UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

SmartBeds

Aplicación de técnicas de minería de datos para la detección de crisis epilépticas y aplicación web

Presentado por José Luis Garrido Labrador
en Universidad de Burgos — 24 de junio de 2019
Tutores: Álvar Arnaiz González y José Francisco Díez Pastor

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

Dr. D. Álvar Arnáiz González, profesor del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. José Luis Garrido Labrador, con DNI 71707244Y, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado «Smart-Beds - Aplicación de técnicas de minería de datos para la detección de crisis epilépticas y aplicación web».

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 24 de junio de 2019

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
Dr. D. Álvar Arnaiz González	Dr. D. José Francisco Díez Pastor

Resumen

La epilepsia es una enfermedad que provoca crisis repentinas con convulsiones violentas y pérdidas del conocimiento. Ante estas situaciones es necesario suministrar unos primeros auxilios, sin embargo, durante las sesiones nocturnas es difícil detectar y tratar a tiempo estas crisis que pueden provocar daños graves a los pacientes que sufren esta enfermedad.

En este trabajo se investiga, a partir de datos reales provenientes de sensores en un colchón, métodos que puedan detectar situaciones de crisis en tiempo real de tal manera que se puedan suministrar las atenciones necesarias con la mayor celeridad posible.

Además de esto se ha creado una API REST y una página web que permite la gestión de camas (adición, borrado, edición), así como mostrar el estado actual de los sensores y las probabilidades de que el paciente esté sufriendo una crisis.

Descriptores

Crisis epilépticas, minería de datos, desequilibrados, monitorización, clasificador, API REST.

Abstract

Epilepsy is a disease whose symptoms are violent seizures and fainting. In that's situations is necessary to apply first aid, but, overnight is difficult to detect and treat in time these crisis. That can cause serious damage to the patients.

In this paper is researched, based on real data from sensors in a mattress, methods which can detect seizures in real time in such a way that caregivers can supply first aids as soon as possible.

In addition, it has been created a REST API and a web that allows the management of beds (creation, modification and elimination) as well as showing the current status of the sensors and the probabilities that the patient is suffering a crisis.

Keywords

Epileptic seizures, data mining, unbalanced, monitoring, clasificator, API REST.

Índice general

Índice de figuras

Índice de tablas

Introducción

Gracias al avance de las técnicas y algoritmos de minería de datos, disciplinas no directamente relacionadas con la computación se han ido beneficiando de las ciencias de datos, a modo particular, la medicina está desarrollándose hacia modelos más preventivos gracias a las predicciones que se pueden generar utilizando estos métodos.

Es por este motivo que podemos ver en la literatura científica de los últimos años como se relaciona medicina y ciencia de datos, por ejemplo en estudios de detección de caídas [33] o la motorización del sueño para la prevención de apneas [18]. En este trabajo de fin de grado, el objetivo es la detección de crisis epilépticas a partir de sensores de presión en un colchón y constantes vitales del paciente. Este TFG está adscrito al proyecto homónimo vencedor del concurso universitario *Desafío Universidad Empresa* [5]

Aunque el análisis y detección de crisis epilépticas de manera automática ha sido ampliamente explorada por la comunidad científica, esta se ha centrado o en el uso de pulseras inteligentes [25] o el uso de encefalogramas (*EEG*) [16, 19, 34] para la predicción de estos eventos. Por tanto, ante pacientes con diversos problemas de salud que impiden el uso de pulseras ya que son fácilmente removibles y de dispositivos de control de actividad cerebral al entorpecer el sueño del paciente (Tabla 1.1), se realiza este pro-

Hardware	**Coste**	**Fiabilidad**	**Observación**
Detector de EEG	Elevado	Elevada	Incómodo para dormir
Pulseras inteligentes	Bajo	Elevada	Los pacientes son propensos a quitársela
Cama inteligente	Elevado	Pendiente de estudio	Es cómodo y no invasivo

Tabla 1.1: Distintos métodos para la detección de crisis epilépticas y sus situaciones según *PRONISA*.

Figura 1.1: Publicidad de la cama de la cual obtenemos los datos.

yecto que enfoca la detección de las crisis epilépticas en el uso de sensores de presión y biométricos en la cama (Fig. 1.1 donde duerme el paciente.

1.1. Material adjunto

Junto a esta memoria se incluyen:

- **Cuaderno de investigación** con la evolución y pasos realizados durante la investigación junto con los resultados de cada experimento.
- **Anexos** donde se incluyen:
  - Plan de proyectos
  - Requisitos del sistema
  - Diseño del sistema
  - Manual para el programador
  - Manual para el usuario

- **API REST** en **Python-Flask**
- **Aplicación web** que funciona sobre la *API*.
- **Experimentos** en *Jupyter Notebook*

Además se puede acceder a través de internet a la página web en producción y al repositorio GitHub del proyecto.

Objetivos del proyecto

Los objetivos del proyecto se han dividido en tres apartados siendo estos los objetivos generales, los técnicos y los personales.

2.1. Objetivos generales

- Estudio del estado del arte en detección de crisis epilépticas, tanto en materia de *hardware*, datos utilizados como en técnicas y modelos existentes desarrollados por otros científicos. Con esto se busca explorar técnicas no antes utilizadas como también optimizar esfuerzos por los métodos que ya han probado su utilidad.
- Exploración e interpretación de los datos, las formas por las cuales se pueden representar y como se distribuyen los distintos ejemplos que representan crisis respecto a las situaciones normales. Aplicación de filtros, análisis estadísticos, proyecciones de n-variedad (*manifold learning*), etc.
- Exploración de técnicas de balanceo de datos existentes y aprendizaje automático para conjuntos de datos desequilibrados. Buscar la mejor combinación de técnicas de balanceo y modelos de aprendizaje automático para optimizar la detección de situaciones de crisis.
- Búsqueda de un modelo que haga una clasificación lo más correcta posible centrando que la predicción sea acertada en situaciones de crisis mediante la optimización del valor del ratio de verdaderos positivos.
- Creación de una *API REST* con la que poder distribuir de los datos de las camas y sus predicciones.

- Implementar una interfaz web por la cual se puedan ver datos en tiempo real de las camas y sus predicciones.

2.2. Objetivos técnicos

- Hacer uso de las herramientas de minería de datos de *scikit-learn* y *Weka*.
- Crear una serie de transformadores de datos de *scikit-learn* para facilitar el preprocesado.
- Desarrollar una *API REST* sencilla y fácil de utilizar.
- Crear una interfaz web que permita ver los datos en tiempo real de los pacientes con una baja latencia.

2.3. Objetivos personales

- Contribuir a la mejora de la calidad de vida de pacientes que sufren de epilepsia.
- Profundizar en el trabajo de investigador, sus metodologías y las fases de por las que pasa un proyecto de investigación.
- Comprender más técnicas de minería de datos, nuevos modelos y nuevas formas de abordar el análisis de datos.
- Completar mi formación académica con el desarrollo de una aplicación que englobe la mayor cantidad del conocimiento adquirido en el estudio del Grado.

Conceptos teóricos

En este capítulo se explicarán superficialmente los conceptos por los cuales se han desarrollado este proyecto, una explicación más completa de los modelos que se han usado se encuentran en la sección 5.1.

Definiciones

Crisis epiléptica [21]: se trata de un evento imprevisto de corta duración que comienza y termina de forma súbita. Se originan en el cerebro y pueden provocar convulsiones, rigidez, desvanecimiento o espasmos musculares según el foco de origen del mismo. Si la duración de la crisis fuese de más de cinco minutos se considera *status epilepticus* (puede provocar daños neuronales y es improbable que paren por si solas) y se necesita atención médica inmediata para mitigar los efectos.

Preprocesado [27]: proceso por el cual se realizan operaciones sobre los datos con el fin de facilitar la interpretación de los mismos. Engloba procesos como la eliminación de instancias ruidosas, suavizado de señales mediante filtros, normalización de las características, eliminación de aquellas que poseen una baja variabilidad, el análisis estadístico y la proyección de los datos en espacios de menor dimensionalidad para su visualización, etc.

Análisis de componentes principales [43]: se trata de una técnica estadística que se utiliza para crear una descripción del conjunto de datos utilizando unas variables no correlacionadas. El objetivo es encontrar los ejes de máxima varianza y realizar proyecciones de menor dimensionalidad de los datos facilitando la interpretación de los datos.

Proyección a 2-variedad (*Manifold learning*): se define como variedad a un objeto geométrico que representa un espacio que se parece localmente,

la idea intuitiva sobre este concepto es el de un mapa, que proyecta en dos dimensiones un objeto que existe en tres dimensiones [38]. Por tanto, una proyección a 2-variedad es el proceso por el cual podemos disminuir la dimensionalidad de un conjunto de datos a dos dimensiones y poder estudiarlo mejor [23]. De la misma manera que no existe una única forma de transformar una esfera a un plano, tampoco existe una única forma de proyectar el conjunto de dimensiones a dos dimensiones, esto se explora en el capítulo 5 del Apéndice *Cuaderno de investigación* adjuntado.

Clasificador: se trata de un algoritmo de aprendizaje supervisado que tiene de entrada los datos de una instancia y obtiene de salida la predicción de la clase a la que pertenece.

Detección de anomalías [42]: también conocido como clasificación de clase única (*one-class clasification*), es un tipo de clasificador que se centra en acotar el espacio en el cual las instancias de una única clase existen de tal manera que detecte como anomalía cualquier instancia que no esté dentro de ese espacio.

Ensemble [26]: consiste en un método que se basa en la premisa que un conjunto de clasificadores débiles al combinarse genera un clasificador fuerte. Bajo esta definición se pueden crear diversas técnicas que permitan crear estos clasificadores. Algunas de estas técnicas son: crear subconjuntos aleatorios de las instancias para entrenar el mismo clasificador con diversos datos (*bagging*); entrenar a los clasificadores con pesos diferentes para las instancias o remuestreando los datos según este criterio (*boosting*); usar diferentes semillas para los clasificadores (comité aleatorio); creación de conjuntos de árboles de decisión según diferentes criterios (bosques).

Conjunto de datos desequilibrados: también conocidos como desbalanceados, trata de un conjunto de datos en el cual las clases no están igualmente representadas existiendo una gran desproporción. Esta diferencia puede ser, como ejemplo, con proporciones de un 1 % para la clase minoritaria y un 99 % para la mayoritaria. Esto provoca en clasificadores normales que las instancias de la clase minoritaria sean ignoradas ya que un de predecir siempre las instancias como clase mayoritaria el acierto sería muy alto (en el ejemplo anterior tendríamos una precisión del 99 %).

Balanceo de datos [7, 8, 12]: este es el proceso por el cual se pretende equilibrar la proporción entre las clases de un conjunto de datos desequilibrados. Estos métodos pueden ser de sobremuestreo creando nuevas instancias de la clase minoritaria a partir de los datos existentes, submuestreo eliminando instancias de la clase mayoritaria o el muestreo aleatorio creando un *ensemble* utilizando diversos conjuntos balanceados de manera aleatoria.

Técnicas y herramientas

4.1. Investigación

El proyecto ha sido desarrollado aplicando diversas técnicas de minería de datos siguiendo el proceso KDD [27] (Fig. 4.2). Aunque en esta memoria se explicará superficialmente, el desarrollo completo de la investigación se encuentra en el cuaderno de investigación adjuntado.

Para el proceso del clasificador final el flujo KDD ha sido el siguiente:

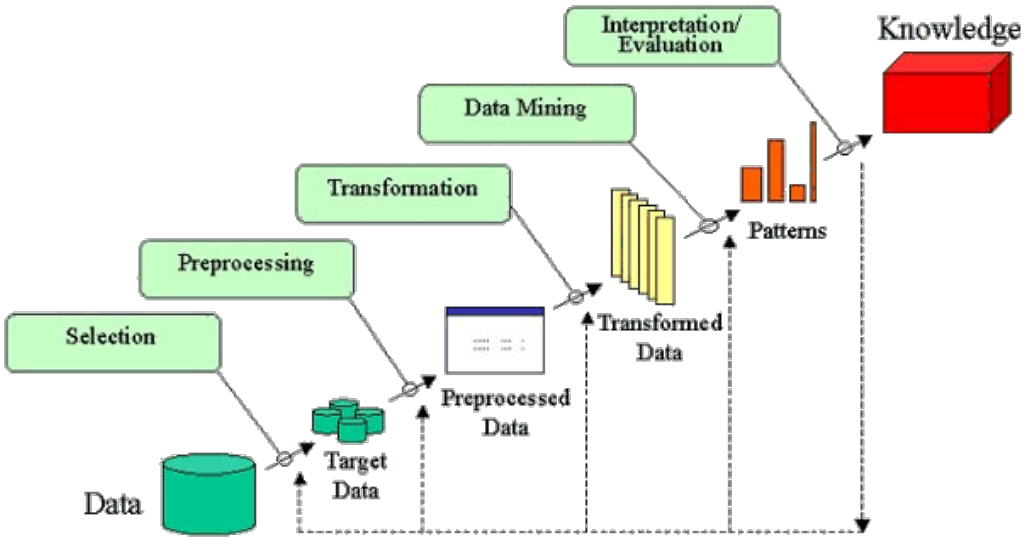

Figura 4.2: Flujo KDD [10].

1. **Selección** Los datos han sido cedidos por la asociación abulense *PRONISA* que contiene jornadas nocturnas con identificadores de la cama, datos de presiones y datos vitales. Estos datos no están balanceados ya que existe una mayor cantidad de datos del paciente durmiendo que bajo una crisis epiléptica.

2. **Preprocesado** Los datos se han limpiado reduciendo ruidos mediante filtros de señal.

3. **Transformación** Se han generado datos estadísticos para series temporales que optimizaban el valor del área bajo la curva PCR [31] desde los datos preprocesados.

4. **Minería de datos** Se ha aplicado un sistema de clasificación doble, en primer lugar se utiliza un árbol de clasificación muy simple que divide entre las situaciones de despierto (bajas presiones en la cama) y acostado, en esta situación se aplica un clasificador *Random Forest* [4] para determinar si hay crisis o no.

5. **Interpretación** Se ha desplegado una aplicación web con datos en tiempo real para evaluar de manera constante la situación actual del paciente.

Este proceso de investigación se realizó sobre *Python* utilizando el ecosistema de bibliotecas *SciPy* [17], la biblioteca homónima se uso para crear filtros sobre los datos.

scikit-learn [**23**] biblioteca con funciones de aprendizaje automático para la creación de nuestros modelos.

NumPy [**22**] biblioteca de computación especialmente útil para el almacenamiento de los datos y sus operaciones.

Pandas [**20**] biblioteca de análisis de datos que usamos para almacenar los datos cargados desde los ficheros *csv*.

MatPlotLib [**15**] herramienta de dibujado usada para representar los datos y sus proyecciones.

Además, en algunas partes de la investigación se utilizó *Weka* [14] para estudiar métodos que no se encontraban en *scikit-learn* como el *Random Balance* [7] o *Rotation Forest* [28].

4.2. Servicio web

Backend

Las herramientas utilizadas para programar el servidor han sido las siguientes:

Flask [30] *Microframwork* de código abierto (BSD) que ofrece una capa de abstracción muy alta de un servicio web, se utiliza para la creación de la lógica de negocio mediante la gestión de las rutas.

Jinja [29] Gestor de plantillas de código abierto (BSD) para Python, se utiliza para la creación de la interfaz web mediante la creación de páginas dinámicas en HTML.

Flask-SocketIO [13] Integración del servicio de *sockets*, *Socket.IO* [2], compatible con los *WebSockets*, se utiliza para la difusión de los datos en pacientes.

Gevent y Eventlet [3, 24] Bibliotecas para el uso de tiempo real, asíncrono de hilos para el uso de *Socket.IO*, el uso de estas bibliotecas es facilitar el procesado y difusión en tiempo real de los datos de los pacientes.

Para la programación del sistema de hilos que distribuyen datos en tiempo real se siguieron los paradigmas de la programación orientada a objetos y de programación funcional. El sistema de rutas siguió las guías del *microframework Flask*.

Algunos patrones de diseño utilizados han sido el *Singleton* para la API así como un *Proxy* entre la interfaz web y la lógica de el API.

Frontend

Para el desarrollo de la parte visible de la aplicación se han usado otra serie de herramientas:

Bootstrap [36] *Framework* de *CSS* de código abierto (MIT) creado por *Twitter* para la creación de aplicaciones web redimensionables. Todos los estilos de la web se apoyan en este *framework*.

jQuery [37] *Framework* de *JavaScript* de código abierto (MIT) que simplifica el acceso al *HTML DOM* de la página. La gestión de los eventos de la web se gestionan mediante este *framework*.

Chart.js Biblioteca de *JavaScript* de código abierto (MIT) para la creación de grafos en `canvas`. Se usa para la visualización de las gráficas de presiones y constantes vitales.

4.3. Herramientas generales

Para el desarrollo general del proyecto se han utilizado las siguientes herramientas según el ámbito al que pertenecen:

Servidor

Nginx servidor web y de proxy reverso ligero de alto rendimiento [41] de código abierto (BSD simplificada).

MariaDB sistema de gestión de bases de datos derivado de *MySQL* [39] de código abierto (GPLv2). Este motor es extremadamente compatible con *MySQL* porque es creado como una bifuración de esto para garantizar la existencia de este motor bajo GPL.

Proxmox es un entorno de virtualización de servidores [44] de código abierto (AGPL). Su función principal es el despliegue y gestión de máquinas virtuales y contenedores.

Anarchy Arch sistema GNU/Linux derivado de *ArchLinux* [35] sobre el cual se ejecuta todo el servidor, está alojado en una máquina virtual del entorno *Proxmox*.

Miscelánea

- **Jupyter Notebooks**: IDE de programación de *Python* basado en *iPython* de código abierto (BSD).
- **PyCharm Professional**: IDE de programación para *Python* avanzado basado en *IntelliJ*.
- **Visual Studio Code**: Editor de código genérico de código abierto (MIT).

- **Postman**: IDE para la ejecución de request *HTTP*.
- **Selenium**: IDE de pruebas sobre web de código abierto (APACHE).
- **CertBot**: Sistema para la firma *SSL* sobre *HTTP* gratuito de *LetsEncrypt* de código abierto (MPL).
- **Overleaf**: Editor de LaTeX online para el trabajo colaborativo.
- **TeXStudio**: Editor de LaTeX de código abierto (GPLv2).
- **Dia**: Editor de diagramas genérico de código abierto (GPL).
- **StartUML**: Editor de diagramas UML.
- **Codesketch DB**: Traductor bidireccional de código-diagrama para bases de datos.
- **Filezilla**: Aplicación para la trasferencia de ficheros sobre *FTP* y *SFTP* de código abierto (GPLv2).
- **GitHub**: Servicio online de *hosting* para repositorios Git.
- **ZenHub**: Servicio online de integración de herramientas *SCRUM* sobre GitHub.

Aspectos relevantes del desarrollo del proyecto

5.1. Introducción

En este capítulo se explicarán los aspectos más importantes tanto de la investigación como del desarrollo. Los detalles más específicos y los resultados obtenidos se encuentran en el *Cuaderno de investigación* y los *Anexos*.

5.2. Investigación

La fase de investigación ha ocupado la mayor cantidad de tiempo de este proyecto ya que ha sido también una introducción a la carrera investigadora. Ha sido desarrollada conjuntamente con Alicia Olivares Gil bajo la dirección de nuestros tutores el Dr. Álvar Arnáiz González y el Dr. José Francisco Díez Pastor.

El comienzo de esta investigación comenzó con un estudio del arte con el fin de conocer métodos y soluciones ya existen y se han probado. Sin embargo, la mayoría de artículos que existen actualmente para ese problema se centran en datos de encefalogramas, un tipo de datos que no poseemos. También exploramos problemas semejantes como detecciones de caídas mediante sensores de presión aunque la mayoría de estos sistemas se basaban en otros sensores como cámaras o acelerómetros. Algunos de estos artículos están resumidos en la sección 6.1.

Tras esta exploración bibliográfica nos centramos en el estudio de los datos. Primero haciendo una limpieza eliminando datos con poca variabilidad, normalizarlos y eliminar datos ruidosos. Tras esto generamos proyecciones

de los mismos a dos dimensiones tanto con datos limpiados, estadísticos o filtrados.

Los datos estadísticos que generamos en un comienzo fueron medias y desviaciones móviles de ventana 25. Los filtros tratados fueron el *butterworth* y el *savgol*, ambos de la librería *SciPy* [17].

Las primeras conclusiones que obtuvimos era que los datos intermedios de las crisis documentadas tenían ciertas propiedades que las diferenciaba del resto, sin embargo esto se perdía aumentando la ventana del estudio. Además que este proceso era muy lento y necesitaba una gran cantidad de datos por lo cual realmente podíamos obtener si hubo una crisis y no hacer una detección en directo.

A partir de aquí, para facilitar los procesos de preprocesado se crearon un conjunto de transformadores de datos que incluyen un normalizador por característica como por grupos de las mismas, un filtro de ruido, un eliminador de características de poca varianza, un calculador de estadísticos móviles, un modificador del valor objetivo según una ventana temporal y dos compuestos que concatenan resultados o los convierte en una línea de tuberías de tal forma que se ejecuten secuencialmente.

Antes de pasar a la exploración de métodos para clasificar, acotamos las duraciones de las crisis que nos habían reportado, la primera razón es que la misma empresa nos informó de que los datos eran estimados y estos superaban el umbral para *status epilepticus* [21]. Esta acotación se hizo analizando manualmente los datos dentro del rango que nos dieron determinando como crisis las situaciones con presiones anómalas.

Detección de anomalías

Lo siguiente que se realizó fue un estudio de detección de anomalías en el cual se intentó tanto predecir una crisis a partir de entrenar un clasificador `OneClassSVM` entrenando con todos los datos de no crisis y también predecir una situación de no crisis entrenando el mismo clasificador con los datos de crisis.

Lo primero que obtuvimos es que una crisis es un subconjunto de una situación no crisis ya que el clasificador no lo detecta como anomalía al ser entrenado con las situaciones de no crisis.

El segundo obtenido fue que las diversas crisis no coexisten en el mismo espacio, ya que al entrenar el clasificador con una crisis determinaba como anomalía a todas las demás crisis al igual que al resto de datos. Además, en

el caso de combinar dos situaciones de crisis el entrenamiento ya tiene un gran índice de fallo si los datos no son estadísticos (media y desviación).

Ensembles para desequilibrados

Debido a que la detección de anomalías no dio resultados correctos se pasó a hacer un estudio de clasificadores combinados (*ensemble*) optimizados para la resolución de problemas con datos desequilibrados [8, 12].

Debido a que el conjunto de datos ya habíamos hecho un submuestreo al limitar el conjunto de datos a los del día de las crisis probamos a hacer un sobremuestreo mediante SMOTE [12] con los algoritmos de *Bagging*, *AdaBoostM1* y *Rotation Forest* [28].

Los resultados de este experimento fue que los clasificadores sobreajustaban los datos de las crisis y no eran capaces de predecir correctamente otras crisis. Además pudimos comprobar que el algoritmo de *boosting* no predecía correctamente los datos de entrenamiento lo que, junto con el hecho de que la etiquetación sabíamos que había sido manual, da pie a la conclusión de que los datos están mal etiquetados [26].

También se probó a realizar una validación cruzada con todo el conjunto de datos usando el clasificador de *Rotation Forest* [28] obteniendo unos buenos resultados.

Búsqueda de mejores características

Otro proceso que se hizo fue la búsqueda de mejores características a partir de un análisis estadístico tratando a los datos como una serie temporal. A partir primero de la búsqueda de la mejor ventana temporal (obtenido el valor de noventa) mediante la optimización del valor del AUC [12] y de PRC [6].

Ensembles para desequilibrados - estadísticos de series temporales

Para finalizar la investigación se volvieron a probar los *ensembles* para conjuntos desequilibrados de Galar et. al. [12] y Díez et. al. [7, 8] usando sobremuestreo con *SMOTE*, submuestreo con *RUS* y muestreo aleatorio con *Random Balance*. Se usaron los clasificadores de comité aleatorio, *bagging* y *rotation forest*.

Los resultados fueron semejantes a la primera ocasión ya que el testeo con la otra crisis siempre obtenía un ratio de aciertos con las crisis de 0.

5.3. Desarrollo de la aplicación

La fase de desarrollo ha ocupado alrededor de un cuarto del tiempo del proyecto. A su vez se ha dividido en cuatro partes: una primera de búsqueda de herramientas para la difusión en tiempo real de datos, una segunda de diseño, una tercera de implementación y finalmente el despliegue.

La primera parte fue la más corta, consistió en explorar herramientas que fueran compatibles con websocket [46] ya que este protocolo es el más eficaz para distribución en tiempo real de datos mediante HTTP, que es la tecnología más sencilla para la creación de una *API*. La herramienta seleccionada fue *socketIO* [2] que daba una interfaz sencilla y multilenguaje.

Diseño

La fase de diseño fue la que más tiempo llevó de este apartado ya que hubo que discutir muchos detalles con mi compañera Alicia Olivares, ya que se debía crear una interfaz sencilla y funcional para que pudiese desarrollar una aplicación Android lo más rápidamente posible.

El primer paso fue determinar los casos de uso, que debido a la limitación temporal, se resumieron lo más posible. Decidimos hacer una gestión de usuarios y camas con un sistema de permisos simplificado de forma tal que los usuarios pueden estar asociados a camas. También se creó un usuario maestro administrador que tuviese permisos sobre todo.

Otro punto importante fueron los datos. Los usuarios son formados por un identificador único, un nombre de usuario único, una contraseña y un token dinámico y único. Este último es el que permite identificar al usuario para realizar operaciones *CRUD*. Este cambia por cada inicio de sesión del usuario de tal forma que solo puede existir una sesión abierta a la vez.

Las camas, por otro, lado están formadas por un par de identificadores de los sensores de presión (*MAC*) y de los sensores biométricos (*UUID*) que son usados también como el nombre del espacio donde se difundirán los datos de esa cama. También identifican a la cama un número secuencial y un nombre de la misma. Cada cama tiene a su vez una combinación de IP-puerto del rango de *multicast* que especifica a la aplicación a donde tiene que escuchar.

Como parte final del diseño se determinó cuales serían los pasos para el *handshake* del proceso de obtención de datos en tiempo real. Se decidió que este sería una petición a la API para solicitar el espacio de nombres, luego una apertura de conexión a *socketIO*, si la conexión era correcta lanzar una petición de solicitud de datos (`give_me_data`) para finalmente escuchar paquetes (`package`) al espacio de nombres solicitado.

Implementación

El último proceso fue la implementación, que se dividió en cuatro partes: primero la creación de la *API REST* para la gestión de camas y de usuarios, la segunda fue la implementación de la difusión de datos en tiempo real, la tercera la creación de ventanas para la interacción vía web y por último la creación de test unitarios y de integración (interfaz).

Implementación de la *API*

Esta *API* se implementó utilizando dos partes, una primera como una clase *singleton* que sirve de conexión a la base de datos y realiza las operaciones especificadas controlando los permisos y una segunda parte que hace de adaptador para traducir peticiones *HTTP* a la signatura de las funciones de la clase.

El funcionamiento global de una petición es:

1. Se hace una petición `POST` mediante `x-www-form-urlencoded` a la ruta de la operación deseada.
2. La función asociada a la ruta desempaqueta el mensaje y crea una llamada al objeto de la *API*.
3. La *API* realiza la operación solicitada y devuelve el resultado deseado.
4. La función de adaptación recoge el resultado y crea un objeto de respuesta *JSON* con un código *HTTP* y un mensaje.

En el caso de que alguno de los procesos fallasen, se generaría un código de error según el tipo de error. Si fuese 400 implicaría que la petición realizada no tiene los parámetros esperados, 401 si el valor del *token* no está correctamente asociado, 403 si no se tuviesen permisos para realizar la operación, 404 si no existiese la operación deseada, 405 si se hace con un método erróneo, 418 si la operación no estuviese disponible por causa de

la versión y 500 en el caso de que existiese un error en el servidor. Si todo fuese correcto el número del código sería 200. Todos estos valores provienen del estándar RFC 7231 [11].

Difusión en tiempo real

Para poder hacer la difusión en tiempo real se necesita de la librería de `flask-socketio` [13] y de `gevent` [24].

El primer paso es crear los hilos del *pipeline* del procesamiento de los datos (uno de escucha y otro de procesado). Esto se hace usando la interfaz `Thread` pero compatibilizado con `gevent` mediante un *mokey patching* (explicado en la sección 5.5).

Tras esto es necesario crear las capturas de eventos de *socketIO* para poder detectar cuando se solicitan datos y enviarlos. Este envío se hace mediante otro hilo, el de difusión que emite por *broadcast* a todos los clientes de cada cama. Estos hilos funcionan solo dentro del entorno de petición (*request context*) por lo que morían al cerrar el usuario la conexión, por tanto, para evitar que los datos se acumulasen en el servidor se usaron colas en los hilos de procesamiento con un tamaño máximo a la ventana temporal (90) de tal forma que se borrasen los más viejos si entraban nuevos datos.

Un detalle importante es que como los datos no podemos tenerlos en tiempo real, ya que pertenecen a la empresa que nos los suministra. Por ello hemos simulado un *streaming* de datos local usando UDP emitiendo una nueva linea del CSV cada 0,4 segundos, que es una estimación del intervalo por el cual nos llegan los datos.

Otra modificación sobre el diseño ha sido que realmente existe una única cama y las distintas camas que existen realmente usan los mismos hilos ya que los *Broadcaster* realmente emiten a todos los *namespaces* que existen. Se ha hecho esto debido a las limitaciones técnicas del equipo de despliegue ya que cada cama implicaba una serie de tres hilos que ralentizan mucho el sistema.

También se utiliza actualmente un clasificador aleatorio por intervalos de tal manera que cambie la probabilidad de crisis cada noventa instancias entre los tres estados posibles: crisis, no crisis, despierto; con el fin de poder probar como cambian las interfaces en los distintos estados, sobre todo porque el clasificador obtenido al final no era capaz de predecir situaciones de crisis.

Ventanas *HTML*

La interfaz web de la aplicación se hizo mediante plantillas *Jinja* [29] de *HTML*. Se apoyó con el *framework* de *boostrap 4* [36] para el *CSS* y *jQuery* para la implementación de funciones *JavaScript*.

Para facilitar este desarrollo se crearon varios componentes separados y una plantilla maestra de la que heredarían todas las páginas. Esto evita el código repetido así como facilita el desarrollo de la aplicación. Los componentes que se crearon fueron la barra de navegación, el pie de página, las funciones generales de para la validación de los formularios y un modal para mostrar los errores.

Otro aspecto importante del desarrollo de la interfaz fue la creación de los gráficos en tiempo real. Se usó la librería *Graph.js* para hacer gráficos en objetos `canvas` de *HTML*. En un primer momento la librería actualizaba muy lentamente, pero el problema estaba en la programación de los hilos de *Python* con los hilos, esto se solucionó con la llamada `eventlet.sleep(0)` en el `Broadcaster` para que los datos fuesen en tiempo real.

Para optimizar el renderizado es estas gráficas se optó por eliminar las animaciones y los valores de `hover`[1] para que no se ejecutasen nuevas llamadas a eventos del `DOM` que mostraban los datos de esa instancia temporal.

Otra tecnología que se usó en el desarrollo de la interfaz web fue el uso de llamadas asíncronas al servidor mediante *AJAX*. De esta manera se pueden evitar cargas completas de la página de manera innecesaria. Especialmente en las situaciones que la operación no requería mostrar nuevos datos en la pantalla como la creación, modificación o borrado de usuarios y camas.

Pruebas

Las pruebas consistieron en pruebas unitarias y de integración. Para las primeras se usó `unnittest` de *Python* y para las de integración se hicieron test sobre la interfaz web con *Selenium IDE*. Este por motivos de tiempo no se ha trasladado a código y se puede ejecutar importando el fichero .side al *IDE* y ejecutarlo desde el mismo. En el Anexo *Manual del Programador* este punto está más explicado.

Los test unitarios se basaron en un test parametrizado y un fichero *JSON* con la configuración de todos los test deseados.

[1] Evento al estar el puntero del ratón sobre el objeto

```
{
    "tests": [
        {
            "name": "nombre del test",
            "description": "descripcion del test",
            "func": "funcion de la clase API",
            "input": ["parametros de entrada en orden"
                ],
            "output": ["salida esperada"],
            "excepts": "excepciones esperadas, si es
                null no se espera ninguna"
        },
        ...
    ],
    "tokens":
        {
            "usuario": "token",
            ...
        }
}
```

Cada test tiene seis campos, un nombre y descripción para identificar al test, una función de la clase `smartbeds.api.API`, una entrada en el orden de los parámetros, la salida esperada (si espera una) y la excepción que espera (si espera una). También tiene una serie de tokens asociados a los usuarios, aunque ese campo es solamente útil para el programador de los test para saber que token está asociado a cada usuario.

Antes de ejecutar los tests se instala una base de datos de pruebas que borra la base de datos anterior.

Despliegue

El despliegue de la aplicación fue una de las partes más complicadas del proceso de desarrollo. En primer lugar se intentó desplegar sobre un dispositivo embebido, en particular la *Raspberry Pi 2B*. Sin embargo, este *hardware* no fue capaz de soportar el rendimiento necesario para enviar los datos en un tiempo real.

Por eso se traspasó el sistema a otro servidor personal[2] con varias máquinas virtuales llamado *Al-Juarismi*. Se incorporó la base de datos a una de ellas de nombre *V-Lovelace* y la lógica de la aplicación se almacenó en la máquina *V-Babbage*. En esta última también se añadió la simulación de la cama como un demonio semejante al que se puede ver en el anexo del *Manual del programador*.

Este servidor está configurado de tal forma que se requiera un certificado *SSL* sobre todas las páginas que albergue. Por tanto se usó un certificado de *LetsEncrypt* para el mismo.

5.4. Test de usabilidad

TODO: Pendiente de realizar

5.5. *Mokey Patching*

También conocido como el parches de mono en español, es la técnica por la cual se extiende o modifica la funcionalidad del sistema de manera local y en ejecución. Solo se permite en lenguajes dinámicos como *Python* o *Ruby* [40].

Bajo esta definición las librerías *Gevent* [24] y *Eventlet* [3] aportar una función de parcheo para cambiar las funciones de las librerías nativas de *Python*: `sockets`, `dns`, `time`, `select`, `thread`, `os`, `ssl`, `httplib`, `subprocess`, `sys`, `aggresive`, `Event`, `builtins`, `signal` y `queue`.

Gracias a esto todas esas funciones y clases que no son compatibles con el sistema de eventos de *socketIO* se pueden utilizar sin necesidad de aprender todas las funciones y clases alternativas de *gevent* o *eventlet*.

[2]Este servidor forma parte del grupo *JKA Network*

Trabajos relacionados

En la introducción se han presentado diversos trabajos que sobre la detección de crisis epilépticas, a continuación se presentarán con más detalles el estado del arte en este campo.

6.1. Artículos científicos

Epileptic seizure detection [32]

Este artículo de 2007 se centra en la detección de epilepsia y sus momentos previos mediante el uso de los datos de encefalograma (EEG) descompuesto con la transformada ondícula [45] con redes *RBF*. Exploran diversos conjuntos de características combinadas con diversas cantidades de neuronas obteniendo resultados de un 89 % de precisión media tanto en la detección de situaciones de crisis como de los momentos previos a una crisis.

Los datos usados han sido provenientes de ratas, en concreto dos sanas y cinco epilépticas de las cuales obtuvieron 2 356 segmentos de crisis.

Automated Epileptic Seizure Detection Methods: A Review Study [34]

Este estudio de revisión del 2012 hace una compartativa de 45 artículos de métodos de detección de crisis epilépticas, siendo un punto muy importante para comenzar una exploración ya que incluye una gran bibliografía revisada. Contiene el proceso de extracción de características, todos probados con un conjunto de datos de encefalogramas, también documenta las precisiones obtenidas.

Direction Sensitive Fall Detection Using a Triaxial Accelerometer and aBarometric Pressure Sensor [33]

Otro artículo explorado es este de 2011 centrado en la detección de caídas debido a que se centra en entradas mediante presiones y acelerómetros. El objetivo de la investigación está tanto en la detección de la caída como en la dirección de la misma.

El principal reto en este trabajo estaba en discriminar correctamente las situaciones que eran una caida de las que no debido a que no es lo mismo tumbarse en la cama, que caerse de la misma.

El proceso que sigue es extraer características de los datos del acelerómetro obteniendo la amplitud y el ángulo del mismo, hacer dos procesos sobre estos como un árbol de decisión de tal manera que de concluir posible caída se para a un sensor de presión que determina si es una caída o no lo es.

La precisión de los resultados ronda el 85 % al incluir los sensores de presión mejorando en 4 puntos al solo utilizar el acelerómetro.

Seizure detection, seizure prediction, and closed-loop warning systemsin epilepsy [25]

Este artículo se centra en una revisión de productos comerciales para la detección de crisis epilépticas así como diversos métodos para esta detección.

Presenta un rango de dieciséis años de estudios con múltiples modelos y datos. La mayoría están centrado en datos de encefalogramas y máquinas de vectores soporte. También presenta la variedad de productos del mercado aunque en la mayoría de los casos no existe un artículos científicos sobre los mismos. La mayoría de estos dispositivos son *wereables*[3] como pulseras, si que presenta algunos sistemas basados en camas.

[3]Dispositivo electrónico que se utiliza como una prenda.

Conclusiones y Líneas de trabajo futuras

Para finalizar la exposición de esta memoria se comentará en este capítulo las conclusiones del trabajo así como líneas futuras que se pueden realizar.

7.1. Conclusiones

De este proyecto se pueden obtener las siguientes conclusiones.

- No se ha podido crear un modelo útil para la monitorización de pacientes con epilepsia por diversos motivos. El primero es la escasez de datos para el estudio ya que de todas las crisis documentadas solo se tenían datos documentados de tres. Segundo, la etiquetación deficiente de las mismas ya que solo se poseía un rango temporal. En tercer lugar no se poseían datos de las constantes vitales o estos eran de mala calidad lo que impedía contrastar la información con datos médicos fiables.
- Los algoritmos de clasificación desarrollados en la universidad de Burgos, el *Rotation Forest* y el *Random Balance* han sido los que mejores resultados han dado para este problema, teniendo en cuenta que no ha existido ninguno probado que haya sido capaz de predecir bien una crisis nueva.
- Los conjuntos de datos muy desbalanceados y muy ruidosos son difíciles de analizar. Más aún es encontrar modelos capaces de interpretarlos correctamente. Esto nos ha derivado en clasificadores que sobreajustan demasiado o que no son capaces de ajustar.

- Desarrollar la API usando el *framework Flask* para *Python* ha facilitado el desarrollo gracias a su simplicidad, en especial gracias a que no se requieren unos grandes conocimientos de programación de servicios *HTTP*.
- *SocketIO* es una herramienta que ha resultado muy útil para hacer uso *websockets* y hacer la difusión en tiempo real de los datos. La principal complicación que tuvo fue la integración con *Flask* al usar hilos, pero la programación de las funciones que usan de esta librería ha sido un proceso sencillo.
- TODO: conclusiones de la usabilidad de la página

7.2. Líneas futuras

En el caso de continuar este proyecto se podrían realizar:

- Probar *Bag of Words* para series temporales [9].
- Utilizar métodos de selección de instancias para eliminar ejemplos ruidosos de manera más correcta [1].
- En el caso de obtener nuevos datos mejor etiquetados y constantes vitales correctas se podría realizar los experimentos documentados.
- Internacionalizar la aplicación.
- Crear un sistema de localizaciones (hospitales, centros de discapacidad) para mejorar la gestión de las camas y los usuarios.

Bibliografía

[1] Álvar Arnaiz-González, José-Francisco Díez-Pastor, Juan J Rodríguez, and César García-Osorio. Instance selection of linear complexity for big data. *Knowledge-Based Systems*, 107:83–95, 2016.

[2] Damien Arrachequesne and Erik Little. Socket.io. https://socket.io/docs/, feb 2018.

[3] Karl J Aström. Event based control. In *Analysis and design of nonlinear control systems*, pages 127–147. Springer, 2008.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Radio Amiga Burgos. Álvar Arnaiz González y José Francisco Díez Pastor – Premiados del Concurso Desafío Universidad Empresa, Marzo 2018.

[6] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM.

[7] José F Díez-Pastor, Juan J Rodríguez, César García-Osorio, and Ludmila I Kuncheva. Random balance: ensembles of variable priors classifiers for imbalanced data. *Knowledge-Based Systems*, 85:96–111, 2015.

[8] José F Díez-Pastor, Juan J Rodríguez, César I García-Osorio, and Ludmila I Kuncheva. Diversity techniques improve the performance of the best imbalance learning ensembles. *Information Sciences*, 325:98–117, 2015.

[9] Johann Faouzi. pyts: a Python package for time series transformation and classification. https://pyts.readthedocs.io/en/latest/auto_examples/classification/plot_bossvs.html, May 2018.

[10] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–37, 1996.

[11] R. Fielding and J. Reschke. Hypertext transfer protocol (http/1.1): Semantics and content. RFC 7231, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7231.txt.

[12] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2012.

[13] Miguel Grinberg. Flask-socketio. https://flask-socketio.readthedocs.io/, jan 2019.

[14] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[15] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[16] Jesper Jeppesen, Sándor Beniczky, A Fuglsang Frederiksen, Per Sidenius, and Peter Johansen. Modified automatic r-peak detection algorithm for patients with epilepsy using a portable electrocardiogram recorder. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4082–4085. IEEE, 2017.

[17] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 07/02/2019].

[18] Juha M Kortelainen, Mark Van Gils, and Juha Pärkkä. Multichannel bed pressure sensor for sleep monitoring. In *2012 Computing in Cardiology*, pages 313–316. IEEE, 2012.

[19] Yatindra Kumar, ML Dewal, and RS Anand. Epileptic seizures detection in eeg using dwt-based apen and artificial neural network. *Signal, Image and Video Processing*, 8(7):1323–1334, 2014.

[20] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[21] National Institute of Neurological Disorders and Stroke NIH. Epilepsy information page. https://www.ninds.nih.gov/Disorders/All-Disorders/Epilepsy-Information-Page, may 2019.

[22] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] Daniel Pope. Gevent: asynchronous i/o made easy. https://doi.org/10.5446/19958, 2014. [Online; accessed 16 May 2019].

[25] Sriram Ramgopal, Sigride Thome-Souza, Michele Jackson, Navah Ester Kadish, Iván Sánchez Fernández, Jacquelyn Klehm, William Bosl, Claus Reinsberger, Steven Schachter, and Tobias Loddenkemper. Seizure detection, seizure prediction, and closed-loop warning systems in epilepsy. *Epilepsy & behavior*, 37:291–307, 2014.

[26] Juan José Rodríguez. Clasificación bayesiana, basada en instancias y por combinación. Universidad de Burgos. [Online; accessed 15-Jun-2019].

[27] Juan José Rodríguez. Introducción a la minería de datos. Universidad de Burgos. [Online; accessed 15-Jun-2019].

[28] Juan José Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. Rotation forest: A new classifier ensemble method. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1619–1630, 2006.

[29] Armin Ronacher, David Lord, Adrian Mönnich, and Markus Unterwaditzer. Welcome to jinja2. http://jinja.pocoo.org/docs/2.10/, 2008.

[30] Armin Ronacher, David Lord, Adrian Mönnich, and Markus Unterwaditzer. Welcome to flask. http://flask.pocoo.org/docs/1.0/, 2010.

[31] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[32] Ronald Schuyler, Andrew White, Kevin Staley, and Krzysztof J Cios. Epileptic seizure detection. *IEEE Engineering in medicine and Biology Magazine*, 26(2):74, 2007.

[33] Marie Tolkiehn, Louis Atallah, Benny Lo, and Guang-Zhong Yang. Direction sensitive fall detection using a triaxial accelerometer and a barometric pressure sensor. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 369–372. IEEE, 2011.

[34] Alexandros T Tzallas, Markos G Tsipouras, Dimitrios G Tsalikakis, Evaggelos C Karvounis, Loukas Astrakas, Spiros Konitsiotis, and Margaret Tzaphlidou. Automated epileptic seizure detection methods: a review study. In *Epilepsy-histological, electroencephalographic and psychological aspects*. IntechOpen, 2012.

[35] Wikipedia contributors. Arch linux — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Arch_Linux&oldid=896735398, 2019. [Online; accessed 16-May-2019].

[36] Wikipedia contributors. Bootstrap (front-end framework) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Bootstrap_(front-end_framework)&oldid=895052706, 2019. [Online; accessed 16-May-2019].

[37] Wikipedia contributors. Jquery — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=JQuery&oldid=896325154, 2019. [Online; accessed 16-May-2019].

[38] Wikipedia contributors. Manifold — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Manifold&oldid=877548508, 2019. [Online; accessed 17-January-2019].

[39] Wikipedia contributors. Mariadb — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=MariaDB&oldid=897072367, 2019. [Online; accessed 16-May-2019].

[40] Wikipedia contributors. Monkey patch — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Monkey_patch&oldid=897650268, 2019. [Online; accessed 18-June-2019].

[41] Wikipedia contributors. Nginx — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Nginx&oldid=896866646, 2019. [Online; accessed 16-May-2019].

[42] Wikipedia contributors. One-class classification — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=One-class_classification&oldid=897781715, 2019. [Online; accessed 15-June-2019].

[43] Wikipedia contributors. Principal component analysis — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Principal_component_analysis&oldid=878535871, 2019. [Online; accessed 17-January-2019].

[44] Wikipedia contributors. Proxmox virtual environment — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Proxmox_Virtual_Environment&oldid=896960348, 2019. [Online; accessed 16-May-2019].

[45] Wikipedia contributors. Wavelet transform — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Wavelet_transform&oldid=900879949, 2019. [Online; accessed 24-June-2019].

[46] Wikipedia contributors. Websocket — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=WebSocket&oldid=897985416, 2019. [Online; accessed 31-May-2019].