Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Facultad de Ingeniería

Trabajo práctico 2: Búsqueda bibliografica

Taller de Tesis 2, *Maestria en Explotación de Datos y Descubrimiento del Conocimiento*

Adrian Norberto Marino

Buenos Aires, 2023

Índice general

1. INTRODUCCIÓN

Los sistemas de recomendación tienen como objetivo principal proporcionar a los usuarios productos, promociones y contenidos relevantes a sus preferencias o necesidades. Estos sistemas permiten a los usuarios encontrar de forma ágil y eficiente lo que están buscando. Formalizando esta definición, podemos decir que los sistemas de recomendación buscan ayudar a un usuario o grupo de usuarios a descubrir ítems que se ajusten a sus preferencias, dado un conjunto de ítems que puede ser extenso o un amplio espacio de búsqueda.

Este objetivo puede variar dependiendo de cada negocio: Si consideramos un *e-commerce* de *delivery* gastronómico, su propósito sería ofrecer a los clientes platos relevantes a un precio asequible y con un tiempo de entrega aceptable.

Si hablamos de un *e-commerce* de productos, su objetivo consiste en proporcionar a los usuarios aquellos productos que satisfacen sus necesidades, a un precio que estén dispuestos a pagar. Además, se busca garantizar una experiencia satisfactoria con los vendedores.

En el negocio de visualización de contenido (audio, video, texto, etc..), el objetivo es acercar a sus usuarios contenido a fin a sus preferencias para mejorar su experiencia en la plataforma.

El objetivo principal en todos los casos es mejorar la conversión. En el campo del *marketing*, se define la conversión como las acciones realizadas por los usuarios que están alineadas con los objetivos de la empresa. Por ejemplo, aumentar el volumen de compras en un *e-commerce* de productos, incrementar la cantidad de entregas mensuales en un *e-commerce* de *delivery* gastronómico, aumentar las impresiones de publicidad en aplicaciones de visualización de contenido, prolongar el tiempo de permanencia en plataformas de *streaming* de audio o video, entre otros. Existen numerosos ejemplos en los que el objetivo común es mejorar la conversión y el compromiso del usuario con la marca, es decir, el *engagement*.

Desde un enfoque técnico, los sistemas de recomendación se utilizan para predecir el grado de preferencia de un usuario con un artículo específico. Esto se logra aplicando algoritmos de optimización que minimizan la diferencia entre el grado de preferencia esperado y el grado de preferencia real del usuario. También existen otros enfoques que utilizan medidas de distancia para determinar este grado de preferencia. En secciones posteriores, exploraremos estos conceptos con mayor detalle.

1.1. Tipos de sistemas de recomendación

A continuación, en la figura 1.1, se pueden observar las diferentes categorías y subcategorías de los sistemas de recomendación:

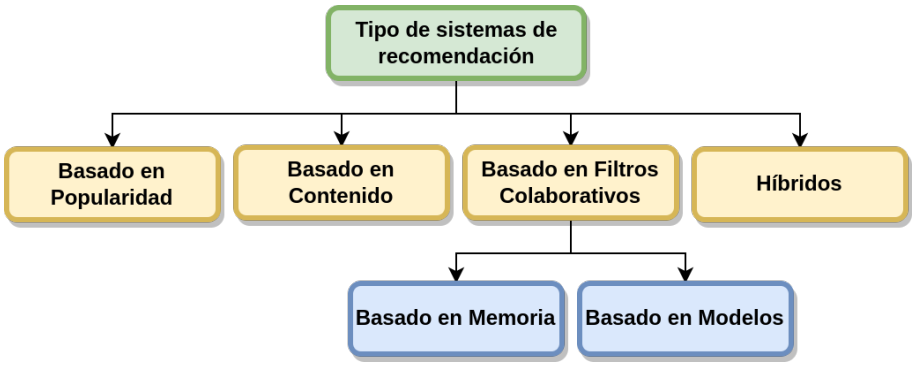

Fig. 1.1: Clasificación de tipos de sistemas de recomendaciones.

1.1.1. Basados en Popularidad

Este tipo de sistema de recomendación utiliza alguna característica de popularidad de los ítems en cuestión. Algunos ejemplos de estas características podría ser la cantidad de vistas, la cantidad de compras o la cantidad de comentarios positivos, o una combinación de ellas. Luego, estos sistemas buscan los K elementos más populares. Si bien este tipo de enfoque proporciona buenos resultados para nuevos usuarios, sus recomendaciones no tienen en cuenta las preferencias individuales de cada usuario, ya que se basan en estadísticas comunes a todos los usuarios. Por esta razón, a menudo no se consideran sistemas de recomendación en sentido estricto. No obstante, siguen siendo ampliamente utilizados debido a su capacidad para generar una alta tasa de conversión, a pesar de la falta de personalización.

1.1.2. Basados en Contenido

Este tipo de sistema de recomendación [1, 5] necesita un trabajo previo de ingeniería de *features* sobre los ítems. Se busca definir cuales son los *features* mas significativos para la tarea en cuestión, y cual es el grado de adecuación de cada ítems a los *features* seleccionados.

Por otro lado, existen dos formas de utilizas estos modelos:

- El usuario esta registrado pero no realizo interacciones con los ítems: Este escenario se da con usuarios que se registraron pero aun no han calificado ningún ítems. En estos casos muchas aplicaciones optan por realizar una encuesta inicial. Esta encuentra se utiliza para consultar al usuario *features* relevantes que permitirán comenzar a realizar recomendaciones. Por ejemplo, se podría presentan una lista de géneros, actores, directores, etc. Luego, el usuario realiza una puntuación de acuerdo a su preferencia. Estas encuestas se suelen realizar cada cierto tiempo, dado que hay usuarios que tienen una frecuencia muy baja de interacción con el sistema.

- El usuario se registro hace un tiempo y tiene cierta frecuencia de interacciones con los ítems: En esta caso, se utilizan las clasificaciones del usuario para realizar las recomendaciones. Este enfoque suele tener mejores resultados, ya que muchas veces (dependiendo del dominio de las recomendaciones) el usuario no conoce inicialmente cuales son sus preferencias o bien puede cambiar de parecer con el tiempo. En comparación, este enfoque es mejor al enfoque anterior, dado que el modelo tiene un conocimiento mas actualizado sobre las preferencias del usuario, permitiendo realizar recomendaciones mas alineadas a sus gustos.

Ambos enfoques puede combinarse. Supongamos que el científico de datos seleccionar los géneros como *features*. Inicialmente, puede realizar una encuesta para conocer el grado de preferencia del usuario por cada género. Luego, el usuario comienza a calificar ítems y estos valores se pueden actualizar utilizando alguna medida basada en las calificaciones. De esta forma, se podrían utilizar ambos enfoques de manera complementaria.

Luego, teniendo interacciones de los usuarios, se puede definir el grado de preferencia de los usuarios a cada *feature* definido para los ítems. Con esta información, es posible encontrar tanto ítems como usuarios similares y realizar recomendaciones del tipo:

- Para el *Usuario A* el cual tiene ciertos niveles de preferencia por cada *features*, se pueden recomendar los *Ítem X* e *Ítem Y*. El modelo puede inferir el grado de preferencia del *Usuario A* para cada item existente y luego ordenarlos. En este caso, el *Ítem X* es de mayor preferencia para el usuario que el *Ítem Y*.
- Dado el *Usuario A*, el cual tiene preferencia por el *Ítem X*, también podría tener preferencia por el *Ítem Y*, por ser muy cercano o similar al *Ítem X*.
- Dos *Usuarios A y B* cercanos o similares, tendrán preferencias similares. De esta forma es posible recomendar ítem consumidos por el *Usuario A* al *Usuario B* y vise versa.

La principal desventaja de este enfoque, es la necesidad de realizar ingeniería de *features* para encontrar los *features* que produzcan recomendaciones relevantes al usuario. El modelo no encuentra estos *features* automáticamente, sino que deben ser definidos de antemano manualmente. Se puede apreciar que esto introduce un sesgo al momento de seleccionar los *features* o construirlos en base a datos referentes a los ítems. Como ventaja, si se encuentran los *features* correctos se pueden lograr muy buenos resultados.

1.1.3. Basados en Filtrado Colaborativos

Estos modelos [1, 5], a diferencia de los modelos basados en contenido, no requirieren ingeniería de *features*, lo que hace muy simple su implementación, ya que únicamente es necesario registrar las interacciones de los usuarios para con los ítems. Luego, el propio modelo encuentra automáticamente los *features* mas relevantes dependiendo de la cantidad de columnas que se especifiquen (dimensiones de un vector *Embedding*). Ejemplos de interacciones podrían ser:

- El *Usuario A* visualizo el *Ítem X* el dia 2 de marzo de 2022.
- El *Usuario A* compro el *Ítem X* el dia 10 de marzo de 2022.
- El *Usuario A* califico al *Ítem X* con 5 puntos el dia 25 de marzo de 2022.

Ambos tipo de modelos, basados en contenido y filtros colaborativos, personalizan sus recomendaciones. Es decir, ajustan las recomendaciones a las preferencias de cada usuario particular. Además, ambos permiten encontrar usuarios e ítems similares y recomendar ítems entre usuarios similares.

Por otro lado, los modelos basados en filtros colaborativos, descubren un espacio latente de soluciones sin necesidad de recolectar datos y definir *features* en forma manual, a diferencia de los modelos basados en contenido. La selección o construcción manual de *features* puede llevar a una solución sesgada, ya que no esta basada en datos sino en el juicio experto del científico de datos. Esto puede llevar a una selección subjetiva de los *features* que se aleje de la realidad, introduciendo un sesgo en la predicción.

No todo son rosas con estos modelos, dado que sufren un problema llamado *Cold start* o arranque en frio. Los usuarios nuevos son aquellos que aun no han realizado ninguna interacción con el sistema. Estos modelos no podrán realizar recomendaciones a estos usuarios, dado que requieren un mínimo de interacciones para comenzar a ofrecer recomendaciones con cierta precisión.

Además, existen otros problemas referidos al cambiar la cantidad de interacciones de los usuarios. Si pensamos en una solución donde alimentamos al modelo con una ventana de interacciones para los últimos N meses, tendremos las siguiente situaciones:

- Usuarios nuevos: Los usuarios nuevos no tendrán interacciones. Por lo tanto, este modelo no podrá realizar ninguna recomendación. En general, se establece un mínimo de interacciones para que el modelo pueda realizar recomendaciones de forma acertada.
- Usuarios con pocas interacciones: Por otro lado, tenemos a los usuarios que tienen una baja taza de interacciones con el sistema o aplicación. Por ejemplo, en un *e-commerce* de venta de productos, hay usuarios que compran con mucha frecuencia y otros muy de vez en cuando. Estos últimos, en general tendrán una baja taza de interacción pudiendo caer por debajo del umbral mínimo que requiere el modelo. De esta forma, tendremos usuarios que quedarán fuera del modelo actual.
- Usuarios con muchas interacciones: En este caso, el usuario tiene una gran cantidad de interacciones con ítems. Para estos usuarios, el modelo podrá ofrecer recomendaciones relevantes, ya que cuanto mas interacciones se tenga, el modelo se ajusta con mas facilidad a sus preferencias. Por otro lado, esto puede ser una gran desventaja, ya que se produce un efecto de túnel. Es decir, el usuario obtiene recomendaciones muy ajustadas a sus preferencias, perdiendo la capacidad de descubrir nuevos ítems que podrían ser relevantes. Por esta cuestión se suelen mezclar tanto recomendaciones personalizadas como no-personalizadas, para favorecer el descubrimiento de nuevos ítems.

1.1.4. Categorías dentro de los modelos basados en filtros colaborativos

Dentro de los sistemas de recomendación basados en filtros colaborativos, tenemos dos sub-clasificaciones referidas a la forma en la que se realizan las predicciones.

Basados en Memoria

Este tipo de modelos, como su nombre lo indica, mantiene sus datos en memoria. Se recorren todos los datos (*full scan*) cada vez que se necesita realizar un inferencia o pre-

dicción (fijando un número de vecinos a comparar). Un ejemplo de estos modelos es el algoritmo de k vecinos cercanos (*KNN*), el cual mantiene una matriz rala de distancias en memoria, la cual se recorre completamente para comparar las distancias entre filas o columnas, usando alguna medida de distancia como puede ser la *distancia coseno*, *coseno ajustada*, *manhattan*, etc.. Para mitigar el problema de búsqueda exhaustiva (*full scan*), se puede utilizar una memoria *cache* y asi realizar estas búsquedas una única vez. Otro problema es su limitación al tamaño máximo de la memoria con la que se cuenta, es decir, que el tamaño de la matriz depende de la memoria máxima disponible. Esto puede mitigarse utilizando implementaciones de matrices rala, las cuales comprimen los datos en memoria guardando unicamente las celdas que tienen datos. Además, es posible utilizar un memoria *cache* que mantenga en memoria las búsqueda mas frecuentes y baje a almacenamiento secundario las menos frecuentes. Todos estos problemas de *performance* y uso de recursos se deben a que *KNN* no reduce la dimensionalidad de los datos, como si lo hacen varias implementaciones basadas en *embeddings*, *auto-encoder*, redes neuronales etc.., donde lo que se busca una representación mas compacta de los ítems y usuarios sin perder información. Mas allá de estos problemas, los resultados obtenidos por estos modelos no están muy alejados de aquellos que se encuentran en el estado del arte. Puede recomendarse su uso cuando tenemos un dominio reducido, dada su simplicidad.

Basados en Modelos

Algunos ejemplos de estos modelos son los clasificadores bayesianos, redes neuronales, algoritmos genéticos, sistemas difusos y la técnica de descomposición matricial (*SVD*). Estos modelos en general buscan directa o indirectamente reducir la dimensionalidad de los datos. De esta forma, es posible utilizarlos en dominios con una gran cantidad de datos.

1.1.5. Modelos Híbridos o Ensambles

Lod modelos híbridos o ensambles [2, 4], son aquellos modelos que combinan mas de una técnica de recomendación. Comúnmente son utilizados para resolver el problema de *Cold start* o arranque en frio que sufren los modelos de recomendación basados en filtros colaborativos.

Los modelos de recomendación colaborativos, no puede realizar recomendaciones a usuarios que aun no han calificado ítems o que no han realizado ninguna interacción con la aplicación en cuestión. Para solucionar esta problemática, se utilizan modelos de recomendación que no dependan de las interacciones de los usuarios, como pueden ser modelos basados en contenido [5].

El uso de ensambles de modelos puede solucionar este problema de *Cold start* o arranque en frio, y al mismo tiempo, es posible realizar recomendaciones de mayor calidad a aquellas resultado de la predicción de cada modelo por separado.

Por otro lado, utilizar ensambles no asegura una mejora en las recomendaciones, ya que es algo muy dependiente de los datos, de la heterogeneidad de los resultados de los modelos y las técnicas de ensamblado que se utilicen.

1.1.6. Estrategias de ensamble de modelos

A continuación se definen las estrategias de ensamble mas comunes:

Switching

La técnica de *switching* consiste en intercambiar los modelos de recomendación según la cantidad actual de interacciones de cada usuario. Es recomendable filtrar las interacciones dentro de un período de tiempo de N horas, días o meses, para luego calcular el número de interacciones del usuario. Un ejemplo sería:

- Si el usuario tiene menos de N interacciones, se utiliza un recomendador por popularidad.
- Si el usuario tiene entre 5 y 10 interacciones, se utiliza un recomendador basado en contenido.
- Si el usuario tiene mas de 20 interacciones, se utiliza un recomendador basado en filtros colaborativos.

El principal problema de este enfoque es el cambio abrupto en el patrón de las recomendaciones al cambiar de un modelo de recomendación a otro, ya que la calidad de las recomendaciones puede variar mucho de un recomendador a otro. En el ejemplo anterior, el cambio del recomendador por popularidad al recomendador basado en contenido supone una mejora en la calidad de las recomendaciones, ya que a partir de ese momento se estarían personalizando los resultados. Sin embargo, ¿qué sucede cuando un usuario disminuye su frecuencia de interacción con la aplicación? En este caso, se cambiaría de modelos más personalizados a menos personalizados. Para disminuir los efectos de este problema, se podría aumentar la ventana de tiempo para filtrar la interacciones, ocasionando el efecto contrario a utilizar un ensamble.

Mixing

Esta técnica combina los *ratings* predichos por cada modelo de recomendación para cada ítem recomendado. Normalmente, se realiza una normalización dividiendo por el *rating* más alto, lo que da a cada ítem un *score* entre 0 y 1, donde 1 es la puntuación más alta. Otra alternativa seria calcular el *score* mediante la media, promedio pesado o mediana de los *ratings* predichos por cada modelo y para cada ítem. Esta técnica puede ser más efectiva si se combina con la técnica de *switching*, que suaviza la transición entre modelos mezclando las recomendaciones de dos modelos cuando se alcanza un rango de interacciones preestablecido como frontera entre estos.

Weighted

La técnica de *Weighted* en un caso particular de la técnica de *Mixing*. Se refiere al caso en que se realiza un promedio pesado de los *ratings* predichos por cada modelo, para el mismo ítem. El problema es que los pesos se asignan manualmente, ya sea arbitrariamente, o por conocimiento del dominio. Es decir, no existe ningún proceso para optimizar o descubrir estos hiper parámetros guiado por datos.

Regresión Lineal

Esta técnica es similar al enfoque *Weighted*, pero en este caso, si se optimizan los pesos utilizando un modelo de regresión linear. El proceso consta de los siguientes pasos:

1. Se entrena cada modelo de recomendación por separado utilizan el mismo conjunto de entrenamiento y evaluación.

2. Con cada modelo se predice valor de *rating* que cada usuario asignara a cada ítem del conjunto de entrenamiento. De esta forma como resultado, se tiene un nuevo conjunto de entrenamiento con las siguientes columnas: usuario, item, los *ratings* predichos por cada modelos como columnas, y finalmente el *rating* real realizado por el usuario. Lo mismo sucede con el conjunto de evaluación.

3. Finalmente, se entrena y evalúa una regresión lineal como los conjuntos construidos en el paso anterior. A diferencia del enfoque *Weighted* ahora si los pesos se ajustan de acuerdo a los datos de entrenamiento.

Stacking

El enfoque *Stacking* es una generalización del enfoque de *Regresión Lineal*. Este consta de aplicar cualquier modelo (incluido regresión lineal) para ajustar los pesos, pudiéndose aplicar cualquier modelos lineal o no lineal como: redes neuronales, regresiones polinómicas, etc.

Feature-Weighted Linear Stacking

La técnica *Feature-Weighted Linear Stacking* [6, 4] a grandes rasgos guarda similitud con los últimos 3 métodos expuestos (*Weighted*, *Regresión Lineal*, *Stacking*). Consta de aplicar un función lineal similar a una regresión, donde cada modelo se puede pensar como una componentes de la regresión, cada una multiplicada por una meta función o función

feature-weights. Estas meta funciones pueden ser pensadas como pesos guiados por reglas duras. Es decir, sus valores están pesados por reglas estáticas definidas previamente. Esta reglas ayudan a manejar el nivel de contribución que tiene cada modelo sobre el *rating* resultado de la predicción. Por ejemplo, podríamos utilizar reglas duras para aumentar la contribución de un modelo basado en contenido cuando el usuario tiene pocas interacciones o aumentar la contribución de un modelo de filtros colaborativos cuando el número de interacciones del usuario aumenta. En definitiva, es una formas de realizar lo que se llama *blend* o mezcla de las contribuciones de cada modelo al *rating* resultado aplicando una transición suave.

K-Arm Bandit utilizando Thompson sampling

La técnica propuesta se basa en el conocido problema de los bandidos multi brazo o también llamado el problema de las maquinas traga monedas.

El problema consiste en que tenemos una cantidad K de maquinas traga monedas y contamos con una cantidad limitadas de fichas. Se busca producir la mayor cantidad de ganancias, y en el proceso descubrir cual es la probabilidad de existo de cada maquina. Todo esto utilizando la menor cantidad de fichas posible. En definitiva, se busca maximizar la ganancia descubriendo de forma temprana las probabilidades de éxito de cada maquina [9].

Este es un método iterativo, donde en cada iteración, el método selecciona una máquina, realiza una jugada y registra si se ganó o se perdió.

La selección de la máquina puede ser al azar, o estar guiada por una heurística o método estadístico.

Cuando la selección es al azar, se dice que se está en modo o etapa de exploración, es decir, se gastan fichas para acumular resultados de cada máquina y así registrar una serie de éxitos y pérdidas en cada máquina. Cada máquina se puede pensar como una variable binomial, es decir toma dos valores: gano o perdió.

Luego, con esta sucesión de eventos, se estima una distribución beta [10] para cada máquina, y finalmente se toma una muestra de un valor de esa distribución. Este valor sera el valor mas probable o la probabilidad de éxito de la máquina.

Ya realizada una primera etapa de exploración, en las próximas iteraciones se puede hacer uso del conocimiento actual y seleccionar la máquina con mayor probabilidad de éxito. A esta etapa se le llama explotación, ya que se está explotando o haciendo uso del conocimiento aprendido con anterioridad.

El proceso realiza una mezcla de estas dos etapas de forma estocástica, haciendo uso de la estrategia *epsilon greedy* (mejorada). Esta estrategia, elige de forma aleatoria entre ambas técnicas de selección, con una probabilidad de exploración *epsilon*. Esta probabilidad ira disminuyendo a medida que aumenta el número de iteraciones. Esta es una mejora que permite restringir la exploración. Tiene mucho sentido, ya que a medida que se realizan más jugadas, hay una mejor comprensión del problema, y es menos necesario realizar una selección al azar.

En conclusión, se busca determinar la probabilidad de éxito de cada máquina, realizando la menor cantidad de jugadas posibles.

Este enfoque se utiliza en el ámbito de sistemas de recomendación para determinar el modelo con mayor probabilidad de éxito para cada usuario [3, 7, 8, 10, 11, 12, 13]. Es decir, cuál es el modelo que tiene mayor probabilidad de realizar una recomendación similar a la que realizaría el usuario. Esta medición de éxito depende de la métrica a evaluar.

A diferencia de varios de los modelos de ensamble anteriormente expuestos, donde se requiere realizar una misma predicción para todos los modelos, este en foque se realiza la predicción en un único modelo, debido a que es una estrategia de selección de modelos. Esto se traduce en un uso mas eficiente de los recursos de CPU, GPU y memoria del sistema, ya que estos modelos en general se ejecutan como servicios *cloud* (en la nube) en *AWS* (*Amazon Web Services*) o *GCP* (*Google Cloud Platform*), donde se paga unicamente por el uso de recursos que realizan los servicios o modelo en este caso.

REFERENCIAS

[1] D./Dª. Josefa Aguado Ferrón. (2016). Introducción a los sistemas de recomendación, Universidad de Granada. https://fcd.ugr.es/sites/centros/fcd/public/2021-05/TFG-JOSEFA-AGUADO-FERRON.pdf

[2] Criado González, Marta. (2016). Análisis e implementación de un sistema de recomendación para la lista de la compra, UNIVERSIDAD DE GRANADA. https://core.ac.uk/download/pdf/288501906.pdf

[3] Brodén, Björn and Hammar, Mikael and Nilsson, Bengt J and Paraschakis, Dimitris. (2018). Ensemble recommendations via thompson sampling: an experimental study within e-commerce. https://www.diva-portal.org/smash/get/diva2:1409392/FULLTEXT01.pdf

[4] Falk, Kim. (2019). Practical recommender systems. Simon and Schuster. https://github.com/GeorgeQLe/Textbooks-and-Papers/blob/master/%5BML%5D%20Practical%20Recommender%20Systems.pdf

[5] Glauber, Rafael and Loula, Angelo. (2019). Collaborative Filtering vs. Content-Based Filtering: differences and similarities. arXiv preprint arXiv:1912.08932. https://arxiv.org/pdf/1912.08932.pdf

[6] Sill, Joseph and Takács, Gábor and Mackey, Lester and Lin, David. (2009). Feature-weighted linear stacking. arXiv preprint arXiv:0911.0460 https://arxiv.org/pdf/0911.0460.pdf

[7] Cañamares, Rocío and Redondo, Marcos and Castells Pablo. (2019). Multi-armed recommender system bandit ensembles. Proceedings of the 13th ACM Conference on Recommender Systems https://castells.github.io/papers/recsys2019-rcanamares.pdf

[8] Chakrabarti, Deepayan and Kumar, Ravi and Radlinski, Filip and Upfal, Eli. (2008). Mortal multi armed bandits. Advances in neural information processing systems. https://proceedings.neurips.cc/paper_files/paper/2008/file/788d986905533aba051261497ecffcbb-Paper.pdf

[9] Markel Sanz Ausin (Deep Learning Engineer, NVIDIA) (2020). Introducción al aprendizaje por refuerzo. Parte 1: el problema del bandido multibrazo. https://markelsanz14.medium.com/introducci%C3%B3n-al-aprendizaje-por-refuerzo-parte-1-el-problema-del-bandido-multibrazo-afe05c0c372e

[10] Sophia Yang, Ph.D. (2022). Multi-Armed Bandit Algorithms: Thompson Sampling. Intuition, Bayes, and an example. https://towardsdatascience.com/multi-armed-bandit-algorithms-thompson-sampling-6d91a88145db

[11] James McCaffrey. (2019). How to Do Thompson Sampling Using Python. https://visualstudiomagazine.com/articles/2019/06/01/thompson-sampling.aspx

[12] Charl De Villiers. (2019). How to Build a Product Recommender Using Multi-Armed Bandit Algorithms. https://www.offerzen.com/blog/how-to-build-a-product-recommender-using-multi-armed-bandit-algorithms

[13] Anson Wong. (2017). Solving the Multi-Armed Bandit Problem. https://towardsdatascience.com/solving-the-multi-armed-bandit-problem-b72de40db97c