA), if for any functions f, g, both monotonically increasing or decreasing, and disjoint sets $I, J \subset \{1, \dots, c\}$,

$$\mathbb{E}[f(q_I)g(q_J)] \leq \mathbb{E}[f(q_I)] \cdot \mathbb{E}[g(q_J)],$$

where q_I, q_J are the variables restricted to these sets.

Lemma 1. (1) *Let $\{q_{0,i}\}_{i=1}^c, \{q_{1,i}\}_{i=1}^c \subset \{0,1\}^c$ be two independent random bitarrays that are both NA. Then $q_0|q_1$, the elementwise 'or' operation, and $q_0\&q_1$, the elementwise 'and' operation, are both NA. So NA is closed under elementwise 'or' and 'and' operations.*

(2) *Let q_i be the ith bit in any Bloom filter of the set $\mathcal{N}$ with independent hash functions, then the random bits, $\{q_i\}_{i=1}^c$, are NA.*

Proof. **(1)** We show that NA is closed under both elementwise operations. First, note that the concatenation $\{q_{0,1}, \dots, q_{0,c}, q_{1,1}, \dots, q_{1,c}$ is NA, by closure of NA under independent union (Property P7 in Joag-Dev et al. ((1983))). Then on the disjoint sets, $\{q_{0,i}, q_{1,i}\}_{i=1}^c$, apply the bit operation to produce the resulting array. Operation 'or' is monotonically increasing because, $q_{0,i}|q_{1,i} = 1\{q_{0,i} + q_{1,i} > 0\}$, 'and' is as well because $q_{0,i}\&q_{1,i} = 1\{q_{0,i}+q_{1,i} > 1\}$. Finally we conclude by closure of NA under monotonic increasing functions on disjoint sets (Property P6 in Joag-Dev et al. ((1983))).

(2) Consider hash function $j \in \{1, \dots, k\}$ for node $v \in \mathcal{N}$. Let B_v^j be the c-bit Bloom filter bitarray for this vertex and hash function only, then B_v^j has only a single bit that is 1 and the rest are 0. By the 0-1 property for binary bits, we know that B_v^j has NA entries (Lemma 8 in Dubhashi and Ranjan ((1998))), since $\sum_i B_{v,i}^j = 1$. Then the Bloom filter, B, of $\mathcal{N}$ is $B = |_{j=1}^k|_{v \in N(x)} B_v^j$—the 'or' operation applied to all hashes and vertices, and we conclude by property (1). □

Proof of Theorem 2. Consider the partition of $N(x) \cup N(y)$ into $A_1 = N(x) \backslash N(y)$, $A_2 = N(y) \backslash N(x)$, $A_3 = N(x) \cap N(y)$. Let B_x, B_y be the Bloom filter bitarrays for $N(x), N(y)$ and let B_1, B_2, B_3 be those for A_1, A_2, A_3 respectively.

Notice that $B_x\&B_y = B_3|(B_1\&B_2)$, where the bit operations are elementwise. If all hash functions are independent, then B_1, B_2, B_3 are independent. Notice that for a given node and hash function the bit selected is random, but unique, which means that the elements of the bitarrays are not necessarily independent for any Bloom filter. However, the bitarray $B_3|(B_1\&B_2)$ is negatively associative by Lemma 1. Let $q_i = (B_{1,i}\&B_{2,i})|B_{3,i}$, then we have that,

$$\mathbb{E}q_i = 1 - (1 - \mathbb{E}[B_{1,i}] \cdot \mathbb{E}[B_{2,i}])(1 - \mathbb{E}[B_{3,i}]).$$

The probability that bit i in one of the bitarrays is 0 is

$$1 - \mathbb{E}[B_{j,i}] = \left(1 - \frac{1}{c}\right)^{k|A_j|}, \quad j = 1, 2, 3.$$

This can give us an expression in terms of $c, k, |A_1|, |A_2|, |A_3|$ for the expectation of $Q = \sum_{i=1}^c q_i$. We have that by Hoeffding's inequality for negatively associative random variables Dubhashi and Ranjan ((1998)),

$$\mathbb{P}\{Q \geq (1+\delta)\mathbb{E}Q\} \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{\mathbb{E}Q},$$

$$\mathbb{P}\{Q \leq (1-\delta)\mathbb{E}Q\} \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mathbb{E}Q}.$$

It remains to provide intelligible bounds on $\mathbb{E}Q$. By the inequalities $1 - 1/x \leq \log x \leq x - 1$,

$$-\frac{k|A_3|}{c-1} \leq \log(1 - \mathbb{E}[B_{3,i}]) \leq -\frac{k|A_3|}{c}$$

Also,

$$\mathbb{E}[B_{1,i}] \cdot \mathbb{E}[B_{2,i}] = \left(1 - \left(1 - \frac{1}{c}\right)^{k|A_1|}\right)\left(1 - \left(1 - \frac{1}{c}\right)^{k|A_2|}\right)$$

so by the inequality,

$$\log(1 - (1-(1-x)^a)(1-(1-x)^b)) \geq -abx^2, \quad a, b > 0, x \in [0,1];$$

we have that

$$-k^2 \frac{|A_1||A_2|}{c^2} \leq \log(1 - \mathbb{E}[B_{1,i}] \cdot \mathbb{E}[B_{2,i}]) \leq 0.$$

Furthermore, notice that the LHS is minimized when $|A_1| = |A_2| = |N(x) \triangle N(y)|/2$,

$$-k^2 \frac{|A_1||A_2|}{c^2} \geq -k^2 \frac{|N(x) \triangle N(y)|^2}{4c^2}.$$

We then have that

$$\begin{aligned} &\log(c - \mathbb{E}Q) \\ &= \log c + \log(1 - \mathbb{E}[B_{1,i}] \cdot \mathbb{E}[B_{2,i}]) + \log(1 - \mathbb{E}[B_{3,i}]) \\ &\leq \log c - \frac{k|N(x) \cap N(y)|}{c} \end{aligned}$$

and

$$\log(c - \mathbb{E}Q) \geq \log c - k^2 \frac{|N(x) \triangle N(y)|^2}{4c^2} - \frac{k|N(x) \cap N(y)|}{c-1}.$$

□

Proof of Theorem 1. We can see that there exist C_0, C_1 such that for any $\delta \geq 0$,

$$\left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{\mathbb{E}Q} \leq C_1 e^{-C_0 \delta \mathbb{E}Q}.$$

Then we have that with probability $1 - \gamma$,

$$Q \leq \left(1 + \frac{1}{C_0} \log \frac{C_1}{\gamma}\right)\mathbb{E}Q \leq \left(1 + \frac{1}{C_0} \log \frac{C_1}{\gamma}\right)\Gamma_1.$$

Note that because $1 - e^{-x} \leq x$,

$$\Gamma_1 \leq k^2 \frac{|N(x) \triangle N(y)|^2}{4c} + \frac{ck|N(x) \cap N(y)|}{c-1}.$$

Moreover, for $\delta \in (0,1)$,

$$\left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\mathbb{E}Q} \leq e^{-\frac{\delta^2}{3}\mathbb{E}Q} \leq e^{-\frac{\delta^2}{3}\Gamma_0}.$$

Hence,

$$\mathbb{P}\{Q \leq (1-\delta)\Gamma_0\} \leq e^{-\frac{\delta^2}{3}\Gamma_0}.$$

Suppose that for some $\alpha \in (0,1)$, $\alpha c > k|N(x) \cap N(y)|$ then we have that

$$\Gamma_0 \geq \frac{(1-e^{-\alpha})}{\alpha} k|N(x) \cap N(y)|.$$

The function $(1-e^{-\alpha})/\alpha$ is decreasing and the limit as $\alpha \to 0$ is 1. Thus, for any $\delta \in (0,1)$, there exists an $\alpha \geq 0$ such that if $\alpha c > k|N(x) \cap N(y)|$ then $\Gamma_0 \geq (1-\delta)k|N(x) \cap N(y)|$. If this is the case then

$$\mathbb{P}\{Q \leq (1-\delta)k|N(x) \cap N(y)|\} \leq e^{-\frac{1}{3}(1-\delta)\delta^2 k|N(x)\cap N(y)|}.$$

□

Algorithm 2 A Standard Bloom Filter

```
class BloomFilter:
  def constructor(self, c, {h_t(·) : t = 1, ..., k}):
    self.b[i] = 0   ∀i = 1, ..., c
    self.h_t = h_t   ∀i = 1, ..., k

  def add(self, x):
    self.b[self.h_t(x)] = 1   ∀t = 1, ..., k

  def union(self, bf):
    self.b[i] ← self.b[i] | bf.b[i]   ∀i = 1, ..., c

  def size(self):
    return ⌈−(c/k) log(1 − nnz(self.b)/c)⌉
```

6.2 Simulation Study

In the simulation we carried out, we set the number of users $n = 10,000$ and the number of items $m = 2,000$. We uniformly sample 5% for training and 2% for testing out of the total nm ratings. We choose $T = 3$ so the graph contains at most 6-hop information among n users. We use rank $r = 50$ for both user and item embeddings. We set influence weight $w = 0.6$, i.e. in each propagation step, 60% of one user's preference is decided by its friends (i.e. neighbors in the friendship graph). We set $p = 0.001$, which is the probability for each of the possible edges being chosen in Erdős-Rényi graph G. A small edge probability p, influence weight $w < 1.0$, and a not too-large T is needed, because we don't want that all users become more or less the same after T propagation steps.

6.3 Metrics

We omit the definitions of RMSE, Precision@k, NDCG@k, MAP as those can be easily found online. HLU: Half-Life Utility Breese et al. ((1998)), Shani et al. ((2008)) is defined as:

$$\text{HLU} = \frac{1}{n}\sum_{i=1}^{n} \text{HLU}_i, \tag{6}$$

where n is the number of users and HLU_i is given by:

$$\text{HLU}_i = \sum_{l=1}^{k} \frac{max(R_{i\Pi_{il}} - d, 0)}{2^{(j-1)/(\alpha-1)}}, \tag{7}$$

where $R_{i\Pi_{il}}$ follows previous definition, d is the neural vote (usually the rating average), and α is the viewing halflife. The halflife is the number of the item on the list such that there is a 50-50 chance the user will review that item Breese et al. ((1998)).

Algorithm 3 Simulation of Synthetic Data

Input: n users, m items, rank r, influence weight w, T propagation steps
Output: $R_{\text{tr}} \in \mathbb{R}^{n\times m}$, $R_{\text{te}} \in \mathbb{R}^{n\times m}$, $G \in \mathbb{R}^{n\times n}$

1: Randomly initialize $U \in \mathbb{R}^{n\times r}, V \in \mathbb{R}^{m\times r}$ from standard normal distribution
2: Generate a random undirected Erdős-Rényi graph G with each edge being chosen with probability p
3: **for** $t = 1, ..., T$ **do**
4: **for** $i = 1, ..., n$ **do**
5: $\tilde{U}_i = w \cdot \sum_{j:(i,j)\in G} U_j + (1-w) \cdot U_i$
6: Set $U = \tilde{U}$
7: Generate rating matrix $R = UV^T$
8: Random sample observed user/item indices in training and test data: $\Omega_{\text{tr}}, \Omega_{\text{te}}$
9: Obtain $R_{\text{tr}} = \Omega_{\text{tr}} \circ R, R_{\text{te}} = \Omega_{\text{te}} \circ R$
10: **return** rating matrices $R_{\text{tr}}, R_{\text{te}}$, user graph G

6.4 Graph Regularized Weighted Matrix Factorization for Implicit feedback

We use the rank $r = 10$, negatives' weight $\rho = 0.01$ and measure the prediction performance with metrics MAP, HLU, Precision@k and NDCG@k (see definitions of metrics in Appendix 6.3).

We follow the similar procedure to what is done before in GRMF and co-factor: we run all combinations of tuning parameters of $\lambda_l \in \{0.01, 0.1, 1, 10, 100\}$ and $\lambda_g \in \{0.01, 0.1, 1, 10, 100\}$ for each method on validation data for fixed number 40 epochs and choose the best combination as the parameters to use on test data. We then report the best prediction results during first 40 epochs on test data with the chosen parameter combination.

6.5 Reproducibility

To reproduce results reported in the paper, one need to download data (douban and flixster) and third-party C++ Matrix Factorization library from the link https://www.csie.ntu.edu.tw/~cjlin/papers/ocmf-side/. One can simply follow README there to compile the codes in Matlab and run one-class matrix factorization library in different modes (both explicit feedback and implicit feedback works).

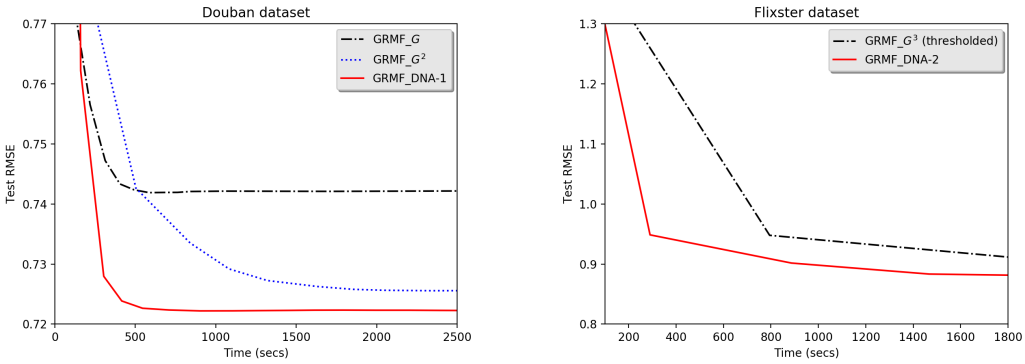

Figure 4: Compare Training Speed of GRMF, with and without Graph DNA.

Table 6: Compare Bloom filters of different depths and sizes an on Synthesis Dataset. Note that the number of bits of Bloom filter is decided by Bloom filter's maximum capacity and tolerable error rate (i.e. false positive error, we use 0.2 as default).

methods	max capacity	c bits	nnz ratio	RMSE ($\times 10^{-3}$)	% Relative Graph Gain
GRMF_G^2	-	-	-	2.6543	59.5903
GRMF_DNA-1	20	135	0.217	2.4303	163.8734
GRMF_DNA-1	50	336	0.093	2.4795	140.9683
GRMF_DNA-2	20	135	0.880	2.4921	135.1024
GRMF_DNA-2	50	336	0.608	2.4937	134.3575
GRMF_DNA-2	100	672	0.381	2.4510	154.2365
GRMF_DNA-2	200	1,341	0.215	2.4541	152.7933
GRMF_DNA-3	200	1,341	0.874	2.4667	146.9274
GRMF_DNA-3	600	4,020	0.525	2.4572	151.3500
GRMF_DNA-3	1,000	6,702	0.364	2.4392	159.7299
GRMF_DNA-3	**1,500**	**10,050**	**0.262**	**2.4247**	**166.4804**
GRMF_DNA-4	2,000	13,401	0.743	2.5532	106.6573
GRMF_DNA-4	4,000	26,799	0.499	2.4466	156.2849

Table 7: Compare nnz of different methods on Douban and Flixster datasets. GRMF_G^4 and GRMF_DNA-2 are using the same 4-hop information in the graph but in different ways. Note that we do not exclude potential overlapping among columns.

Dataset	methods	R_{tr}	G	G^2	G^3	G^4	B	total nnz
Douban	MF	9,803,098	-	-	-	-	-	9,803,098
	GRMF_G	9,803,098	1,711,780	-	-	-	-	11,514,878
	GRMF_G^2	9,803,098	1,711,780	106,767,776	-	-	-	118,282,654
	GRMF_G^3	9,803,098	1,711,780	106,767,776	2,313,572,544	-	-	2,431,855,198
	GRMF_G^4	**9,803,098**	**1,711,780**	**106,767,776**	**2,313,572,544**	**8,720,553,105**	-	**11,152,408,303**
	GRMF_DNA-1	9,803,098	0	-	-	-	8,834,740	18,637,838
	GRMF_DNA-2	9,803,098	1,711,780	-	-	-	142,897,900	154,412,778
	GRMF_DNA-3	9,803,098	1,711,780	-	-	-	928,159,604	939,674,482
Flixster	MF	3,619,304	-	-	-	-	-	3,619,304
	GRMF_G	3,619,304	2,538,746	-	-	-	-	6,158,050
	GRMF_G^2	3,619,304	2,538,746	130,303,379	-	-	-	136,461,429
	GRMF_G^3	3,619,304	2,538,746	130,303,379	2,793,542,551	-	-	3,060,307,359
	GRMF_G^4	**3,619,304**	**2,538,746**	**130,303,379**	**2,793,542,551**	**12,691,844,513**	-	**15,752,151,872**
	GRMF_DNA-1	3,619,304	0	-	-	-	12,664,952	16,284,256
	GRMF_DNA-2	3,619,304	2,538,746	-	-	-	181,892,883	188,050,933
	GRMF_DNA-3	3,619,304	2,538,746	-	-	-	1,185,535,529	1,191,693,579

The advantage of using this library is that the codes support multi-threading and runs quite fast with very efficient memory space allocations. It also supports with graph or other side information. All three methods' baseline can be simply run with the tuning parameters we reported in the Table 9, 10, 11 in Appendix.

To reproduce results of our DNA methods, one need to generate Bloom filter matrix B following Algorithm 1. We will provide our python codes implementing Algorithm 1 and Matlab codes converting into the formats the library requires.

For baselines and our DNA methods, We perform a parameter sweep for $\lambda_l \in \{0.01, 0.1, 1, 10, 100\}$ and $\lambda_g \in \{0.01, 0.1, 1, 10, 100\}$ as well as for $\alpha \in \{0.0001, 0.001, 0.01, 0.1, 0.3, 0.7, 1\}$, for $\beta \in \{0.005, 0.01, 0.03, 0.05, 0.1\}$ when needed. We run all combinations of tuning parameters for each method on validation set for 40 epochs and choose the best combination as the parameters to use on test data. We then report the best test RMSE in first 40 epochs on test data with the chosen parameter combination. We provide all the chosen combinations of tuning parameters that achieves reported optimal results in results tables in the Table 9, 10, 11 in Appendix. One just need to exactly follow our procedures in Section 3 to construct new $\dot{G}, \dot{U}$ to replace the G, U in baseline methods before feeding into Matlab.

As to simulation study, we will also provide python codes to repeat our Algorithm 3 to generate synthesis dataset. One can easily simulate the data before converting into Matlab data format and running the codes as before. The optimal parameters can be found in Table 8. For all the methods, we select the best parameters λ_l and λ_g from $\{0.01, 0.1, 1, 10, 100\}$. For method GRMF_$G^2$, we tune an additional parameter $\alpha \in \{0.0001, 0.001, 0.01, 0.1, 0.3, 0.7, 1\}$. For the thrid-order method GRMF_G^3, we tune $\beta \in \{0.005, 0.01, 0.03, 0.05, 0.1\}$ in addition to $\lambda_l, \lambda_G, \alpha$. Due to the speed constraint, we are not able to tune a broader range of choices for α and β as it is too time-consuming to do so especially for douban and flixster datasets. For example, it takes takes about 3 weeks using 16-cores CPU to tune both α, β on flixster dataset. We run each method with every possible parameter combination for fixed 80 epochs on the same training data, tune the best parameter combination based on a small predefined validation data and report the best RMSE results on test data with the best tuning parameters during the first 80 epochs. Note that only on the small synthesis dataset, we calculate full G^3 and report the results. On real datasets, there is no way to calculate full G^4 to utilize the complete 4-hop information, because one can easily spot in Table 7 the number of non-zero elements (nnz) is growing exponentially when the hop increases by 1, which makes it impossible for one to utilize complete 3-hop and 4-hop information.

In Table 9, one can compare magnitude of optimal α and β to have a good idea of whether G or G^2 is more useful. G represents shallow graph information and G^2 represents deep graph information. If one already run GRMF_G^2, one can then use this as a preliminary test to decide whether to go deep with DNA-3 ($d = 3$) to capture deep graph information or simply go ahead with DNA-1 ($d = 1$) to fully utilize shallow information. For douban dataset, we have $\alpha = 0.05 > 0.0005 = \beta$, which implies shallow information is important and we should fully utilize it. It explains why DNA-1 is performing well both in terms of performance and speed on douban dataset. It is worth noting that GRMF_DNA-1's Bloom filter matrix B contains much more nnz than that of G in Table 7 though 20% less than that of G^2. On the other hand, for flixster dataset, we have $\alpha = 0.01 < 0.1 = \beta$, which implies in this dataset deeper information is more important and we should go deeper. That explains why here GRMF_DNA-3 (6-hop) achieves about 10 times more gain than using 1-hop GRMF_G.

6.6 Code

Part of our code is already made available on Github.

Table 8: Compare Matrix Factorization for Explicit Feedback on Synthesis Dataset. The synthesis dataset has $10,000$ users and $2,000$ items with user friendship graph of size $10,000 \times 10,000$. Note that the graph only contains at most 6-hop valid information. GRMF_G^6 means GRMF with $G + \alpha \cdot G^2 + \beta \cdot G^3 + \gamma \cdot G^4 + \epsilon \cdot G^5 + \omega \cdot G^6$. GRMF_DNA-$d$ means depth d is used.

methods	test RMSE ($\times 10^{-3}$)	λ_l	λ_g	α	β	γ	ϵ	ω	% gain over baseline
MF	2.9971	0.01	-	-	-	-	-	-	-
GRMF_G	2.7823	0.01	0.01	-	-	-	-	-	7.16693
GRMF_G^2	2.6543	0.01	0.01	0.3	-	-	-	-	11.43772
GRMF_G^3	2.5687	0.01	0.01	0.01	0.05	-	-	-	14.29382
GRMF_G^4	2.5562	0.01	0.01	0.01	0.05	0.1	-	-	14.71088
GRMF_G^5	2.4853	0.01	0.01	0.01	0.05	0.1	0.1	-	17.07651
GRMF_G^6	2.4852	0.01	0.01	0.01	0.05	0.1	0.1	0.01	17.07984
GRMF_DNA-1	2.4303	0.01	0.01	-	-	-	-	-	18.91161
GRMF_DNA-2	2.4510	0.01	0.01	-	-	-	-	-	18.22095
GRMF_DNA-3	**2.4247**	**0.01**	**0.01**	-	-	-	-	-	**19.09846**
GRMF_DNA-4	2.4466	0.01	0.01	-	-	-	-	-	18.36776

Table 9: Compare Matrix Factorization methods for Explicit Feedback on Douban and Flixster data. We use rank $r = 10$.

Dataset	methods	test RMSE ($\times 10^{-1}$)	λ_l	λ_g	α	β	% gain over baseline
Douban	MF	7.3107	1	-	-	-	-
	GRMF_G	7.2398	0.1	100	-	-	0.9698
	GRMF_G^2	7.2381	0.1	100	0.001	-	0.9930
	GRMF_G^3 (full)	7.2432	0.1	100	0.05	0.0005	0.9350
	GRMF_G^3 (thresholded)	7.2382	0.1	100	0.05	0.0005	0.9917
	GRMF_DNA-1	7.2191	0.1	100	-	-	1.2689
	GRMF_DNA-2	7.2359	1	10	-	-	1.0232
	GRMF_DNA-3	**7.2095**	**0.01**	**100**	-	-	**1.3843**
Flixster	MF	8.8111	0.1	1	-	-	-
	GRMF_G	8.8049	0.01	1	-	-	0.0704
	GRMF_G^2	8.7849	0.01	1	0.05	-	0.2974
	GRMF_G^3 (full)	8.7932	0.1	1	0.01	0.1	0.2032
	GRMF_G^3 (thresholded)	8.7920	0.01	1	0.01	0.1	0.2168
	GRMF_DNA-1	8.8013	0.01	1	-	-	0.1112
	GRMF_DNA-2	8.8007	0.1	1	-	-	0.1180
	GRMF_DNA-3	**8.7453**	**0.1**	**100**	-	-	**0.7468**

Table 10: Compare Co-factor Methods for Explicit Feedback on Douban and Flixster Datasets. We use rank $r = 10$ for both methods.

Dataset	methods	test RMSE ($\times 10^{-1}$)	λ_l	% gain over baseline
Douban	co-factor_G	7.2743	1	-
	co-factor_DNA-3	**7.2674**	**1**	**0.5923**
Flixster	co-factor_G	8.7957	0.01	-
	co-factor_DNA-3	**8.7354**	**0.01**	**0.8591**

Table 11: Compare Weighted Matrix Factorization with Graph for Implicit Feedback on Douban and Flixster Datasets. We use rank $r = 10$ for both methods and all metric results are in %.

Dataset	Methods	MAP	HLU	P@1	P@5	NDCG@1	NDCG@5	λ_l	λ_g
Douban	WMF_G	8.340	13.033	14.944	10.371	14.944	12.564	0.01	10
	WMF_DNA-3	**8.400**	**13.110**	**14.991**	**10.397**	**14.991**	**12.619**	**1**	**1**
Flixster	WMF_G	10.889	14.909	12.303	7.9927	12.303	12.734	10	0.1
	WMF_DNA-3	**11.612**	**15.687**	**12.644**	**8.1583**	**12.644**	**13.399**	**1**	**1**