Factorización Matricial para RecSys

IIC 3633 - Sistemas Recomendadores - PUC Chile

Denis Parra

TOC

En esta clase

1. Contexto Histórico (Background)
2. SVD
3. SVD en sistemas recomendadores
4. FunkSVD: descenso del gradiente estocástico
5. Ejemplo en pyreclab
6. Extensiones del modelo básico: biases, implicit feedback, concept drift
7. Consideraciones para implementación en sistemas en producción

Background I

- Los primeros trabajos que usaron factorización matricial en sistemas recomendadores datan de comienzos de los 2000 (Sarwar et al. 2000; Sarwar et al. 2002) y mostraron resultados promisorios, pero no fue sino hasta el Netflix Prize (2006-2009) que los métodos de factorización matricial se volvieron más populares.

- El Blog Post de un participante del Netflix Prize, Simon Funk (http://sifter.org/~simon/journal/20061211.html), que describió su solución basado en un cálculo incremental de SVD usando regularización, basado en el trabajo de Gorrell et al. (2005), ha sido frecuentemente citado en trabajos de sistemas recomendadores.

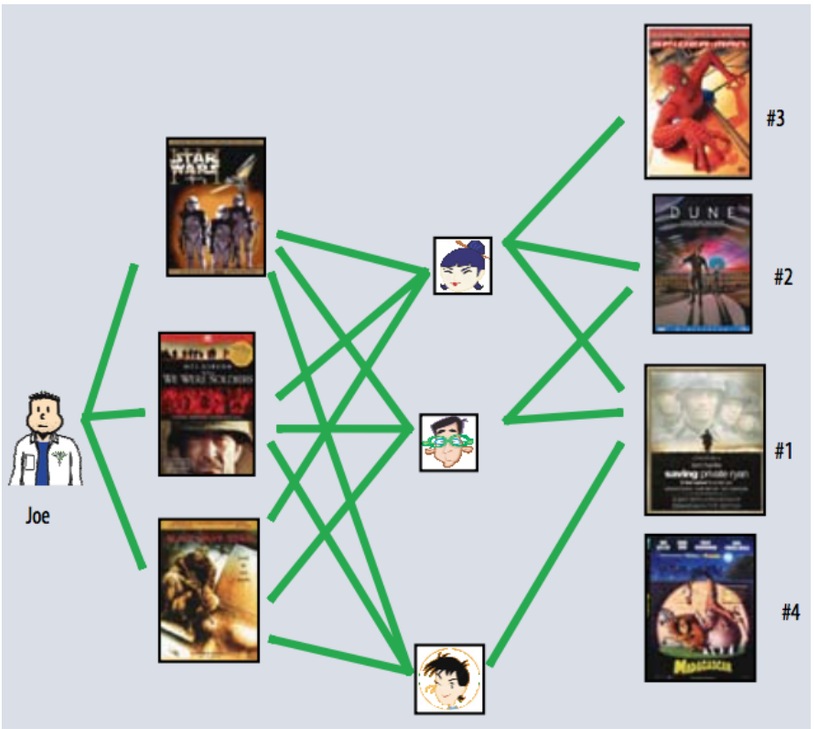
Background II

- El trabajo ganador del Netflix Prize se basa en una composición (blending) de varios modelos, pero la técnica de factorización matricial es la más importante junto con Restricted Boltzmann Machines; de hecho, fueron las únicas 2 que Netflix puson realmente en producción terminado el concurso.

- De ahí en adelante, una gran parte de los trabajos en sistemas recomendadores se fundan en estos modelos factorización matricial, también llamados de factores latentes.

Introducción I

- En el filtrado colaborativo usamos la noción de "lo que le gusta a usuarios similares a mí podría también gustarme".



Introducción II

- En el los modelos de factores latentes, el objetivo es encontrar un **embedding** donde tanto usuarios como ítems puedan mapearse en conjunto.

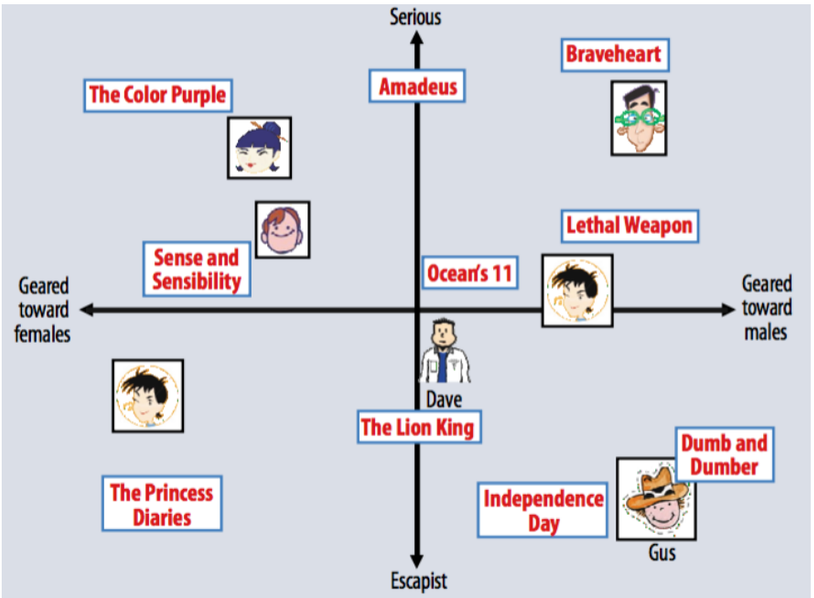


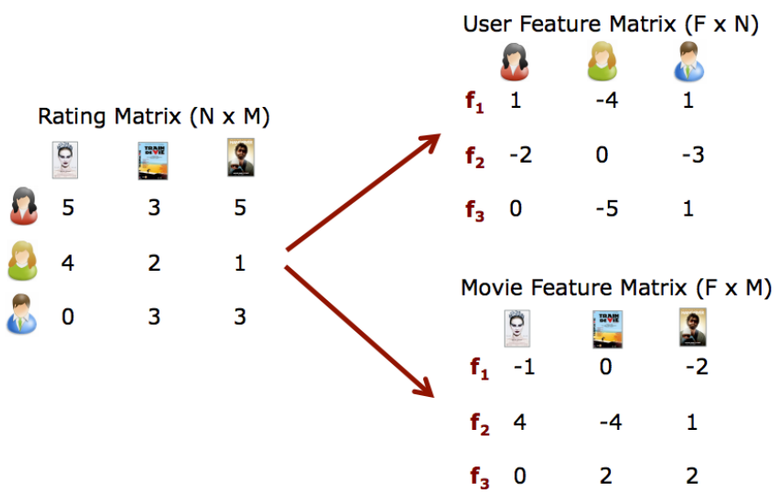
Figure 2. A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.

Introducción III

- Bajo el paradigma de factores latentes de usuarios e ítems, la predicción de ratings podría realizarse de esta manera:

$$r_{ui} = q_i^T \cdot p_u$$

- donde
  - q_i corresponde al vector de factores del item i en el espacio latente y, análogamente,
  - p_u al vector de factores latentes del usuario u.



Introducción IV

- ¿Cómo encontrar los valores de los vectores q_i y p_u?
  - Opción 1: Usando directamente **SVD**, técnica de factorización matricial utilizada frecuentemente en recuperación de información (Latent Semantic Analysis).
  - Opción 2: Usar la siguiente función de pérdida, como un problema de optimización con regularización l_2 que podemos resolver con stochastic gradient descent u otras técnicas.

$$\min_{q*,p*} \sum_{(u,i)\in K} (r_{ui} - q_i^T \cdot p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

donde K es el conjunto de pares (u, i) para los cuales r_{ui} es conocido (training set).

SVD I

- **Teorema de diagonalización de matrices**: Sea A una matriz $m \times m$ cuadrada con valores reales, con m vectores propios linealmente independientes (en inglés eigenvectors). Luego, existe una descomposición

$$A = USU^{-1}$$

, donde las columnas de U son los vectores propios de A y S es una matriz diagonal cuyas entradas son los valores propios de A en orden decreciente.

$$\begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \dots & \\ & & & \lambda_M \end{pmatrix}, \lambda_i \geq \lambda_{i+1}.$$

ref: Introduction to Information Retrieval, Manning et. al (2010)

SVD II

- **Teorema de diagonalización simétrica**: Sea A una matriz $m \times m$ cuadrada y simétrica con valores reales, con m vectores propios linealmente independientes. Luego, existe una descomposición

$$A = QSQ^T,$$

donde las columnas de Q son los vectores propios ortogonales y normalizados de A, y S es una matriz diagonal cuyas entradas son los valores propios de A. Más aún, todas las entradas de Q son reales y tenemos que $Q^{-1} = Q^T$.

ref: Introduction to Information Retrieval, Manning et. al (2010)

SVD III

- **Teorema**: Una matriz arbitraria $A \in \mathbb{R}^{m \times n}$ permite una descomposición matricial de la forma:

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T = U \tilde{S} V^T, \quad \tilde{S} := \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix},$$

donde $U \in \mathbb{R}^{m \times m}$ y $V \in \mathbb{R}^{n \times n}$ son ambas matrices ortogonales, y la matriz {S} es diagonal:

$$S = diag = (\sigma_1, \dots, \sigma_r),$$

donde los números positivos $\sigma_1 \geq \dots \geq \sigma_r > 0$ son únicos, y son llamados los valores singulares de A. El número $r \geq min(m, n)$ es igual al ranking de A y la tripleta $(U, \tilde{S}, V)$ es llamada la **singular value decomposition** (SVD) de A.

Las primeras r columnas de $U : u_i, i = 1, \dots, r$ (respectivamente $V : v_i, i = 1, \dots, r$) son llamados los vectores izquierdos (resp. derechos) singulares de A, y satisfacen:

$$Av_i = \sigma_i u_i, \quad u_i^T A = \sigma_i v_i, \quad i = 1, \dots, r.$$

ref: https://inst.eecs.berkeley.edu/~ee127a/book/login/thm_svd.html

12/34

Conectando los teoremas

- Las matrices que usualmente usamos de usuario-item no son cuadradas ni simétricas.
- Pero si consideramos el teorema SVD

$$A = U\tilde{S}V^T$$

y luego que AA^T es una matriz cuadrada y simétrica, tenemos:

$$AA^T = U\tilde{S}V^T \; V\tilde{S}U^T = U\tilde{S}^2U^T$$

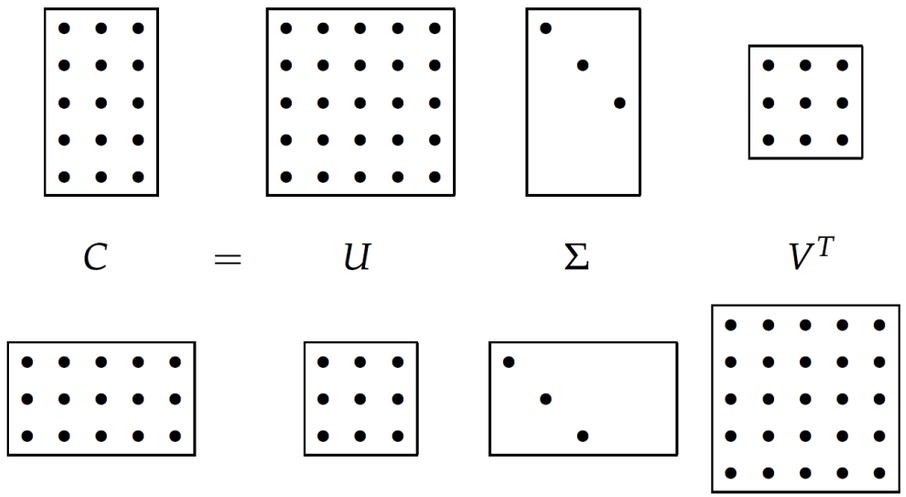
y de forma análoga

$$A^TA = V\tilde{S}U^T \; U\tilde{S}V^T = V\tilde{S}^2V^T$$

podemos calcular las matrices U, V y $\tilde{S}$.

ref: Introduction to Information Retrieval, Manning et. al (2010)

SVD Visualmente



► **Figure 18.1** Illustration of the singular-value decomposition. In this schematic illustration of (18.9), we see two cases illustrated. In the top half of the figure, we have a matrix C for which $M > N$. The lower half illustrates the case $M < N$.

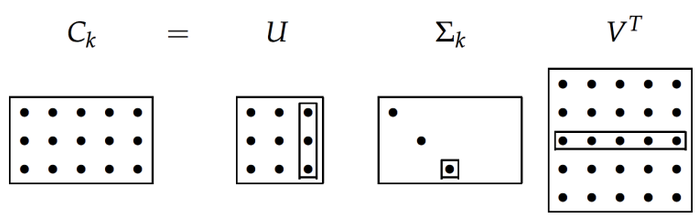
ref: Introduction to Information Retrieval, Manning et. al (2010)

Aproximación Low-rank

- Dada una matriz A de $m \times n$, deseamos encontrar una matriz A_k de ranking a lo más k y que minimice la norma Frobenius de la matriz de diferencias $X = A - A_k$, definida como

$$||X||_F = \sqrt{\sum_{i=1}^{M} \sum_{j=1}^{n} X_{ij}^2}$$

- la matriz $low - rank$ aproximada A_k la podemos encontrar reemplazando por cero los valores singulares más pequeños de la matriz diagonal resultante de SVD:



► **Figure 18.2** Illustration of low rank approximation using the singular-value decomposition. The dashed boxes indicate the matrix entries affected by "zeroing out" the smallest singular values.

SVD en Sistemas Recomendadores I

- Sarwar et al (2000) presentaron un modelo para hacer recomendaciones usando SVD:

  - factor R_{norm} using SVD to obtain U, S and V.
  - reduce the matrix S to dimension k
  - compute the square-root of the reduced matrix S_k, to obtain $S_k^{1/2}$
  - compute two resultant matrices: $U_k S_k^{1/2}$ and $S_k^{1/2} V_k'$

$$C_{P_{pred}} = \overline{C} + U_K \cdot \sqrt{S_k}' (c) \cdot \sqrt{S_k} . V_k' (P)$$

- Una desventaja importante es cómo calcular la representación latente para nuevos usuarios.

  ref: Sarwar et al (2000). Application of Dimensionality Reduction in Recommender System -- A Case Study.

16/34

SVD en Sistemas Recomendadores II

- Sarwar et al (2001) presentan un nuevo modelo para hacer recomendaciones usando SVD, y que permite imputar usuarios nuevos en el espacio latente:

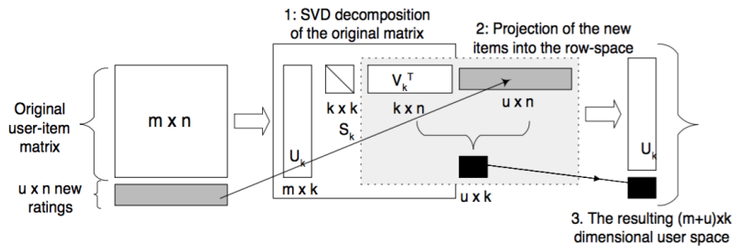
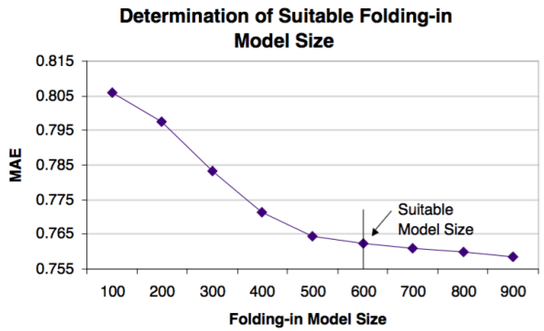


Figure 1: Schematic diagram of the SVD folding-in technique.

$$P = U_k \times U_k^T \times N_u.$$

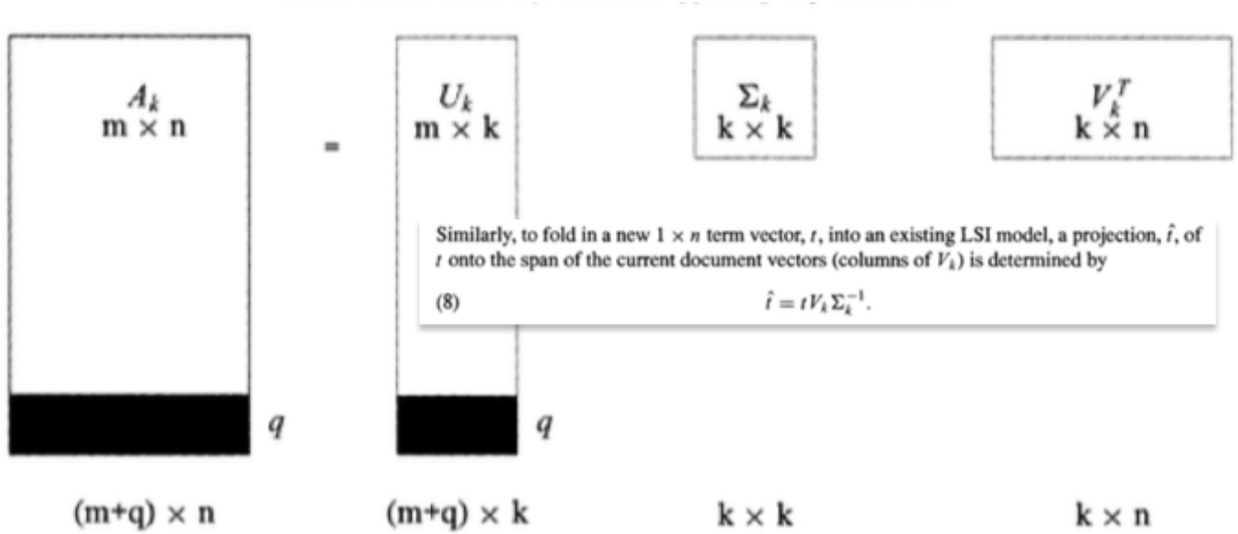


- ref: Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems

17/34

SVD en Sistemas Recomendadores III

- Folding in: Using Linear Algebra for Intelligent Information Retrieval



Similarly, to fold in a new $1 \times n$ term vector, t, into an existing LSI model, a projection, $\hat{t}$, of t onto the span of the current document vectors (columns of V_k) is determined by

(8) $$\hat{t} = t V_k \Sigma_k^{-1}.$$

FIG. 3. *Mathematical representation of folding-in q terms.*

Debilidades de SVD tradicional para RecSys

- Calcular la SVD de una matriz es muy costoso, $O(m^2 n)$
- Problemas debido a la gran cantidad de celdas vacías en la matriz usuario-ítem obligaron a los autores en (Sarwar, 2000) a pre-llenarla.
- El modelo tradicional de SVD no está definido cuando hay un conocimiento parcial de la matriz.
- Utilizar un método convencional también corre el peligro de over-fitting.

Solución de Simon Funk: SVD incremental

- Basada en Stochastic Gradient Descent, considerando trabajo de Gorrell (2005)

```
/*
 * Where:
 *   real *userValue = userFeature[featureBeingTrained];
 *   real *movieValue = movieFeature[featureBeingTrained];
 *   real  lrate = 0.001;
 */
static inline
void train(int user, int movie, real rating)
{
        real err = lrate * (rating - predictRating(movie, user));

        userValue[user] += err * movieValue[movie];
        movieValue[movie] += err * userValue[user];
}
```

  - Itera sobre todos los ratings observados r_{ui}, prediciendo $r_{u,i}$ y calculando el error.

$$e_{ui} \overset{def}{=} r_{ui} - q_i^T p_u$$

  - Luego actualiza los factores latentes usando las siguientes reglas de actualización.

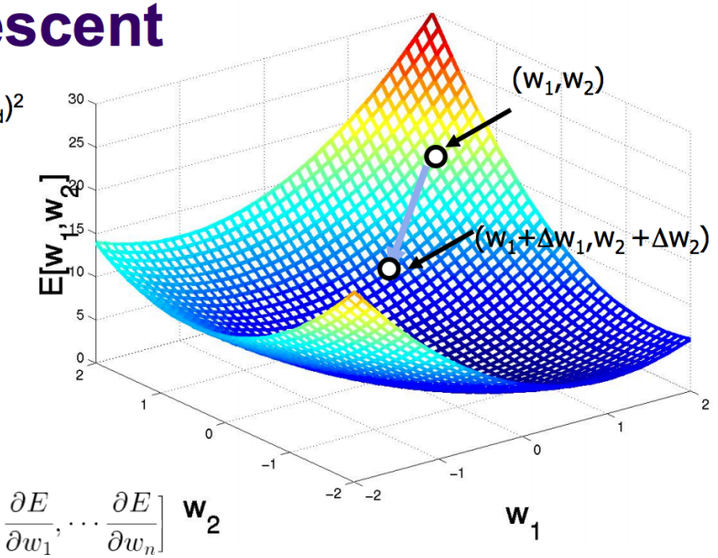
$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

Descenso del gradiente

- Basado en ejemplo de Andrew Ng:

Gradient Descent

$E[w_1,\ldots,w_n] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Derivación de las reglas de actualización

- Tenemos

$$\Delta\vec{w} = -\eta\nabla E[\vec{w}]$$

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d(t_d - o_d)^2 \\ &= \frac{1}{2}\sum_d\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\ &= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\ &= \sum_d(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\ \frac{\partial E}{\partial w_i} &= \sum_d(t_d - o_d)(-x_{i,d})\end{aligned}$$

22/34

Pseudocódigo para el algoritmo completo

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, *where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

Regularización

- Al tener tan pocos datos la matriz (densidad del 1%-2%), el procedimiento corre el riesgo de sobre-entrenarse (over-fitting). Un regularizador permite penalizar valores muy altos de los coeficientes de los vectores latentes (Bishop, 2007;Goodfellow et al, 2016)

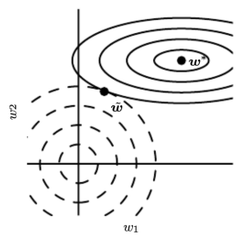


Figure 7.1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal $\boldsymbol{w}$. The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L^2 regularizer. At

---

- Por eso se agregan los términos $||p_u||$ y $||q_i||$ a la función de pérdida (loss function)

$$\min_{q*,p*} \sum_{(u,i)\in K} (r_{ui} - q_i^T \cdot p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2)$$

- De esa forma, las reglas de actualización del SGD incluyen (siendo γ el learning rate):

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

Extensiones I

- El modelo de Koren et al. para el Netflix prize incorpora los biases de usuarios e items:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u$$

$$\min_{p^*, q^*, b^*} \sum_{(u,i) \in \kappa} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda$$

$$(|| p_u ||^2 + || q_i ||^2 + b_u^2 + b_i^2)$$

Código del FunkSVD en paquete pyreclab

- https://github.com/gasevi/pyreclab/blob/master/algorithms/AlgFunkSvd.cpp

```
for( size_t u = 0 ; u < nusers ; ++u )
{
   SparseRow< boost::numeric::ublas::mapped_matrix<double, boost::numeric::ublas::row_major> > row = m_ratingMatrix.userVector( u );
   SparseRow< boost::numeric::ublas::mapped_matrix<double, boost::numeric::ublas::row_major> >::iterator ind;
   SparseRow< boost::numeric::ublas::mapped_matrix<double, boost::numeric::ublas::row_major> >::iterator end = row.end();
   for( ind = row.begin() ; ind != end ; ++ind )
   {
      int i = ind.index();

      double pred = predict( u, i );
      double eui = *ind - pred;

      // squared error: ( rui - ( mu + bu + bi + pu*qi ) )^2
      loss += eui * eui;

      double bu = m_userBias[u];
      double bi = m_itemBias[i];

      // bias regularization: lambda * ( bu^2 + bi^2 )
      loss += m_lambda * bu * bu;
      loss += m_lambda * bi * bi;
```

Código del FunkSVD en paquete pyreclab II

- https://github.com/gasevi/pyreclab/blob/master/algorithms/AlgFunkSvd.cpp

```
// update biases
double buNext = eui - m_lambda * bu;
m_userBias[u] += m_learningRate * buNext;
double biNext = eui - m_lambda * bi;
m_itemBias[i] += m_learningRate * biNext;

for( size_t f = 0 ; f < m_nfactors ; ++f )
{
   double puf = m_userP[u][f];
   double qif = m_itemQ[i][f];

   double delta_u = eui * qif - m_lambda * puf;
   double delta_i = eui * puf - m_lambda * qif;

   // update user and item matrices
   m_userP[u][f] += m_learningRate * delta_u;
   m_itemQ[i][f] += m_learningRate * delta_i;

   // user and item vectors regularization: lambda * ( ||pu||^2 + ||qi||^2 )
   loss += m_lambda * puf * puf + m_lambda * qif * qif;

   if( !m_running )
   {
      return STOPPED;
   }
```

Extensiones II

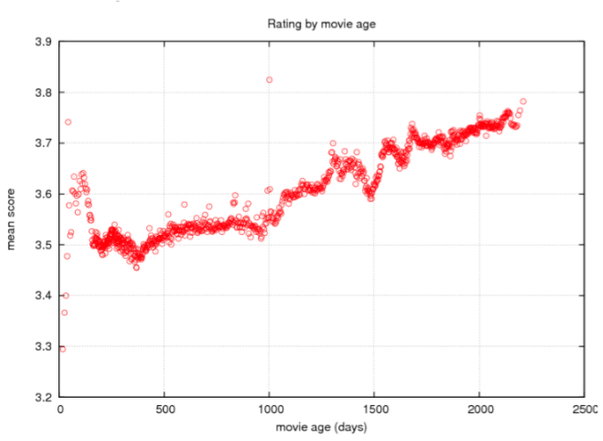
- Luego de incluye información implícita en el modelo incorporando $N(u)$

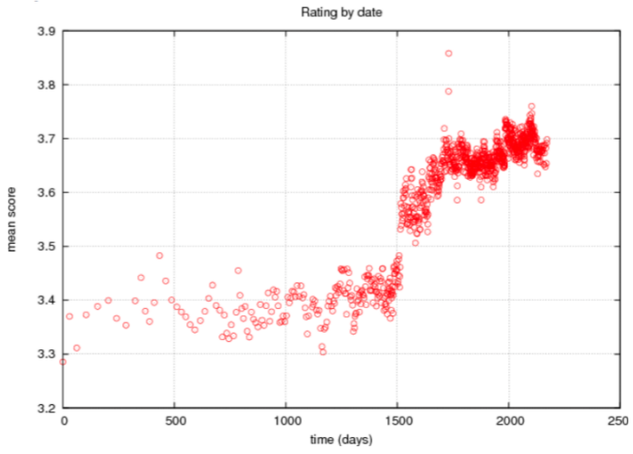
$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T [p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i + \sum_{a \in A(u)} y_a]$$

Extensiones III

- Luego de incluye información temporal (concept drift)

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T p_u(t)$$





Resultados Finales

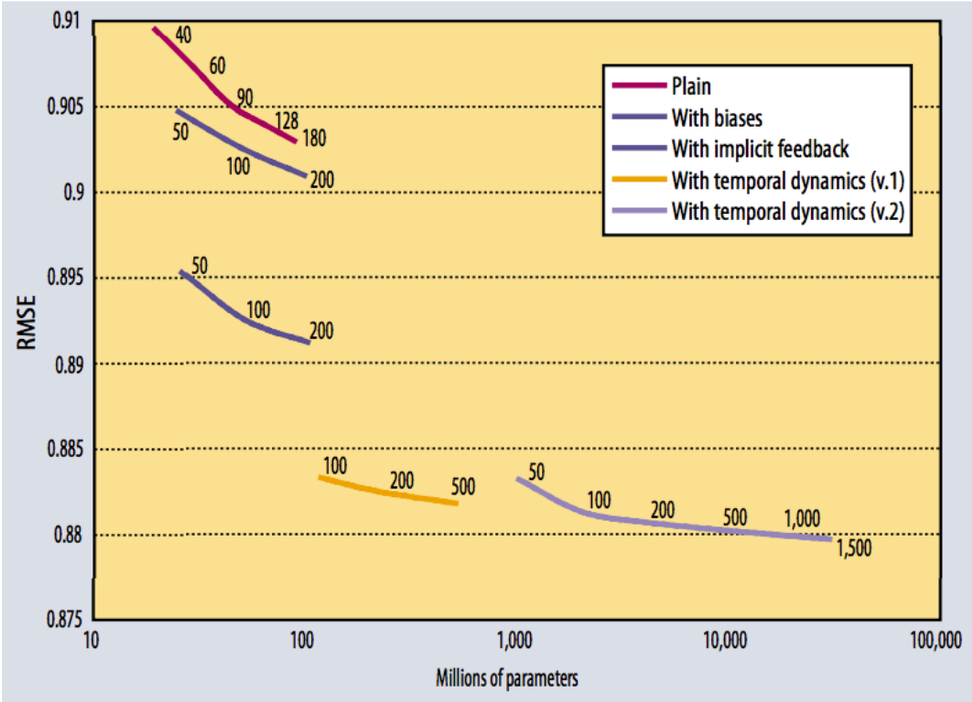


Figure 4. Matrix factorization models' accuracy. The plots show the root-mean-square error of each of four individual factor models (lower is better). Accuracy improves when the factor model's dimensionality (denoted by numbers on the charts) increases. In addition, the more refined factor models, whose descriptions involve more distinct sets of parameters, are more accurate. For comparison, the Netflix system achieves RMSE = 0.9514 on the same dataset, while the grand prize's required accuracy is RMSE = 0.8563.

Muchos nuevos modelos!

- Collaborative Filtering for Implicit Feedback Datasets
- BPR
- Factorization Machines
- Tensor Factorization (context)
- NMF
- SLIM

¿Implementación en el Mundo Real?

- Sugiero estas diapositivas:
  - Spotify: Collaborative Filtering with Spark http://www.slideshare.net/MrChrisJohnson/collaborative-filtering-with-spark y
  - Algorithmic Music Recommendations at Spotify http://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify/36-Open_ProblemsHow_to_go_from
  - Netflix: Building Large-scale Real-world Recommender Systems - Recsys2012 tutorial http://www.slideshare.net/xamat/building-largescale-realworld-recommender-systems-recsys2012-tutorial

Referencias

- Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2000). Application of Dimensionality Reduction in Recommender System -- A Case Study
- Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2002). Incremental singular value decomposition algorithms for highly scalable recommender systems. In Fifth International Conference on Computer and Information Science (pp. 27-28).
- Manning, C. D., Raghavan, P. and Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA.
- Gorrell, G., & Webb, B. (2005). Generalized hebbian algorithm for incremental latent semantic analysis. In INTERSPEECH (pp. 1325-1328).
- Funk, S. (2006). Try this at home. Blog post: http://sifter.org/~simon/journal/20061211.html. December11.
- Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37. Chicago
- recommenderlab https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf
- rrecsys https://cran.r-project.org/web/packages/rrecsys/index.html

Referencias II

- SVD https://inst.eecs.berkeley.edu/~ee127a/book/login/l_svd_def.html
- SVD https://inst.eecs.berkeley.edu/~ee127a/book/login/thm_svd.html
- Goodfellow, I., Bengio, Y., Courville, A. (2016) Deep Learning. http://www.deeplearningbook.org/
- Dimensionality Reduction and the Singular Value Decomposition http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/svd.html
- Timely Development on the Netflix Prize http://www.timelydevelopment.com/demos/NetflixPrize.aspx
- SVD and the Netflix dataset http://www.slideshare.net/bmabey/svd-and-the-netflix-dataset-presentation
- Derivation on SVD update rules http://sifter.org/~simon/journal/20070815.html
- UM (mike Ekstrand) video on FunkSVD, Coursera https://www.coursera.org/learn/recommender-systems/lecture/PYZI5/funksvd-training-algorithm
- Chih-Chao, M. 2008. A Guide to Singular Value Decomposition for Collaborative Filtering.
- Collaborative Filtering for Netflix, Michael Percy (2009) https://classes.soe.ucsc.edu/cmps242/Fall09/proj/mpercy_svd_paper.pdf