UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR

INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Contribuciones a la selección distribuida de proxy en el acceso a la web a través de redes de comunidad inalámbricas

Autor:

D. Diego Bores Quijano

Tutor:

Dr. D. Eduardo Gómez Sánchez
Dr. D. Miguel Luis Bote Lorenzo

Valladolid, 18 de julio de 2018

TÍTULO:	**Contribuciones a la selección distribuida de proxy en el acceso a la web a través de redes de comunidad inalámbricas**
AUTOR:	**D. Diego Bores Quijano**
TUTOR:	**Dr. D. Eduardo Gómez Sánchez** **Dr. D. Miguel Luis Bote Lorenzo**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

Tribunal

PRESIDENTE:	**Dr. D. Juan Ignacio Asensio Pérez**
VOCAL:	**Dr. D. Federico Simmross Wattenberg**
SECRETARIO:	**Dr. D. Rodrigo de Luis García**
FECHA:	**18 de julio de 2018**
CALIFICACIÓN:	

Resumen del TFG

Las redes de comunidad inalámbricas son redes en malla en las que es frecuente que el acceso a la Web se haga a través de unos pocos *proxies*, siendo los usuarios quienes eligen su *proxy* preferido. Esto puede hacer que la carga se concentre en unos pocos *proxies* dando lugar a una calidad de servicio percibida por los usuarios finales degradada. Para elegir el *proxy* menos cargado sin necesidad de que estos informen de su carga, algunos autores proponen estimarla a partir del *Time to First Byte* (TTFB). En este trabajo se exploran las posibilidades de estimación y de predicción de dicho TTFB, usando datos de la red de comunidad Guifi.net. En primer lugar se estudian los diferentes métodos de recomendación colaborativa, evaluando a continuación el rendimiento de BRISMF. Este algoritmo alcanza cierta mejora frente a un algoritmo simple de referencia, aunque no demasiado significativa en la mayoría de los casos estudiados. Por otra parte, se propone el estudio del problema como la predicción de una serie temporal, probando algunos de los algoritmos de predicción populares. De nuevo se observa una mejora frente al algoritmo de referencia que tampoco es especialmente relevante. No obstante, este estudio abre oportunidades para mejorar la estimación o predicción del TTFB, reduciendo el número de sondeos desde los clientes o la cantidad de información compartida entre estos.

Palabras clave

Redes de comunidad, *proxy*, predicción, aprendizaje automático, sistemas de recomendación, series temporales

Abstract

Wireless Community Networks are mesh networks in which web access is tipically acomplished through a few proxies, with users choosing their preferred proxy. This may overload a few proxies, yielding to a poor quality of service, as perceived by end users. In order to choose the least loaded proxy without them having to report their load, some authors propose estimating it form Time to First Byte (TTFB). In this work we explore the possibilities of estimating and predicting the TTFB, using data from the Guifi.net community network. First, different methods of collaborative recommendation are studied, evaluating then performance of BRISMF. This algorithm achives certain improvement over a simple reference algorithm, although not too large in most of the cases studied. On the other hand, we also approach the problem as time series prediction, testing some of the popular algorithms for it. Once again, an improvement over the reference algorithm is observed, altought is not particularly relevant either. Nevertheless, this study opens opportunities to improve both the estimation and the prediction of the TTFB, reducing the number of polls made by clients or the amount of information shared among them.

Keywords

Community Networks, proxy, prediction, machine learning, recommender systems, time series

Agradecimientos

En primer lugar, me gustaría dar las gracias a todos los que me han apoyado de una manera u otra en esta etapa de mi vida. En lo académico, me gustaría dar la gracias a Miguel y a Eduardo, tutores de este trabajo fin de grado, por el esfurzo dedicado en resolver las dudas que me han ido surgiendo, y a los Drs. Manos Dimos y Roc Meseguer por su generosidad al capturar y compartir el conjunto de datos de la red Guifi.net usado en este proyecto, así como su disponibilidad para resolver dudas sobre el funcionamiento de dicha red. También me gustaría dar las gracias a mis compañeros de clase, por haber hecho más llevaderos estos cuatro años de carrera, especialmente este último.

Me gustaría agradecer también a mi familia más cercana, y en especial a mi hermana, por su apoyo incondicional y por haber sabido ayudarme en los momentos de más estrés, y a mis amigos por haberme ayudado en los momentos de evasión tan necesarios para poder ver los problemas con una nueva perspectiva.

Muchas gracias a todos.

Índice general

1. Introducción **1**
- 1.1. Las redes de comunidad inalámbricas . . . 1
- 1.2. Objetivos . . . 3
- 1.3. Estructura del documento . . . 3

2. Planteamiento del problema **5**
- 2.1. Introducción . . . 5
- 2.2. El problema de elección de *proxy* . . . 5
- 2.3. Datos . . . 8
- 2.4. Algoritmo de referencia . . . 14
- 2.5. Evaluación del rendimiento . . . 15
- 2.6. Conclusiones . . . 16

3. Estimación de TTFB mediante filtrado colaborativo **17**
- 3.1. Introducción . . . 17
- 3.2. Formulación del problema . . . 18
  - 3.2.1. El problema de la recomendación . . . 18
  - 3.2.2. Estimación del TTFB como un problema de recomendación . . . 19
- 3.3. Selección de algoritmo . . . 19
  - 3.3.1. Tipos de sistemas de recomendación . . . 19
  - 3.3.2. Adecuación de los tipos de sistemas de recomendación para la estimación de TTFB . . . 21
  - 3.3.3. Tipos de algoritmos de filtrado colaborativo . . . 22
  - 3.3.4. Adecuación de los tipos de algoritmos de filtrado colaborativo para la estimación del TTFB . . . 24
  - 3.3.5. Factorización matricial . . . 25
  - 3.3.6. Algoritmos de factorización matricial . . . 25
  - 3.3.7. Adecuación de los algoritmos de factorización matricial para la estimación del TTFB . . . 26
  - 3.3.8. BRISMF . . . 27
- 3.4. Marco experimental . . . 29
  - 3.4.1. Entorno experimental . . . 29
  - 3.4.2. Parámetros de experimentación . . . 30
- 3.5. Resultados . . . 30
- 3.6. Discusión y Conclusiones . . . 32

4. Predicción de TTFB mediante series temporales 34
4.1. Introducción . . . 34
4.2. Formulación del problema . . . 35
4.2.1. La predicción de series temporales . . . 35
4.2.2. Predicción del TTFB como una serie temporal . . . 35
4.3. Selección de algoritmos . . . 36
4.3.1. Algoritmos de predicción de series temporales . . . 36
4.4. Marco experimental . . . 37
4.4.1. Datos . . . 37
4.4.2. Entorno experimental y parámetros de experimentación . . . 37
4.5. Resultados . . . 38
4.6. Discusión y conclusiones . . . 40

5. Conclusiones y trabajo futuro 41
5.1. Trabajo futuro . . . 42

Referencias 43

A. Figuras de TTFB 48
A.1. Evolución del TTFB para cada *proxy* . . . 48
A.2. Diagramas de cajas de las muestras por pareja cliente-*proxy* . . . 54

B. Figuras de evolución del MAE en el uso de sistemas de recomendación para la estimación del TTFB 56
B.1. Evolución del MAE por pareja cliente-*proxy* . . . 56
B.2. Diagramas de cajas de los errores . . . 62

C. Figuras de evolución del MAE en el uso de series temporales para la predicción del TTFB 65

D. Información adicional sobre los experimentos 67
D.1. Código de la estimación mediante BRISMF . . . 67
D.2. Parámetros de los algoritmos de predicción . . . 69
D.2.1. Perceptron . . . 69
D.2.2. FIMT-DD . . . 69
D.2.3. ORTO . . . 70
D.2.4. AMRules . . . 71

Índice de figuras

2.1. Tiempo transcurrido entre muestras sucesivas desde el cliente 1 10
2.2. Tiempo transcurrido entre muestras sucesivas desde el cliente 2 10
2.3. Tiempo transcurrido entre muestras sucesivas desde el cliente 3 11
2.4. Tiempo transcurrido entre muestras sucesivas desde el cliente 4 11
2.5. Tiempo transcurrido entre muestras sucesivas para el *proxy* 1 12
2.6. Diagrama de cajas del total de muestras. 12
2.7. Diagramas de cajas de las muestras recogidas por clientes y *proxies* . . . 13
2.8. Evolución del TTFB al *proxy* 1 14

A.1. Evolución del TTFB al *proxy* 2 48
A.2. Evolución del TTFB al *proxy* 3 49
A.3. Evolución del TTFB al *proxy* 4 49
A.4. Evolución del TTFB al *proxy* 5 50
A.5. Evolución del TTFB al *proxy* 6 50
A.6. Evolución del TTFB al *proxy* 7 51
A.7. Evolución del TTFB al *proxy* 9 51
A.8. Evolución del TTFB al *proxy* 10 52
A.9. Evolución del TTFB al *proxy* 11 52
A.10. Evolución del TTFB al *proxy* 12 53
A.11. Evolución del TTFB al *proxy* 13 53
A.12. Diagrama de cajas de las muestras por *proxy* para el cliente 1 54
A.13. Diagrama de cajas de las muestras por *proxy* para el cliente 2 54
A.14. Diagrama de cajas de las muestras por *proxy* para el cliente 3 55
A.15. Diagrama de cajas de las muestras por *proxy* para el cliente 4 55

B.1. Evolución del MAE frente a la evolución del TTFB en el *proxy* 1 56
B.2. Evolución del MAE frente a la evolución del TTFB en el *proxy* 2 57
B.3. Evolución del MAE frente a la evolución del TTFB en el *proxy* 3 57
B.4. Evolución del MAE frente a la evolución del TTFB en el *proxy* 4 58
B.5. Evolución del MAE frente a la evolución del TTFB en el *proxy* 5 58
B.6. Evolución del MAE frente a la evolución del TTFB en el *proxy* 6 59
B.7. Evolución del MAE frente a la evolución del TTFB en el *proxy* 7 59
B.8. Evolución del MAE frente a la evolución del TTFB en el *proxy* 9 60
B.9. Evolución del MAE frente a la evolución del TTFB en el *proxy* 10 60
B.10. Evolución del MAE frente a la evolución del TTFB en el *proxy* 11 61
B.11. Evolución del MAE frente a la evolución del TTFB en el *proxy* 12 61

B.12. Evolución del MAE frente a la evolución del TTFB en el *proxy* 13 62
B.13. Diagrama de cajas de los errores para el cliente 1 62
B.14. Diagrama de cajas de los errores para el cliente 2 63
B.15. Diagrama de cajas de los errores para el cliente 3 63
B.16. Diagrama de cajas de los errores para el cliente 4 64

C.1. Evolución del MAE frente a la evolución del TTFB en el *proxy* 2 65
C.2. Evolución del MAE frente a la evolución del TTFB en el *proxy* 5 66
C.3. Evolución del MAE frente a la evolución del TTFB en el *proxy* 13 66

Índice de tablas

2.1. Número de muestras recogidas por pareja cliente-*proxy* 9

3.1. Valores del error absoluto medio obtenidos con el algoritmo de referencia 31
3.2. Valores del error absoluto medio obtenidos con BRISMF 31
3.3. Tabla con el porcentaje de mejora en términos de MAE en el uso de BRISMf respecto del algoritmo de referencia 32

4.1. Tabla con los resultados de MAE de cada uno de los algoritmos y para cada tamaño de ventana para la pareja cliente 1 - *proxy* 2. El MAE del algoritmo de referencia para esta pareja cliente-*proxy* es 0.19690. 38
4.2. Tabla con los resultados de MAE de cada uno de los algoritmos y para cada tamaño de ventana para la pareja cliente 1 - *proxy* 5. El MAE del algoritmo de referencia para esta pareja cliente-*proxy* es 0.02436. 38
4.3. Tabla con los resultados de MAE de cada uno de los algoritmos y para cada tamaño de ventana para la pareja cliente 1 - *proxy* 13. El MAE del algoritmo de referencia para esta pareja cliente-*proxy* es 0.06341. 39
4.4. Tabla con el porcentaje de mejora de cada uno de los algoritmos con respecto al algoritmo de referencia para el tamaño de ventana temporal que mejor resultado reporta. 39

Capítulo 1

Introducción

1.1. Las redes de comunidad inalámbricas

Las redes de comunidad son redes distribuidas, descentralizadas y a gran escala que permiten a comunidades locales crear su propia infraestructura de red [Veg15], proporcionando además interconexión con Internet de forma asequible incluso en áreas rurales [RM13]. Esto se consigue gracias al abaratamiento de costes que se produce al compartir recursos como la infraestructura o el acceso a Internet [Uni09, Par14]. La operación de estas redes se basa, por tanto, en el principio de cooperación entre sus miembros. Estas comunidades suelen tener ciertas reglas de participación como licencias de pertenencia o acuerdos de comunicación entre pares, que definen la libertad, neutralidad y franqueza de los miembros de la red [Neu16].

Las redes de comunidad tienen dos propósitos principales [Mac13]. Uno de ellos es permitir interacciones entre los usuarios. En este sentido hay multitud de aplicaciones para estas redes que incluyen *e-learning* o mensajería instantánea [Lin08]. El segundo de los propósitos es reducir el coste del acceso a Internet, o simplemente facilitarlo en los lugares donde no hay una gran infraestructura.

En multitud de ciudades europeas hay redes de comunidad inalámbricas, como la existente en Atenas con más de 2.000 nodos, la austriaca Funk-Feuer[1] o la italiana Ninux con más de 200 nodos [Mac13]. En España Guifi.net cuenta con más de 30.000 nodos [Dim17], está situada mayormente en Cataluña y contiene varias puertas de enlace a Internet, que típicamente actúan también como *proxies* web, compartidas por los participantes [Veg15].

En cuanto a su topología, Guifi.net tiene forma de malla con enlaces típicamente punto a punto, y se divide en sistemas autónomos formados por cientos de nodos que utilizan un protocolo de enrutado interno, como OSPF[2](*Open Shortest Path First*, primer camino más corto) u OLSR[3] (*Optimized Link State Routing Protocol*, protocolo de enrutado de estado de enlace optimizado), y que se conectan con sistemas autónomos cercanos vía protocolos de enrutado externo como BGP [4] (*Border Gateway Protocol*, protocolo de

[1]https://www.funkfeuer.at/

[2] OSPF versión 2: disponible en https://tools.ietf.org/html/rfc2328

[3]OSLR versión 1: disponible en https://tools.ietf.org/html/rfc3626

[4]BGP versión 4: disponible en https://tools.ietf.org/html/rfc4271

puerta de enlace de frontera) [Neu16]. El método más común de acceso a la Web es a través de *proxies*, siendo los usuarios los que eligen su *proxy* preferido de entre todos los disponibles en la red. Para acceder a un servidor web externo es necesario recorrer una ruta dentro de Guifi.net que atraviesa varios enlaces y nodos encaminadores hasta llegar al *proxy* seleccionado, que realizará la petición al servidor externo en nombre de los usuarios.

En las redes de comunidad inalámbricas, todos los nodos inalámbricos que forman parte de ella actúan como encaminadores, recibiendo paquetes en todas sus interfaces. Si el paquete está dirigido al nodo al que llega, éste lo procesa. En caso contrario lo reenvía a uno de sus vecinos [Jai03].

Por otra parte, en las redes de comunidad, y en particular en Guifi.net, los enlaces son aportados y organizados por los participantes, lo que hace que los caminos entre nodos, o de cliente a *proxy*, puedan ser no fiables [Aky05]. De esta manera, los accesos a la Web a través de *proxies* dependen de usuarios u organizaciones que comparten toda su conexión a Internet con otros usuarios [Dim17]. Por tanto, Guifi.net se enfrenta a dos retos principales. El primero es el hecho de que los enlaces son volátiles, por lo que es importante predecir su calidad [Mil15, BL18]. El segundo radica en el hecho de que los usuarios eligen un *proxy* de una lista con los *proxies* disponibles, y existe la posibilidad de congestión en estos nodos de salida, lo que causaría que el acceso a la Web presentase un rendimiento degradado [Dim17]. Como consecuencia de cierta falta de regulación al respecto, y pese a ser un servicio crítico para los usuarios, podría ocurrir que mientras unos *proxies* estuviesen subocupados, otros acabaran saturados por la demanda. Por todo ello, la elección del *proxy* es un reto abierto que puede afectar de manera relevante a la percepción de calidad de servicio que tienen los usuarios finales.

Este trabajo se centra en la solución del segundo problema. Una posible aproximación podría ser estimar la carga de los *proxies* y elegir los de menor carga [Dim17]. De esta forma se contribuye al buen rendimiento global de la red, puesto que se reparte la carga entre todos los *proxies*. Otra solución podría ser elegir desde cada cliente el *proxy* con menor latencia. En este caso se busca optimizar la percepción del servicio por parte del usuario final. Para la primera podría ser de ayuda que los *proxies* informasen de su carga, pero esto implicaría modificar los protocolos y se busca una solución que sea incremental y compatible hacia atrás, que funcione tanto con los clientes mejorados como con los existentes, y que sea independiente de los algoritmos de transporte y enrutado, es decir, que no implique realizar cambios incompatibles con el entorno existente [Dim17]. Sin embargo, existe cierta relación entre ambas soluciones. En [Dim17] se muestra que el TTFB (*Time to First Byte*, tiempo hasta el primer *byte*) es una métrica que puede obtener el cliente sin necesidad de modificar los protocolos de comunicación, y además es una medida razonable de la carga de un *proxy*.

Sin embargo, y como se explica más adelante, la solución de selección de *proxy* mediante estimación de la carga a partir del TTFB tiene ciertas limitaciones. La primera es que un cliente puede conocer la estimación de la carga actual de *proxies* que no sondea y que puede recibir de otros clientes que sí lo hacen si estos comparten esta información, pero sería deseable conocer la estimación de la carga actual de *proxies* no sondeados sin tener que esperar a recibir todas las actualizaciones de las medidas realizadas por otros clientes, ya que esto podría tardar más de una ronda Vivaldi. La segunda es que un cliente

conoce la estimación actual de los *proxies* que él sondea, pero sería deseable tener una estimación de su carga futura que permitiese al cliente anticiparse a cambios de carga en los *proxies*. La solución a ambas limitaciones podría ser predecir el valor del TTFB, tanto a futuro para los *proxies* que un cliente mide como a presente para los que no mide, y esto es lo que aborda este trabajo.

1.2. Objetivos

El objetivo del presente trabajo es **abordar el problema de elección de *proxy* en las redes de comunidad inalámbricas desde una perspectiva de aprendizaje automático mediante la predicción del TTFB**, con la intención de mejorar la calidad de la elección del *proxy* utilizado y de esta manera mejorar la calidad de servicio percibida por los usuarios de las redes de comunidad.

Este objetivo se va a abordar de dos formas distintas. Por una parte, bajo la suposición de que un cliente realiza sondeos (mediciones del TTFB) a un determinado *proxy*, se tratará de predecir los valores en el futuro inmediato de este TTFB. De acuerdo con [Dim17] en que el TTFB es un buen estimador de la carga de un *proxy*, predecir los valores del TTFB permitirá tener en cuenta no solo la situación actual, sino la situación futura, a la hora de elegir un *proxy*, con la esperanza de anticiparse a los cambios en su carga. Por otra parte, y bajo la suposición de que algunos clientes sondean a algunos *proxies* y a continuación comparten entre ellos los resultados de las medidas, se va a tratar de que un cliente pueda estimar en el instante actual el TTFB de todos los *proxies*, incluyendo aquellos a los que no sondea. Esto permitiría mantener bajo el número de sondeos realizados sin reducir la información con la que cuentan los clientes a la hora de elegir un *proxy*.

1.3. Estructura del documento

En cuanto a la estructura del documento, en el capítulo 2 se describen los datos utilizados para el entrenamiento y evaluación de los diferentes modelos utilizados para la estimación y la predicción del TTFB, así como el algoritmo de referencia con el que se comparan los resultados, la métrica de evaluación del rendimiento y el *software* empleado.

En el capítulo 3 se realiza una aproximación a la estimación del TTFB mediante el uso de las herramientas de los sistemas de recomendación. Se presentan los diferentes tipos de sistemas de recomendación, y se justifica la elección del algoritmo concreto empleado. También se realiza una descripción del código empleado para la estimación, así como de los parámetros implicados en el proceso. A continuación se plantean los resultados y por último las conclusiones extraídas.

En el capítulo 4 se plantea la predicción del TTFB como un problema de series temporales. Se explican los algoritmos de aprendizaje automático empleados, se exponen los resultados obtenidos y por último se realiza una discusión sobre las conclusiones derivadas.

Por último, en el capítulo 5 se exponen las conclusiones globales obtenidas y el trabajo futuro. En los apéndices se presentan figuras adicionales que podrían ser de utilidad a la hora de entender tanto el problema, como los datos de partida y los resultados obte-

nidos. También se adjunta el código empleado en los experimentos del capítulo 3 y los parámetros de los experimentos del capítulo 4.

Capítulo 2

Planteamiento del problema

2.1. Introducción

Como se indica en el capítulo de introducción, en las redes de comunidad surge el problema de la elección de un *proxy* que permita un acceso de calidad a la Web. En [Dim17] se plantea una forma de estimación de la carga a partir del TTFB, que se basa en que los clientes sólo sondean una porción de los *proxies* existentes para medir su TTFB, y esta información la comparten entre ellos para reducir la sobrecarga introducida.

Este trabajo parte de la hipótesis de que el TTFB es un buen estimador de la carga de un *proxy* que permite elegir el mejor [Dim17], y pretende mejorarlo proponiendo, por una parte, la estimación de la carga no medida a partir de valores pasados compartidos por otros clientes, y por otra, la predicción de este estimador a partir de los valores pasados medidos por cada cliente. Al igual que [Dim17], el estudio se realiza sobre la red Guifi.net, debido a la posibilidad de instalar *software* de prueba en los clientes, lo que facilita la recogida de un conjunto de datos sobre los que evaluar los métodos propuestos y que se describirán más adelante.

En este capítulo se describe en primer lugar en la sección 2.2 la investigación realizada en [Dim17] sobre la elección del *proxy* a partir de la estimación de su carga basada en el TTFB, y se discuten las limitaciones de esta aproximación. A continuación, en la sección 2.3 se describen los datos utilizados en el estudio de la predicción del TTFB. En la sección 2.4 se explica el algoritmo de referencia utilizado en los siguientes capítulos, y por último en la sección 2.5 se muestra el método de evaluación de los resultados obtenidos.

2.2. El problema de elección de *proxy*

En las redes de comunidad, los usuarios eligen su *proxy* de acceso a la Web de una lista que contiene los *proxies* disponibles [Dim17]. A consecuencia de la falta de regulación podría ocurrir que mientras unos *proxies* estén saturados por un alto número de peticiones, otros permanezcan subocupados o inactivos, degradando la QoE (*Quality of Experience*, calidad de experiencia) percibida por los usuarios en el acceso a la Web [Dim17].

El objetivo es, por tanto, diseñar una solución práctica, no óptima pero de mejor esfuerzo o *best-effort*, donde los clientes puedan elegir un cierto *proxy* utilizando métricas de uso de red y rendimiento de los *proxies* que no requieran modificar ningún componente

de la red y que puedan funcionar en un entorno heterogéneo [Dim17] como son las redes de comunidad. Para ello los clientes cooperan compartiendo el rendimiento de la red y de los *proxies*, y utilizan esta información para ordenar su lista de *proxies* conocidos. La información que comparten es sólo la que ellos mismos miden, o la que reciben de otros clientes. Esto es así porque los clientes no tienen acceso a las métricas de carga o rendimiento de los *proxies*, ya que esto implicaría modificar los protocolos e incumplir el requisito del uso de métricas que no requieran modificar ningún componente de la red.

Por tanto, se utilizan métricas adecuadas para estimar la carga del *proxy* únicamente a partir de variables medidas por los clientes, y en función de dichas métricas se elige, de manera conservadora, el *proxy* con menor carga estimada. Esta elección conservadora podría consistir en cambiar de *proxy* sólo si la métrica utilizada es mejorada por encima de un umbral, o durante un periodo de tiempo lo suficientemente grande, que permita evitar cambios innecesarios de *proxy* por parte de los clientes.

En consecuencia, también es necesario obtener métricas sobre la latencia desde el cliente a otros elementos de la red, puesto que han de ser utilizadas en la elección tanto de la ruta como del *proxy*. Sin embargo, sondear a todos los elementos generaría una sobrecarga importante. La solución es que cada cliente participe en una red Vivaldi para estimar su RTT (*Round-Trip Time*, tiempo de ida y vuelta) con otros clientes. Vivaldi es un algoritmo que asigna coordenadas a cada *host* de forma que la distancia euclídea entre dos *hosts* en este sistema sirve como estimación de la latencia entre ellos [Dab04]. Cada nodo mantiene una lista de $C + R$ clientes conocidos, C que se estiman más cercanos en cuanto a distancia euclídea entre coordenadas Vivaldi, y otros R aleatorios en cualquier lugar de la red. En cada ronda un nodo selecciona de forma aleatoria un vecino de la lista, realiza N sondeos, y le pide sus coordenadas Vivaldi y las de sus vecinos.

De forma similar se puede obtener el RTT desde un nodo a un *proxy* [Dim17], aunque estos no participen de forma activa en el sistema Vivaldi. Cada cliente mantiene C+R coordenadas que representan a *proxies*, de forma similar a los clientes. En cada ronda, cada nodo realiza N consultas al *proxy* elegido para esa ronda, actualiza las coordenadas que tiene para ese *proxy* y comparte la latencia con el vecino seleccionado en la ronda actual.

El estimador de rendimiento del *proxy* propuesto en [Dim17] se basa en la suposición de que el TTFB en una petición HTTP puede reflejar la calidad del servicio. El TTFB es el tiempo que transcurre desde que se realiza una petición hasta que se recibe el primer *byte* de la respuesta, y se podría ver afectado de forma proporcional por la carga del *proxy* [Dim17]. De este modo los *proxies* con baja carga ofrecerán, en principio y siempre que el RTT entre cliente y *proxy* sea bueno, bajos valores del TTFB, y *proxies* con alta carga ofrecerán altos valores de TTFB. Cada cliente calcula de forma pasiva el TTFB de las respuestas HTTP que recibe del *proxy*, y utiliza este valor para estimar la carga del *proxy* y compartirlo con sus vecinos en la red Vivaldi.

Para una toplogía estática de una WMN (*Wireless Mesh Network*, redes inalámbricas en malla), con ciertas suposiciones sobre su calidad y condiciones dinámicas del enlace (un enlace muy lento no es distinguible de uno muy congestionado), se puede esperar que la latencia sea la suma del tiempo requerido para establecer la conexión con el *proxy* y enviar la petición, el tiempo que tarda el *proxy* en procesar la petición hasta que inicia la

petición al servidor remoto, y el tiempo que tarda el *proxy* en completar la petición.

$$t_{lat} \approx t_{request_c_p} + t_{proxy_p} + t_{response_c_p} \tag{2.1}$$

El primer sumando correspondería a cierto número de veces por el RTT entre cliente y *proxy*,

$$t_{request_c_p} \approx A * t_{mesh_rtt_c_p} \tag{2.2}$$

y este valor de RTT podría emplearse para evitar caminos cliente-*proxy* lentos. El segundo sumando incluye el tiempo que la petición espera para empezar a ser servida, con lo que podría ser un buen indicador de la carga del *proxy*, y por tanto se podría utilizar para evitar *proxies* sobrecargados.

En cuanto al rendimiento del *proxy*, se utiliza el TTFB para estimarlo. El TTFB funciona bien como indicador de la responsividad de servicios web [Dim17], ya que combina el tiempo de conexión TCP y el de procesamiento del servidor remoto. Pero en nuestro caso el escenario es más complicado, pues aspiramos a utilizar el TTFB medido por el cliente para estimar el rendimiento del *proxy* que media entre el cliente y el contenido pedido.

El tiempo que el *proxy* necesita para completar la petición se puede aproximar por la suma del tiempo que el *proxy* tarda en recibir el primer *byte* y el tiempo transcurrido en el *proxy* desde que recibe el primer *byte* hasta que acaba de recibir la respuesta,

$$t_{response_c_p} \approx t_{proxy,ttfb} + t_{transport_response} \tag{2.3}$$

Ambos dependen de la capacidad de la conexión a Internet del *proxy*, de los retardos en el camino entre el *proxy* y el servidor remoto, así como de la capacidad de respuesta del servidor remoto.

Si asumimos que el contenido al que intentan acceder todos los clientes siempre está disponible, en servidores remotos a una distancia similar de todos los *proxies* y con la misma capacidad de conexión a Internet, el tiempo que tarda el *proxy* en recibir la respuesta completa del servidor remoto desde que recibe el primer *byte* de dicha respuesta será igual para todos, y por tanto se puede obviar en la estimación de la carga. Por tanto, el TTFB entre cliente y *proxy* se puede aproximar por dos veces el RTT entre cliente y *proxy* más el tiempo de respuesta del proxy y más el TTFB entre el *proxy* y el servidor remoto:

$$t_{ttfb_c_p} \approx 2 * t_{mesh_rtt_c_p} + t_{proxy_p} + t_{proxy_ttfb} \tag{2.4}$$

y de esta forma, el tiempo de respuesta del *proxy* se puede aproximar como el TTFB del cliente al *proxy* menos dos veces el RTT entre cliente y *proxy*:

$$t_{proxy_p} \approx t_{ttfb_c_p} - 2 * t_{mesh_rtt_c_p} \tag{2.5}$$

Pero las medidas del TTFB pueden ser muy ruidosas, ya que a veces los paquetes se ven retrasados por la carga de la red, y en ocasiones los *proxies* pueden completar una petición rápidamente pese a una alta carga. En [Dim17] se filtran los valores del tiempo de respuesta del *proxy* con una media exponencial para reducir el efecto de dicho ruido.

Cuando los retardos de las peticiones HTTP son mayores que el tiempo de Vivaldi, que corresponde al tiempo entre dos rondas de propagación de la información, para esa ronda

no se obtiene medida. Como solución, en [Dim17] se va incrementando el valor del TTFB en un valor igual al tiempo de ronda Vivaldi hasta que se consigue obtener una nueva medida. Uno de los problemas de este estimador es que se asume que el cliente realiza peticiones HTTP de forma secuencial, algo poco realista. Por otro lado, en [Dim17] se comprueba experimentalmente que este estimador es malo en términos de valor absoluto, si bien hay cierta correlación entre las medidas de este estimador y las medidas de carga.

Para elegir el mejor *proxy*, no sólo nos interesa su tiempo de respuesta, sino todo el tiempo que transcurre en la comunicación desde el cliente al servidor pasando por el *proxy*. De nuevo, para obtener información de carácter agnóstico en cuanto a métricas de carga internas del *proxy* o de velocidad de conexión de éste con el servidor, el uso del TTFB puede ser una buena métrica. Pero el sondeo realizado por los clientes al *proxy* debe tener las mismas características en todos, por ejemplo mediante *pings* al *proxy* o solicitando recursos que se encuentren cacheados y del mismo tamaño en todos los casos. Una forma de agilizar el proceso de elección de *proxy* es mediante la estimación del TTFB en los clientes a los *proxies* que no sondea en la correspondiente ronda.

2.3. Datos

Los datos utilizados, como se ha indicado anteriormente, provienen de la red Guifi.net. Para tomar las medidas se instrumentaron cuatro clientes, si bien no fue necesario realizar cambios en los *proxies*. Se realizaron sondeos a 13 *proxies* distintos con un periodo de 10 segundos, que es el periodo de la ronda Vivaldi. Cada muestra obtenida es almacenada en un fichero de *log* con información sobre el instante en que se tomó la muestra en forma de sello temporal Unix, dirección IP del *proxy* al que se sondea, el TTFB medido, el tiempo total incluyendo el RTT, y el código HTTP recibido. La primera muestra se tomó el 16 de marzo de 2018 a las 16:28:24, y la última el 19 de marzo de 2018 a las 11:55:00. En total se recogieron 1.134.888 muestras. Sin embargo, teniendo en cuenta los instantes de inicio y fin de toma de muestras, y el tiempo transcurrido, se deberían haber tomado en torno a 23.920 muestras por cada pareja cliente-*proxy*, es decir, un total de 1.243.840 muestras. Esta diferencia se puede deber a instantes de mal funcionamiento de los *proxies* o de los clientes utilizados en la obtención de muestras. Para la realización de los experimentos de estimación y predicción, asumiremos que en esos instantes no hay muestras y por tanto no se actualizará el modelo utilizado. Además hay que destacar que en los datos recibidos, el TTFB medido por el *proxy* 8 es siempre igual a 0, por lo que se decide eliminarlo de los datos y mantener el identificador empleado en el resto de *proxies*.

En la tabla 2.1 se puede ver el número de muestras recogidas por pareja cliente-*proxy*, además del total recogido por cliente, por *proxy* y el número de muestras recogidas global. Se puede ver cómo el número de muestras recogidas se mantiene constante en cada cliente para cada uno de los *proxies*, y que para cada *proxy*, el número de muestras tomadas depende del cliente que las toma.

	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Total por proxy
Proxy 1	23.632	23.632	23.638	23.672	94.574
Proxy 2	23.632	23.632	23.638	23.672	94.574
Proxy 3	23.632	23.632	23.638	23.672	94.574
Proxy 4	23.632	23.632	23.638	23.672	94.574
Proxy 5	23.632	23.632	23.638	23.672	94.574
Proxy 6	23.632	23.632	23.638	23.672	94.574
Proxy 7	23.632	23.632	23.638	23.672	94.574
Proxy 8	23.632	23.632	23.638	23.672	94.574
Proxy 9	23.632	23.632	23.638	23.672	94.574
Proxy 10	23.632	23.632	23.638	23.672	94.574
Proxy 11	23.632	23.632	23.638	23.672	94.574
Proxy 12	23.632	23.632	23.638	23.672	94.574
Proxy 13	23.632	23.632	23.638	23.672	94.574
Total por cliente	307.216	307.216	307.294	307.736	1.229.462

Tabla 2.1: Número de muestras recogidas por pareja cliente-*proxy*

Para ver cómo se producen las pérdidas, en las figuras 2.1 a 2.4 se muestra el tiempo transcurrido entre muestras sucesivas del TTFB medido en cada cliente para todos los *proxies*. Se observa que en la mayoría de ocasiones, el tiempo entre muestras se encuentra en torno a 10 segundos. Sin embargo, hay picos que superan este valor y que indican que se han producido pérdidas. En estas figuras se puede ver que las pérdidas se producen de forma simultánea en cada uno de los clientes para todos los *proxies*, lo que podría indicar un mal comportamiento de los clientes. Además, hay instantes en los que hay pérdidas en los cuatro clientes a la vez, como ocurre en torno al final. Para apoyar esta última idea se obtiene la figura 2.5, que muestra lo mismo que las otras pero superponiendo el tiempo entre muestras tomadas por los cuatro clientes para un sólo *proxy*. Se puede ver cómo efectivamente hay instantes en que se producen pérdidas para todos los clientes y *proxies* a la vez.

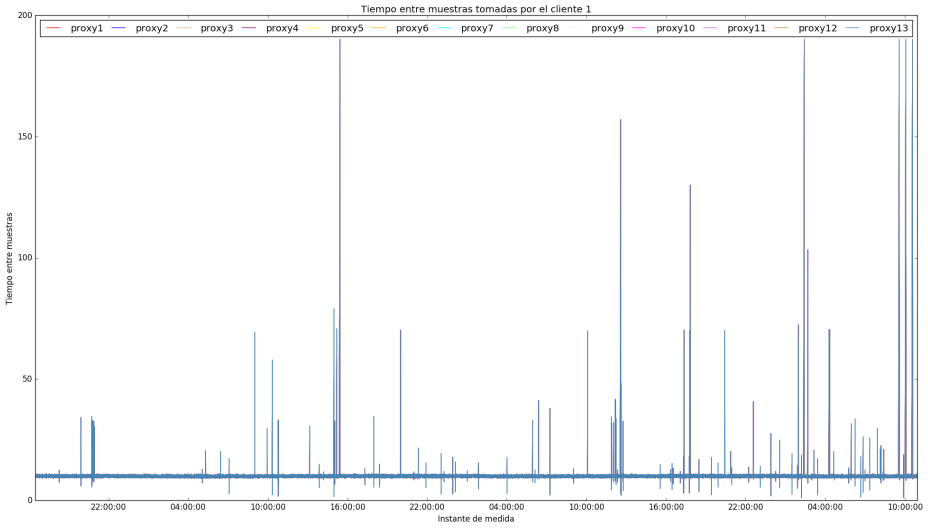


Figura 2.1: Tiempo transcurrido entre muestras sucesivas desde el cliente 1

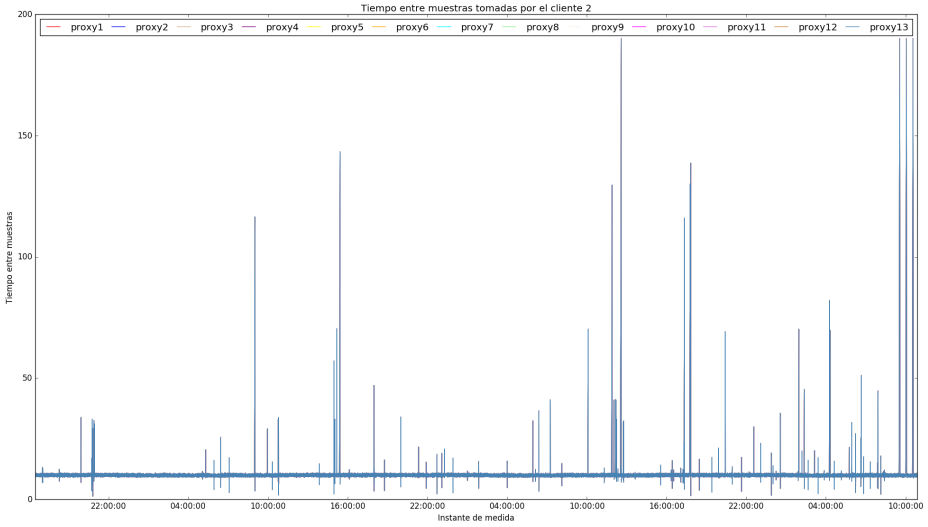


Figura 2.2: Tiempo transcurrido entre muestras sucesivas desde el cliente 2

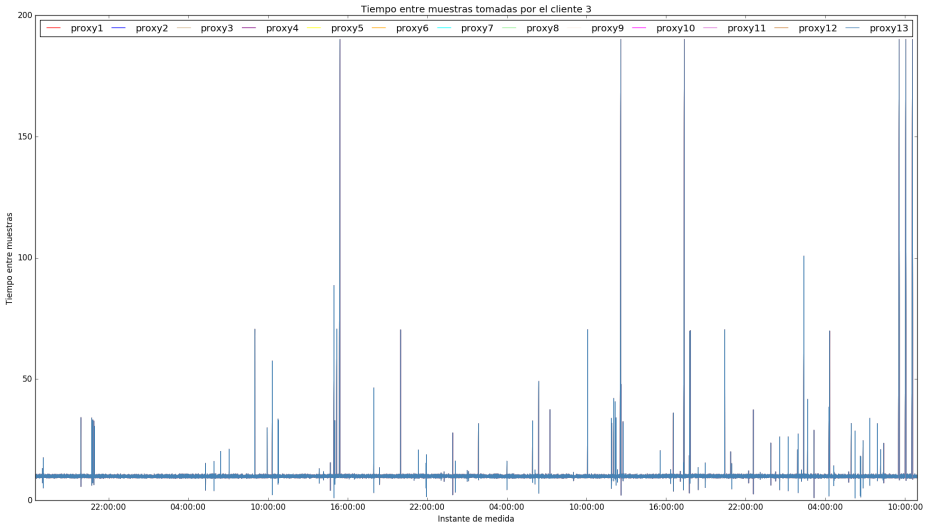


Figura 2.3: Tiempo transcurrido entre muestras sucesivas desde el cliente 3

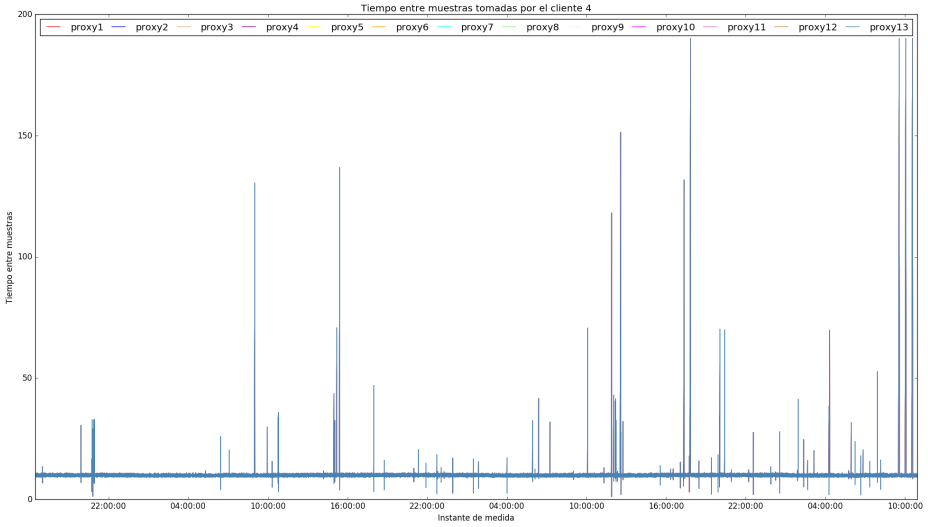


Figura 2.4: Tiempo transcurrido entre muestras sucesivas desde el cliente 4

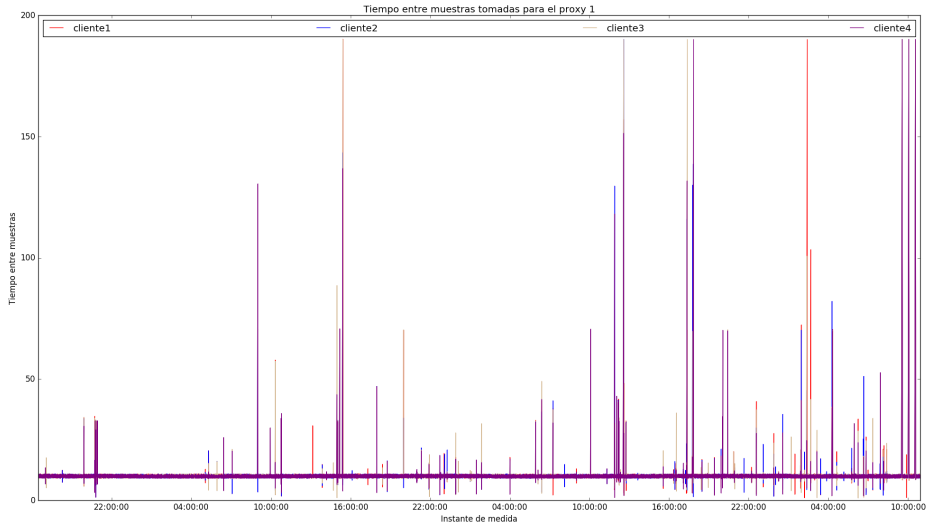


Figura 2.5: Tiempo transcurrido entre muestras sucesivas para el *proxy* 1

Para ver qué distribución tiene el global de las muestras recogidas, se puede obtener un diagrama de cajas como el de la figura 2.6. Cabe destacar que para la obtención de este diagrama de cajas se han obviado los *outliers* o datos atípicos en la representación. Sin embargo, la existencia de estos valores grandes explica el hecho de que el valor medio se encuentre fuera del rango entre cuartiles.

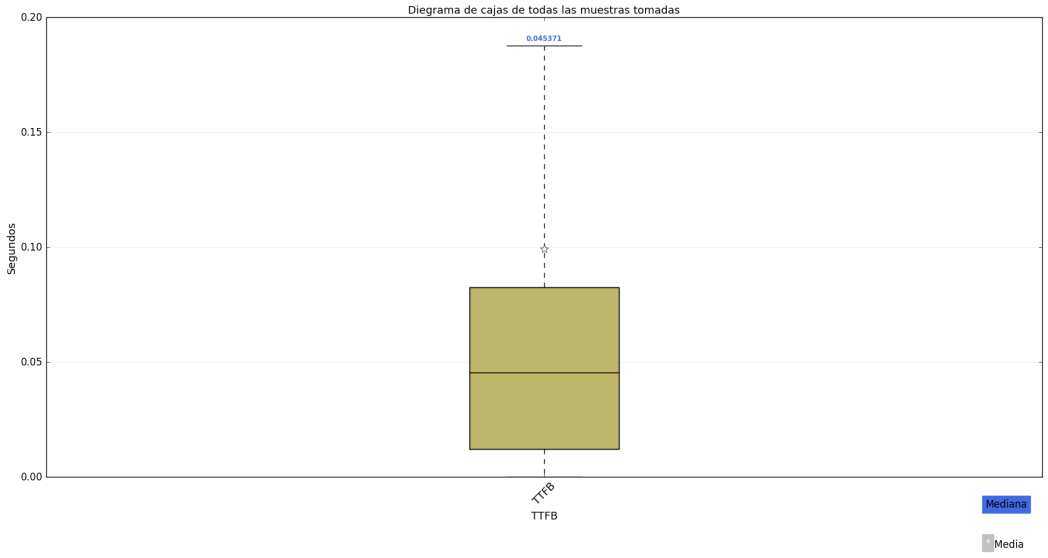
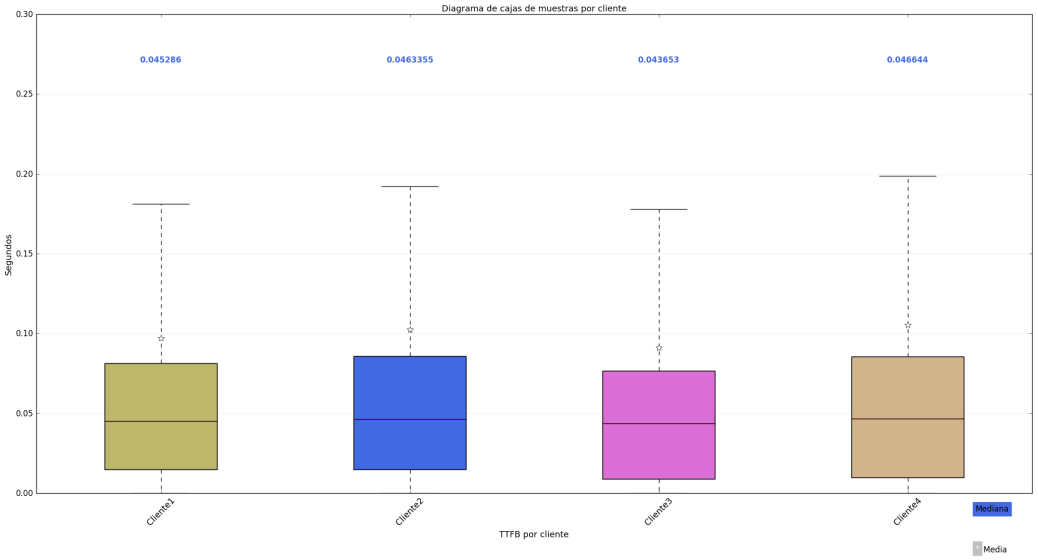


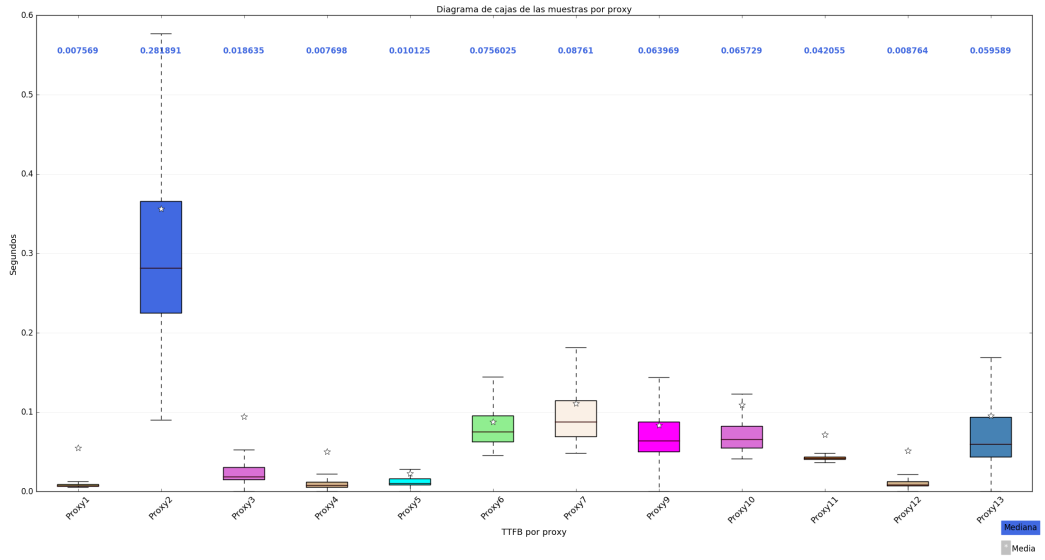
Figura 2.6: Diagrama de cajas del total de muestras.

Repitiendo el procedimiento para las muestras recogidas por cada cliente y para cada *proxy* se obtiene la figura 2.7. Se puede observar que los rangos en los que se encuentran los valores, obviando los datos atípicos, son parecidos para los cuatro clientes, y que

el valor medio también es similar. Para el caso de los *proxies* el resultado es algo más complejo. Se puede ver cómo los valores de TTFB del *proxy* 2 son más elevados que para el resto, lo que podría indicar una peor conexión a dicho *proxy* o una mayor carga durante el periodo de medida. Sin embargo, los valores en general son muy parecidos y bajos, una vez más obviando los datos atípicos. Al igual que en el caso de las muestras por cliente, el valor medio supera el rango entre cuartiles, y el motivo es el mismo: la existencia de valores elevados que hacen que la media crezca. En el anexo A se muestran los diagramas de cajas para cada una de las parejas cliente-*proxy*.



(a) Diagrama de cajas por cliente



(b) Diagrama de cajas por *proxy*

Figura 2.7: Diagramas de cajas de las muestras recogidas por clientes y *proxies*

De la figura 2.7a se puede extraer que los clientes reciben muestras muy parecidas, quizás debido a una gran cercanía entre ellos. En la figura 2.7b se pueden observar tres grupos de *proxies* atendiendo a sus rangos intercuartiles. El primero de ellos estaría formado por el *proxy* 2, cuyo rango entre cuartiles se encuentra por encima del resto, así como su valor medio. El siguiente grupo estaría formado por los *proxies* 6, 7, 9, 10, 11 y 13, cuyos rangos intercuartiles se superponen. Estos *proxies* podrían presentar un rendimiento intermedio. Por último, el tercer y mejor grupo estaría formado por los *proxies* 1, 3, 4, 5 y 12. En estos casos el rango intercuartil es pequeño y tiene valores bajos, si bien hay casos en los que el valor medio es relativamente elevado. De entre todos los *proxies* sondeados destacan el 2 como peor opción por sus elevados valores, y el 5 como mejor opción por sus valores más pequeños.

También es interesante observar cómo evoluciona el TTFB con el tiempo. En la figura 2.8 se puede observar dicha evolución para el *proxy* 1, y el resto de figuras se pueden ver en el anexo A. Se pueden observar ciertos picos de valores elevados motivados bien por un periodo de peor conexión, bien por un instante de mayor carga en el *proxy* al que se sondea. Igualmente se puede ver cómo en dichos instantes de TTFB elevado, todos los clientes perciben la subida en el valor medido, lo que indica cierta correlación entre medidas tomadas entre distintos clientes a un mismo *proxy* y podría apoyar la teoría de cercanía entre ellos, puesto que el valor medido por los clientes en dichos instantes es bastante parecido.

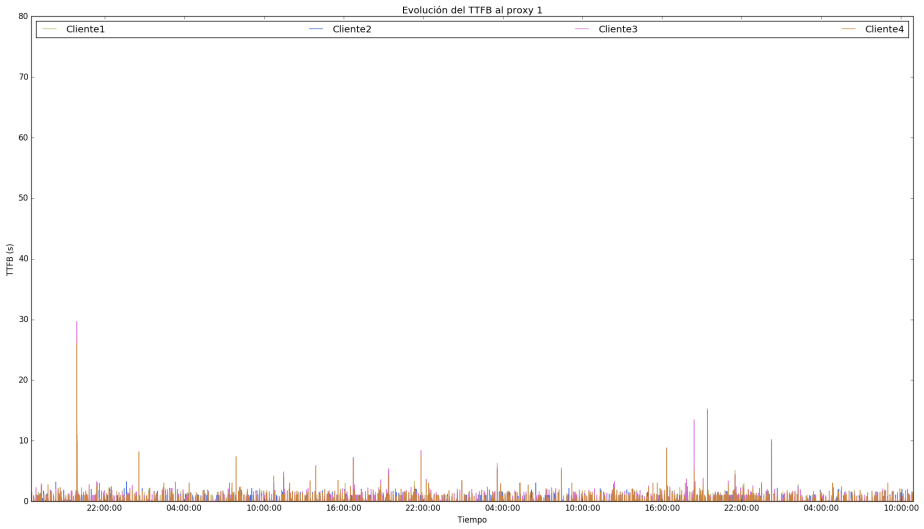


Figura 2.8: Evolución del TTFB al *proxy* 1

2.4. Algoritmo de referencia

Para comprobar el buen funcionamiento del algoritmo empleado en la predicción del TTFB se utiliza un algoritmo de referencia. Este algoritmo consiste en predecir el valor de una muestra de TTFB de la pareja cliente-*proxy* como el valor de la muestra anterior.

Se trata de un algoritmo muy sencillo que permitirá ver el error que se comete al elegir un *proxy* en función de los valores de TTFB predichos con los métodos que se presentarán más adelante en comparación con el que se comete al elegir un *proxy* en función de la última muestra de TTFB obtenida por el cliente. Para utilizarlo basta con desplazar el vector de muestras de TTFB para cada pareja cliente-*proxy* una posición a la derecha, añadiendo un cero en la posición de más a la izquierda.

2.5. Evaluación del rendimiento

El problema se tratará como un problema de regresión. Esto significa que se intentará predecir el valor exacto de TTFB, y por tanto es necesario evaluar la calidad de las predicciones con alguna métrica adecuada. Las opciones más populares son el MAE (*Mean Absolute Error*, error absoluto medio) y el RMSE (*Root Mean Square Error*, raíz del error cuadrático medio). Si N es el tamaño del vector de errores, y e_i el i-ésimo elemento de dicho vector, el MAE se define como:

$$MAE = \frac{\sum_{i=0}^{N-1} |e_i|}{N} \tag{2.6}$$

y el RMSE como:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{N-1} |e_i|^2}{N}} \tag{2.7}$$

En este trabajo se utiliza el MAE como medida de la calidad de las predicciones por dos motivos principales [JW05]. El primero de ellos es la facilidad de su cálculo comparado con el cálculo del RMSE. El MAE requiere sumar la magnitud de los errores para obtener el error total y a continuación dividirlo entre el número de muestras, mientras que para calcular el RMSE hay que calcular la suma de los errores elevados al cuadrado, dividirlo entre el número de muestras total y obtener su raíz cuadrada. El segundo motivo es el hecho de que en el RMSE los errores grandes influyen al total en proporción a su cuadrado, con lo que su influencia es mayor que la de los errores pequeños. Debido al carácter ruidoso del TTFB, y a la presencia de picos difíciles de predecir, el uso del RMSE podría dar lugar a una medida de la calidad afectada en gran medida por los errores grandes que muy probablemente se producirán al realizar las predicciones.

Nótese que el MAE del algoritmo de referencia puede obtenerse fácilmente obteniendo la resta del vector de muestras original y el vector desplazado. A continuación se obtiene la suma de los valores absolutos de los elementos del vector resultado, y se divide entre su longitud.

Por otra parte, a la hora de evaluar los métodos de aprendizaje automático es habitual dividir el conjunto de datos en dos subconjuntos, uno de entrenamiento utilizado para ajustar los parámetros del modelo, y otro de test, sobre el que se calculan los errores. Sin embargo, en este trabajo se utilizan métodos de aprendizaje en línea, caracterizados por ser incrementales, por las características dinámicas de nuestro problema y por la gran cantidad de datos que se deberán manejar. Esto elimina la necesidad de dividir los datos en un conjunto de entrenamiento y otro de test, puesto que se evalúa el modelo existente con la llegada de cada una de las muestras y a continuación se actualiza o entrena dicho modelo con esta nueva muestra.

2.6. Conclusiones

A partir de la definición del problema de selección de *proxy* se puede destacar la importancia de obtener un método que permita al cliente elegir de forma automática el *proxy* que mejore la calidad del servicio percibido por éste. Además, se puede decir que el TTFB es una buena medida de la carga del *proxy* si bien no en términos de valor absoluto, sí en términos de determinar cuál es el mejor *proxy* en cada instante. Es aquí donde surge la posibilidad de predecir el TTFB.

A partir de el análisis de los datos se observa que la medida del TTFB presenta un comportamiento altamente ruidoso y aleatorio, lo que dificulta en principio la tarea del predictor, pero que abre nuevos caminos alternativos a la estimación del TTFB como un problema de regresión. También cabe destacar que los datos tomados por los diferentes clientes son muy parecidos, de forma que se podría aumentar el interés del trabajo realizado escogiendo clientes con características más diferentes.

Capítulo 3

Estimación de TTFB mediante filtrado colaborativo

3.1. Introducción

Los grandes proveedores web de contenido como películas, series, música o libros, junto con los grandes comercios en línea ofrecen infinidad de productos. Esta gran variedad de opciones, junto con su carácter dinámico y heterogéneo [Kor09, Lop11] hacen difícil que los usuarios encuentren lo que buscan. Una consecuencia directa de esta gran cantidad de posibilidades es una baja satisfacción de los usuarios al adquirir unos productos que realmente no cubren sus necesidades [Ric15]. Los sistemas de recomendación pueden ser buenos aliados en la toma de decisiones de este tipo [Res97]. De hecho, conseguir emparejar los usuarios con los artículos más apropiados para ellos permite un alto grado de satisfacción de los clientes y una mayor fidelidad [Kor09]. Normalmente un sistema de recomendaciones que ofrezca recomendaciones rápidas y precisas atraerá el interés de los consumidores, lo que favorecerá un incremento de los beneficios reportados [Su09].

Un ejemplo de sistema de recomendación es el utilizado por la plataforma de vídeo bajo demanda Netflix [Agg16]. Netflix permite a los usuarios puntuar películas y series en una escala de 5 puntos, y almacena información sobre los artículos que han sido vistos por un determinado usuario. Todos estos datos son utilizados para realizar recomendaciones sobre otros artículos. Otro ejemplo es el sistema de recomendación utilizado por la web de ventas *online* Amazon.com [Agg16], pionera en este campo, donde las recomendaciones realizadas se basan tanto en las valoraciones de los usuarios como en sus patrones de compra y búsqueda.

Un sistema de recomendación se puede describir como un sistema de procesado de información que reúne varios tipos de datos para construir sus recomendaciones [Ric15]. Su función es obtener un perfil de las preferencias de los usuarios sobre los artículos, además de modelar la relación entre usuarios y artículos [Tak09], para poder realizar recomendaciones. Los artículos son los objetos que son recomendados, y se pueden caracterizar por su complejidad y su valor o utilidad. En el ejemplo de Netflix, los artículos son las películas y series. Los usuarios, por su parte, pueden tener una serie de características que los describen, y que pueden ser utilizadas por el sistema de recomendación a la hora de

ofrecer recomendaciones. La relación entre los usuarios y los artículos viene dada por las valoraciones, que a su vez son la forma de interacción entre el usuario y el sistema de recomendación. En el ejemplo de Netflix, las valoraciones son las puntuaciones que los usuarios dan a las películas y series que han visto.

Los métodos de predicción de los sistemas de recomendación han sido utilizados ya en varios problemas relacionados con el que se aborda en este capítulo. En [Zhe17] se adopta una aproximación de filtrado colaborativo basado en usuario para obtener recomendaciones para servicios en la nube. Aquí se identifican los usuarios como los clientes de la nube, y los artículos como los servicios proporcionados. Por otro lado, en [Luo16] se utiliza un mecanismo de filtrado colaborativo basado en el coeficiente de correlación de Pearson para obtener los valores desconocidos en una matriz de valores de QoS (*Quality of Service*, calidad de servicio) donde los artículos son servicios web. En [Vad16] se presenta un sistema de recomendación basado en filtrado colaborativo que utiliza las preferencias individuales de los usuarios sobre atributos no funcionales para recomendar servicios. Por último, en [Zhu17] se propone una aproximación a la predicción en línea de QoS para ciertos servicios basada en una factorización matricial adaptativa.

En este capítulo se busca estudiar la posibilidad de abordar el problema de la estimación del TTFB como un problema de recomendación, y comprobar si se pueden utilizar los mecanismos empleados en este tipo de problemas. Para ello, se realiza un recorrido a través de los distintos tipos de sistemas de recomendaciones que permita, por un lado, ver con perspectiva la multitud de posibilidades existentes, y por el otro, seleccionar un algoritmo adecuado para la estimación del TTFB. Además, una vez elegido el algoritmo, se realizarán pruebas de estimación cuya calidad, cuantificada por el error cometido, se comparará con la de un algoritmo sencillo de referencia para evaluar en qué grado se mejoran los resultados al emplear este tipo de algoritmos en el ámbito de la predicción del TTFB.

Por ello, en la segunda sección se plantea el problema de la recomendación y se exponen los motivos por los que es razonable el uso de los mecanismos de los sistemas de recomendación en la estimación del TTFB. En la siguiente sección se realiza una aproximación descendente desde los múltiples tipos de sistemas de recomendación hasta llegar al algoritmo utilizado en la estimación del TTFB. A continuación, en la cuarta sección, se describen los datos con los que contamos, el *software* utilizado y el algoritmo de referencia con el que comparamos los resultados. En la quinta sección se muestran y discuten los resultados obtenidos y por último, en la sexta sección, se plantean las conclusiones alcanzadas.

3.2. Formulación del problema

3.2.1. El problema de la recomendación

El problema de recomendación se puede formular de la siguiente manera [Ado05]. Llamamos C al conjunto de todos los usuarios y S al conjunto de todos los artículos que pueden ser recomendados. Sea u la función de utilidad que mide la utilidad del artículo s para el usuario c. La utilidad de un artículo es normalmente representada por una valoración, que indica cuánto le gustó a un usuario un artículo en particular. Esta valoración,

que ha de ser numérica, puede ser dada por el usuario de forma explícita o deducida por el algoritmo de recomendación a partir de las acciones pasadas realizadas por dicho usuario [Kor09]. Entonces, para cada usuario *c* queremos elegir el artículo *s'* que maximice la utilidad a ese cliente:

$$\forall c \in C, s'_c = \underset{s \in S}{\operatorname{argmax}}\, u(c, s) \tag{3.1}$$

Cada elemento del espacio de usuarios se puede definir con un perfil que incluye varias características del usuario como la edad, el género, etc. En el caso más simple puede contener un único elemento como el identificador de usuario. De la misma forma cada elemento del espacio de artículos se define con un conjunto de características [Ado05]. El problema principal de los sistemas de recomendación recae en que la utilidad normalmente no está definida para todos los usuarios y artículos. Esto significa que la utilidad necesita ser extrapolada. Una vez que las valoraciones desconocidas son estimadas, se realiza la recomendación de uno o más artículos a un usuario eligiendo el o los de mayor puntuación [Ado05].

3.2.2. Estimación del TTFB como un problema de recomendación

En el problema de selección de *proxy* contamos con *C* clientes y *P proxies*. Además, cada uno de los clientes tiene información de los *proxies* que directamente sondea, junto con la información que recibe de otros clientes de los sondeos que estos realizan a los mismos *proxies* o a otros. Al plantear de esta forma el problema, se podría identificar a los clientes en el problema de selección de *proxy* con los usuarios de un sistema de recomendaciones, y a los artículos con los *proxies*. De esta forma, el TTFB medido directamente por un cliente a un *proxy* se podría interpretar como una métrica de preferencia de un cliente hacia un *proxy*, y se podría identificar con la valoración de un usuario a un artículo. Por tanto, el problema se reduciría a estimar el TTFB entre los clientes y los *proxies* como si de puntuaciones de usuarios se tratara.

Las estimaciones se generarían en cada cliente, y puesto que el periodo de actualización de la información de la que éste dispone es de 10 segundos, el modelo se actualizará con esa frecuencia. Con cada actualización, el cliente refresca la información sobre TTFB a los *proxies* que sondea tanto él como los clientes vecinos con los que se comunica. Para los experimentos contamos con un conjunto de datos en el que todos los clientes sondean a todos los *proxies*, de forma que se permita analizar el desempeño del estimador, si bien en un caso real cada cliente sólo sondea a una parte de los *proxies*. De esta forma, la predicción podría tener sentido para los *proxies* que el cliente no sondea en el momento actual pero de los que recibe información tomada desde otros clientes.

3.3. Selección de algoritmo

3.3.1. Tipos de sistemas de recomendación

En función de cómo se hace la predicción de la utilidad de una recomendación, se pueden distinguir seis clases principales de sistemas de recomendación [Ric15]:

- **Basados en contenido**: El sistema aprende a recomendar artículos que se parecen a los que le gustaron al usuario en el pasado. La similitud entre artículos se calcula basándose en las características asociadas a los artículos. Por ejemplo, si un usuario ha valorado positivamente una película que pertenece al genero de drama, el sistema puede recomendarle otras películas de este género. En los sistemas de recomendación basados en contenido, es necesario recoger información sobre cada artículo. Esto es laborioso, y más aún cuando hay hasta millones de artículos diferentes. Una ventaja de este tipo de sistemas de recomendación es que puede recomendar artículos que aún no han sido valorados. Un ejemplo de este tipo de sistemas de recomendación es LIBRA, que utiliza las descripciones de los productos de páginas de venta en línea como Amazon.com para recomendar libros [Lop11].

- **Filtrado colaborativo**: Los métodos de filtrado colaborativo producen recomendaciones basadas en los patrones de puntuación de los usuarios, sin la necesidad de información exógena sobre usuarios o artículos. La idea más simple de esta aproximación consiste en recomendar al usuario artículos que gustaron a otros usuarios con gustos similares en el pasado. Esta similitud entre gustos se calcula de acuerdo con el historial de valoraciones de los usuarios. En los sistemas de recomendación basados en filtrado colaborativo se hace uso sólo de las actividades pasadas de los usuarios, identificando relaciones entre usuarios y artículos y haciendo asociaciones mediante el uso de esta información para predecir las preferencias de lo usuarios. Un ejemplo de este tipo de sistemas de recomendación es MovieLens [Sch07], un sistema de recomendación de películas en la que un usuario puntúa del 1 al 5 las películas, y el algoritmo utiliza las puntuaciones de toda la comunidad para recomendar otras películas que podrían interesar al usuario, predecir cómo puntuaría ese usuario otras películas o realizar otras tareas.

- **Demográficos**: Este tipo de sistemas recomiendan artículos de acuerdo con el perfil demográfico del usuario como el país de procedencia, el género, el idioma o la edad. Asume que recomendaciones diferentes deberían ser generadas para diferentes nichos demográficos. Por ejemplo, algunos usuarios son enviados a diferentes páginas web en función de su idioma o país de procedencia, o se realizan sugerencias en función de la edad del usuario.

- **Basados en conocimiento**: El sistema recomienda artículos de acuerdo con el conocimiento sobre cómo ciertas características de un artículo satisfacen las necesidades y preferencias de un usuario, y cómo de útil es el artículo para el usuario. Hay dos subtipos principales, los sistemas basados en caso y los sistemas basados en restricción. Los basados en caso establecen una función de similitud para estimar cómo de bien encajan las necesidades de los usuarios con los artículos recomendados. Los basados en restricción utilizan reglas específicas sobre cómo relacionar los requerimientos de un usuario con las características de los artículos. Un ejemplo de uso de sistemas de recomendación basados en caso es el del fabricante de componentes electrónicos Analog Devices, que utiliza un sistema de recomendación de este tipo para facilitar a sus clientes un acceso más intuitivo y flexible a su catálogo de amplificadores operacionales [Bri05].

- **Basados en comunidad**: Estos sistemas recomiendan artículos teniendo en cuenta las preferencias de los amigos del usuario. La evidencia sugiere que la gente tiende a confiar más en las recomendaciones de sus amigos que en las recomendaciones de individuos similares pero anónimos. Estos tipos de sistemas de recomendación modelan y adquieren información sobre las relaciones sociales de los usuarios y de las preferencias de sus amigos.

- **Sistemas de recomendación híbridos**: se basan en la combinación de las técnicas mencionadas. Un sistema híbrido que combina dos técnicas trata de utilizar las ventajas de ambas y reducir las desventajas de cada una.

3.3.2. Adecuación de los tipos de sistemas de recomendación para la estimación de TTFB

En el apartado anterior se han descrito varios tipos de sistemas de recomendación. De entre todos ellos, los más adecuados para su uso en la estimación del TTFB son los sistemas de recomendación basados en filtrado colaborativo, ya que no requieren información sobre los usuarios y los artículos más allá de las valoraciones que los relacionan, y se puede identificar el TTFB desde un cliente a un *proxy* con las valoraciones. Además, en la sección 3.1 se presentaban varios casos de aplicación de filtrado colaborativo en situaciones parecidas al problema que nos atañe.

Por su parte, los basados en contenido requieren conocer características sobre los artículos, en nuestro caso los *proxies*, para recomendar artículos similares a los que al usuario, en nuestro caso el cliente, le gustaron en el pasado, es decir, ofrecieron un menor TTFB. Sin embargo, la obtención de esta información no es posible sin modificar los protocolos, lo que implicaría una mayor dificultad y una violación de las restricciones del problema.

El uso de sistemas de recomendación demográficos en nuestro problema no tendría sentido, ya que no se pueden agrupar los clientes de acuerdo con perfiles demográficos como los mencionados en la sección anterior. Por este motivo se descarta este tipo de sistemas de recomendación.

Los sistemas de recomendación basados en conocimiento requieren información adicional, como podría ser la ocupación de las colas, para determinar el grado en que estas características encajan con los requerimientos de los clientes. Pero esto de nuevo implica incumplir la condición de no modificar los protocolos.

Por otro lado, los sistemas de recomendación basados en comunidad deben tener en cuenta las preferencias de los amigos del usuario. En sistemas de recomendación de contenido web, por ejemplo, esta información se podría extraer de redes sociales. Sin embargo, en nuestro problema no parece lógico establecer relaciones de amistad entre los clientes más allá de las que se pueden extraer en relación a la cercanía en términos de coordenadas Vivaldi.

Por todo esto, en este capítulo se explorará la utilidad del uso de los sistemas de filtrado colaborativo para el problema de la estimación del TTFB. A continuación se profundizará un poco más sobre las distintas variantes de este tipo de sistemas de recomendación.

3.3.3. Tipos de algoritmos de filtrado colaborativo

Se puede establecer una división entre los diferentes tipos de algoritmos de filtrado colaborativo: basados en memoria y basados en modelo. Mientras que los primeros emplean las puntuaciones de los usuarios para calcular la similitud entre usuarios o artículos para realizar predicciones o recomendaciones, los basados en modelo emplean las valoraciones para estimar o crear un modelo para hacer predicciones [Su09].

3.3.3.1. Algoritmos basados en memoria

Los algoritmos de filtrado colaborativo basados en memoria utilizan la base de datos de usuarios y artículos, o una porción de esta, para generar una predicción. Cada usuario es parte de un grupo con intereses similares, de modo que identificando los vecinos de un usuario se puede realizar una predicción de preferencias sobre nuevos artículos.

La primera generación de algoritmos de filtrado colaborativo utilizaba las puntuaciones para calcular la similitud entre usuarios y artículos. Debido a su fácil implementación y efectividad, su aplicación está bastante extendida en los sistemas comerciales como Amazon.com [Su09]. Sin embargo, hay bastantes limitaciones para este tipo de técnicas de filtrado colaborativo, como es el hecho de que la similitud entre usuarios se basa en artículos comunes, y si los datos son dispersos habrá menos artículos comunes y por tanto estas técnicas dejan de ser fiables [Su09]. Dentro de esta familia de algoritmos de filtrado colaborativo se encuentran los basados en usuario y los basados en artículo.

- **Basados en usuario**

  Los sistemas de filtrado colaborativo basados en usuario predicen los intereses de un usuario de prueba en un artículo de prueba de acuerdo con información de valoraciones de usuarios con un perfil similar. Cada perfil de usuario es ordenado de acuerdo con su nivel de diferencia respecto al perfil del usuario de prueba. Las valoraciones de otros usuarios similares contribuyen más a predecir la puntuación sobre el artículo de prueba [Wan06]. Para elegir el conjunto de usuarios similares se puede utilizar un umbral o selección los mejores N usuarios en cuanto a similitud con el usuario de prueba.

- **Basados en artículo**

  En los sistemas de filtrado colaborativo basados en artículo la idea principal es la misma que en los basados en usuario, pero empleando la similitud entre artículos en lugar de entre usuarios. En este caso, las puntuaciones conocidas de un artículo de prueba realizadas por un usuario de prueba se pueden predecir promediando las puntuaciones de otros artículos similares puntuados por dicho usuario [Wan06]. De nuevo, se ordenan los artículos de acuerdo con su nivel de diferencia respecto al artículo de prueba, y las puntuaciones de los artículos más similares son ponderadas con más peso [Wan06].

3.3.3.2. Algoritmos basados en modelo

Este tipo de algoritmos se basan en la idea del *factor latente*, que trata de explicar las puntuaciones caracterizando tanto a los usuarios como a los artículos con una serie de

factores inferidos a partir de la forma de puntuar de los usuarios [Kor09, Han11]. Estos *factores latentes* se pueden interpretar como relaciones entre distintas puntuaciones. El uso de modelos como los empleados en aprendizaje automático puede permitir al sistema de recomendaciones reconocer patrones complejos basados en los datos de entrenamiento que le permitan hacer predicciones inteligentes para los datos de prueba o del mundo real [Su09]. Se pueden emplear algoritmos de clasificación si las puntuaciones son categóricas, o modelos de regresión y SVD si las puntuaciones son numéricas. En este tipo de algoritmos de filtrado colaborativo destacan los algoritmos basados en agrupamiento o *clustering*, los basados en reglas de asociación, los basados en factorización matricial, los basados en máquinas de Boltzman restringidas y los basados en redes neuronales recurrentes.

- **Clustering**

  Estos algoritmos tratan de separar los datos en grupos o *clusters* de forma que los objetos pertenecientes a un grupo son similares entre sí, y diferentes a los del resto. Las diferencias y similitudes entre objetos se basan en atributos cuantificables que los describen y que se calculan típicamente como medidas de distancia [Han11, Su09]. Una vez definidos los grupos y clasificados los objetos, se recomiendan aquellos que pertenecen al mismo grupo que, por ejemplo, otro objeto bien puntuado por el usuario. Dado que la partición no es realizada por seres humanos sino por un algoritmo, se pueden descubrir grupos desconocidos en los datos.

- **Reglas de asociación**

  Se basan en un tipo de técnica de mineria de datos (*data minig*) que consiste en descubrir asociaciones y correlaciones entre artículos en un conjunto de datos grande de transacciones comerciales o relaciones de otro tipo. Con grandes cantidades de datos continuamente recolectados y almacenados, surge la posibilidad de buscar patrones de relación o correlación entre dichos datos. El descubrimiento de relaciones interesantes entre grandes cantidades de registros puede ayudar en muchas tomas de decisiones [Han11]. Un ejemplo típico es el análisis de las cestas de la compra, en el que el análisis de los hábitos de compra de los usuarios encontrando asociaciones entre diferentes artículos permite a los vendedores desarrollar estrategias de marketing basadas en el conocimiento sobre, por ejemplo, qué artículos son comprados juntos por los clientes [Han11].

- **Factorización matricial**

  La factorización matricial consiste en descomponer de la matriz de valoraciones, en la que cada fila es el vector de puntuaciones de cada usuario y cada columna es el vector de puntuaciones recibidas por cada artículo, en dos matrices que contengan las características (factores latentes) de cada usuario y de cada artículo por separado. De esta forma, se puede estimar la puntuación de cada usuario a todos los artículos del sistema. Más adelante se describirá este tipo de algoritmos con más detalle.

- **Máquina de Boltzmann restringida**

  Las máquinas de Boltzmmann restringidas (RBM, *Restricted Boltzmann Machines*) son modelos gráficos probabilísticos que pueden ser interpretados como redes

neuronales estocásticas [Fis12]. Las máquinas de Boltzman se pueden utilizar para aprender aspectos importantes de una distribución de probabilidad desconocida a partir de muestras de dicha distribución. Están formadas por dos tipos de unidades: visibles y ocultas. Las unidades visibles constituyen la primera capa y corresponden a las componentes de una observación, por ejemplo una unidad visible para cada píxel de una imagen digital. Las unidades ocultas, por su parte, modelan las dependencias entre las componentes de las observaciones, que el ejemplo anterior corresponderían a las dependencias entre los pixeles de la imagen. Puesto que los factores latentes son relaciones entre distintas puntuaciones, se pueden identificar las puntuaciones realizadas por los usuarios con las unidades visibles, y los factores latentes que relacionan distintas puntuaciones con las unidades ocultas.

- **Redes neuronales recurrentes**

  Se trata de un tipo de redes neuronales [Isa04] caracterizadas por tener bucles, de forma que pueden aparecer conexiones de una neurona con ella misma, conexiones entre neuronas de una misma capa o conexiones de las neuronas de una capa con la capa anterior. Estas redes tienen un comportamiento temporal, que facilita el tratamiento de patrones dinámicos, en los que el valor en un determinado instante depende de sus valores en instantes anteriores de tiempo. Esto puede ayudar a la hora de recomendar un artículo en un instante de tiempo mediante el uso de la información adicional sobre los artículos recomendados o valorados por el usuario en instantes anteriores.

3.3.4. Adecuación de los tipos de algoritmos de filtrado colaborativo para la estimación del TTFB

En el apartado anterior se presentan distintos tipos de algoritmos de filtrado colaborativo. En cuanto a los basados en modelo, el filtrado colaborativo mediante agrupamiento no es útil en este caso, puesto que únicamente agrupa los artículos en grupos con características similares inferidas de las valoraciones. Por su parte, el filtrado colaborativo basado en reglas de asociación busca correlaciones entre los artículos de una transacción, algo que tampoco es útil en la estimación del TTFB. Los métodos basados en RBM son parecidos a los de factorización matricial, y los basados en redes neuronales recurrentes permiten modelar también el comportamiento temporal de las puntuaciones, si bien la factorización matricial presenta un menor error en la predicción [Bok15].

En cuanto a los métodos basados en memoria, obtienen mejor resultado que los basados en modelo cuando la matriz es muy densa, pero funcionan peor en matrices de puntuaciones dispersas [Cac11]. Además como pasaba en los métodos de agrupamiento y de reglas de asociación, no buscamos artículos similares sino el artículo con mejor puntuación.

En este capítulo se explora el uso de algoritmos de factorización matricial para su aplicación al problema de la estimación del TTFB, puesto que presentan buen rendimiento para matrices muy grandes y muy dispersas [Kor09], como podría ser nuestro caso. A continuación se formula el problema de la factorización matricial y se presentan algunos algoritmos.

3.3.5. Factorización matricial

En el modelo básico de factorización matricial, de la matriz de puntuaciones *R* de tamaño $m \times n$ se obtienen por factorización las matrices *P* y *Q* de tamaños $m \times k$ y $k \times n$ respectivamente, de forma que:

$$R \approx PQ$$

Cada columna de la matriz *P* y cada fila de la matriz *Q* se conoce como *vector latente*, mientras que cada fila de *P* y cada columna de *Q* es un *factor latente*. Nos referimos a la u-ésima fila de *P*, $\mathbf{p_u}$, como *factor de usuario*, donde cada elemento *k* contiene la afinidad del usuario a *k* diferentes conceptos en los que se puede dividir la matriz de puntuaciones *R*. De manera análoga, la i-ésima columna de la matriz *Q*, $\mathbf{q_i}$, se conoce como *factor de artículo* y contiene la afinidad del i-ésimo artículo a los *k* conceptos correspondientes de la matriz *R*.

Para explicar qué es un concepto podemos utilizar un ejemplo. Imaginemos una matriz con valoraciones de varios usuarios a varias películas. Estos conceptos serían, por ejemplo, los géneros a los que pertenece cada película. Así, cuando hablamos de afinidad de un usuario a un concepto hablamos de cuánto le gusta a este usuario determinado género, y cuando hablamos de afinidad de un artículo a un concepto hablamos de en qué grado pertenece la película correspondiente a un género. Por tanto, cada puntuación r_{ui} de la matriz *R* se puede aproximar como, [Agg16]:

$$r_{ui} \approx \mathbf{p_u q_i}$$

Una forma más intuitiva de expresar esta aproximación es:

$$r_{ui} \approx \sum_{s=1}^{k} p_{us} \cdot q_{si}$$

Es decir, se aproxima la valoración que realizaría el usuario *u* sobre el artículo *i* como la suma de los productos de las afinidades del usuario *i* y el artículo *j* a cada uno de los *k* conceptos que se pueden distinguir en la matriz *R*. Cabe destacar que estos conceptos no son siempre tan intuitivos como en el caso de la matriz de valoraciones de películas, y además no siempre son interpretables semánticamente, si bien siempre representan un patrón de correlación dominante en la matriz de puntuaciones [Agg16].

Las diferencias clave entre los diversos métodos de factorización matricial radican en las restricciones impuestas a las matrices *P* y *Q*, como la ortogonalidad o la no negatividad de los vectores latentes, así como en la naturaleza de la función objetivo a optimizar [Agg16].

3.3.6. Algoritmos de factorización matricial

Existen multitud de algoritmos de filtrado colaborativo basados en factorización matricial. Algunos de ellos son:

- **SVD**

  Se trata de una técnica que factoriza la matriz de puntuaciones en tres matrices. Este método de factorización proporciona la mejor aproximación de menor rango

de la matriz de puntuaciones en términos de norma de Frobenius. En [Sar00] se plantea una aproximación a este tipo de factorización, en la que se utiliza SVD para producir una representación de menor dimensión del espacio original de usuarios y artículos y capturar las relaciones latentes entre los usuarios y los artículos. También se computa la relación de vecindad en el espacio reducido para generar una lista de *N* mejores productos que recomendar.

- **BRISMF**

  BRISMF (*Biased Regularized Incremental Simultaneous Marix Factorization*, Factorización de Matrices Simultánea Incremental e Insesgada) es un algoritmo de factorización matricial que se plantea en [Tak09]. Se realiza una descomposición de la matriz de puntuaciones, que es muy dispersa, en dos matrices de menor rango que representan los factores latentes de usuarios y artículos. Esta factorización se hace de forma incremental, mediante un algoritmo de descenso de gradiente. Los detalles concretos se presentarán más adelante.

- **Factorización matricial probabilística**

  Se puede ver como una extensión probabilística del modelo SVD. El desarrollo se explica en [Sal07], y consiste en establecer una función de distribución de probabilidad de la matriz de puntuaciones condicionada a las matrices de características de usuario y artículo, que se modela como un modelo lineal con ruido Gaussiano de observación. Se obtiene la función de distribución a posteriori, y se minimiza el logaritmo neperiano, lo que es equivalente a minimizar la suma de errores cuadráticos, que es la función objetivo.

- **SVD++**

  Mientras que los algoritmos basados en factorización SVD sólo pueden manejar información explícita como las puntuaciones de un usuario, existe también información implícita como su historial de búsqueda que puede aportar conocimiento a la hora de realizar recomendaciones. En este sentido, en [Kor08] se desarrolla un algoritmo que permite tener en cuenta este tipo de información. Para ello se altera la formulación del problema de factorización matricial de forma que se incluya la información implícita de la que hablamos. Además, se representa cada artículo mediante tres vectores, y no se realiza una parametrización explícita de los usuarios, sino que son caracterizados por los artículos que prefieren. Esto implica una mayor facilidad para manejar nuevos usuarios en cuanto estos comienzan a puntuar en el sistema.

3.3.7. Adecuación de los algoritmos de factorización matricial para la estimación del TTFB

En el apartado anterior se presentan algunos algoritmos de factorización matricial. Mientras que BRISMF es una mejora del algoritmo SVD básico, la factorización matricial probabilística no se podría emplear en nuestro caso, en el que no conocemos *a priori* ningún modelo lineal ni la distribución probabilística para el TTFB medido. Por otro

lado, el uso de información implícita, aunque posible, sería muy costoso y puesto que la evolución del TTFB no parece seguir ningún tipo de patrón temporal la información sobre los *proxies* elegidos anteriormente por un cliente podría no ser de gran ayuda a la hora de predecir el TTFB. Por este motivo se descarta SVD++.

En BRISMF, el uso del algoritmo de descenso de gradiente permite explotar la información de orden cronológico, además de ser más adecuado para problemas con matrices de gran tamaño. Por otro lado, el uso de una versión semipositiva de la factorización matricial permite acelerar el proceso de entrenamiento sin deteriorar la precisión [Tak09]. Por todo esto, el algoritmo elegido para la estimación del TTFB es BRISMF. A continuación se explica el funcionamiento de dicho algoritmo, y se realizan los experimentos de predicción exponiendo los resultados obtenidos.

3.3.8. BRISMF

El algoritmo BRISMF permite resolver el problema de filtrado colaborativo de forma eficiente y con bajo error. En primer lugar se explica el algoritmo básico de factorización de matrices, a partir del cual se desarrolla BRISMF en [Tak09].

Como indicamos en la sección 3.3.5, llamamos *R* a la matriz de puntuaciones, y buscamos aproximar la matriz *R* por el producto de dos matrices *P* y *Q* de forma que $R \approx PQ$. Aquí *P* sería la matriz de características de los usuarios, y *Q* la de los artículos, y contendrían los vectores latentes de usuarios y artículos respectivamente. El principal problema al que se enfrenta la factorización matricial es el hecho de que la matriz *R* contiene elementos que no son conocidos. En estos casos se trata de estimar dichas puntuaciones. Si $\mathbf{p}_u$ es un vector fila de *P* que contiene las características del usuario *u* , y $\mathbf{q}_i$ un vector columna de *Q* que contiene las características del artículo *i*, la puntuación del usuario *u* al artículo *i* se estima como:

$$\widehat{r}_{ui} = \mathbf{p}_u \mathbf{q}_i \tag{3.2}$$

Definiremos el error de entrenamiento en la puntuación (u,i) como:

$$e_{ui} = r_{ui} - \widehat{r}_{ui} \tag{3.3}$$

La suma de errores cuadráticos de entrenamiento se puede expresar como:

$$SSE = \sum_{u,i} e_{ui}^2 \tag{3.4}$$

Y el error cuadrático medio, que será el que tratemos de minimizar será:

$$RMSE = \sqrt{SSE/\mid T \mid} \tag{3.5}$$

Siendo T el conjunto de pares usuario-artículo para los que existe una puntuación. Para minimizar el RMSE se puede emplear un mecanismo de aprendizaje automático conocido como descenso de gradiente que permite encontrar un mínimo local de la función a minimizar RMSE, o equivalentemente, SSE. Para simplificar esta optimización podemos definir SSE' como la mitad de la suma de errores cuadráticos de entrenamiento:

$$SSE' = \frac{1}{2} SSE \tag{3.6}$$

Para minimizar el RMSE se puede aplicar un algoritmo incremental de descenso de gradiente simple a SSE', donde cada paso trata de disminuir el error de predicción de una única puntuación,e_{ui}^2, en el sentido opuesto al de máximo crecimiento. Se puede definir:

$$e'_{ui} = \frac{1}{2} e_{ui}^2 \tag{3.7}$$

de modo que:

$$SSE' = \sum_{u,i} e'_{ui} \tag{3.8}$$

Por tanto buscamos:

$$(P^*, Q^*) = \underset{(P,Q)}{\operatorname{argmin}} RMSE = \underset{(P,Q)}{\operatorname{argmin}} SSE = \underset{(P,Q)}{\operatorname{argmin}} SSE' \tag{3.9}$$

Decimos que el algoritmo de descenso de gradiente empleado es incremental porque se actualizan los pesos en la dirección opuesta al gradiente. Esto es, para cada ejemplo de entrenamiento (u, i), se cambian los pesos en las matrices *P* y *Q* de manera simultánea para reducir el cuadrado del error actual, y conseguir una mejor aproximación a la valoración real. De esta forma se pueden encontrar las matrices *P* y *Q*, y una vez se ha acabado el entrenamiento se obtienen cada uno de los valores de *R* a partir de la fórmula 3.2. Este algoritmo de factorización matricial se conoce por la siglas ISMF (*Incremental Simultaneous Matrix Factorization*, Factorización Matricial Incremental Simultánea).

Esta factorización de matrices puede sufrir de sobreajuste para usuarios con pocas valoraciones [Tak09]. El sobreajuste es un problema que se da cuando un algoritmo de aprendizaje automático se ajusta demasiado bien a los ejemplos de entrenamiento, pero no es capaz de generalizar correctamente ante nuevos casos. Es un problema habitual cuando hay pocos ejemplos de entrenamiento. Una forma común de solucionar el sobreajuste es el uso de regularización, que penaliza el cuadrado de la norma Euclídea de los pesos [Tak09]. De la aplicación de esta penalización surge un nuevo problema de optimización:

$$e'_{ui} = (e_{ui}^2 + \lambda \cdot \mathbf{p_u} \cdot \mathbf{p_u^T} + \lambda \cdot \mathbf{q_i^T} \cdot \mathbf{q_i})/2 \tag{3.10}$$

donde λ es el factor de regularización. SSE' se define igual que en 3.8, pero la ecuación 3.6 ya no se cumple, y minimizar SSE' no equivale a minimizar SSE si $\lambda \neq 0$. Este nuevo algoritmo de factorización matricial se llama RISMF donde la R hace referencia a la regularización.

Otro problema al que se enfrenta ISMF es el hecho de que un usuario pueda realizar puntuaciones que estén por encima o por debajo de la media. De igual forma un artículo puede ser muy popular. Una forma de extender el algoritmo RISMF [Tak09] para modelar este fenómeno consiste en extender la factorización añadiendo sesgos en usuarios y artículos. Para ello, se fijan la primera columna de *P* y la segunda fila de *Q* a un valor constante de 1. Este método se denomina BRISMF, y consigue acelerar la fase de entrenamiento obteniendo un mejor resultado a la hora de generalizar.

El algoritmo de descenso de gradiente empleado en RISMF tiene una desventaja importante, y es que las características de los artículos pueden cambiar. Si el cambio es significativo, las características de los usuarios al inicio de una iteración podría ser inapropiadas al final de la misma. En [Tak09] se plantean dos formas de resolverlo:

- Evaluar las valoraciones de prueba de un usuario inmediatamente después de iterar entre las valoraciones del conjunto de entrenamiento, y antes de iterar entre las valoraciones del siguiente usuario.

- Volver a calcular las características del usuario después del proceso de aprendizaje. Este método puede servir de método de entrenamiento incremental para un sistema de recomendación.

Mediante este segundo método se pueden incorporar de forma eficiente nuevos usuarios y valoraciones de los usuarios existentes sin necesidad de volver a entrenar el modelo completo. Sin embargo, no soporta la adición de artículos, y después de la adición de muchas nuevas valoraciones la matriz Q estará obsoleta y habría que volver a entrenar el modelo completo.

3.4. Marco experimental

3.4.1. Entorno experimental

En los experimentos de predicción se utiliza el *software* proporcionado por MOA [1] (*Massive Online Analysis*, análisis masivo en línea) en su versión 2017.06b. Se trata de un entorno de desarrollo que permite utilizar algoritmos y realizar experimentos de aprendizaje en línea. Contiene una amplia colección de algoritmos tanto en línea como fuera de línea para realizar tareas de clasificación y agrupación, así como herramientas de evaluación de algoritmos de minería de datos (*data mining*) en grandes conjuntos de datos [Bif10].

MOA ofrece bibliotecas con una API (*Application Programming Interface*, interfaz de programación de aplicaciones) pública, de forma que el usuario puede programar sus experimentos mediante su uso. Adicionalmente, MOA ofrece una interfaz gráfica, que a su vez utiliza la API, y que permite manejar de forma cómoda las funcionalidades más relevantes, si bien no es posible controlar todos los parámetros que si se pueden controlar mediante el uso directo de la API. Para su uso, en primer lugar se elige la fuente de datos, a continuación se selecciona el algoritmo de aprendizaje, y finalmente se escoge el método de evaluación para analizar su desempeño en la tarea deseada [Bif10].

LA API de MOA proporciona una implementación del algoritmo BRISMF [Tak09] que se utilizará en los experimentos de este capítulo. Para la obtención de las predicciones se crea una insancia de la clase `BRISMFPredictor`. Esta clase lleva asociada una instancia de la clase `MemRecommenderData` que contiene la matriz de puntuaciones. Inicialmente se crea una instancia de cada una de estas clases, y se pasa al constructor los parámetros necesarios que caracterizan al predictor y que más adelante se describirán.

A continuación, se recorre el conjunto de datos, de forma que se almacenan en un mapa todos los TTFB medidos por un cliente en un instante concreto con ayuda del sello temporal asociado a los datos, e indexados por el identificador del *proxy*. En este punto se obtienen predicciones de TTFB para cada pareja cliente-*proxy* para los que se tienen medidas en esta iteración con el método `predictRating()` del predictor, se computa

[1] https://moa.cms.waikato.ac.nz

el error absoluto para cada una de las predicciones comparándolas con los valores reales contenidos en el fichero de datos, y se entrena el modelo de cada uno de los *proxies* y del cliente correspondiente mediante el método `setRating()` de la instancia que contienen la matriz de puntuaciones. Este proceso se repite para cada conjunto de medidas de cada cliente en cada instante de medición hasta que se llega al final del fichero que contiene las muestras. En el apéndice D se muestra el código utilizado en la realización de las predicciones.

3.4.2. Parámetros de experimentación

Para la predicción, en la creación de la instancia del predictor hay que pasar una serie de parámetros que son clave en su funcionamiento. Aunque los resultados dependen en gran medida de estos valores, se está realizando un estudio exploratorio y se eligen los parámetros que, tras una pequeña experimentación, reportan mejores resultados. En concreto estos parámetros son:

- `numFeatures`: es el número de características de los vectores de usuario y artículo. Es la dimensión interna *k* de la que se hablaba en la sección 3.3.5. En los experimentos se fija a 10.

- `lRate`: se trata del factor de aprendizaje. Establece la velocidad de descenso en dirección contraria al gradiente que se realiza para obtener las matrices P y Q. En los experimentos se fija a 0.001.

- `rFactor`: es el factor de regularización λ del que se hablaba en la sección 3.10. En los experimentos se fija a 0.02.

Además, es necesario establecer el número de iteraciones utilizado para el entrenamiento de usuarios y artículos mediante el método `setNIterations()`. En los experimentos se fija a 1000.

3.5. Resultados

Una vez realizadas las predicciones con ambos algoritmos, se computa el MAE para cada pareja cliente-*proxy*, además del MAE por *proxy*, por cliente y total. En la tabla 3.1 se puede ver el MAE obtenido para el algoritmo de referencia, y en la tabla 3.2 el obtenido con BRISMF. En las casillas de *Agregado* se indica el MAE por cliente, por *proxy* y total. En un color sombreado aparecen los casos en los que el algoritmo correspondiente obtiene un menor MAE, mientras que en negrita aparece, por cada fila, el cliente que mejor resultado obtiene.

Se puede observar que en la mayoría de casos se obtiene un mejor resultado al emplear BRISMF, si bien la mejoría no es demasiado elevada en todos. Además, el MAE agregado por cliente nos indica que BRISMF funciona peor que el algoritmo de referencia para el cliente 4, y el MAE agregado por *proxy* indica que hay *proxies* para los que el algoritmo de referencia funciona mejor que BRISMF, en concreto los *proxies* 5, 6, 9 y 11. Además, el cliente para el que se obtienen los mejores resultados es, en la mayoría de casos, el

	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Agregado
Proxy 1	0.08391	0.08672	**0.08265**	0.08804	0.08533
Proxy 2	0.19690	**0.18717**	0.19927	0.19583	0.19479
Proxy 3	0.15066	0.15982	**0.12317**	0.15905	0.14818
Proxy 4	0.08646	0.08510	**0.08438**	0.09066	0.08665
Proxy 5	0.02436	0.02379	**0.02169**	0.02658	0.02411
Proxy 6	0.03182	0.03316	**0.03040**	0.04293	0.03458
Proxy 7	0.05660	0.05832	**0.05409**	0.06695	0.05899
Proxy 9	0.04162	0.04819	**0.03809**	0.05895	0.04672
Proxy 10	0.10042	**0.08313**	0.10103	0.08380	0.09209
Proxy 11	0.05224	0.06606	**0.04506**	0.07680	0.06005
Proxy 12	0.07720	0.08257	**0.07131**	0.07930	0.07760
Proxy 13	0.06341	0.06396	**0.06065**	0.07399	0.06551
Agregado	0.08047	0.08150	**0.07598**	0.08691	0.08122

Tabla 3.1: Valores del error absoluto medio obtenidos con el algoritmo de referencia

	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Agregado
Proxy 1	0.07526	0.08115	**0.07379**	0.08652	0.07918
Proxy 2	0.16560	**0.16324**	0.16638	0.17852	0.16844
Proxy 3	0.14110	0.15081	**0.11733**	0.15481	0.14102
Proxy 4	0.08477	0.08419	**0.08143**	0.09214	0.08564
Proxy 5	0.02971	0.03131	**0.02872**	0.03971	0.03237
Proxy 6	**0.03797**	0.03876	0.03800	0.04906	0.04095
Proxy 7	0.05773	0.05804	**0.05658**	0.06697	0.05983
Proxy 9	0.04529	0.04889	**0.04438**	0.06011	0.04967
Proxy 10	0.10010	**0.08253**	0.10023	0.08412	0.09174
Proxy 11	0.06579	0.07728	**0.06123**	0.08805	0.07309
Proxy 12	0.07103	0.07665	**0.06765**	0.07839	0.07343
Proxy 13	0.06126	0.06213	**0.05931**	0.07125	0.06349
Agregado	0.07797	0.07958	**0.07459**	0.08747	0.07990

Tabla 3.2: Valores del error absoluto medio obtenidos con BRISMF

cliente 3. En la tabla 3.3 se puede observar el porcentaje de mejora del MAE, donde aparece sombreado y en positivo el caso en que el uso de BRISMF supone una mejora, y en negativo el caso contrario. También aparece resaltado en negrita, por cada fila, el cliente para el que se obtiene una mayor mejora, o un menor empeoramiento en los casos en los que el algoritmo de referencia obtiene un mejor resultado.

Si se considera que el problema de estimación es más fácil cuando el MAE es más pequeño para ambos algoritmos, en estos casos el resultado obtenido mediante el uso del algoritmo de referencia es mejor, y el porcentaje de empeoramiento puede ser realmente grande, de entorno al 25 % como es el caso del agregado en el *proxy* 5. Sin embargo, para la mayoría de casos donde el problema de predicción es más difícil (mayor MAE

	Cliente 1	Cliente 2	Cliente 3	Cliente 4	Agregado
Proxy 1	11.493 %	6.864 %	**12.007 %**	1.757 %	7.767 %
Proxy 2	18.901 %	14.659 %	**19.768 %**	9.696 %	15.644 %
Proxy 3	**6.775 %**	5.974 %	4.977 %	2.739 %	5.077 %
Proxy 4	1.994 %	1.081 %	**3.623 %**	-1.606 %	1.179 %
Proxy 5	**-18.007 %**	-24.018 %	-24.478 %	-33.065 %	-25.517 %
Proxy 6	-16.197 %	-14.448 %	-20.000 %	**-12.495 %**	-15.556 %
Proxy 7	-1.957 %	**0.482 %**	-4.401 %	-0.030 %	-1.404 %
Proxy 9	-8.103 %	**-1.432 %**	-14.173 %	-1.930 %	-5.939 %
Proxy 10	0.320 %	0.727 %	**0.798 %**	-0.380 %	0.382 %
Proxy 11	-20.596 %	-14.519 %	-26.409 %	**-12.777 %**	-17.841 %
Proxy 12	**8.686 %**	7.723 %	5.410 %	1.161 %	5.679 %
Proxy 13	3.510 %	2.945 %	2.259 %	**3.846 %**	3.182 %
Agregado	**3.206 %**	2.413 %	1.864 %	-0.640 %	1.652 %

Tabla 3.3: Tabla con el porcentaje de mejora en términos de MAE en el uso de BRISMf respecto del algoritmo de referencia

para ambos algoritmos), como el caso del *proxy* 2 en el que el MAE se encuentra en torno a 0,2 para el algoritmo de referencia, BRISMF presenta mejoras en algunos casos considerables, como en el agregado de este *proxy* donde se mejora en torno a un 15 % el resultado respecto al algoritmo de referencia.

3.6. Discusión y Conclusiones

En este capítulo hemos realizado un recorrido por los distintos tipos de algoritmos de recomendación del que se puede extraer, en primer lugar, que los sistemas de recomendación basados en filtrado colaborativo permiten realizar recomendaciones sin más información que la matriz de puntuaciones. De entre ellos destacan los de factorización matricial por su buen funcionamiento en casos de matrices dispersas y por su capacidad de predecir las puntuaciones ausentes en la matriz de puntuaciones con un bajo error. De entre ellos destaca el algoritmo BRISMF, con un aprendizaje más rápido y capacidad para tener en cuenta la evolución temporal de las puntuaciones.

También se han realizado predicciones aplicando el algoritmo BRISMF, y de los resultados obtenidos se puede extraer que la calidad de sus estimaciones es similar a la de las del algoritmo de referencia, si bien lo mejora en un porcentaje no despreciable cuando la dificultad del problema de predicción es mayor.

Hay que destacar el hecho de que la comparación con el algoritmo de referencia empleado no es del todo apropiada. Si bien permite tener la idea del desempeño de un algoritmo de filtrado colaborativo frente al de una solución básica, los requisitos para cada una de las aproximaciones son diferentes. Para el uso del algoritmo de referencia cada cliente debería sondear en un caso real a todos los *proxies*, mientras que para el sistema de recomendación sería necesario distribuir la medidas de cada cliente a los vecinos. Además, en los experimentos realizados se han utilizado todas las medidas que todos los clientes

realizan de todos los *proxies* para construir la matriz de valoraciones, pero en un escenario más realista se deberían considerar únicamente las medidas que algunos clientes hacen de algunos *proxies*, obteniendo una matriz de puntuaciones más dispersa, y predecir el valor para los *proxies* no medidos. De esta forma se estaría sacando provecho a las ventajas del filtrado colaborativo.

Por otro lado, el interés en el uso de sistemas de recomendación aumenta cuando hay un alto número de clientes, siendo algunos de ellos parecidos entre sí y deferentes de otros, situación que aparentemente no se da en el conjunto de datos empleado. No obstante, los resultados mostrados en este capítulo constituyen un indicio de la utilidad de los sistemas de recomendación para la predicción del TTFB en la selección de *proxy* de acceso a la Web en las redes de comunidad inalámbricas.

Capítulo 4

Predicción de TTFB mediante series temporales

4.1. Introducción

Las series temporales son colecciones de observaciones, que pueden estar equiespaciadas o no, y que se realizan de forma secuencial a lo largo del tiempo. Ejemplos de series temporales son los registros de ventas de un determinado producto en meses sucesivos, la temperatura en un lugar determinado al mediodía durante varios días, el consumo de electricidad en un área en particular durante periodos de una hora seguidos, o el número semanal de accidentes de carretera [Cha00, Box94]. Entre los objetivos del estudio de las series temporales se encuentran la comprensión y descripción del mecanismo de generación, la predicción de valores futuros, y el control óptimo del sistema [Wei06]. El trabajo realizado en este capítulo se centra en la predicción de valores futuros.

Las series temporales se han utilizado en varios problemas relacionados con el que se aborda en este capítulo. En [Ye16] se propone un marco de trabajo en la nube basado en la predicción de QoS a largo plazo mediante el uso de series temporales. En [Zha17] se propone un método de predicción de la QoS de diferentes servicios web mediante el uso de series temporales multivariantes. Por su parte, [Syu16] propone el uso de un método de aprendizaje automático, GP (*Genetic Programming*, Programación Genética), en la predicción de la calidad de servicio como una serie temporal. En [Zad10] se utiliza predicción de series temporales como método para reducir la sobrecarga durante la monitorización de la calidad de servicio en una arquitectura orientada a servicio. Por otro lado, en [BL18] se proponen varias técnicas para la predicción de la calidad del enlace en redes de comunidad.

En este capítulo se trata la posibilidad de abordar la predicción del TTFB modelándolo como una serie temporal, y realizar predicciones utilizando técnicas de regresión. Para ello, se realiza un estudio de los distintos tipos de algoritmos de predicción de series temporales, y se eligen aquellos que mejor encajan con el problema de predicción del TTFB. Una vez elegidos los algoritmos se realizan pruebas para determinar su capacidad de predicción y comparar la calidad de las predicciones con la del algoritmo de referencia, igual que en el capítulo anterior.

La estructura de este capítulo es la siguiente: en la sección 4.2 se define formalmente

el problema de las series temporales y se discuten los motivos por los que el uso de algoritmos de predicción de series temporales es razonable para la predicción del TTFB. En la sección 4.3 se mencionan varios tipos de serie temporal y se exponen los distintos algoritmos para su uso en la predicción. A continuación, en la sección 4.4 se presenta el *software* utilizado, en la sección 4.5 se exponen los resultados obtenidos, y por último en la sección 4.6 se plantean las conclusiones extraídas.

4.2. Formulación del problema

4.2.1. La predicción de series temporales

Se puede definir una serie temporal como un conjunto de medidas tomadas durante un intervalo de tiempo. Atendiendo a cómo se toman las medidas, la series temporales se pueden dividir entre series temporales continuas o discretas. En las primeras, las medidas son tomadas de forma continua a lo largo del tiempo, mientras que en las segundas las medidas son tomadas en un conjunto discreto de instantes temporales [Cha00]. Típicamente, las series temporales son discretas. Por otro lado, en función de si sus propiedades estadísticas (ej. media, varianza) se mantienen o no a lo largo del tiempo, se puede distinguir entre series temporales estacionarias y no estacionarias, respectivamente [Cha00].

Para la predicción de los valores futuros de una serie temporal, se pueden utilizar técnicas de aprendizaje automático. Estas técnicas se pueden dividir entre métodos en línea, que procesan cada muestra cuando llega sin la necesidad de almacenarla, y métodos por lotes, que utilizan el total o una parte de las muestras almacenadas para obtener y actualizar el modelo.

Independientemente del algoritmo de aprendizaje automático empleado, un parámetro que tiene un gran impacto en la calidad del modelo es la ventana temporal de entrenamiento. Sea una serie temporal constituida por N muestras $d = (d_1, d_2, ..., d_N)$ sobre la que se busca realizar predicciones. Se conoce como ventana temporal el intervalo comprendido por las W muestras a partir de las que se obtienen las predicciones y los S instantes futuros donde se pretende predecir el valor que tomará la serie temporal [BL18]. Este parámetro tiene un impacto no despreciable en la predicciones.

4.2.2. Predicción del TTFB como una serie temporal

Las muestras tomadas para cada pareja cliente-*proxy* se pueden representar como una serie temporal, puesto que son medidas de una magnitud tomadas a lo largo del tiempo que dan una descripción de la evolución temporal de dicha magnitud. Esta serie se caracteriza por presentar cierta variación en el periodo de muestreo, y presenta intervalos de tiempo sin muestras. Se trataría de una serie temporal discreta, puesto que se toman las muestras en intervalos de tiempo discretos, cuyas muestras no están perfectamente equiespaciadas. La predicción del TTFB, por tanto, se puede plantear como un problema de predicción de series temporales mediante regresión, y el objetivo es el de predecir el siguiente valor del TTFB en función de los valores medidos anteriormente.

4.3. Selección de algoritmos

4.3.1. Algoritmos de predicción de series temporales

Se pueden distinguir varias familias de algoritmos de regresión, entre las que destacamos las redes neuronales, los árboles de regresión, y los basados en reglas de decisión.

- **Redes neuronales**:

  Una red neuronal [Fau94] es un sistema de procesado de la información consistente en un conjunto de elementos de procesado simples llamados *neuronas*. Cada neurona está conectada a otras mediante conexiones con un peso asociado. Estos pesos representan la información que la red utiliza para resolver el problema. Se pueden utilizar en gran variedad de problemas, entre los que se encuentra la predicción. De acuerdo con [Zha98], las redes neuronales presentan una gran adaptabilidad, y su no linealidad implica un mejor modelado de las situaciones de tipo no lineal. Sin embargo, por su carácter no paramétrico en el sentido de que no se intenta ajustar un modelo, tienden a sufrir de alta varianza o de sesgo [Gem92].

- **Árboles de regresión**:

  Se trata de un tipo de algoritmo de predicción que combina los árboles de decisión con la regresión lineal. Un árbol de decisión es una estructura similar a un diagrama de flujo donde en cada nodo interno se realiza una prueba sobre un atributo, cada rama representa la salida de dicha prueba, y cada nodo *hoja* o terminal contiene una etiqueta de clase [Han11]. Combinando esta clasificación con una técnica de regresión lineal se pueden obtener predicciones a partir de un conjunto de variables de entrada. Los árboles de regresión realizan selección automática de características, y son robustos ante datos atípicos y características irrelevantes [Alm13]. Sin embargo, no pueden representar de forma robusta algunas funciones como los modelos lineales. Además, el modelo es discreto aunque produce predicciones continuas, lo que da lugar a regiones discretas con discontinuidades en los límites [Wei95].

- **Basados en reglas de decisión**:

  Al igual que los árboles de regresión, combinan un clasificador y una técnica de regresión, aprovechando la cercana relación entre ambas técnicas de aprendizaje automático [Wei95]. En este caso la clasificación se realiza mediante una serie de reglas de decisión que dividen el espacio de partida, pero a diferencia de los árboles de regresión, los espacios no tienen por qué ser disjuntos. Presentan ventajas similares a las de los árboles de regresión, como la robustez ante datos atípicos y características irrelevantes [Alm13]. Al igual que en los árboles de regresión, la principal desventaja radica del hecho de que, en muchos casos, se trata de aproximar una región del espacio de valores por un valor constante o por una función lineal, lo que puede dar lugar a un elevado error [Wei95].

En todas las familias expuestas ellas existen algoritmos de aprendizaje por lotes y algoritmos de aprendizaje en línea. Los métodos en línea resultan adecuados para la predicción del TTFB, ya que presentan ventajas en situaciones en las que las muestras llegan de

forma continua o cuando el conjunto total de datos es muy grande [Oza01]. Además, permiten comenzar a realizar predicciones sin esperar a tener un conjunto de entrenamiento, aunque inicialmente el rendimiento no será muy alto, y se adaptan a cambios en el modelo subyacente. Otra ventaja frente a los algoritmos por lotes es que los algoritmos de aprendizaje en línea implican una menor carga computacional, facilitando la posibilidad de utilizarlos en los nodos de la red con baja capacidad de cómputo [BL18].

En este capítulo se han utilizado en los experimentos llevados a cabo para la predicción del TTFB el perceptron en línea como algoritmo basado en redes neuronales, FIMT-DD (*Fast Incremental Model Trees with Drift Detection*, árbol de modelado rápido incremental con detección de cambios) [Iko11b] y ORTO (*On-line Regression/Model Trees wih Options*, árboles de modelado/regresión con opciones) [Iko11a] como algoritmos basados en árboles de regresión, y AMRules (*Adaptative Model Rules*, reglas de modelo adaptativo) [Alm13] como algoritmo basado en reglas de decisión. Esta elección se debe a que estos algoritmos han sido utilizados con cierto éxito en un problema de predicción similar en [BL18].

4.4. Marco experimental

4.4.1. Datos

Los datos empleados son los mismos que en el capítulo 3. Puesto que este capítulo está dedicado a explorar la predicción del TTFB mediante regresión para cada pareja cliente-*proxy* de forma independiente, realizaremos las pruebas con un *proxy* de cada uno de los grupos explicados en el capítulo 2, en concreto los *proxies* 2, 5 y 13. Además, y dada la similitud de las distribuciones de las muestras tomadas por los cuatro clientes, se utilizarán únicamente las realizadas por el cliente 1.

No se realiza ningún tipo de rellenado ni de normalización de los datos, al igual que se hacía en el caso del filtrado colaborativo, puesto que se trata de un trabajo exploratorio de la posibilidad de predecir el TTFB como si de una serie temporal se tratase, aunque se podrían haber eliminado los datos atípicos (*outliers*) y haber normalizado las muestras para mejorar el rendimiento de los algoritmos de predicción.

4.4.2. Entorno experimental y parámetros de experimentación

En este capítulo se utiliza la interfaz de usuario proporcionada por MOA, que permite realizar tareas de regresión a partir de un fichero .arff[1], y obtener tanto las predicciones realizadas en cada instante como la evolución del MAE y del RMSE conforme se realizan las predicciones. Como se ha justificado en la sección anterior, los algoritmos empleados son Perceptron, FIMT-DD, ORTO y AMRules.

De igual manera que en el capítulo anterior, la elección de los parámetros de los algoritmos empleados es de gran importancia, puesto que estos influyen en gran medida en los resultados obtenidos. Pero puesto que el trabajo realizado es meramente exploratorio, se ha decidido no invertir esfuerzos en buscar los valores óptimos de estos parámetros,

[1]Fichero de texto en el que cada línea contiene las muestras de la ventana temporal y el valor de salida correcto.

dejando los valores que la interfaz gráfica de MOA proporciona por defecto. El valor concreto de estos parámetros y su función se detallan en el apéndice D. No obstante, sí es interesante explorar el efecto del tamaño de la ventana temporal en las predicciones obtenidas. Por ello, los experimentos realizados en este capítulo se repiten para distintos tamaños de ventana.

4.5. Resultados

Una vez realizadas las predicciones se computa el MAE de acuerdo con la fórmula 2.6 presentada en el capítulo 2. Los resultados obtenidos se presentan en las tablas 4.1, 4.2 y 4.3. Las celdas con fondo sombreado indican los casos en los que el algoritmo correspondiente supera en rendimiento al algoritmo de referencia, mientras que el mejor resultado para cada algoritmo aparece resaltado en negrita.

	Perceptron	**FIMT-DD**	**ORTO**	**AMRules**
W=1	0.18736	0.16111	0.16111	0.14959
W=2	**0.18166**	0.16046	**0.15525**	0.14769
W=3	0.20652	**0.15680**	0.16031	**0.14713**
W=4	0.19684	0.16516	0.15854	0.14829
W=5	0.24492	0.16550	0.16057	0.14748
W=6	0.23787	0.16502	0.15916	0.14751

Tabla 4.1: Tabla con los resultados de MAE de cada uno de los algoritmos y para cada tamaño de ventana para la pareja cliente 1 - *proxy* 2. El MAE del algoritmo de referencia para esta pareja cliente-*proxy* es 0.19690.

	Perceptron	**FIMT-DD**	**ORTO**	**AMRules**
W=1	**0.01869**	0.01796	**0.01817**	**0.01651**
W=2	0.01938	0.01938	0.02189	0.01702
W=3	0.01962	0.01867	0.02012	0.01733
W=4	0.01990	**0.01792**	0.02095	0.01732
W=5	0.02079	0.01819	0.02183	0.01683
W=6	0.02093	0.01866	0.02155	0.01655

Tabla 4.2: Tabla con los resultados de MAE de cada uno de los algoritmos y para cada tamaño de ventana para la pareja cliente 1 - *proxy* 5. El MAE del algoritmo de referencia para esta pareja cliente-*proxy* es 0.02436.

	Perceptron	FIMT-DD	ORTO	AMRules
W=1	**0.05439**	**0.04964**	0.04914	0.05193
W=2	0.05876	0.05027	**0.04824**	**0.04735**
W=3	0.05599	0.05410	0.05078	0.04850
W=4	0.06108	0.05367	0.07885	0.04839
W=5	0.05896	0.05023	0.07920	0.04811
W=6	0.07765	0.05090	0.06518	0.04741

Tabla 4.3: Tabla con los resultados de MAE de cada uno de los algoritmos y para cada tamaño de ventana para la pareja cliente 1 - *proxy* 13. El MAE del algoritmo de referencia para esta pareja cliente-*proxy* es 0.06341.

Se puede observar cómo en la mayoría de los casos los algoritmos de regresión utilizados mejoran la calidad de las predicciones del algoritmo de referencia. También se puede observar la influencia de la ventana temporal, que hace que para un mismo algoritmo los resultados varíen al utilizar mayor o menor número de muestras anteriores para predecir la siguiente muestra. Sin embargo, no hay un tamaño de ventana que sea consistentemente mejor para todos los algoritmos y todos los *proxies*.

En la tabla 4.4 se muestra el porcentaje de mejora para cada uno de los algoritmos de aprendizaje utilizados respecto del algoritmo de referencia para las ventanas que mejor resultado dan en cada caso, resaltando en negrita el mejor porcentaje de mejora para cada pareja cliente-*proxy*. Se ve cómo el algoritmo AMRules obtiene un mejor resultado que el resto, y que Perceptron es el que peor resultado obtiene. Se observa también cómo el mejor resultado se obtiene para el *proxy* 5, que según argumentamos en el capítulo 2, podría ser el mejor de los *proxies* sondeados.

	Perceptron	FIMT-DD	ORTO	AMRules
Proxy 2	7,74 % (W=2)	20,37 % (W=3)	21,15 % (W=2)	**25,28 %** (W=3)
Proxy 5	23,28 % (W=1)	26,44 % (W=4)	25,41 % (W=1)	**32,22 %** (W=1)
Proxy 13	14,22 % (W=1)	21,72 % (W=1)	23,92 % (W=2)	**25,33 %** (W=2)

Tabla 4.4: Tabla con el porcentaje de mejora de cada uno de los algoritmos con respecto al algoritmo de referencia para el tamaño de ventana temporal que mejor resultado reporta.

Por otro lado, se puede realizar un análisis interesante a partir de las gráficas de la evolución del MAE con el tiempo para cada uno de los algoritmos de predicción utilizados y la evolución del TTFB ubicadas en el apéndice C. En estas gráficas hay instantes en los que un cambio en el TTFB implica un crecimiento del MAE, que a continuación se vuelve a estabilizar. Se podría argumentar que la nueva situación estable se debe a una adaptación del modelo de predicción a las nuevas propiedades estadísticas de la variable muestreada, característica de los métodos de predicción en línea. Sin embargo, al observar la evolución del TTFB, no parece que haya un cambio en su distribución, sino más bien un periodo de mayor variabilidad, y que la estabilización del MAE se debe al final de este periodo y no a la adaptación del modelo.

4.6. Discusión y conclusiones

En este capítulo se ha realizado una descripción de las series temporales, así como de algunas de las técnicas utilizadas para su predicción, mencionando sus ventajas e inconvenientes. También se ha planteado la predicción del TTFB como un problema de predicción de series temporales.

Se han realizado experimentos de predicción que se basan en otros realizados en el mismo campo de investigación. Los resultados obtenidos muestran que se produce una mejora respecto al algoritmo de referencia que si bien no es demasiado elevada, constituye una señal de que la predicción de series temporales puede ser útil de alguna forma en la predicción del TTFB para la selección del *proxy* de acceso a la Web en las redes de comunidad inalámbricas.

Mientras que en el capítulo anterior se discutía la idoneidad del algoritmo de referencia utilizado, en este caso se puede afirmar que es un buen ejemplo de algoritmo de predicción sencillo, ya que en esta aproximación se supone un sondeo de todos los clientes a todos los *proxies*. De los resultados obtenidos se puede deducir que la predicción del valor exacto del TTFB no es viable en todos los casos. El conjunto de datos utilizado podría no ser el más adecuado para evaluar la predicción con series temporales, puesto que se observa un número elevado de valores altos de TTFB medidos durante una sola muestra, que pueden deberse a problemas en la red o situaciones de carga en el *proxy* muy puntuales, y que aparecen rodeados de valores muy bajos. Como se puede ver en las figuras del apéndice C, los algoritmos son incapaces de predecir estos grandes picos del TTFB, y esto contribuye en gran medida el error.

También es importante destacar que el periodo de medida es largo, y que el comportamiento que aparece reflejado es el habitual de Guifi.net, según confirma el grupo de la UPC que ha generado los datos. En [Dim17] se añaden cargas sintéticas en los *proxies* que generan un comportamiento que permite estudiar la correlación entre el TTFB y la carga. Pero este procedimiento no tiene sentido para el trabajo realizado en este capítulo, puesto que se estaría generando una serie temporal artificial cuyo aprendizaje no sería de utilidad para situaciones reales.

Podría ser interesante realizar un estudio del problema con un enfoque de clasificación de una serie temporal, en el que se intente predecir una etiqueta que describa la carga del *proxy* (ej. baja, media, alta) a partir de los valores pasados de estas etiquetas. También podría ser interesante estudiar la posibilidad de predecir si en un intervalo (por ejemplo, en el próximo minuto) la carga va a estar en una de estas tres categorías, en lugar de predecirlo para la siguiente muestra de forma aislada.

Capítulo 5

Conclusiones y trabajo futuro

En este trabajo se ha abordado la predicción del TTFB medido desde un cliente al acceder a un *proxy* web en el contexto de las redes de comunidad inalámbricas. La predicción del TTFB tiene cierta importancia, ya que guarda cierta correlación con la carga del *proxy* y está claramente relacionado con la calidad de servicio percibida por el usuario final. Así, la predicción se podría utilizar para cambiar automáticamente de *proxy* que se utiliza para acceder a la Web, balanceando la carga entre los *proxies* disponibles y mejorando la percepción del servicio.

Para abordar el estudio, se han propuesto dos alternativas. La primera de ellas consiste en el uso de los mecanismos de los sistemas de recomendación, en particular de filtrado colaborativo, para la estimación del TTFB a los *proxies* que un cliente de prueba no sondea. Esta aproximación permite explotar tanto las medidas que otros clientes toman para dichos *proxies*, como las semejanzas entre los clientes extraídas de las medidas tomadas por ellos a los *proxies* que el cliente de prueba sí sondea. Esto permitiría reducir la sobrecarga en la red, puesto que se reduciría el número de sondeos realizados por los clientes y se reducirían las retransmisiones de los resultados de esos sondeos. Sin embargo, aunque los resultados obtenidos mejoran ligeramente los resultados del algoritmo de referencia, la naturaleza de los datos utilizados no permite alcanzar unas conclusiones muy sólidas sobre la conveniencia de los sistemas de recomendación. Esto es debido por un lado a la semejanza entre todos los clientes, y por el otro al bajo número de clientes utilizados para tomar las medidas.

La segunda de las alternativas consiste en el uso de los mecanismos de predicción de series temporales para predecir los valores futuros del TTFB medido desde un cliente. Esta alternativa permitiría al cliente saber con antelación el valor del TTFB al *proxy* que está sondeando y anteponerse a cambios en su valor para evitar así *proxies* cuyo TTFB predicho es alto. Esto podría ser útil si se va a iniciar una comunicación que se prevé larga, y que por tanto se puede ver afectada por el cambio en el TTFB. Sin embargo, esta solución implica que un cliente sondee a todos los *proxies* del sistema, lo cual aumentaría notablemente la sobrecarga en la red respecto del caso en que sólo sondea a unos pocos *proxies*. En los resultados obtenidos se puede observar cierta influencia de la ventana temporal utilizada en los algoritmos, y cierta mejora en la mayoría de los casos respecto al algoritmo de referencia. Sin embargo, la forma ruidosa y altamente variable del TTFB medido supone un reto muy grande para los sistemas de predicción, lo cual se puede ver en los cambios bruscos en el MAE cuando el TTFB presenta mayor variabilidad.

Por otra parte, y a la vista de los datos, el problema que se plantea es más académico que real, ya que no parece que los *proxies* ofrezcan un rendimiento degradado en ningún momento. Esto no significa que el problema no exista en potencia, sino que cualquier sistema de elección de *proxy* que combine la proximidad física del cliente y un factor aleatorio que evite que la carga se concentre en algunos *proxies* podría ser suficiente si el número de *proxies* disponible es alto en comparación con la demanda de los usuarios finales.

5.1. Trabajo futuro

Respecto al trabajo futuro, los resultados obtenidos mediante ambas aproximaciones plantean varios caminos de investigación a seguir. En primer lugar, cabe plantear la opción de repetir los experimentos de estimación mediante filtrado colaborativo, esta vez con un conjunto de datos que contenga un mayor número de clientes y cuyas características sean diferentes. Se podría tomar medidas en varios clientes, por ejemplo ocho, formando grupos de dos clientes físicamente próximos entre sí y lejanos del resto. Además, sería interesante dispersar la matriz de puntuaciones, de forma que clientes similares sondeasen a varios *proxies* comunes y varios *proxies* distintos, para poder observar por un lado el funcionamiento del filtrado colaborativo y por otro cuánto se puede reducir el tráfico en la red (por los sondeos evitados) manteniendo la calidad de los estimadores. También se podría optimizar los valores de los parámetros de BRISMF, y probar otros algoritmos de filtrado colaborativo.

Por otro lado, en el ámbito de las series temporales, se podría tratar de optimizar el valor de los parámetros de entrenamiento de cada algoritmo empleado y extender el estudio a más parejas cliente *proxy*, con series temporales más largas y utilizando otros algoritmos. Se podría comprobar también si es posible reducir, a partir de unas predicciones fiables, el número de veces que se sondea sustituyendo de forma intercalada una medida real y una predicción. También se podría plantear la posibilidad de enfocar la predicción del TTFB mediante series temporales como un problema de clasificación, en el que no se busque predecir el valor exacto del TTFB, sino si el valor medio del TTFB durante un periodo futuro, como por ejemplo el siguiente minuto, va a estar en un determinado rango (ej. bajo, medio o alto). Esto podría ser más útil que la predicción del valor exacto e instantáneo a la hora de elegir el siguiente *proxy*, ya que el cliente dispondría de información sobre el estado medio del *proxy* durante un periodo, y no de las variaciones exactas del TTFB.

Otro camino a seguir podría ser combinar los dos métodos propuestos en este trabajo. Así, se podría utilizar el filtrado colaborativo para reducir el número de *proxies* a los que un cliente sondea, y la predicción para reducir el número de sondeos realizados (ej. se podrían realizar sondeos cada dos rondas Vivaldi, y difundir la información cada ronda de forma que una vez se difundiese el valor medido y otra el valor predicho).

Referencias

[Ado05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, Junio 2005. 18, 19

[Agg16] Charu C. Aggarwal. *An Introduction to Recommender Systems*, páginas 1–28. Springer International Publishing, Cham, 2016. 17, 25

[Aky05] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: A survey. *Computer Networks*, 47(4):445–487, Marzo 2005. 2

[Alm13] Ezilda Almeida, Carlos Ferreira, and João Gama. Adaptive model rules from data streams. En Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editores, *Machine Learning and Knowledge Discovery in Databases*, páginas 480–492, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 36, 37

[Bif10] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. Moa: Massive online analysis, a framework for stream classification and clustering. En Tom Diethe, Nello Cristianini, and John Shawe-Taylor, editores, *Proceedings of the First Workshop on Applications of Pattern Analysis*, volumen 11 of *Proceedings of Machine Learning Research*, páginas 44–50, Cumberland Lodge, Windsor, UK, 01–03 Sep 2010. PMLR. 29

[BL18] Miguel L. Bote-Lorenzo, Eduardo Gómez-Sánchez, Carlos Mediavilla-Pastor, and Juan I. Asensio-Pérez. Online machine learning algorithms to predict link quality in community wireless mesh networks. *Computer Networks*, 132:68 – 80, 2018. 2, 34, 35, 37

[Bok15] Dheeraj Bokde, Sheetal Girase, and Debajyoti Mukhopadhyay. Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science*, 49:136 – 146, 2015. Proceedings of 4th International Conference on Advances in Computing, Communication and Control (ICAC3'15). 24

[Box94] George Edward Pelham Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 3rd edición, 1994. 34

[Bri05] Derek Bridge, Mehmet H. Göker, Lorraine McGinty, and Barry Smyth. Case-based recommender systems. *The Knowledge Engineering Review*, 20(3):315–320, Septiembre 2005. 20

[Cac11] Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Transactions on the Web*, 5(1):2:1–2:33, Febrero 2011. 24

[Cha00] C. Chatfield. *Time-Series Forecasting*. CRC Press, 2000. 34, 35

[Dab04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. *SIGCOMM Computer Communication Review*, 34(4):15–26, Agosto 2004. 6

[Dim17] E. Dimogerontakis, J. Neto, R. Meseguer, L. Navarro, and L. Veiga. Client-side routing-agnostic gateway selection for heterogeneous wireless mesh networks. En *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, páginas 377–385, May 2017. 1, 2, 3, 5, 6, 7, 8, 40

[Fau94] Laurene Fausett, editor. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994. 36

[Fis12] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. En Luis Alvarez, Marta Mejail, Luis Gomez, and Julio Jacobo, editores, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, páginas 14–36, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 24

[Gem92] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, Enero 1992. 36

[Han11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edición, 2011. 23, 36

[Iko11a] Elena Ikonomovska, João Gama, Bernard Ženko, and Sašo Džeroski. Speeding up hoeffding-based regression trees with options. En *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, páginas 537–544, USA, 2011. Omnipress. 37

[Iko11b] Elena Ikonomovska, João Gama, and Sašo Džeroski. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168, Jul 2011. 37

[Isa04] Pedro Isasi and Inés Galván. *Redes de neuronas artificiales: un enfoque práctico*. Editorial Alhambra S. A. (SP), 2004. 24

[Jai03] Saurabh Jain and D. P. Agrawal. Wireless community networks. *Computer*, 36(8):90–92, Aug 2003. 2

[JW05] C J. Willmott and K Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. 30:79, 12 2005. 15

[Kor08] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. En *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, páginas 426–434, New York, NY, USA, 2008. ACM. 26

[Kor09] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug 2009. 17, 19, 23, 24

[Lin08] Yen-Wen Lin, Li-Su Kuo, Jhih-Siang Wang, and Wei-Ting Shiu. P2p-based instant messaging for wireless community networks. En *2008 Third International Conference on Communications and Networking in China*, páginas 1330–1334, Aug 2008. 1

[Lop11] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems: State of the Art and Trends*, páginas 73–105. Springer US, Boston, MA, 2011. 17, 20

[Luo16] Xiong Luo, Ji Liu, Dandan Zhang, and Xiaohui Chang. A large-scale web qos prediction scheme for the industrial internet of things based on a kernel machine learning algorithm. *Computer Networks*, 101:81 – 89, 2016. Industrial Technologies and Applications for the Internet of Things. 18

[Mac13] L. Maccari. An analysis of the ninux wireless community network. En *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, páginas 1–7, Oct 2013. 1

[Mil15] Pere Millan, Carlos Molina, Esunly Medina, Davide Vega, Roc Meseguer, Bart Braem, and Chris Blondia. Time series analysis to predict link quality of wireless community networks. *Computer Networks*, 93(P2):342–358, Diciembre 2015. 2

[Neu16] A. Neumann, E. Lopez, L. Cerda-Alabern, and L. Navarro. Securely-entrusted multi-topology routing for community networks. En *2016 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, páginas 1–8, Jan 2016. 1, 2

[Oza01] Nikunj C. Oza and Stuart Russell. Experimental comparisons of online and batch versions of bagging and boosting. En *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, páginas 359–364, New York, NY, USA, 2001. ACM. 37

[Par14] European Parliament and Council. Directive 2014/61/eu of the european parliament and the council of 15 may 2014 on measures to reduce the cost of deploying high-speed electronic communication networks. Informe técnico 2017/61/eu, Mayo 2014. 1

[Res97] Paul Resnick and Hal R. Varian. Recommender systems. *Magazine Communications of the ACM*, 40(3):56–58, Marzo 1997. 17

[Ric15] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*. Springer Publishing Company, Incorporated, 2nd edición, 2015. 17, 19

[RM13] Carlos Rey-Moreno, Zukile Roro, William D. Tucker, Masbulele Jay Siya, Nicola J. Bidwell, and Javier Simo-Reigadas. Experiences, challenges and lessons from rolling out a rural wifi mesh network. En *Proceedings of the 3rd ACM Symposium on Computing for Development*, ACM DEV '13, páginas 11:1–11:10, New York, NY, USA, 2013. ACM. 1

[Sal07] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. En *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, páginas 1257–1264, USA, 2007. Curran Associates Inc. 26

[Sar00] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Application of dimensionality reduction in recommender systems: A case study. En *WebKDD Workshop at the ACM SIGKKD*, 2000. 26

[Sch07] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. *Collaborative Filtering Recommender Systems*, páginas 291–324. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. 20

[Su09] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:4:2–4:2, Enero 2009. 17, 22, 23

[Syu16] Y. Syu, Y. Y. Fanjiang, J. Y. Kuo, and J. L. Su. Quality of service timeseries forecasting for web services: A machine learning, genetic programming-based approach. En *2016 Annual Conference on Information Science and Systems (CISS)*, páginas 343–348, March 2016. 34

[Tak09] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *The Journal of Machine Learning Research*, 10:623–656, Junio 2009. 17, 26, 27, 28, 29

[Uni09] International Telecommunication Union. Trends in telecommunication reform 2008: Six degrees of sharing. Informe técnico D-PREF-TTR.10, Julio 2009. 1

[Vad16] G. Vadivelou. Collaborative filtering based web service recommender system using users' satisfaction on qos attributes. En *2016 International Conference*

on Inventive Computation Technologies (ICICT), volumen 3, páginas 1–5, Aug 2016. 18

[Veg15] Davide Vega, Roger Baig, LlorenÃ§ CerdÃ -Alabern, Esunly Medina, Roc Meseguer, and Leandro Navarro. A technological overview of the guifi.net community network. *Computer Networks*, 93:260 – 278, 2015. Community Networks. 1

[Wan06] Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. En *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, páginas 501–508, New York, NY, USA, 2006. ACM. 22

[Wei95] Sholom M. Weiss and Nitin Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3(1):383–403, Diciembre 1995. 36

[Wei06] W.W.S. Wei. *Time Series Analysis: Univariate and Multivariate Methods*. Pearson Addison Wesley, 2006. 34

[Ye16] Z. Ye, S. Mistry, A. Bouguettaya, and H. Dong. Long-term qos-aware cloud service composition using multivariate time series analysis. *IEEE Transactions on Services Computing*, 9(3):382–393, May 2016. 34

[Zad10] M. Hosein Zadeh and M. A. Seyyedi. Qos monitoring for web services by time series forecasting. En *2010 3rd International Conference on Computer Science and Information Technology*, volumen 5, páginas 659–663, July 2010. 34

[Zha98] Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 14(1):35 – 62, 1998. 36

[Zha17] P. Zhang, L. Wang, W. Li, H. Leung, and W. Song. A web service qos forecasting approach based on multivariate time series. En *2017 IEEE International Conference on Web Services (ICWS)*, páginas 146–153, June 2017. 34

[Zhe17] X. Zheng, L. D. Xu, and S. Chai. Qos recommendation in cloud services. *IEEE Access*, 5:5171–5177, 2017. 18

[Zhu17] J. Zhu, P. He, Z. Zheng, and M. R. Lyu. Online qos prediction for runtime service adaptation via adaptive matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 28(10):2911–2924, Oct 2017. 18

Apéndice A

Figuras de TTFB

En este apéndice se incluyen las figuras de la evolución del TTFB para cada *proxy* y los diagramas de cajas de las muestras tomadas para cada pareja cliente-*proxy*.

A.1. Evolución del TTFB para cada *proxy*

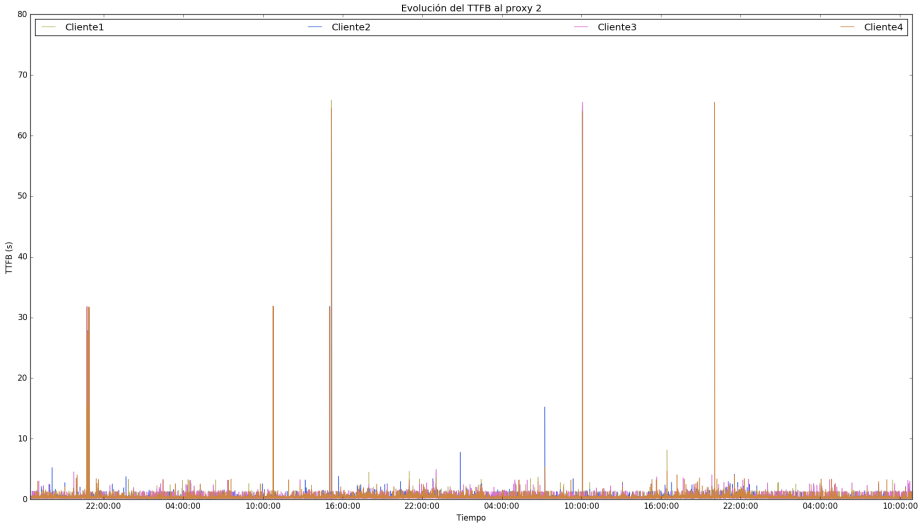


Figura A.1: Evolución del TTFB al *proxy* 2

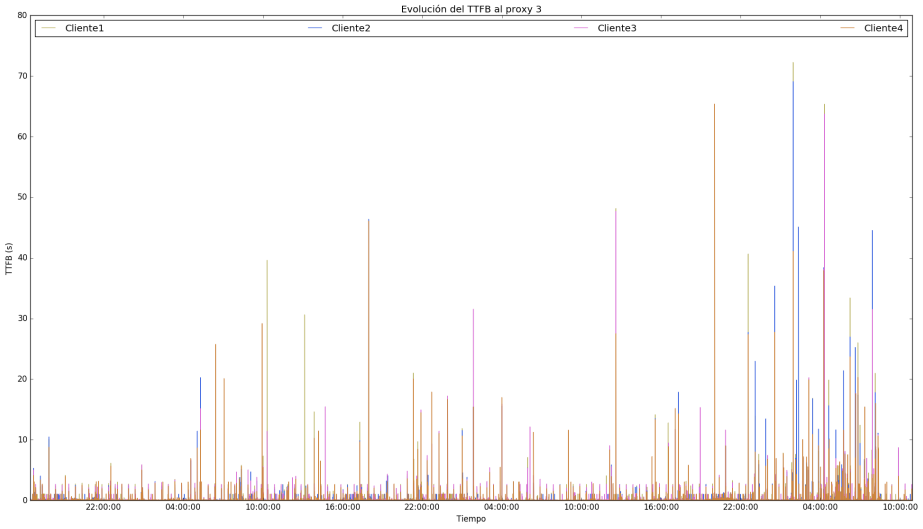


Figura A.2: Evolución del TTFB al *proxy* 3

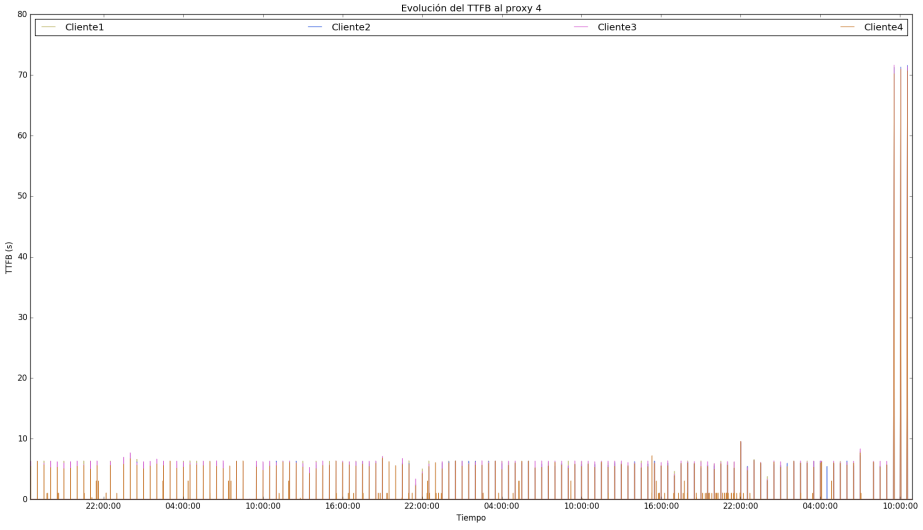


Figura A.3: Evolución del TTFB al *proxy* 4

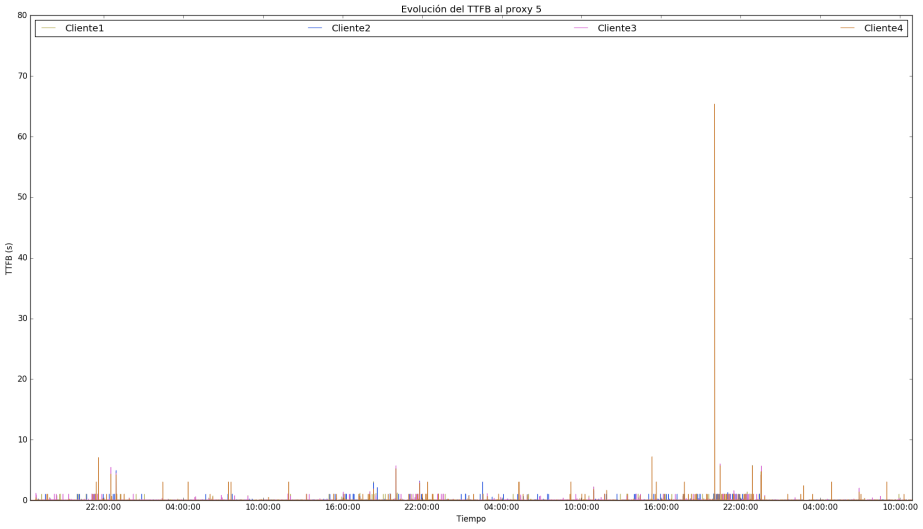


Figura A.4: Evolución del TTFB al *proxy* 5

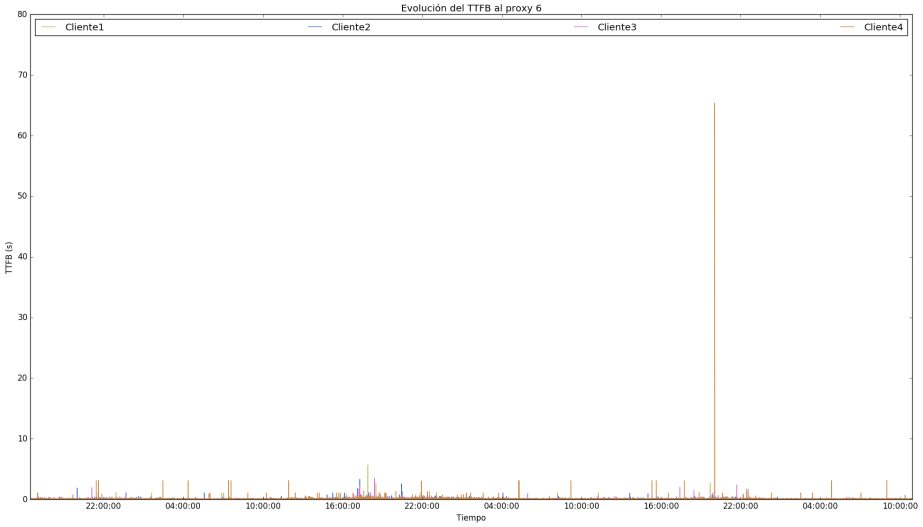


Figura A.5: Evolución del TTFB al *proxy* 6

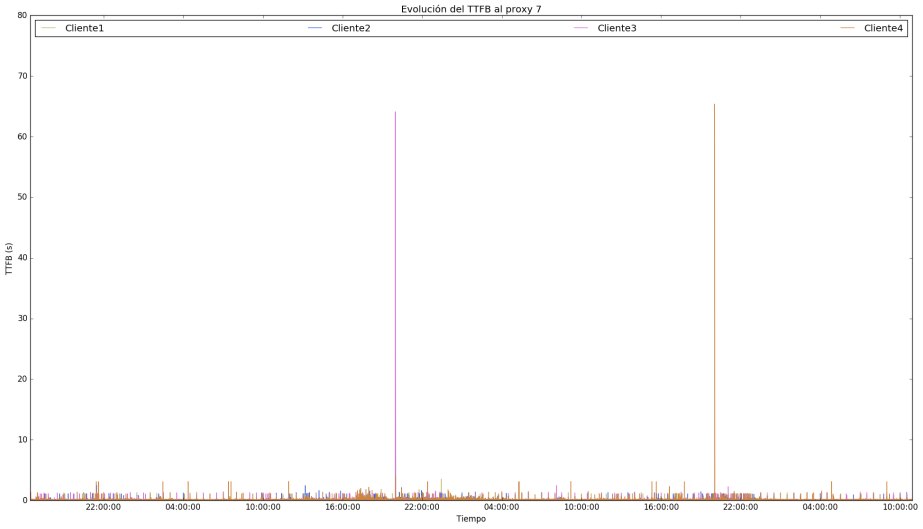


Figura A.6: Evolución del TTFB al *proxy* 7

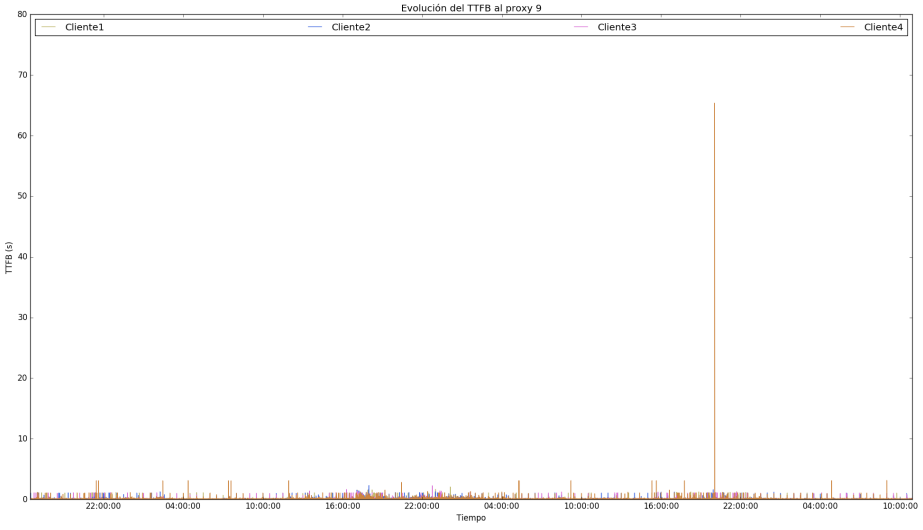


Figura A.7: Evolución del TTFB al *proxy* 9

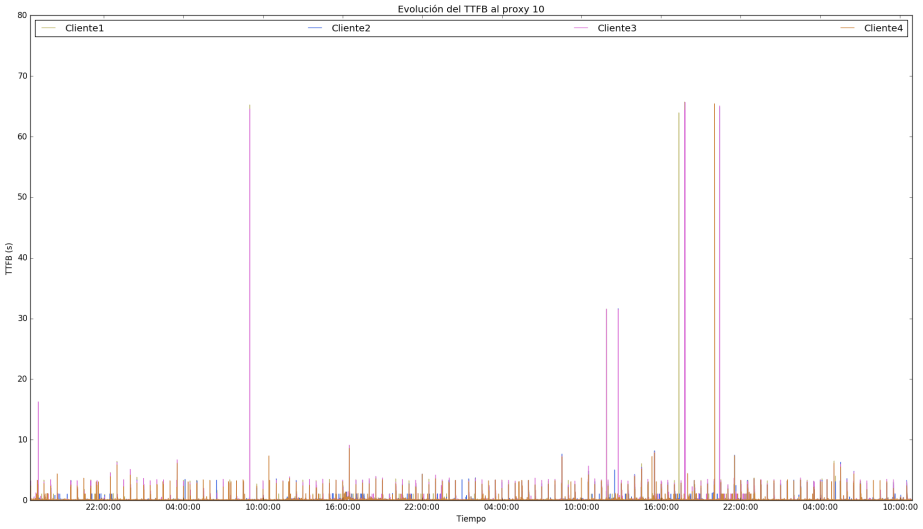


Figura A.8: Evolución del TTFB al *proxy* 10

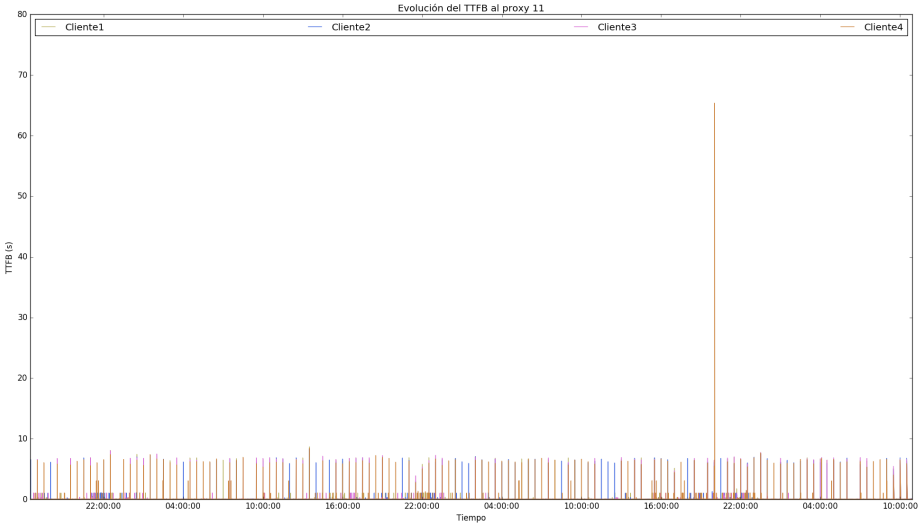


Figura A.9: Evolución del TTFB al *proxy* 11

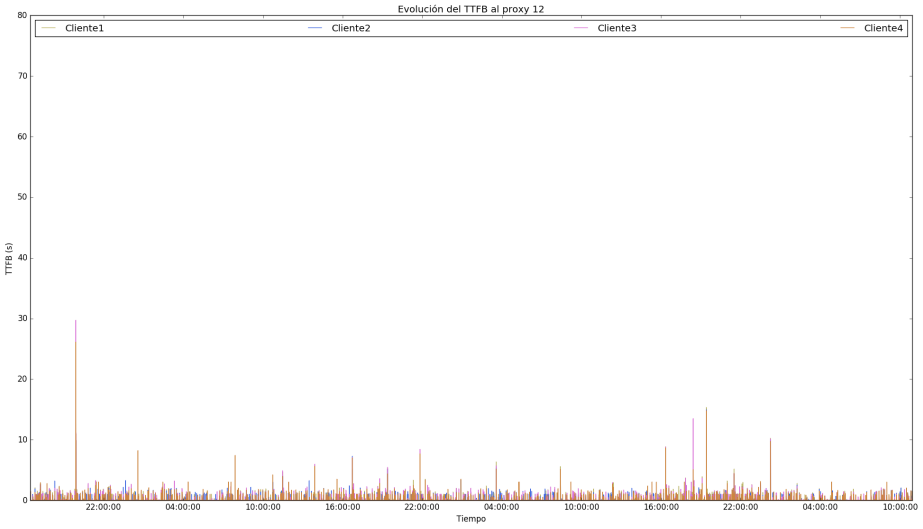


Figura A.10: Evolución del TTFB al *proxy* 12

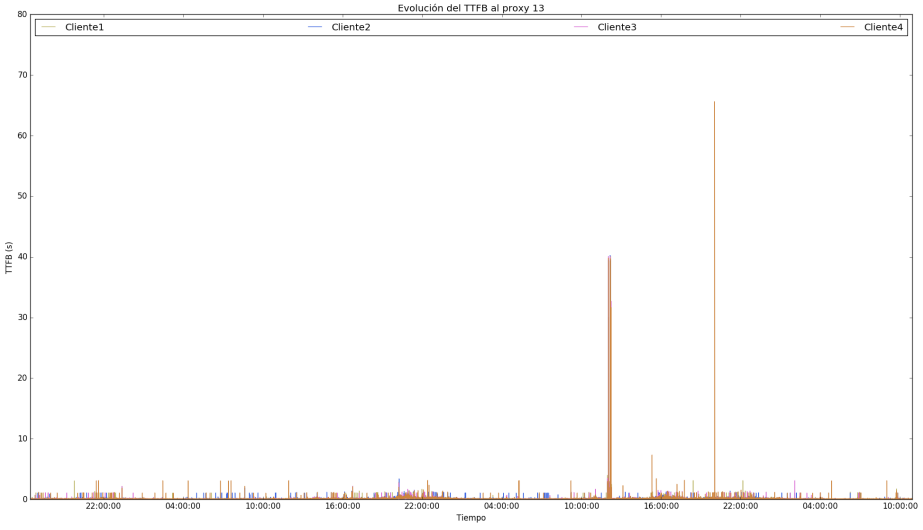


Figura A.11: Evolución del TTFB al *proxy* 13

A.2. Diagramas de cajas de las muestras por pareja cliente-*proxy*

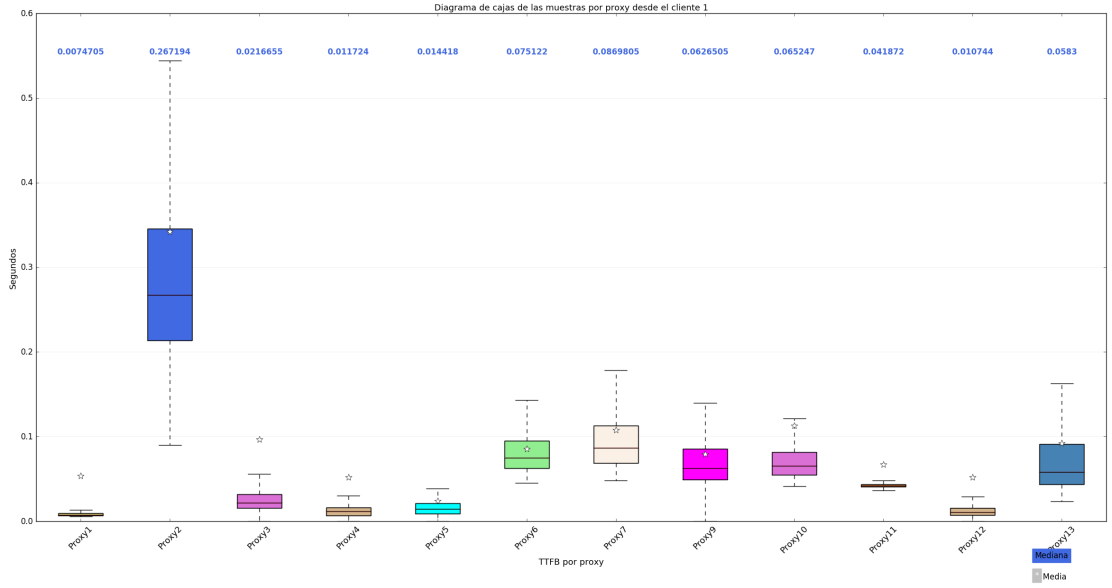


Figura A.12: Diagrama de cajas de las muestras por *proxy* para el cliente 1

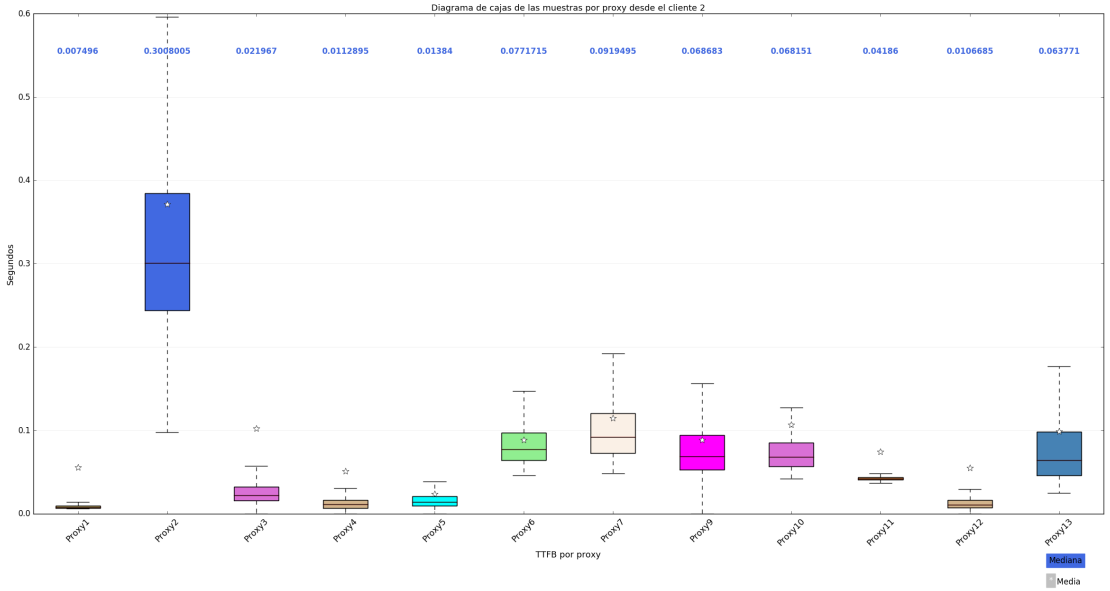


Figura A.13: Diagrama de cajas de las muestras por *proxy* para el cliente 2

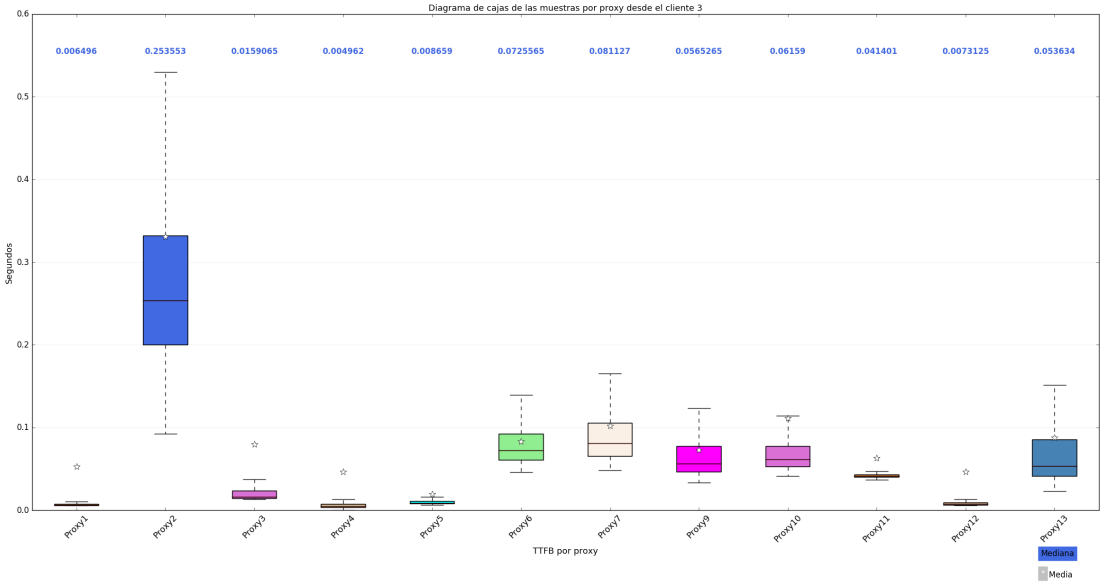


Figura A.14: Diagrama de cajas de las muestras por *proxy* para el cliente 3

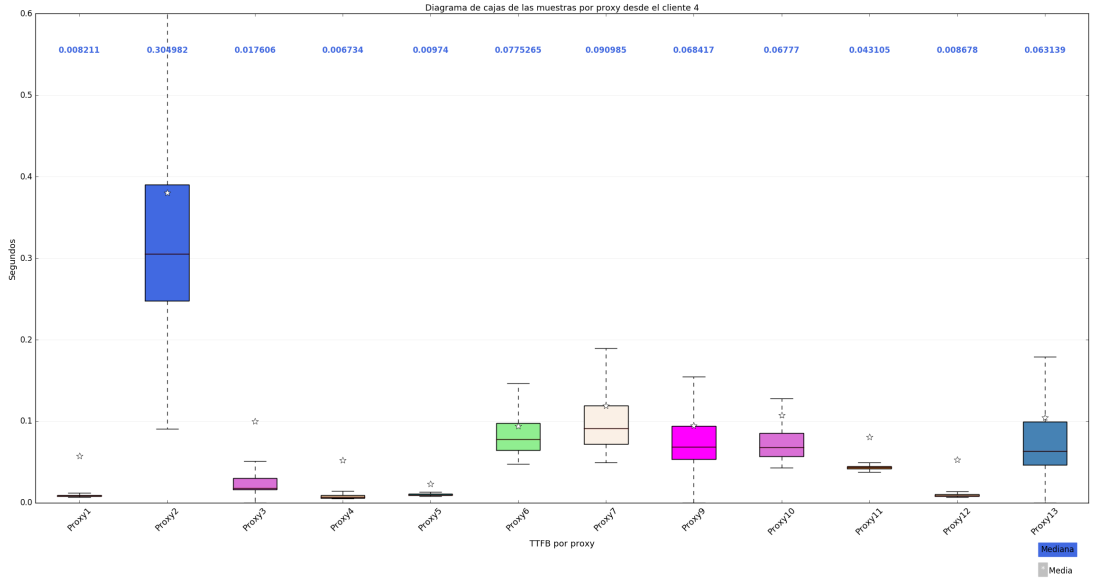


Figura A.15: Diagrama de cajas de las muestras por *proxy* para el cliente 4

Apéndice B

Figuras de evolución del MAE en el uso de sistemas de recomendación para la estimación del TTFB

En este anexo se añaden las figuras de la evolución del MAE y los diagramas de cajas de los errores cometidos tanto por el algoritmo de referencia como por BRISMF.

B.1. Evolución del MAE por pareja cliente-*proxy*

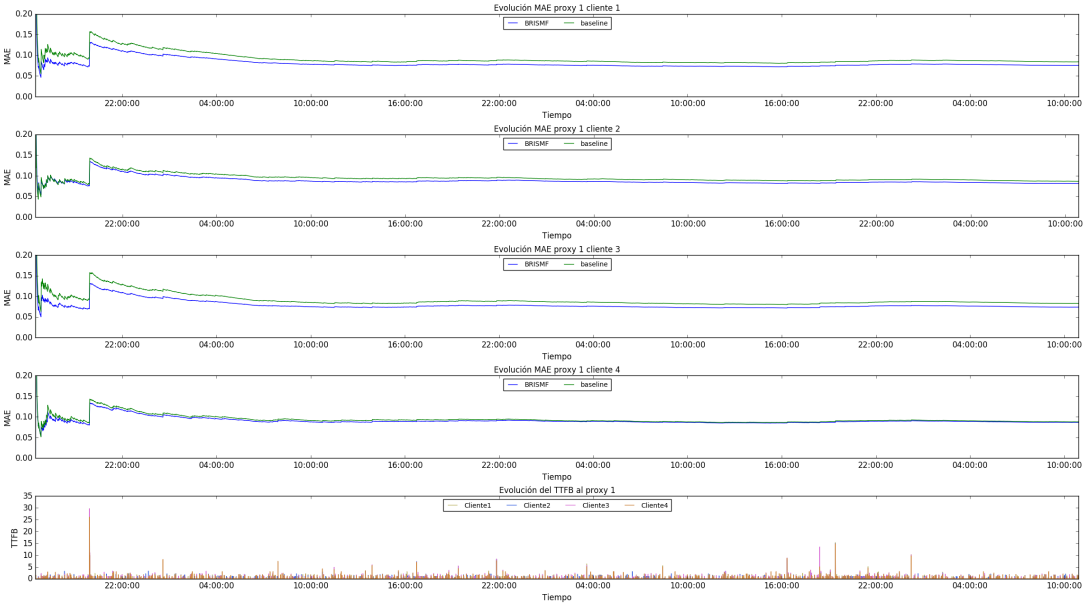


Figura B.1: Evolución del MAE frente a la evolución del TTFB en el *proxy* 1

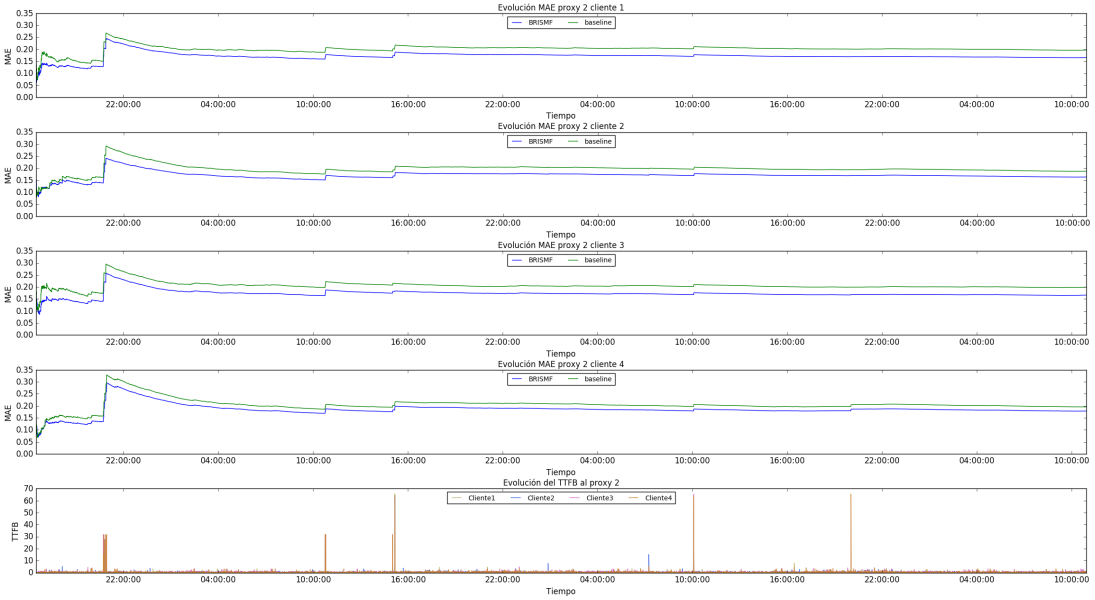


Figura B.2: Evolución del MAE frente a la evolución del TTFB en el *proxy* 2

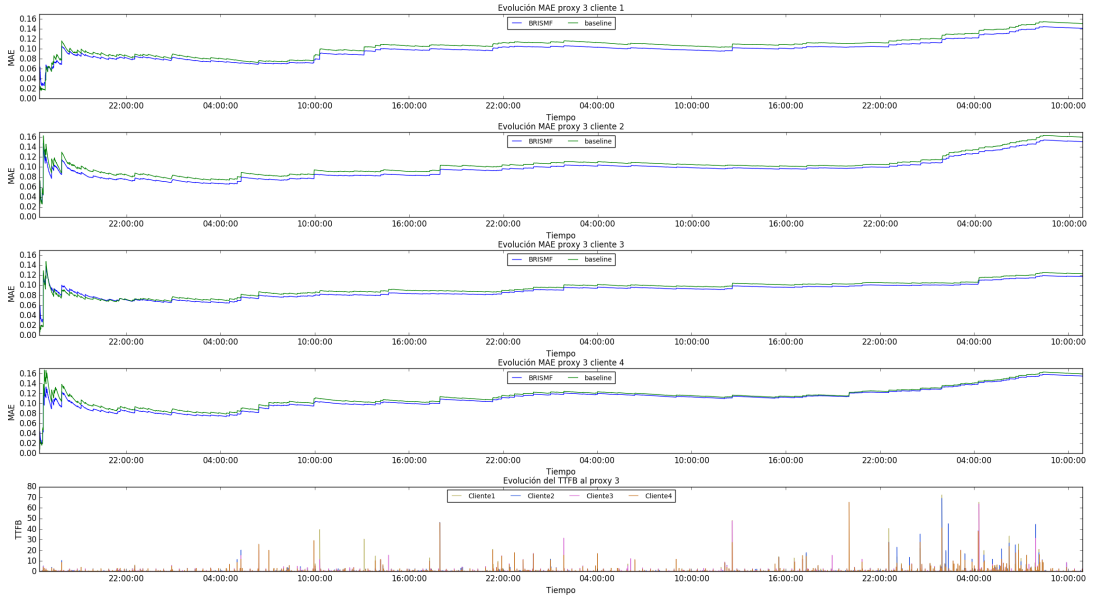


Figura B.3: Evolución del MAE frente a la evolución del TTFB en el *proxy* 3

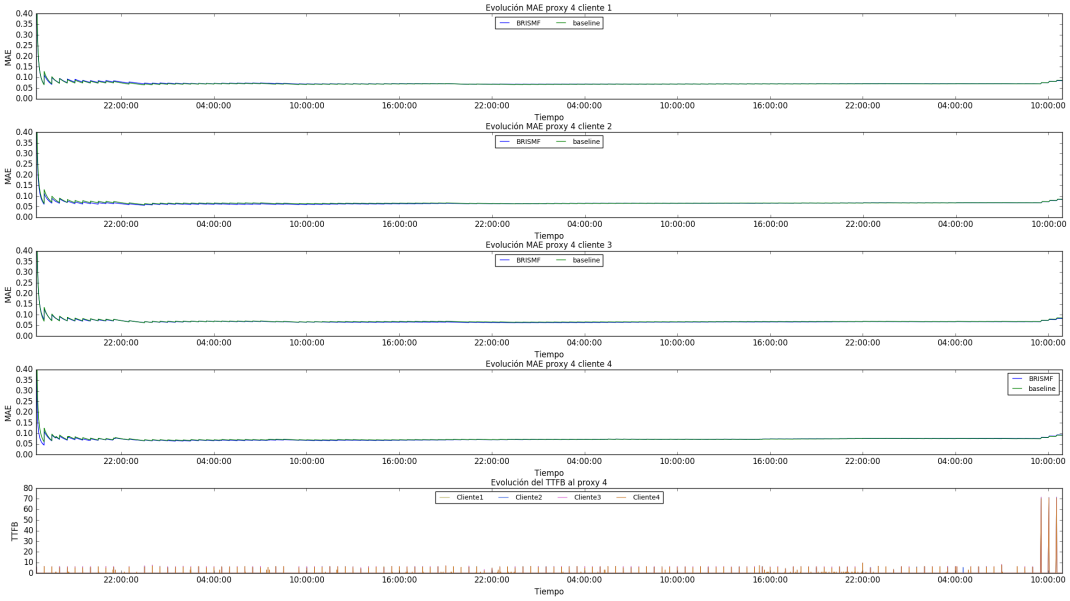


Figura B.4: Evolución del MAE frente a la evolución del TTFB en el *proxy* 4

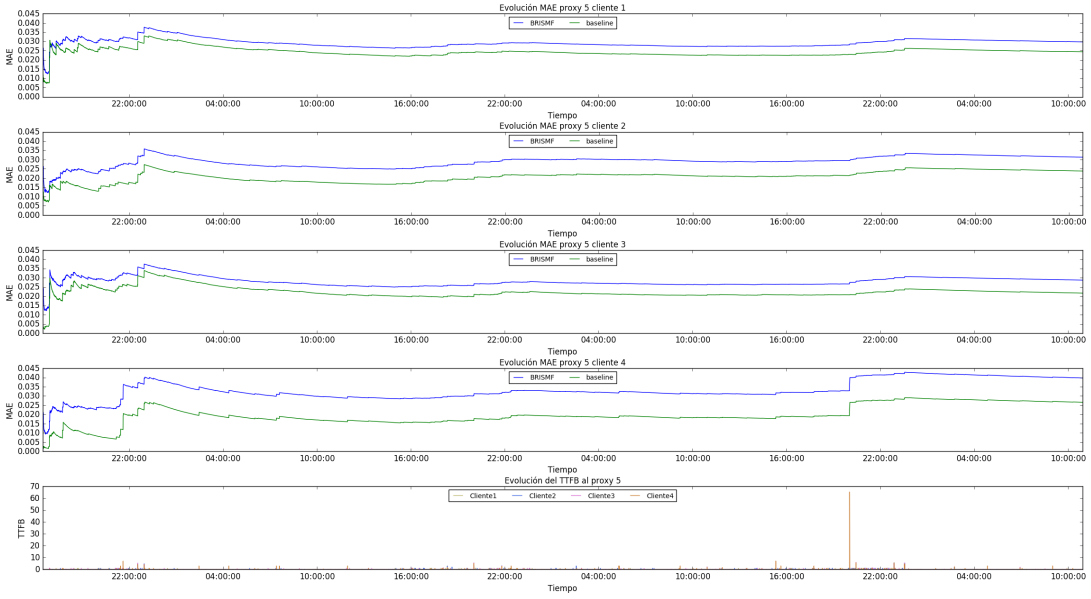


Figura B.5: Evolución del MAE frente a la evolución del TTFB en el *proxy* 5

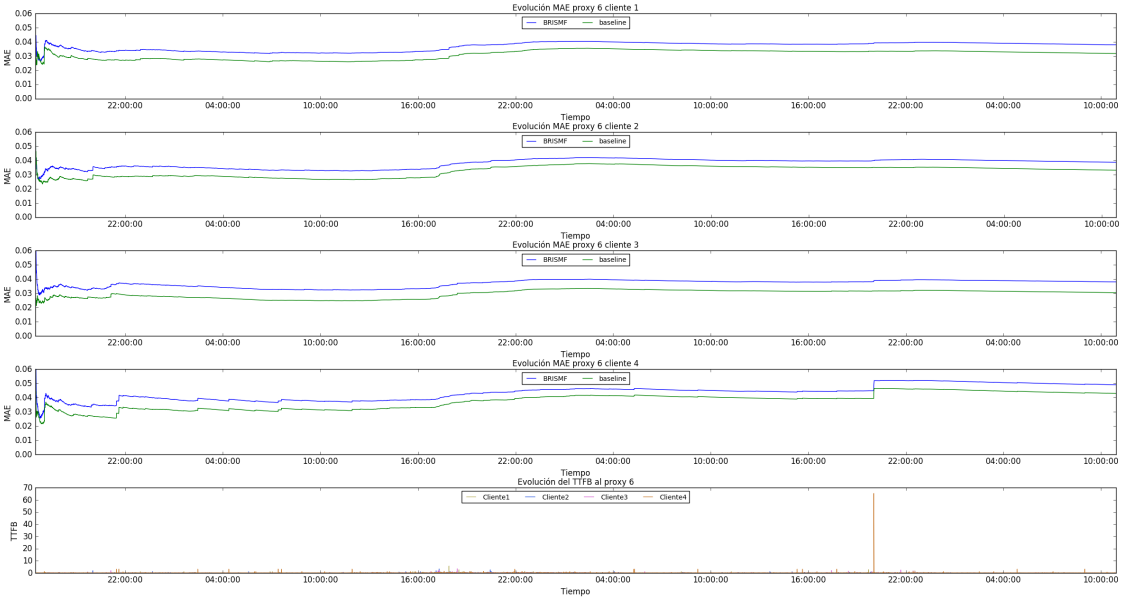


Figura B.6: Evolución del MAE frente a la evolución del TTFB en el *proxy* 6

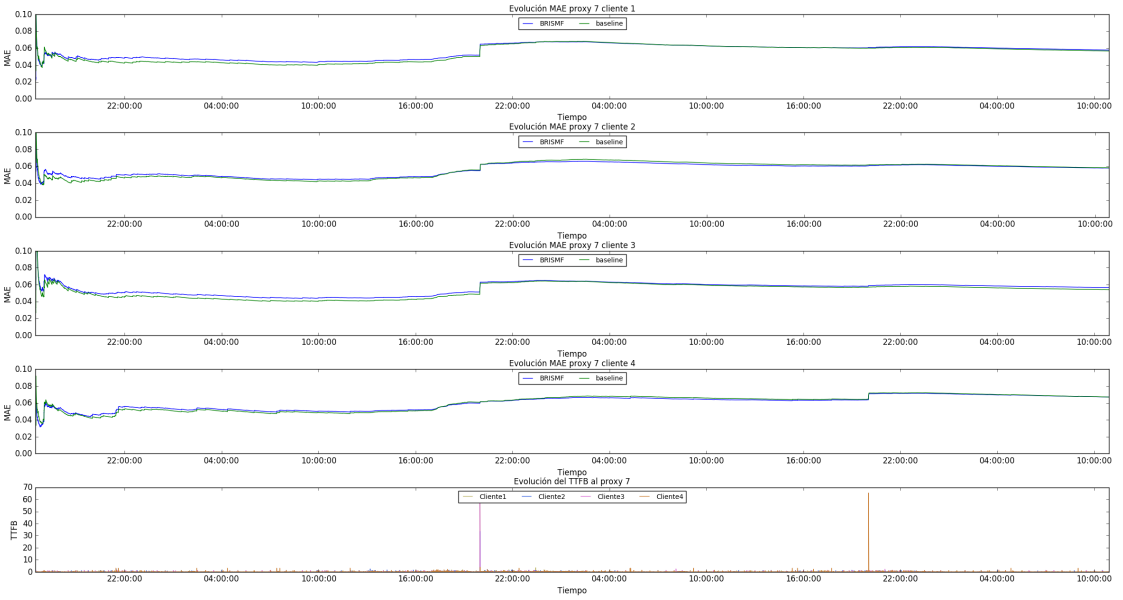


Figura B.7: Evolución del MAE frente a la evolución del TTFB en el *proxy* 7

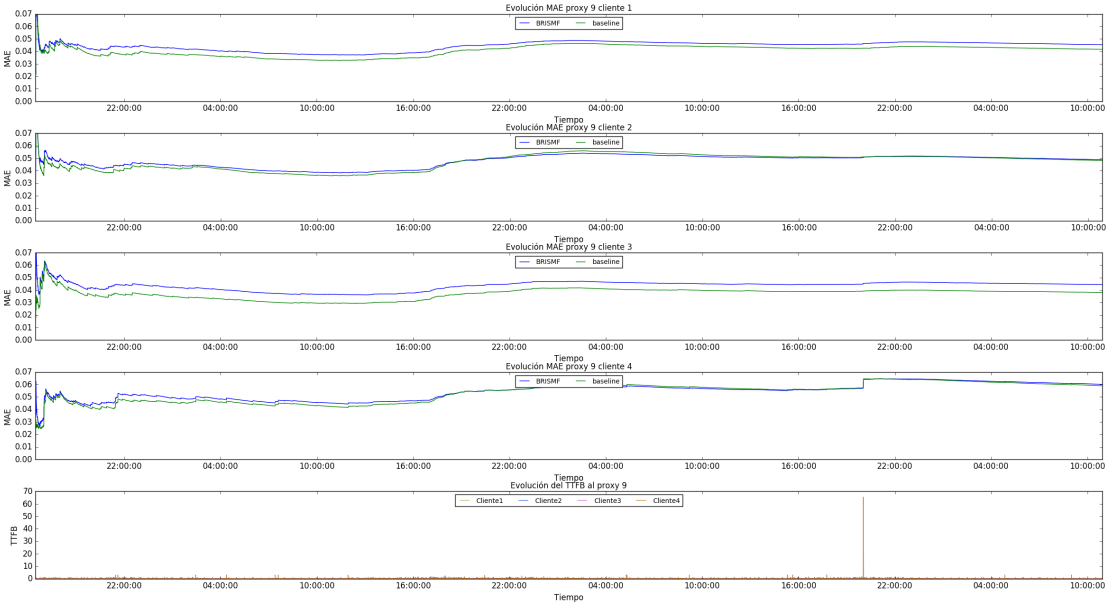


Figura B.8: Evolución del MAE frente a la evolución del TTFB en el *proxy* 9

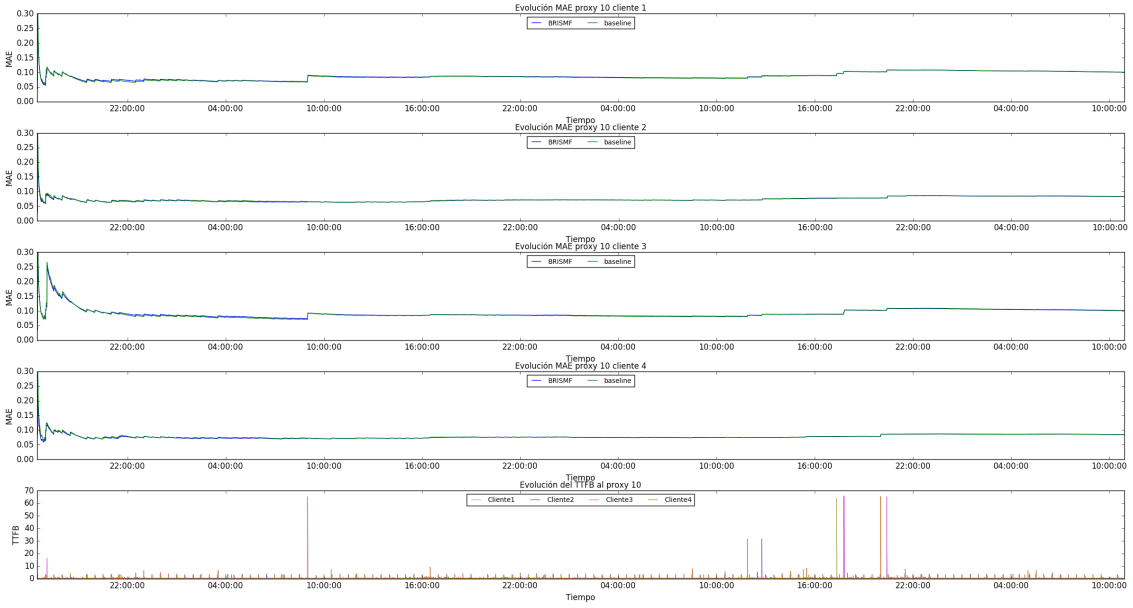


Figura B.9: Evolución del MAE frente a la evolución del TTFB en el *proxy* 10

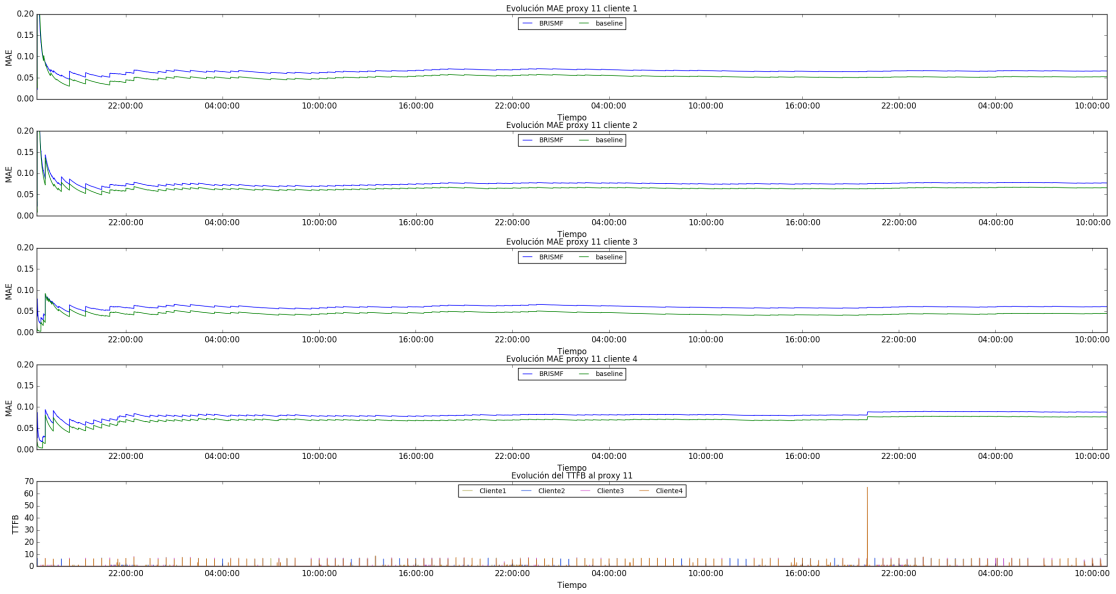


Figura B.10: Evolución del MAE frente a la evolución del TTFB en el *proxy* 11

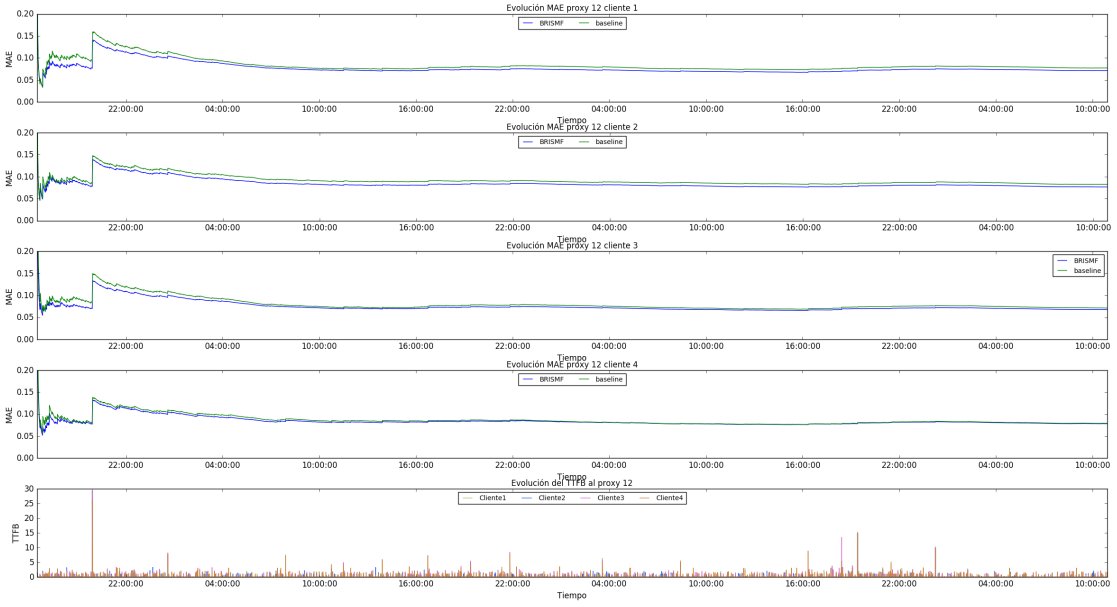


Figura B.11: Evolución del MAE frente a la evolución del TTFB en el *proxy* 12

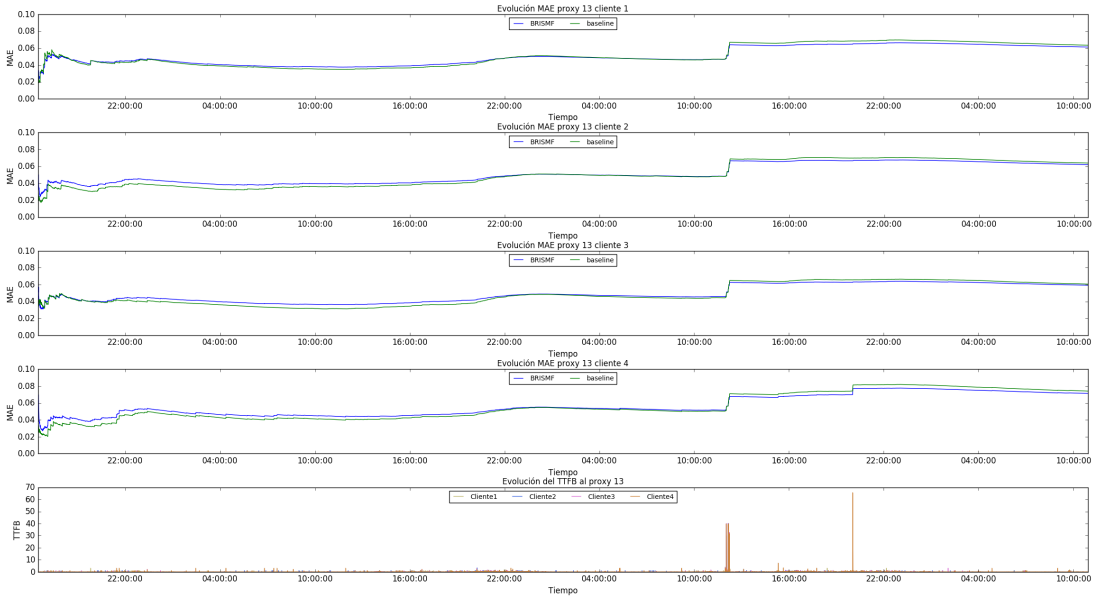


Figura B.12: Evolución del MAE frente a la evolución del TTFB en el *proxy* 13

B.2. Diagramas de cajas de los errores

En esta sección se muestran los diagramas de cajas de los errores cometidos tanto por el algoritmo de referencia como por BRISMF para cada uno de los clientes:

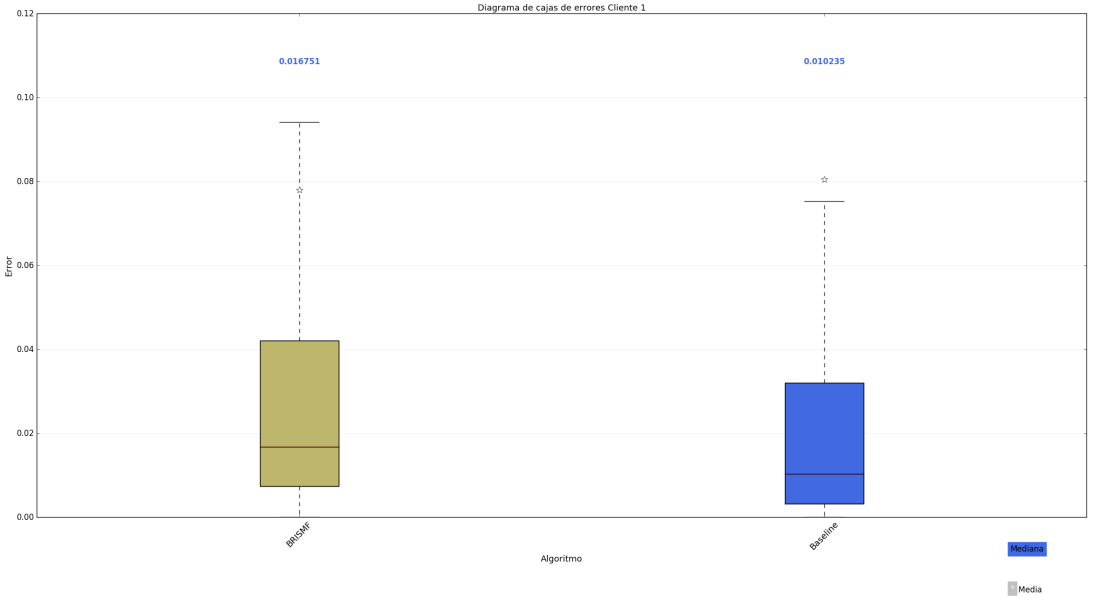


Figura B.13: Diagrama de cajas de los errores para el cliente 1

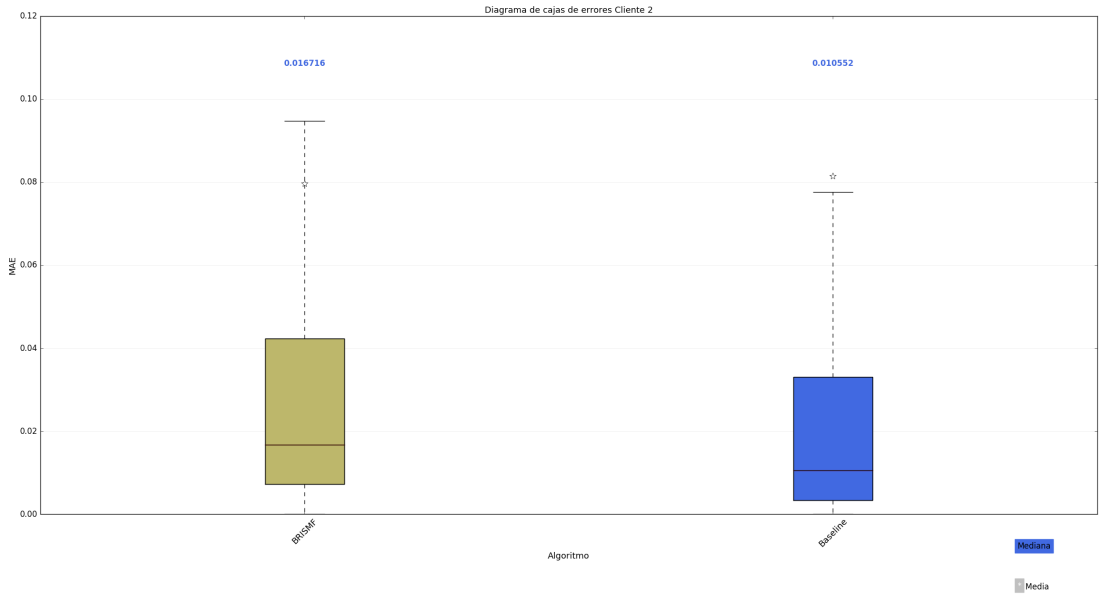


Figura B.14: Diagrama de cajas de los errores para el cliente 2

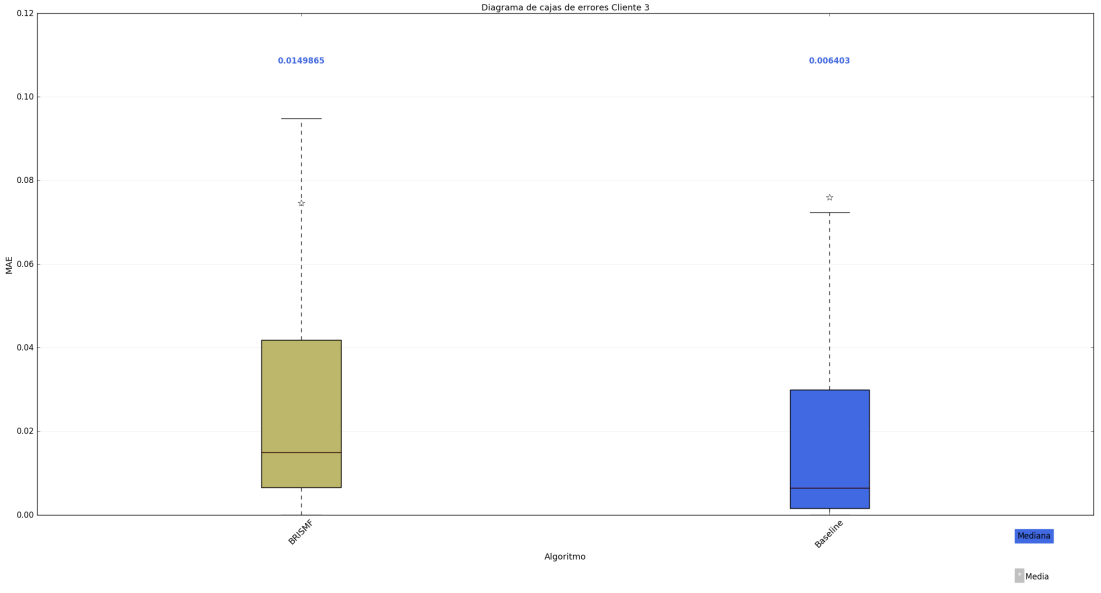


Figura B.15: Diagrama de cajas de los errores para el cliente 3

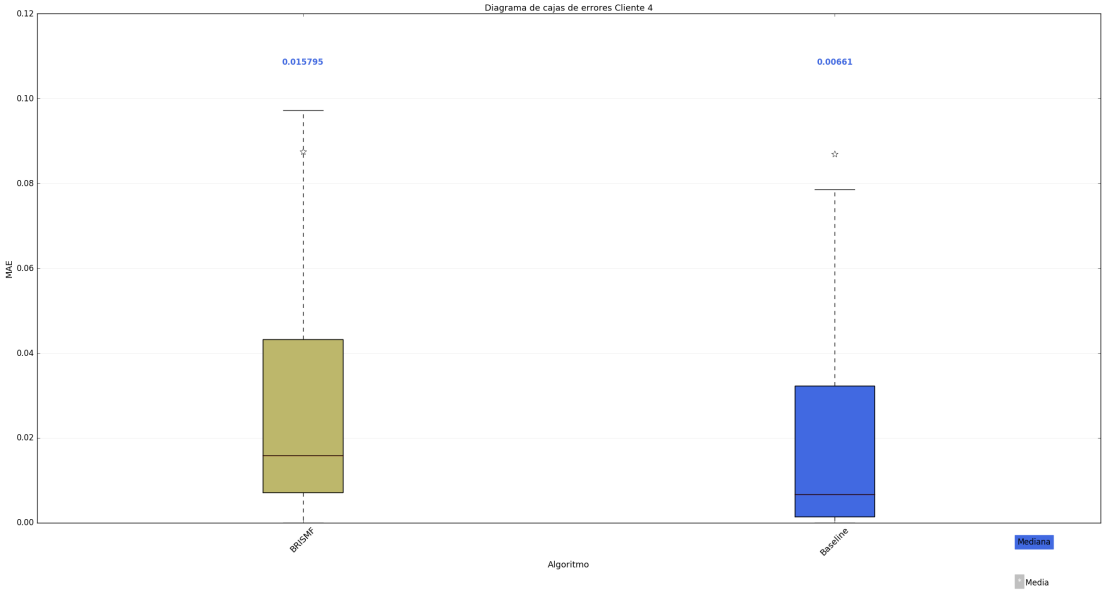


Figura B.16: Diagrama de cajas de los errores para el cliente 4

Apéndice C

Figuras de evolución del MAE en el uso de series temporales para la predicción del TTFB

En este anexo se añaden las figuras de la evolución del MAE en la aproximación a la predicción del TTFB como serie temporal para los algoritmos utilizados (Perceptron, FIMT-DD, ORTO y AMRules) y para el algoritmo de referencia.

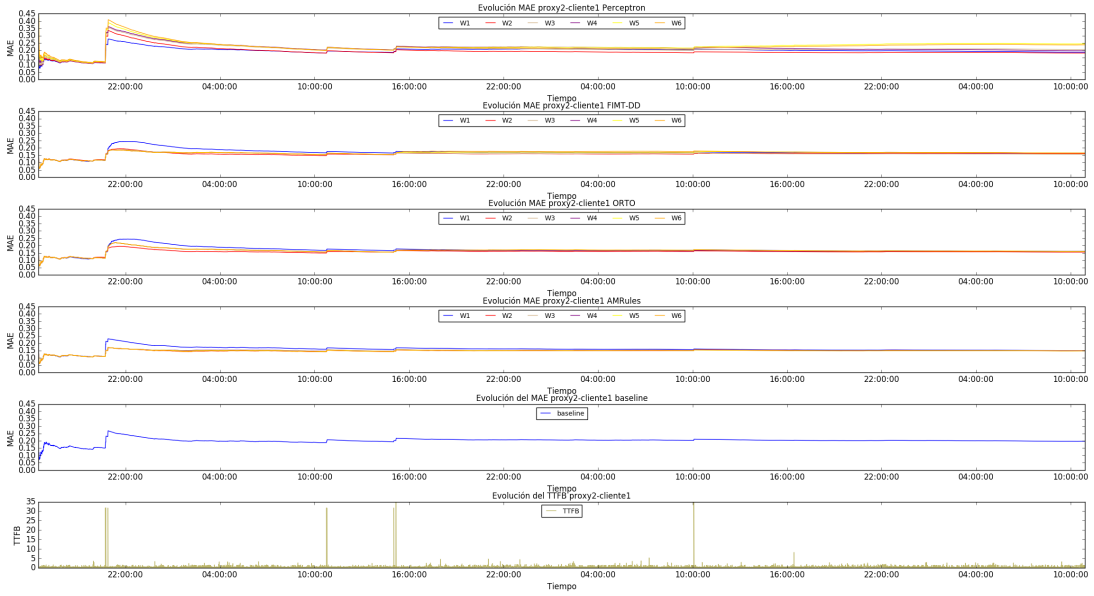


Figura C.1: Evolución del MAE frente a la evolución del TTFB en el *proxy* 2

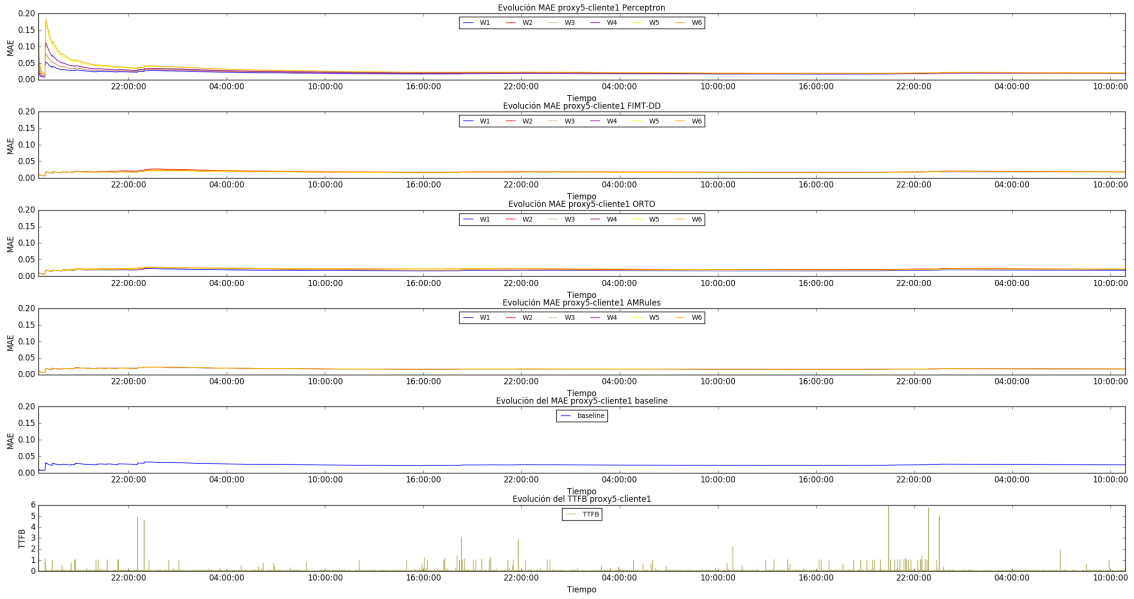


Figura C.2: Evolución del MAE frente a la evolución del TTFB en el *proxy* 5

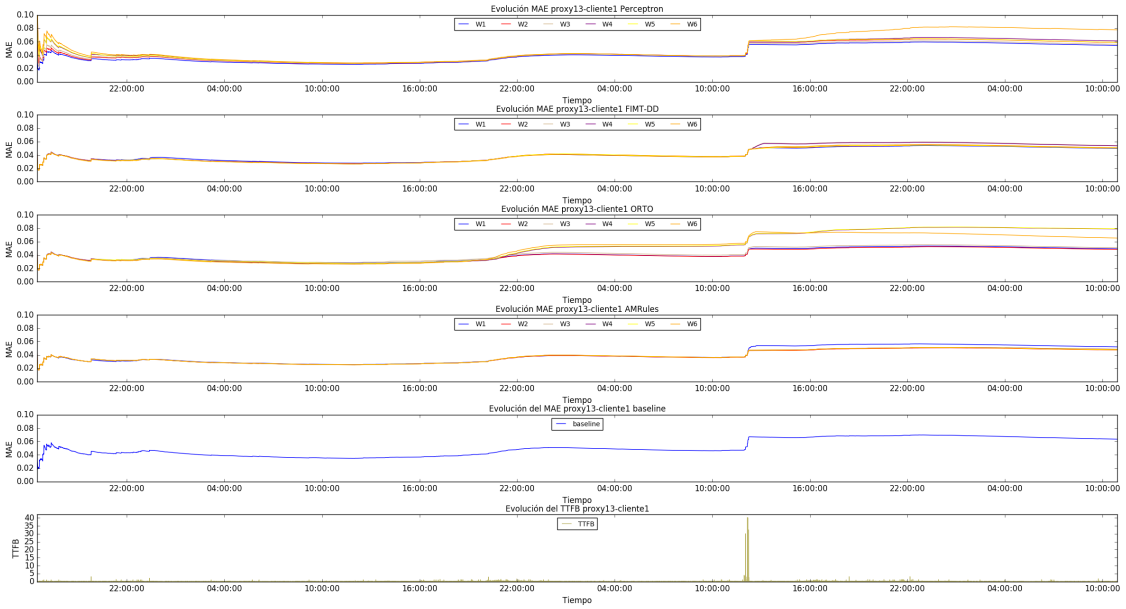


Figura C.3: Evolución del MAE frente a la evolución del TTFB en el *proxy* 13

Apéndice D

Información adicional sobre los experimentos

D.1. Código de la estimación mediante BRISMF

```
import moa.core.TimingUtils;
import java.io.IOException;
import moa.recommender.dataset.*;
import moa.recommender.rc.predictor.impl.BRISMFPredictor;
import moa.recommender.rc.data.impl.*;
import moa.recommender.dataset.impl.*;
import com.github.javacliparser.FileOption;
import  moa.tasks.TaskMonitor;
import java.util.*;
import java.lang.Math.*;
import java.io.*;
public class Predictor {

public Predictor(){
}
private int curCID;//Almacena el identificador del cliente de la linea actual
private double curRating;//Almacena el valor del TTFB de la linea actual
private double curPred;//Almacena la prediccion realizada
private double curTime;//Almacena el sello temporal de la linea actual
int prevCID;//Almacena el identificador del cliente de la linea anterior
double prevTime;//Almacena el valor del sello temporal de la linea anterior
private double lRate=0.001;//Valor del factor de aprendizaje
private double rFactor=0.02;//Valor del factor de regularizacion
private int numFeatures=10;//Tamano del vector latente
Map<Integer,Double> TTFB = new HashMap<Integer,Double>();//Almacena el TTFB correspondiente a las medidas de un
    cliente en un determinado instante


public void run(){
MemRecommenderData dataset = new MemRecommenderData();//Instancia que de la clase MemRecommenderData que contiene la
     matriz de puntuaciones
FileWriter fichero = null;
PrintWriter pw = null;
FileReader f = null;
BufferedReader b = null;
double error;//Almacena el valor del error absoluto instantaneo
try
{
fichero = new FileWriter("./Results_.txt");//Fichero donde se guardan los resultados
pw = new PrintWriter(fichero);
f= new FileReader("./Dataset_ordenado_sin_p8.txt");//Fichero donde se encuentran los datos
b= new BufferedReader(f);
BRISMFPredictor predictor = new BRISMFPredictor(numFeatures, dataset, lRate, rFactor, false);//Instancia de
    BRISMFPredictor que servira de predictor
predictor.setNIterations(1000);

String linea;//Almacena la linea del fichero de datos en cada iteracion
linea=b.readLine();//Se lee la primera linea de los datos

do {
```

```
    do {
49    String[] split = linea.split("::");//Se divide la linea de los datos
      curCID=Integer.valueOf(split[0]);//Se obtiene el identificador de cliente de la linea actual
51    //Si el cliente no esta en el sistema se anade
      if (dataset.userExists(curCID)==false)
53      {
        dataset.addUser(curCID, new ArrayList<Integer>(), new ArrayList<Double>());
55      }
      curTime=Double.valueOf(split[3]);//Se obtinen el valor del sello temporal de la linea actual
57    //Se almacenan los valores anteriores de ID del cliente y sello temporal para comparar
      prevCID=curCID;
59    prevTime=curTime;
      //Carga en el mapa TTFB las medidas de cada proxy para el miemo cliente y timestam
61    TTFB.put(Integer.valueOf(split[1]), Double.valueOf(split[2]));
      //Si el proxy no esta en el sistema se anade
63    if(dataset.itemExists(Integer.valueOf(split[1]))==false)
      {
65      dataset.addItem(Integer.valueOf(split[1]), new ArrayList<Integer>(), new ArrayList<Double>());
      }
67    //Se comprueba si los datos de la nueva linea corresponden al mismo cliente y al mismo instante
    }while((linea=b.readLine())!=null && Integer.valueOf(linea.split("::")[0])==prevCID && Double.valueOf(linea.split(
      "::")[3])==curTime);
69
    //Cuando se cambia de cliente:
71
    //En primer lugar se obtienen predicciones y errores absolutos antes de entrenar para cada cliente
73  for (int k=0; k<14;k++)
    {
75    if (TTFB.containsKey(k))//Si hay medidas del proxy con identificador k
      {
77      curPred=predictor.predictRating(curCID, k);//Se predice el TTFB
        error=Math.abs(curPred-TTFB.get(k));//Se calcula el error
79      pw.println(curCID + " " + k + " " + TTFB.get(k) + " " + curPred + " " +  error);//Se almacenan los resultados
      }
81  }
    //A continuacion se anaden al sistema las medidas.
83  //El metodo dataset.setRating() se encarga de entrenar los vaectores de usuario y articulo correspondientes
    for (int k=0; k<14;k++)
85  {
      if (TTFB.containsKey(k))
87    {
        dataset.setRating(curCID, k, TTFB.get(k));
89    }
    }
91  //Se resetea el mapa TTFB y se itera con las medidas del siguiente cliente
    TTFB.clear();
93  }while (linea!=null);//Se repite el preceso mientras no se haya acabado el documento
    b.close(); //Se cierra el fichero
95  } catch (Exception e) {
    e.printStackTrace();
97  } finally {
    try {
99    if (null != fichero)
        fichero.close();
101 } catch (Exception e2) {
      e2.printStackTrace();
103   }
  }
105
107
  }
109
  public static void main(String[] args) throws IOException {
111 Predictor exp = new Predictor();
  exp.run();
113 }
  }
115
```

D.2. Parámetros de los algoritmos de predicción

En esta sección se describen los parámetros utilizados para cada uno de los algoritmos de regresión utilizados. El valor de todos ellos es el que la interfaz de usuario indica por defecto, y que también se indicará a continuación.

D.2.1. Perceptron

Los parámetros de Perceptron que la interfaz permite modificar son:

- `learningRatio_Decay_set_constant`: permite indicar si la tasa de decrecimiento del factor de aprendizaje debe ser constante. Por defecto se indica que no.
- `learningRatio`: factor de aprendizaje. Por defecto vale 0,025.
- `learningRateDecay`: tasa de decrecimiento del factor de aprendizaje durante el entrenamiento del Perceptron. Por defecto vale 0,001.
- `fadingFactor`: factor de desvanecimiento del error acumulado. Por defecto vale 0,99.
- `randomSeed`: semilla para el comportamiento aleatorio. Por defecto vale 1.

D.2.2. FIMT-DD

Los parámetros de FIMT-DD que la interfaz permite modificar son:

- `splitCriterion`: criterio de división a utilizar por el árbol. Por defecto se utiliza `VarianceReductionSplitCriterion`.
- `gracePeriod`: número de instancias que una hoja debe observar antes de intentar divisiones. Por defecto vale 200.
- `splitConfidence`: error permitido en la decisión de división. Por defecto vale 0.
- `tieThreshold`:umbral por debajo del que una división es forzada a romper vínculos. por defecto vale 0,05.
- `PageHinckleyAlpha`: valor Alpha utilizado en el test de detección de cambios de Page Hinckley. Por defecto vale 0,005.
- `PageHinckleyThreshold`: valor del umbral utilizado en el test de detección de cambios de Page Hinckley. Por defecto vale 50.
- `alternateTreeFadingFactor`: factor de desvanecimiento utilizado para decidir si un árbol alternativo debería reemplazar al original. Por defecto vale 0,995.
- `alternateTreeTMin`: valor de Tmin utilizado para determinar si un árbol alternativo debería sustituir al original. Por defecto vale 150.

- `alternateTreeTime`: número de instancias utilizadas para decidir si un árbol alternativo debería ser descartado. Por defecto vale 1500.
- `regressionTree`: indica si se debe construir un árbol de regresión en lugar de uno de modelado. En los experimentos realizados se utiliza un árbol de regresión.
- `learningRatio`: factor de aprendizaje. Por defecto vale 0,02
- `learningRatioDecayFactor`: tasa de decrecimiento del factor de aprendizaje durante el entrenamiento. Por defecto vale 0,001.
- `learningRatioConst`: mantener la tasa de aprendizaje constante en lugar de hacerla decrecer. Por defecto no se mantiene constante.

D.2.3. ORTO

Los parámetros de ORTO que la interfaz permite modificar son:

- `maxTrees`: número máximo de árboles contenidos en el árbol opción. Por defecto vale 10.
- `maxOptionLevel`: Profundidad máxima a la que se pueden crear nodos de opción. Por defecto vale 10.
- `optionNodeAggregation`: método de agregación utilizado para combinar las predicciones en los nodos opción. Por defecto se utiliza la media.
- `optionFadingFactor`: factor de desvanecimiento utilizado para comparar subárboles de un nodo opción. Por defecto vale 0,9995.
- `splitCriterion`: criterio de división a utilizar por el árbol. Por defecto se utiliza `VarianceReductionSplitCriterion`.
- `gracePeriod`: número de instancias que una hoja debe observar antes de intentar divisiones. Por defecto vale 200.
- `splitConfidence`: error permitido en la decisión de división. Por defecto vale 0.
- `tieThreshold`: umbral por debajo del que una división es forzada a romper vínculos. por defecto vale 0,05.
- `PageHinckleyThreshold`: valor del umbral utilizado en el test de detección de cambios de Page Hinckley. Por defecto vale 50.
- `alternateTreeFadingFactor`: factor de desvanecimiento utilizado para decidir si un árbol alternativo debería reemplazar al original. Por defecto vale 0,995.
- `alternateTreeTMin`: valor de Tmin utilizado para determinar si un árbol alternativo debería sustituir al original. Por defecto vale 150.

- `alternateTreeTime`: número de instancias utilizadas para decidir si un árbol alternativo debería ser descartado. Por defecto vale 1500.
- `regressionTree`: indica si se debe construir un árbol de regresión en lugar de uno de modelado. En los experimentos realizados se utiliza un árbol de regresión.
- `learningRatio`: factor de aprendizaje. Por defecto vale 0,02.
- `learningRatioDecayFactor`: tasa de decrecimiento del factor de aprendizaje durante el entrenamiento. Por defecto vale 0,001.
- `learningRatioConst`: mantener la tasa de aprendizaje constante en lugar de hacerla decrecer. Por defecto no se mantiene constante.

D.2.4. AMRules

Los parámetros de AMRules que la interfaz permite modificar son:

- `splitConfidence`: error permitido en la decisión de división. Por defecto vale 1.0E-7.
- `tieThreshold`: umbral por debajo del que una división es forzada a romper vínculos. por defecto vale 0,05.
- `gracePeriod`: número de instancias que una hoja debe observar antes de intentar divisiones. Por defecto vale 200.
- `learnerOption`: algoritmo de aprendizaje. Por defecto se utiliza un regresor multiobjetivo adaptativo.
- `setUnorderedRulesOn`: permite el uso de reglas sin orden. Por defecto no se permiten.
- `dropOldRuleAfterExpansion`: descartar la regla antigua si se expande. Por defecto no se hace.
- `changeDetector`: detector de cambios. Por defecto se utiliza el de Page Hinkley.
- `anomalyDetector`: se trata del detector de anomalías utilizado. Por defecto se utiliza `OddsRatioScore`.
- `splitCriterionOption`: criterio de división utilizado para determinar el mérito de una división. Por defecto se utiliza `MultiTargetVarianceRatio`.
- `errorMeasurer`: medida del error para decidir que algoritmo debería predecir. Por defecto se usa `RelativeMeanAbsoluteDeviationMT`.
- `weightedVoteOption`: la opción que en la interfaz aparece por defecto es `InverseErrorWeightedVoteMultiLabel`.

- `numericObserver`: se trata del observador numérico. Por defecto la opción que se utiliza es `MultiLabelBSTree`.
- `nominalObserver`: se trata del observador nominal. Por defecto se utiliza la opción `MultiLabelNominalAttributeObserver`.
- `verbosity`: nivel de información a la salida. Por defecto vale 1.
- `outputSelector`: selector de atributos de salida. Por defecto la opción utilizada es `SelectAllOutputs`.
- `inputSelector`: selector de atributos de salida. Por defecto se utiliza la opción `SelectAllInputs`.
- `randomSeedOption`: Por defecto vale 1.
- `featureRanking`: algoritmo de ordenación de características. Por defecto la opción que se utiliza es `NoFeatureRanking`.