UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA

DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Predicción de calidad de enlace en redes comunitarias inalámbricas basadas en OLSR

Autor:

D. Carlos Mediavilla Pastor

Tutores:

Dr. D. Miguel Luis Bote Lorenzo

Dr. D. Eduardo Gómez Sánchez

Valladolid, 18 de julio de 2017

TÍTULO:	**Predicción de calidad de enlace en redes comunitarias inalámbricas basadas en OLSR**
AUTOR:	**D. Carlos Mediavilla Pastor**
TUTORES:	**Dr. D. Miguel Luis Bote Lorenzo** **Dr. D. Eduardo Gómez Sánchez**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dr. D. Juan Ignacio Asensio Pérez**
VOCAL:	**Dr. D. Ignacio de Miguel Jiménez**
SECRETARIO:	**Dr. D. Federico Simmross Wattenberg**
FECHA:	**24 de julio de 2017**
CALIFICACIÓN:	

Resumen de TFG (150 palabras)

Las redes comunitarias inalámbricas son redes en malla, creadas por varios usuarios que residen en una zona común, dando acceso a servicios como el acceso a Internet, entre otros. Los enlaces de estas redes son muy poco fiables, por lo que se utilizan protocolos de encaminamiento como OLSR, que utiliza la calidad de enlace como métrica de coste. El uso de técnicas de aprendizaje automático puede ser útil a la hora de predecir el estado futuro de los enlaces, y consecuentemente, mejorar el encaminamiento. A este respecto, trabajos previos han estudiado cómo el aprendizaje automático en lote puede predecir la calidad de los enlaces de una red. No obstante, los modelos creados implican una carga computacional no asumible en entornos reales. Además, estos modelos se degradan con el tiempo. El objetivo de este TFG es proponer nuevos métodos que mantengan una carga computacional reducida y que además realicen una buena predicción.

Palabras clave

Redes de comunidad, redes en malla inalámbricas, predicción de calidad de enlace, aprendizaje automático, coste computacional.

Abstract (150 words)

Wireless Community Networks are mesh networks, built by several users living in the same area, providing access to services such as Internet access, among others. The links of these networks are very unreliable, that is why routing protocols like OLSR, which uses link quality as cost metric, are used. Machine learning techniques can be useful for predicting future network state, and therefore, improving the operation of it. In this context, previous works have studied how batch machine learning can forecast link quality of a network. Nevertheless, the created models imply unacceptable computational cost in real environments. Besides, these models degrade as time passes. The objective of this project is to propose new methods which maintain low computational cost, and make good predictions.

Keywords

Community networks, wireless mesh networks, link quality prediction, online machine learning, computational cost.

Agradecimientos

Antes de comenzar este trabajo, quería dar las gracias a todas las personas, que, de una u otra manera, me han ayudado a terminar esta etapa de mis estudios. En primer lugar, en lo académico, gracias a todos los profesionales de la ETSIT que han puesto todo su empeño para formarnos como Ingenieros. Especialmente quería dar las gracias a los tutores de este Trabajo Fin de Grado, Miguel y Eduardo, que previamente fueron mis profesores, por todas las dudas resueltas a lo largo de sus asignaturas, así como durante este Trabajo, que no han sido pocas. Del mismo modo, quería dar las gracias a Juan Ignacio, que, si bien no ha sido tutor del Trabajo, ha participado activamente en las reuniones. Por otro lado, y sin alejarme de la Universidad, quería dar las gracias a mis compañeros de clase, ya que sin ellos estos cuatro años se habrían hecho mucho más largos.

En lo personal, quería dar las gracias en primer lugar a mi familia más cercana, mi padre, mi madre, mi hermano. Gracias por aguantarme este año, que ha sido complicado y demasiado atareado. También quería dar las gracias a mi novia, por entenderme como nadie y darme su apoyo en los momentos que más lo necesitaba, gracias de corazón.

Finalmente, quería dar las gracias a mis amigos de toda la vida, que, pese a que en los últimos años nos vemos menos, cuando salimos todo sigue siendo igual que siempre; esperemos que no se pierda nunca.

Gracias a todos.

Índice

Lista de figuras

Lista de tablas

"There are three things all wise men fear: the sea in storm, a night with no moon, and the anger of a gentle man."

Patrick Rothfuss, The Wise Man's Fear

Capítulo 1. Introducción

1.1. Motivación

Una red comunitaria es una infraestructura de red extendida a lo largo de varias localizaciones en un vecindario, y a la cual los miembros de la comunidad tienen permiso de acceso [1]. Las redes comunitarias o redes de comunidad son infraestructuras distribuidas, descentralizadas y a gran escala compuestas de varios nodos, enlaces y servicios cuyos recursos están disponibles a un grupo de personas viviendo en la misma zona [2]. Las redes comunitarias son construidas por una gran variedad de razones. Algunas proporcionan acceso a Internet a localizaciones remotas donde el coste es demasiado alto para los proveedores comerciales de conexión a Internet [1], otras simplemente son creadas debido a las posibilidades técnicas de las redes [3]. Hay redes comunitarias construidas con unos pocos nodos y otras con miles de ellos. Las redes comunitarias crecen y cambian con el paso del tiempo. Los nodos y los enlaces pueden fallar y necesitan mantenimiento, además de que nuevos nodos y enlaces pueden ser añadidos. Aunque algunas redes comunitarias utilizan profesionales a sueldo para instalaciones y mantenimiento, la mayoría de las redes de comunidad son administradas por voluntarios [3]. En muchas ciudades europeas se están creando redes de comunidad inalámbricas con decenas de nodos. En Atenas una única red comunitaria inalámbrica (AWMN) incluye más de 2400 nodos, mientras que en España la red Guifi es una composición de varias redes comunitarias inalámbricas con más de 33000 nodos y creciendo [2]. Esto permite que miles de nodos conecten decenas de miles de familias, asociaciones y oficinas públicas, con un enfoque no lucrativo y basado en la organización de las comunidades.

Aunque la variedad de redes comunitarias es alta, la mayoría de ellas tienen objetivos y desafíos comunes [3]. Los principales retos son en primer lugar los organizacionales, fundamentalmente debidos a la necesidad de obtener fondos para mantener y expandir la red. Por otra parte, el mantenimiento también destaca como uno de los problemas principales, incluyendo monitorización, actualizaciones y configuración. Debido a la naturaleza de estas redes, muchos de los enlaces son inalámbricos, y por ello presentan algunos retos como la alta variabilidad de la calidad del enlace (LQ, del inglés *link quality*), lo que supone el principal problema para muchas de ellas. El protocolo de encaminamiento utilizando en este tipo de redes es mayoritariamente OLSR (*Optimized Link State Routing*) [3], el cual calcula las mejores rutas desde cada nodo hasta el resto de nodos de la red, minimizando el

coste de cada una de ellas. Este coste se puede definir de varias formas, entre ellas, a partir de la calidad de los enlaces, es decir, su valor de LQ.

A este respecto, trabajos previos [4] afirman que el seguimiento de la LQ permite a los protocolos de encaminamiento evitar los enlaces asimétricos (dado que la métrica de coste los penaliza), aunque puede que el seguimiento no sea suficiente, y se usen datos anticuados para encaminar los nuevos paquetes [5]. En este sentido, la predicción de LQ se ha demostrado como una mejor técnica frente al seguimiento de la LQ [6], ya que permite tomar las decisiones de encaminamiento con el conocimiento del futuro próximo de la red, que es cuando los paquetes van a ser encaminados, y no con la información actual, que es posible que varíe hasta dicho momento [7]. En otro contexto semejante [8], el de las redes inalámbricas de sensores (WSN, *Wireless Sensor Networks*), también se ha propuesto la predicción de la calidad del enlace como una aproximación interesante de cara a disminuir la probabilidad de pérdida de datos. Hay que hacer notar que las redes de comunidad y las redes de sensores, pese a tener objetivos diferentes (unas comunican usuarios, otras transmiten parámetros medidos), comparten la misma problemática (topología dinámica, comunicación multisalto, dispositivos poco potentes).

Por otro lado, en [7] siguen un enfoque basado en aprendizaje automático para la predicción de la LQ en la red comunitaria FunkFeuer de Austria, utilizándose técnicas aplicadas a la predicción de series temporales, concretamente, técnicas de regresión (*regression*) y agrupación (*clustering*). En dichos experimentos se obtienen buenos resultados utilizando técnicas de regresión como Árboles de Regresión (*Regression Trees*) o Máquinas de Soporte Vector (*Support Vector Machines*).

No obstante, como muestra [7], para poder mantener la calidad de la predicción a lo largo del tiempo se hace necesario construir modelos actualizados cada cierto tiempo, con una frecuencia tal que puede hacer inviable su aplicación en entornos reales dado el alto coste computacional necesario. Una de las principales características de un predictor es su adaptabilidad [9], dado que en redes muy dinámicas el predictor debería ser capaz de adaptarse a los cambios. Otra característica importante es la capacidad del predictor de ser *plug and play*, es decir, que sea capaz de funcionar en cualquier red sin necesidad de realizar un proceso previo de adaptación. Esto último es muy relevante, ya que en redes de comunidad donde los enlaces pueden aparecer y desaparecer de manera constante, nos interesa ser capaces de poder predecir lo antes posible. Los dispositivos de red en las redes comunitaria no son grandes ordenadores de altas prestaciones, sino pequeños encaminadores económicos con poca capacidad de procesamiento. Es en este contexto en el que el planteamiento de una predicción incremental y de bajo coste

computacional tiene sentido, con el objetivo de ser implementable. Si finalmente se logra proponer un sistema de predicción incremental de LQ suficientemente preciso, los algoritmos de encaminamiento serán capaces de evitar los enlaces que vayan a fallar en el futuro, y de esta manera, minimizará el número de paquetes perdidos, dando un mejor servicio a los usuarios de la red de comunidad.

1.2. Objetivos

En este trabajo se pretende continuar el previamente realizado en [7], centrándose en la mejora de los métodos propuestos tanto en términos de coste computacional (para poder adaptar periódicamente los modelos de predicción) como de eficacia en la predicción. Es por ello que el objetivo principal de la investigación será el siguiente:

- Proponer algoritmos incrementales de predicción de LQ que mejoren en primer lugar los resultados obtenidos por [7] y por otra parte disminuyan su coste computacional permitiendo su uso en entornos reales.

Para poder llevar a cabo el objetivo propuesto, hay varios objetivos parciales que han de tenerse en cuenta:

- En primer lugar, tenemos que replicar los experimentos realizados por [7], bajo las mismas condiciones y asumiendo los mismos parámetros, para obtener unos resultados semejantes a los reportados por estos autores. Estos resultados serán la línea de partida sobre la que comparar el resto de experimentos realizados
- En segundo lugar, se plantearán algoritmos de aprendizaje incremental que, por una parte, obtengan buenos resultados (comparables o mejores a los anteriores) y por otra parte sean capaces de mantener una carga computacional reducida (significativamente menor a los anteriores)
- Finalmente, se podrán probar otras alternativas derivadas del trabajo precedente, como reentrenar los modelos cada cierto tiempo, o con otros parámetros.

1.3. Metodología de trabajo

Durante el desarrollo de este Trabajo Fin de Grado se ha seguido la metodología de trabajo propuesta por el método de ingeniería [10], adaptándola a nuestro contexto particular, teniendo en cuenta que el objetivo de nuestro trabajo no es crear un producto sino el estudio de nuevos métodos para el problema planteado.

El método de ingeniería es una aproximación sistemática que da soporte a un ingeniero o a un equipo en obtener la solución deseada a un problema, que ha sido especificado por clientes, patrocinadores o inversores que perciben valor en resolver el problema. El método de ingeniería tiene seis etapas:

1. Idea:
   La fase de "Idea" suele empezar con un problema. El problema requiere investigación sobre su viabilidad y su factibilidad. La viabilidad sugiere que existe un valor significativo en obtener la solución. La factibilidad sirve para comprobar si la idea puede ser realizada.

2. Concepto:
   La fase de "Concepto" consta de la generación de numerosos modelos (matemáticos, físicos, simulaciones, simples dibujos o esquemas), todos los cuales deberían expresar los requerimientos del cliente.

3. Planificación:
   La fase de "Planificación" se centra en definir el plan de implementación: identificar el capital humano, las tareas, la duración de las tareas, las dependencias de las tareas, las interconexiones de las tareas, y el presupuesto necesario para conseguir realizar el proyecto.

4. Diseño:
   En la fase de "Diseño" se especifican los detalles, las especificaciones son establecidas. El propósito de esta fase es trasladar los requerimientos del cliente en especificaciones de ingeniería con las que un ingeniero (diseñador) pueda trabajar para diseñar y construir un prototipo.

5. Desarrollo:
   El propósito de la fase de desarrollo es generar la documentación ingenieril: esquemas, dibujos, código fuente, y cualquier otra cosa que demuestre la solución al problema. La solución puede ser un prototipo funcional tangible o bien una simulación funcional intangible. Específicamente, consiste del ciclo iterativo: diseño, testeo, depuración y rediseño.

6. Lanzamiento:

El “Lanzamiento” incluye la emisión del diseño ingenieril y el paquete de documentación a instalaciones de fabricación para producción. En este punto, todas las pruebas de calificación están completas y el prototipo de trabajo ha demostrado funcionalidad.

En nuestro caso particular, no todas las etapas del método de ingeniería aparecen reflejadas en nuestro trabajo. A continuación, se muestran las distintas etapas del método de ingeniería y cómo se ha particularizado el método para los objetivos de esta investigación.

1. Idea:
   En la etapa de “Idea” se ha entendido el problema existente presentado en la Motivación y detallado en los Objetivos, consistente en obtener una predicción incremental de la LQ con una baja carga computacional. Es una idea viable, ya que proporciona un valor añadido sobre el estado del arte actual y es factible, ya que los medios materiales necesarios son únicamente un ordenador con las herramientas *software* requeridas.

2. Concepto:
   Sobre la etapa de “Concepto” no se ha hecho un gran hincapié, sino que ha sido a lo largo de todo el proceso cuando se han planteado diferentes ideas sobre qué mejoras o variaciones había que realizar sobre la iteración anterior para mejorar los resultados.

3. Planificación:
   En la etapa de “Planificación” no se ha incidido en gran medida, dado que el trabajo realizado ha sido desarrollado de manera individual, y no había costes económicos que financiar.

4. Diseño:
   De manera paralela a la etapa de “Desarrollo”, en la etapa de “Diseño” se han diseñado los experimentos que posteriormente se han implementado. Dado que se han realizado varios experimentos, sobre esta etapa se ha iterado aproximadamente el mismo número de veces que sobre la etapa de “Desarrollo”, exceptuando aquellos casos de depuración de errores.

5. Desarrollo:

En la etapa de “Desarrollo” se ha centrado la mayor parte del tiempo de este Trabajo Fin de Grado. Para lograr completar esta etapa, se han realizado varias subtareas que vale la mención reseñar:

- En primer lugar, se han extraído los datos relevantes de los ficheros de registro disponibles de la red Funkfeuer. El dato principal que se ha extraído de estos ficheros ha sido la LQ.
- Después, se ha seleccionado un subconjunto de enlaces del total, de acuerdo con los criterios establecidos en [6], para poder realizar los experimentos desde el mismo punto de partida que el propuesto.
- A continuación, se han realizado los experimentos relativos a la predicción de la LQ de los diferentes enlaces seleccionados, entrenando el predictor, y después testeando con una porción disjunta de la utilizada en el entrenamiento. Dado que en estos casos la ordenación temporal de los valores es relevante, el conjunto de test es el conjunto de datos consecutivo en el tiempo respecto al de entrenamiento.
- Finalmente, para cada uno de los experimentos realizados, se han sacado las conclusiones pertinentes, que han permitido o bien realizar nuevos experimentos con diferentes parámetros, o bien extraer información sobre el problema en cuestión.

En esta etapa se han realizado varias iteraciones, tanto para ajustar diversos parámetros de los experimentos en cuestión, como para depurar errores. Además, se han realizado diferentes experimentos para extraer información relevante sobre diferentes aspectos de la predicción de la LQ.

6. Lanzamiento:
   La etapa de “Lanzamiento” no ha existido como tal en el desarrollo de este trabajo dado que no va a pasar a producción, se podría considerar este documento como la etapa de “Lanzamiento” de este Trabajo Fin de Grado.

1.4. Estructura del documento

En la parte restante del documento se detallará el trabajo realizado en relación con la predicción de la LQ.

En el capítulo 2 se analizará el estado del arte desde diferentes perspectivas. Por una parte, se estudiarán las redes de comunidad, su desarrollo, sus utilidades y sus problemáticas También se analizarán los protocolos de encaminamiento utilizados en las redes de comunidad, Además, se analizarán los diferentes tipos de aprendizaje automático existentes en la actualidad, así como los conjuntos de datos disponibles sobre los que poder realizar aprendizaje. Por otra parte, se analizarán los estudios previos existentes sobre la predicción de la LQ, que incluyen tanto el punto de partida de nuestro trabajo (condiciones de partida), como los resultados con los que hemos de compararnos.

En el capítulo 3 se mostrará la reproducción realizada de los experimentos realizados por [7], incidiendo en las semejanzas y diferencias encontradas en los resultados obtenidos.

En el capítulo 4 se expondrán los nuevos experimentos realizados, y su comparativa con los desarrollados previamente en el capítulo 3.

Finalmente, en el capítulo 5 desarrollaremos las conclusiones extraídas del estudio realizado. Además, se plantearán las posibles líneas de trabajo futuro para obtener una perspectiva global donde se localice el trabajo.

Capítulo 2. Estado del Arte

2.1. Introducción

Como se ha mencionado anteriormente, el objetivo de este Trabajo Fin de Grado es mejorar los resultados obtenidos por [7], a la par que se reduce la carga computacional de los algoritmos utilizados. Para poder cumplir este objetivo, es necesario en primer lugar profundizar un poco más en las características del problema en cuestión.

En primer lugar, en este capítulo se va a hablar de las redes de comunidad, sus problemáticas y los protocolos de encaminamiento que se utilizan en ellas. Por otra parte, se va a profundizar en el problema de la predicción de la calidad del enlace, para lo cual se va a estudiar el aprendizaje automático, así como los *toolkits* existentes en la actualidad. Finalmente, se comentarán brevemente los conjuntos de datos disponibles y se extraerán unas breves conclusiones.

2.2. Redes de comunidad

Para las comunicaciones inalámbricas de área local, la emergencia del estándar IEEE 802.11[1], conocido comercialmente como Wi-Fi, supuso un gran paso hacia adelante [11]. Conducido por su bajo coste y su facilidad de despliegue, los dispositivos compatibles con el estándar 802.11 se convirtieron en equipamiento estándar en portátiles y dispositivos móviles, presentándose como la tecnología predominante para conexiones inalámbricas locales. La operación en bandas del espectro sin licencia facilitó el despliegue de Wi-Fi, dado que era un camino sencillo para los operadores comerciales, instituciones académicas, o incluso entusiastas de las radiocomunicaciones.

En las modernas zonas urbanas con mucha densidad de población, la cobertura que las WLAN proporciona es en constante crecimiento. Una simple búsqueda de redes inalámbricas nos revela un elevado número de puntos de acceso (AP, *Access Point*), por lo que actualmente el problema de la cobertura no es muy relevante. La facilidad de despliegue de Wi-Fi lo ha hecho omnipresente en áreas urbanas densamente pobladas, y solo era cuestión de tiempo que las redes comunitarias inalámbricas emergieran.

Las redes comunitarias inalámbricas (WCN, *Wireless Community Networks*) han sido desarrolladas gracias a grupos de entusiastas, que, usando equipamiento

[1] Estándar IEEE 802.11: disponible en https://standards.ieee.org/about/get/802/802.11.html (Última consulta:17 de julio de 2017)

de redes barato para realizar interconexiones gratuitas, crean redes inalámbricas autónomas [11]. Estas redes proporcionan una gran variedad de servicios, destacando entre ellos las llamadas VoIP entre los miembros de la comunidad y la compartición de ficheros.

El incremento de cobertura Wi-FI y la emergencia de las WCN hace que surja la cuestión de si alternativas de bajo coste a los servicios tradicionales ofrecidos por los sistemas de telefonía móvil (3G, 4G) pueden ser conseguidas. Además, las WCN podrían utilizarse como proveedor de acceso residencial a Internet, compartiendo el acceso entre varios hogares. No obstante, antes de que este escenario pueda ser realidad, han de resolverse algunos problemas como los incentivos a los propietarios privados de WLAN para que den acceso libre a sus AP o algunos problemas de seguridad. A continuación, clasificaremos las WCN en función de quién sea el organizador de la iniciativa, y también en función de qué alternativa de diseño se haya adoptado.

En primer lugar, la red puede ser iniciativa de la comunidad, es decir, ser el resultado de los esfuerzos de voluntarios individuales, sin ánimo de lucro. Por otra parte, a veces la iniciativa es comercial, siendo en este caso las empresas mediadoras para el desarrollo de WCN. A este respecto, han surgido iniciativas que permiten a los usuarios compartir sus WLAN a cambio de una pequeña compensación económica, donde podría destacar FON[2]. Finalmente, los gobiernos municipales o regionales a veces ponen AP en espacios públicos, permitiendo acceso gratuito o económico a los usuarios.

Por otra parte, la red puede estar diseñada como una malla inalámbrica, típicamente cuando es iniciativa de la comunidad, dando alguno de los nodos acceso a Internet. Por otra parte, puede estar basada en *hotspots*. En la Figura 1 podemos apreciar la diferencia entre las dos arquitecturas presentadas.

Independientemente de quien organice la iniciativa y de la arquitectura de la WCN, existe un aspecto, existe una problemática común, que es el cálculo de las rutas que han de seguir los paquetes. Cuando haya más de un salto entre el AP al que se conecta el usuario y la salida a Internet, se hace necesario el uso de protocolos de encaminamiento que calcule la mejor ruta en cada caso.

[2] FON, disponible en http://www.fon.com (Última consulta: 17 de julio de 2017)

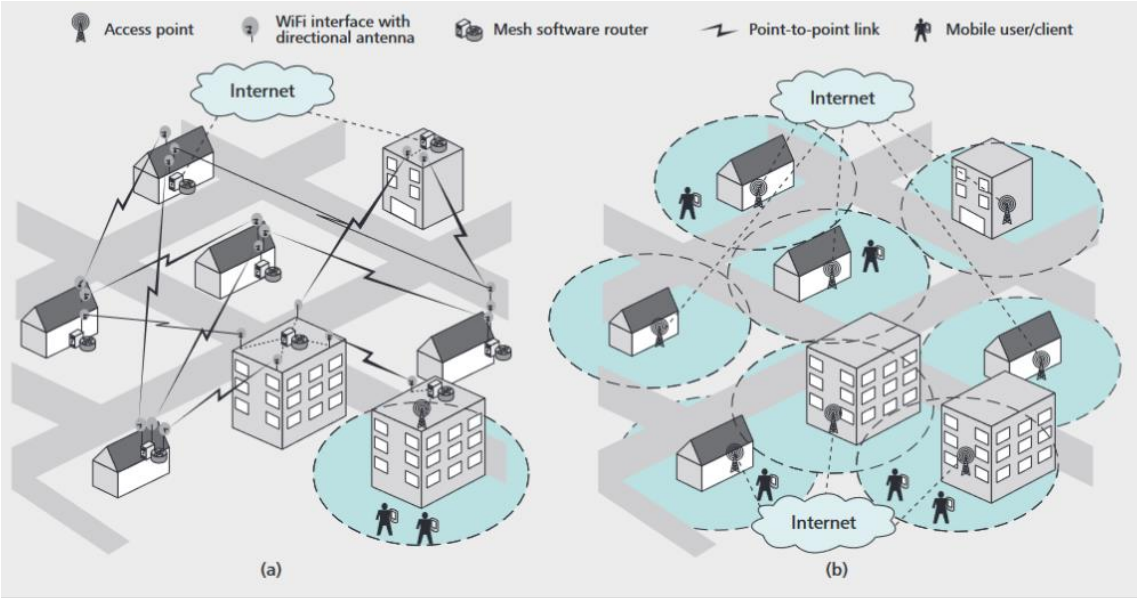


Figura 1: alternativas estructurales para redes comunitarias inalámbricas. Una arquitectura basada en una malla inalámbrica se muestra en (a), mientras que la alternativa basada en hotspots se muestra en (b).[3]

2.2.1. Protocolos de encaminamiento

Para poder comunicar dos nodos en una red, es necesario encontrar rutas que atraviesen nodos intermedios, optimizando algún criterio relacionado con el coste económico o con la calidad de servicio ofrecida (por ejemplo, menor número de saltos, menor retardo extremo a extremo, menor probabilidad de pérdida, etc).

Este problema, que en una red cableada podría resolverse de forma estática o semi-estática, se complica en gran medida cuando los nodos que se comunican son inalámbricos, pudiéndose destacar dos posibles fuentes de problemas. En primer lugar, la ubicación de los nodos esta potencialmente sujeta a cambios de posición, por lo que pueden salir de la cobertura del resto de nodos. Por otra parte, dado que en una WCN normalmente los equipos los ponen los usuarios, puede que no estén disponibles todo el tiempo, ya que estos usuarios pueden apagarlos.

Respecto al primer problema, en una WCN no habría que preocuparse, dado que los equipos suelen estar fijos, y solo los terminales finales son los que se mueven, si se estuviese estudiando otro tipo de redes, como las MANET (*Mobile Ad hoc NETwork*), que son redes malladas en las que todos los dispositivos están potencialmente en movimiento, sí que habría que tener esta consideración.

Sea como fuere, en las WCN se hace necesario establecer algún mecanismo con el que poder establecer las rutas más fiables, estables y con menor probabilidad de pérdida, entre el origen y el destino. De esto se encargan los protocolos de

[3] Figura tomada de [11].

encaminamiento, que a continuación vamos a explicar. No obstante, en primer lugar, se hace preciso hablar de los algoritmos de encaminamiento.

Para formular los problemas de encaminamiento se utilizan grafos [12]. Un grafo *G = (N, E)* es un conjunto de *N* nodos y una colección de *E* aristas, donde cada arista une dos nodos de N. En el contexto del encaminamiento de la capa de red, los nodos del grafo representan los encaminadores (es decir, el lugar donde las decisiones de reenvío de paquetes son tomadas) y las aristas representan los enlaces físicos entre dichos encaminadores. En la Figura 2 se puede ver un ejemplo de un grafo que representa una red de ordenadores, donde cada arista también tiene un valor que representa su coste, que podría reflejar la longitud física de un enlace (por ejemplo, un enlace transoceánico debería tener un mayor coste que un corto enlace terrestre), pero también podría tener en cuenta otras consideraciones como la velocidad del enlace, o incluso el coste económico asociado al uso del enlace. No obstante, de cara a los algoritmos de encaminamiento es irrelevante de qué manera se haya calculado. Un algoritmo de encaminamiento optimiza el cálculo de rutas entre cualquier par de todos en función de dicho coste.

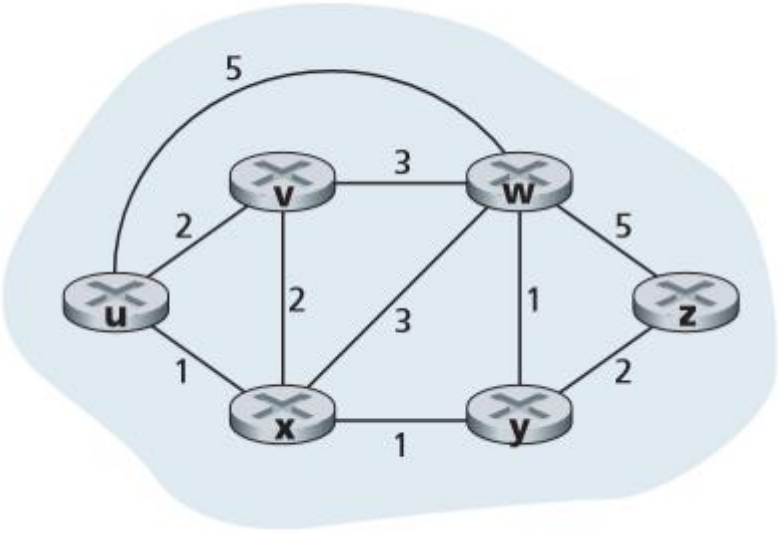


Figura 2: grafo de una red de ordenadores[4].

Por otra parte, los algoritmos de encaminamiento pueden calcularse de manera distribuida (cada nodo obtiene la información de la parte de la red más cercana a él, y optimiza las rutas de su entorno con la esperanza de que esto optimice la red de manera global), Típicamente a los algoritmos distribuidos se les conoce como algoritmos de vector de distancias (DV, *Distance-Vector*), dado que cada nodo

[4] Figura tomada de [12].

mantiene un vector de los costes estimados (distancias) al resto de nodos en la red. Uno de más importante es el algoritmo de Bellman-Ford distribuido. También pueden calcularse de manera centralizada (un nodo obtiene la información de la red completa, calcula las rutas aplicando el algoritmo de encaminamiento sobre el grafo, y finalmente distribuye las rutas). Además, podría seguirse un enfoque centralizado en cada nodo, por lo que cada uno de ellos debería obtener la información completa de la red, pero después ya no sería necesario distribuir las rutas. En la práctica, los algoritmos con información de estado global son mencionados como de estado de enlace (LS, *Link-State*), dado que el algoritmo debe ser consciente del coste de cada enlace en la red. Uno de más importante es el algoritmo de Dijkstra.

Por otra parte, los algoritmos de encaminamiento pueden ser estáticos (cuando las rutas cambian de manera muy lenta a lo largo del tiempo) o dinámicos (las rutas varían a medida que varía el tráfico o la topología de la red). Al contrario que los algoritmos utilizados por los protocolos más frecuentes de Internet, como RIP[5], OSPF[6], o BGP[7], que únicamente intercambian información sobre el número de saltos de cada ruta, en una WCN se hace necesario considerar métricas que reflejen la calidad de los enlaces presentes en la misma, con el objetivo de maximizar la fiabilidad y la estabilidad de las rutas.

2.4. OLSR (*Optimized Link State Routing Protocol*)

Como se ha comentado anteriormente, los protocolos que se utilizan mayoritariamente en Internet son RIP, OSPF y BGP. No obstante, estos protocolos no están optimizados para tipos concretos de redes, en concreto, para las WCN, debido a que las técnicas de inundación utilizadas para transmitir la información no son eficientes, y pueden sobrecargar la red. En esta sección se describirá detalladamente el funcionamiento del protocolo OLSR, el principal protocolo de encaminamiento utilizado en WCN [3], pese a que se utilicen otras alternativas, como pueden ser BGP - ya mencionado anteriormente - o BATMAN (*Better Approach to Mobile Adhoc Networking*), que pretende ser el reemplazo futuro de OLSR. No obstante, algunos autores no comparten la idea de crear un nuevo protocolo para este propósito, por lo que hay otro tipo de propuestas que pretenden extender el funcionamiento del protocolo OSPF, como por ejemplo las extensiones de OSPF para

[5] RIP versión 2: disponible en https://tools.ietf.org/html/rfc2453 (Última consulta: 17 de julio de 2017)

[6] OSPF versión 2: disponible en https://tools.ietf.org/html/rfc2328 (Última consulta: 17 de julio de 2017)

[7] BGP versión 4: disponible en https://tools.ietf.org/html/rfc4271 (Última consulta: 17 de julio de 2017)

MANET, que se pueden consultar en los RFC 5614[8] y 5820[9]. Se explicarán tanto los estándares de OLSR como las implementaciones existentes.

Originalmente, OLSR fue propuesto en [13], como un protocolo para redes móviles inalámbricas. Esta primera concepción, así como la que acabó convirtiéndose en estándar en el RFC 3626[10], no empleaban el valor de la calidad del enlace a la hora de realizar el encaminamiento [14]. No obstante, primero se explicará el funcionamiento primigenio del protocolo y después de qué manera se añadió esta característica.

De acuerdo a la publicación original [13], OLSR es un protocolo de encaminamiento de estado de enlace proactivo (va recalculando periódicamente las rutas, aunque no se necesiten). El protocolo OLSR es una optimización de un protocolo de estado de enlace, pensado para redes móviles *ad hoc* (MANET). En primer lugar, cada nodo en la red selecciona un conjunto de nodos de entre sus nodos vecinos, los cuales retransmitirán sus paquetes. Este conjunto de nodos vecinos seleccionados es llamado conjunto de *multipoint relays* (MPRs) de este nodo. El resto de vecinos de un nodo que no estén dentro de su conjunto MPR, leen y procesan el pero no retransmiten los paquetes de *Broadcast*. Para esta finalidad, cada nodo mantiene un conjunto de sus vecinos que son llamados el MPR *Selectors* del nodo. Cada mensaje de *Broadcast* con origen un vecino del MPR *Selectors* es retransmitido. Este conjunto puede cambiar a lo largo del tiempo, que es indicado por los nodos selectores en los mensajes HELLO que se explicarán a continuación.

Cada nodo elige su conjunto MPR de entre sus nodos vecinos de manera que cubra (en términos de cobertura de radio) todos los nodos que están a una distancia de dos saltos. Es decir, que cada nodo en el vecindario de dos saltos de un nodo tiene que tener un enlace bidireccional con el conjunto MPR. Cuando menor es el conjunto MPR, más óptimo es el protocolo de encaminamiento. En la Figura 3 se muestra el conjunto MPR alrededor del nodo N.

La idea de los *multipoint relays* es minimizar el reenvío de los paquetes de *Broadcast*, reduciendo la duplicidad de retransmisiones en la misma zona. Solamente los *multipoint relays* de un nodo retransmiten los mensajes *Broadcast*. Esta técnica reduce de manera significativa el número de retransmisiones en un procedimiento de inundación. El protocolo es particularmente apropiado para redes

[8] *MANET Extension of OSPF using CDS Flooding*: disponible en http://www.rfc-editor.org/rfc/rfc5614.txt (Última consulta: 17 de julio de 2017)

[9] *Extensions to OSPF to Support Mobile Ad Hoc Networking*: disponible en http://www.rfc-editor.org/rfc/rfc5820.txt (Última consulta: 17 de julio de 2017)

[10] OLSR versión 1: disponible en http://www.rfc-editor.org/rfc/rfc3626.txt (Última consulta: 17 de julio de 2017)

grandes y densas, ya que la optimización realizada con los *multipoint relays* funciona bien en este contexto. Cuanto más grande y densa sea la red, más optimización se consigue en comparación con el algoritmo normal de estado de enlace. El protocolo está diseñado para funcionar de una manera totalmente distribuida, sin depender de ninguna entidad central.

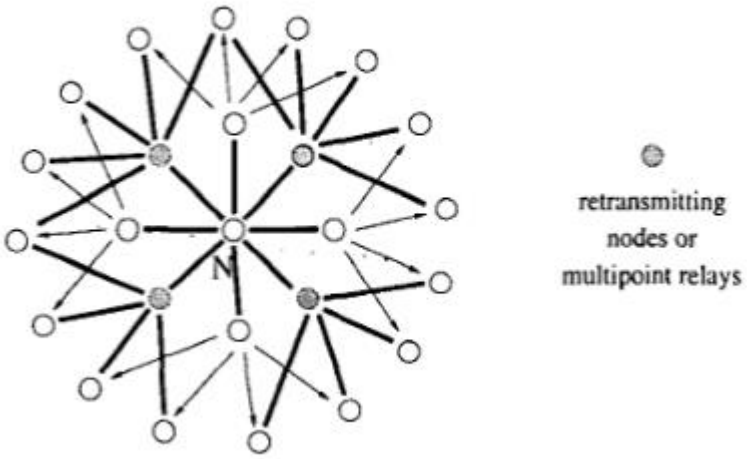


Figura 3: multipoint relays del nodo N[11].

Cada nodo debe detectar los nodos vecinos con los que tiene un enlace directo y bidireccional. Para lograr esto, cada nodo periódicamente difunde sus mensajes HELLO, conteniendo información acerca de sus vecinos y el estado de sus enlaces. Estos mensajes son recibidos por los vecinos, pero no son reenviados más allá. Además de los mensajes HELLO, el protocolo utiliza los mensajes TC (*Topology Control*). Los mensajes TC son reenviados por *Broadcast* por la red completa. Esta técnica es similar a la técnica de estado de enlace utilizada por ARPANET, pero con la ventaja de los MPR, que permite una mayor escalabilidad en el encaminamiento intra-AS.

Un mensaje TC es enviado periódicamente por cada nodo de la red, en el que declara su conjunto de selectores MPR, es decir, el mensaje contiene la lista de nodos vecinos que el nodo emisor del mensaje TC ha elegido como un *multipoint relay*. La información difundida a lo largo de la red por los mensajes TC permite a cada nodo construir la topología completa de la red. A partir de esta información, ya se puede ejecutar el algoritmo de encaminamiento correspondiente (Dijkstra) para obtener las rutas al resto de nodos de la red.

[11] Figura tomada de [13].

Como ya se comentó anteriormente, en los orígenes del protocolo OLSR no se utilizaba la información de calidad del enlace para establecer las rutas, sino que se utilizaba la métrica del número de saltos (como en RIP). Esto quiere decir que una implementación que cumpla el RFC, minimizará el número de saltos entre el nodo origen y el resto de nodos de la red, incluso si esto significa que se prefiera una ruta a través de un enlace muy "malo" frente a otra ruta a través de dos enlaces muy "buenos" [14]. Esta elección probablemente no sea óptima. Por ello, en la implementación 0.4.8 de `olsrd` (implementación del protocolo OLSR del proyecto OLSR.org[12]) se introdujo una implementación experimental de una métrica basada en ETX (*Expected Transmission Count*).

Para solucionar el problema mencionado anteriormente, se planteó una medida de la calidad del enlace basada en medir la pérdida de paquetes. Como cada nodo periódicamente recibe mensajes HELLO de sus nodos vecinos (por defecto el estándar especificaba cada 2 segundos), existe suficiente información como para determinar la tasa de paquetes perdidos. Si por ejemplo 3 de cada 10 paquetes son perdidos existe un 70% de transmisión satisfactoria, valor conocido como calidad del enlace (LQ, *Link Quality*). Es importante conocer la LQ del enlace en la dirección opuesta, es decir, cuántos de los paquetes que enviamos son recibidos por nuestros nodos vecinos A esta calidad la llamamos calidad del enlace del vecino (NLQ, *Neighbor Link Quality*).

Estos dos valores, tanto la LQ como la NLQ, oscilan entre 0 (pérdida total de paquetes) y 1 (transmisión correcta de todos los paquetes). Estos valores pueden tener sentido por separado, pero la idea es combinarlos, debido a que el tráfico de información puede ir en ambos sentidos (petición/respuesta), nos interesa la probabilidad de éxito de un viaje de ida y vuelta. La probabilidad se calcula simplemente como el producto de la LQ y la NLQ, es decir $P_{éxito} = LQ \times NLQ$. A partir de este valor se puede calcular el número medio de transmisiones necesarias para que pueda producirse un intercambio de ida y vuelta entre dos nodos, valor conocido como número esperado de transmisiones, ETX, definido por primera vez en [15].

Para poder utilizar ETX como métrica de coste de los enlaces, es necesario además de saber el valor de LQ, calculado por el mismo nodo, el valor de NLQ, del nodo vecino. Por ello, en esta implementación se introdujeron variaciones a los mensajes HELLO que hicieran posible a cada nodo enviar los valores de LQ a sus

[12] proyecto OLSR.org: disponible en http://www.olsr.org/mediawiki/index.php/Projects (Última consulta 17 de julio de 2017)

nodos vecinos. Nótese que debido a esto una versión que cumpliese el estándar a priori no sería compatible.

Años después, se publicó la segunda versión de OLSR en el RFC 7181[13], que ya permitió más flexibilidad a la hora de definir la métrica de coste. A raíz de esta publicación, el proyecto OLSR.org comenzó a desarrollar la nueva versión de su implementación, olsrd2. Cabe destacar que existen otras implementaciones del protocolo como por ejemplo NLR-OLSR[14] y NET-OLSR[15], pero parece que la que más acogida ha tenido por parte de la comunidad es la primeramente mencionada.

2.5. Predicción de Calidad del Enlace

Tal y como se ha comentado en la sección anterior, OLSR utiliza como valor de coste el valor de ETX obtenido a la hora de calcular las rutas para el encaminamiento. La variabilidad a lo largo del tiempo de la LQ dificulta la optimización del encaminamiento de los paquetes, y, por tanto, dificulta la reducción de la pérdida de estos. Esta variabilidad es debida en parte al carácter inalámbrico de los enlaces y a su carácter asimétrico en ocasiones.

A este respecto, trabajos previos [4] afirman que el seguimiento de la LQ permite a los protocolos de encaminamiento evitar los enlaces asimétricos. Esto es debido a que la métrica de coste (por ejemplo, ETX, presentado en la sección anterior) penaliza a este tipo de enlaces. No obstante, puede que en el tiempo que llega la información de LQ de nodos lejanos, se reconstruye la topología de la red, y se ejecuta el algoritmo de encaminamiento, las rutas calculadas pasen por enlaces cuya calidad haya variado en ese tiempo, siendo por ello la ruta inadecuada. Es decir, que las rutas óptimas hace un tiempo pueden no ser las óptimas ahora.

En este sentido, parece razonable que, si la LQ varía a lo largo del tiempo, se puedan realizar predicciones a corto plazo que mejoren el desempeño de los protocolos de encaminamiento, ya que permitiría tomar las decisiones de encaminamiento con el conocimiento del futuro próximo de la red, que es cuando los paquetes van a ser encaminados, y no con la información actual, que es posible que varíe hasta dicho momento [7].

En [7] se realiza un estudio basado en la predicción de LQ utilizando técnicas de aprendizaje automático aplicadas a series temporales, sobre los enlaces de la

[13] OLSR versión 2: disponible en http://www.rfc-editor.org/info/rfc7181 (Última consulta 17 de julio de 2017)

[14] NRL OLSR *Routing Protocol Implementation*: disponible en https://www.nrl.navy.mil/itd/ncs/products/olsr (Última consulta 17 de julio de 2017)

[15] Windows Mobile OLSR Daemon: disponible en https://sourceforge.net/projects/wmolsr/ (Última consulta 17 de julio de 2017)

WCN FunkFeuer de Austria. Concretamente, se utilizan técnicas de regresión (*regression*) y agrupación (*clustering*). En dichos experimentos, se obtienen buenos resultados utilizando técnicas de regresión como Árboles de Regresión (*Regression Trees*) o Máquinas de Soporte Vector (*Support Vector Machines*).

Por otra parte, también se analiza la degradación a lo largo del tiempo de los modelos construidos con aprendizaje automático, mostrando claramente la necesidad de actualizar los modelos a lo largo del tiempo. Este problema no se resuelve, planteando los autores la problemática de resolverlo siguiendo el enfoque planteado en su investigación. Si se utilizasen los mismos algoritmos de aprendizaje propuestos en [7] y se pretendiesen reconstruir los modelos cada cierto tiempo, el coste computacional sería prohibitivo para dispositivos poco potentes como puedan ser los encaminadores. Por otro lado, en [9] se realiza una investigación acerca de la predicción de la calidad del enlace en redes de sensores. El enfoque seguido es similar al de [7], sin embargo, en este caso se utilizan técnicas de aprendizaje automático en línea, para superar las limitaciones que pueden producir los algoritmos de aprendizaje en lote, como ya se ha comentado anteriormente. Es en este problema en el que se centra este Trabajo Fin de Grado.

2.6. Aprendizaje Automático

El aprendizaje automático (*Machine Learning*), según Arthur L. Samuel, pionero en el campo de la inteligencia artificial y considerado como el creador del término *Machine Learning* , se puede definir como “el campo de estudio que da a los ordenadores la capacidad de aprender sin ser explícitamente programados para ello” [16]. Más formalmente, Tom Mitchell [17] dijo que “un programa de ordenador aprende de la experiencia E respecto a una tarea T con una medida del rendimiento P, si su rendimiento en la tarea T medido por P mejora con E”. El aprendizaje automático tiene muchas aplicaciones actualmente [18], como por ejemplo en robótica, biología, minería de datos, reconocimiento de escritura, visión por ordenador, buscadores para clasificar las páginas web, etcétera. En resumen, el aprendizaje automático está formado por un conjunto de algoritmos que permiten extraer modelos a partir de datos u observaciones de un sistema de cualquier tipo, con el fin de realizar tareas de clasificación o predicción de ese sistema.

El aprendizaje automático se ha vuelto muy importante debido al crecimiento de la cantidad de datos disponible [19]. En el año 2000, la cantidad total de datos disponible en la Web variaba entre 25 y 50 terabytes. En 2005, el tamaño aproximado era de 600 terabytes. Hoy en día, la cantidad total es casi incalculable.

En primer lugar, se puede hacer una primera distinción entre los diferentes tipos de aprendizaje automático entre los que se ejecutan sobre un conjunto de

datos fijo, que se conoce como aprendizaje en lote (*Batch*) y sobre los que se ejecutan sobre un flujo de datos (*stream* u *online*). El aprendizaje en lote se emplea para aprender modelos sobre un conjunto de datos fijo, que se ha recolectado a lo largo de un periodo de tiempo. Este enfoque puede ser suficiente, pero existen aplicaciones en las que se vuelve insuficiente. Por una parte, los datos pueden llegar a una velocidad tan elevada, que el almacenarlos conlleve un gasto inasumible. Además, actualizar periódicamente los modelos puede ser demasiado costoso, por lo que una alternativa el tiempo real (*stream*) es la solución ideal, en la que los datos no se almacenen, y el modelo se actualice progresivamente a medida que se obtienen nuevos datos.

Por otra parte, se puede hacer una distinción entre aprendizaje supervisado y no supervisado. El aprendizaje supervisado es el tipo más común de aprendizaje automático [18], en el cual se le entregan datos correctos al algoritmo (es decir, se conoce la correspondencia entre la entrada y la salida) y tiene que ser capaz de realizar predicciones cuando se le alimenta con nuevos datos. Por otro lado, el aprendizaje no supervisado se le entregan datos al algoritmo y el algoritmo tiene que ser capaz encontrar estructura en ellos, formando distintos grupos. Esto es lo que se conoce como problema de agrupamiento (*clustering*). El aprendizaje no supervisado es utilizado en multitud de situaciones, como por ejemplo clasificación automática de noticias, genoma humano, análisis de redes sociales, astronomía, etcétera

Finalmente, cabe destacar que todos estos algoritmos se pueden ejecutar de manera distribuida. Tradicionalmente, el cuello de botella que impedía el desarrollo de sistemas más inteligentes era la cantidad limitada de datos disponibles [19]. No obstante, ahora el factor limitante es la imposibilidad de los algoritmos de aprendizaje de usar todos los datos para aprender en un tiempo razonable. En este contexto, el aprendizaje distribuido parece ser una línea prometedora de investigación, dado que distribuye el proceso de aprendizaje a lo largo de varias estaciones de trabajo, como una manera natural de escalar los algoritmos.

2.7. *Toolkits* de Aprendizaje Automático

Existe un gran número de *toolkits* o herramientas que permiten probar un conjunto elevado de algoritmos de aprendizaje automático. En este documento vamos a destacar tanto los más importantes en la actualidad, como aquellos que han sido utilizados para el desarrollo de este Trabajo Fin de Grado. Vamos a exponer las distintas alternativas en función de la clasificación realizada en el apartado anterior, de acuerdo a lo que se puede ver en la Figura 4.

- En lote: existe una gran variedad de toolkits de aprendizaje automático en lote no distribuido, ya que se trata del caso "más sencillo" de todos los posibles en el campo de la minería de datos. De entre todas las alternativas posibles, vamos a destacar tres:
    - WEKA[16]: es una de las herramientas que se ha utilizado en el desarrollo de este Trabajo Fin de Grado, ya que es la herramienta que los autores de [7] utilizaron en sus experimentos. Dispone de interfaz gráfica, API Java, e interfaz de línea de comandos, por lo que es una herramienta muy flexible en tanto en cuanto puede ser utilizada con varias finalidades y por distintos tipos de usuarios. Incluye una gran variedad de algoritmos para regresión, clasificación y *clustering*, así como muchos módulos instalables para aumentar las funciones base.
    - R[17]: es un lenguaje de programación enfocado a la computación estadística (modelado lineal y no lineal, tests estadísticos, análisis de series temporales, clasificación, *clustering*, etc.) y a los gráficos. Pese a ser un lenguaje creado en 1993, su popularidad se ha incrementado enormemente en los últimos años con el auge de la minería de datos.
    - scikit-learn[18] (Python): es una biblioteca de Python enfocada al aprendizaje automático, construida sobre las bibliotecas NumPy, SciPy y matplotlib. Últimamente, ha gozado de una gran popularidad, siendo incluso utilizada por Spotify[19] para realizar las recomendaciones de música.
- En línea:
    - MOA[20]: esta es la segunda herramienta que se ha utilizado en el desarrollo de este trabajo Fin de Grado. Es similar a WEKA, pero enfocado en el aprendizaje automático sobre flujos de datos. Implementa una gran variedad de algoritmos de aprendizaje automático en línea de "última generación", para regresión, clasificación y *clustering*. Al

[16] WEKA: disponible en http://www.cs.waikato.ac.nz/ml/weka/ (Última consulta: 17 de julio de 2017)

[17] R: disponible en https://www.r-project.org/ (Última consulta: 17 de julio de 2017)

[18] scikit-learn: disponible en http://scikit-learn.org/stable/ (Última consulta: 17 de julio de 2017)

[19] Spotify: disponible en https://www.spotify.com/es/ (Última consulta: 17 de julio de 2017)

[20] MOA: disponible en http://moa.cms.waikato.ac.nz/ (Última consulta: 17 de julio de 2017)

igual que WEKA, dispone de interfaz gráfica, API Java e interfaz de línea de comandos.

A modo de resumen, en la Figura 4 se puede ver un gráfico que ilustra la clasificación expuesta en esta sección y en la anterior. Nótese que en la referencia empleada se refieren a estos algoritmos como minería de datos y no como aprendizaje automático.

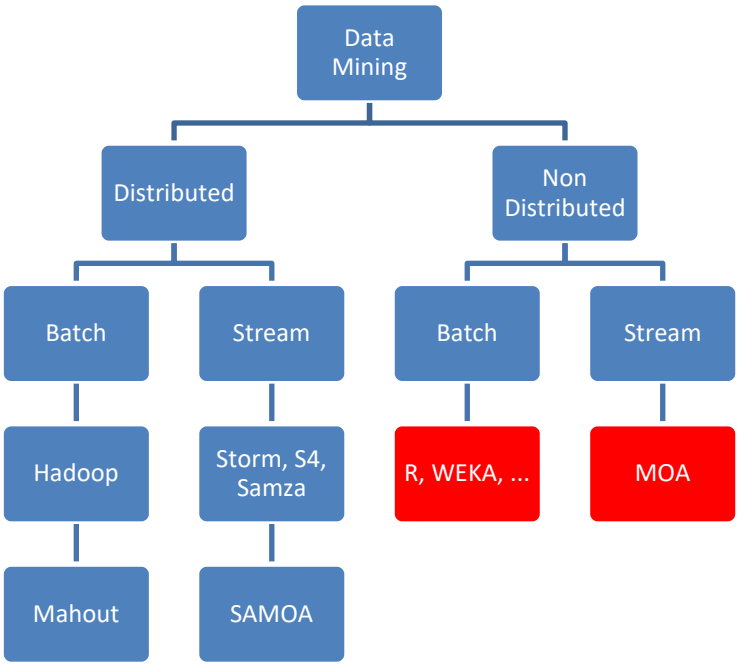


Figura 4: clasificación del aprendizaje automático[21].

2.8. Conjuntos de datos disponibles

FunkFeuer es una WCN experimental desplegada en varias ciudades de Austria. En el estudio realizado por [7], existían unos datos disponibles a través del proyecto CONFINE[22], que lamentablemente ya no estaban disponibles en el momento del inicio de este Trabajo Fin de Grado. No obstante, sí que había datos disponibles de la misma red (Austria), de un periodo más reciente, por lo que se han podido realizar los experimentos sin ningún problema. Al término de este Trabajo Fin de Grado, dichos datos ya no están disponibles.

[21] Figura tomada de [29].

[22] Proyecto CONFINE: disponible en https://confine-project.eu/ (Última consulta: 17 de julio de 2017)

Por otra parte, existe un sitio web[23], correspondiente a la red FunkFeuer de la ciudad de Graz, en el que se actualizando los datos de calidad de enlace correspondientes a sus enlaces, pero lamentablemente, la web tiene prohibido el acceso a robots de recogida de datos, por lo que sin permiso no se podría acceder al uso de estos datos.

2.9. Conclusiones

A lo largo de este capítulo, se han ido presentando las diferentes problemáticas existentes en cuanto al encaminamiento en las redes de comunidad. Se ha incidido en el problema que supone la predicción de la calidad del enlace en las mismas, así como en las técnicas y herramientas existentes que podrían utilizarse de cara a solucionarlo.

En la parte restante del documento se presentará el trabajo realizado a partir del trabajo previo de [7], empezando por la réplica de parte de los experimentos realizados en su investigación, y se continuará con nuevos experimentos que pongan en manifiesto sus debilidades y permitan obtener mejores soluciones.

[23] FunkFeuer Graz: disponible en http://stats.ffgraz.net/index.php?module=olsr&view=topo (Última consulta: 17 de julio de 2017)

Capítulo 3. Reproducción de experimentos previos

3.1. Introducción

En el capítulo anterior se han estudiado las principales problemáticas existentes en cuanto al encaminamiento en las redes de comunidad, destacando la predicción de la calidad del enlace entre todas ellas. Para hacer frente a este problema, los autores de [7] realizan un enfoque basado en predicción de series temporales utilizando métodos de aprendizaje automático en lote. Sin embargo, dicho enfoque tiene una serie de problemas, como son la degradación de los modelos construidos a medida que avanza el tiempo y el elevado coste computacional asociado a algunos de los algoritmos. Como se ha mencionado anteriormente, en primer lugar, se van a reproducir los experimentos previos realizados por [7], con el objetivo de entender mejor la problemática de la calidad del enlace, así como las fortalezas y debilidad del enfoque seguido en su investigación.

En primer lugar, en este capítulo se expondrán los datos utilizados, mostrando las semejanzas y diferencias con los utilizados por los autores del artículo. Posteriormente, se detallará el equipamiento utilizado, con el fin de que las medidas de coste computacional sean relativas a la potencia del equipo. A continuación, se iniciará la réplica de los principales experimentos del artículo, en concreto, se realizarán 4 experimentos: comparativa de 4 algoritmos de aprendizaje automático en lote, estudio de la influencia del tamaño de la ventana temporal, predicciones con un horizonte más alejado y degradación del modelo con el paso del tiempo. Finalmente, se extraerán unas breves conclusiones.

3.2. Datos utilizados

Los datos utilizados en [7] correspondían a medidas de LQ tomadas cada 5 minutos durante 7 días completos, desde el día 28 de abril hasta el 4 de mayo, del año 2014. En el momento del desarrollo de este Trabajo Fin de Grado, estos datos ya no estaban disponibles, por lo que se han utilizado datos más recientes que podrían haber condicionado los resultados obtenidos. Los datos que se obtuvieron corresponden a medidas de LQ tomadas cada 5 minutos durante 14 días completos, desde el día 15 (viernes) hasta el 28 (jueves) de enero, del año 2016. No obstante, dado que los experimentos originales utilizaban únicamente una semana de datos, se han usado los datos del 15 (viernes) al 21 (jueves) de enero. En primer lugar, cabe destacar que a lo largo de los 7 días el número de enlaces encendidos ha ido variando, como se puede ver en la Figura 5. Las gráficas se han realizado con

Anaconda[24], una distribución de Python con numerosas bibliotecas, entre ellas matplotlib, la biblioteca para dibujar gráficos de forma similar a MATLAB[25].

Por otra parte, el número total de enlaces únicos en la red ha sido de 1928, frente a los 2095 de [7]. En su artículo, después de obtener todos los enlaces, se realiza una selección de ellos previa a los experimentos. En primer lugar, se eliminan los enlaces que no varían a lo largo de los 7 días de experimentos, ya que la predicción en estos casos es trivial y se obtendría una mayor precisión en las predicciones de manera engañosa. En nuestro caso, después de eliminar los enlaces cuya LQ no ha variado a lo largo de los 7 días, nos quedan 1090 enlaces, frente a los 1068 que obtiene los autores de [7]. Finalmente, se eliminan los enlaces de los que no hay suficientes datos para entrenar. Como en el trabajo previo no se especifica qué criterio han seguido a la hora de determinar que "no hay suficientes datos para entrenar", se ha considerado que se eliminan los enlaces que están apagados durante todo el conjunto de entrenamiento (los seis primeros días de los experimentos). Además, se han eliminado los enlaces con valor de LQ constante durante el entrenamiento, ya que WEKA nos devolvía errores en estos casos. Después de este procedimiento, nos quedan 979 enlaces frente a los 1032 con los que se quedan en [7]. En la Tabla 1 se puede ver un resumen de lo previamente explicado.

Selección de enlaces	**Artículo [7]**	**Nuestro experimento**
Número de enlaces disponibles inicialmente	2095	1928
Número de enlaces después de eliminar los invariables	1068	1090
Número de enlaces final	1032	979

Tabla 1: resumen del proceso de selección de enlaces.

[24] Anaconda: disponible en https://www.continuum.io/downloads (Última consulta: 17 de julio de 2017)

[25] MATLAB: disponible en https://es.mathworks.com/products/matlab.html (Última consulta: 17 de julio de 2017)

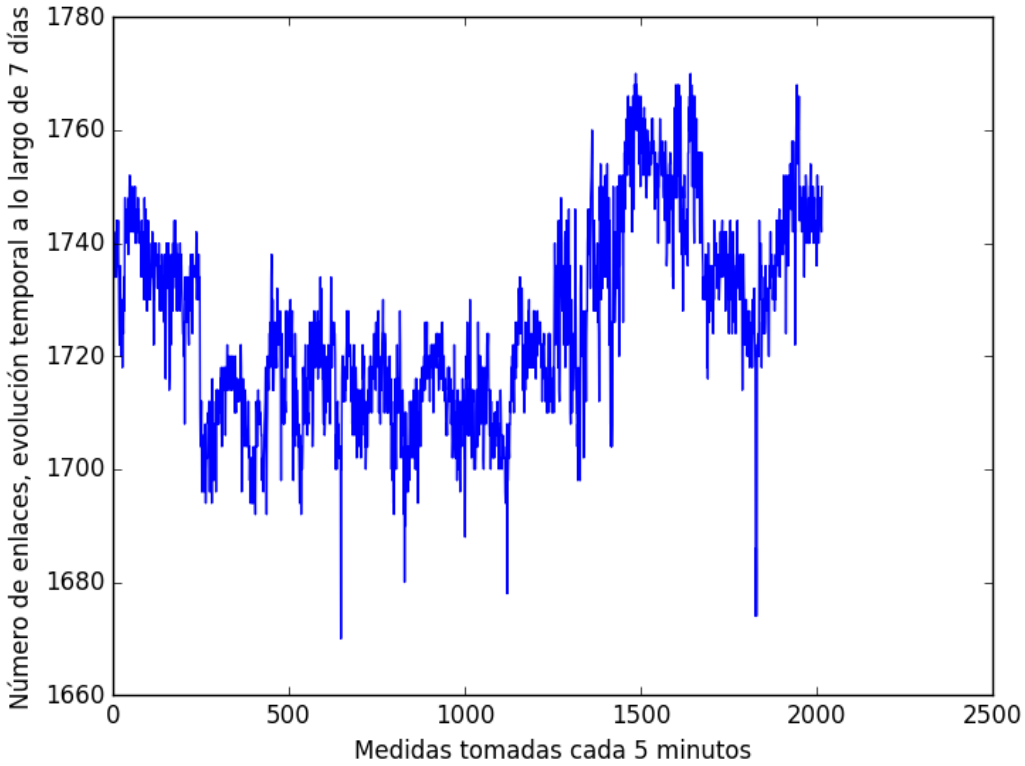


Figura 5: evolución temporal del número de enlaces en la red FunkFeuer a lo largo de 7 días.

Por otra parte, en los instantes temporales que el enlace está apagado, se ha optado por asignarle un valor de LQ igual a 0. En [7] no se indica nada a este respecto, aunque sí se intuye que pueda haberse realizado de esta manera, ya que inciden claramente en que el conjunto de entrenamiento está formado por 1728 instancias (lo correspondiente a 6 días). Es decir, se ha supuesto un estado estacionario de la red en la que se conocen todos los posibles enlaces entre los nodos, cosa que en una red real no podría considerarse a priori.

Por tanto, se ha tomado esta decisión pensando en la simplificación de los experimentos, ya que, de otro modo, cada enlace tendría datos en unos instantes de tiempo diferentes al resto de enlaces, tanto en número como en ubicación temporal. Además, esta parecía la forma más justa de computar los errores de los predictores, pese a que en algunos casos se esté prediciendo sin que el enlace en sí esté disponible. No obstante, este enfoque podría también ser el más adecuado en tanto en cuanto el predictor podría detectar periodicidades en los ciclos de encendido apagado de los enlaces, pero puede perjudicar a la métrica de error del predictor, ya que se está considerando de la misma manera a los errores que se cometen cuando el enlace está activo que cuando está apagado. Respecto a las consideraciones

previas, si un enlace está parte de los 7 días apagado y cuando está encendido tiene siempre el mismo valor de LQ, se ha considerado que no es un enlace de calidad constante.

A modo de cierre a este apartado, se ha dibujado con la librería nx de Anaconda un grafo de la red completa (se agregan los enlaces existentes a lo largo de los 7 días). Además, se han dibujado por colores los enlaces que se han ido eliminando en cada etapa como se ha indicado anteriormente. Nótese que la ubicación de los nodos es aleatoria y no responde a ninguna localización geográfica de los mismos, es simplemente una forma de representación. Además, en cada ejecución la ubicación de los nodos varía, pero para nuestro propósito no supone ningún problema. En la Figura 6 se ve el estado completo de la red antes de haber eliminado ningún enlace. Por otra parte, en la Tabla 2 se destacan los enlaces que se eliminan y los que quedan finalmente.

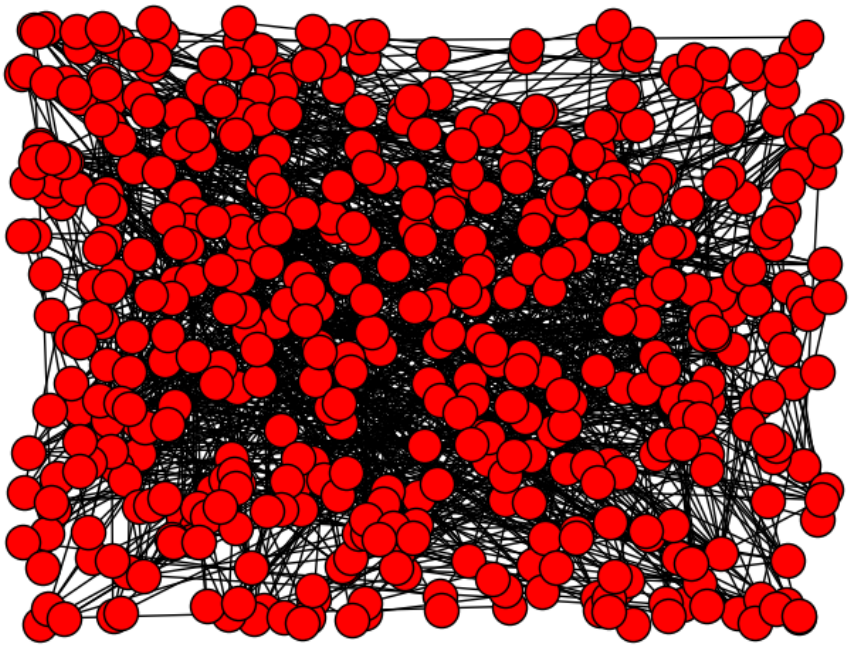


Figura 6: grafo completo de la red FunkFeuer.

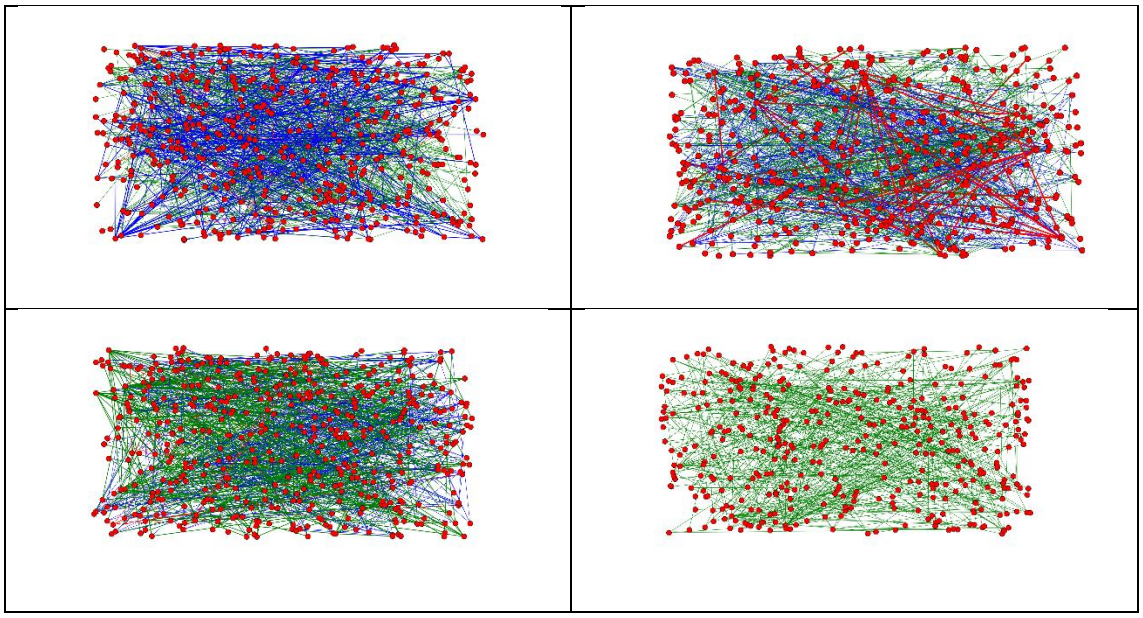


Tabla 2: de izquierda a derecha y de arriba a abajo: a) Red completa, destacando en azul los enlaces con calidad constante; b) red completa, destacando en rojo los enlaces con información insuficiente; c) red completa, destacando en verde los enlaces que cumplen todas las condiciones; d) red parcial final fruto de la selección de enlaces realizada.

3.3. Descripción del equipamiento utilizado

Dado que en las posteriores secciones se va a evaluar el tiempo de computación empleado por los algoritmos además del error en la predicción, es necesario especificar con qué especificaciones principales contaba el equipo utilizado, dado que dichas medidas temporales van a depender en última instancia de la potencia del equipo utilizado. Las características se han obtenido el *software* CPU-Z[26]. Además de las características mostradas en la Figura 7 y en la Figura 8, la máquina cuenta con un disco duro Western Digital de 500GB a 5400 rpm y un disco de estado sólido Samsung 850 EVO mSATA de 250 GB, donde se alojan tanto los ficheros utilizados como los programas utilizados en este Trabajo Fin de Grado.

Por otra parte, es necesario mostrar las versiones del *software* utilizado para la realización tanto de estos experimentos. Se ha utilizado la versión de Java 8, actualización 121. Se ha utilizado la versión 3.8.1 de WEKA, así como la versión 1.0.25 del módulo de series temporales de WEKA (timeseriesForecasting). Para el uso de este módulo, se ha seguido la documentación disponible en la Wiki de Pentaho[27], desarrollador del módulo.

[26] CPU-Z: disponible en http://www.cpuid.com/softwares/cpu-z.html (Última consulta: 17 de julio de 2017)

[27] Time Series Forecasting: disponible en http://wiki.pentaho.com/display/DATAMINING/Time+Series+Analysis+and+Forecasting+with+Weka#TimeSeriesAnalysisandForecastingwithWeka-4UsingtheAPI (Última consulta: 17 de julio de 2017)

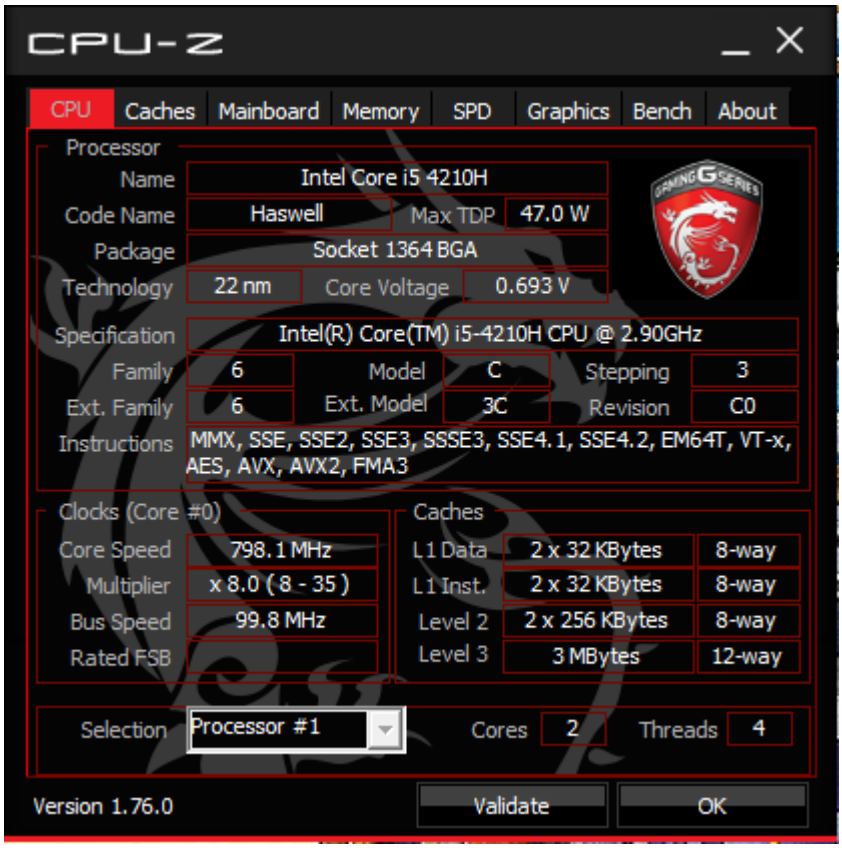


Figura 7: características de la CPU de la máquina utilizada.

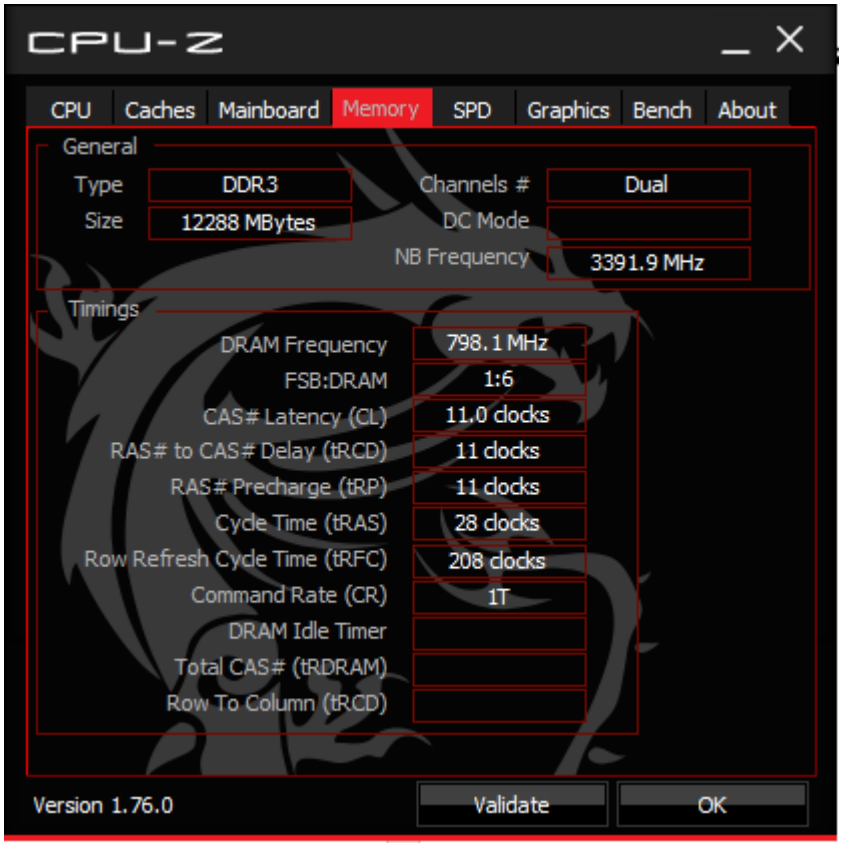


Figura 8: características de la memoria principal de la máquina utilizada

3.4. Algoritmos de aprendizaje basados en series temporales

En esta sección se van a replicar los experimentos realizados en la sección 4.1 de [7]. En ella, se aplican 4 algoritmos de aprendizaje automático con el fin de predecir la LQ del instante siguiente. Estos cuatros algoritmos son: *Support Vector Machines* (SVM), k-*Nearest Neigbours* (kNN), *Regression Trees* (RT) y *Gaussian Processes for Regression*). El algoritmo SVM se ha convertido recientemente en uno de los más populares y ampliamente utilizados. Realiza una división lineal o no lineal del espacio de entrada y construye un modelo de predicción que asigna los valores en una u otra categoría. Aunque SVM es un algoritmo de clasificación, tiene su versión de regresión llamada SVR (*Support Vector Regression*) [20]. El algoritmo kNN es uno de los algoritmos de aprendizaje automático más simples, seleccionando los k vecinos más cercanos a la entrada introducida. Dependiendo si se usa para clasificación o para regresión, entrega a la salida la clase a la que pertenecen la mayoría de los vecinos, o la media de los valores de los vecinos, respectivamente. RT es un tipo de árbol de decisión donde el valor de salida puede ser continuo. Recursivamente particiona el espacio de datos y ejecuta un modelo de predicción simple en cada partición. Sus ventajas son que produce resultados rápidos y es resistente a valores irrelevantes. El algoritmo GPR es una aproximación flexible que puede fácilmente lidiar con conjuntos de datos complejos. La salida es una distribución normal definida por su media y su varianza. En el módulo de series temporales, los algoritmos que se han utilizado para aplicar estos algoritmos son SMOreg [21], IBk [22], REPTree [23] y GaussianProcesses [24], respectivamente. Podría haberse utilizado M5P [25] en vez de REPTree, ya que ambos son árboles de regresión, pero se ha optado por el primero, ya que en [7] no especifican los algoritmos concretos utilizados. En cuanto al resto, no hay duda posible.

Los experimentos han sido realizados de la siguiente manera:

- Por cada enlace, tenemos 2016 medidas de calidad del enlace (1 cada 5 minutos, durante 7 días).
- De estas 2016 medidas de calidad del enlace, se han tomado las 1728 primeras (6 días) para realizar el entrenamiento.
- Las 288 últimas muestras se han utilizado de conjunto de test.
- Se ha elegido una ventana temporal (*lag window* según la notación de WEKA) de 12 unidades. La ventana temporal es el número de medidas (instancias) pasadas que se utilizan para crear los vectores de entrenamiento, es decir, cada vector de entrenamiento está formado por 12 valores pasados y el siguiente. Por tanto, a la hora de alimentar el

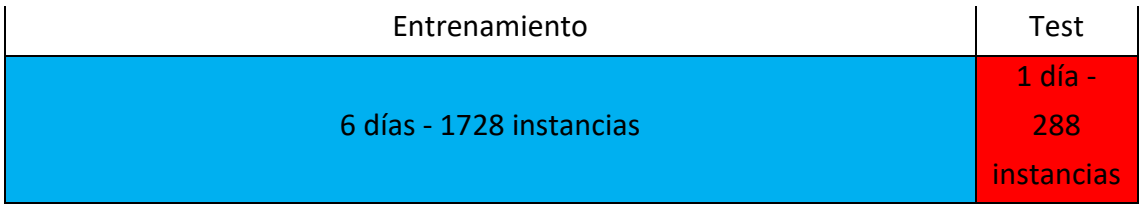
predictor, se le dará una secuencia de 12 valores y con ellos tendrá que predecir el siguiente valor.

- Se ha realizado una predicción a 1 instante vista. Es decir, con cada 12 medidas, se ha predicho una nueva medida. Este valor predicho se ha comparado con el valor real del conjunto de test, y a continuación se ha desplazado la ventana temporal 1 unidad hacia el futuro, con el fin de predecir una nueva medida. Es decir, se van a utilizar las medidas del conjunto de test también como entrada del predictor, pero estas medidas habrán sido testeadas en el instante inmediatamente anterior. La$y t= f(y(t-1).y(t-2).y(t-12))$
- Figura 9 ilustra perfectamente lo aquí explicado.
- Se ha realizado un predictor por cada enlace.

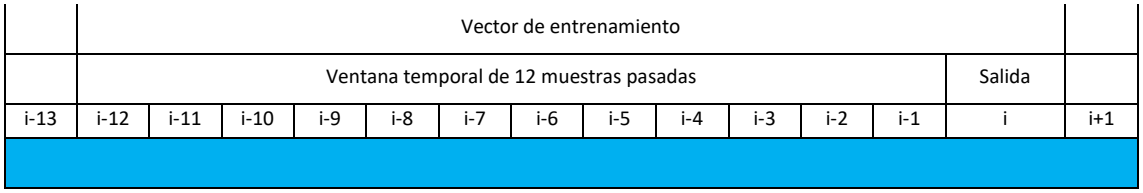
Hay que destacar que estos experimentos realizados tendrían todo el sentido en un contexto como el de una WCN, ya que utilizando el protocolo OLSR, que es un protocolo de estado de enlace, cada nodo obtiene la información de calidad del enlace de todos los enlaces de la red. Con esta información, sería posible que cada nodo realizase las predicciones que se están realizando, con el objetivo de que se haga el encaminamiento con el estado futuro predicho de la red y no con los valores actuales.

Por defecto, el módulo de series temporales de WEKA, además de construir un modelo con las 12 instancias de la ventana temporal, también utiliza un valor de timestamp, que lo obtiene remapeando los instantes de tiempo que se incluyen como entrada (los correspondientes a las fechas en las que se obtuvieron los datos de LQ. Además, se incluyen las potencias al cuadrado y al cubo de este timestamp, así como el producto del mismo con cada una de las instancias de la ventana temporal. Esto se puede ver en la Figura 10, que muestra la descripción de los parámetros que van a usarse en el conjunto de entrenamiento, concretamente para el entrenamiento del predictor del enlace número 0. El uso de estas entradas tiene el objetivo de poder crear un modelo más potente y sofisticado, que pueda detectar tendencias generales a lo largo del tiempo (por ejemplo, que la LQ siempre aumentase a lo largo del tiempo), pero puede llevar a un sobreajuste de los datos, que resultará en modelos incapaces de generalizar a partir del conjunto de entrenamiento. Además, al incluir estos parámetros, ya no se está realizando un análisis de series temporales puro, ya que un análisis de series temporales solo utiliza los valores pasados de las variables a predecir. No obstante, y dado que en [7] no indican lo contrario, en primer lugar, se ejecutarán los experimentos con los parámetros por defecto, y después se explorará la eliminación de los mismos. Con lo explicado anteriormente se han realizado las predicciones. Se han utilizado los

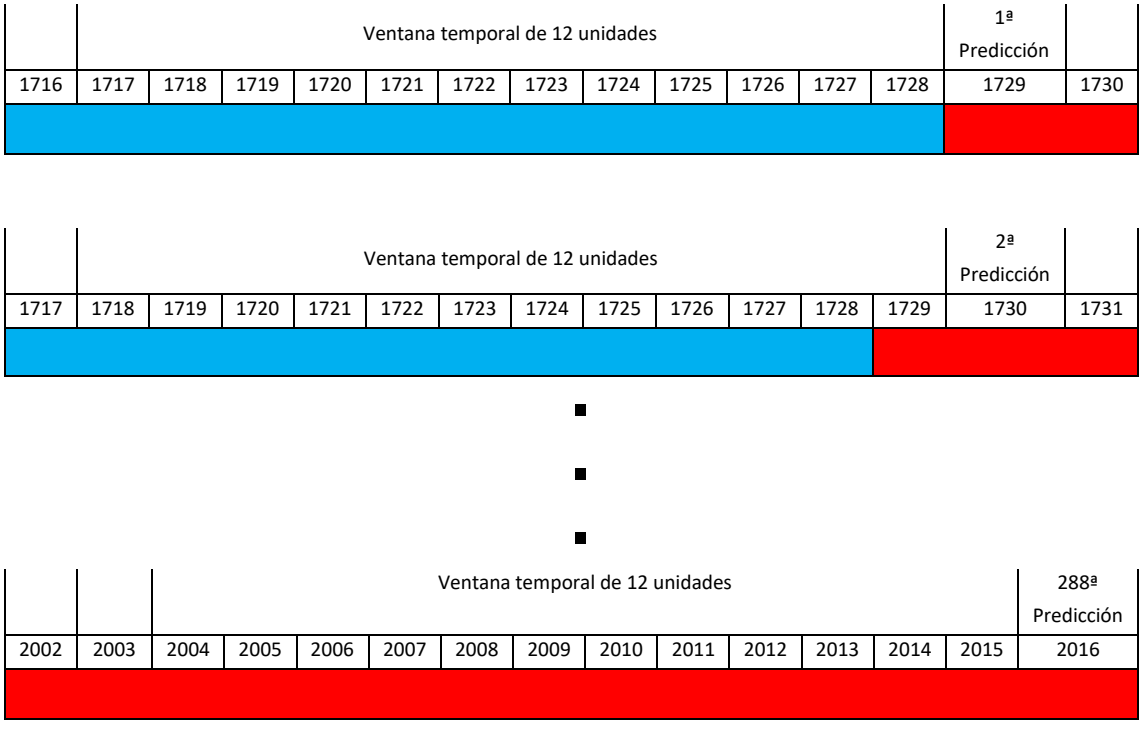
parámetros por defecto de todos los algoritmos de aprendizaje automático, exceptuando el número de vecinos del algoritmo kNN, que se ha fijado a un valor de 10, lo cual se explicará posteriormente.



Entrenamiento:



Predicción y test:



$$y(t) = f(y(t-1), y(t-2) \dots y(t-12))$$

Figura 9: explicación del funcionamiento del predictor.

Los resultados obtenidos se van a mostrar de dos formas. Por una parte, obtenidos los valores de MAE (*Mean Absolute Error*, error medio absoluto) de cada uno de los enlaces, vamos a realizar el MAE promedio por cada algoritmo utilizado, lo cual se representará con un gráfico de barras. Cabe destacar que se ha elegido la métrica de MAE por ser la métrica usada por los autores de [7], que obtenían resultados semejantes usando RMSE (*Root Mean Squared Error, raíz del error cuadrático medio*), aunque esta métrica da más peso a los errores grandes que a los pequeños, a diferencia del MAE que los trata por igual. Por otra parte, vamos a dibujar un diagrama de caja o *boxplot*, que es una representación que permite ver a simple vista la distribución de los datos, de forma semejante a un histograma. Cada *boxplot* está formado por una caja, dentro de la cual se encuentran los datos entre el primer y el tercer cuartil. Dentro de la caja, se marca con una línea el segundo cuartil (mediana). De cada lado de la caja salen unos bigotes o *whiskers*, que se extienden hasta 1.5 veces el rango intercuartil (el rango entre el primer y el tercer cuartil). Si algún dato no cae dentro, se representan de manera independiente, a los que se conoce como puntos singulares o *outliers*. La Figura 11: explicación de un diagrama de caja o boxplot.Figura 11 clarifica esta explicación, en este caso representando un *boxplot* de una distribución Normal.

```
Transforming input data...
Transformed training data:

            Link0
            timestamp-remapped
            Lag_Link0-1
            Lag_Link0-2
            Lag_Link0-3
            Lag_Link0-4
            Lag_Link0-5
            Lag_Link0-6
            Lag_Link0-7
            Lag_Link0-8
            Lag_Link0-9
            Lag_Link0-10
            Lag_Link0-11
            Lag_Link0-12
            timestamp-remapped^2
            timestamp-remapped^3
            timestamp-remapped*Lag_Link0-1
            timestamp-remapped*Lag_Link0-2
            timestamp-remapped*Lag_Link0-3
            timestamp-remapped*Lag_Link0-4
            timestamp-remapped*Lag_Link0-5
            timestamp-remapped*Lag_Link0-6
            timestamp-remapped*Lag_Link0-7
            timestamp-remapped*Lag_Link0-8
            timestamp-remapped*Lag_Link0-9
            timestamp-remapped*Lag_Link0-10
            timestamp-remapped*Lag_Link0-11
            timestamp-remapped*Lag_Link0-12
```

Figura 10: características del conjunto de entrenamiento por defecto.

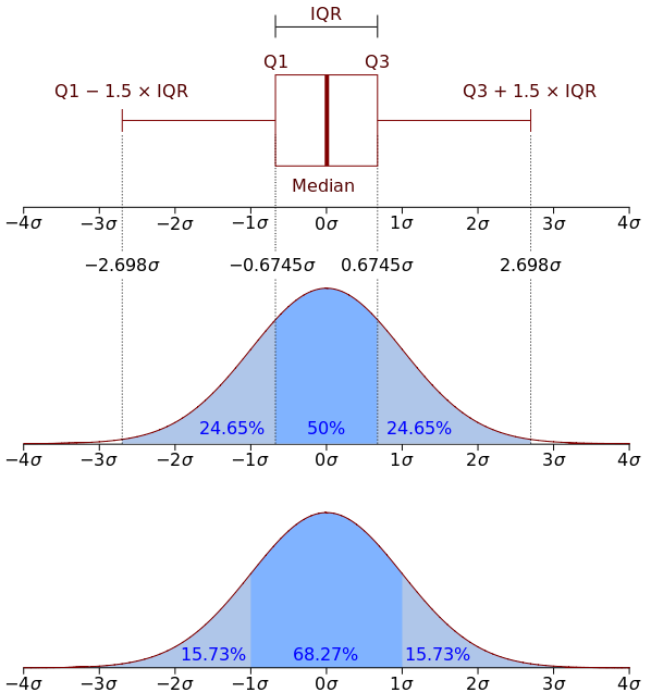


Figura 11: explicación de un diagrama de caja o boxplot.

En la Figura 12 se muestra el gráfico de barras, mientras que en la Figura 13 se muestra el *boxplot*. Por otra parte, en la Tabla 3 se muestra una comparativa entre los resultados de MAE promedio obtenidos y los que se muestran en [7]. Respecto a los resultados obtenidos en estos experimentos, y en comparación con los reportados por [7], cabe destacar los siguientes puntos:

- En primer lugar, en comparación con los resultados reportados por [7], a groso modo se observan unos resultados parecidos, por lo que los experimentos han sido satisfactorios. Los niveles de MAE promedio reportado son similares, y los *boxplots* tienen la misma distribución. No obstante, hay algunas diferencias que merece la pena destacar.
- Obtenemos un resultado mucho mejor para el algoritmo SVM que el reportado por los otros autores. Dado que SVM es un algoritmo muy costoso computacionalmente, como más tarde se podrá comprobar (solo por detrás de GPR), puede que hayan utilizado una versión menos costosa computacionalmente (o que hayan utilizado algún parámetro que produzca este efecto).
- En cuanto al algoritmo RT, los resultados son prácticamente idénticos, lo que nos hace pensar que los experimentos han seguido el mismo enfoque, y, por tanto, están bien realizados.

- Con el algoritmo kNN, nuestro resultado es bastante mejor que el reportado por los otros autores. Esta diferencia podría ser debida a una mala selección del valor de k, que como ya se mencionó anteriormente, más tarde comentaremos.
- Finalmente, con el algoritmo GPR obtenemos los peores resultados, de la misma manera que los autores del artículo (y con valores semejantes), lo que nos reafirma en la idea de que los experimentos están bien realizados.
- No obstante, cualquier diferencia podría ser debida a la diferencia en los datos de partida y no a lo indicado anteriormente.

MAE promedio	**Artículo [7]**	**Obtenido**
SVM	3.2%	2%
RT	2.7%	2.6%
kNN	4%	2.7%
GPR	4.5%	4.1%

Tabla 3: comparativa de los resultados de MAE promedio de predicción de LQ obtenidos en [7] y los obtenidos en nuestros experimentos.

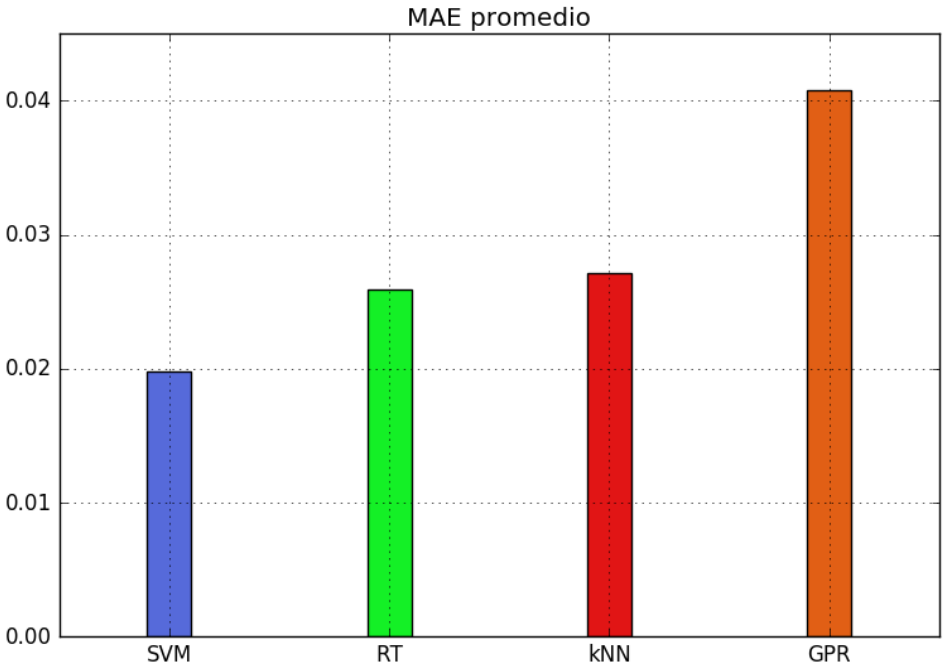


Figura 12: MAE promedio de los 4 algoritmos estudiados.

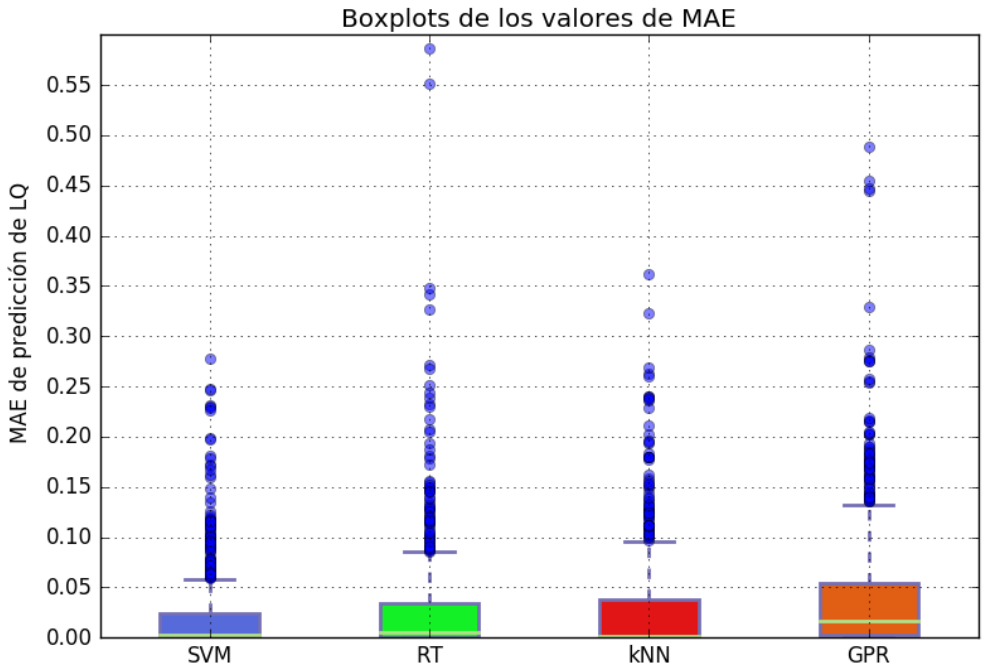


Figura 13: : Boxplots comparando el MAE de la predicción de LQ de los 4 algoritmos estudiados.

Una vez realizados los experimentos, en el artículo utilizan el T-test [26] para ver si estadísticamente las medias de error (MAE promedio) obtenidas de los diferentes algoritmos son realmente diferentes. El T-test se utiliza sobre muestras cuya población subyacente está distribuida de manera normal (Gaussiana), y además el tamaño de las muestras es pequeño. Cabe plantearse la pregunta de, si tener una muestra de tamaño 979 por cada algoritmo es pequeña, pero, aun así, en primer lugar, vamos a comprobar si los datos son Gaussianos. En primer lugar, en la Figura 14 vemos los histogramas de los valores de MAE promedio para cada algoritmo. A la vista de los histogramas, podemos ver que los datos no son Gaussianos, pero vamos a realizar el test de Kolmogorov-Smirnov (KS-test) [27] para ver si se adaptan los datos a una distribución normal estándar (media cero y desviación típica 1). Para ello, podemos o bien estandarizar nuestros datos (restamos la media y dividimos por la desviación típica) o bien pasarle estos dos argumentos a la función que realiza el test.

En este caso, la hipótesis nula es que las dos distribuciones son iguales, si el p-valor es menor que nuestro nivel de significatividad estadística, que tomamos por ejemplo a 0.05 (Valor que se toma en el artículo al realizar el T-test) Con cualquiera de los algoritmos obtenemos un p-valor de 0, por lo que podemos rechazar la hipótesis nula y, por tanto, los datos no son Gaussianos, como era de esperar.

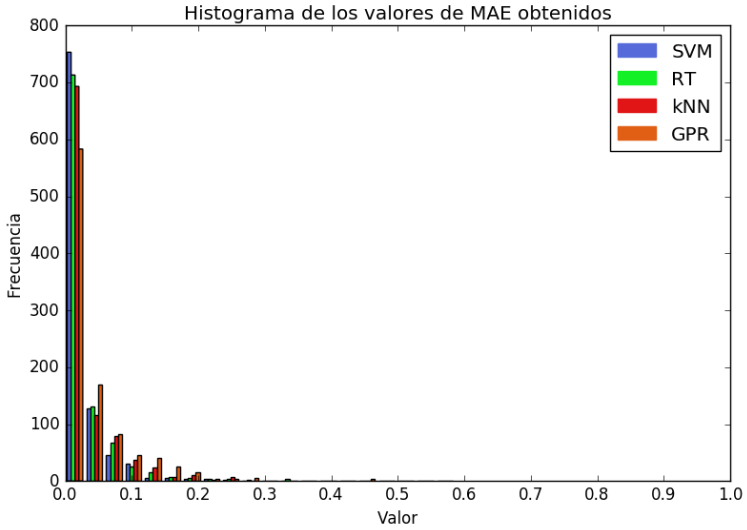


Figura 14: histograma de los valores de MAE obtenidos para los 4 algoritmos.

Dado que los datos no son Gaussianos, no podemos emplear el T-test, pero en cambio podemos emplear el Z-test de acuerdo a lo expuesto en [28]. Para poder aplicar el Z-test, necesitamos que las muestras (el resultado de cada algoritmo, formado por 979 valores de MAE, es una muestra) estén distribuidas de forma normal (Gaussiana). De acuerdo al teorema del límite central, la suma de variables aleatorias independientes tienden a tener una distribución Gaussiana, aunque las variables en sí mismas (en este caso el resultado de un experimento) no se distribuyan de manera Gaussiana. Además, debe ser posible obtener la media y la varianza de cada una de las muestras de manera precisa. Al tener muestras formadas por casi 1000 elementos, podemos afirmar que es posible.

Para una significatividad estadística del 5%, obtenemos que se puede rechazar la hipótesis nula, y, por tanto, las medias de MAE de SVM y RT son distintas (Obtenemos un p-valor de 0.0018). Podríamos haber tomado un nivel de significatividad estadística menor, pero hemos tomado el mismo que se utilizaba en el artículo como referencia. Comprobando el resto de combinaciones, únicamente no se puede rechazar la hipótesis nula de que la media de RT sea diferente a la de kNN (obtenemos un p-valor de 0.5943). Estos resultados son los que a simple vista podíamos inferir del gráfico de barras y de los *boxplots*, pero es importante constatarlo estadísticamente.

Finalmente, y antes de cerrar esta sección, vamos a explicar por qué se ha elegido el valor 10 para el parámetro k del algoritmo kNN. Para ello, hay que explicar qué significa este parámetro y que rol juega en el algoritmo kNN. El algoritmo kNN es un algoritmo sencillo que simplemente realiza un promedio entre las salidas de sus "k" vecinos más cercanos del conjunto de entrenamiento. La métrica de cercanía

es variable, pero por defecto se utiliza la distancia Euclídea. Inicialmente se probó con el valor por defecto de k, igual a 1, pero los errores obtenidos por este algoritmo eran muy superiores al del resto de algoritmos.

Analizando los datos de los enlaces que daban más error, nos dimos cuenta de que el error se debía a que como solo se utilizaba un vecino, la predicción se veía estropeada porque el vecino muy cercano era muy malo para realizar la predicción, ya que hacía que el algoritmo predijese 1 cuando lo lógico viendo el conjunto de entrenamiento era que predijese 0. Por lo tanto, la solución pasaba por aumentar el valor de k, haciendo que el peso de dicho vecino malo se vea reducido. Para tal fin se han realizado experimentos en WEKA, variando k desde 1 hasta 10, y después se ha utilizado la validación cruzada para elegir el mejor valor de k para cada enlace. En la Figura 15 y en la Figura 16 se pueden ver los valores promedios de MAE obtenidos y los *boxplots* representando la dispersión del error, respectivamente.

Como se puede observar, los resultados son muy satisfactorios al aumentar el valor de k, obteniéndose con k igual a 10 el mejor resultado. Por otra parte, podemos comprobar que el uso de la validación cruzada no ayuda, porque para los enlaces que el comportamiento es malo con k igual a 1, elige k igual a 1, al utilizarse la validación cruzada únicamente con el conjunto de entrenamiento, es decir, sobre el conjunto de entrenamiento el valor k igual a 1 es el más óptimo, pero el algoritmo no es capaz de generalizar bien (se produce un sobreajuste al conjunto de entrenamiento). Ya que con k igual a 10 obtenemos el mejor valor, utilizamos éste, pese a que es posible que haya un k mayor que sea aún mejor, aunque el aumento de k eleva la carga computacional.

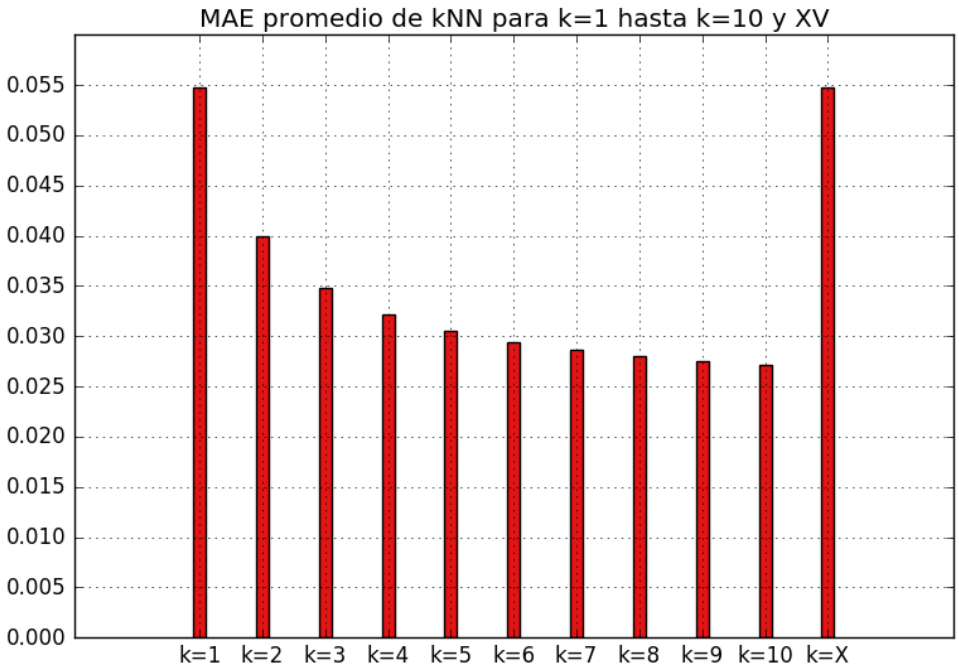


Figura 15: MAE promedio del algoritmo kNN, variando k de 1 hasta 10, y con validación cruzada.

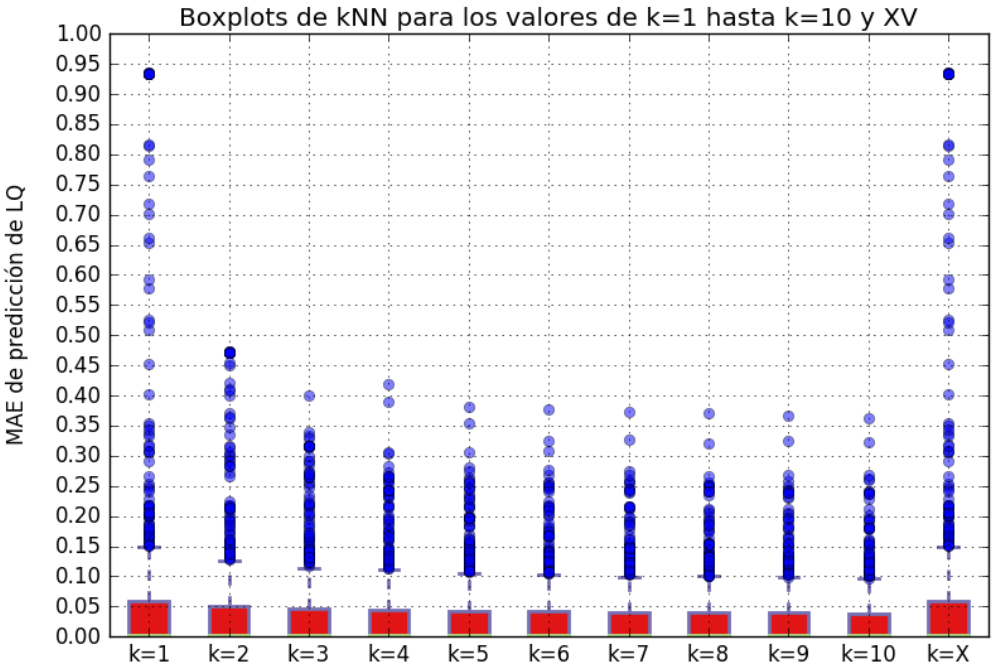


Figura 16: : Boxplots comparando el MAE de la predicción de LQ del algoritmo kNN, variando k de 1 hasta 10, y con validación cruzada.

3.5. Análisis del impacto del tamaño de la ventana temporal

Siguiendo los experimentos realizados en [7], se ha realizado el correspondiente a la sección 4.2. En dicho experimento se variaba la ventana temporal (*lag window*) desde 1 unidad hasta 24 unidades, y se analizaba cómo afecta a las predicciones realizadas. En su caso, el algoritmo RT era el que mejores resultados obtenía en la sección anterior, pero dado que en nuestro caso era SVM, y que kNN obtenía un valor semejante al de RT, vamos a realizar por completitud el mismo análisis para los 3 algoritmos, es decir, todos menos GPR. Cabe destacar que este experimento ha sido computacionalmente costoso en el caso del algoritmo SVM, tardando más de un día en realizarse.

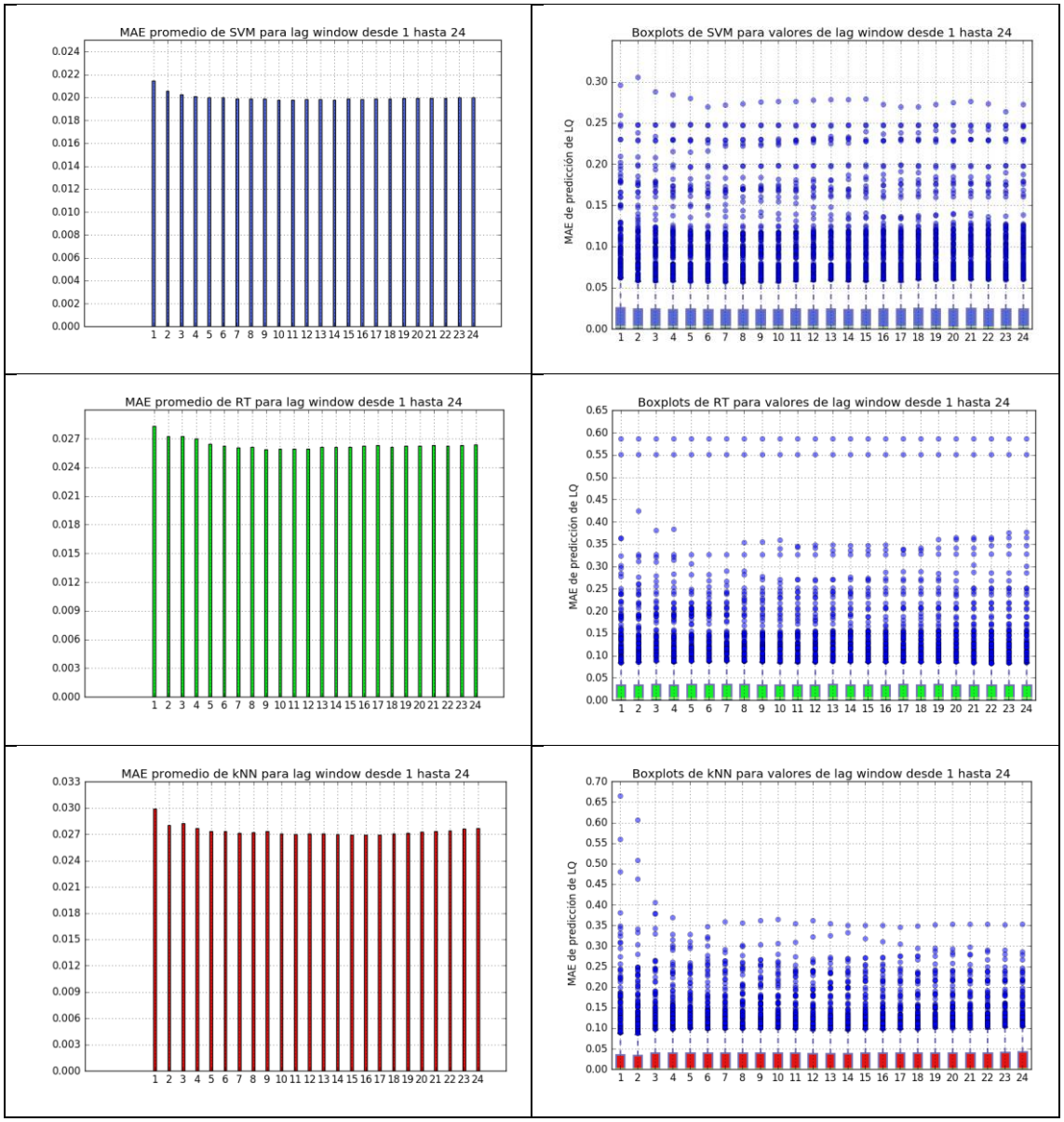


Tabla 4: gráficos de barras y boxplots representando el MAE promedio y el MAE de la predicción de LQ, respectivamente, para los algoritmos SVM, RT y kNN, con lag window de 1 a 24 unidades

En la Tabla 4 se pueden ver los resultados obtenidos. A la vista de los resultados, podemos ver que se obtienen unos resultados ligeramente peores para tamaños de *lag window* muy pequeños, pero en cuanto aumenta un poco los resultados son extremadamente parecidos entre sí. Cabe destacar que en el algoritmo kNN se producen unos cuantos valores atípicos con ventana de tamaño 1 y 2, pero nada especialmente representativo. Los autores de [7] reportan que después de realizar un T-test (que como ya hemos indicado anteriormente no es lo adecuado) no se puede elegir un valor óptimo de ventana temporal, y que mantienen el valor 12 para los siguientes experimentos. No se justifica de manera formal la elección de este valor, que si bien puede tener sentido en el contexto del problema (12 instancias corresponden a una hora de valores de LQ), no se sostiene si tenemos en cuenta los buenos resultados obtenidos con ventanas más pequeñas. En nuestro caso, realizando un Z-test, llegamos a la misma conclusión que los autores del estudio, por lo que, pese a que pensamos que podría emplearse un valor de ventana menor (lo que disminuiría la carga computacional de los algoritmos), mantenemos el valor 12 para reproducir los siguientes experimentos.

3.6. Predicción con un horizonte más alejado

Siguiendo los pasos de la sección 4.3 de [7], se va a estudiar si es posible predecir el valor de LQ a varios instantes vista en el futuro. Para ello, en el artículo se continúa con el mismo experimento base que en secciones anteriores, ventana temporal de 12 unidades, conjunto de entrenamiento de 1728 instancias (6 días) y conjunto de entrenamiento de 288 instancias (1 día). En el artículo se utiliza el algoritmo RT para realizar los experimentos, dado que es el algoritmo con el que mejores resultados habían obtenido. Aunque en nuestro caso no era el mejor algoritmo, como en la sección anterior hicimos el experimento con el resto de algoritmos con los que obteníamos buenos resultados, y vimos que prácticamente no había diferencia de comportamiento entre ellos, en este caso sólo realizamos los experimentos con RT. A continuación, predicen desde 1 hasta 8 unidades hacia el futuro. Para realizar predicciones con un horizonte superior a 1, el módulo de series temporales simplemente calcula una predicción tal y como se explicó en secciones anteriores, y después de desplazar la ventana, esta predicción forma parte los datos con los que se alimenta al predictor, en lugar de con los datos reales. Con los resultados obtenidos, se han dibujado la Figura 17 y la Figura 18 , que muestran el gráfico de barras, y los *boxplots*, respectivamente.

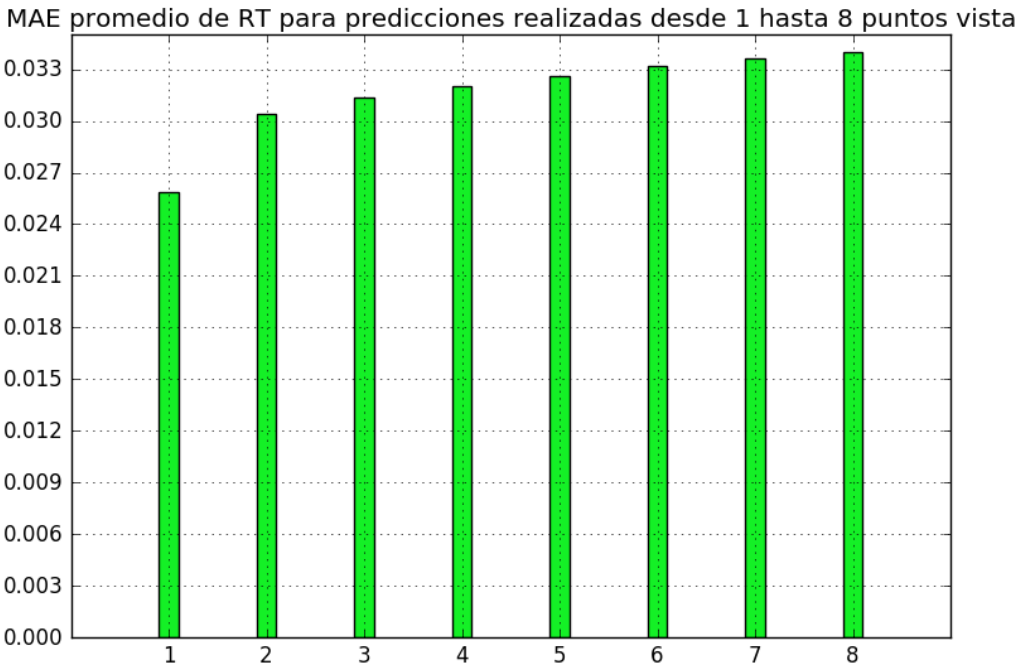


Figura 17: MAE promedio de RT para predicciones realizadas desde 1 hasta 8 puntos vista.

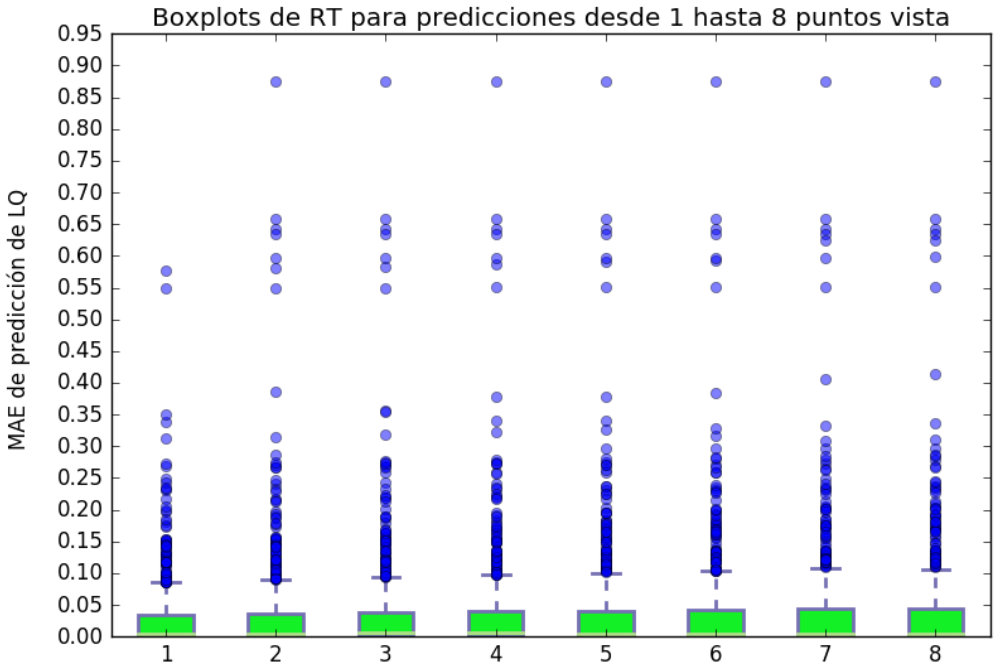


Figura 18: : Boxplots comparando el MAE de la predicción de LQ de RT para predicciones desde 1 hasta 8 puntos vista.

A la vista de los resultados obtenidos, en comparación con los resultados reportados por los autores del artículo, podemos decir lo siguiente:

En nuestro caso, al igual que en el artículo, el MAE promedio aumenta poco a poco al realizar las predicciones cada vez a más instantes vista. La mayor diferencia la observamos entre las predicciones a 1 y 2 instantes, que es lo mismo que se observa en la gráfica del artículo. Por otro lado, atendiendo a la variabilidad del error, al igual que en el artículo, los valores para la mediana y el primer cuartil son prácticamente iguales y se produce un aumento en el valor del tercer cuartil y de los valores atípicos, que son los que contribuyen a aumentar el MAE promedio. Todo esto nos hace concluir que es viable realizar predicciones con un error razonable con un horizonte más alejado en el tiempo, lo que podría beneficiar al encaminamiento en la red, ya que, por una parte, la mayor parte de los errores son pequeños, y apenas aumentan con el tiempo, y por otra, los errores grandes, son pocos, y dado que sus predicciones no son útiles en última instancia (ya que son tan grandes que es mejor no utilizar predicción), nos da igual que aumenten un poco más.

3.7. Degradación del modelo con el paso del tiempo

Finalmente, en esta sección se han realizado los experimentos correspondientes a la sección 4.4 de [7]. La idea de estos experimentos era constatar que a medida que pasa el tiempo, el modelo construido se degrada, en el sentido de que deja de ser capaz de representar fielmente la evolución de la LQ. Para ello, lo que se propone en el artículo es construir un conjunto de entrenamiento formado por las primeras 288 medidas de LQ, es decir, lo correspondiente a un día de medidas. Por otra parte, se utiliza la ventana temporal de 12 unidades y se realizan predicciones a un instante vista. Para poder medir cómo se va degradando a lo largo del tiempo el modelo, se plantea un conjunto de test variable, desde 144 medidas de LQ (medio día) hasta 1728 (6 días). De esta manera, es posible ver que las predicciones que se realizan más alejadas del instante del entrenamiento obtienen un error mayor. Otra posible manera de ver esta degradación habría sido calcular el error en intervalos disjuntos del mismo tamaño, viendo cómo a medida que pasa el tiempo el error es mayor. No obstante, por ahora nos circunscribiremos a las condiciones establecidas por el artículo. Al igual que en la sección anterior, realizamos los experimentos con RT. En la Figura 19 y en la Figura 20 se pueden observan el gráfico de barras y los *boxplots*, respectivamente.

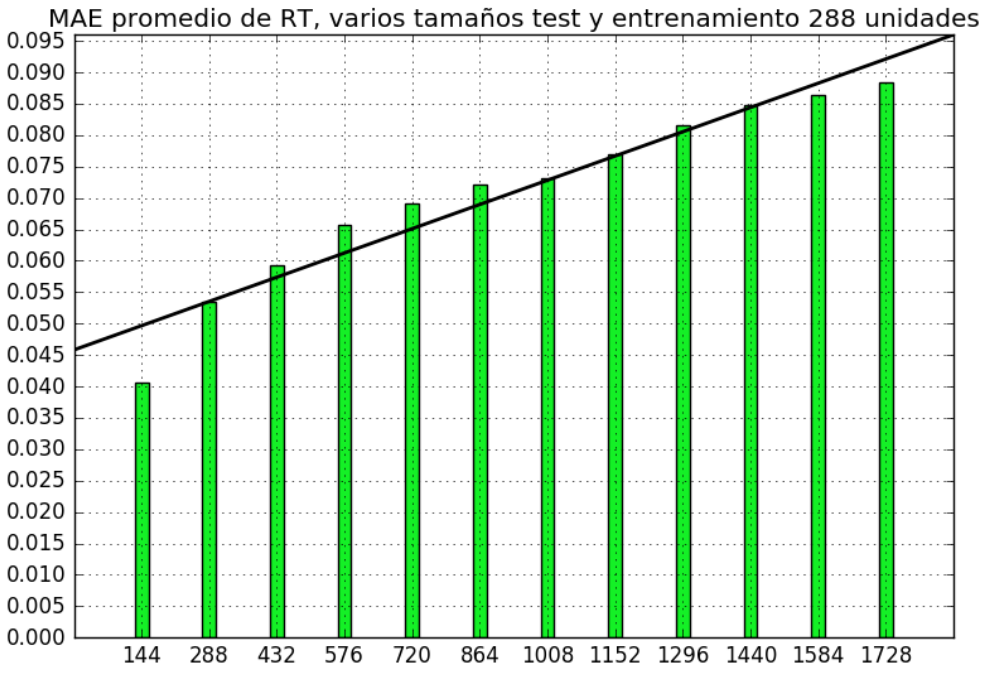


Figura 19: MAE promedio de RT para predicciones realizadas con un conjunto de entrenamiento de tamaño 288 y varios tamaños de conjunto de test.

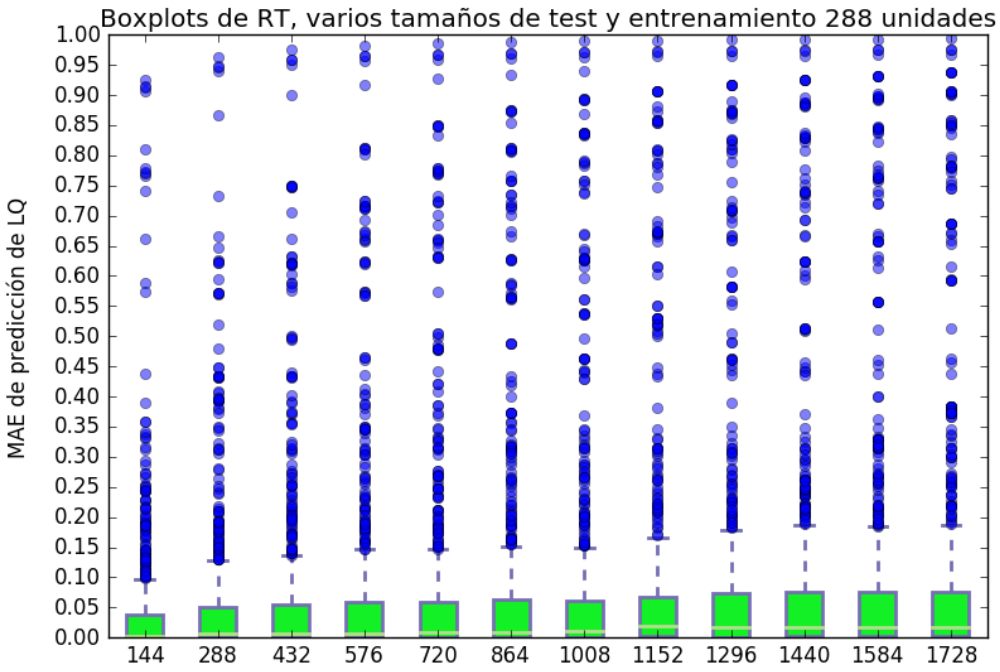


Figura 20: : Boxplots comparando el MAE de la predicción de LQ de RT para predicciones realizadas con un conjunto de entrenamiento de tamaño 288 y varios tamaños de conjunto de test.

A la vista de los resultados obtenidos, podemos destacar varias conclusiones. En el artículo se observa un aumento del MAE promedio perfectamente lineal, que utilizando regresión lineal se obtiene una recta de regresión MAE = 0.0132 + 0.0212 * test, variando test de 1 a 12 unidades (Tenemos 12 conjuntos de test diferentes). En nuestro caso, el aumento del MAE promedio no es tan lineal como en el caso del artículo, como se puede ver en la Figura 3. Además, la recta de regresión que obtenemos no se parece, obteniéndose MAE = 0.045843 + 0.0038 * test.

A partir de estos resultados, se puede concluir que en nuestro caso el error aumenta de manera más lenta de lo que aumenta en el artículo, lo que podría ser debido a que en el conjunto de datos que estamos utilizando existen menos variaciones, y, por tanto, el modelo no se degrada tan rápido como en el artículo. También podría ser debido a que la selección de enlaces realizada en la etapa inicial ha sido diferente que la de los otros autores, pero esto no lo podemos confirmar.

Finalmente, a la vista de los *boxplots* obtenidos, vemos que, a partir del conjunto de test de 1152 medidas, la mediana y el tercer cuartil permanecen prácticamente en el mismo valor, y el aumento del error promedio es debido a los puntos singulares, es decir, el error sigue aumentando porque hay un conjunto de enlaces que cada vez son representados peor por el modelo que se creó del conjunto de entrenamiento de 288 unidades. El resto de enlaces, parece que siempre se mantienen en la misma cota de error.

3.8. Conclusiones

Con la réplica de los experimentos del trabajo de [7], se ha conseguido entender alguna de las limitaciones de su enfoque, como la degradación de los modelos con el tiempo. No obstante, aún queda trabajo en ese sentido, que continuará en el capítulo siguiente.

En cuanto a la réplica en sí de los experimentos, ha sido completamente satisfactoria. Los resultados obtenidos inicialmente, si bien no son idénticos que los reportados por los autores, se acercan bastante. No obstante, sí que hemos observado algunas diferencias, como que con algún algoritmo obtenemos mejores resultados que los otros autores, quizá debido a una selección de parámetros diferente. Por otra parte, en cuanto al experimento del tamaño de la ventana temporal, llegamos a la misma conclusión que ellos, que es que no se puede elegir un valor óptimo de ventana. En el experimento de predicción con un horizonte más alejado también obtenemos resultados parecidos, destacándose la conclusión de que es posible predecir a más de un instante vista sin perjudicar en gran medida a las predicciones. Finalmente, obtenemos resultados ligeramente diferentes en el experimento de degradación del modelo, pero las conclusiones extraídas son

similares: el modelo se degrada a medida que se predice en tiempos más alejados del conjunto de entrenamiento.

En la parte restante del documento se presentarán los nuevos experimentos realizados, que, por una parte, permitirán comprender finalmente todas las limitaciones del trabajo de [7], como por ejemplo la importancia del coste computacional; y por otra nos permitirán abrir nuevas vías de investigación, proponiendo nuevos algoritmos y técnicas para solucionar el problema de la predicción de la calidad del enlace.

Capítulo 4. Propuesta de nuevos algoritmos

4.1. Introducción

En el capítulo anterior se han replicado los principales experimentos realizados en [7], permitiéndonos extraer algunas conclusiones acerca de las debilidades de su trabajo. No obstante, no todos los aspectos relevantes estaban presentes en sus experimentos, como, por ejemplo, la carga computacional, o la posibilidad de reentrenamiento de los modelos.

En primer lugar, en este capítulo se analizará la influencia del *timestamp* en los algoritmos de aprendizaje en lote, y posteriormente, se analizará el efecto que tiene entrenar los algoritmos durante un menor tiempo. Después, se establecerá un *baseline* muy sencillo que nos permita comparar todos los algoritmos con él, de manera que un algoritmo de aprendizaje automático deba ser mejor que el *baseline* para que su complejidad esté justificada. A continuación, se estudiarán los resultados obtenidos por los algoritmos de aprendizaje en línea, en comparación con los algoritmos de aprendizaje en lote, y con estos mismos reentrenados periódicamente. Finalmente, se extraerán unas breves conclusiones.

4.2. Estudio de la influencia del *timestamp* en algoritmos en lote

Tal y como se comentó en el capítulo anterior, a la hora de realizar experimentos con el módulo de series temporales de WEKA, nos encontramos con que, en la configuración por defecto, además de crearnos el vector de características con los valores pasados de la variable (la ventana temporal), nos creaba características derivadas, formadas por el *timestamp*, los valores de *timestamp* al cuadrado y al cubo, y producto de las variables temporales con el *timestamp*, como ya se ilustró en la Figura 10. Dado que los autores de [7] no especifican si han utilizado el programa con los parámetros por defecto, en el capítulo anterior lo hicimos de dicha manera, pero en este momento vamos a pasar a estudiarlo. Por completitud, compararemos el caso estudiado en el capítulo anterior, es decir, conjunto de entrenamiento de 6 días y conjunto de test de 1 día, ventana temporal de 12 unidades, predicción a 1 unidad vista, potencias del *timestamp* y productos del *timestamp* con las variables temporales con otros dos casos: el caso en el que se utiliza únicamente el *timestamp* además de las variables temporales, y el caso de series temporales puras, sin *timestamp*. En la Figura 21 se pueden ver las características utilizadas en cada caso.

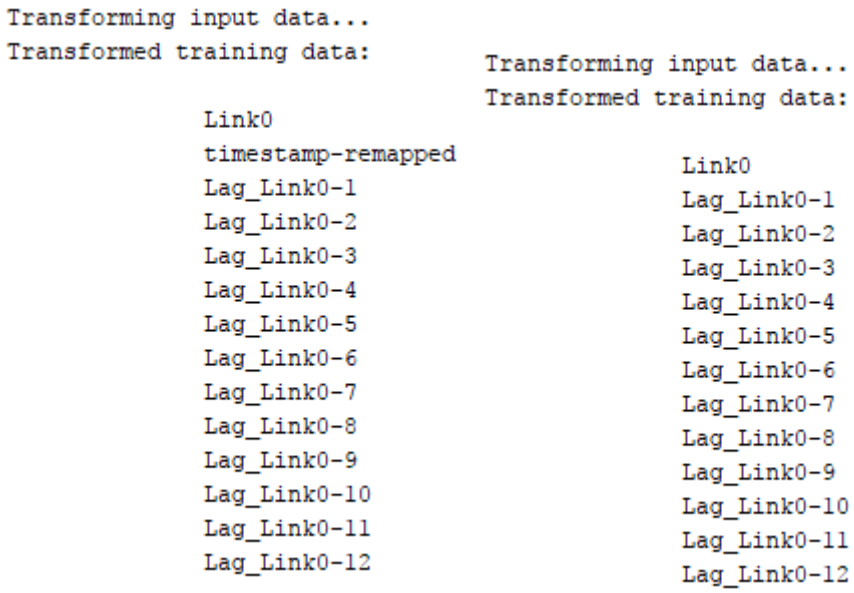


Figura 21: características de los conjuntos de entrenamiento utilizados.

A partir de ahora, y por simplicidad, se representarán de manera conjunta los *boxplots* junto al MAE promedio. En la Figura 22 se observan los resultados obtenidos. Por cada algoritmo, tenemos tres *boxplots*, de izquierda a derecha son: en primer lugar, uso de *timestamp*, potencias del *timestamp* y productos del *timestamp* por las variables temporales; a continuación, únicamente uso del *timestamp* junto a las variables temporales; y finalmente uso sólo de las variables temporales. El diamante color rosa claro identifica el valor de la media.

Además de los errores, consideramos relevante el tiempo empleado para realizar los entrenamientos y los test, al contrario que [7], que no lo considera en su estudio pese a ser un aspecto muy importante a la hora de una posible implantación de un sistema de predicciones en un entorno real. Como se puede ver en la Figura 23, el coste computacional difiere en gran medida de un algoritmo a otro, por lo que hay que tenerlo en consideración a la hora de analizar los resultados. El coste computacional se medirá a partir de ahora en segundos de CPU. Las medidas se han realizado con una utilidad proporcionada por MOA. De acuerdo a los autores de la herramienta, la utilidad mide el tiempo transcurrido, y no el tiempo que realmente ha pasado en la CPU el algoritmo, por lo que dicha medida podría verse afectada por otros programas ejecutándose simultáneamente. Por ello, los resultados podrían no ser fiables para comparar tiempos en el mismo orden de magnitud, pero si nos servirán para observar la carga generada por algoritmos cuando haya diferencias de varios órdenes de magnitud. Además, se ha intentado que todas las simulaciones se ejecutaran con las mismas condiciones (no otros programas ejecutándose, misma configuración de rendimiento del ordenador).

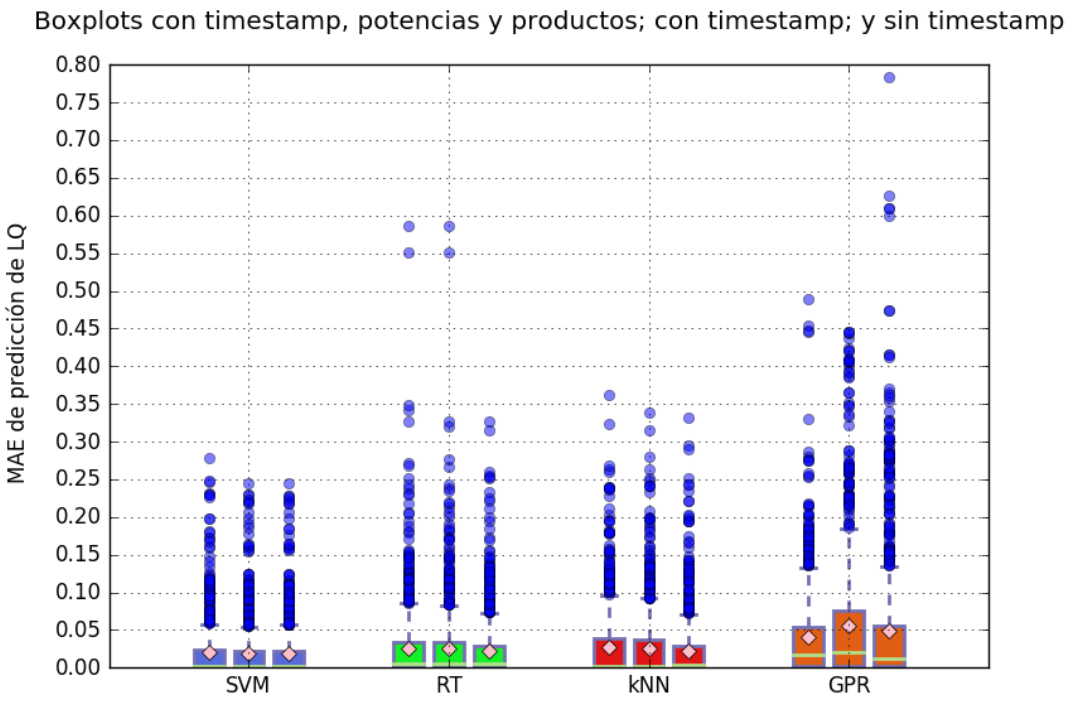


Figura 22: : Boxplots comparando el MAE de la predicción de LQ de los 4 algoritmos de aprendizaje en lote (6 días entrenamiento, 1 día test) para 3 casos: uso de timestamp, potencias del timestamp y productos del timestamp con las variables temporales;con timestamp; y sin timestamp

Una vez obtenidos los resultados, podemos extraer varias conclusiones. En primer lugar, el uso del *timestamp*, tanto en la versión con potencias y productos como en la versión simple, perjudica a los tres algoritmos que obtienen buenos resultados, SVM, RT y kNN, tanto en el tiempo de computación como en el MAE promedio obtenido. Esto es debido a que las características adicionales añadidas al espacio de entrada no tienen una buena capacidad predictiva. Al añadir más dimensiones al espacio de entrada, se necesita más tiempo para entrenar los algoritmos, es por ello que se tarda más. Con el peor algoritmo, GPR, ocurre algo curioso, ya que en el tiempo de computación se comporta al revés que el resto de algoritmos, y en cuanto al MAE promedio no se observa el mismo comportamiento que con el resto de algoritmos. No obstante, dado que este es el peor algoritmo tanto en tiempo como en MAE, no es relevante profundizar en ello. En cuanto a SVM, vemos que al eliminar las potencias y productos del *timestamp*, la carga computacional disminuye enormemente, aunque, por otra parte, sigue siendo mucho mayor que el obtenido por RT y kNN. Es decir, por un lado, tenemos un algoritmo muy bueno en MAE, que es SVM, pero que tarda mucho, y, por otra parte, otros dos algoritmos no tan buenos en cuanto al MAE obtenido, que son RT y kNN, pero su coste computacional es mucho más asumible.

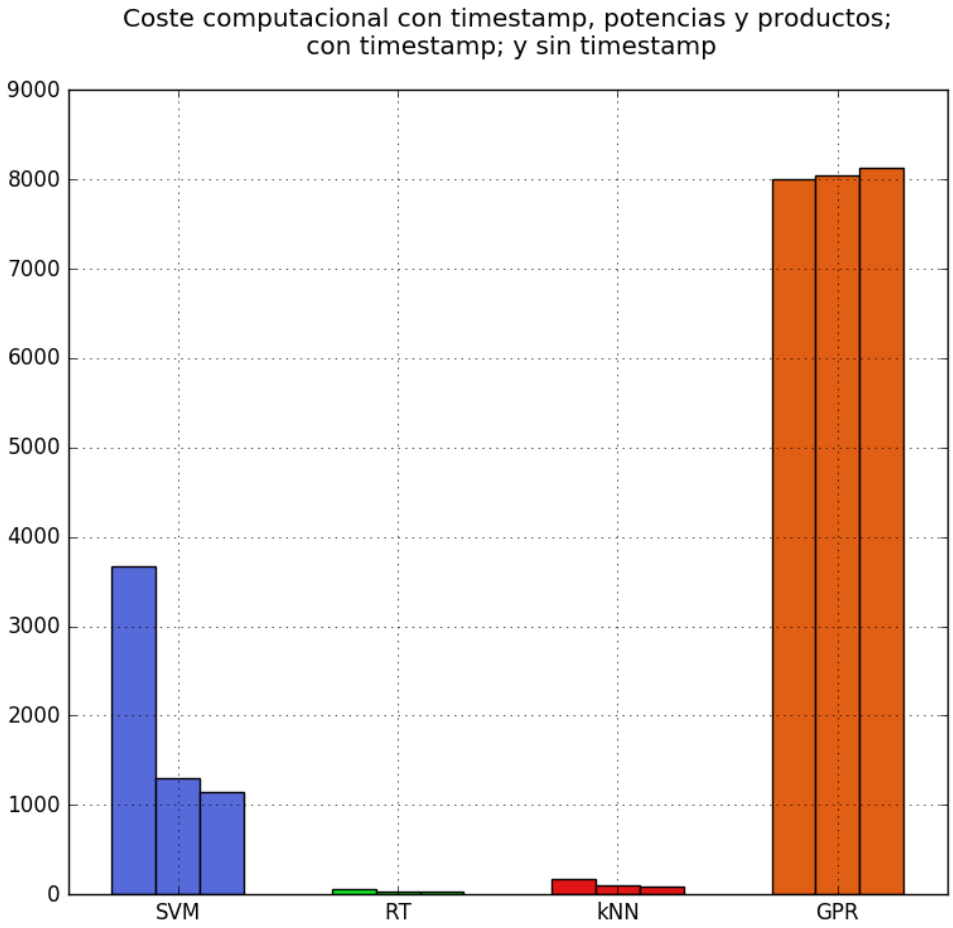


Figura 23: Coste computacional de los 4 algoritmos de aprendizaje en lote (6 días entrenamiento, 1 día test) para 3 casos: uso de timestamp, potencias del timestamp y productos del timestamp con las variables temporales;con timestamp; y sin timestamp

Por completitud, en la Tabla 5 se muestran los datos utilizados para realizar los gráficos, ya que debido a la diferencia de escalas algunos de ellos son difíciles de apreciar. Finalmente, cabe destacar que se han realizado Z-test de la manera realizada anteriormente, y para una significatividad estadística del 5%, solo obtiene que son distintos los MAE promedio de kNN con productos y potencias del que no tiene *timestamp*; y los de GPR con productos y potencias del *timestamp* del que tiene *timestamp* y del que no tiene *timestamp*, pero estos dos últimos entre sí no. Pese a que estadísticamente no haya evidencia de que son mejores sin *timestamp*, dado que en los algoritmos que obtienen buenos resultados el coste computacional es menor y la media es menor en el experimento ejecutado, de aquí en adelante si hay que hacer algún experimento se hará sin *timestamp*, excepto que se realicen los mismos tres experimentos reportados en esta sección.

Algoritmo	**MAE sin *timestamp***	**Tiempo CPU (s)**	**MAE con *timestamp***	**Tiempo CPU (s)**	**MAE con t^ y t*Lag**	**Tiempo CPU (s)**
SVM	**1.85%**	1150.28	1.88%	1300.86	1.98%	3673.33
RT	**2.26%**	30.88	2.54%	31.66	2.59%	54.42
kNN	**2.25%**	85.34	2.54%	102.81	2.71%	175.78
GPR	4.98%	8124.97	5.62%	8047.20	**4.08%**	8007.67

Tabla 5: resultados de MAE promedio y coste computacional de los 4 algoritmos de aprendizaje en lote (6 días entrenamiento, 1 día test) para 3 casos: uso de timestamp, potencias del timestamp y productos del timestamp con las variables temporales;con timestamp; y sin timestamp.

4.3. Algoritmos de aprendizaje en lote con entrenamiento más corto

Una vez visto el desempeño de los algoritmos de aprendizaje en lote en las condiciones anteriores, cabe preguntarse si un algoritmo que necesite 6 días de entrenamiento es apropiado para una red telemática. Además, hemos visto que el coste computacional del mejor algoritmo es demasiado alto, lo que podría imposibilitar su implementación en entornos reales. Por ello, hemos planteado dos nuevos experimentos, el primero de ellos consiste en entrenar durante 1 día y testear durante 6, mientras que el segundo consiste en entrenar durante 1 hora y testear durante 6 días y 23 horas. Para ello, en primer lugar, hemos de realizar la misma selección que se hizo en el capítulo previo. Siguiendo las mismas normas, en este caso nos quedamos únicamente con 610 enlaces para el experimento con 1 día de entrenamiento y 325 para el experimento con 1 hora de entrenamiento, por lo que además de entrenar menos tiempo, tenemos enlaces potencialmente más variables que en el caso anterior. Por completitud, y pese a que ya quedó demostrado en la sección anterior que el uso del *timestamp* no mejora los resultados, hemos realizado los experimentos en los tres casos que se desarrollaron anteriormente.

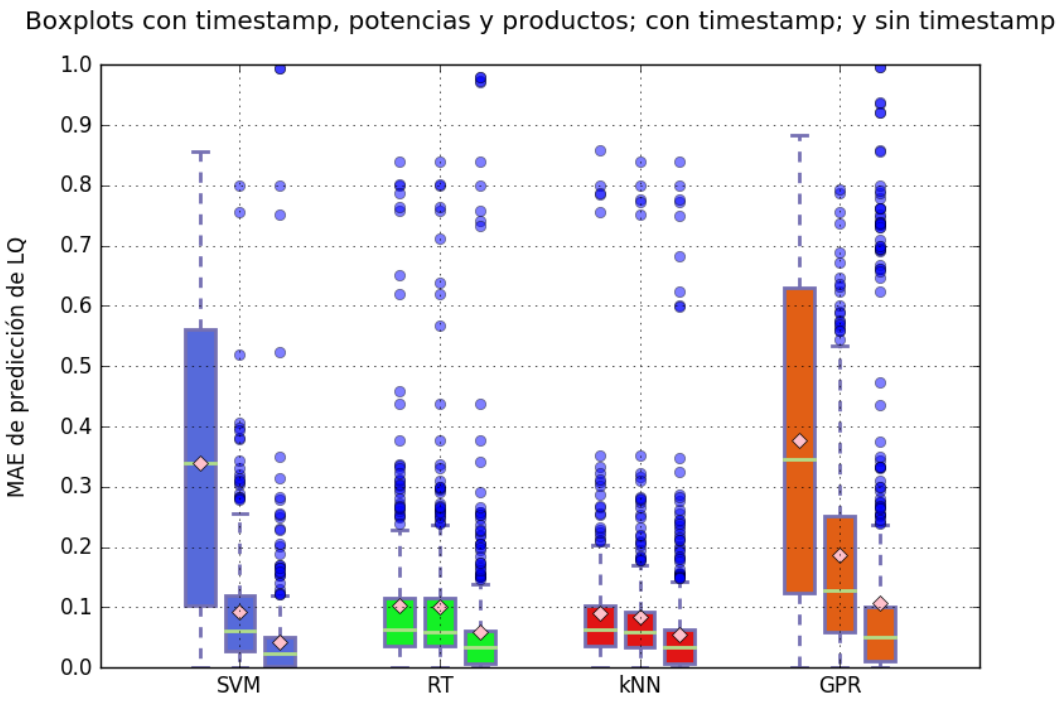


Figura 24: : Boxplots comparando el MAE de la predicción de LQ de los 4 algoritmos de aprendizaje en lote (1 día entrenamiento, 6 días test) para 3 casos: uso de timestamp, potencias del timestamp y productos del timestamp con las variables temporales;con timestamp; y sin timestamp

Algoritmo	**MAE sin timestamp**	**Tiempo CPU (s)**	**MAE con timestamp**	**Tiempo CPU (s)**	**MAE con t^ y t*Lag**	**Tiempo CPU (s)**
SVM	**4.23%**	67.15	9.23%	160.22	34.02%	153.81
RT	**5.85%**	52.92	10.14 %	106.73	10.27%	121.94
kNN	**5.55%**	90.42	8.41%	194.11	9.07%	187.83
GPR	**10.77%**	80.61	18.65%	137.03	37.69%	126.97

Tabla 6: resultados de MAE promedio y coste computacional de los 4 algoritmos de aprendizaje en lote (1 día entrenamiento, 6 días test) para 3 casos: uso de timestamp, potencias del timestamp y productos del timestamp con las variables temporales;con timestamp; y sin timestamp.

Al disminuir el tiempo de entrenamiento a 1 día, en la Figura 24 vemos como los resultados empeoran considerablemente, aunque los resultados obtenidos siguen el mismo patrón observado anteriormente: SVM es el algoritmo que mejor se comporta, RT y kNN se comportan de manera similar y GPR es el algoritmo que peor se comporta; por otra parte, los algoritmos se comportan mejor si no se utiliza *timestamp*. Destaca que SVM y GPR obtienen muy malos resultados con *timestamp* + potencias + productos.

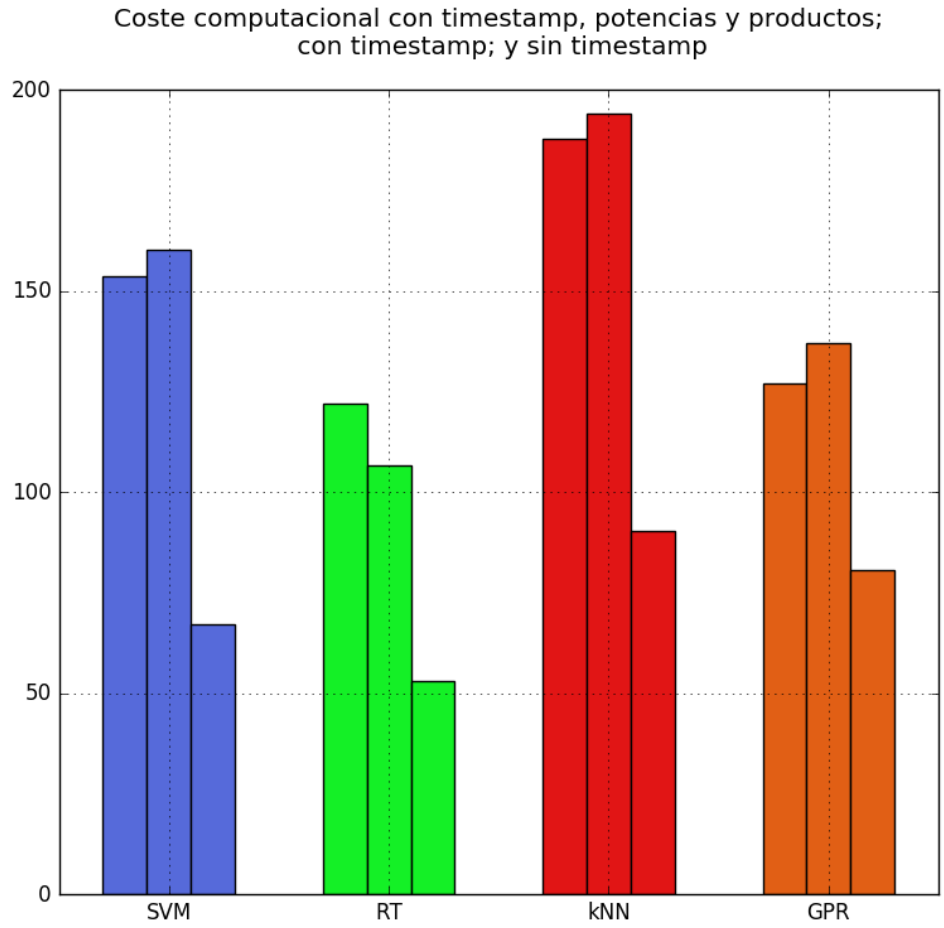


Figura 25: Coste computacional de los 4 algoritmos de aprendizaje en lote (1 día entrenamiento, 6 días test) para 3 casos: uso de timestamp, potencias del timestamp y productos del timestamp con las variables temporales;con timestamp; y sin timestamp

En cuanto al coste computacional, a la vista de la Figura 25, sigue la tendencia marcada en el caso anterior, si no se utiliza *timestamp* se consumen menos recursos, y los casos de utilizar *timestamp* y *timestamp* + potencias + productos se mantienen en niveles similares. No obstante, ahora los costes de todos los algoritmos son similares, y no hay unas diferencias tan acusadas como en el caso previo. En la Tabla 6 se pueden ver los valores utilizados para dibujar las gráficas.

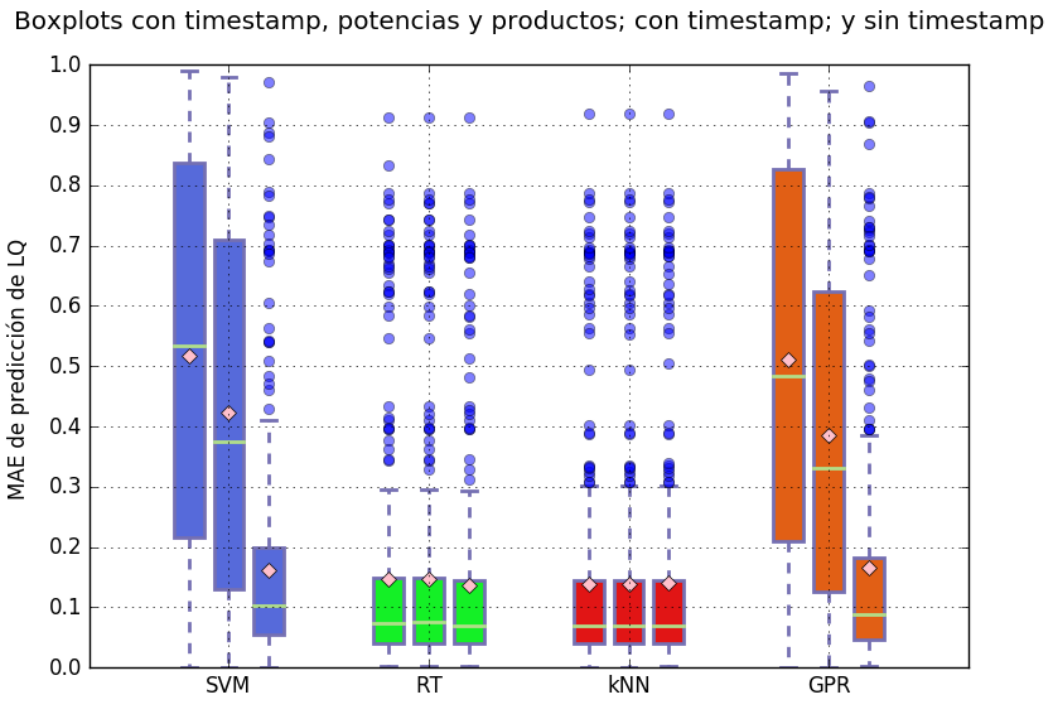


Figura 26: : Boxplots comparando el MAE de la predicción de LQ de los 4 algoritmos de aprendizaje en lote (1 hora entrenamiento, 6 días y 23 horas test) para 3 casos: uso de timestamp, potencias del timestamp y productos del timestamp con las variables temporales;con timestamp; y sin timestamp

Algoritmo	**MAE sin timestamp**	**Tiempo CPU (s)**	**MAE con timestamp**	**Tiempo CPU (s)**	**MAE con t^ y t*Lag**	**Tiempo CPU (s)**
SVM	**16.25%**	31.39	42.21%	35.86	51.67%	63.56
RT	**13.72%**	33.16	14.67%	37.78	14.72%	65.43
kNN	14.01%	32.375	**13.94%**	37.88	13.94%	64.70
GPR	**16.53%**	32.00	38.59%	37.03	51.03%	62.76

Tabla 7: resultados de MAE promedio y coste computacional de los 4 algoritmos de aprendizaje en lote (1 hora entrenamiento, 6 días y 23 horas test) para 3 casos: uso de timestamp, potencias del timestamp y productos del timestamp con las variables temporales;con timestamp; y sin ti

Finalmente, con el entrenamiento de una hora los resultados son muy malos, como se puede ver en la Figura 26. Podemos destacar que tanto RT como kNN se mantienen prácticamente iguales en cualquiera de los 3 casos, mientras que SVM y GPR lo hacen mucho peor en los dos primeros casos.

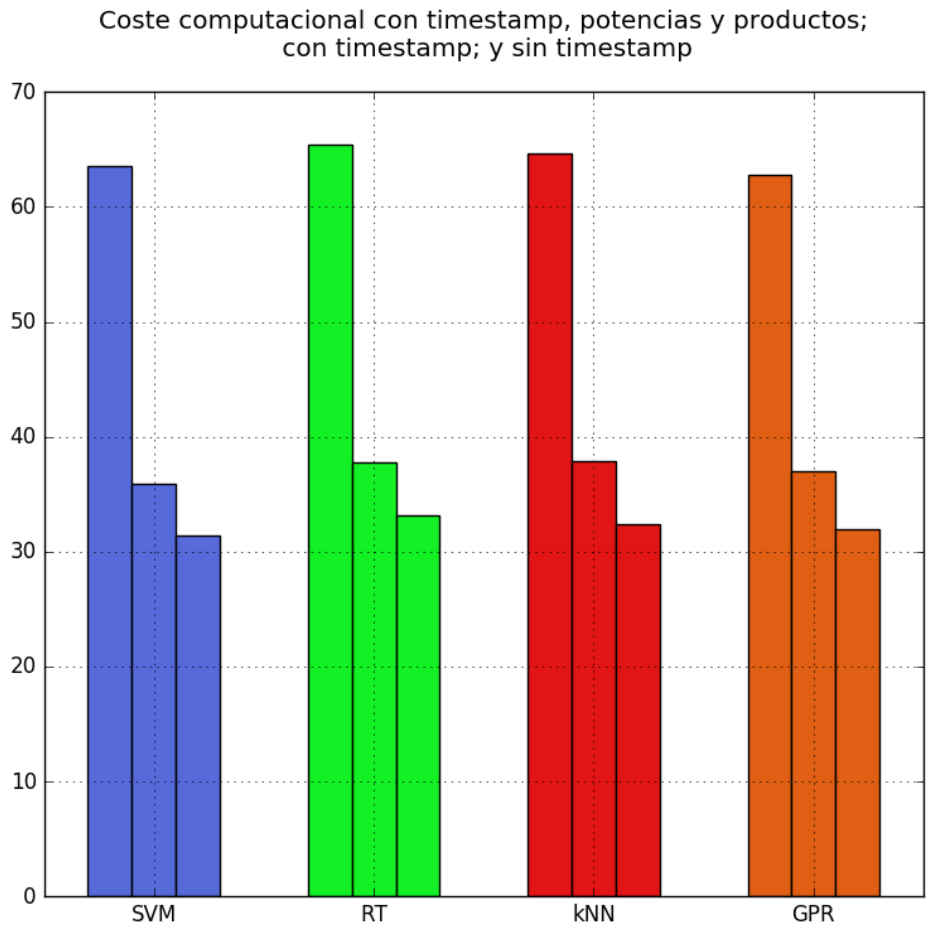


Figura 27: Coste computacional de los 4 algoritmos de aprendizaje en lote (1 hora entrenamiento, 6 días y 23 horas test) para 3 casos: uso de timestamp, potencias del timestamp y productos del timestamp con las variables temporales;con timestamp; y sin timestamp

En cuanto al coste computacional, si bien el caso sin *timestamp* sigue siendo el menos exigente, el caso en el que se usa el *timestamp* no queda muy lejos, como se ve en la Figura 27. Los datos utilizados quedan recogidos en la Tabla 7.

Finalmente, en la Figura 28 comparamos el resultado de los algoritmos sin *timestamp* para los tres casos considerados: entrenamiento de 6 días y test de 1 día; entrenamiento de 1 día y test de 6 días; y entrenamiento de 1 hora y test de 6 días y 23 horas. Como se puede ver, a medida que disminuimos el conjunto de entrenamiento, los resultados son peores. Esto es debido a dos cosas, en primer lugar, como ya se comentó en el capítulo anterior, a medida que testeamos más lejos del conjunto de entrenamiento, el modelo está más degradado. Por otra parte, dado que entrenamos con menos información, el modelo construido es potencialmente menos completo, ya que se puede enfrentar a situaciones no contempladas en el conjunto de entrenamiento.

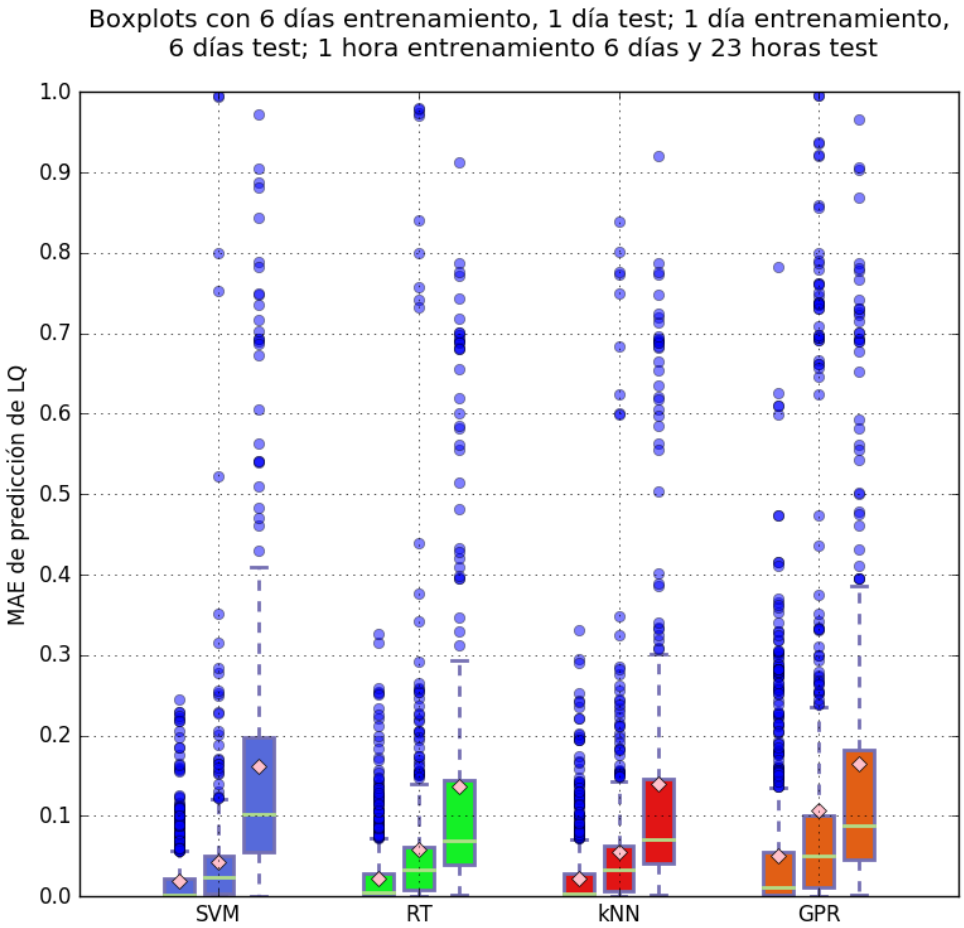


Figura 28: Boxplots comparando el MAE de la predicción de LQ para tres casos, de izquierda a derecha: entrenamiento de 6 días y test de 1 día; entrenamiento de 1 día y test de 6 días; y entrenamiento de 1 hora y test de 6 días y 23 horas.

4.4. Fijando un *baseline* comparativo

Uno de los fallos que presenta el trabajo desarrollado por [7], es que no compara los algoritmos que propone con lo que pretende sustituir. En la implementación existente de OLSR, las rutas se calculan teniendo en cuenta el valor actual de LQ, es decir, se asume que el valor de LQ no va a variar mucho con el paso del tiempo, por lo que el valor LQ actual es un buen predictor del LQ futuro. En [7] se afirma que esto puede no ser cierto, siendo este el punto de partida de su investigación, y también de este Trabajo Fin de Grado, por lo que es necesario comprobar que es posible y viable construir mejores predictores de la LQ que el último valor observado. Por ello, en esta sección se pretende estudiar cómo de mejores son los algoritmos propuestos respecto a este simple mecanismo de predicción, que a partir de ahora consideraremos como *baseline*. Para ello, hemos simulado un predictor que realiza exactamente eso, tomar el valor anterior

conocido, y después calcular el error que existe con el valor real. En la Figura 29 se pueden ver los resultados obtenidos, el diamante representa el MAE promedio.

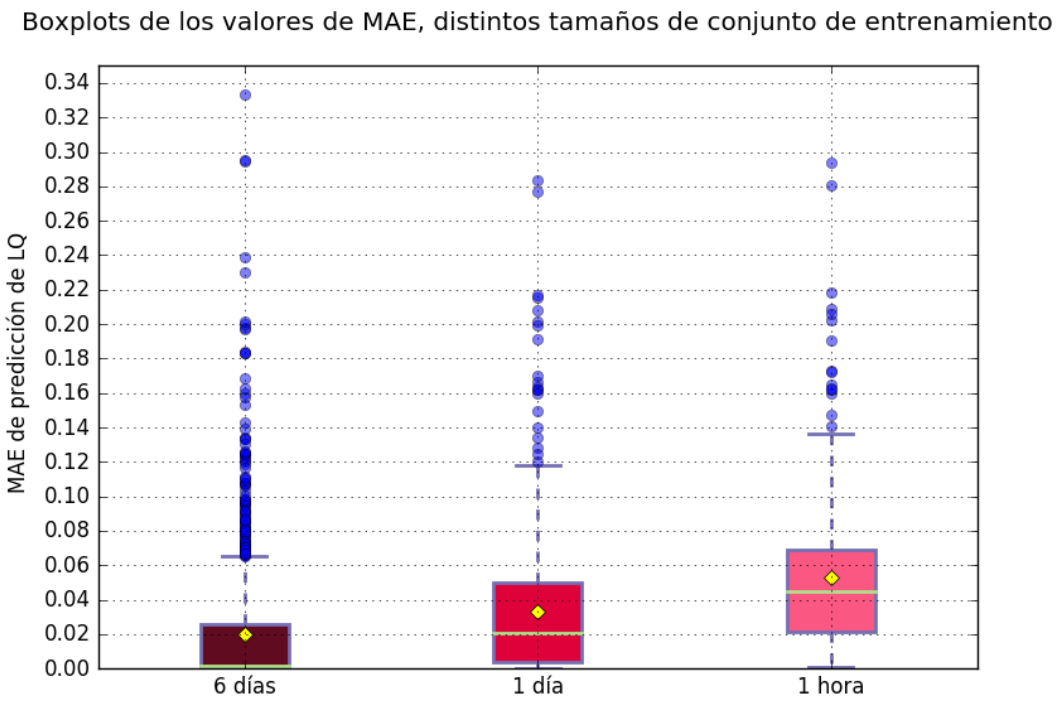


Figura 29: : Boxplots comparando el MAE de la predicción de LQ del algoritmo baseline, para distintos tamaños de entrenamiento y test.

Los resultados obtenidos son muy buenos, en el caso del entrenamiento de 1 día y entrenamiento de 1 hora, son mejores que cualquiera de los algoritmos utilizados. Concretamente, los valores de MAE promedio son 2.02%, 3.31% y 5.31%. Por tanto, de todos los algoritmos planteados hasta el momento, los únicos que pueden competir con el *baseline* son SVM y quizá RT y kNN, pero solo si se entrena con 6 días y se testea en el último, y sin utilizar *timestamp*. En la Figura 30, se muestra la comparativa entre el *baseline* y los algoritmos mencionados. Realizando Z-test, aparte de obtener lo que ya sabíamos (SVM distinto de RT y kNN, RT y kNN no distintos), obtenemos que para una significatividad estadística de 5%, las medias de los 3 algoritmos y la del *baseline* no son diferentes. Es decir, pese a que en promedio SVM es mejor que el *baseline*, el test nos dice que no hay evidencia estadística que lo confirme, seguramente debido a una concentración de errores en torno a 0.23 que se observa en SVM. Si el *baseline* presenta buenos resultados, es porque en promedio, los enlaces son muy estables. Es posible que para los enlaces con mayor variabilidad no funcione bien, pero no son excesivos.

Por tanto, vistos los resultados obtenidos, necesitamos conseguir algoritmos que obtengan mejores resultados que el *baseline*, y que, además, esta diferencia sea estadísticamente comprobable. Además, es importante recalcar que, pese a que el

baseline obtiene unos buenos resultados en promedio, si comete errores frecuentes en unos pocos enlaces puede afectar al resultado del algoritmo de encaminamiento, pese a que los valores de MAE reportado apenas varíen.

Por otra parte, el algoritmo que mejores resultados obtiene es SVM, y ya hemos visto que tiene una carga computacional muy elevada para conjuntos de entrenamientos grandes. Además, con conjuntos de entrenamiento pequeños, ninguno de los algoritmos se acerca al resultado del *baseline*. Por todo esto, se hace evidente la necesidad de otro tipo de algoritmos, que nos permitan obtener buenos resultados y por otra parte no tengan un gran coste computacional. Los algoritmos de aprendizaje en línea son unos buenos candidatos, ya que además de que están optimizados para ejecutarse en tiempo real (es decir, el procesamiento de una nueva instancia debe ejecutarse lo más rápido posible), nos permiten actualizar el modelo de manera continua, lo que nos permite solucionar los dos problemas observados anteriormente: al actualizar el modelo la degradación no va a ocurrir, porque siempre vamos a tener un modelo actualizado; y al no tener un conjunto de entrenamiento acotado, siempre vamos a poder aprender de todos los datos recibidos (aunque probablemente las tendencias observadas en tiempos pasados cada vez tengan menos peso en las predicciones).

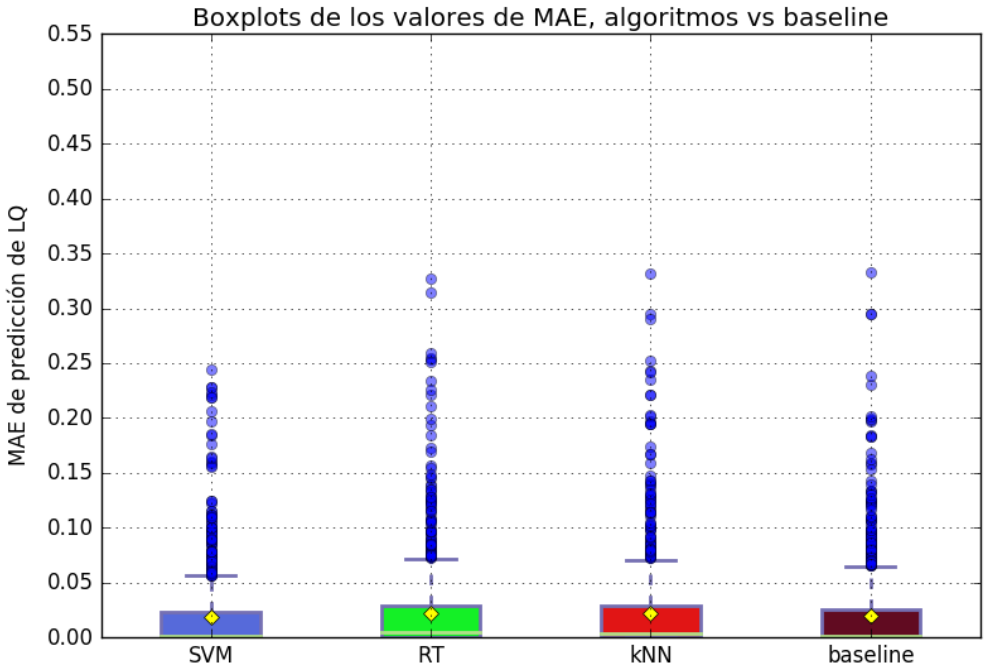


Figura 30: : Boxplots comparando el MAE de la predicción de LQ entre los algoritmos de aprendizaje en lote y el baseline.

4.4. Algoritmos de aprendizaje en línea

Tal y como se comentó en la sección anterior, los algoritmos de aprendizaje en línea pueden ser la solución que nos permita obtener errores pequeños, manteniendo una carga computacional baja, y además mejorando los valores del *baseline*. Para comprobar si es así, se han realizado unos nuevos experimentos con MOA, el equivalente de WEKA para aprendizaje en línea. Pese a que la filosofía del aprendizaje en línea rompe totalmente con la concepción de conjunto de entrenamiento y conjunto de test, dado que con cada nueva instancia recibida se actualiza el modelo, vamos a mantener los conjuntos de datos generados para WEKA, con el fin de que la comparativa sea lo más justa posible, es decir, se ha utilizado la selección previa de 979 enlaces.

Tras probar varios algoritmos de aprendizaje automático en línea, se han elegido cuatro que presentaban buenos resultados: Perceptron, AMRulesRegressor, ORTO y FadingTargetMean, en algunos de ellos se ha ajustado algún parámetro[28]. Se ha realizado un experimento variando los tamaños de ventana utilizada, de 1 a 12 unidades, cuyos resultados aparecen en la Tabla 8. Este experimento se ha vuelto a realizar aquí, ya que, por una parte, dado que estamos utilizando algoritmos de aprendizaje automático aplicado a la predicción de series temporales, necesitamos saber cuál es el tamaño óptimo de ventana, es decir, cuantos valores pasados son necesarios para realizar una buena predicción. Asimismo, cuanto menor sea la ventana, menor potencia de cálculo se necesitará, lo que beneficiara al coste computacional de los algoritmos. Para que la comparativa fuese justa, hemos preentrenado los algoritmos durante 6 días, para a continuación empezar a testear a la vez que se actualizaba el modelo durante el último día (*test-then-train*).

[28] Configuración de los algoritmos:

- Perceptron (parámetros por defecto).
- AMRulesRegressor:
  - learnerOption = rules.multilabel.functions.AdaptiveMultiTargetRegressor -l (rules.multilabel.functions.MultiLabelTargetMeanRegressor -l (FadingTargetMean -f 0.5))
- ORTO:
  - gradePeriorOption = 20000
- FadingTargetMean:
  - fadingFactorOption = 0.5

Algoritmo	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12
Perceptron	2.4	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.2	2.2
AMRulesRegressor (optimizado)	**1.88**	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.2	2.2
ORTO (optimizado)	2.5	2.4	2.4	2.4	2.3	2.5	2.5	2.6	2.2	2.7	2.8	2.8
FadingTargetMean (optimizado)	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9	1.9

Tabla 8: resultados de MAE promedio de algoritmos de aprendizaje en línea para tamaños de ventana temporal de 1 a 12 unidades, testeando el último día.

Tanto AMRulesRegressor (con ventana temporal de 1 unidad) como FadingTargetMean (con cualquier ventana temporal se obtiene el mismo resultado) obtienen mejores resultados que los obtenidos por el *baseline*, y se acercan a los resultados obtenidos por SVM. Los resultados nos indican, dado que la ventana temporal obtenida es pequeña, que no hace falta mucha información del pasado, sino que basta con el pasado más reciente, lo que, además, redundará en una menor carga computacional.

No obstante, para una significatividad estadística del 5%, y realizando un Z-test, siguen sin ser distintos estos resultados, pese a que la media sea menor. En la Figura 31 se pueden ver los *boxplots* representando el error del algoritmo SVM con el algoritmo AMRulesRegressor (ventana 1), FadingTargetMean (ventana1) y el *baseline*. No se han indicado los tiempos de computación ya que son muy pequeños en cualquiera de los casos, no llegando a superarse los 30 segundos en ningún caso.

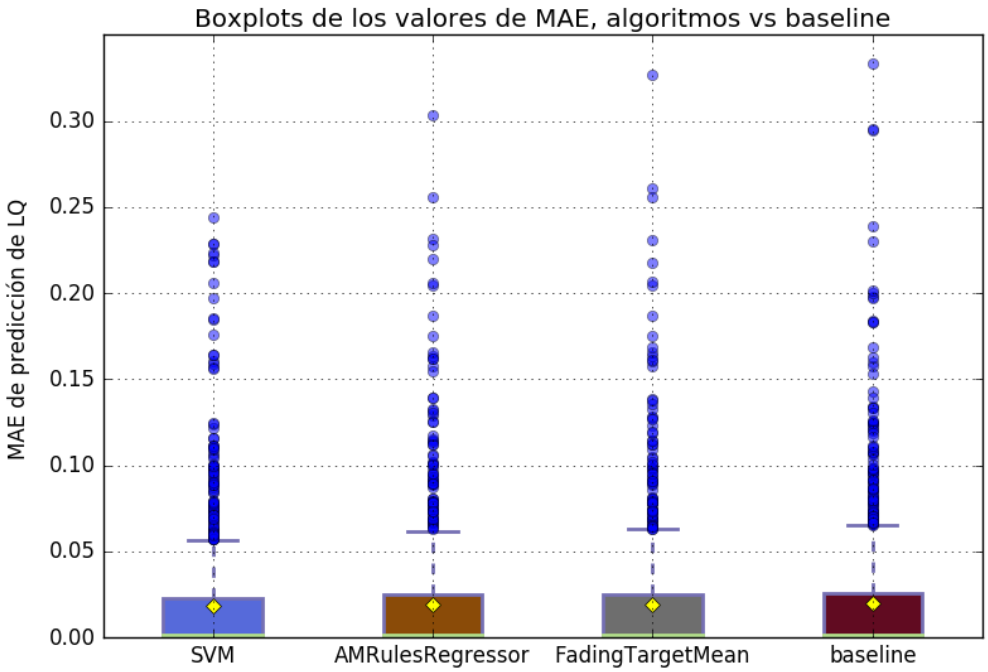


Figura 31: : Boxplots comparando el MAE de la predicción de LQ de los algoritmos SVM, AMRulesRegressor, FadingTargetMean testeando sobre el último día; y el baseline.

4.5. Comparativa entre algoritmos de aprendizaje en lote y de aprendizaje en línea

Como se vio en el apartado anterior, para el caso de un entrenamiento de 6 días y un test de 1 día, no hay diferencias estadísticamente significativas entre el mejor algoritmo en lote, SVM, y el mejor algoritmo en línea, AMRulesRegressor. No obstante, el coste computacional del primero frente al segundo es mucho mayor, pero, aun así, el del *baseline* sigue siendo menor, y no hay diferencias estadísticamente significativas entre AMRulesRegressor y el *baseline*. Por ello, en esta sección nos centraremos sobre los conjuntos de enlaces estudiados anteriormente, para los casos de entrenamiento de 1 día y 1 hora (610 y 325 enlaces respectivamente). En estos conjuntos de enlaces, los algoritmos de aprendizaje en lote se comportaban mucho peor que el *baseline*, y queremos comprobar si los algoritmos de aprendizaje en línea superan esta limitación. En la Tabla 9 se muestran los resultados para 1 día de entrenamiento y 6 días de test, y en la Tabla 10 se muestran los resultados para 1 hora de entrenamiento y 6 días y 23 horas de test. Recordemos que los valores promedio de MAE obtenidos con *baseline* para estos conjuntos de datos era de 3.31 y 5.31 % respectivamente.

Algoritmo	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**	**L8**	**L9**	**L10**	**L11**	**L12**
Perceptron	3.7	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.6	3.6
AMRulesRegressor (optimizado)	**3.1**	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.3	3.4	3.4
ORTO (optimizado)	4.0	3.8	4.0	4.0	**3.8**	4.3	4.2	4.3	4.3	4.3	4.3	4.3
FadingTargetMean (optimizado)	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1	3.1

Tabla 9. resultados de MAE promedio (tanto por ciento) algoritmos de aprendizaje en línea para tamaños de ventana temporal de 1 a 12 unidades, testeando desde el segundo día.

Algoritmo	**L1**	**L2**	**L3**	**L4**	**L5**	**L6**	**L7**	**L8**	**L9**	**L10**	**L11**	**L12**
Perceptron	5.4	5.2	5.1	5.1	5.1	5.2	5.2	5.2	5.2	5.2	5.3	5.3
AMRulesRegressor (optimizado)	**4.8**	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
ORTO (optimizado)	5.6	5.6	5.7	6.0	5.8	6.5	6.5	6.8	6.7	6.6	6.9	7.0
FadingTargetMean (optimizado)	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8	4.8

Tabla 10: resultados de MAE promedio (tanto por ciento) algoritmos de aprendizaje en línea para tamaños de ventana temporal de 1 a 12 unidades, testeando desde la segunda hora.

Como se puede ver en las tablas, con los algoritmos de aprendizaje en línea se obtienen mejores valores que con el *baseline*, y que cualquiera de los algoritmos de aprendizaje en lote. Por completitud, hemos representado los *boxplots*, en la Figura 32 y en la Figura 33. Realizando Z-test con los valores de MAE y para el valor de significatividad estadística que hemos venido utilizando, 5%, las medias siguen sin ser estadísticamente diferentes. Ahora bien, el valor que se obtiene (p-valor 0.1) es lo suficientemente pequeño para que podamos decir que las medias son distintas y que los algoritmos de aprendizaje en línea lo hacen mejor que el *baseline*, para este conjunto de enlaces más variable.

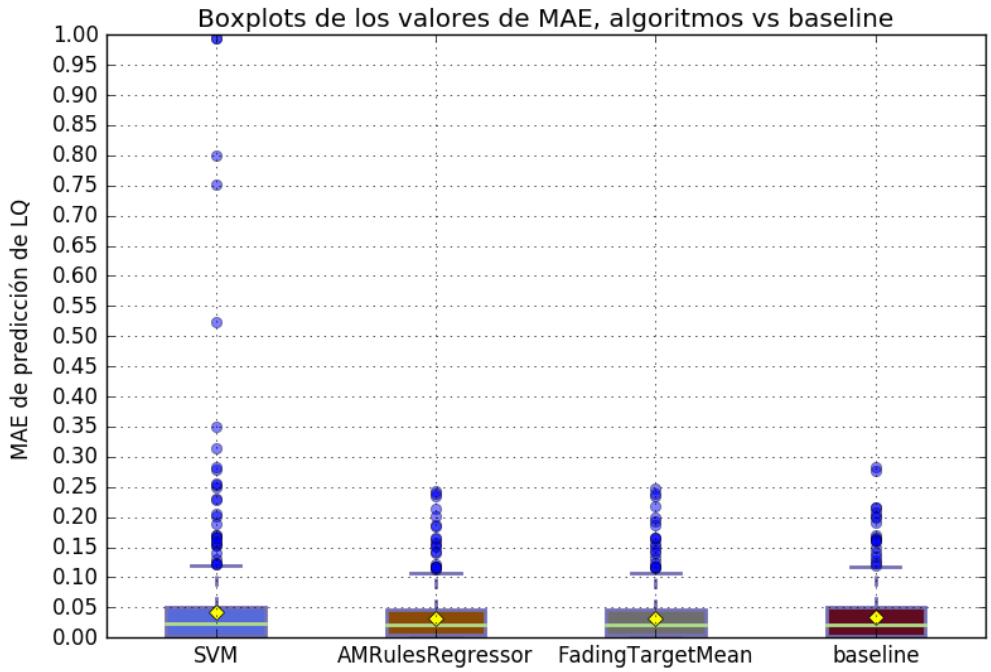


Figura 32: : Boxplots comparando el MAE de la predicción de LQ de los algoritmos SVM, AMRulesRegressor, FadingTargetMean testeando sobre a partir del segundo día; y el baseline.

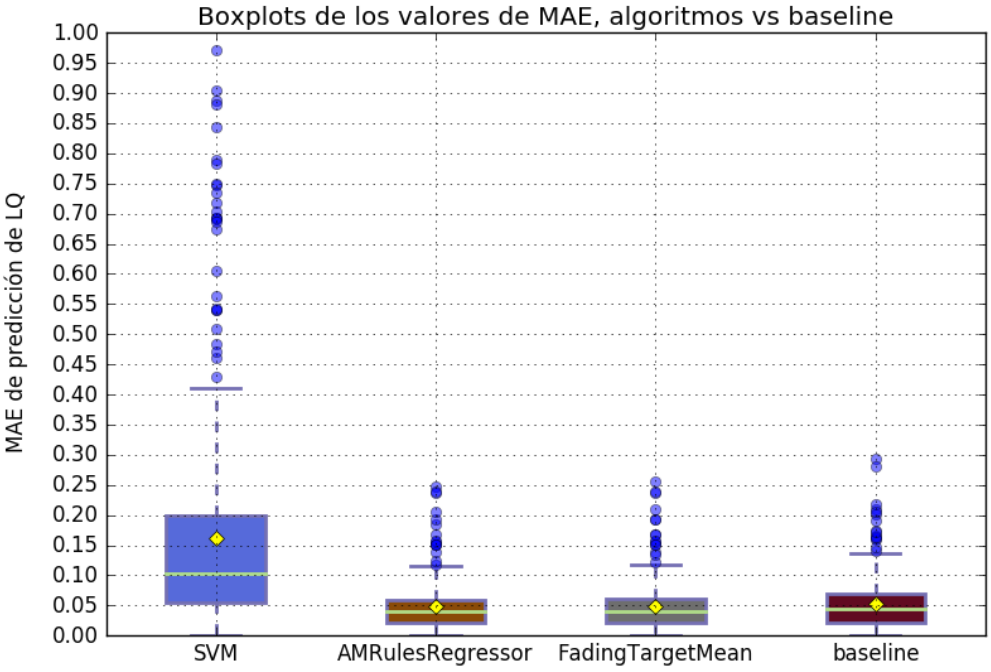


Figura 33: : Boxplots comparando el MAE de la predicción de LQ de los algoritmos SVM, AMRulesRegressor, FadingTargetMean testeando a partir de la segunda hora; y el baseline.

4.6. Algoritmos en lote reentrenados frente a algoritmos de aprendizaje en línea

Finalmente, un aspecto del que hemos hablado a lo largo de este Trabajo Fin de Grado ha sido acerca de la degradación de los algoritmos de aprendizaje en lote. Como hemos visto, cuando el test se realiza en un horizonte más alejado respecto al momento en el que se realizó el entrenamiento, los resultados son peores. Además, si el entrenamiento es menor, peor aún. Para analizar este escenario, se ha creado un subconjunto de enlaces que varíen cada uno de los 6 primeros días, para que los algoritmos de WEKA puedan funcionar, resultando en un conjunto final de 407 enlaces, que como podemos ver, es una gran reducción desde los 610 enlaces que teníamos cuando forzábamos a que hubiera variaciones en el primer día.

Con estos datos, y ventana temporal de 12 unidades, se han realizado experimentos en WEKA. Hemos realizado tres tipos de experimentos. En primer lugar, se han realizado los experimentos que se habían hecho anteriormente, entrenamiento con 6 días y test de 1 día y viceversa. A continuación, reentrenando con los datos del día anterior y testeando con el siguiente día. Y finalmente, reentrenamos con toda la información disponible hasta el momento. Los resultados aparecen en la Tabla 11, en la Tabla 12 y en la Tabla 13, respectivamente.

Algoritmo	**MAE 6 train 1 test**	**Tiempo CPU (s)**	**MAE 1 train 6 test**	**Tiempo CPU (s)**
SVM	3.87%	726.125	4.46%	44.125
RT	4.61%	14.51	5.76%	34.67
kNN	4.57%	28.73	5.58%	57.62
GPR	6.75%	3240.47	7.16%	48.73

Tabla 11: resultados de MAE promedio obtenidos para los algoritmos de WEKA, en 2 casos distintos: en ambos sin timestamp y sobre un conjunto de datos de 407 enlaces, en el primer caso con 6 días de entrenamiento y 1 de test, en el segundo caso 1 día de entrenamiento y 6 de test.

Algoritmo	**Día 2**	**Día 3**	**Día 4**	**Día 5**	**Día 6**	**Día 7**	**Media**	**T total**
SVM	4.53%	4.30%	3.77%	3.95%	4.02%	4.29%	4.14%	113.22
RT	5.52%	5.47%	4.83 %	4.87%	5.13%	5.58%	5.23%	43.86
kNN	5.29%	5.51%	4.69%	4.72%	4.91 %	5.40%	5.09%	66.09
GPR	6.72%	6.97%	6.68%	6.08%	6.25%	6.65%	6.56%	82.19

Tabla 12: resultados de MAE promedio obtenidos para los algoritmos de WEKA, entrenando cada día con el día anterior. Se realiza un test con los datos del día.

Algoritmo	Día 2	Día 3	Día 4	Día 5	Día 6	Día 7	Media	T total
SVM	4.53%	4.21%	3.75%	3.89%	3.96%	3.87%	4.03%	1776.28
RT	5.52%	5.25%	4.63 %	4.67%	4.73%	4.61%	4.90%	57.35
kNN	5.29%	5.14%	4.48%	4.50%	4.62 %	4.57%	4.77%	124.16
GPR	6.72%	6.95%	6.87%	6.70%	6.42%	6.75%	6.73%	5320.14

Tabla 13: resultados de MAE promedio obtenidos para los algoritmos de WEKA, entrenando cada día con todos los días anteriores. Se realiza un test con los datos del día.

Como se puede ver, los resultados reentrenando cada día no son mucho mejores que en el caso anterior de entrenamiento de 1 día. Si bien se aprecian mejoras, son de aproximadamente 0.5% en todos los casos. No obstante, viendo el tiempo estimado de ejecución se puede concluir que esta mejora no es muy costosa, por lo que puede realizarse sin mayor problema cada día. Por otro lado, no se alcanzan los valores obtenidos en el caso anterior de entrenamiento de 6 días.

Respecto al reentrenamiento acumulado, los resultados van mejorando hasta que se llega al valor obtenido en el caso inicial de entrenamiento de 6 días, excepto en el caso de GPR, que se comportaba mejor en el caso anterior; la información pasada le penaliza levemente, aunque no hay que preocuparse mucho ya que es el algoritmo que peores resultados reporta. En cuanto a los tiempos de ejecución, se puede ver que tanto RT como kNN escalan de una manera adecuada, como ya se ha comentado con anterioridad, mientras que SVM y GPR aumentan en gran cantidad su tiempo de ejecución. Este enfoque de reentrenar con todos los datos anteriores puede ser coherente con una situación de inicialización de un sistema, hasta que se tenga suficiente información para solo entrenar con N datos anteriores y no con el total.

A modo de comparativa, se ha elegido el algoritmo AMRulesRegressor, que era de los algoritmos con los que mejores resultados se obtenían anteriormente con ventana de tamaño 1, y el *baseline*. Dado que reentrenar cada día con todo lo anterior es computacionalmente muy costoso, y además es poco escalable, la comparativa se va a realizar con el reentrenamiento con el día anterior, con el mejor algoritmo, SVM. Con AMRulesRegressor se obtiene un MAE promedio del 4% empezando a testear desde el segundo día, menor que cualquier caso de reentrenamiento de los algoritmos en lote. Además, el *baseline* de este conjunto de datos comenzando a testear desde el segundo día es del 4.5%. En la Figura 34, se dibujan los *boxplots* de los resultados obtenidos.

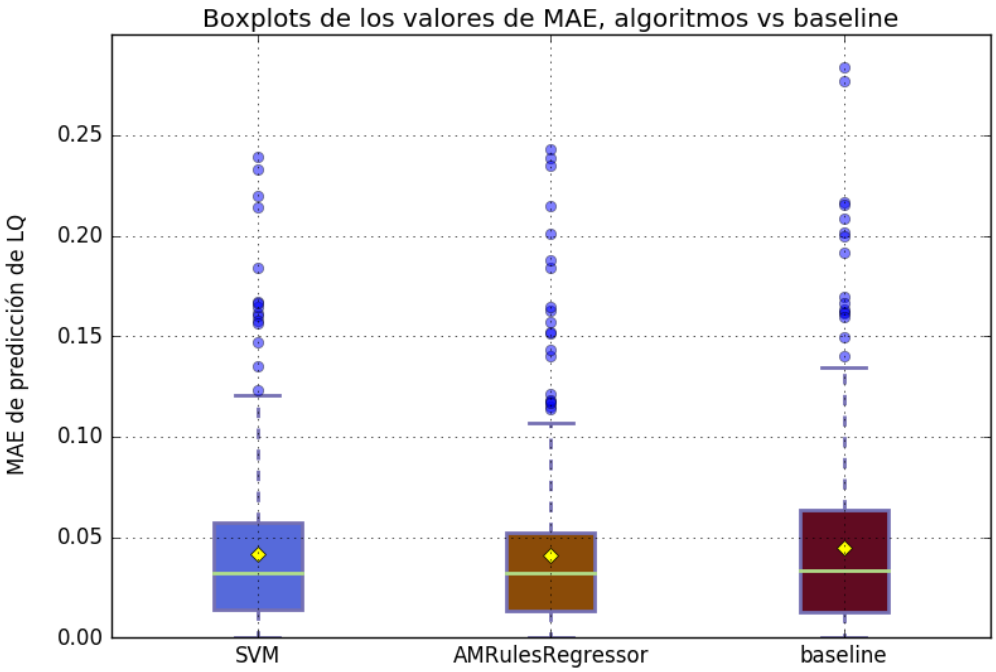


Figura 34: : Boxplots comparando el MAE de la predicción de LQ obtenido con reentrenamiento en el caso de entrenamiento en lote, en los otros dos casos, testeo desde el segundo día.

A la vista de los resultados, queda claro que en este caso los mejores resultados los obtiene el algoritmo en línea. No obstante, al realizar un Z-test, para una significatividad estadística del 5%, siguen sin ser distintas las medias.

4.7. Conclusiones

Con los nuevos experimentos realizados se ha conseguido entender mejor la degradación que se produce en el modelo a medida que pasa el tiempo desde que fue entrenado. Además, se han medido los valores de coste computacional, que han resultado ser extremadamente altos para alguno de los algoritmos de aprendizaje en lote, concretamente, para el algoritmo que mejores resultados producía.

Por otra parte, se ha analizado la influencia del *timestamp*, siendo negativa en la mayoría de los casos, tanto en valores de MAE promedio (diferencia no significativa estadísticamente) como en coste computacional. Además, se ha completado un aspecto que los autores de [7] no cubrían, como es la necesidad de establecer un criterio comparativo justo, es decir, determinar cuando la predicción es mejor que la no predicción. A este respecto, ninguno de los algoritmos previos mostraba ser mejor que el *baseline*, si bien alguno de ellos se mantenía en niveles semejantes. Por ello, se han analizado algoritmos de aprendizaje en línea, que nos han permitido obtener modelos incrementales, que implican una carga computacional baja y permiten mejores resultados que el *baseline*, pese a que no

sean estadísticamente significativos. Además, los experimentos realizados sobre los conjuntos de datos más variables (forzando a que varíen el primer día, la primera hora, o los 6 primeros días) nos han demostrado que, en estos casos, los algoritmos de aprendizaje en línea funcionan mejor que los algoritmos de aprendizaje en lote, y que el *baseline*, pese a que, en este caso, tampoco la diferencia sea estadísticamente significativa.

En la parte restante del documento se extraerán las conclusiones finales de este Trabajo Fin de Grado. También se detallarán las posibles vías de trabajo futuro, centrándonos en la implementación práctica de los modelos de predicción, dado que el fin último del desarrollo de este trabajo es mejorar el encaminamiento en redes comunitarias inalámbricas, es necesario comprobar que los algoritmos propuestos son viables en los dispositivos de estas redes.

Capítulo 5. Conclusiones

El objetivo de este Trabajo Fin de Grado era proponer algoritmos de aprendizaje incrementales, que fuesen capaces de mejorar en primer lugar los resultados obtenidos por [7] y, a su vez, disminuyesen el coste computacional, de manera que pudiesen implementarse en entornos reales.

En primer lugar, se han constatado las limitaciones existentes en el trabajo de [7], tanto las que reportaban los propios autores como otras adicionales. Hemos observado que los modelos propuestos por los autores se degradan con el paso del tiempo, y, además, el coste computacional asociado es muy elevado en el algoritmo con el que mejores resultados obtienen.

Otro punto sobre el que queremos hacer hincapié es sobre la falta de un criterio de admisibilidad de los algoritmos, es decir, un *baseline* que permita determinar cuándo la predicción que estamos obteniendo es mejor que no hacer nada (simplemente recordar el último valor). A este respecto, los resultados obtenidos con este método son bastante buenos, no habiendo diferencia estadísticamente significativa con los algoritmos de aprendizaje automático en lote, poniendo en duda si en esta red en concreto sería necesario el uso de predicciones, ya que en general, los valores de calidad no varían mucho, o al menos, si no sería necesario distinguir unos enlaces de otros, aplicando el *baseline* para enlaces con poca variabilidad y un método más avanzado para los enlaces más variables.

Por otra parte, dada la carga computacional que implican los algoritmos de aprendizaje automático en lote cuando se entrenan durante 6 días, hemos propuesto nuevos conjuntos de entrenamiento de 1 día y 1 hora. Los resultados de estos algoritmos han sido muy negativos, tanto que el *baseline* los superaba con facilidad. En búsqueda de una solución mejor, hemos introducido los algoritmos de aprendizaje en línea, que esperábamos que obtuviesen mejores resultados que los algoritmos en lote, y que además mantuviesen una carga computacional reducida. Lo segundo ha quedado claramente constatado, pero respecto a lo primero, si bien se han conseguido valores de MAE promedio inferiores al *baseline*, no han sido estadísticamente significativos. Esto nos ha hecho plantearnos si no sería más apropiado replicar este experimento sobre una red con mayores variaciones, en el caso de que existan redes de comunidad más variables, en la que los algoritmos de aprendizaje en línea pudiesen demostrar su potencial. Si no, se podría estudiar otro tipo de redes inalámbricas como las redes de sensores. Además, el enfoque seguido en cuanto a la generación de los conjuntos de datos, derivada del artículo de [7], no creemos que sea la más apropiada para el aprendizaje en línea, ya que, en primer

lugar, con este tipo de algoritmos no vamos a tener un conjunto de entrenamiento completo, sino que vamos a ir recibiendo los datos progresivamente. Por otra parte, el asignar a los enlaces apagados el valor 0 puede que esté beneficiando de manera engañosa al método *baseline* en intervalos largos de tiempo dónde un enlace este apagado, y perjudicando a los algoritmos de aprendizaje, al atribuírseles errores cuando los enlaces no están presentes.

Finalmente, y dado que en [7] intuían la posibilidad de reentrenar los algoritmos de aprendizaje en lote, lo hemos hecho. Los resultados son positivos en comparación con los experimentos previos en los que se entrenaba con conjuntos pequeños, pero siguen siendo peores que con el método *baseline*. Podemos destacar que tanto con este conjunto de datos como con el que se utilizaba un entrenamiento de una única hora, hemos obtenido resultados más favorables a favor de los algoritmos de aprendizaje en línea, lo que nos reafirma en la necesidad de obtener datos de redes más variables (como estos conjuntos de enlaces), para poder ver mejor lo que los algoritmos de aprendizaje incremental pueden ofrecernos. No obstante, en este caso reentrenado los algoritmos de aprendizaje en lote se han comportado mejor que cuando no se reentrenaba, por lo que en algunos casos podría contemplarse esta alternativa.

5.1. Trabajo Futuro

A la vista de los resultados obtenidos, son varias las posibles líneas de investigación. En primer lugar, el uso de algoritmos de aprendizaje automático en línea se ha mostrado beneficioso, pero es preciso hacer experimentos en redes (tanto redes de comunidad como otro tipo de redes inalámbricas) con más variabilidad en la calidad de los enlaces, con el fin de analizar si dichos algoritmos son capaces de predecir en estas condiciones, y de esta manera obtener diferencias estadísticamente significativas con el método *baseline*.

Por otra parte, dado que el fin mismo del desarrollo de este Trabajo Fin de Grado es el estudio de algoritmos de aprendizaje automático que permitan predecir la calidad del enlace en dispositivos ligeros de redes de comunidad, un objetivo claro es el testeo de los algoritmos estudiados en dispositivos con menos potencia que el equipamiento utilizado hasta el momento. En este sentido, dado que formar una red de comunidad completa no es viable, podría simularse la entrega de los datos de calidad de enlace, tal y como ocurre en una red de comunidad real. Otra posible alternativa sería crear una red comunitaria en miniatura, aunque puede que los resultados extraídos de ella no fuesen generalizables. Finalmente, podría simularse una red de comunidad completa.

Finalmente, queremos añadir que podría iniciarse otra investigación semejante, pero en el campo de las redes de sensores, en las que la predicción de la calidad del enlace también es importante. A este respecto, quizá fuese más sencilla la implementación de una red de este tipo en vez de una red de comunidad.

Referencias

[1] B. Braem, C. Blondia, C. Barz, H. Rogge, F. Freitag, L. Navarro, J. Bonicioli, S. Papathanosiou, P. Escrich, R. Baig Viñas, A. L. Kaplan, A. Neumann, I. Vilata i Balaguer, B. Tatum y M. Matson, «A case for research with on community networks,» vol. 43, nº 3, pp. 68-73, jul 2013.

[2] L. Maccari y R. Lo Cigno, «A week in the life of three large Wireless Comunnity Networks,» *Ad Hoc Networks,* vol. 24, pp. 175-190, jan 2015.

[3] J. Avonts, B. Braem, Blondia y Chris, «A Questionnaire based Examination of Community Networks,» de *International Workshop on Community Networks and Bottom-up-Broadband*, 2013.

[4] A. Woo, T. Tong y D. Culler, «Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks,» de *International Conference on Embedded Networked Sensor Systems*, 2003.

[5] R. Draves, J. Padhye y B. Zill, «Comparison of Routing Metrics for Static Multi-Hop Wireless Networks,» de *SIGCOMM*, 2004.

[6] K. Farkas, T. Hossmann, F. Legendre, B. Plattner y S. K. Das, «Link quality prediction in mesh networks,» *Computer Communications,* vol. 31, nº 8, pp. 1497-1512, 25 may 2008.

[7] P. Millan, C. Molina, E. Medina, D. Vega, R. Meseguer, B. Braem y C. Blondia, «Time series analysis to predict link quality of Wireless community networks,» *Computer Networks,* vol. 93, nº 2, pp. 342-358, dec 2015.

[8] Y. Zhao, S. Li y J. Hou, «Link Quality Prediction via a Neighborhood-Based Nonnegative Matrix Factorization Model for Wireless Sensor Networks,» *International Jorunal of Distributed Sensor Networks,* vol. 11, nº 10, jan 2015.

[9] T. Liu y A. Cerpa, «Temporal Adaptive Link Quality Prediction with Online Learning,» *ACM Transactions on Sensor Networks (TOSN),* vol. 10, nº 3, apr 2014.

[10] R. Lasser, «Engenieering Method, Electrical and Computer Engineering Design Handbook,» Tufts University School of Engineering, [En línea]. Available: https://sites.tufts.edu/eeseniordesignhandbook/2013/engineering-method/. [Último acceso: 28 apr 2017].

[11] P. A. Frangoudis, G. C. Polyzos y V. P. Kemerlis, «Wireless Community Networks: An Alternative Approach for Nomadic Broadband Network Access,» *IEEE Communications Magazine,* vol. 49, nº 5, pp. 206-2013, may 2011.

[12] J. F. Kurose y K. W. Ross, Computer Networking: A Top-Down Approach, 6th ed., Boston: Pearson, 2013.

[13] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum y L. Viennot, «Optimized Link State Routing Protocol for Ad Hoc Networks,» de *Multi Topic COnference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International*, Lahore, Pakistan, 2001.

[14] T. Lopatic y A. Tonnesen, «olsrd Link Quality Extensions,» 5 oct 2004. [En línea]. Available: http://www.olsr.org/docs/README-Link-Quality.html. [Último acceso: 26 jun 2017].

[15] D. S. De Couto, High-Throughput Routing for Multi-Hop Wireless Networks, 2004.

[16] A. L. Samuel, «Some Studies in Machine Learning Using the Game of Checkers,» *IBM Journal of Research and Development,* vol. 3, nº 3, p. 210, 1959.

[17] T. Mitchell, Machine Learning, McGraw-Hill, 1997.

[18] A. Ng, «Machine Learning. Stanford University,» [En línea]. Available: https://www.coursera.org/learn/machine-learning. [Último acceso: 14 apr 2017].

[19] D. Peteiro- Barral y B. Guijarro-Berdiñas, «A survey of methods for distributed machine learning,» *Progress in Artificial Intelligence,* vol. 2, nº 1, pp. 1-11, mar 2013.

[20] H. Drucker, C. J. Burger, L. Kaufman, A. J. Smola y V. N. Vapnik, «Support Vector Regression Machines,» *Advances in Neural Information Processing Systems (NIPS),* vol. 9, pp. 155-161, 1996.

[21] S. Shevade, S. Keerthi, C. Bhattacharyya y K. Murthy, «Improvements to the SMO Algorithm for SVM Regression,» *IEEE Transaction on Neural Networks,* vol. 11, nº 5, pp. 1188-1193, sep 2000.

[22] D. Aha y D. Kibler, «Intance-based learning algorithms,» *Machine Learning,* vol. 6, nº 1, pp. 37-66, jan 1991.

[23] E. Frank, «Class REPTree,» WEKA, [En línea]. Available: http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/REPTree.html. [Último acceso: 17 jul 2017].

[24] D. J. Mackay, «Introduction to Gaussian Processes,» *Department of Physics, Cambridge University,* 1998.

[25] J. Quilan, «Learning with continuous classes,» de *5th Australian Joint Conference on Artificial Intelligence*, Singapore, 1992.

[26] J. De Winter, «Using the Student's t-test with extremely small sample sizes,» *Practical Assessment, Research & Evaluation,* vol. 18, nº 10, aug 2013.

[27] W. W. Daniel, «Kolmogorov-Smirnov one-sample test,» de *Applied Nonparametric Statistics*, Boston, PWS-Kent, 1990, pp. 319-330.

[28] W. Navidi, Statistics for Engineers and Scientists, 3rd ed., McGraw-Hill, 2011.

[29] G. De Francisci Morales y A. Bifet, «SAMOA: Scalable Advanced Massive Online Analysis,» *Journal of Machine Learning Research,* vol. 16, pp. 149-153, 2015.