Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

Matchbox: un modelo bayesiano de recomendaciones

Tesis de Licenciatura en Ciencias de la Computación

Macarena Piaggio

Directores: Esteban Mocskos y Gustavo Landfried

Buenos Aires, 2024

MATCHBOX: UN MODELO BAYESIANO DE RECOMENDACIONES

Los sistemas de recomendación son modelos probabilísticos que ordenan un conjunto de opciones en base a su impacto sobre un conjunto de objetivos. Mediante este ordenamiento se busca optimizar decisiones y acciones a tomar para intervenir en las áreas objetivo de manera eficaz. Por ejemplo, en identificación de personas desaparecidas, se desarrollan modelos basados en información del contexto forense que priorizan las muestras de ADN, buscando minimizar la cantidad de comparaciones requeridas para la identificación. En problemas de búsqueda espacial se desarrollan modelos basados en información del dominio; estos modelos ordenan los puntos de exploración para realizar la búsqueda en la menor cantidad de pasos posible. En redes sociales se utilizan algoritmos basados en el historial de actividades personal para ofrecer interacciones que mejoren la experiencia del usuario. Estos últimos han adquirido una gran relevancia en el modo en que interactuamos a través de internet, impactando profundamente la vida individual y social en general.

Una de las clases de modelos de recomendación de ítems más utilizadas son aquellos basados en la descomposición en valores singulares (SVD en inglés) de la matriz de puntajes que los usuarios asignan a los ítems (e.g. (3)). Debido al impacto que estos sistemas tienen en la vida de las personas es importante evaluar correctamente el espacio de hipótesis. Sin embargo, en muchos casos la inferencia no puede realizarse de forma exacta, siendo necesario utilizar aproximaciones. Las simulaciones por el método de Monte Carlo son una clase de métodos de aproximación que, si bien son de propósito general, pueden resultar impracticables debido a su excesivo costo computacional.

En el año 2009 Microsoft desarrolló un sistema de recomendación por factorización de matrices (de tipo SVD), *Matchbox* (25). Este es un sistema de recomendación de ítems que estima puntuaciones de usuario eficientemente mediante pasaje de mensajes variacionales (utilizando aproximaciones analíticas) entre las variables y las funciones del modelo probabilístico. La metodología usada garantiza que la distribución conjunta aproximada q minimice la divergencia Kullback-Leibler respecto a la solución por inferencia exacta p.

En este trabajo: documentamos de forma completa la matemática del algoritmo utilizado por Microsoft para los sistemas de recomendación de tipo SVD; realizamos una implementación propia del modelo probabilístico; y evaluamos la implementación oficial de Microsoft respecto a nuestra implementación y a otros modelos alternativos.

Palabras clave: sistemas de recomendación, inferencia bayesiana, inferencia probabilística aproximada, factor graphs, factorización de matrices, modelos de caja blanca

MATCHBOX: A BAYESIAN RECOMMENDER SYSTEM

Recommendation systems are probabilistic models that rank a number of options according to their impact on a set of goals. Ranking is used to optimize decisions and actions to be taken in order to effectively intervene in target areas. For example, to minimize the number of comparisons required to identify missing persons, models are developed to prioritize DNA samples based on information from the forensic context. In spatial search problems, models are used to order exploration points based on domain information to perform searches in as few steps as possible. In social networks, algorithms based on personal user activity are used to provide interactions that enhance user experience. The latter play a significant role in how we interact on the Internet, and have a profound impact on our individual and social lives.

For item recommendation, models that decompose the user-item rating matrix using Singular-Value Decomposition (SVD) or similar techniques have long been popular (e.g. (3)), These systems have an extraordinary impact on the lives of their users, so it is crucial to properly evaluate the hypothesis space. However, it's often not possible to perform exact inference, forcing us to resort to approximations. Monte Carlo simulations are general-purpose approximation methods that may be impractical due to their high computational costs.

In 2009 Microsoft developed a (SVD-based) matrix factorization recommender system, *Matchbox* (25). Matchbox is an item recommender that efficiently estimates user ratings through variational message passing (using analytic approximations) between variables and functions of the probabilistic model. The methodology guarantees that the joint approximate distribution q minimizes the Kullback-Leibler divergence with respect to its exact inference counterpart p.

In this work we fully document the methematics of the Matchbox algorithm; we develop our own implementation of the probabilistic model; and we evaluate the official Microsoft implementation against ours and other alternative models.

Keywords: recommendation systems, Bayesian inference, approximate probabilistic inference, factor graphs, matrix factorization, white-box models.

AGRADECIMIENTOS

Esta tesis no sería lo que es si no fuera por todos aquellos que me acompañaron en este largo camino: el LICAR, que me recibió con los brazos abiertos; la comunidad de Exactas, siempre alegre de compartirme su conocimiento; mi familia, cuyo apoyo incondicional me impulsó a seguir adelante; mis amigos, que me alentaron y aconsejaron cada vez que un experimento se rompió o que un objetivo cambió.

Un agradecimiento especial corresponde a mis directores de tesis, quienes me abrieron las puertas al mundo de la investigación. Es gracias a su dedicación y paciencia que hoy puedo presentar un trabajo del que estoy sumamente orgullosa. Gracias Gustavo, por acompañarme día a día y enseñarme todo lo que ahora sé sobre inferencia bayesiana a lo largo de tantísimas reuniones, conversaciones y mensajes. Gracias Esteban, por compartir tu sabiduría cuando mi entusiasmo por hacer más y falta de experiencia amenazaron con descarrilar este trabajo. Es un privilegio estar cerrando esta etapa junto a un laboratorio de semejante nivel.

Índice general

1.. Introducción 1
1.1. Objetivos 4
1.2. Aplicando las reglas de la probabilidad 4
1.2.1. Comparando modelos 7
1.2.2. Evitando overfitting 9
1.2.3. *Factor graphs* e inferencia por pasaje de mensajes 10
1.3. Métodos eficientes de aproximación 16
1.3.1. Caso de aprendizaje: Matchbox 17
1.3.2. Revisión de estado del arte 18

2.. Matchbox 21
2.1. Sistema de recomendación Matchbox 22
2.1.1. Caso *collaborative filtering* 23
2.1.2. Factor Graph 24
2.1.3. Propagación en el tiempo 28
2.2. Aproximaciones analíticas 30
2.2.1. Medidas de divergencia 31
2.2.2. Aproximación en Matchbox 33
2.3. Propiedades de las Gaussianas 34

3.. Implementación 37
3.1. Documentación del algoritmo 37
3.1.1. Mensajes descendentes 37
3.1.2. Mensajes ascendentes 42
3.1.3. Modelos de umbrales 46
3.2. Implementación computacional 48
3.3. Validación 48

4.. Comparativa 53
4.1. Datasets de puntuaciones 53
4.2. Notas sobre el cálculo de evidencia 53
4.3. Baseline: *Random Forest* y *Gradient Boosting* 54
4.3.1. Comparación sobre dataset sintético 55
4.3.2. Comparación sobre dataset de puntuaciones binarias 56
4.4. Comparación entre versiones de Matchbox 57
4.4.1. Notas sobre la evidencia en `test` en nuestra implementación 57
4.4.2. Comparación sobre dataset sintético 58
4.4.3. Comparación sobre MovieLens 59

5.. Conclusiones 61
5.1. Infer.NET 61
5.1.1. El tutorial de sistemas de recomendación 62
5.1.2. La clase `RecommenderSystem` 63
5.1.3. Elección de aproximaciones 64
5.2. Conclusiones finales 66
5.3. Trabajo a futuro 68

Referencias 69

Apéndice 71
5.4. General Message-Passing algorithm 71
5.5. Experimentos 72
5.5.1. Historial de puntuaciones 72
5.5.2. Espacios de hiperparámetros para comparaciones 73
5.5.3. Comparaciones sobre MovieLens 74
5.5.4. Comparaciones sobre dataset sintético 76

1. INTRODUCCIÓN

Desde el descubrimiento de las reglas de la probabilidad a finales del siglo XVIII, la teoría de la probabilidad es el enfoque más utilizado para razonar en contextos de incertidumbre. Si bien en todo este tiempo no se ha propuesto nada mejor en términos prácticos, el costo computacional asociado a la evaluación de todo el espacio de hipótesis ha limitado históricamente la aplicación estricta de las reglas de la probabilidad (o enfoque bayesiano). A fines del siglo XX, con el proceso de masificación de las computadoras personales, el enfoque bayesiano comenzó a ocupar el centro de atención de la inteligencia artificial, hasta que en el año 2012 se presenta una red neuronal convolucional profunda entrenada sobre GPU (17) que mueve el foco del área hacia ellas. Nuevamente, la complejidad computacional actuó como un límite a la aplicación estricta de las reglas de la probabilidad, que obligan a evaluar todo el espacio de hipótesis. Aún así, el auge de las redes neuronales profundas no produjo la aparición de un nuevo sistema de razonamiento bajo incertidumbre con mejor desempeño que el de las reglas de la probabilidad. En este sentido, el enfoque bayesiano sigue siendo el corazón de la inteligencia artificial y de toda ciencia basada en datos.

> Understanding how the assumptions in a machine learning model affect its behaviour is no longer just a useful skill for developing machine learning systems. It has become a critically important way of making sure that a machine learning system is transparent, interpretable and fair. As people's lives are influenced by machine-made decisions to an ever greater extent, the call to understand the reasoning behind these systems is going to become deafeningly loud. The assumptions in these systems need to be clear, transparent and available for all to see – and made accessible through clear explanations of each decision or prediction. Model-based machine learning is a crucial tool in ensuring that transparency and fairness lie at the foundations of all machine learning systems. (Winn, Model-Based Machine Learning)

En esta tesis nos proponemos estudiar algunos de los algoritmos eficientes de aproximación de la inferencia exacta en modelos causales o probabilísticos que no tienen solución analítica. En particular nos enfocaremos en sistemas de recomendaciones personalizadas. Los sistemas de recomendación han adquirido una gran relevancia en nuestro día a día: Google, Netflix, Amazon, Youtube, Twitter, Instagram ofrecen una incontable cantidad de películas, publicaciones, mensajes, videos, noticias de las que solo interactuaremos con una pequeña fracción en nuestras vidas. De manera casi transparente confiamos en recomendaciones para navegar el océano de contenido que Internet pone al alcance de nuestras manos. ¿Qué le gustará o interesará a un usuario y por qué? ¿Cómo podemos lidiar con un catálogo en constante expansión e intereses siempre en cambio? Se han propuesto multitud de respuestas a estas preguntas, cada una con sus propios objetivos e impacto en quienes

las reciben. Los sistemas de recomendación tienen gran influencia en nuestra vida cotidiana, por lo que además de querer soluciones precisas, flexibles y eficientes es importante que su comportamiento sea explicable.

Un hito en el desarrollo de los sistemas de recomendación fue la competencia *Netflix Prize* (2006-2009). Todos los años de su duración Netflix publicó un dataset anonimizado de puntuaciones de películas y ofreció un premio a quienes desarrollaran un algoritmo que superara significativamente las predicciones que la empresa podía obtener hasta aquel momento. De esta competencia surgieron varios algoritmos, entre ellos el ganador del Progress Prize de 2008 del concurso, SVD++ (15), que popularizaría las técnicas de factorización de matrices para sistemas de recomendación (16).

Cuando hablamos de Descomposición por Valores Singulares (SVD)[1] nos referimos a lo siguiente: dada una matriz $M \in \mathbb{R}^{\times\times\geqslant}$ existe una factorización $U\Sigma V$ tal que $U \in \mathbb{R}^{K\times n}$, $V \in \mathbb{R}^{K\times m}$, $\Sigma \in \mathbb{R}^{K\times K}$. En el contexto de recomendación, M será la matriz de puntajes que n usuarios asignan a m ítems. Las matrices U y V describirán a los n usuarios y m películas respectivamente (las columnas) mediante un conjunto de $K \in \mathbb{R}$ "características" latentes (las filas). La matriz Σ es una matriz diagonal que contiene los valores singulares, números que indican la importancia de cada una de estas características. Veremos en el desarrollo del trabajo que la predicción a posteriori (de aquí en más *posterior*) sobre las características latentes no tiene solución analítica, motivo por el cual las primeras implementaciones de esta clase de solución realizaban estimaciones puntuales sin modelar la incertidumbre asociada a realizar dichas estimaciones.

$$\mathbf{U} \in \mathbb{R}^{2\times 2} : \begin{pmatrix} 0.5 & 3 \\ -2 & 5 \end{pmatrix} \qquad \mathbf{V} \in \mathbb{R}^{2\times 3} : \begin{pmatrix} -1 & 0.3 & 2 \\ 1.2 & -5 & 4 \end{pmatrix}$$

$$\mathbf{\Sigma} \in \mathbb{R}^{2\times 2} : \begin{pmatrix} -0.5 & 0 \\ 0 & 3.6 \end{pmatrix}$$

Matrices de descomposición SVD para recomendación

Una de las primeras aproximaciones a la inferencia exacta (o bayesiana) en los modelos de recomendación de ítems basados en factorización de matrices fue el sistema desarrollado por Microsoft conocido como *Matchbox* (25), originalmente planteado para la recomendación de películas. Se puede decir que realiza una descomposición en valores singulares bayesiana, ya que parte de las mismas matrices U y V antes mencionadas pero modela a las características latentes como distribuciones de probabilidad Gaussianas, reflejando así la incerteza inherente a la estimación de estos factores. Podemos pensar en los factores de U como los gustos de usuario y V como los géneros de las películas. Opcionalmente, para generar mejores recomendaciones para nuevos usuarios (arranque "en frío") el modelo

[1] En este contexto, SVD no se refiere a la descomposición matemática clásica, si no a una aproximación de esta que fue popularizada en el contexto del Netflix Prize por la publicación de Simon Funk "Netflix Update: Try This at Home" (8). La solución ganadora del Progress Prize de 2008, SVD++, cita esta publicación como punto de partida.

puede tomar descripciones de los usuarios ($x \in \mathbb{R}^n$) y de las películas ($y \in \mathbb{R}^m$).

$$\mathbf{U} \in \mathbb{R}^{2\times 2} : \begin{pmatrix} \mathcal{N}(0,1) & \mathcal{N}(3,0.5) \\ \mathcal{N}(-2,0.1) & \mathcal{N}(5,2) \end{pmatrix} \qquad \boldsymbol{x} \in \mathbb{R}^2 = (1,0)$$

$$\mathbf{V} \in \mathbb{R}^{2\times 3} : \begin{pmatrix} \mathcal{N}(-1,2) & \mathcal{N}(0.3,1) & \mathcal{N}(2,0.9) \\ \mathcal{N}(1,2) & \mathcal{N}(-5,1.3) & \mathcal{N}(4,0.3) \end{pmatrix} \qquad \boldsymbol{y} \in \mathbb{R}^3 = (0,1,0)$$

$$z \ \in \mathbb{R}^2 = (\mathbf{U}\boldsymbol{x})^T \mathbf{V}\boldsymbol{y} = (\mathcal{N}(0.150, 0.707),\ \mathcal{N}(-2.018, 0.100))$$

Descomposición SVD versión Matchbox: U es la matriz de características latentes de usuario (gustos), V de película (géneros). Cuando no se cuenta con descripciones de usuario (o película) ($\boldsymbol{x} = e_i$ con i su identificador de usuario) los gustos latentes del usuario están dados por la columna correspondiente de la matriz U. Multiplicando las características latentes correspondientes al usuario y película, se obtiene la afinidad ($\boldsymbol{z}$) del par.

Matchbox realiza recomendaciones personalizadas ordenando las películas que el usuario aún no ha visto de acuerdo al puntaje que predice será asignado a cada una. Para poder realizar estas recomendaciones, el modelo se entrena a partir de triplas (usuario, película, puntaje numérico asignado) que llamaremos eventos de puntuación.

Matchbox puede ser representado en términos gráficos intuitivos, y computa la incertidumbre asociada a cada una de las variables haciendo que el modelo sea explicable y auditable, lo cual es de especial interés en el área de recomendación. El algoritmo ha sido reconocido como de buen funcionamiento en la literatura (20), pero la primer (y única a nuestro saber) implementación de Matchbox se hizo de código abierto en 2018 como parte del paquete de inferencia bayesiana y programación probabilística `Infer.NET` (21) de Microsoft.

Matchbox especifica matemáticamente la distribución conjunta del modelo probabilístico usando *factor graphs*. Estos son grafos bipartitos que contienen ejes entre distribuciones condicionales (nodos funciones) y los parámetros de esas distribuciones (nodos hipótesis o variables). Además de que los *factor graphs* permiten expresar visualmente las hipótesis del modelo probabilístico, existen algoritmos para realizar la inferencia de forma eficiente mediante pasaje de mensajes entre sus nodos. El *sum-product algorithm* es uno de tales algoritmos. Sus mensajes son la descomposición exacta de la aplicación estricta de las reglas de la probabilidad, lo que permite ejecutarlos en paralelo. Matchbox utiliza este algoritmo para todos los mensajes que pueden ser calculados de forma exacta. Lo interesante de Matchbox es la aplicación de algoritmos de pasaje de mensajes aproximados que garantizan la minimización de la divergencia Kullback-Leibler (de aquí en más divergencia KL) entre una familia de distribuciones (por ejemplo la gaussiana) y la distribución exacta. En particular Matchbox combina dos tipos de mensajes aproximados, *Expectation Propagation* cuando la distribución exacta intratable es unimodal y *Variational Message Passing* cuando es bimodal. Estos algoritmos de aproximación permiten resolver una gran variedad de problemas comunes a muchos modelos probabilísticos, como la suma, resta, multiplicación, división y truncamiento de variables. Este fue uno de los principales motivos por el que nos interesó Matchbox en nuestro estudio de sistemas probabilísticos.

1.1. Objetivos

El **objetivo general** de esta tesis es aprender sobre los algoritmos de aproximación eficiente de la inferencia probabilística exacta. Para aprender sobre estos algoritmos nos avocamos al estudio del sistema de recomendación bayesiano Matchbox, desarrollado por Microsoft en 2009 (25). Nos resultó de particular interés la manera en que resuelve la suma, resta, multiplicación y división de variables mediante una combinación de *Expectation Propagation* y *Variational Message Passing*, dos métodos de inferencia eficientes por pasaje descentralizado de mensajes entre los nodos de la red bayesiana. Estas operaciones son un problema común a otros modelos probabilísticos y la solución propuesta por Matchbox es generalizable.

Los **objetivos específicos** de esta tesis son:

1. *Documentar la matemática detrás de Matchbox*, tanto la inferencia aproximada como exacta que no se encuentran detalladas en el artículo original.
2. *Implementar el algoritmo*, dado que hoy en día existe una única versión disponible, escrita en `C#`, difícil de integrar a proyectos que involucran otros lenguajes de programación.
3. *Comparar Matchbox con otros sistemas de recomendación*, para contrastar el desempeño que tiene sobre datos reales respecto de otros modelos.
4. *Estudiar el funcionamiento de Matchbox a través de simulaciones*, de modo de ganar intuición del comportamiento que tiene algoritmo ante diferentes escenarios controlados.

1.2. Aplicando las reglas de la probabilidad

La teoría de la probabilidad puede resumirse en dos reglas fundamentales: la regla de la suma y la regla del producto. La *regla del producto* (o condicional), garantiza la coherencia de las creencias con la información disponible: preservamos la creencia previa que sigue siendo compatible con los nuevos datos (y la porción de creencia que sobrevive la volvemos a consider como nuestra creencia total).

$$P(X,Y) = P(Y|X)P(X) \qquad \text{(regla producto)}$$

La *regla de la suma* garantiza no perder creencia cuando la distribuimos entre hipótesis mutuamente contradictorias: las predicciones se realizan con la contribución de todas las hipótesis.

$$P(X) = \sum_Y P(X,Y) \underset{\text{(regla producto)}}{=} \sum_Y P(Y|X)P(X) \qquad \text{(regla suma)}$$

Por la simetría de la regla del producto ($P(X,Y) = P(Y,X)$) se sigue el *Teorema de Bayes*. Esto permite actualizar la probabilidad sobre una hipótesis H dado los datos D.

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} \qquad \text{(teorema Bayes)}$$

En general, las hipótesis H y los datos D forman parte de un modelo (o conjunto de restricciones) M particulares. Si quisiéramos hacer explícita esta dependencia deberíamos escribir el teorema de Bayes de la siguiente forma:

$$P(H|D,M) = \frac{P(D|H,M)P(H|M)}{P(D|M)}$$

Veamos al enfoque bayesiano en acción con un ejemplo similar al problema de Monty Hall. El siguiente ejemplo y el resto del marco teórico se basan en la sección introductoria de la tesis de Landfried (18) y en los primeros tres capítulos del libro de Bishop PRML (4):

Supongamos que tenemos tres cajas *exactamente* iguales y una persona esconde un regalo dentro de una de ellas. Una vez escondido, el regalo permanece en la misma caja, su posición no es aleatoria. ¿Dónde está el regalo? En este escenario tenemos tres hipótesis elementales r_i, donde cada una se interpreta como *el regalo se encuentra en la caja i*, siendo el conjunto de hipótesis elementales $H_R = \{r_1, r_2, r_3\}$. En la figura 1.1 representamos las cajas con cuadrados negros con la probabilidad $P(r_i)$ de cada una de las hipótesis elementales.

Si por algún motivo supiéramos que el regalo está en la caja del medio, podemos expresar certeza total asignando toda la probabilidad a la caja del medio. Esta situación se representa en la figura 1.1a en la que las hipótesis r_1 y r_3 tienen asignada probabilidad nula y la hipótesis r_2 tiene asignada probabilidad 1.

Sin certeza, en cambio, nos vemos obligados a dividir la probabilidad de alguna forma. Si elegimos alguna de las cajas estaríamos afirmando más de lo que sabemos porque en los hechos no tenemos información que nos haga preferir ninguna de ellas. Esto podemos expresarlo dividiendo la creencia en partes iguales $P(r_1) = P(r_2) = P(r_3)$.

a Incertidumbre nula *b* Incertidumbre máxima

Fig. 1.1: Posibles distribuciones de creencias

En términos epistemológicos, para no mentir en contextos de incertidumbre debemos no afirmar más de lo que se sabe (maximizando incertidumbre o entropía) sin dejar de decir todo lo que sí se sabe (incorporando toda la información disponible). Esto se corresponde con la distribución de creencias de la figura 1.1b, que maximiza la incertidumbre al dividir la probabilidad en partes iguales, e incorpora la información de que el regalo está en alguna de las tres cajas al no asignar creencia por fuera de ellas.

Ahora, complejicemos levemente el modelo. Supongamos que se nos pide elegir una caja y que, una vez que elegimos, quien guardó el regalo nos muestra el interior de una de las cajas que **no** contiene el regalo. Se incorporan, entonces, las hipótesis elementales c_i que corresponde a *"elegimos la caja i"* y s_i a *"muestran el interior de la caja i"*. Luego,

nuestro experimento $\mathcal{H}$ consiste de tres hipótesis, $\mathcal{H} = \{H_R, H_C, H_S\}$, cada una con tres hipótesis elementales, $H_R = \{r_1, r_2, r_3\}$, $H_C = \{c_1, c_2, c_3\}$ y $H_S = \{s_1, s_2, s_3\}$.

A esta información inicial la llamaremos Modelo A, o simplemente M_A. Necesitamos entonces la distribución de probabilidad $P(H_R, H_C, H_S|M_A)$. Para simplificar el problema, y sin perder generalidad, podemos suponer que siempre elegimos la caja 1, es decir, que es cierta c_1. De esta manera, reducimos el problema a solo dos hipótesis: $P(H_R, H_S|c_1, M_A)$. Debido a que no tenemos ninguna otra restricción adicional, si el regalo se encontrara en la caja 1, es decir que es cierta r_2, no hay preferencia por ninguna de las hipótesis elementales s_1 o s_3 (y lo mismo ocurre cuando son ciertas r_1 o r_3).

Nada nos obliga a suponer que el Modelo A es correcto. Podemos suponer que la persona que muestra la caja no nos mostrará tampoco el interior de la caja que seleccionamos, por lo que si es cierta c_i, no puede ocurrir s_i, que se muestre el interior de esa misma caja. Esto plantea una restricción (o dependencia) adicional. Llamaremos Modelo B (M_B) al sistema de restricciones que incluye esta restricción además todas las restricciones del modelo A. Este modelo tiene asociado la distribución de probabilidad $P(H_R, H_S|c_1, M_B)$.

Bajo el sistema de restricciones M_B, dada c_1, si el regalo se encontrara en la caja 1, es decir si r_1 fuera cierta, no tenemos información que nos haga preferir ninguna de las hipótesis elementales s_2 o s_3, por lo tanto $P(s_2|r_1, c_1, M_B) = P(s_3|r_1, c_1, M_B) = 1/2$. En cambio, si el regalo se encontrara en una caja distinta a la que elegimos (i.e., $r_i \,/\, i \neq 1$), entonces por la restricción preexistente $s_j \,/\, j \neq i$ (sea r_i la caja elegida) sólo se puede mostrar el interior de la caja que no contiene el regalo ni fue elegida ($s_j \,/\, j \neq i \wedge j \neq 1$).

$P(s|r, c_1, M_A) =$

	r_1	r_2	r_3
s_1	0	1/2	1/2
s_2	1/2	0	1/2
s_3	1/2	1/2	0

$P(s|r, c_1, M_B) =$

	r_1	r_2	r_3
s_1	0	0	0
s_2	1/2	0	1
s_3	1/2	1	0

Notar que estamos expresando 3 distribuciones de probabilidad, una por cada posible valor de la posición del regalo. Podemos usar la regla del producto para calcular la distribución de probabilidad conjunta cuando c está fijo, dado que $P(s_i, r_j|c_1) = P(s_i|r_j, c_1)\, P(r_j)$. La regla de la suma establece que cualquier creencia marginal puede ser obtenida integrando la creencia conjunta. Es por esto que la probabilidad conjunta es sumamente importante, ya que podemos responder cualquier pregunta relacionada al problema a partir de ella y la aplicación de las reglas de la probabilidad.

$P(s, r|c_1, M_A) =$

	r_1	r_2	r_3	$P(s_j\|c_1, M_A)$
s_1	0	1/6	1/6	1/3
s_2	1/6	0	1/6	1/3
s_3	1/6	1/6	0	1/3
$P(r_i\|c_1, M_A)$	1/3	1/3	1/3	1

$P(s, r|c_1, M_B) =$

	r_1	r_2	r_3	$P(s_j\|c_1, M_B)$
s_1	0	0	0	0
s_2	1/6	0	1/3	1/2
s_3	1/6	1/3	0	1/2
$P(r_i\|c_1, M_B)$	1/3	1/3	1/3	1

Probabilidad conjunta y marginales para M_A y M_B.

Veamos cómo se actualiza nuestra creencia sobre la posición del regalo en cada momento del problema de Monty Hall.

Al comienzo del juego el regalo puede estar dentro de cualquiera de las tres cajas, $P(r_i|M_k) = 1/3 \,\forall\, i \in \{1,2,3\}$. Elegir una caja no modifica esta probabilidad, por lo que $P(r_i|c_1, M_k) = P(r_i|M_k)$. Esto se ve reflejado en la última fila de las tablas 1.2.

Si, por ejemplo, como pista se señalara y mostrara el interior de la segunda caja (s_2), la segunda fila de las tablas 1.2 nos indicará la probabilidad de que el regalo esté en cada caja y se señale la segunda caja (dado que se eligió la primer caja). Esta es la creencia que "sobrevive" a la observación del valor de s. Notemos que esta creencia no suma 1 porque es una creencia conjunta; una vez que observamos que efectivamente ocurrió s_2 debemos normalizarla, es decir dividir por la evidencia, para obtener nuestra nueva creencia total sobre la posición del regalo.

Nuestra nueva creencia respecto a la posición del regalo está dada por $P(r_i|s_2, c_1)$; aplicando el Teorema de Bayes podemos obtener el normalizador que nos falta y calcular la distribución de probabilidad del regalo luego de haber recibido la pista:

$$\underbrace{P(r_i|s_2, c_1, M_k)}_{\text{Nueva creencia}} = \frac{\overbrace{P(s_2, r_i|c_1, M_k)}^{\text{Creencia que sobrevive}}}{\underbrace{P(s_2|c_1, M_k)}_{\substack{\text{Creencia total que sobrevive} \\ \text{(normalizador)}}}}$$

$$= \frac{\overbrace{P(s_2|r_i, c_1, M_k)}^{\text{Verosimilitud}}\overbrace{P(r_i|c_1, M_k)}^{\text{Prior}}}{\underbrace{P(s_2|c_1, M_k)}_{\text{Evidencia}}}$$

$P(s_2, r|c_1, M_A) =$

	r_1	r_2	r_3	$P(s_2\|c_1, M_A)$
s_2	1/6	0	1/6	1/3

$P(s_2, r|c_1, M_B) =$

	r_1	r_2	r_3	$P(s_2\|c_1, M_B)$
s_2	1/6	0	1/3	1/2

Entonces, nuestras nuevas creencias sobre la posición del regalo para cada modelo resultan:

$P(r_i|s_2, c_1, M_A) =$

	r_1	r_2	r_3
s_2	1/2	0	1/2

$P(r_i|s_2, c_1, M_B) =$

	r_1	r_2	r_3
s_2	1/3	0	2/3

1.2.1. Comparando modelos

En lugar de elegir un modelo arbitrariamente, podemos evaluar ambos mediante la aplicación de las reglas de la probabilidad, agregando la hipótesis $H_M = \{M_A, M_B\}$ a nuestro experimento (es decir, ahora $\mathcal{H} = \{H_R, H_S, c_1, H_M\}$).

Para evaluar los distintos modelos querremos comparar la probabilidad de cada uno dados los datos con los que contamos: $P(M_k|\text{Datos})$ - el posterior del modelo. Para calcularlo podemos aplicar estrictamente las reglas de probabilidad, siguiendo el enfoque bayesiano. Conceptualmente, utilizaremos las probabilidades para representar cuanta certeza tenemos

respecto a la elección de cada modelo.

$$\underbrace{P(M_k|\text{Datos})}_{Posterior} = \frac{\overbrace{P(M_k)}^{Prior}\overbrace{P(\text{Datos}|M_k)}^{Evidencia}}{P(\text{Datos})}$$

La creencia a priori del modelo (de aquí en más *prior*), $P(M_k)$, expresa nuestra preferencia inicial por cada modelo. Al igual que con los priors de las cajas en el ejemplo de la sección 1.2, como no tenemos información que nos haga preferir alguno de los modelos a priori, dividimos la creencia en partes iguales para cada modelo.

La evidencia, $P(\text{datos}|M_k)$, es la predicción que el modelo realizó "a priori" (**sin** haber observado los datos) de los datos que finalmente fueron observados. Podemos pensarlo como *"la preferencia que expresan los datos por cada modelo"* ((4) sección 3.4), o como *"la creencia que sobrevive en cada modelo al observar el dato"*. En el ejemplo basado en Monthy Hall de la sección 1.2, podemos ver su efecto cuando se nos da la pista: como el regalo no puede estar tras la puerta señalada por la pista, perdemos toda creencia en el caso en que el regalo está detrás de esa puerta. $P(s_i|r_i) = 0$, no hay evidencia que apoye esa hipótesis.

Cuando los priors de ambos modelos son iguales, la comparación de los posterior de dos modelos resulta $\frac{P(M_A|\text{Datos})}{P(M_B|\text{Datos})} = \frac{P(\text{Datos}|M_A)}{P(\text{Datos}|M_B)}$. Es decir, basta con comparar las evidencias de los distintos modelos. Esta expresión se conoce como el *Bayes factor* y se suele expresar en escala logarítmica para reportar la diferencia en órdenes de magnitud.

Estrictamente, la evidencia es equivalente a que el modelo observe los datos uno a uno y calcule la probabilidad de los datos aún no observados en cada paso, del siguiente modo:

$$\begin{aligned} P(\text{datos} = \{d_1, d_2, d_3, ...\}|M_k) &= P(d_1|M_k)P(d_2|d_1, M_k)P(d_3|d_1, d_2, M_k)... \\ &= \prod_{i=1}^{numObs} P(d_i|\ d_1, ..., d_{i-1}, M_k) \end{aligned} \tag{1.1}$$

y, si nos interesa su logaritmo:

$$log(P(\text{datos} = \{d_1, d_2, d_3, ...\}|M_k)) = log(\prod_{i=1}^{numObs} P(d_i|\ d_1, ..., d_{i-1}, M_k))$$

$$\sum_{i=1}^{numObs} log(P(d_i|\ d_1, ..., d_{i-1}, M_k))$$

En (18) se muestra experimentalmente que, generando datos usando el Modelo B, a medida que se observan datos nuevos la evidencia de este crece y la del Modelo A decrece. El Modelo B respeta las restricciones del Modelo A, así que ninguno de los modelos observará eventos imposibles, que generarían un 0 en la productoria 1.1.

Esta misma metodología se utilizará para las demás comparaciones de modelos de este trabajo. También reportaremos la media geométrica de las predicciones de los datos observados, que puede interpretarse como la probabilidad del modelo de predecir acertadamente

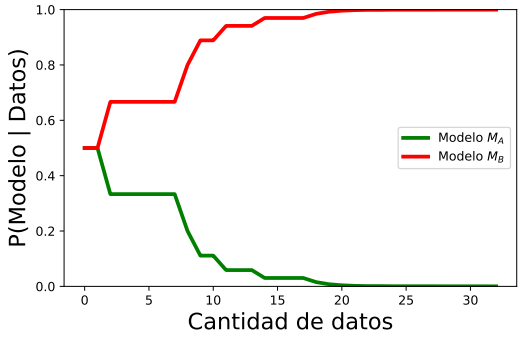


Fig. 1.2: Distribución de probabilidad a posteriori sobre los sistemas de restricciones (o modelos) alternativos M_A y M_B dado los datos.

el dato a observar. Dado que la evidencia es la productoria de estas probabilidades, esta métrica resulta:

$$\begin{aligned}
\text{Media geométrica(predicciones)} &= \left(\prod_{i=1}^{numObs} P(d_i |\ d_1, ..., d_{i-1}, model) \right)^{\frac{1}{numObs}} \\
&= (\text{evidencia})^{\frac{1}{numObs}} \\
&\underset{x\ \equiv\ exp(log(x))}{=} exp\left(log\left((\text{evidencia})^{\frac{1}{numObs}} \right) \right) \\
&\underset{log(x^k)\ \equiv\ k\ log(x)}{=} exp\left(\frac{\log(\text{evidencia})}{numObs} \right)
\end{aligned}$$

1.2.2. Evitando overfitting

Una gran ventaja de la selección de modelo bayesiana por sobre la selección de modelo de máxima verosimilitud es que no produce overfitting. Veamoslo con un ejemplo clásico de selección de modelo lineal, presente tanto en el capítulo 3 del libro de Bishop (4) como en la sección 1.3.3 de la Tesis de Landfried (18).

Supongamos que tenemos una muestra de datos de tamaño N generado a partir de una función sinoidal con un ruido aleatorio que tiene desvío estándar β y media nula.

$$p(y_i|x_i, \beta) = \mathcal{N}(y_i\,|\,\text{sen}(x_i), \beta^2) \qquad i \in \{0, \ldots, N-1\}$$

Supongamos además que tenemos 10 modelos lineales alternativos M_D, con $D \in \{0, \ldots, 9\}$, y queremos hallar aquel que más se ajusta a los datos. Definimos $M_D : y(\boldsymbol{x}, \boldsymbol{w}) = \sum_{d=0}^{D} w_d x_d^d$ al modelo lineal de grado D con $x_i \in \mathbb{R}$ constantes cuyo valor conocemos con exactitud al momento de observar cada dato de la muestra.

¿Cuál de todos los modelos lineales "ajusta" mejor los datos? Los modelos lineales de grado 0 a 2 no tienen la flexibilidad suficiente para "ajustar" (o acercarse) a la sinoidal en el rango $x_i \in [-\pi, \pi]$, mientras que todos los modelos de grado mayor o igual a grado

3 sí tienen la flexibilidad suficiente para "ajustar" (o acercarse) a la sinoidal en el rango $x_i \in [-\pi, \pi]$.

La selección de modelo por máxima verosimimilitud selecciona el modelo que maximiza la función de costo

$$\max_{\boldsymbol{w}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}, \beta, M_D)$$

y a partir de este predice nuevos datos minimizando

$$p(y_N|x_N, \beta, M_D) \approx p(y_N|x_N, \underbrace{\max_{\boldsymbol{w}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}, \beta, M_D)}_{\substack{\text{Hipótesis elementales } \boldsymbol{w} \text{ que} \\ \text{que maximizan la función} \\ \text{de costo}}}, \beta, M_D) \tag{1.2}$$

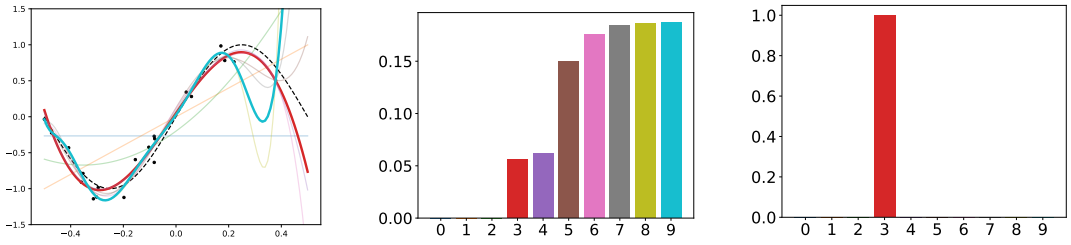
En la figura 1.3b se grafican los valores de verosimilitud para cada modelo, y en la figura 1.3a se ven los ajustes de todos los modelos. Destacamos los modelos de grado 3 (línea roja) y de grado 9 (línea celeste). Se ve que al aumentar la complejidad de los modelos aumenta su flexibilidad y, por lo tanto, aumenta su verosimilitud. El modelo de grado 9 es el que pasa por más puntos de la muestra, sin embargo no se ajusta bien a la función que generó los datos (la sinoidal). A diferencia de la selección de modelo bayesiana, la selección por máxima verosimilitud tiende a preferir modelos innecesariamente complejos.

En lugar de seleccionar el modelo de máxima verosimilitud, podríamos realizar una selección de modelo mediante la aplicación de las reglas de la probabilidad, como vimos en la sección 1.2.1. Vamos a observar los datos de la muestra de a uno, **aproximando en cada paso** el nuevo prior de los modelos usando la ecuación de predicción 1.2 sobre los datos vistos hasta el momento. Calculamos la evidencia final usando la ecuación (1.1), y la graficamos en la figura 1.3c. Podemos ver que una vez observados todos los datos se le asigna probabilidad 1 al modelo de grado 3, lo cual tampoco es ideal: si la evidencia de un modelo es 0, seguirá siendolo para cualquier dato que vea en el futuro también, ya que la fórmula de la evidencia es una productoria. Aún si en el futuro se observara un dato fuera del rango $x_i \in [-\pi, \pi]$, se anuló la posibilidad de seleccionar un modelo más complejo, por lo que se sigue sobreajustando el modelo a los datos de entrenamiento (*overfitting*).

Ahora bien, estos modelos lineales tienen una solución analítica, por lo que **no necesitamos aproximar las predicciones** que hacen los modelos en cada paso. Calculamos entonces los ajustes y evidencia exactos de cada modelo para nuestra muestra y los graficamos en las figuras 1.4a y 1.4b. Hay dos puntos a destacar: los modelos de grado 1 y 2, cuya expresividad es insuficiente para adaptarse a una sinoidal, tienen probabilidad 0 (la evidencia desmiente estos modelos). Sin embargo, la probabilidad de los modelos complejos es baja **pero no nula**, dejando abierta la posibilidad de que sean la mejor elección para datos nuevos, especialmente si el modelo de grado 3 tampoco tuviera la expresividad necesaria para adaptarse a nuevos datos (evitamos que se produzca *overfitting*).

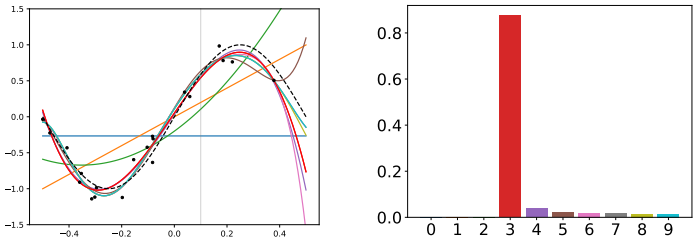
1.2.3. *Factor graphs* e inferencia por pasaje de mensajes

En el enfoque de aprendizaje automático basado en modelos se crean modelos a medida del problema, generalmente utilizando métodos gráficos que lo describan. Estas descripciones



a Ajuste de modelos *b* Máxima verosimilitud *c* Validación en línea

Fig. 1.3: Modelos lineal de grado $D = 0$ a $D = 9$ ajustados por máxima verosimilitud a los datos. En a se grafica la muestra con línea punteada, y los ajustes obtenidos por máxima verosimilitud de los modelos. En b se muestra el valor de máxima verosimilitud obtenido por cada uno de los modelos. En c se muestra la probabilidad de los modelos aproximando cada una de las predicciones individuales a priori mediante la ecuación ad hoc (1.2), usando como dato de entrenamiento en cada paso solo los datos observados previamente.



a Ajuste bayesiano *b* Posterior de modelos

Fig. 1.4: Modelos lineal de grado $D = 0$ a $D = 9$ ajustados mediante la aplicación estricta de las reglas de la probabilidad a los datos. En a se grafica la sinoidal con línea punteada, y los ajustes obtenidos usando la solución analítica. En b se muestra la probabilidad de los modelos calculada mediante la aplicación estricta de las reglas de la probabilidad.

gráficas son la especificación matemática de la probabilidad conjunta, y nuestro objetivo es actualizarla luego de ver nuevos datos. Generalmente la actualización no tiene solución analítica. Sin embargo, actualmente existen lenguajes de programación probabilística que se encargan de resolverlo ya sea con algún método de aproximación por sampleo o mediante aproximaciones analíticas, como es el caso del paquete Infer.NET (21).

Matchbox utiliza el enfoque basado en modelos. En particular, especifica la probabilidad conjunta mediante *factor graph*. Cualquier distribución de probabilidad conjunta puede factorizarse como producto de distribuciones de probabilidad condicional. Un *factor graph* es un grafo bipartito que vincula nodos factores (esas distribuciones de probabilidad condicional) con las variables que son argumento de esos factores.

Representemos el Modelo B del ejemplo de Monty Hall de la subsección 1.2 usando un *factor graph*. Lo primero que vamos a querer hacer es factorizar la probabilidad conjunta $P(r, c, s)$ de alguna manera que nos sea útil y tenga sentido para nuestro problema. Ya vimos que "dónde se coloca el regalo" (H_r) y "qué caja elegimos inicialmente" (H_c) son decisiones que se toman sin ninguna información previa y sin ninguna dependencia a otros

factores del problema. Por otro lado, la elección de caja señalada (H_s) depende de dónde está el regalo y, en el caso del modelo B, de cuál fue la caja elegida inicialmente. Teniendo todo esto en cuenta, una factorización que deja expresada toda esta información es la siguiente:

$$P(r,c,s) \underset{\text{(regla producto)}}{=} \underbrace{P(s|r,c)}_{f_s(s,r,c)} \underbrace{P(r)}_{f_r(r)} \underbrace{P(c)}_{f_c(c)}$$

Los *factor graphs* representan cada variable de la probabilidad conjunta con un círculo y cada factor de la descomposición con un cuadrado. Se colocan ejes conectando a los factores con aquellas variables que forman parte de sus argumentos. Vemos en la figura 1.5 como resulta la representación gráfica de esta factorización.

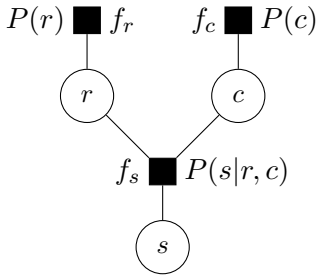


Fig. 1.5: Factor graph de Monty Hall

El *sum-product algorithm* es una forma general de descomponer las reglas de la probabilidad como mensajes que se envían los nodos del *factor graph* entre sí para calcular cualquier marginal de la distribución conjunta. Estos mensajes codifican la mínima cantidad de pasos que se requieren para calcular cualquier distribución de probabilidad marginal.

Citando a Landfried (18):

> Hay dos tipos de mensajes: los mensajes que envían los nodos variables a sus funciones vecinas ($m_{v\to f}(v)$); y los mensajes que envían los nodos funciones a sus variables vecinas ($m_{f\to v}(v)$). El primero codifica una porción de la regla del producto.
>
> $$m_{v\to f}(v) = \prod_{h\in n(v)\setminus\{f\}} m_{h\to v}(v) \qquad \text{(paso del producto)}$$
>
> donde $n(v)$ representa el conjunto de vecinos del nodo v. Es decir, el mensaje que envía una variable v a un factor f es simplemente la multiplicación de los mensajes que recibe del resto de sus vecinos $h \in n(v)$ salvo f. Los mensajes que envían los nodos funciones codifican una parte de la regla de la suma.
>
> $$m_{f\to v}(v) = \int \cdots \int \left(f(\boldsymbol{h}, v) \prod_{h\in n(f)\setminus\{v\}} m_{h\to f}(h) \right) d\boldsymbol{h} \qquad \text{(paso de la suma)}$$
>
> donde $\boldsymbol{h} = n(f)\setminus\{v\}$ es el conjunto de todos los vecinos de f salvo v, y $f(\boldsymbol{h}, v)$ representa la función f, evaluada en todos sus argumentos. En pocas palabras,

el mensaje que envía una función f a una variable v también es la multiplicación de los mensajes que recibe del resto de sus vecinos $h \in n(f)$ salvo v, incluyendo en el producto el factor mismo $f(\cdot)$, integrando (o sumando en el caso discreto) sobre los vecinos $\boldsymbol{h}$. Finalmente, la distribución de probabilidad marginal de una variable v es simplemente la multiplicación de los mensajes que v recibe de todos sus vecinos.

$$p(v) = \prod_{h \in n(v)} m_{h \to v} \qquad \text{(probabilidad marginal)}$$

Veamos entonces cuales son los mensajes que se envían en el *factor graph* del ejemplo de Monty Hall:

Mensajes que "bajan"

$$\begin{aligned}
m_{f_r \to r}(r) &= m_{r \to f_s}(r) = P(r) \\
m_{f_c \to c}(c) &= m_{c \to f_s}(c) = P(c) \\
m_{f_s \to s}(s) &= \sum_{r,\,c} f_s \, m_{r \to f_s}(r) \, m_{c \to f_s}(c) \\
&= \sum_{r,\,c} P(s|r,c) P(r) P(c) \\
&= \sum_{r,\,c} P(s,r,c) \\
&= P(s)
\end{aligned}$$

Mensajes que "suben"

$$\begin{aligned}
m_{s \to f_s}(s) &= 1 \\
m_{f_s \to c}(c) &= \sum_{r,\,s} f_s \, m_{r \to f_s}(r) \, m_{s \to f_s}(s) \\
&= \sum_{r,\,s} P(s|r,c) P(r) 1 \\
&= \sum_{r} P(r) \underbrace{\sum_{s} P(s|r,c)}_{1} \\
&= \sum_{r} P(r) = 1 \\
m_{f_s \to r}(r) &\underset{\text{(análogo)}}{=} 1
\end{aligned}$$

Podemos comprobar, entonces, que las marginales de las variables se obtienen multiplicando los mensajes entrantes a cada uno de sus nodos:

$$\begin{aligned}
P(c) &= \underbrace{m_{f_c \to c}(c)}_{P(c)} \underbrace{m_{f_s \to c}(c)}_{1} \\
P(r) &= \underbrace{m_{f_r \to r}(r)}_{P(r)} \underbrace{m_{f_s \to r}(r)}_{1} \\
P(s) &= \underbrace{m_{f_s \to s}(s)}_{P(s)}
\end{aligned}$$

Para el funcionamiento de Matchbox nos interesará calcular las probabilidades actualizadas luego de observar el puntaje real del par (usuario, película) que estábamos evaluando. Una vez que observemos el puntaje, podremos calcular la "sorpresa" o creencia que sobrevive a esta observación, y usarla para actualizar los priors de las características latentes que estamos estimando.

Para entender como lo hace Matchbox, veamos como quedan expresadas en el *factor graph*

nuestras creencias sobre la posición del regalo en cada momento del problema de Monty Hall.

Al comienzo del juego el regalo puede estar dentro de cualquiera de las tres cajas, $P(r_i) = 1/3 \, \forall i \in \{1, 2, 3\}$. Este, el prior de r, queda guardado en el primer factor que envía mensaje a r (primer mensaje que baja).

Una vez que se escondió el regalo nos toca elegir una caja. Como ya vimos, la posición del regalo y la elección de la caja son eventos independientes, por lo que no hay relación entre estas variables por ahora, y $P(r_i|c_1) = P(r_i)$.

Para calcular nuestra creencia sobre la posición del regalo luego de ver cuál caja es señalada vamos a querer modelar a s como variable observable. En verdad, para modelar una variable como observable basta con tomar los mensajes del *factor graph* y dejar de integrar por la variable observable. Pero también podemos dejar expresado que la variable es observable en el *factor graph*, agregando un factor f_{obs} que represente el realizar la observación, y asignandole la función $\mathbb{I}(s = s^*)$ donde s^* indica el valor de la observación. Dado que la función tiene a s de variable, f_{obs} va a estar conectado al nodo s.

Además, en lugar de fijar c en c_1, modelemosla también como variable observable. Agregamos un factor $\mathbb{I}(c = c^*)$ conectado a la variable c. En la figura 1.6 podemos ver el *factor graph* resultante.

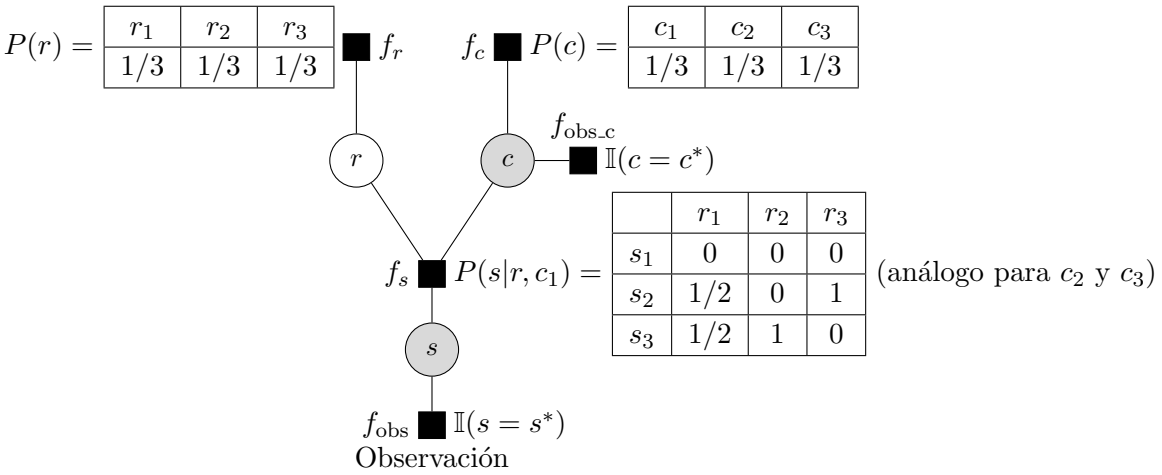


Fig. 1.6: Factor graph de Monty Hall con variables observadas

Para calcular la probabilidad de que el regalo esté en la i-ésima caja, que se elija la primer caja y se señale la segunda caja como pista, podemos usar nuestro *factor graph* y el algoritmo de suma producto. Veamos como quedan los mensajes:

Mensajes que "bajan"

$$
\begin{aligned}
m_{f_r \to r}(r) &= m_{r \to f_s}(r) = P(r) \\
m_{f_c \to c}(c) &= P(c) \\
m_{f_{\text{obs\_c}} \to c}(c) &= \mathbb{I}(c = c^*) \\
m_{c \to f_s}(c) &= P(c)\mathbb{I}(c = c^*) \\
m_{f_s \to s}(s) &= \sum_{r,\,c} f_s \, m_{r \to f_s}(r) \, m_{c \to f_s}(c) \\
&= \sum_{r,\,c} P(s|r,c)P(r)P(c)\mathbb{I}(c = c^*) \\
&\overset{*}{=} \sum_{r,\,c} P(s|r,c)P(r|c)P(c)\mathbb{I}(c = c^*) \\
&\underset{\text{(regla producto)}}{=} \sum_{r,\,c} P(s,r,c)\mathbb{I}(c = c^*) \\
&\underset{\text{(regla suma)}}{=} P(s,c^*) \\
&= P(s,c)\mathbb{I}(c = c^*)
\end{aligned}
$$

$*$: $P(r) = P(r|c^*)$ por independencia.

Mensajes que "suben"

$$
\begin{aligned}
m_{s \to f_s}(s) &= f_{\text{obs}} = \mathbb{I}(s = s^*) \\
m_{f_s \to r}(r) &= \sum_{c,\,s} f_s \, m_{c \to f_s} \, m_{s \to f_s} \\
&= \sum_{c,\,s} P(s|r,c)\mathbb{I}(s = s^*)P(c)\mathbb{I}(c = c^*) \\
&\underset{\text{(Función identificadora)}}{=} P(s_2|r,c^*)P(c_1) \\
m_{f_s \to c}(c) &= \sum_{r,\,s} f_s \, m_{r \to f_s} \, m_{s \to f_s} \\
&= \sum_{r,\,s} P(s|r,c)P(r)\mathbb{I}(s = s^*) \\
&= P(s_2|c)
\end{aligned}
$$

El producto de los mensajes que le llegan a r nos darán la marginal de r cuando s y c son observables.

$$
\begin{aligned}
m_{f_r \to r}(r) m_{f_s \to r}(r) &= P(r)P(s^*|r,c^*)P(s^*,c^*) \\
&\overset{*}{=} P(r|c^*)P(s^*|r,c^*)P(s^*,c^*) \\
&\underset{\text{(regla producto)}}{=} P(s^*,r|c^*)P(s^*,c^*)
\end{aligned}
$$

$*$: $P(r) = P(r|c^*)$ por independencia.

$P(s^*, r, c^*, M_A)$

	r_1	r_2	r_3	$P(s^*\|c^*, M_A)$
s_1	0	1/18	1/18	1/9
s_2	1/18	0	1/18	1/9
s_3	1/18	1/18	0	1/9

$P(s^*, r, c^*, M_B)$

	r_1	r_2	r_3	$P(s^*\|c^*, M_B)$
s_1	0	0	0	0
s_2	1/18	0	1/9	1/6
s_3	1/18	1/9	0	1/6

Probabilidad marginal de r con s y c observables. Se muestran las probabilidades para $c = c_1$, análogo para c_2 y c_3.

Llegamos así a la marginal de r. Si, por ejemplo, observamos que $c = c_1$ y $s = s_2$, la creencia que sobrevive será la segunda fila de las tablas mostradas. Una vez que observamos los valores de c y s podemos normalizar (dividir por $P(s_2, c_1|M_k)$) para obtener nuestra nueva creencia sobre la posición del regalo, que será la misma que calculamos anteriormente.

1.3. Métodos eficientes de aproximación

El obstáculo histórico de la aplicación estricta de las reglas de la probabilidad ha sido el costo computacional asociado a la evaluación completa de las hipótesis de investigación $\mathcal{H} = \{H_1, H_2, \ldots, H_N\}$. En el caso más simple, con hipótesis de investigación binarias $|H_i| = 2$, la cantidad de combinaciones a evaluar, $P(h_1, h_2, \ldots h_N)$, crece de forma exponencial como 2^N. No solo hay que asignar probabilidades iniciales a cada una de esas combinaciones de hipótesis elementales; cuando observamos alguna de las hipótesis elementales, se debe incorporar la información que transmite a través del grado de sorpresa (o predicción posterior) a cada una de la 2^{N-1} combinaciones de hipótesis elementales no observadas. En casos más complejos, que involucran además preguntas (o hipótesis) de investigación continuas ni siquiera es posible aplicar la regla de la suma cuando sus integrales no tienen solución analítica.

Para afrontar estas situaciones, en las que nos vemos impedidos de calcular la probabilidad posterior mediante la aplicación estricta de las reglas de la probabilidad (es decir, no nos es posible realizar inferencia exacta), han surgido numerosos métodos eficientes de aproximar esta probabilidad:

Aproximaciones por sampleo *(Cadenas de Markov)* Se construye una cadena de Markov que tiene la distribución posterior como su distribución estacionaria. *Hamiltonian Monte Carlo*, por ejemplo, utiliza el gradiente para definir guiar la evolución de la cadena de Markov, permitiendo saltos más largos y eficientes en el espacio de parámetros. *Sequential Monte Carlo*, por su parte, representa la distribución posterior mediante un conjunto de partículas, cada una con su peso asociado, y se actualiza iterativamente a medida que se reciben nuevas observaciones.

Ventajas En teoría permiten aproximar cualquier distribución de probabilidad con alta precision.

Desventajas Suelen ser computacionalmente costosas.

Aproximaciones analíticas Las métodos analíticos suelen aproximar la distribución de probabilidad p, que es difícil de calcular, con una distribución más fácil de manejar q. Se busca la distribución q que mejor se ajusta a la distribución p. Esta búsqueda se realiza en base a diferentes tipos de divergencia, lo que determina el comportamiento de la aproximación. Por ejemplo *Expectation Propagation* minimiza la divergencia $\mathrm{KL}(p||q)$, mientras que *Variational Inference* minimiza la divergencia $\mathrm{KL}(q||p)$.

Ventajas Son computacionalmente más eficientes que los métodos de muestreo.

Desventajas Si la familia q es muy chica (por ejemplo la familia de las Gaussianas), las aproximaciones pueden ser muy sesgadas.

Otras aproximaciones eficientes. Existen muchas otras aproximaciones eficientes. En particular, destacamos la reciente publicación de *Stein Variational Gradient Descent* (SVGD) (19), en la cual se aproxima cualquier distribución de probabilidad con alta precisión de forma muy eficiente mediante el transporte de un conjunto finito de

partículas, de manera que converjan hacia la distribución objetivo (minimizando la divergencia de Stein entre la distribución de referencia y la distribución objetivo).

En esta tesis estudiaremos los métodos de aproximación analíticos *Expectation Propagation* y *Variational Inference*, sobre los cuales entraremos en detalle en la sección 2.2.

1.3.1. Caso de aprendizaje: Matchbox

Como ya presentamos, Matchbox es un sistema de recomendación de ítems, originalmente presentado para realizar predicciones personalizadas de los puntajes que usuarios darán a películas. Es un algoritmo de factorización de matrices que utiliza las técnicas de pasaje de mensajes aproximados *Expectation Propagation* y *Variational Message Passing*. Todos los mensajes se aproximan mediante distribuciones Gaussianas, las cuales tienen propiedades de interés que veremos más adelante. Al guardar las matrices de características U y V en lugar de la matriz de puntuaciones M, se consigue una representación de los datos más compacta, que escala en función de la cantidad de usuarios y películas en lugar de en función de la cantidad de puntuaciones.

La multiplicación y división de variables, centrales para esta implementación, frecuentemente generan problemas en modelos probabilísticos, y puntualmente en Matchbox aparecen multiplicaciones de variables cuyo resultado es caro de computar (lo cual impactará tanto al mensaje puntual como todos aquellos que dependan de él). Ante esta situación, Matchbox propone combinar el uso generalizado de *Expectation Propagation* con un uso localizado de *Variational Message Passage* para aproximar las distribuciones bimodales (resultantes de la multiplicación y división de variables), ya que la aproximación de *Expectation Propagation* no convergería nunca para este caso (veremos por qué en la sección 2.2.1). Esta solución es generalizable y es utilizada por trabajos posteriores que se encuentran con la misma situación, entre ellos algunos que mencionaremos en la próxima sección.

Enumeramos los aportes del trabajo estudiado, incluyendo los que aún no hemos mencionado:

1. Resuelve el problema común de "arranque en frío" (*cold-start problem*) incorporando metadata de usuarios e ítems para hacer buenas recomendaciones a usuarios nuevos, transicionando a personalización a partir de puntuaciones para los usuarios más antiguos.

2. Realiza inferencia eficiente mediante una innovadora combinación de Variational Message Passing (VMP) y Expectation Propagation (EP).

3. Permite modelar dinámicamente cambios en la popularidad de los ítems, los gustos de los usuarios y el sistema de puntuación propio del usuario en el tiempo.

4. Es capaz de adaptarse a distintos modelos de puntuación. Puntualmente, presenta tres alternativas: observación de puntuación absoluta (escala contínua), preferencia binaria (me gusta/no me gusta) y preferencia ordinal en una escala específica al

usuario, mediante umbrales personalizados.

5. Es capaz de utilizar Assumed-Density Filtering (ADF) para realizar entrenamiento on-line, incorporando datos incrementalmente para permitir reflejar inmediatamente las preferencias de usuario actualizadas (no exploraremos este punto en el presente trabajo).

1.3.2. Revisión de estado del arte

Existen varios trabajos de interés que retoman estas técnicas y extienden Matchbox, mostrando la adaptabilidad de las innovaciones propuestas a diversos problemas. GroupBox (28), de 2015, realiza recomendaciones personalizadas para grupos encadenando el modelo Matchbox dentro de un sistema que modela dinámicas entre usuarios. Christakopoulou (5), de 2016, construye un sistema de recomendación "conversacional" en el que el modelo toma el rol de interlocutor con el usuario, estimando tanto las preferencias de usuario latentes (usando la representación de factores latentes de Matchbox) como las preguntas que le conviene realizar para alcanzar una buena recomendación rápidamente dado el contexto. Kasneci (12), de 2011, da herramientas probabilísticas para construir y mantener bases de conocimiento de gran escala generadas a partir de conocimiento extraído de internet. Las herramientas que propone permiten modelar las incertidumbres que suelen surgir durante la confección de estas bases de datos; ya sea por las técnicas de extracción empleadas, por la incerteza inherente a ciertos datos (por ejemplo, no se sabe exactamente cuando nació Platón), o por el desconocimiento que se tiene de cuán confiables son las fuentes de las que proviene la información en internet. El paper utiliza Matchbox para modelar la pericia de los usuarios respecto a la información que aportan.

Además de Matchbox, los mismos autores, junto a otros en Microsoft Research, presentaron otros dos modelos de predicción mediante modelado probabilístico en la misma época: TrueSkillThroughTime en 2007 (6) y AdPredictor en 2010 (9). Ambos utilizan modelos gráficos casi iguales para distintas aplicaciones. TrueSkillThroughTime se encuentra en uso hoy como sistema de predicción de habilidad y emparejamiento para Xbox Network (anteriormente Xbox LIVE), la plataforma de juegos multijugador de Microsoft que cuenta con más de 120 millones de usuarios. A nuestro saber, AdPredictor sigue en uso en Bing para realizar la selección de advertising y precios estimando *Click-through rates* (CTR). El estudio de TrueSkillThroughTime nos resultó invaluable para poder comprender el funcionamiento y la matemática de Matchbox.

Matchbox fue utilizado en *Project Emporia*, un portal lanzado en 2010 que utilizaba los datos públicos del perfil de twitter del usuario para generar recomendaciones personalizadas de noticias. No hay mucha información disponible de primera mano sobre *Project Emporia*; el proyecto ya no se encuentra disponible y parece haber tenido una corta vida. Sin embargo, en su momento esta iniciativa era de suma vanguardia; hoy en día todas las grandes redes sociales utilizan algoritmos de recomendación para ordenar sus *"timelines"*, pero en ese momento sólo facebook contaba con una iniciativa de este estilo, con *EdgeRank* (2007), un sistema de ordenamiento rudimentario basado en inter-

acciones, decay y preferencia del usuario por cierto tipo de publicación. No fue hasta 2011 que facebook comenzó a usar Machine Learning para ordenar el feed de noticias; en el caso de Twitter, ocurrió recién en 2016. En el minuto 2:42 del siguiente video subido por Microsoft Research se puede ver una explicación y demostración del proyecto: https://www.youtube.com/watch?v=ku8iIG6iHMc.

Al día de hoy, Matchbox está disponible para su uso en el portal de Machine Learning de Microsoft https://studio.azureml.net/, y es el único sistema de recomendación basado en modelos allí que permite incorporar metadata de usuarios e ítems. Además, desde 2012 se utiliza la metodología de modelado de características latentes de Matchbox en el sistema de recomendación de la tienda de juegos de Xbox Network (14).

2. MATCHBOX

Los sistemas de recomendación se suelen caracterizar según qué información utilizan para seleccionar el contenido a recomendar. Existen dos enfoques principales para realizar este filtrado, a saber:

- *Content-Based Filtering*, en que se generan predicciones analizando metadata sobre los ítems a recomendar y estimando características de cada usuario a partir de sus puntuaciones anteriores. Algunos ejemplos de algoritmos que usan esta técnica son los árboles de decisiones (más adelante hablaremos de *Random Forest* (10)), clasificadores bayesianos (7) y redes neuronales (4).
- *Collaborative Filtering*, en que no se cuenta con metadata, por lo que se observan las preferencias de los usuarios por ciertos ítems y se agrupan usuarios con intereses similares en "vecindarios", recomendandoles ítems que otros miembros del vecindario hayan puntuado altamente. Durante el Netflix Prize se popularizaron las técnicas de factorización de matrices para realizar *collaborative filtering*.

Matchbox combina ambos enfoques, por lo que se considera que realiza *hybrid filtering*. Cuando un usuario es nuevo, las estimaciones de características latentes serán malas, por lo que Matchbox utiliza la metadata del usuario para hacerle recomendaciones (*content-based filtering*), solucionando el problema del arranque "en frío" (*cold-start problem*). A medida que este puntua más ítems, el algoritmo podrá refinar sus estimaciones y realizar recomendaciones personalizadas basandose principalmente en la información obtenida de las puntuaciones (*collaborative filtering*). El uso de metadata de usuarios/ítems es opcional; en caso de no contar con ella, Matchbox realiza solo *collaborative filtering*.

Se puede observar este modelo híbrido en acción en la demostración de *Project Emporia*: de acuerdo al presentador, cuando se iniciaba la aplicación por primera vez esta "revisa si sabe suficiente de mi para realizarme recomendaciones personalizadas en arranque". En caso de encontrar estos datos, los utilizaba como descriptores; en caso de que no, se puede ver que *Project Emporia* recomendaba de los ítems más populares, lo cual se alinea con el uso de *collaborative filtering* para un usuario nuevo. Por descripciones que encontramos del proyecto, que mencionan el uso de datos públicos de Facebook y Twitter, sospechamos que la búsqueda de información del usuario revisaba si era posible scrapear datos de las redes sociales instaladas en el teléfono o informadas a la aplicación de alguna otra manera. Sin embargo, no hallamos información pública para confirmar esta sospecha.

En la literatura de sistemas de recomendación, se suele representar a los usuarios e ítems mediante vectores de atributos (*features*) observables, asociandole a cada atributo un vector de características (*traits*) latentes. En particular, un sistema de recomendación tiene un vector de atributos observables $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\Phi}, r)$, donde
$\boldsymbol{x} \in \mathbb{R}^n$*:* vector de atributos (descripción) de una persona.

$\boldsymbol{y} \in \mathbb{R}^m$: vector de atributos (descripción) de un ítem.

$\boldsymbol{\Phi} \in \mathbb{R}^f$: vector de atributos del contexto.

$r \in \mathbb{R}$: puntaje que la persona da al ítem (rating).

Estos son los datos de entrada con los que contamos para entrenar y testear nuestro modelo.

2.1. Sistema de recomendación Matchbox

Matchbox modela las variables observables mediante un vector de características latentes $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{w})$ con,

$\boldsymbol{s} = \boldsymbol{U}\boldsymbol{x} \in \mathbb{R}^k$: gustos de las personas, con $\boldsymbol{U} \in \mathbb{R}^{k \times n}$ las características de usuario (gustos) latentes.

$\boldsymbol{t} = \boldsymbol{V}\boldsymbol{y} \in \mathbb{R}^k$: géneros de los ítems, con $\boldsymbol{V} \in \mathbb{R}^{k \times m}$ las características de ítem (géneros) latentes.

$b = \boldsymbol{\Phi}^T \boldsymbol{w} = \sum_f \Phi_f \, w_f \in \mathbb{R}$: sesgo de contexto, con $\boldsymbol{w} \in \mathbb{R}^f$ los pesos latentes.

La predicción de la puntuación r se modela como,

$$p(r|\boldsymbol{s}, \boldsymbol{t}, b) = \mathcal{N}(r|\boldsymbol{s}^T \boldsymbol{t} + b, \beta^2) \tag{2.1}$$

con β el desvío estándar del ruido observacional, un parámetro del modelo. Entenderemos como afinidad (nombrada z) a la puntuación estimada antes de agregar el sesgo y el ruido, $\boldsymbol{s}^T \boldsymbol{t}$. Se obtiene:

$$\boldsymbol{z} = \boldsymbol{s}^T \boldsymbol{t} = (\boldsymbol{U}\boldsymbol{x})^T \boldsymbol{V}\boldsymbol{y} = \sum_k \underbrace{\sum_{i=1}^{n} u_{ki} x_i \sum_{j=1}^{m} v_{kj} y_j}_{z_k} \tag{2.2}$$

Para realizar recomendaciones personalizadas, el modelo actualiza las distribuciones de creencias de las variables latentes $(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{w})$ a medida que recibimos datos.

$$p(\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{w}|\text{Datos}, \text{Modelo}) \tag{2.3}$$

Estas variables latentes se pueden descomponer en variables elementales,

$$\boldsymbol{U} \in \mathbb{R}^{K \times n},\ \boldsymbol{U} = \begin{pmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & & \vdots \\ u_{k1} & \cdots & u_{kn} \end{pmatrix} \qquad \boldsymbol{V} \in \mathbb{R}^{K \times m},\ \boldsymbol{V} = \begin{pmatrix} v_{11} & \cdots & v_{1m} \\ \vdots & & \vdots \\ v_{k1} & \cdots & v_{km} \end{pmatrix} \tag{2.4}$$

$$\boldsymbol{w} \in \mathbb{R}^f = (w_1,\ \ldots, w_f)$$

La distribución de creencias a priori de cada una de estas variables se representan mediante una distribución gaussiana.

$$p(u_{ki}) = \mathcal{N}(u_{ki}|\mu_{U_{ki}}, \sigma^2_{U_{ki}}) \qquad p(v_{kj}) = \mathcal{N}(v_{kj}|\mu_{V_{kj}}, \sigma^2_{V_{kj}})$$
$$p(w_d) = \mathcal{N}(w_d|\mu_{w_d}, \sigma_{w_d})$$

Por defecto, Matchbox elige $\mathcal{N}(0,1)$ como prior de las variables latentes, lo que supone que el verdadero valor puede ser tanto positivo como negativo. A posteriori, si a un usuario le gustaran las películas de cierto género, esperaríamos que la variable latente correspondiente fuera una gaussiana con media positiva y varianza chica, indicando mucha certeza.

El modelo considera que cada uno de los elementos de $\boldsymbol{U}$, $\boldsymbol{V}$ y $\boldsymbol{w}$ son independientes entre sí, por lo tanto:

$$p(\boldsymbol{U}) = \prod_k \prod_i \mathcal{N}(u_{ki} | \mu_{u_{ki}}, \sigma^2_{u_{ki}}) \tag{2.5}$$

$p(\boldsymbol{V})$ y $p(\boldsymbol{w})$ se obtienen de manera análoga.

Por otra parte, frecuentemente la descripción del contexto es una combinación de la descripción del usuario y del ítem. Cuando es así, resulta:

$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{e}_i \\ \boldsymbol{e}_j \end{pmatrix}$*:* siendo un vector de tamaño $f = n + m$

$\boldsymbol{w} = \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}$*:* usando $\mathbf{u}$ y $\mathbf{v}$ como renombres de $\boldsymbol{w}$

$b = \boldsymbol{\Phi}^T \boldsymbol{w} = \mathrm{u}_i + \mathrm{v}_j = w_i + w_{n+j}$**:** con u_i y w_i el i-ésimo elemento de $\mathbf{u}$ y $\boldsymbol{w}$

Durante el resto del trabajo consideraremos que este es el caso.

La distribución conjunta de todas las variables del modelo, $p(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{U}, \boldsymbol{V}, \boldsymbol{w}, \boldsymbol{z}, \tilde{r}, r \mid \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\Phi})$ (donde nombramos r a la puntuación con sesgo y $\tilde{r}$ a la puntuación con sesgo y ruido observacional) puede ser factorizada del siguiente modo:

$$p(r|\tilde{r})p(\tilde{r}|\boldsymbol{z}, b) \underbrace{p(b|\boldsymbol{w}, \boldsymbol{\Phi})}_{\text{prior sesgo}} \underbrace{p(\boldsymbol{U})p(\boldsymbol{V})p(\boldsymbol{w})}_{\text{priors variables latentes}} \prod_{k=1}^{K} \underbrace{p(z_k|s_k, t_k)}_{\text{afinidad}} \underbrace{p(s_k|\boldsymbol{U}, \boldsymbol{x})}_{\text{prior usuario}} \underbrace{p(t_k|\boldsymbol{V}, \boldsymbol{y})}_{\text{prior ítem}}$$

En la sección 2.1.2 veremos la representación gráfica de esta factorización mediante el *factor graph* del modelo.

2.1.1. Caso *collaborative filtering*

Cuando se realiza *content-based filtering*, se utilizan las descripciones de usuarios/ítems para relacionarlos entre sí, formando "vecindarios". Entonces, para hacerle una recomendación a un usuario, podemos mirar los ítems que otros usuarios del vecindario calificaron positivamente. En este caso los gustos/géneros serán una combinación lineal de las columnas de la matriz de características latentes. Podemos pensar en las columnas de las matrices como "arquetipos" de usuarios/ítems que se combinan para modelar a cada usuario particular.

Como mencionamos anteriormente, cuando no tenemos descripciones iniciales de los usuarios/películas Matchbox realiza únicamente *collaborative filtering*. Esto se representa describiendo a las personas y los ítems mediante vectores de identidad, $x = \boldsymbol{e_i}$ que tienen todos ceros y un único 1 en la posición i. Cuando este es el caso, a cada usuario/ítem le corresponderá su propia columna en la matriz de características latentes. Formalmente:

$\boldsymbol{x} \in \mathbb{R} = \boldsymbol{e}_i$*:* identidad de las personas

$\boldsymbol{y} \in \mathbb{R} = \boldsymbol{e}_j$: identidad de los ítems

$\boldsymbol{s} = \boldsymbol{U}\boldsymbol{e}_i = \boldsymbol{u}_i$: con $\boldsymbol{u}_i$ la i-ésima columna de $\boldsymbol{U}$

$\boldsymbol{t} = \boldsymbol{V}\boldsymbol{e}_j = \boldsymbol{v}_j$: con $\boldsymbol{v}_j$ la j-ésima columan de $\boldsymbol{V}$

Luego, la afinidad z se reduce a la suma de los productos entre los gustos y las caracterísiticas.

$$\boldsymbol{z} = \boldsymbol{s}^T \boldsymbol{t} \overset{*}{=} \boldsymbol{u}_i^T \boldsymbol{v}_j = \sum_k \underbrace{u_{ki} v_{kj}}_{z_k} \tag{2.6}$$

*: caso *collaborative filtering*

2.1.2. Factor Graph

Otra manera de especificar matemáticamente el modelo probabilístico de Matchbox es mediante su *factor graph*, que se muestra en la figura 2.5

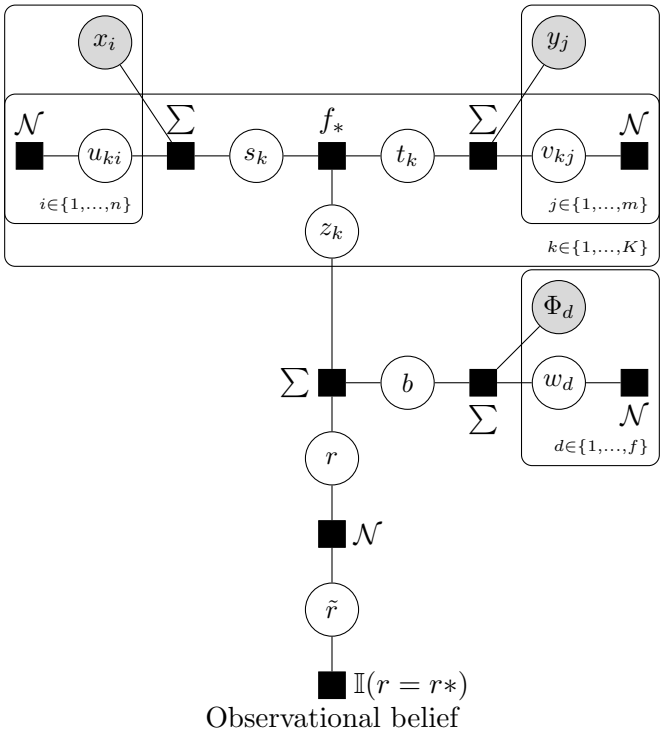


Fig. 2.1: Factor graph de Matchbox (sin discretización de puntuación)

Hagamos un paseo por el funcionamiento de Matchbox para un par (usuario, película). Partimos de las matrices $\boldsymbol{U}$ y $\boldsymbol{V}$ que contienen K características latentes para n (arquetipos de) usuarios y m (arquetipos de) ítems. Las matrices guardan los priors de estas características latentes, que son distribuciones normales, como se ve reflejado en los factores indicados con una $\mathcal{N}$ conectados a los nodos u_{ki} y v_{kj}. Llamaremos $f_{u_{ki}}$ y $f_{v_{ki}}$ a estos factores. Además, para el usuario y película actual tenemos los vectores x e y que se

corresponden a las descripciones de cada uno. Para predecir la puntuación correspondiente al par (usuario, película) debemos primero calcular la afinidad de dicho par respecto a cada una de las K características latentes que modelamos, z_k. Vemos que la parte superior del grafo está rodeada por un rectángulo, cuya esquina inferior derecha lee $k \in \{1, ..., K\}$. Esto indica que esta sección del grafo se repite K veces, cada una correspondiente a un subíndice k distinto. El funcionamiento de todas estas secciones será el mismo, pero hay que tener en cuenta que al factor que se encuentra debajo de z_k le llegarán K mensajes, cada uno de un nodo z_k distinto.

La afinidad del usuario por la k-ésima característica, s_k, se obtiene haciendo una suma "pesada" de las k-ésimas características de cada uno de los n arquetipos de usuario con los elementos de la descripción x del usuario como pesos. Esto se refleja en la función asignada al factor $\sum$ conectado a los nodos u y x, $\mathbb{I}(s_k = \sum_i u_{ki} x_i)$ La afinidad de la película por la k-ésima caracterísitca, t_k, se obtiene de manera análoga usando la matriz V y el vector de descripción y.

z_k representa la **afinidad del usuario y la película** en cuanto a esa característica. La función asignada al factor f_* es $\mathbb{I}(z_k = s_k t_k)$, es decir se multiplican la afinidad de usuario e ítem para la k-ésima característica. Cuando la característica k es muy alta para la película j ($v_{kj} \gg 0$) y a una persona i le gusta mucho esa característica k ($u_{ki} \gg 0$), entonces es muy alta la afinidad del usuario por la película **respecto a la k-ésima característica**, es decir z_k es grande. Si una caracteristica k es muy negativa en el ítem j ($v_{kj} \ll 0$) y el gusto por esa caracteristica también es muy negativa ($u_{ki} \ll 0$) la afinidad también será alta. Para que la afinidad sea baja los signos deben estar invertidos, es decir que el ítem presente mucho de algo que a la perona no le gusta, o que haya poco de lo que a la persona sí le gusta.

Tanto la afinidad alta como la afinidad baja pueden ser generadas por dos combinaciones distintas de preferencia de usuario - característica del ítem. Esto queda representado en los mensajes relacionados a la afinidad (puntualmente en el factor $*$): la multiplicación de variables generará una distribución bimodal donde cada moda representa una de estas combinaciones. En las próximas secciones veremos que será necesario aproximar estas distribuciones, y que aproximar implicará quedarnos con una única moda que represente a la combinación más probable de acuerdo a la información que se tiene en cada momento.

Una vez calculadas las afinidades z_k para cada característica, estas se suman para crear el puntaje (continuo) estimado que el usuario asignará a la película, r. Además, se modela un componente extra, el *sesgo* del usuario, b. Este sesgo está dado por la descripción del contexto de la puntuación $\mathbf{\Phi}$ y por pesos latentes (es decir, estimados) $\boldsymbol{w}$. Se obtiene b multiplicando cada d-ésimo elemento de la descripción Φ_d con el d-ésimo peso w_d y sumando todas estas multiplicaciones. Como mencionamos anteriormente, es común que la descripción del contexto $\mathbf{\Phi}$ esté dada por una combinación de la descripción del usuario y del ítem; en estos casos es posible reescribir $b = \boldsymbol{x}^T \boldsymbol{u} + \boldsymbol{y}^T \boldsymbol{v}$ donde $\boldsymbol{u}$ y $\boldsymbol{v}$ son sesgos latentes de usuario e ítem. Asumimos en el resto del trabajo, sin pérdida de generalidad, que este es el caso.

A la puntuación estimada no discretizada, r, que obtuvimos sumando los z_k y el sesgo, b, le agregamos un ruido observacional gaussiano $\mathcal{N}(\tau|\tilde{\tau}, \beta^2)$. Este ruido representa la incertidumbre generada por factores no modelados. Hecho esto, por fin tenemos nuestra estimación (continua) final del puntaje correspondiente al par (usuario, pelicula). Como esta estimación es continua, es probable que querramos discretizarla para poder traducir el valor continuo a algún sistema de puntuación, como puede ser el binario (me gusta/no me gusta) o numérico (1-5 estrellas, por ejemplo). Explicaremos como se realiza esta discretización en la sección 2.1.2.1.

Finalmente, se observa el valor real de la puntuación, r^* y se actualizan las probabilidades a priori asociadas a las características latentes del modelo, según la sorpresa que genera el valor observado.

2.1.2.1. Sistemas de puntuación

Si el sistema de puntuación fuera continuo, bastaría con colocar la identidad $\mathbb{I}(\tilde{r} = r*)$ en el factor $\tilde{r}$ para observar el puntaje real. Pero este no suele ser el caso; es más común que se utilice algún sistema de puntuación discreto. Por ejemplo MovieLens (versión 100.000 observaciones), el dataset que utilizamos para la experimentación, utiliza puntuaciones basadas en estrellas, de 1 a 5 estrellas (sistema ordinal). En contraste, hoy en día Netflix utiliza un sistema binario (me gusta/no me gusta). Estos modelos pueden utilizarse para modelar tanto *feedback* de usuario explícito (como los casos mencionados anteriormente) como implícito (por ejemplo la interacción/no interacción con ciertos ítems).

Cuando el sistema utiliza puntuaciones discretas, contaremos con un modelo gráfico separado que estimará el valor discreto más probable. El mensaje que subirá del factor marcado como *Observational belief* provendrá de este modelo - más adelante detallaremos el modelo de discretización mediante umbrales personalizados que propone Matchbox, pero este modelo podría ser reemplazado por otro o se podría utilizar la estimación continua directamente, según el caso. En la figura 2.1 se conecta a a la variable $\tilde{r}$ un factor con la función $\mathbb{I}(r = r^*)$ - esto es lo que se realiza cuando no se desea discretizar la puntuación.

El modelo generativo del sistema de puntuación ordinal que permite puntuaciones $r \in 1, \ldots, L$ está dado por la siguiente ecuación

$$p(l = a|\, \boldsymbol{b}_u, r) \begin{cases} \prod_{i=1}^{r^*-1} \mathbb{I}(r > \tilde{b}_{u(i-1)}) \prod_{i=r^*}^{L-1} \mathbb{I}(r < \tilde{b}_{u(i-1)}) & 1 < r^* < L \\ \prod_{i=1}^{L-1} \mathbb{I}(r < \tilde{b}_{u(i-1)}) & r^* = 1 \\ \prod_{i=1}^{L-1} \mathbb{I}(r > \tilde{b}_{u(i-1)}) & r^* = L \end{cases}$$

donde

r^*: el puntaje real (observado).

$\boldsymbol{b}_u \in \mathbb{R}^{L-1}$: umbrales de puntuación de un usuario específico u, donde L es la cantidad de puntuaciones posibles (por ejemplo, para 5 estrellas necesitamos 4 umbrales).

Es decir, los umbrales dividen el espacio de puntuación continuo en regiones consecutivas dentro de las cuales el usuario le da cierta puntuación a un ítem. Esta manera de modelar los umbrales nos permite que la distribución del espacio de puntuación no sea uniforme, ya que en la realidad tampoco lo suele ser. En (26) se muestra que, para los sitios estudiados, alrededor del 80 % de las puntuaciones de los clientes chinos y estadounidenses son de 4 o 5 estrellas, mientras que para los clientes japoneses aproximadamente 43 % de las reseñas son de 4 estrellas, mientras que un 30 % son de 5 estrellas y un 21 %, de 3 estrellas.

Los umbrales serán caracterísitcas latentes de nuestros usuarios, por lo que asignamos un prior independiente Gaussiano a cada uno con $p(b_{ui}) = \mathcal{N}(b_{ui}|\,\mu_i, \sigma_i^2)$. Intuitivamente, la puntuación discretizada estará dada por cuantos umbrales supera la distribución continua. Si no supera ninguno, el puntaje dado será el mínimo, por ejemplo 1 estrella en un sistema de 1 a 5 estrellas; si supera los cuatro umbrales el puntaje será de 5 estrellas, si supera dos umbrales será de 3, y así.

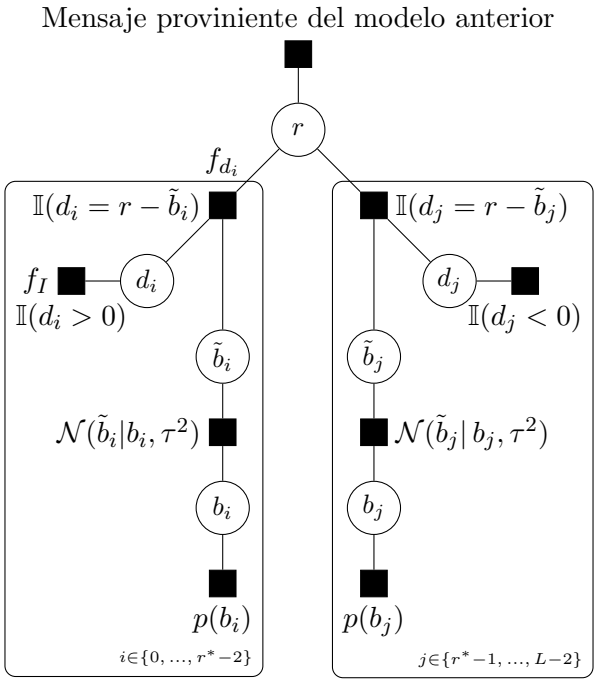
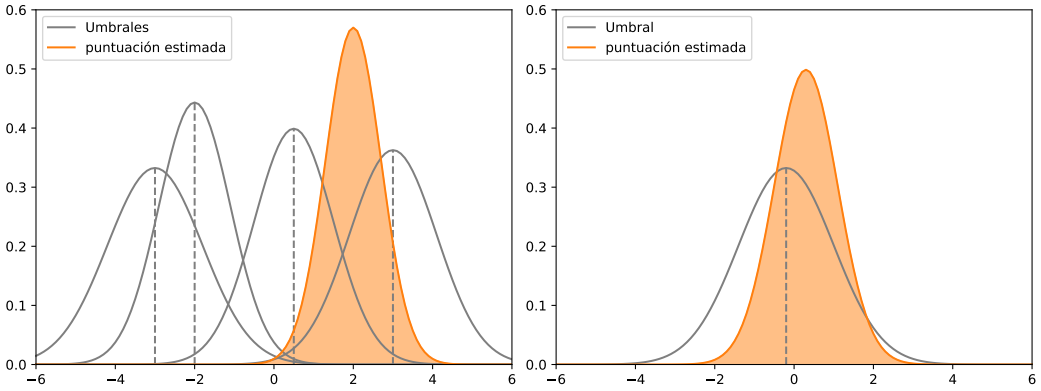


Fig. 2.2: Factor graph para inferir el sistema de puntuación de un usuario específico.

En la figura 2.2 mostramos el *factor graph* para un sistema ordinal de discretización de la puntuación. Cada rectángulo representa uno de los $L - 2$ umbrales, y se distinguen en la figura los umbrales superados por la puntuación real (a la izquierda) y los no superados (a la derecha). Para representar cuando se supera el umbral, $r > b_{u(i)}$, y cuando no, $r < b_{u(i)}$, se usan en conjunto $\mathbb{I}(d_i = r - \tilde{b}_i)$ y $\mathbb{I}(d_i > 0)$. Necesariamente, por cada umbral aparecerá una Gaussiana truncada que querremos aproximar para poder simplificar el cómputo. Entraremos en detalle en esto en la sección 3.1.3.



a Sistema 5 estrellas *b* Sistema binario

Fig. 2.3: Puntaución continua estimada (naranja) y los posteriors de umbrales de usuario (en gris) usados para traducirla al sistema de puntuación correspondiente.

2.1.3. Propagación en el tiempo

Comprendimos cómo procesa Matchbox un "evento" de puntuación, y hablamos de guardar los posteriors de las variables latentes luego de observar el puntaje real dado, para usarlos como priors de esas mismas variables a la hora de realizar la próxima predicción. Incorporar puntuaciones de a una, utilizando el posterior de la observación i-ésima como prior de la observación $i+1$-ésima, es lo que se conoce como *entrenamiento online* o *filtering*.

El prior de cada característica latente para la próxima estimación será el posterior de dicha característica luego de observar el valor del puntaje real. El posterior actual, posterior_t, está dado por la siguiente ecuación:

$$\text{posterior}_t = \text{Likelihood}_t \overbrace{\text{Likelihood}_{t-1}, \ldots, \text{Likelihood}_1 \text{prior}_1}^{\text{Posterior}_{t-1} \text{ como Prior}_t} \tag{2.7}$$

En nuestro *factor graph*, el *likelihood* (o versimilitud) va a estar dado por el mensaje que le llega al factor donde origina el prior. Podemos considerar a la verosimilitud como el filtro del prior; es la sorpresa del modelo al observar el dato real. Entonces, cada posterior puede ser visto como una acumulación de los filtros de creencia anteriores.

Al entrenar de esta manera, nuestras predicciones mejorarán a medida que incorporemos puntuaciones a nuestro modelo, resultando en predicciones finales de calidad. Esto implica que las predicciones que realizamos cuando contábamos con menos información habrán tenido mucha incerteza, y no ganaremos mucha información de ellas.

Si pudieramos volver a procesar los primeros eventos, incorporando la información que observamos posteriormente, podríamos sacarle mayor jugo a esas puntuaciones y mejorar nuestras estimaciones de variables latentes sin necesidad de incorporar nuevos datos. Matchbox hace exactamente esto con el concepto de "historial" de puntuaciones, también llamado *smoothing*, propagando los posteriors de las variables no solo hacia adelante (o hacia el futuro), sino también hacia atrás, es decir hacia observaciones pasadas. Reflexionemos sobre el funcionamiento y la utilidad del historial de puntuaciones con un ejemplo.

Fig. 2.4: Puntuaciones del ejemplo

Supongamos que tenemos en nuestro sistema dos usuarios: Facundo, a quien le gustan las películas de terror, y Camila, a quien no. Facundo y Camila se unieron a la plataforma para puntuar dos películas que vieron en un festival de películas indie: Beyond the Black Rainbox y Felicidad. Ninguna de estas películas había sido puntuada previamente en nuestro sistema. Facundo puntuó la primera positivamente y la segunda negativamente, mientras que Camila las puntuó de manera contraria.

Luego de esto Camila dejó de utilizar la plataforma, pero Facundo fue a un festival de películas clásicas de terror, las cuales puntuó positivamente, y un festival de comedias clásicas, que puntuó negativamente. Otra nueva usuaria, Maia, fue a los mismos festivales y puntuó de manera idéntica las mismas películas. Esto nos lleva a deducir que tanto Facundo como Maia son fanáticos de las películas de terror y probablemente no les gusten las comedias.

Ahora, el sistema quiere recomendarle películas a Maia que aún no haya visto, entre ellas Beyond the Black Rainbow y Felicidad. Como ambas películas fueron puntuadas cuando el sistema aún no conocía a Facundo ni a las películas, no fue un evento que nos diera mucha información de usuario ni película. Sin embargo, si Facundo puntuara altamente Beyond the Black Rainbow ahora, sería un indicio de que esa película es de terror y, además, de que le podría gustar a Maia. Esto sería, justamente, aplicar *smoothing*: quitamos del sistema la "información" que nos dió Facundo al puntuar Beyond the Black Rainbow la primera vez (que de todos modos fue poca) y pretendemos que Facundo vuelve a puntuar la película, **teniendo en cuenta todas sus demás puntuaciones**. De esta manera, el mismo evento nos dará mucha más información y nos llevará a hacer una predicción mejor para Maia. De la misma manera, podemos simular que se vuelve a puntuar Felicidad, y así evitar recomendarla.

Por otra parte, el paper de Matchbox habla de la posibilidad de modelar el cambio de las variables latentes en el tiempo; para esto se pueden modelar "sesiones" en las que se puntuan varios ítems y generar un posterior de la sesión para cada variable que aparece en alguno de los "eventos" de puntuación de la sesión. A estos posteriors se les agregará

	Beyond the Black Rainbow		Felicidad	
	P(r=1)	Terror latente	P(r=1)	Terror latente
Sin historial	0.51251	N(mu=0.33, var=0.72)	0.51387	N(mu=-0.38, var=0.70)
Con historial	0.53241	N(mu=0.75, var=0.65)	0.45298	N(mu=-0.86, var=0.64)

Tab. 2.1: Para películas nuevas al sistema que fueron puntuadas por un usuario a quien le gustan las películas de terror y uno al que no, se muestra el posterior latente de género "terror" de cada una y la probabilidad de recomendarlas a un usuario a quien le gustan las películas de terror. Detalles del experimento en apéndice sección 5.5.1

incerteza, haciendo que $\sigma^2_{(t+1)u_{ki}} = \sigma^2_{(t)u_{ki}} + \gamma^2$ donde $\sigma^2_{(t)u_{ki}}$ es la varianza de la característica latente u_{ki} en la sesión en tiempo t y γ^2 la incerteza que agregamos. Esto se puede usar por ejemplo para modelar cambios en el tiempo de gustos o del sistema personal de puntuación del usuario.

2.2. Aproximaciones analíticas

Para realizar inferencia exacta podemos utilizar el *sum-product algorithm*, como vimos en el ejemplo de Monty Hall en la sección 1.2.3. Las propiedades de las distribuciones gaussianas permiten calcular los mensajes de forma exacta y por lo tanto eficiente. Sin embargo, los mensajes que envía el factor f_* son distribuciones bimodales, a diferencia de (casi) todos los demás mensajes que son distribuciones Gaussianas. Para poder aprovechar las propiedades de las distribuciones Gaussianas, que nos permiten simplificar los mensajes propagados significativamente y por lo tanto abaratar el costo computacional de la inferencia, aproximaremos estos mensajes con distribuciones Gaussianas. Mencionamos en la sección 2.1.2.1 que querremos aproximar las Gaussianas truncadas provinientes de la discretización de la puntuación continua usando *Expectation Propagation*, que minimiza la divergencia $KL(p||q)$. Además, aproximaremos los mensajes salientes del factor f_* mediante *Variational Message Passing*, que buscará la distribución gaussiana que minimiza la divergencia KL inversa $KL(q||p)$ en lugar de la divergencia $KL(p||q)$.

Como hablamos en la sección 1.3, la inferencia bayesiana exacta puede ser prohibitivamente cara computacionalmente. En su reporte técnico sobre divergencias, Minka (22) explica:

> Fortunately, many belief networks benefit from an averaging effect. A network with many interacting elements can behave, on the whole, like a simpler network. This insight has led to a class of approximation methods called variational methods (Jordan et al., 1999) which approximate a complex network p by a simpler network q, optimizing the parameters of q to minimize information loss.
>
> ...the simpler network q can then act as a surrogate for p in a larger inference process. This makes variational methods well-suited to large networks, especially ones that evolve through time. A large network can be divided into

> pieces, each of which is approximated variationally, yielding an overall variational approximation to the whole network. This decomposition strategy leads us directly to message-passing algorithms.

Los algoritmos de pasaje de mensaje permiten ajustar aproximaciones variacionales de manera distribuida, lo cual es particularmente útil para redes de gran tamaño. Existen varios algoritmos de pasaje de mensaje, que se diferencian por la función de costo que minimizan y en consecuencia los mensajes que propagan. Puntualmente, nos interesan *Variational message-passing* (Winn & Bishop, 2005), una implementación con pasaje de mensajes de *mean-field* (Peterson & Anderson, 1987), y *Expectation Propagation* (Minka, 2001b). Minka indica cuatro pasos para construir un algoritmo de pasaje de mensajes:

1. Elegir una familia de funciones (exponencial) a la cual pertenecerá q.
2. Elegir una medida de divergencia a minimizar.
3. Construir un algoritmo de optimización para la medida de divergencia y familia de funciones elegidas.
4. Distribuir la optimizción en la red, subdividiendo la red p en factores y minimizando la medida de divergencia localmente en cada factor.

Una explicación general de los algoritmos de pasaje de mensajes se encuentra en el apéndice, sección 5.4.

2.2.1. Medidas de divergencia

Cada algoritmo de pasaje de mensajes minimiza una métrica de divergencia distinta, y estas dictan el carácter de las aproximaciones que se utilizan para cada mensaje propagado en el *factor graph*. *Expectation Propagation* propagará la distribución $q(x)$ que minimice la divergencia $\mathrm{KL}(p||q)$:

$$\underset{\mu,\sigma}{\mathrm{Arg\,min}}\ KL(p(x)||\mathcal{N}(x|\mu,\sigma))$$

Donde:

$$KL(p||q) = \int_x p(x) log \frac{p(x)}{q(x)} dx$$

Mientras que *Variational Message Passing* minimiza la divergencia $\mathrm{KL}(q||p)$ (es decir la misma divergencia, con sus argumentos intercambiados). Dado que las divergencias minimizadas son distintas, las distribuciones seleccionadas también lo serán; veamos qué características tienen las distribuciones resultantes con un ejemplo sencillo.

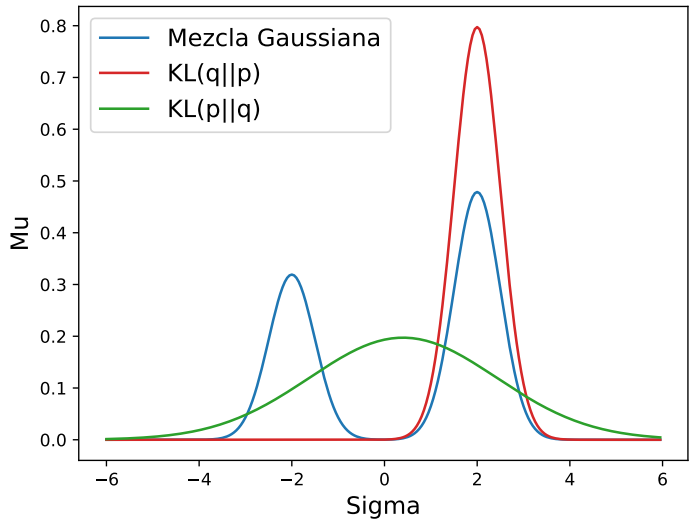
Sea la siguiente Gaussiana mixta nuestra distribución a aproximar $p(x)$

$$p(x) = \mathcal{N}(x|\mu_1, \sigma_1^2) \cdot 0.4 + \mathcal{N}(x|\mu_2, \sigma_2^2) \cdot 0.6 \tag{2.8}$$

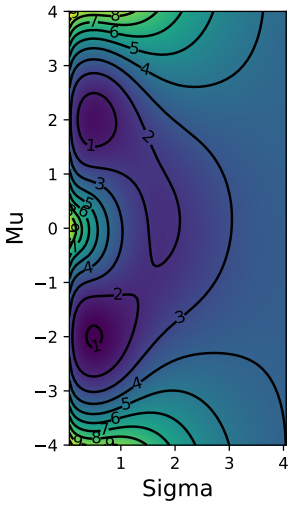
Busquemos las distribuciones q que mejor aproximan p de acuerdo a las medidas de divergencia utilizadas en *Expectation Propagation* y *Variational Message Passing*. Restringiremos la distribución aproximada a la familia de las Gaussianas, con media y varianza libres. En la figura 2.5b se grafican las distribuciones aproximadas resultantes.

Minka (22) explica en su reporte técnico sobre divergencias que cuando minimizamos la divergencia $KL(p||q)$, q tenderá a cubrir la mayor parte de p posible. Esta divergencia requiere $q > 0$ siempre que $p > 0$, evitando "falsos negativos". Decimos que la divergencia KL es una divergencia *inclusiva*, y por lo tanto vemos que en nuestro caso se estira, intentando cubrir ambas modas de la distribución.

Por otro lado, cuando minimizamos la divergencia inversa $KL(q||p)$, la distribución que mejor aproxima a p cubrirá la moda con mayor densidad. Si ambas modas tienen igual densidad, elegirá alguna de ellas, por lo que podemos decir que tiene la capacidad de "romper simetría". Esta divergencia busca que q sea chica cuando p es chica; $p(x) = 0$ fuerza que $q(x) = 0$, por lo que se dice que es una divergencia que causa *zero-forcing*. Podemos decir que la divergencia evita "falsos positivos", lo cual causa que partes de p se excluyan. Por esto, se dice que la divergencia KL inversa es una divergencia *excluyente*.



a Aproximación de una mezcla Gaussiana mediante una distribución Gaussiana



b KL($q||p$)

Fig. 2.5: Ejemplo de aproximación analítica. a) En azul la mezcla de gaussianas, en verde la gaussiana que minimiza la divergencia KL($p||q$), y en rojo la gaussiana que minimiza la divergencia KL($q||p$). b Todos los valores de la divergencia KL($q||p$), en violeta valores bajos, en verde valores altos.

La elección de divergencia a minimizar al realizar pasaje de mensajes no es trivial y depende de varios factores. Minka propone tres a tener en cuenta: el costo de computar cada métrica para el problema dado, si la familia de distribuciones que se usa para aproximar se parece a la distribución aproximada, y cuál es el objetivo de la inferencia. Puntualmente, cuando se busca computar distribuciones marginales a partir de una distribución q factorizada (como es nuestro caso), Minka recomienda utilizar una divergencia inclusiva como $KL(p||q)$, la cual preserva las marginales en mayor medida.

En contraste, recomienda usar divergencias exclusivas, como $KL(q||p)$, para aprendizaje bayesiano de modelos mixtos donde cada componente tiene parámetros distintos (como

la mezcla gaussiana de nuestro ejemplo). Una divergencia inclusiva intentará cubrir todas las modas de p, lo cual será más caro computacionalmente y posiblemente indeseado. Por lo tanto una divergencia exclusiva, que se concentre en una única moda, puede ajustarse bien a esta situación.

2.2.2. Aproximación en Matchbox

Matchbox usa, mayormente, *Expectation Propagation (EP)*, es decir minimiza la divergencia $KL(p||q)$ para la mayoría de los factores. Su innovación está en el uso localizado de *Variational Message-Passing (VMP)* en el factor f_*. Teniendo en cuenta las características de cada tipo de divergencia que comentamos en la sección anterior, esta elección cobra sentido.

El factor f_* propaga la función $I(z = s * k)$, la multiplicación de variables. Esta multiplicación de variables resulta en distribuciones bimodales similares a la mezcla gaussiana de la figura 2.5a. En dicha figura podemos ver que, si minimizaramos la divergencia $KL(p||q)$, la varianza crecería más y más en cada iteración del algoritmo, ya que la aproximación que minimiza $KL(p||q)$ intentará cubrir ambas modas. Matchbox elige minimizar la divergencia $KL(q||p)$ (divergencia KL inversa), que ante una distribución bimodal modela únicamente la moda más densa, rompiendo simetría. Ya comentamos en la sección 2.1.2 que cada una las modas de esta distribución representa una combinación de preferencia del usuario por la característica y presencia de dicha característica en el ítem; efectivamente, la minimización de la divergencia $KL(q||p)$ implica que Matchbox elige en cada iteración del algoritmo de pasaje de mensajes la combinación que explique la afinidad resultante con mayor probabilidad.

Por otra parte, respecto a la utilización de *Expectation Propagation*, es interesante notar que, salvo por los mensajes enviados por el factor f_* y por la variable d_i de la sección 2.1.2.1, los mensajes propagados son siempre distribuciones Gaussianas, por lo que las aproximaciones de *Expectation Propagation* y los mensajes enviados por el algoritmo de suma-producto coinciden. Por lo tanto, las Gaussianas truncadas del *factor graph* de umbrales son el único lugar donde se utiliza *Expectation Propagation* para aproximar una distribución no Gaussiana.

Refiriendonos al *factor graph* de umbrales (figura 2.2) se puede calcular $m_{d_i \to f_{d_i}}$, el mensaje que sube de la variable d_i, como:

$$m_{d_i \to f_{d_i}} \overset{\text{(paso del producto) y (probabilidad marginal)}}{=} \frac{\hat{p}(d_i)}{m_{f_{d_i} \to d_i}} = \frac{m_{f_I \to d_i}}{m_{f_{d_i} \to d_i}}$$

La marginal $p(d_i)$ es la creencia en la estimación continua de la puntuación que sobrevive a la observación de si el puntaje real supera el i-ésimo umbral. Por lo tanto, es una Gaussiana truncada en la posición estimada del umbral en la recta.

Siguiendo los pasos de Trueskill, Matchbox aproxima gaussianas truncadas mediante *Expectation Propagation*. Esta aproximación, $\hat{p}(d_i)$, será la Gaussiana con la **misma media y varianza que la Gaussiana truncada**. Basta esta aproximación para que todos los demás mensajes del *factor graph* de umbrales puedan ser calculados exactamente mediante el algoritmo de suma-producto.

2.3. Propiedades de las Gaussianas

Enumeramos las propiedades que necesitaremos para derivar los mensajes exactos que surgen del *sum-product algorithm*. En cuanto a la primera propiedad, la función indicadora $\mathbb{I}(\cdot = \cdot)$ vale 1 cuando la igualdad es verdadera y 0 cuando no. Se usa para representar distribuciones de probabilidad de variables **discretas** no aleatorias, como el valor de la observación de la caja en que se encuentra el regalo en el ejemplo de la sección 1.2.3. De la misma forma, la función delta de Dirac $\delta(\cdot = \cdot)$ se usa para representar distribuciones de probabilidad de variables **continuas** no aleatorias, como la afinidad del par (usuario, ítem) $p(z_k|s_k, t_k)$. Siguiendo el ejemplo del paper de Matchbox, haremos un abuso de notación y representaremos distribuciones de probabilidad de variables no aleatorias siempre con la función indicadora $\mathbb{I}$ independientemente de si la variable es continua o no.

Estas funciones nos permiten reducir la dimensionalidad del problema:

$$\iint_{x\ y} \mathbb{I}(x = h(y,z)) f(x) g(y) = \int_{y} f(h(y,z)) g(y) \qquad \text{(Función identificadora)}$$

Por otra parte, el producto de dos distribuciones gaussianas evaluadas en el mismo punto x puede expresarse como producto de otras dos distribuciones gaussianas, donde sólo una de ellas está evaluada en x:

$$\mathcal{N}(x|\mu_1, \sigma_1^2)\ \mathcal{N}(x|\mu_2, \sigma_2^2) = \mathcal{N}(\mu_1|\mu_2, \sigma_1^2 + \sigma_2^2)\ \mathcal{N}(x|\mu_*, \sigma_*^2) \qquad \text{(Producto de gaussianas)}$$

$$\text{Donde } \mu_* = \frac{\mu_1}{\sigma_1^2} + \frac{\mu_2}{\sigma_2^2} \text{ y } \sigma_*^2 = \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right)^{-1}$$

Utilizaremos además la propiedad que se deriva de la simetría de gaussianas:

$$\mathcal{N}(x|\mu, \sigma^2) = \mathcal{N}(\mu|x, \sigma^2) \qquad \text{(Simetría de gaussianas)}$$

y la multiplicación por constantes:

$$\mathcal{N}(x|\mu, \sigma^2) * c = \mathcal{N}(x|\, c * \mu, (c * \sigma)^2) \qquad \text{(Gaussiana * const.)}$$

Por último, utilizando las propiedades anteriores se puede demostrar que la suma de gaussianas resulta:

$$\iint_{x_1\ x_2} \mathbb{I}\,(t = x_1 + x_2) \mathcal{N}(x_1|\mu_1, \sigma_1^2) \mathcal{N}(x_2|\mu_2, \sigma_2^2) = \mathcal{N}(t|\mu_1 + \mu_2, \sigma_1 + \sigma_2^2)$$

(Suma de gaussianas)

Y de manera equivalente, para la resta:

$$\iint_{x_1\ x_2} \mathbb{I}\,(t = x_1 - x_2)\mathcal{N}(x_1|\mu_1, \sigma_1^2)\mathcal{N}(x_2|\mu_2, \sigma_2^2) = \mathcal{N}(t|\mu_1 - \mu_2, \sigma_1 + \sigma_2^2)$$

Más generalmente, sean c_i constantes,

$$\iint_{x_1 \cdots x_n} \mathbb{I}\left(t = \sum_{i=1}^{n} c_i\, x_i\right) \prod_{i=1}^{n} \mathcal{N}(x_i|\mu_i, \sigma_i^2) = \mathcal{N}(t|\sum_{i=1}^{n} c_i\, \mu_i, \sum_{i=1}^{n} c_i^2\, \sigma_i^2)$$

3. IMPLEMENTACIÓN

3.1. Documentación del algoritmo

La primera parte de la tesis (primer objetivo) estuvo dedicada a documentar e implementar el modelo Matchbox. El artículo de MatrchBox no incluye la documentación de los mensajes exactos, que corresponden al algoritmo de suma-producto, por lo que los presentamos en esta sección. La implementación de Matchbox fue desde un principio el objetivo principal de la tesis, debido a que cuando comenzamos no encontramos una implementación de código abierto disponible.

En la figura 3.1 se numeran los mensajes principales, a documentar.

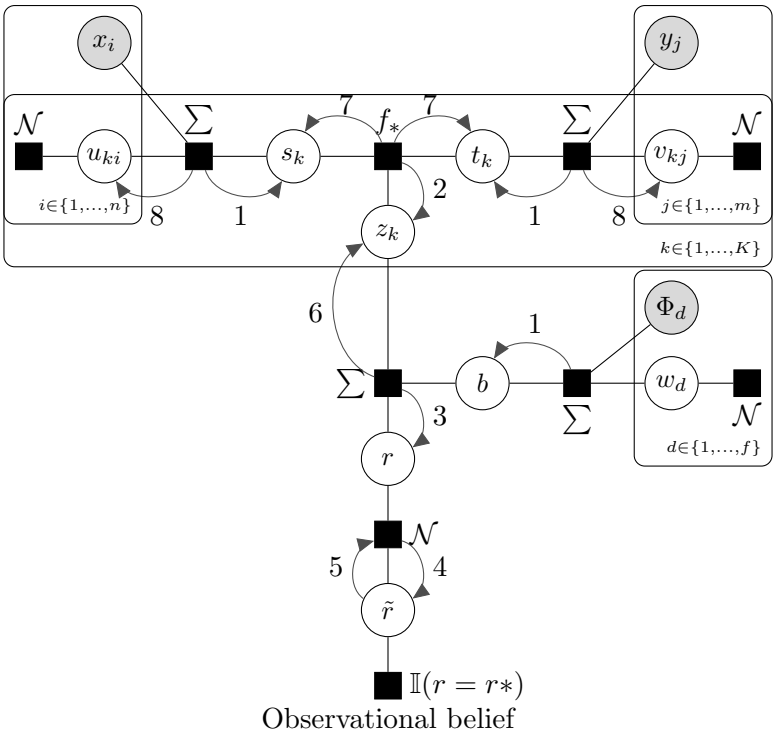


Fig. 3.1: Factor graph de Matchbox con mensajes numerados

3.1.1. Mensajes descendentes

3.1.1.1. Mensaje (1)

Dada la k-ésima característica, los mensajes que los factores $f_{u_{ki}}$ y $f_{v_{kj}}$, que representan las predicciones a priori (*priors*) de las características latentes, envían en el modelo 3.1 a

sus variables son:

$$m_{f_{u_{ki}} \to u_{ki}}(u_{ki}) = \mathcal{N}(u_{ki}|\mu_{U_{ki}}, \sigma^2_{U_{ki}}) \quad \forall i \in 1..n$$
$$m_{f_{v_{kj}} \to v_{kj}}(v_{kj}) = \mathcal{N}(v_{kj}|\mu_{V_{kj}}, \sigma^2_{V_{kj}}) \quad \forall j \in 1..m$$

Al recibir un único mensaje, las variables no hacen más que propagarlos. Por lo tanto, el mensaje señalado con el número 1 en la figura 2.5 es

$$\begin{aligned} m_{f_{s_k} \to s_k}(s_k) &= \int_{u_{k1}} \cdots \int_{u_{kn}} \mathbb{I}\Big(s_k = \sum_{l=1}^{n} u_{kl} x_l\Big) \prod_{l=1}^{n} m_{u_{kl} \to f_{u_{kl}}}(u_{kl}) \; du_{k1} \ldots du_k \\ &= \int_{u_{k1}} \cdots \int_{u_{kn}} \mathbb{I}\Big(s_k = \sum_{l=1}^{n} u_{kl} x_l\Big) \prod_{l=1}^{n} \mathcal{N}(u_{kl}|\mu_{U_{kl}}, \sigma^2_{U_{kl}}) \; du_{k1} \ldots du_k \end{aligned}$$

Cuando la descripción de las personas $\boldsymbol{x}$ se reduce a la identidad $\boldsymbol{e}_i$ (*collaborative filtering*), el mensaje se reduce a,

$$= \int_{u_{k1}} \cdots \int_{u_{kn}} \mathbb{I}\Big(s_k = u_{ki}\Big) \prod_{l=1}^{n} \mathcal{N}(u_{kl}|\mu_{U_{kl}}, \sigma^2_{U_{kl}}) \; du_{k1} \ldots du_k$$

Dada la función indicadora podemos reemplazar la variable u_{ki} por s_k (propiedad función identificadora). Al reemplazar todas las apariciones de la variable u_{ki} ya no queda ninguna aparición ligada, por lo que podemos sacar la integral correspondiente. La distribución centrada en s_k es la única que no está ligada a ninguna integral, por lo que la movemos fuera de estas. Resulta entonces que todas las distribuciones integradas se están integrando sobre todo su dominio, por ende:

$$= \mathcal{N}(s_k|\mu_{U_{ki}}, \sigma^2_{U_{ki}}) \underbrace{\int_{u_{kl}} \cdots \int \prod_{l=1}^{i-1} \mathcal{N}(u_{kl}|\mu_{kl}, \sigma^2_{kl}) du_{kl} \prod_{l=i+1}^{n} \mathcal{N}(u_{kl}|\mu_{kl}, \sigma^2_{kl}) du_{kl}}_{=1}$$

Cuando sí se cuenta con descripciones de usuario, el mensaje resulta:

$$m_{f_{s_k} \to s_k}(s_k) \overset{\text{Suma de gaussianas}}{=} \mathcal{N}(s_k|\sum_{i=1}^{n} x_i \mu_{U_{ki}}, \sum_{i=1}^{n} x_i^2 \sigma^2_{U_{ki}})$$

Dado que la matemática es muy similar a la del caso *collaborative filtering*, no continuaremos explorando este caso.

Por otra parte, el mensaje (1) correspondiente al sesgo es:

$$m_{f_b \to b}(b) = \int_{w_1} \cdots \int_{w_f} \mathbb{I}\Big(b = \sum_{d=1}^{f} \Phi_d w_d\Big) \prod_{d=1}^{f} \mathcal{N}(w_d|\mu_{w_d}, \sigma^2_{w_d}) \; dw_1 \ldots dw_f$$

Cuando $b = \boldsymbol{x}^T\boldsymbol{u} + \boldsymbol{y}^T\boldsymbol{v}$, se resuelve de manera similar a los mensajes de las características latentes:

$$m_{f_b \to b}(b) \overset{\text{Suma de gaussianas}}{=} \mathcal{N}(b|\underbrace{\mu_{\mathrm{u}_i} + \mu_{\mathrm{v}_j}}_{\mu_b}, \underbrace{\sigma^2_{\mathrm{u}_i} + \sigma^2_{\mathrm{v}_j}}_{\sigma^2_b})$$

En cambio, cuando $b = \boldsymbol{\Phi}^T\boldsymbol{w}$:

$$m_{f_b \to b}(b) = \mathcal{N}(b|\sum_{d=1}^{f} \Phi_d \mu_{w_d}, \sum_{d=1}^{f} \Phi_d^2 \sigma^2_{w_d})$$

Luego, los mensajes 1 respectivos a las características latentes y al sesgo que nos interesarán (caso *collaborative filtering* y $b = \boldsymbol{x}^T\boldsymbol{u} + \boldsymbol{y}^T\boldsymbol{v}$):

$$\begin{aligned}
m_{f_{s_k} \to s_k}(s_k) &= \mathcal{N}(s_k|\mu_{U_{ki}}, \sigma^2_{U_{ki}}) && \text{(Mensaje 1 a } s_k\text{)} \\
m_{f_{t_k} \to t_k}(t_k) &= \mathcal{N}(t_k|\mu_{V_{kj}}, \sigma^2_{V_{kj}}) && \text{(Mensaje 1 a } t_k\text{)} \\
m_{f_b \to b}(b) &= \mathcal{N}(b|\underbrace{\mu_{\mathrm{u}_i} + \mu_{\mathrm{v}_j}}_{\mu_b}, \underbrace{\sigma^2_{\mathrm{u}_i} + \sigma^2_{\mathrm{v}_j}}_{\sigma^2_b}) && \text{(Mensaje 1 a } b\text{)}
\end{aligned}$$

Para la i-ésima persona, el j-ésimo ítem y la k-ésima característica.

3.1.1.2. Mensaje (2)

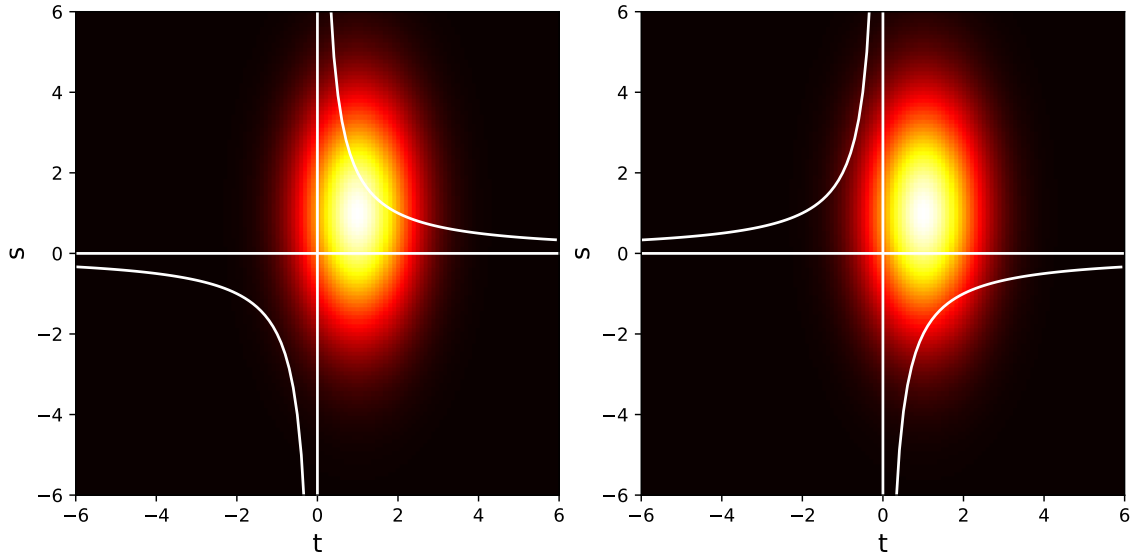
El mensaje señalado con el número 2 en la figura 2.5 es

$$\begin{aligned}
m_{f_* \to z_k}(z_k) &= \iint_{s_k t_k} \mathbb{I}(z_k = s_k t_k)\ \mathcal{N}(s_k|\mu_{U_{ki}}, \sigma^2_{U_{ki}})\ \mathcal{N}(t_k|\mu_{V_{kj}}, \sigma^2_{V_{kj}})\ ds_k dt_k \\
&= \iint_{s_k t_k} \mathbb{I}(\frac{z_k}{t_k} = s_k)\ \mathcal{N}(s_k|\mu_{U_{ki}}, \sigma^2_{U_{ki}})\ \mathcal{N}(t_k|\mu_{V_{kj}}, \sigma^2_{V_{kj}})\ ds_k dt_k \qquad (3.1) \\
&\overset{\text{Función identificadora}}{=} \int_{t_k} \mathcal{N}(\frac{z_k}{t_k}|\mu_{U_{ki}}, \sigma^2_{U_{ki}})\ \mathcal{N}(t_k|\mu_{V_{ki}}, \sigma^2_{V_{ki}})\, dt_k
\end{aligned}$$

Este, el mensaje 7, y la gaussiana truncada del modelo de umbrales, son los únicos mensajes que no se reducen a una distribución gaussiana. z_k, la afinidad entre persona e ítem respecto a la k-ésima característica, se obtiene del producto entre las variables s_k y t_k. Por lo tanto, el mensaje 2 evaluado en z_k es la integral de la multiplicación de dos distribuciones gaussianas, es decir la integral de una distribución bidimensional.

$$\mathcal{N}(s_k|\mu_{U_{ki}}, \sigma^2_{U_{ki}})\ \mathcal{N}(t_k|\mu_{V_{kj}}, \sigma^2_{V_{kj}}) \qquad (3.2)$$

Multiplicación de mensajes 1 provinientes de s_k y t_k. Es una distribución bidimensional.

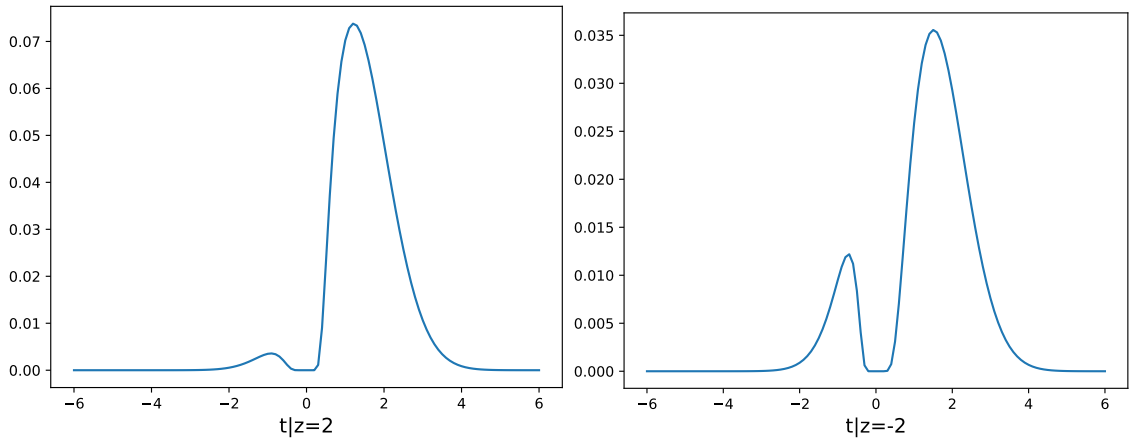


a : Con $z_k = 2$ b : Con $z_k = -2$

Fig. 3.2: La distribución de creencias bidimensional de la ecuación 3.2 con $\mathcal{N}(s_k|0,4)$, $\mathcal{N}(t_k|1,1)$. Las curvas representan la isolínea $z_k = s_k t_k$ sobre la cual se integra.

Para ganar intuición sobre la forma del mensaje 2, veamos primero en la figura 3.2 lo que ocurre con la multiplicación de las gaussianas que están al interior de la integral, antes de hacer el cambio de variables mediante la propiedad (Función identificadora).

Al hacer el cambio de variable en la ecuación 3.1 nos quedamos con los valores de la distribución bidimensional que se encuentran atravesados por la isolínea $z = s * t$. En la figura 3.3 graficamos los valores bajo la isolínea y vemos que aparecen dos modas. El valor final del mensaje 2 será la integral de estos valores.



a : Con $z_k = 2$ b : Con $z_k = -2$

Fig. 3.3: La densidad sobre la isolínea de la figura 3.2 con $\mu_{U_{ki}} = 1$, $\mu_{V_{kj}} = 0$, $\sigma_{U_{ki}} = 1$, $\sigma_{V_{kj}} = 1$.

En las figuras 3.3a y 3.3b graficamos el valor del mensaje 2 cuando $z_k = 2$ y $z_k = -2$. Existen muchas formas de aproximar esta integral, tanto numéricas como analíticas.

El paper elige aproximar este mensaje analíticamente mediante una Gaussiana, lo cual permite representar al mensaje utilizando sólo dos variables. Además, eligiendo aproximar analíticamente ciertos mensajes utilizando Gaussianas, se logra que **todos** los mensajes del esquema de pasaje de mensajes sean Gaussianas. Esto permite representar todos los mensajes mediante sólo dos parámetros, su media y su varianza, y que los cálculos a realizar sean sencillos y eficientes.

En el artículo original se decide aproximar este mensaje minimizando la divergencia KL inversa $KL(q||p)$ (*Variational Message Passing*) en lugar de la divergencia $KL(p||q)$ (*Expectation Propagation*), por lo que el mensaje 2 aproximado resulta:

$$\begin{aligned} m_{*\to z}(z) &\approx \mathcal{N}(z|\langle s\rangle\langle t\rangle,\ \langle s^2\rangle\langle t^2\rangle - \langle s\rangle^2\langle t\rangle^2) \\ &= \mathcal{N}(z|\mu_{s_k}\mu_{t_k},\ (\sigma^2_{s_k}+\mu^2_{s_k})(\sigma^2_{t_k}+\mu^2_{t_k}) - \mu^2_{s_k}\mu^2_{t_k}) \\ &= \mathcal{N}(z|\underbrace{\mu_{s_k}\mu_{t_k}}_{\mu_{z_k}}, \underbrace{\sigma^2_s\sigma^2_{t_k} + \sigma^2_{s_k}\mu^2_{t_k} + \sigma^2_{t_k}\mu^2_{s_k}}_{\sigma^2_{z_k}}) = \mathcal{N}(z|\mu_{z_k}, \sigma^2_{z_k}) \end{aligned}$$

Donde μ_{s_k} y σ_{s_k} son la media y la varianza de la distribución marginal $p(s_k)$ (análogo para t_k), y μ_{z_k}, σ_{z_k} son renombres.

Esta aproximación se calcula a partir de las marginales de las variables s y t pero, como estas marginales las calculamos multiplicando el mensaje 1 por el mensaje 7, se genera una dependencia cíclica. Esta dependencia cíclica se resuelve mediante el esquema de pasaje de mensajes iterativo abarcado en la sección 2.2.

En la primera iteración de dicho esquema $\mu_{s_k} = \mu_{U_{ki}}$ y $\sigma_{s_k} = \sigma_{U_{ki}}$, pues el mensaje ascendente que se necesita para computar la marginal $p(s_k)$ no está definido. En las siguientes iteraciones se actualiza la marginal y se vuelve a enviar el mensaje, iterando hasta que los mensajes del esquema convergen. En la figura 3.4 se puede visualizar la aproximación de la primera iteración.

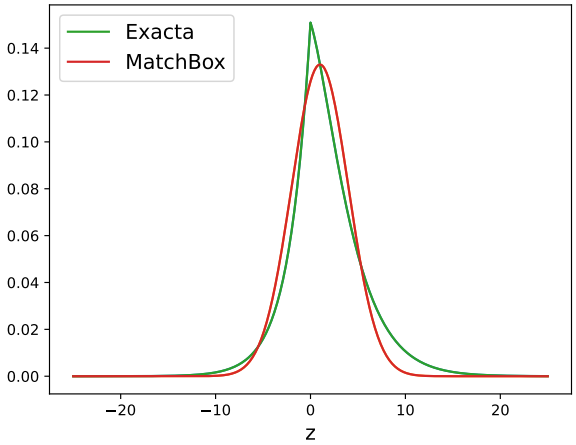


Fig. 3.4: Mensaje (2) exacto y aproximación propuesta por Matchbox

3.1.1.3. Mensaje (3)

El mensaje señalado con el número 3 es el prior de la variable $r = b + \sum_{k=1}^{K} z_k$, la afinidad $\boldsymbol{z}$ más el sesgo b, antes de agregarle el ruido β.

$$\begin{aligned} m_{f_r \to r}(r) &= \int_{\substack{z_k \\ k\in[1,K]}} \cdots \int_{b} \mathbb{I}\left(r = b + \sum_{k=1}^{K} z_k\right) m_{b\to f_r}(b) \left(\prod_{k=1}^{K} m_{z_k \to f_r}(z_k)\right) dz_1 \dots dz_K \, db \\ &= \int_{\substack{z_k \\ k\in[1,K]}} \cdots \int_{b} \mathbb{I}\left(r = b + \sum_{k=1}^{K} z_k\right) \mathcal{N}(b|\mu_b, \sigma_b^2) \prod_{k=1}^{K} \mathcal{N}(z_k|\mu_{z_k}, \sigma_{z_k}^2) \, dz_1 \dots dz_K \, db \\ &\underset{\text{Suma de gaussianas}}{=} \mathcal{N}(r| \underbrace{\mu_b + \sum_{k=1}^{K} \mu_{z_k}}_{\mu_r}, \underbrace{\sigma_b^2 + \sum_{k=1}^{K} \sigma_{z_k}^2}_{\sigma_r^2}) = \mathcal{N}(r|\mu_r, \sigma_r^2) \end{aligned}$$

3.1.1.4. Mensaje (4)

Este mensaje agrega a la puntuación estimada r el ruido β. β representa la incertidumbre generada por factores no modelados que podrían estar afectando la puntuación. Se modela a través de una distribcuión normal, resultando la predicción final $\tilde{r} \sim \mathcal{N}(r, \beta^2)$.

$$\begin{aligned} m_{f_{\tilde{r}} \to \tilde{r}}(\tilde{r}) &= \int_r \mathcal{N}(\tilde{r}|r, \beta^2) \, m_{r \to f_{\tilde{r}}}(r) \, dr \\ &= \int_r \mathcal{N}(\tilde{r}|r, \beta^2) \, \mathcal{N}(r|\mu_r, \sigma_r^2) \, dr \\ \text{(Prop. Simetría de gaussianas)} &= \int_r \mathcal{N}(r|\tilde{r}, \beta^2) \, \mathcal{N}(r|\mu_r, \sigma_r^2) \, dr \\ \text{(Prop. Producto de gaussianas)} &= \int_r \underbrace{\mathcal{N}(\tilde{r}|\mu_r, \beta^2 + \sigma_r^2)}_{\text{doesn't contain } r} \underbrace{\mathcal{N}(r|\mu_*, \sigma_*^2)}_{\text{integrates to 1}} dr \\ &= \mathcal{N}(\tilde{r}|\mu_r, \beta^2 + \sigma_r^2) = \mathcal{N}(\tilde{r}| \underbrace{\mu_b + \sum_{k=1}^{K} \mu_{z_k}}_{=\mu_r}, \beta^2 \underbrace{+\sigma_b^2 + \sum_{k=1}^{K} \sigma_{z_k}}_{=\sigma_r^2}) \end{aligned}$$

3.1.2. Mensajes ascendentes

3.1.2.1. Mensaje (5)

El mensaje señalado con el número 5 $m_{\tilde{r} \to f_{\tilde{r}}(\tilde{r})}$ no es más que el mensaje ascendente inicial, el cual el modelo de discretización de puntuación envía a la variable $\tilde{r}$. Cuando se utiliza un sistema de puntuación contínuo (es decir, no se discretiza la puntuación), el mensaje resulta:

$$m_{f_{ob} \to \tilde{r}}(\tilde{r}) = \mathbb{I}(\tilde{r} = r^*) \tag{3.3}$$

donde r^* es el valor real (observado) de la puntuación.

Los detalles del sistema discretización de puntuación mediante umbrales propuesto por el paper de Matchbox se documentan en la sección 3.1.3.

3.1.2.2. Mensaje (5′)

Llamamos 5′ al mensaje que envía el factor de puntuación estimada con ruido $f_{\tilde{r}}$, a la variable de puntuación sin ruido r.

$$\begin{aligned} m_{f_{\tilde{r}}\to r}(r) &= \int_r \mathbb{I}(\tilde{r} = r^*)\mathcal{N}(\tilde{r}|r,\, \beta^2)\, d\tilde{r} \\ &= \mathcal{N}(r^*|r,\, \beta^2) = \mathcal{N}(r|r^*,\, \beta^2) \end{aligned}$$

La verosimilitud de la puntuación estimada está entonces centrada en el valor observado r^*, con una incertidumbre equivalente al ruido β.

3.1.2.3. Mensaje (6)

El mensaje 6 enviado a z_k contiene la información de la puntuación observada, el sesgo, y de los mensajes descentenes 2. Los relaciona siguiendo la ecuación $\tilde{r} = b + \sum_l z_l$. Como vamos a enviar el mensaje a una de las variables z_k, despejamos por esa variable, por lo que resulta $z_k = \tilde{r} - b - \sum_{l\neq k} z_l$.

$$m_{f_r\to z_k}(z_k) = \iiint_{r\, b\, z_l} \mathbb{I}\left(z_k = r - b - \sum_{l\neq k} z_l\right) \mathcal{N}(r|r^*, \beta^2)\, \mathcal{N}(b|\mu_b,\, \sigma_b^2) \prod_l \mathcal{N}(z_l|\mu_{z_l},\, \sigma_{z_l}^2) dr\, db\, dz_l$$

(Prop Función identificadora, Suma de gaussianas)

$$= \mathcal{N}(z_k|\, \underbrace{r^* - \left(\mu_b + \sum_{l\neq k} \mu_{z_l}\right)}_{\mu_{6_k}},\ \underbrace{\beta^2 + \sigma_b^2 + \sum_{l\neq k} \sigma_{z_l}^2}_{\sigma_{6_k}^2}) = \mathcal{N}(z_k|\mu_{6_k},\, \sigma_{6_k}^2)$$

Este mensaje, la verosimilitud de la afinidad z_k, está centrado en la diferencia entre el valor de la puntuación observada r^* y la puntuación estimada **sin el aporte de** z_k, que se calcula $\mu_{6_k} = \mu_b + \sum_{l\neq k} \mu_{z_l}$. Su varianza (incertidumbre) incluye al ruido observacional y a la varianza de todas las estimaciones salvo la varianza de z_k, resultando $\sigma_{6_k}^2 = \beta^2 + \sigma_b^2 + \sum_{l\neq k} \sigma_{z_l}^2$.

3.1.2.4. Mensaje (7)

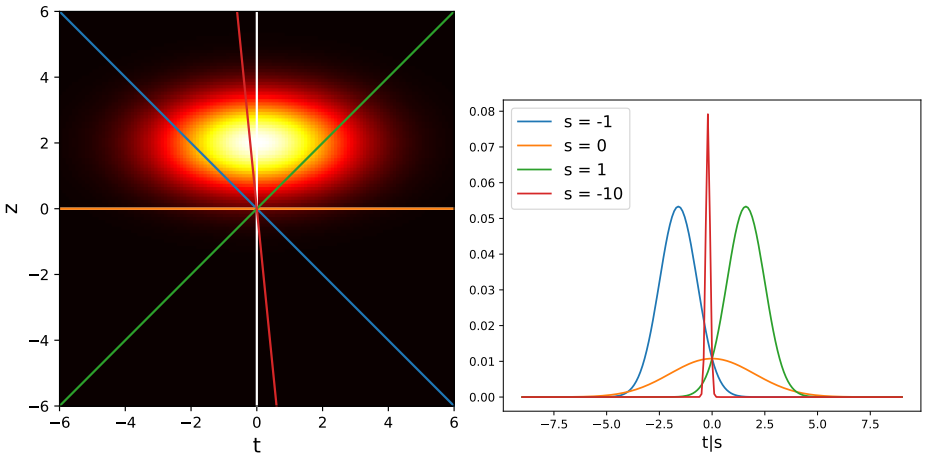
El mensaje 7 evaluado en un determinado s_k es

$$\begin{aligned} m_{f_*\to s_k}(s_k) &= \iint_{z_k t_k} \mathbb{I}(z_k = s_k t_k)\mathcal{N}(z_k|\mu_{6_k},\, \sigma_{6_k}^2)\mathcal{N}(t_k|\mu_{t_k},\, \sigma_{t_k}^2)\, dz_k\, dt_k && (3.4) \\ &= \int_{t_k} \mathcal{N}(s_k t_k|\mu_{6_k},\, \sigma_{6_k}^2)\mathcal{N}(t_k|\mu_{t_k},\, \sigma_{t_k}^2)\, dt_k && (3.5) \end{aligned}$$

De manera similar al mensaje 2, este mensaje no resulta en una distribución Gaussiana, por lo que Matchbox lo aproximará por una distribución Gaussiana. Una vez más esta

aproximación se elige minimizando la divergencia KL inversa $KL(q||p)$, y una vez más se calcula a partir de las marginales de s o t (según corresponda). Por lo tanto, el esquema de pasaje de mensajes iterativo es necesario para calcular este mensaje también.

En la figura 3.5a graficamos la distribución conjunta (3.4) y 4 isolineas ($s \in \{-10, -1, 0, 1\}$) para ganar una intuición de la forma del mensaje. En dicho gráfico se toma $\mathcal{N}(t_k|0, 2)$ como prior de t_k y $\mathcal{N}(t_k|2, 1)$ como valor del mensaje 6. En la figura 3.5b graficamos el área debajo de las mismas 4 isolíneas. El mensaje 7 será la integral de todas las posibles isolíneas.



a Mensaje 7 e isolíneas $s_k = z_k/t_k$ b Área debajo de isolíneas s_k

Fig. 3.5: $\mu_{t_k} = 0, \sigma_{t_k} = 2, \mu_{6_k} = 2, \sigma_{6_k} = 1, s_k \in \{-10, -1, 0, 1\}$

Observemos lo que ocurre con el área debajo de las isolíneas. Claramente las isolineas $s = -1$ y $s = 1$ tienen más área que la isolínea $s = 0$. Pero también tienen más área que la isolínea $s = -10$ que pasa cerca del centro de la distribución conjunta. Para entender por qué ocurre esto, hay que pensar en la "velocidad" con la que se pasa por la región de alta densidad a medida que movemos la variable sobre la que tenemos que integrar, en este caso t. En el límite, $s \approx -\infty$, la densidad debajo de la isolínea siempre es 0 para todos los valores de t, salvo para una región infinitecimal cercana al cero donde la densidad toca el máximo que tiene la distribcuión conjunta.

Por ese motivo, cuando graficamos el valor de las integrales debajo de todas las posibles isolíneas vemos que aparece una doble moda. La aproximación propuesta por el paper de Matchbox, que minimiza la divergencia KL inversa, es la siguiente:

$$m_{f_* \to s_k}(s_k) \approxeq \mathcal{N}(s_k|\frac{\langle m_{z_k \to *}\rangle \langle t_k\rangle}{\langle t_k^2\rangle}, \frac{\langle m_{z_k \to *}^2\rangle - \langle m_{z_k \to *}\rangle^2}{\langle t_k^2\rangle}) \tag{3.6}$$

$$= \mathcal{N}\Big(s_k|\frac{(\mu_{z_k \to *})(\mu_{t_k})}{\sigma_{t_k}^2 + \mu_{t_k}^2}, \frac{\left(\mu_{z_k \to *}^2 + \sigma_{z_k \to *}^2\right) - \mu_{z_k \to *}^2}{\left(\mu_{t_k}^2 + \sigma_{t_k}^2\right)}\Big) \tag{3.7}$$

El mensaje que se envía a la variable t_k es idéntico, reemplazando el uso de la marginal de t_k por la de s_k (el mensaje 7 para el usuario se calcula a partir de la marginal del ítem

y viceversa). Como ocurre en el mensaje 2, esta aproximación se construye a partir de las marginales de las variables s y t, por lo que es necesario iterar el esquema de pasaje de mensajes hasta alcanzar la convergencia. En la figura 3.6 mostramos el mensaje 7 exacto y la aproximación propuesta por Matchbox luego de la primer iteración del esquema de pasaje de mensajes (ecuación 3.6).

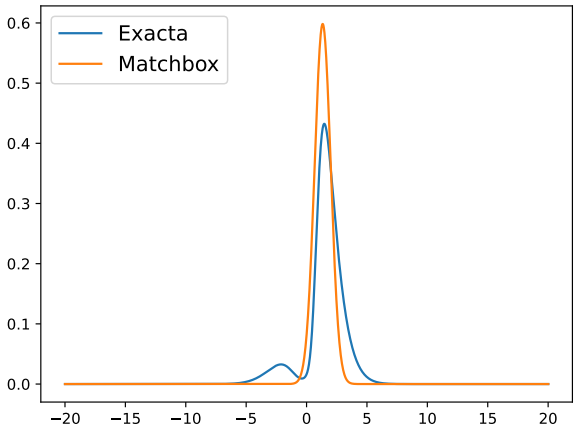


Fig. 3.6: Mensaje (7) exacto y aproximación propuesta por Matchbox

3.1.2.5. Mensaje (8)

Finalmente, llegamos a los mensajes 8, que se corresponden a las verosimilitudes de las variables latentes.

El mensaje ascendente que recibe la variable b es:

$$
\begin{aligned}
m_{f_r \to b}(b) &= \iint_{z_k\, r} \mathbb{I}\Big(b \;= r \;- \sum_k z_k\Big) \mathcal{N}\left(r \mid r^*,\; \beta^2\right) \prod_k \mathcal{N}\left(z_k \mid \mu_{z_k},\; \sigma^2_{z_k}\right) dz_k\, dr \\
&= \mathcal{N}\Big(b \mid \underbrace{r^* \;- \sum_k \mu_{z_k}}_{\mu_{\mathcal{L}b}},\; \underbrace{\beta^2 \;+ \sum_k \sigma^2_{z_k}}_{\sigma^2_{\mathcal{L}b}}\Big)
\end{aligned}
$$

Donde $\mu_{\mathcal{L}b}$ se lee como la media (μ) del *likelihood* ($\mathcal{L}$) de b. Luego, la verosimilitud que se envía a uno de los sesgos es,

$$
\begin{aligned}
m_{f_b \to\, \mathrm{u}_i}(\mathrm{u}_i) \;&=\; \iint_{b\;\; \mathrm{v}_j} \mathbb{I}(b \;=\; \mathrm{u}_i + \mathrm{v}_j) \mathcal{N}\left(\mathrm{v}_j \mid \mu_{\mathrm{v}_j},\; \sigma^2_{\mathrm{v}_j}\right) \mathcal{N}\left(b \mid \mu_{\mathcal{L}b},\; \sigma^2_{\mathcal{L}b}\right) db\, d\mathrm{v}_j \\
&= \iint_{b\;\; \mathrm{v}_j} \mathbb{I}(\mathrm{u}_i \;=\; b - \mathrm{v}_j) \mathcal{N}\left(\mathrm{v}_j \mid \mu_{\mathrm{v}_j},\; \sigma^2_{\mathrm{v}_j}\right) \mathcal{N}\left(b \mid \mu_{\mathcal{L}b},\; \sigma^2_{\mathcal{L}b}\right) db\, d\mathrm{v}_j \\
&= \mathcal{N}\left(\mathrm{u}_i \mid \mu_{\mathcal{L}b} - \mu_{\mathrm{v}_j},\; \sigma^2_{\mathcal{L}b} + \sigma^2_{\mathrm{v}_j}\right)
\end{aligned}
$$

Ahora veamos la verosimilitud de los ítems, $m_{f_{t_k} \to v_{k_j}}(v_{k_j})$. Al igual que en el mensaje 1, en el modelo *collaborative filtering*, vale que $\mathbb{I}(t_k = v_{k_i})$. Por lo tanto, este mensaje no

modifica en nada el mensaje ascendente que recibe t_k. Va a funcionar como un cambio de variable.

$$
\begin{aligned}
m_{f_{t_k}\to v_{k_j}}(v_{k_j}) &= \int_{V_{k_l}} \cdots \int \mathbb{I}(t_k = v_{k_j})\mathcal{N}(t_k|\mu_{7_{t_k}}, \sigma^2_{7_{t_k}})d_{t_k} \prod_{l\neq j} \mathcal{N}(v_{k_l}|\mu_{V_{k_l}}, \sigma^2_{V_{k_l}})d_{V_{k_l}} \\
&= \mathcal{N}(v_{k_j}|\mu_{7_{t_k}}, \sigma^2_{7_{t_k}}) \underbrace{\int_{V_{k_l}} \cdots \int \prod_{l\neq j} \mathcal{N}(v_{k_l}|\mu_{V_{k_l}}, \sigma^2_{V_{k_l}})d_{V_{k_l}}}_{\text{integra 1}} \\
&= \mathcal{N}(v_{k_j}|\mu_{7_{t_k}}, \sigma^2_{7_{t_k}})
\end{aligned}
$$

3.1.3. Modelos de umbrales

Cuando el modelo Matchbox considera umbrales, se hace necesario además aproximar la variable que representa la diferencia entre el umbral y el rating continuo, d_i. En la siguiente figura (3.7) se especifica el modelo con un único umbral en el caso $i \leq r^*$ para simplificar la visualización. Los mensajes son idénticos para más umbrales, salvando que cuando $i > r^*$ se cambia $\mathbb{I}(d_i > 0)$ por $\mathbb{I}(d_i < 0)$. En particular, siguiendo el algoritmo de suma-producto,

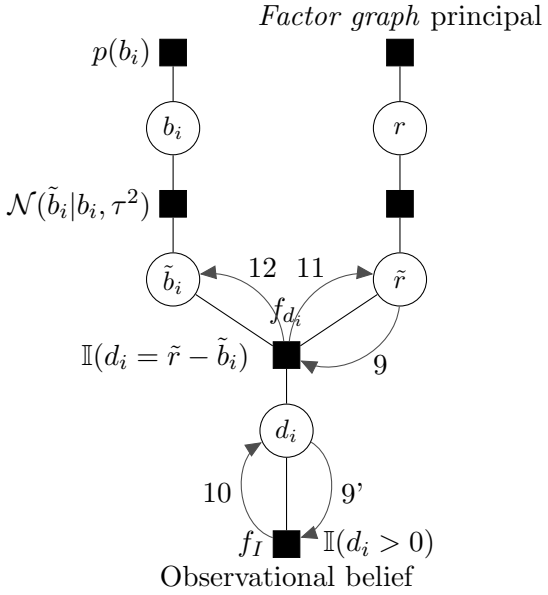


Fig. 3.7: Factor graph del i-ésimo umbral de un usuario, caso $i \leq r^*$. El *factor graph* con múltiples umbrales se puede observar en la figura 2.2.

la marginal de esta variable d_i se puede calcular como el producto de los mensajes que recibe:

$$
\begin{aligned}
m_{f_I\to d_i}\, m_{f_{d_i}\to d_i} &= p(d_i) \\
\mathbb{I}(d_i > 0)\mathcal{N}(d_i|\mu_{d_i}, \sigma^2_{d_i}) &= p(d_i)
\end{aligned}
$$

Matchbox, ante una Gaussiana truncada, utiliza la aproximación de *Expectation Propagation* para enviar la Gaussiana que mejor la aproxima, que será aquella con la misma media

y varianza que la Gaussiana truncada. Primero notemos que el mensaje $m_{f_I \to d_i} = m_{d_i \to f_{d_i}}$. Luego,

$$m_{f_I \to d_i} = m_{d_i \to f_{d_i}} = \frac{\widehat{p}(d_i)}{m_{f_{d_i} \to d_i}}$$

Donde $\widehat{p}(d_i)$ es la gaussiana con misma media y varianza que la gaussiana truncada exacta $p(d_i)$. Basta esta aproximación para que todos los demás mensajes del *factor graph* de thresholds puedan ser calculados exactamente mediante el algoritmo de suma-producto.

Los mensajes que se propagan por este grafo son los siguientes:

$$(9) \qquad m_{r \to f_{d_i}}(r) = \underbrace{\mathcal{N}\left(r|\mu_{\tilde{r}}, \sigma^2_{\tilde{r}}\right)}_{\text{mensaje 4}} \prod_{j \neq i} \underbrace{m_{f_{d_j} \to r}(r)}_{\text{mensaje 11}}$$

$$(9') \qquad m_{f_{d_i} \to d_i}(d_i) = \iint_{b_i \; r} \mathbb{I}\left(d_i = r - \tilde{b}_i\right) \; \Updownarrow_{\nabla \to \{\lceil_\rangle}(\nabla) \; \mathcal{N}\left(\tilde{b}_i|\mu_{b_i}, \tau^2\right) db_i \, dr$$
$$= \mathcal{N}\left(d_i|\mu_{\tilde{r}} - \mu_{b_i}, \sigma^2_{\tilde{r}} + \tau^2\right)$$

$$(10) \qquad m_{d_i \to f_{d_i}}(d_i) = \frac{p(d_i)}{m_{f_{d_i} \to d_i}(d_i)} \underset{\text{truncada}}{\approx} \frac{\hat{p}(d_i)}{m_{f_{d_i} \to d_i}(d_i)} = \mathcal{N}\left(d \mid \psi_i, \gamma_i^2\right)$$

$$(11) \qquad m_{f_{d_i} \to r}(r) = \iint_{d_i \; b_i} \mathbb{I}\left(d_i = r - \tilde{b}_i\right) \mathcal{N}\left(d_i \mid \psi_i, \gamma_i^2\right) \underbrace{\mathcal{N}\left(\tilde{b}_i|\mu_{b_i}, \tau^2\right)}_{\text{prior umbral + ruido}} dd_i \, db_i$$
$$= \mathcal{N}\left(r|\psi_i - \mu_{b_i}, \gamma_i^2 + \tau_i^2\right)$$

$$(12) \qquad m_{f_{d_i} \to \tilde{b}_i}\left(\tilde{b}_i\right) = \iint_{d_i \; r} \mathbb{I}\left(d_i = r - \tilde{b}_i\right) \mathcal{N}\left(r|\mu_{\tilde{r}}, \sigma^2_{\tilde{r}}\right) \mathcal{N}\left(d_i \mid \psi_i, \gamma_i^2\right) dd_i \, dr$$
$$= \mathcal{N}\left(\tilde{b}_i|\mu_{\tilde{r}} - \psi_i, \sigma^2_{\tilde{r}} + \gamma_i^2\right)$$

Cuando existen varios umbrales, como se muestra en la figura 2.2 se produce una mutua dependencia entre las distintas variables d_i (el mensaje 9 incorpora información de los mensajes 11 de los demás umbrales), por lo que es necesario iterar el esquema de pasaje de mensajes hasta alcanzar la convergencia. Se inicializan los mensajes aún no computados (mensajes 11 de umbrales aún no visitados) como distribuciones uniformes.

Para calcular la probabilidad los distintos puntajes es necesario aplicar un algoritmo que respete la regla del producto, de modo que el orden de envío de mensajes respete el cómputo de las distribuciones condicionales. Por ejemplo, si tenemos 3 umbrales y el rating supera los primeros 2 umbrales y queda por debajo del tercero, debe ocurrir:

$$P(d_1 > 0, d_2 > 0, d_3 < 0) = P(d_1 > 0)P(d_2 > 0|d_1 > 0)P(d_3 < 0|d_2 > 0, d_1 > 0)$$

Para lograr este efecto es necesario que, al implementar, los umbrales se visiten en orden relativo a su puntaje asociado (primero el umbral b_0, luego b_1, y así) y se propaguen los mensajes 11 de los umbrales visitados en cada paso.

3.2. Implementación computacional

Nuestra implementación del algoritmo sigue la especificación descripta en la sección anterior. Implementamos una clase `Gaussian` con las propiedades de Gaussianas presentadas en la sección 2.3, una clase `Messages` que implementa cada uno de los mensajes documentados, y finalmente la clase `Matchbox` que contiene la implementación completa del algoritmo.

Al inicializar una instancia de la clase Matchbox se pueden pasar por parámetro del constructor las siguientes configuraciones:

- `traitCount`: número de gustos implícitos a modelar
- `numThresholds`: número de umbrales de usuario a modelar (se corresponde a la cantidad de puntuaciones posibles en nuestro sistema de puntuación -1).
- `betaNoise2`: varianza del ruido incorporado a las puntuaciones (ver mensajes 4, 5)
- `biasNoise2`: varianza del ruido incorporado a los sesgos (se suma al mensaje 1 saliente de la variable bias)
- `tauNoise2`: varianza del ruido incorporado a los umbrales de usuario (incorporado por el factor que conecta las variables $\tilde{b_i}$ y b_i)

Los gustos implícitos se guardan en los diccionarios `U` (gustos de usuario), `V` (características de películas), `U_thr` (umbrales de usuario), `ubias` (sesgo de usuario) y `vbias` (sesgo de película).

Cada vez que se observa un evento de puntuación se suma el logaritmo de la probabilidad de predecir el puntaje correcto a la evidencia acumulada (de acuerdo a la fórmula 1.1). De este modo, la evidencia del modelo esta actualizada en todo momento. Guardamos tanto la evidencia on-line (que se calcula a partir de las predicciones realizadas al observar un evento por primera vez) como la evidencia con propagación (actualizando las predicciones de los eventos cada vez que se "vuelven a observar").

Para implementar la propagación, puntualmente el momento de volver a observar eventos previamente incorporados, es necesario quitar del modelo la información que aportó dicho evento cuando lo observamos por última vez. Dado que posterior $=$ prior $*$ likelihood, podemos recuperar las estimaciones a priori (antes de observar el evento) de las variables latentes dividiendo la estimación actual (posterior) por el likelihood generado por el modelo al observar la puntuación anteriormente. En nuestra implementación, guardamos un historial de eventos observados, y cada elemento de clase `Evento` guarda los likelihoods generados por el modelo la última vez que se observó este.

3.3. Validación

El paquete Infer.NET es un paquete de .NET desarrollado por Microsoft, gratuito y de código abierto, que provee herramientas para realizar programación probabilística y rea-

lizar inferencia bayesiana sobre modelos gráficos, como los *factor graphs* presentados en este trabajo, de manera eficiente y escalable (21). Además, es la herramienta utilizada por Microsoft internamente en soluciones para incorporar Machine Learning en Office, Xbox y Azure (27).

La documentación del paquete incluye varios tutoriales, entre ellos uno en el que construye una versión simplificada del sistema Matchbox (11). Puntualmente, el tutorial implementa la versión sin descripciones (*collaborative filtering*) de Matchbox, completa con umbrales personalizados, historial y propagación de información en el tiempo. Dado que el tutorial implementa todas las facetas del algoritmo que nos interesan, podemos utilizarlo para verificar nuestra propia implementación y por tanto nuestra comprensión de la matemática del mismo.

Encontramos que, efectivamente, nuestra implementación reproduce los resultados del tutorial. Partiendo de los mismos datos de entrenamiento, ambos algoritmos realizan estimaciones idénticas para las características latentes de usuarios e ítems presentes en el sistema, los sesgos latentes, los umbrales personalizados y las puntuaciones. Mostramos en la figura 3.8 las estimaciones realizadas por ambas implementaciones para un dataset pequeño de ejemplo.

Parámetros reales		Parámetros estimados	
Carac. 1 real	Carac. 2 real	Carac. 1 estimada	Carac. 2 estimada
PointMass(1.00)	PointMass(0.00)	PointMass(1.00)	PointMass(0.00)
PointMass(0.00)	PointMass(1.00)	PointMass(0.00)	PointMass(1.00)
-0.42	0.73	N(-0.235,0.269)	N(-0.072,0.280)
-0.06	-0.03	N(-0.418,0.299)	N(-0.044,0.341)
0.80	-0.92	N(0.043,0.282)	N(0.861,0.249)

Fig. 3.8: 100 observaciones - medias de estimaciones de características de primeros cinco elementos puntuados, modelando dos características latentes. Las 100 observaciones fueron sintetizadas a partir de modelar características de ítems y gustos de usuario. Dadas las mismas puntuaciones y priors, ambas implementaciones realizan las mismas estimaciones para todas las variables latentes (se muestran características de ítems de manera ilustrativa). Las características de los primeros k ítems observados son una matriz identidad de para dar un marco de referencia

Una vez que confirmamos que nuestra implementación reproducía las estimaciones de variables latentes del tutorial, quisimos verificar que la evidencia resultante para ambas implementaciones fuera la misma. Aquí encontramos que, a pesar de estar reproduciendo con exactitud las variables latentes y su evolución durante la propagación y agregado de información, la evidencia de ambas implementaciones difiere. La forma de obtener la evidencia de un modelo en Infer.NET no es directa siguiendo la definición de la evidencia como hacemos para nuestra implementación, sino que se inicializa otro modelo que tome al que nos interesa como variable con una Bernoulli(0.5) como estimación a priori y se pide

al motor de inferencia que estime la probabilidad de esa variable. Este cálculo pasa por el motor de inferencia como una estimación genérica, por lo que nos es imposible seguir el funcionamiento de su cómputo, pero tenemos motivo para confiar en nuestro valor de evidencia por sobre aquel del tutorial, como explicaremos.

En la figura 3.2 se muestran las probabilidades de cada combinación de umbral superado + umbral sin superar, para el caso en que tenemos un sistema de dos umbrales (donde las puntuaciones posibles serán 0, 1 y 2 estrellas) que acaba de ser inicializado, e incorporamos un primer evento de puntuación.

Estrellas	Umbral 1	Umbral 2	Probabilidad
0	no superado	no superado	0.35186
?	no superado	superado	0.03358
1	superado	no superado	0.19733
2	superado	superado	0.35186
Total			0.93463

Tab. 3.1: Probabilidades para el tutorial (dos umbrales, dos características latentes)

Estrellas	Umbral 1	Umbral 2	Probabilidad
0	no superado	no superado	0.38198
?	no superado	superado	0.03715
1	superado	no superado	0.18668
2	superado	superado	0.39419
Total			1

Tab. 3.2: Probabilidades para nuestra implementación (dos umbrales, dos características latentes)

Notar que incluímos la probabilidad del caso en que el primer umbral no es superado y el segundo sí, caso que no tiene correlación con ningun puntaje posible del sistema de puntuación propuesto. El modelo, como está planteado por el paper, admite esta posibilidad como válida y por lo tanto le asigna una parte de la creencia total. En esta posibilidad, que nunca se dará por como funciona nuestro sistema de puntuaciones, estamos "perdiendo" algo de probabilidad, lo cual nos lleva a creer que este modelo podría mejorarse. No exploraremos posibles mejoras en este trabajo.

Por otra parte, aunque ambas implementaciones difieren en la probabilidad asignada a cada uno de los casos, las probabilidades asignadas por nuestra implementación siempre suman 1, lo cual no es el caso para la implementación del tutorial de Matchbox, como se puede apreciar en el ejemplo. De hecho, al agregar más umbrales, la creencia total se aleja aún más de 1 que en el ejemplo mostrado. Por esto, y considerando que las estimaciones de variables latentes y puntuaciones estimadas son idénticas para ambos tutoriales, y la dificultad de auditar el cálculo de la evidencia para el paquete, preferimos nuestro cálculo

de evidencia por sobre el del tutorial.

Además del tutorial, Infer.NET ofrece la clase `RecommenderSystem`, que es una implementación de Matchbox optimizada para escalabilidad y eficiencia y que permite incorporar descripciones de usuario, a diferencia del tutorial. Esta implementación es mucho menos transparente que la implementación del tutorial, entre otros motivos porque en esta el algoritmo está implementado como una especie de máquina de estados. Cada paso del algoritmo es un llamado sin retorno que actualiza el estado interno del sistema de recomendación, de maneras que dificultan el acceso a resultados intermedios y dificultan encontrar las implementaciones de mensajes particulares. Sin embargo, validando nuestra implementación contra la del tutorial, tenemos confianza que nuestra implementación y la del paquete de Microsoft son comparables. Por lo tanto, para la comparativa que realizaremos contra otros algoritmos de recomendación (sección 4), utilizaremos la clase `RecommenderSystem`, la cual está más preparada para tomar datos a mayor escala.

4. COMPARATIVA

Para evaluar el desempeño de Matchbox decidimos compararlo primero con algunos algoritmos clásicos de la literatura, a modo de *baseline* de referencia, y luego con algoritmos de punta más utilizados por la comunidad de data science y en competencias de recomendación.

Sobre la interpretación del logaritmo de la evidencia

En las tablas que siguen se muestran el logaritmo de la evidencia de los modelos y la media geométrica de las predicciones. La media geométrica se interpreta como la predicción característica del modelo. El logaritmo de la evidencia se incluye para evidenciar los órdenes de magnitud de las diferencias: cuanta más evidencia tiene un modelo, más creencia sobrevivió a todas las observaciones (multiplicaciones) realizadas. Al observar el logaritmo base 10, una diferencia de 1 entre logaritmos de evidencias implica que la predicción de la implementación con **menor** (logaritmo de) evidencia es 10 veces mejor.

Dado que las magnitudes de esta métrica son muy grandes nos pareció más informativo reportar, para cada algoritmo, la diferencia entre el valor de esta para la implementación de Matchbox de Infer.NET y su valor para el algoritmo correspondiente. La reportaremos como diferencia de la evidencia en órdenes de magnitud.

4.1. Datasets de puntuaciones

Existen varios *datasets* públicos y de diversos tamaños disponibles libremente. En este trabajo utilizamos *MovieLens*, puntualmente la versión de 100.000 puntuaciones, que tiene un formato compatible con Infer.NET sin necesidad de modificaciones. El dataset está compuesto de varios archivos, pero específicamente nos interesará u.data que contiene los identificadores de usuario, ítem, puntajes y timestamps en ese orden, separados mediante tabulaciones. Se cuenta con datos de 943 usuarios y 1682 ítems, y cada usuario ha puntuado al menos 20 películas.

Para los experimentos sobre datasets simulados incluiremos el detalle de los parámetros utilizados para generar los datos en las secciones del apéndice correspondientes.

4.2. Notas sobre el cálculo de evidencia

Hablamos en la introducción del cálculo de la evidencia para un conjunto de datos: estrictamente, la predicción a priori se consigue prediciendo un primer dato del conjunto, luego prediciendo un segundo dato dado observar el valor real del primero, luego un tercero dados el primero y el segundo, y así. Computar la evidencia de esta manera es costoso o directamente imposible para ciertos algoritmos: *Random Forest*, por ejemplo, crea sus

árboles de decisión a partir de las características de los datos de entrenamiento observados como conjunto. Por ende, optamos por utilizar en su lugar una técnica clásica de cross-validation. Dado un conjunto de datos de entrenamiento `train_data` y un conjunto de datos de validación `test_data` = $\{t_1, t_2, t_3, ...\}$ calcularemos:

$$P(\texttt{test\_data}|\text{Modelo}) = \prod_{t_i}^{\texttt{test\_data}} P(t_i|\text{Modelo}, \texttt{train\_data})$$

En otras palabras, entrenaremos cada modelo sobre la `train_data` y luego obtendremos la predicción a priori del modelo entrenado para cada una de las observaciones del set de datos de validación por separado. Entonces, a diferencia del método descripto antes:

- Se entrena sobre parte del dataset total de una vez, sin calcular las predicciones a priori incrementales.
- El modelo nunca llega a observar los datos de **test_data** - **no** los va incorporando a medida que obtenemos las predicciones a priori. Es decir, hay parte de las observaciones que nunca se "aprenden".

4.3. Baseline: *Random Forest* y *Gradient Boosting*

Decidimos comenzar comparando Matchbox contra los clasificadores *Random Forest* y *Gradient Boosting*, dos algoritmos populares, flexibles y de buen desempeño para esta tarea.

Random Forest es un algoritmo sumamente popular, capaz de resolver problemas tanto de clasificación como de regresión. Durante la etapa de entrenamiento, *Random Forest* crea múltiples árboles de decisión y los entrena con subconjuntos aleatorios de los datos de entrenamiento (*bootstraping*), eligiendo de manera aleatoria sobre qué *features* hacer las divisiones. Para alcanzar un único resultado, el algoritmo promedia los resultados de los árboles que lo componen, consiguiendo menor varianza y *overfitting* respecto al uso de un único árbol de decisión.

Por su parte, *Gradient Boosting* construye árboles de decisión secuencialmente, donde cada árbol corrige los errores de los anteriores mediante la optimización de una *loss function* dada. Esta secuencialidad permite a *Gradient Boosting* encontrar relaciones complejas en el dataset, pero tiene mayor probabilidad de realizar *overfitting* sobre datos ruidosos.

Utilizamos la implementacioón de *Random Forest* de la librería `sklearn` (23) y la librería `LightGBM` (13) para *Gradient Boosting*. Esperamos que Matchbox supere el desempeño de ambos sistemas, considerando que ninguno toma consideraciones particulares para el caso de recomendación.

Para la comparación realizamos optimización de hiperparámetros; se presentan en este capítulo solo los mejores resultados para cada corrida, y se incluyen en el apéndice (capítulo

5.3) los espacios de búsqueda de los hiperparámetros optimizados y los resultados para las mejores 10 combinaciones encontradas. Para realizar la optimización, se utilizó la librería de python `Hyperopt`.

Utilizando el dataset Movielens, descripto en la sección 4.1, obtuvimos las predicciones características mostradas en la figura 4.1. Vemos que mientras *Gradient Boosting* y *Random Forest* presentaron *overfitting*, Matchbox tuvo resultados igual de buenos en test que en train. Esto se alínea con nuestra anterior observación de que la inferencia bayesiana no genera overfitting, como mencionamos en 1.2.2. Reflexionaremos respecto a este resultado, el cuál se repetirá en las próximas corridas, más adelante.

Por otra parte, la probabilidad de predicción correcta promedio (media geométrica de la evidencia) fue muy baja: hay cinco puntuaciones posibles (1 a 5 estrellas inclusive) por lo que querríamos superar la probabilidad de adivinar el puntaje eligiendo uno aleatoriamente, que es de $1/5 = 0.20$. Para MovieLens, vemos que las predicciones de Matchbox no superan las de *Random Forest*, las de *Gradient Boosting*, ni al azar. Los detalles de los resultados presentados se encuentran en el apéndice, sección 5.5.3

	Infer.NET	*Random Forest*	*Gradient Boosting*
Geometric mean (test)	0.14894	0.26211	0.26674
Geometric mean (train)	0.14894	0.44019	0.30983
diferencia de la evidencia en órdenes de magnitud (test)	0	-14130	-14568
diferencia de la evidencia en órdenes de magnitud (train)	0	-81969	-55630

Tab. 4.1: Evaluación de modelos con dataset MovieLens, 100.000 puntuaciones (75:25 train:test), sistema de puntuación de 5 estrellas

4.3.1. Comparación sobre dataset sintético

Ante los resultados de la comparación inicial perdió sentido avanzar con la comparación contra sistemas más complejos, así que decidimos en su lugar repetir el experimento anterior sobre un dataset sintetizado siguiendo el modelo de la realidad propuesto por Matchbox. Nuestra hipótesis de cara a este experimento es que Matchbox debería poder estimar razonablemente bien las variables latentes para este dataset, dado que las puntuaciones se generaron a partir de usuarios que siguen los comportamientos exactos que Matchbox asume se dan en la realidad.

Para sintetizar los datos seguimos de cerca la metodología utilizada en el tutorial de Matchbox provisto por la documentación de Infer.NET.

En la tabla 4.2 se observan los mejores resultados obtenidos para cada algoritmo. Detalles de los parámetros usados para la generación de los datos y de las cuales combinaciones de parámetros resultaron mejores se encuentran en el apéndice, sección 5.5.4.

	Infer.NET	*Random Forest*	*Gradient Boosting*
Geometric mean (test)	0.06891	0.22455	0.21880
Geometric mean (train)	0.06652	0.36071	0.23232
diferencia de la evidencia en órdenes de magnitud (test)	0	-29530	-28881
diferencia de la evidencia en órdenes de magnitud (train)	0	-126788	-93792

Tab. 4.2: Evaluación de modelos con dataset sintético, 100.000 puntuaciones (75:25 train:test), sistema de puntuación de 5 estrellas

Sorprendentemente, la performance de Matchbox empeoró para este dataset. La de *Gradient Boosting* y *Random Forest* también disminuyó, aunque menos dramáticamente. Curiosamente, las predicciones de Matchbox no mejoran ni siquiera cuando el K del modelo es el mismo que fue utilizado para la generación de datos.

En principio, que la performance empeore para todos los algoritmos nos indicaría que el modelo no refleja particularmente bien la realidad (mal diseño). Pero dado que en nuestras pruebas del tutorial el algoritmo parecía estar prediciendo razonablemente, surge otra hipótesis: podría ser que el diseño sea bueno, pero en la clase `RecommenderSystem` se introduzca algún error implementativo.

Realizando pruebas con la implementación del tutorial encontramos que esta tiene algunos problemas cuando la cantidad de puntuaciones observadas es grande, los cuales discutimos es más detalle en la sección 5.1.1. Gracias a que reportamos estos problemas en el *issue tracker* del repositorio, nos enteramos de que en la implementación de la clase `RecommenderSystem` se estaba utilizando *damping*, una técnica no mencionada en el paper de Matchbox. Esto nos hace sospechar que podría haber otras técnicas agregadas a la implementación de Microsoft que no estan declaradas en el trabajo original, y deja en evidencia que el tutorial (y por tanto nuestra implementación) y la clase `RecommenderSystem` no son equivalentes. Además, es posible que la implementación del paquete, en sus diferencias, esté incorporando algún error de implementación.

Nos interesa, entonces, hacer una nueva comparación incluyendo nuestra propia implementación también. Si el problema es de diseño, nuestra implementación no debería hacer buenas predicciones para un gran número de observaciones. Si, en cambio, el problema es implementativo, es probable que nuestra implementación supere el desempeño de la clase `RecommenderSystem`.

4.3.2. Comparación sobre dataset de puntuaciones binarias

Antes de avanzar con la comparación de versiones de Matchbox, revisitemos otro posible causante del mal desempeño, el modelo de umbrales. En la sección 3.3 mostramos que el modelo de umbrales asigna creencia a casos que no tienen asignado un puntaje, y por

lo tanto nunca se darán en el dataset; esto implica una "pérdida de creencia". Evidentemente, este modelo de umbrales se podría mejorar, de manera de excluir estos casos. Para tener una idea del impacto de esta pérdida de creencia en la capacidad de predicción de Matchbox, decidimos hacer las mismas pruebas "binarizando" el dataset, ya que con un sólo umbral las únicas posibilidades (no superar/superar el umbral) tienen correspondencia directa con puntajes posibles (no me gusta/me gusta). Para binarizar el dataset asignamos a cada evento un puntaje de 1 ("me gusta") si el puntaje original era de 4 o más estrellas, y un puntaje de 0 ("no me gusta") si era menor.

En la figura 4.3 podemos observar que la implementación de Infer.NET (clase `RecommenderSystem`) sigue sin superar al azar (que en este caso es 0.5), mientras que *Gradient Boosting* y *Random Forest* sí. Concluímos que el modelo no tiene un impacto positivo en las puntuaciones, pero tampoco es la principal causa de las malas predicciónes. Utilizar más de un umbral es computacionalmente costoso (implica realizar propagación de mensajes en el grafo de umbrales por cada iteración en el grafo principal), y no genera una mejora de la capacidad predictiva. Por lo tanto, durante los siguientes experimentos utilizaremos datasets binarios directamente. Los detalles de este experimento se encuentran en el apéndice, sección 5.5.3.

	Infer.NET	*Random Forest*	*Gradient Boosting*
Geometric mean (test)	0.39364	0.54940	0.55137
Geometric mean (train)	0.39019	0.66200	0.57974
diferencia de la evidencia en órdenes de magnitud (test)	0	-8335	-8424
diferencia de la evidencia en órdenes de magnitud (train)	0	-39647	-29695

Tab. 4.3: Evaluación de modelos con dataset MovieLens, 100.000 puntuaciones (75:25 train:test), sistema de puntuación binario (me gusta/no me gusta)

4.4. Comparación entre versiones de Matchbox

4.4.1. Notas sobre la evidencia en `test` en nuestra implementación

Ninguno de los paquetes que utilizamos para la comparativa provee, a nuestro conocer, formas de realizar entrenamiento incremental (*on-line*) - una vez que el modelo se encuentra entrenado, sus estimaciones internas quedarán fijas y será necesario entrenar de nuevo con el dataset completo para ajustarlas. Nuestra implementación, en contraste, ajusta sus estimaciones siempre que observa un nuevo evento - se "entrena" continuamente. Esto tiene una gran ventaja: durante la evaluación de `test`, nuestra implementación puede incorporar la información que brindan los puntajes reales testeados (luego de predecirlos sin

observarlos). En conjunto con la inexistencia de *overfitting* en este modelo, podríamos proponer no realizar una división train/test en primer lugar. Esto es el llamado *entrenamiento on-line* que se menciona en el paper; nos extraña que la implementación de Infer.NET no permita incorporar nuevas observaciones a un modelo ya entrenado.

Al fin y al cabo, lo que estamos haciendo con nuestra implementación no es más que calcular la evidencia real en lugar de recurrir al método de cross-validation alternativo que propusimos antes: recordemos que, sean $d_1, \ldots, d_n$ nuestros datos y M nuestro modelo:

$$P(\text{Datos}|M) = P(d_1|M)P(d_2|d_1, M)P(d_3|d_2, d_1, M)\ldots$$

Por otra parte, notamos que en ambas implementaciones de Matchbox se realizó propagación de creencias sobre el historial de puntuaciones del dataset de entrenamiento, pero no sobre el dataset de testeo.

4.4.2. Comparación sobre dataset sintético

Comenzamos evaluando el desempeño de nuestra implementación contra la clase `RecommenderSystem`, *Gradient Boosting* y *Random Forest*, con 100.000 puntuaciones sintetizadas. Quisimos evaluar también el desempeño del tutorial de Matchbox en este experimento, pero esto no fue posible debido a comportamiento que el tutorial no parece reportar resultados confiables para datasets de gran tamaño. Nos explayaremos al respecto al hablar de las dificultades que nos encontramos durante el uso de Infer.NET, en la sección 5.1.1. Los resultados se encuentran en la tabla 4.4, y los detalles de las corridas presentadas en el apéndice, sección 5.5.4. La performance de nuestra implementación es considerablemente

	Matchbox			
	Infer.NET	Nuestra	*Random Forest*	*Gradient Boosting*
Geometric mean (test)	0.36860	0.63185*	0.52487	0.51826
Geometric mean (train)	0.35864	0.62645	0.63672	0.52974
diferencia de la evidencia en órdenes de magnitud (test)	0	-13473	-8835	-8519
diferencia de la evidencia en órdenes de magnitud (train)	0	-41830	-43050	-29255

Tab. 4.4: Evaluación de modelos con dataset sintético, 100000 puntuaciones (75:25 train:test), sistema de puntuación binario (me gusta/no me gusta)
$*$ sin propagación en historial: con propagación se alcanza una media geométrica de **0.67817**!

mejor que la de la implementación de Infer.NET, e incluso supera a *Gradient Boosting* y *Random Forest* también. Efectivamente, nuestra implementación se comporta como esperábamos que se comportara la implementación de Infer.NET. Esto nos da la pauta de

que el problema del modelo no tiene que ver con el diseño, sino con el estado de la implementación. Dado que los datos fueron sintetizados siguiendo el modelo de la realidad del sistema, era esperable que nuestra implementación (de ser correcta) tuviera un buen desempeño en este dataset. Para entender el desempeño real de Matchbox, veamos si estos mismos resultados se repiten sobre MovieLens.

4.4.3. Comparación sobre MovieLens

Para concluir, evaluamos nuestra propia implementación de Matchbox en el train-test split de MovieLens binarizado antes utilizado, y comparamos los resultados con los de los demás algoritmos. En la tabla 4.5 podemos apreciar que la capacidad predictiva de nuestra implementación de Matchbox supera a todos los demás algoritmos estudiados en test, y a todos salvo *Random Forest* en train. Además, a diferencia de *Random Forest* y *Grandient Boosting*, presenta muy poco *overfitting*. Podemos comprobar entonces que los malos resultados de la implementación de Infer.NET provienen de la implementación particular y no del diseño del algoritmo en sí.

A lo largo de esta tesis reportamos bugs y consultamos dudas sobre el diseño del algoritmo con quienes mantienen el paquete, y los resultados mostrados en este trabajo evolucionaron drásticamente a lo largo de su realización en consecuencia. Dada la complejidad de la implementación de Microsoft y la extensión del trabajo ya realizado, continuar la búsqueda de errores implementativos o mejorar la velocidad de nuestra implementación, que es considerablemente más lenta que la de Microsoft, quedan como trabajo a futuro.

	Matchbox			
	Infer.NET	Nuestra	*Random Forest*	*Gradient Boosting*
Geometric mean (test)	0.39364	0.57190	0.54940	0.55137
Geometric mean (train)	0.39019	0.58102	0.66200	0.57974
diferencia de la evidencia en órdenes de magnitud (test)	0	-9734	-8335	-8424
diferencia de la evidencia en órdenes de magnitud (train)	0	-103674	-114647	-104695

Tab. 4.5: Evaluación de modelos con dataset MovieLens, 100.000 puntuaciones (75:25 train:test), sistema de puntuación binario (me gusta/no me gusta)

5. CONCLUSIONES

En este trabajo investigamos métodos de inferencia eficiente mediante aproximación analítica a través del estudio del sistema de recomendación Matchbox. Documentamos en profundidad la matemática de este algoritmo, lo implementamos y comparamos su desempeño contra otros algoritmos destacados del área.

En el curso de la comparativa encontramos que la única implementación de código abierto de este sistema, provista por el mismo laboratorio que presentó el articulo estudiado, no realiza predicciones adecuadamente. Mediante nuestra propia implementación, pudimos comprobar que la implementación del algoritmo de acuerdo a las especificaciones del artículo original sí resulta en un sistema de recomendación de buen desempeño. Al compararlo con algoritmos robustos de mucho uso en la literatura comprobamos una gran ventaja de Matchbox - en verdad, de los métodos bayesianos en general: la poca o nula aparición del fenómeno de *overfitting*.

Ya estudiamos en la sección 1.2.2 un ejemplo de selección de modelo lineal. A partir de dicho ejemplo mostramos que la aplicación estricta de las reglas de la probabilidad premia a modelos más sencillos sin descartar modelos más complejos que puedan ser necesarios para expresar datos aún por observar, y de esta manera evita sobreajustar la solución a datos ruidosos o que indican un patrón que no tenemos suficiente información para dilucidar aún. Nos sorprendió observar este efecto de manera concreta en los resultados de nuestra implementación: efectivamente, y a diferencia de los demás algoritmos estudiados, Matchbox preserva flexibilidad suficiente para tener un desempeño igual o aún mejor en test que en train. En el dataset sintético, que está generado a partir de un modelo relativamente simple, Matchbox supera en test el desempeño en train. En un dataset real, la media geométrica en test es peor que en train por 1.56 %; en contraste, *Gradient Boosting* muestra una desmejora de 4.86 % en el mejor caso, y *Random Forest* una desmejora del 17 %.

Otra ventaja del sistema propuesto es su capacidad de incorporar información incrementalmente de manera constante - lo que se conoce como funcionamiento *on-line* - por lo que no hay una distinción real entre etapas de "entrenamiento" y "testeo". Esto permite aprovechar el dataset en su totalidad, lo cual es especialmente interesante cuando se cuenta con pocos datos. Se incorpora además la técnica de propagación del historial de puntuaciones, la cual permite maximizar la información obtenida de las observaciones ya realizadas.

5.1. Infer.NET

Lamentablemente, no encontramos una comunidad activa dedicada puntualmente a Infer.NET, pero encontramos que los desarrolladores del paquete (especialmente Tom Minka)

respondían todo tipo de consultas relacionadas al paquete en el *issue tracker* del repositorio. Nuestra primer consulta dio lugar a intercambios de mensajes con Minka a lo largo de todo 2023; detallaremos a continuación las conclusiones de estas comunicaciones. Las *issues* completas pueden encontrarse en https://github.com/dotnet/infer/issues/428 y https://github.com/dotnet/infer/issues/449.

Minka es parte del equipo de investigación de Microsoft que presentó el paper de Matchbox, y es autor del trabajo citado como marco teórico en la sección de mensajes aproximados (mensajes 2 y 7) del paper. Es por esto que pudimos hacer consultas tanto sobre la implementación (sección 5.1.1) en sí como sobre el diseño, puntualmente la construcción de los mensajes (sección 5.1.3).

5.1.1. El tutorial de sistemas de recomendación

Al realizar las comparaciones, estábamos interesados en comparar la clase `RecommenderSystem` con la implementación de Matchbox del tutorial. Este funciona bien para una cantidad de observaciones baja, lo cual comprobamos reproduciendo los resultados del tutorial y verificando que las estimaciones de características latentes resultaran idénticas a las realizadas por nuestra implementación en datasets pequeños. Sin embargo, nos encontramos que al complejizar el modelo (ya fuera subiendo la cantidad de características latentes, la cantidad de umbrales, o la cantidad de puntuaciones observadas) este comenzaba dar predicciones poco confiables. Basta con observar la tabla 5.1: jugando con los hiperparámetros del modelo podemos alcanzar una media geométrica de hasta *0.97203* para 4 umbrales, es decir 5 posibles puntajes: esto no tiene ningún sentido.

Affinity noise variance	thresholds noise variance	user thresholds prior variance	numLevels	geometric mean	bias prior var	trait prior var
1	0	1	1	roto	1.0	1.0
0.1	0.1	1	1	0.44109	1.0	1.0
1	0	1	4	roto	1.0	1.0
0.1	0.1	1	4	0.97203	1.0	1.0
1	0	0	1	0.63499	1.0	1.0
0.1	0.1	0	1	0.57561	1.0	1.0
1	0	0	4	0.33192	1.0	1.0
0.1	0.1	0	4	0.46966	1.0	1.0

Tab. 5.1: Resultados de predicciones realizadas por el tutorial dado un dataset sintético de 100.000 observaciones, con 943 usuarios y 1682 ítems.

Tuvimos un largo intercambio de mensajes en el *issue tracker* de Infer.NET en el intento de conseguir explicar distintos comportamientos extraños del tutorial. Al comenzar este trabajo, era posible reproducir las estimaciones resultantes en el tutorial para 100 observaciones, pero no las de la segunda tabla de resultados de este, entrenada con 20000 observaciones. Esta era una clara inconsistencia en la documentación, así que la reportamos. A partir de este reporte, Minka encontró y arregló *bugs* en el motor de inferencia

de Infer.NET y un caso borde en el producto de Gaussianas. Esos *fixes* hicieron que el tutorial volviera a reproducir los resultados ya presentes en la documentación, lo cual por un lado nos permitió confirmar el buen funcionamiento de nuestra implementación (que ya reproducía los resultados del tutorial).

Es importante notar que estos cambios arreglan las **estimaciones** del modelo, pero no las **estimaciones**. Aunque para los parámetros del ejemplo (cantidad de usuarios, ítems, observaciones, e hiperparámetros dados) el tutorial realiza ahora las estimaciones esperadas y da valores de evidencia razonables, si intentamos subir la cantidad de características latentes modeladas a 5 o más, la evidencia del modelo cae a 0. Comentamos este resultado en la *issue* anterior, y la respuesta que obtuvimos es que al modelar varios gustos latentes se debía incorporar *damping*, una técnica que no es mencionada en el paper de Matchbox y sólo es nombrada en el paper citado como marco teórico para las aproximaciones (22). No tuvimos oportunidad de hacer más consultas al respecto, ya que fue cerrada la issue, pero recalcamos que nos ocurre que la evidencia da números sospechosos o inválidos incluso al mantener fija la cantidad de características latentes y tener una cantidad baja de umbrales, como se puede observar en la primer fila de la tabla 5.1.

Es sumamente extraño que cuando tenemos estimaciones idénticas de características latentes la evidencia de nuestro modelo y la del tutorial sean distintas, ya que las predicciones se derivan de manera directa de las características latentes. No tenemos claro qué clase de error podría afectar a las predicciones pero no a las estimaciones, considerando que las estimaciones se retroalimentan con las predicciones mismas (a través del *likelihood*). Encontramos que, aparentemente, desde 2012 se utiliza la metodología de modelado de características latentes de Matchbox en el sistema de recomendación de la tienda de juegos de Xbox Network (14) Quizás que la parte de estimaciones de características sea la que mejor funciona viene de que esa parte continua en uso y por ende está mantenida. Es imposible afirmar esta teoría, pero nos pareció interesante mencionarla.

5.1.2. La clase `RecommenderSystem`

A pesar de los *bugfixes* anteriores, la clase `RecommenderSystem` de Infer.NET continuó su mal desempeño. Nos quedamos con la impresión de que la implementación del sistema de recomendación no necesariamente está mantenida, considerando que los resultados al momento de escribir la documentación parecen ser correctos. Por conversaciones presentes en el *issue tracker*, se da a entender que Microsoft utiliza Infer.NET internamente para hacer inferencia, pero utilizando únicamente *Expectation Propagation* - originalmente se planeó dar igual soporte a otros algoritmos, lo cual se puede ver a lo largo del código, pero al utilizarse más EP se dió más atención a su mantenimiento, causando que fuera el algoritmo con mayor funcionalidad. En este círculo vicioso, *Variational Inference* terminó quedando desatendido y los desarrolladores en Microsoft pasaron a utilizar otras herramientas para ese caso. Más detalles se pueden encontrar en `https://github.com/dotnet/infer/issues/208`. Sospechamos que en estas idas y vueltas algo en el motor de inferencia del paquete podría haber cambiado de manera tal que *Expectation Propagation* continua funcionando correctamente, pero *Variational Inference*

no. Dado que *Variational Inference* es central para Matchbox, esto afectaría a la implementación de la clase `RecommenderSystem` y a ciertas facetas de la implementación del tutorial. Citando a Minka mismo en la *issue* antes mencionada:

> There isn't any roadmap for Infer.NET. Microsoft folks make changes based on what they need at the time. Folks outside Microsoft can make pull requests anytime for any part of the code or documentation.
>
> The original vision for Infer.NET was to start with 3 inference algorithms [EP, Gibbs and VI] and include more over time. But as people in Microsoft started using Infer.NET, they preferred EP for most models. This created a feedback loop where EP received the most attention and development, making it better and thus even more attractive relative to the other two algorithms. After a certain point, no one in Microsoft was using any inference algorithm besides EP in Infer.NET. There are still plenty of Microsoft products that use EP internally to this day, and they depend on the continual improvements we are making to Infer.NET.
>
> After other PPLs rose up, anyone who wanted to use a different algorithm like ADVI or HMC would just use those other PPLs, so there was no pressure to add those algorithms to Infer.NET (and there still isn't, in my opinion). Essentially what has happened is that people first decide what inference algorithm they want to use, then choose the appropriate PPL.

Creemos que debe ser posible retomar la implementación de Microsoft y llegar a una implementación muy rápida y correcta, pero dicha tarea queda por fuera del tiempo y alcance de esta tesis. Considerando que Microsoft ha quitado Matchbox de su Azure Machine Learning Service moderno, es probable que ya no haya interés por mantener la implementación del lado de Microsoft tampoco. Nosotros disponibilizamos nuestra implementación, creada con el fin de demostrar que el diseño de Matchbox es funcional, pero vale notar que es una implementación sin el nivel de optimización que presenta incluso el tutorial - dejamos como trabajo futuro mejorar esta implementación.

5.1.3. Elección de aproximaciones

Vimos en la sección 3.1 que los mensajes 2 y 7 se aproximan mediante *Variational Message Passing*. Este algoritmo que encontrará, al converger, la Gaussiana que minimice la divergencia KL inversa respecto a la probabilidad conjunta, pero no aquella expresada por el *factor graph* que mostramos, sino la expresada por una refactorización de este. Esta refactorización es necesaria para la aplicación de aproximaciones analíticas por una restricción de los métodos utilizados; en su reporte de divergencias (22) Minka entra en más detalle en los comos y por qués, que no vienen al caso en el presente trabajo.

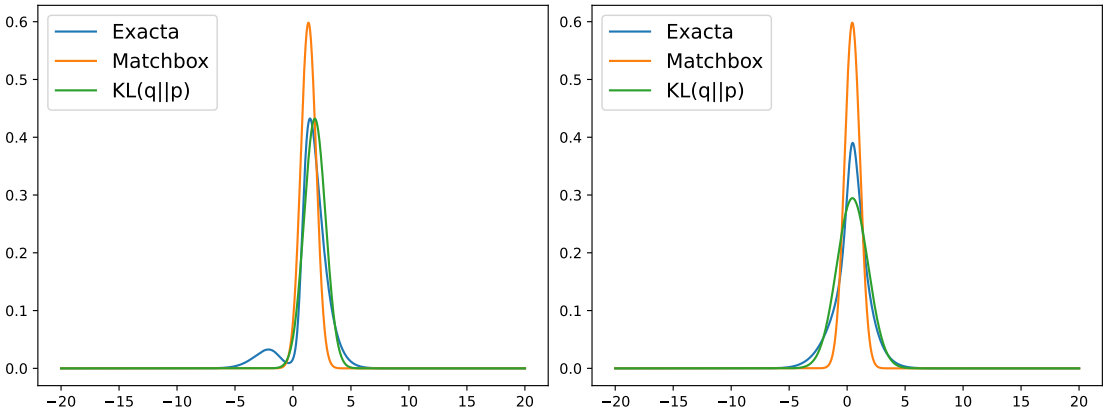
Nosotros contamos con la distribución conjunta exacta a minimizar, aquella expresada por el *factor graph* antes presentado. Encontrar una distribución que aproxime a la distribución conjunta exacta implicará que también se minimicen los posteriors exactos de las variables

del grafo. Por ejemplo, para el prior de s_k, partiendo de la función objetivo a minimizar y siguiendo la notación del reporte de divergencia de Minka de 2005 (22), la cuál está explicada en el apéndice (sección 5.4):

$$\begin{aligned}
\hat{f}_*(s_k) &= \arg\min_{\hat{f}} \left(KL(\hat{f}_*(s_k)\ q^{\setminus *}(s_k) \ ||\ f_*(s_k)\ q^{\setminus *}(s_k)) \right) \\
&\underset{\text{(55 Minka2005)}}{=} \arg\min_{\hat{f}} \left(KL(\hat{f}_*(s_k)\ m_{s_k \to *}(s_k) \ ||\ f_*(s_k)\ m_{s_k \to *}(s_k)) \right) \\
&\underset{\text{(53 Minka2005)}}{=} \arg\min_{\hat{m}} \left(KL(\underbrace{\hat{m}_{* \to s_k}(s_k)}_{\substack{\text{Mensaje 7} \\ \text{aproximado}}}\ \underbrace{m_{s_k \to *}(s_k)}_{\substack{\text{Mensaje 1 a } s_k}} \ ||\ \underbrace{m_{* \to s_k}(s_k)}_{\substack{\text{Mensaje 7} \\ \text{aproximado}}}\ \underbrace{m_{s_k \to *}(s_k)}_{\substack{\text{Mensaje 1 a } s_k}}) \right) \\
&= \arg\min_{\hat{q}} (\ KL(\hat{q}(s_k) \ ||\ q(s_k)))
\end{aligned}$$

Coloquialmente, debería ocurrir que la distribución q que minimiza la divergencia para la probabilidad conjunta minimice también la divergencia entre el prior exacto y aproximado de la variable s_k.

Podemos computar una buena aproximación numérica (no analítica) del mensaje 7 exacto utilizando una grilla. Además, a partir de esta aproximación, podemos encontrar la Gaussiana que minimiza la divergencia KL inversa mediante cualquier técnica de optimización (nosotros utilizamos *Stochastic Gradient Descent*). Nos pareció interesante graficar el posterior exacto y ambas aproximaciones: en la figura 5.1 se puede observar que la aproximación propuesta no se acerca a la encontrada mediante SGD, y que esta última se alinea mejor con el comportamiento esperado de la divergencia KL inversa frente a una distribución bimodal.



a Mensaje 7 bimodal *b* Mensaje 7 unimodal

Fig. 5.1: Mensajes exactos y aproximaciones obtenidas mediante pasaje de mensajes iterativo sobre mensaje 7

En el curso del trabajo comprobamos que la aproximación analítica funciona correctamente, a pesar de que no se estaría realizando la "mejor aproximación posible" de los posteriors correspondientes a inferencia exacta. Incluímos en nuestra implementación la posibilidad

de intercambiar la aproximación de Matchbox por la aproximación numérica del mensaje exacto. Esta implementación no es perfecta - quisimos evaluarla contra Movielens binarizado pero solo pudimos evaluarla hasta 44.000 observaciones, probablemente por algún bug que no encontramos, y el tiempo de ejecución fue sumamente largo. Sin embargo, nos pareció interesante mostrar los resultados parciales, teniendo en cuenta que habría que revisar estos resultados luego de corregir la optimización numérica. Queda como trabajo futuro revisar y mejorar esta primer implementación. En la figura 5.2 podemos observar que los resultados son muy similares, pero la aproximación analítica es ligeramente mejor.

	Aprox. analítica	Aprox. numérica
Geometric mean	0.54726	0.54647
diferencia de la evidencia en órdenes de magnitud	0	64.68

Tab. 5.2: Resultados en dataset Movielens binario 44.000 observaciones (sólo train, sin *smoothing*)

5.2. Conclusiones finales

Es importante tener en cuenta que los principales competidores e innovadores en el área de recomendación (Youtube, TikTok, Instagram (Meta), Amazon, por nombrar algunos) nunca hacen públicos los detalles completos de sus sistemas de recomendación, tratándolos como secretos del oficio. En general, se sabe que los sistemas de recomendación de estas compañías suelen conformarse de varios algoritmos que optimizan distintas métricas y se componen para generar las recomendaciones finales. Aunque conseguir el detalle de los sistemas completos de una de estas grandes compañias es prácticamente imposible, en algunos casos podemos encontrar detalles de las técnicas utilizadas en algún componente de estos sistemas. Por ejemplo, un reporte publicado por Amazon (24) revela como ellos modelan la probabilidad de realizar compras repetidas utilizando Bayes empírico. El año pasado X (antes Twitter) publicó un repositorio mostrando la estructura de su sistema de recomendación, y mostrando la implementación de uno de los varios algoritmos que lo componen. En general, varias compañias han realizado publicaciones “revelando el secreto de su algoritmo”, pero en verdad estas no hacen más que contar a grandes rasgos algunas de las métricas tenidas en cuenta para ciertos aspectos de sus recomendaciones, sin ningún detalle técnico que nos permita auditar los sesgos e intereses que dictan qué contenido nos es presentado día a día.

Dada la actual falta de transparencia de la industria, en el área de recomendación se pone especial énfasis en conocer tanto técnicas clásicas como revisar el estado del arte en la academia constantemente, ya que esto permite intuir qué técnicas pueden estar siendo utilizadas en la industria. Un gran lugar para medir el pulso de la escena de recomendaciones son las competencias de recomendación. La más importante fue, sin lugar a dudas, el

Netflix Prize (2007-2009), una de las primeras de su tipo. Aunque ya han pasado muchos años desde su finalización, es innegable el impacto que tuvo en el área, y muchas de las técnicas popularizadas por los equipos ganadores de aquel momento se siguen usando hoy en día. El equipo ganador del Progress Prize de 2007, BellKor, muestra en el reporte de su solución de dicho año que su algoritmo de recomendación ganador también es un sistema compuesto. Más aún, el algoritmo ganador del Grand Prize de la competencia, *BellKor's Pragmatic Chaos*, está conformado por 104 predictores combinados mediante una red neuronal de una capa. Volviendo a la solución de BellKor de 2007, esta fue adaptada y llevada a producción en Netflix, de acuerdo a una serie de posts en su blog de tecnología (1). El reporte técnico de BellKor (2) revela que uno de los componentes de este sistema es un modelo de estimación de factores latentes, a los cuales modela utilizando distribuciones normales, mediante factorización de matrices. Si esto suena familiar, efectivamente, esta estrategia es idéntica a la de Matchbox. La diferencia se encuentra, por supuesto, en las técnicas utilizadas - para realizar las estimaciones no utiliza técnicas bayesianas, sino que realiza selecciónd de parámetros mediante *Gibbs Sampling* y *Expectation Maximization*. Pero sería posible reemplazar este mismo componente en la solución de BellKor por Matchbox - cuánto sentido tiene esto dependerá del dominio de recomendación y los objetivos puntuales de la solución, pero podría ser un experimento interesante para realizar a futuro.

Matchbox es una implementación relativamente sencilla de *collaborative filtering* y recomendaciones personalizadas mediante modelado causal probabilístico. A pesar de ser superada por otras técnicas de punta actuales, es un punto de partida valioso para el aprendizaje de técnicas bayesians, modelado gráfico, y modelos de estimación de factores latentes mediante factorización de matrices, los cuales son muy populares en el mundo de *collaborative filtering* especialmente a partir del Netflix Prize. Destacamos particularmente el aprendizaje que logramos durante la realización de este trabajo sobre aproximaciones analíticas para modelos con pasaje de mensajes. Exploramos sus fortalezas y debilidades, comprendiendo mejor en qué casos pueden ser beneficiosas (cuando la distribución conjunta es completamente factorizable, por ejemplo) pero también ganando consciencia sobre su inflexibilidad: en el caso de Matchbox, en que la mayoría de los mensajes son Gaussianos y solo dos no, ya no es posible minimizar sobre el factor graph sobre el que se hace pasaje de mensajes, sino que hay que recurrir a uno simplificado. En el futuro, nos gustaría explorar los métodos de aproxiamción por sampleo, puntualmente Monte Carlo, que parecen ser los más utilizados al momento de la escritura de esta tesis.

En lineas generales, en el área de recomendación las redes bayesianas no ganan hoy en día a redes profundas en cuanto a performance, pero sí en ajustabilidad y explicabilidad. Por dar un ejemplo, en lo que refiere a retención de clientes, saber por qué ciertos clientes dejan de hacer compras en nuestro portal es tanto o más importante que predecir qué clientes es más probable dejen de hacerlo. Una red bayesiana con peor desempeño puede de todas maneras llevar a más acciones más acertadas que un sistema que predice mejor pero no nos da pautas sobre las que podamos actuar.

En lo que respecta a explicabilidad, la necesidad hoy es clara: tan solo en los dos años de realización de esta tesis, de 2022 a 2024, el lugar que ocupa la inteligencia artificial en

nuestras vidas cambió radicalmente con la presentación de herramientas como ChatGPT y DALL-E. A medida que se continua trabajando en legislación que obligue transparencia en lo que respecta a recomendaciones, se volverá necesario encontrar nuevas técnicas que permitan realizar recomendaciones eficientes y explicables simultáneamente.

5.3. Trabajo a futuro

El producto de esta tesis sienta las bases para que se realice en lo inmediato una implementación optimizada de Matchbox, ya sea partiendo de nuestra implementación o de la de Infer.NET. Además propone una aproximación alternativa para el sistema de recomendación estudiado, la cual se podría retomar y mejorar en un trabajo posterior. A partir del aprendizaje realizado en este trabajo, se podrá continuar el estudio de métodos de aproximación para poder aportar en esta área incipiente. Por último, queda abierta la posibilidad de explorar modelos causales alternativos para sistemas de puntuación con umbrales personalizados.

5. REFERENCIAS

[1] Amatriain, X.: Netflix recommendations: Beyond the 5 stars (part 2) (2012), https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-2-d9b96aa399f5

[2] Bell, R.M., Koren, Y., Volinsky, C.: The bellkor solution to the netflix prize (2007), https://api.semanticscholar.org/CorpusID:11877045

[3] Bennet, J., Lanning, S.: The netflix prize. In: Proceedings of the KDD Cup Workshop 2007. ACM (2007), http://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf

[4] Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, 1 edn. (2007), http://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0387310738

[5] Christakopoulou, K., Radlinski, F., Hofmann, K.: Towards conversational recommender systems. vol. 13-17-August-2016, p. 815 – 824 (2016), https://www.scopus.com/inward/record.uri?eid=2-s2.0-84984985159&doi=10.1145%2f2939672.2939746&partnerID=40&md5=2586e3f785fb6d56f7aac9ae375032c7, cited by: 268

[6] Dangauthier, P., Herbrich, R., Minka, T., Graepel, T.: Trueskill through time: Revisiting the history of chess. In: Platt, J.C., Koller, D., Singer, Y., Roweis, S.T. (eds.) Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems. pp. 337–344. Curran Associates, Inc. (2007), https://proceedings.neurips.cc/paper/2007/file/9f53d83ec0691550f7d2507d57f4f5a2-Paper.pdf

[7] Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine Learning 29(2-3), 131 – 163 (1997), https://www.scopus.com/inward/record.uri?eid=2-s2.0-0031276011&doi=10.1023%2fa%3a1007465528199&partnerID=40&md5=fc2314843318a55a209feba0b743dd35, cited by: 3987; All Open Access, Bronze Open Access

[8] Funk, S.: Netflix update: Try this at home (2006), https://sifter.org/simon/journal/20061211.html

[9] Graepel, T., Quiñonero Candela, J., Borchert, T., Herbrich, R.: Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In: Proceedings of the 27th International Conference on Machine Learning ICML 2010, Invited Applications Track (unreviewed, to appear) (June 2010), https://www.microsoft.com/en-us/research/publication/web-scale-bayesian-click-through-rate-prediction-for-sponsored-search-advertising-in-microsofts-bing-search-engine/

[10] Ho, T.K.: Random decision forests. In: Proceedings of 3rd International Conference on Document Analysis and Recognition. vol. 1, pp. 278–282 vol.1 (1995)

[11] Infer.NET: Recommender system tutorial (2018), https://dotnet.github.io/infer/userguide/Recommender%20System.html

[12] Kasneci, G., Van Gael, J., Stern, D., Graepel, T.: Cobayes: Bayesian knowledge corroboration with assessors of unknown areas of expertise. p. 465 – 474 (2011), https://www.scopus.com/inward/record.uri?eid=2-s2.0-79952398813&doi=10.1145%2f1935826.1935896&partnerID=40&md5=dd4b851ac6a354f9924a87151668fae8, cited by: 34

[13] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. Advances in neural information processing systems 30, 3146–3154 (2017)

[14] Koenigstein, N., Nice, N., Paquet, U., Schleyen, N.: The xbox recommender system (09 2012)

[15] Koren, Y.: Factorization meets the neighborhood: A multifaceted collaborative filtering model. p. 426 – 434 (2008), https://www.scopus.com/inward/record.uri?eid=2-s2.0-65449121157&doi=10.1145%2f1401890.1401944&partnerID=40&md5=5e52057856b4f3eb4b28daafcc942c1c, cited by: 3288

[16] Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer 42(8), 30–37 (2009)

[17] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25 (2012)

[18] Landfried, G.: Análisis bayesiano del aprendizaje en comunidades de video juegos (2023)

[19] Liu, Q., Wang, D.: Stein variational gradient descent: A general purpose bayesian inference algorithm (2019)

[20] Liu, X., Xue, W., Xiao, L., Zhang, B.: PBODL : Parallel bayesian online deep learning for click-through rate prediction in tencent advertising system. CoRR abs/1707.00802 (2017), http://arxiv.org/abs/1707.00802

[21] Minka, T., Winn, J., Guiver, J., Zaykov, Y., Fabian, D., Bronskill, J.: Infer.NET 0.3 (2018), http://dotnet.github.io/infer, microsoft Research Cambridge

[22] Minka, T., et al.: Divergence measures and message passing. Tech. rep., Technical report, Microsoft Research (2005), https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2005-173.pdf

[23] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011)

[24] Rahul Bhagat, Alex Lobzhanidze, S.M., Vishwanath, S.: Buy It Again: Modeling Repeat Purchase Recommendations (2018), https://assets.amazon.science/40/e5/89556a6341eaa3d7dacc074ff24d/buy-it-again-modeling-repeat-purchase-recommendations.pdf

[25] Stern, D., Herbrich, R., Graepel, T.: Matchbox: Large scale bayesian recommendations. In: Proceedings of the Eighteenth International World Wide Web Conference (January 2009), https://www.microsoft.com/en-us/research/publication/matchbox-large-scale-bayesian-recommendations/

[26] Wan, Y., Nakayama, M.: A sentiment analysis of star-rating: a cross-cultural perspective (01 2022)

[27] Zaykov, Y.: The microsoft infer.net machine learning framework goes open source (2018), https://www.microsoft.com/en-us/research/blog/the-microsoft-infer-net-machine-learning-framework-goes-open-source/

[28] Zhang, C., Guiver, J., Minka, T., Zaykov, Y.: Groupbox: A generative model for group recommendation. Tech. Rep. MSR-TR-2015-61 (July 2015), https://www.microsoft.com/en-us/research/publication/groupbox-a-generative-model-for-group-recommendation/

5. APÉNDICE

5.4. General Message-Passing algorithm

El funcionamiento general de los algoritmos de pasaje de mensajes es el siguiente: partimos de una distribución p y queremos hallar $q \in \mathcal{F}$ tal que minimice $D(p||q)$ donde D es alguna medida de divergencia. Factorizamos a $p(x) = \prod_a f_a(\boldsymbol{x})$, como hacíamos para construir un *factor graph*. No hay una única factorización para p, pero cuál elijamos definirá como queda dividido el grafo sobre el que hacemos pasaje de mensajes. Dada la factorización, vamos a aproximar a cada factor con un miembro de $\mathcal{F}$, de manera que al multiplicar todas estas aproximaciones obtengamos una distribución $q \in \mathcal{F}$ tal que $D(p||q)$ sea pequeño. Como cada mensaje (y por lo tanto cada aproximación) depende del resto de la red, se genera una dependencia cíclica. Esta se puede resolver iterativamente, haciendo que cada factor envíe su aproximación al resto de la red y luego recompute su aproximación a partir de los mensajes que recibe. Si definimos $q^{\backslash a}$ como el producto de todos los factores aproximados menos f_a:

$$q^{\backslash a}(\boldsymbol{x}) = q(\boldsymbol{x})/\hat{f}_a(\boldsymbol{x}) = \prod_{b \neq a} \hat{f}_b(\boldsymbol{x}) \tag{5.1}$$

y de manera análoga podemos definir $p^{\backslash a}(\boldsymbol{x}) = \prod_{b \neq a} f_a(\boldsymbol{x})$, hallar la mejor aproximación $\hat{f}_a$ se reduce a:

$$\hat{f}_a(\boldsymbol{x}) = \text{argmin} D(f_a(\boldsymbol{x}) q^{\backslash a}(\boldsymbol{x}) \,||\, \hat{f}_a(\boldsymbol{x}) q^{\backslash a}(\boldsymbol{x})) \tag{5.2}$$

Los factores aproximados se pueden, a su vez, descomponer en mensajes $m_{f_a \to i}$ enviados del factor a a la variable x_i:

$$\hat{f}(\boldsymbol{x}) = \prod_i m_{f_a \to x_i}(x_i) \tag{5.3}$$

Estos mensajes no necesariamente están normalizados ni necesitan ser distribuciones. Los mensajes que envían las variables a los factores se definen:

$$m_{x_i \to f_a}(x_i) = \prod_{b \neq a} m_{f_b \to x_i}(x_i) \tag{5.4}$$

Y por lo tanto $q^{\backslash a}(\boldsymbol{x})$ se puede redefinir como:

$$\begin{aligned} q^{\backslash a}(\boldsymbol{x}) &\overset{(5.1)}{=} \prod_{b \neq a} \hat{f}_b(\boldsymbol{x}) \\ &\overset{(5.3)}{=} \prod_{b \neq a} \prod_i m_{f_b \to x_i}(x_i) \\ &\overset{(5.4)}{=} \prod_i m_{x_i \to f_a}(x_i) \end{aligned}$$

Minka demuestra que para realizar inferencia basta con propagar los mensajes entre un factor y las variables que utiliza.

La técnica de pasaje de mensajes aproximado parte de la suposición de que minimizar las divergencias locales $D(f_a(\boldsymbol{x})q^{\backslash a}(\boldsymbol{x}) \,||\, \hat{f}_a(\boldsymbol{x})q^{\backslash a}(\boldsymbol{x}))$ es una buena aproximación a minimizar la divergencia global $D(p||q)$.

La divergencia KL inversa $KL(q||p)$ tiene la propiedad especial de que la divergencia global y local son equivalentes, es decir minimizar las divergencias locales se corresponde exactamente a minimizar la divergencia global. La divergencia $KL(p||q)$ no cuenta con tal propiedad; minimizar las divergencias locales da una **aproximación** a minimizar la divergencia global.

5.5. Experimentos

En esta sección daremos los detalles de los experimentos realizados, para que sea posible reproducirlos. Se provee el código de estos experimentos como parte de este trabajo.

5.5.1. Historial de puntuaciones

En un sistema limpio, modelando 4 características de usuario/película y con sistema de puntuación binario ("me gusta"/"no me gusta") se agregan los siguientes eventos:

- El usuario 100 puntua las películas 100,101,102 y 103 alternadamente positiva y negativamente. Esto es únicamente para evitar que las películas que se usen para romper simetría sean las próximas puntuadas.
- Dos usuarios puntuan las películas 1 y 2 de manera opuesta: el usuario 1 puntua (positivo, negativo) y el usuario 2 (negativo, positivo).
- El usuario 1 puntua otras 4 películas, en las cuales el prior de su primer característica es $N(3, 0.5)$, positivamente.
- El usuario 1 puntua otras 4 películas, en las cuales el prior de su primer característica es $N(-3, 0.4)$, negativamente.
- El usuario 3, que es nuevo, puntua las últimas 8 películas de manera idéntica al usuario 1.

Hecho esto, miramos cual era la probabilidad de recomendar las películas 1 y 2 al usuario 3. Esta probabilidad es equivalente, en este caso, a la probabilidad de que la película sea puntuada negativamente. Para hacer esto, incorporamos al sistema los eventos en que el usuario 3 puntua la película 1 de manera positiva y la 1 de manera negativa (de manera idéntica al usuario 1) y observamos la evidencia resultante. También observamos el posterior sobre la primer carácterística de ambas películas luego de realizar las puntuaciones.

5.5.2. Espacios de hiperparámetros para comparaciones

Los espeacios de hiperparámetros considerados para los algoritmos mostrados en las comparaciones son los siguientes:

Matchbox

- **traitCount**: hp.quniform(4, 10, 1)
- **iterationCount**:hp.quniform(5, 30, 5)
- **UserTraitFeatureWeightPriorVariance**: hp.choice([0.25,0.5,0.75,1])
- **ItemTraitFeatureWeightPriorVariance**: hp.choice([0.25,0.5,0.75,1])
- **ItemTraitVariance**: hp.choice([0.25,0.5,0.75,1])
- **UserTraitVariance**: hp.choice([0.25,0.5,0.75,1])
- **AffinityNoiseVariance**: 1.0

Random Forest

- **bootstrap**: hp.choice([True, False])
- **n_estimators**: hp.quniform(100, 500, 100)
- **max_features**: "sqrt"
- **max_depth**: hp.quniform(10, 110, 10)
- **min_samples_split**: hp.choice([2, 5, 10])
- **min_samples_leaf**: hp.choice([1, 2, 4])

LGBM

- **n_estimators**: hp.quniform(100, 500, 100)
- **subsample**: hp.quniform(0.7, 0.90, 0.02)
- **learning_rate**: hp.qloguniform(np.log(0.04), np.log(0.17), 0.01)
- **reg_alpha**: hp.choice([0, hp.quniform('reg_alpha', 0.01, 0.1, 0.01)])

Implementación propia

Dado que nuestra implementación no está optimizada y tarda varias horas en correr (específicamente, en converger) para un dataset de este tamaño, la evaluamos contra un único set de hiperparámetros que encontramos eran de buen funcionamiento durante la realización de la implementación. De todos modos, con este único set cumplimos nuestros objetivos de demostrar el buen funcionamiento de Matchbox y superar la performance de los demás algoritmos. Los parámetros utilizados son los siguientes:

trait count	num thresholds	observational noise^2	bias noise^2	threshold noise^2
4	1	1	0	0

5.5.3. Comparaciones sobre MovieLens

En las siguientes tablas, `loss` y `train loss` se corresponden al cálculo hecho por `log_loss` de la librería `sklearn`, que es el logaritmo de la evidencia negado y normalizado. `geo mean` y `geo mean train` se refieren a la media geométrica en test y train respectivamente, como las reportamos a lo largo del trabajo. `data_size` indica el tamaño de dataset de train + test y `status` que la corrida para ese conjunto de hiperparámetros fue exitosa. El resto de los hiperparámetros de las tablas se corresponden a hiperparámetros de las implementaciones correspondientes, cuyo significado se puede consultar en las documentaciones oficiales o en el cuerpo de este trabajo según corresponda.

5.5.3.1. Resultados extra - sistema de puntuación de estrellas

Mathbox (Infer.NET `RecommenderSystem`)

rmse	loss	train loss	Affinity Noise Var	Item Trait Feature Weight Prior Var	Item trait Var	User Trait Feature Weight Prior Var	User Trait Var	iteration count	trait count	min rating	max rating	train time (s)	status	geo mean	geo mean train	data size
2.27948	1.90422	1.91346	1.0	0.25	0.25	0.25	0.25	25.0	4.0	5	296.19119	ok	0.14894	0.14894	99999	
2.26632	1.93181	1.94557	1.0	0.75	0.25	0.25	0.25	25.0	7.0	5	301.9789	ok	0.14489	0.14291	99999	
2.29308	1.94966	1.95965	1.0	0.25	0.5	0.5	0.75	25.0	4.0	5	294.16568	ok	0.14232	0.14091	99999	
2.29944	1.95479	1.96484	1.0	1.0	0.5	0.25	1.0	25.0	4.0	5	296.81226	ok	0.14159	0.14018	99999	
2.28064	1.97866	1.9967	1.0	0.25	0.75	1.0	0.5	10.0	7.0	5	118.50594	ok	0.13825	0.13578	99999	
2.30708	1.9815	1.99204	1.0	0.25	1.0	0.5	1.0	20.0	5.0	5	238.5939	ok	0.13786	0.13642	99999	
2.28268	1.98798	2.00237	1.0	0.25	0.5	1.0	0.5	25.0	7.0	5	298.68455	ok	0.13697	0.13501	99999	
2.2816	2.02776	2.0514	1.0	0.25	0.75	0.5	0.75	10.0	10.0	5	119.55536	ok	0.13163	0.12856	99999	
2.27908	2.03458	2.05584	1.0	1.0	0.5	0.75	1.0	25.0	9.0	5	299.96985	ok	0.13074	0.12799	99999	
2.27828	2.04181	2.0649	1.0	0.25	0.5	0.5	1.0	15.0	10.0	5	183.24074	ok	0.12979	0.12683	99999	

Random Forest

rmse	loss	train loss	bootstrap	max depth	max features	min samples leaf	min samples split	n estimators	train time	status	geo mean	geo mean train	data size
1.32628	1.339	0.82054	True	30	sqrt	1	10	500	49.45648	ok	0.26211	0.44019	99999
1.32104	1.33938	0.88303	True	80	sqrt	4	5	300	29.39363	ok	0.26201	0.41353	99999
1.3284	1.33995	0.83854	True	80	sqrt	2	10	400	40.79138	ok	0.26186	0.43234	99999
1.33764	1.3541	0.78896	False	20	sqrt	2	5	400	50.02835	ok	0.25818	0.45432	99999
1.33496	1.35418	0.78981	False	20	sqrt	2	5	300	37.66318	ok	0.25816	0.45393	99999
1.29488	1.38207	1.33816	False	10	sqrt	4	5	200	15.05727	ok	0.25106	0.26233	99999
1.38024	1.38254	0.64228	False	90	sqrt	2	10	300	42.3683	ok	0.25094	0.52609	99999
1.39944	1.38514	0.58276	True	40	sqrt	1	5	400	44.4551	ok	0.25029	0.55835	99999
1.47072	1.49192	0.38836	False	60	sqrt	2	2	200	30.31494	ok	0.22494	0.67817	99999
1.58444	2.5229	0.0	False	110	sqrt	1	2	300	51.40289	ok	0.08023	1.0	99999

LGBM

rmse	loss	train loss	learn rate	n estimators	reg alpha	subsample	train time	status	geo mean	geo mean train	data size
1.30836	1.32147	1.17172	0.16	200	0.04	0.86	1.62525	ok	0.26674	0.30983	99999
1.29792	1.32233	1.18252	0.1	300	0.0	0.74	2.59817	ok	0.26651	0.30651	99999
1.28804	1.3225	1.20792	0.06	400	0.0	0.86	3.20242	ok	0.26647	0.29882	99999
1.29912	1.32384	1.20811	0.12	200	0.1	0.88	1.75478	ok	0.26611	0.29876	99999
1.2788	1.3247	1.22765	0.05	400	0.08	0.78	3.59309	ok	0.26588	0.29298	99999
1.28192	1.3256	1.22787	0.05	400	0.03	0.88	3.53275	ok	0.26564	0.29292	99999
1.33192	1.32789	1.07067	0.16	400	0.07	0.72	4.00853	ok	0.26504	0.34278	99999
1.33928	1.32967	1.07081	0.16	400	0.0	0.8	3.12016	ok	0.26457	0.34273	99999
1.27684	1.33397	1.27004	0.12	100	0.07	0.72	1.39754	ok	0.26343	0.28082	99999
1.27128	1.33477	1.27216	0.04	300	0.0	0.78	2.63336	ok	0.26322	0.28023	99999

5.5.3.2. Binarización de MovieLens

Para binarizar el dataset asignamos a cada evento un puntaje de 1 ("me gusta") si el puntaje original era de 4 o más estrellas, y un puntaje de 0 ("no me gusta") si era menor.

5.5.3.3. Resultados extra - sistema de puntuación binario

Mathbox (Infer.NET `RecommenderSystem`)

rmse	loss	train loss	Affinity Noise Var	Item Trait Feature Weight Prior Var	Item trait Var	User Trait Feature Weight Prior Var	User Trait Var	iteration count	trait count	min rating	max rating	train time (s)	status	geo mean	geo mean train	data size
0.49056	0.93232	0.94112	1.0	0.75	0.25	1.0	0.75	10.0	4.0	0	1	19.05924	ok	0.39364	0.39019	99999
0.49088	0.93487	0.94918	1.0	0.25	0.5	0.5	0.25	10.0	8.0	0	1	23.32073	ok	0.39264	0.38706	99999
0.4918	0.94021	0.94944	1.0	1.0	1.0	0.5	0.75	5.0	5.0	0	1	10.20606	ok	0.39055	0.38696	99999
0.49068	0.95872	0.96905	1.0	0.5	0.75	1.0	0.5	10.0	6.0	0	1	21.09617	ok	0.38339	0.37944	99999
0.49084	0.96312	0.9792	1.0	0.25	0.25	0.25	1.0	25.0	9.0	0	1	59.76024	ok	0.3817	0.37561	99999
0.49032	0.96812	0.9833	1.0	0.75	0.5	0.5	1.0	10.0	7.0	0	1	22.24057	ok	0.3798	0.37407	99999
0.48988	0.96966	0.98003	1.0	0.5	1.0	0.25	0.5	20.0	6.0	0	1	42.02173	ok	0.37921	0.3753	99999
0.49264	0.97904	0.99597	1.0	0.25	1.0	0.25	0.75	10.0	8.0	0	1	23.03694	ok	0.37567	0.36936	99999
0.49148	0.98382	0.99998	1.0	0.75	0.5	1.0	1.0	25.0	9.0	0	1	60.38235	ok	0.37388	0.36789	99999
0.48996	0.9902	1.00488	1.0	0.5	1.0	0.75	1.0	25.0	7.0	0	1	54.42108	ok	0.3715	0.36609	99999

Random Forest

rmse	loss	train loss	bootstrap	max depth	max features	min samples leaf	min samples split	n estimators	train time	status	geo mean	geo mean train	data size
0.32492	0.59892	0.41249	True	90	sqrt	4	5	400	35.91334	ok	0.5494	0.662	99999
0.32548	0.59948	0.4128	True	60	sqrt	4	5	200	17.85591	ok	0.5491	0.66179	99999
0.32448	0.59973	0.38497	True	100	sqrt	2	10	200	18.18736	ok	0.54896	0.68047	99999
0.33168	0.60952	0.33565	False	90	sqrt	4	10	500	62.35939	ok	0.54361	0.71488	99999
0.33336	0.61188	0.32242	False	50	sqrt	4	5	500	62.58493	ok	0.54233	0.72439	99999
0.33196	0.61363	0.27634	True	80	sqrt	1	5	300	29.31731	ok	0.54138	0.75855	99999
0.35932	0.63195	0.61399	False	10	sqrt	1	10	400	28.48466	ok	0.53156	0.54119	99999
0.34216	0.64516	0.17614	False	90	sqrt	2	2	400	54.05696	ok	0.52458	0.8385	99999
0.34272	0.64741	0.17639	False	80	sqrt	2	2	200	26.96662	ok	0.5234	0.83829	99999
0.35388	0.83532	0.00011	False	50	sqrt	1	2	200	28.401	ok	0.43374	0.99989	99999

LGBM

rmse	loss	train loss	learn rate	n est-imators	reg alpha	sub-sample	train time	status	geo mean	geo mean train	data size
0.31636	0.59535	0.54518	0.13	400	0.0	0.86	0.65729	ok	0.55137	0.57974	99999
0.31636	0.59535	0.54518	0.13	400	0.0	0.84	0.65502	ok	0.55137	0.57974	99999
0.32088	0.59579	0.53424	0.16	400	0.0	0.88	0.70965	ok	0.55113	0.58612	99999
0.31952	0.59618	0.55244	0.11	400	0.0	0.88	0.89685	ok	0.55091	0.57555	99999
0.3206	0.59694	0.5577	0.13	300	0.05	0.74	0.57415	ok	0.55049	0.57252	99999
0.32264	0.59912	0.5732	0.09	300	0.07	0.72	0.5328	ok	0.5493	0.56372	99999
0.32304	0.60309	0.58468	0.05	400	0.1	0.84	0.74433	ok	0.54712	0.55729	99999
0.32388	0.60322	0.5852	0.05	400	0.05	0.9	0.72633	ok	0.54705	0.557	99999
0.32536	0.60679	0.59476	0.14	100	0.0	0.84	0.20326	ok	0.5451	0.55169	99999
0.33236	0.61279	0.60484	0.05	200	0.04	0.74	0.42034	ok	0.54184	0.54616	99999

5.5.4. Comparaciones sobre dataset sintético

5.5.4.1. Generación de dataset sintético

El dataset sintético fue generado a partir del código de generación de datos del tutorial de sistemas de recomendación de Infer.NET.

Los parámetros utilizados para la generación del dataset de **5 estrellas** son los siguientes:

- 943 usuarios
- 1682 ítems
- 4 características latentes
- 100.000 observaciones
- 1 umbral
- Varianza de prior de características latentes: 1
- Varianza de ruido observacional: 1
- Varianza de ruido observacional para umbrales: 0

Los parámetros utilizados para la generación del dataset **binario** son los siguientes:

- 943 usuarios
- 1682 ítems
- 2 características latentes
- 100.000 observaciones
- 1 umbral
- Varianza de prior de características latentes: 1

- Varianza de ruido observacional: 1
- Varianza de ruido observacional para umbrales: 0

5.5.4.2. Resultados extra - sistema de puntuación de estrellas

Mathbox (Infer.NET `RecommenderSystem`)

rmse	loss	train loss	Affinity Noise Var	Item Trait Feature Weight Prior Var	Item trait Var	User Trait Feature Weight Prior Var	User Trait Var	iter-ation count	trait count	min rating	max rating	train time (s)	status	geo mean	geo mean train	data size
5.14	2.67488	2.71019	1.0	0.25	0.25	0.25	0.25	10.0	4.0	1	5	80.78189	ok	0.06891	0.06652	100000
5.11424	2.68932	2.72671	1.0	0.5	0.25	0.25	0.25	20.0	6.0	1	5	166.04595	ok	0.06793	0.06543	100000
5.0904	2.84173	2.88419	1.0	0.5	0.25	0.25	0.5	10.0	4.0	1	5	80.79756	ok	0.05832	0.0559	100000
5.0628	2.84505	2.89222	1.0	0.75	0.5	1.0	0.25	15.0	5.0	1	5	123.01597	ok	0.05813	0.05545	100000
5.03692	2.99512	3.05226	1.0	0.75	1.0	0.5	0.25	15.0	6.0	1	5	123.25533	ok	0.05003	0.04725	100000
5.03112	3.11543	3.17462	1.0	0.75	0.75	0.25	0.5	20.0	7.0	1	5	167.91639	ok	0.04436	0.04181	100000
5.02172	3.15823	3.22751	1.0	0.75	0.75	0.5	0.75	15.0	8.0	1	5	126.4503	ok	0.0425	0.03966	100000
5.02592	3.20435	3.26814	1.0	0.75	0.75	0.25	0.75	25.0	7.0	1	5	208.72796	ok	0.04059	0.03808	100000
5.0494	3.25857	3.3045	1.0	0.75	1.0	0.25	1.0	10.0	4.0	1	5	80.11921	ok	0.03844	0.03672	100000
4.99476	3.28276	3.35944	1.0	1.0	1.0	0.25	1.0	30.0	9.0	1	5	255.50718	ok	0.03752	0.03475	100000

Random Forest

rmse	loss	train loss	boots-trap	max depth	max features	min samples leaf	min samples split	n estim-ators	train time	status	geo mean	geo mean train	data size
4.20848	1.49368	1.01968	True	70	sqrt	4	10	500	54.19208	ok	0.22455	0.36071	100000
4.51984	1.49645	1.24531	True	20	sqrt	2	10	200	18.09265	ok	0.22392	0.28785	100000
4.55076	1.49834	1.24576	True	20	sqrt	2	10	100	9.02973	ok	0.2235	0.28772	100000
4.17176	1.5003	0.98556	True	60	sqrt	4	2	400	44.22117	ok	0.22306	0.37323	100000
4.53252	1.50614	1.17307	True	20	sqrt	2	5	100	9.21019	ok	0.22176	0.30942	100000
4.1774	1.51749	0.86672	True	30	sqrt	2	5	300	33.53179	ok	0.21926	0.42033	100000
4.08312	1.56406	0.70471	True	50	sqrt	2	2	300	36.19303	ok	0.20928	0.49425	100000
4.16784	1.57368	0.84	False	100	sqrt	4	10	400	65.44772	ok	0.20728	0.43171	100000
4.19584	1.59382	0.82945	False	40	sqrt	4	5	400	65.86373	ok	0.20315	0.43629	100000
4.06756	2.01322	0.44249	False	100	sqrt	2	2	500	90.87217	ok	0.13356	0.64244	100000

LGBM

rmse	loss	train loss	learn rate	n est-imators	reg alpha	sub-sample	train time	status	geo mean	geo mean train	data size
4.85628	1.51962	1.45963	0.07	200	0.08	0.84	1.68691	ok	0.2188	0.23232	100000
4.81736	1.52017	1.43795	0.04	500	0.1	0.7	3.91215	ok	0.21868	0.23741	100000
4.786	1.52032	1.43369	0.07	300	0.02	0.78	2.34622	ok	0.21864	0.23843	100000
4.81184	1.52035	1.43354	0.07	300	0.0	0.8	2.31108	ok	0.21864	0.23846	100000
4.9022	1.52048	1.46805	0.04	300	0.09	0.88	2.54791	ok	0.21861	0.23038	100000
4.7708	1.5219	1.42435	0.06	400	0.05	0.82	3.09476	ok	0.2183	0.24066	100000
4.75352	1.52537	1.3982	0.11	300	0.07	0.72	2.29833	ok	0.21754	0.24704	100000
5.10016	1.52667	1.50435	0.05	100	0.02	0.86	0.91956	ok	0.21726	0.22216	100000
4.69432	1.53374	1.36805	0.15	300	0.0	0.78	2.16123	ok	0.21573	0.2546	100000
4.66492	1.54636	1.33007	0.13	500	0.05	0.82	3.5825	ok	0.21302	0.26446	100000

5.5.4.3. Resultados extra - sistema de puntuación binario

Mathbox (Infer.NET `RecommenderSystem`)

rmse	loss	train loss	Affinity Noise Var	Item Trait Feature Weight Prior Var	Item trait Var	User Trait Feature Weight Prior Var	User Trait Var	iteration count	trait count	min rating	max rating	train time (s)	status	geo mean	geo mean train	data size
0.48072	0.99804	1.02543	1.0	1.0	0.25	0.5	0.25	30.0	5.0	0	1	59.33511	ok	0.3686	0.35864	100000
0.4812	1.05991	1.09331	1.0	0.25	0.25	0.25	0.5	20.0	5.0	0	1	39.38314	ok	0.34649	0.3351	100000
0.48052	1.06942	1.11324	1.0	1.0	0.25	0.5	1.0	20.0	10.0	0	1	49.08685	ok	0.34321	0.32849	100000
0.48	1.07508	1.11476	1.0	1.0	0.25	0.25	0.75	30.0	7.0	0	1	64.41936	ok	0.34127	0.32799	100000
0.48056	1.07518	1.11512	1.0	0.5	0.25	0.5	0.75	15.0	6.0	0	1	31.0706	ok	0.34124	0.32788	100000
0.4802	1.08868	1.13791	1.0	1.0	0.75	0.5	0.5	25.0	10.0	0	1	61.18469	ok	0.33666	0.32049	100000
0.48076	1.09542	1.15091	1.0	0.5	0.5	0.75	1.0	10.0	9.0	0	1	23.86679	ok	0.3344	0.31635	100000
0.48188	1.09619	1.13898	1.0	0.25	0.25	1.0	1.0	10.0	6.0	0	1	21.09479	ok	0.33414	0.32014	100000
0.48152	1.13374	1.1931	1.0	1.0	1.0	0.5	0.75	15.0	8.0	0	1	33.86051	ok	0.32183	0.30328	100000
0.482	1.21374	1.25822	1.0	1.0	1.0	1.0	0.75	10.0	5.0	0	1	19.93357	ok	0.29709	0.28416	100000

Random Forest

rmse	loss	train loss	bootstrap	max depth	max features	min samples leaf	min samples split	n estimators	train time	status	geo mean	geo mean train	data size
0.36996	0.64461	0.45143	True	60	sqrt	4	10	200	21.94663	ok	0.52487	0.63672	100000
0.3728	0.64614	0.44093	True	90	sqrt	4	5	500	55.55322	ok	0.52406	0.64344	100000
0.37624	0.64713	0.52327	True	20	sqrt	1	5	200	18.21475	ok	0.52355	0.59258	100000
0.384	0.65966	0.54217	False	20	sqrt	4	2	400	52.6331	ok	0.51703	0.58149	100000
0.38384	0.65966	0.54187	False	20	sqrt	4	2	300	39.39137	ok	0.51702	0.58166	100000
0.3774	0.66181	0.34523	True	60	sqrt	2	5	500	59.07356	ok	0.51592	0.70806	100000
0.40308	0.66809	0.6524	False	10	sqrt	1	10	200	15.73843	ok	0.51269	0.52079	100000
0.38088	0.67276	0.41444	False	30	sqrt	4	2	100	15.81304	ok	0.5103	0.66071	100000
0.38316	0.6884	0.3521	False	90	sqrt	4	5	500	84.07343	ok	0.50238	0.70321	100000
0.3924	0.71836	0.17916	True	90	sqrt	1	2	300	37.78957	ok	0.48755	0.83597	100000

LGBM

rmse	loss	train loss	learn rate	n estimators	reg alpha	subsample	train time	status	geo mean	geo mean train	data size
0.39048	0.65727	0.63536	0.07	300	0.09	0.84	0.50938	ok	0.51826	0.52974	100000
0.3906	0.65756	0.64685	0.05	200	0.1	0.76	0.37731	ok	0.51811	0.52369	100000
0.39192	0.65805	0.63318	0.06	400	0.06	0.72	0.66328	ok	0.51786	0.5309	100000
0.39224	0.65854	0.62797	0.15	200	0.0	0.8	0.32135	ok	0.51761	0.53368	100000
0.39072	0.65865	0.62635	0.08	400	0.0	0.74	0.63052	ok	0.51755	0.53454	100000
0.39336	0.6592	0.65251	0.06	100	0.0	0.72	0.20576	ok	0.51727	0.52074	100000
0.393	0.65932	0.62567	0.11	300	0.03	0.82	0.47965	ok	0.5172	0.5349	100000
0.39508	0.65979	0.6216	0.13	300	0.0	0.9	0.46677	ok	0.51696	0.53708	100000
0.3928	0.66065	0.61896	0.14	300	0.0	0.86	0.51852	ok	0.51652	0.5385	100000
0.39656	0.66487	0.60486	0.14	500	0.07	0.74	0.75101	ok	0.51434	0.54615	100000