통계계산특론 기말 프로젝트

- Matrix Factorization 을 이용한 추천 시스템 알고리즘 구현

212STG13 박지원

212STG18 예지혜

218STG01 김지민

목차

I. 서론

1. 주제 소개

현대인들은 정보의 홍수 속에서 살고 있다. 최근 가장 인기있는 스트리밍 미디어인 넷플릭스에 대해서 '넷플릭스 증후군'이라는 신조어가 등장할 정도다. 넷플릭스 증후군이란 넷플릭스에서 제공하는 컨텐츠를 보는 시간보다 무엇을 볼지 결정하지 못해 예고편만 보다 끝나는 상황을 가리킨다. 따라서, 이용자들의 취향에 적합한 컨텐츠를 추천해주는 추천 시스템은 이들의 만족도를 높일 수 있는 강력한 경쟁도구가 될 것이다.

추천 시스템 알고리즘은 크게 content filtering 방법과 collaborative filtering 방법으로 나뉜다. Content filtering 방법은 user 또는 item 의 명시적인 profile 을 작성해야한다는 문제점이 있는데, 이에 대한 대안책이 collaborative filtering 이다. Collaborative filtering 은 domain-free 로 특정 item 의 세부적인 특징을 알 필요가 없이 알고리즘 내부적으로 이를 잡아내어 올바른 추천을 가능하게 한다.

Collaborative filtering 은 다시 Neighborhoods method 와 latent factor model 로 나뉜다. 이 중 matrix factorization 의 가장 성공적인 구현이라 불리는 것은 Latent factor method 이다.

Latent factor model 은 잠재 요인을 찾아내는 방법으로, 사용자와 영화에 숨겨져있는 특징들을 파악하여 그 특징들을 이용해 어떤 영화를 좋아할지 추측하고, 추천하는 방식이다. 이 모델의 가장 성공적인 방법은 matrix factorization 방법으로, 간단히 말하면 별점 데이터를 사용자와 영화 행렬로 분해하는 방법이다. 이 방법의 세 가지 알고리즘에 대해 이해하고 실제 데이터에 적용하여 추천 시스템 방식을 이해해보고자 한다.

2. Matrix Factorization

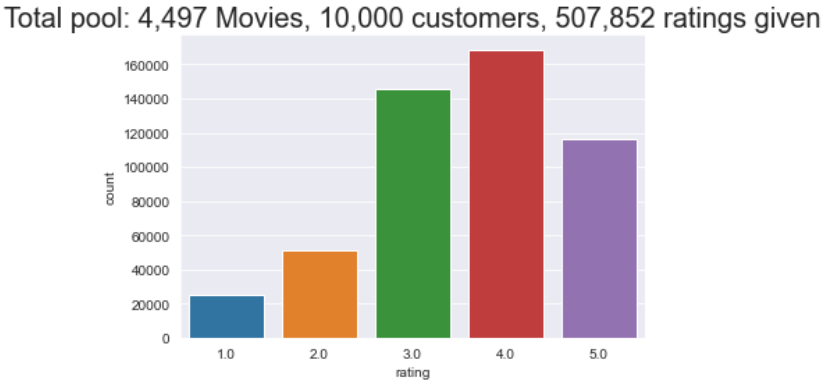
먼저, 데이터 형태는 다음과 같다. 행을 사용자, 열을 영화로 두고, 각 셀의 수치는 사용자가 해당 영화에 부여한 별점이다. 사용자가 모든 영화에 대해 별점을 평가하지는 않으므로 0 이 굉장히 많은 sparse matrix 이다.

movie_id	1	2	3	4	5	6	7	8	9	10	...	4490	4491	4492	4493	4494	4495	4496	4497	4498	4499
user_id																					
97	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
201	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
379	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
781	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
933	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2648312	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2648719	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2648927	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2649110	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2649328	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

이 행렬을 R 이라 하면 $R_{ij} = U_i^T V_j$로 분해할 수 있다. U 는 사용자, V 는 영화에 대한 행렬로, 각 요소에 대한 잠재 요인 정보를 담고 있는 행렬이다. 사용자를 N 명, 영화가 M 개, 잠재 요인을 d 개라 할 때 U 는 dxN, V 는 dxM 행렬이 된다. 이러한 U 와 V 를 찾으면 다시 곱해서 모든 셀의 값을 채울 수 있다. 이렇게 R 을 U 와 V 로 분해하는 것을 matrix factorization 이라 한다.

3. 데이터 설명

본 프로젝트에서 사용한 데이터는 Netflix 에서 제공한 데이터로 Kaggle 에서 로드하였다. 1998 년 10 월부터 2005 년 12 월까지 17000 개의 영화에 대하여 이용자들이 1-5 까지의 숫자로 별점을 매긴 약 100 만개의 자료이다. 학습 환경의 메모리 문제와 학습 속도를 위해 전체 데이터 중 10 만 명의 사용자만 잘라 4497 개 영화와 507852 개의 별점 데이터만 사용하였다. 모델의 성능을 평가하기 위해 25%의 test data 는 학습에 사용하지 않고, test rmse 를 모델 성능 척도로 사용하였다.



II. 본론 - 알고리즘 구현

1. Basic Matrix Factorization

(1) 알고리즘

분해하고자 하는 행렬 R 은 결측치가 매우 많은 Sparse matrix 로, 전통적인 singular value decomposition (SVD) 방법이 정의되지 않는다. 따라서, SVD 를 적용하기 위해 결측치를 imputation 하게 되면, 계산량이 늘어나고 데이터에 왜곡이 생길 수 있다. 최근에는 이 문제를 해결하기 위해 이미 알고 있는 별점에 대해서만 학습하는 방법을 제안한다. 또한 과적합을 막기 위해 규제항을 추가하여 학습한다. 이 내용은 2009 년의 Matrix factorization techniques for recommender systems 논문을 참고하였다.

$$minimize \sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}[(R_{ij} - U_i^T V_j)^2 + \lambda(||U_i||^2 + ||V_j||^2)]$$

위의 목적 함수를 최소화하는 U 와 V 를 찾는 것이 목표이며, 지시함수 I_{ij}는 관측된 별점이면 1, 관측되지 않았으면 0 을 의미한다. 이는 이미 알고 있는 별점에 대해서만 학습하기 위함이다. λ 는 규제 정도를 조절하며, 일반적으로 cross-validation 을 통해 결정된다. 이 프로젝트에서는 λ를 여러가지 값으로 시도하여 test rmse 를 비교해보았다.

$$e_{ij} = R_{ij} - U_i^T V_j$$
$$U_i \leftarrow U_i + \alpha(e_{ij} V_j - \lambda U_i)$$
$$V_j \leftarrow V_j + \alpha(e_{ij} U_i - \lambda V_j)$$

이 목적 함수를 최소화하는 U 와 V 를 찾기 위해서는 Stochastic gradient descent 알고리즘을 사용하였다. SGD 방법은 매 단계에서 모든 데이터를 사용하지 않고 계산이 간단하기 때문에 수렴 속도가 빠르다.

이 논문에서는 사용자간 차이와 영화간 차이를 반영할 또 하나의 term 을 제안한다. R 을 U 와 V 로 분해하는 것은 사용자와 영화의 상호작용을 분석하는 것으로 볼 수 있는데 사용자 중에도 깐깐한 사용자가 있을 수 있고, 별점을 후하게 주는 사용자가

있을 수 있다. 또한 영화 중에도 유명해서 일반적으로 더 높은 별점을 받는 영화가 있을 수 있다. 이러한 편차를 잡아내기 위해 사용자와 영화를 의미하는 추가적인 term 을 타겟 함수에 반영한다.

$$b_{ij} = \mu + b_i + b_j$$

$$\widehat{R_{ij}} = \mu + b_i + b_j + U_i^T V_j$$

$$minimize \sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}[(R_{ij} - \mu - b_i - b_j - U_i^T V_j)^2 + \lambda(||U_i||^2 + ||V_j||^2 + b_i^2 + b_j^2)]$$

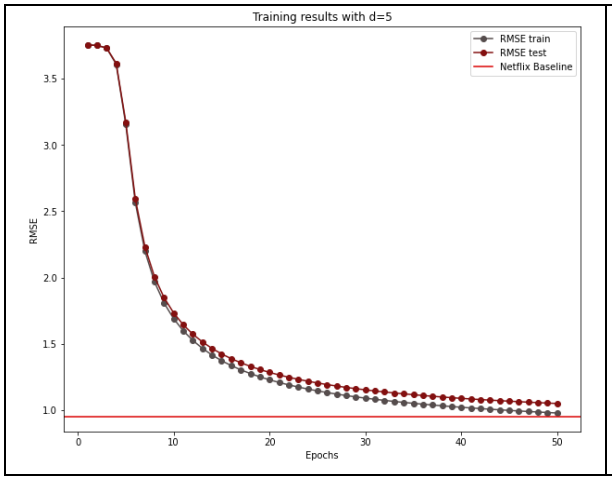
예를 들어, 어떤 사용자가 일반적으로 0.5 점을 더 박하게 주고, 타이타닉이라는 영화는 0.3 점을 후하게 받는 영화라 한다면, 추정치는 모든 영화의 평균 별점 μ에 0.5 를 더하고 0.3 을 뺀 값에 상호작용 term 을 더한 값이 될 것이다. 정리하면, 추정치는 전체 평점, 사용자의 편차, 영화의 편차, 영화와 사용자간 상호작용의 합산이 되는 것이다. 이 방법에 대해서도 Stochastic gradient descent 알고리즘을 사용하였다.

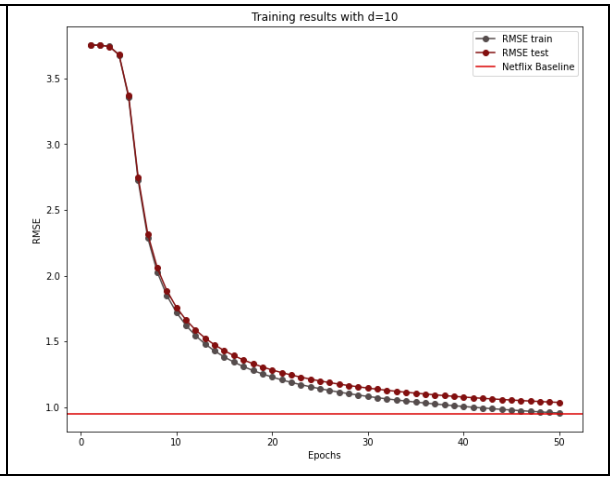
(2) 실험 결과

1) 상호작용 항만 학습

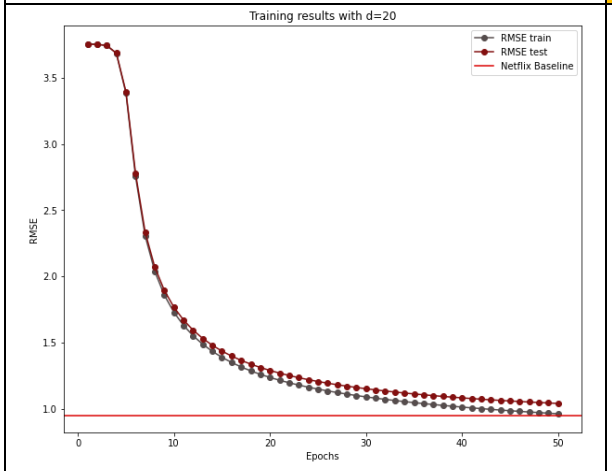
matrix factorization 구현에 있어 먼저 정해할 부분은 상호작용 효과를 몇 차원까지 추정할 것인가이다. 이 차원을 d 라 칭하면, 사용자에 대해서 d 개, 영화에 대해서 d 개의 특징을 학습한다고 볼 수 있다. 이러한 특징은 영화로 예를 들면 장르적 특징이나 캐릭터, 참신성 등도 가능하지만, 우리가 이해하기 힘든 잠재적 특징도 잡아낼 수 있다. d를 결정하는 방법은 따로 고려하지 않았으며 일반적으로 넷플릭스 데이터에 대해 20 개 정도의 특징을 고려한다고 알려져 있어 여러 개의 d 를 시도하고 성능을 비교해보았다.

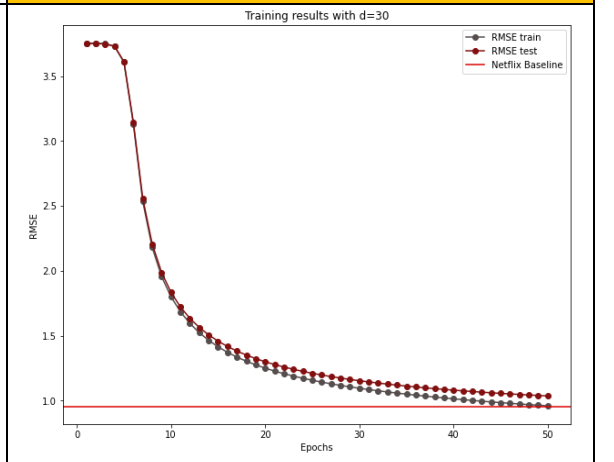
d = 5	d = 10
최종 test rmse = 1.0518	최종 test rmse = 1.0366



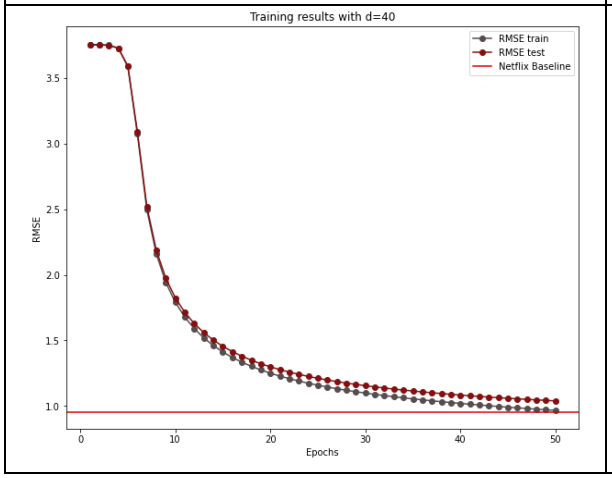


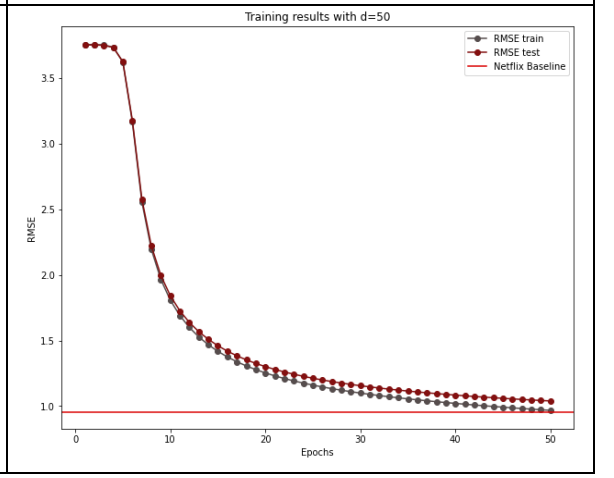
d = 20	d = 30
최종 test rmse = 1.0403	최종 test rmse = 1.0358





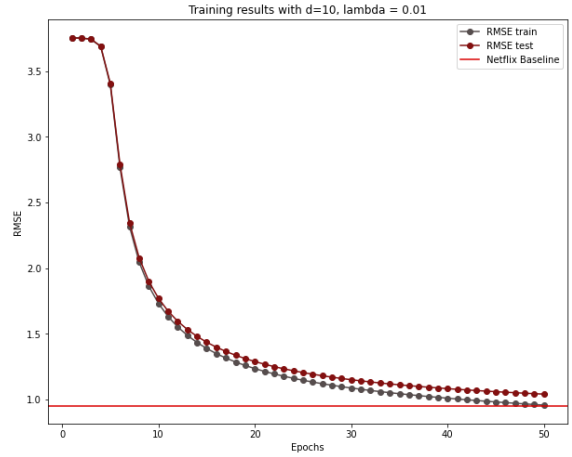
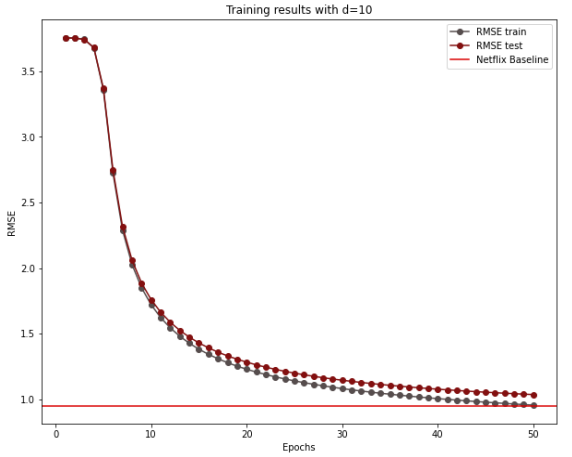
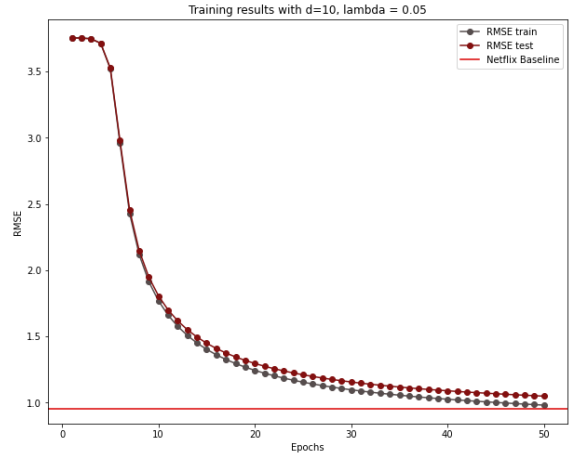
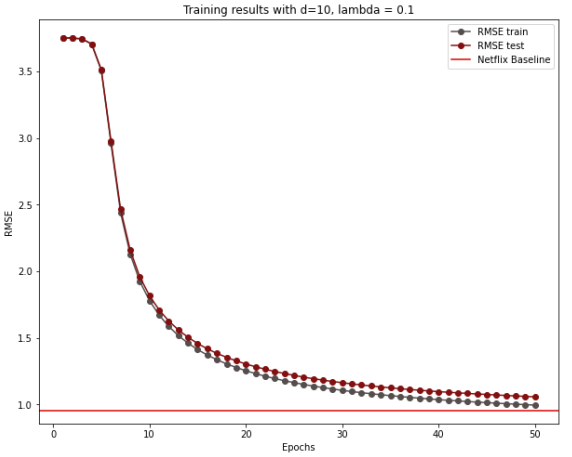
d = 40	d = 50
최종 test rmse = 1.0391	최종 test rmse = 1.0391





λ는 0.02 로 고정한 채로 d = 5, 10, 20, 30, 40, 50 을 시도하였을 때 대부분 성능이 비슷했고, 가장 성능이 좋은 것은 d = 30 이었다. 그러나 d=10 인 경우의 성능도 거의 비슷하며, 본 프로젝트는 논문에 비해 데이터를 일부만 사용했다는 점을 고려했을 때, d = 10 인 경우도 나쁘지 않다고 판단하여 d = 10 이후의 학습을 진행하였다.

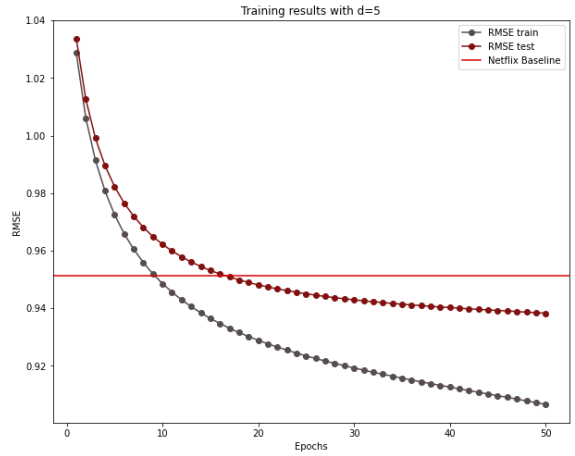
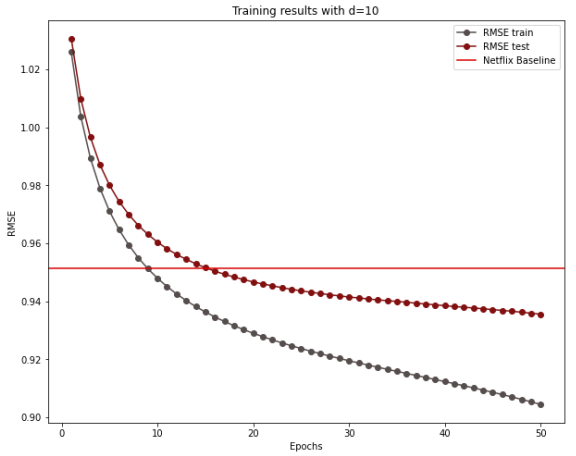
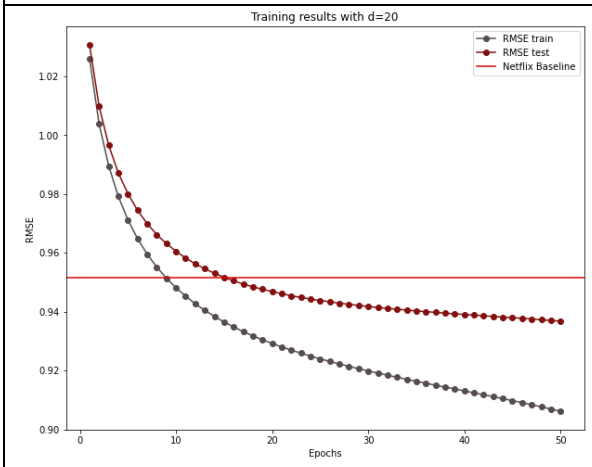
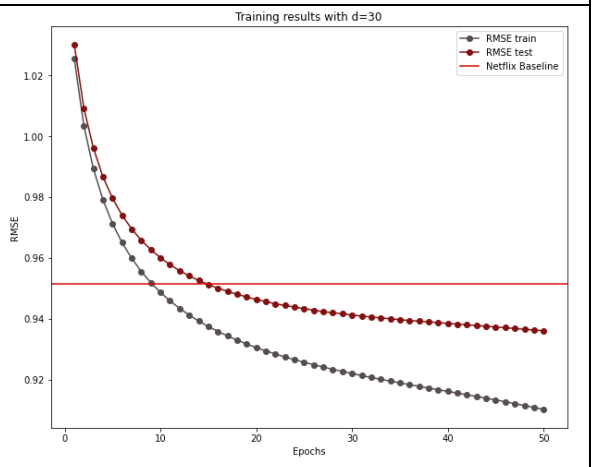
두 번째로 λ의 값에 변형을 주었다. λ를 변화시키는 것은 과적합 방지를 위한 규제를 의미한다.

λ = 0.01	λ = 0.02
최종 test rmse = 1.0408	최종 test rmse = 1.0366
	
λ = 0.05	λ = 0.1
최종 test rmse = 1.0499	최종 test rmse = 1.0579
	

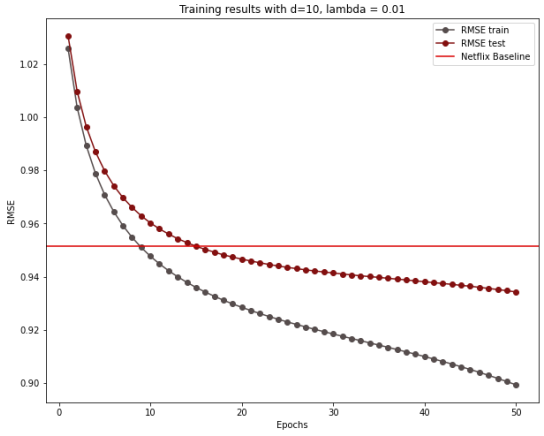
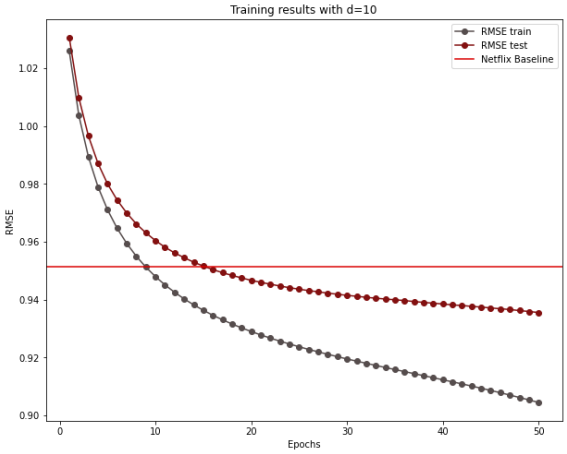
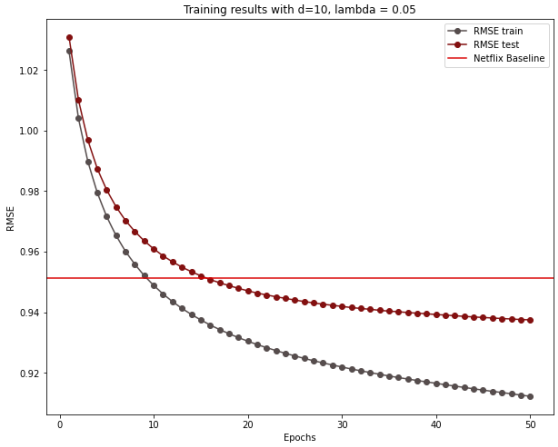
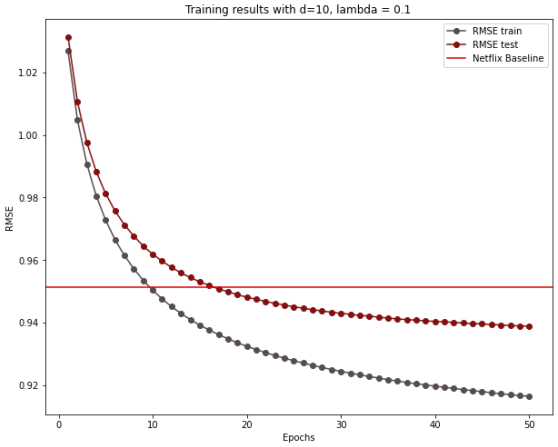
대부분의 경우 크게 과적합되지 않고 train rmse 와 test rmse 가 비슷하게 나타났다. λ = 0.02 일 때의 model 에서 가장 좋은 성능을 보였고, 이는 다른 논문에서 언급한 수치와 비슷하였으므로 이 λ값을 선택한다.

2) 개인 편차 추가 학습

사용자와 영화의 개별 편차를 추가한 모델에 대해서도 동일한 기준으로 학습을 시도하였다. 먼저 여러 개의 d 를 시도하고, λ 의 값을 변경한다. 다만, 이전의 학습 결과를 반영하여 d = 5, 10, 20, 30 의 값을 시도하고 λ = 0.01, 0.02, 0.05, 0.1 의 값을 대입하여 학습을 진행한다.

d = 5	d = 10
최종 test rmse = 0.9383	최종 test rmse = 0.9356
	
d = 20	d = 30
최종 test rmse = 0.9368	최종 test rmse = 0.960
	

test data 에 대한 rmse 가 0.93 대에 머무르며 앞선 기본 모델 결과보다 훨씬 좋은 성능을 보여주었다. 가장 성능이 좋은 경우는 d = 10 이었으며 앞의 모델과 동일하게 d=10 에 대해 다양한 λ를 학습해보았다.

λ = 0.01	λ = 0.02
최종 test rmse = 0.9343	최종 test rmse = 0.9356
	
λ = 0.05	λ = 0.1
최종 test rmse = 0.9375	최종 test rmse = 0.9389
	

λ에 따른 차이는 크지 않았으며 0.01 인 경우와 0.02 인 경우가 가장 좋았다. test rmse 는 0.01 인 경우가 더 좋았으나 이전 모델과의 원활한 비교를 위해 0.02 를 최종 모델로 비교해보겠다. 따라서 최종 모델은 d = 10, λ = 0.02 인 편차항이 추가된 모델이다. 해당 모델로 특정 사용자의 별점을 추정한 결과이다.

	movie_id	title	rating	rating_pred
18	1084	Walking with Prehistoric Beasts	5.0	3.661165
73	2862	The Silence of the Lambs	5.0	3.529433
23	1391	Yanni: Live at the Acropolis	5.0	3.434659
78	3113	Dante's Peak	5.0	3.478117
79	3184	Desert Hearts	5.0	3.413284
26	1470	Bend It Like Beckham	5.0	3.458431
27	1482	Beyond Borders	5.0	3.457147
92	3671	Laughing Matters	5.0	3.394318
57	2452	Lord of the Rings: The Fellowship of the Ring	5.0	3.936750
31	1709	Clash of the Titans	5.0	3.393939

이 사용자가 5 점을 준 영화들에 대한 별점 예측값이다. 대부분 3 점대에 머무르는데, 다른 별점대에 대해서도 3 점대로 대부분 예측하는 경향이 있었다. 다음은 이 사용자가 별점을 매기지 않은 영화에 대해 예측하여 추천한 리스트이다.

	movie_id	title	rating	rating_pred
3363	3453	The Man Who Knew Too Much	NaN	4.620457
4307	4424	Elton John: Live in Barcelona	NaN	4.511962
1906	1945	From Hell: Bonus Material	NaN	4.477076
2055	2100	Bliss	NaN	4.474524
2484	2546	The Second Coming	NaN	4.423051
1446	1474	Classic Country Comedy	NaN	4.414985
4236	4350	The Best of the New Scooby-Doo Movies	NaN	4.394918
2965	3044	The Inheritance	NaN	4.360222
2392	2450	Inserts	NaN	4.359082
3351	3441	Kicking & Screaming	NaN	4.353238

이후 이 결과를 다른 모델의 결과와 비교해보도록 한다.

2. Probabilistic Matrix Factorization

(1) 알고리즘

두 번째 알고리즘은 2008 년 Ruslan Salakgutdinov 가 Probabilistic Matrix Factorization 이라는 논문을 통해 소개한 방법으로 분포 가정을 추가한 것이다. 각 U 와 V 에 대해 평균이 0 인 가우시안 분포를 사전 분포로 가정하면 U 와 V 의 결합 분포는 베이지안 방법을 통해 사후 분포로 계산할 수 있다.

$$p(U,V|R,\sigma^2) = p(R|U,V,\sigma^2)\, p(U,V|\sigma_U^2,\sigma_V^2) = \; p(R|U,V,\sigma^2)\, p(U|\sigma_U^2)\, p(V|\sigma_V^2)$$

$$p(R|U,V,\sigma^2) = \prod_{i=1}^{N}\prod_{j=1}^{M}[\mathcal{N}\left(R_{ij}|U_i^T V_j,\sigma^2\right)]^{I_{ij}}$$

$$p(U|\sigma_U^2) = \prod_{i=1}^{N}\mathcal{N}(U_i|0,\sigma_U^2)$$

$$p(V|\sigma_v^2) = \prod_{j=1}^{M}\mathcal{N}(V_j|0,\sigma_v^2)$$

사용자와 영화는 독립적으로 발생하므로 두 분포는 독립이라 가정하면, 우리가 목표로 하는 사후 분포는 가우시안 분포의 합으로 표현할 수 있다. 베이지안 방법을 이용하므로 해당 분포의 likelihood 를 최대화하는 것이 목표이다. 따라서, 정규분포의 확률밀도 함수를 사용하고, 로그를 씌운 log likelihood 를 최대화하는 것을 목표로 하며, 목적 함수는 최소화하는 방식으로 표현하기 위해 -1 을 곱하고 상수항은 제거하면 목적 함수는 다음과 같이 표현할 수 있다.

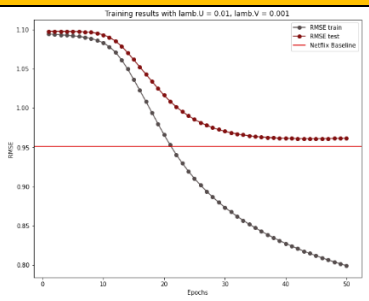
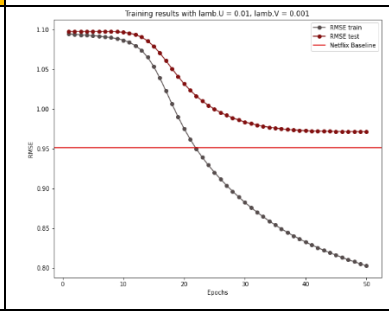
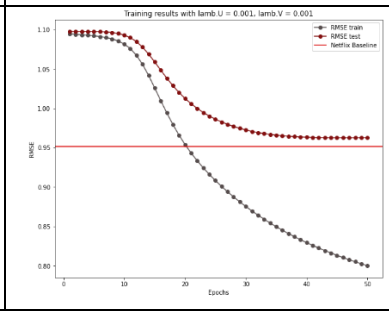
$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N}||U_i||^2 \; + \frac{\lambda_V}{2}\sum_{j=1}^{M}||V_j||^2$$

이 목적 함수를 최소화하는 U 와 V 를 찾는 것이 최종 목표이며, 이를 찾기 위한 최적화 방법으로는 논문을 참고하여 momentum gradient descent 방법을 사용하였다. 약 50만 개의 데이터 중 만 개의 데이터를 하나의 batch로 사용하였으며, 각 epoch에 대해

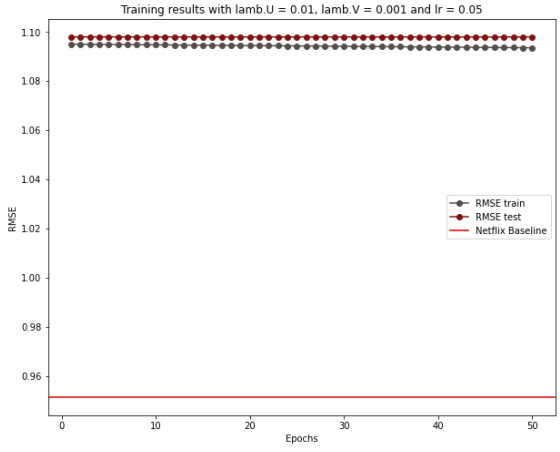
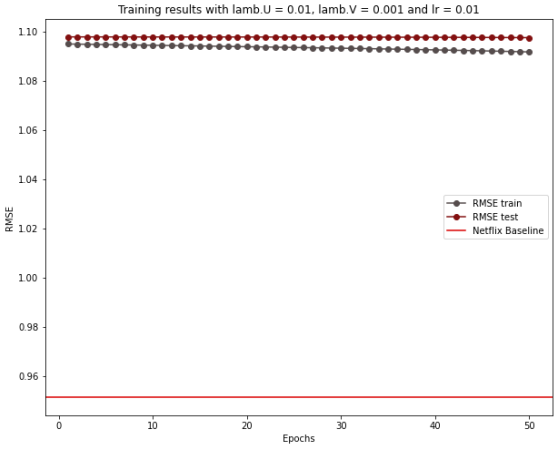
50 번의 batch 학습을 진행하였다. Learning rate 와 momentum 에 해당하는 값은 논문을 참고하였다. 그러나, 수렴 속도가 느리다는 문제점이 발생해 여러 값들을 대입하여 적절한 값을 찾았다.

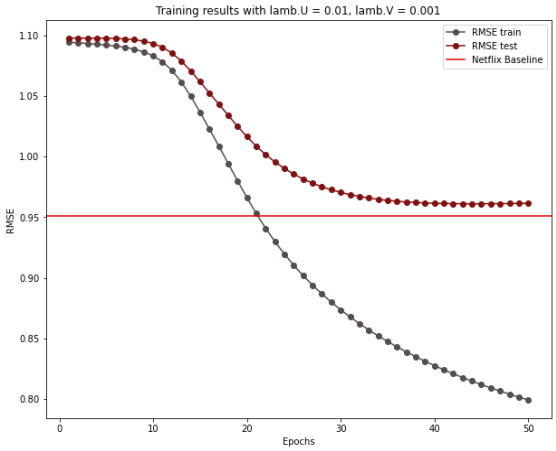
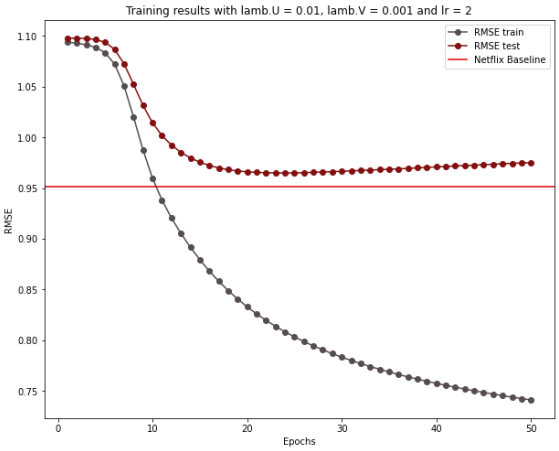
(2) 실험 결과

우선, 사전 분포의 모수에 해당하는 λ 값들을 바꾸어 가며 학습을 진행하였다. 논문에 제시된 (λ_u=0.01, λ_v=0.001), (λ_u=0.001, λ_v=0.0001)에 추가로 (λ_u=0.001, λ_v=0.001)을 시도하여 총 세가지 결과를 비교한다.

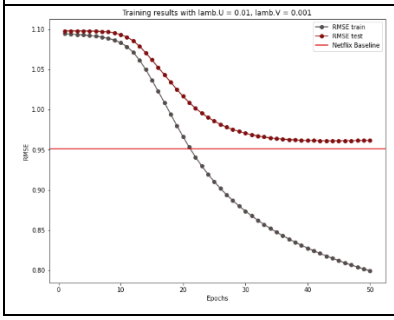
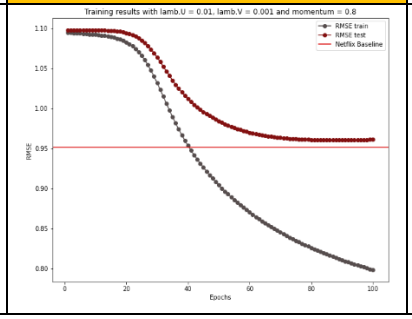
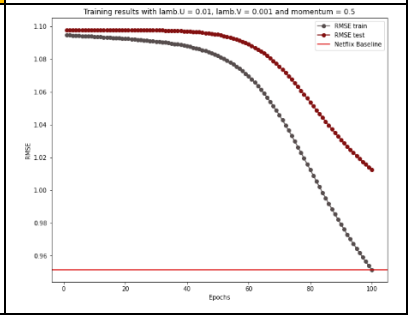
λ_u=0.01, λ_v=0.001	λ_u=0.001, λ_v=0.0001	λ_u=0.001, λ_v=0.001
최종 test rmse = 0.9610	최종 test rmse = 0.9716	최종 test rmse = 0.9626
		

비교 결과, 논문에서 첫번째로 제시한 값이 미세하게 가장 좋은 성능을 보여주었다. 따라서 해당 λ 값을 고정하고 다양한 learning rate 에 대입하였다.

Learning rate = 0.05	Learning rate = 0.1
최종 test rmse = 1.0980	최종 test rmse = 1.0976
	

Learning rate = 1	Learning rate = 2
최종 test rmse = 0.9610	최종 test rmse = 0.9650
	

이번에는 갱신 정도를 조절하는 momentum 값을 바꿔가며 시도하였다. 논문에서 제시한 값은 0.9 였기 때문에 해당 값으로 학습해왔는데, 그 외에도 추가적으로 0.8 인 경우와 0.5 인 경우에 대해 학습을 진행하였다.

Momentum = 0.9	Momentum = 0.8	Momentum = 0.5
최종 test rmse = 0.9610	최종 test rmse = 0.9605	최종 test rmse = 1.0124
		

실험 결과 momentum 이 작아질수록 느리게 수렴하는 것을 확인할 수 있었고, 가장 좋은 수치는 0.8 이었다. 이러한 결과들을 바탕으로 PMF 방법을 사용한 최종 모델은 λ (u = 0.01, v = 0.001), learning rate = 1, momentum = 0.8 일 때이며, 가장 좋은 test rmse 는 0.9605 였다. 이 모델을 가지고 특정 사용자에 대해 예측한 결과를 살펴본다.

	movie_id	title	rating	rating_pred
18	1082	Walking with Prehistoric Beasts	5.0	3.832549
73	2860	The Silence of the Lambs	5.0	4.207895
23	1389	Yanni: Live at the Acropolis	5.0	3.579948
78	3110	Dante's Peak	5.0	3.291114
79	3181	Desert Hearts	5.0	3.580797
26	1468	Bend It Like Beckham	5.0	3.705519
27	1480	Beyond Borders	5.0	3.492514
92	3668	Laughing Matters	5.0	3.733208
57	2450	Lord of the Rings: The Fellowship of the Ring	5.0	4.484694
31	1707	Clash of the Titans	5.0	3.726379

이번에는 4 점대로 예측된 영화도 몇 개 발견할 수 있었다. 이 사용자가 별점을 평가하지 않은 영화 중 높은 평점으로 예측되어 넷플릭스가 추천하게 될 영화를 확인해보자.

	movie_id	title	rating	rating_pred
2115	2160	CSI: Season 1	NaN	4.631397
4193	4303	The Sixth Sense	NaN	4.477036
2708	2780	Braveheart	NaN	4.430554
2486	2546	Gilmore Girls: Season 1	NaN	4.411368
1448	1474	Six Feet Under: Season 4	NaN	4.360109
265	269	Sex and the City: Season 4	NaN	4.336044
3027	3103	Ghost	NaN	4.332624
2068	2112	Firefly	NaN	4.318385
1765	1796	Lethal Weapon	NaN	4.307304
3000	3077	The Lion King: Special Edition	NaN	4.303823

앞서 Basic 모델이 예측한 결과와 동일한 작품은 찾아보기 힘들었다. 이후 이 결과를 다른 모델의 결과와 비교해보겠다.

3. Bayesian Probabilistic Matrix Factorization

Bayesian Probabilistic Matrix Factorization Model 은 모수 추정 자체보다는 prediction 에 관심을 둔다. 따라서 Loss function 을 최소화하는 parameter 를 찾는 것이 아니라, R_{ij} 의 예측값, 즉 사후 기대치를 구하는 것이 목적이다. $\mathrm{R_{ij}}$ 의 예측 분포를 구하기 위해서는 복잡한 적분을 풀어야 하는데, analytic solution 이 존재하지 않기 때문에 MCMC approximation 을 이용한다.

(1) 알고리즘

Bayesian PMF 모델은 다음과 같이 계층적인 prior 를 가정한다. U, V 에 대한 prior distribution 뿐 아니라 U,V 의 prior 에 대한 모수 즉 hyperparameter $\{\mu_U, \Lambda_U\}, \{\mu_V, \Lambda_V\}$ hyperprior 를 추가적으로 가정한다. U, V 에 대해서는 PMF 에서와 마찬가지로 Gaussian prior 를 가정하고, $\{\mu_U, \Lambda_U\}, \{\mu_V, \Lambda_V\}$에 대해서는 Normal-Wishart hyperprior 를 가정한다.

- **Data (Likelihood)** : $R\,|U, V, \sigma^2 \sim \prod_{i=1}^{N} \prod_{j=1}^{M} \left[Normal\left(R_{ij} \middle| U_i^T V_j,\ \sigma^2\right) \right]^{I_{ij}}$

- **Gaussian Prior** : $U_i \sim N(\mu_U,\ \Lambda_U^{-1}),\ V_j \sim N(\mu_V,\ \Lambda_V^{-1})$

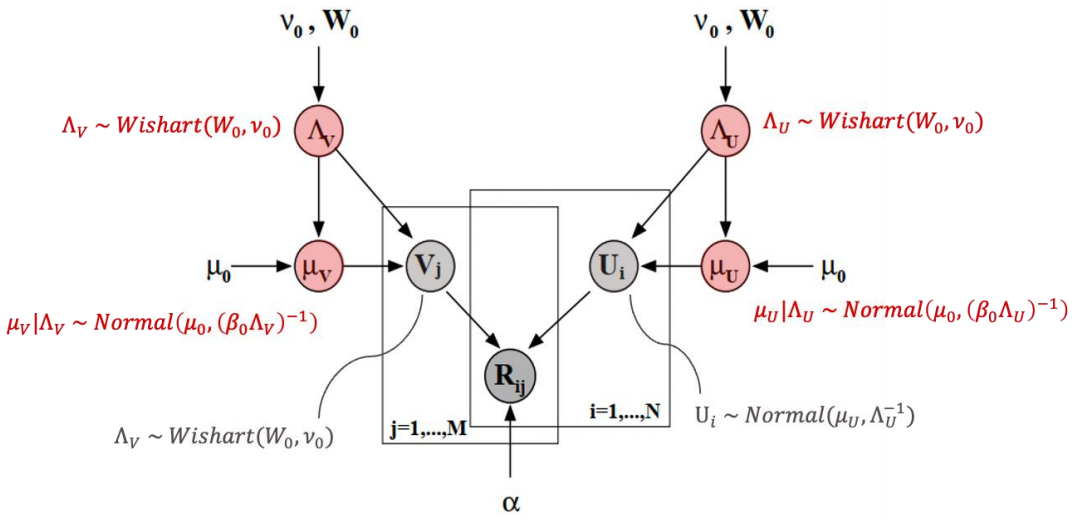
- **Gaussian-Wishart Prior**:

$\mu_U|\Lambda_U \sim Normal(\mu_0,\ (\beta_0 \Lambda_U)^{-1}),\ \mu_V|\Lambda_V \sim Normal(\mu_0,\ (\beta_0 \Lambda_V)^{-1})$

$\Lambda_U \sim Wishart(W_0, \nu_0),\ \Lambda_V \sim Wishart(W_0, \nu_0)$

$\{\mu_U,\ \Lambda_U\} \sim N(\mu_0,\ (\beta_0 \Lambda_U)^{-1})\, W(W_0, \nu_0),\ \{\mu_V,\ \Lambda_V\} \sim N(\mu_0,\ (\beta_0 \Lambda_V)^{-1})\, W(W_0, \nu_0)$

Bayesian PMF 의 모델 구조를 도식화하면 다음과 같다.



Bayesian PMF

이 때 R_{ij}의 예측 분포는

$p(R_{ij}^*|R, \Theta_0) = \iint p(R_{ij}^*|U_i, V_j)\, p(U,V|R, \Theta_U, \Theta_V)\, p(\Theta_U, \Theta_V|\Theta_0)\, d\{U,V\}\, d\{\Theta_U, \Theta_V\}$

이다. 이 적분에 대한 analytic solution이 존재하지 않기 때문에 식 (1)로 주어지는 MCMC approximation을 이용한다.

$p(R_{ij}^*|R, \Theta_0) \approx \frac{1}{K} \sum_{k=1}^{K} p\left(R_{ij}^* \middle| U_i^{(k)}, V_j^{(k)}\right), \quad \ldots (1)$

where $\left\{U_i^{(k)}, V_j^{(k)}\right\}$: MCMC samples

Markov chain 에 의해 생성되는 $\left\{U_i^{(k)}, V_j^{(k)}\right\}$는 $\{U, V, \Theta_U, \Theta_V\}$의 결합 사후 분포 즉

$$p(U, V, \Theta_U, \Theta_V|R, \Theta_0) = p(U,V|R, \Theta_U, \Theta_V) p(\Theta_U, \Theta_V|\Theta_0)$$

로부터 생성된 sample 이다.

MCMC 방법으로는 Gibbs sampling 을 이용하였다. 모수들을 조건부 사후분포로부터 차례대로 샘플링하는 사이클을 반복하는 방법인데, 가장 최근에 뽑힌 값을 조건부로 하여 샘플링 하는 것이 핵심이다.

목적 함수인 joint posterior 는 복잡한 형태이나, 각 모수의 conditional posterior 가 sampling 하기에 쉬운 형태로 주어지므로 Gibbs sampling 을 적용하기에 적절하다.

Gibbs sampling 을 적용하기 위해 각 모수들의 conditional posterior 는 다음과 같다.

■ Conditional posterior of $\{U, V\}$

$$p(U_i|R, V, \Theta_U, \alpha) \sim N(U_i \mid \mu_i^*,\ [\Lambda_i^*]^{-1})$$
$$p(V_j|R, U, \Theta_V, \alpha) \sim N\left(V_j \,\middle|\, \mu_j^*,\ [\Lambda_j^*]^{-1}\right)$$

Where

$$\Lambda_i^* = \Lambda_U + \sum_{j=1}^{M} [V_j V_j^T]^{I_{ij}}$$
$$\mu_i^* = [\Lambda_i^*]^{-1}\left(\alpha \sum_{j=1}^{M} [V_j V_j^T]^{I_{ij}} + \Lambda_U \mu_U\right)$$

■ Conditional posterior of $\{\Theta_U, \Theta_V\}$

$$\Theta_U = \{\mu_U,\ \Lambda_U\}$$
$$p(\mu_U, \Lambda_U|U,\ \Theta_0) = N(\mu_U|\mu_0^*,\ (\beta_0^*\Lambda_U)^{-1})\ W(\Lambda_U|W_0^*, \nu_0^*)$$

Where

$$\mu_0^* = \frac{\beta_0\mu_0 + N\bar{U}}{\beta_0 + N},\ \beta_0^* = \beta_0 + N,\ \nu_0^* = \nu_0 + N$$

$$[W_0^*]^{-1} = W_0^{-1} + N\bar{S} + \frac{\beta_0 N}{\beta_0 + N}(\mu_0 - \bar{U})(\mu_0 - \bar{U})^T$$

$$\bar{U} = \frac{1}{N}\sum_{i=1}^{N} U_i\,,\ \bar{S} = \frac{1}{N}\sum_{i=1}^{N}(U_i - \bar{U})(U_i - \bar{U})^T$$

$$\Theta_V = \{\mu_V,\ \Lambda_V\}$$
$$p(\mu_V, \Lambda_V|V, \Theta_0) = N(\mu_V|\mu_0^*,\ (\beta_0^*\Lambda_V)^{-1})\ W(\Lambda_V|W_0^*, \nu_0^*)$$

Where

$$\mu_0^* = \frac{\beta_0\mu_0 + M\bar{V}}{\beta_0 + M},\ \beta_0^* = \beta_0 + M,\ \ \nu_0^* = \nu_0 + M$$

$$[W_0^*]^{-1} = W_0^{-1} + M\bar{S} + \frac{\beta_0 M}{\beta_0 + M}(\mu_0 - \bar{V})(\mu_0 - \bar{V})^T$$

$$\bar{V} = \frac{1}{M}\sum_{j=1}^{M} V_i\,,\ \bar{S} = \frac{1}{M}\sum_{j=1}^{M}(V_i - \bar{V})(V_i - \bar{V})^T$$

각 모수들의 conditional posterior 를 이용하여 다음과 같이 Gibb sampling 알고리즘을 시행할 수 있다.

■ Gibbs sampling Algorithm

Gibbs sampling Algorithm

$\Theta_0 = \{\mu_0, \nu_0, W_0\}$

Step 1. Initialize parameters: $\{U^1, V^1\}$

Step 2. For t = 1,...,T
sample the hyperparameters :

$$\Theta_U^t = \{\mu_U, \Lambda_U\}^t \sim p(\Theta_U | U^t, \Theta_0)$$

$$\Theta_V^t = \{\mu_V, \Lambda_V\}^t \sim p(\Theta_V | V^t, \Theta_0)$$

For each i = 1, ... ,N user features :

$$U_i^{t+1} \sim \mathrm{p}(\mathrm{U_i} | \mathrm{R}, \mathrm{V^t}, \Theta_U^t)$$

For each j = 1, ..., M movie features:

$$V_j^{t+1} \sim \mathrm{p}(\mathrm{V_i} | \mathrm{R}, \mathrm{U^{t+1}}, \Theta_V^t)$$

(2) 실험 결과

다음 값들을 초기치로 하여 epoch 40 의 Gibbs sampling 을 시행하였다.

■ Initial values

$U_i \sim N(0,1)$

$V_j \sim N(0,1)$

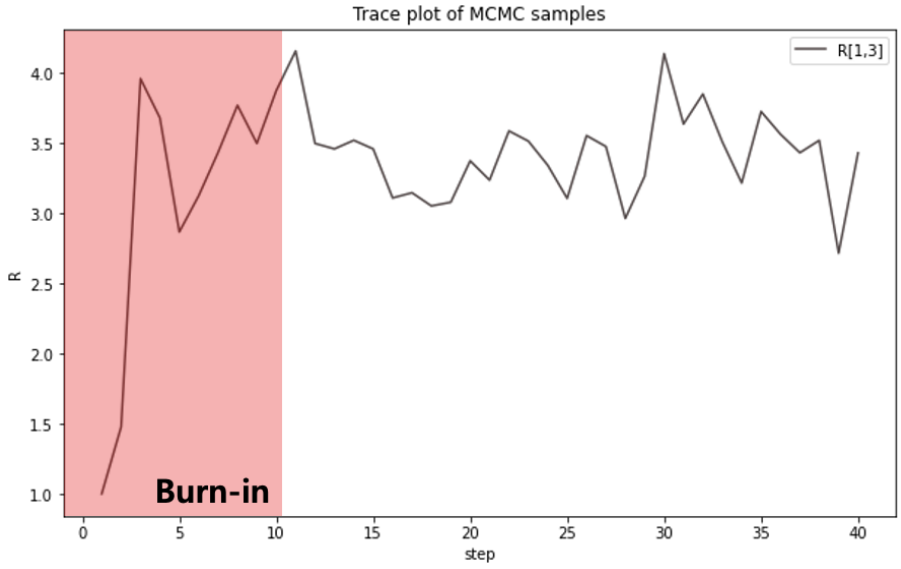
$\mu_0 = 0$

$\nu_0 = D$

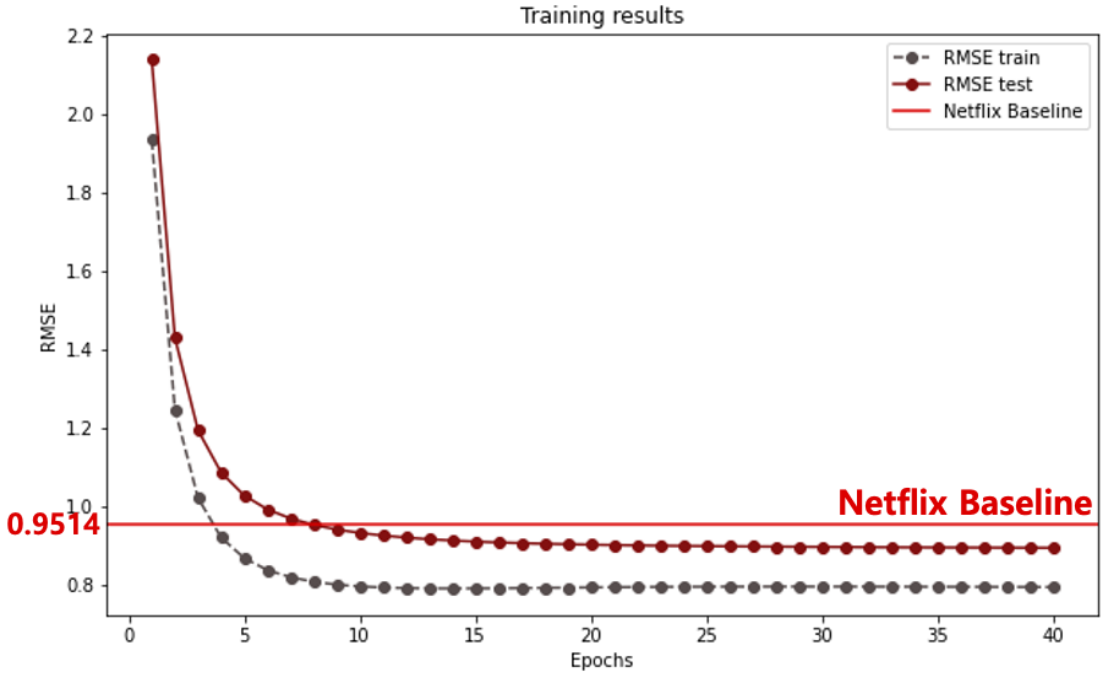
$W_0 = I\ (Identity\ Matrix)$

$\alpha = 2$

Gibbs sampling 결과 추정된 rating matrix R 의 임의로 한 값(예: user 1 이 movie 3 에 줄 것으로 예상되는 평점)에 대한 trace plot 을 그려보면 다음과 같이 3 점 초반대로 수렴하는 것으로 보인다. epoch 40 중 처음 10 step 정도는 큰 변동을 보이므로, burn-in 하여 step 10 이후의 값들만 취하여 추론을 진행할 수 있다.



D=10, epoch=40 으로 하여 BPMF 를 시행한 결과, train RMSE=0.7968, test RMSE=0.8942 로 netflix baseline RMSE 인 0.9514 보다 좋은 결과를 보여주었다. 앞서 실험한 PMF 방법에 비해서도 0.6 가량의 RMSE 성능 향상을 확인할 수 있었다.



Gibbs sampling 을 통해 얻어진 MCMC sample 의 평균을 취하여 최종 예측값을 도출 후, 특정 사용자에 대한 결과를 살펴보면 다음과 같다.

	movie_id	title	rating	rating_pred
0	1	Dinosaur Planet	3.0	3.102412
1	30	Something's Gotta Give	3.0	3.821087
2	188	Dead Birds	3.0	2.546090
3	191	X2: X-Men United	4.0	3.857685
4	283	If These Walls Could Talk	4.0	3.682491
5	457	Kill Bill: Vol. 2	3.0	3.300550
6	486	Journey to the Center of the Earth	4.0	3.470178
7	514	Santana: Supernatural Live	3.0	3.385765
8	528	The Hitchhiker's Guide to the Galaxy	4.0	2.127235
9	594	By Hook or By Crook	2.0	3.015638

이 사용자가 아직 보지 않은 영화 중 높은 평점을 예측되어 넷플릭스가 추천하게 될 영화는 다음과 같다.

	movie_id	title	rating	rating_pred
2858	2937	The Gate	NaN	4.728234
3364	3454	Kenny Chesney: Greatest Hits	NaN	4.687521
2998	3077	Turbulence	NaN	4.661542
2862	2941	Love Me Tonight	NaN	4.655401
1232	1255	Better Off Dead	NaN	4.641759
1465	1494	Appalachian Journey: Yo-Yo Ma/Edgar Meyer/Mark...	NaN	4.629214
2393	2451	Stealing Candy	NaN	4.601193
265	270	Sex and the City: Season 4	NaN	4.598906
3857	3960	Pollyanna	NaN	4.591619
4308	4425	Miranda (Tinto Brass)	NaN	4.586969

결론

1. 구현 결과 비교

1) 성능 비교

Matrix Factorization 방법을 기반으로 한 추천 시스템 알고리즘 세가지를 구현해보았다. 다음은 각 알고리즘의 최종 test RMSE 와 학습시간을 비교한 결과이다. BPMF 가 가장 성능이 뛰어나며 Basic MF 도 편차항을 추가하며 성능이 많이 개선되었다.

	test RMSE	학습 시간
Basic MF	0.9356	25m 51s (epoch 50)
PMF	0.9605	15m 5s (epoch 100)
BPMF	0.8942	55m 35s (epoch 40)

2) 추천 결과 비교

Basic MF

	movie_id	title	rating	rating_pred
3363	3453	The Man Who Knew Too Much	NaN	4.620457
4307	4424	Elton John: Live in Barcelona	NaN	4.511962
1906	1945	From Hell: Bonus Material	NaN	4.477076
2055	2100	Bliss	NaN	4.474524
2484	2546	The Second Coming	NaN	4.423051
1446	1474	Classic Country Comedy	NaN	4.414985
4236	4350	The Best of the New Scooby-Doo Movies	NaN	4.394918
2965	3044	The Inheritance	NaN	4.360222
2392	2450	Inserts	NaN	4.359082
3351	3441	Kicking & Screaming	NaN	4.353238

PMF

	movie_id	title	rating	rating_pred
2115	2160	CSI: Season 1	NaN	4.631397
4193	4303	The Sixth Sense	NaN	4.477036
2708	2780	Braveheart	NaN	4.430554
2486	2546	Gilmore Girls: Season 1	NaN	4.411368
1448	1474	Six Feet Under: Season 4	NaN	4.360109
265	269	Sex and the City: Season 4	NaN	4.336044
3027	3103	Ghost	NaN	4.332624
2068	2112	Firefly	NaN	4.318385
1765	1796	Lethal Weapon	NaN	4.307304
3000	3077	The Lion King: Special Edition	NaN	4.303823

BPMF

	movie_id	title	rating	rating_pred
2858	2937	The Gate	NaN	4.728234
3364	3454	Kenny Chesney: Greatest Hits	NaN	4.687521
2998	3077	Turbulence	NaN	4.661542
2862	2941	Love Me Tonight	NaN	4.655401
1232	1255	Better Off Dead	NaN	4.641759
1465	1494	Appalachian Journey: Yo-Yo Ma/Edgar Meyer/Mark...	NaN	4.629214
2393	2451	Stealing Candy	NaN	4.601193
265	270	Sex and the City: Season 4	NaN	4.598906
3857	3960	Pollyanna	NaN	4.591619
4308	4425	Miranda (Tinto Brass)	NaN	4.586969

이 유저에 대해 추천할 상위 10 개의 영화 중에 PMF 와 BPMF 는 하나의 작품이 겹치는 것을 확인할 수 있다. 실제로 넷플릭스에선 굉장히 많은 영화를 추천하기 때문에 더 많은 항목을 비교하면 유사한 영화가 있을 것으로 보인다. 또한 각 방법론의 결과를 앙상블하여 상위 영화들을 추천하는 것도 하나의 아이디어로 생각해볼 수 있다.

2. 한계점 및 제언

1) 데이터 크기와 컴퓨터 자원

원래 Netflix 데이터는 48 만 명의 사용자와 17,700 여개의 영화로 매우 큰 데이터셋이나, 컴퓨터 자원의 한계로 인해 100,000 명의 사용자만 추출하여 모델을 구현하였다. 시간과 컴퓨터 자원이 뒷받침된다면 실제 대규모 데이터셋을 사용해 모델을 학습시키고 보다 향상된 결과를 기대해볼 수 있을 것이다.

2) 제언

Basic MF 방법의 경우 규제항을 통해 과적합을 방지하는데, 규제항이 너무 커지면 전체적인 성능이 나빠질 수밖에 없다. 따라서 λ를 잘 조절하여 이를 결정해야 하는데 이 과정 자체가 여러 번의 실험을 필요로 한다.

PMF 방법의 경우 분포 가정을 추가한 모델이므로 이 가정이 틀리면 알고리즘 전체를 믿을 수 없게 된다. 사전 분포로 가우시안 분포를 가정하고, 모수 자체도 평균 0 과 특정 표준편차를 가정하기 때문에 가정이 잘못되었을 위험이 있다. 실제로 가우시안 분포가 잘 맞지 않는 결과를 보여 이를 반영하고자 BPMF 가 등장하였다. 모수는 동일 논문에서 모수 또한 사전 분포를 가정하여 자동적으로 선택하는 모델을 제안하고 있다.

BPMF 방법의 가장 큰 한계점은 Markov chain 이 언제 수렴했는지 판단하는 것에 명확한 기준이 없다는 점이다. Trace plot 을 통해 시각적으로 판단하는 등 명확한 기준 없이 수렴했다고 진단하면 잘못된 사후 분포로 수렴할 위험이 존재한다. 또한, MCMC 방법을 사용하기 때문에 다른 알고리즘에 비해 학습 시간이 오래 걸린다는 단점 또한 존재한다.

참고 문헌

[1] Y. Koren, R. Bell, and C. Volinsky, Matrix factorization techniques for recommender systems, IEEE Computing, 2009.

[2] Salakhutdinov, R., & Mnih, A., Probabilistic matrix factorization. *Advances in Neural Information Processing Systems 20,* 2008

[3] Salakhutdinov, R., & Mnih, A. Bayesian probabilistic matrix factorization using MCMC. *ICML'08*.

[4] Python 을 이용한 개인화 추천시스템, 임일 저

[5] https://github.com/fuhailin/Probabilistic-Matrix-Factorization

[6] https://github.com/LoryPack/BPMF

[7] https://towardsdatascience.com/pmf-for-recommender-systems-cbaf20f102f0

코드

Basic Matrix Factorization using sgd optimization

In [3]:

```
import numpy as np
import pandas as pd
```

In [4]:

```
ratings = pd.read_csv('netflix_data.csv')
ratings = ratings[['user_id', 'movie_id', 'rating']].astype(int)            # timestamp 제거
```

In [5]:

```
ratings
```

Out[5]:

	user_id	movie_id	rating
0	2442	1	3
1	1719610	1	2
2	1011918	1	4
3	479924	1	5
4	2389367	1	1
...	...	...	...
507847	295393	4499	5
507848	305344	4499	1
507849	1627987	4499	1
507850	1988633	4499	4
507851	1796454	4499	1

507852 rows × 3 columns

In [6]:

```
# train test 분리
from sklearn.utils import shuffle
TRAIN_SIZE = 0.75
ratings = shuffle(ratings, random_state=1)
cutoff = int(TRAIN_SIZE * len(ratings))
ratings_train = ratings.iloc[:cutoff]
ratings_test = ratings.iloc[cutoff:]
```

Algorithm

In [7]:

```
# New MF class for training & testing
class NEW_MF():
    def __init__(self, ratings, K, alpha, beta, iterations, verbose=True):
        self.R = np.array(ratings)
##### >>>>> (2) user_id, item_id를 R의 index와 매핑하기 위한 dictionary 생성
        item_id_index = []
        index_item_id = []
        for i, one_id in enumerate(ratings):
            item_id_index.append([one_id, i])
            index_item_id.append([i, one_id])
        self.item_id_index = dict(item_id_index)
        self.index_item_id = dict(index_item_id)
        user_id_index = []
        index_user_id = []
        for i, one_id in enumerate(ratings.T):
            user_id_index.append([one_id, i])
            index_user_id.append([i, one_id])
        self.user_id_index = dict(user_id_index)
        self.index_user_id = dict(index_user_id)
#### <<<<< (2)
        self.num_users, self.num_items = np.shape(self.R)
        self.K = K
        self.alpha = alpha
        self.beta = beta
        self.iterations = iterations
        self.verbose = verbose

    # train set의 RMSE 계산
    def rmse(self):
        xs, ys = self.R.nonzero()
        self.predictions = []
        self.errors = []
        for x, y in zip(xs, ys):
            prediction = self.get_prediction(x, y)
            self.predictions.append(prediction)
            self.errors.append(self.R[x, y] - prediction)
        self.predictions = np.array(self.predictions)
        self.errors = np.array(self.errors)
        return np.sqrt(np.mean(self.errors**2))

    # Ratings for user i and item j
    def get_prediction(self, i, j):
        prediction = self.P[i, :].dot(self.Q[j, :].T)
        return prediction

    # Stochastic gradient descent to get optimized P and Q matrix
    def sgd(self):
        for i, j, r in self.samples:
            prediction = self.get_prediction(i, j)
            e = (r - prediction)

            self.P[i, :] += self.alpha * (e * self.Q[j, :] - self.beta * self.P[i,:])
            self.Q[j, :] += self.alpha * (e * self.P[i, :] - self.beta * self.Q[j,:])

##### >>>>> (3)
    # Test set을 선정
    def set_test(self, ratings_test):
        test_set = []
        for i in range(len(ratings_test)):      # test 데이터에 있는 각 데이터에 대해서
```

```
            x = self.user_id_index[ratings_test.iloc[i, 0]]
            y = self.item_id_index[ratings_test.iloc[i, 1]]
            z = ratings_test.iloc[i, 2]
            test_set.append([x, y, z])
            self.R[x, y] = 0                    # Setting test set ratings to 0
        self.test_set = test_set
        return test_set                          # Return test set

    # Test set의 RMSE 계산
    def test_rmse(self):
        error = 0
        for one_set in self.test_set:
            predicted = self.get_prediction(one_set[0], one_set[1])
            error += pow(one_set[2] - predicted, 2)
        return np.sqrt(error/len(self.test_set))

    # Training 하면서 test set의 정확도를 계산
    def test(self):
        # Initializing user-feature and item-feature matrix
        self.P = np.random.normal(scale=1./self.K, size=(self.num_users, self.K))
        self.Q = np.random.normal(scale=1./self.K, size=(self.num_items, self.K))


        # List of training samples
        rows, columns = self.R.nonzero()
        self.samples = [(i, j, self.R[i,j]) for i, j in zip(rows, columns)]

        # Stochastic gradient descent for given number of iterations
        training_process = []
        for i in range(self.iterations):
            np.random.shuffle(self.samples)
            self.sgd()
            rmse1 = self.rmse()
            rmse2 = self.test_rmse()
            training_process.append((i+1, rmse1, rmse2))
            if self.verbose:
                if (i+1) % 5 == 0:
                    print("Iteration: %d ; Train RMSE = %.4f ; Test RMSE = %.4f" % (i+1, rmse1, rmse
        return training_process

    # Ratings for given user_id and item_id
    def get_one_prediction(self, user_id, item_id):
        return self.get_prediction(self.user_id_index[user_id], self.item_id_index[item_id])

    # Full user-movie rating matrix
    def full_prediction(self):
        return self.P.dot(self.Q.T)
```

In [26]:

```
R_temp
```

Out[26]:

movie_id	1	2	3	4	5	6	7	8	9	10	...	4490	4491	4492	4493	4494	449
user_id																	
97	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.
201	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.
379	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.
781	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.
933	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2648312	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.
2648719	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.
2648927	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.
2649110	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.
2649328	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.0	0.0	0.0	0.0	0.0	0.

10000 rows × 4497 columns

d=5, lambda = 0.02

In [8]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=5, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)
result5 = mf.test()
```

```
Iteration: 5 ; Train RMSE = 3.1543 ; Test RMSE = 3.1730
Iteration: 10 ; Train RMSE = 1.6903 ; Test RMSE = 1.7335
Iteration: 15 ; Train RMSE = 1.3742 ; Test RMSE = 1.4252
Iteration: 20 ; Train RMSE = 1.2301 ; Test RMSE = 1.2859
Iteration: 25 ; Train RMSE = 1.1468 ; Test RMSE = 1.2063
Iteration: 30 ; Train RMSE = 1.0921 ; Test RMSE = 1.1546
Iteration: 35 ; Train RMSE = 1.0530 ; Test RMSE = 1.1180
Iteration: 40 ; Train RMSE = 1.0233 ; Test RMSE = 1.0906
Iteration: 45 ; Train RMSE = 0.9997 ; Test RMSE = 1.0692
Iteration: 50 ; Train RMSE = 0.9801 ; Test RMSE = 1.0518
```

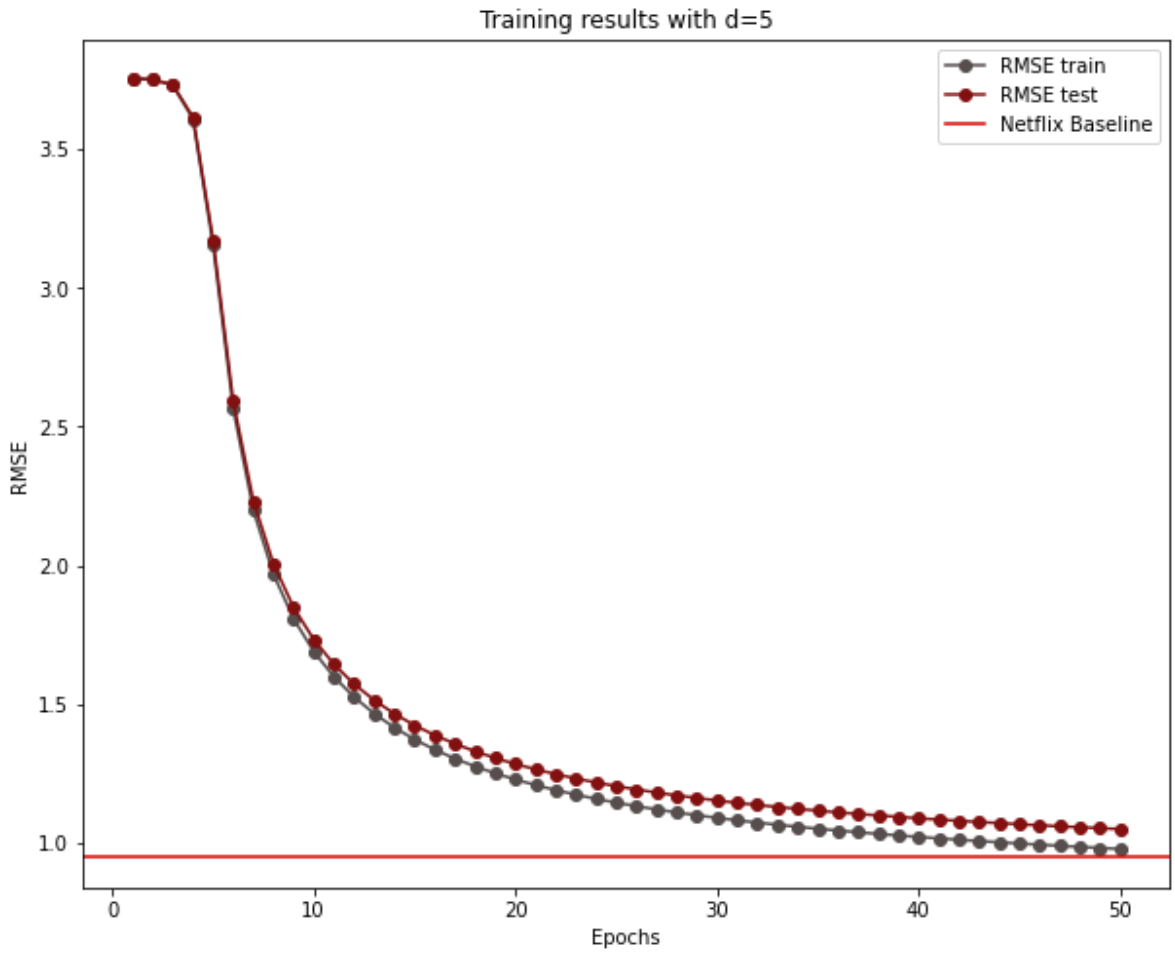
In [9]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result5)[:,0]
train_rmse_list = np.array(result5)[:,1]
test_rmse_list = np.array(result5)[:,2]

plt.title('Training results with d=5')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=10, lambda = 0.02

In [10]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)

import time
start = time.time()
result10 = mf.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 3.3568 ; Test RMSE = 3.3671
Iteration: 10 ; Train RMSE = 1.7205 ; Test RMSE = 1.7592
Iteration: 15 ; Train RMSE = 1.3821 ; Test RMSE = 1.4299
Iteration: 20 ; Train RMSE = 1.2294 ; Test RMSE = 1.2835
Iteration: 25 ; Train RMSE = 1.1410 ; Test RMSE = 1.1999
Iteration: 30 ; Train RMSE = 1.0822 ; Test RMSE = 1.1457
Iteration: 35 ; Train RMSE = 1.0393 ; Test RMSE = 1.1070
Iteration: 40 ; Train RMSE = 1.0058 ; Test RMSE = 1.0777
Iteration: 45 ; Train RMSE = 0.9787 ; Test RMSE = 1.0547
Iteration: 50 ; Train RMSE = 0.9563 ; Test RMSE = 1.0366
```

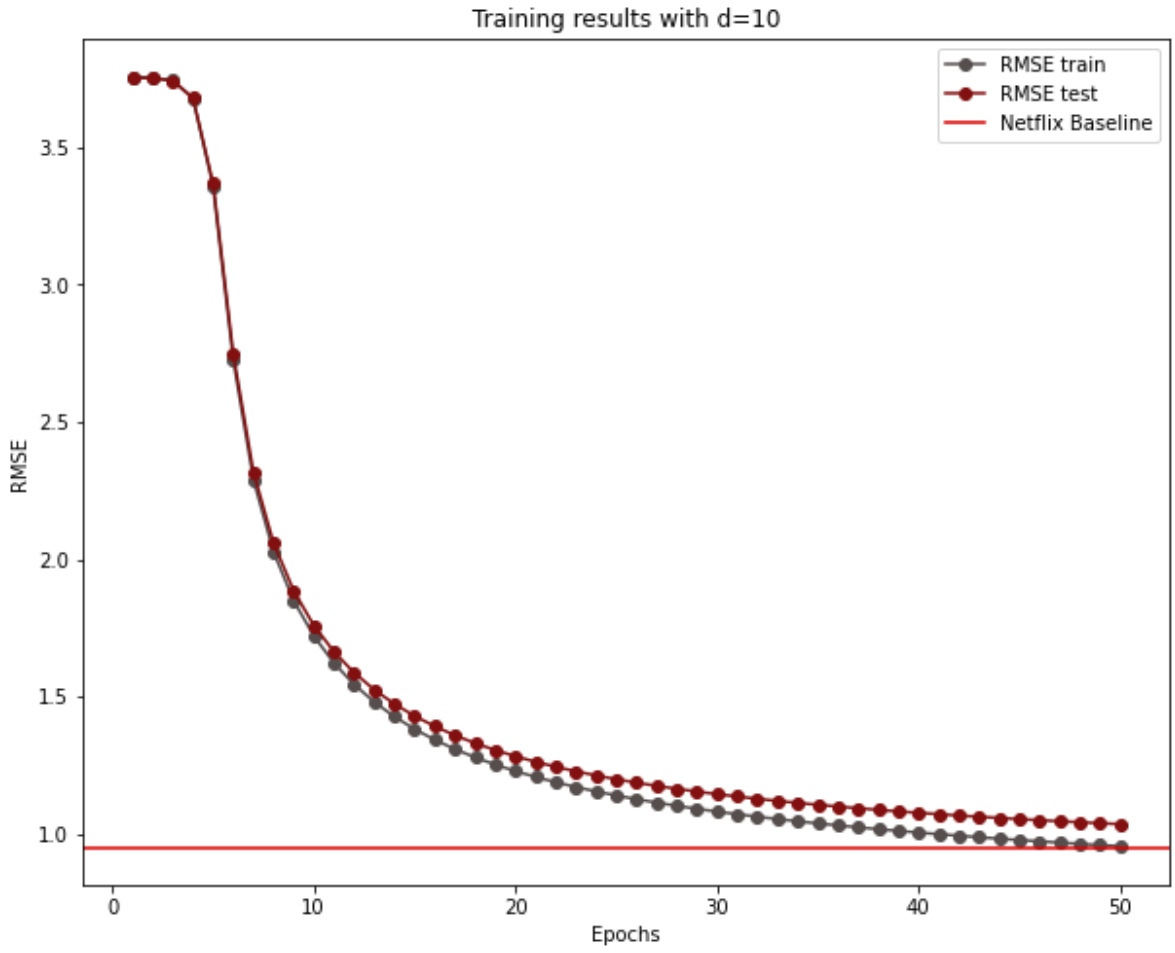
In [11]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result10)[:,0]
train_rmse_list = np.array(result10)[:,1]
test_rmse_list = np.array(result10)[:,2]

plt.title('Training results with d=10')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=20, lambda = 0.02

In [12]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)

import time
start = time.time()
result20 = mf.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 3.3830 ; Test RMSE = 3.3933
Iteration: 10 ; Train RMSE = 1.7279 ; Test RMSE = 1.7663
Iteration: 15 ; Train RMSE = 1.3884 ; Test RMSE = 1.4355
Iteration: 20 ; Train RMSE = 1.2364 ; Test RMSE = 1.2892
Iteration: 25 ; Train RMSE = 1.1485 ; Test RMSE = 1.2059
Iteration: 30 ; Train RMSE = 1.0897 ; Test RMSE = 1.1513
Iteration: 35 ; Train RMSE = 1.0465 ; Test RMSE = 1.1121
Iteration: 40 ; Train RMSE = 1.0127 ; Test RMSE = 1.0822
Iteration: 45 ; Train RMSE = 0.9852 ; Test RMSE = 1.0589
Iteration: 50 ; Train RMSE = 0.9623 ; Test RMSE = 1.0403
```

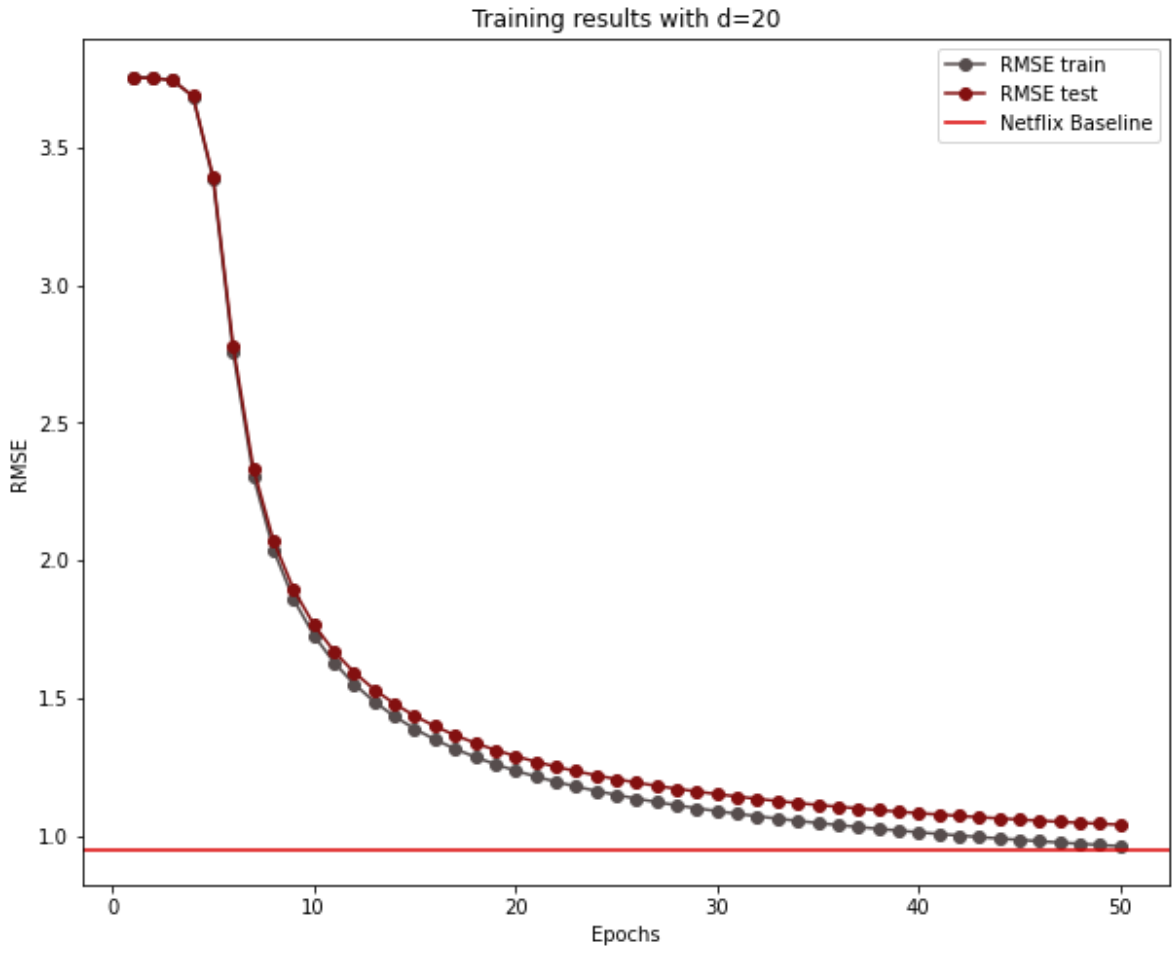
In [13]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result20)[:,0]
train_rmse_list = np.array(result20)[:,1]
test_rmse_list = np.array(result20)[:,2]

plt.title('Training results with d=20')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=30, lambda = 0.02

In [14]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=30, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)

import time
start = time.time()
result30 = mf.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 3.6100 ; Test RMSE = 3.6119
Iteration: 10 ; Train RMSE = 1.7997 ; Test RMSE = 1.8342
Iteration: 15 ; Train RMSE = 1.4143 ; Test RMSE = 1.4580
Iteration: 20 ; Train RMSE = 1.2499 ; Test RMSE = 1.2992
Iteration: 25 ; Train RMSE = 1.1566 ; Test RMSE = 1.2104
Iteration: 30 ; Train RMSE = 1.0949 ; Test RMSE = 1.1528
Iteration: 35 ; Train RMSE = 1.0494 ; Test RMSE = 1.1117
Iteration: 40 ; Train RMSE = 1.0134 ; Test RMSE = 1.0802
Iteration: 45 ; Train RMSE = 0.9840 ; Test RMSE = 1.0554
Iteration: 50 ; Train RMSE = 0.9595 ; Test RMSE = 1.0358
```

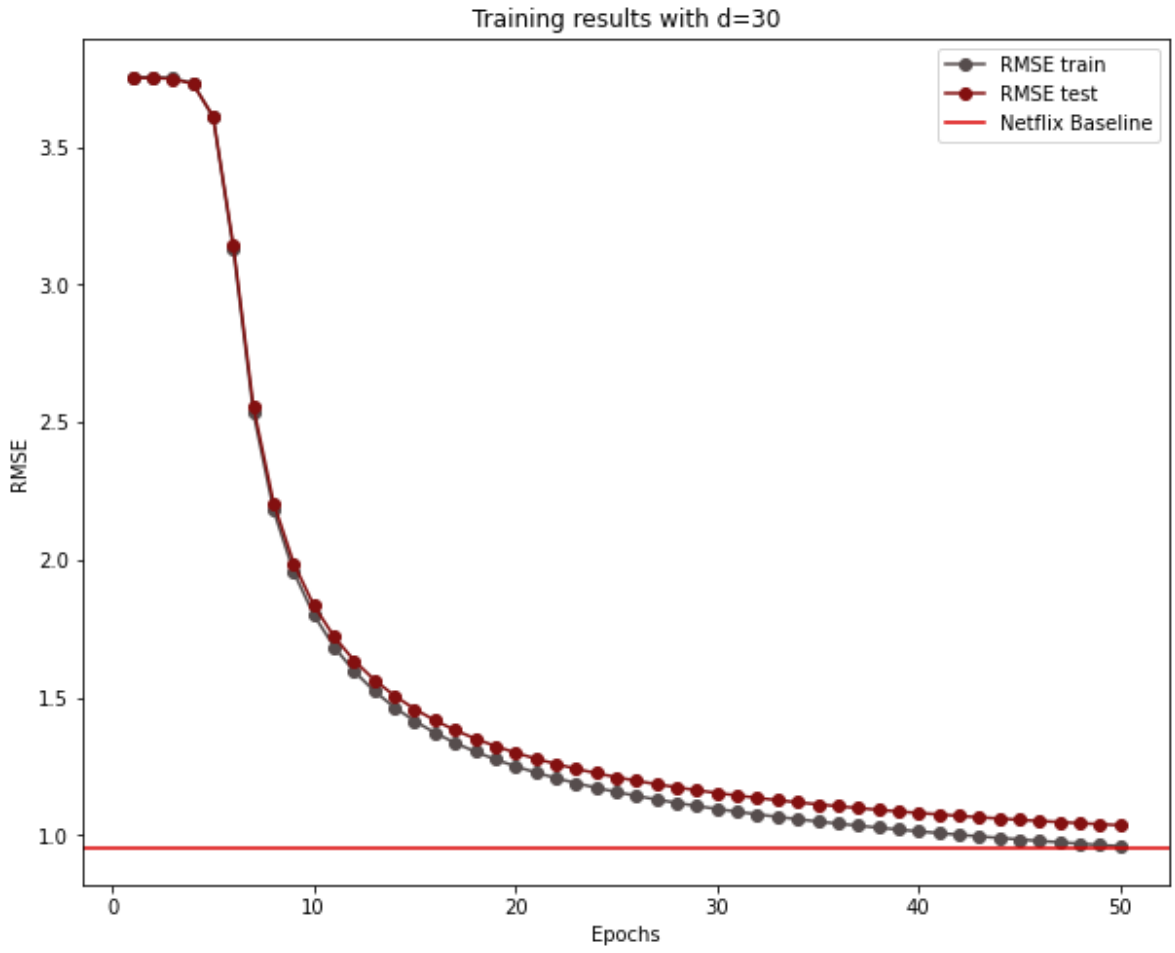
In [15]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result30)[:,0]
train_rmse_list = np.array(result30)[:,1]
test_rmse_list = np.array(result30)[:,2]

plt.title('Training results with d=30')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=40, lambda = 0.02

In [16]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=40, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)

import time
start = time.time()
result40 = mf.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 3.5873 ; Test RMSE = 3.5898
Iteration: 10 ; Train RMSE = 1.7890 ; Test RMSE = 1.8235
Iteration: 15 ; Train RMSE = 1.4115 ; Test RMSE = 1.4547
Iteration: 20 ; Train RMSE = 1.2499 ; Test RMSE = 1.2985
Iteration: 25 ; Train RMSE = 1.1585 ; Test RMSE = 1.2111
Iteration: 30 ; Train RMSE = 1.0983 ; Test RMSE = 1.1546
Iteration: 35 ; Train RMSE = 1.0543 ; Test RMSE = 1.1142
Iteration: 40 ; Train RMSE = 1.0196 ; Test RMSE = 1.0834
Iteration: 45 ; Train RMSE = 0.9909 ; Test RMSE = 1.0588
Iteration: 50 ; Train RMSE = 0.9668 ; Test RMSE = 1.0391
```

Proobablistic Matrix Factorization

In [19]:

```
import numpy as np
import pandas as pd
```

In [59]:

```
ratings = pd.read_csv('netflix_data.csv')
ratings.head()
```

Out[59]:

	user_id	movie_id	rating
0	2442	1	3.0
1	1719610	1	2.0
2	1011918	1	4.0
3	479924	1	5.0
4	2389367	1	1.0

In [60]:

```
ratings.shape
```

Out[60]:

```
(507852, 3)
```

In [61]:

```
user_dict = dict(zip(ratings.user_id.unique(), range(ratings.user_id.nunique())))
movie_dict = dict(zip(ratings.movie_id.unique(), range(ratings.movie_id.nunique())))
```

In [62]:

```
ratings_dict = ratings.replace({'user_id':user_dict})
ratings_dict = ratings_dict.replace({'movie_id':movie_dict})
```

In [63]:

```
ratings_dict
```

Out[63]:

	user_id	movie_id	rating
0	0	0	3.0
1	1	0	2.0
2	2	0	4.0
3	3	0	5.0
4	4	0	1.0
...	...	...	...
507847	257	4496	5.0
507848	8	4496	1.0
507849	3822	4496	1.0
507850	352	4496	4.0
507851	408	4496	1.0

507852 rows × 3 columns

https://github.com/fuhailin/Probabilistic-Matrix-Factorization/blob/master/RunExample.py
(https://github.com/fuhailin/Probabilistic-Matrix-Factorization/blob/master/RunExample.py)

In [64]:

```
# train test 분리
from sklearn.utils import shuffle
TRAIN_SIZE = 0.75
ratings_dict = shuffle(ratings_dict, random_state=1)
cutoff = int(TRAIN_SIZE * len(ratings_dict))
ratings_train = ratings_dict.iloc[:cutoff]
ratings_test = ratings_dict.iloc[cutoff:]
```

In [67]:

```
class PMF(object):
    def __init__(self, num_feat=10, epsilon=1, lamb_U=0.01, lamb_V = 0.01, momentum=0.8, maxepoch=2
        self.num_feat = num_feat  # Number of latent features
        self.epsilon = epsilon  # learning rate
        self.lamb_U = lamb_U
        self.lamb_V = lamb_V
        self.momentum = momentum  # momentum of the gradient
        self.maxepoch = maxepoch
        self.num_batches = num_batches  # Number of batches in each epoch (for SGD optimization)
        self.batch_size = batch_size  # Number of training samples used in each batches (for SGD opt

        self.w_Item = None  # Item feature vectors
        self.w_User = None  # User feature vectors

        self.training_process = []

    def fit(self, train_vec, test_vec):
        # mean subtraction
        self.mean_inv = np.mean(train_vec[:, 2])  # mean of grading

        pairs_train = train_vec.shape[0]
        pairs_test = test_vec.shape[0]

        # 1-p-i, 2-m-c
        num_user = int(max(np.amax(train_vec[:, 0]), np.amax(test_vec[:, 0]))) + 1  # Column 0, Tota
        num_item = int(max(np.amax(train_vec[:, 1]), np.amax(test_vec[:, 1]))) + 1  # Column 1, Tota

        incremental = False
        if ((not incremental) or (self.w_Item is None)):
            # initialize
            self.epoch = 0
            self.w_Item = 0.1 * np.random.randn(num_item, self.num_feat) # U 초기치로 정규분포 난수
            self.w_User = 0.1 * np.random.randn(num_user, self.num_feat) # V 초기치로 정규분포 난수

            self.w_Item_inc = np.zeros((num_item, self.num_feat)) # gradient 담을 공간
            self.w_User_inc = np.zeros((num_user, self.num_feat))

        while self.epoch < self.maxepoch:
            self.epoch += 1

            # Shuffle training truples
            shuffled_order = np.arange(train_vec.shape[0])
            np.random.shuffle(shuffled_order)

            # Batch update
            for batch in range(self.num_batches):

                test = np.arange(self.batch_size * batch, self.batch_size * (batch + 1))
                batch_idx = np.mod(test, shuffled_order.shape[0])

                batch_UserID = np.array(train_vec[shuffled_order[batch_idx], 0], dtype='int32')
                batch_ItemID = np.array(train_vec[shuffled_order[batch_idx], 1], dtype='int32')

                # Compute Objective Function
                pred_out = np.sum(np.multiply(self.w_User[batch_UserID, :],
                                              self.w_Item[batch_ItemID, :]), axis=1)

                rawErr = pred_out - train_vec[shuffled_order[batch_idx], 2] + self.mean_inv
```

```
            # Compute gradients
            lx_User = 2 * np.multiply(rawErr[:, np.newaxis], self.w_Item[batch_ItemID, :]) ₩
                  + self.lamb_U * self.w_User[batch_UserID, :]
            lx_Item = 2 * np.multiply(rawErr[:, np.newaxis], self.w_User[batch_UserID, :]) ₩
                  + self.lamb_V * (self.w_Item[batch_ItemID, :])  # np.newaxis :increase the di

            dw_Item = np.zeros((num_item, self.num_feat))
            dw_User = np.zeros((num_user, self.num_feat))

            # loop to aggreate the gradients of the same element
            for i in range(self.batch_size):
                dw_Item[batch_ItemID[i], :] += lx_Item[i, :]
                dw_User[batch_UserID[i], :] += lx_User[i, :]

            # Update with momentum
            self.w_Item_inc = self.momentum * self.w_Item_inc + self.epsilon * dw_Item / self.ba
            self.w_User_inc = self.momentum * self.w_User_inc + self.epsilon * dw_User / self.ba

            self.w_Item = self.w_Item - self.w_Item_inc
            self.w_User = self.w_User - self.w_User_inc


            if batch == self.num_batches - 1:

                # Compute Objective Function after
                pred_out = np.sum(np.multiply(self.w_User[np.array(train_vec[:, 0], dtype='int32
                                              self.w_Item[np.array(train_vec[:, 1], dtype='int32
                                  axis=1)  # mean_inv subtracted
                rawErr = pred_out - train_vec[:, 2] + self.mean_inv
                obj = np.linalg.norm(rawErr) ** 2 + 0.5 * self.lamb_U * np.linalg.norm(self.w_U
                    + 0.5 * self.lamb_V * np.linalg.norm(self.w_Item) ** 2
                rmse1 = np.sqrt(obj / pairs_train)

                # Compute validation error
                pred_out = np.sum(np.multiply(self.w_User[np.array(test_vec[:, 0], dtype='int32'
                                              self.w_Item[np.array(test_vec[:, 1], dtype='int32'
                                  axis=1)  # mean_inv subtracted
                rawErr = pred_out - test_vec[:, 2] + self.mean_inv
                rmse2 = np.linalg.norm(rawErr) / np.sqrt(pairs_test)

                self.training_process.append((self.epoch, rmse1, rmse2))

                if self.epoch % 5 == 0:
                    print('iteration %f : Training RMSE %f, Test RMSE %f' % (self.epoch, rmse1,
        return self.training_process, self.w_User, self.w_Item

    def predict(self, invID):
        return np.dot(self.w_Item, self.w_User[int(invID), :]) + self.mean_inv

    # ****************Set parameters by providing a parameter dictionary.  ***********#
    def set_params(self, parameters):
        if isinstance(parameters, dict):
            self.num_feat = parameters.get("num_feat", 10)
            self.epsilon = parameters.get("epsilon", 1)
            #self._lambda = parameters.get("_lambda", 0.1)
            self.lamb_U = parameters.get("lamb_U", 0.01)
            self.lamb_V = parameters.get("lamb_V", 0.01)
            self.momentum = parameters.get("momentum", 0.8)
            self.maxepoch = parameters.get("maxepoch", 20)
            self.num_batches = parameters.get("num_batches", 10)
            self.batch_size = parameters.get("batch_size", 1000)
```

lambda_U = 0.01, lambda_V = 0.001

In [101]:

```
pmf1 = PMF()
pmf1.set_params({"num_feat": 10, "epsilon": 1, "lamb_U": 0.01, "lamb_V" : 0.001, "momentum": 0.9, "m
                 "batch_size": 10000})
```

In [102]:

```
import time
start = time.time()

pmf_result1, U_1, V_1 = pmf1.fit(ratings_train.values, ratings_test.values)

print(time.time()-start)
```

```
iteration 5.000000 : Training RMSE 1.092139, Test RMSE 1.097748
iteration 10.000000 : Training RMSE 1.083200, Test RMSE 1.093431
iteration 15.000000 : Training RMSE 1.036673, Test RMSE 1.061763
iteration 20.000000 : Training RMSE 0.966133, Test RMSE 1.016425
iteration 25.000000 : Training RMSE 0.910264, Test RMSE 0.985555
iteration 30.000000 : Training RMSE 0.873517, Test RMSE 0.970445
iteration 35.000000 : Training RMSE 0.847331, Test RMSE 0.963829
iteration 40.000000 : Training RMSE 0.827433, Test RMSE 0.961478
iteration 45.000000 : Training RMSE 0.811755, Test RMSE 0.961048
iteration 50.000000 : Training RMSE 0.799079, Test RMSE 0.961526
93.45875072479248
```

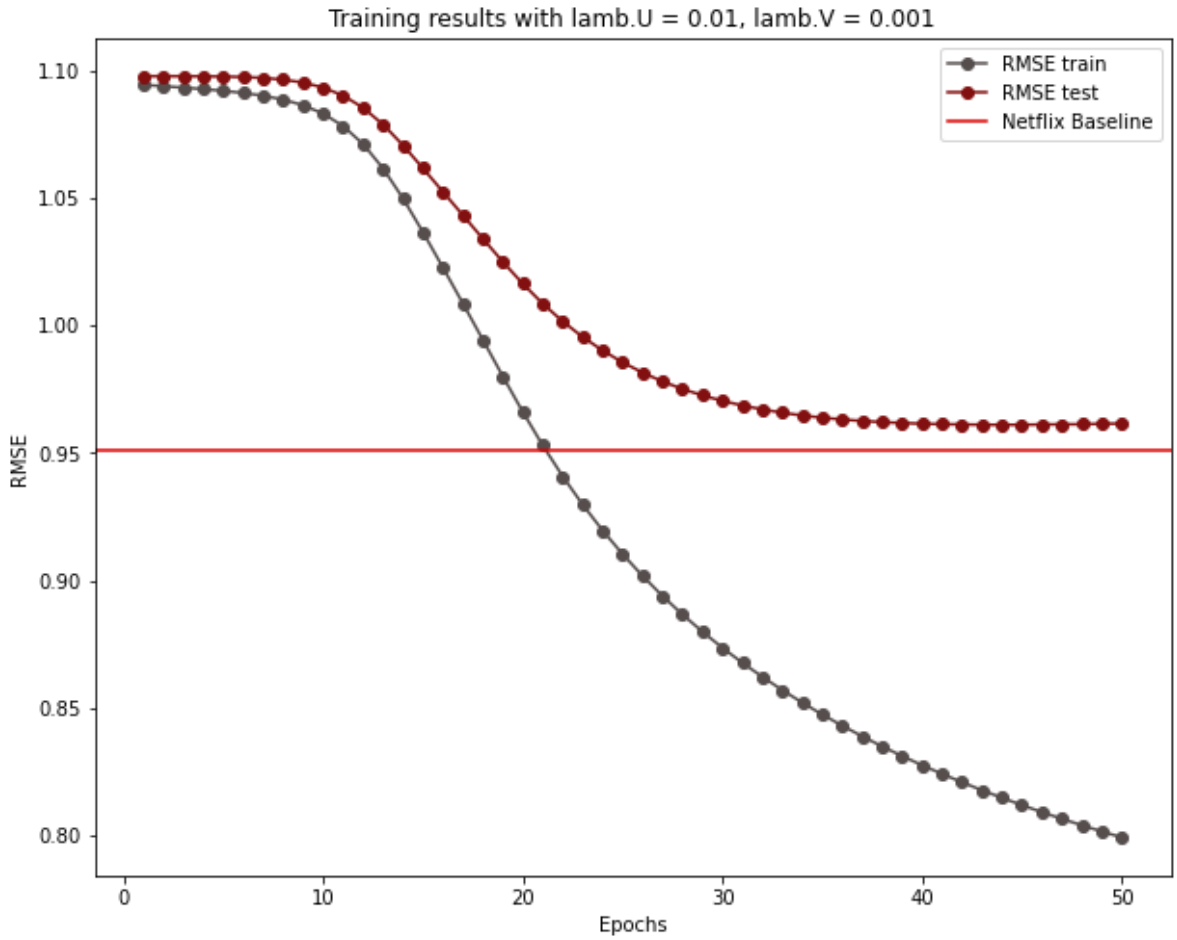
In [103]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(pmf_result1)[:,0]
train_rmse_list = np.array(pmf_result1)[:,1]
test_rmse_list = np.array(pmf_result1)[:,2]

plt.title('Training results with lamb.U = 0.01, lamb.V = 0.001')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



lambda_U = 0.001, lambda_V = 0.0001

In [81]:

```
pmf2 = PMF()
pmf2.set_params({"num_feat": 10, "epsilon": 1, "lamb_U": 0.001, "lamb_V" : 0.0001, "momentum": 0.9,
                 "batch_size": 10000})
```

In [82]:

```
import time
start = time.time()

pmf_result2, U_2, V_2 = pmf2.fit(ratings_train.values, ratings_test.values)

print(time.time()-start)
```

```
iteration 5.000000 : Training RMSE 1.092469, Test RMSE 1.097938
iteration 10.000000 : Training RMSE 1.086773, Test RMSE 1.096717
iteration 15.000000 : Training RMSE 1.053803, Test RMSE 1.079220
iteration 20.000000 : Training RMSE 0.975602, Test RMSE 1.031999
iteration 25.000000 : Training RMSE 0.920326, Test RMSE 0.999957
iteration 30.000000 : Training RMSE 0.882587, Test RMSE 0.983795
iteration 35.000000 : Training RMSE 0.854072, Test RMSE 0.976082
iteration 40.000000 : Training RMSE 0.832624, Test RMSE 0.972781
iteration 45.000000 : Training RMSE 0.816059, Test RMSE 0.971884
iteration 50.000000 : Training RMSE 0.802856, Test RMSE 0.971628
87.4956464767456
```

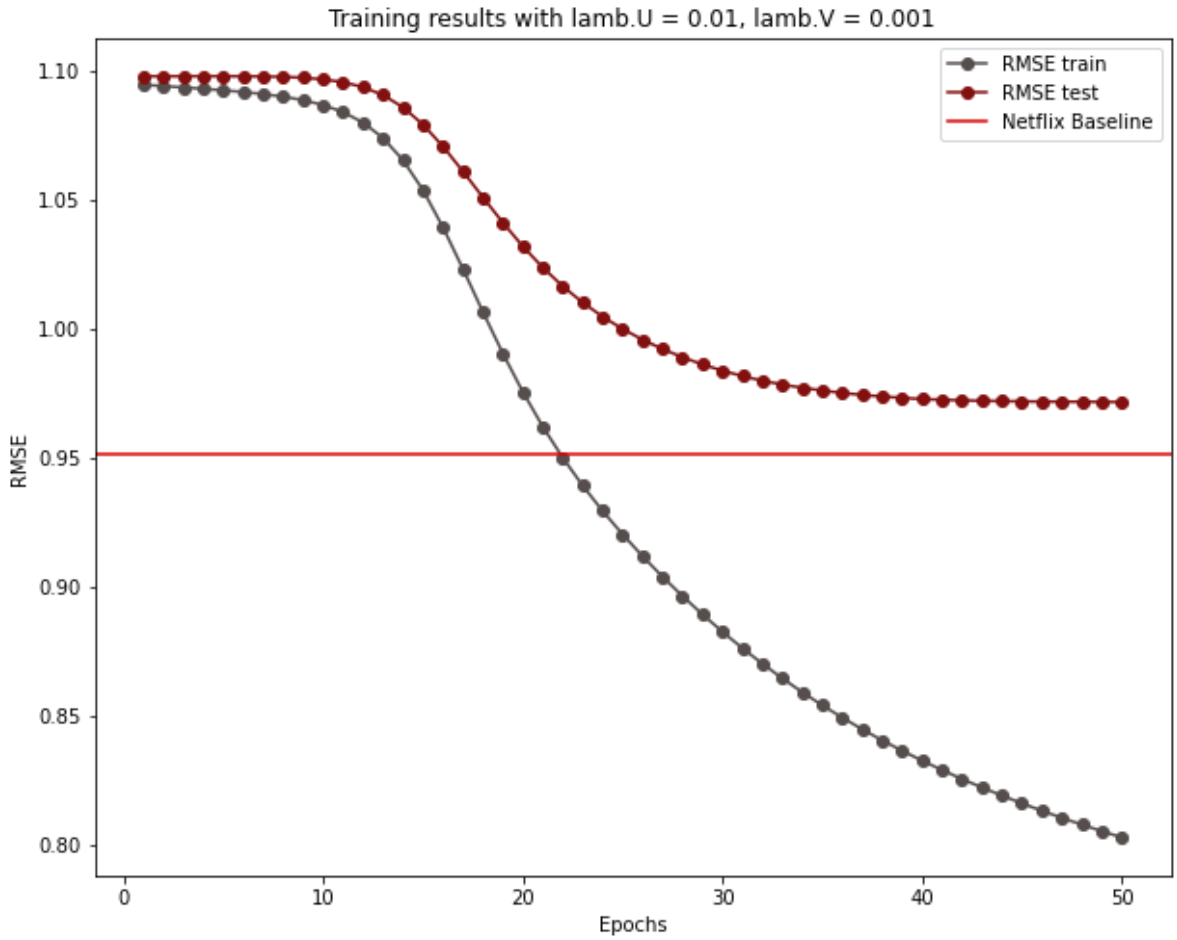
In [83]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(pmf_result2)[:,0]
train_rmse_list = np.array(pmf_result2)[:,1]
test_rmse_list = np.array(pmf_result2)[:,2]

plt.title('Training results with lamb.U = 0.01, lamb.V = 0.001')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



lambda_U = 0.001, lambda_V = 0.001

In [84]:

```
pmf3 = PMF()
pmf3.set_params({"num_feat": 10, "epsilon": 1, "lamb_U": 0.001, "lamb_V" : 0.001, "momentum": 0.9, "
                 "batch_size": 10000})
```

In [85]:

```
import time
start = time.time()

pmf_result3, U_3, V_3 = pmf3.fit(ratings_train.values, ratings_test.values)

print(time.time()-start)
```

```
iteration 5.000000 : Training RMSE 1.092101, Test RMSE 1.097619
iteration 10.000000 : Training RMSE 1.081648, Test RMSE 1.093030
iteration 15.000000 : Training RMSE 1.026036, Test RMSE 1.058925
iteration 20.000000 : Training RMSE 0.954062, Test RMSE 1.012528
iteration 25.000000 : Training RMSE 0.908270, Test RMSE 0.986242
iteration 30.000000 : Training RMSE 0.875367, Test RMSE 0.972549
iteration 35.000000 : Training RMSE 0.849699, Test RMSE 0.965967
iteration 40.000000 : Training RMSE 0.829462, Test RMSE 0.963364
iteration 45.000000 : Training RMSE 0.813321, Test RMSE 0.962620
iteration 50.000000 : Training RMSE 0.800215, Test RMSE 0.962753
87.41289448738098
```

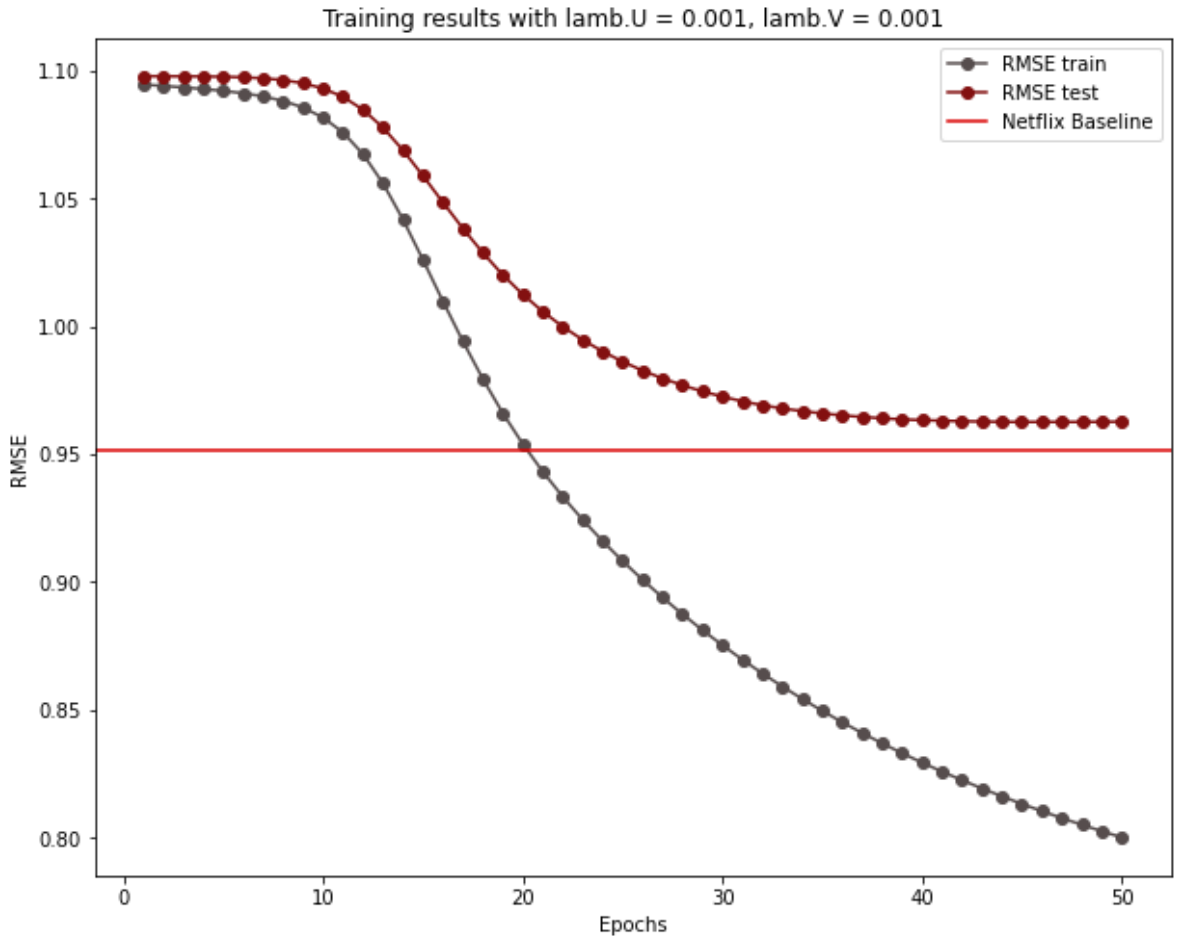
In [86]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(pmf_result3)[:,0]
train_rmse_list = np.array(pmf_result3)[:,1]
test_rmse_list = np.array(pmf_result3)[:,2]

plt.title('Training results with lamb.U = 0.001, lamb.V = 0.001')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



lambda_U = 0.01, lambda_V = 0.001 -- 학습률 0.1

In [218]:

```
pmf3_2 = PMF()
pmf3_2.set_params({"num_feat": 10, "epsilon": 0.1, "lamb_U": 0.01, "lamb_V" : 0.001, "momentum": 0.9
                   "batch_size": 10000})
```

In [219]:

```
import time
start = time.time()

pmf_result3_2, U_3_2, V_3_2 = pmf3_2.fit(ratings_train.values, ratings_test.values)

print(time.time()-start)
```

```
iteration 5.000000 : Training RMSE 1.094744, Test RMSE 1.097850
iteration 10.000000 : Training RMSE 1.094453, Test RMSE 1.097837
iteration 15.000000 : Training RMSE 1.094164, Test RMSE 1.097826
iteration 20.000000 : Training RMSE 1.093873, Test RMSE 1.097814
iteration 25.000000 : Training RMSE 1.093576, Test RMSE 1.097798
iteration 30.000000 : Training RMSE 1.093267, Test RMSE 1.097778
iteration 35.000000 : Training RMSE 1.092938, Test RMSE 1.097749
iteration 40.000000 : Training RMSE 1.092584, Test RMSE 1.097706
iteration 45.000000 : Training RMSE 1.092194, Test RMSE 1.097644
iteration 50.000000 : Training RMSE 1.091755, Test RMSE 1.097558
346.0560326576233
```

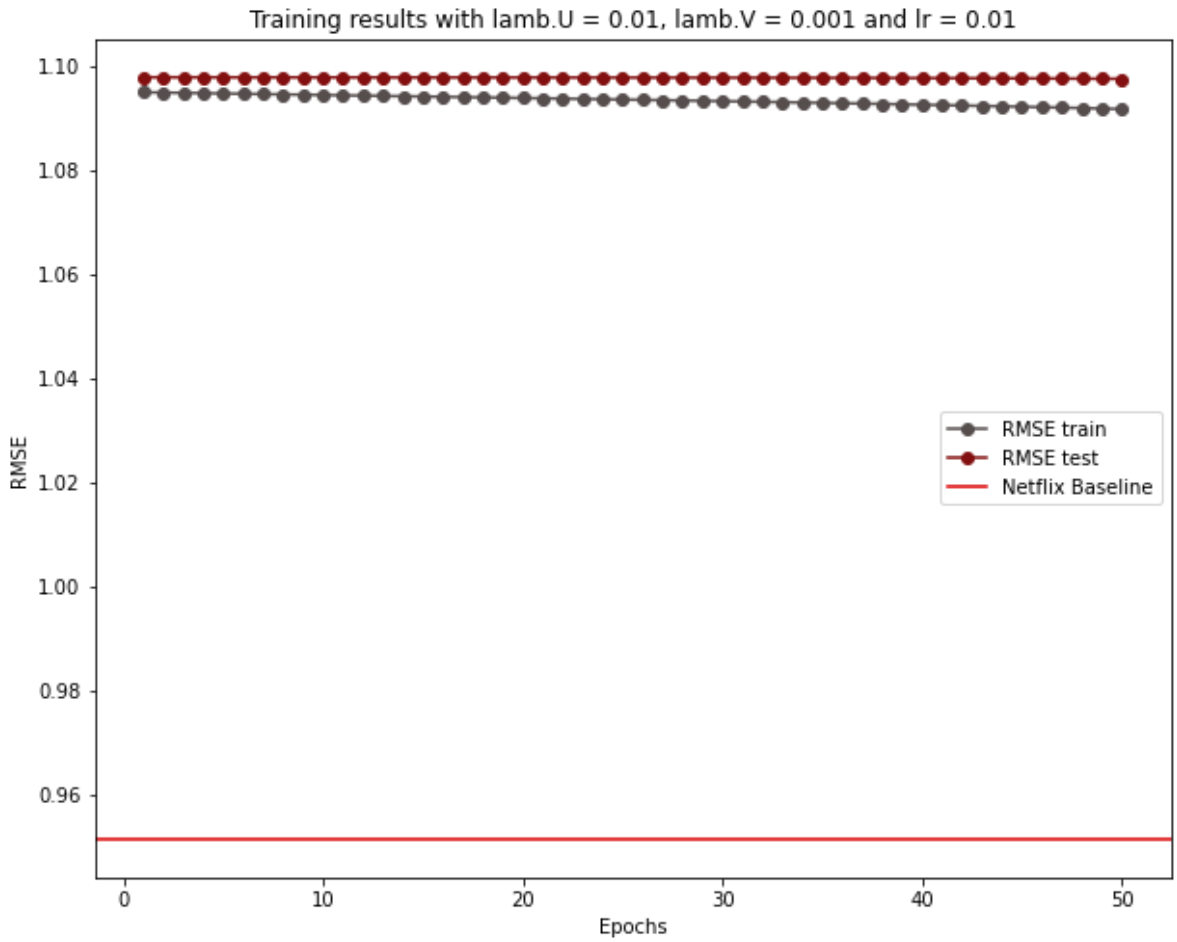
In [221]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(pmf_result3_2)[:,0]
train_rmse_list = np.array(pmf_result3_2)[:,1]
test_rmse_list = np.array(pmf_result3_2)[:,2]

plt.title('Training results with lamb.U = 0.01, lamb.V = 0.001 and lr = 0.01')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



lambda_U = 0.01, lambda_V = 0.001 -- 학습률 0.05

In [222]:

```
pmf3_3 = PMF()
pmf3_3.set_params({"num_feat": 10, "epsilon": 0.05, "lamb_U": 0.01, "lamb_V" : 0.001, "momentum": 0.
                   "batch_size": 10000})
```

In [223]:

```
import time
start = time.time()

pmf_result3_3, U_3_3, V_3_3 = pmf3_3.fit(ratings_train.values, ratings_test.values)

print(time.time()-start)
```

```
iteration 5.000000 : Training RMSE 1.094916, Test RMSE 1.097988
iteration 10.000000 : Training RMSE 1.094761, Test RMSE 1.097982
iteration 15.000000 : Training RMSE 1.094609, Test RMSE 1.097978
iteration 20.000000 : Training RMSE 1.094459, Test RMSE 1.097974
iteration 25.000000 : Training RMSE 1.094311, Test RMSE 1.097971
iteration 30.000000 : Training RMSE 1.094162, Test RMSE 1.097969
iteration 35.000000 : Training RMSE 1.094014, Test RMSE 1.097966
iteration 40.000000 : Training RMSE 1.093865, Test RMSE 1.097964
iteration 45.000000 : Training RMSE 1.093714, Test RMSE 1.097962
iteration 50.000000 : Training RMSE 1.093562, Test RMSE 1.097958
342.7715289592743
```

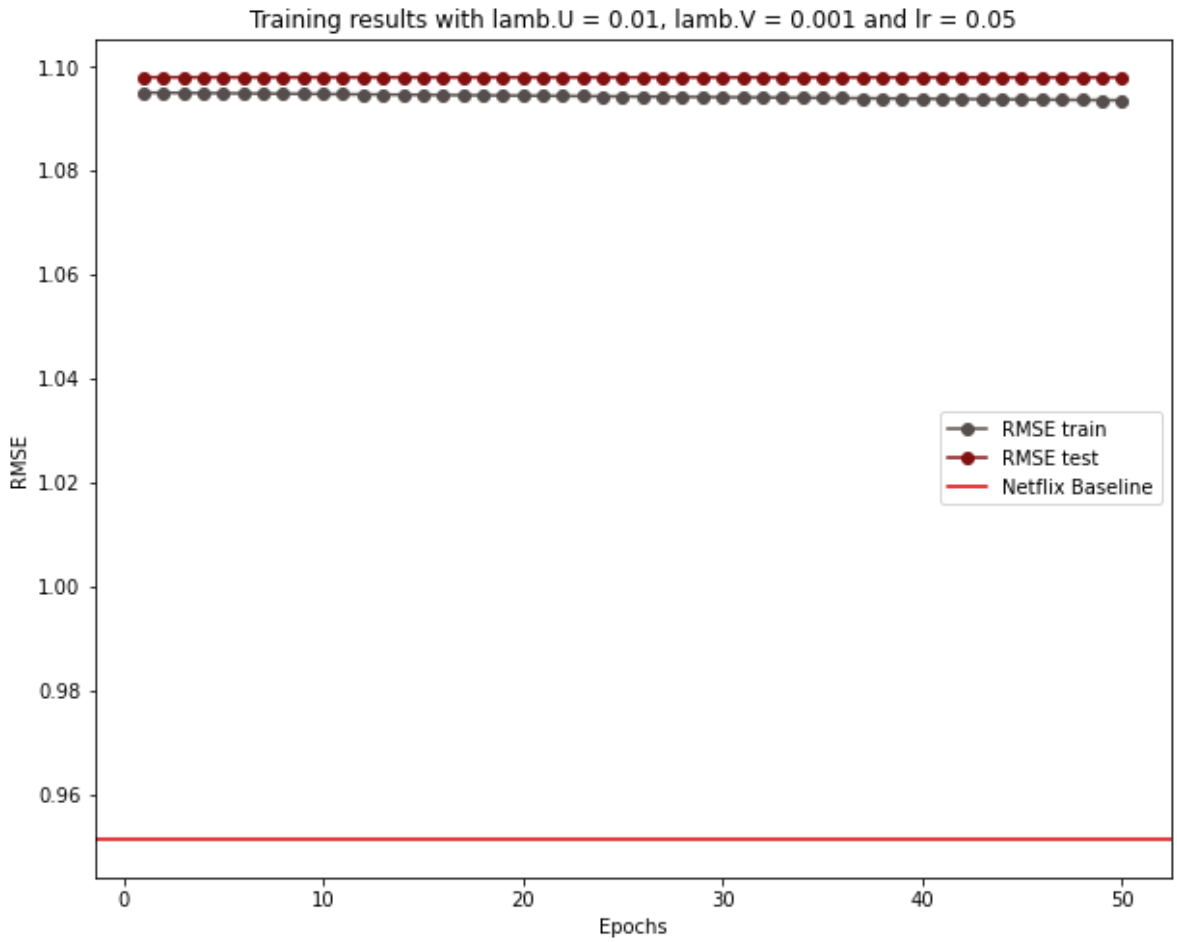
In [224]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(pmf_result3_3)[:,0]
train_rmse_list = np.array(pmf_result3_3)[:,1]
test_rmse_list = np.array(pmf_result3_3)[:,2]

plt.title('Training results with lamb.U = 0.01, lamb.V = 0.001 and lr = 0.05')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



lambda_U = 0.01, lambda_V = 0.001 -- 학습률 2

In [225]:

```
pmf3_4 = PMF()
pmf3_4.set_params({"num_feat": 10, "epsilon": 2, "lamb_U": 0.01, "lamb_V" : 0.001, "momentum": 0.9,
                   "batch_size": 10000})
```

In [226]:

```
import time
start = time.time()

pmf_result3_4, U_3_4, V_3_4 = pmf3_4.fit(ratings_train.values, ratings_test.values)

print(time.time()-start)
```

```
iteration 5.000000 : Training RMSE 1.083466, Test RMSE 1.093764
iteration 10.000000 : Training RMSE 0.959821, Test RMSE 1.014407
iteration 15.000000 : Training RMSE 0.879455, Test RMSE 0.975586
iteration 20.000000 : Training RMSE 0.833035, Test RMSE 0.966096
iteration 25.000000 : Training RMSE 0.803421, Test RMSE 0.965001
iteration 30.000000 : Training RMSE 0.783386, Test RMSE 0.966616
iteration 35.000000 : Training RMSE 0.768821, Test RMSE 0.968616
iteration 40.000000 : Training RMSE 0.757597, Test RMSE 0.970956
iteration 45.000000 : Training RMSE 0.748618, Test RMSE 0.973160
iteration 50.000000 : Training RMSE 0.741224, Test RMSE 0.975219
414.8878321647644
```

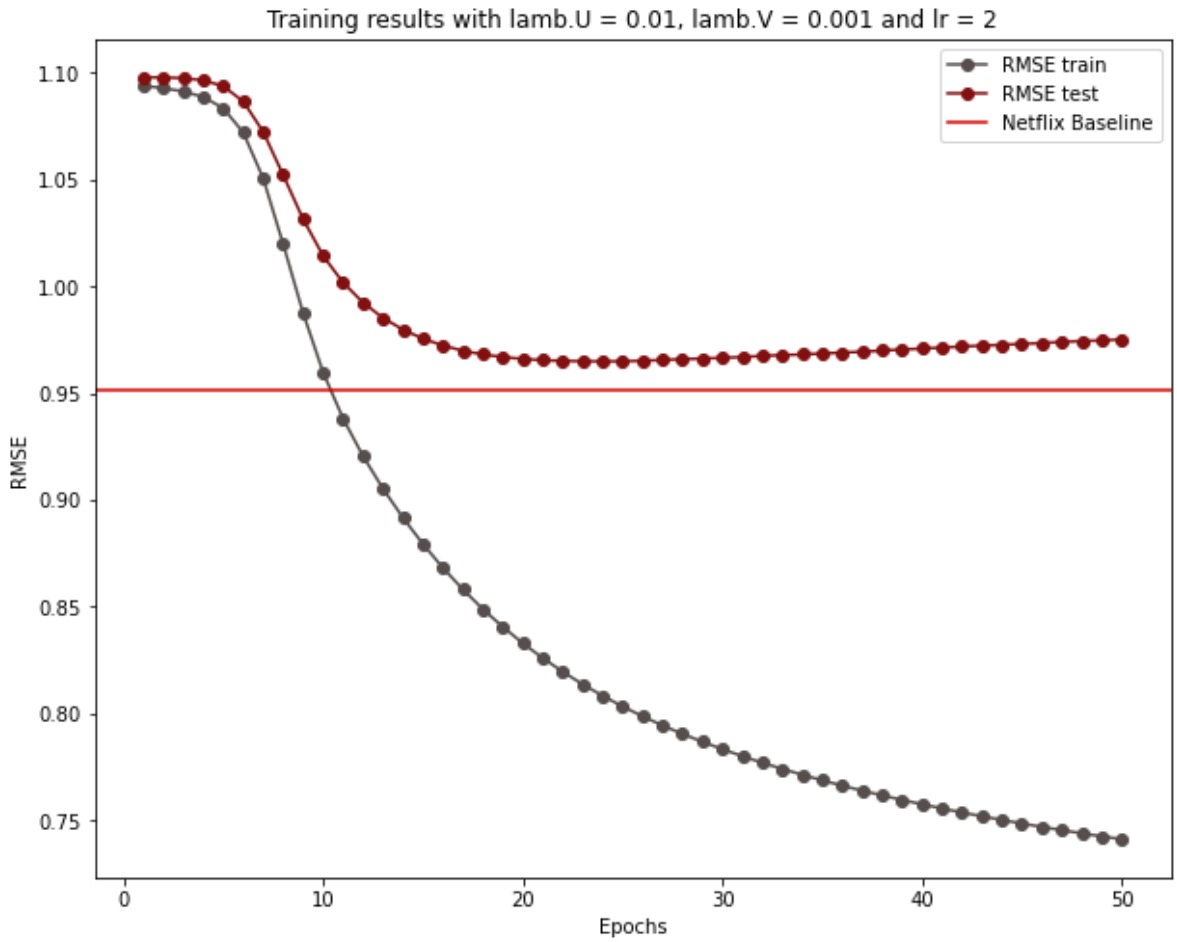
In [227]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(pmf_result3_4)[:,0]
train_rmse_list = np.array(pmf_result3_4)[:,1]
test_rmse_list = np.array(pmf_result3_4)[:,2]

plt.title('Training results with lamb.U = 0.01, lamb.V = 0.001 and lr = 2')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



lambda_U = 0.01, lambda_V = 0.001 -- momentum 0.8

In [269]:

```
pmf4 = PMF()
pmf4.set_params({"num_feat": 10, "epsilon": 1, "lamb_U": 0.01, "lamb_V" : 0.001, "momentum": 0.8, "m
                 "batch_size": 10000})
```

In [229]:

```
#### import time
start = time.time()

pmf_result4, U_4, V_4 = pmf4.fit(ratings_train.values, ratings_test.values)

print(time.time()-start)
```

```
iteration 5.000000 : Training RMSE 1.093582, Test RMSE 1.097825
iteration 10.000000 : Training RMSE 1.091947, Test RMSE 1.097682
iteration 15.000000 : Training RMSE 1.089144, Test RMSE 1.096997
iteration 20.000000 : Training RMSE 1.082557, Test RMSE 1.094202
iteration 25.000000 : Training RMSE 1.065710, Test RMSE 1.084927
iteration 30.000000 : Training RMSE 1.031744, Test RMSE 1.063578
iteration 35.000000 : Training RMSE 0.989350, Test RMSE 1.035103
iteration 40.000000 : Training RMSE 0.953531, Test RMSE 1.011821
iteration 45.000000 : Training RMSE 0.926222, Test RMSE 0.995666
iteration 50.000000 : Training RMSE 0.904230, Test RMSE 0.984106
iteration 55.000000 : Training RMSE 0.885840, Test RMSE 0.975752
iteration 60.000000 : Training RMSE 0.870245, Test RMSE 0.969903
iteration 65.000000 : Training RMSE 0.856818, Test RMSE 0.965863
iteration 70.000000 : Training RMSE 0.845176, Test RMSE 0.963349
iteration 75.000000 : Training RMSE 0.834949, Test RMSE 0.961743
iteration 80.000000 : Training RMSE 0.825932, Test RMSE 0.960836
iteration 85.000000 : Training RMSE 0.817907, Test RMSE 0.960544
iteration 90.000000 : Training RMSE 0.810723, Test RMSE 0.960505
iteration 95.000000 : Training RMSE 0.804259, Test RMSE 0.960696
iteration 100.000000 : Training RMSE 0.798410, Test RMSE 0.961010
904.6325843334198
```

In [230]:

```
pd.DataFrame(pmf_result4, columns = ['epoch', 'train_rmse', 'test_rmse']).to_csv('pmf_result.csv', i
```

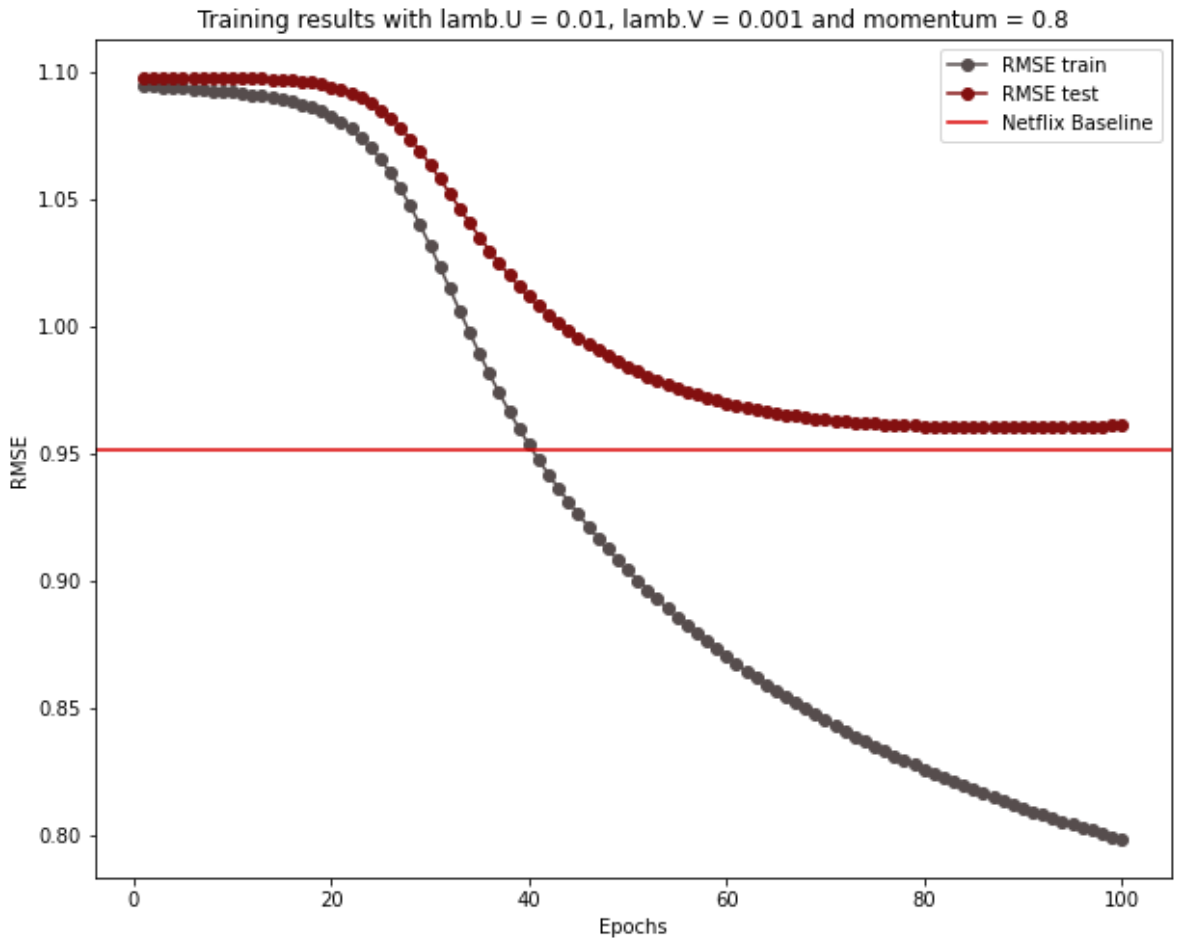
In [231]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(pmf_result4)[:,0]
train_rmse_list = np.array(pmf_result4)[:,1]
test_rmse_list = np.array(pmf_result4)[:,2]

plt.title('Training results with lamb.U = 0.01, lamb.V = 0.001 and momentum = 0.8')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



lambda_U = 0.01, lambda_V = 0.001 -- momentum 0.5

In [232]:

```
pmf4_2 = PMF()
pmf4_2.set_params({"num_feat": 10, "epsilon": 1, "lamb_U": 0.01, "lamb_V" : 0.001, "momentum": 0.5,
                   "batch_size": 10000})
```

In [233]:

```
import time
start = time.time()

pmf_result4_2, U_4_2, V_4_2 = pmf4_2.fit(ratings_train.values, ratings_test.values)

print(time.time()-start)
```

```
iteration 5.000000 : Training RMSE 1.094399, Test RMSE 1.097937
iteration 10.000000 : Training RMSE 1.093829, Test RMSE 1.097931
iteration 15.000000 : Training RMSE 1.093238, Test RMSE 1.097925
iteration 20.000000 : Training RMSE 1.092597, Test RMSE 1.097907
iteration 25.000000 : Training RMSE 1.091850, Test RMSE 1.097856
iteration 30.000000 : Training RMSE 1.090925, Test RMSE 1.097746
iteration 35.000000 : Training RMSE 1.089718, Test RMSE 1.097526
iteration 40.000000 : Training RMSE 1.088050, Test RMSE 1.097114
iteration 45.000000 : Training RMSE 1.085665, Test RMSE 1.096364
iteration 50.000000 : Training RMSE 1.082167, Test RMSE 1.095052
iteration 55.000000 : Training RMSE 1.077020, Test RMSE 1.092814
iteration 60.000000 : Training RMSE 1.069579, Test RMSE 1.089168
iteration 65.000000 : Training RMSE 1.059232, Test RMSE 1.083570
iteration 70.000000 : Training RMSE 1.045814, Test RMSE 1.075613
iteration 75.000000 : Training RMSE 1.029801, Test RMSE 1.065410
iteration 80.000000 : Training RMSE 1.012419, Test RMSE 1.053759
iteration 85.000000 : Training RMSE 0.995146, Test RMSE 1.041906
iteration 90.000000 : Training RMSE 0.978996, Test RMSE 1.030802
iteration 95.000000 : Training RMSE 0.964381, Test RMSE 1.020944
iteration 100.000000 : Training RMSE 0.951259, Test RMSE 1.012351
888.2746319770813
```

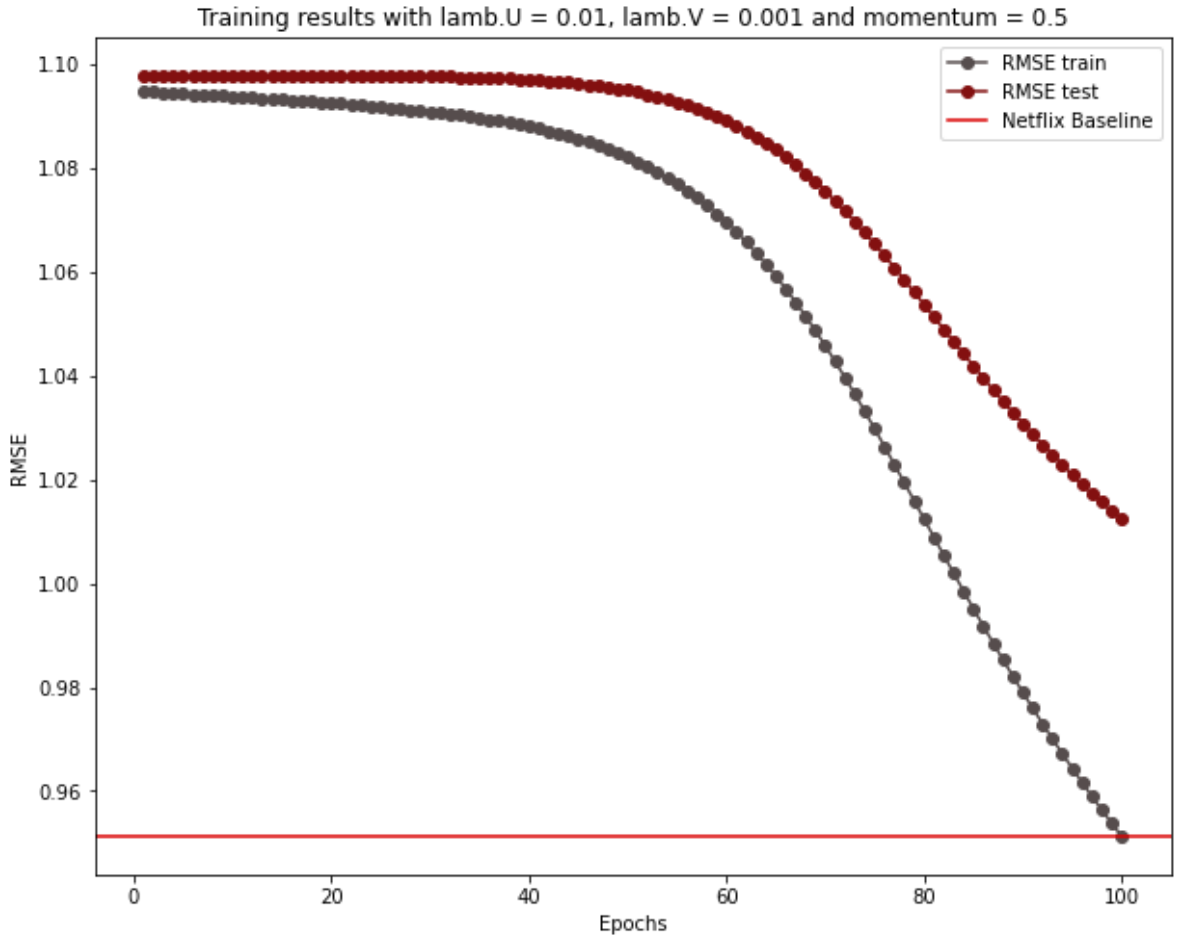
In [234]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(pmf_result4_2)[:,0]
train_rmse_list = np.array(pmf_result4_2)[:,1]
test_rmse_list = np.array(pmf_result4_2)[:,2]

plt.title('Training results with lamb.U = 0.01, lamb.V = 0.001 and momentum = 0.5')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



최종 모델 결과 확인

In [256]:

```
df_movies = pd.read_csv('netflix_data/movie_titles.csv', encoding = "ISO-8859-1", header = None, na
df_movies = df_movies.replace({'user_id':user_dict})
df_movies = df_movies.replace({'movie_id':movie_dict})
```

In [257]:

```
user_id = user_dict.get(2442)
print(user_id)
```

0

In [258]:

```
R_2442 = pd.DataFrame(pmf4.predict(user_id)).reset_index()
R_2442.columns = ['movie_id','rating_pred']
```

In [259]:

```
R_2442.head()
```

Out[259]:

	movie_id	rating_pred
0	0	3.565371
1	1	3.641833
2	2	3.778583
3	3	3.624064
4	4	3.646987

In [260]:

```
df_movies.head()
```

Out[260]:

	movie_id	year	title
0	0	2003.0	Dinosaur Planet
1	1	2004.0	Isle of Man TT 2004 Review
2	2	1997.0	Character
3	3	1994.0	Paula Abdul's Get Up & Dance
4	4	2004.0	The Rise and Fall of ECW

In [261]:

```
ratings_dict.movie_id.unique()
```

Out[261]:

```
array([4303, 3667, 4352, ..., 1810,  548, 3745], dtype=int64)
```

In [262]:

```
R_result = pd.merge(pd.merge(df_movies, R_2442, how='inner', on='movie_id'),
                    ratings_dict[ratings_dict['user_id']==user_id].drop('user_id',axis=1), how='lef
```

In [263]:

```
R_fitted = R_result[R_result['rating'].isna()==False]
R_fitted = R_fitted[['movie_id','title','rating','rating_pred']].reset_index(drop=True)
```

In [265]:

```
R_fitted.head(10)
```

Out[265]:

	movie_id	title	rating	rating_pred
0	0	Dinosaur Planet	3.0	3.565371
1	29	Something's Gotta Give	3.0	3.823515
2	187	Dead Birds	3.0	3.408520
3	190	X2: X-Men United	4.0	4.054227
4	282	If These Walls Could Talk	4.0	3.710069
5	456	Kill Bill: Vol. 2	3.0	3.366525
6	485	Journey to the Center of the Earth	4.0	3.456025
7	513	Santana: Supernatural Live	3.0	3.543549
8	527	The Hitchhiker's Guide to the Galaxy	4.0	2.736385
9	593	By Hook or By Crook	2.0	3.527677

In [266]:

```
# 원래 취향
R_fitted.sort_values(by=['rating'], ascending=False).head(10)
```

Out[266]:

	movie_id	title	rating	rating_pred
18	1082	Walking with Prehistoric Beasts	5.0	3.832549
73	2860	The Silence of the Lambs	5.0	4.207895
23	1389	Yanni: Live at the Acropolis	5.0	3.579948
78	3110	Dante's Peak	5.0	3.291114
79	3181	Desert Hearts	5.0	3.580797
26	1468	Bend It Like Beckham	5.0	3.705519
27	1480	Beyond Borders	5.0	3.492514
92	3668	Laughing Matters	5.0	3.733208
57	2450	Lord of the Rings: The Fellowship of the Ring	5.0	4.484694
31	1707	Clash of the Titans	5.0	3.726379

In [267]:

```
R_pred = R_result[R_result['rating'].isna()==True]
R_pred = R_pred[['movie_id','title','rating','rating_pred']].reset_index(drop=True)
```

In [268]:

```
# 추천
R_pred.sort_values(by=['rating_pred'], ascending=False).head(10)
```

Out[268]:

	movie_id	title	rating	rating_pred
2115	2160	CSI: Season 1	NaN	4.631397
4193	4303	The Sixth Sense	NaN	4.477036
2708	2780	Braveheart	NaN	4.430554
2486	2546	Gilmore Girls: Season 1	NaN	4.411368
1448	1474	Six Feet Under: Season 4	NaN	4.360109
265	269	Sex and the City: Season 4	NaN	4.336044
3027	3103	Ghost	NaN	4.332624
2068	2112	Firefly	NaN	4.318385
1765	1796	Lethal Weapon	NaN	4.307304
3000	3077	The Lion King: Special Edition	NaN	4.303823

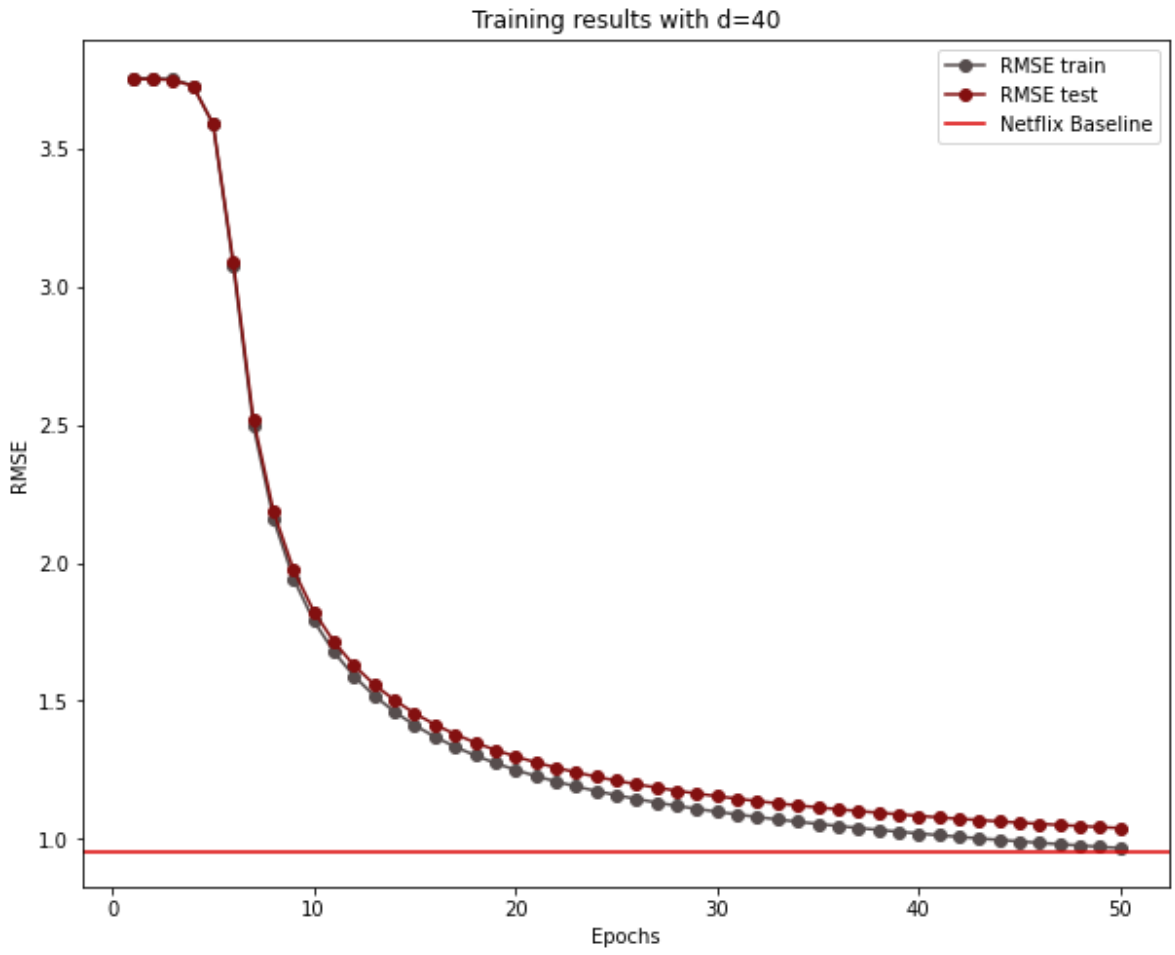
In [17]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result40)[:,0]
train_rmse_list = np.array(result40)[:,1]
test_rmse_list = np.array(result40)[:,2]

plt.title('Training results with d=40')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=50, lambda = 0.02

In [18]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=50, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)
result50 = mf.test()
```

```
Iteration: 5 ; Train RMSE = 3.6222 ; Test RMSE = 3.6235
Iteration: 10 ; Train RMSE = 1.8062 ; Test RMSE = 1.8401
Iteration: 15 ; Train RMSE = 1.4171 ; Test RMSE = 1.4599
Iteration: 20 ; Train RMSE = 1.2525 ; Test RMSE = 1.3007
Iteration: 25 ; Train RMSE = 1.1599 ; Test RMSE = 1.2121
Iteration: 30 ; Train RMSE = 1.0991 ; Test RMSE = 1.1552
Iteration: 35 ; Train RMSE = 1.0550 ; Test RMSE = 1.1145
Iteration: 40 ; Train RMSE = 1.0203 ; Test RMSE = 1.0837
Iteration: 45 ; Train RMSE = 0.9917 ; Test RMSE = 1.0590
Iteration: 50 ; Train RMSE = 0.9674 ; Test RMSE = 1.0391
```

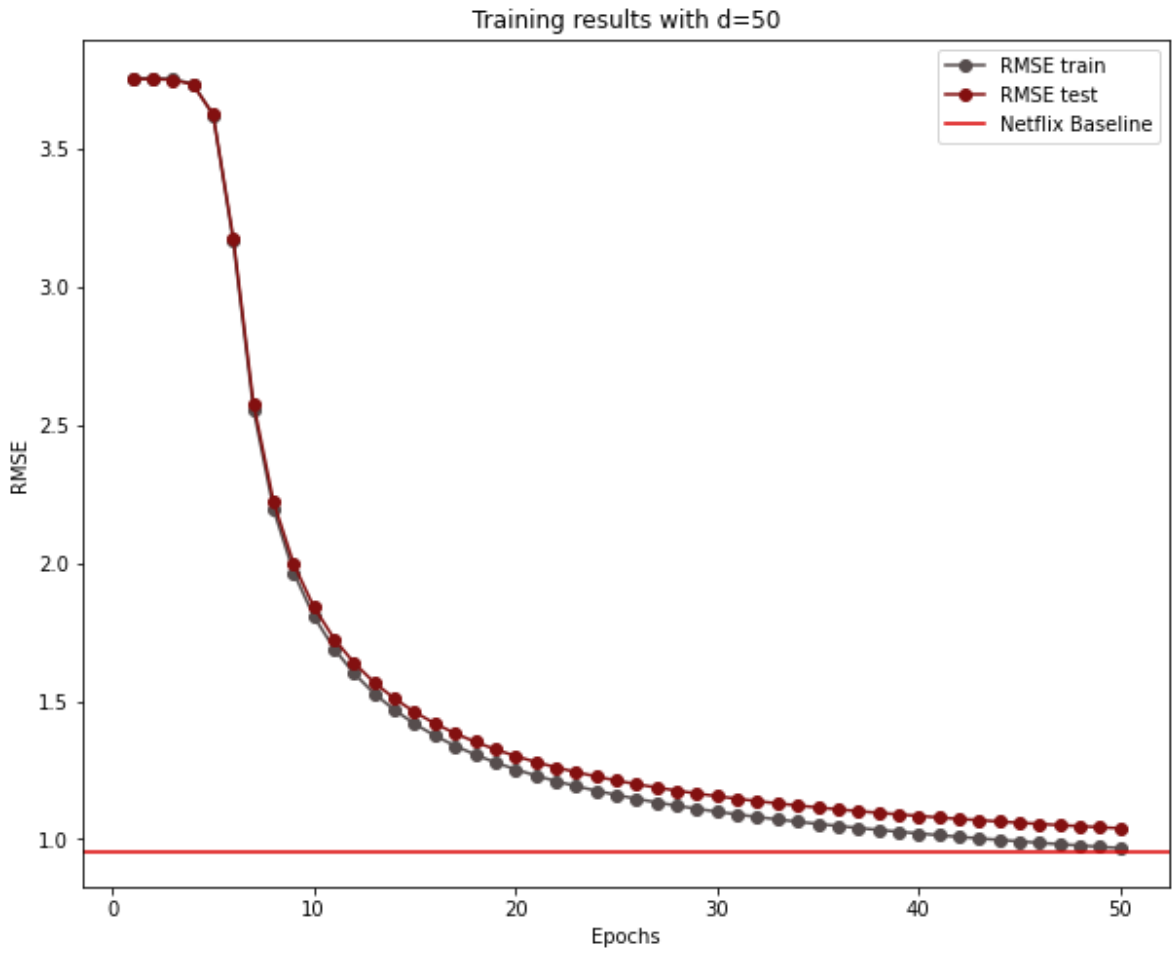
In [19]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result50)[:,0]
train_rmse_list = np.array(result50)[:,1]
test_rmse_list = np.array(result50)[:,2]

plt.title('Training results with d=50')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=10, lambda = 0.01

In [20]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.01, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)
result01 = mf.test()
```

```
Iteration: 5 ; Train RMSE = 3.3959 ; Test RMSE = 3.4064
Iteration: 10 ; Train RMSE = 1.7309 ; Test RMSE = 1.7701
Iteration: 15 ; Train RMSE = 1.3885 ; Test RMSE = 1.4361
Iteration: 20 ; Train RMSE = 1.2352 ; Test RMSE = 1.2888
Iteration: 25 ; Train RMSE = 1.1464 ; Test RMSE = 1.2051
Iteration: 30 ; Train RMSE = 1.0873 ; Test RMSE = 1.1505
Iteration: 35 ; Train RMSE = 1.0439 ; Test RMSE = 1.1117
Iteration: 40 ; Train RMSE = 1.0097 ; Test RMSE = 1.0825
Iteration: 45 ; Train RMSE = 0.9813 ; Test RMSE = 1.0594
Iteration: 50 ; Train RMSE = 0.9571 ; Test RMSE = 1.0408
```

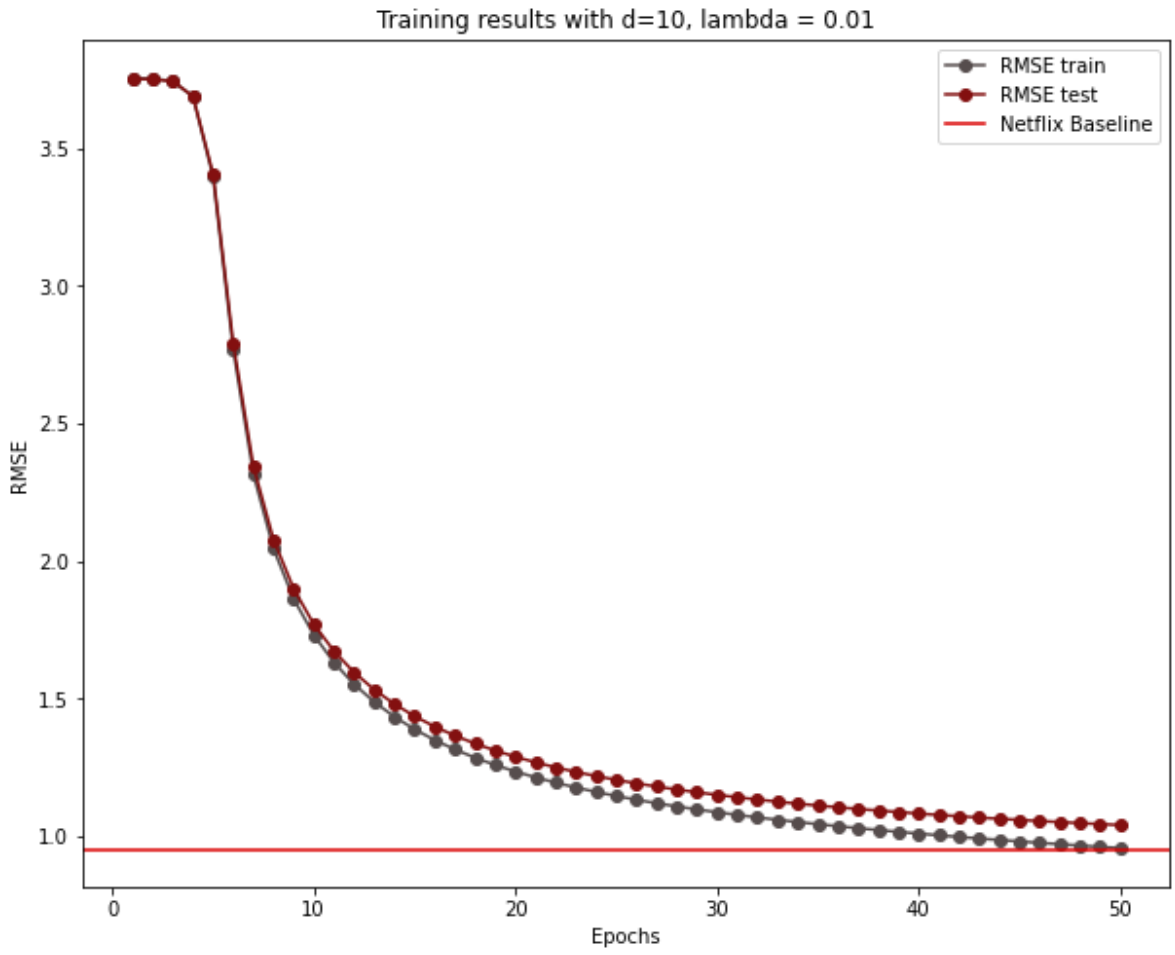
In [21]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result01)[:,0]
train_rmse_list = np.array(result01)[:,1]
test_rmse_list = np.array(result01)[:,2]

plt.title('Training results with d=10, lambda = 0.01')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=10, lambda = 0.05

In [22]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.05, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)
result05 = mf.test()
```

```
Iteration: 5 ; Train RMSE = 3.5194 ; Test RMSE = 3.5258
Iteration: 10 ; Train RMSE = 1.7689 ; Test RMSE = 1.8063
Iteration: 15 ; Train RMSE = 1.4027 ; Test RMSE = 1.4491
Iteration: 20 ; Train RMSE = 1.2446 ; Test RMSE = 1.2964
Iteration: 25 ; Train RMSE = 1.1552 ; Test RMSE = 1.2110
Iteration: 30 ; Train RMSE = 1.0979 ; Test RMSE = 1.1568
Iteration: 35 ; Train RMSE = 1.0572 ; Test RMSE = 1.1185
Iteration: 40 ; Train RMSE = 1.0267 ; Test RMSE = 1.0903
Iteration: 45 ; Train RMSE = 1.0023 ; Test RMSE = 1.0680
Iteration: 50 ; Train RMSE = 0.9821 ; Test RMSE = 1.0499
```

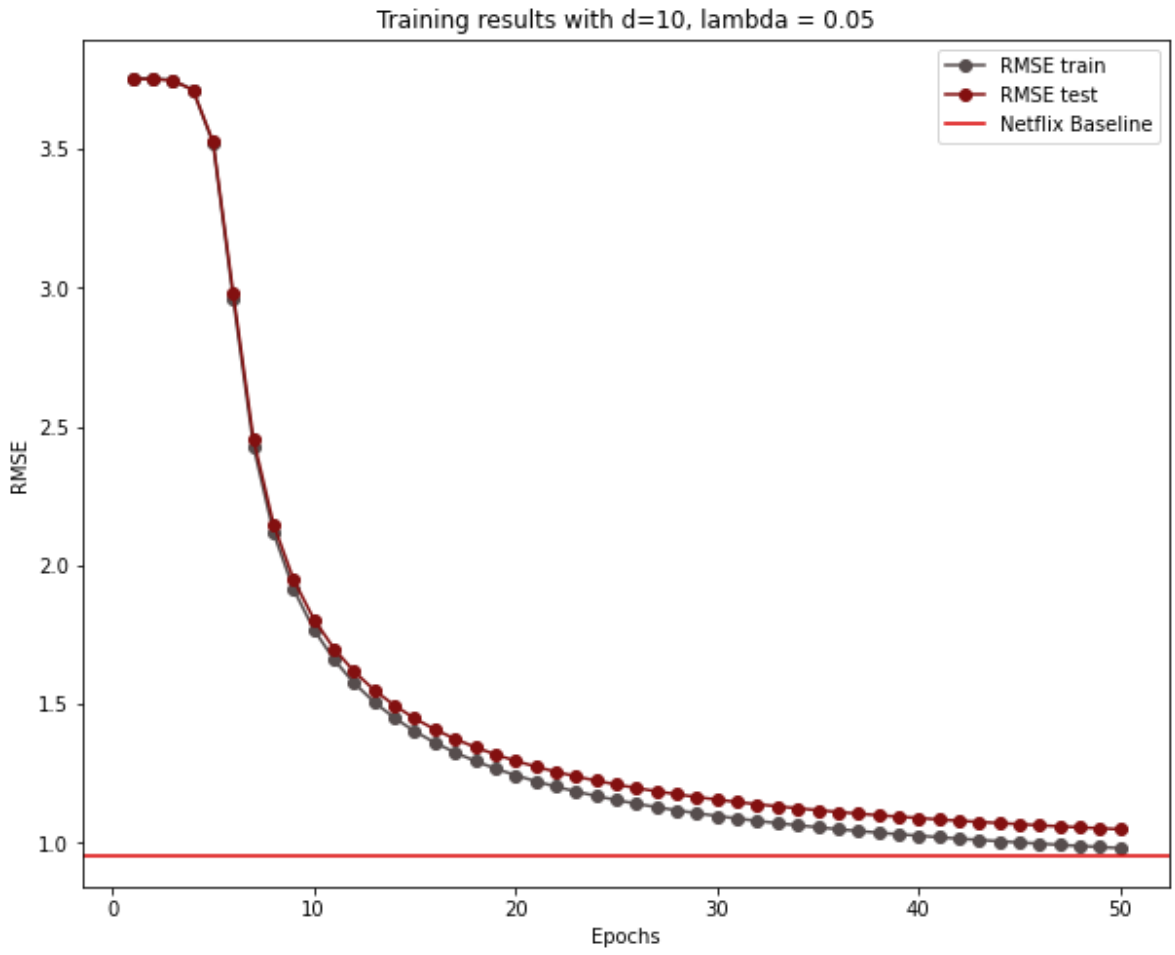
In [23]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result05)[:,0]
train_rmse_list = np.array(result05)[:,1]
test_rmse_list = np.array(result05)[:,2]

plt.title('Training results with d=10, lambda = 0.05')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=10, lambda = 0.1

In [24]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.1, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)
result_1 = mf.test()
```

```
Iteration: 5 ; Train RMSE = 3.5084 ; Test RMSE = 3.5141
Iteration: 10 ; Train RMSE = 1.7802 ; Test RMSE = 1.8167
Iteration: 15 ; Train RMSE = 1.4128 ; Test RMSE = 1.4583
Iteration: 20 ; Train RMSE = 1.2539 ; Test RMSE = 1.3046
Iteration: 25 ; Train RMSE = 1.1647 ; Test RMSE = 1.2190
Iteration: 30 ; Train RMSE = 1.1069 ; Test RMSE = 1.1639
Iteration: 35 ; Train RMSE = 1.0669 ; Test RMSE = 1.1256
Iteration: 40 ; Train RMSE = 1.0369 ; Test RMSE = 1.0971
Iteration: 45 ; Train RMSE = 1.0144 ; Test RMSE = 1.0757
Iteration: 50 ; Train RMSE = 0.9958 ; Test RMSE = 1.0579
```

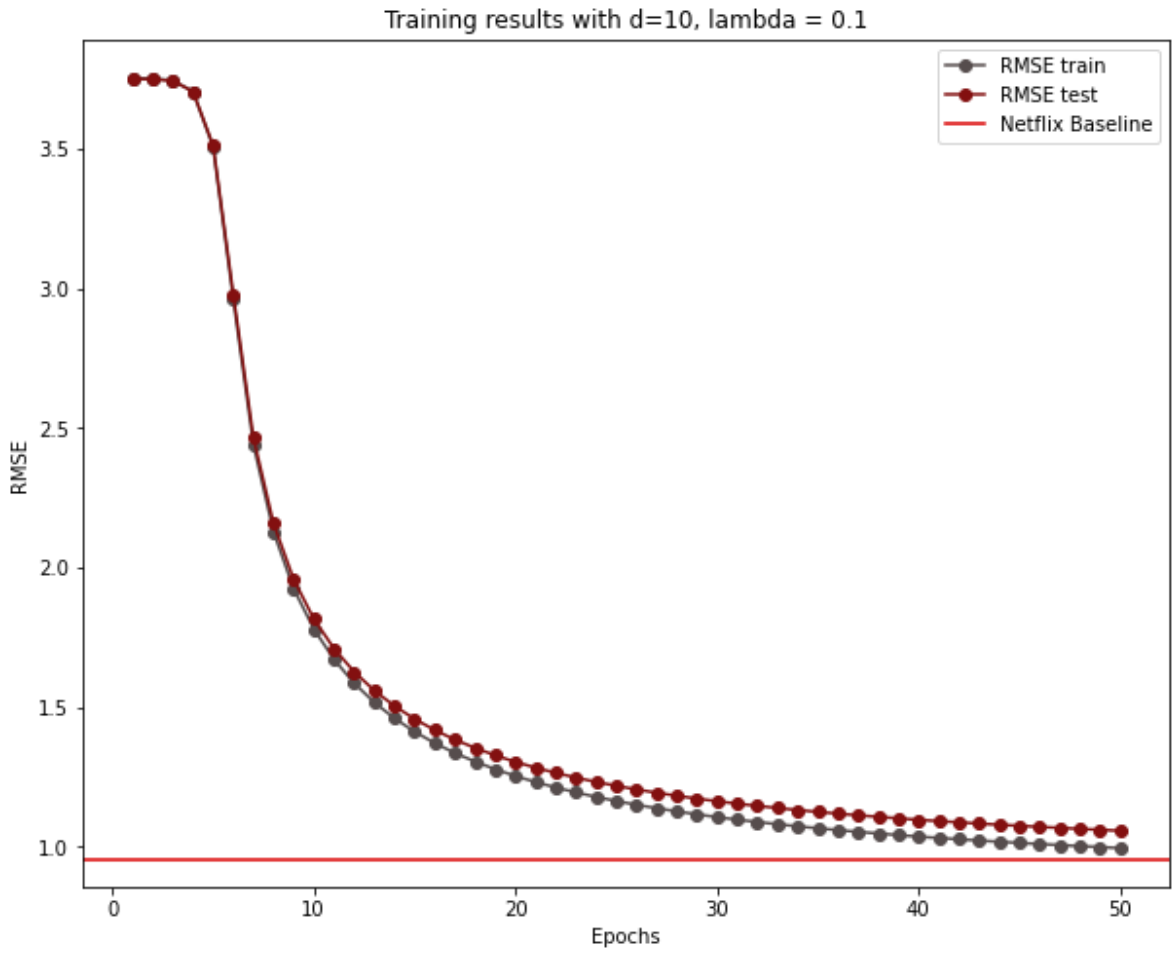
In [25]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result_1)[:,0]
train_rmse_list = np.array(result_1)[:,1]
test_rmse_list = np.array(result_1)[:,2]

plt.title('Training results with d=10, lambda = 0.1')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



Matrix Factorization using sgd optimization

In [16]:

```
import numpy as np
import pandas as pd
```

In [17]:

```
ratings = pd.read_csv('netflix_data.csv')
ratings = ratings[['user_id', 'movie_id', 'rating']].astype(int)            # timestamp 제거
```

In [18]:

```
ratings
```

Out[18]:

	user_id	movie_id	rating
0	2442	1	3
1	1719610	1	2
2	1011918	1	4
3	479924	1	5
4	2389367	1	1
...	...	...	...
507847	295393	4499	5
507848	305344	4499	1
507849	1627987	4499	1
507850	1988633	4499	4
507851	1796454	4499	1

507852 rows × 3 columns

In [19]:

```
# train test 분리
from sklearn.utils import shuffle
TRAIN_SIZE = 0.75
ratings = shuffle(ratings, random_state=1)
cutoff = int(TRAIN_SIZE * len(ratings))
ratings_train = ratings.iloc[:cutoff]
ratings_test = ratings.iloc[cutoff:]
```

Algorithm

In [20]:

```
# New MF class for training & testing
class NEW_MF():
    def __init__(self, ratings, K, alpha, beta, iterations, verbose=True):
        self.R = np.array(ratings)
##### >>>>> (2) user_id, item_id를 R의 index와 매핑하기 위한 dictionary 생성
        item_id_index = []
        index_item_id = []
        for i, one_id in enumerate(ratings):
            item_id_index.append([one_id, i])
            index_item_id.append([i, one_id])
        self.item_id_index = dict(item_id_index)
        self.index_item_id = dict(index_item_id)
        user_id_index = []
        index_user_id = []
        for i, one_id in enumerate(ratings.T):
            user_id_index.append([one_id, i])
            index_user_id.append([i, one_id])
        self.user_id_index = dict(user_id_index)
        self.index_user_id = dict(index_user_id)
#### <<<<< (2)
        self.num_users, self.num_items = np.shape(self.R)
        self.K = K
        self.alpha = alpha
        self.beta = beta
        self.iterations = iterations
        self.verbose = verbose

    # train set의 RMSE 계산
    def rmse(self):
        xs, ys = self.R.nonzero()
        self.predictions = []
        self.errors = []
        for x, y in zip(xs, ys):
            prediction = self.get_prediction(x, y)
            self.predictions.append(prediction)
            self.errors.append(self.R[x, y] - prediction)
        self.predictions = np.array(self.predictions)
        self.errors = np.array(self.errors)
        return np.sqrt(np.mean(self.errors**2))

    # Ratings for user i and item j
    def get_prediction(self, i, j):
        prediction = self.b + self.b_u[i] + self.b_d[j] + self.P[i, :].dot(self.Q[j, :].T)
        return prediction

    # Stochastic gradient descent to get optimized P and Q matrix
    def sgd(self):
        for i, j, r in self.samples:
            prediction = self.get_prediction(i, j)
            e = (r - prediction)

            self.b_u[i] += self.alpha * (e - self.beta * self.b_u[i])
            self.b_d[j] += self.alpha * (e - self.beta * self.b_d[j])

            self.P[i, :] += self.alpha * (e * self.Q[j, :] - self.beta * self.P[i,:])
            self.Q[j, :] += self.alpha * (e * self.P[i, :] - self.beta * self.Q[j,:])

##### >>>>> (3)
    # Test set을 선정
```

```
    def set_test(self, ratings_test):
        test_set = []
        for i in range(len(ratings_test)):      # test 데이터에 있는 각 데이터에 대해서
            x = self.user_id_index[ratings_test.iloc[i, 0]]
            y = self.item_id_index[ratings_test.iloc[i, 1]]
            z = ratings_test.iloc[i, 2]
            test_set.append([x, y, z])
            self.R[x, y] = 0                    # Setting test set ratings to 0
        self.test_set = test_set
        return test_set                         # Return test set

    # Test set의 RMSE 계산
    def test_rmse(self):
        error = 0
        for one_set in self.test_set:
            predicted = self.get_prediction(one_set[0], one_set[1])
            error += pow(one_set[2] - predicted, 2)
        return np.sqrt(error/len(self.test_set))

    # Training 하면서 test set의 정확도를 계산
    def test(self):
        # Initializing user-feature and item-feature matrix
        self.P = np.random.normal(scale=1./self.K, size=(self.num_users, self.K))
        self.Q = np.random.normal(scale=1./self.K, size=(self.num_items, self.K))

        # Initializing the bias terms
        self.b_u = np.zeros(self.num_users)
        self.b_d = np.zeros(self.num_items)
        self.b = np.mean(self.R[self.R.nonzero()])

        # List of training samples
        rows, columns = self.R.nonzero()
        self.samples = [(i, j, self.R[i,j]) for i, j in zip(rows, columns)]

        # Stochastic gradient descent for given number of iterations
        training_process = []
        for i in range(self.iterations):
            np.random.shuffle(self.samples)
            self.sgd()
            rmse1 = self.rmse()
            rmse2 = self.test_rmse()
            training_process.append((i+1, rmse1, rmse2))
            if self.verbose:
                if (i+1) % 5 == 0:
                    print("Iteration: %d ; Train RMSE = %.4f ; Test RMSE = %.4f" % (i+1, rmse1, rmse
        return training_process

    # Ratings for given user_id and item_id
    def get_one_prediction(self, user_id, item_id):
        return self.get_prediction(self.user_id_index[user_id], self.item_id_index[item_id])

    # Full user-movie rating matrix
    def full_prediction(self):
        return self.b + self.b_u[:,np.newaxis] + self.b_d[np.newaxis,:] + self.P.dot(self.Q.T)
```

d=5, lambda = 0.02

In [21]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf5 = NEW_MF(R_temp, K=5, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf5.set_test(ratings_test)

import time
start = time.time()
result5 = mf5.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 0.9725 ; Test RMSE = 0.9823
Iteration: 10 ; Train RMSE = 0.9486 ; Test RMSE = 0.9622
Iteration: 15 ; Train RMSE = 0.9365 ; Test RMSE = 0.9531
Iteration: 20 ; Train RMSE = 0.9289 ; Test RMSE = 0.9481
Iteration: 25 ; Train RMSE = 0.9234 ; Test RMSE = 0.9450
Iteration: 30 ; Train RMSE = 0.9192 ; Test RMSE = 0.9429
Iteration: 35 ; Train RMSE = 0.9157 ; Test RMSE = 0.9414
Iteration: 40 ; Train RMSE = 0.9126 ; Test RMSE = 0.9402
Iteration: 45 ; Train RMSE = 0.9096 ; Test RMSE = 0.9392
Iteration: 50 ; Train RMSE = 0.9066 ; Test RMSE = 0.9383
1667.2828891277313
```

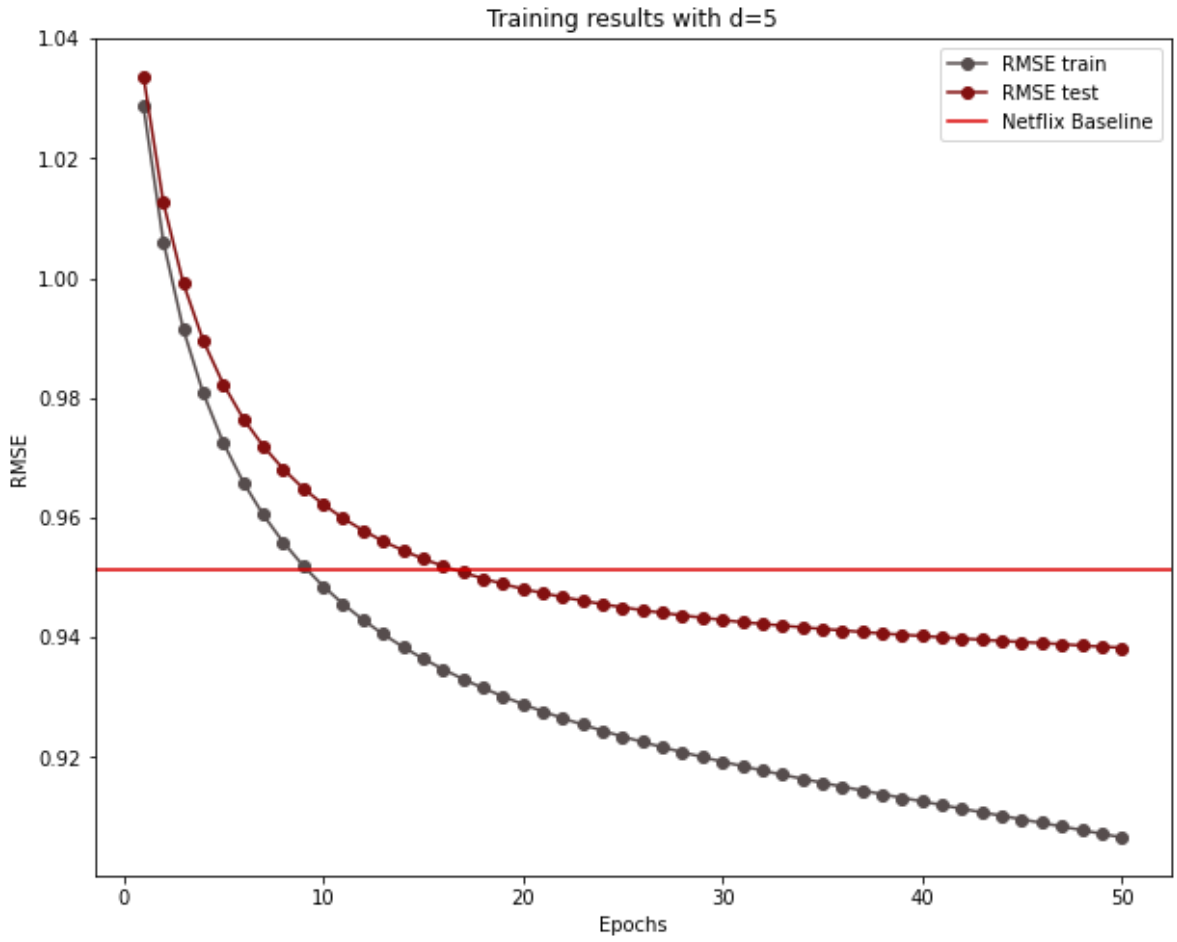
In [22]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result5)[:,0]
train_rmse_list = np.array(result5)[:,1]
test_rmse_list = np.array(result5)[:,2]

plt.title('Training results with d=5')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=10, lambda = 0.02

In [23]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf10 = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf10.set_test(ratings_test)

import time
start = time.time()
result10 = mf10.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 0.9710 ; Test RMSE = 0.9800
Iteration: 10 ; Train RMSE = 0.9480 ; Test RMSE = 0.9604
Iteration: 15 ; Train RMSE = 0.9363 ; Test RMSE = 0.9516
Iteration: 20 ; Train RMSE = 0.9290 ; Test RMSE = 0.9467
Iteration: 25 ; Train RMSE = 0.9237 ; Test RMSE = 0.9436
Iteration: 30 ; Train RMSE = 0.9195 ; Test RMSE = 0.9415
Iteration: 35 ; Train RMSE = 0.9159 ; Test RMSE = 0.9400
Iteration: 40 ; Train RMSE = 0.9123 ; Test RMSE = 0.9385
Iteration: 45 ; Train RMSE = 0.9087 ; Test RMSE = 0.9371
Iteration: 50 ; Train RMSE = 0.9045 ; Test RMSE = 0.9356
1551.2333669662476
```

BPMF

netflix data 구현

In [11]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt

from numpy.random import multivariate_normal
from scipy.stats import wishart
```

데이터 전처리

netflix data 불러오기

In [12]:
```
ratings = pd.read_csv('./data/netflix_data_small.csv')
ratings.head(10)
```

Out[12]:

	user_id	movie_id	rating
0	2442	1	3.0
1	1719610	1	2.0
2	1011918	1	4.0
3	479924	1	5.0
4	2389367	1	1.0
5	563962	1	5.0
6	369646	1	5.0
7	1430587	1	4.0
8	305344	1	1.0
9	389872	1	2.0

In [14]:
```
ratings['rating'].mean()
```

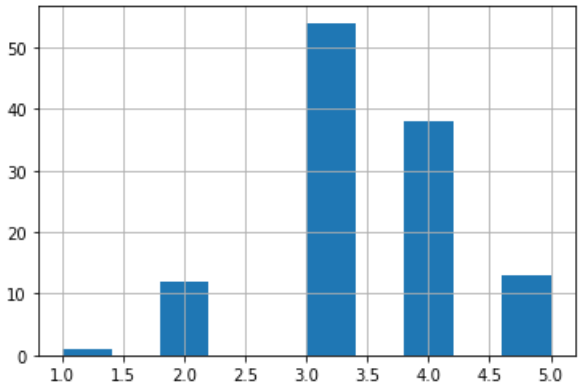
Out[14]:
```
3.591302190401928
```

In [15]:
```
ratings['rating'].std()
```

Out[15]:
```
1.0952602247926584
```

In [16]:
```
ratings.loc[ratings['user_id']==2442,'rating'].hist()
```

Out[16]:
```
<AxesSubplot:>
```



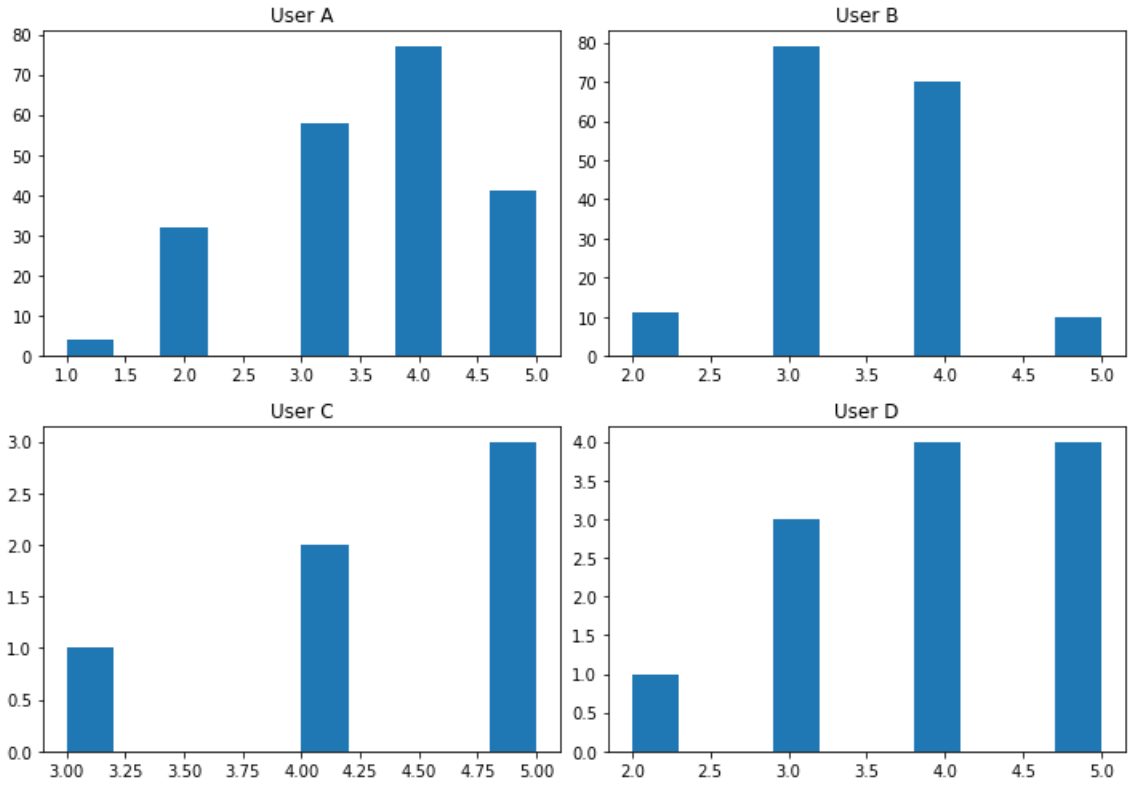
In [17]:
```
np.random.seed(123)
random_user_list = np.random.choice(ratings['user_id'].unique(),4)
fig, axs = plt.subplots(2,2, figsize=(10,7))
axs[0,0].hist(ratings.loc[ratings['user_id']==random_user_list[0],'rating'])
axs[0,0].set_title("User A")

axs[0,1].hist(ratings.loc[ratings['user_id']==random_user_list[1],'rating'])
axs[0,1].set_title("User B")

axs[1,0].hist(ratings.loc[ratings['user_id']==random_user_list[2],'rating'])
axs[1,0].set_title("User C")

axs[1,1].hist(ratings.loc[ratings['user_id']==random_user_list[3],'rating'])
axs[1,1].set_title("User D")

fig.tight_layout()
```



train, test 분리

In [18]:
```
# train test 분리
from sklearn.utils import shuffle
```

```
np.random.seed(123)
TRAIN_SIZE = 0.75
ratings = shuffle(ratings, random_state=1)
cutoff = int(TRAIN_SIZE * len(ratings))
ratings_train = ratings.iloc[:cutoff]
ratings_test = ratings.iloc[cutoff:]
```

In [19]:
```
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
R = np.array(R_temp)
R_test = np.zeros(R.shape)

item_id_index = []
index_item_id = []
for i, one_id in enumerate(R_temp):
    item_id_index.append([one_id, i])
    index_item_id.append([i, one_id])
item_id_index = dict(item_id_index)
index_item_id = dict(index_item_id)

user_id_index = []
index_user_id = []
for i, one_id in enumerate(R_temp.T):
    user_id_index.append([one_id, i])
    index_user_id.append([i, one_id])
user_id_index = dict(user_id_index)
index_user_id = dict(index_user_id)
```

In [20]:
```
test_set = []
for i in range(len(ratings_test)):      # test 데이터에 있는 각 데이터에 대해서
    x = user_id_index[ratings_test.iloc[i, 0]]
    y = item_id_index[ratings_test.iloc[i, 1]]
    z = ratings_test.iloc[i, 2]
    test_set.append([x, y, z])
    R[x, y] = 0                    # Setting test set ratings to 0
    R_test[x, y] = z
```

In [23]:
```
print(R.shape, R_test.shape)
```

```
(10000, 4497) (10000, 4497)
```

BPMF 구현

In [25]:
```
def BPMF(R, R_test, U_in, V_in, T, D, lowest_rating, highest_rating,
         mu_0=None, Beta_0=None, W_0=None, nu_0=None, seed=None):

    def Normal_Wishart(mu_0, lamb, W, nu, seed=None):
        """Function extracting a Normal_Wishart random variable"""
        Lambda = wishart(df=nu, scale=W, seed=seed).rvs()
        cov = np.linalg.inv(lamb * Lambda)
        mu = multivariate_normal(mu_0, cov)
        return mu, Lambda, cov

    def ranked(i, j):  # function telling if user i ranked movie j in the train
```

```
dataset.
        if R[i, j] != 0:
            return True
        else:
            return False

    def ranked_test(i, j):  # function telling if user i ranked movie j in the test
dataset.
        if R_test[i, j] != 0:
            return True
        else:
            return False

    N = R.shape[0] ; M = R.shape[1]
    U_old = np.array(U_in) ; V_old = np.array(V_in)
    R_pred_sum = np.zeros((N, M))
    train_rmse_list = []
    test_rmse_list = []
    train_epoch_list = []

    ### mcmc sample list
    ## predictions
    mcmc_samples_Rij = []
    ## hyperparams
    #mcmc_samples_mu_u = np.array([])
    #mcmc_samples_Lambda_U = np.array([])
    #mcmc_samples_mu_v = np.array([])
    #mcmc_samples_Lambda_V = np.array([])
    ## params
    mcmc_samples_U_new = []
    mcmc_samples_V_new = []

    # initialize now the hierarchical priors:
    alpha = 2  # observation noise, they put it = 2 in the paper
    mu_u = np.zeros((D, 1))
    mu_v = np.zeros((D, 1))
    Lambda_U = np.eye(D)
    Lambda_V = np.eye(D)

    # COUNT HOW MAY PAIRS ARE IN THE TEST AND TRAIN SET:
    pairs_train = (R != np.zeros((R.shape[0], R.shape[1]))).sum()
    pairs_test = (R_test != np.zeros((R_test.shape[0], R_test.shape[1]))).sum()

    # SET THE DEFAULT VALUES for Wishart distribution
    # we assume that parameters for both U and V are the same.

    if mu_0 is None:
        mu_0 = np.zeros(D)
    if nu_0 is None:
        nu_0 = D
    if Beta_0 is None:
        Beta_0 = 2
```

```
    if W_0 is None:
        W_0 = np.eye(D)

    ## Gibbs sampling
    np.random.seed(seed)
    for t in range(T):
        """ SAMPLE HYPERPARAMETERS """
        Beta_0_star = Beta_0 + N
        nu_0_star = nu_0 + N
        W_0_inv = np.linalg.inv(W_0)

        ### user hyperparameters : mu_U, Lambda_U
        U_average = np.sum(U_old, axis=1) / N
        dot_temp = np.zeros((D,D))
        for i in range(N):
            U_dev = (U_old[:,i] - U_average).reshape((D,1))
            dot_temp += np.dot(U_dev, np.transpose(U_dev))
        S_bar_U = dot_temp / N

        mu_0_star_U = (Beta_0 * mu_0 + N * U_average) / (Beta_0 + N)
        W_0_star_U_inv = W_0_inv + N * S_bar_U + Beta_0 * N / (Beta_0 + N) * np.dot(
            np.transpose(np.array(mu_0 - U_average, ndmin=2)),
            np.array((mu_0 - U_average), ndmin=2))
        W_0_star_U = np.linalg.inv(W_0_star_U_inv)

        # sample from Gaussian-Wishart distribution
        mu_U, Lambda_U, cov_U = Normal_Wishart(mu_0_star_U, Beta_0_star,
                                               W_0_star_U, nu_0_star, seed=seed)

        ### movie hyperparameters : mu_V, Lambda_V
        V_average = np.sum(V_old, axis=1) / M
        dot_temp = np.zeros((D,D))
        for j in range(M):
            V_dev = (V_old[:,j] - V_average).reshape((D,1))
            dot_temp += np.dot(V_dev, np.transpose(V_dev))
        S_bar_V = dot_temp / M

        mu_0_star_V = (Beta_0 * mu_0 + M * V_average) / (Beta_0 + M)
        W_0_star_V_inv = W_0_inv + M * S_bar_V + Beta_0 * M / (Beta_0 + M) * np.dot(
            np.transpose(np.array(mu_0 - V_average, ndmin=2)),
            np.array((mu_0 - V_average), ndmin=2))
        W_0_star_V = np.linalg.inv(W_0_star_V_inv)

        # sample from Gaussian-Wishart distribution
        mu_V, Lambda_V, cov_V = Normal_Wishart(mu_0_star_V, Beta_0_star,
                                               W_0_star_V, nu_0_star, seed=seed)


        """ SAMPLE PARAMETERS : U, V """
        U_new = np.zeros((D,N))
        V_new = np.zeros((D,M))
```

```
        ## sample user features
        for i in range(N):
            vvt_temp = np.zeros((D,D))
            vr_temp = np.zeros(D)
            for j in range(M):
                if ranked(i,j):
                    vvt_temp += np.dot(V_old[:,j].reshape((D,1)),
                                       np.transpose(V_old[:,j].reshape((D,1))))
                    vr_temp += V_old[:,j]*R[i,j]
            Lambda_i_star_U = Lambda_U + alpha * vvt_temp
            Lambda_i_star_U_inv = np.linalg.inv(Lambda_i_star_U)
            mu_i_star = np.dot( Lambda_i_star_U_inv, alpha*vr_temp + np.dot(Lambda_U,
mu_U))

            # sample new U
            new_U_i = multivariate_normal(mu_i_star, Lambda_i_star_U_inv) # Dx1
            U_new[:,i] = new_U_i

        ## sample movie features
        for j in range(M):
            uut_temp = np.zeros((D,D))
            ur_temp = np.zeros(D)
            for i in range(N):
                if ranked(i,j):
                    uut_temp += np.dot(U_new[:, i].reshape((D,1)),
                                       np.transpose(U_new[:, i].reshape((D,1))))
                    ur_temp += U_new[:, i]*R[i,j]
            Lambda_j_star_V = Lambda_V + alpha * uut_temp
            Lambda_j_star_V_inv = np.linalg.inv(Lambda_j_star_V)
            mu_j_star = np.dot( Lambda_j_star_V_inv, alpha*ur_temp + np.dot(Lambda_V,
mu_V) )

            # sample new V
            new_V_j = multivariate_normal(mu_j_star, Lambda_j_star_V_inv) # Dx1
            V_new[:,j] = new_V_j

        # update U, V
        U_old = np.array(U_new)
        V_old = np.array(V_new)

        mcmc_samples_U_new.append(U_new[1,5])
        mcmc_samples_V_new.append(V_new[3,5])

        R_step = np.dot(np.transpose(U_new), V_new)
        for i in range(N):  # reduce all the predictions to the correct ratings range.
            for j in range(M):
                if R_step[i, j] > highest_rating:
                    R_step[i, j] = highest_rating
                elif R_step[i, j] < lowest_rating:
                    R_step[i, j] = lowest_rating

        mcmc_samples_Rij.append(R_step[1,3])
```

```
        R_pred_sum += R_step
        k = t+1 # num iter
        R_pred = R_pred_sum / k

        # train RMSE
        train_err = 0
        for i in range(N):
            for j in range(M):
                if ranked(i, j):
                    train_err += (R_pred[i,j] - R[i,j])**2 # sum of squared error
        train_rmse = np.sqrt(train_err / pairs_train)
        train_rmse_list.append(train_rmse)
        print("\n Training RMSE at iteration ", k, " :    ", "{:.4}".format(train_rmse))

        # test RMSE
        test_err = 0
        for i in range(N):
            for j in range(M):
                if ranked_test(i, j):
                    test_err += (R_pred[i,j] - R_test[i,j])**2 # sum of squared error
        test_rmse = np.sqrt(test_err / pairs_test)
        test_rmse_list.append(test_rmse)
        print("Test RMSE at iteration ", k, " :    ", "{:.4}".format(test_rmse))

        train_epoch_list.append(k)

    return R_pred, train_rmse_list, test_rmse_list, train_epoch_list, mcmc_samples_Rij,
mcmc_samples_U_new, mcmc_samples_V_new
```

train

초기값 설정

In [26]:
```
'''# PMF 결과 추정된 U,V를 initial value로 사용
U_in = np.load("U_02.npy")
V_in = np.load("V_02.npy")'''
```

Out[26]:
```
'# PMF 결과 추정된 U,V를 initial value로 사용\nU_in = np.load("U_02.npy")\nV_in = np.load("V_02.npy")'
```

In [27]:
```
D=10
mu_u = np.zeros((D, 1)) ; mu_v = np.zeros((D, 1))
N = R.shape[0] ; M = R.shape[1]
U_in=np.zeros((D, N)) ; V_in=np.zeros((D, M))
np.random.seed(123)
for i in range(N):
    U_in[:,i] = np.random.normal(mu_u, size=(D,1)).reshape(D)

for j in range(M):
    V_in[:,j] = np.random.normal(mu_v, size=(D,1)).reshape(D)
```

In [28]:
```
import time
start = time.time()
```

```
result = BPMF(R, R_test, U_in, V_in, T=40, D=10,
     lowest_rating=1, highest_rating=5,
     mu_0=None, Beta_0=None, W_0=None, nu_0=None, seed=123)

print(time.time()-start)
```

```
 Training RMSE at iteration  1  :    2.212
Test RMSE at iteration  1  :    2.414

 Training RMSE at iteration  2  :    1.378
Test RMSE at iteration  2  :    1.564

 Training RMSE at iteration  3  :    1.097
Test RMSE at iteration  3  :    1.268

 Training RMSE at iteration  4  :    0.9687
Test RMSE at iteration  4  :    1.131

 Training RMSE at iteration  5  :    0.9004
Test RMSE at iteration  5  :    1.057

 Training RMSE at iteration  6  :    0.8605
Test RMSE at iteration  6  :    1.013

 Training RMSE at iteration  7  :    0.8357
Test RMSE at iteration  7  :    0.9835

 Training RMSE at iteration  8  :    0.8198
Test RMSE at iteration  8  :    0.9635

 Training RMSE at iteration  9  :    0.8094
Test RMSE at iteration  9  :    0.9491

 Training RMSE at iteration  10  :    0.8026
Test RMSE at iteration  10  :    0.9383

 Training RMSE at iteration  11  :    0.7982
Test RMSE at iteration  11  :    0.9303

 Training RMSE at iteration  12  :    0.7954
Test RMSE at iteration  12  :    0.924

 Training RMSE at iteration  13  :    0.7938
Test RMSE at iteration  13  :    0.919

 Training RMSE at iteration  14  :    0.7929
Test RMSE at iteration  14  :    0.9151

 Training RMSE at iteration  15  :    0.7925
Test RMSE at iteration  15  :    0.9118

 Training RMSE at iteration  16  :    0.7925
Test RMSE at iteration  16  :    0.9092

 Training RMSE at iteration  17  :    0.7927
Test RMSE at iteration  17  :    0.9071

 Training RMSE at iteration  18  :    0.7931
Test RMSE at iteration  18  :    0.9054

 Training RMSE at iteration  19  :    0.7935
Test RMSE at iteration  19  :    0.9039

 Training RMSE at iteration  20  :    0.7939
Test RMSE at iteration  20  :    0.9027

 Training RMSE at iteration  21  :    0.7944
```

```
Test RMSE at iteration  21  :    0.9017

 Training RMSE at iteration  22  :    0.7948
Test RMSE at iteration  22  :    0.9008

 Training RMSE at iteration  23  :    0.7952
Test RMSE at iteration  23  :    0.9

 Training RMSE at iteration  24  :    0.7956
Test RMSE at iteration  24  :    0.8993

 Training RMSE at iteration  25  :    0.7959
Test RMSE at iteration  25  :    0.8987

 Training RMSE at iteration  26  :    0.7962
Test RMSE at iteration  26  :    0.8982

 Training RMSE at iteration  27  :    0.7965
Test RMSE at iteration  27  :    0.8977

 Training RMSE at iteration  28  :    0.7967
Test RMSE at iteration  28  :    0.8972

 Training RMSE at iteration  29  :    0.7968
Test RMSE at iteration  29  :    0.8968

 Training RMSE at iteration  30  :    0.7969
Test RMSE at iteration  30  :    0.8965

 Training RMSE at iteration  31  :    0.7971
Test RMSE at iteration  31  :    0.8962

 Training RMSE at iteration  32  :    0.7971
Test RMSE at iteration  32  :    0.8959

 Training RMSE at iteration  33  :    0.7971
Test RMSE at iteration  33  :    0.8956

 Training RMSE at iteration  34  :    0.7972
Test RMSE at iteration  34  :    0.8954

 Training RMSE at iteration  35  :    0.7972
Test RMSE at iteration  35  :    0.8952

 Training RMSE at iteration  36  :    0.7972
Test RMSE at iteration  36  :    0.895

 Training RMSE at iteration  37  :    0.7971
Test RMSE at iteration  37  :    0.8947

 Training RMSE at iteration  38  :    0.797
Test RMSE at iteration  38  :    0.8946

 Training RMSE at iteration  39  :    0.7969
Test RMSE at iteration  39  :    0.8944

 Training RMSE at iteration  40  :    0.7968
Test RMSE at iteration  40  :    0.8942
3335.2690579891205
```

In [29]:

```
R_pred, train_rmse_list, test_rmse_list, train_epoch_list, mcmc_samples_Rij,
mcmc_samples_U_new, mcmc_samples_V_new = result
```

결과 시각화

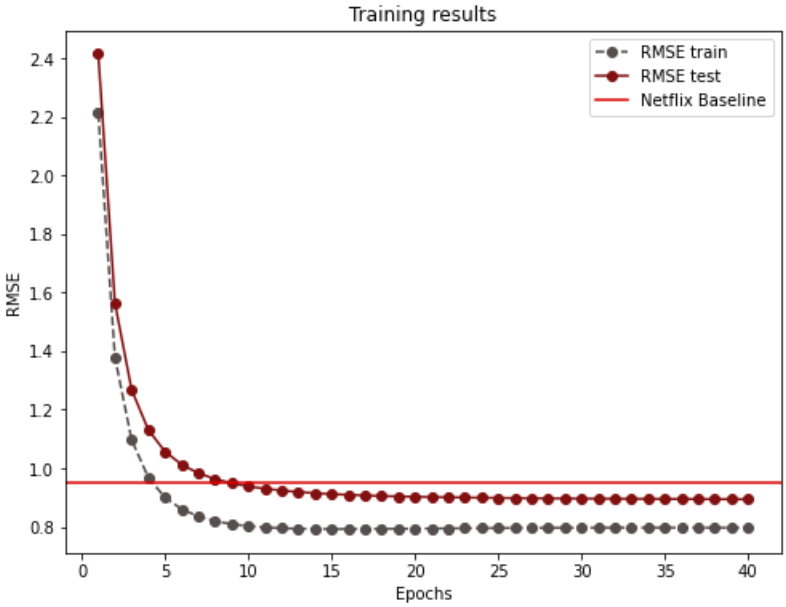
In [32]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
```

```
plt.title('Training results')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='--')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
```

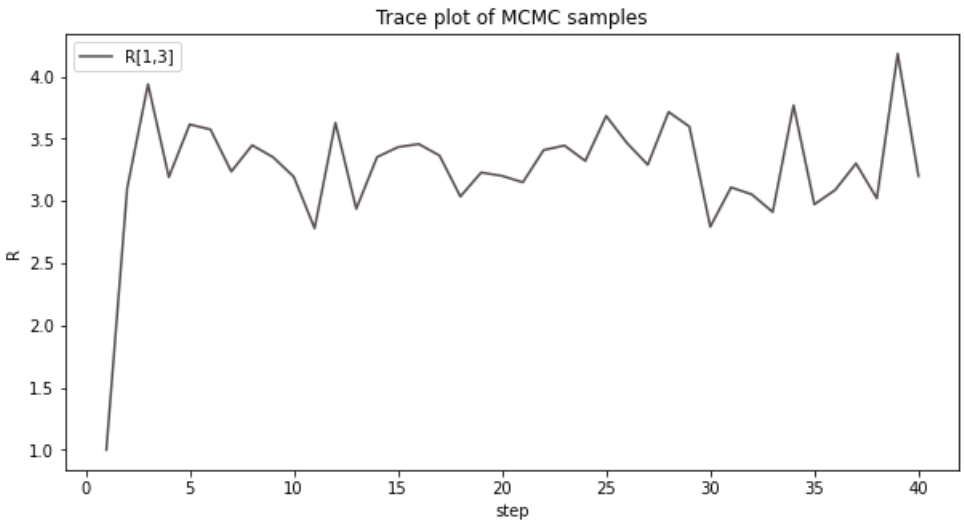
Out[32]: Text(0, 0.5, 'RMSE')



In [33]:

```
plt.figure(figsize=(10, 5))
plt.title('Trace plot of MCMC samples')
plt.plot(train_epoch_list, mcmc_samples_Rij, label='R[1,3]',
         color='#564d4d')
plt.legend()
plt.xlabel('step')
plt.ylabel('R')
```

Out[33]: Text(0, 0.5, 'R')



예측 결과 확인

In [35]:
```
df_movies = pd.read_csv('movie_titles.csv', encoding = "ISO-8859-1", header = None,
names = ['movie_id', 'year', 'title'])
df_movies.drop('year', axis=1, inplace=True)
df_movies.head()
```

Out[35]:

	movie_id	title
0	1	Dinosaur Planet
1	2	Isle of Man TT 2004 Review
2	3	Character
3	4	Paula Abdul's Get Up & Dance
4	5	The Rise and Fall of ECW

In [36]:
```
user_dict = dict(zip(ratings.user_id.unique(), range(ratings.user_id.nunique())))
#user_dict = dict(zip(range(ratings.user_id.nunique()),ratings.user_id.unique()))
#movie_dict = dict(zip(ratings.movie_id.unique(), range(ratings.movie_id.nunique())))
#ratings_dict = ratings.replace({'user_id':user_dict})
#ratings_dict = ratings_dict.replace({'movie_id':movie_dict})
```

user_id=2442 유저에 대한 예측값 확인

In [37]:
```
R_pred_2442 = pd.DataFrame(R_pred[user_dict[2442]],np.arange(1,M+1)).reset_index()
R_pred_2442.columns = ['movie_id','rating_pred']
R_pred_2442 = pd.merge(R_pred_2442, df_movies, how='left', on='movie_id')
R_pred_2442
```

Out[37]:

	movie_id	rating_pred	title
0	1	3.102412	Dinosaur Planet
1	2	2.887548	Isle of Man TT 2004 Review
2	3	3.511813	Character
3	4	3.014915	Paula Abdul's Get Up & Dance
4	5	3.537195	The Rise and Fall of ECW
...	...	...	...
4492	4493	3.629701	Ju-on: The Grudge
4493	4494	3.094832	Cartoon Crazys: Vol. 1
4494	4495	2.957297	Clifford: Happy Birthday Clifford / Puppy Love
4495	4496	3.181694	Farewell My Concubine
4496	4497	2.876507	Texasville

4497 rows × 3 columns

user 2442의 취향

In [38]:
```
R_result = pd.merge(R_pred_2442,
ratings[ratings['user_id']==2442].drop('user_id',axis=1), how='left', on='movie_id')
R_result = R_result[R_result['rating'].isna()==False]
R_result = R_result[['movie_id','title','rating','rating_pred']].reset_index(drop=True)

R_result.sort_values(by='rating', ascending=False).head(10)
```

Out[38]:

	movie_id	title	rating	rating_pred

	movie_id	title	rating	rating_pred
18	1084	Walking with Prehistoric Beasts	5.0	2.649816
73	2862	The Silence of the Lambs	5.0	3.078828
23	1391	Yanni: Live at the Acropolis	5.0	3.008299
78	3113	Dante's Peak	5.0	3.252302
79	3184	Desert Hearts	5.0	3.160749
26	1470	Bend It Like Beckham	5.0	2.993537
27	1482	Beyond Borders	5.0	3.272943
92	3671	Laughing Matters	5.0	3.303395
57	2452	Lord of the Rings: The Fellowship of the Ring	5.0	2.928999
31	1709	Clash of the Titans	5.0	3.504033

In [40]:
```
R_result.head(10)
```

Out[40]:

	movie_id	title	rating	rating_pred
0	1	Dinosaur Planet	3.0	3.102412
1	30	Something's Gotta Give	3.0	3.821087
2	188	Dead Birds	3.0	2.546090
3	191	X2: X-Men United	4.0	3.857685
4	283	If These Walls Could Talk	4.0	3.682491
5	457	Kill Bill: Vol. 2	3.0	3.300550
6	486	Journey to the Center of the Earth	4.0	3.470178
7	514	Santana: Supernatural Live	3.0	3.385765
8	528	The Hitchhiker's Guide to the Galaxy	4.0	2.127235
9	594	By Hook or By Crook	2.0	3.015638

user 2442가 좋아할만한 영화는?

In [39]:
```
R_result_recom = pd.merge(R_pred_2442,
ratings[ratings['user_id']==2442].drop('user_id',axis=1), how='left', on='movie_id')
R_result_recom = R_result_recom[R_result_recom['rating'].isna()==True]
R_result_recom =
R_result_recom[['movie_id','title','rating','rating_pred']].reset_index(drop=True)

R_result_recom.sort_values(by='rating_pred', ascending=False).head(10)
```

Out[39]:

	movie_id	title	rating	rating_pred
2858	2937	The Gate	NaN	4.728234
3364	3454	Kenny Chesney: Greatest Hits	NaN	4.687521
2998	3077	Turbulence	NaN	4.661542
2862	2941	Love Me Tonight	NaN	4.655401
1232	1255	Better Off Dead	NaN	4.641759
1465	1494	Appalachian Journey: Yo-Yo Ma/Edgar Meyer/Mark...	NaN	4.629214
2393	2451	Stealing Candy	NaN	4.601193
265	270	Sex and the City: Season 4	NaN	4.598906
3857	3960	Pollyanna	NaN	4.591619
4308	4425	Miranda (Tinto Brass)	NaN	4.586969

In [ ]:
```

```

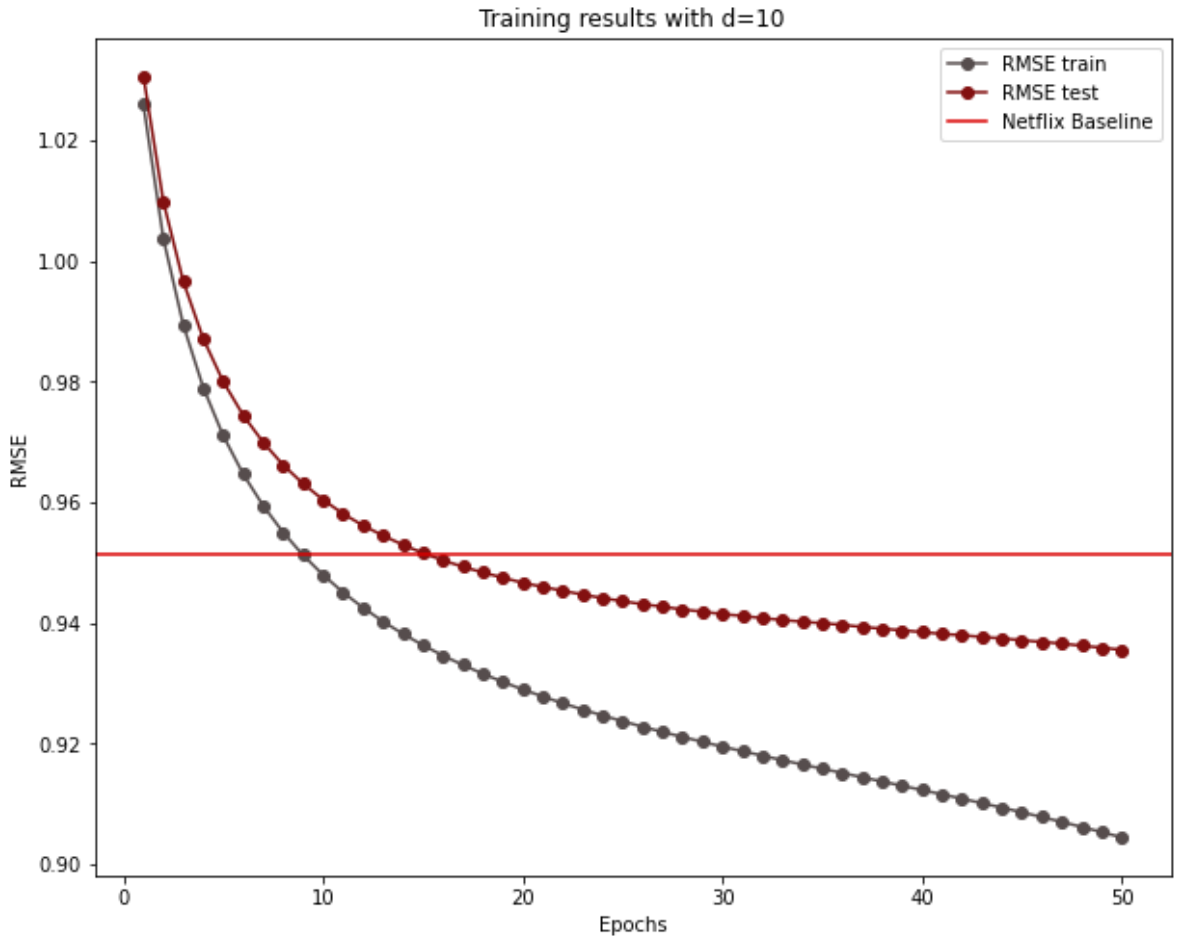
In [24]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result10)[:,0]
train_rmse_list = np.array(result10)[:,1]
test_rmse_list = np.array(result10)[:,2]

plt.title('Training results with d=10')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



In [63]:

```
pd.DataFrame(result10, columns = ['epoch','train','test']).to_csv("mf_b.csv", index=False)
```

d=20, lambda = 0.02

In [25]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf20 = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf20.set_test(ratings_test)

import time
start = time.time()
result20 = mf20.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 0.9712 ; Test RMSE = 0.9801
Iteration: 10 ; Train RMSE = 0.9481 ; Test RMSE = 0.9605
Iteration: 15 ; Train RMSE = 0.9365 ; Test RMSE = 0.9517
Iteration: 20 ; Train RMSE = 0.9292 ; Test RMSE = 0.9468
Iteration: 25 ; Train RMSE = 0.9240 ; Test RMSE = 0.9438
Iteration: 30 ; Train RMSE = 0.9199 ; Test RMSE = 0.9418
Iteration: 35 ; Train RMSE = 0.9163 ; Test RMSE = 0.9402
Iteration: 40 ; Train RMSE = 0.9131 ; Test RMSE = 0.9391
Iteration: 45 ; Train RMSE = 0.9098 ; Test RMSE = 0.9380
Iteration: 50 ; Train RMSE = 0.9062 ; Test RMSE = 0.9368
1534.759910106659
```

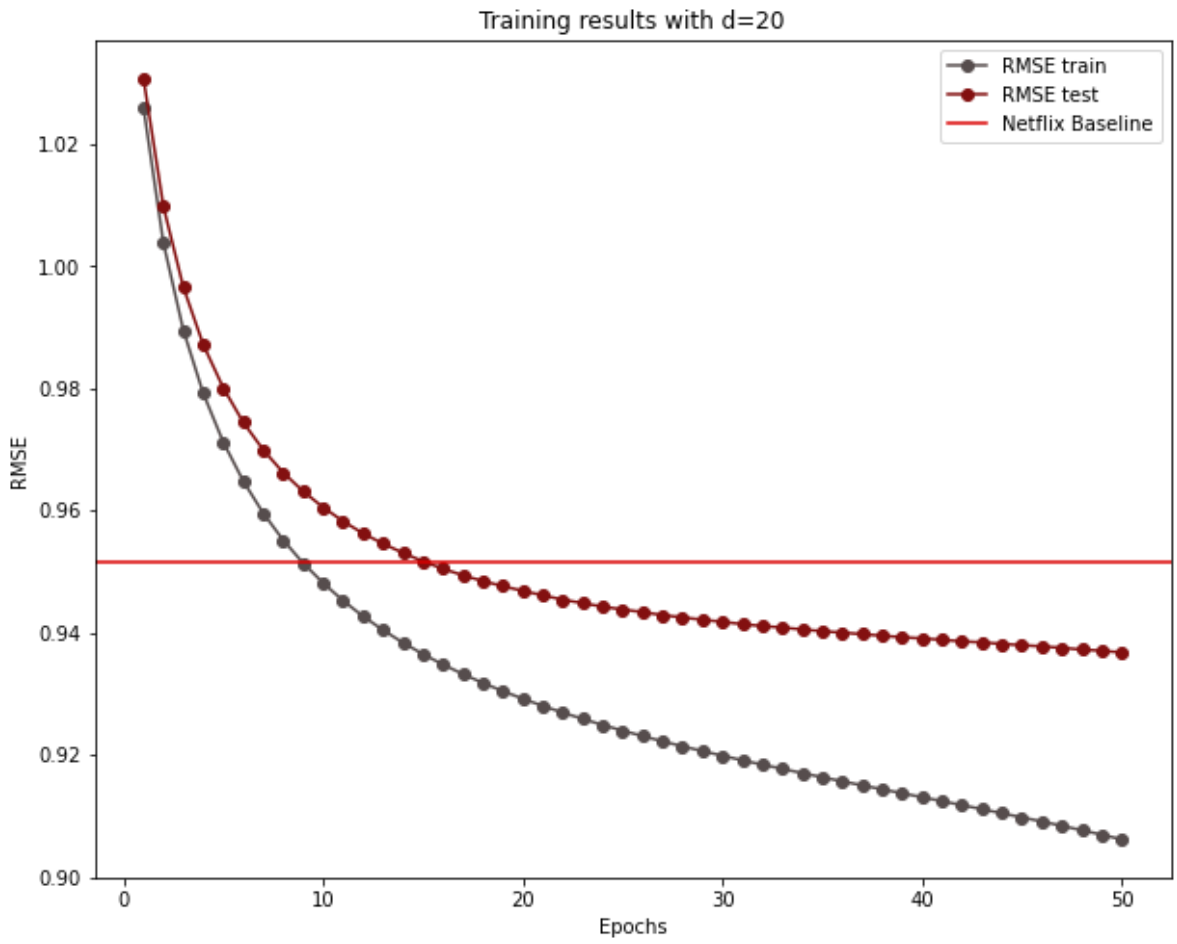
In [26]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result20)[:,0]
train_rmse_list = np.array(result20)[:,1]
test_rmse_list = np.array(result20)[:,2]

plt.title('Training results with d=20')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=30, lambda = 0.02

In [14]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf30 = NEW_MF(R_temp, K=30, alpha=0.001, beta=0.02, iterations=50, verbose=True)
test_set = mf30.set_test(ratings_test)

import time
start = time.time()
result30 = mf30.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 0.9713 ; Test RMSE = 0.9796
Iteration: 10 ; Train RMSE = 0.9487 ; Test RMSE = 0.9601
Iteration: 15 ; Train RMSE = 0.9374 ; Test RMSE = 0.9513
Iteration: 20 ; Train RMSE = 0.9305 ; Test RMSE = 0.9464
Iteration: 25 ; Train RMSE = 0.9257 ; Test RMSE = 0.9433
Iteration: 30 ; Train RMSE = 0.9220 ; Test RMSE = 0.9412
Iteration: 35 ; Train RMSE = 0.9189 ; Test RMSE = 0.9397
Iteration: 40 ; Train RMSE = 0.9161 ; Test RMSE = 0.9385
Iteration: 45 ; Train RMSE = 0.9133 ; Test RMSE = 0.9373
Iteration: 50 ; Train RMSE = 0.9102 ; Test RMSE = 0.9360
```

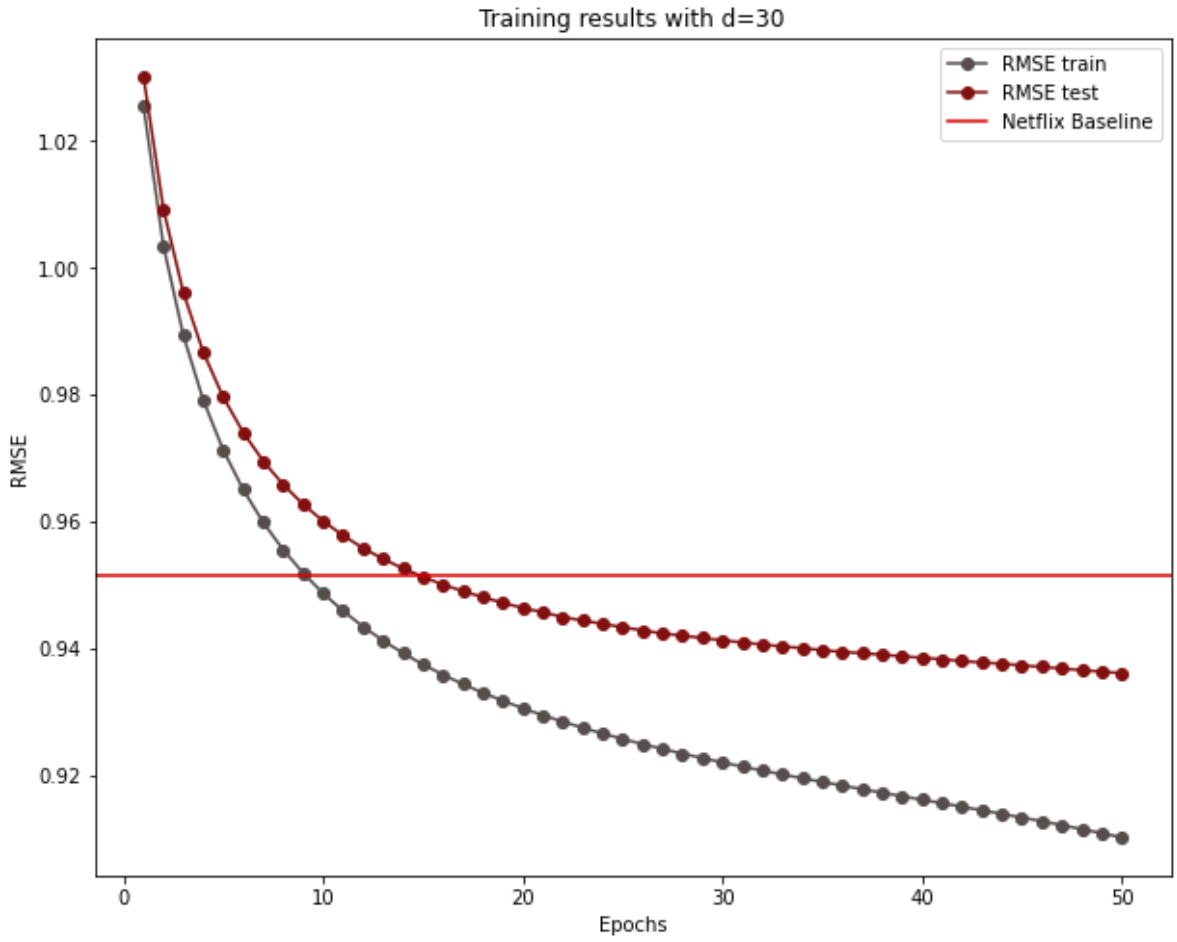
In [15]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result30)[:,0]
train_rmse_list = np.array(result30)[:,1]
test_rmse_list = np.array(result30)[:,2]

plt.title('Training results with d=30')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=10, lambda = 0.01

In [16]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.01, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)
result01 = mf.test()
```

```
Iteration: 5 ; Train RMSE = 0.9709 ; Test RMSE = 0.9798
Iteration: 10 ; Train RMSE = 0.9477 ; Test RMSE = 0.9602
Iteration: 15 ; Train RMSE = 0.9360 ; Test RMSE = 0.9515
Iteration: 20 ; Train RMSE = 0.9285 ; Test RMSE = 0.9466
Iteration: 25 ; Train RMSE = 0.9230 ; Test RMSE = 0.9435
Iteration: 30 ; Train RMSE = 0.9185 ; Test RMSE = 0.9414
Iteration: 35 ; Train RMSE = 0.9143 ; Test RMSE = 0.9398
Iteration: 40 ; Train RMSE = 0.9100 ; Test RMSE = 0.9381
Iteration: 45 ; Train RMSE = 0.9051 ; Test RMSE = 0.9364
Iteration: 50 ; Train RMSE = 0.8994 ; Test RMSE = 0.9343
```

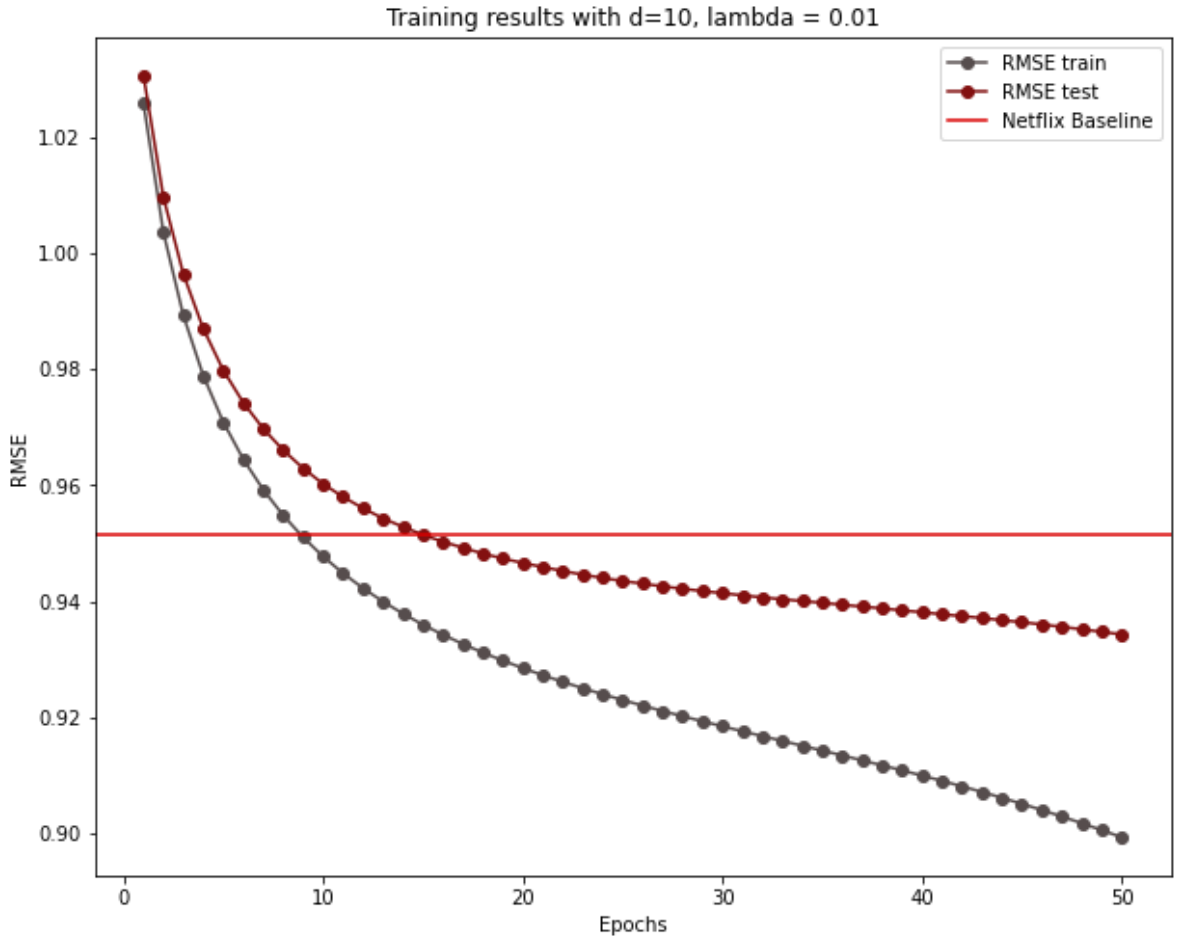
In [18]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result01)[:,0]
train_rmse_list = np.array(result01)[:,1]
test_rmse_list = np.array(result01)[:,2]

plt.title('Training results with d=10, lambda = 0.01')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=10, lambda = 0.05

In [20]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.05, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)
result05 = mf.test()
```

```
Iteration: 5 ; Train RMSE = 0.9716 ; Test RMSE = 0.9804
Iteration: 10 ; Train RMSE = 0.9489 ; Test RMSE = 0.9610
Iteration: 15 ; Train RMSE = 0.9375 ; Test RMSE = 0.9521
Iteration: 20 ; Train RMSE = 0.9305 ; Test RMSE = 0.9471
Iteration: 25 ; Train RMSE = 0.9256 ; Test RMSE = 0.9441
Iteration: 30 ; Train RMSE = 0.9220 ; Test RMSE = 0.9420
Iteration: 35 ; Train RMSE = 0.9190 ; Test RMSE = 0.9404
Iteration: 40 ; Train RMSE = 0.9166 ; Test RMSE = 0.9393
Iteration: 45 ; Train RMSE = 0.9143 ; Test RMSE = 0.9383
Iteration: 50 ; Train RMSE = 0.9123 ; Test RMSE = 0.9375
```

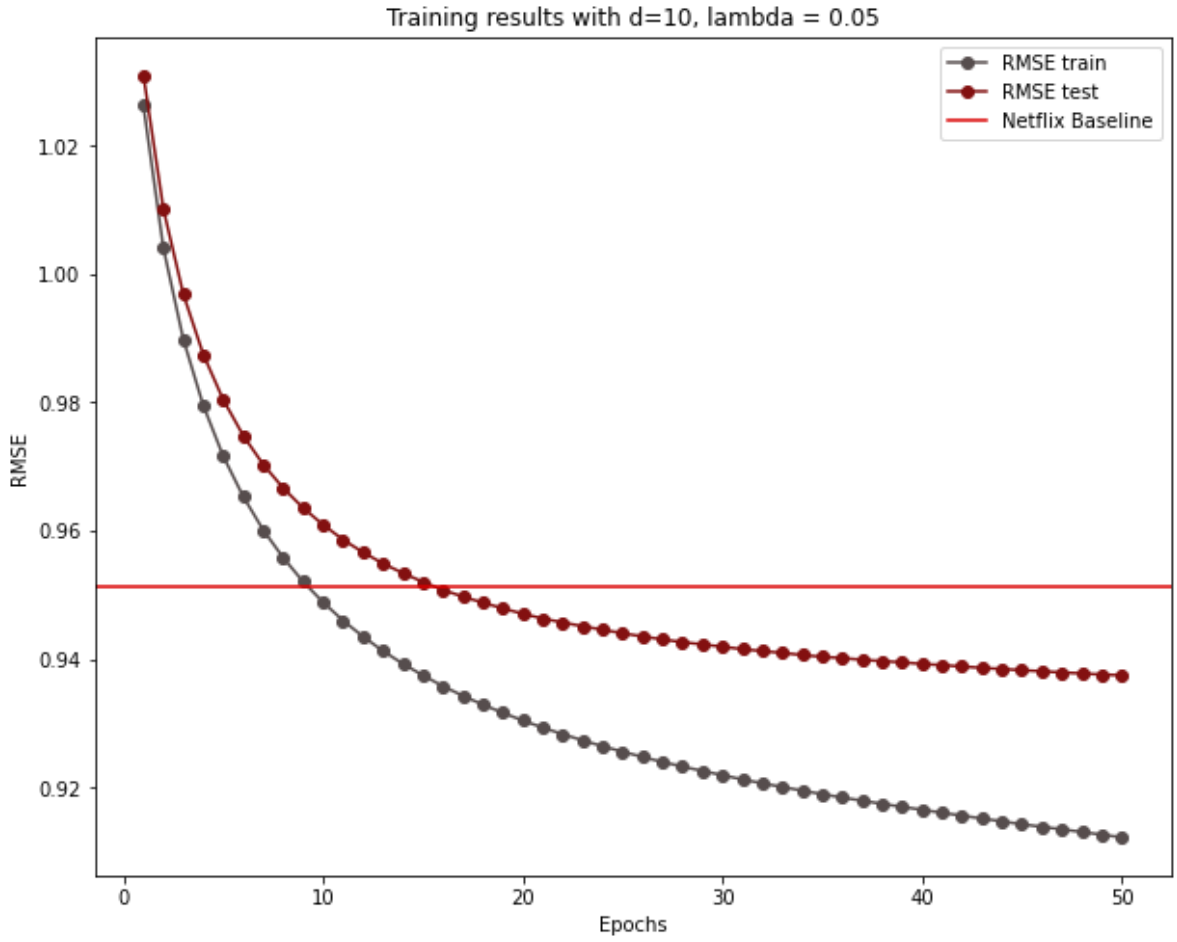
In [21]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result05)[:,0]
train_rmse_list = np.array(result05)[:,1]
test_rmse_list = np.array(result05)[:,2]

plt.title('Training results with d=10, lambda = 0.05')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=10, lambda = 0.1

In [22]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=10, alpha=0.001, beta=0.1, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)
result_1 = mf.test()
```

```
Iteration: 5 ; Train RMSE = 0.9727 ; Test RMSE = 0.9812
Iteration: 10 ; Train RMSE = 0.9504 ; Test RMSE = 0.9619
Iteration: 15 ; Train RMSE = 0.9392 ; Test RMSE = 0.9531
Iteration: 20 ; Train RMSE = 0.9325 ; Test RMSE = 0.9482
Iteration: 25 ; Train RMSE = 0.9279 ; Test RMSE = 0.9451
Iteration: 30 ; Train RMSE = 0.9245 ; Test RMSE = 0.9430
Iteration: 35 ; Train RMSE = 0.9218 ; Test RMSE = 0.9415
Iteration: 40 ; Train RMSE = 0.9198 ; Test RMSE = 0.9404
Iteration: 45 ; Train RMSE = 0.9180 ; Test RMSE = 0.9397
Iteration: 50 ; Train RMSE = 0.9165 ; Test RMSE = 0.9389
```

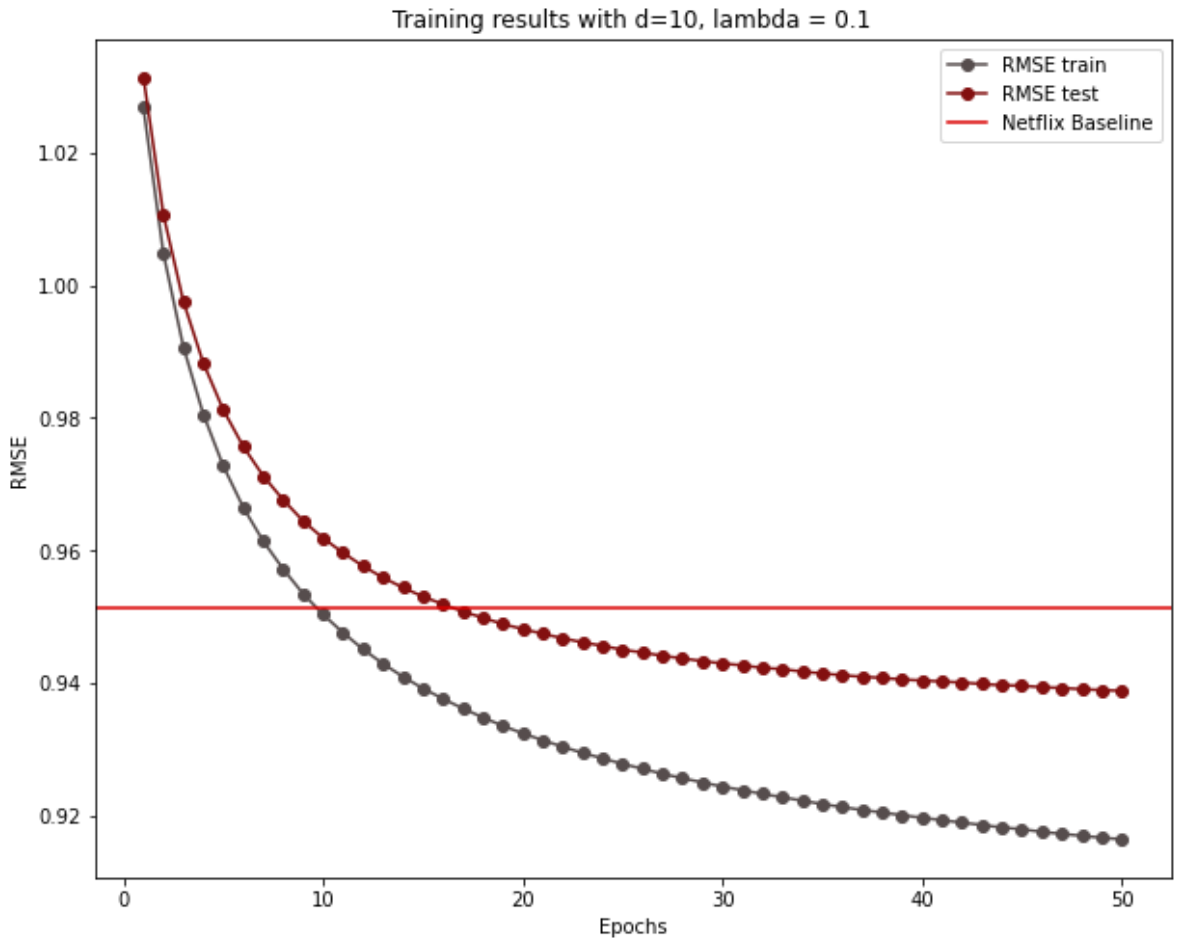
In [23]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result_1)[:,0]
train_rmse_list = np.array(result_1)[:,1]
test_rmse_list = np.array(result_1)[:,2]

plt.title('Training results with d=10, lambda = 0.1')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=20, lambda = 0.01

In [7]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=20, alpha=0.001, beta=0.01, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)

import time
start = time.time()
result20_01 = mf.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 0.9710 ; Test RMSE = 0.9795
Iteration: 10 ; Train RMSE = 0.9482 ; Test RMSE = 0.9600
Iteration: 15 ; Train RMSE = 0.9368 ; Test RMSE = 0.9511
Iteration: 20 ; Train RMSE = 0.9296 ; Test RMSE = 0.9462
Iteration: 25 ; Train RMSE = 0.9245 ; Test RMSE = 0.9432
Iteration: 30 ; Train RMSE = 0.9203 ; Test RMSE = 0.9411
Iteration: 35 ; Train RMSE = 0.9166 ; Test RMSE = 0.9395
Iteration: 40 ; Train RMSE = 0.9128 ; Test RMSE = 0.9380
Iteration: 45 ; Train RMSE = 0.9085 ; Test RMSE = 0.9364
Iteration: 50 ; Train RMSE = 0.9032 ; Test RMSE = 0.9345
354.4637825489044
```

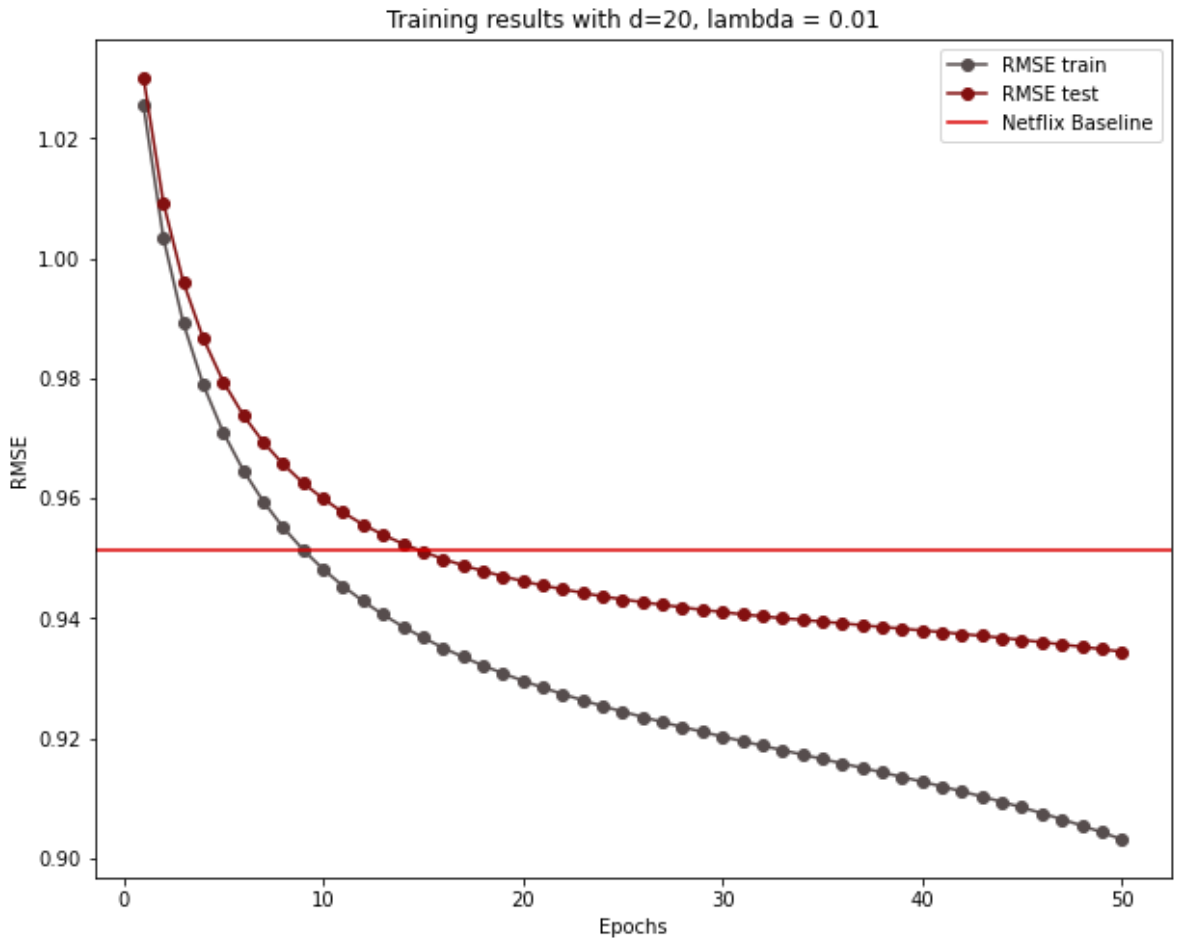
In [8]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result20_01)[:,0]
train_rmse_list = np.array(result20_01)[:,1]
test_rmse_list = np.array(result20_01)[:,2]

plt.title('Training results with d=20, lambda = 0.01')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



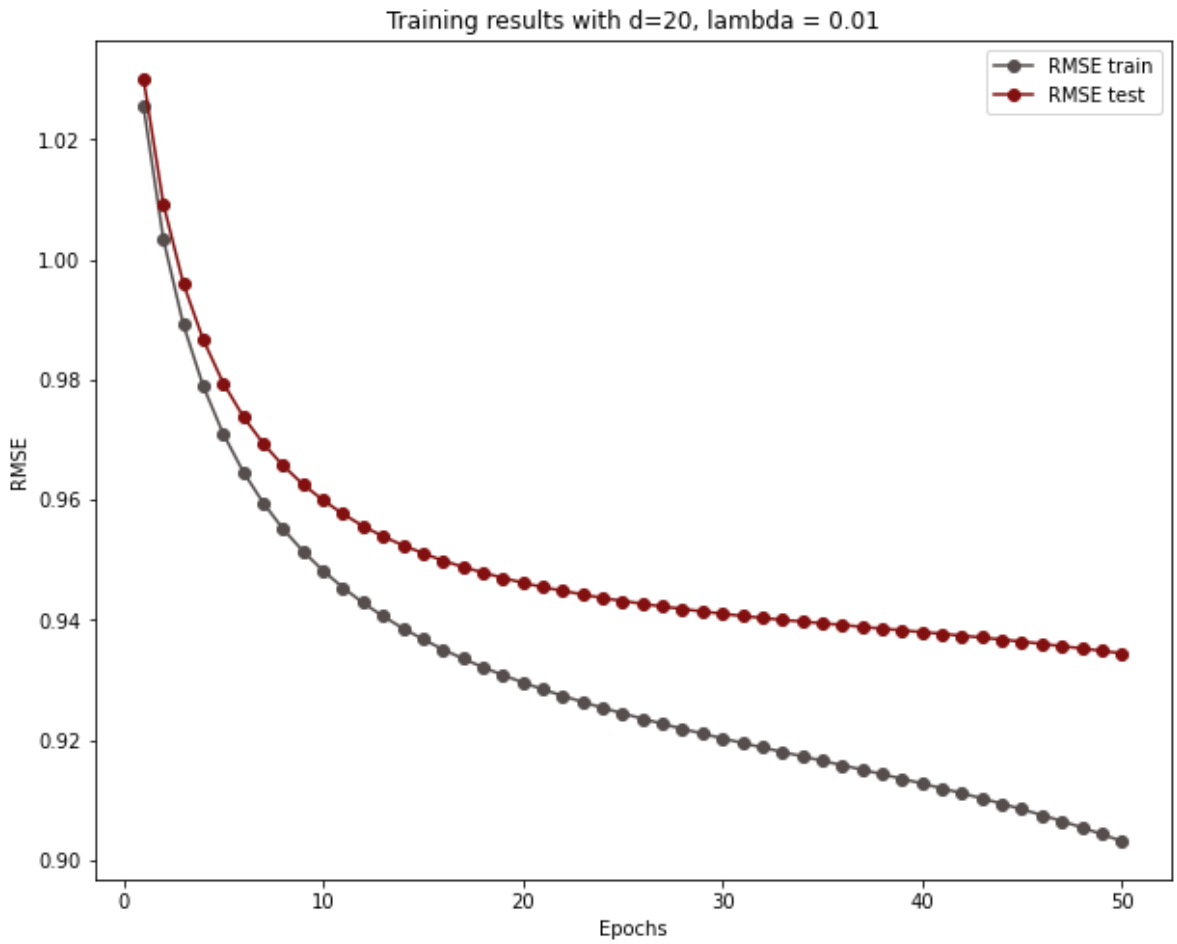
In [9]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result20_01)[:,0]
train_rmse_list = np.array(result20_01)[:,1]
test_rmse_list = np.array(result20_01)[:,2]

plt.title('Training results with d=20, lambda = 0.01')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
#plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=20, lambda = 0.05

In [10]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=20, alpha=0.001, beta=0.05, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)

import time
start = time.time()
result20_05 = mf.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 0.9717 ; Test RMSE = 0.9802
Iteration: 10 ; Train RMSE = 0.9492 ; Test RMSE = 0.9607
Iteration: 15 ; Train RMSE = 0.9381 ; Test RMSE = 0.9518
Iteration: 20 ; Train RMSE = 0.9312 ; Test RMSE = 0.9469
Iteration: 25 ; Train RMSE = 0.9265 ; Test RMSE = 0.9439
Iteration: 30 ; Train RMSE = 0.9230 ; Test RMSE = 0.9418
Iteration: 35 ; Train RMSE = 0.9202 ; Test RMSE = 0.9403
Iteration: 40 ; Train RMSE = 0.9179 ; Test RMSE = 0.9392
Iteration: 45 ; Train RMSE = 0.9159 ; Test RMSE = 0.9383
Iteration: 50 ; Train RMSE = 0.9140 ; Test RMSE = 0.9375
353.8110542297363
```

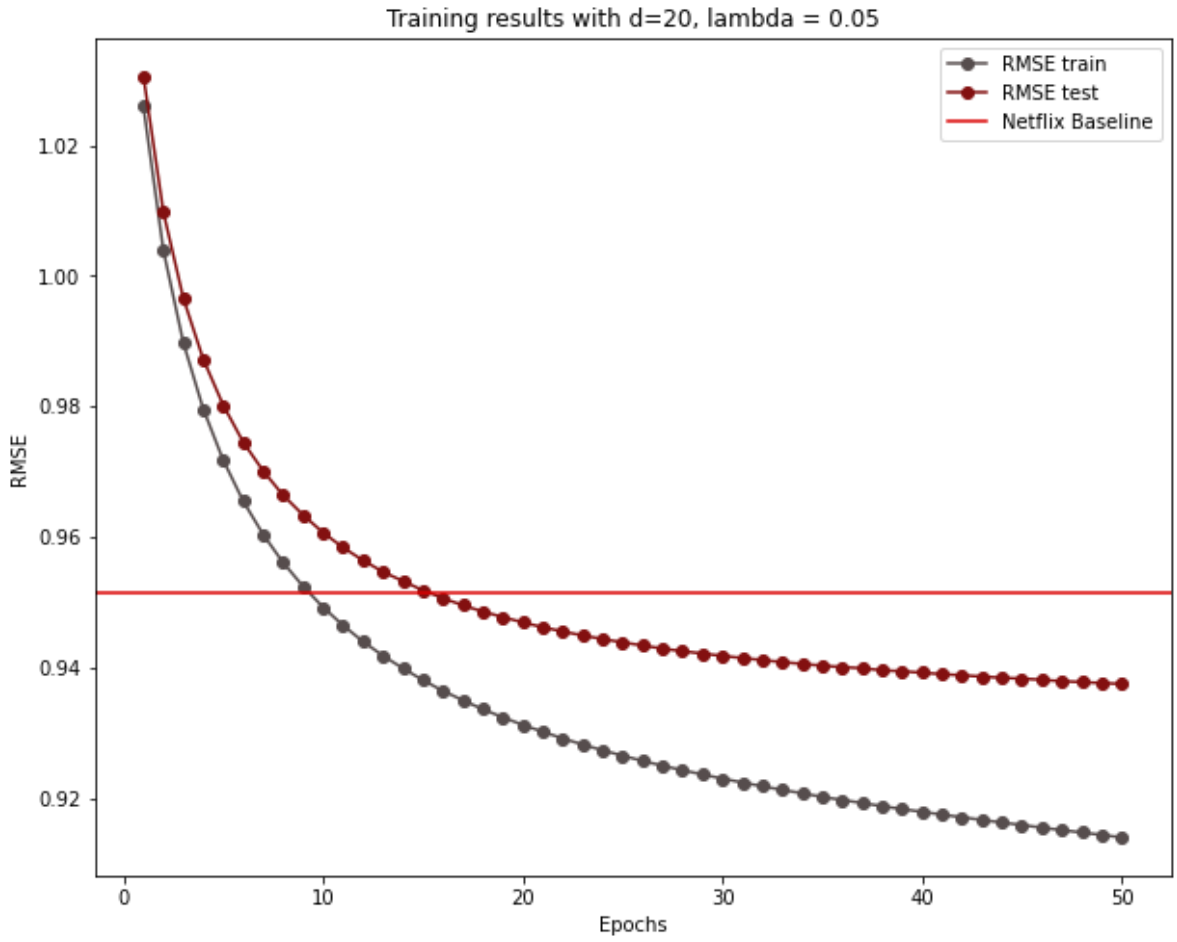
In [11]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result20_05)[:,0]
train_rmse_list = np.array(result20_05)[:,1]
test_rmse_list = np.array(result20_05)[:,2]

plt.title('Training results with d=20, lambda = 0.05')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



d=20, lambda = 0.1

In [13]:

```
# Testing MF RMSE
R_temp = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
mf = NEW_MF(R_temp, K=20, alpha=0.001, beta=0.1, iterations=50, verbose=True)
test_set = mf.set_test(ratings_test)

import time
start = time.time()
result20_1 = mf.test()
print(time.time()-start)
```

```
Iteration: 5 ; Train RMSE = 0.9729 ; Test RMSE = 0.9811
Iteration: 10 ; Train RMSE = 0.9507 ; Test RMSE = 0.9618
Iteration: 15 ; Train RMSE = 0.9398 ; Test RMSE = 0.9531
Iteration: 20 ; Train RMSE = 0.9331 ; Test RMSE = 0.9482
Iteration: 25 ; Train RMSE = 0.9285 ; Test RMSE = 0.9450
Iteration: 30 ; Train RMSE = 0.9252 ; Test RMSE = 0.9430
Iteration: 35 ; Train RMSE = 0.9226 ; Test RMSE = 0.9415
Iteration: 40 ; Train RMSE = 0.9206 ; Test RMSE = 0.9405
Iteration: 45 ; Train RMSE = 0.9189 ; Test RMSE = 0.9396
Iteration: 50 ; Train RMSE = 0.9175 ; Test RMSE = 0.9390
358.29302644729614
```

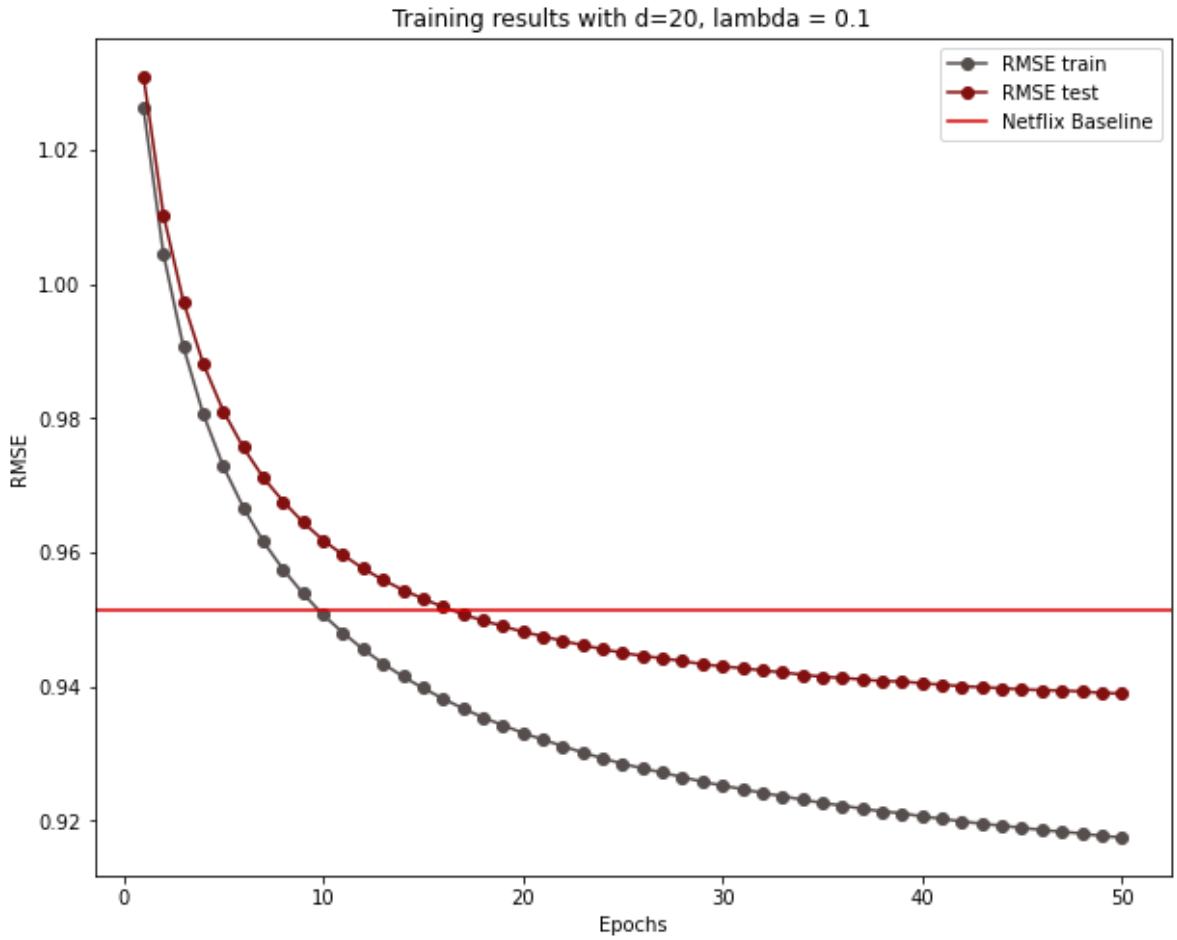
In [14]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,8))
train_epoch_list = np.array(result20_1)[:,0]
train_rmse_list = np.array(result20_1)[:,1]
test_rmse_list = np.array(result20_1)[:,2]

plt.title('Training results with d=20, lambda = 0.1')
plt.plot(train_epoch_list, train_rmse_list, label='RMSE train',
         color='#564d4d', marker='o', linestyle='-')
plt.plot(train_epoch_list, test_rmse_list, label='RMSE test',
         color='#831010', marker='o')
plt.axhline(0.9514, color='#db0000', label='Netflix Baseline')

plt.legend()
plt.xlabel('Epochs')
plt.ylabel('RMSE')
plt.show()
```



최종 모델 결과 확인

In [35]:

```
user_id = 2442
R_2442 = []
for i in mf10.item_id_index.keys():
    R_2442.append(mf10.get_one_prediction(2442, i))
```

In [38]:

```
df_movies = pd.read_csv('netflix_data/movie_titles.csv', encoding = "ISO-8859-1",
                        header = None, names = ['movie_id', 'year', 'title'])
```

In [40]:

```
R_2442 = pd.DataFrame(R_2442).reset_index()
R_2442.columns = ['movie_id','rating_pred']
```

In [44]:

```
R_result = pd.merge(pd.merge(df_movies, R_2442, how='inner', on='movie_id'),
                    ratings[ratings['user_id']==user_id].drop('user_id',axis=1), how='left', on='mo
```

In [45]:

```
R_fitted = R_result[R_result['rating'].isna()==False]
R_fitted = R_fitted[['movie_id','title','rating','rating_pred']].reset_index(drop=True)
```

In [46]:

```
R_fitted.head(10)
```

Out[46]:

	movie_id	title	rating	rating_pred
0	1	Dinosaur Planet	3.0	3.307235
1	30	Something's Gotta Give	3.0	3.589936
2	188	Dead Birds	3.0	3.304152
3	191	X2: X-Men United	4.0	3.462255
4	283	If These Walls Could Talk	4.0	3.451830
5	457	Kill Bill: Vol. 2	3.0	3.664112
6	486	Journey to the Center of the Earth	4.0	3.288731
7	514	Santana: Supernatural Live	3.0	3.405161
8	528	The Hitchhiker's Guide to the Galaxy	4.0	3.445424
9	594	By Hook or By Crook	2.0	3.819438

In [47]:

```
# 원래 취향
R_fitted.sort_values(by=['rating'], ascending=False).head(10)
```

Out[47]:

	movie_id	title	rating	rating_pred
18	1084	Walking with Prehistoric Beasts	5.0	3.661165
73	2862	The Silence of the Lambs	5.0	3.529433
23	1391	Yanni: Live at the Acropolis	5.0	3.434659
78	3113	Dante's Peak	5.0	3.478117
79	3184	Desert Hearts	5.0	3.413284
26	1470	Bend It Like Beckham	5.0	3.458431
27	1482	Beyond Borders	5.0	3.457147
92	3671	Laughing Matters	5.0	3.394318
57	2452	Lord of the Rings: The Fellowship of the Ring	5.0	3.936750
31	1709	Clash of the Titans	5.0	3.393939

In [48]:

```
R_pred = R_result[R_result['rating'].isna()==True]
R_pred = R_pred[['movie_id','title','rating','rating_pred']].reset_index(drop=True)
```

In [49]:

```
# 추천
R_pred.sort_values(by=['rating_pred'], ascending=False).head(10)
```

Out[49]:

	movie_id	title	rating	rating_pred
3363	3453	The Man Who Knew Too Much	NaN	4.620457
4307	4424	Elton John: Live in Barcelona	NaN	4.511962
1906	1945	From Hell: Bonus Material	NaN	4.477076
2055	2100	Bliss	NaN	4.474524
2484	2546	The Second Coming	NaN	4.423051
1446	1474	Classic Country Comedy	NaN	4.414985
4236	4350	The Best of the New Scooby-Doo Movies	NaN	4.394918
2965	3044	The Inheritance	NaN	4.360222
2392	2450	Inserts	NaN	4.359082
3351	3441	Kicking & Screaming	NaN	4.353238