UNIVERSIDAD TORCUATO DI TELLA

UNIVERSIDAD TORCUATO DI TELLA, ESCUELA DE NEGOCIOS

MASTER IN MANAGEMENT AND ANALYTICS

---

# Aplicación empírica de metodologías de Machine Learning to Rank

## Comparación con otras técnicas de aprendizaje supervisado en diferentes aplicaciones

---

## Resumen

Existen muchos problemas de negocio que requieren predecir u ordenar un conjunto de opciones. Para resolver este tipo de preguntas los modelos tradicionales de aprendizaje supervisado pueden no ser ideales, ya que no están diseñados para esta tarea. Learning to Rank, por otro lado, es una rama del aprendizaje supervisado que busca ordenar un conjunto de opciones dado un criterio de relevancia. En esta tesis se evalúa el desempeño de estos modelos en tres aplicaciones diferentes. Dos de ellas requieren ordenar páginas web respecto a criterios de búsqueda, y la tercera es un sistema de recomendación de productos en un marketplace. Los resultados obtenidos en este estudio sugieren que estos modelos tienen un mejor desempeño en tareas de ordenamiento que técnicas tradicionales de regresión y clasificación. Utilizar Learning to Rank con la metodología planteada en esta tesis, puede ayudar a industrias de diferentes tipos a obtener mejores resultados en múltiples áreas como marketing y finanzas.

**Alumno:** Umaña Zamora, Gabriela M.

**Directores:** Galvez, Ramiro H. - Escuela de Negocios, Universidad Torcuato di Tella

Soulès, Lucas M. - Mgtr. en Management and Analytics

**Fecha de entrega:** 31 de mayo de 2021

UNIVERSIDAD TORCUATO DI TELLA, BUSINESS SCHOOL

MASTER IN MANAGEMENT AND ANALYTICS

---

# Empirical application of Machine Learning to Rank algorithms

## Comparison with other supervised learning techniques in different applications

---

## Abstract

Multiple business questions require predicting or sorting a list of options. To solve this kind of problem, traditional supervised learning, like regression and classification, may not be optimal, because these are not designed for this task. On the other hand, Learning to Rank is a supervised learning technique intended to sort a list of items based on criteria of relevance. This thesis conducted a study that evaluates the performance of these algorithms on three different databases. Two of which require sorting a list of web URLs based on search parameters. The third one, on the other hand, is an application of product recommendations, based on user's views and searches in a marketplace. The results presented in this document, suggests that Learning to Rank algorithms have better performance on ranking tasks than regression and classification approaches. Multiple industries can find a benefit by using Learning to Rank models instead of traditional supervised models, by applying them to different areas of the business like marketing and finance.

**Student:** Umaña Zamora, Gabriela M.

**Thesis advisors:** Galvez, Ramiro H. - Business School, Universidad Torcuato di Tella
Soulès, Lucas M. - M.S. Management and Analytics

**Date:** May 31st, 2021

# Índice

# 1. Introducción

El aprendizaje automático permite a empresas de diferentes industrias analizar datos complejos y de gran magnitud. Utilizando técnicas algorítmicas de esta naturaleza, las empresas pueden construir modelos que ayuden a resolver diversos problemas como satisfacción y retención de clientes, detección de enfermedades, optimización de rutas y de producción, entre muchas más. Sin embargo, para obtener los mejores resultados posibles, al aplicar estos modelos es de suma importancia identificar qué tipo de algoritmo es el más adecuado para resolver el problema en cuestión. En el ámbito de aprendizaje automático existen dos ramas principales que se utilizan para resolver numerosos problemas de negocio: Aprendizaje supervisado y no supervisado.

Los modelos de aprendizaje no supervisado tienen como objetivo encontrar relaciones entre los diferentes registros. Es decir, buscan estudiar la estructura intrínseca de una base de datos. La particularidad de estos modelos es que se utilizan donde, a priori, no se conoce ningún valor objetivo para cada observación. Es por esto que se suelen utilizar, principalmente, en problemas de agrupación y reducción de dimensionalidad. Ejemplos de esto puede ser, para hacer campañas de marketing focalizado al identificar grupos de usuarios con comportamientos similares de compra. También, se pueden utilizar para detección de fraudes o en biología para clasificar plantas y animales con características similares (Berry et al., 2020).

La segunda rama es la técnica de aprendizaje supervisado. Estos modelos utilizan datos etiquetados y buscan aprender una función que, dado unos valores de entrada conocidos, devuelvan una etiqueta adecuada. Estos algoritmos se utilizan para predecir variables categóricas o continuas. Cuando la variable a predecir es categórica se trata de un problema de clasificación. Entre algunas de las aplicaciones más comunes se pueden mencionar: predecir si un cliente dará de baja un servicio en un momento determinado, predecir si un préstamo entrará en default o si una red eléctrica presentará fallas (Berry et al., 2020). Por otro lado, cuando la variable objetivo es continua, se utilizan modelos supervisados de regresión. Con estos últimos se pueden resolver problemas tales como: predecir el balance de una cuenta de ahorro, predecir el rendimiento de un cultivo agrícola (Berry et al., 2020), entre muchos más.

Existe una tercera categoría de modelos de aprendizaje supervisado, menos conocida, que se denomina *Learning to Rank* (LTR). Estos algoritmos, a diferencia de los de clasificación o regresión, tienen como objetivo devolver una lista de items ordenados de acuerdo a un criterio de relevancia. Por ejemplo: ordenar los resultados de una búsqueda.

El concepto de LTR se comenzó a utilizar en motores de búsqueda comerciales en la década de los 2000. El primero en utilizarlo fue Altavista (Silverstein et al., 1998), un motor de búsqueda posteriormente comprado por Yahoo!. Luego, Microsoft Research creó en 2005 su propio modelo y lo implementó en Bing (Burges, 2010). A partir de entonces, estos algoritmos han tomado gran importancia en aplicaciones de Information Retrieval[1] (IR). Yahoo los utiliza para ordenar búsquedas web (Yin et al., 2016), mientras que Slack lo utiliza para mejorar la búsquedas de mensajes (Tromba et al., 2015).

La razón por la que estos algoritmos tomaron tanta importancia en estas aplicaciones es porque cubren necesidades que no están resueltas por modelos tradicionales de clasificación y regresión. Estos últimos están diseñados para aprender una función que predice un valor o etiqueta específica. Los modelos de LTR, por otro lado, aprenden una función que compara todas las opciones disponibles, y devuelve una lista ordenada. Por lo tanto, son capaces de resolver de mejor manera los problemas de ordenamiento. Más allá de aplicaciones de IR, investigadores han demostrado que estos modelos tienen un buen desempeño en otro tipo de aplicaciones. Por ejemplo, Daniel Poh de la Universidad de Oxford, demuestra cómo al utilizar modelos de ranking en el diseño de portfolio de acciones, se obtienen mejores retornos que utilizando modelos de clasificación y regresión (Poh et al., 2020). En el área de marketing, Tian Gan, de la universidad de Shandog, realizó un estudio en 2019, que muestra cómo se pueden aplicar modelos de LTR para seleccionar, entre múltiples usuarios de redes sociales, a los mejores micro-influencers[2] para promocionar campañas específicas (Gan et al., 2019).

El objetivo de este trabajo es, en primer lugar, comparar los algoritmos de LTR con las técnicas de aprendizaje supervisado más utilizadas, y evaluar si estos modelos son competitivos. Como segundo objetivo, este trabajo busca mostrar cómo LTR puede ser

[1] Definicion dada en *Modern information retrieval*, por Baez Yates: "Dada una necesidad de información (consulta + perfil del usuario + ... ) y un conjunto de documentos, ordenar los documentos de más a menos relevantes para esa necesidad y presentar un subconjunto de aquellos de mayor relevancia".
[2] Usuarios de las redes sociales que se especializan en un nicho de mercado o área específica

aplicado en problemas de recomendación y de IR. Estos dos problemas son fundamentales en negocios de e-commerce, ya que directamente afectan la experiencia del usuario e influyen en métricas de conversión, retención y ganancias. Pinterest, por ejemplo, obtuvo un incremento del 30% en retención de usuarios al implementar modelos de ranking para recomendar *pins* (Liu et al., 2017).

Para lograr estos objetivos se seleccionaron tres bases de datos: Yahoo[3], Microsoft[4] y Mercado Libre[5]. Las dos primeras contienen datos correspondientes a búsquedas de internet y *urls.* El objetivo es ordenar, para cada búsqueda, las diferentes opciones de páginas web. Sobre estas bases se evaluaron modelos de clasificación, regresión y ranking, con el objetivo de comparar el desempeño de estos modelos en aplicaciones de IR.

Por otro lado, la base de Mercado libre contiene los datos publicados en la competencia realizada por esta empresa en 2020. Esta base se utiliza para mostrar cómo los algoritmos de LTR presentan una opción competitiva para resolver problemas de recomendación en un marketplace. El problema que se busca resolver es cómo seleccionar y ordenar las 10 publicaciones que tengan la mayor probabilidad de compra, basándose en la navegación previa del usuario. Se utilizó la misma ingeniería de atributos de una de las soluciones con mejores resultados en la competencia. De esta manera, utilizando las mismas variables, se evaluó si alguna de las técnicas de ranking lograba superarla.

El resto del documento se encuentra estructurado de la siguiente manera: en la Sección 2 se presentará el marco teórico de las diferentes técnicas de LTR; en la Sección 3 se introducen los datos, incluyendo descripción de las variables, magnitud de las bases y un pequeño análisis descriptivo; en la Sección 4 se describe la implementación, que incluye ingeniería de atributos, parámetros de los modelos entrenados, y métrica de evaluación; en la Sección 5 se presentan los resultados para cada una de las bases; y finalmente, en la Sección 6 se discuten los resultados y se presentan las conclusiones y futuras mejoras.

[3] https://webscope.sandbox.yahoo.com/catalog.php?datatype=c&did=44
[4] https://www.microsoft.com/en-us/research/project/mslr/
[5] https://ml-challenge.mercadolibre.com/

# 2. Métodos

## 2.1 Modelos de ranking

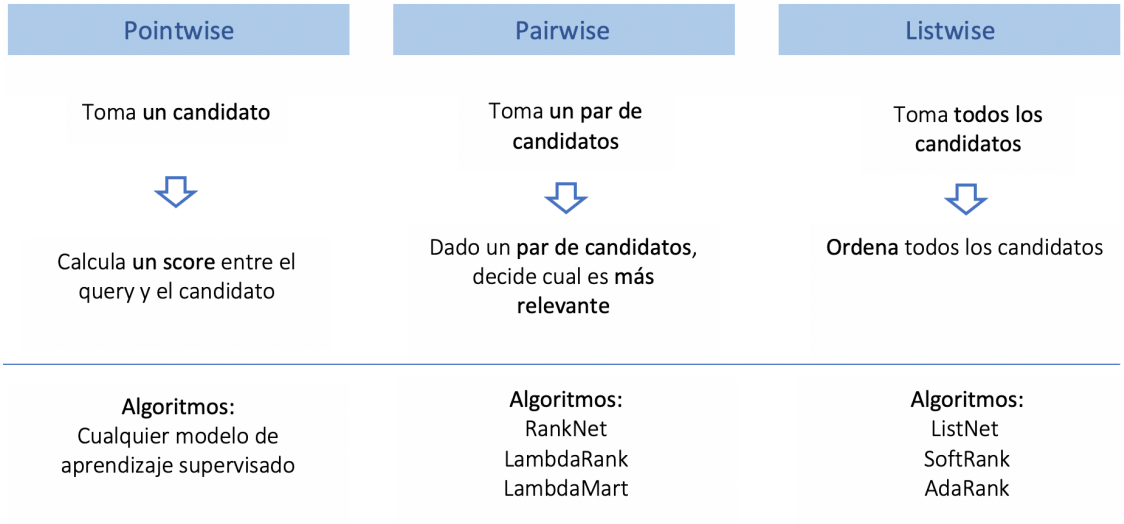
Learning to Rank (LTR) es una técnica de aprendizaje supervisado para resolver problemas de ordenamiento. La principal diferencia de estas técnicas con otros modelos tradicionales, como clasificación y regresión, es que estos últimos buscan evaluar cada instancia en particular y predecir una clase o un valor para cada una de ellas, mientras que los modelos de ranking buscan ordenar un conjunto de ítems. Es decir, estas técnicas no se enfocan en asignar un score o clase, sino en predecir el ordenamiento óptimo basado en un criterio de relevancia. En pocas palabras, la diferencia radica en la variable objetivo. En estos modelos la función que se busca aprender es $f(q, D)$ donde $q$ representa una búsqueda (*query* en inglés) y $D$ una lista de posibles resultados (*documents* en inglés). La función deberá ordenar la lista $D$ basada en la relevancia de cada uno de los ítems respecto a los parámetros de $q$.

La aplicación más común de estos modelos es en problemas de *Information Retrieval* (IR), los cuales tienen como objetivo recolectar información que cumpla ciertos criterios de búsqueda. Un ejemplo de esto son los motores de búsqueda de internet. En este caso $q$ representa los parámetros de búsqueda del usuario y $D$ todas las posibles páginas web afines. El modelo de ranking devolverá una lista ordenada de estas opciones, tratando de colocar en las posiciones superiores, aquellas páginas que estén más relacionadas con la búsqueda del usuario. Otras aplicaciones menos comunes son traductores (Shen et al., 2004), algoritmos para autocompletar texto (Shokouhi, 2013), sistemas de recomendación de productos (Kanti et al., 2017) y selección de portafolio de acciones (Poh et al., 2020).

A la fecha, hay tres formas diferentes de abordar el problema de ranking. La primera es usar modelos de machine learning tradicionales; técnica denominada *Pointwise*. Por otro lado, las soluciones más avanzadas incluyen evaluar la relevancia relativa de dos documentos a la vez; esto se lo conoce como *Pairwise*. Por último, la alternativa conocida como *Listwise* propone evaluar todos los documentos relacionados al *query* ($q$) simultáneamente.

A continuación, se explica cada una de estas técnicas en detalle.

*Figura 1. Metodologías de Learning to Rank*




### 2.1.1 Pointwise

Este tipo de algoritmos busca resolver el problema de ordenamiento utilizando modelos tradicionales de aprendizaje supervisado. La característica principal de estos modelos es que se entrenan viendo un documento a la vez. En este caso se pueden utilizar tres tipos de modelos:

- Regresión: La variable a predecir es continua, por ejemplo un score de relevancia.
- Clasificación: La variable a predecir es categórica no ordenada, por ejemplo documento relevante (1) o no relevante (0).
- Regresión ordinal: La variable a predecir es categórica ordenada, por ejemplo categoría de relevancia de 0 a 5.

El input para entrenar estos algoritmos está compuesto por los features de cada par *query-documento*. Por ejemplo, si para una búsqueda de internet $q_1$ hay tres posibles resultados $d_1, d_2, d_3$, y para una segunda búsqueda $q_2$ hay dos resultados $d_1, d_2$, un modelo pointwise se entrenará con tres registros correspondientes al primer *query* y dos registros del segundo. Es decir el input del modelo será: $x_{1:}\, q_1 d_1$ , $x_{2:}\, q_1 d_2$ , $x_{3:}\, q_1 d_3$ , $x_{4:}\, q_2 d_1$ , $x_{5:}\, q_2 d_2$

Una vez evaluados los documentos, se ordenan en base a la probabilidad o score asignado por el modelo. Estos modelos buscarán que las predicciones sean lo más cercanas posibles al objetivo, pero no se enfocan en ordenar los documentos.

A pesar de que esta metodología presenta una gran ventaja por poder implementarse con cualquier algoritmo de machine learning, no es la forma óptima de abordar el problema de ordenamiento. Esto es así ya que la función de pérdida no tiene visibilidad sobre la posición que ocupa cada documento en la lista, es decir pierde todo contexto. Además, cuando el número de documentos por *query* varía mucho, la función de pérdida se enfocará más en los *queries* con más documentos (Liu, 2009).

### 2.1.2 Pairwise

A diferencia de los modelos *pointwise*, esta metodología evalúa los documentos en pares. Es decir, la función que aprende el modelo toma todos los pares de documentos posibles para un *query*, y predice cuál de ambos es más relevante para la búsqueda. De esta manera, el problema de ordenamiento se reduce a un problema de clasificación binaria, en el que se tiene que predecir si un documento es más relevante que otro. El objetivo de estos modelos es hacer predicciones positivas para el documento más relevante en el par y negativas para el menos relevante (Liu, 2009). Al clasificar cada par de documentos es posible ordenar toda la lista en base a estas predicciones .

Estos modelos presentan una ventaja respecto a las técnicas *pointwise*, ya que evalúan la relevancia relativa de cada documento, en lugar de enfocarse en predecir una variable objetivo. Esto contribuye a obtener un mejor ordenamiento (Liu, 2009).

A continuación, se describen los modelos más utilizados con la técnica *pairwise*.

#### *RankNet*

Es una red neuronal de dos capas desarrollada por Chris Bruges de Microsoft Research en 2010 (Burges, 2010). Recibe como input los atributos asociados a dos documentos del mismo *query* $x_u$, $x_v$, y devuelve dos outputs: un score para cada documento $f(x_u)$ y $f(x_v)$. Luego estos scores son activados por la siguiente función sigmoide, que devolverá la probabilidad de que el documento $x_u$ sea más relevante que $x_v$:

$$P_{uv} = \frac{1}{1+ e^{-\sigma(f(x_u)-f(x_v))}}$$

Como función de pérdida se utiliza la entropía cruzada, siendo $\overline{P_{uv}} = \frac{1}{2}(1 + S_{uv})$, $S_{uv} \in \{0, \pm 1\}$, tomando valor 1 si $x_u$ es más relevante que $x_v$, -1 de lo contrario y 0 si son iguales (Burges, 2010).

$$C = (-\overline{P_{uv}} \, log P_{uv}) - (1 - \overline{P_{uv}}) \, log(1 - P_{uv})$$

$$C = \frac{1}{2}(1 + S_{uv}) \, \sigma(f(x_u) - f(x_v)) + log(1 + e^{-\sigma(f(x_u)-f(x_v))})$$

Una vez calculada la entropía, se actualizan los pesos *w* de la red utilizando el gradiente descendente de la función de costo y *back propagation*. Este proceso se repite con todos los documentos de cada *query* y finalmente se ordenan en base al score obtenido de la red.

Una desventaja de este modelo es que intrínsecamente busca optimizar el error entre dos pares de documentos, en lugar de considerar el ordenamiento de toda lista. Es por esto que se desarrolla *LambdaNet*.

***LambdaNet***

Este modelo es una versión mejorada de *RankNet*. Al igual que el anterior, es una red neuronal, pero introduce un nuevo elemento llamado *lambda* (λ). Este modelo, a pesar de que es una técnica pairwise, porque evalúa un par de documentos a la vez, con el término λ , analiza cómo la predicción obtenida para cada documento, influye en el ordenamiento de toda la lista.

En *RankNet*, la función de costo se optimiza utilizando un gradiente descendente que se enfoca en reducir el error de pares. Por otro lado, *LambdaNet* utiliza un gradiente λ que considera el orden óptimo de toda la lista. Para lograr esto, el gradiente de la función de

costo (C) es actualizado con un término $|NDCG|$[6], que indica como esta métrica de information retrieval se ve afectada si las posiciones del documento $u$ y $v$se intercambian. Es decir, si el modelo predice que $u$ es más relevante que $v$, el algoritmo calcula el cambio en la métrica NDCG, que considera toda la lista de documentos, si $v$se coloca antes que $u$.

$$\lambda_{uv} = \frac{\partial C(f(x_u) - f(x_v))}{\partial f(f(x_u))} |\Delta NDCG|$$

La posición de cada documento se actualizará con un peso λ. Si el documento $x_u$ es más relevante que $x_v$, éste se moverá hacia arriba de la lista con un peso |λ| y el documento $x_v$ se moverá hacia abajo con el mismo peso. (Burges C. J., 2010). De esta manera se logra que se optimice el ordenamiento general de los documentos en lugar de enfocarse en la pérdida de pares. Además al utilizar el término $|\Delta NDCG|$, se le asigna más peso a los documentos ubicados al inicio de la lista

En la siguiente figura se ilustra cómo los gradientes tienen un rol diferente en *RankNet* y *LambdaNet*.

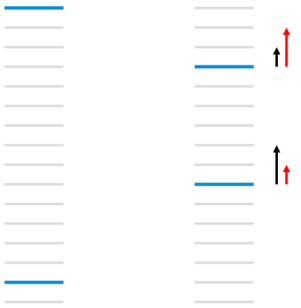


*Figura 2. RankNet vs LambdaNet*

En la Figura 2 las líneas grises representan documentos irrelevantes para el *query* y las líneas azules documentos relevantes. En el ordenamiento de la izquierda, el error de los pares es 13 (documentos grises clasificados como más relevantes que la última línea azul). En el ordenamiento de la derecha, el error de pares se redujo a 11. A pesar de que el error se redujo, este ordenamiento no es el deseado ya que en el top de la lista no están incluidas

[6] Métrica que evalúa la calidad del ordenamiento de una lista, al otorgar más peso a que los documentos más relevantes se ubiquen en las primeras posiciones. Más información en 4.4 Métricas de evaluación

las líneas azules. Las flechas negras representan el gradiente de *RankNet*, que se incrementa cuando se reduce el error de pares. Las flechas rojas muestran el gradiente λ de *LambdaNet* que tendrá más peso si los documentos relevantes se mueven a las primeras posiciones de la lista. (Burges C. J., 2010)

***LambdaMART***

Este modelo es simplemente un *LambdaNet* que, en lugar de utilizar redes neuronales, combina las técnicas de optimización *gradient boosting forest* con el término *lambda* de *LambdaNet*. *Gradient boosting forest* es una técnica de aprendizaje supervisado que está formado por un conjunto de árboles de decisión entrenados de forma secuencial, de manera que cada árbol aprende de los errores del anterior. Estos modelos se optimizan utilizando un gradiente descendente, que en el caso de *LambdaMART* es el término λ explicado anteriormente.

### 2.1.3 Listwise

A pesar de que los modelos *pairwise* tienen un buen desempeño en tareas de ordenamiento, presentan la desventaja de no evaluar todos los elementos de la lista simultáneamente. Además, al entrenar el modelo con cada par *query-documento*, los *queries* con mayor número de documentos van a tener mayor influencia en la función de pérdida (Cao et al., 2007). Es por esto que surgen las técnicas *Listwise*. Esta metodología, a diferencia de *pairwise*, busca ordenar una lista de documentos evaluando todos los elementos a la vez.

Existen dos subcategorías de esta técnica. La primera consiste en entrenar los modelos con una función de pérdida que directamente optimice una métrica de *Information Retrieval* (IR). La otra categoría consiste en evaluar todas las combinaciones de ordenamiento posibles. En este caso, la función de costo busca minimizar la diferencia entre la combinación propuesta por el modelo y el ordenamiento óptimo.

1. Listwise con optimización directa

La primera subcategoría es un proceso de optimización directo, donde al entrenar el modelo se busca optimizar la misma métrica con la que se evalúa. Entre las métricas más

comunes de IR se pueden mencionar *NDCG* (*Normalized Discounted Cumulative Gain*) y *MAP* (*Mean Average Precision*). La primera calcula la ganancia acumulada que se obtiene por la relevancia de cada documento en la lista; de tal forma que documentos relevantes en las primeras posiciones aportarán más ganancia que si se ubican al final. Este valor acumulado se normaliza dividiendo por la ganancia del ordenamiento ideal. Por otro lado, *MAP*, en lugar de utilizar la relevancia de los documentos, utiliza una clasificación binaria en la que los documentos relevantes toman el valor de 1, y 0 en caso contrario. De igual forma que *NDCG*, esta métrica aporta más peso a documentos relevantes en las primeras posiciones.

El hecho de que estas métricas están basadas en la posición de los documentos presenta un desafío para las técnicas de optimización, ya que no están pensadas para casos discontinuos o no diferenciables. Para superar este desafío se han propuesto diferentes técnicas que permiten hacer aproximaciones continuas de las métricas de IR utilizando algoritmos como *SoftRank*, *AppRank* y *AdaRank*. (Liu, 2009)

2. Listwise con optimización de permutaciones

Con esta técnica los modelos evalúan todas las permutaciones posibles para ordenar la lista. Es decir, calcula la probabilidad de que cada ordenamiento posible sea el correcto. La función de pérdida evalúa la diferencia entre la propuesta del modelo y el verdadero ordenamiento. En este caso el modelo no está optimizando directamente una métrica de ranking como *NDCG*, sin embargo, suelen tener buenos resultados. El modelo más utilizado con esta técnica es *ListNet* (Liu, 2009). Este documento se enfocará principalmente en esta subcategoría.

***ListNet***

Este algoritmo, propuesto por Zhe Cao, de Microsoft en 2007 (Cao et al., 2007), es un modelo de redes neuronales, en donde la función de score se basa en el concepto de *top one probability*. Utiliza gradiente descendente como algoritmo de optimización y entropía cruzada como función de pérdida.

Como se mencionó anteriormente, este modelo se basa en las probabilidades de permutación. Sin embargo, ya que calcular la probabilidad de cada ordenamiento es muy costoso computacionalmente, este modelo hace una simplificación llamada *top one probability*. Esto consiste en calcular para cada elemento de la lista, únicamente la

probabilidad de estar en la primera posición. De esta manera dado un *query* $q$ con $n$ documentos, la función $f(x)$ de la red generará una lista de scores $z$. Por tanto, *top one probability* para cada documento en base a esta lista será:

$$P_{z(f)}\left(x_j\right) = \frac{exp(f\left(x_j\right))}{\sum_{k=1}^{n} exp(f\left(x_j\right))}$$

En base a esta probabilidad se formula la función de pérdida utilizando entropía cruzada, donde $P_y$ es la verdadera probabilidad de que $x_j$ sea el primero en la lista.

$$L(y, z(f)) = -\sum_{j=1}^{n} P_y\left(x_j\right) log(P_{z(f)}\left(x_j\right))$$

El resultado final del modelo será cada documento con una probabilidad de ser el primero en la lista. De esta forma, se ordenan todos los documentos y se obtiene el ranking de cada *query*.

La única diferencia de este modelo con *RankNet* es que utiliza una función de pérdida *listwise*, pero en el caso de que los *queries* solo tengan dos documentos, el modelo será exactamente igual que *RankNet*. (Cao, 2007).

## 3. Aplicaciones

Con el fin de evaluar los modelos de ranking, en este trabajo se presentan dos aplicaciones que tienen un gran impacto en negocios de e-commerce o marketplace. La primera es Information Retrieval y la segunda es modelos de recomendación.

Information Retrieval (IR) consiste en, dado una necesidad de información y un conjunto de opciones, ordenar las opciones de más a menos relevantes y presentar un subconjunto de aquellas con mayor relevancia (Baeza-Yates, 1999). Aunque la aplicación más común de IR son búsquedas de páginas web, éste también juega un papel esencial en plataformas de e-commerce. Por ejemplo, presentar los resultados de una búsqueda en Amazon o en Mercado Libre requiere contar con un modelo que seleccione y ordene los productos para presentar al usuario los que más se acerquen a sus necesidades. Wayfair, un e-commerce multimillonario de venta de muebles, implementa técnicas de LTR como parte fundamental de su motor de búsqueda (Sonawane, 15). Otro ejemplo de éxito es Foursquare, una aplicación basada en geolocalización, que entre otras cosas, permite a los

usuarios buscar comercios cercanos. En 2013 presentó un estudio donde muestra que, al aplicar un modelo *LambdaMart*, la probabilidad de que la opción elegida por el usuario se encuentre en el top 5 seleccionado por el modelo es 82%; superando otros siete modelos bajo estudio (Shaw et al., n.d.).

Con el objetivo de comparar el desempeño de LTR con modelos de regresión y clasificación en problemas de IR, se aplicaron estas técnicas en dos bases diferentes. Ambas son bases públicas que se han utilizado en múltiples estudios de este ámbito, y que representan las mejores prácticas en temas de construcción de variables para este tipo de aplicaciones. Éstas son: Microsoft Learning to Rank -Web 30K[7] y Yahoo! Machine Learning to Rank versión 1.0[8]. Ambas bases contienen información sobre *queries* de búsquedas web y *urls* de páginas relacionadas. El objetivo es ordenar las páginas de acuerdo con la relevancia que representan para el *query*.

La otra aplicación bajo estudio es en modelos de recomendación. Estos sistemas juegan un papel fundamental en diferentes tipos de negocio. En un estudio realizado por Dokyun Lee de Carnegie Mellon University, se muestra que el uso de modelos de recomendación en plataformas de e-commerce incrementa tanto el número de vistas de productos como la probabilidad de conversión del usuario (Lee, 2020). En estos sistemas, es fundamental contar con un algoritmo que ordene las recomendaciones y muestre al usuario aquellas que más se relacionen a sus necesidades. Pinterest, por ejemplo, obtuvo un incremento del 30% en retención de usuarios al incorporar modelos de ranking en su algoritmo de recomendación de *pins* (Liu et al., 2017). Otro caso de éxito es Skyscanner, una aplicación para búsqueda de vuelos que implementó modelos de ranking para recomendar itinerarios de viaje; obteniendo como resultado un incremento en las tasas de conversión (Lathia, 2017).

Con el propósito de evaluar estos modelos en un problema real de recomendación de productos, se seleccionó como tercera base, los datos publicados en la competencia realizada por Mercado Libre en 2020[9]. Mercado Libre reconoce la importancia de contar con un buen sistema de recomendación para mejorar la experiencia de los usuarios y obtener mayores tasas de retención y conversión. Es por esto que organizaron una competencia en la

[7] https://www.microsoft.com/en-us/research/project/mslr/
[8] https://webscope.sandbox.yahoo.com/catalog.php?datatype=c&did=44
[9] https://ml-challenge.mercadolibre.com/downloads

que se propongan soluciones que seleccionen y ordenen las publicaciones que más se relacionan a la actividad del usuario. Con estos datos, en este trabajo, se evalúa el desempeño de modelos de ranking y se compara con otras técnicas de aprendizaje supervisado. A continuación se brinda una explicación más detallada de estas bases.

## 3.1 Microsoft Learning to Rank - WEB 30K

### 3.1.1 Descripción de los datos

Esta base fue publicada por Microsoft en junio del 2010. Contiene registros que consisten en pares de *queries* de búsqueda y *urls*, acompañados por la relevancia del *url* respecto a la búsqueda. Esta base cuenta con 136 variables descriptivas del *url* y de su relación con los parámetros del *query*. La variable que representa la relevancia del *url* proviene de un sistema de etiquetado de Microsoft Bing que toma cinco valores (0-4), entre más alto sea el valor, más relevancia tiene. La Figura 3 muestra el formato de los datos, donde cada línea corresponde a un par *query-documento*. El primer elemento de la línea es la importancia del documento, seguido por el número de identificación (id) del *query*. Las variables se muestran en el formato $[id\ de\ la\ variable\colon\ valor]$.

*Figura 3. Datos de Microsoft*

```
============================

0 qid:1 1:3 2:0 3:2 4:2 ... 135:0 136:0

2 qid:1 1:3 2:3 3:0 4:0 ... 135:0 136:0

============================
```

A continuación, en la Tabla 1, se presentan las variables que contiene este dataset. Cada una de las siguientes descripciones están disponibles para el *body*, *anchor*, *title*, *url* y todo el documento. Todas están normalizadas.

*Tabla 1. Descripción de variables de Microsoft*

| Variable | Descripción |
|---|---|
| Covered query term number | Cuántos términos de la búsqueda están cubiertos en el *url* |
| Covered query term ratio | Número de términos de la búsqueda encontrados en el *url*, dividido por el número total de términos de la búsqueda |
| stream length | Longitud del texto |
| IDF | 1 dividido el número de documentos que contienen términos de la búsqueda |
| Sum/ min/ max/ mean/ variance of term frequency | Suma/ min/ max/ media/ varianza de la cantidad de veces que aparece cada término en el documento |
| Normalized Sum/ min/ max/ mean/ variance of stream length | Suma/ min/ max/ media/varianza dividida entre la longitud del texto |
| Sum/ min/ max/ mean/variance of tf*idf | Suma/ min/ max/ media/varianza del producto entre la frecuencia y el *idf* de cada término |
| Boolean model | La descripción de esta variable es de uso privado de Microsoft |
| vector space model | Producto entre el vector del *query* y el del documento. Son privados de Microsoft |
| BM25 | Okapi BM25[10] |
| LMIR.ABS | Modelo de lenguaje estadístico para *IR* utilizando *absolute discounting smoothing* [11] |

[10] https://nlp.stanford.edu/IR-book/html/htmledition/okapi-bm25-a-non-binary-model-1.html
[11] https://sigir.org/wp-content/uploads/2017/06/p268.pdf

| | |
|---|---|
| LMIR.DIR | Modelo de lenguaje estadístico para IR utilizando *Bayesian smoothing direct priors* [12] |
| LMIR.JM | Modelo de lenguaje estadístico para IR utilizando *Jelinek-Mercer smoothing*[13] |
| Number of slashes in url | Número de "/" que contiene el *url* |
| Length of url | Número de caracteres del *url* |
| Inlink number | Número de páginas web que han citado este *url* |
| Outlink number | Cuántas páginas web cita este *url* |
| Page Rank | Centralidad de esta página web basado en los links de Internet |
| QualityScore | Este score calcula la calidad de la página web. Este cálculo es realizado por un clasificador privado de Microsoft |
| QualityScore2 | Este score calcula el "badness" de la página web, es calculado por un clasificador privado de Microsoft |
| Query-url click count | Número de clics del par *query-url* en un periodo determinado |
| url click count | Número de clics a la página web en un periodo determinado |
| url dwell time | Tiempo promedio en el que el usuario interactúa con la página después de haber sido sugerida por el motor de búsqueda (dwell time) |

[12] Ibiden
[13] Ibiden

Los datos publicados por Microsoft están divididos en 5 *folds*, cada uno de éstos contiene un dataset de entrenamiento, validación y test. A modo de referencia se presenta el número de *queries* y documentos contenidos en el primer fold.

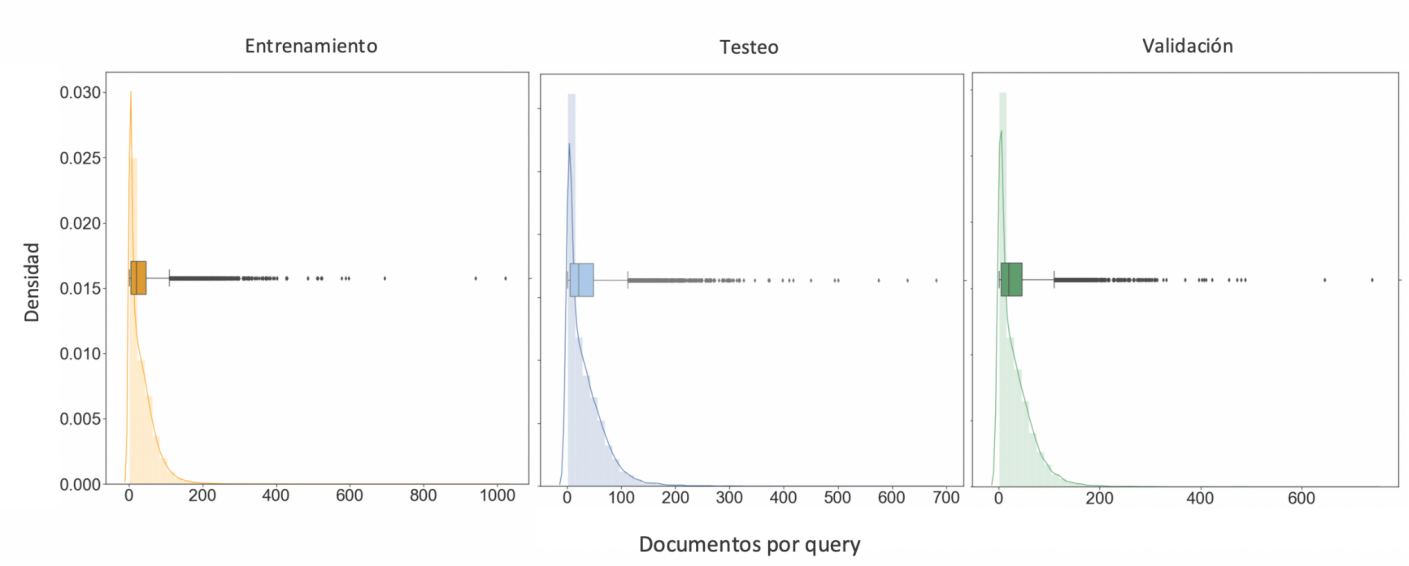
*Tabla 2. Dimensión de datasets de Microsoft*

| | **Entrenamiento** | **Validación** | **Test** |
|---|---|---|---|
| Queries | 70.688 | 23.599 | 23.740 |
| Pares query-documento | 2.270.296 | 747.218 | 753.611 |

### 3.1.2 Análisis exploratorio

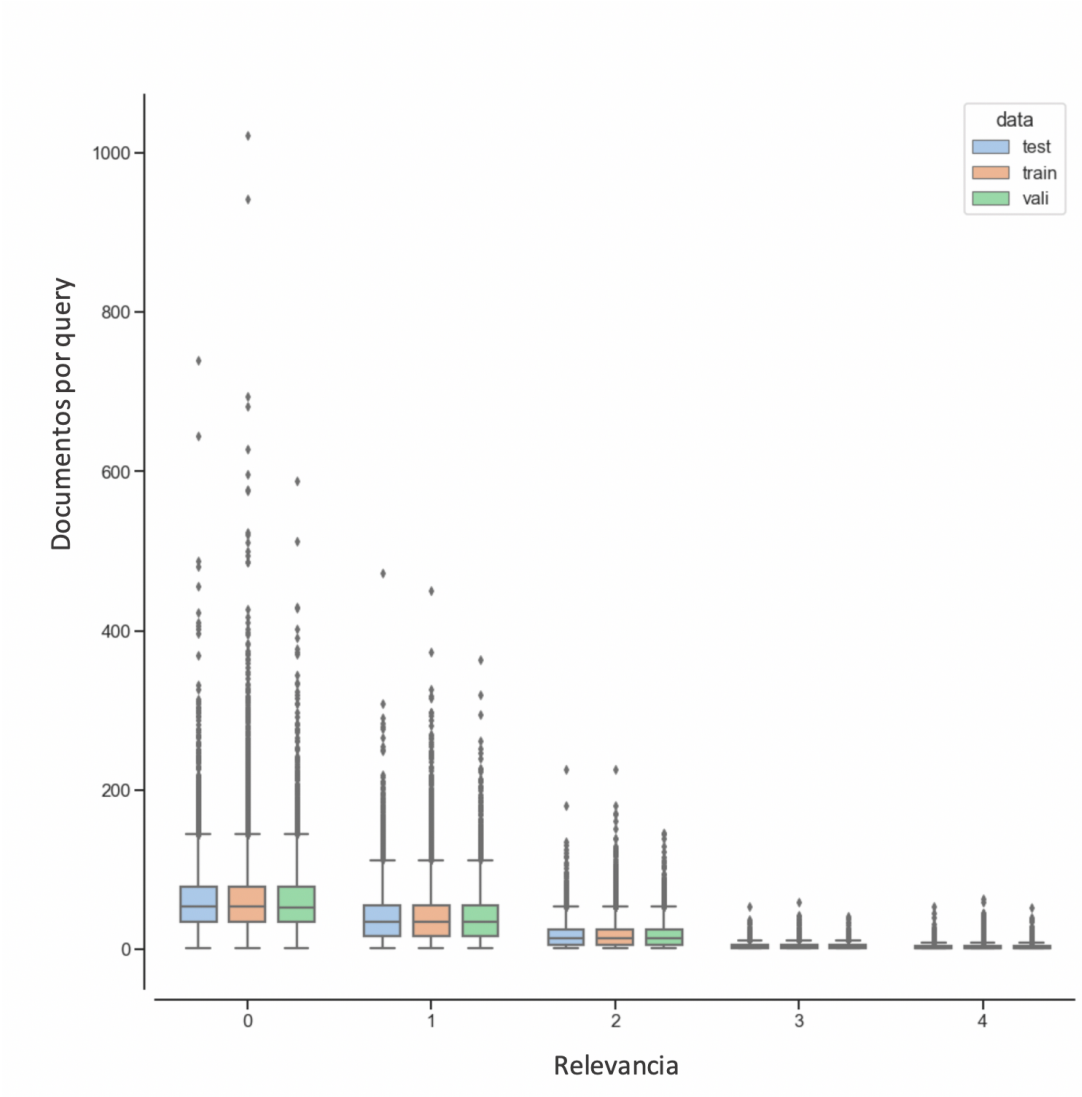
El siguiente gráfico muestra la distribución del número de documentos por *query* para cada dataset, tomando como referencia el primer fold. En promedio cada *query* tiene 35 documentos, sin embargo hay muchos considerados outliers ya que tienen más de 50 *urls*. Esta disparidad puede causar en modelos *pairwise* y *pointwise* que la función de pérdida se vea más influenciada por estos *queries* con muchos documentos.

*Gráfico 1. Microsoft - distribución de documentos por query*



El Gráfico 2 muestra la distribución del número de documentos de cada relevancia por *query*. Como se puede ver hay mayor cantidad de documentos con relevancia baja (0-2) y muy pocos documentos con relevancia entre 3 y 4. Los modelos de ranking deberían ser capaces de identificar estos pocos documentos y colocarlos en las primeras posiciones de la lista.

*Gráfico 2. Microsoft - distribución de documentos por relevancia*



## 3.2 Yahoo! Machine Learning to Rank Challenge version 1.0

### 3.2.1 Descripción de los datos

Este dataset, publicado por Yahoo!, contiene pares de *queries* y *url*. Al igual que la base de Microsoft, el problema que se busca resolver es proporcionar al usuario una lista de páginas web ordenadas, colocando al principio las más relevantes para la búsqueda.

El formato de los datos es igual al de Microsoft, con la diferencia de ser esparzas. Es decir, no todos los documentos tienen todas las variables.

*Figura 4. Datos de Yahoo*

```
==============================
0 qid:1 1:3 3:2 4:2 5:6 … 518:0 519:0
2 qid:1 1:3 2:3 3:0 4:0 … 517:0 518:0
==============================
```

Los datos están divididos en tres conjuntos: entrenamiento, validación y test.

*Tabla 3. Dimensión de datasets de Yahoo*

| | **Train** | **Validación** | **Test** |
|---|---|---|---|
| Queries | 64.989 | 9.836 | 22.889 |
| URLS | 473.134 | 71.083 | 165.660 |
| Variables | 519 | 519 | 519 |

La descripción de las variables no es proporcionada con tanto detalle como en el caso de Microsoft. Yahoo proporciona únicamente una descripción general de los diferentes tipos de variables (Chapelle & Chang, 2011).

- **Variables de grafo:** Variables de red como *PageRank*[14], *indegree*[15] y *outdegree*[16].
- **Estadísticas propias del documento**: Por ejemplo, cantidad de palabras y de "/" en el *url*.

[14] Patente de Google, familia de algoritmos que a través de métricas de red, asigna de forma numérica la relevancia de los documentos indexados por un motor de búsqueda
[15] Número de urls que dirigen hacia una página web
[16] Número de urls que cita una página web

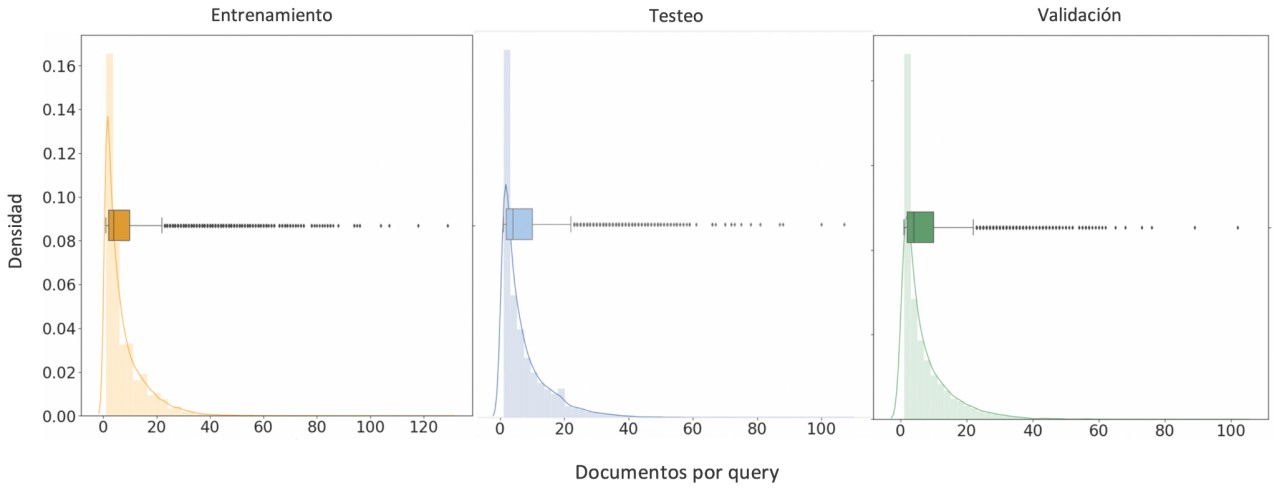
- **Clasificadores del documento:** Variables binarias que indican características del documento, por ejemplo: idioma, tema principal, calidad, tipo de página, contenido para adultos y spam.
- **Descripción del query:** Variables propias de la búsqueda, por ejemplo: número de palabras, frecuencia de las palabras, *click-through-rate* [17]
- **Similitudes de texto entre el query y los urls:** Estas variables evalúan con diferentes métricas como, suma, máximo, mínimo y promedio, cuántas veces los términos de la búsqueda están contenidos en la página web. Estas variables están construidas para diferentes partes del documento: título, contenido principal y *url*.
- **Tema del documento:** Este set de variables, compara si el tema principal del *url* coincide con la temática de la búsqueda.
- **Clicks:** Este conjunto de variables mide las siguientes probabilidades para los *urls*, dado los parámetros de búsqueda: probabilidad de que reciba al menos un clic, probabilidad de que sea el último o el primero en recibir un clic y probabilidad de no recibir ningún clic, pero que un *url* ubicado más abajo en la lista si lo reciba.
- **Variables de tiempo:** Entre estas variables se encuentra, por ejemplo, cuánto tiempo toma en cargar la página y la "edad" de la misma.

### 3.2.2 Análisis exploratorio

En el siguiente gráfico se puede observar la distribución del número de documentos por *query* en cada dataset. Cada *query* tiene en promedio 7 documentos relacionados. Sin embargo, al igual que en el dataset de Microsoft, hay *queries* considerados outliers que tienen más de 20 documentos.

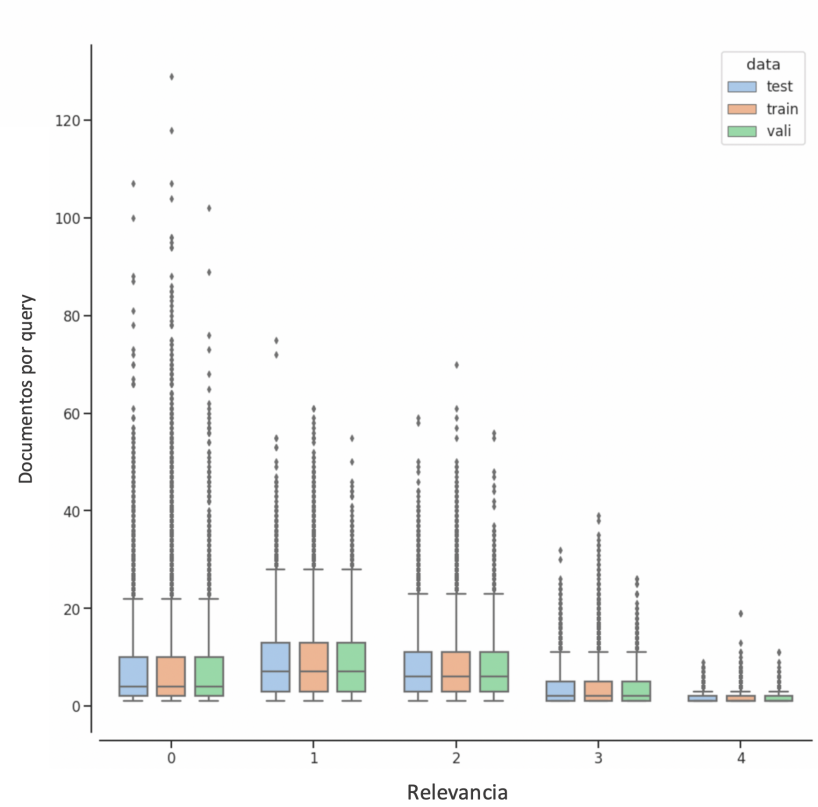
[17] Ratio entre el número de clicks y el número de impresiones o resultados de la búsqueda

*Gráfico 3. Yahoo distribución de documentos por query*




En el gráfico 4 se puede observar la distribución de documentos por *query* y por relevancia. Como se puede ver hay una considerable diferencia entre el número de documentos con el mayor grado de relevancia (4) comparado con *urls* menos relevantes (0-3).

*Gráfico 4. Yahoo distribución de documentos por relevancia*



## 3.3 Mercado Libre Data Challenge 2020

### 3.3.1 Descripción de los datos

Mercado Libre, la empresa de e-commerce referente en Latinoamérica, realiza anualmente desafíos para motivar a la comunidad de científicos de datos a resolver problemas reales de este ámbito. En 2020 el desafío consistió en realizar un modelo para hacer 10 recomendaciones personalizadas a sus usuarios, basadas en las publicaciones que son más cercanas a las necesidades y preferencias de cada cliente. La base contiene un dataset de entrenamiento con 413.163 historiales de navegación y un dataset de test con 177.070.

Los datos fueron proporcionados en dos archivos de formato *JSON.* A continuación se describe cada uno de ellos.

*Historial de navegación*

Este archivo contiene una línea por cada compra. Los atributos consisten en el historial de navegación de una semana previa y hasta 2 horas antes de realizar la compra. Además, el dataset de entrenamiento contiene el identificador del ítem comprado.

*Tabla 4. Variables de historial de navegación de Mercado Libre*

| Atributo | Descripción |
|---|---|
| user_history | Lista de tuplas, cada una contiene:<br>● *event_type:* view o search<br>● *event_timestamp:* momento en el que tuvo lugar el evento<br>● *event_info:* si event_type es search contiene el query de la búsqueda, de lo contrario contiene el item_id visto. |
| item_bought (variable target) | Número único de identificación del ítem comprado |

La figura 4 muestra el formato original de este archivo

*Figura 4. Formato de datos de historial de navegaciones*


```
{
  "user_history": [
    {
      "event_info": 1986443,
      "event_timestamp": "2019-10-16T18:14:51.587-0400",
      "event_type": "view"
    },
    {
      "event_info": "ACCESORIOS MAZDA3 2019",
      "event_timestamp": "2019-10-16T18:16:18.340-0400",
      "event_type": "search"
    },
    {
      "event_info": "ACCESORIOS MAZDA3 2019",
      "event_timestamp": "2019-10-16T18:16:20.052-0400",
      "event_type": "search"
    },
    {
      "event_info": 991246,
      "event_timestamp": "2019-10-16T18:16:34.598-0400",
      "event_type": "view"
    }
  ],
  "item_bought": 678901
}
```


*Publicaciones*

Esta base contiene el detalle de todos los productos publicados, con un total de 2.102.277 registros.

*Tabla 5. descripción de variables de dataset de publicaciones*

| Atributo | Descripción |
|---|---|
| item_id | Identificación única del producto |
| title | Título de la publicación |
| price | Precio del artículo en USD |
| category_id | id de la categoría |
| product_id | id del producto. No todos los productos tienen este atributo |
| domain_id | id de dominio. Un dominio es una agrupación de publicaciones. No tiene una relación explícita con el árbol de categoría. |
| condition | Si la publicación se refiere a un producto nuevo o usado. |

La siguiente figura muestra el formato original de esta base

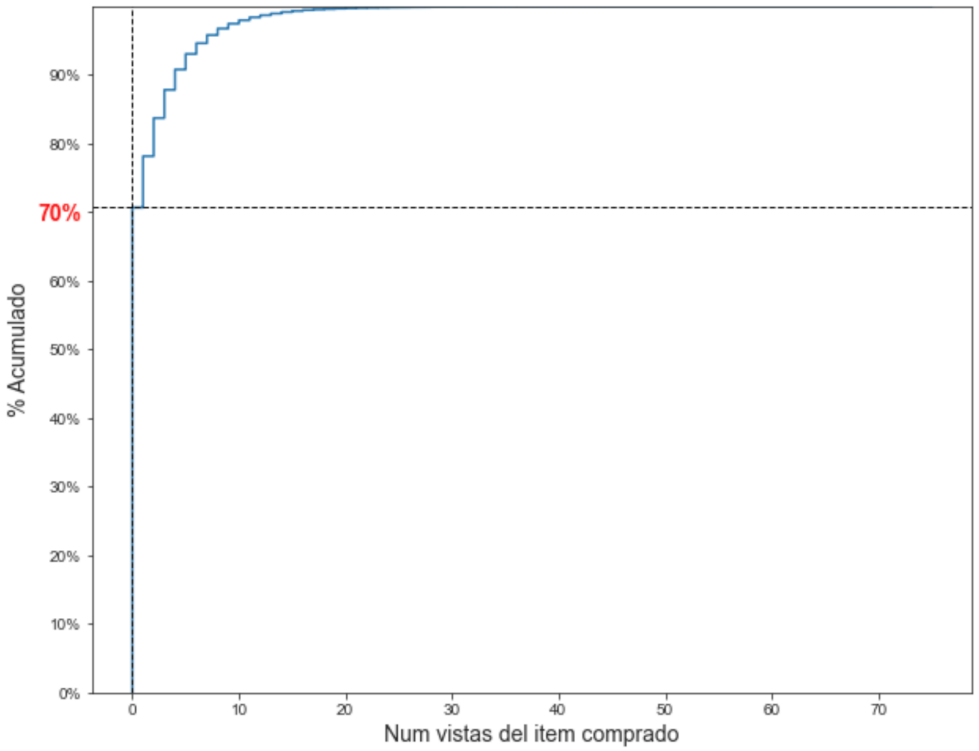
*Figura 5. Formato de datos de publicaciones de Mercado Libre*


```
{
  "item_id": 71031,
  "title": "Corriente Galvanica Tipo Nuskin + Regalo ",
  "domain_id": "MLM-ESTHETIC_ELECTRIC_DEVICES",
  "product_id": null,
  "price": "2499.00",
  "category_id": "MLM194097",
  "condition": "new"
}
```


### 3.3.2 Análisis exploratorio

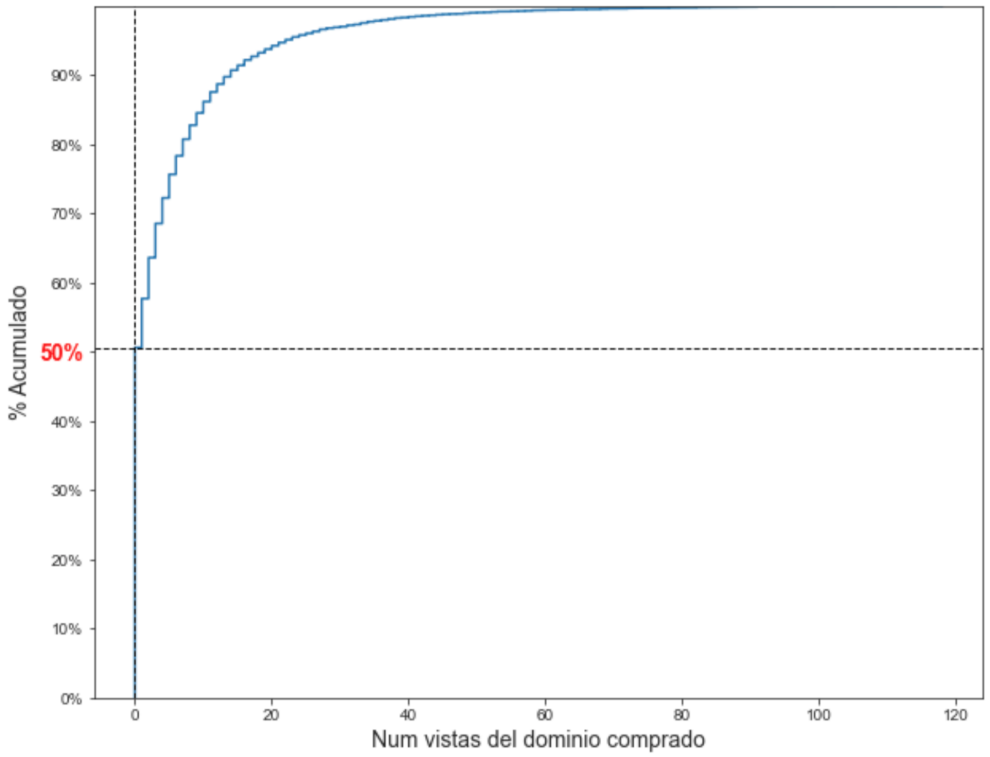
El Gráfico 5 muestra la distribución acumulada del número de veces que el usuario visitó el ítem comprado. Como se puede ver, solamente el 30% de los historiales de sesión visitaron el producto que compraron. Esto se debe a que los datos incluyen únicamente las actividades del usuario hasta 2 horas antes de la compra.

*Gráfico 5. Mercado Libre - Porcentaje acumulado de vistas al ítem comprado*



En el Gráfico 6 se puede observar el porcentaje acumulado del número de veces que los usuarios visitan ítems del mismo dominio que el producto comprado. El porcentaje de usuarios que visitaron el mismo dominio por lo menos una vez es 50%. Si comparamos esto con el gráfico anterior, podemos intuir que al momento de entrenar el modelo, tendrá mucha importancia considerar ítems del mismo dominio que los ítems visitados.

*Gráfico 6. Mercado Libre - Porcentaje acumulado de vistas del dominio comprado*



Dado que la base de publicaciones contiene más de dos millones de registros, es de suma importancia analizar si es necesario incluir todos los productos en el entrenamiento del modelo. El siguiente gráfico muestra el porcentaje de publicaciones que fueron compradas al menos una vez, de acuerdo a la base de historiales de navegación. Como se puede ver, únicamente 3.1% de los 2 millones de publicaciones tienen una compra. Esto reduce el universo de productos a 65 mil. Al hacer este mismo análisis con los dominios, se obtiene que aproximadamente el 60% de ellos tienen al menos una compra.

*Gráfico 7. Mercado Libre - Porcentaje de publicaciones con al menos una compra*

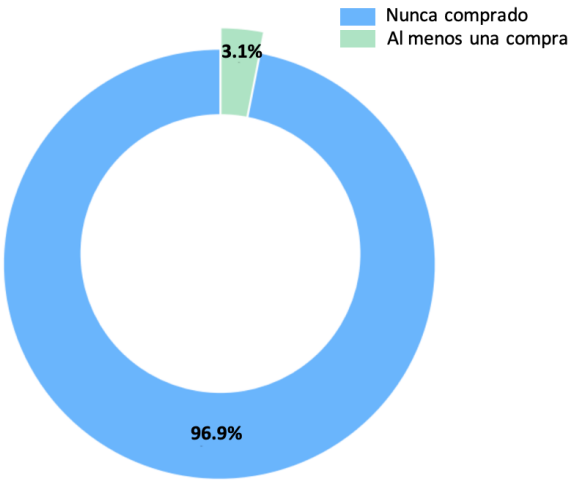
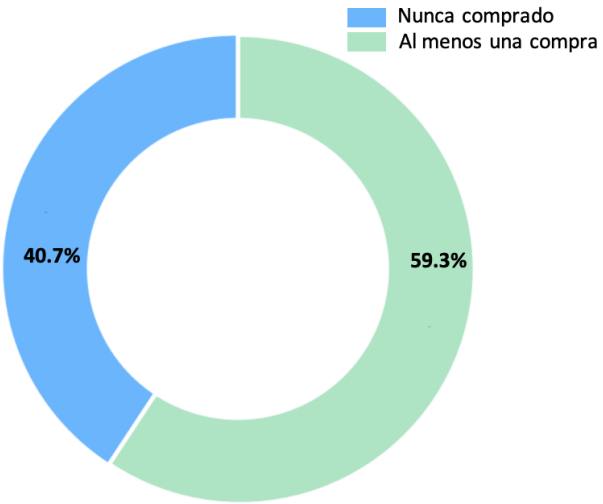
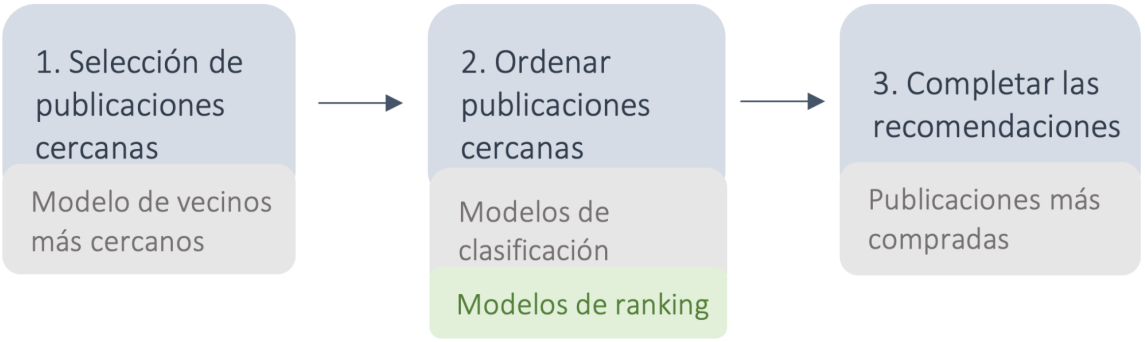



*Gráfico 8. Mercado Libre - Porcentaje de dominios con al menos una compra*



### 3.3.3 Solución propuesta

Con el propósito de evaluar el desempeño de modelos de ranking para resolver este problema de recomendación de productos, se optó por utilizar la solución del tercer lugar de la competencia como punto de partida. Esta solución propone el siguiente esquema, en verde se indica la modificación propuesta en este trabajo:




1. *Selección de publicaciones cercanas*

   Dado que el conjunto de datos de Mercado Libre cuenta con millones de publicaciones, el primer paso es reducir estas opciones. Para esto, la solución de referencia propone hacer un modelo de vecinos más cercanos que compare las historias de navegación. Este algoritmo va a identificar, para cada registro, las 50 navegaciones más parecidas y propondrá como recomendación los productos comprados por estas navegaciones. De esta manera, cada registro del dataset tendrá hasta 50 opciones para recomendar.

2. *Ordenar publicaciones cercanas*

   Una vez preseleccionadas las publicaciones es necesario un modelo que identifique cuáles de estas son las más relevantes para el usuario. Para esto, la solución original propone un ensamble de modelos de clasificación en los cuales la variable a predecir es igual a 1 si el producto propuesto es el comprado y 0 en caso contrario.

   De esta manera, en la solución de referencia, se construyen tres modelos de clasificación: *LightGBM*, *XGBoost* y *Catboost*. Luego se obtiene un score final sumando las probabilidades obtenidas en cada uno.

3. *Completar recomendaciones*

Dado que esta solución utiliza modelos de clasificación binaria, confía únicamente en las dos recomendaciones que obtuvieron la mejor puntuación. Estos dos ítems son colocados en las primeras posiciones de la lista. Las recomendaciones restantes se completan con los ítems más vendidos del dominio de la publicación que obtuvo la mayor puntuación en el modelo. Por ejemplo, si la primera recomendación es del dominio "smart watch" las últimas 8 posiciones de la lista serán ocupadas por productos del dominio "smart watch".

***Mejoras propuestas***

Es claro que los modelos de clasificación no son ideales para ordenar los productos en base a su relevancia, ya que consideran una variable objetivo binaria que deja de lado aquellos productos que, a pesar de no ser la compra del usuario, también pueden ser relevantes por pertenecer a la misma categoría o dominio. Para solucionar esta deficiencia, este trabajo propone una variable a predecir con distintos niveles de relevancia.

*y = 12 si el producto evaluado es el ítem comprado*
*y = 1 si el producto evaluado es del mismo dominio del producto comprado*
*y = 0 si no cumple ninguno de los criterios anteriores*

Esta definición es la misma que utiliza Mercado Libre para evaluar los resultados de la competencia[18]. Utilizando esta variable, este trabajo propone evaluar, además de la solución del tercer lugar, los siguientes modelos:

- *Metodología pointwise - Regresión lineal:* Se utilizó la librería de Python LightGBM, específicamente el módulo LightGBMRegressor, que utiliza técnicas de *Gradient Boosting Machines*.

[18] Más información sobre esta métrica, en 4.4 Métricas de evaluación

- *Metodología pairwise - LambadaMart:* Con el propósito de evaluar un modelo de ranking con función de pérdida *pairwise*, se seleccionó *LambdaMart* y se utilizó el modulo de ranking de LightGBM.
- *Metodología listwise - ListNet:* Se utilizó Tensor Flow Ranking, un módulo de Tensor Flow que permite implementar el modelo *ListNet* mencionado anteriormente.

Dado que el objetivo de este estudio es comparar el desempeño de los modelos por sí mismos, los resultados presentados en este trabajo omiten el último paso de "completar las recomendaciones". Sin embargo, en el Anexo 2, se muestran los resultados de todos los modelos después de completar este último paso.

# 4. Implementación

## 4.1 Microsoft Learning to Rank - WEB 30K

Para recomendar las mejores páginas web en cada búsqueda se implementaron tres modelos, manteniendo la separación original de 5 *folds.* De esta manera, se entrenaron 5 modelos para cada técnica. Para optimizar los hiperparámetros, se utilizó la técnica de *random search,* probando 100 combinaciones. A continuación se presenta la grilla y los valores óptimos para cada uno.

- *Metodología pointwise - Regresión lineal:* Se utilizó la librería de Python *LightGBM*, específicamente el módulo *LightGBMRegressor*, que utiliza técnicas de *Gradient Boosting Machines*. La variable a predecir es la relevancia del *url* (0-4). Se utilizaron los siguientes parámetros:

*Tabla 6. Microsoft - hiperparámetros del modelo de regresión*

| Hiperparámetro | Grilla | Óptimos |
|---|---|---|
| n_estimators | [200 - 1100] | 900 |
| colsample_bytree | [0.5 - 0.9] | 0.6 |
| max_depth | [10 - 55] | 45 |
| num_leaves | [10 - 55] | 50 |
| reg_alpha | [1.0 - 1.8] | 1.75 |
| reg_lambda | [1.0 - 1.8] | 1.6 |
| min_split_gain | [0.3 - 0.8] | 0.55 |
| subsample | [0.3 - 0.9] | 0.8 |
| metirc | | rmse |
| learning_rate | | 0.1 |
| num_threads | | 2 |

- *Metodología pairwise - LambadaMart:* Con el propósito de evaluar un modelo de ranking con función de pérdida *pairwise*, se seleccionó *LambdaMart* y se utilizó LightGBM, especificando los siguientes hiperparámetros:

*Tabla 7. Microsoft - hiperparámetros del modelo LambdaMart*

| Hiperparámetro | Grilla | Óptimos |
|---|---|---|
| n_estimators | [200- 1100] | 300 |
| colsample_bytree | [0.5 - 0.9] | 0.70 |
| max_depth | [10 - 55] | 15 |
| num_leaves | [10 - 55] | 25 |
| reg_alpha | [1.0 - 1.8] | 1.40 |
| reg_lambda | [1.0 - 1.8] | 1.60 |
| min_split_gain | [0.3 - 0.8] | 0.675 |
| subsample | [0.3 - 0.9] | 0.450 |
| objective | | lambdarank |
| metirc | | ndcg |
| learning_rate | | 0.1 |
| num_threads | | 2 |

- *Metodología listwise - ListNet:* Se utilizó Tensor Flow Ranking, un módulo de Tensor Flow que permite implementar el modelo *ListNet* mencionado anteriormente. Por limitaciones de memoria no se optimizaron los hiperparámetros en este modelo.

*Tabla 8. Microsoft - hiperparámetros del modelo ListNet*

| Hiperparámetro | Valor |
|---|---|
| list_size | 100 |
| batch_size | 32 |
| hidden_layers_dims | [256,128,64] |
| num_train_steps | 10 |
| dropout_rate | 0.5 |
| learning_rate | 0.01 |
| loss_function | approx_ndcg_loss |

## 4.2 Yahoo! Machine Learning to Rank Challenge version 1.0

Al igual que con la base de Microsoft, se probaron tres modelos manteniendo los datasets originales. Se optimizaron probando 30 combinaciones de hiperparámetros con *random search.* A continuación se presentan los valores utilizados:

- Regresión lineal

*Tabla 8. Yahoo - hiperparámetros de modelo de regresión*

| Hiperparámetro | Grilla | Óptimos |
|---|---|---|
| n_estimators | [200 - 1100] | 1000 |
| colsample_bytree | [0.3 - 0.9] | 0.50 |
| max_depth | [10 - 55] | 50 |
| num_leaves | [20 - 75] | 50 |
| reg_alpha | [0.5 - 1.5] | 1.0 |
| reg_lambda | [1.3 - 1.9] | 1.80 |
| min_split_gain | [0.3 - 0.8] | 0.425 |
| subsample | [0.3 - 0.9] | 0.80 |
| metirc | | rmse |
| learning_rate | | 0.1 |
| num_threads | | 2 |

- LambdaMART

*Tabla 9. Yahoo - hiperparámetros del modelo LambdaMart*

| Hiperparámetro | Grilla | Óptimos |
|---|---|---|
| n_estimators | [200 - 1100] | 400 |
| colsample_bytree | [0.3 - 0.9] | 0.45 |
| max_depth | [10 - 55] | 25 |
| num_leaves | [10 - 75] | 20 |
| reg_alpha | [0.5 - 1.5] | 0.75 |
| reg_lambda | [1.1 - 1.9] | 1.30 |
| min_split_gain | [0.3 - 0.8] | 0.525 |
| subsample | [0.3 - 0.9] | 0.35 |
| objective | | lambdarank |
| metirc | | ndcg |
| learning_rate | | 0.1 |
| num_threads | | 2 |

- ListNet

*Tabla 10. Yahoo - hiperparámetros del modelo ListNet*

| Hiperparámetro | Valor |
|---|---|
| list_size | 20 |
| batch_size | 32 |
| hidden_layers_dims | [256,128,64] |
| num_train_steps | 10 |
| dropout_rate | 0.5 |
| learning_rate | 0.01 |
| loss_function | approx_ndcg_loss |

## 4.3 Mercado Libre Data Challenge 2020

### 4.3.1 Ingeniería de atributos

Con el propósito de evaluar el desempeño de los modelos de ranking, se utilizó la misma ingeniería de atributos de una de las soluciones más competitivas (Veiga, 2020). A continuación se presenta una descripción de los atributos.

***Atributos para pre-selección de publicaciones***

Como se mencionó anteriormente, el primer paso es entrenar un modelo de vecinos más cercanos, que tiene como objetivo encontrar historiales de navegación similares y seleccionar los ítems comprados por estas sesiones como publicaciones candidatas a recomendar. Para esto se construyen los siguientes atributos:

- Publicaciones vistas tf_idf: Para cada historial de navegación se recopilan los últimos 20 productos visitados, y se filtran por aquellos que, en conjunto con todos los registros del dataset, han sido visitados más de una vez, con el fin de reducir las publicaciones a aquellas más importantes. Estos productos se modelan como palabras de un *Bag Of Words*[19]. En esta analogía el *corpus*[20] sería el conjunto de publicaciones visitadas más de una vez, los documentos serían los historiales de sesión y los términos serían los productos. Con la información del número de veces que el ítem fue visitado durante la navegación se calcula TF-IDF (Term Frequency-Inverse Document)[21].
- Dominios vistos tf_idf: La misma lógica utilizada con las publicaciones vistas, se replica para calcular TF-IDF de los dominios de las publicaciones. En este caso los términos corresponden a la variable domain_id.

[19] Método que se utiliza en el procesamiento del lenguaje para representar documentos (páginas web, libros o cualquier elemento que contenga texto) ignorando el orden de las palabras. Con este modelo se obtiene una representación de cada documento, en función de las palabras que contiene.
[20] Colección de documentos
[21] Es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección o corpus, al comparar el número de veces que una palabra aparece en el documento con el número de documentos que contienen dicha palabra

- Queries de búsqueda tf_idf: Otra información que puede ser relevante para encontrar similitudes entre sesiones son los términos de búsqueda. Para procesar esta información se recurre de nuevo a la técnica de *Bag of Words*. Se concatenan todas las búsquedas de cada sesión en un solo texto. En este caso, el corpus sería el conjunto de todas las búsquedas del dataset, los términos serían cada palabra del corpus y los documentos, los historiales de búsqueda.

Como resultado de la construcción de estos atributos, el modelo de vecinos más cercanos es entrenado con un arreglo esparzo que contiene una línea por cada historial de sesión y una columna por cada ítem visitado, dominio visitado y términos de las búsquedas realizadas.

***Atributos para ordenar las publicaciones***

Con las recomendaciones obtenidas en el modelo de vecinos más cercanos se construye una base que contiene un registro por cada par sesión-recomendación. Además, a cada sesión se le agrega como recomendación los últimos 10 items visitados. Para cada uno de estos registros se crean los siguientes atributos:

Variables propias de la publicación

- **Precio:** Precio del producto.
- **Condición:** Condición del producto; 0 si es nuevo, 1 si es usado y -1 si no está definido.

Variables relacionadas a otras sesiones

- **Compras en otras sesiones:** Número de veces que la publicación fue comprada por otros historiales de sesión.
- **Vistas en otras sesiones:** Número de veces que la publicación fue visitada por otros historiales de sesión.

Variables comparativas

- **Distancia knn:** Distancia entre la sesión en análisis y la sesión similar obtenida en el modelo de vecinos más cercanos.

- **Recomendaciones knn:** Número de veces que el ítem fue sugerido por el modelo de vecinos más cercanos.
- **Número de vistas:** Número de veces que el producto fue visitado en el historial de navegación.
- **Mejor producto:** Variable booleana que toma el valor de 1 si es el mejor ítem visitado. De los últimos 10 items visitados se evalúa cual de estos es el más comprado en otras sesiones, este es considerado el mejor ítem.
- **Producto más visitado:** Variable booleana que toma el valor de 1 si es el ítem más visitado en el historial de sesión .
- **Último producto visitado:** Variable booleana que toma el valor de 1 si es la última publicación visitada.
- **Mejor dominio:** Variable booleana que toma el valor de 1 si es el dominio del mejor producto.
- **Dominio más visitado:** Variable booleana que toma el valor de 1 si es el dominio de la publicación más visitada en el historial de sesión.
- **Última visita repetida:** Variable booleana que toma el valor de 1 si corresponde a la última publicación que el usuario visitó más de una vez.
- **Secuencia de visita:** Secuencia de la visita; 0 si fue la última sesión visitada, 1 si fue la penúltima y así sucesivamente.
- **Score final:** Para cada uno de los últimos 20 ítems se busca qué publicaciones compraron otros usuarios que visitaron el mismo producto. Con esta información se calcula un score que corresponde al número de veces que estos ítems fueron comprados. Por ejemplo, para una sesión que visitó $a$ y $b$, se obtendrá el siguiente diccionario: $\{ a: \{c: 4,\ e: 3\},\ b: \{c: 1,\ f: 2\}\}$

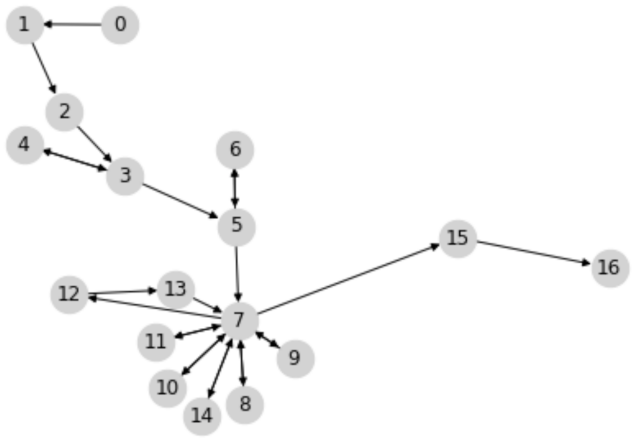
  Esto indica que hay 4 sesiones que visitaron$a$ y compraron $c$, mientras que hay 1 sesión que visitó $b$ y compró $c$.

  Se obtiene esta información para las últimas 20 publicaciones visitadas, y con esto se calcula un score que suma las compras de cada producto. En el ejemplo anterior el score de $c$ es 5, el de $e$ es 3 y el de $f$ es 2.
- **Número de búsquedas:** Número de búsquedas en el historial de sesión.

### Variables de grafo

Para cada historial de navegación se construye un grafo dirigido que relaciona el orden en el que el usuario visitó las publicaciones. Los nodos representan las publicaciones vistas, los enlaces dirigen hacia el siguiente producto visitado. En la figura 6 se puede ver un ejemplo. El primer ítem visitado (nodo 0) se dirige hacia la segunda visita (nodo 1). El nodo 5 y 6 tienen una relación bipartita, lo que significa que el usuario, después de visitar el séptimo ítem (nodo 6), vuelve de nuevo al sexto ítem visitado (nodo 5).

*Figura 6. Mercado Libre - Grafo dirigido de vistas en un historial de sesión*




Con esta información se calculan las siguientes métricas para cada nodo o vista:

- **indegree:** Número publicaciones que dirigen hacia esa vista.
- **outdegree:** Número de publicaciones que salen de esa vista.
- **betweenness:** Número de veces que la vista se encuentra en el camino más corto entre otras dos publicaciones.
- **shortest path to last item:** Cuántas publicaciones hay entre el nodo en análisis y la última publicación visitada.
- **closeness:** Número de vistas del camino más corto hacia el resto de vistas.
- **pagerank y eigenvector centrality:** Medidas de centralidad basadas en el número de conexiones que la publicación en análisis tienen con las otras vistas.

### 4.3.2 Entrenamiento de modelos

***Base de entrenamiento, validación y test***

Dado que la ingeniería de atributos involucra analizar otras sesiones en la base de datos, es de suma importancia separar los historiales de navegación en entrenamiento, validación y test desde el principio para evitar *data leakage*[22]. Los datos se separaron en 80% (330.530) de las sesiones para entrenamiento, 10% (41.317) para validación y 10% (41.316) para test.

Además, los modelos se evaluaron en otra base de validación que corresponde a la utilizada en el *leaderboard* público de la competencia. Esta base cuenta con 177.077 sesiones, de las cuales se evalúa únicamente el 70%.

***Modelo de vecinos más cercanos***

Para obtener las recomendaciones del dataset de validación y test, se entrena el modelo con todos los registros de la base de entrenamiento. Es decir, una sesión de la base de validación tendrá, únicamente, recomendaciones provenientes de la base de entrenamiento. Por otro lado, para generar las recomendaciones de las sesiones de entrenamiento, se separó esta base en 7 *folds*. De esta manera, se evita que el modelo utilice información de la misma sesión que se desea predecir. Las recomendaciones del dataset de entrenamiento provienen del 85% (6/7) de las sesiones de la base de entrenamiento.

Para simplificar el problema de vecinos más cercanos se utilizan únicamente los 40 mil productos más comprados. Los hiperparámetros utilizados son los siguientes:

*Tabla 11. Mercado Libre - hiperparámetros de modelo KNN*

| Hiperparámetro | Valor |
|---|---|
| n_neighbors | 11 |
| n_jobs | 1 |
| leaf_size | 30 |
| metric | cosine |

[22] Entrenar el modelo utilizando datos fuera de la base de entrenamiento, que uno se supone no debería de conocer. Esto generará una mayor precisión en dichos datos, pero que no se verá reflejada al aplicar el modelo en producción.

***Modelos para ordenar las recomendaciones***

Los siguientes modelos se optimizaron con una muestra de 10% de la base de entrenamiento y de validación. La técnica utilizada fue *random search,* evaluando 100 modelos. A continuación se presentan las grillas y los valores óptimos para cada uno.

*Modelos pointwise*

- **Regresión lineal:** Se utilizó el módulo *LGBMRegressor*, con la variable a predecir mencionada anteriormente, que toma los siguientes valores$\{0, 1, 12\}$.

*Tabla 12. Mercado Libre - hiperparámetros de modelo de regresión*

| Hiperparámetro | Grilla | Óptimos |
|---|---|---|
| n_estimators | [200 - 1100] | 300 |
| colsample_bytree | [0.7 - 0.9] | 0.80 |
| max_depth | [10 - 50] | 15 |
| num_leaves | [10 - 50] | 30 |
| reg_alpha | [1.0 - 1.8] | 1.20 |
| reg_lambda | [1.0 - 1.8] | 1.20 |
| min_split_gain | [0.1 - 0.8] | 0.30 |
| subsample | [0.3 - 0.9] | 0.80 |
| objective | | regression |
| metirc | | rmse |
| learning_rate | | 0.1 |
| num_threads | | 2 |

- **Clasificación:** Se utilizaron tres modelos de clasificación para replicar la solución propuesta por el tercer lugar de la competencia. La variable a predecir de estos modelos toma el valor de 1 si la publicación es la comprada por la sesión y 0 en caso contrario. Una vez obtenidas las probabilidades para cada uno de los pares sesión-recomendación, se suman los resultados de los tres modelos para construir un score final, al cual nos referiremos como **Ensamble de clasificación**.

  Se utilizan los siguientes hiperparámetros:

*Tabla 12. Mercado Libre - hiperparámetros de modelos de clasificación*

| Hiperparámetro | Grilla | LightGBM | XGBoost |
|---|---|---|---|
| n_estimators | [200 - 1100] | 300 | 900 |
| colsample_bytree | [0.7 - 0.9] | 0.80 | 0.75 |
| num_leaves | [10 - 55] | 45 | 10 |
| max_depth | [10 - 55] | 15 | 10 |
| subsample | [0.7 - 0.9] | 0.82 | 0.86 |
| subsample_Freq | [0.7 - 0.9] | 35 | 35 |
| min_split_gain | [0.1 - 0.9] | 0.70 | 0.30 |
| reg_alpha | [1.1 - 2.0] | 1.33 | 1.50 |
| reg_lambda | [1.0 - 1.9] | 1.75 | 1.75 |
| metric | | auc | auc |

*Tabla 13. Mercado Libre - hiperparámetros de modelo Catboost*

| Hiperparámetro | Grilla | CatBoost |
|---|---|---|
| learning_rate | [0.03, 0.1] | 0.05 |
| iterations | 10 -150 | 90 |
| depth | [15,10,20] | 20 |
| l2_leaf_reg | [1, 3, 5, 7, 9] | 7 |
| metric | | auc |

*Modelo pairwise*

- **LambdaMart:** Se utiliza el módulo de ranking de *LightGBM*. El modelo recibe como input la misma base que los modelos *pointwise* (un registro por cada par sesión-recomendación). Sin embargo, difiere en que a éste se le indica cuántas recomendaciones corresponden a cada sesión. De esta manera, se puede entrenar evaluando pares de recomendaciones de la misma sesión. La variable a predecir es la misma utilizada en el modelo de regresión lineal.

  Se utilizan los siguientes hiperparámetros:

*Tabla 14. Mercado Libre - hiperparámetros de modelo LamdaMart*

| Hiperparámetro | Grilla | Óptimo |
|---|---|---|
| n_estimators | [200, 1100] | 500 |
| max_bin | [10 -150] | 96 |
| num_leaves | [10-100] | 90 |
| max_depth | [10-50] | 20 |
| min_data_in_leaf | [50-100] | 40 |
| feature_fraction | [0.5 - 0.8] | 0.6242 |
| subsample | [0.5 - 0.8] | 0.7212 |
| learning_rate | [0.005 - 0.1] | 0.083 |
| reg_alpha | [0 - 1] | 0.7142 |
| reg_lambda | [0 - 1] | 0.1428 |

- **Ensamble de clasificación más LambdaMart:** Como segundo ensamble se construye una combinación entre el ensamble de clasificación mencionado anteriormente y el modelo de *LambdaMART*. Para obtener estas predicciones, simplemente se suman las probabilidades obtenidas en los modelos.

*Modelo Listwise*

- **ListNet:** Se utilizó el módulo de *Tensor Flow Ranking* para construir el algoritmo de *ListNet* mencionado anteriormente.

*Tabla 15. Mercado Libre - hiperparámetros de modelo ListNet*

| Hiperparámetro | Valor |
|---|---|
| list_size | 50 |
| batch_size | 32 |
| hidden_layers_dims | [256,128,64] |
| num_train_steps | 100.000 |
| dropout_rate | 0.5 |
| learning_rate | 0.01 |
| loss_function | approx_ndcg_loss |

## 4.4 Métricas de evaluación

Dado que el objetivo de las tres aplicaciones propuestas es ordenar un conjunto de recomendaciones, no sería ideal utilizar métricas de evaluación de problemas binarios o continuos, como AUC, accuracy o RMSE. La métrica más utilizada para evaluar modelos de ranking es *Normalized Discounted Cumulative Gain (NDCG).* Ésta permite obtener un score que indica qué tan bueno es el modelo en ordenar las recomendaciones de mayor a menor relevancia. A continuación se explica cómo se calcula.

El primer paso para entender *NDCG* es entender qué es *Cumulative Gain* (CG). CG hace alusión a la relevancia ganada por cada documento en las recomendaciones. Es decir, si se tiene una lista de 5 recomendaciones ordenadas por las probabilidades obtenidas del modelo, la ganancia acumulada es la suma de la relevancia de las recomendaciones.

$$Ra = [2, 4, 4, 1, 1]$$
$$Rb = [4, 2, 1, 1, 1]$$
$$CGa = 2 + 4 + 4 + 1 + 1 = 12$$
$$CGb = 4 + 4 + 2 + 1 + 1 = 12$$

Como se puede observar, CG por sí solo no considera la posición de los elementos dentro de la lista, por lo tanto no es una buena métrica de ordenamiento. Para corregir esto, se introduce la siguiente ecuación que considera en el denominador la posición de la recomendación. De esta manera, documentos con alta relevancia en las primeras posiciones generan muchos puntos, pero un documento con alta relevancia colocado al final de la lista será penalizado.

$$DCG = \sum_{i=1}^{n} \frac{relevance_i}{log_2(i+1)}$$

$$DCGa = \frac{2}{log_2(1+1)} + \frac{1}{log_2(2+1)} + \frac{4}{log_2(3+1)} + \frac{4}{log_2(4+1)} + \frac{1}{log_2(5+1)} = 6.73$$

$$DCGb = \frac{4}{log_2(1+1)} + \frac{4}{log_2(2+1)} + \frac{1}{log_2(3+1)} + \frac{2}{log_2(4+1)} + \frac{1}{log_2(5+1)} = 8.27$$

Como se puede observar en el ejemplo anterior, esta métrica es capaz de evaluar qué ordenamiento es mejor. Sin embargo, no es un número estándar que se pueda utilizar para comparar diferentes soluciones. Por ejemplo, si una lista en lugar de tener relevancias entre 4 y 1, utilizara {12,1,0}, no sería posible comparar el DCG de ambos. Es por esto que se normaliza el DCG dividiéndolo por el DCG del ordenamiento óptimo. De esta manera se obtienen resultados entre 0 y 1 y es posible hacer comparaciones.

$$NDCG = \frac{DCG}{iDCG}$$

Para el ejemplo anterior el ordenamiento óptimo sería $[4, 4, 2, 1, 1]$ con lo que se obtiene $iDCG$ de 8.34, por tanto $NDCGa$ = 0.81 y $NDCGb$ = 0.99.

El $iDCG$ depende de la relevancia de los documentos recomendados. Por ejemplo, un *query* con recomendaciones con relevancias entre 3 y 1 y otro con relevancias entre 4 y 3, tendrán un denominador diferente. Por lo tanto, esta métrica estará evaluando únicamente qué tan bueno es el modelo en ordenar las opciones disponibles. Tiene sentido aplicar esta lógica en bases como la de Microsoft y Yahoo, ya que en estas aplicaciones no se evalúa qué tan bueno es el modelo en recomendar, sino qué tan bueno es en ordenar las opciones disponibles. Por otro lado, para adaptar este problema a uno como el de Mercado Libre, en el que el objetivo es que el ítem comprado se encuentre siempre en una posición alta dentro de las recomendaciones, tiene más sentido mantener el $iDCG$ constante para todas las sesiones. En este caso una recomendación ideal de 10 items seria: $[12, 1, 1, 1, 1, 1, 1, 1, 1, 1]$ , lo que significa un $iDCG$ igual a 15.54. Utilizando este denominador, se calcula el NDCG de cada historial de navegación, los cuales se promedian para obtener una métrica final.

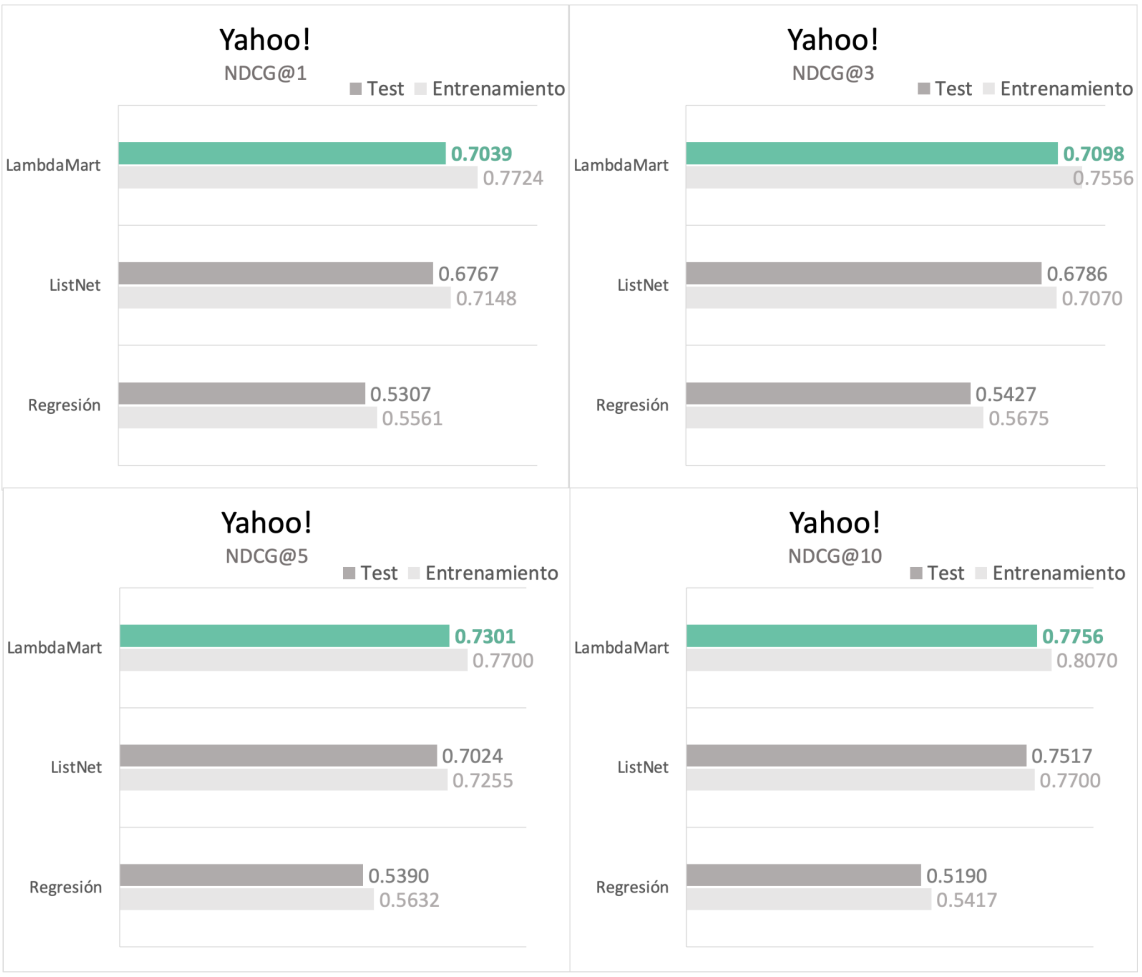
El NDCG se puede calcular en diferentes posiciones de la lista de recomendación. Por ejemplo, $NDCG@1$evalúa únicamente el primer ítem; mientras que $NDCG@5$ evalúa los primeros 5. Como criterio para elegir el mejor modelo se evaluará $NDCG@10$en el dataset de test.

# 5. Resultados

Todos los modelos fueron evaluados con NDCG de 1 hasta 10. A continuación se muestran los resultados obtenidos en cada una de las aplicaciones en estudio.

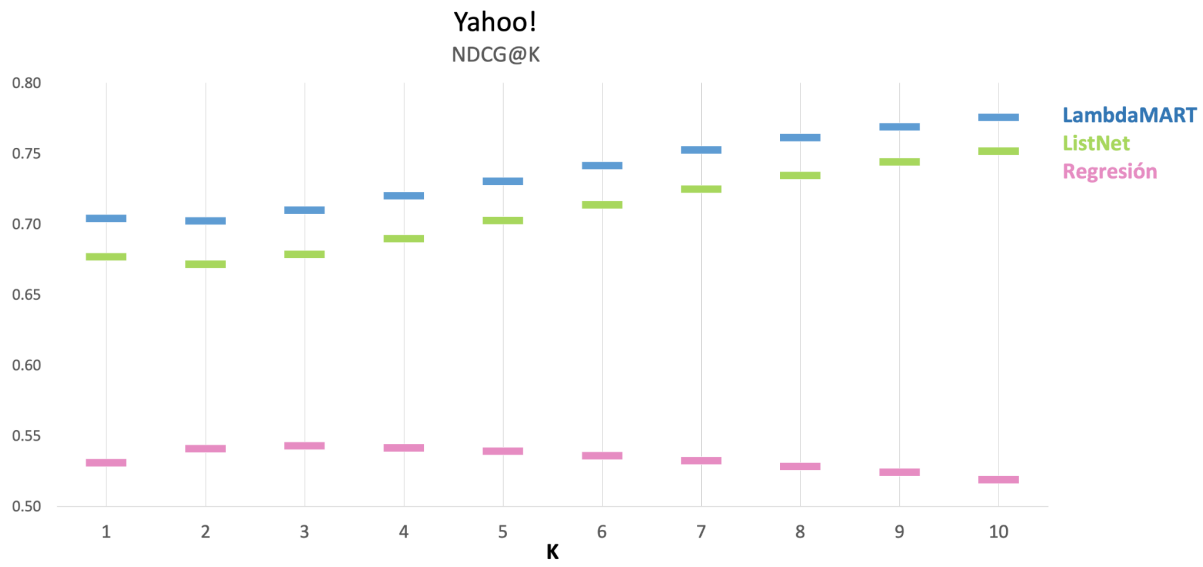
En el siguiente gráfico se puede observar el NDCG@1, 3, 5 y 10 de los modelos de *LambdaMart*, *ListNet* y Regresión en la base de entrenamiento y test de Yahoo. El modelo que se desempeña mejor es el de *LambdaMart*, seguido por *ListNet*; siendo el modelo de Regresión el que obtiene el peor resultado. Sin embargo, cabe destacar que la diferencia entre los primeros dos modelos es muy chica. Por ejemplo, en NDCG@10, *LambdaMart* está por delante de *ListNet* únicamente por 0.0239, lo que representa que este último modelo tiene un performance de 3% menor que *LambdaMart*. Por otro lado, el modelo de regresión es 33% más bajo.

*Gráfico 9. Yahoo - NDCG 1,3,5,10*



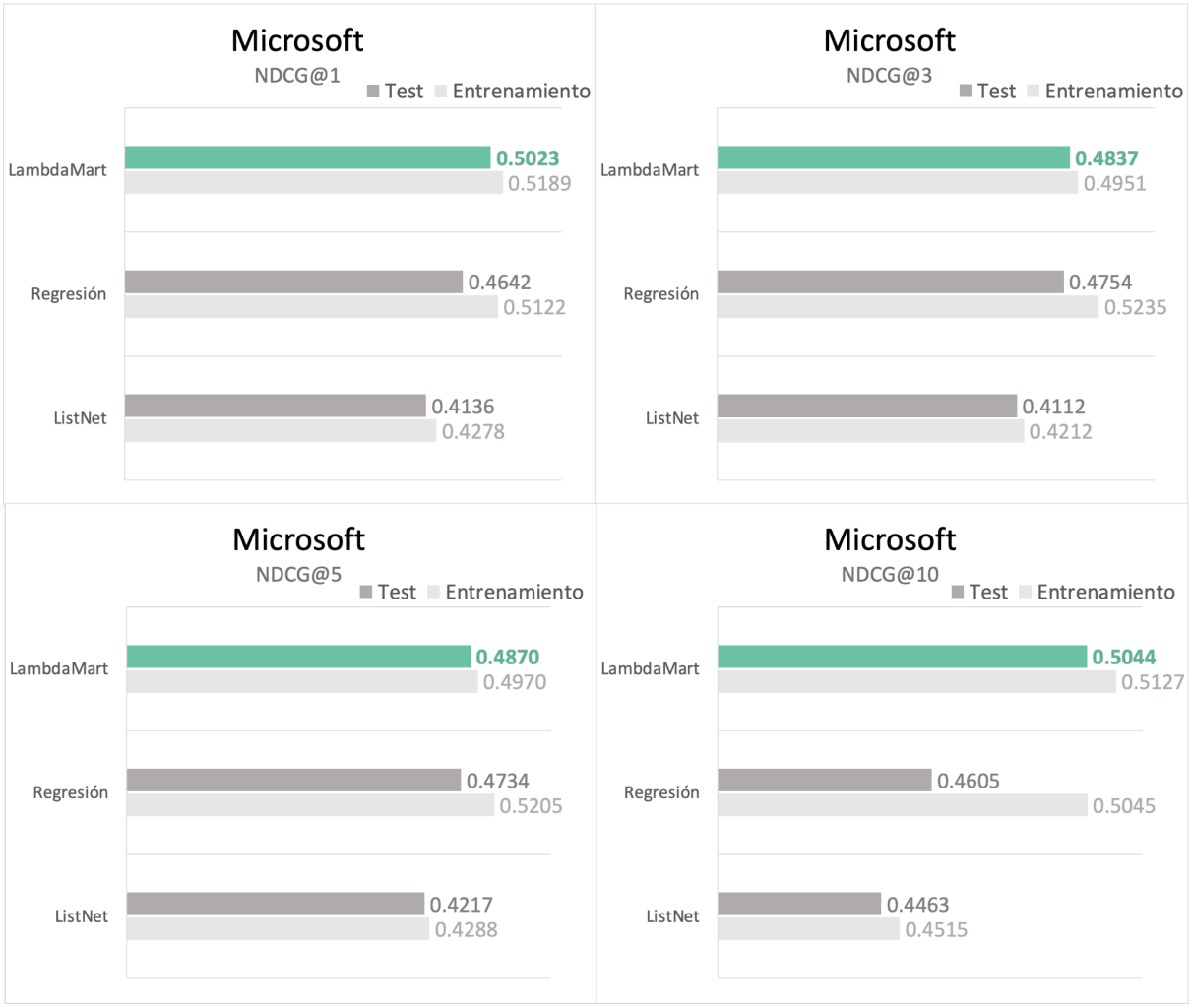
Al observar los resultados en diferentes niveles de *NDCG*, se obtiene la misma conclusión respecto al desempeño de los modelos. Sin embargo, el Gráfico 10 muestra que en el caso de *LambdaMart* y *ListNet*, el *NDCG* aumenta a medida que aumenta el tamaño de la lista ($k$). Lo contrario sucede con Regresión, la métrica comienza a decrecer a partir de $k = 4$

*Gráfico 10. Yahoo - Tendencia NDCG@K*



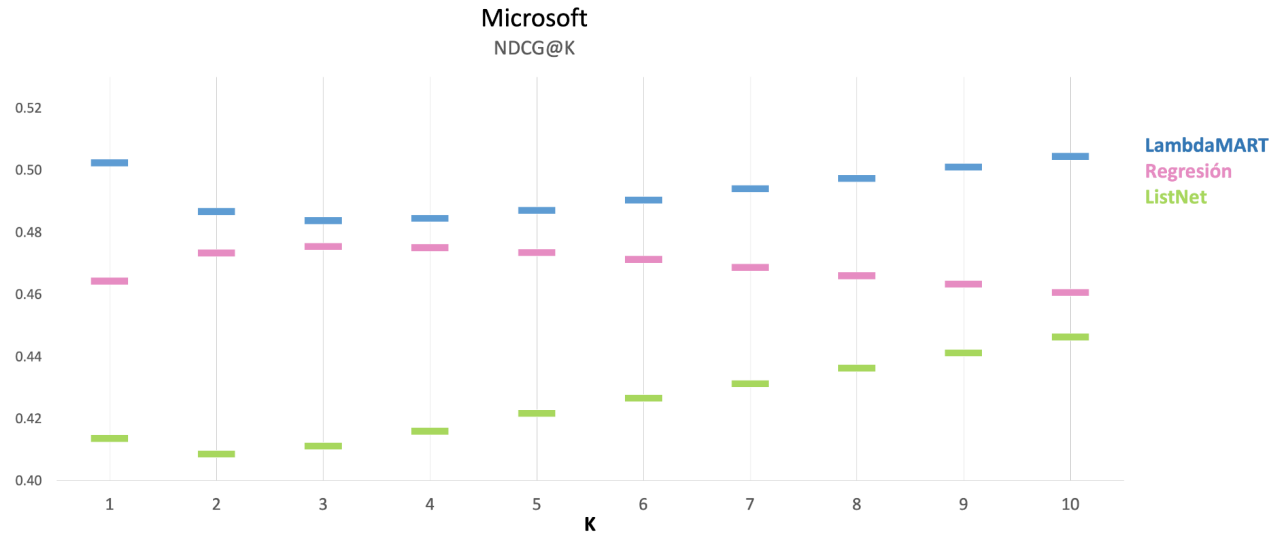
Un comportamiento similar se observa en la base de Microsoft. *LambdaMart* sigue siendo el mejor modelo en todos los niveles de $k$. En algunos niveles, el modelo de Regresión se desempeña mejor que *ListNet*. En el Gráfico 11 se resume los resultados de los modelos en las bases de entrenamiento y test de Microsoft. El valor mostrado es un promedio simple de los 5 *folds*. Como se puede ver, LambdaMart obtiene el mejor score en todos los niveles de NDCG.

*Gráfico 11. Microsoft - NDCG 1,3,5 y 10*



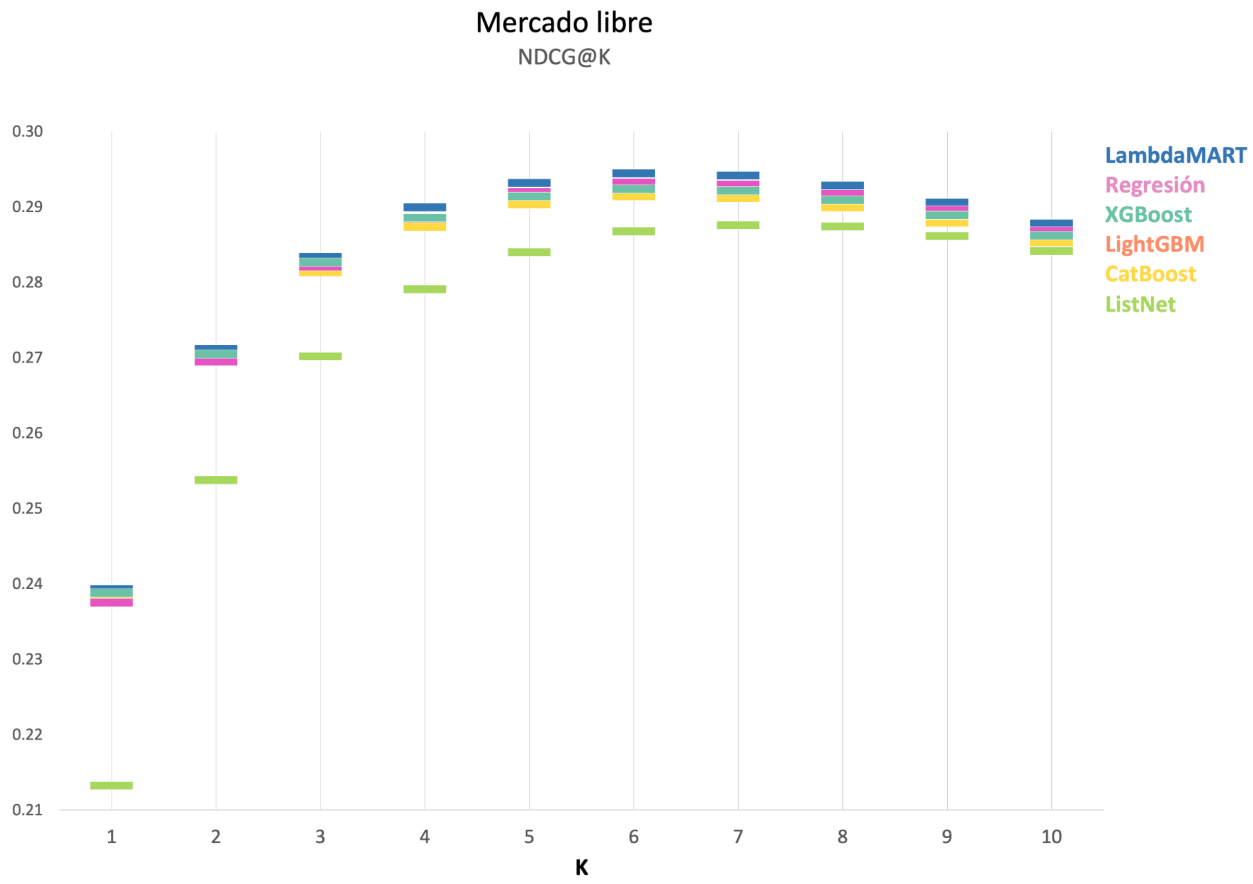
Al analizar los resultados en más niveles de $k$, se observa que *ListNet* es el peor modelo en todos los niveles. Al igual que en la base de Yahoo, el *NDCG* es creciente respecto a $k$ para *LambdaMart* y *ListNet,* y decreciente para Regresión. Cabe destacar que, al contrario del resultado obtenido en Yahoo, en esta base es más clara la ventaja de *LambdaMart* con respecto a los otros modelos. En *NDCG@10,* por ejemplo, el desempeño de *ListNet* es 11.5% menor que *LambdaMart*, mientras que Regresion es 8.6% menor.

*Gráfico 12. Microsoft - Tendencia NDCG@K*



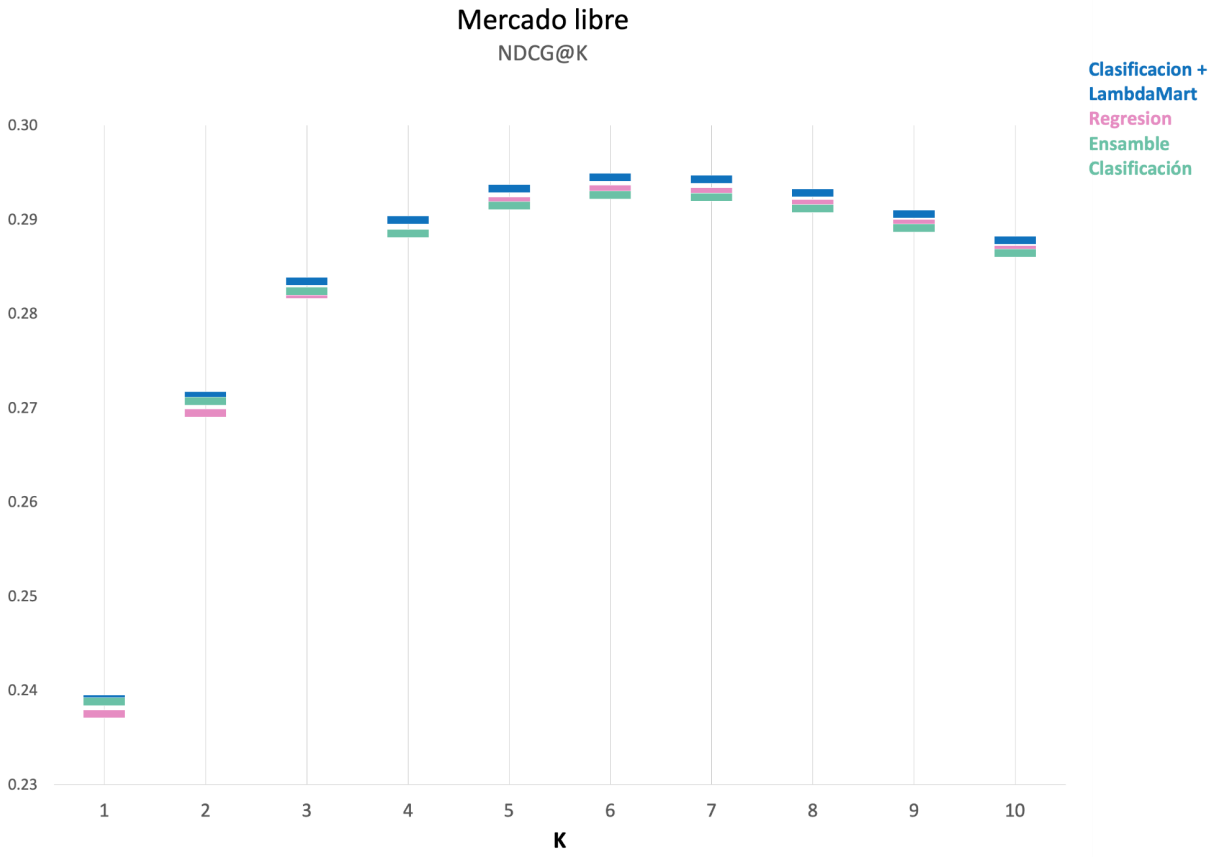
Los resultados obtenidos en la base de Mercado Libre presentan tendencias diferentes. En el Gráfico 13 se puede observar el *NDCG* en la base de test de cada modelo en diferentes niveles de $k$. Como se puede ver, a excepción de *ListNet*, los modelos tienen un desempeño muy similar, especialmente entre $k = 1$ $y$ $k = 3$. A partir de $k = 4$ se puede observar que la diferencia entre los modelos se va haciendo más notable. *LambdaMart* se mantienen siempre a la delantera, pero es interesante notar, que entre $k = 1$ y $k = 4$ los modelos de clasificación tienen un mejor desempeño que el modelo de Regresión. Sin embargo, en los siguientes niveles de $k$ el modelo de Regresión se mantiene en segundo lugar, siempre muy cerca del modelo de *LambdaMart*.

*Gráfico 13. Mercado Libre - Tendencia NDCG@K en test*



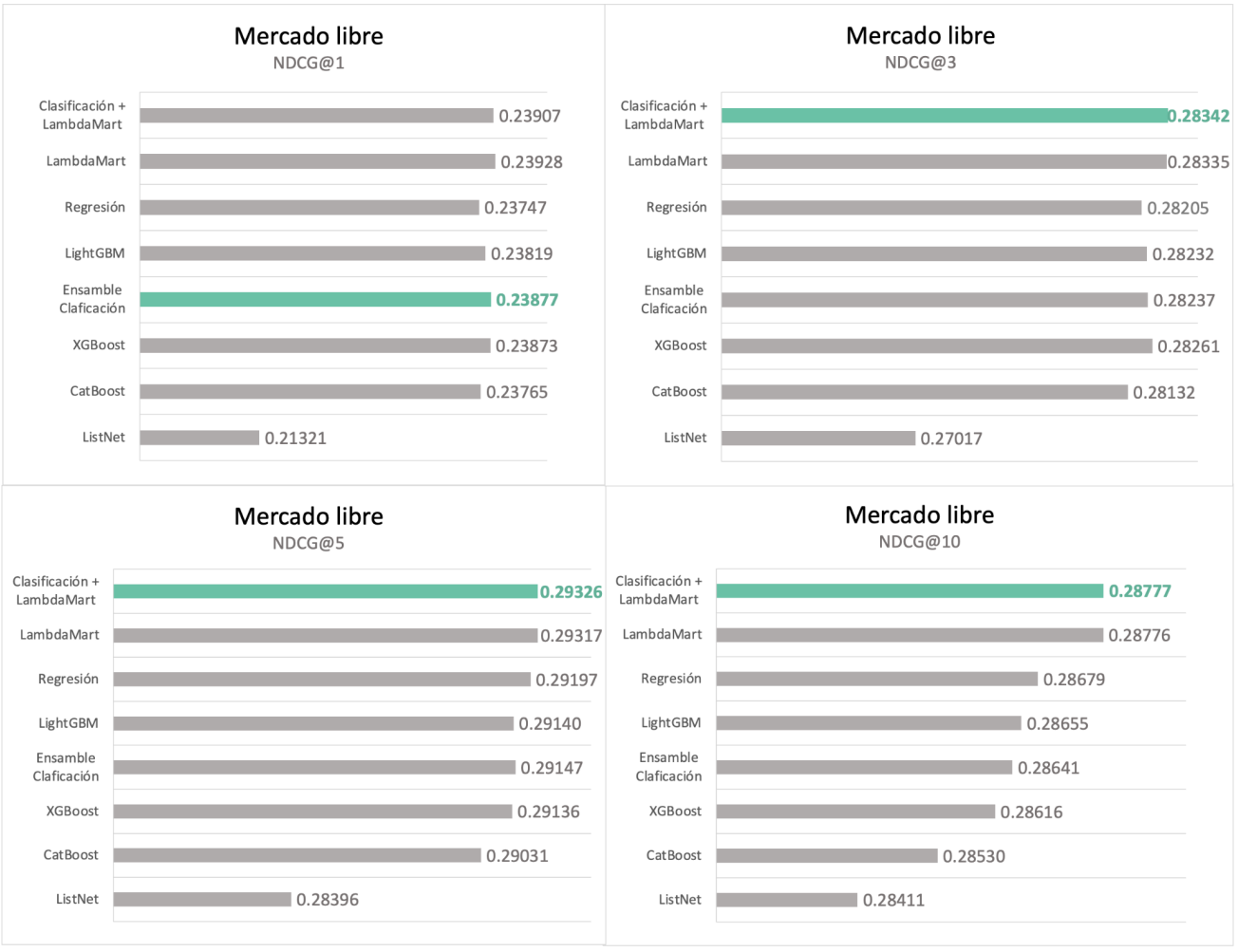
A continuación se presenta el mismo gráfico, con únicamente los tres modelos que con mejor desempeño en *NDCG@10*, incluyendo el ensamble de clasificación (*LightGBM + XGBoost + Catboost*) y el ensamble de clasificación más *LambdaMart*. Como se puede observar, este último es el mejor modelo a partir de $k = 2$, seguido por el modelo de Regresión y el ensamble de clasificación en tercer lugar. Este último es el que presenta mejores resultados en *NDCG@1*.

*Gráfico 14. Mercado Libre - Mejores modelos tendencia NDCG@K en test*



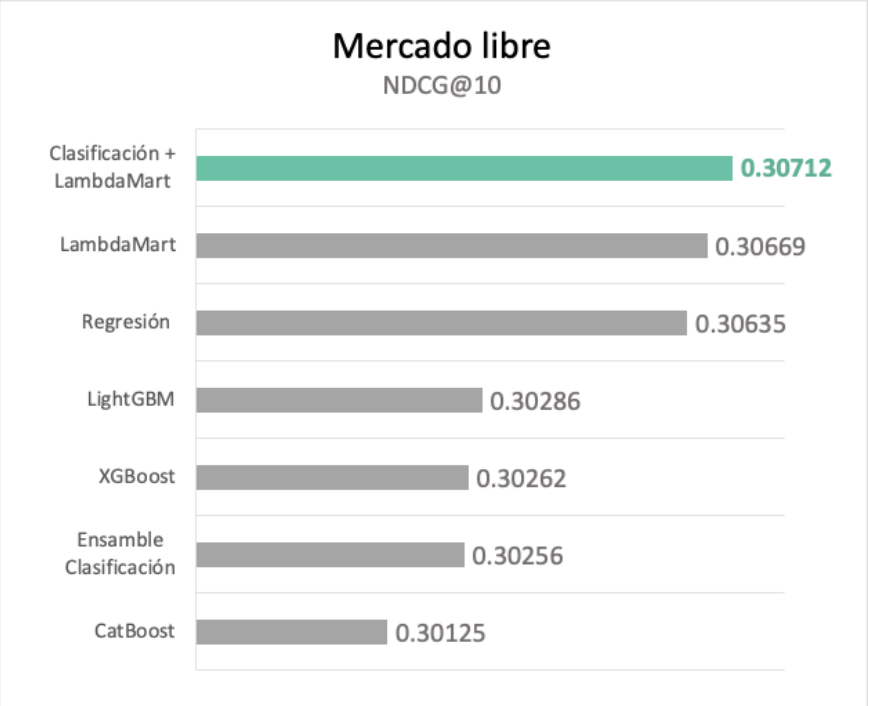
En el Gráfico 15 se presenta un resumen de los resultados en *NDCG@1, 3, 5* y *10,* en la base de test. Se puede apreciar claramente que en *NDCG@1* el mejor modelo es el ensamble de clasificación. Pero al considerar únicamente los modelos individuales, *LambdaMart* es el que obtiene mejor resultado. Por otro lado, en *NDCG@3, 5* y *10*, el mejor modelo es el ensamble de clasificación más *LambdaMart*, seguido por el modelo individual de *LambdaMart* y Regresión. Cabe destacar que la diferencia entre los modelos no es tan significativa como en las bases de Yahoo y Microsoft. Por ejemplo, comparando el mejor modelo individual (*LambdaMart*) en *NDCG@10*, Regresión es solo 0.34% más bajo, mientras que el mejor modelo de clasificación, *LightGBM,* es únicamente 0.42% menor. Los resultados en la base de entrenamiento se pueden observar en el Anexo 1.

*Gráfico 15. Mercado Libre - NDCG 1, 3, 5 y 10 en test*



Con el fin de replicar exactamente las condiciones de la competencia, los resultados de los modelos fueron enviados a Mercado Libre, quienes evaluaron el *NDCG@10* en la base del leaderboard público (70% de los datos de test). Los resultados se presentan a continuación.

*Gráfico 16. Mercado Libre - NDCG 10 en leaderboard público*




Dados estos resultados, tanto el ensamble de Clasificación + *LambdaMart*, como el modelo individual de *LambdaMart*, se hubieran ubicado en la cuarta posición del *leaderboard* público[23]. El modelo de *LambdaMart* difiere únicamente en 0.00673 puntos del primer lugar, mientras que el ensamble de Clasificación + *LambdaMart* difiere en 0.00630.

[23] https://ml-challenge.mercadolibre.com/leaderboard

# 6. Discusión

Los resultados anteriores permiten confirmar la hipótesis de que los modelos de ranking ofrecen una solución competitiva para resolver problemas de recomendación y de IR. Tanto en las bases de Yahoo y Microsoft, como en la base de Mercado Libre, los modelos de ranking con algoritmos *Pairwise* fueron los que obtuvieron el mejor resultado, incluso superando otros algoritmos como *XGBoost* que se conocen por ser muy competitivos.

Es sorprendente el mal desempeño que tuvo *ListNet*, especialmente en la base de Mercado Libre. Sin embargo, además de que este comportamiento se replica en Microsoft y Yahoo, también hay otros ejemplos en la literatura, donde la performance de *ListNet* es mucho más baja que *LambdaMART*. Por ejemplo, en un estudio realizado por Shuo Sun de Johns Hopkins University, sobre la base Istella Letor[24]. El *NDCG@10* obtenido por *ListNet* es 0.6317, mientras que el de *LambdaMart* es 0.6574 (Sun & Duh, 2020). *ListNet* tiene principalmente un mal desempeño en el dataset de Mercado Libre, un factor que puede estar afectando estos resultados, es que al contrario de Yahoo y Microsoft, las variables de esta base no fueron normalizadas.

Otro resultado inesperado fue el buen desempeño que obtuvo el modelo de regresión en la base de Mercado Libre. Sin embargo, esto tiene sentido ya que la variable objetivo tiene tres niveles claramente distanciados entre sí (12, 1, 0), permitiéndole al modelo diferenciar las publicaciones relevantes. Esto no pasa en las bases de Microsoft y Yahoo, porque la variable objetivo toma valores entre 0 y 4. Por tanto, el modelo de regresión puede predecir para un documento de relevancia 4, un valor más cercano a 3 y para uno de relevancia 3 un valor más cercano a 4. De esta forma, al ordenar las predicciones, los documentos quedarán invertidos, obteniendo un mal desempeño en NDCG.

[24] http://quickrank.isti.cnr.it/istella-dataset/

## 6.1 Limitaciones y posibles mejoras

Un gran desafío al evaluar modelos de ranking fue que estas técnicas no se encuentran tan bien implementadas como otros algoritmos de aprendizaje supervisado. Las únicas librerías de Python que tienen un módulo de ranking son *LightGBM* y recientemente *XGBoost*, ambas ofrecen únicamente el algoritmo de *LambdaMART*. La otra opción disponible es el módulo de ranking de Tensor Flow, que permite entrenar y evaluar modelos *listwise*. Sin embargo, esta opción requiere mayor esfuerzo de programación, ya que se debe construir la red neuronal de cero.

Es debido a lo anterior que en este informe se evaluaron únicamente algoritmos de ranking que se encuentran implementados en Python. Sin embargo, existen otros en lenguajes como C++, que han demostrado ser muy competitivos y que se utilizan ampliamente en aplicaciones de Information Retrieval. Por ejemplo, con la librería RankLib de C++ se pueden implementar otros modelos *pairwise* y *listwise* como *RankNet, RankBoost y AdaRank* (Modry, 2014). Una posible mejora en este trabajo sería evaluar otros modelos de ranking e identificar si algunos de éstos tienen un mejor desempeño.

En cuanto al mal desempeño de *ListNet*, una futura mejora es implementar la solución propuesta por Zhen Qin de Google Research en el paper *"Are neural rankers still outperformed by gradient boosted decision trees?"* (Qin et al., 2021). En este estudio, los autores analizan las debilidades de los modelos de redes neuronales en problemas de ranking. Proponen tres pasos para mejorar los resultados: primero normalizar las variables, segundo aumentar los datos con distribución gaussiana y por último, utilizar la técnica *Multi Head Self Attention (MHSA)*[25] para la arquitectura de la red.

Por último, otro desafío en la implementación de *ListNet*, fue la capacidad computacional. Las redes neuronales consumen mucha memoria, especialmente si el dataset es muy grande. Por esta razón, estos modelos se entrenaron con los hiperparámetros utilizados en la literatura. Una posible mejora sería entrenar estos modelos haciendo optimización de hiperparámetros y evaluar si se obtienen mejores resultados.

[25]Multi Head Self Attention: https://www.mql5.com/es/articles/8909#para2

## 6.2 Aplicaciones prácticas

El objetivo principal de este estudio era evaluar si los modelos de ranking pueden ser aplicados en diferentes problemas de negocio, más allá de su aplicación principal en Information Retrieval. Los resultados obtenidos con los datos de Mercado Libre indican que estos modelos se podrían aplicar a diferentes problemas que normalmente son abordados con modelos tradicionales. Parte de los hallazgos de este trabajo es que en ciertos tipos de problemas, en el cual el objetivo final es obtener un listado de ítems, los modelos de ranking son una opción viable y competitiva.

Como ejemplo de lo anterior, estos modelos se pueden utilizar en recomendación de productos en e-commerce, similar al ejemplo analizado en este informe. En este caso, el hecho de tener un algoritmo que predice y ordena mejor las recomendaciones, tiene un impacto positivo en la experiencia del usuario. Un mejor modelo significa que el usuario obtiene recomendaciones más personalizadas y genera una mejor experiencia de compra; lo que se traduce a mayores tasas de conversión y retención, y por lo tanto, mayores ganancias. Otra aplicación que genera un impacto similar es ordenar los resultados de búsqueda en un marketplace o cualquier sitio de e-commerce. Esto sería muy similar al problema de Yahoo y Microsoft, con la diferencia de que los *urls* son publicaciones de productos.

Estos algoritmos también podrían utilizarse en estrategias de marketing para seleccionar micro-influencers en redes sociales. En este caso, se debería contar con datos de publicaciones de los influencers y cómo éstas se relacionan con la marca que se desea promocionar. Con esta información se puede entrenar un modelo de ranking que seleccione a aquellas personas que mejor cumplen con los objetivos de la marca.

Los modelos de ranking también pueden ser aplicados en finanzas, para seleccionar el portfolio que genere mejor rentabilidad.

En conclusión, este trabajo servirá como referencia de que los modelos de ranking pueden generar, en determinados problemas relevantes de la industria, mejores resultados que modelos tradicionales. Esta ventaja se puede aprovechar en diferentes áreas para incrementar métricas de negocio, como retorno de inversión, retención y captación de clientes.

# 7. Anexos

## 7.1 Anexo 1: Resultados en base de entrenamiento de Mercado Libre

En verde se resalta el modelo con mejor desempeño y en azul el resultado más bajo. La última línea indica el cambio porcentual entre el mayor y el menor.

| | Entrenamiento | | | |
|---|---|---|---|---|
| | ndcg@1 | ndcg@3 | ndcg@5 | ndcg@10 |
| **Clasificación + LambdaMart** | 0.241217 | 0.284913 | 0.294529 | 0.290710 |
| **LambdaMart** | **0.241509** | **0.285111** | **0.294638** | **0.290811** |
| **Regresión** | 0.238713 | 0.282917 | 0.292739 | 0.289365 |
| **Ensamble Clasificación** | 0.240310 | 0.283471 | 0.292344 | 0.288856 |
| **LightGBM** | 0.241171 | 0.284326 | 0.293182 | 0.289780 |
| **CatBoost** | 0.238723 | 0.281775 | 0.290535 | 0.287207 |
| **XGBoost** | 0.239586 | 0.282930 | 0.292017 | 0.288417 |
| **ListNet** | **0.228638** | **0.276463** | **0.288048** | **0.284880** |
| %Δ | **5.6%** | **3.1%** | **2.3%** | **2.1%** |

## 7. 2 Anexo 2: Resultados después de completar recomendaciones en Mercado Libre

| | Entrenamiento | | | |
|---|---|---|---|---|
| | ndcg@1 | ndcg@3 | ndcg@5 | ndcg@10 |
| **Clasificación + LambdaMart** | 0.243112 | 0.266424 | 0.281219 | 0.299288 |
| **LambdaMart** | **0.244591** | **0.267190** | **0.281935** | **0.300042** |
| **Ensamble Clasificación** | 0.243112 | 0.262104 | 0.277026 | 0.295015 |
| **XGBoost** | 0.239040 | 0.262439 | 0.277320 | 0.295332 |
| **Regresión** | 0.238178 | 0.261955 | 0.277313 | 0.296323 |
| **LightGBM** | 0.237312 | 0.261031 | 0.276024 | 0.294117 |
| **CatBoost** | 0.238723 | 0.261616 | 0.276511 | 0.294516 |
| **ListNet** | **0.232872** | **0.256253** | **0.273565** | **0.291405** |
| **%Δ** | **5.0%** | **4.3%** | **3.1%** | **3.0%** |

| | Testeo | | | |
|---|---|---|---|---|
| | ndcg@1 | ndcg@3 | ndcg@5 | ndcg@10 |
| **Clasificación + LambdaMart** | **0.240390** | 0.286674 | **0.301301** | **0.318900** |
| **LambdaMart** | 0.239822 | 0.286174 | 0.300926 | 0.318550 |
| **Regresión** | 0.240372 | **0.286726** | 0.300955 | 0.318217 |
| **Ensamble Clasificación** | 0.239940 | 0.286323 | 0.300656 | 0.318013 |
| **LightGBM** | 0.239357 | 0.285922 | 0.300171 | 0.317595 |
| **CatBoost** | 0.239410 | 0.285970 | 0.300300 | 0.317571 |
| **XGBoost** | 0.239361 | 0.285880 | 0.300150 | 0.317449 |
| **ListNet** | **0.228638** | **0.276892** | **0.292381** | **0.310168** |
| | | | | |
| %Δ | **5.1%** | **3.6%** | **3.1%** | **2.8%** |

# 8. Bibliografía


- Baeza-Yates, R. (1999, Julio). *Modern Information Retrieval*. Research Gate. https://www.researchgate.net/publication/2352627_Modern_Information_Retrieval
- Berry, M. W., Mohamed, A., & Yap, B. W. (2020). *Supervised and Unsupervised Learning for Data Science*. Springer. ISBN 978-3-030-22475-2
- Burges, C. J.C. (2010). *From RankNet to LambdaRank to LambdaMART: An Overview*. Microsoft Research. https://www.microsoft.com/en-us/research/uploads/prod/2016/02/MSR-TR-2010-82.pdf
- Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). *Learning to Rank: From Pairwise Approach to Listwise Approach*. Microsoft Research. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2007-40.pdf
- Chapelle, O., & Chang, Y. (2011). *Yahoo! Learning to Rank Challenge Overview*. https://www.ccs.neu.edu/home/vip/teach/IRcourse/6_ML/other_notes/chapelle11a.pdf
- Gan, T., Wang, S., Liu, M., Song, X., Yao, Y., & Nie, L. (2019). *Seeking Micro-influencers for Brand Promotion*. Association for Computing Machinary. https://dl.acm.org/doi/proceedings/10.1145/3343031
- Kanti, S., Sondhi, P., & Zhai, C. (2017, Agosto). *On Application of Learning to Rank for E-Commerce Search*. SIGIR. https://arxiv.org/pdf/1903.04263.pdf
- Lathia, N. (2017, Septiembre 26). *Learning to rank for flight itenerary search*. https://hackernoon.com/learning-to-rank-for-flight-itinerary-search-8594761eb867
- Lee, D. (2020, Mayo 1). *Effects of Recommender Systems in E-Commerce Vary by Product Attributes and Review Ratings*. Tepper School of Business, Carnegie Mellon

University. https://www.cmu.edu/tepper/news/stories/2020/may/e-commerce-recommenders-research-lee.html#:~:text=Recommender%20systems%20are%20used%20in,or%20review%20ratings%20influence%20the

- Liu, D. C., Rogers, S., Shiau, R., Kislyuk, D., Ma, K. C., Zhong, Z., Liu, J., & Jing, Y. (2017). *Related Pins at Pinterest: The Evolution of a Real-World Recommender System* [In Proceedings of the 26th International Conference on World Wide Web Companion]. https://arxiv.org/pdf/1702.07969.pdf
- Liu, T.-Y. (2009). Learning to Rank for Information Retrieval. In *Foundations and Trends in Information Retrieval*. The essence of knowledge. http://didawiki.di.unipi.it/lib/exe/fetch.php/magistraleinformatica/ir/ir13/1_-_learning_to_rank.pdf
- Modry, M. (2014, Mayo 12). *Learning to rank algorithms*. Czech Technical University in Prague. https://core.ac.uk/download/pdf/47178464.pdf
- Poh, D., Poh, B., Zohren, S., & Robertshttps://arxiv.org/abs/2012.07149, S. (2020, Diciembre 13). *Building Cross-Sectional Systematic Strategies By Learning to Rank*. Cornell University - Quantitative Finance. https://arxiv.org/abs/2012.07149
- Qin, Z., Yan, L., Zhuang, H., Tay, Y., Pasumarthi, R. K., Wang, X., Bendersky, M., & Najork, M. (2021). *Are Neural Rankeres Still Outperformed by Gradient Boosted Decision Trees?* Google Research.
- Shaw, B., Shea, J., Sinha, S., & Hogue, A. (n.d.). *Learning to Rank for spationtemporal search*. Broadway, New Yourk. https://www.metablake.com/foursquare/wsdm2013-final.pdf

- Shen, L., Sarkar, A., & Och, F. J. (2004). *Discriminative Reranking for Machine Translation*. Association for Computational Linguistics. https://www.aclweb.org/anthology/N04-1023.pdf
- Shokouhi, M. (2013, Agosto 1). *Learning to Personalize Query Auto-Completion*. Microsoft Research. https://www.microsoft.com/en-us/research/wp-content/uploads/2013/01/SIGIR2013-Shokouhi-PersonalizedQAC.pdf
- Silverstein, C., Henzinger, M., Marais, H., & Moricz, M. (1998, Octubre 26). *Analysis of a Very Large AltaVista Query Log*. https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-TN-1998-014.pdf
- Sonawane, S. (15, Octubre 2018). *How we use machine learning and natural language processing to empower search*. Wayfair tech blog. https://www.aboutwayfair.com/tech-innovation/how-we-use-machine-learning-and-natural-language-processing-to-empower-search
- Sun, S., & Duh, K. (2020, Mayo 8). *Modeling Document Interactions for Learning to Rank with Regularized Self-Attention*. Cornell University. https://arxiv.org/pdf/2005.03932.pdf
- Tromba, I., Gallagher, J., & Liszka, J. (2015). *Search at Slack*. Slack Engineering. https://slack.engineering/search-at-slack/
- Veiga, T. (2020, Diciembre 9). *MeLi 2020 Winner Solution*. Github. https://github.com/tobiasveiga/MeLi2020_WinnerSolution
- Yin, D., Hu, Y., Tang, J., & Daily Jr., T. (2016, August 16). *Ranking Relevance in Yahoo Search*. Relevance Science, Yahoo! http://www.kdd.org/kdd2016/papers/files/adf0361-yinA.pdf