Árboles de decisión adaptativos en políticas de admisión a caché

Francisco Tonin Monzón, Santiago Banchero, and Gabriel Tolosa

Departamento de Ciencias Básicas, Universidad Nacional de Luján, Argentina
{ftonin, sbanchero,tolosoft}@unlu.edu.ar

Abstract. Millones de consultas son procesadas diariamente por los motores de búsqueda web. En éstos la utilización de memoria caché es crucial para reducir el tiempo de respuesta y aumentar el rendimiento. En la literatura, diversos autores han propuesto la utilización de técnicas de aprendizaje automático para aumentar la eficiencia de la caché. Hasta el momento, los trabajos en el área consisten en la utilización de algoritmos con funcionamiento por lote para gestionar las políticas de diferentes niveles de caché. Estos algoritmos construyen modelos estáticos que reducen su efectividad ante cambios en la distribución de los datos de entrada. Por otro lado, la investigación en el área de minería de flujos de datos ha aportado nuevos algoritmos, técnicas y plataformas para hacer frente a ambientes con generación continua de datos, altas tasas de arribo y elevados niveles de fluctuación en éstas, características que a su vez definen a la resolución de consultas en motores de búsqueda web. Basado en esto, se propone la utilización de un árbol de decisión adaptativo para generar reglas que permiten predecir futuras apariciones de las consultas. El rendimiento del mismo es comparado contra un árbol de decisión estático clásico mostrando las ventajas de reconocer dinámicamente patrones que identifican consultas frecuentes.

1 Introducción

Desde el surgimiento de los primeros motores de búsqueda masivos, el crecimiento en cantidad de datos a procesar y usuarios no ha hecho más que incrementarse. Aproximadamente el 90% de ellos utilizan motores de búsqueda para obtener información [29] y tienen grandes expectativas respecto a la calidad de sus resultados y al tiempo en el que le son entregados [2]. La web indexable (contenido accesible por los motores de búsqueda) es cada vez más grande, más de 45 mil millones de páginas según [7], por lo que resolver una consulta requiere el procesamiento de un inmenso volumen de datos. Otra dificultad que incrementa la complejidad de las operaciones son las altas tasas de consultas que arriban de manera ininterrumpida provocando que cualquier actividad complementaria o de mantenimiento deba hacerse, en lo posible, en tiempo real. Estas últimas características asociadas a los

motores de búsqueda son referidas en la literatura como volumen y velocidad, dos de las dimensiones que caracterizan un problema de grandes datos (Big Data) [11, 27].

Por lo tanto, se requiere de técnicas sumamente optimizadas para gestionar altas tasas de consultas sobre un conjunto de datos en constante crecimiento. Esto ha generado avances algorítmicos y de estructuras de datos que permiten procesar eficientemente y responder las consultas con restricciones de tiempo (milisegundos). Una de las técnicas más utilizadas para abordar este problema es el uso de caches, que consiste básicamente en mantener en una memoria de rápido acceso *ciertos* elementos previamente utilizados. En el ámbito de un motor de búsqueda existen, básicamente, tres niveles de caché [21]. A nivel del nodo que recibe las consultas (*broker*) existe un caché de resultados el cual almacena la información de respuesta para n consultas. Por otro lado, a nivel de los nodos de búsqueda (*search nodes*) existen dos niveles de caché, de intersecciones y de listas de postings. Luego, las memorias caché pueden ser estáticas, en las cuales su contenido no cambia conforme se ejecuta el proceso o dinámicas, las cuales son gestionadas mediante alguna *política de reemplazo* (por ejemplo, LRU) la cual decide qué elemento desalojar para hacer lugar para uno nuevo. Existen también enfoques híbridos en los cuales una porción del caché es estático y la restante, dinámico [13]. En general, estas políticas se basaban en criterios de frecuencia, tiempo de aparición o costo [32].

Otra posibilidad, es complementar un caché dinámico con una *política de admisión*, la cual establece si el ítem actual es suficientemente *bueno* para ser incorporado (teniendo sentido el desalojo de otro). Estas políticas no han sido extensamente estudiadas en el ámbito de los motores de búsqueda por lo que existen oportunidades de investigación.

Recientemente, diversos autores [17, 20, 32] han propuesto, como alternativa a los criterios tradicionales, la utilización de técnicas de aprendizaje automático para decidir para la gestión de los elementos en la memoria caché. En el entorno de un motor de búsqueda la frecuencia de aparición de las consultas evoluciona, los usuarios expresan un aumento en su interés en consultas relacionadas con eventos recientes, con efecto visible en la frecuencia de aparición de las mismas [30], lo cual se puede modelar como un problema de *streaming*. Por este motivo, un modelo de predicción entrenado en modo por lotes (estático) es limitado en cuanto su performance de predicción en el tiempo y solo puede mantenerse ajustado a un periodo finito. Por este motivo, un modelo más real para una política de caché de un motor de búsqueda debe considerar la evolución en el tiempo, capturando las nuevas tendencias y olvidando aquellas antiguas que ya no están en vigencia.

Desde hace algunos años el interés en la investigación y desarrollo de software para tratamiento streaming se ha incrementado, y se han desarrollado algoritmos de aprendizaje automático para diversas problemáticas como clustering, clasificación y minería de secuencias [28]. Las etapas que tradicionalmente existen en los algoritmos,

entrenamiento y ejecución, se diluyen constituyéndose como un continuo de ejecución y aprendizaje [5].

Entre los diversos tipos de algoritmos adaptados para el tratamiento de streaming, los arboles de decisión han sido importante foco de investigación y desarrollo. La aptitud para poder aprender de flujos de datos conlleva cumplir con los siguientes restricciones: leer cada instancia como máximo una vez, usar un cantidad limitada de memoria, funcionar en tiempo limitado y contar con un modelo listo para predecir en cualquier momento [5]. Hoeffding Tree [12] es el primer árbol decisión con la capacidad de aprender a partir de flujos de datos. Este árbol decisión puede aprender eficazmente de datos si su distribución es estacionaria. Esta limitación se debe a que carece de la capacidad de detectar o adaptarse a cambios en el tiempo de las propiedades estadísticas asociadas a la variable objetivo (cambio de concepto). Para suplir esta falencia, Domingos, Hulten y otros [19] desarrollan una segunda versión que utiliza un mecanismo de ventanas deslizantes, para detectar y responder a cambios de concepto. Sin embargo, el tamaño de ventana óptimo es un parámetro difícil de determinar, ligado a la versatilidad del entorno en el que se desempeña el algoritmo.

Sobre esta base, se ha desarrollado Hoeffding Adaptive Tree (HAT) [4], algoritmo que resuelve el problema de encontrar el tamaño de ventana óptimo individualmente por cada nodo del árbol. Para ello, incluye instancias de detectores de cambio [3] que ajustan el tamaño de ventana en base al comportamiento de la distribución de los datos por cada nodo dependiendo de la rama y altura donde se encuentra. Las condiciones propias de HAT lo hacen una alternativa prometedora para capturar las características cambiantes que permiten predecir parámetros de las consultas (por ejemplo, su frecuencia), para aplicar a políticas de caché de un motor de búsqueda.

1.1 Objetivos

El objetivo principal de este trabajo es evaluar la performance de un árbol de decisión adaptativo (HAT) para aplicar al diseño de una política de admisión para un motor de búsqueda web que recibe (y procesa) consultas en modo *streaming*. De acuerdo a nuestro conocimiento, es la primera vez que se propone un árbol de decisión adaptativo para el problema abordado.

Se modela la admisión como un problema de clasificación binario y se intenta capturar los cambios de concepto en el tiempo, manteniendo un modelo siempre ajustado. Se evalúa al algoritmo Hoeffding Adaptive Tree utilizando un registro de consultas de un motor de búsqueda real [24], ampliamente utilizado en trabajos relacionados, comparando su rendimiento con el de un árbol de decisión tradicional con funcionamiento por lotes (C4.5) [25]. Finalmente, se integra el mecanismo de decisión como política de admisión y se evalúa la performance del caché de resultados en ambos casos.

El resto del trabajo se encuentra organizado de la siguiente manera: en la Sección 2 se introducen los trabajos relacionados con esta propuesta. A continuación (Sección 3), se introducen conceptos sobre árboles adaptativos. Luego, se propone una metodología en la Sección 4 que determina la parte experimental de la Sección 5. Finalmente, se presentan las conclusiones y los trabajos futuros propuestos (Sección 6).

2 Trabajos Relacionados

En la literatura se puede encontrar una vasta cantidad de trabajos que tienen como objetivo mejorar la performance de la caché de los motores de búsqueda Web [1, 2, 6, 17, 20, 23, 31]. Los acercamientos tratados por los mismos varían en nivel de jerarquía (caché de resultados [17, 20], listas de posteo o intersección de listas de posteo [32]) y tipos de políticas de caché: admisión o desalojo, incluso con *prefetching*, que pueden ser dinámicas o estáticas) [17]. Aún así, hay que destacar que existen muy pocos trabajos relacionados que abordan directamente el problema estudiado en este trabajo.

Los tres trabajos más cercanos a esta propuesta son el de Tolosa [32], Kucukyilmaz [20] y Zhou [34]. En este sentido, el primero de ellos aborda el problema de aumentar el rendimiento de la caché de intersecciones. Para esto, se propone una política de admisión que utiliza algoritmos de aprendizaje automático para decidir qué ítem (o intersecciones) deberían ser cacheados y cuáles no. A partir de estos modelos es posible prevenir intersecciones infrecuentes o incluso *singleton*[1] sean admitidas en caché. El trabajo modela la eficiencia del caché en términos del costo ahorrado en la resolución de consultas.

En el trabajo de Kucukyilmaz [20] se propone un enfoque de aprendizaje automático para mejorar el *hit rate* de un caché de resultados a partir de una gran cantidad de *features* extraídas de los registros de consultas de un motor de búsqueda. En este trabajo los autores modelan a partir de las características del un registro de consultas y el algoritmo *Gradient Boosted Decision Trees* (GBDT) [15], configurado con un total de 40 árboles y no más de 20 nodos. En el caso del almacenamiento en caché estático, el modelo de aprendizaje se usa para seleccionar las consultas que se admitirán en la caché. Mientras que en caché dinámico, el modelo de aprendizaje es utilizado para predecir la probabilidad de observar una consulta en un futuro cercano para facilitar la admisión y el desalojo.

El trabajo de Zhou [34] integra el caché de resultados de consultas, de *posting list* e intersecciones en memoria. Se enfoca principalmente en las políticas de caché de intersecciones, proponiendo una estrategia de selección y reemplazo que se basan en asociación de ítem frecuentes $top - N$ e incremental, respectivamente.

[1] Se denomina *singleton* a una consulta que aparece sólo una vez.

3 Árboles de Clasificación Adaptativos

El aprendizaje automático es un campo de las ciencias de la computación que aborda la construcción de programas que automáticamente mejoren con la experiencia [22], es decir, que reduzcan su tasa de error conforme a la experiencia acumulada. En este campo se distinguen tres tipos de problemas de aprendizaje: supervisado, no supervisado y aprendizaje por refuerzo [26]. Los árboles de decisión corresponden al aprendizaje supervisado y son ampliamente utilizados en problemas de clasificación. Estos algoritmos intentan, a partir de instancias vistas, generar hipótesis con las cuales hacer predicciones de futuras instancias.

Los algoritmos clásicos para construir árboles, como ID3, C4.5 y CART asumen que todas las instancias de entrenamiento se encuentran simultáneamente en memoria, lo que limita el número de ejemplos sobre los que pueden aprender [12]. Los procesos fundamentales que generan los flujos de datos pueden cambiar a través de los años, meses o incluso segundos drásticamente (cambio de concepto) [9] lo que produce que los modelos estáticos pierdan su efectividad.

El cambio de concepto en las distribuciones de datos a través del tiempo es uno de los principales problemas que deben resolver los algoritmos de aprendizaje automático. Aprender de datos en los que se producen cambios de concepto requiere de estrategias dirigidas a llevar a cabo las siguientes tareas:

- Detectar cuando ocurre un cambio de concepto
- Decidir qué instancias-ejemplo mantener y cuales olvidar
- Controlar y actualizar el modelo cuando se detecten cambios significativos

En aprendizaje sobre flujo de datos (*stream learning*), en general no es necesario computar estadísticas sobre todo el pasado, siendo suficiente con hacerlo sobre el pasado reciente [16]. Una de las formas mas clásicas y simples de mantener los ejemplos correspondientes a ese pasado es almacenar solo una ventana de instancias. En la literatura, esta técnica se conoce como ventanas deslizantes. El tamaño de la ventana influye en la rapidez con la que se puede detectar esos cambios. Debido a esto, utilizar un tamaño de ventaja fijo es ineficiente ya que los datos puede tener una combinación de cambios abruptos, graduales y largos periodos de estacionalidad. Tamaños de ventana pequeños son preferibles frente a cambios de concepto rápidos, mientras que ventanas de mayor dimensión permiten generalizar mejor en largos periodos sin cambios.

HAT [4] es una variante de Hoeffding Tree que utiliza ventanas deslizantes para mantener ajustado el modelo al concepto vigente, adaptando el tamaño de ventana dinámicamente. El tamaño de ventana óptimo es calculado individualmente para cada nodo, utilizando detectores de cambios y estimadores llamados ADWIN [3].

Este árbol de decisión adaptativo encapsula en cada nodo todos los cálculos estadísticos que permiten detectar cambios en la distribución de los datos. El algoritmo

decide individualmente los valores de cuantas instancias es necesario almacenar. De esta forma, cada instancia incrementa el tamaño de ventana en periodos sin cambios y lo reduce al detectarse uno. El algoritmo intenta encontrar el tamaño óptimo que tenga un balance entre tiempo de reacción y alta sensibilidad a pequeñas varianzas [3]. HAT implementa la segunda versión del algoritmo ADWIN, en la cual se introducen mejoras en cuanto a la eficiencia en el uso de recursos. La primera versión, controla exhaustivamente posibles divisiones de la ventana de instancias y almacena de forma explícita cada una. Esto resulta costoso para el computo y en el espacio de almacenamiento necesario. La segunda versión incorpora una variación de histograma exponencial [10] que le permite comprimir los datos con una cota superior asintótica de $O(logW)$ en uso de memoria y tiempo de procesamiento.

A diferencia de CVFDT, HAT crea árboles alternativos tan pronto como se detecta un cambio, sin la necesidad de esperar a tener un número previamente definido de instancias. De igual manera, apenas existe evidencia de un árbol alternativo alcanza mayor exactitud, se reemplaza el árbol actual. Es decir, los cambios abruptos aceleraran la creación de un árbol alternativo que reemplace al vigente.

4 Metodología

El modelo propuesto para la gestión de un caché de resultados de un motor de búsqueda se basa en realizar una evaluación previa de las consultas antes de enviarlas efectivamente a caché. Para ello, se implementa una política de admisión que *decide* cuál consulta debe ser almacenada en caché y cuál no. Esta decisión se modela como un problema de aprendizaje automático (Machine Learning) que se resuelve con un algoritmo de clasificación binaria. Entonces, para una consulta dada q, el clasificador determina la clase (frecuente/infrecuente) como:

$$C(q) = \begin{cases} 0 \text{ si } q \rightarrow Infrecuente \\ 1 \text{ si } q \rightarrow Frecuente \end{cases} \quad (1)$$

Luego, todas las q, tal que $C(q) = 1$ son enviadas a la memoria caché y gestionadas luego por la política de reemplazo. Este flujo se muestra en la Figura 1. En el ejemplo se puede notar que si bien llegan al caché cuatro consultas ($q_1 \dots q_4$) solo dos (q_1 y q_3) son admitidas en caché (y oportunamente reemplazadas).

Para la implementación de la política de admisión se utilizan, entonces, un árbol de decisión estándar (C4.5) y uno adaptativo (HAT). El algoritmo HAT, desarrollado específicamente para el aprendizaje a partir de flujos de datos continuos y potencialmente infinitos tiene la capacidad de alternar entrenamiento y evaluación indefinidamente. En cambio, C4.5 el modelo se entrena por única vez con necesariamente la totalidad de instancias en memoria. Ambas formas de aprendizaje poseen beneficios y desventajas. Por un lado, el aprendizaje continuo permite la actualización del modelo, aunque obliga a tomar decisiones con reducido número de

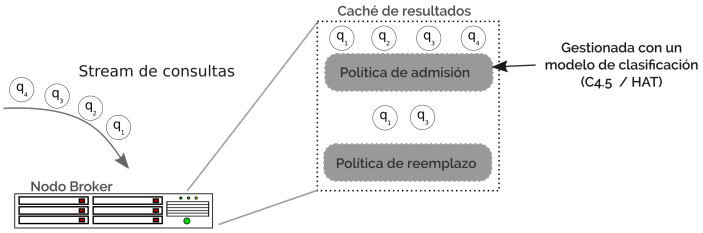

Fig. 1: Esquema de interacción entre la política de admisión y la de reemplazo

instancias. Por otro lado, el aprendizaje en lote permite recorrer iterativamente el conjunto de instancias de entrenamiento en busca de patrones que definen la clase objetivo. Sin embargo, el modelo estático construido por C4.5 pierde efectividad apenas se produce un cambio en las propiedades estadísticas de la variable objetivo, siendo necesario reconstruir el modelo para recuperar su exactitud de clasificación.

En ambos casos, se define un conjunto de *features* a evaluar para el entrenamiento del modelo y que pertenecen a la clase "frecuente" aquellas consultas con más de dos apariciones. El criterio usado para asignar la clase es altamente influyente en la evaluación. En un entorno real, la primera aparición de una consulta, resulta naturalmente en un fallo de caché (*cache miss*). En la literatura, se denomina a esta situación fallo obligatorio (*compulsory miss* [1]). Se intenta reproducir esta situación etiquetando una consulta efectivamente frecuente a partir de su segunda aparición.

Para la extracción de features se procesa el registro de consultas de un motor de búsqueda. En este caso, se utiliza el AOL Query Log [24] que comprende mas de 20M de consultas acumuladas en un periodo de tres meses. En la Tabla 1 se introducen las features extraídas.

Cantidad de caracteres
Número de términos
Presencia de URL
Presencia de errores de ortografía
Largo promedio de términos
Número de ranking
Hora de la consulta
Cantidad resultados seleccionados
Cantidad resultados seleccionados en la primera posición
Frecuencia {max/min/avg} de los término de la consulta por día
Frecuencia {max/min/avg} de los términos de la consulta por hora
Frecuencia {max/min/avg}de los término de la consulta por minuto
Frecuencia de la consulta por {dia/hora/minuto}

Tabla 1: Features extraídas del registro de consultas

Para las pruebas, se utilizan implementaciones desarrolladas por la universidad Waikato[2], disponibles en las librerías para minería de datos Weka [18] y Moa [5], ampliamente utilizadas.

4.1 Puesta a punto

En el aprendizaje de máquina, la optimización de hiperparámetros consiste en buscar el valor de los parámetros que frente a un conjunto independiente de datos genere un modelo que minimice una función de error predefinida [8]. Todas las consultas del registro fueron utilizadas con HAT para definir qué criterio de división y método para manejo de atributos numéricos brinda mejores resultados. Los resultados se presentan en las Figuras 2 y 3 respectivamente. De aquí, se selecciona "Índice de Gini" como criterio de división y "ExponentialHistogram" para el manejo de los atributos numéricos.

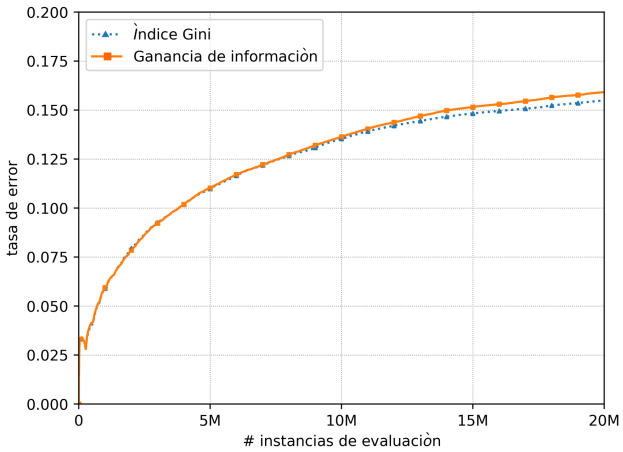

Fig. 2: Tasa de error de HAT variando criterios de división

[2] https://www.waikato.ac.nz/

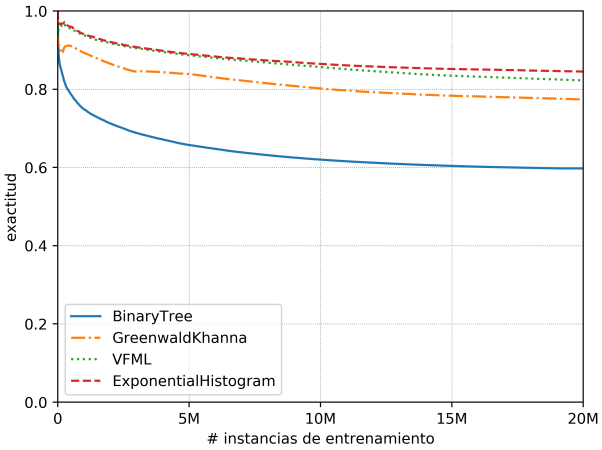

Fig. 3: Exactitud de HAT variando métodos de manejo de atributos numéricos

4.2 Métricas

En esta sección, se exponen las métricas utilizadas para la evaluación de la clasificación. La exactitud y la tasa de error son dos métricas complementarias utilizas para la puesta a punto de los modelos. La primera de ellas es la proporción de instancias correctamente clasificadas. Se define como:

$$\textbf{Exactitud} = \frac{vp + vn}{vp + vn + fp + fn} \tag{2}$$

Por otro lado, la tasa de error es la proporción de instancias erróneamente clasificadas, expresada como:

$$\textbf{Tasa de Error} = \frac{fp + fn}{vp + vn + fp + fn} \tag{3}$$

En algunas ocasiones, la exactitud obtenida por un clasificador por sí sola, resulta una métrica exigua para evaluar su performance [14]. La Sensibilidad es la proporción de instancias positivas correctamente clasificados como tal. Para esta tarea, la sensibilidad indica el porcentaje de consultas frecuentes correctamente clasificadas.

$$\textbf{Sensibilidad} = \frac{vp}{vp + fn} \tag{4}$$

Por otra parte, la especificidad es la proporción de instancias negativas que son correctamente clasificados como tal. La métrica es una indicador de la capacidad del clasificador para predecir consultas no frecuentes. Se define a la especificidad como:

$$\textbf{Especificidad} = \frac{vn}{vn + fp} \tag{5}$$

5 Experimentos y Resultados

5.1 Comparación del rendimiento de clasificación

Utilizando el registro de consultas preprocesado se entrena a C4.5 con 100K instancias (*C4.5-single*) y es evaluado con un conjunto disjunto al anterior. Para una comparación equitativa, se contrasta la performance de HAT a partir del mismo conjunto de evaluación con un precalentamiento de 200 instancias.

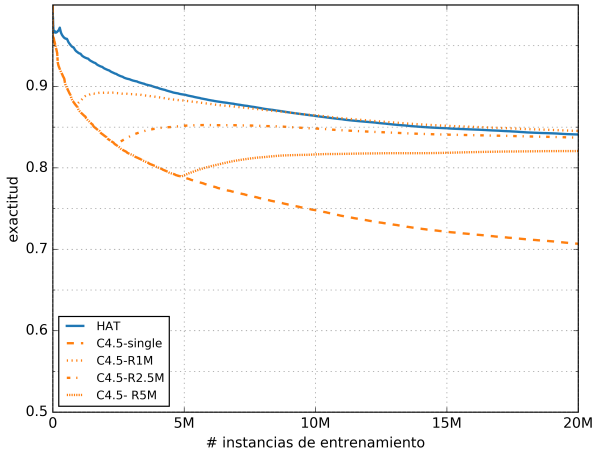

Fig. 4: Comparación de la exactitud de la clasificación entre C4.5 y HAT utilizando 20M de instancias del query log de AOL. *C4.5-single* corresponde al modelo entrenado solo al inicio mientras que *C4.5-R1M*, *C4.5-R2.5M* y *C4.5-R5M* corresponden al modelo reentrenado cada 1, 2.5 y 5 millones de consultas vistas.

A los efectos de ofrecer un análisis más robusto, se incluyen tres versiones del árbol C4.5 con reentrenamiento periódico. En primera instancia, se reentrena el

modelo por cada millón de consultas clasificadas (*C4.5-R1M*), lo que se toma como baseline. En un escenario mas realista se debe considerar que un motor de búsqueda comercial procesa un millón de consultas en un lapso de tiempo efímero. Por esta razón, se agregan dos escenarios más donde el árbol C4.5 es reentrenado cada 2.5 M y 5 M de consultas (*C4.5-R2.5M* y *C4.5-R5M*, respectivamente).

Como se exhibe en la Figura 4, ambos clasificadores mantienen su exactitud por encima de un 90% al clasificar las primeras 500K instancias, aunque HAT lo mantiene hasta las 3,5M primeras. Los modelos tienen una tendencia a deteriorar su performance, *C4.5-single* se ve más afectado conforme más instancias son clasificadas, deteriorándose hasta un 20% por debajo del valor inicial y finalmente, obteniendo una exactitud del 70% al cabo de clasificar 20M de instancias. Es decir, un 30% de las instancias son incorrectamente clasificadas por C4.5. Este porcentaje tiene significativas implicaciones en la tasa de aciertos de la caché, al ser este clasificador utilizado para administrar las políticas de admisión. Por otra parte, HAT al cabo de clasificar 20M reduce su exactitud en sólo un 10%, logrando aproximadamente un 84,5% de instancias correctamente clasificadas. Como señala Tolosa [32], al clasificar un elemento como falso negativo se podría estar excluyendo indeterminadamente de la caché a uno con aparición recurrente. Para C4.5 este error resulta de mayor gravedad, dado que un elemento con cierta combinación de valores sera siempre clasificado de la misma manera hasta un reconstrucción del modelo.

Comparando las variantes de C4.5 y HAT, se puede apreciar que la performance entre los algoritmos es similar cuando C4.5 es reentrenado en periodos breves (*C4.5-R1M*). Por otra parte, HAT supera aquellas variantes de C4.5 que son entrenadas más espaciadamente, evidenciando su capacidad de mantenerse ajustado al concepto vigente en el flujo de consultas.

		Predicción	
		Frecuente	Infrecuente
Verdad	Frecuente	0,1319 (VP)	0,2927 (FN)
	Infrecuente	0,0006 (FP)	0,5748 (VN)

Tabla 2: Tabla de contingencia de la clasificación de *C4.5-single*.

		Predicción	
		Frecuente	Infrecuente
Verdad	Frecuente	0,3072 (VP)	0,1157 (FN)
	Infrecuente	0,0434 (FP)	0,5337 (VN)

Tabla 3: Tabla de contingencia de la clasificación de HAT

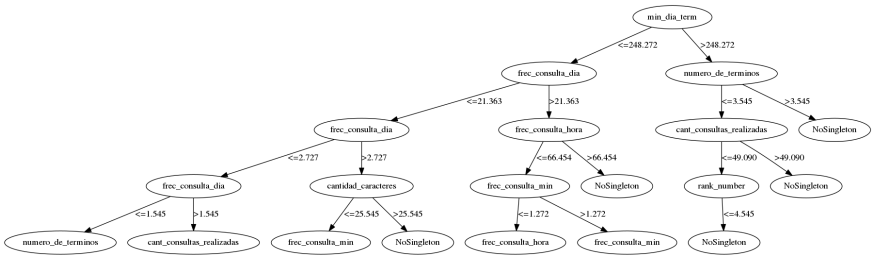

Fig. 5: Fragmento del árbol de decisión generado por HAT en evaluación predictiva secuencial de 20M instancias

Se analiza en detalle las clasificación realizada por ambos algoritmos utilizando tablas de contingencia. En las Tablas 2 y 3, se puede apreciar que ambos clasificadores tienen mejor especificidad que sensibilidad. Es decir, ambos identifican mejor a consultas no frecuentes, aunque la especificidad de C4.5 es superior a la de HAT, que es acompañada por una tasa de falsos positivos baja (aproximadamente un $0,1\%$). En cambio, al clasificar consultas frecuentes HAT se desempeña mejor, alcanzando una sensibilidad considerablemente mayor (cercana a un 18%). La misma es influida por los falsos negativos, que como se comenta antes es el error más severo para la clasificación de consultas. C4.5 tiene casi un 30% de falsos negativos contra un 11% de HAT, es decir casi tres veces superior. En consecuencia, esta relación se mantiene para la tasa de falsos negativos (o falsas consultas *singleton*), ya que para HAT es de un 26% versus un 69% de C4.5. Para el objetivo al que se destinan estos clasificadores, es apreciable que HAT tenga un porcentaje menor de falsos negativos con relación a C4.5.

5.2 Rendimiento ante cambios de concepto

En el ambiente de los motores de búsqueda, se producen cambios de concepto ligados a la variabilidad de las necesidades de información de los usuarios. Frente al conjunto de datos utilizado para la evaluación de los clasificadores, HAT resulta superior. Sin embargo, es probable que no se presenten cambios lo suficientemente importantes

como para que se evidencie el verdadero potencial de la adaptación a los cambios de concepto.

Un problema con la mayoría de los conjuntos de datos del mundo real disponibles para la experimentación es que poseen cambios de concepto diminutos [33]. Una opción para juzgar el rendimiento en un entorno realmente adverso es introducir artificialmente cambios de concepto en la distribución de los datos. Este método ha sido utilizado por otros autores para cuantificar la capacidad de los algoritmos para adaptar los modelos conforme se suceden los cambios [4]. En este caso, se evalúa a HAT se modificando la forma en que es definida la clase objetivo, dejándola solo ligada a los valores que tome uno de los atributos. Dicho de otra forma, se introduce una dependencia completa hacia el valor de un atributo por parte de la variable objetivo. De esta forma, se escoge al atributo 'cantidad de términos'.

El experimento consiste en tomar todas las instancias, y clasificar como frecuentes (no singletons) sólo a aquellas con cantidad de términos mayor a tres. Este atributo se escoge entre los demás por dos motivos. El primero, la cantidad de términos es ponderada para la asignación de la clase por C4.5 y HAT. Dado que el atributo se ubica en el segundo y tercer nivel de los arboles generados. Al introducirse un cambio de concepto, la ubicación cercana a la base del árbol obliga a que el modelo desactualizado deba cambiar radicalmente para poder capturar la nueva tendencia. El segundo motivo es que el cambio planteado podría producirse en un ambiente real. Por ejemplo, podría originarse una inclinación de los usuarios sobre un tema que requiere ser consultado con un número de términos superior a tres.

Para generar el conjunto de datos de evaluación utilizado en este experimento, se duplican 20M de instancias (obteniendo 40M) y sobre las 20M últimas se modifica el valor de la clase objetivo en base al del atributo cantidad de términos. Una vez que se introduce el cambio de concepto, la adaptación de HAT se percibe antes de las 100K instancias, apreciable en la Figura 6. Como se esperaba, el algoritmo actualiza el modelo para ajustarse a la nueva condición que define la clase. Para hacerlo éste poda antiguas ramas, eliminando reglas inválidas, hasta quedarse con solo la raíz que contiene la regla determinada por el atributo 'cantidad de términos'. Al modificarse correctamente las reglas que definen la clase, la exactitud de clasificación comienza un crecimiento continuo. La conformación final del modelo es representada en la Figura 7. A diferencia de HAT, C4.5 no tiene la capacidad de adaptarse a los cambios y su modelo aún se encuentra ajustado al antiguo concepto. Por lo tanto, el clasificador C4.5 comienza a deteriorar su rendimiento en cuanto enfrenta a las instancias con el desvío. No obstante, es de destacar que el entrenamiento de C4.5 le permite mantener prestaciones sin deteriorarse abruptamente, mostrando tolerancia al cambio aunque, obviamente, no puede mejorar su performance.

Finalmente, se evalúa la performance del caché de resultados comparando ambos algoritmos de decisión gestionando la política de admisión. Como política de reemplazo se utiliza LRU (Least Recently Used), ampliamente utilizada en una di-

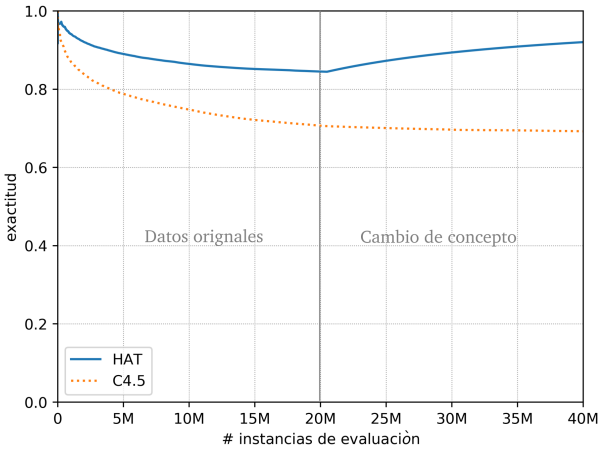

Fig. 6: Evaluación de HAT y C4.5 frente a un cambio de concepto

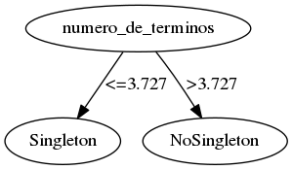

Fig. 7: Modelo resultante de la adaptación de HAT al cambio de concepto en la distribución de las consultas

versidad de aplicaciones. En este caso, la métrica utilizada para la evaluación es la tasa de acierto (*hit rate*), que corresponde a la cantidad de ítem encontrados en caché, respecto del total de intentos.

La Figura 8 muestra los resultados. Como se puede apreciar, con tamaños de caché pequeños las tasas de aciertos son bajas, resultando superior el árbol C4.5. Luego, por sobre las 7.5K entradas, HAT prevalece, con crecimiento ininterrumpido en la tasa de aciertos para mayores capacidades de caché. Esto último, refleja el comportamiento descripto en el análisis de la clasificación (Tablas 2 y 3). Es interesante notar que las prestaciones de LRU con la política de admisión basada en C4.5 comienza a estabilizar su performance a partir de 7.5K de instancias, mientras que aquella basas en HAT la supera. Una posible explicación es que el entrenamiento de C4.5 ha capturado la frecuencia de aparición de las consultas en las primeras

100K apariciones, factor que impacta positivamente en LRU. El estudio más profundo de este comportamiento utilizando otras políticas de reemplazo está fuera de los alcances de este trabajo pero se considera interesante.

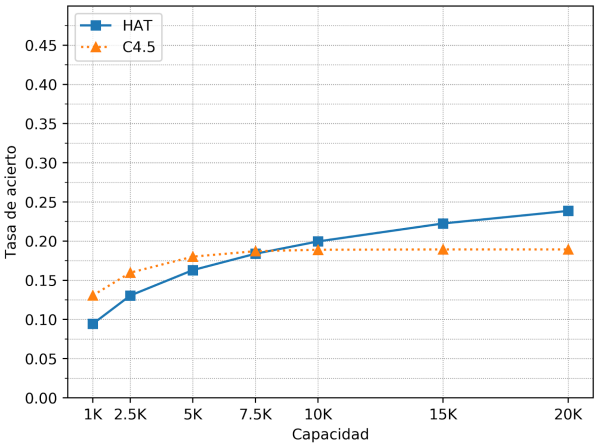

Fig. 8: Tasa de acierto en caché de resultados gestionado con la política LRU, usando C4.5 y HAT en la admisión.

6 Conclusiones y Trabajos Futuros

Los servicios digitales actuales producen grandes volúmenes de datos que desafían las infraestructuras y las técnicas de aprendizaje automático tradicionales para extraer conocimiento. Los algoritmos de *stream mining* aportan soluciones a problemas en entornos en los que se requiere tomar decisiones cercanas a tiempo real.

En este trabajo, se busca optimizar la performance de la caché de resultados de un motor de búsqueda utilizando un algoritmo de árboles de decisión adaptativo (HAT) a partir de un flujo de consultas. La comparación de este algoritmo con otro, considerado de los clásicos, muestra una superioridad por parte HAT para desempeñar esta tarea, aportando una mejora en exactitud del 14,5%, con una tendencia que indica que la diferencia podría incrementarse con mas instancias. Complementariamente, se muestra cómo el modelo se ajusta a los cambios de concepto. Finalmente, se evalúa la performance del caché de resultados considerando la admisión usando ambas estrategias de decisión.

En trabajos futuros se propone optimizar el proceso de admisión a los efectos de disminuir los falsos negativos en la clasificación ya que nunca son admitidos en caché, siendo que son frecuentes. Además, se propone analizar el impacto de la admisión selectiva ante diferentes criterios de desalojo (frecuencia, tamaño, costo) y se planea expandir el estudio a los demás niveles de caché en una máquina de búsqueda.

References

1. Baeza-Yates, R., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., Silvestri, F.: The impact of caching on search engines. In: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 183–190. ACM (2007)
2. Baeza-Yates, R., Gionis, A., Junqueira, F.P., Murdock, V., Plachouras, V., Silvestri, F.: Design trade-offs for search engine caching. ACM Transactions on the Web (TWEB) 2(4), 20 (2008)
3. Bifet, A., Gavalda, R.: Learning from time-changing data with adaptive windowin g. In: Proceedings of the 2007 SIAM international conference on data mining. pp. 443–448. SIAM (2007)
4. Bifet, A., Gavalda, R.: Adaptive parameter-free learning from evolving data streams. In: Conf. Advances in Intelligent Data Analysis VIII, 8th Int. Symp. on Intelligent Data Analysis, Lyon, France (2009)
5. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: Moa: Massive online analysis. Journal of Machine Learning Research 11(May), 1601–1604 (2010)
6. Blanco, R., Bortnikov, E., Junqueira, F., Lempel, R., Telloli, L., Zaragoza, H.: Caching search engine results over incremental indices. In: Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval. pp. 82–89. ACM (2010)
7. Van den Bosch, A., Bogers, T., De Kunder, M.: Estimating search engine index size variability: a 9-year longitudinal study. Scientometrics 107(2), 839–856 (2016)
8. Claesen, M., De Moor, B.: Hyperparameter search in machine learning. arXiv preprint arXiv:1502.02127 (2015)
9. Cohen, L., Avrahami-Bakish, G., Last, M., Kandel, A., Kipersztok, O.: Real-time data mining of non-stationary data streams from sensor networks. Information Fusion 9(3), 344–353 (2008)
10. Datar, M., Gionis, A., Indyk, P., Motwani, R.: Maintaining stream statistics over sliding windows. SIAM journal on computing 31(6), 1794–1813 (2002)
11. De Francisci Morales, G.: Samoa: A platform for mining big data streams. In: Proceedings of the 22nd International Conference on World Wide Web. pp. 777–778. ACM (2013)
12. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 71–80. ACM (2000)
13. Fagni, T., Perego, R., Silvestri, F., Orlando, S.: Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. ACM Transactions on Information Systems (TOIS) 24(1), 51–78 (2006)

14. Fawcett, T.: An introduction to roc analysis. Pattern recognition letters 27(8), 861–874 (2006)
15. Friedman, J.H.: Stochastic gradient boosting. Computational Statistics & Data Analysis 38(4), 367–378 (2002)
16. Gama, J., Gaber, M.M.: Learning from data streams: processing techniques in sensor networks. Springer (2007)
17. Gan, Q., Suel, T.: Improved techniques for result caching in web search engines. In: Proceedings of the 18th international conference on World wide web. pp. 431–440. ACM (2009)
18. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1), 10–18 (2009)
19. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 97–106. ACM (2001)
20. Kucukyilmaz, T., Cambazoglu, B.B., Aykanat, C., Baeza-Yates, R.: A machine learning approach for result caching in web search engines. Information Processing & Management 53(4), 834–850 (2017)
21. Long, X., Suel, T.: Three-level caching for efficient query processing in large web search engines. World Wide Web 9(4), 369–395 (2006)
22. Mitchell, T.M., et al.: Machine learning. 1997. Burr Ridge, IL: McGraw Hill 45(37), 870–877 (1997)
23. Ozcan, R., Altingovde, I.S., Ulusoy, Ö.: Static query result caching revisited. In: Proceedings of the 17th international conference on World Wide Web. pp. 1169–1170. ACM (2008)
24. Pass, G., Chowdhury, A., Torgeson, C.: A picture of search. In: InfoScale. vol. 152, p. 1 (2006)
25. Quinlan, J.R.: C4. 5: programs for machine learning. Elsevier (2014)
26. Russell, S.J., Norvig, P.: Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited, (2016)
27. Sagiroglu, S., Sinanc, D.: Big data: A review. In: Collaboration Technologies and Systems (CTS), 2013 International Conference on. pp. 42–47. IEEE (2013)
28. Sayed-Mouchaweh, M., Lughofer, E.: Learning in non-stationary environments: methods and applications. Springer Science & Business Media (2012)
29. Shih, B.Y., Chen, C.Y., Chen, Z.S.: Retracted: an empirical study of an internet marketing strategy for search engine optimization. Human Factors and Ergonomics in Manufacturing & Service Industries 23(6), 528–540 (2013)
30. Subasic, I., Castillo, C.: The effects of query bursts on web search. In: Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on. vol. 1, pp. 374–381. IEEE (2010)
31. Tolosa, G., Feuerstein, E., Becchetti, L., Marchetti-Spaccamela, A.: Performance improvements for search systems using an integrated cache of lists+ intersections. Information Retrieval Journal 20(3), 172–198 (2017)
32. Tolosa, G.H.: Técnicas de caching de intersecciones en motores de búsqueda. Ph.D. thesis, Universidad Nacional de Buenos Aires (2016)

33. Tsymbal, A.: The problem of concept drift: definitions and related work. Computer Science Department, Trinity College Dublin 106(2) (2004)
34. Zhou, W., Li, R., Dong, X., Xu, Z., Xiao, W.: An intersection cache based on frequent itemset mining in large scale search engines. In: Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on. pp. 19–24. IEEE (2015)