Implicit feedback

Patricio López Juri

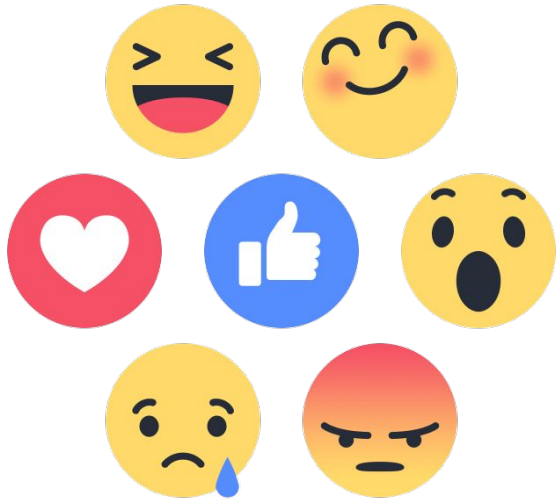
Signals in the Silence: Models of Implicit Feedback in a Recommendation System for Crowdsourcing (2014) por:

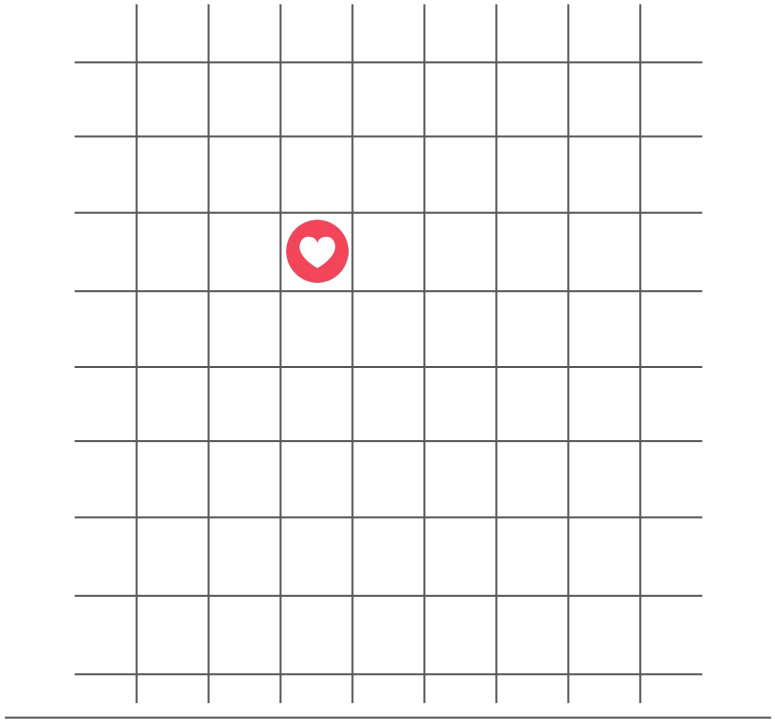
- Christopher H. Lin
- Ece Kamar
- Eric Horvitz

Feedback

Explícito

- El usuario declara sus gustos y preferencias
- La GUI presenta elementos visuales para interactuar

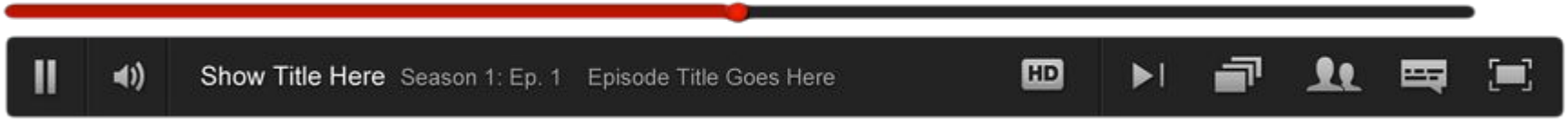


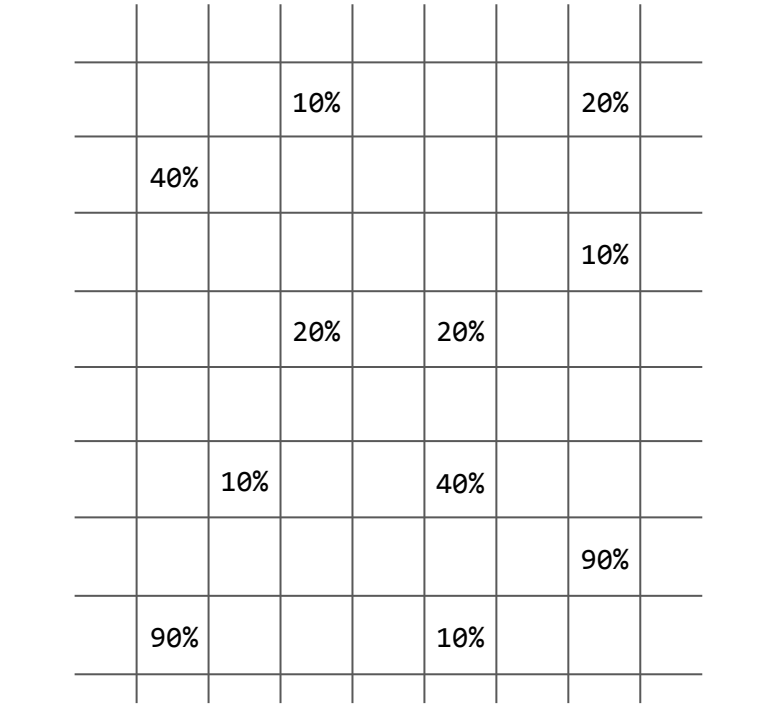


Data Sparsity

Implícito

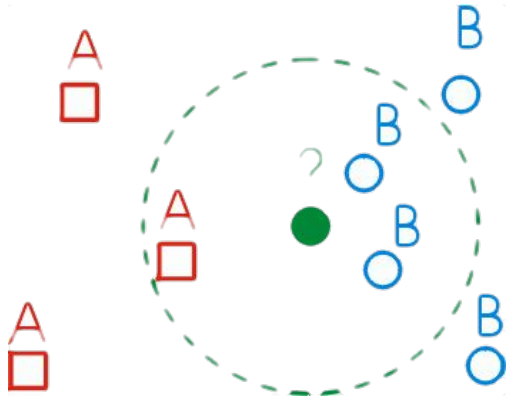
- Acciones que muestren interés (o desinterés) son registradas



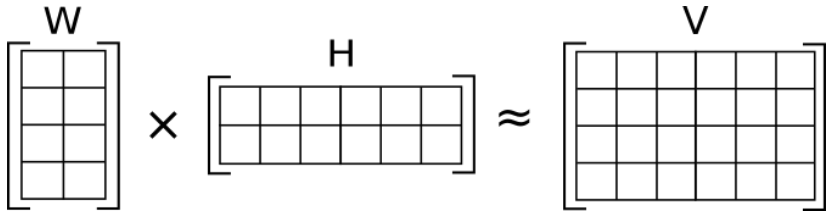


“Solucionado”

Técnicas



Vecinos más cercanos



Matrix factorization

Queremos minimizar esto

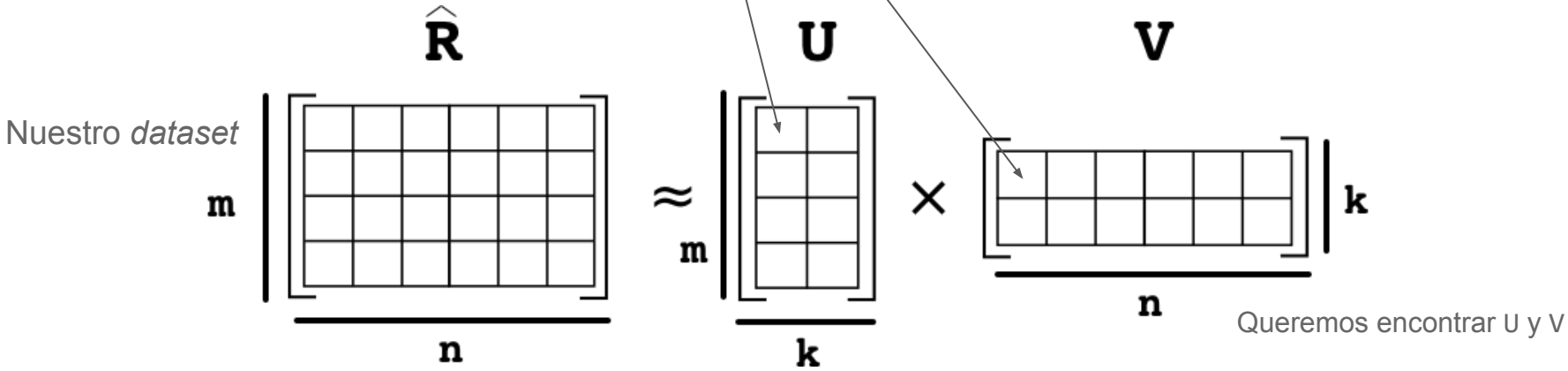
Rating explícito

Parámetro regulador

$$L(U,V) = \sum_{i,j} (r_{ij} - u_i v_j)^2 + \lambda(||U||_2^2 + ||V||_2^2)$$

row

column



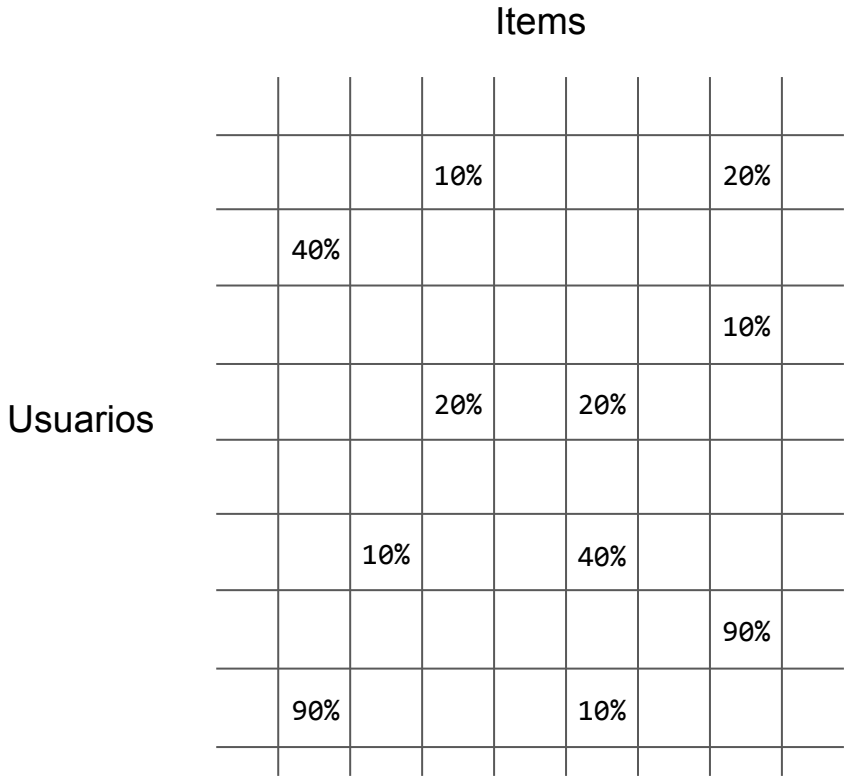
Queremos encontrar U y V

- ‘m’ usuarios
- ‘n’ items
- ‘k’ factores latentes (número definido por nosotros)
- Lambda: parámetro regulador (definido por nosotros)

¿Cómo integramos ***feedback*** **implícito**?

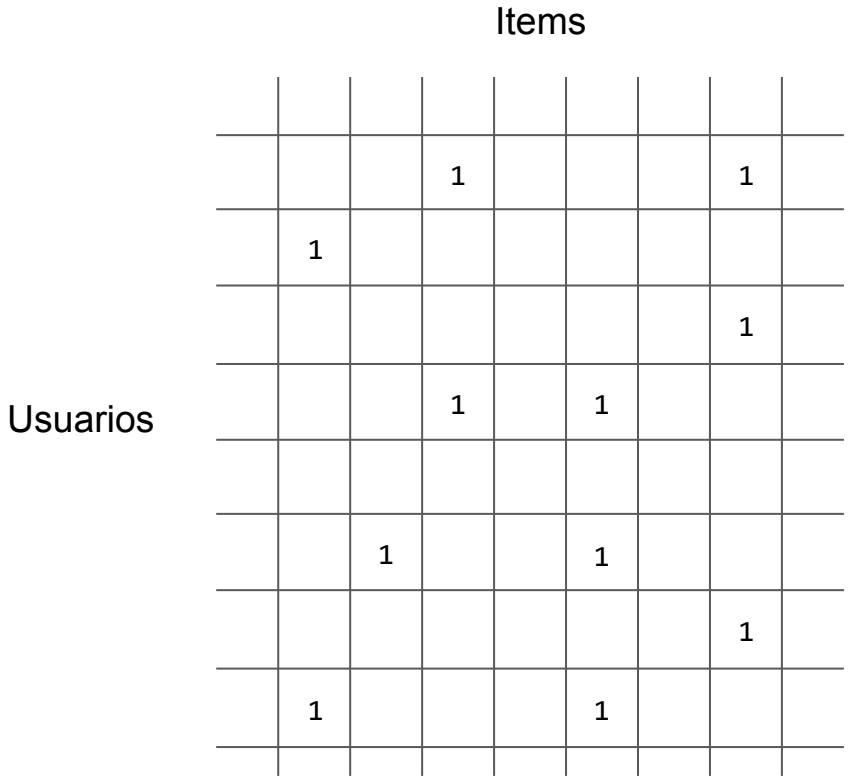
Matriz de señales: *C*

- `c_{i,j} ≥ 0`
  - La interacción implícita del usuario ‘i’ con el ítem ‘j’.



Matriz de activación: *P*

- $P_{i,j} \geq 0$
  - Indica si hubo interacción del usuario ‘i’ con el ítem ‘j’.



Peso: *w*

- Aquí se crea su propia función de peso respecto al valor de interacción implícita.

$$w_{ij} = f(c_{ij})$$

```
// TODO: Implement float f(int i, int j)
```

Peso

Interacción o no

Queremos minimizar esto

$$L(U,V) = \sum_{i,j} w_{ij}(p_{ij} - u_i v_j)^2 + \lambda(||U||_2^2 + ||V||_2^2)$$

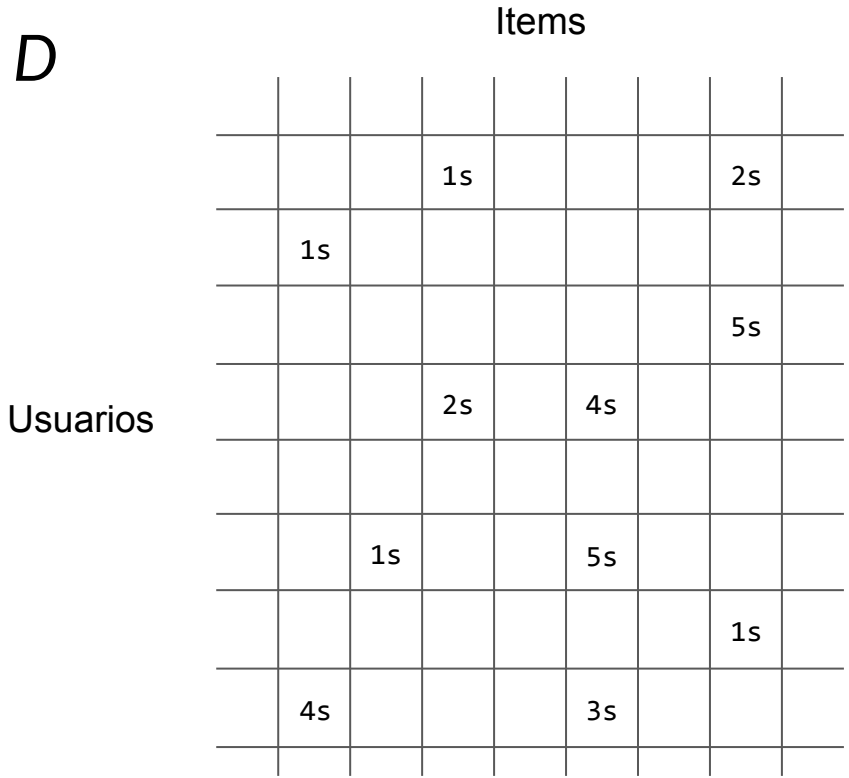
Complejidad: $O(k^2\omega + k^3(n+m))$

Cantidad de valores distintos a 0 en P

Feedback **negativo**

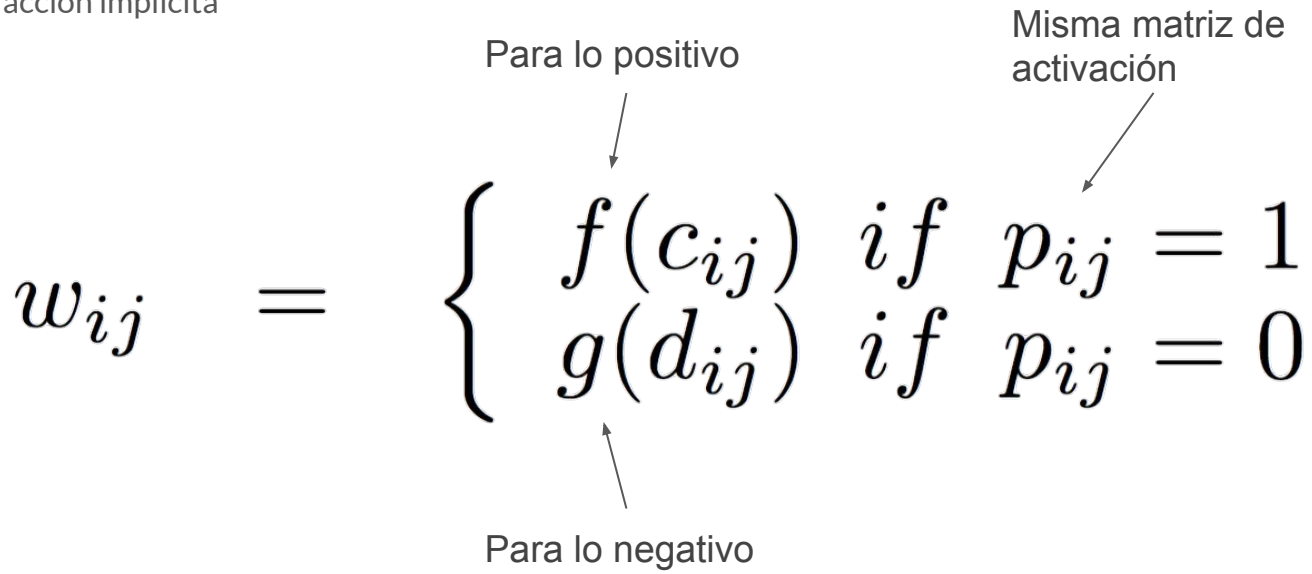
Matriz de señales negativas: *D*

- d_{i,j} ≥ 0
  - La interacción implícita negativa del usuario ‘i’ con el ítem ‘j’.



Peso: *w*

- Aquí se crea su propia función de peso respecto al valor de interacción implícita positiva y negativa.



Sampling-Based Learning

Sampling-Based Learning

Generamos varias matrices de P. Donde para cada celda de cada matriz (`gamma`):

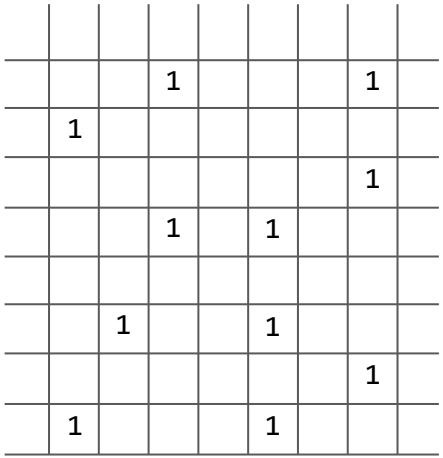
- Si `p_{i,j}` es 1:

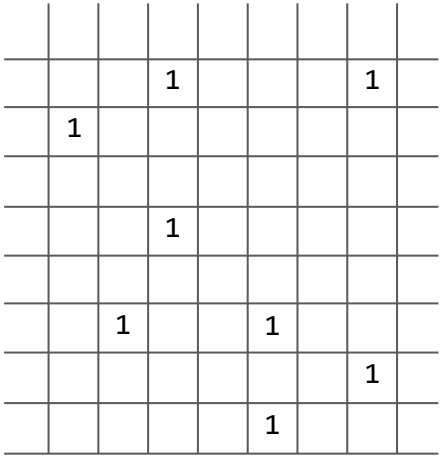
$$p_{ij}^{\gamma} = \begin{cases} 1 & \text{with probability } f(c_{ij}) \\ \text{Missing} & \text{with probability } 1 - f(c_{ij}) \end{cases}$$

- Si `p_{i,j}` es 0:

$$p_{ij}^{\gamma} = \begin{cases} 0 & \text{with probability } g(d_{ij}) \\ \text{Missing} & \text{with probability } 1 - g(d_{ij}) \end{cases}$$

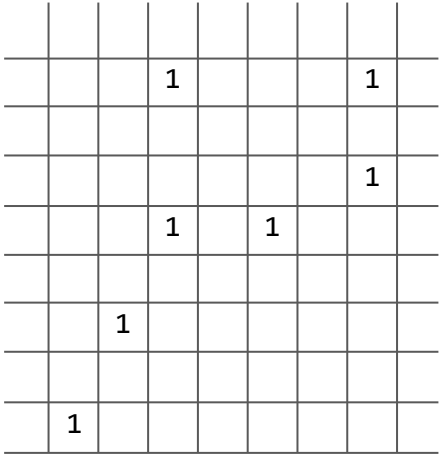
P





P^1

...



P^gamma

Omitido el feedback negativo para simplicidad

Sampling-Based Learning

- Corremos el mismo algoritmo para cada matriz P^gamma
- Tenemos una distribución para cada posible rating en la matriz de Usuarios x Items

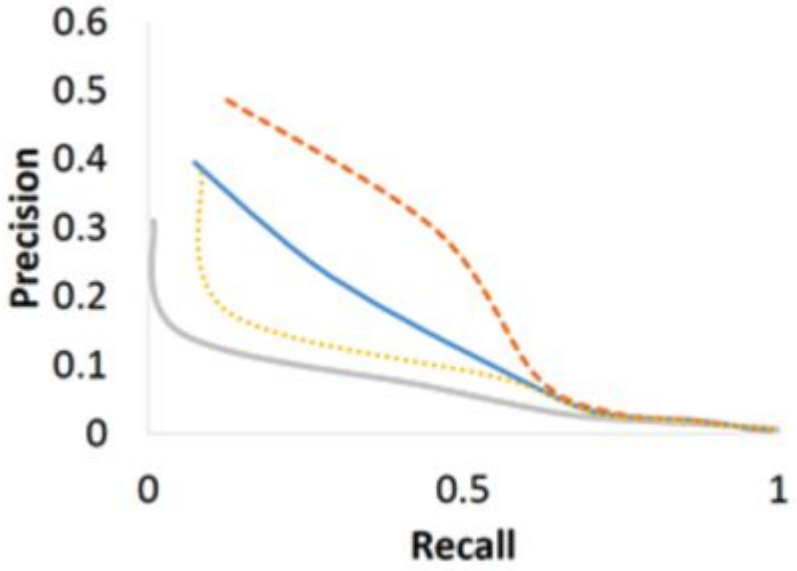
Experimento

- Recomendar tareas en un sistema *crowdsourcing*
- Si trabajadores hacen muchas *tasks*, es *feedback* positivo
- Tasks que aparecen harto pero son ignoradas, es *feedback* negativo.

Resultados MPR (menos es mejor)

	10000	20000	All
Global	45.693	27.711	39.055
Neighborhood	16.402	12.915	4.605
IFMF	11.482	8.790	3.351
IFMF2	10.528	8.803	3.365
IFMFS	9.928	6.267	2.189
IFMF2S	7.736	6.180	1.902
LB	0.0325	0.0216	0.0256

Resultados con pocos datos



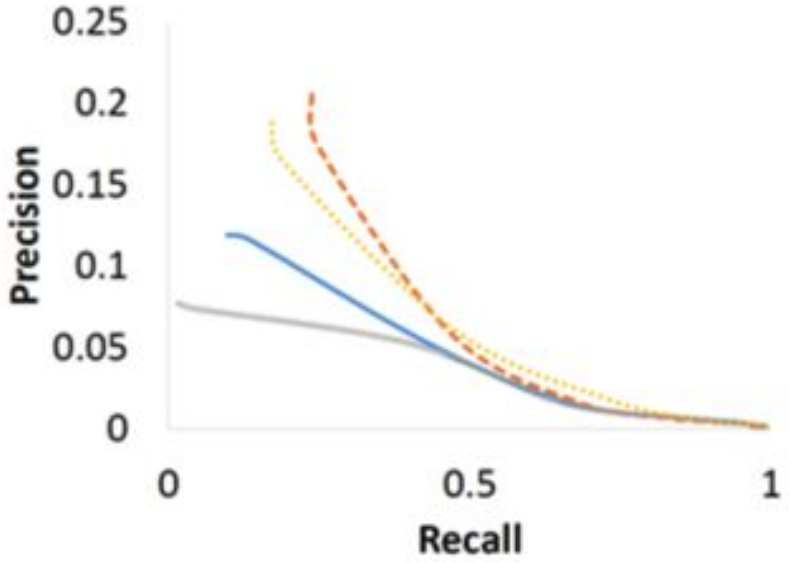
Con negativo

Con negativos y sampled

Solo positivos

Solo positivos y sampled

Resultados con muchos datos



Con negativo

Con negativos y sampled

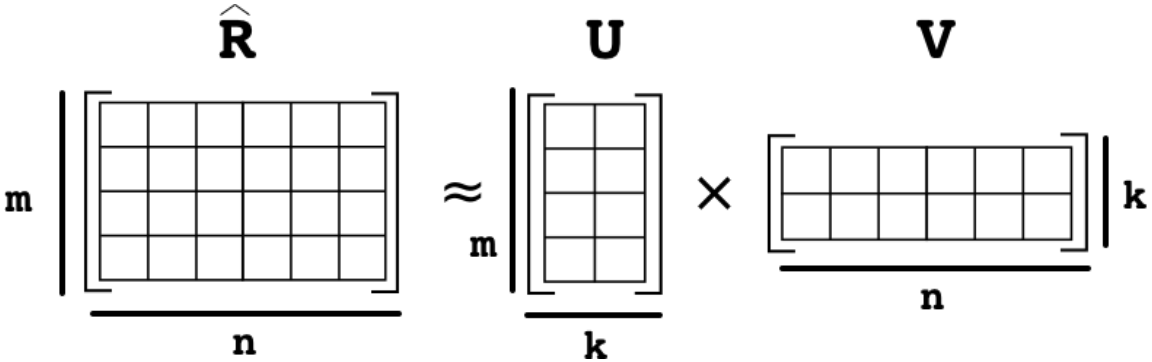
Solo positivos

Solo positivos y sampled

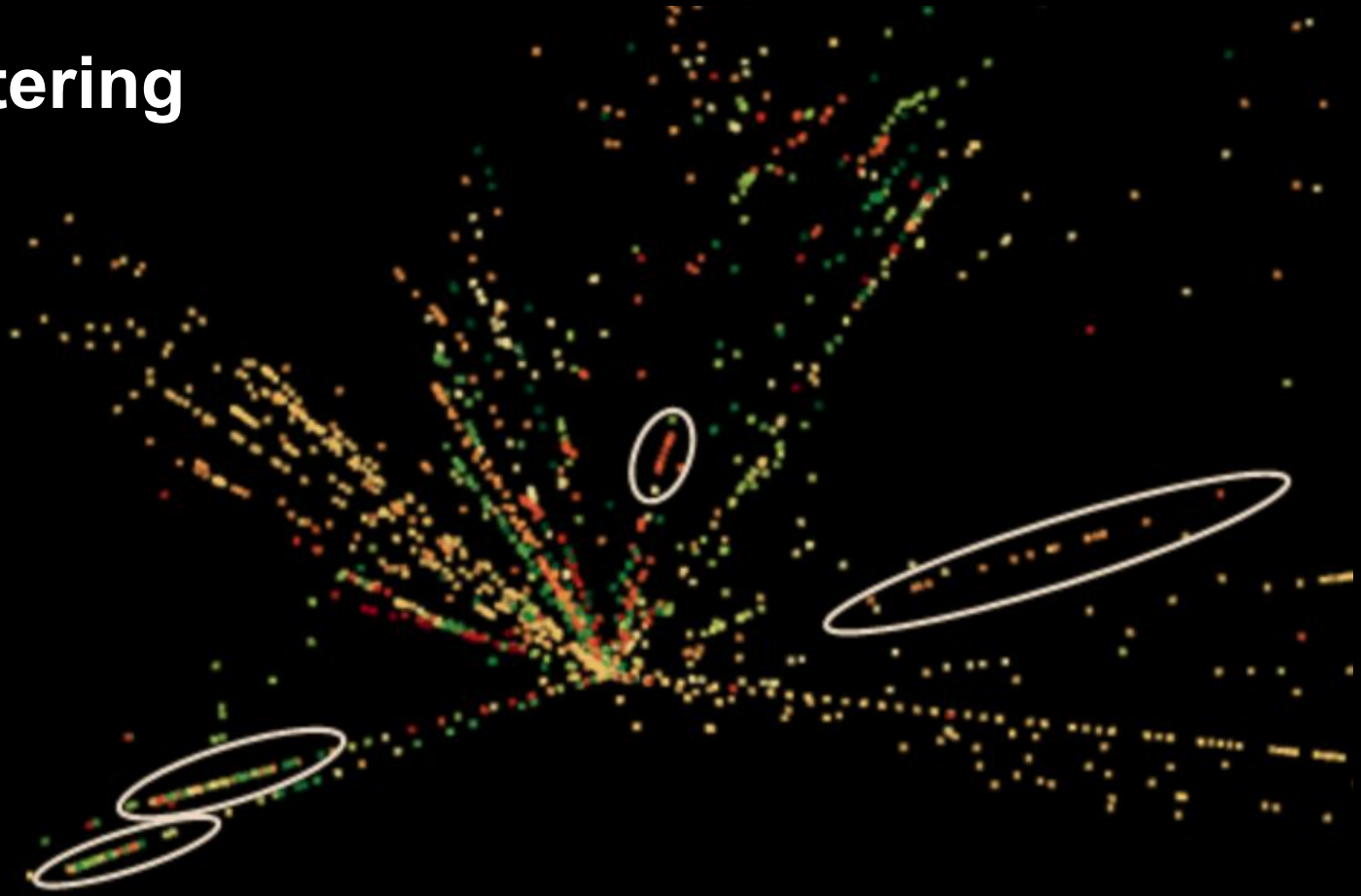
Otras ventajas de MF

Clustering

- Podemos utilizar U y V para hacer clusters
- Podemos usar **PCA** (*Principal component analysis*) para reducir aún más la dimensionalidad hasta 2 dimensiones



Clustering



¿Cómo partir aplicando esto?

Ejemplo:

Tiempo viendo una publicación en Facebook

Facebook captura el tiempo que le dedicas a cada publicación, pues da indicios sobre cierto interés tuyo sobre esta.

Podemos usar *Javascript* y *React.js* para hacer algo similar.

http://www.slate.com/blogs/future_tense/2015/06/16/facebook_s_new_attention_tracking_feature_is_super_creepy.html

feedback.js

```
import React, { Component, PropTypes } from 'react';
import ReactDOM from 'react-dom';
import VisibilitySensor from 'react-visibility-sensor';

export default class ViewFeedback extends Component {
  static propTypes = { onDisplayTimeChange: PropTypes.func }
  state = { mark: new Date() }

  onVisibilityChange = (visible) => {
    const now = new Date();
    // Notify on disappearance
    if (!visible && this.props.onDisplayTimeChange) {
      const diff = now - this.state.mark;
      this.props.onDisplayTimeChange(diff);
    }
    this.setState({ mark: now });
  }

  render = () => (
    <VisibilitySensor onChange={this.onVisibilityChange}>
      {this.props.children}
    </VisibilitySensor>
  )
}
```

Medimos el tiempo que se mantiene en la pantalla

```
<div>
  <ViewFeedback onDisplayTimeChange={time => /* Send to server */}>
    <article className="post">
      Hello world
    </article>
  </ViewFeedback>

  <ViewFeedback onDisplayTimeChange={time => /* Send to server */}>
    <article className="post">
      Hello world
    </article>
  </ViewFeedback>

  <ViewFeedback onDisplayTimeChange={time => /* Send to server */}>
    <article className="post">
      Hello world
    </article>
  </ViewFeedback>
</div>
```

Conclusiones e Interrogantes

- Podemos obtener más datos que con solo feedback explícito.
- El método de *sampling* puede pasar por alto recomendaciones *obvias*.

- ¿Qué pasó con el *feedback* explícito? ¿Cómo lo volvemos a incorporar?
- ¿Si tengo más fuentes de *feedback* implícito (positivo y/o negativo)?
  - ¿Debería bastar con modificar la función de peso o hay que darle un enfoque ‘híbrido’?.
- ¿Es correcto monitorear tanto al usuario sin su consentimiento?