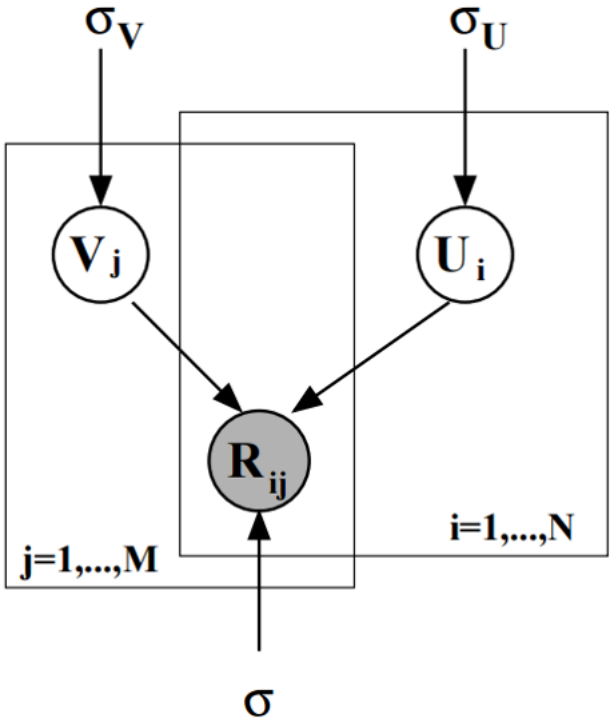
Stan notebook for Probabilistic Matrix Factorizatioin (PMF)

Probabilistic Matrix Factorization

- The goal is to fill out a matrix based on its few given data points. In movie rating application, we aim to predict the score for movies that have not been seen by users using a given set of ratings.
- U : collection of N vectors each representing a user's feature.
- V : collection of M vectors each representing a movie's feature.
- R : given set of known ratings (input data).
- The model is as follow:

$$p(U) = \Pi_{i=1}^{N}\ N(U_i|0, \sigma_u^2 I) p(V) = \Pi_{i=1}^{M}\ N(V_i|0, \sigma_\nu^2 I) p(R) = \Pi_{i=1}^{N}\Pi_{j=1}^{M}\ N(R_{ij}|U_i^T V_j, \sigma^2 I)$$

- The objective is to maximize likelihood over U and V.
- Following figure show its graphical model.



code to read the data and model

stan code

```
data{
  int<lower=1> N;               // number of users
  int<lower=1> M;               // number of movies
  int<lower=1> S;               // number of ratings samples
  int<lower=1,upper=N> user[S]; // user number for ratings
  int<lower=1,upper=M> movie[S]; // movie number for ratings
  vector<lower=1,upper=5>[S] R;     // ratings
  int<lower=1> d;               // dimensionality for vector of users and movies
```

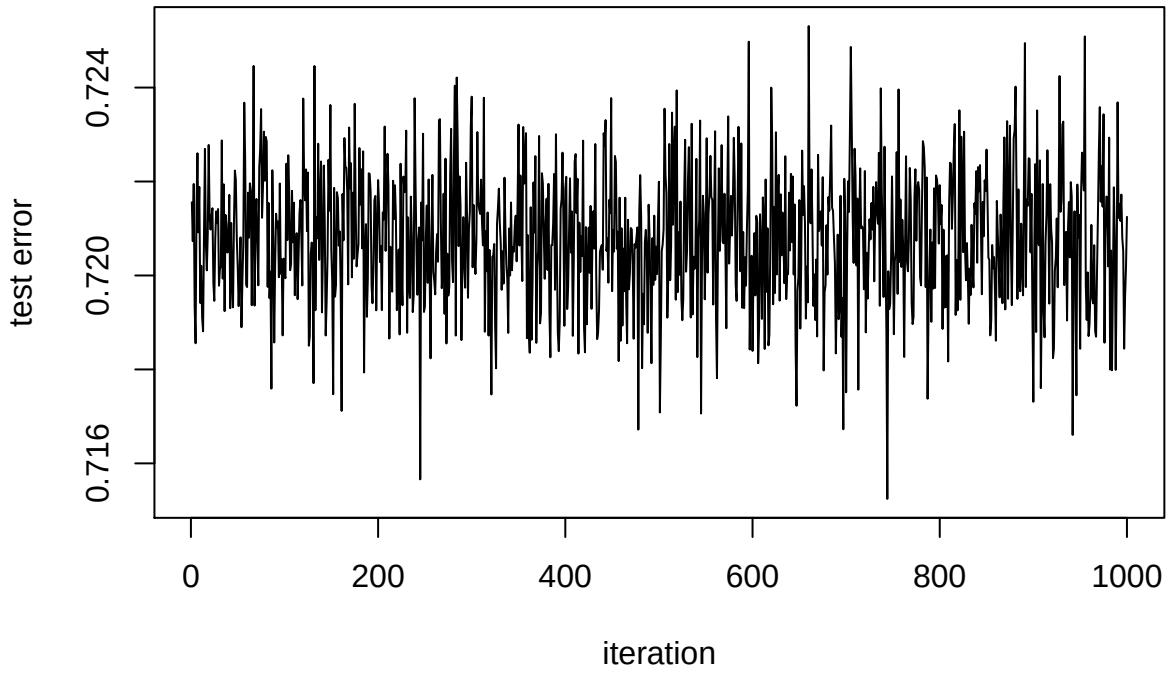
```
  real<lower=0> sigmau;        // variance for users
  real<lower=0> sigmav;        // variance for movies
  real<lower=0> sigma;
  real realzero;
}

parameters{
  matrix[N,d] u;
  matrix[M,d] v;
}

transformed parameters{
  vector[S] prod_u_vt;
  matrix[d,M] vt;
  vt = v'; # transpose of movie vectors
  for (i in 1:S){
    prod_u_vt[i] = u[user[i],:]*(vt[:,movie[i]]); # dot product of user i and movie j to represent mean

  }

}

model{
  for (i in 1:d){
  u[:,i] ~ normal(realzero , sigmau);
  }
  for (j in 1:d){
  v[:,j] ~ normal(realzero , sigmav);
  }
  R ~ normal(prod_u_vt , sigma);
}
```

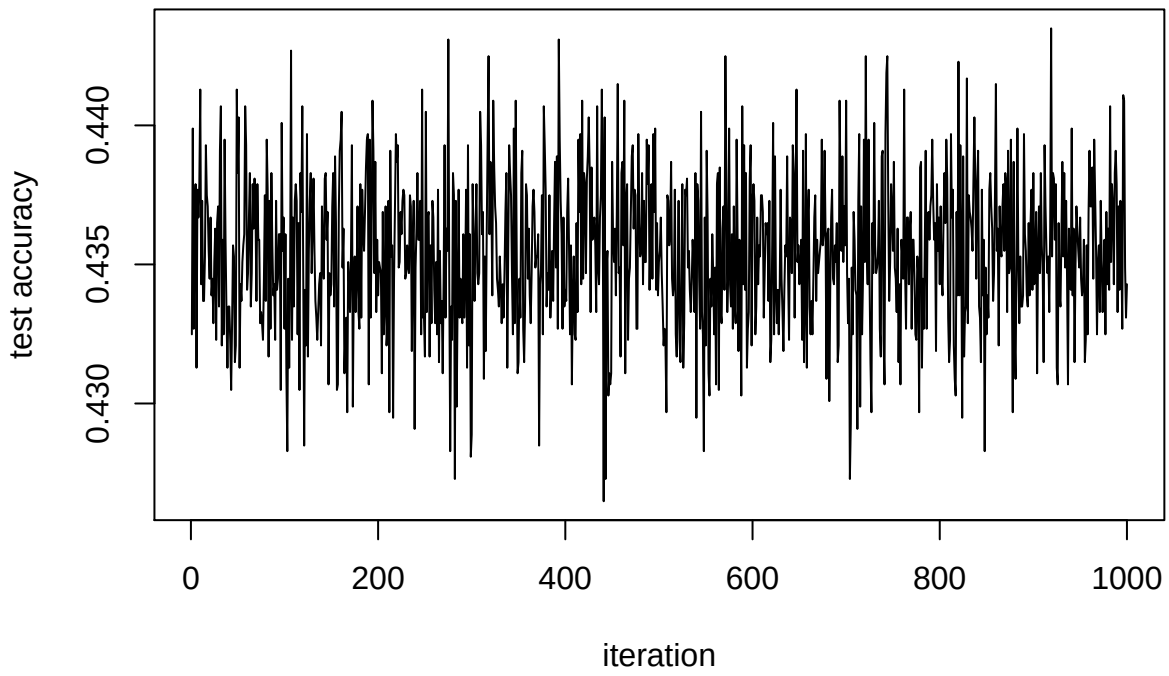
fitting the model

```
## Loading required package: ggplot2

## Loading required package: StanHeaders

## rstan (Version 2.14.1, packaged: 2016-12-28 14:55:41 UTC, GitRev: 5fa1e80eb817)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## rstan_options(auto_write = TRUE)
## options(mc.cores = parallel::detectCores())

## hash mismatch so recompiling; make sure Stan code ends with a blank line

## In file included from C:/Users/Ghazal/Documents/R/win-library/3.3/BH/include/boost/config.hpp:39:0,
##                  from C:/Users/Ghazal/Documents/R/win-library/3.3/BH/include/boost/math/tools/config
##                  from C:/Users/Ghazal/Documents/R/win-library/3.3/StanHeaders/include/stan/math/rev/
##                  from C:/Users/Ghazal/Documents/R/win-library/3.3/StanHeaders/include/stan/math/rev/
##                  from C:/Users/Ghazal/Documents/R/win-library/3.3/StanHeaders/include/stan/math/rev/
##                  from C:/Users/Ghazal/Documents/R/win-library/3.3/StanHeaders/include/stan/math/rev/m
##                  from C:/Users/Ghazal/Documents/R/win-library/3.3/StanHeaders/include/stan/math.hpp:4
##                  from C:/Users/Ghazal/Documents/R/win-library/3.3/StanHeaders/include/src/stan/model/
##                  from file13ec54c5d85.cpp:8:
## C:/Users/Ghazal/Documents/R/win-library/3.3/BH/include/boost/config/compiler/gcc.hpp:186:0: warning:
##  #  define BOOST_NO_CXX11_RVALUE_REFERENCES
##  ^
```

```
## <command-line>:0:0: note: this is the location of the previous definition
##
## SAMPLING FOR MODEL 'pmf' NOW (CHAIN 1).
##
## Chain 1, Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1, Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1, Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1, Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1, Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1, Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1, Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1, Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1, Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1, Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1, Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1, Iteration: 2000 / 2000 [100%]  (Sampling)
##  Elapsed Time: 4292.3 seconds (Warm-up)
##                4853.31 seconds (Sampling)
##                9145.61 seconds (Total)
```

evaluating results




test accuracy
0.430
0.435
0.440
0
200
400
600
800
1000
iteration