

# Factorización Matricial para RecSys

IIC 3633 - Sistemas Recomendadores - PUC Chile

Denis Parra

Profesor Asistente, DCC, PUC Chile

# Memo del Semestre

- **Tarea 1:** Deadline nuevo, Jueves 8 de Septiembre.
- **Lecturas en el semestre:** Ya fueron actualizadas en el sitio web del curso.

| Week | Fecha semana   | Clase Martes                       | Clase Jueves                  | Presentador 1  | Presentador 2 | Presentador 3 |           |
|------|----------------|------------------------------------|-------------------------------|----------------|---------------|---------------|-----------|
| I    | 2 - 4 Ago      | Intro + CF                         | CF + Clustering               |                |               |               |           |
| II   | 9 - 11 Ago     | CF item-based                      | Slope One + RecSys            |                |               |               |           |
| III  | 16 - 18 Ago    | Evaluacion de RecSys               | Evaluacion de RecSys          |                |               |               |           |
| IV   | 23 - 25 Ago    | Content-based                      | Tag-based                     |                |               |               |           |
| V    | 30 Ag - 1 Sept | Hybrid                             | Context-aware                 |                |               |               |           |
| VI   | 6 - 8 Sept     | Factorizacion Matricial            | Implicit Feedback             |                |               |               |           |
| VII  | 13 - 15 Sept   | student presentation (Context, MF) | RECSYS Conf                   | V. Dominguez   | J. Schellman  | P. Lopez      |           |
| VIII | 20 - 22 Sept   | RECSYS Conf                        | student presentation (IF, MF) | F. Lucchini    | V. Claro      | V. Castillo   |           |
| IX   | 27 - 29 Sept   | Presentaciones: Proy. Final        | Presentaciones: Proy. Final   |                |               |               |           |
| X    | 4 - 6 Oct      | User-centric RecSys/Interfaces     | student presentation          | J. Lee         | C. Kutscher   | R. Carmona    |           |
| XI   | 11 - 13 Oct    | Active Learning/Ranking            | student presentation          | F. Rojos       | J. Navarro    | N. Morales    |           |
| XII  | 18 - 20 Oct    | Graph-based                        | student presentation          | P. Messina     | S. Martí      | J. Castro     |           |
| XIII | 25 - 27 Oct    | Applications: Social/Trust/Music   | student presentation          | J.M. Herrera   | V. Dragicevic | L. Zorich     |           |
| XIV  | 1 - 3 Nov      | Applications: POI/Tourism          | student presentation          | P. Sanabria    | T. Hepner     | M. Troncoso   | I. Becker |
| XV   | 8 - 10 Nov     | Applications: Educ/Soft.Eng.       | student presentation          | R. Perez       | P. Sanabria   | J. Diaz       |           |
| XVI  | 15 - 17 Nov    | Deep Learning                      | student presentation          | Felipe del Río | L. Pose       | G. Sepulveda  |           |
| XVII | 29 Nov - 1 Dic | Presentacion Final                 | Presentacion Final            |                |               |               |           |

# TOC

## En esta clase

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

# Background I

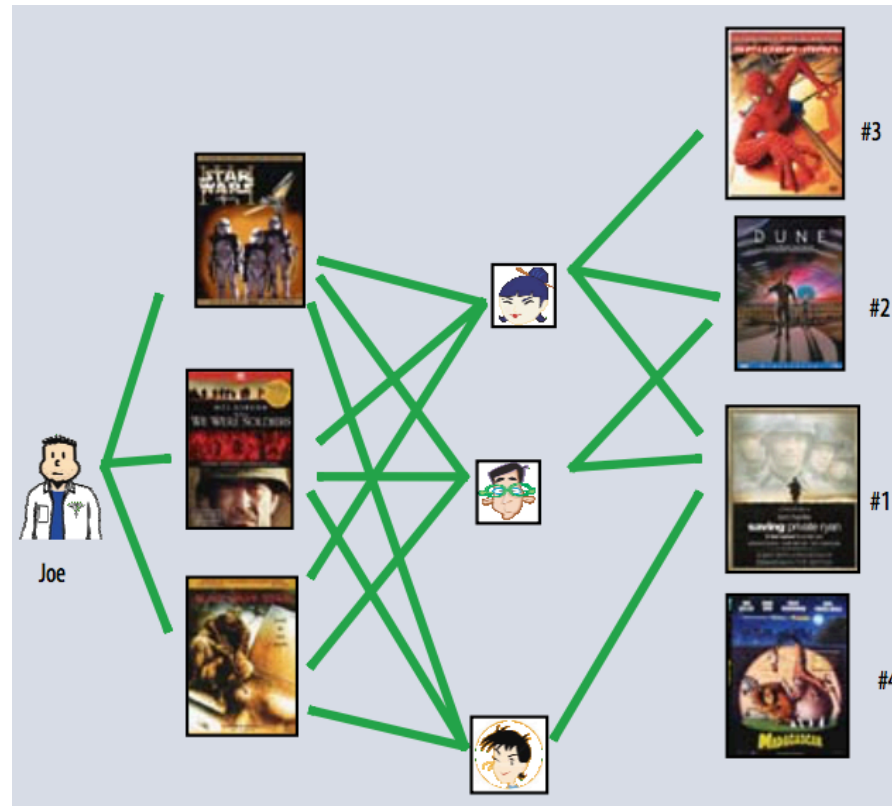
- Los primeros trabajos que usaron factorización matricial en sistemas recomendadores datan de comienzos de los 2000 (Sarwar et al. 2000; Sarwar et al. 2002) y mostraron resultados promisorios, pero no fue sino hasta el Netflix Prize (2006-2009) que los métodos de factorización matricial se volvieron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

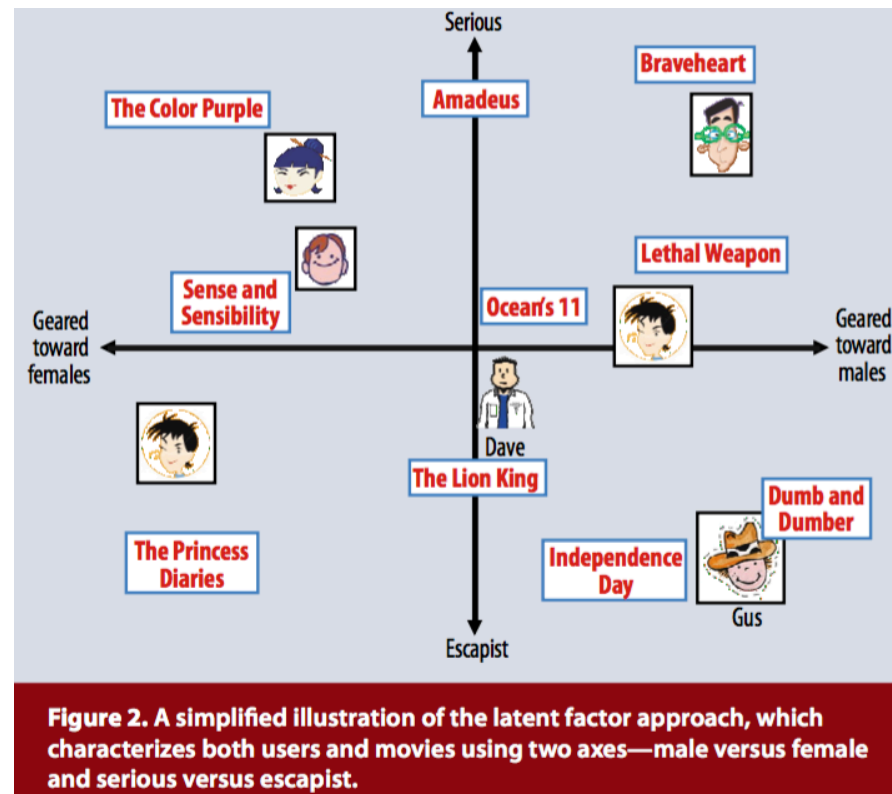
# Introducción I

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



# Introducción II

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

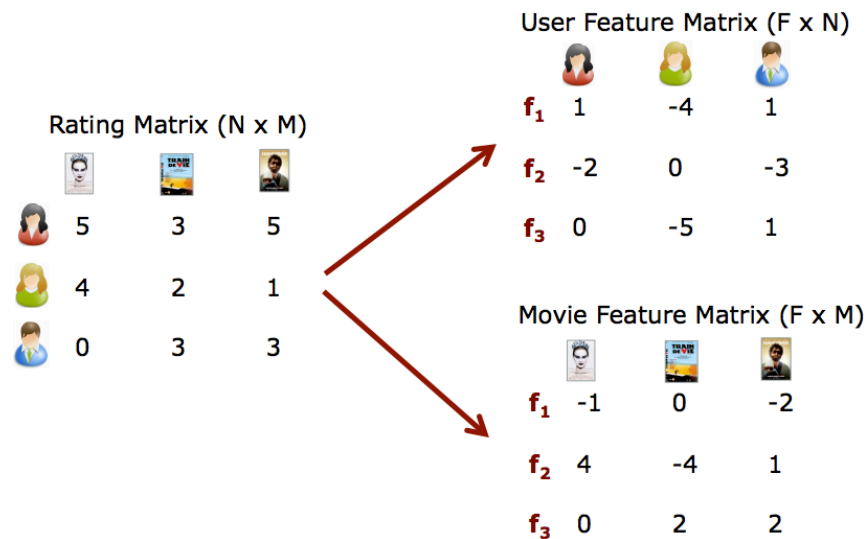


# Introducción III

- 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 ítem  $i$  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:

$$S = \text{diag} = (\sigma_1, \dots, \sigma_r),$$

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:

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

ref: [https://inst.eecs.berkeley.edu/~ee127a/book/login/thm\\_svd.html](https://inst.eecs.berkeley.edu/~ee127a/book/login/thm_svd.html)

# 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:

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

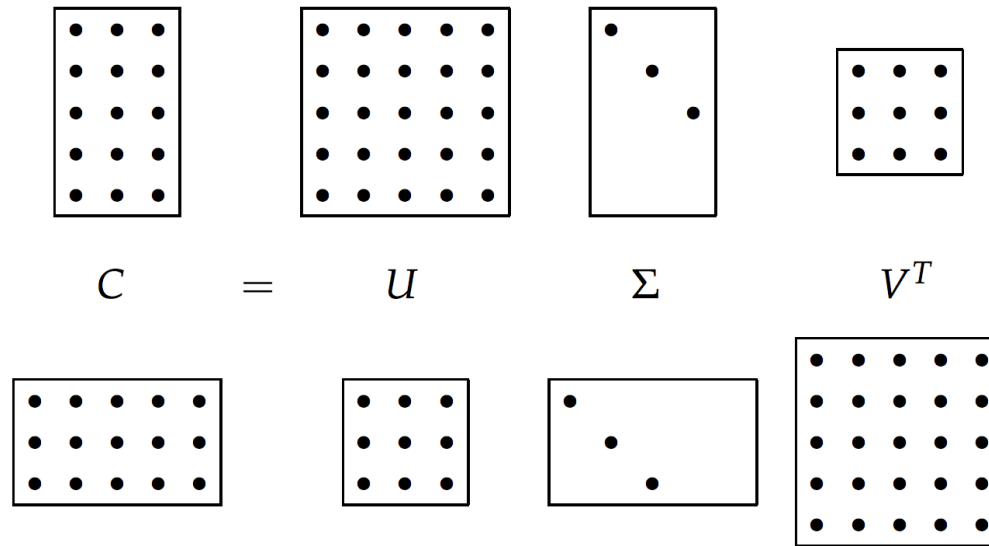
y de forma análoga

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

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  $M < N$ .

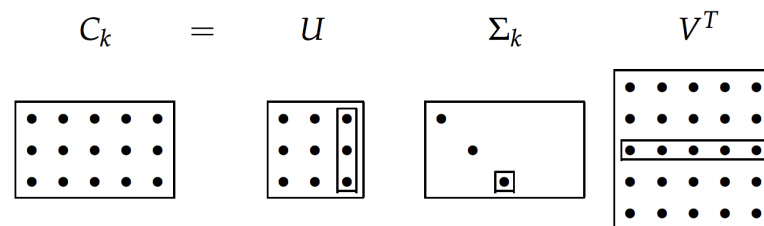
ref: Introduction to Information Retrieval, Manning et. al (2010)

# 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  $S_k^{1/2} V_k'$

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

- 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.



# 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:

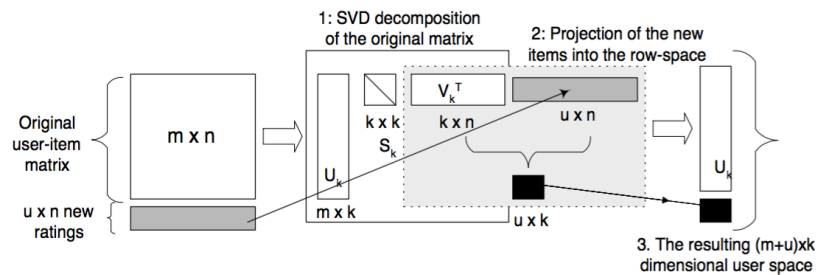
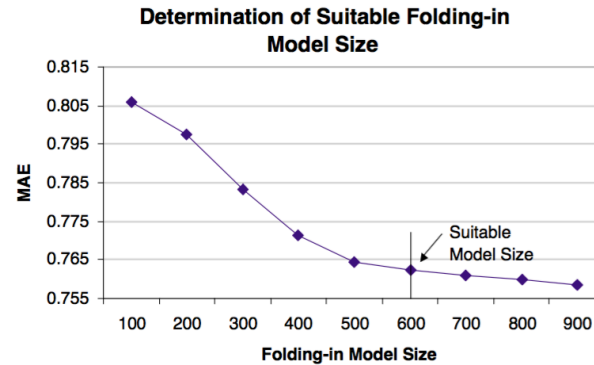


Figure 1: Schematic diagram of the SVD folding-in technique.

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



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

# Debilidades de SVD tradicional para RecSys

- Calcular la SVD de una matriz es muy costoso,  $O(m^2n)$
- Problemas debido a la gran cantidad de celdas vacías en la matriz usuario-ítem obligaron a los autores en (Sarwar, 2000) a prellenarla.
- 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.

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

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

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

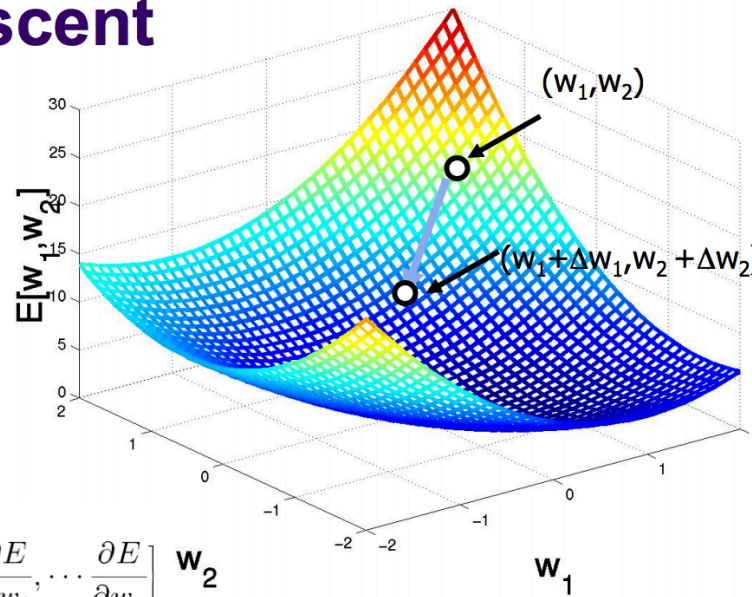
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

# Descenso del gradiente

- Basado en ejemplo de Andrew Ng:

## Gradient Descent

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



Gradient

$$\nabla E[\vec{w}] \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right] \mathbf{w}_2$$

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 I

- Considerando

$$r(i, j) = 1 \text{ if user } j \text{ has rated movie } i \text{ (0 otherwise)}$$

$$y^{(i,j)} = \text{rating by user } j \text{ on movie } i \text{ (if defined)}$$

$\theta^{(j)}$  = parameter vector for user  $j$

$x^{(i)}$  = feature vector for movie  $i$

For user  $j$ , movie  $i$ , predicted rating:  $(\theta^{(j)})^T x^{(i)}$

$m^{(j)}$  = no. of movies rated by user  $j$

Task is to learn  $\theta^{(j)}$

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2$$

# Derivación de las reglas de actualización II

- Tenemos

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

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

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \end{aligned}$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d) (-x_{i,d})$$

# 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.  $\eta$  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  $\Delta w_i$  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)

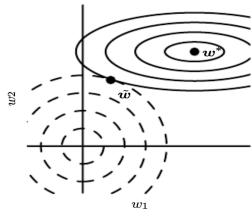


Figure 7.1: An illustration of the effect of  $L^2$  (or weight decay) regularization on the value of the optimal  $w$ . The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the  $L^2$  regularizer. At

- 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  $\gamma$  el learning rate):

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



# Código del FunkSVD en paquete rrecsys I

- [https://github.com/ludovikoba/rrecsys/blob/master/R/ALG\\_funkSVD.R](https://github.com/ludovikoba/rrecsys/blob/master/R/ALG_funkSVD.R)

```
for (f in 1:k) {
  #restart counter and error for new feature
  resetrrecsysenv()
  p <- matrix(100, nrow = row_x, ncol = col_x)
  # convergence check
  ptm <- Sys.time()
  while (!isConverged(x, p)) {

    p <- U %%% t(V)
    error <- x - p

    # update user features

    temp_U <- U

    for (j in 1:col_x) {

      delta_Uik <- lambda * (error[userIDX[[j]], j] * V[j, f] - gamma * U[userIDX[[j]], f])

      U[userIDX[[j]], f] <- U[userIDX[[j]], f] + delta_Uik

    }
  }
}
```

# Código del FunkSVD en paquete rrecsys II

- [https://github.com/ludovikcoba/rrecsys/blob/master/R/ALG\\_funkSVD.R](https://github.com/ludovikcoba/rrecsys/blob/master/R/ALG_funkSVD.R)

```
# update item features
for (i in 1:row_x) {

  delta_Vjk <- lambda * (error[i, itemIDX[[i]]] * temp_U[i, f] - gamma * V[itemIDX[[i]], f])

  V[itemIDX[[i]], f] <- V[itemIDX[[i]], f] + delta_Vjk

}

} #end convergence loop

cat("Feature:", f, "/", k, " trained. Time:", as.numeric(Sys.time() - ptm, units = "secs"), "seconds. \n")

} #end feature loop
```

# Ejemplo de recomendaciones de bromas

- <https://mhahsler-apps.shinyapps.io/Jester/>

## Joke Recommender Using Jester Data

**Rate the following jokes:**

"May I take your order?" the waiter asked.  
 "Yes, how do you prepare your chickens?"  
 "Nothing special sir," he replied. "We just tell them straight out that they're going to die."

-10  10

Two attorneys went into a diner and ordered two drinks. Then they produced sandwiches from their briefcases and started to eat. The owner became quite concerned and marched over and told them, "You can't eat your own sandwiches in here!" The attorneys looked at each other, shrugged their shoulders and then exchanged sandwiches.

-10  10

How many teddybears does it take to change a lightbulb? It takes only one teddybear, but it takes a whole lot of lightbulbs.

-10  10

Select recommender algorithm

SVD

Number of recommended jokes

3

Change jokes to rate

|     | Recommended Joke   | Predicted Rating |
|-----|--|------------------|
| j10 | Two cannibals are eating a clown, one turns to other and says: "Does this taste funny to you?"                           | 4.8              |
| j23 | Q: What is the Australian word for a boomerang that won't come back? A: A stick  | 4.7              |
| j11 | Q: What do a hurricane, a tornado, and a redneck divorce all have in common? A: Someone's going to lose their trailer... | 4.6              |

[recommenderlab example by Michael Hahsler. See the source code.](#)

# Extensiones I

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

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

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

# Extensiones II

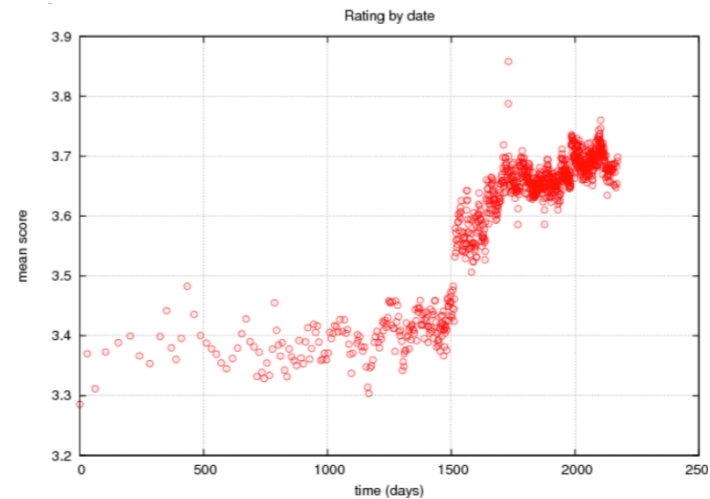
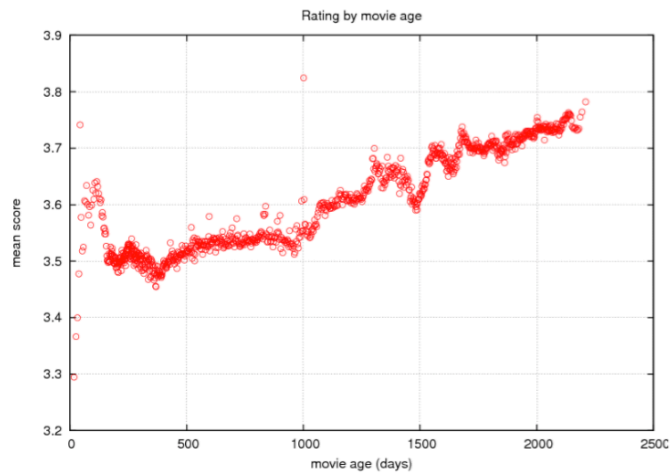
- Luego de incluye información implícita en el modelo incorporando  $N(u)$

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

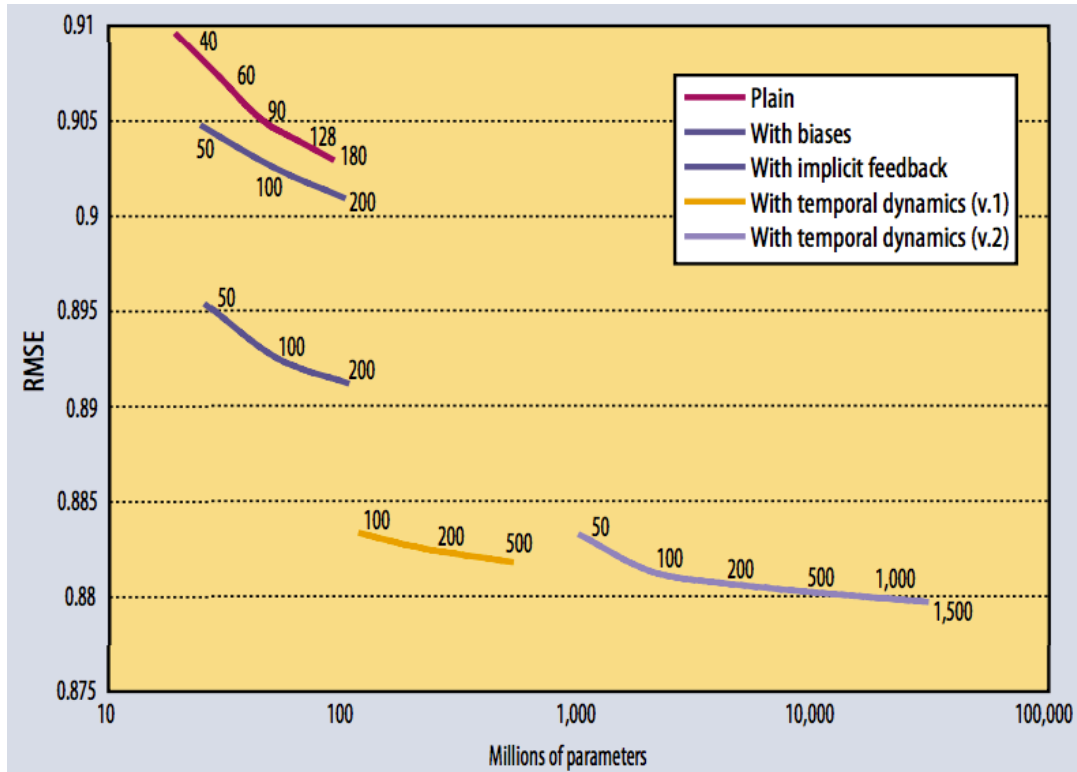
# Extensiones III

- 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](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](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>. December 11.
- 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](https://inst.eecs.berkeley.edu/~ee127a/book/login/l_svd_def.html)
- SVD [https://inst.eecs.berkeley.edu/~ee127a/book/login/thm\\_svd.html](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](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/cms242/Fall09/proj/mpercy\\_svd\\_paper.pdf](https://classes.soe.ucsc.edu/cms242/Fall09/proj/mpercy_svd_paper.pdf)