Proyecto Final

# Modelos de Deep Learning para la predicción del comportamiento sedentario futuro



# Universidad Nacional del Centro de la Provincia de Buenos Aires

# Facultad de Ciencias Exactas

# Ingeniería de Sistemas

**Alumno:** Santillán Cooper, Martín

**Directores**

Armentano, Marcelo

Tommasel, Antonela

[Introducción](#_bwxenck50ema) 4

[Motivación](#_q5c5w8glzi8i) 4

[Objetivos](#_dam41j7go72q) 5

[Solución propusta](#_a0czsxpqqexh) 6

[Marco Teórico](#_1gedkm8lbj6g) 6

[Inteligencia Artificial](#_sj1n6sg1145v) 6

[Machine Learning](#_96a4zfw1rsd0) 8

[Aprendizaje Supervisado y No Supervisado](#_ziax18pwx0k) 9

[Tipos de tareas](#_ziax18pwx0k) 9

[Entrenamiento de los modelos](#_ez6zzb4h85ot) 11

[Evaluación de los modelos](#_ziax18pwx0k) 13

[Generalizacion, underfitting y overfitting](#_ziax18pwx0k) 14

[Regularización (debajo de función de costo/pérdida)](#_8mcmbggk6h84) 15

[Clasificación y Regresión](#_ziax18pwx0k) 17

[Métricas de evaluación](#_fy1hu0h7xblr) 17

[Métricas de regresión](#_2ep9687rdxfr) 17

[Métricas de clasificación](#_npj5wxjjdw8) 19

[Deep Learning](#_ziax18pwx0k) 23

[Hiperparámetros](#_vu07wdmaldgb) 24

[Modelos de DL utilizados](#_96a4zfw1rsd0) 25

[Perceptrón Multicapa](#_ziax18pwx0k) 25

[Redes Neuronales Recurrentes](#_ziax18pwx0k) 28

[Redes Neuronales Convolucionales](#_ziax18pwx0k) 29

[Redes Temporales Convolucionales](#_cxfot1s9xzgl) 30

[Predicción de actividades](#_ukao3nv2lmz0) 32

[Trabajos relacionados](#_bwxenck50ema) 33

[Monitoreo y modelado del comportamiento humano utilizando smartphones](#_esa6d6panlb7) 33

[Redes Neuronales y series temporales](#_p1n00otb4p2g) 36

[Predicción del comportamiento sedentario](#_nkjsb1qugtvx) 37

[Predicción de Comportamiento Sedentario Futuro](#_bwxenck50ema) 41

[Evaluación Experimental](#_bwxenck50ema) 44

[Descripción del Dataset](#_s093l2b6kutm) 44

[Introduccion](#_n87t2tpgyp4o) 44

[Tipos de datos del dataset](#_sz6kovk3e9at) 44

[Datos no disponibles](#_l5x3lmcvuo7) 45

[Datos del acelerómetro](#_uisfuehtonv6) 45

[Datos de sensado](#_mvmik7j0jxx8) 46

[Datasets descartados](#_cc15la8gqxco) 48

[Pre-procesamiento del dataset](#_myvoi0gmjuzq) 48

[Análisis de datos](#_jlx3orj8yxc8) 48

[Tamaño de buckets](#_yvslk3ahs9dp) 48

[Disponibilidad de los datos](#_k7vn35flm781) 50

[Registros desconocidos](#_drbiv2psrh3l) 53

[Depuracion de buckets](#_jlx3orj8yxc8) 56

[Nivel de MET](#_2nqs14x4733i) 57

[Análisis del nivel de MET](#_a7ggrn221cnc) 59

[Inconsistencias encontradas](#_ykt2faadw8ix) 62

[Pipeline del preprocesamiento](#_5akxfwxylnb8) 63

[Selección y cómputo de características](#_iu4m6aih2hf4) 64

[Características Discretas](#_sfg3oyy7xfof) 64

[Características obtenidas a partir del GPS [Saeb 2016, cualquiera de los dos]:](#_yjq202r50uhd) 64

[Características basadas en el tiempo:](#_sjg563uez25j) 66

[Características de actividad física:](#_vw0jkvh8j8cr) 67

[Características de audio:](#_fj6szpa9f5r3) 68

[Otras características](#_nccr5fced68a) 68

[Características por intervalo](#_bwxenck50ema) 68

[Variable objetivo](#_y9duidtplvg7) 69

[Eliminación de valores nulos](#_isddz3rgyeb7) 70

[Tratamiento de variables categóricas](#_dh0p9774ynwl) 70

[Generación de datasets con retrasos](#_d9404x3ykd5) 72

[Separación entre X e y](#_c0k06mjirh2) 74

[Separación entre entrenamiento y testeo](#_r8nb29o58liv) 74

[Normalización](#_d8qo7gu7xjoy) 75

[Ajuste de las dimensiones de X](#_qriu9j6uzsv4) 75

[Validación de los modelos](#_jwjstti4vmcc) 76

[Métricas de evaluación](#_b10fvbibeday) 78

[Tunning de los modelos](#_6q1vpo9qlc4e) 79

[Optimizacion de Bayes](#_m8jwp7j0anwz) 79

[Selección de experimentos modelo](#_9plflhsz77t8) 80

[Selección de usuarios modelo](#_s66ur8g37jy7) 81

[Diseño de los experimentos](#_usyp7uyz8uuf) 82

[Resultados](#_9ycog7juxxqz) 83

[Resultados del proceso de tunning](#_qnqzosz30ohi) 84

[Resultados de los experimentos](#_t5ali0z4gpu0) 86

[Discusión](#_aoikelspm5xw) 89

[Conclusiones](#_bwxenck50ema) 90

[Bibliografía](#_bwxenck50ema) 91

## Introducción

## Motivación

La definición de comportamiento sedentario ha ido evolucionando a lo largo de los años, al mismo tiempo que lo hizo la forma de medirlo. La Red de investigación del comportamiento sedentario[[1]](#footnote-0) define el comportamiento sedentario como cualquier comportamiento de vigilia caracterizado por un gasto de energía ≤1.5 MET (equivalente metabólico de tareas) mientras se está sentado o reclinado [(Sedentary Behaviour Research Network ...)](https://paperpile.com/c/FQGDmo/EDKC). MET mide la intensidad de una actividad en múltiplos del gasto energético en reposo. Ejemplos de actividades sedentarias son mirar televisión (1.0 MET), comer sentado (1.5 MET), jugar videojuegos (1.0 MET) y conducir (1.3 MET).

La investigación realizada en el área demuestra asociaciones sólidas y consistentes entre el tiempo sedentario y la diabetes, las enfermedades cardiovasculares y la mortalidad en general [(Wilmot et al., 2012)](https://paperpile.com/c/FQGDmo/P5fD) [(Carter et al., 2017)](https://paperpile.com/c/FQGDmo/lXon). Sin embargo, las asociaciones informadas fueron en gran medida independientes de la actividad física. Por lo tanto, es importante tener en cuenta que el comportamiento sedentario no representa lo contrario de la actividad física y que es posible que un individuo tenga simultáneamente niveles altos de actividad física moderada a vigorosa (MVPA, por sus siglas en inglés) y comportamiento sedentario. En general, se ha demostrado que el tiempo en el que un individuo se encuentra en estado sedentario está asociado de forma perjudicial con la salud y con marcadores de riesgo metabólico en diversos grupos poblacionales. Además, se ha destacado la importancia de no solo estimular la MVPA, sino también de reducir el tiempo de sedentarismo, ya que la conducta sedentaria es un factor de riesgo para la mortalidad, independiente de la MVPA [(Koster et al. 2012)](https://paperpile.com/c/FQGDmo/8ipe).

Otros trabajos se han centrado en las asociaciones entre los descansos breves en el tiempo sedentario con los resultados metabólicos (Paing et al. 2018) y con la optimización de las operaciones cognitivas [(Felez-Nobrega et al., 2018; Magnon et al., 2018; Paing et al., 2018)](https://paperpile.com/c/FQGDmo/xvIZ+fa6g+Hfl9). [(Benatti & Ried-Larsen, 2015)](https://paperpile.com/c/FQGDmo/Nwq4) afirmaron que existe evidencia suficiente para demostrar los efectos positivos en el metabolismo de reducir los tiempos de conductas sedentarias.

Los métodos utilizados para medir la conducta sedentaria se pueden clasificar en subjetivos (cuestionarios y diarios de autoevaluación) y objetivos (sensores comunes de dispositivos ubicuos). Los métodos subjetivos están siendo superados por las nuevas tecnologías que pueden proporcionar, para todos los grupos de la población, información segundo a segundo sobre la postura, el movimiento (o su falta), y los patrones dentro y entre días [(Atkin et al., 2012)](https://paperpile.com/c/FQGDmo/45GF).

El área de *mobile health (mHealth)* ha crecido mucho en los últimos años gracias a la miniaturización de los dispositivos móviles, su poder de cómputo y su conectividad. *MHealth*  puede definirse como “el uso de comunicaciones móviles y tecnologías de red emergentes para el cuidado de la salud. *MHealth* tiene la capacidad de beneficiar la comunicación entre paciente y entidades de la salud, proveer servicios de salud y para promover comportamientos preventivos de la salud [(Dutta et al., 2018)](https://paperpile.com/c/FQGDmo/QgUc). De esta forma, existe un gran potencial para que las tecnologías de *mHealth* rediseñen casi todas las facetas de la atención médica [(Steinhubl et al., 2015)](https://paperpile.com/c/FQGDmo/QxBt). En particular, aplicaciones de *mHealth* han demostrado tener el potencial de promover cambios en el comportamiento sedentario y la actividad física [(Yerrakalva et al., 2019)](https://paperpile.com/c/FQGDmo/Kpxj).

Si bien los dispositivos móviles pueden considerarse como una de las causas del comportamiento sedentario [(He & Agu, 2016b)](https://paperpile.com/c/FQGDmo/GSfPw), también ofrecen nuevas oportunidades para prevenirlo. Hoy en día, los dispositivos móviles portátiles, como teléfonos inteligentes, relojes inteligentes y rastreadores de ejercicios están equipados con una amplia variedad de sensores que se pueden usar para registrar la actividad humana y llevar a cabo análisis de comportamiento a través de una técnica conocida como *lifelogging*. Se define a *lifelogging* como una forma pervasiva de realizar cómputos, que consiste en un registro digital unificado de la totalidad de las experiencias de un usuario, capturado a partir de sensores multi-modales digitales almacenados como un archivo personal multimedia. Estos sensores permiten a los *smartphones* capturar información sobre nosotros y sobre nuestros contextos y entornos de manera no intrusiva y continua [(Ali et al., 2014)](https://paperpile.com/c/FQGDmo/TYzA). El uso de métodos objetivos para evaluar el comportamiento sedentario está creciendo a medida que los costos de los dispositivos móviles portátiles disminuyen y son más fáciles de usar. En febrero de 2019, el 96% de las personas en los EE. UU. poseía un teléfono, de los cuales el 81% eran teléfonos inteligentes, también llamados *smartphones*. Los porcentajes asciende a 99% y 96% respectivamente en la población de entre 18 a 49 años[[2]](#footnote-1). En este contexto, estos dispositivos pueden verse como una oportunidad para desarrollar métodos objetivos complejos para medir y predecir el comportamiento sedentario.

## Objetivos

Se han implementado muchas aplicaciones para teléfonos inteligentes con el objetivo de alertar al usuario cuando se reconoce un comportamiento sedentario [(He and Agu 2014; Grundgeiger et al. 2017; Fahim et al. 2017)](https://paperpile.com/c/FQGDmo/F6qP+V35l+JRXi). Predecir conductas sedentarias futuras puede ayudar a habilitar intervenciones preventivas, como recordatorios y sugerencias para diferentes actividades basadas en la Teoría de la conducta planificada (TPB) (Ajzen 1991). TPB postula que es más probable que un sujeto participe en las intervenciones recomendadas para reducir las conductas sedentarias si tales actividades están incluidas en sus planes. Siguiendo esta idea, la hipótesis que motiva esta tesis es que *si se pudiese predecir en el momento que un sujeto será sedentario en el tiempo , se podrían recomendar actividades a modo de intervenciones con la finalidad de cambiar su rutina sedentaria a largo plazo*. Este tipo de intervenciones puede dar lugar a mejores oportunidades para cambiar el comportamiento de los sujetos y así tener estilos de vida más saludables.

El objetivo principal de esta tesis es:

* Predecir el comportamiento sedentario futuro mediante modelos de *Deep Learning* entrenados con datos de Lifelogging.

Para lograr el anterior objetivo principal, se buscan lograr los siguientes objetivos específicos:

* El análisis de datos de *lifelogging* para la definición de perfiles dinámicos de usuario capaces de capturar y representar el comportamiento del usuario, basado en fuentes de datos heterogéneas.
* La identificación de datasets apropiados para llevar a cabo el objetivo anterior.
* La utilización de los perfiles definidos para identificar patrones de comportamiento sedentario.
* El estudio, análisis y comparación de algoritmos de Deep Learning para la tarea de predicción del comportamiento sedentario.
* La definición de un proceso que asegure la correcta evaluación y comparación de los diferentes modelos de Deep Learning.
* La proposición de diferentes marcos donde los modelos propuestos pueden ser utilizados.

## Solución propusta

El problema de predecir el comportamiento sedentario futuro se ha abordado previamente analizando solo el tiempo inactivo / estacionario de un sujeto y se ha medido y comparado el rendimiento de varios modelos. Alternativamente, en esta tesis se define la Predicción del Comportamiento Sedentario Futuro (PCSF) como la tarea de predecir si la actividad física de un usuario superará o no, en promedio, 1,5 MET en un futuro próximo. Aunque MET es una métrica estándar en el área de la salud para medir la intensidad de una actividad en términos del gasto energético, el uso de ésta para predecir el comportamiento sedentario mediante el uso de dispositivos portátiles y dispositivos móviles permanece inexplorada.

En este trabajo, se presenta un enfoque novedoso para predecir el comportamiento sedentario futuro de un sujeto en términos de su nivel de MET a partir de diferentes modelos de Deep Learning, basado en la observación de valores obtenidos de múltiples sensores de dispositivos portátiles y móviles.

El resto de la tesis está organizada de la siguiente manera: en el capítulo 2 se expone el Marco Conceptual. En el capítulo 3 se muestran los trabajos relacionados. En el capítulo 4. En el capítulo 5.

## Marco Teórico

## Inteligencia Artificial

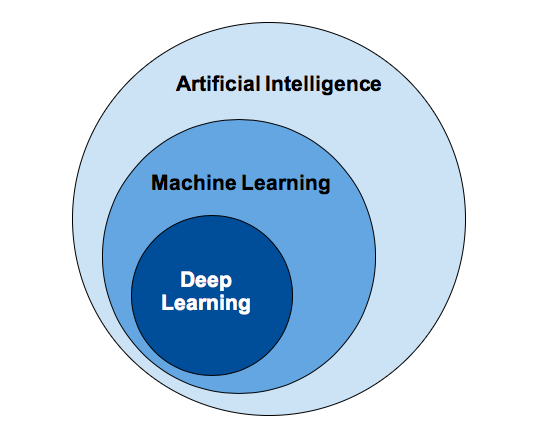
El objetivo principal de esta tesis es evaluar el uso de modelos de *Deep Learning* para el problema de predicción del comportamiento sedentario futuro a través de datos de Lifelogging. Por lo tanto, es importante dar una base a los conceptos más importantes en ambas áreas. Es debido notar que tanto Deep Learning como Machine Learning pertenecen al campo de Inteligencia Artificial y que a su vez Deep Learning pertenece al campo de Machine Learning. Por lo tanto, a continuación se presentan los conceptos más importantes de Machine Learning, que servirán para comprender el proceso llevado a cabo para generar los modelos. Luego, se presentan los conceptos más importantes de Deep Learning, que son una extensión de los presentados en la sección de Machine Learning.

[(Mitchell, 1997)](https://paperpile.com/c/FQGDmo/tsbr) define que un programa de computadora aprende a partir de una experiencia E con respecto a una tarea T y un determinado desempeño medido por D, si su desempeño en la tarea T, mejora con la experiencia E. Esta definición puede aplicarse a la totalidad del área de la Inteligencia Artificial. Los primeros programas de Inteligencia Artificial en tener éxito fueron aquellos basados en conocimiento o *knowldedge based.* En estos, se definía el conocimiento al que tenía acceso el programa a partir de lenguajes formales. Luego, el programa de Inteligencia Artificial resonaba a partir de reglas de inferencia lógica. Estos primeros programas consiguieron resolver problemas que son difíciles de resolver para seres humanos, como jugar al ajedrez. Irónicamente, los problemas más difíciles de resolver para un programa de inteligencia artificial son aquellos que los seres humanos llevan a cabo sin ninguna dificultad en la vida diaria, como reconocer objetos. Esto se da ya que el conocimiento utilizado para llevar a cabo estas tareas es subjetivo e intuitivo, y por lo tanto, difícil de articular de manera formal. Esto sugiere que los sistemas de Inteligencia Artificial deben ser capaces de adquirir su propio conocimiento, extrayendo patrones de los datos. Esta capacidad es conocida como *Machine Learning*. Por lo tanto, en el área de *Machine Learning*, el algoritmo de aprendizaje debe encontrar patrones en los datos de entrada, generando así conocimiento.

Una característica importante de Machine Learning es que el desempeño de sus algoritmos depende fuertemente de la representación de los datos de entrada (hand-crafted). Es decir que dicha representación ha sido calculada a partir de los datos crudos, ya que los algoritmos de Machine Learning no pueden aprender de ellos o les es más difícil. En este contexto, cada uno de los datos de entrada calculados es denominado característica o feature.

En [Deep Learning BOok Introduction] los autores incluyen, además, el área Representation Learning, que es un subconjunto de Machine Learning pero a su vez un superconjunto de Deep Learning. En Representation Learning se utiliza Machine Learning no solo para aprender del mapeo de los datos de entrada a la salida (predicción), sino que también para descubrir la representación óptima de los datos de entrada. De esta forma se puede obtener mejor desempeño, sobretodo para tareas complejas, para las cuales es difícil decidir cuál es el conjunto de características óptimo. Igualmente, existen casos en los cuales es igual de difícil aprender la representación óptima que resolver el problema original. Esta forma de aprendizaje de la representación óptima de los datos de entrada (por ejemplo, una foto, audio, datos de sensores) es llamada comúnmente end-to-end e implica que no es necesario realizar la búsqueda manual de la representación de los datos de entrada, sino que son hallados automáticamente.

*Deep Learning* resuelve el problema de Representation Learning, introduciendo representaciones que son expresadas en términos de otras representaciones más simples. De esta forma, el algoritmo de aprendizaje puede aprender conceptos más complejos a partir de conceptos más simples a través de diferentes niveles de representación. Un ejemplo de esto son las Redes Neuronales Convolucionales aplicadas a imágenes, donde se aprenden automáticamente filtros que transforman las imágenes originales. Diferentes niveles de filtros son aprendidos, de forma que a imágenes ya filtradas se les aplican nuevos filtros, de esta forma, llegando a encontrar la representación óptima de las imágenes para la tarea de aprendizaje que se quiere llevar a cabo. Por lo tanto, la representación aprendida por un mismo algoritmo va a diferir si la tarea en cuestión es, por ejemplo, la de encontrar objetos en la imagen o si esta es la de predecir si en la imagen es de noche o de día.



## Machine Learning

Machine Learning puede ser definido de varias maneras, dependiendo del autor y del campo de estudio desde el cual se aproxime uno. En [Machine Learning a probabilistic approach] es definida como un conjunto de métodos que pueden detectar patrones en los datos de entrada automáticamente, pudiendo usar luego esos patrones descubiertos para realizar diferentes tipos de decisiones bajo incertidumbre, dependiendo de si la tarea que realiza el modelo es supervisada o no supervisada. En (Libro Deep Learning), Machine Learning es definido como “una forma de estadística aplicada que hace gran énfasis en el uso de computadoras para estimar estadísticamente funciones complejas y un menor énfasis en probar intervalos de confianza en dichas funciones”.

En general, en el área de Machine Learning las tareas son descritas a partir de cómo el modelo de Machine Learning debería procesar un determinado caso o ejemplo. Un caso puede ser descrito como un conjunto de características que han sido cuantitativamente medidas a partir de cierto evento u objeto, que queremos que el modelo de Machine Learning procese. Dichas características pueden o no haber sido calculadas a partir de los datos originales.

Típicamente, cada caso se representa como un vector perteneciente a donde cada componente del vector representa una característica diferente. Además, los casos pueden ser representados en forma matricial u en otros espacios con mayores dimensiones. La representación de los datos va a diferir de acuerdo a la naturaleza de los casos y a la estructura de los datos de entrada que espera el algoritmo de aprendizaje. Por ejemplo, las características de una imagen son usualmente los valores de los píxeles de dicha imagen. Dependiendo de las imágenes, pueden estar representadas en 3 dimensiones (alto x ancho) o en 3 dimensiones (alto x ancho x canales) si la imagen tiene más de un canal, como es el caso de las imágenes satelitales. A su vez, el algoritmo de aprendizaje puede esperar como entrada un vector (como sería el caso de un Perceptron Multicapa) o alguna de las dos estructuras anteriormente nombradas. De esta forma, puede apreciarse que la estructura de los datos de entrada puede ser muy variada de acuerdo a la naturaleza de los datos, si los datos han sido preprocesados para generar nuevas características y del algoritmo de aprendizaje que se desea utilizar.

## Aprendizaje Supervisado y No Supervisado

Los diferentes algoritmos pueden ser clasificados a partir de la forma de los datos de entrada. En el caso del aprendizaje supervisado, los algoritmos utilizan *datasets* donde cada caso está asociado a una etiqueta . Dicha etiqueta es llamada variable objetivo. Las tareas de aprendizaje supervisado se basan en lograr predecir el valor de a partir de las características de . Durante la fase de entrenamiento, los algoritmos de este tipo pueden saber cuán acertadas son las predicciones de acuerdo a cuán alejadas están de la variable objetivo y así cambiar su forma de predecir para minimizar su error de predicción. En el caso de los algoritmos no supervisados se utilizan *datasets*  que no poseen una variable objetivo. Este tipo de algoritmos lleva a cabo la tarea de reconocimiento de patrones. Es decir, las variables de entrada son utilizadas para identificar grupos a partir del aprendizaje de propiedades útiles acerca de la estructura del *dataset*. Ejemplos: encontrar grupos con ejemplos similares (k nearest neighbours) o aprender la distribución de probabilidades del dataset (denoising, synthesis) [(Berry et al., 2019)](https://paperpile.com/c/FQGDmo/GuPX) .

## Tipos de tareas

Muchos tipos de problemas o tareas pueden ser resueltos utilizando *Machine Learning.* A continuación se presentan brevemente las tareas más comunes, ya que serán de ayuda al momento de discutir cuál es el tipo de tarea que se busca abordar en esta tesis:

* *Clasificación*: en este tipo de tarea un programa de computadora debe especificar a qué categoría pertenece un determinado valor de entrada. Para resolver esta tarea el algoritmo de aprendizaje debe aprender una función . Si, para un determinado ejemplo, Quiere decir que el modelo asigna un ejemplo descrito por el vector una categoría identificada como y. En general, la función , en lugar de dar como resultado una categoría específica, genera una distribución de probabilidades sobre las clases, que puede ser interpretada como la confianza del clasificador sobre el hecho de que la entrada pertenezca a cada clase. Luego, a partir de dicha distribución se decide la clase predicha de la entrada. Un ejemplo de tarea de clasificación es el reconocimiento de objetos, donde la entrada del algoritmo de aprendizaje es una imagen, generalmente descrita como un conjunto de píxeles, y la salida es un código numérico identificando al objeto en la imagen.
* *Clasificación con valores faltantes:* la tarea de clasificación se vuelve más difícil si al clasificador no se le garantiza que cada característica en el vector de entrada va a estar siempre presente. En la categoría anterior, el algoritmo de aprendizaje solo tiene que definir una sola función para mapear un vector de entrada a una salida de tipo categórica. Cuando algunos de los valores de entrada faltan, el algoritmo de aprendizaje debe aprender un conjunto de funciones donde cada función corresponde a clasificar al lector de entrada a partir de un subconjunto diferente de sus características faltantes. Una forma de definir eficientemente este gran conjunto de funciones que deben ser aprendidas es aprender la distribución de probabilidad sobre las variables de entrada y luego resolver la tarea de clasificación completando las variables faltantes a partir de la distribución de probabilidad aprendida.

* *Regresión:* Este tipo de tarea el algoritmo de aprendizaje debe predecir un valor numérico dado un cierto vector de entrada. Para resolver esta tarea el algoritmo de aprendizaje debe aprender una función . Este tipo de tarea es similar a las tareas de clasificación, excepto por el hecho de que el formato de salida es diferente.. Un ejemplo de tarea de regresión es la de predecir el valor monetario al cual debería venderse una casa dadas las características de dicha casa cómo entrada para el algoritmo de aprendizaje.
* *Transcription:* en este tipo de tarea el sistema de *Machine Learning* observa una representación relativamente desestructurada de algún tipo de dato y transcribe la información al formato textual. Por ejemplo, en reconocimiento óptico de caracteres el programa de computadora recibe una fotografía que contiene una imagen con texto y debe generar como resultado el texto en la forma de una secuencia de caracteres. A forma de otro ejemplo, Google Street utiliza técnicas de Deep learning para procesar la dirección de los números de las casas utilizando transcripción.
* *Traducción de máquina:* en este tipo de tarea la entrada consiste en una secuencia de símbolos en un determinado lenguaje y el programa de computadora debe convertirlos en una secuencia de símbolos de otro lenguaje. Este tipo de tarea es comúnmente aplicada a lenguajes naturales como, por ejemplo, traducir del español al portugués.
* *Salida estructurada:* las tareas de salida estructurada involucran cualquier tarea donde la salida es un vector con importantes relaciones entre sus diferentes componentes. Está categoría subsume a otras como transcripción y traducción, entre otras. Un ejemplo de esta tarea es el de la función de un *parser,* donde la entrada es un texto en lenguaje natural y la salida es un árbol que escribe la estructura gramatical de dicho texto etiquetando cada nodo del árbol como verbos, sustantivos, adverbios, etcétera.
* *Detección de anomalías:* en este tipo de tarea el programa de computadora analiza un conjunto de eventos u objetos y reporta algunos de ellos como inusuales o atípicos. Un ejemplo de tarea de detección de anomalías es la detección de fraudes de tarjetas de crédito. Al modelar los hábitos de compra de una compañía de tarjetas de crédito - es decir, modelando su distribución de probabilidad- es posible detectar el mal uso de dichas tarjetas. De esta forma, si un ladrón roba una tarjeta de crédito las compras del susodicho, éstas serán detectadas como *outliers* u anomalías de la distribución de probabilidad modelada.
* *Síntesis y sampleo de puntos:* en este tipo de tarea el algoritmo aprende de generar nuevos ejemplos que son similares a aquellos pertenecientes al conjunto de datos entrenamiento.
* *Denoising:* En este tipo de tarea el algoritmo de aprendizaje tiene como entrada un ejemplo corrompido obtenido a partir de un proceso de corrupción no conocido aplicado a un ejemplo limpio. El algoritmo debe, entonces, predecir el ejemplo limpio a partir del ejemplo corrompido.
* *Estimación de densidad o Estimación de la función de probabilidad de masa:* en esta tarea el algoritmo debe aprender una función que puede ser interpretada cómo es la función de densidad de probabilidad (si la entrada es continua) o como la función de probabilidad de masa (si la entrada es discreta). En general todos los tipos de tareas descritos arriba requieren que el algoritmo de aprendizaje al menos capture la estructura de la distribución de probabilidades. Este tipo de tarea es también utilizado para completar valores faltantes en los datos.

## Entrenamiento de los modelos

El entrenamiento de un modelo paramétrico de Machine Learning para una tarea supervisada consiste en encontrar el valor de los parámetros del modelo para los cuales el desempeño del modelo es el mayor. La **función de costo** (también llamada función de pérdida o función de error) indica cuán erradas son las predicción de dicho modelo dado el valor de los parámetros , se define al entrenamiento de un modelo de Machine Learning como el proceso de optimización de la función . Este proceso de optimización puede ser de maximización o de minimización. Aunque por lo general, si se trata de un problema de maximización se lo transforma en uno de minimización simplemente cambiando el signo de la función de costo.. En algunos casos, la optimización de los parámetros de un modelo para encontrar el menor error posible puede ser hallado analiticamente, como se da en el caso de la regresión lineal, aunque en la mayoría de los casos debe usarse un proceso iterativo ya que no es posible hayar una solucion analitica.

Por lo general, los métodos de optimización de la función de costo de un modelo de Machine Learning están basados en un algoritmo conocido como Descenso por el Gradiente, o *Gradient Descent*. Este algoritmo utiliza la idea de gradiente para decidir cómo deben variar los parámetros del modelo para minimizar . El gradiente es la extensión del concepto de derivada para espacios multidimensionales. El gradiente es, entonces, un vector director unitario, donde cada uno de sus componentes es calculado como la derivada parcial de cada dimensión. En cada paso del Descenso por el Gradiente, se computa el valor de la función de costo y luego, el gradiente. A partir de él se actualizan los parámetros del modelo utilizando la siguiente fórmula:

,

Donde representa a la tasa de aprendizaje o *learning rate* y es el gradiente de la función de costo con respecto a los parámetros. La tasa de aprendizaje es un escalar positivo y decide el tamaño del paso que dará el vector de los parámetros en la dirección en la que la función de costo disminuye.

El valor de debe ser elegido cuidadosamente. Un valor demasiado grande puede provocar que el proceso de minimización nunca llegue a converger y que el valor de la función de costo se vuelva caótico aumentando y disminuyendo iteración tras iteración. Por otra parte, un valor demasiado pequeño puede provocar que el proceso de minimización sea extremadamente lento, ya que, aunque los parámetros del modelo se ajustarán -en dirección del gradiente-, se necesitarán demasiadas iteraciones para que la función de costo converja . Para evitar este problema, suele utilizarse un valor de expresado a través de una función de la cantidad de épocas de entrenamiento, donde disminuye en función del número de iteración. Además, muchos de los algoritmos que extienden la idea del Descenso por el Gradiente utilizan otros parámetros que son aprendidos.

La función de costo se calcula a partir de la fórmula

,

donde es un caso de entrenamiento e su etiqueta, la cantidad de casos de entrenamiento disponibles y es la función que computa el error o costo para un caso de entrenamiento. El resultado de esta fórmula es, entonces, la esperanza de la sumatoria del error de cada uno de los casos de entrenamiento.

Se considera que el proceso de optimización que se realiza al entrenar un modelo de Machine Learning no responde a un proceso de optimización normal. La optimización de un modelo de Machine Learning se realiza indirectamente, ya que se lleva a cabo a partir del error del predictor con respecto al conjunto de casos de entrenamiento, cuando, en realidad, su desempeño está dado por su rendimiento en el conjunto de casos de testeo. Otra clara diferencia es que al entrenar un modelo de Machine Learning no se optimiza su función de costo hasta hallar la convergencia, o hasta que los gradientes sean muy pequeños, es decir, hasta llegar a un mínimo local o global. Sino, que se optimiza la función de costo hasta que ciertas condiciones sean cumplidas, como pueden ser que se haya llegado al desempeño deseado, que se hayan llevado a cabo la cantidad de iteraciones planeadas o que algún algoritmo como *Early Stopping* indique que el proceso de entrenamiento debe detenerse debido a que detectó *overfitting.*

Existe una buena cantidad de algoritmos de optimización de modelos de Machine Learning que utilizan la idea del Descenso por el Gradiente. Al método que, para actualizar el cálculo de , utiliza todos los casos de entrenamiento disponibles se lo conoce como Descenso por el Gradiente Determinístico. En la práctica, realizar el cómputo de la función de costo para realizar una única actualización de los parámetros puede ser demasiado costoso cuando la cantidad de casos de entrenamiento asciende desde cientos de miles a millones. En su lugar, se utiliza un menor a la cantidad total de casos de entrenamiento disponibles para ajustar los parámetros, es decir que se calcula, en cada iteración, a partir de uno a varios cientos de casos de entrenamiento. Cuando el valor de es uno se le llama Descenso por el Gradiente Estocástico (SGD, por sus siglas en inglés), mientras que cuando es mayor a 1 se lo conoce como Descenso por el Gradiente Mini-Batch (aunque, en ocasiones, también llamado estocástico).

Utilizar un más pequeño permite que el algoritmo converja más rápidamente en términos de la computación requerida -y no en el número de actualizaciones requeridas-. Además, una ventaja de poder seleccionar arbitrariamente es que,en caso de que se esté usando una unidad de procesamiento que permita realizar cálculos paralelamente como GPUs y CPUsde varios núcleos, puede aprovecharse la oportunidad de paralelización del cálculo del error de la predicción de cada caso de entrenamiento perteneciente al *batch*.

Los algoritmos de optimización utilizados en la práctica extienden la idea de SGD. El algoritmo *Momentum* utiliza el promedio de decadencia exponencial del gradiente para actualizar los parámetros y no el gradiente en si, lo que permite reducir la varianza introducida por SGD. La función de costo es muy sensible con respecto a algunos parámetros mientras que es insensible para otros. *Momentum* tiene lo anterior en cuenta, a expensas de introducir otro hiperparámetro (el que decide cuánto se tiene en cuanta los gradientes pasados). Otros algoritmos introducen la idea de tener una tasa de aprendizaje individual para cada parámetro. Algunos ejemplos de este último tipo de algoritmo de optimización son AdaGrad, RMSProp, ADAM y NADAM [(Mustapha et al. 2020)](https://paperpile.com/c/FQGDmo/eIpe).

## Evaluación de los modelos

Existen diferentes métodos para evaluar el desempeño de un modelo de Machine Learning. El uso de tal o cual modelo depende de la naturaleza de los datos y la cantidad de ellos.

La forma más simple de evaluar un modelo es dividirlo en un dataset de entrenamiento y otro de testeo. En este caso, se entrena el modelo utilizando como datos de entrada el dataset de entrenamiento y luego se evalúa al modelo a partir de los datos de testeo. Es importante notar que los datos de testeo deben ser datos a partir de los cuales el modelo no fue entrenado, por lo que estos dos conjuntos deben ser siempre disjuntos. De esta forma, se evalúa la capacidad de **generalizar** del modelo. La habilidad de un modelo para desempeñarse bien en datos que no habían sido observados es llamada generalización.

En el proceso de entrenamiento se buscan los valores óptimos de los parámetros que permitan el mejor desempeño del modelo en el conjunto de entrenamiento, a partir de un método de optimización, como el descenso por el gradiente. Para llevar a cabo dicho proceso de entrenamiento es necesario definir antes el valor de los hiperparámetros, ya que diferentes valores suponen diferentes modelos a ser entrenado, y por lo tanto, posiblemente diferente desempeño en la tarea a realizar. Bajo este esquema, lo más común es entrenar y testear al modelo a partir de diferentes combinaciones de hiperparámetros, seleccionando al final la combinación a partir de la cual el modelo ha obtenido el mejor desempeño en el conjunto de testeo. Estas combinaciones pueden crearse de manera manual o mediante algún método de optimización, como la Optimización de Bayes.

Puede observarse ahora que, de la misma forma que se optimizan los parámetros del modelo para obtener un buen desempeño en el conjunto de entrenamiento, los hiperparámetros también deben ser optimizados para alcanzar un buen desempeño en el conjunto de testeo. Por lo tanto, existe la necesidad de un tercer conjunto, que permita verificar el poder de generalización del modelo que mayor desempeño alcanzó en el conjunto de testeo. Este último conjunto es llamado dataset de validación.

Cuando los datos disponibles son escasos, el método anterior no debe ser utilizado, ya que las diferentes particiones dejarán de ser estadísticamente representativas de la fuente original de los datos (sobretodo los conjuntos de testeo y validación, que, en general, representan un porcentaje mínimo del *dataset*). En estos casos, se utiliza la validación cruzada o *cross-validation.* La validación cruzada es un método estadístico para evaluar y comparar modelos de *Machine Learning* [*(Refaeilzadeh et al. 2009)*](https://paperpile.com/c/FQGDmo/tMvw). La técnica más básica de validación cruzada es llamada *k-fold.* Esta técnica consiste en dividir el *dataset* en particiones. Para cada partición , se entrena el modelo en las particiones restantes y se lo evalúa en la partición . La evaluación final del modelo es igual a la media del desempeño en cada una de las iteraciones. La técnica anterior tiene la característica de tener baja replicabilidad. Una extensión a *k-fold* es conocida como *multiple run k-fold* y consiste en realizar la validación cruzada sobre las particiones repetidas veces, donde cada corrida genera diferentes particiones, aumentando así la replicabilidad [(Bouckaert 2003)](https://paperpile.com/c/FQGDmo/o9Vb).

En el caso en que se esté prediciendo el futuro a partir del pasado, como es el caso de las series de tiempo*,* ciertas consideraciones especiales deben ser tomadas en cuenta. Para empezar, los datos no deben ser reordenados aleatoriamente . Y, más importante, sin importar la técnica de validación, los datos de testeo deben ser siempre temporalmente posteriores a los datos de entrenamiento. Existen técnicas de validación específicas para las series de tiempo que serán explicadas en la sección [(Chollet, 2017)](https://paperpile.com/c/FQGDmo/KYgf).

## Generalizacion, *underfitting* y *overfitting*

El objetivo principal en Machine Learning es que el modelo entrenado para llevar a cabo una tarea determinada tenga un buen desempeño en datos de entrada nuevos. Esto es, en casos que no hayan sido vistos antes por el modelo, y no solamente en los casos que fueron utilizados para entrenar a dicho modelo.

Como se explicó en la sección anterior, al entrenar un algoritmo de Machine Learning, se utiliza un conjunto de datos de entrenamiento, a partir del cual se intenta minimizar el error, lo que puede considerarse un problema de optimización. A partir de esto, se calcula el error de generalización o de testeo. El error de testeo es el error esperado de un modelo al predecir a partir de un caso no observado, que es calculado a partir del conjunto de testeo. Los parámetros del modelo (aquellos que se ajustan en el proceso de optimización) son ajustados de acuerdo al conjunto de entrenamiento, por lo que se espera que el error de testeo sea siempre mayor al error de entrenamiento.

Por lo tanto, se busca lograr dos objetivos claros al momento de validar un algoritmo de Machine Learning:

* Lograr que el error de entrenamiento sea bajo, y
* Lograr que el error de testeo sea lo más similar posible al error de entrenamiento.

Estos dos objetivos se corresponden con dos desafíos importantes a superar en Machine Learning: *underfitting* y *overfitting*. Underfitting se da cuando el modelo no logra obtener un error de entrenamiento lo suficientemente bajo, mientras que overfitting ocurre cuando la diferencia entre el error de entrenamiento y el de testeo es demasiado grande.

La tarea a la hora de entrenar un modelo es encontrar un punto medio de tal modo de que no se produzca ni overfitting ni underfitting. Esto último se logra aumentando o disminuyendo la capacidad del modelo. La capacidad se refiere a la complejidad del espacio de hipótesis, esto es, todas las funciones que el algoritmo de aprendizaje puede llegar a seleccionar como posible solución hipótesis.

Por ejemplo, el espacio de hipótesis de la regresión lineal es el conjunto de todas las posibles funciones lineales de la forma

.

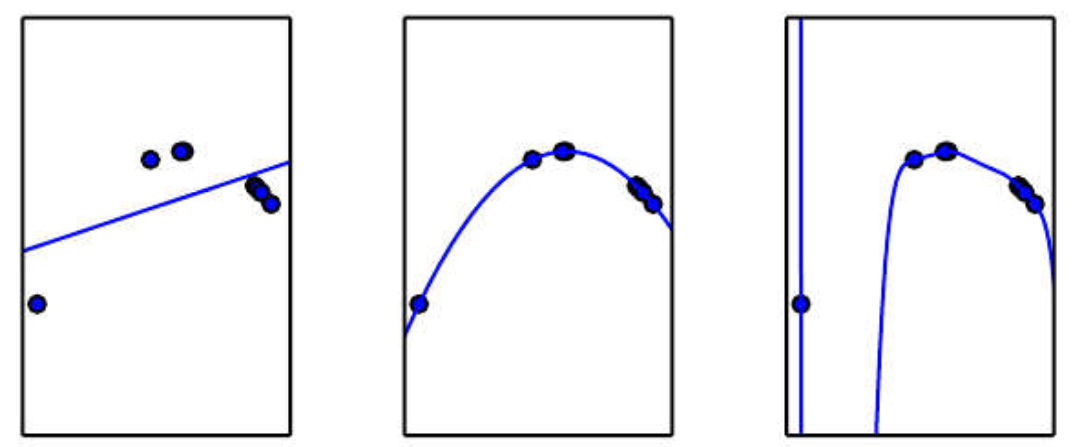
La capacidad de un algoritmo de este tipo puede aumentarse generalizando el espacio de hipótesis para incluir polinomios de grado de la forma

.

El desempeño de los modelos de Machine Learning va a ser mejor cuanto mejor se adapte su capacidad a la complejidad de la tarea a llevar a cabo. De esta forma, los modelos con una baja capacidad son incapaces de resolver tareas complejas, por lo que va a producirse underfitting. Mientras que los modelos con alta capacidad pueden resolver tareas complejas, pero sí su capacidad es más alta que la necesaria para resolver una determinada tarea, pueden cometer overfitting, ya que la función aprendida no tendrá un buen desempeño en los casos de testeo.

En la Fig X pueden verse tres diferentes modelos entrenados a partir de un conjunto de datos. Dicho conjunto de datos fue generado evaluando valores aleatorios de x en una función cuadrática. En este caso, la tarea de los modelos es la de predecir el valor de para cada valor de . Los modelos fueron ajustados a una función lineal, una función cuadrática y un polinomio de grado 9, respectivamente. Cada modelo ha debido ajustar un número diferente de parámetros, uno por cada grado del polinomio. Es decir, el primer modelo debió ajustar 2 parámetros (el peso asociado a la variable de grado 1 y aquel asociado a la variable de grado 0, también llamado bias), el segundo 3 y el último 9. Por lo tanto, la capacidad de los modelos aumenta de acuerdo al grado del polinomio, ya que el conjunto de hipótesis de un polinomio de grado incluye al de uno de grado , si . Puede observarse entonces que:

* El modelo de la izquierda produce underfitting, ya que debido a su baja capacidad no puede obtener un bajo error de entrenamiento.
* El modelo de de la derecha produce overfitting, ya que tendrá un error alto en los casos de testeo por no haber podido aprender la función verdadera a partir de la cual el conjunto de datos fue generado.
* Por último, el modelo del centro la capacidad es la adecuada para la tarea a realizar, por lo que el modelo ha logrado generalizar a partir de los casos de entrenamiento.



Función lineal, Polinomio grado 2, grado 9 [(Goodfellow et al., 2016)](https://paperpile.com/c/FQGDmo/ZFoL).

## Regularización (debajo de función de costo/pérdida)

En la sección anterior, se ha presentado un trade-off entre overfitting y underfitting a partir de aumentar o disminuir la capacidad de un algoritmo de aprendizaje al cambiar el modelo del cual aprende (la función). Este proceso puede llevarse a cabo de forma más general, ya que es posible configurar a un algoritmo para preferir algunas funciones pertenecientes a su espacio de hipótesis por sobre otras de acuerdo a cierto criterio utilizando una técnica de regularización. Es decir, las funciones menos preferidas van a ser elegidas sólo si se desempeñan mucho mejor que las preferidas en el conjunto de entrenamiento. Puede interpretarse a la regularización como una forma general de variar la capacidad de un algoritmo de aprendizaje, ya que quitar a una función del espacio de hipótesis puede interpretarse como una preferencia infinita de las funciones que aún pertenecen al espacio de hipótesis por sobre la función excluida. Por lo tanto, las técnicas de regularización tienen como objetivo disminuir la capacidad de un algoritmo de aprendizaje sin excluir explícitamente ninguna función del espacio de hipótesis, sino que dejando que el algoritmo defina cuales son las funciones más apropiadas a aprender en el proceso de entrenamiento.

Se define a la regularización como cualquier cambio que se le haga al algoritmo de aprendizaje para reducir el error de testeo o generalización, pero no su error de entrenamiento. Dicho de otra manera, se busca que el modelo aprenda la función más simple posible que consiga un bajo error de entrenamiento, de tal forma que pueda aun generalizar, es decir, tener un bajo error de testeo.

A modo de ejemplo, supóngase que se generan conjuntos de entrenamiento y testeo a partir de una función cuadrática. A continuación, se entrena un modelo cuya capacidad son todas las funciones polinómicas. Si no se utilizase ninguna técnica de regularización, el algoritmo aprendería una función utilizando un polinomio de grado (es decir, no tendrá ninguna preferencia por aprender polinomio de grado 2). De esta forma, el error de testeo será elevado, ya que el modelo predecirá el conjunto de testeo a partir del polinomio aprendido y no específicamente a partir de la función cuadrática. En este caso, una técnica de regularización podría penalizar al algoritmo por cuanto más grande sea el grado del polinomio. Debido a esto, el algoritmo deberá minimizar, no solo el error de entrenamiento, sino que también el grado del polinomio. Por lo tanto, el algoritmo terminará aprendiendo la función cuadrática. Nótese que el algoritmo no debería aprender un polinomio de grado 1 (función lineal) ya que de esta forma aumentaría el error de testeo (suponiendo que el conjunto de entrenamiento es lo suficientemente grande como para no parecerse a una función lineal).

A continuación se presentan las técnicas de regularización más comunes que pueden ser utilizadas tanto en algoritmos de *Machine Learning* como de *Deep Learning*. Las técnicas de regularización que son aplicadas específicamente para algoritmos de Deep Learning serán detalladas en su propia sección.

La “Penalización de la norma de los parámetros”consiste en añadir una penalidad de la norma de los parámetros a la función de pérdida , de tal manera que el algoritmo de optimización deba minimizar, no solo el error, sino que también la norma de los parámetros. Si llamamos a esta nueva función objetivo, entonces:

Donde es un hiper parámetro que modifica la contribución del término de penalización a la función objetivo. Si es igual a 0 entonces no hay regularización y cuanto mayor es mayor es la regularización aplicada. Existen diferentes funciones de la norma de los parámetros que pueden ser usadas como el término en la función objetivo regularizada. A continuación se presentan varias de ellas:

* RegularizacionEsta técnica de regularización es también conocida como *weight decay* o ridge regression. La regularización acerca el vector de los parámetros al origen añadiendo el siguiente término de regularización

donde w es el vector de los parámetros.

* Regularizacion : esta técnica de regularización tiende a que algunos parámetros tomen el valor de 0, por lo que puede ser usada como para seleccionar las características más importantes para el modelo (feature selection). Está definida por el término

,

* Selección de un tamaño de bache pequeño: puede considerarse a la selección de un tamaño de bache pequeño como un tipo de regularización, ya que añade ruido al proceso de aprendizaje.

## Clasificación y Regresión

En esta tesis, se define a la predicción del comportamiento sedentario futuro (PCSF) como el problema de predecir si la actividad de un usuario va a superar, o no, 1.5 METS en el futuro cercano. En este punto, podemos divergir el camino de investigación en dos. En el primero, se pueden tomar los datos de la actividad física, calcular el nivel de MET y generar modelos que puedan predecir el valor de MET calculado en el futuro cercano. En este primer caso, la variable a predecir es un valor continuo y por lo tanto abordaríamos un problema de regresión. En el segundo camino, se puede utilizar el valor de MET calculado para generar dos clases, una que indique comportamiento sedentario y otra que no. En este caso, la variable a predecir es discreta y puede tomar solo dos valores diferentes y por lo tanto abordaríamos un problema de clasificación.

Estos dos tipos de caminos representan diferentes formas de atacar el problema cada una con sus ventajas y desventajas. Al requerir cada problema modelos con diferentes características propias, tanto la forma en la que cada modelo se entrena como la evaluación de su desempeño va a diferir. A continuación, se describen las características, similitudes y diferencias entre estos dos tipos de modelos.

FALTA COMPLETAR (similitudes y diferecias)

## Métricas de evaluación

Para evaluar las habilidades de un algoritmo de Machine Learning debe utilizarse una métrica cuantitativa de su desempeño. Las métricas utilizadas poseen diferentes cualidades y existen diferentes de ellas para cada tipo de tarea. Además, cada una de las métricas existentes se desempeña mejor o peor dependiendo de las características específicas del problema. A continuación, se detallan y discuten algunas de las métricas de evaluación más utilizadas para los las tareas de regresión.

## Métricas de regresión

Las métricas de regresión son utilizadas para comparar los valores predichos por modelos de regresión con los valores reales. Sean la función aprendida por el modelo e la variable objetivo, con e , la métricas de regresión tienen por objetivo evaluar cuán precisamente los valores predichos por el modelo se asemejan a la variable objetivo . En otras palabras, buscan resumir la distancia entre e para cada uno de los casos de testeo. Para todas las métricas, el resultado va a ser mejor cuanto más cercanos sean e , pero difieren en distintos aspectos, como el método de agregación de la distancia entre los puntos para el *dataset* de testeo y el método para obtener la distancia entre dos puntos.

A continuación se presentan las métricas de regresión más simples o primarias:

* Magnitud del error: esta métrica no es utilizada pero sirve a modo de comprensión de las siguiente métricas. Esta métrica se basa en obtener la magnitud para todos los casos de testeo y obtener la media. De esta forma, para cada caso se puede determinar claramente si el modelo tiende a predecir un valor mayor o menor que el valor real. El problema de este método se da en la agregación, dónde errores negativos y positivos se cancelan entre ellos, por lo que el resultado no es confiable. Por esta razón, las métricas usadas utilizan ya sea el cuadrado o el valor absoluto al momento de obtener cada error individual.
* Error cuadrático medio o *Mean Squared Error (MSE):* esta métrica se calcula como la media de la suma de los cuadrados de la diferencia entre los valores e . Su formula es:

En este caso, podemos observar que la función de distancia es el cuadrado de la diferencia. De esta forma, los errores no pueden ser negativos y cancelarse. Por otro lado, la función de agregación, que resume los errores individuales, es la media.

Como consecuencia de utilizar el cuadrado de la diferencia, el error individual crece cuadráticamente con respecto a la diferencia entre e . De esta forma, se sobre-penalizan los errores grandes. Por lo tanto, esta no es una métrica recomendada en casos donde el *dataset* posea muchos valores atípicos, también conocidos como *outliers,* ya que puede darse un error muy grande aun cuando el modelo haya aprendido bien la tarea a realizar. En otras palabras, se produce una sobreestimación de cuán erróneo es el modelo.

Una importante ventaja de la métrica MSE es que es diferenciable en todos sus puntos, por lo que puede ser utilizada sin problemas como función de optimización.

* Raíz del Error cuadrático medio o *Root Mean Squared Error (RMSE):* esta métrica es muy similar a MSE. A diferencia de esta última, el método de agregación aplica la raíz cuadrada luego de calcular la media. Por lo tanto, su formula es:

Aplicar la raíz cuadrada tiene la propiedad de que el error estará expresado en la misma unidad que la variable objetivo, lo que le otorga más significado al error. Este no es el caso de MSE, ya que el error estará expresado en el cuadrado de la dimensión de la variable objetivo.

* Error medio absoluto o Mean absolute Error (MAE): este error difiere de las dos métricas anteriores en el método utilizado para calcular la distancia entre el valor de la variable objetivo y el del valor predicho. En este caso, en lugar del cuadrado de la distancia, se utiliza el valor absoluto de ella. MAE, por lo tanto, está definida por la siguiente fórmula

Al utilizar valor absoluto, el error estará expresado en la misma unidad que lo está la variable objetivo, como se da en el caso de RMSE. La no utilización del cuadrado para calcular la distancia entre los puntos implica ciertas consecuencias. Por un lado, MAE no es diferenciable, por lo que presenta ciertas dificultades si se la quiere utilizar para la optimización de parámetros de modelos de *Deep Learning.* Por otro, MAE penaliza linealmente los errores, por lo que si se quiere que el modelo preste más atención a los *outliers* tal vez no es la métrica adecuada.

Existen multitud de métricas de normalización que no están mencionadas en esta sección. Por tal razón, a continuación se listan ciertas características por las cuales difieren de las métricas previamente presentadas. Existen métricas que utilizan otros métodos de agregación. Por ejemplo, en lugar de la media, se utiliza la media geométrica o simplemente la suma de todos los errores. Otras métricas, conocidas como adimensionales o *dimensionless*, aplican una normalización a los errores individuales. Por ejemplo, puede dividirse a cada error por el valor de la variable objetivo, a forma de normalizar el resultado y hacerlo adimensional. La ventaja de la normalización es que permite la comparación de resultados de la performance del modelo entre diferentes *datasets*.

## Deep Learning

Como fue notado brevemente en la introducción, *Deep Learning* se diferencia de *Machine Learning* en que el primero utiliza una sucesión de representaciones, lo que permite conseguir representaciones más complejas. Igualmente, como es en el caso de *Machine Learning*, todos los algoritmos de aprendizaje tienen por objetivo resolver una tarea específica al aproximar un función .

Todos los modelos utilizados en esta tesis son casos de redes neuronales. Cada uno de los modelos tiene características que los diferencian de los demás y que los hacen más o menos propensos a ser usados en ciertos tipos de tarea. A su vez, estas diferencias implican la existencia de ventajas y desventajas a la hora de utilizarlos, y no solo teniendo en cuenta los tipos de tareas en los cuales se desempeñan mejor, sino que también entran en juego otros factores como el tiempo de entrenamiento necesario, el número de parámetros a entrenar, los hiperparámetros a ajustar, el tamaño del modelo, etc.

La palabra red de “Redes Neuronales” se debe a la estructura de grafo a partir de la cual estos modelos pueden ser representados,

Durante el entrenamiento de una red neuronal, se intenta que la función sea lo más parecida posible a la función subyacente que da origen a los datos de entrada, . Como en el caso de *Machine Learning*, en las tareas supervisadas, como la que se ataca en esta tesis, cada caso de entrenamiento está acompañado por una etiqueta o *label .* En *Deep Learning*, se busca que el resultado de la *output layer* sea lo más parecido posible a .

## Hiperparámetros

La mayoría de los modelos de *Deep Learning* tienen ciertas configuraciones que deben ser ajustadas para que el modelo se desempeñe de la mejor manera posible. Estos hiperparametros no son aprendidos por el algoritmo de aprendizaje que utiliza el modelo. Por esta razón, no son considerados parámetros. Como se verá más adelante, pueden utilizarse algoritmos de aprendizaje para encontrar los mejores valores para los hiperparámetros para el modelo.

Como ya se ha visto en la sección [SECCION Evaluación de los modelos] los hiperparametros introducen la necesidad de un nuevo *dataset*, diferente al *dataset* de testeo, en el cual sea posible evaluar la mejor configuración de los hiperparametros hallada. Este tercer *dataset -* en adición a los de entrenamiento y testeo- es conocido como *dataset* de validación.

Los hiperparametros pueden ser configuraciones relacionadas al algoritmo de aprendizaje o a la arquitectura del modelo. Por lo tanto, al existir diferentes arquitecturas, cada modelo tendrá hiperparametros específicos. A continuación, se definen los hiperparametros más generales y que forman parte de todos los modelos. Los hiperparámetros que difieren de modelo en modelo serán introducidos en su respectiva sección.

* Número de capas: el número de capas que tiene el modelo, donde cada capa esta forma neuronas.
* Número de neuronas: número de neuronas de cada capa. La cantidad de neuronas puede diferir de la cantidad de neuronas presentes en otra capa.
* Función de activación: es la función que se aplica a la salida de las neuronas que pertenecen a las capas ocultas de la red neuronal. La función de activación tiene la finalidad de introducir no-linealidades a la red neuronal. Es decir, sea la funcion de activacion, entonces es no-lineal.

Típicamente, el valor de una neurona es la suma de sus entradas ponderadas por sus respectivos pesos, esto es, un número real. Por lo tanto, es una función tal que e , donde no puede ser expresada como una función lineal, de la forma .

Existen muchas funciones de activación. Por ejemplo, la función sigmoide , la función tangente . Asimismo, la funcion de activacion que se ha vuelto el estándar en la actualidad es llamada ReLU, por su nombre en inglés “*Rectified Linear Unit*” donde . Esta función tiene la propiedad de ser ‘casi’ lineal, ya que está compuesta por dos funciones lineales, aunque sigue siendo no lineal. De esta forma, comparte ciertas propiedades con las funciones lineales, lo que hace que los modelos que las utilicen sean fáciles de optimizar a partir de métodos de optimización basados en el gradiente [(Goodfellow et al., 2016)](https://paperpile.com/c/FQGDmo/ZFoL).

* Número de épocas: número de veces que se recorre el *dataset* de entrenamiento completo en el proceso de entrenamiento.
* Tamaño de bache: número de baches en el que se divide el *dataset* de entrenamiento. Para usar baches, el algoritmo de optimizacion de parametros divide el *dataset* de entrenamiento en baches. Luego, para cada bache, se computa el error del modelo y se actualizan los parámetros mediante la optimización basada en el gradiente. La ventaja de actualizar los parámetros al final de cada bache y no luego de recorrer el *dataset* de entrenamientoentero es que permite realizar los parámetros se actualizan con mayor frecuencia, lo que permite que la función de optimización converja más rápidamente. Si la cantidad de baches es 1, se realizan tantas actualizaciones a los parámetros como número de épocas.
* Tasa de aprendizaje o *learning rate*: define el factor en que se actualizan los parámetros del modelo en relación al gradiente computado en cada iteración. Sean los parámetros del modelo, el gradiente computado y la tasa de aprendizaje, entonces los parámetros del modelo se actualizan como al final de cada iteración.

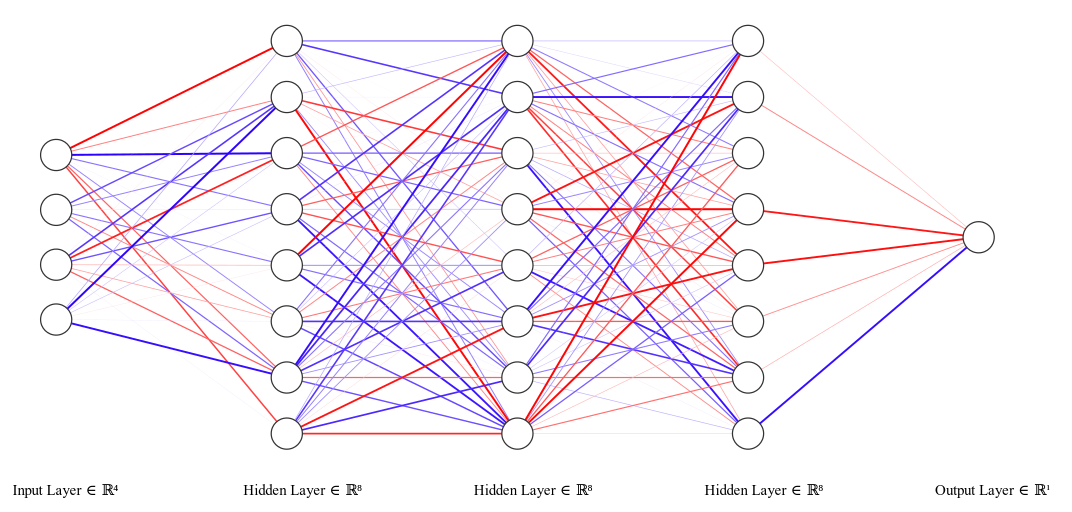
## Modelos de DL utilizados

A continuación, se presentan los diferentes modelos utilizados, todos ellos pertenecientes al área de Aprendizaje Profundo o *Deep Learning.* Para cada uno de ellos, se realiza una breve introducción y se desarrollan sus características, ventajas y desventajas de su utilización. Además, se discute el uso de cada tipo de modelo en el marco de problemas donde los datos de entrada poseen una organización secuencial.

## Perceptrón Multicapa

El Perceptrón Multicapa o Multilayer Perceptron -MLP, por sus siglas en inglés- es el modelo más sencillo dentro del campo de *Deep Learning* y puede aprender múltiples niveles de representación de los datos de entrada para modelar relaciones complejas entre dichos datos. De esta forma, las características de alto nivel están definidas en términos de características de bajo nivel. En este sentido, se toma a cada capa como un nivel de representación que utiliza la representación de la capa anterior. Por lo tanto, los MLP se componen de una secuencia de capas, donde cada capa modela una función que utiliza como entrada la salida de la capa/función anterior. De esta forma, puede representarse a los MLP como una composición de funciones (Goodfellow, Bengio, and Courville 2016).

En las MLP, cada capa está compuesta por un cierto número de unidades computacionales o neuronas. A su vez, cada neurona posee conexiones directas con todas las neuronas de la capa anterior. Cuando existen conexiones entre todas las neuronas de la capa y la capa , la capa es llamada capa densa. Cada neurona lleva a cabo cierto cálculo y produce como salida un valor que es luego emitido a hacia todas sus conexiones salientes. Cada conexión tiene un peso que corresponde a cuán fuertemente están conectadas dos neuronas. Estos pesos son los parámetros de las MLP, y más particularmente, de las capas densas, que deben ser ajustados por el algoritmo de aprendizaje. Típicamente, la computación que lleva a cabo cada neurona está separada en dos etapas: primero realiza la suma ponderada de todas sus conexiones entrantes y luego aplica una función de activación.



*Figura X*

Como ya se ha mencionado, la función aplicada por cada capa puede ser vista como una representación diferente de la salida de la capa anterior. Es interesante analizar esto gráficamente a través de elementos del álgebra lineal. Para ello, es necesario mostrar qué cálculos lleva a cabo cada capa densa. Sean:

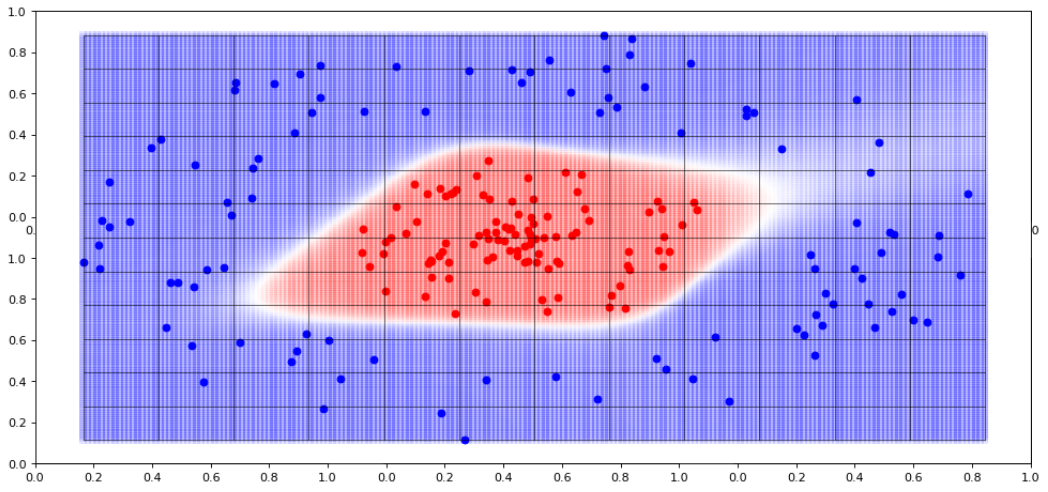
* la dimensión de entrada y la cantidad de neuronas de la capa densa
* el vector de salida la capa , perteneciente a .
* la matriz que contiene todos los pesos entre la capa e , perteneciente a .
* el *bias,* perteneciente a .
* una función de activación no lineal.

Entonces, la salida de la capa es la función :

Donde puede ser interpretada como una matriz de transformación lineal que mapea un vector de a , es un desplazamiento del vector resultante de la transformación lineal, y es aplicada a cada componente del vector resultado. Por lo tanto, .

En al figura X puede observarse un ejemplo de red MLP, compuesta por 5 capas, la capa de entrada, la capa de salida, y 3 capas ocultas o *hidden layers.* Los vectores de entrada de la red pertenecen a , los de salida a y las dimensiones de entrada de las capas ocultas pertenecen a . Los pesos -es decir, la matriz - están representados en la figura a partir de las líneas que unen a las neuronas de dos capas adyacentes. El color rojo significa un valor positivo del peso, mientras que el azul un valor negativo. Cuanto más visible es la línea entre dos neuronas, mayor es el valor absoluto del peso. La red neuronal del ejemplo puede estar representando a un modelo ya entrenado o a uno sin entrenar, donde los pesos fueron inicializados aleatoriamente.

A modo de visualizar lo explicado anteriormente, implementé una *Jupyter Notebook*[[3]](#footnote-2) en la cual se entrena a un MLP simple cuya tarea es la de clasificar un dataset artificial formado por 2 clases de puntos concéntricos pertenecientes a . Una vez que el modelo ha sido entrenado, puede observarse la predicción para cada punto del espacio, junto a su confianza, como muestra la figura. Pero lo interesante es que pueden obtenerse las salidas de las capas intermedias y verse, paso a paso, cómo se van llevando a cabo las transformaciones y cuáles son las representaciones de cada capa. En la ***FIGURA***, la primera fila muestra las transformaciones lineales de los vectores generados por la capa anterior más un desplazamiento, mientras que las columnas muestran el resultado de la aplicación de la función de activación (en este caso la función es sigmoide). Debe notarse que, como las transformaciones aplicadas por cada capa se muestran de forma acumulativa, los espacios no pertenecen a la salida de una transformación lineal. Por el tipo de dataset, las capas internas de la red entrenada tiene 3 neuronas y no 2. Por lo tanto, todas las representaciones que se muestran son parciales, es decir, se grafican 2 de las 3 dimensiones. Sin importar cual sea el dataset, la MLP intentará hallar un hiperplano que separe ambas clases[[4]](#footnote-3). Como en este caso no existe una recta que separe ambas clases, es necesario proyectar el dataset a una dimensión mayor, en cuyo caso la red termina aprendiendo un plano que divide ambas clases. En las 2 dimensiones graficadas, se puede observar que las dos clases son linealmente separables.





Una de las desventajas de utilizar este tipo de redes para el modelado de secuencias es que ignora completamente la topología de la entrada, es decir, *timesteps* y características se representan en la misma dimensión, por lo que no es posible “obligar” a la red a que tenga en cuenta estas representaciones. Dicho de otra manera, las variables de entrada podrían ser presentadas en cualquier orden sin afectar el resultado del entrenamiento. En este trabajo se consideró dicha arquitectura con el fin de evaluar si utilizar arquitecturas que tengan en cuenta la estructura secuencial de los datos, representa o no una mejora en la precisión de los modelos.

## Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes -o RNN por sus siglas en inglés- son modelos donde la entrada de la red son secuencias. Las unidades de las RNN mantienen un estado interno que es propagado a través del tiempo. Dicho estado interno funciona como la memoria de las unidades y puede actuar como una representación de todo aquello que la red haya visto en la secuencia de entrada hasta el momento. En síntesis, las RNN tiene como entrada un vector que representa una secuencia (por ejemplo: una oración) y las neuronas o unidades procesan cada elemento de dicho vector uno a la vez e iterativamente [Elman, 1990]. Las RNN son representadas computacionalmente mediante grafos cíclicos, es decir que poseen ciclos. Dichos ciclos representan la influencia del valor actual de una variable en su valor en un momento futuro.

Las RNN han ganado mucha popularidad por su buen desempeño en tareas relacionadas al modelado del lenguaje (Hermans y Schrauwen 2013) y traducción de máquina [(Mahata et al., 2019)](https://paperpile.com/c/FQGDmo/OEcq). Desgraciadamente, las arquitecturas más simples de RNN son difíciles de entrenar (Pascanu et al. 2013), ya que presentan problemas como *vanishing gradient* y *exploding gradient* [*(Bengio et al., 1994)*](https://paperpile.com/c/FQGDmo/NwkN), por lo que en la práctica se utilizan arquitecturas más complejas que logran sortear en cierta medida dichos problemas. Estas variantes utilizadas son las LSTM [(Hochreiter & Schmidhuber, 1997)](https://paperpile.com/c/FQGDmo/XGNz) y las GRU [(Cho et al., 2014)](https://paperpile.com/c/FQGDmo/aBhN) que añaden mayor cantidad de información a cada neurona en comparación con las RNN más simples, como las *forget gate,* que son parámetros que se aprenden cuando se entrena el modelo y deciden que parte del vector de entrada será descartado.

Más allá del tipo de neurona que se use en una RNN, el mecanismo es similar a todas ellas. Como ya se ha dicho, la entrada de cada neurona es una secuencia que es procesada iterativamente, es decir, un elemento de la secuencia a la vez. Cada neurona tiene un estado interno que va cambiando a medida que la secuencia es procesada (más o menos complejo dependiendo del tipo de neurona recurrente), emulando la memoria de la red. Es importante señalar aquí una diferencia clara entre las RNN y el tipo de arquitectura explicada en la subsección anterior, las MLP. Como ya ha sido explicado, las unidades de las RNN reciben y procesan la información de manera iterativa. De esta manera, el orden en que la secuencia de entrada es procesada es importante y debe respetar su orden natural (por ejemplo, el del tiempo, o de izquierda a derecha en el caso del procesamiento de texto) ya que la red está diseñada para sacar provecho de dicho orden. Si se compara este punto con las MLP, puede verse que, en ellas, cada neurona recibe toda la información “al mismo tiempo”, impidiendo que pueda tomarse en cuenta el orden secuencial. De esto se desprende otra importante diferencia: las RNN comparten los mismo parámetros para cada componente de la secuencia, a diferencia de las MLP que poseen parámetros diferentes para cada parte de ella [(Goodfellow et al., 2016)](https://paperpile.com/c/FQGDmo/ZFoL).



## Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales -CNN, por sus siglas en inglés- son muy comunes en el procesamiento de imágenes y presentan varias diferencias con respecto a las MLP. Por un lado, las CNN toman como entrada datos claramente estructurados, en los que las variables cercanas están fuertemente correlacionadas (por ejemplo, imágenes). Por otro lado, este tipo de redes reduce en gran medida el número de parámetros a entrenar, ya que éstos son compartidos por todas las neuronas de una determinada capa. Estos parámetros son llamados filtros, y a partir de aplicar la convolución a cada sector de los datos de entrada (sean de la entrada de la red o de una capa intermedia), extraen características locales que luego son combinadas sucesivamente generando características globales [(Lecun & Bengio, 1995)](https://paperpile.com/c/FQGDmo/BoTl). Por lo tanto, dichos filtros pueden ser interpretados como características que crecen en complejidad con cada capa de la red. Comúnmente, entre cada capa de convolución se aplica un método llamado *pooling* que reduce la cantidad de neuronas y, por lo tanto, la resolución de los datos procesados, obteniendo invariancia ante la translación y mejorando la capacidad de generalización de la red.

Las CNN han sido utilizadas con éxito en tareas de predicción de series de tiempo en diversas investigaciones [16, 17, 18]. Más recientemente, este tipo de redes, han mostrado un rendimiento del estado del arte en tareas que eran dominadas por las Redes Neuronales Recurrentes (RNN), como las síntesis de audio, modelado de lenguaje a nivel de palabra, traducción de máquina [(Dauphin et al., 2017; Kalchbrenner et al., 2016; van den Oord & Dieleman, n.d.)](https://paperpile.com/c/FQGDmo/tNCm+RfyA+pq2Q). A partir de estos trabajos, se han implementado diferentes tipos de CNN para el modelado de secuencia, y, en especial, para la predicción en series de tiempo.

A related idea is the use of convolution across a 1-D temporal sequence. This convolutional approach is the basis for time-delay neural networks (Lang and Hinton, 1988; Waibel et al., 1989; Lang et al., 1990). The convolution operation allows a network to share parameters across time but is shallow. The output of convolution is a sequence where each member of the output is a function of a small number of neighboring members of the input. The idea of parameter sharing manifests in the application of the same convolution kernel at each time step. Recurrent networks share parameters in a diﬀerent way. Each member of the output is a function of the previous members of the output. Each member of the output is produced using the same update rule applied to the previous outputs. This recurrent formulation results in the sharing of parameters through a very deep computational graph.

## Redes Temporales Convolucionales

Las redes temporales convolucionales -o TCN, por sus siglas en inglés-, son un tipo de CNN con ciertas características que la hacen especialmente aplicables a tareas donde la entrada del modelo es una secuencia. [(Bai et al., 2018)](https://paperpile.com/c/FQGDmo/XcA5) realizan una evaluación sistemática de arquitecturas convolucionales (TCN) y recurrentes. Específicamente, se compara ambos tipos de modelos en tareas que han sido históricamente utilizadas como *benchmark* para comparar diferentes arquitecturas RNN. Los resultados muestran que la común asociación entre el modelado de secuencias y las RNN debe ser reconsiderada, ya que fueron las TCN las arquitecturas que alcanzaron el mayor desempeño en los *benchmarks*. Las TCN ya habían sido utilizadas en otros trabajos [(Lea et al., 2017; van den Oord & Dieleman, n.d.)](https://paperpile.com/c/FQGDmo/tNCm+VXWI) con ciertas variaciones. En [(Bai et al., 2018)](https://paperpile.com/c/FQGDmo/XcA5) se implementa una versión de una TCN genérica. En dicha versión, se busca que las TCN puedan aprovechar las ventajas que presentan las CNN más simples, pero siendo adaptadas para el modelado de secuencias y la predicción de series de tiempo. Estos cambios introducen ciertas diferencias entre las TCN genéricas de [(Bai et al., 2018)](https://paperpile.com/c/FQGDmo/XcA5) y las CNN tradicionales. Las diferencias más importantes son:

• Las TCN no utilizan capas de pooling, de forma que la dimensión de entrada es igual a la de

salida.

• Las TCN siempre aplican la convolución de una sola dimensión (la del tiempo).

• Las convoluciones llevadas a cabo por las TCN son causales, es decir, las convoluciones nunca consideran información del futuro.

• Utilizan dilataciones para aumentar el campo receptivo exponencialmente y así

poder cubrir largas dependencias en los datos secuenciales.

Las TCN propuestas en [(Bai et al., 2018)](https://paperpile.com/c/FQGDmo/XcA5) están compuestas por capas convolucionales densas, esto es, todas las capas de la red son convolucionales, con excepción de las capas de activación y aquellas utilizadas para regularización. Las TCN están organizadas en bloques residuales. Cada bloque residual está compuesto por dos bloques formados por: una capa convolucional causal con tamaño de kernel y una dilatación , una capa de activación Relu, una capa de regularización - la técnica de regularización por normalización utilizada varía entre *Batch Normalization, Layer Normalization y Weight Normalization* según la implementación, aunque en [Bai et al] se propone la última-*,* yun *Dropout* espacial. La diferencia entre una capa de Dropout normal y una espacial es que la última le da el valor de 0 a mapeos de característica enteras, es decir, al resultado completo de la aplicación de un filtro sobre la entrada. Luego la salida de cada bloque utiliza conexiones residuales en las que la entrada y la salida se suman. Esta suma puede no ser posible si la entrada y la salida del bloque residual tienen dimensiones diferentes, en cuyo caso, se le aplica al vector de entrada una convolución de dimensión 1x1 para hacer coincidir las dimensiones de la entrada y la salida. Esta inconsistencia entre las dimensiones de entrada y la salida de los bloques residuales solo se da en la primera entrada, es decir, en el input de la red, ya que, todas capas internas tienen el mismo tamaño y en todas ellas se aplica zero *padding*.

Un concepto muy importante para las TCN es el de campo receptivo, o *receptive field,* que indica la cantidad de componentes de la entrada a los que el filtro de una capa convolucional arbitraria tiene acceso. Esto es, el campo receptivo indica de cuantos componentes de la entrada un filtro está utilizando información al llevar a cabo la convolución. A diferencia de las CNN comunes, donde el campo receptivo aumenta linealmente con la cantidad de capas convolucionales -aunque esto puede variar a partir del uso de técnicas como *stride* o *max-pooling-* [*(Goodfellow et al., 2016)*](https://paperpile.com/c/FQGDmo/ZFoL) el uso de las dilataciones en las TCN permite que el campo receptivo aumente exponencialmente con cada bloque residual. Esto se da gracias a que las dilataciones se determinan a partir de potencias de 2 (por ejemplo: 1, 2, 4, 8, etc.). Esto permite que las TCN puedan tener una historia larga con pocas capas computacionales, pudiendo reducir la cantidad de parámetros necesarios a ajustar.

En las TCN, el campo receptivo es un hiperparámetro a ajustar. Este hiperparámetro no se ajusta directamente, sino que varía a través de la cantidad de dilataciones y el tamaño del filtro. Es importante que el campo receptivo sea lo suficientemente largo como para cubrir toda la secuencia de entrada [(Bai et al., 2018)](https://paperpile.com/c/FQGDmo/XcA5).

Hay diversas ventajas de las TCN sobre las RNN que se suman al mejor desempeño en diferentes *benchmark* [*(Bai et al., 2018)*](https://paperpile.com/c/FQGDmo/XcA5):

* A diferencias de las RNN donde las predicciones para un *timestep* futuro tienen que esperar a que se hayan llevado a cabo todas las predicciones anteriores, las convoluciones pueden llevarse a cabo en paralelo dado que se utiliza el mismo filtro para todas ellas. Esto permite que tanto el entrenamiento como la evaluación de la red pueda llevarse a cabo más rápidamente.
* Es sencillo ajustar el campo receptivo de las TCN a partir de la variación del tamaño del filtro y el valor de las dilataciones, lo que permite un mejor control del tamaño de la memoria del modelo, permitiendo la fácil adaptación para diferentes dominios.
* Como en las TCN el camino de *backpropagation* es diferente de la dirección temporal de la secuencia, éstas no sufren de los llamados problemas *exploding/vanishing gradient.*
* Al entrenarse, las RNN deben mantener en memoria los resultados parciales de cada capa, pudiendo llegar a requerir mucha memoria, sobretodo para secuencias largas. Este hecho no ocurre en las TCN porque los filtros son compartidos por toda la capa.
* Las TCN pueden, al igual que la mayoría de las RNN, recibir secuencias de longitud variable.

En [(Bai et al., 2018)](https://paperpile.com/c/FQGDmo/XcA5) también se muestran dos desventajas de la utilización de las TCN:

* En el momento de la evaluación, las TCN deben mantener en memoria la secuencia entera para realizar una predicción. En cambio, las RNN solo necesitan mantener en memoria un *timestamp* de la totalidad de la secuencia.
* Si se desea cambiar de dominio a partir de un modelo ya entrenado, por ejemplo para realizar *fine-tuning,* el campo receptivo requerido por la nueva tarea a realizar puede ser mayor que el campo receptivo del modelo ya entrenado, por lo que el modelo puede no tener un buen desempeño en la nueva tarea.

## Predicción de actividades

La predicción de actividades es un área de investigación que consiste en hipotizar información sobre las actividades en las que un sujeto va a estar involucrado en el futuro basándose en datos de sensores. No debe confundirse al área de predicción de actividades con el de reconocimiento de actividades. La diferencia entre ambas es que, en la segunda, se utilizan datos de sensores tanto del pasado como del presente y se los mapea con con un tipo de actividad que está ocurriendo en el momento actual. Mientras que en la primera no se tiene ningún dato de sensor sobre la actividad que va a llevarse a cabo en el futuro y que es la que se se quiere predecir. La predicción de actividades puede ser utilizada, por ejemplo, para anticipar las actividades de una persona en una casa, facilitando la automatización de dispositivos IOT, como calentando la casa antes de que la persona llegue del trabajo.

En el capítulo 7 de  *Activity learning Discovering, recognizing, and predicting human behavior from sensor data* [*(Cook & Krishnan, 2015)*](https://paperpile.com/c/FQGDmo/DZqSa) se le da un enfoque al área de Predicción de Actividades basado en algoritmos de compresión para la predicción de actividades. Más específicamente, los algoritmos de compresión que se propone utilizar son LZ78 y ALZ. Ambos algoritmos pueden ser interpretados como tipos de modelos de Markov. En este trabajo, se analizan las ventajas de cada unos de estos algoritmos para la predicción de secuencias de una manera general. Es decir, se generalizan para la predicción de símbolos, sean cuales sean (por ejemplo, símbolos que representan ubicaciones).

Similarmente a lo expuesto anteriormente, la predicción de actividades también puede ser estudiada como la predicción de cuando cierta actividad va a volver a suceder. En este caso, a diferencia de predecir el siguiente elemento de una secuencia, se busca hallar la cantidad de unidades de tiempo luego de la cual cierta actividad se repetirá.

## Trabajos relacionados

En este capítulo se introducen los trabajos relacionados al problema abordado en esta tesis. La predicción del comportamiento sedentario futuro es un problema complejo y puede ser descompuesto en diferentes aristas que lo caracterizan. A fin de tener un paneo general de cada una de esas aristas, las primeras 3 subsecciones muestran los trabajos relacionados a cada una de ellas. Mientras que en la subsección 4 se muestran los trabajos relacionados que abordaron precisamente el problema de la predicción del comportamiento sedentario futuro.

## Monitoreo y modelado del comportamiento humano utilizando smartphones

A partir de que los dispositivos móviles y “*wearables*”, como *smartphones*, comenzaron a volverse populares, se desarrollaron numerosos estudios para encontrar los patrones de uso que permiten relacionar, inferir y predecir diferentes tipos de comportamientos humanos en el contexto de la salud y el bienestar.

Harari et al. [(Harari et al., 2017)](https://paperpile.com/c/FQGDmo/VGeQ), en su investigación, parte del hecho de que los recientes avances en tecnología de sensado móvil revolucionaron la evaluación del comportamiento al permitir el seguimiento no obstrusivo y continuo del comportamiento a través de dispositivos móviles (como acelerómetros y micrófonos) incluidos en smartphones.

Para examinar la viabilidad de usar métodos móviles de sensado para obtener estilos de vida diarios de comportamiento de estudiantes, los investigadores presentaron un estudio utilizando un dataset generado por una aplicación de sensado en smartphones llamada *StudentLife* para medir, de forma objetiva, la actividad natural de estudiantes y comportamientos sociales a lo largo de un ciclo académico de 10 semanas de duración. Al hacerlo, abordaron una brecha existente en la literatura de *mHealth* al proporcionar una descripción de los patrones de granularidad fina de estabilidad y cambio que caracterizan las conductas relacionadas con la salud de los estudiantes durante un período académico.

Los resultados de este estudio indican que, a medida que el ciclo académico progresa, los estudiantes tendían a participar en más comportamientos sedentarios y relacionarse menos con otros estudiantes, lo que puede estar relacionado a la necesidad de concentrarse en estudiar y preparar exámenes.

En conclusión, Harari et. al. demostraron la viabilidad de utilizar métodos objetivos de sensado basados en smartphones para realizar el seguimientos de comportamientos relacionados a la salud en el contexto de la vida cotidiana de estudiantes. Los investigadores identificaron varias correlaciones significativas entre los datos objetivos de los sensores de los dispositivos móviles y el bienestar mental y el rendimiento académico.

Gong et al. [(Gong et al., 2019)](https://paperpile.com/c/FQGDmo/TxFr), en una colaboración entre ingenieros y psicólogos, utilizaron los datos del acelerómetro y del GPS de smartphones en los momentos antes, durante y después de realizar llamadas y enviar mensajes de texto. La hipótesis de esta investigación fue que los movimientos sutiles (medidos con acelerómetros) covarían con el nivel de ansiedad social y el contexto social. Por ejemplo, las manos pueden temblar más cuando el estudiante está en un ambiente público, debido a que la probabilidad de evaluación negativa por otros es alta. Los autores denominan a estos movimientos como comportamientos dinámicos del usuario.

Para realizar el estudio, 52 estudiantes, con edad media de 20.5±3.8, fueron reclutados del grupo de participantes del departamento de psicología de la Universidad de Virginia. De cada participante se tomaron los datos del acelerómetro, el GPS y datos relacionados a las llamadas y a los mensajes de texto. Luego, modelaron los datos como un sistema lineal dinámico. Para ello, primero utilizaron Foursquare [[5]](#footnote-4)para obtener los Puntos de Interés a partir de los datos crudos del GPS de cada participante, y luego, los dividieron en sus locaciones semánticas (hogar, transporte, vida personal, lugar de comida). Respecto a los datos sociales (llamadas y mensajes de texto), los autores analizan por separado períodos de observación minutos antes y después de una llamada, para detectar, por ejemplo, cómo varían los indicadores de ansiedad de un sujeto cuando debe realizar una llamada y no recibe una respuesta a la llamada realizada. Además, antes de que comenzase el experimento, cada estudiante fue evaluado de acuerdo a la Escala de Ansiedad de Interacción Social (SIAS, por sus siglas en inglés).

Los resultados obtenidos en el estudio mostraron una relación estadística significativa entre los datos de los sensores de smartphones y la ansiedad social de los estudiantes. Estos resultados sugieren que es posible integrar datos de acelerómetros y de GPS para identificar marcadores de comportamiento asociados con la ansiedad social en varias locaciones. Por lo anterior, este trabajo abre nuevas posibilidades para monitorear pasivamente marcas de comportamiento de ansiedad social a través de la integración de los datos del acelerómetro y los de GPS.

Kanjo et al. [(Kanjo et al., 2019)](https://paperpile.com/c/FQGDmo/zeAi) adoptó un enfoque de Deep Learning para la clasificación de emociones a partir de datos de diferentes tipos de sensores portables. La clasificación de emociones a menudo requiere del modelado de tipos de datos de entrada de diferentes modalidades, como señales fisiológicas, datos ambientales, video y datos de movimiento y ubicación. Este problema ya había sido abordado desde la utilización de algoritmos del área de Machine Learning, pero aunque los procesos de ingeniería de características a menudo integrados en estos algoritmos son beneficiosos para la modelización de emociones, heredan algunas limitaciones críticas que pueden dificultar el desarrollo de modelos fiables y precisos.

En este trabajo, los investigadores adoptaron un enfoque de aprendizaje profundo para la clasificación de emociones a través de un proceso iterativo mediante la adición y eliminación de un gran número de señales de sensores de diferentes modalidades. El conjunto de datos fue recogido en un estudio del mundo real desde teléfonos inteligentes y dispositivos portátiles. Dicho conjunto de datos combina la interacción local de tres modalidades de sensores: en el cuerpo, el medio ambiente y la ubicación en un modelo global que representa la dinámica de la señal junto con las relaciones temporales de cada modalidad. El enfoque propuesto emplea una serie de algoritmos de *Deep Learning* que incluyen un enfoque híbrido usando la Red Neuronal Convolucional y la Red Neuronal Recurrente de Memoria Corta a Largo Plazo (CNN-LSTM) en los datos crudos del sensor. Utilizando los datos crudos del sensor como entrada para las redes neuronales se consigue eliminar las necesidades de extracción manual de característica. Los resultados muestran que la adopción de enfoques de aprendizaje profundo es efectiva en la clasificación de la emoción humana cuando se utiliza un gran número de sensores de entrada y que los modelos híbridos superan a la red neuronal profunda tradicional totalmente conectada. Además, los modelos híbridos superan los algoritmos de tipo *Ensemble* previamente desarrollados que utilizan la ingeniería de características para entrenar el modelo

Wu et al. [(Wu et al., 2018)](https://paperpile.com/c/FQGDmo/H0Qg) utilizaron redes de encuentro basadas en los registros de encuentros Bluetooth para monitorear el nivel de stress de los usuarios. Bluetooth es una red de corto alcance comúnmente disponible en los dispositivos móviles. Al mismo tiempo, la red Bluetooth puede ser utilizada como un sensor móvil ampliamente disponible que puede detectar el entorno social de un individuo basado en la proximidad física. Los autores señalan que en la bibliografía existe una falta de ingeniería sistemática de características, así como también evidencia de su poder predictivo en problemas de salud mental. En este trabajo se extrajeron características de los datos de encuentro de Bluetooth de los teléfonos inteligentes de 49 estudiantes a lo largo de 66 días, provenientes del *dataset* StudentLife. El análisis de correlación sugiere que las características generadas pueden explicar la varianza en el stress significativamente mejor que los modelos nulos.

En este trabajo los autores evaluaron la predictibilidad del nivel de stress que podía conseguirse utilizando las redes de encuentro de Bluetooth, más otros datos tales como la calidad del sueño y otros datos de sensores como GPS, acelerómetro, uso del teléfono, etc. Sus predictores pueden ser clasificados de acuerdo diferentes características: predecir el nivel de stress o el cambio del mismo, utilizar la técnica de evaluación Leave-One-Subject-Out (LOSO) o Leave-one-observation-out (LOOO), predecir el presente o el futuro. Los modelos utilizaron la técnica Random Forest en todos los casos. Los resultados indican que las características extraídas de las redes de encuentro representan una mejora en términos de predictibilidad del nivel de stress y del cambio de éste en el presente, pero no en el futuro. Estos resultados proveen evidencia del valor potencial de mejora al incorporar las redes de encuentro basadas en datos de redes Bluetooth a aplicaciones de monitoreo de stress.

Zia et al. [(Zia et al., 2016)](https://paperpile.com/c/FQGDmo/SRc7) destacan que utilizar los datos de dispositivos portátiles puede ayudar a que pacientes con Parkinson vivan de forma más segura e independientemente. Partiendo de esta idea, el objetivo de este trabajo consistió en determinar la viabilidad de utilizar redes neuronales para predecir la congelación de la marcha, que es un síntoma del Parkinson. Particularmente, los investigadores usaron una clase de red neuronal conocida como Redes Recurrentes en capas (LRN). El dataset utilizado para entrenar las redes fue uno público donado por el Repositorio de Aprendizaje Profundo de la Universidad de California y comprende un registro de los datos de movimiento (3 acelerómetros) de 3 sujetos, donde los momentos en los cuales se predice el Congelamiento de la Marcha están anotados como un timestamp. El trabajo no especifica las características obtenidas a partir de los datos del acelerómetro, por lo que se supone que utilizaron un modelo end-to-end, en el que al igual que en otros trabajos ya expuestos, son las primeras capas de las redes quienes se ocupan de aprender a generar las características. Los parámetros que fueron variando en las diferentes redes neuronales testeadas fueron la cantidad de capas, las cantidad de neuronas por capas, el tamaño de *batch* y el factor de down sampling -esto es, la cantidad de ejemplos de entrenamiento que se quitaron del dataset-.

Los resultados mostraron que las redes que mejor precisión y recall tuvieron son aquellas de una sola capa, baja cantidad de neuronas y alto down sampling. Los autores propusieron, como trabajo futuro, utilizar otros tipos de datos para que se complementen con los datos de movimiento.

A partir de los trabajos antes expuestos es posible notar el potencial que tienen los dispositivos ubicuos equipados con diferentes tipos de sensores, como lo son los *smartphones*, para permitir el monitoreo y modelado del comportamiento de los usuarios que los portan. Estos tipos de comportamiento pueden ser de lo más variados, desde cuestiones relacionadas a la salud mental hasta otras relacionadas a la salud física. La gran disponibilidad de datos heterogéneos y de diferentes poblaciones (por ej: un grupo de estudiantes) ofrece una clara oportunidad para la evaluación de diferentes técnicas del área de *Deep Learning*.

En esta tesis, motivado por lo expuesto en el párrafo anterior, se busca sacar provecho de la ubicuidad de los dispositivos móviles, la heterogeneidad de sus sensores y la facilidad para almacenar y acceder a los datos producidos por ellos. Para ello, se toma el *dataset StudentLife-* utilizado también en otros trabajos, como en Harari et al. [(Harari et al., 2017)](https://paperpile.com/c/FQGDmo/VGeQ) - para procesar los datos disponibles y generar modelos que representen a los usuarios, buscando permitir, a partir de ellos, predecir el comportamiento sedentario futuro a través de su gasto energético.

## Redes Neuronales y series temporales

En esta subsección de los trabajos relacionados, se muestran y analizan una serie de trabajos que utilizan algoritmos del área de Deep Learning para tareas donde los datos disponibles a partir de los cuales es posible entrenar a los algoritmos tienen una fuerte componente temporal. Como ya se ha mostrado en el Marco Teórico, diferentes algoritmos de Deep Learning han ganado popularidad en los últimos años por su gran desempeño alcanzado en tareas donde ha sido posible utilizar explícitamente la componente temporal de los datos. Estos algoritmos han demostrado sistemáticamente un mayor desempeño que los algoritmos tradicionales del área de Machine Learning, como *Support Vector Machines* o *Random Forest*. A continuación se exponen algunos de los trabajos de este tipo, donde busca hacerse hincapié en algoritmos de Deep Learning aplicados aseries temporales.

En Pelletier et al. [(Pelletier et al., 2019)](https://paperpile.com/c/FQGDmo/qKiY) se utilizan Series de Tiempo de Imágenes Satelitales (SITS, por sus siglas en inglés) en sistemas de clasificación que apuntan a obtener mapas actualizados y precisos de la cobertura terrestre de las superficies de la Tierra. La tarea de clasificación en este trabajo fue la de clasificar imágenes SITS, la cobertura terrestre de diferentes porciones de mapa en el este de Toulouse, Francia. En este trabajo se expresa el buen desempeño de métodos de clasificación tradicionales y se propone el uso de métodos de Deep Learning para mejorar dicho desempeño. Más específicamente, los investigadores estudiaron el aporte de las CNN y RNN en las tareas donde la componente temporal es aprovechada y propusieron el uso de CNN temporales. Las CNN temporales propuestas aplican la convolución de una dimensión, la temporal, por lo que pueden denotar como 1D-CNN. En resumen, en este trabajo se compararon modelos que no utilizan explícitamente el carácter secuencial (*Random Forest* y *Support Vector Machines),* RNN, y diferentes tipos de CNN cuyos filtros varían en relación de en qué dimensión se lleva a cabo la convolución. En el caso de las imágenes satelitales la convolución se puede llevar a cabo en la dimensión temporal, en la dimensión espectral o en ambas. Los investigadores compararon también estos tres casos. Los resultados mostraron que las 1D-CNN que toman en cuenta ambas dimensiones obtuvieron un mejor desempeño en la tarea de clasificación llevada a cabo.

En Yan and Ouyang [(Yan & Ouyang, 2017)](https://paperpile.com/c/FQGDmo/FOiI) exploraron la aplicabilidad y validez de aplicar redes LSTM a series de tiempo financieras. En particular, abordaron la tarea de predecir el precio de cierre del Índice Compuesto de Shanghái. Para llevar a cabo esta tarea, los investigadores tomaron datos históricos desde enero del 2012 hasta junio del 2017. A partir de esos datos, diferentes índices fueron computados para cada día, como el precio de cierre del dia anterior, el precio de apertura del día anterior, el mayor precio del día anterior, etc. Luego, a la variable objetivo se le fue aplicada una transformación que permite reducir el ruido en la serie de tiempo, con el fin de aumentar la capacidad de generalización de los modelos, ya que impide que aprenda a partir de una serie de tiempo ruidosa. Finalmente, los modelos LSTM fueron comparados con otros modelos más sencillos, como Perceptrones Multicapa, *Support Vector Machine* y  *K-nearest neighbors*. Los modelos LSTM obtuvieron una mejor desempeño en comparación con los otros modelos gracias a su mejor capacidad para capturar las características no lineales de las series de tiempo financieras.

En Wang et al. [(“Time Series Prediction for the Epidemic Trends of COVID-19 Using the Improved LSTM Deep Learning Method: Case Studies in Russia, Peru and Iran,” 2020)](https://paperpile.com/c/FQGDmo/C6fF) abordaron la tarea de modelar la tendencia epidémica a largo plazo de la epidemia COVID-19 mediante el uso de modelos LSTM. El estudio se realizó a partir de 3 países situados en 3 continentes distintos: Rusia, Perú e Irán. Los modelos fueron entrenados utilizando los datos proveídos por la Universidad John Hopkins. Los investigadores tomaron los registros disponibles desde el 22 de enero al 7 de julio de 2020. Se utilizaron los modelos LSTM para predecir la curva del total de casos confirmados por el virus COVID-19 para los 150 días siguientes. Al ser pocas la cantidad de casos de estudio para entrenar el modelo los investigadores utilizaron el método de optimización continua, mediante el cual los casos ya predichos son utilizados como nuevos casos de entrenamientos. Wang et. al reportan que los resultados de las predicciones son altamente consistentes con los casos positivos diarios notificados, lo que demuestra que el método propuesto puede analizar con precisión la tendencia de la epidemia.

## Predicción del comportamiento sedentario futuro

El comportamiento sedentario puede ser clasificado como un tipo de tarea de predicción de actividades (deberia llegar a que es un tipo de cada una de las secciones de trabajo relacionado anteriormente). En esta subsección se presentan los trabajos relacionados al área en la que se contextualiza esta tesis, es decir, la predicción del comportamiento sedentario.

Cabe aclarar que los trabajos que buscan resolver el problema de predecir el comportamiento sedentario futuro, son escasos y hacen suposiciones diversas a la hora de tratar de resolverlo.

Para realizar la búsqueda de trabajos relacionados a la predicción de comportamiento sedentario futuro se utilizaron las librerías *IEEE Xplore[[6]](#footnote-5)*, *ACM*[[7]](#footnote-6), *Wiley[[8]](#footnote-7)*, *ScienceDirect[[9]](#footnote-8)* y *Scopus[[10]](#footnote-9)* utilizando los motores de búsqueda propios de cada una. Hasta el momento, Q. He and E. Agu son los únicos autores que pueden considerarse como trabajos relacionados a la predicción del comportamiento sedentario a partir de los resultados de dicha búsqueda. A continuación se analizan todos los trabajos de los ya nombrados autores, quizás con más detalle debido a la relación cercana con el trabajo motivo de esta tesis.

A diferencia del enfoque propuesto en el presente trabajo, Q. He y E. Agu determinaron el nivel de sedentarismo como el porcentaje de registros de actividad estacionarios que se tomaron de los datos de actividad física de cada usuario en cada hora, con respecto a los demás tipos de registros de actividad (corriendo, caminando, etc.).

En este primer estudio Q. He y E. O. Agu [(He and Agu 2016)](https://paperpile.com/c/FQGDmo/axGRS) propusieron un algoritmo de dominio de frecuencia para detectar patrones sedentarios recurrentes a partir de los datos de los usuarios. Este trabajo buscó ajustar funciones periódicas (seno, coseno) al porcentaje de registros de actividad estacionaria de cada hora y de cada usuario, tratando así de identificar los patrones subyacentes en su actividad física.

En este trabajo, como en los demás, los autores utilizaron el *dataset StudentLife* para validar sus hipótesis. Con respecto al procesamiento de los datos, primero, discretizaron los datos de actividad física en buckets de distintas escalas de tiempo, ya que, en dicho dataset, cada registro está acompañado de un timestamp. Luego, para cada bucket, computan lo que ellos llaman nivel de sedentarismo, que definen como el porcentaje del total de actividades físicas etiquetadas como estacionarias pertenecientes a ese determinado bucket.

A partir del nivel de sedentarismo computado, los investigadores llevaron a cabo un análisis sobre el nivel de sedentarismo a lo largo de las horas del día y de los días de la semana, que les permitió observar en qué momentos del día los estudiantes eran más sedentarios y cuánto difieren entre sí los patrones de comportamiento sedentario de los diferentes estudiantes y a lo largo del ciclo lectivo.

En base a esto, los autores propusieron un modelo computacional capaz de capturar los patrones recurrentes a diferentes escalas de tiempo. Estos modelos utilizan el nivel de sedentarismo considerándolo como una señal periódica, donde cada elemento de dicha señal corresponde a un bucket de un determinado usuario. Para generar los modelos, los autores aplicaron una transformación de *Fourier* sobre la señal periódica, descomponiendola en una suma de funciones coseno y tomando las primeras funciones con el fin de minimizar el error cuadrado y eliminar el ruido. Los modelos son generados para cada uno de los usuarios y a diferentes escalas de tiempo, es decir, con diferentes tamaños de bucket.

Los investigadores obtuvieron resultados muy variados de acuerdo a los diferentes usuarios y escalas de tiempo utilizadas. Algunos modelos obtuvieron un error cuadrado muy bajo lo que significó que el usuario al cual corresponde el modelo presenta comportamientos sedentarios de forma recurrente y periódica. Sin embargo, otros modelos no lograron conseguir un error cuadrado bajo, lo que mostró que el usuario al cual corresponde el modelo no presenta comportamientos sedentarios de forma recurrente y periódica.

En un segundo estudio, He y Agu [(He & Agu, 2016b)](https://paperpile.com/c/FQGDmo/GSfPw) resaltaron que los teléfonos inteligentes pueden monitorear los comportamientos sedentarios llevados a cabo por sus usuarios, así como también pueden monitorear el contexto en que dichos comportamientos sedentarios ocurren. En este trabajo, los investigadores exploraron si es posible utilizar los datos de contexto para predecir el comportamiento sedentario futuro.

Para probar sus hipótesis, He y Agu utilizaron el *dataset* público *StudentLife*, que contiene datos de actividad física y datos del contexto de 49 estudiantes universitarios a lo largo de 10 semanas. El *dataset StudentLife* ha sido utilizado por todos los trabajos de investigación que se encuentran en esta sección. Además, es el dataset utilizado en esta tesis, por lo que su estructura y contenido será detallado en la sección ….

En el *dataset StudentLife*, todo datos, ya sea de actividad física o de contexto están asociados a un *timestamp*. Dependiendo del tipo de dato, éstos fueron muestreados con cierta frecuencia. Con el fin de normalizar los datos disponibles, el paso siguiente que llevaron a cabo los investigadores fue el de discretizar el tiempo en *buckets* de una hora. Es decir, todos los datos de sensado de los usuarios fue asociado a un hora y día específico. En total, He y Agu obtuvieron como resultado 65,601 buckets.

Luego, a partir de los datos de actividad física, los investigadores computan, para cada bucket, lo que ellos llaman nivel de sedentarismo, que definen como el porcentaje de actividades físicas etiquetadas como estacionario. Esta forma de cuantificar el comportamiento sedentario será discutida más adelante en la sección …Además, discretizaron el nivel de comportamiento sedentario en 3 clases, cada una asociada a un rango específico: muy sedentario (99.99-100.00), sedentario (90.52 - 99.99) y menos sedentario (0 - 92.52). Cabe resaltar que dichos rangos no fueron determinados con ningún tipo de base teórica, sino que fueron así definidos para obtener la misma cantidad de registros de actividad para cada clase.

A continuación, los investigadores analizan los datos de contexto que provee el dataset utilizado y su relación con el nivel de sedentarismo computado. A partir de este análisis, los investigadores definen y calculan, en total, 22 variables de contexto asociadas a cada bucket, como el día de la semana y la actividad física que más llevó a cabo un determinado usuario en una hora específica.

Por último, los investigadores intentaron predecir el nivel de sedentarismo futuro, es decir, si en la siguiente hora el usuario iba a ser muy sedentario, sedentario o menos sedentario, a partir de las variables de contexto que definieron. Para realizar esta tarea, y dado que la variable que se intenta predecir es categórica, los autores de este trabajo utilizaron los modelos Naive Bayes y Regresión Logística. Además, para validar los resultados utilizaron la técnica de validación cruzada. Naive Bayes alcanzó una precisión de 59.7% y recall de 64.2%, mientras que Regresion Logistica obtuvo una precisión de 60.8% y recall de 62.6%. Como último experimento, los investigadores agregaron al conjunto de datos de entrenamiento el nivel de sedentarismo actual, mejorando ligeramente los resultados. Como conclusión de estos resultados, los autores resaltaron que para predecir si el usuario iba a ser muy sedentario en la hora siguiente, consiguieron una precisión de 73.1% y recall de 87.7%.

En otro estudio He y Agu [(He & Agu, 2016c)](https://paperpile.com/c/FQGDmo/2NLm5) destacaron que el tiempo, el ritmo diario y los hábitos sedentarios del pasado son 3 determinantes del comportamiento sedentario. Los investigadores propusieron entonces un método para descubrir automáticamente los 3 determinantes analizando los registros de las series de tiempo de datos obtenidos de sensores de *smartphones*. Dicho método analiza el historial de comportamiento de un usuario para encontrar patrones temporales que puedan ser utilizados para predecir el comportamiento sedentario futuro.

En este trabajo, los investigadores suponen que el comportamiento sedentario de los usuarios está correlacionado linealmente en el dominio temporal de tal forma que la probabilidad de que un usuario tenga un comportamiento sedentario en el siguiente período de tiempo depende de su comportamiento actual y pasado. Para hallar estos patrones, los autores utilizan Modelos Autorregresivos junto al Método de Máxima Entropía para estimar los coeficientes de dichos modelos.

Para evaluar los modelos propuestos Q. He y E. O. utilizaron el dataset StudentLife. Dicho dataset dispone de registros de series de tiempo sensados automaticamente, donde cada registro está compuesto de un timestamp y un tipo de actividad (estacionario, caminando, corriendo). Como en los trabajos anteriores, discretizaron las series de tiempo en buckets de una determinada duración, calculando para cada una el porcentaje de registros etiquetados como estacionario. Luego, generaron en total 324 modelos variando el orden del Modelo Autorregresivo (cantidad de información del pasado que el modelo utiliza) y el tamaño de bucket (1, 2, 3 o 6 horas). Por último, calcularon el Error Cuadrado de cada modelo y para cada usuario.

Los resultados mostraron que los modelos que tuvieron un mayor MSE fueron aquellos que utilizaban buckets de 6 horas. El MSE más bajo obtenido fue sobre el usuario número 23, con un modelo autorregresivo de orden 26 y tamaño de bucket de 6 horas. Los investigadores hipotetizaron que esto se debió a que la influencia de factores aleatorios es menor cuanto más grande sea el tamaño de bucket.

En 2017, He y Agu [(He & Agu, 2017)](https://paperpile.com/c/FQGDmo/oye2) publicaron un último trabajo de investigación en el cual plantearon un nuevo enfoque para poder predecir el comportamiento sedentario de una persona. En este trabajo, los autores propusieron modelos que están basados en el concepto de análisis de ritmo, propuesto por Lefebvre, que postula que muchos comportamientos humanos siguen ritmos naturales. Por lo tanto, en este trabajo los investigadores se concentraron en detectar los ritmos predominantes de comportamiento sedentario y en modelar los ritmos cíclicos y los ritmos lineales utilizando funciones cíclicas y funciones lineales, respectivamente. Como ejemplo de ritmo cíclico, los autores ejemplifican con una persona que se sienta en un sillón todos los días a la misma hora. Como ejemplo de ritmo lineal, los autores ejemplifican con una persona que se acuesta exhausta luego de llevar a cabo un ejercicio de alto requerimiento físico.

Por un lado, los autores de este trabajo denominan a los modelos que utilizan funciones cíclicas como libres de la historia (o *history-free*), ya que una vez determinadas las funciones cíclicas no es necesario guardar los datos de gasto energético del usuario para predecir su siguiente comportamiento sedentario. Por otro, los autores de este trabajo denominan a los modelos que utilizan funciones lineales como dependientes de la historia (o *history-dependent*) ya que estos deben guardar los datos de gasto energético del usuario aun cuando el modelo ya ha sido generado.

Para generar los modelos libres de historia los autores utilizan un procedimiento similar al que siguieron en [(He & Agu, 2016a)](https://paperpile.com/c/FQGDmo/axGRS), mientras que para generar los modelos dependientes de historia utilizan un procedimiento similar al que siguieron en [(He & Agu, 2016c)](https://paperpile.com/c/FQGDmo/2NLm5).

A continuación, los autores propusieron modelos híbridos, que combinan los modelos libres de historia y los modelos dependientes de historia. Analiticamente, los modelos hibridos se obtienen como una suma ponderada de los dos tipos de modelos, donde los primeros buscan calcular los patrones causados por ritmos cíclicos, mientras que los segundos buscan capturar los patrones dentro de esos ciclos.

En total He y Agu generaron 49248 modelos híbridos para los 49 participantes del dataset. Para cada uno de estos modelos, los autores debieron hallar los valores óptimos de l (el orden de los Modelos Autorregresivos) y k (la cantidad de funciones periódicas utilizadas). Los resultados mostraron que el modelo de base, donde k vale 0 y l vale 0, fue el óptimo en casi la mitad de los modelos generados, es decir, el que menos MSE obtuvo. Estos resultados contradicen su hipótesis inicial sobre que el comportamiento sedentario puede ser explicado a partir de los ritmos cíclicos y lineales de las personas. Los autores indicaron que este resultado era de esperar debido a las horas en que los usuarios duermen, en las cuales los modelos pueden simplemente predecir el promedio de los resultados de las horas anteriores.

## Predicción de Comportamiento Sedentario Futuro

El problema que se abordará en esta Tesis es el de predecir el comportamiento sedentario futuro mediante modelos de *Deep Learning* entrenados con datos de *Lifelogging*.

El gasto energético es medido en MET, que es la medida estándar en la comunidad científica que estudia la salud en relación con la actividad física. En el ámbito de la salud, se ha llegado a un acuerdo entre los investigadores en determinar cómo actividad sedentaria a toda aquella actividad cuyo MET asociado sea menor o igual a 1,5.

En el contexto de este trabajo, se define la Predicción del Comportamiento Sedentario Futuro (PCSF) como la tarea de predecir el valor de MET de la actividad física realizada porun usuario en un futuro próximo en pos de estudiar su comportamiento sedentario futuro.

En el marco del aprendizaje de máquina, al ser el MET un valor continuo, el problema a resolver es una tarea de regresión. Dentro de los algoritmos de aprendizaje profundo, las Redes Neuronales Recurrentes (RNN), las Redes Neuronales Convolucionales (CNN) y las Redes Temporales Convolucionales (TCN) son especialmente aptas para el problema a tratar porque los datos registrados de la actividad del usuario están organizados en series de tiempo, en las que la secuencialidad es importante. Por lo tanto, todos estos tipos de redes neuronales serán evaluados y comparados.

En este trabajo, se buscará predecir el valor de MET para un usuario en el siguiente intervalo temporal. Es decir, dados los datos del pasado se utilizarán Redes Neuronales para hallar el gasto energético en el tiempo presente. El objetivo principal es evaluar la factibilidad de utilizar redes neuronales capaces de aprender, a partir de una secuencia de datos de la actividad de un usuario, patrones de comportamiento que sean útiles para el problema de PCSF.

Además de evaluar diferentes tipos de redes neuronales, se evaluarán diferentes tipos de modelos. Dichos tipos de modelos determinan, por un lado, el intervalo de tiempo del cual la arquitectura toma datos y partir del cual es entrenada y realiza las predicciones y, por otro lado, qué datos utiliza la red neuronal para ser entrenada en el contexto de un dataset multi-usuario.

Para seguir con el análisis de la tarea de PCSF se definen a continuación las características que debe posee el dataset a utilizar:

* Es multi-usuario: el *dataset* posee información de diferentes usuarios. De esta forma cada registro del dataset puede ser asociado a un usuario en particular.
* Es de tipo *time-series:* los datos están organizados secuencialmente y en orden temporal. De esta forma cada registro del dataset puede ser asociado a un *timestamp* particular. De preferencia, dichos registros deberían estar igualmente espaciados en el tiempo.
* Poseer datos heterogéneos: el dataset posee información heterogénea sobre cada usuario. De esta forma, dicho *dataset* puede ser descompuesto en varios datasets, donde cada uno de ellos contiene la información de un tipo de sensor.
* Tener una larga duración: como se busca obtener patrones de comportamiento sedentario, es importante que el registro del dataset vaya desde semanas a meses. De esta forma, se puede evaluar la efectividad de los modelos, no solo para aprender patrones de comportamiento sedentario, sino que también para predecir cambios en dichos patrones.

A partir de las características anteriores, los tipos de modelos que se propone evaluar son:

* Por los usuarios utilizados:
  + Personal: en este tipo de modelo tanto los datos utilizados para entrenamiento como aquellos utilizados para testeo pertenecen al mismo usuario. Se pretende aprender modelos particulares para cada usuario del dataset.
  + Impersonal: en este tipo de modelo se utilizan los datos disponibles de todos los usuarios menos uno para entrenar el modelo y se utilizan datos del usuario restante para testearlo. Se pretende aprender un modelo general, que pueda ser útil para predecir el comportamiento de usuarios nuevos.
* Por el dataset:
  + Número de *lags*: define la cantidad de retrasos que son añadidos al *dataset*.
  + Períodos: define el espacio temporal entre cada retraso añadido al *dataset*.
  + Granularidad: define el tamaño del intervalo temporal en los que se divide el *dataset*.
* Por las arquitecturas de Deep Learning
  + MLP
  + CNN
  + RNN
  + TCN

Uno de los hiperparámetros que varía en las distintas arquitecturas propuestas es la cantidad de time-lags utilizados. Los time-lags son la cantidad de información que se le da a la arquitectura sobre el pasado. Es decir, si se quiere predecir el gasto de energía de un usuario en un tiempo , y el número de lags es 3, la entrada de la red neuronal estará formada por las características (o *features*) de los tiempos , e .

En general, los hiperparámetros a configurar son: tipo de capas utilizadas, cantidad de capas, cantidad de unidades por capa, técnica de regularización a utilizar, número a utilizar, algoritmo de optimización y la función de pérdida. Además, las CNN poseen ciertos parámetros que no poseen las demás arquitecturas, como la cantidad de filtros por capa o el tamaño del núcleo (kernel). Más aún, las TCN poseen hiperparámetros que no poseen las CNN: como la cantidad de bloques residuales o la lista de dilataciones. Estos últimos hiperparámetros se configuran de una forma menos aleatoria que los demás, ya que lo que se intenta calcular en este caso es que el campo receptivo sea igual o mayor a la cantidad de time-lags.

En resumen, se evaluarán 4 arquitecturas de redes neuronales: la arquitectura 1 es una RNN, la arquitectura 2 es una CNN, la arquitectura 3 es una TCN y la arquitectura 4 es una NN. La arquitectura 4 se presenta a modo de baseline, y utiliza únicamente información de la hora anterior para predecir el valor del MET de la hora siguiente. Esta arquitectura nos permitirá entonces evaluar la utilidad de utilizar un enfoque de modelado de secuencias para FSBP.

A continuación, la Tabla 1 muestra una descripción de las arquitecturas propuestas. En todos los casos se utiliza el Error Cuadrático Medio (MSE) como función de pérdida, ADAM como algoritmo de optimización, siendo 128 la cantidad de *epochs* y 64 el tamaño de *batch*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Arq. 1 | Arq. 2 | Arq. 3 | Arq. 4 |
| Tipo de capas | LSTM x 2 - FC x 1 | No aplica | Conv1D x 1 - Flatten x 1 - FC x 1 | Dense x 3 |
| Neuronas por capa | 64-32-1 | No aplica | 32 - No aplica - 1 | 64 - 32 - 1 |
| Técnica de regularización | Dropout | Dropout | Dropout y Batch Normalization | Dropout |
| Número de bloques residuales | No aplica | 1 | No aplica | No aplica |
| Lista de dilataciones | No aplica | 1-2-4 | No aplica | No aplica |
| Tamaño del kernel | No aplica | 2 | 2 | No aplica |
| Número de filtros | No aplica | 6 | 32 | No aplica |

FC = Fully Connected

Tabla 1. Descripción de las arquitecturas propuestas

## Evaluación Experimental

En este Capítulo, primero se presenta el *dataset* utilizado para evaluar las diferentes arquitecturas propuestas para la tarea de FSBP. Luego, se describe el pre-procesamiento realizado y las características extraídas del dataset. A continuación, se exponen los usuarios del dataset seleccionados como casos de estudio. Finalmente, se muestran y se discuten los resultados obtenidos. Para la implementación de las arquitecturas se utilizó la librería Keras[[11]](#footnote-10) de Python 3. Los scripts implementados para el procesamiento del dataset y el entrenamiento de los modelos se dejan disponibles públicamente[[12]](#footnote-11).

## Descripción del Dataset

## Introduccion

Para validar la eficiencia de los modelos propuestas fue utilizado el dataset StudentLife [(Wang et al. 2014)](https://paperpile.com/c/FQGDmo/eCuKI). Este dataset fue recolectado a partir de la aplicación de sensado continuo StudentLife. Los participantes en este estudio fueron 48 estudiantes a lo largo de 10 semanas en la primavera de 2013.

Más específicamente, el dataset se recogió de 30 estudiantes de pregrado y 18 graduados. De todo el grupo de estudiantes, 38 eran varones y 10 mujeres. Del grupo de estudiantes de pregrado, dos eran de primer año, 14 de segundo año, 6 de tercer año y 8 de cuarto año. También hubo 13 estudiantes de maestría de primer año y 1 de segundo año, y 3 estudiantes de doctorado. Los participantes fueron racialmente diversos, con 23 caucásicos, 23 asiáticos y 2 afroamericanos.

## Tipos de datos del *dataset StudentLife*

Los tipos de datos disponibles incluidos en el dataset *StudentLife* son los siguientes:

* *App Usage*: contiene información sobre el uso de las aplicaciones de los smartphones.
* *Calendar*: contiene información sobre el calendario de los usuarios. Es decir, cada registro indica una fecha en la que el usuario tiene un evento en el calendario -no especifica el tipo de evento-.
* *Call-log*: contiene información sobre las llamadas producidas desde el teléfono del usuario, así como su duración.
* *Dinning*: contiene información sobre las compras alimenticias dentro del campus universitario de la Universidad de Dartmouth.
* *Education*: contiene información diversa de los diferentes cursos a los cuales los estudiantes asistieron a lo largo del estudio. Esto incluye los cursos a los cuales cada estudiante estaba inscripto, información detallada de cada curso, las fechas límites de cada estudiante, así como también el puntaje obtenido en cada asignatura.
* EMA (Ecological momentary assessments): contiene información sobre diversas y pequeñas encuestas que han sido respondidas por los usuarios a lo largo de las 10 semanas que duró el estudio. Estas encuestas buscan evaluar diferentes cuestiones de orden psicológicos, como el estado anímico, el nivel de estrés y el comportamiento en general. En promedio cada usuario respondió 8 EMAs por dia.
* *Sensing*: contiene información recolectada a partir de los sensores de los smartphones portados por los usuarios. Este tipo de datos ha sido la principal fuente de datos para entrenar los modelos desarrollados en esta tesis, por lo que tiene su sección propia.
* SMS: similar al tipo de datos Call-log, pero conteniendo información sobre los mensajes de texto enviados y recibidos.
* *Survey*: contiene 8 diferentes encuestas que fueron realizadas antes y después de comenzar el estudio. Las encuestas son: *Flourishing Scale* [*(Russell 1996)*](https://paperpile.com/c/FQGDmo/jE3L)*, Loneliness Scale* [*(Russell 1996)*](https://paperpile.com/c/FQGDmo/jE3L)*, Panas* [*(Watson et al. 1988)*](https://paperpile.com/c/FQGDmo/sjtY)*, Perceived Stress Scale* y PHQ-9 [(Kroenke et al. 2001)](https://paperpile.com/c/FQGDmo/wqUG).

## Datos no disponibles

La gran mayoría del total de los datos recolectados durante el estudio StudentLife no se encuentran disponibles en el dataset disponible públicamente, debido a dos razones. La primera razón es que algunos datos no fueron incluidos en el dataset publicado debido a cuestiones de privacidad, como los mensajes de los SMS, los datos de audio o los nombres de los usuarios. Otros datos que no están disponibles fueron utilizados solo para procesar otros datos que sí fueron incluidos en el *dataset* publicado. De esta forma, el total de los datos recolectados a partir de los diferentes sensores del celular es de 52.6 GB, mientras que en el dataset disponible públicamente solo son 2.6 GB aproximadamente.

## Datos del acelerómetro

Los únicos datos sin procesar que se encuentran disponibles son los del acelerómetro, pero no han podido ser utilizados por problemas técnicos. Los datos del acelerómetro para cada usuario fueron puestos a disposición de tal forma que cada usuario tiene asociado un archivo de base de datos de tipo SQLite. Cada base de datos está compuesta por solo una tabla con dos columnas. La primera columna es llamada timestamp y representa el momento en el que se registró la información del acelerómetro. La segunda columna es llamada *feat\_data*, que debiera contener la información del registro del acelerómetro. El problema hallado es que esta última columna es de tipo BLOB, que es utilizado para almacenar cadenas de longitud variable de bytes binarios. No se logró encontrar la forma correcta de extraer los datos del acelerómetro de tipo BLOB, por lo que dichos datos no han sido utilizados. Es importante señalar que los datos del acelerómetro podrían haber representado una importante mejoría para los modelos implementados, ya que podrían haberles aportado información muy importante acerca de la actividad física del usuario. Esto es así debido a que los datos de la actividad provistos por el dataset StudentLife tienen una granularidad muy pobre (solo 3 tipos).

## Datos de sensado

Cada uno de estos tipos de datos de sensado son provistos para cada usuario en formato csv (comma separated values, valores separados por una coma). Por lo tanto, a cada tipo de dato le corresponden 48 archivos csv. Cada uno de estos datos está acompañado de un timestamp que indica el horario y la fecha del registro. Estos timestamp tienen el formato de Unix, por lo que se representan mediante un número entero positivo que indica la cantidad de segundos pasados desde el primero de enero de 1970. Por ejemplo el timestamp 1364357009 equivale a 03/27/2013 4:03 AM.

Por lo tanto, cada registro se compone de un timestamp y diversos datos asociados al tipo de dato, pudiendo ser uno solo (como en el caso de activity) o varios, como en el caso de GPS. Aunque este es el caso para la mayoría de tipos de datos de sensado continuo, algunos presentan otro formato, en el cual existen dos timestamp que indican el momento de inicio y finalización de un evento. Al detallar cada tipo de dato de sensado, se clasificará a estos dos tipos de datos como Tipo de Sensado Discreto (TSD) y Tipo de Sensado por Intervalo (TSI).

Los investigadores utilizaron la aplicación *BeWell* [*(Lane et al. 2012)*](https://paperpile.com/c/FQGDmo/plSy) para proveer un framework para el sensado automático en el estudio StudentLife. Los investigadores aclaran que el estudio StudentLife fue llevado a cabo antes que Google anunciara su servicio de reconocimiento de actividad para teléfonos Android, que fue el utilizado en el estudio. Debido a esto, se utilizó el motor de sensado Jigsaw [2010, J. Lu], desarrollado por los investigadores en un trabajo previo, que permite recolectar y clasificar datos del acelerómetro, el GPS y audio de forma eficiente.

A continuación, se listan los diferentes datos de sensado continuo incluidos en el dataset StudentLife.

* Tipo de Sensado Discreto
  + *Activity*: contiene registros de la actividad física del usuario. Este tipo de dato ha sido recolectado aproximadamente cada 2-3 segundos, dependiendo de la frecuencia de muestreo del acelerómetro del smartphone. El clasificador de actividad del motor de sensado Jigsaw alcanzó un 94% de precisión. Los registros de actividad están clasificados en 4 tipos diferentes y se representan mediante un código en los archivos .csv. Por ejemplo, un registro de actividad física es el siguiente [1364356899, 0], que significa que los datos del acelerómetro que fueron recolectados el 27/03/2013 a las 4:01:39 fueron inferidos como “estacionario” por el clasificador de actividad de Jigsaw. A continuación, se listan los tipos de actividad física junto con el código con el cual están representados en el dataset:
    - *Stationary* (Cód. 0): el usuario se encuentra en estado estacionario.
    - *Walking* (Cód. 1): el usuario se encuentra caminando;
    - *Running* (Cód. 2): el usuario se encuentra corriendo;
    - *Unknown* (Cód. 3): se desconoce el estado de actividad del usuario (en un intercambio de mails con Rui Wang, el investigador explicó que estos registros se encuentran principalmente al principio del estudio StudentLife, por lo que no deberían representar un inconveniente para realizar experimentos utilizando el dataset).
  + *Audio*: Contiene registros de audio del usuario. El clasificador de audio de Jigsaw funciona realizando inferencias de audio durante 1 minuto y luego se pausa por 3 minutos, a no ser que el clasificador de conversación detecte que hay una conversación en curso, en cuyo caso el clasificador de audio no se detiene. Este tipo de dato ha sido recolectado aproximadamente cada 2-3 segundos. A continuación, se listan los tipos de audio junto con el código con el cual están representados en el dataset:
    - *Silence* (Cod. 0): el clasificador de audio infiere silencio.
    - *Voice* (Cod. 1): el clasificador de audio infiere voz (posiblemente una conversación).
    - *Noise* (Cod. 2): el clasificador de audio infiere ruido.
    - *Unknown* (Cod. 3): el clasificador de audio no reconoce el tipo de audio.
  + Bluetooth: contiene información sobre escaneos del dispositivo de bluetooth perteneciente al smartphone de cada usuario. En este caso, y a diferencia de otros tipos TSD, varios registros pueden compartir el mismo *timestamp*. Esto es normal, ya que cada escaneo reporta información sobre las redes bluetooth cercanas halladas, que pueden ser más de una. Para cada red bluetooth escaneada, se obtiene la dirección MAC y la intensidad de la señal. La frecuencia de muestreo para este tipo de sensor es de 1 cada 10 minutos.
  + GPS: contiene información sobre el GPS. Dicha información incluye la proveniencia de la señal (GPS o Network), la latitud, la longitud y la altitud. La frecuencia de muestreo para este tipo de sensor es de 1 cada 10 minutos.
  + Wifi : Similar al tipo de dato Bluetooth, estos registros arrojan información sobre las redes de tipo WIFI escaneadas, como la dirección MAC del router, la frecuencia del canal por donde se realiza la conexión y la intensidad de se la señal. Al igual que en Bluetooth, pueden existir varios timestamp idénticos.
  + Wifi *location*: los autores del dataset StudentLife dividieron el Campus de la ciudad de Dartmouth en diferentes zonas. En los registros de este tipo de sensor, podemos hallar una ubicación aproximada de un usuario en un determinado momento a partir de la red WIFI a la que está conectado. Las zonasse clasifican en In[x] (dentro del edificio x) y Near[x] (dentro del edificio x).

* Tipos de sensado por intervalo
  + *Conversation*: intervalo de tiempo en el que el usuario llevó a cabo una conversación.
  + *Phone charge*: intervalo de tiempo en el que el smartphone se encuentra siendo cargado.
  + *Phone lock*: intervalo de tiempo en el que el smartphone se encuentra bloqueado.
  + *Light*: intervalo de tiempo en el que el smartphone se encuentra en la oscuridad.

## Pre-procesamiento del dataset

A continuación se describe el proceso llevado a cabo para procesar el dataset StudentLife con la finalidad de ser utilizado por los diferentes modelos planteados para resolver el problema de PCSF. Primero, se realiza un análisis de los datos para conocer las características del dataset y definir qué tipos de datos van a ser utilizados.

Como se ha dicho con anterioridad, cada registro del dataset StudentLife posee un timestamp asociado. Por lo tanto, todas las características que se generaron corresponden a una combinación de usuario/hora en particular y son algún cómputo que resume algún aspecto de los datos de algún usuario/hora particular. Por ejemplo, un intervalo específico puede corresponder al usuario 10 y la hora 2013-04-24 19: 00–20: 00.

## Análisis de datos

## Tamaño de buckets

Como ya ha sido mostrado, los tipos de datos que provee el *dataset StudentLife* son un conjunto heterogéneo, es decir, existen muchos tipos diferentes de datos. Por lo tanto, si se quiere utilizar este dataset para generar modelos de Machine Learning con el fin de llevar a cabo la tarea de PCSF, debe llevarse a cabo un proceso previo que permita que dicho *dataset* sea apto para los diferentes modelos. Para esto, a partir del *dataset StudentLife* se debe generar un nuevo *dataset* que esté compuesto por casos, donde cada caso esté asociado a un valor de la variable objetivo. Cada caso de estudio estará compuesto por características que serán utilizadas para predecir la variable objetivo. Por su parte, la variable objetivo representará el nivel de sedentarismo expresado a través del valor de MET.

El problema más grande que trae la gran heterogeneidad de los datos es su disponibilidad. Podemos distinguir 3 tipos de datos presentes en el dataset según su disponibilidad:

* Las encuestas que se hicieron antes y después del estudio StudentLife,
* los datos EMA y,
* los datos de sensores y derivados de ellos.

De estos 3 tipos, solo en el tercer caso existen datos presentes uniformemente a lo largo de todo el estudio -incluyendo los datos de actividad física-, lo que permite generar una gran cantidad de casos o ejemplo de entrenamiento para los modelos, que es crucial para lograr entrenar modelos de Machine Learning que alcancen un buen desempeño. Por esta razón, los datasets para entrenar y testear los modelos serán generados a partir de los datos de sensores y derivados de ellos. Los otros dos tipos de datos no serán utilizados debido a su baja disponibilidad. Es decir, si fuesen utilizados, sólo podrían generarse casos de entrenamiento para aquellos momentos en los que hay datos disponibles, que son muy pocos en comparación a los datos de sensado.

Ahora bien, dentro de los datos de sensores, puede observarse también una disponibilidad variable. Es decir, cada tipo de datos de sensor tiene una cantidad promedio de registros por unidad de tiempo diferente (esta cantidad de registros está dada en la descripción del dataset, pero aun así es calculada junto con otros estadísticos en la siguiente sección a modo de comprobación). Por esta razón, para generar casos es necesario procesar los datos de sensores de forma tal que cada caso pueda incluir información de todos los sensores. Esto último se logra discretizando en tiempo en intervalos, llamados *buckets*. La longitud de estos intervalos debe definirse teniendo en cuenta la frecuencia que tienen cada uno de los datos. Es decir, cada bucket debe contener por lo menos un registro de cada tipo de dato que se quiere usar, ya que la gran mayoría de los modelos de Machine Learning, y en especial los que serán utilizados en esta tesis, no aceptan características sin valor. Así mismo, es válido discutir un número mínimo de registros necesarios para que un tipo de dato en un intervalo de tiempo específico sea válido. Es decir, es necesario responder a la pregunta: ¿A partir de qué cantidad de registros se considera válido un tipo específico de dato? A modo de ejemplo, ¿es válido un bucket de 1 hora de longitud donde se cuenta con solo un registro de actividad física? ¿Puede considerarse que ese registro representa a todo el intervalo (teniendo en cuenta que según la descripción del dataset los registros de actividad física se han registrado cada 3 segundos, es decir, que se esperan 1200)? ¿Si no, cuál debería ser el número mínimo? ¿Es conveniente descartar estos buckets sabiendo que reducen el número de datos de entrenamiento y, en consecuencia, el desempeño de los modelos?

Una vez que el dataset se discretiza en buckets, diferentes características pueden ser calculadas a partir de los registros de cada tipo de dato incluido en cada uno de los buckets. Por ejemplo, para un Tipo de Sensor Discreto en el que se cuentan con registros en un intervalo de tiempo , correspondiente al tamaño de bucket, se generan características a partir de la información disponible. Dichas características pueden ser estadísticos que expliquen los datos disponibles u otros cálculos ad-hoc. Es válido tanto que varios tipos de datos puedan ser combinados para generar una sola característica, así como que varias características sean generadas a partir de un solo tipo de dato.

De acuerdo a las explicaciones anteriores, el tamaño de *bucket* va a definir cuántos registros están presente en cada uno de ellos así como también la cantidad de *bucket* resultantes. De esta manera, se da la siguiente relación: cuanto mayor sea el tamaño de *bucket*, mayor será la cantidad de datos disponibles en cada uno de ellos, a la vez que la agrupación del *dataset* en *buckets* dará como resultado una menor cantidad total de ellos. Esta relación sirve como punto de partida para analizar y definir cuál debería ser el tamaño de *bucket*. Por un lado, que haya más *buckets* en total es beneficioso para los modelos ya que se parte de un *dataset* mas grande del cual aprender y la granularidad a la hora de predecir el comportamiento sedentario es mayor, por lo que un tamaño de *bucket* pequeño es deseado. Por otro lado, hay tipos de datos cuya frecuencia de muestreo es muy baja, por lo que si el tamaño de *bucket* es demasiado pequeño, puede que no haya datos disponibles para ese tipo de sensor.

A partir del análisis anterior, se decidió realizar los experimentos utilizando dos tamaños de buckets diferentes: de 1 hora y de 30 minutos.

Por un lado, se toma el tamaño de *bucket* de 1 hora debido a que es el utilizado en todos los trabajos relacionados a la tarea de PCSF que usan el *dataset StudentLife* [*(He and Agu 2016; He and Agu 2016; He and Agu 2016; He and Agu 2017)*](https://paperpile.com/c/FQGDmo/2NLm5+GSfPw+axGRS+oye2). Al utilizar estos tamaños de *buckets* es posible comparar diferentes cuestiones con trabajos previos, como la cantidad de *buckets* resultantes del procesamiento, a pesar de que estos trabajos difieren en el enfoque propuesto en esta tesis en relación a cómo medir el comportamiento sedentario. Por otro lado, los intervalos de 1 hora son suficientemente grandes como para incluir en promedio siempre más de un registro por bucket para generar las características.

Por otro lado, la división entre *buckets* de media hora fue incluida para comparar el impacto en el desempeño de los modelos de aprendizaje de doblar aproximadamente la cantidad de casos generados a partir del *dataset* original. La desventaja de esto es que habrá más *buckets* que deberán ser descartados por no presentar registros de alguno de los tipos de datos. Como consecuencia, los modelos que sean entrenados a partir de la utilización de varios *buckets* consecutivos (Ej.: CNN, RNN) se verán afectados en cierto sentido porque se darán más casos en los cuales una secuencia de *buckets* deberá ser descartada por la ausencia de uno o más *buckets*.

## Disponibilidad de los datos

La frecuencia de cada uno de los tipos de datos de sensado continuo fue listada en la sección [Datos de sensado]. Como ya ha sido discutido, para entrenar modelos de Machine Learning, es preferible utilizar los tipos de datos para los cuales haya más información disponible. Es por esta razón que se seleccionan del dataset StudentLife principalmente los tipos de datos de sensado continuo, y se descartan aquellos relacionados a test psicológicos o encuestas EMA.

En esta sección se presenta un análisis sobre la disponibilidad de cada tipo de datos de sensado continuo con el objetivo de verificar la información sobre la frecuencia de cada tipo de dato listada en la descripción del dataset. El análisis de la disponibilidad de cada tipo de dato difiere según si el tipo es TSD o TSI. Para realizar un cálculo de la disponibilidad de cada tipo de dato se discretiza el tiempo en dos tipos de buckets: de 1 hora y de 30 minutos.

Para los TSD, se calcula:

* la cantidad de registros disponibles,
* la cantidad de buckets totales resultantes luego de la discretización,
* los *buckets* para los cuales se tiene información -que serán siempre menores o iguales a los *buckets* totales- y,
* los siguientes estadísticos: media, desviación estándar, mínimo, máximo de registros por buckets.

Los resultados pueden verse en la tabla X. En el caso específico del sensor Activity, existen muchos valores nulos, por lo que se incluye una fila con los resultados de eliminar los registros clasificados como unknown.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Cantidad total de registros | Cantidad de buckets para los cuales hay datos | | Cantidad de buckets | | Media y desviación estándar de registros por bucket | | Maximo y minimo de registros por bucket | |
| 1h | 30min | 1h | 30min | 1h | 30min | 1h | 30min |
| Activity | 22.842.191 | 62.807 | 125.009 | 77665 | 155330 | 363.68/193.29 | 182.72/100.09 | 1 / 4820 | 1 / 3538 |
| Activity (sin unknowns) | 22.233.593 | 62.767 | 124.887 | 77665 | 155330 | 354.22/186.45 | 178.02/96.42 | 1 / 4367 | 1 / 3537 |
| Audio | 99.298.223 | 62.792 | 124.968 | 77665 | 155330 | 1581.38/973.7 | 794.58/514.07 | 5 / 45.294 | 1 / 22692 |
| GPS | 202.877 | 59.223 | 115.896 | 77665 | 155330 | 3.42/1.3 | 1.75/0.77 | 1 / 16 | 1 / 9 |
| WIFI | 19.244.309 | 61.750 | 122.526 | 77665 | 155330 | 311.64/522.3 | 157.06/273.09 | 1 / 6144 | 1 / 3500 |
| WIFI location | 1.893.838 | 50.747 | 99.407 | 77665 | 155330 | 37.31/59.74 | 19.05/30.63 | 1 / 357 | 1 / 179 |
| Bluetooth | 1.288.526 | 39.693 | 70.075 | 77665 | 155330 | 32.46/146.35 | 18.38/79.55 | 1 / 4987 | 1 / 2676 |
| Calendar | 1.687 | 563 | 563 | 65660 | 131292 | 2.99/2.31 | 2.99/2.31 | 1 / 20 | 1 / 20 |

Como puede verse en la Tabla X la cantidad de registros disponibles varía mucho entre los diferentes TSD, yendo desde poco más de 200 mil registros disponibles para el GPS, hasta casi 100 millones en el caso de los datos de audio. A pesar de esto, la cantidad de *buckets* para los cuales existe información no se ve tan afectada, tanto en el caso de *buckets* de una hora como para los de 30 minutos. Para los casos de los *buckets* de una hora la cantidad mínima de *buckets* corresponde al sensor Bluetooth, con 39.693 *buckets*, mientras que la cantidad máxima al sensor *Activity*, con 62.807 *buckets*. Sin embargo, para la gran mayoría de los sensores de este tipo la cantidad de *buckets* ronda los 60.000. El mismo resultado se da en el caso de *buckets* de 30 minutos, con la diferencia de que las cantidades son de aproximadamente el doble en cada caso.

La columna “Cantidad de *buckets*” se calcula como la cantidad de *buckets* totales existentes desde el primer registro al último, es decir, teniendo en cuenta también los *buckets* para los cuales no existen datos para el sensor. Puede observarse que son iguales para todos los TSD y en ambos tamaños de buckets. La fecha del primer registro data del 2013-03-27 a las 05:00:00, mientras que la última del 2013-06-01 a las 05:59:59.

Las últimas dos columnas muestran diferentes estadísticos que permiten comprender mejor cómo se distribuye la totalidad de los registros en cada *bucket*. De la media y la desviación estándar se puede observar que en general la cantidad de registros por buckets difiere mucho, tanto que en algunas casos la desviación estándar es mayor que la media, de lo que se puede deducir que hay periodos de tiempo en los que la recolección de datos es muy alta y otra en las que fue casi nula. Además, muestra que en ningún caso la frecuencia de muestreo es igual a la reportada en la descripción del dataset para cada tipo de dato de sensado continuo. Por ejemplo, la frecuencia reportada para el sensor *Activity* es de 3 segundos, por lo que se esperarían 1200 registros por hora, mientras que la media de registros por *buckets* es de 360. Además, si se analizan los máximos y mínimos de cada sensor se pueden observar más inconsistencias, ya que todos los sensores poseen por lo menos un bucket para el cual se dispone del número mínimo de registros, es decir uno. Mientras que el número máximo representa picos, también en todos los sensores, de manera anormal ya que en muchos casos se alejan mucho de la media, siendo el caso más extremo para el sensor *Audio*, para el cual existe un *bucket* de una hora con 45.294 registros. Este análisis no representa en sí un problema a la hora de procesar el *dataset StudentLife*, pero si da la pauta de que el *dataset* posee inconsistencias con respecto a su documentación.

Con respecto a la diferencia que existe entre la cantidad de *buckets* con información disponible para cada sensor, debe realizarse un análisis más detallado. Si bien para que un *bucket* sea apto para ser utilizado en el *dataset* de casos debe poseer información sobre todos los sensores que sean utilizados para generar las características, esto no significa que la falta de registros en un *bucket* equivalga a la falta de información. Esto es, en algunos casos, puede interpretarse la ausencia de datos como información útil que pueda ser calculada para dicho bucket. Por ejemplo, en el caso del sensor Bluetooth, que contiene registros correspondientes a las redes Bluetooth cercanas, la falta de registros en un *bucket* determinado puede interpretarse como que no se han detectado redes Bluetooth cercanas. Por lo tanto, si una característica computada para cada *bucket* es el número de redes Bluetooth detectadas, los *buckets* para los cuales no se tengan registros Bluetooth tendrán un número 0 de redes Bluetooth detectadas y no deberán ser descartados del *dataset*.

En el caso de los sensores que sean usados y no sea posible interpretar los *buckets* para los cuales no se tengan registros disponibles como información útil, deberán ser descartados. Por lo tanto, en estos casos es importante analizar si es mejor no utilizar los datos de ese sensor en pos de tener más *buckets* disponibles o mantenerlos, aumentando así la información de contexto que pueden utilizar los modelos, pero reduciendo la cantidad de casos de entrenamiento disponibles. Como la cantidad de *buckets* en todos los casos es similar, esta decisión desembocará siempre en utilizar el sensor con *buckets* vacíos en lugar de descartarlos. Otro punto importante a destacar es que el número total de casos resultantes no será el número de *buckets* del sensor con menos *buckets* disponibles con información, sino que será la cardinalidad de la unión de todos los *buckets* con información útil disponible para cada uno de los sensores utilizados. Se hace esta aclaración ya que los *buckets* sin información de un sensor pueden no coincidir con los *buckets* sin información de otro sensor.

Para los TSI, se calculan:

* La cantidad total de registros,
* El promedio y la desviacion estandar del tamaño de los intervalos,
* El maximo y minimo del tamaño de los intervalos, y
* El porcentaje de cobertura total.

En la tabla TABLA X pueden observarse los datos calculados. Al analizar la primera columna pueden notarse que la cantidad total de registros del sensor *Conversation* posee desde 8 a 24 veces más registros que los demás sensores. Esto se da porque el sensor *Conversation* es el único TSI para el cual no se limitó el tiempo minimo de duracion. Para los TSI *Dark, PhoneLock* y *PhoneCharge* los únicos registros que se incluyeron en el *dataset* son aquellos cuya duración supera una hora. Esto puede corroborarse gracias a la columna donde se muestran los mínimos, donde puede observarse que el mínimo de estos 3 sensores es de una hora. Es comprensible la decisión de no limitar la mínima duración de las conversaciones ya que tienden a ser más cortas que los eventos que describen los otros sensores. Por ejemplo: los dispositivos requieren más de una hora para ser cargados. Aunque se entiende que la duración mínima de las conversaciones también han sido limitadas, ya que la conversación más corta registrada es de 36 segundos. Con respecto a la duración máxima de los registros, aunque no se informa al respecto en la descripción del *dataset*, parece haber sido truncada a 10 horas.

La mayor granularidad del sensor *Conversation* permite idear diferentes características que pueden ser calculadas para el resto de los TSI. Principalmente, a un determinado *bucket* pueden pertenecer más de un registro del TSI *Conversation,* por lo que puede calcularse la cantidad de conversaciones pertenecientes a cada *bucket* o el porcentaje del *bucket* en el cual el usuario mantuvo una conversación, lo cual enriquece el contexto a partir del cual los modelos realizarán las predicciones*.* En el caso del resto de los otros TSI, la cantidad de registros que pertenezcan a un determinado *bucket* de 30 minutos o una hora siempre va a ser o 1 o 0.

El porcentaje de cobertura total se calculó como la división entre la suma de la duración de todos los registros y el tiempo entre el comienzo del primer registro y el final del último. Este porcentaje fue calculado suponiendo que no hay intersecciones temporales entre los registros por lo que los valores mostrados son aproximados. Por supuesto, el porcentaje de cobertura total puede variar entre los diferentes usuarios. Por ejemplo, aquellos que sean más sociables tendrán un mayor porcentaje de cobertura total para el sensor *Conversation.* Como puede observarse, el sensor *PhoneLock* resultó ser aquel con el mayor porcentaje de cobertura total, mientras que el sensor *Conversation* resultó ser aquel con el mayor porcentaje de cobertura total.

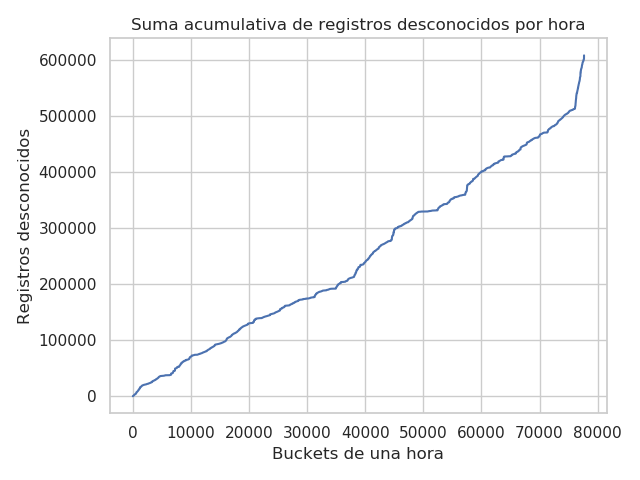
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Cantidad total de registros | Promedio y desviación estándar del tamaño de los intervalos (horas) | | Maximo y minimo del tamaño de los intervalos (horas) | | Porcentaje de cobertura total (%) |
|
| *Dark* | 7.269 | 3.246/2.521 | | 1/9.997 | | 33,7 |
| *PhoneLock* | 9.275 | 3.121/2.37 | | 1/10.0 | | 40,5 |
| *PhoneCharge* | 3.318 | 3.95/2.722 | | 1/9.999 | | 18,9 |
| *Conversation* | 79.023 | 0.17 / 0.27 | | 0.006/6.872 | | 18,8 |

*Tabla*

## Registros desconocidos

Existen dos razones por las cuales es importante realizar un análisis específicamente sobre los datos del sensor Activity. Por un lado, este es el único sensor que presenta registros clasificados como desconocidos, por lo que es importante llevar a cabo un análisis que permita estimar las consecuencias de la presencia de estos registros faltantes en el dataset StudentLife. Por otro, los datos de este sensor son los únicos utilizados para generar la variable objetivo, es decir, la variable a predecir. El nivel de sedentarismo del usuario en un bucket dado se calcula a partir de todos los registros de actividad física disponibles en ese bucket. Claro está, los registros de actividad física clasificados como unknown no pueden ser utilizados ya que se desconoce cuál es la actividad física que el usuario estaba llevando a cabo en ese momento.

En esta sección se analizan los registros clasificados como unknown para entender su distribución a lo largo del estudio StudentLife, así como también cómo varía su distribución para cada uno de los usuarios. Esto permitirá evaluar si es necesario tomar medidas con respecto al manejo de estos registros desconocidos y que problemas puede causar.

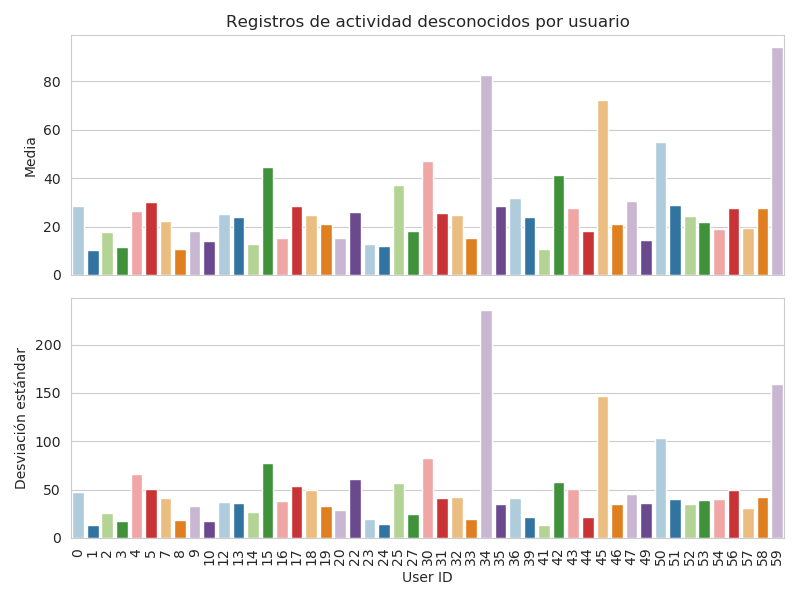
Los registros para los cuales no se tiene información sobre la actividad física son codificados por el número 3. Sabiendo esto, resulta sencillo generar gráficos que muestran claramente la distribución de dichos registros. Cabe aclarar que todos los gráficos que se muestran en esta sección han sido generados a partir de los buckets de una 1 hora. No es necesario realizar el mismo análisis para ambos tipos de buckets ya que arrojaron resultados similares.

En la figura X se muestra la suma acumulativa de registros desconocidos para todas las horas disponibles. En dicha figura, el eje representa a cada *bucket* que es posible generar a partir de los datos del sensor Sensing, sin importar a qué usuario pertenezcan. Mientras que el eje representa la suma cumulativa de registros clasificados como unknown, es decir que el valor de es la suma desde el primer bucket hasta el bucket .

En la figura, puede observarse que la cantidad de registros crece de forma aproximadamente lineal con respecto a la cantidad de buckets a lo largo de todo el dataset. Esto es una buena señal, ya que significa que, *a priori*, los buckets tendrían una cantidad similar de registros a partir de los cuales generar las características de actividad y que los registros unknown no están agrupados en una sección específica del dataset. Esto quiere decir que, al momento de calcular el nivel de sedentarismo, cada bucket tendrá un soporte similar para calcular dicho nivel.

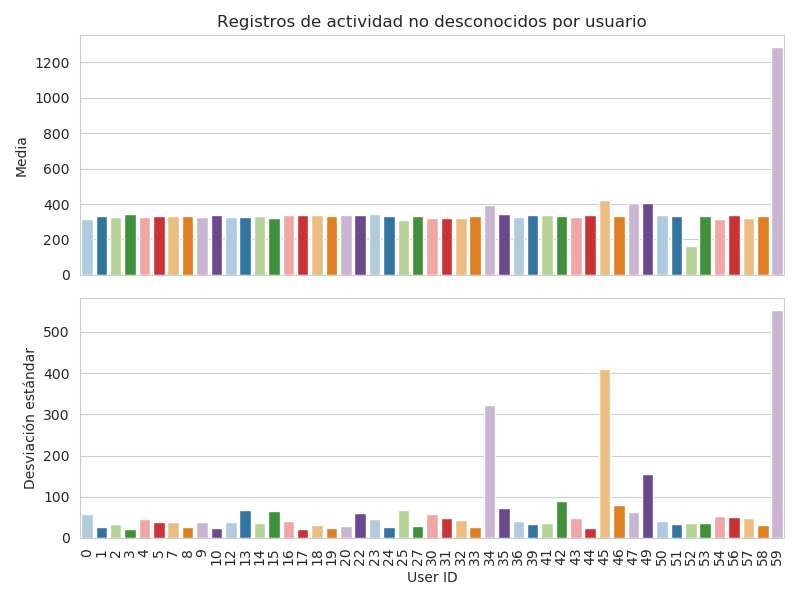
Al final del eje , se puede observar un pico en la cantidad acumulada de registros *unknown*. Estos registros pertenecen al último usuario, que presenta algunas diferencias con los demás usuarios, que podrán ser mejor visualizadas en el siguiente gráfico.

En la Figura Y, se muestran dos gráficos de barras con la media y la desviación estándar de la cantidad de registros desconocidos para cada hora de un determinado usuario. Es decir, si para un usuario se disponen 1000 horas, se calcula la media y la desviación estándar de la cantidad de registros desconocidos disponibles para cada hora.



Varias observaciones pueden hacerse a partir de estas figuras. Con respecto a los valores faltantes, puede verse que la mayor cantidad de los usuarios presenta entre 20 y 40 registros desconocidos por hora, lo que explica el carácter lineal de la función graficada en la Figura X. Sin embargo, existen ciertas excepciones, como los usuarios 34, 45 y 59, donde se puede observar una media mayor de registros faltantes, asimismo como una desviación estándar mayor. El usuario 59 es el que mayor cantidad de registros desconocidos posee, lo que explica el pico observado al final de la figura X.

Por su parte, la figura Z muestra los mismo estadísticos, pero en lugar de ser estos generados a partir de los registros desconocidos, éstos son generados a partir de los registros no desconocidos (con un código de actividad 0, 1 o 2), es decir, a partir de los cuales se tiene información certera. En el primer gráfico, donde se muestra la media, puede verse que todos los usuarios presentan casi la misma cantidad de registros por bucket (aproximadamente 350). Sin embargo, los mismos usuarios que poseen muchos registros desconocidos (34, 54, 59) también presentan una alta desviación estándar en la cantidad de registros no desconocidos por hora, hechos que podrían estar relacionados. Es posible observar, también, que el usuario 59 posee aproximadamente el cuádruple de registros por buckets. Más adelante se discutirá la posibilidad de tratar a estos usuarios que presentan una representación diferente en el dataset como outliers.



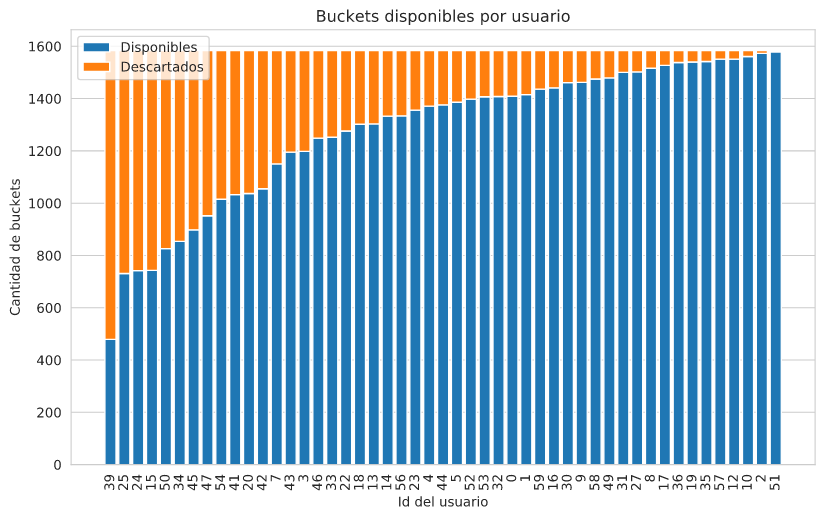
## Depuracion de buckets

Una vez que el dataset StudentLife ha sido discretizado, el siguiente paso del preprocesamiento es eliminar los buckets para los cuales no se posee información sobre alguna de las variables. Es preciso notar que solo se eliminan los buckets sin ningún registro que no pueden ser interpretadas por ninguna característica. Por ejemplo, para los buckets en los que no hubo registros de audio, puede interpretarse que hubo silencio, por lo que las características diseñadas pueden aprovechar esta interpretabilidad y tener un valor para esos buckets. En este caso, el valor para AudioMajor será 0 o “en silencio”.

En la figura X se muestra la cantidad de buckets resultantes disponibles para cada usuario y la cantidad de buckets descartados por la falta de registros para alguna variable, luego de depurar el dataset preprocesado. Como puede observarse, existe una gran diversidad con respecto a la cantidad de buckets para cada usuario. El usuario con menor cantidad de buckets disponibles es el 39, con 479 buckets, mientras que el usuario con más buckets es el 51, con 1579.

Como ya ha sido discutido en el marco teórico, la cantidad de casos de entrenamiento para un algoritmo de aprendizaje puede determinar que dicho algoritmo alcance un buen desempeño o no. Es necesario recordar en este momento que se comparan dos tipos de modelos de acuerdo a de donde provengan los datos de entrenamiento: los modelos personales y los impersonales.

Con respecto a la división anterior, es posible hipotetizar que para los usuarios con muy pocos *buckets* disponibles, como es el caso del usuario 39, los modelos impersonales obtendrán un mayor desempeño que los personales. La explicación de la hipótesis anterior se da porque puede que la baja cantidad de buckets no llegue a darle la información al algoritmo de aprendizaje sobre las idiosincrasias de estos usuarios en particular, que es la ventaja que presentan los modelos personales por sobre los impersonales.



## Nivel de MET

Los registros de actividad de cada usuario están en el dataset clasificados como estacionario, caminando o corriendo. A partir de estos registros se calcula el valor del gasto energético para cada intervalo. Tal como fue explicado anteriormente, el gasto energético es comúnmente medido en términos de MET.

El *dataset StudentLife* no posee información explícita que determine el nivel de MET, por lo que se tomó la decisión de diseñar un marco en el cual fuese posible calcularlo a partir de los datos disponibles. A cada tipo de actividad disponible en el dataset se le asigna un valor estático aproximado de MET de acuerdo al Compendio de Actividades Físicas [(Ainsworth et al., 2011)](https://paperpile.com/c/FQGDmo/zOAD7)[[13]](#footnote-12). El compendio de Actividades Físicas posee 21 categorías, donde cada una de ellas está conformada por tipos de actividades. A su vez, cada actividad está acompañada por el valor de MET asociado a esa actividad. Para cada una de las actividades presentes en el Compendio existe evidencia publicada que soporta la asociación entre dicha actividad y su valor de MET.

Una vez analizado el compendio de actividades físicas, se seleccionaron los tipos de actividades más similares a aquellas presentes en el dataset StudentLife. A las actividades registradas como estacionarias, se les otorgó un valor de MET de 1.3, el cual corresponde a actividades como “sentado en calma”, “acostado en calma”, “no haciendo nada”, “acostado despierto en la cama”, “escuchando musica”. A las actividades registradas como “caminando”, se les otorgó un valor de MET de 5.0, el cual corresponde a actividades como “cargando aproximadamente 6 kilos (por ejemplo, una mochila), en terreno llano o bajando escaleras”. A las actividades registradas como “corriendo” se les otorgó un valor de MET de 8.3, el cual corresponde a actividades como “corriendo, 8 km/h.

Una vez que se definió a qué valores de MET fue asociada cada tipo de actividad, se transformó la información disponible sobre la actividad (variable categórica ordinal) en una variable numérica. Luego, fue necesario definir cómo iba a ser resumida la información de la actividad física -expresada en METs- para cada *bucket*, de tal forma que el resultado fuese un número real. Es decir, cómo todos los datos disponibles son agrupados por buckets debe calcularse un valor de MET representativo para cada uno de ellos. Para ello, se define que el nivel de MET de un bucket está dado por la media del valor de MET de cada uno de los registros de actividad perteneciente a dicho bucket. Por lo tanto el nivel de MET de un bucket está dado por la fórmula:

,

donde es el valor de MET asociado a la actividad , el registro de actividad número y la cantidad de registros de actividad disponibles para el bucket. Por ejemplo, si un bucket contiene 100 registros de tipo “estacionario”, 10 registros de tipo “caminando” y 10 registros de tipo “corriendo”, el nivel de MET ese bucket se calcula como:

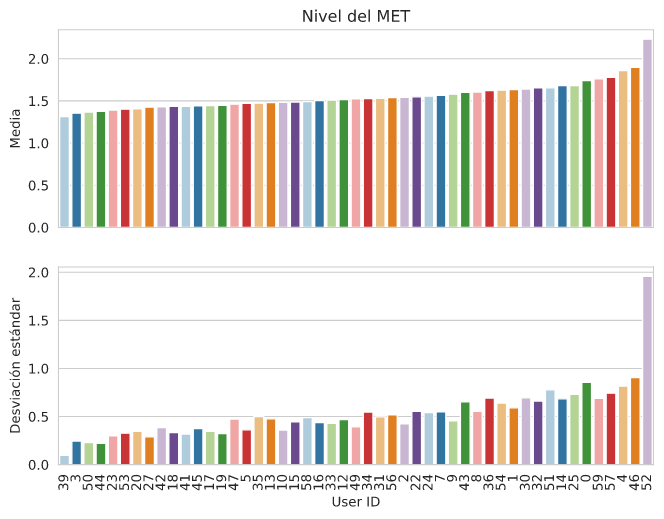
Como ya ha sido explicado con anterioridad, un comportamiento es considerado como sedentario si supera el valor de 1.5 METs. La anterior afirmación es adaptada en esta tesis de la siguiente forma: el comportamiento llevado a cabo en un *bucket* es considerado como sedentario si el nivel de MET para dicho *bucket* es superior a 1.5.

Es necesario recordar en este punto que la baja granularidad en el tipo de actividad física especificadas en el *dataset StudentLife* representa una limitación, ya que el tipo de actividad del Compendio de Actividades Físicas asociadas a cada una de las actividades físicas especificadas en el dataset StudentLife puede no ser certero y distar de la verdadera actividad que el usuario estaba llevando a cabo al momento de ser registrada. Ademas, los valores de MET asociados a cada activiad en el Compendio no estan destinados a ser utilizados de manera individual, ya que segun se explica en la pagina del Compendio “Los valores en el Compendio no estiman el gasto de energia de la actividad fisica en una forma en la cual se tengan en cuenta diferencias en masa coporal, adiposidad, edad, sexo, eficiencia en el movimiento, condiciones geograficas y ambientales en las cuales las actividades son llevadas a cabo. Por lo tanto, las diferencias entre el gasto energético para la misma actividad puede ser grande y el gasto energético real para un individuo puede o no ser cercano al promedio establecido de MET presentado en el Compendio”.

## Análisis del nivel de MET

En esta sección se realiza un análisis de la actividad física, con el fin de comprender cómo se distribuye ésta a lo largo de todo el dataset StudentLife. Este análisis permitirá conocer cuales son las actividades que más se llevan a cabo y en qué medida. Además, a través de los gráficos podrán observarse los patrones de comportamiento de los diferentes usuarios.

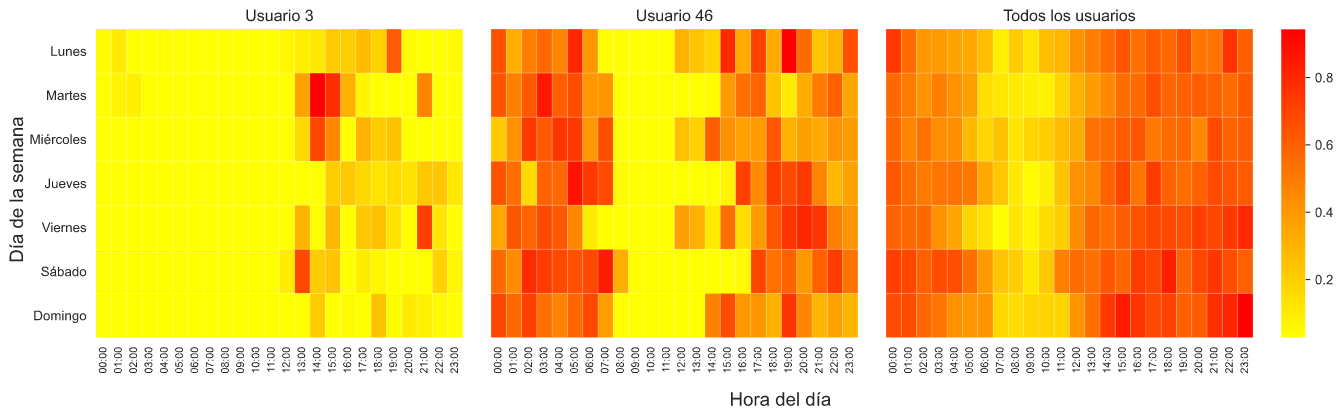
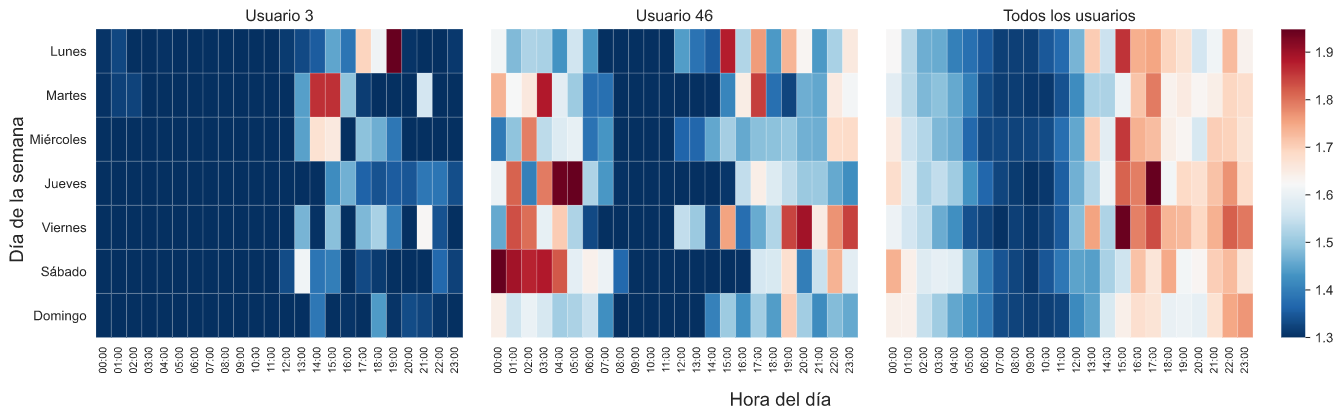
La figura X muestra, de manera similar a las figuras mostradas en la sección anterior, dos gráficos de barras en los que se representa el promedio y la desviación estándar del nivel de MET, ordenados de acuerdo al promedio del nivel MET. Ambos estadísticos son computados a partir del nivel de MET de todos los buckets disponibles para cada usuario. Como puede observarse, el promedio de MET de cada usuario ronda los 1.5, es decir, el nivel de MET que delimita lo que se considera un comportamiento sedentario o no.



Si se observa el gráfico de barras que muestra la desviación estándar, podemos ver que crece aproximadamente proporcionalmente a medida que crece el promedio de MET. Esto puede deberse a que estos usuarios presentan más buckets en los cuales llevaron a cabo actividades físicas. Por lo tanto, esto explicaría la diferencia en la desviación estándar entre los diferentes usuarios, donde los que practican menos actividad física son aquellos que tienen el menor valor para este estadístico.

También, puede interpretarse este crecimiento en la desviación estándar proporcional a la media del gasto energético con respecto cuan rutinario es cada usuario. Por lo tanto, estos gráficos parecen mostrar que cuanto más activo es un usuario, menos rutinario es. Esta hipótesis puede comprenderse mejor si se muestra cuán activo es un usuario a lo largo de las horas del día y los días de la semana. Para ello, se generaron mapas de calor con el promedio y la desviación estándar del nivel de MET para todos los usuarios de acuerdo al día de la semana y las horas del día. Para llevar a cabo esto, se agruparon los *buckets* por dia de la semana (eje ) y la hora del dia (eje ). Los mapas de calor que muestran el promedio del nivel de comportamiento sedentario ya habían sido usados en otros trabajos pero sin tener en cuenta el nivel de MET. En esos trabajos relacionados [(He & Agu, 2016a)](https://paperpile.com/c/FQGDmo/axGRS);[(He & Agu, 2016b)](https://paperpile.com/c/FQGDmo/GSfPw);[(He & Agu, 2016c)](https://paperpile.com/c/FQGDmo/2NLm5) los mapas de calor fueron generados a partir del porcentaje de actividades clasificadas como estacionarias en cada bucket de 1 hora. Los mapas de calor generados a partir de la desviación estándar de cada bucket de 1 hora no han sido aún analizados en otros trabajos relacionados.

En las figura X e Y, se muestran 3 mapas de calor, correspondientes a la media y a la desviación estándar respectivamente. El primer par de mapas corresponden al usuario 3, que es el segundo con menor MET promedio (después del usuario 39, que no fue utilizado para realizar los mapas de calor debido a la poca cantidad de buckets disponible). El segundo usuario seleccionado fue el 46, que es el segundo con mayor MET promedio (después del usuario 52, que no fue utilizado para realizar los mapas de calor debido a las inconsistencias halladas en los datos de actividad reportados). El último par de mapas de calor corresponden a todos los usuarios. Es decir, para generarlos se utilizan todos los buckets disponible de todos los usuarios. Puede observarse que el usuario 3 tiene un comportamiento sedentario todos los días de la semana antes de las 13:00 horas. Mientras que el usuario 46 muestra una alto nivel de MET para las primeras horas del día, sobretodo la madrugada del sábado. Esto puede deberse a que dicho usuario asistiera a eventos sociales en esas horas o que realizaba ejercicio nocturno. Además puede comprobarse que estas actividades no eran llevabas a cabo todas las semanas ya que presentan una alta desviación estándar.



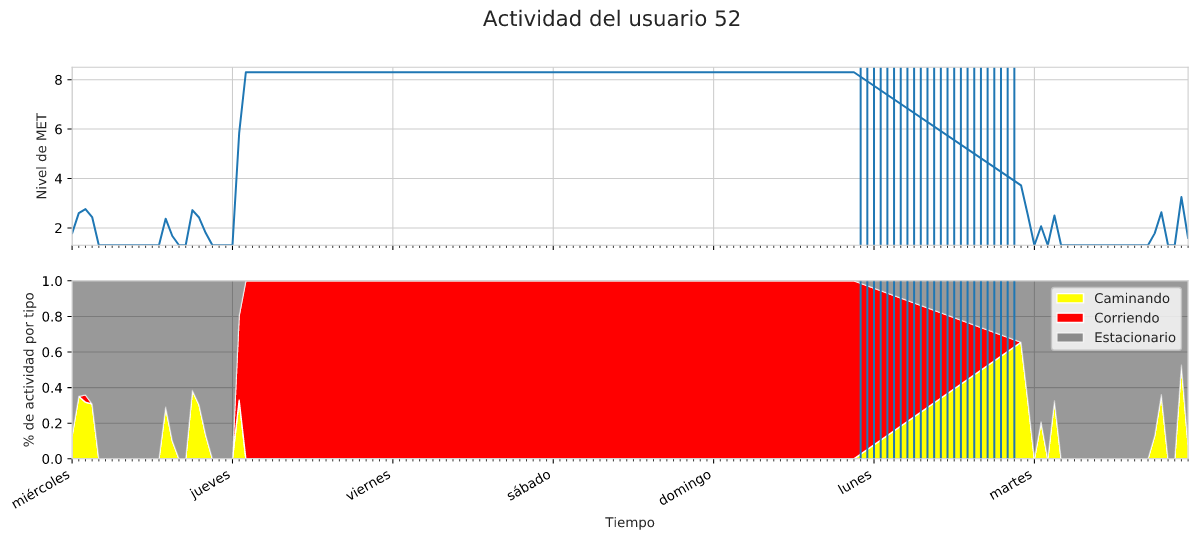
Como puede observarse, las horas del día con el mayor comportamiento sedentario (eso es, niveles de MET menores a 1.5) se corresponden, también, con una desviación estándar baja. Y además, las horas del día con el menor comportamiento sedentario (eso es, niveles de MET mayores 1.5) se corresponden, también, con una desviación estándar alta. Este fenómeno puede notarse tanto en los usuarios 3 y 46, como en el caso general para todos los usuarios. Una baja desviación estándar sugiere que el comportamiento sedentario observado en estos mapas de calor corresponde a un comportamiento de rutina, mientras que el comportamiento no sedentario no forma parte de la rutina de los usuarios debido a la alta desviación estándar.

Para sostener la hipótesis anterior, se realizó el cómputo de la correlación de *Pearson* entre la media y la desviación estándar para todos los usuarios y para todas las posibles combinaciones hora/dia de la semana. Luego, se calculó el promedio de la correlación y resultó ser de 0.87, lo que sirve de soporte a la observación sobre que el comportamiento sedentario está correlacionado positivamente con el comportamiento rutinario. Dado que los usuarios que participaron en los experimentos son estudiantes, esta correlación puede ser explicada por el hecho de que dos posibles comportamientos rutinarios que tienen un bajo nivel de gasto energético son las horas de sueño y los horarios de los cursos. La información sobre las clases a las cuales los usuarios asistieron forman parte del dataset, por lo que se verificó que los horarios de cursada de los estudiantes se corresponden con los horarios de bajo gasto energético. En resumen, los usuarios tienden a ser más sedentarios en actividades rutinarias que en actividades no rutinarias. Esta observación deja lugar a la hipótesis de que el comportamiento sedentario es más predecible que el comportamiento no sedentario en términos de variables de tiempo.

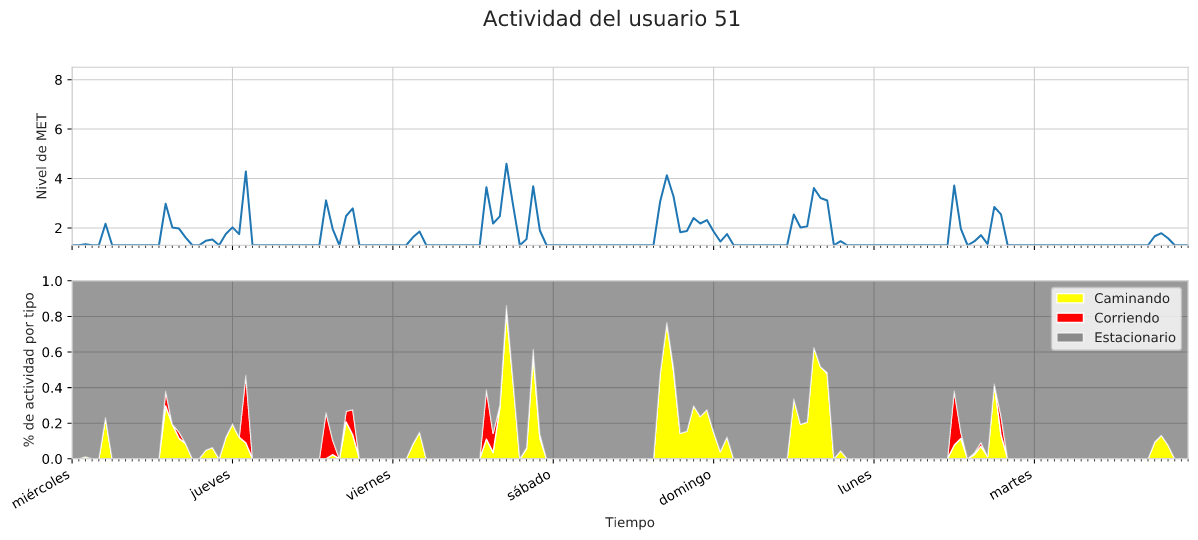
## Inconsistencias encontradas

Puede verse que el usuario 52 se diferencia de los demás en ambos gráficos, ya que posee un promedio de más de 2 METs y una desviación estándar mucho más elevada que la de los demás, cercana a 2 METs. Esto quiere decir que, en promedio, el gasto energético de cada hora de este usuario se aleja en 2 METs de la media. Por lo tanto, puede deducirse que este usuario tiene muchas horas en las que presenta un alto gasto energético. Si se analizan gráficos en los que se muestre que actividad llevó a cabo este usuario a lo largo del tiempo pueden apreciarse inconsistencias. Por ejemplo, en la figura X, se muestra el porcentaje de cada actividad (como suma acumulativa) en el periodo de 2013-05-22 al 2013-05-28. Puede observarse que, según los datos disponibles para este usuario en ese periodo de tiempo, el usuario 52 estuvo corriendo por 4 dias, sin parar, lo cual es imposible y se deba posiblemente a un error en la aplicación de senseo Jigsaw. Además, en la figura, se muestran líneas verticales para cada hora en la que no se posee información de actividad del usuario. Como puede observarse, a continuación de la inconsistencia anterior no se dispone de ningún bucket del día lunes.

Las inconsistencias detalladas en el párrafo anterior se repiten en diferentes porciones de los datos disponibles del usuario 52. Por esta razón, se decidió eliminarlo y no utilizarlo para entrenar los modelos propuestos. Esta decisión se toma en base a que los algoritmos que utilicen los datos de este usuario para realizar predicciones están aprendiendo a partir de una distribución que no corresponde ni al del usuario 52, ni al de ningún otro usuario, por lo que empeora el desempeño de los modelos.



A modo de comparación, en la figura Y se muestra un ejemplo de la actividad llevada a cabo en una semana por el usuario 51. Puede observarse claramente que hay muchos espacios temporales donde el usuario se encuentra totalmente estacionario, lo que puede deberse tanto a horas de sueño como horas de ocio o asistencia a cursos. De hecho, en la figura se observa que las primeras horas del día son, en general, completamente estacionarias, con la excepción del sábado donde el usuario pudo haber llevado a cabo alguna actividad social. Este tipo de patrón es el que está presente en la mayoría de los usuarios.



## Pipeline del preprocesamiento

En esta sección se describe la serie de etapas que atraviesa el procesamiento del dataset StudentLife. Las etapas son: generación de características, tratamiento de variables categóricas, generación de datasets con retrasos, ajustes en la dimensionalidad de los casos de entrenamiento, separación entre casos de entrenamiento y testeo y, por último, separación entre características y variable objetivo.

## Selección y cómputo de características

En esta sección, se definen las características que se generarán a partir del dataset StudentLife. Cada una de las características fue seleccionada por alguna/s de la/s siguiente/s razón/es.:

* Características generadas en base a investigaciones previas [(“Activity Prediction,” 2015; He & Agu, 2016b)](https://paperpile.com/c/FQGDmo/GSfPw+khD0k)
* Características generadas en base al análisis realizado en la sección anterior.
* Características que no incluyen información directamente ligada a la vida estudiantil, con la finalidad de que los resultados puedan ser más universales y no únicamente ligados a los estudiantes.

A su vez, se definen dos tipos de características: aquellas que fueron generadas a partir de los sensores de tipo TSD (Tipo de sensado discreto) y aquellas que fueron generadas a partir de sensores TSI (Tipo de sensado por intervalo). Estos dos tipos de características fueron llamadas Características Discretas y Características por Intervalo, respectivamente. La información utilizada para computar cada una de las siguientes características está ligada al tamaño de bucket que se esté utilizando (ej.: 30 min, 1h., etc.).

## Características Discretas

### Características obtenidas a partir del GPS [(Saeb et al. 2015; Saeb et al. 2016)](https://paperpile.com/c/FQGDmo/TbLIy+ysvF):

* *location\_variance*: utilizado para medir la variabilidad en la ubicación GPS de un participante en cada bucket.
* *location\_mean*: similar a la característica anterior, pero con respecto al promedio.
* *speed\_mean*: promedio de la velocidad instantánea, cómo se computa en [(Saeb et al. 2015)](https://paperpile.com/c/FQGDmo/TbLIy).
* *speed\_variance*: varianza de la velocidad instantánea, cómo se computa en Saeb 2016.
* *total\_distance*: desplazamiento geográfico total.

Para el cálculo de las últimas 3 características se tomaron una serie de decisiones importantes, que se detallan a continuación.

Primero, en todos los casos en los que es necesario calcular las diferencias entre dos registros consecutivos (por ejemplo, entre y ) , hay casos en que el registro anterior al actual no existe. Esto se da en los casos en que el registro es el primero de un usuario y cuando es el primero del bucket. En el caso en el que el registro es el primero del usuario, se tomó la decisión de eliminar el registro, ya que no hay forma de realizar los cálculos. Para el caso en el que el registro sea el primero del bucket, se decidió encontrar una manera de no descartar el registro en cuestión, ya que es muy baja la cantidad de registros disponibles para el sensor GPS. La solución hallada consiste en utilizar el último registro del bucket anterior, lo que por un lado puede parecer erróneo, ya que se está mezclando la información de dos buckets diferentes, pero por otro lado estos dos registros (el primero del bucket y el último del bucket ) están estrechamente relacionados ya que se da en muchos casos en los que se aporta información complementaria al bucket en cuestión. Tomando esta última decisión, se descartan 49 registros (uno por usuario), en lugar de 59.223 (una por cada bucket disponible para el sensor de GPS, como se muestra en la tabla x).

Segundo, para el cálculo de speed\_variance, cuando se calcula la varianza, , si el bucket solo contiene solo un registro y la velocidad espontánea para ese registro es 0 (posiblemente porque el usuario permanece siempre en el mismo sitio) se da que , que da como resultado indeterminado. La forma de expresar computacionalmente este valor es NaN (not a number, no es un número). Obviamente, estos valores no pueden ser almacenados en el dataset generado, por lo que se tomó la decisión de reemplazar estos valores nan por (cero).

Por último, como sucede con muchos de los sensores, hay bucket para los cuales no hay ningún registro disponible. Como ya ha sido discutido, en algunos casos es posible interpretar la ausencia de datos de manera que esos buckets no deben ser descartados. Según la descripción del motor de sensado Jigsaw [(Lu et al. 2010)](https://paperpile.com/c/FQGDmo/KkVi), con el objetivo de utilizar la menor porción de batería, al mismo tiempo que minimizando el error del GPS, este sensor puede permanecer inactivo hasta que se detecte que el usuario está en movimiento a partir del acelerómetro. Por lo tanto, puede interpretarse que para los buckets para los cuales no existen registros del sensor GPS, el usuario permanece en el mismo sitio que la hora anterior. De esta forma, las características generadas a partir del sensor GPS para los cuales no existen registros disponibles se completa de la siguiente manera:

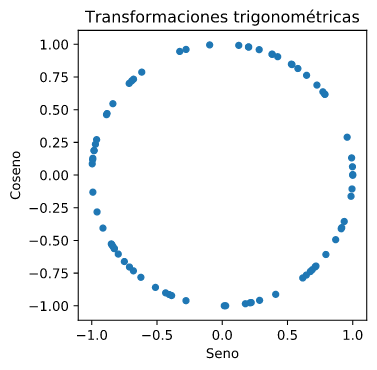
* *location\_variance*: se completa con 0.
* *location\_mean*: se completa con el valor de esta característica para la hora anterior.
* *speed\_mean*: se completa con 0.
* *speed\_variance*: se completa con 0.
* *total\_distance*: se completa con 0.

### Características basadas en el tiempo:

* *second\_sine*: transformación de seno de la cantidad de segundos pasados a partir del comienzo del dia.
* *second\_cosine*: transformación de coseno de la cantidad de segundos pasados a partir del comienzo del dia.
* *weekday\_sin*: transformación de seno del día de la semana.
* *weekday\_cos*: transformación de coseno del día de la semana.

Estas últimas 4 características tienen como objetivo darle a los modelos de Machine Learning la posibilidad de comprender la naturaleza cíclica de las variables relacionadas al tiempo. Sin realizar esta transformación, los modelos de Machine Learning no tienen la información necesaria para comprender que entre el primer valor posible y el ultimo existe la misma distancia que entre cualesquiera otros dos valores consecutivos. Es decir, en el caso de los días de la semana, la diferencia entre martes y miércoles, es la misma que entre domingo y lunes. El problema subyace en que en informática, los días de la semana se expresan como números naturales - los cuales no son cíclicos- donde el número 0 representa al domingo y el número 6 al sábado (esto puede variar dependiendo de las implementaciones).

La transformación llevada a cabo consiste en aplicar por un lado el seno, y por otro el coseno, a la variable cíclica estandarizada (con valores entre 0 y 1) multiplicada por 2 radianes (2). Es decir, primero se transforma a la variable categórica como un ángulo entre 0 y 2, y luego se le aplican las dos funciones trigonometricas. De esta forma, observando el seno como el coseno, el valor del primer valor es similar al último valor, ya que ambas funciones trigonométricas son periódicas. Aunque con una de las dos funciones alcanzaría para plasmar el carácter cíclico de las variables de tiempo, se da el fenómeno de que hay diferentes partes del día que no se diferencian entre sí. Por ejemplo, con respecto al tiempo pasado desde el comienzo del día, si se utiliza el seno, las 20 horas tienen el mismo valor que las 0 horas. El mismo fenómeno se da para el coseno y las demás variables, como los días de la semana. Por lo tanto, es importante usar ambas transformaciones para completar la información y permitir a los modelos diferencias todos los momentos del días, a su vez que permitir deducir que las variables son periódicas. En la Figura X se muestra un gráfico de dos dimensiones, donde el eje representa el valor de transformar la cantidad de segundos pasados desde el comienzo del día a partir del coseno, mientras que el eje muestra la misma transformación pero a partir del seno. Como puede observarse los puntos toman la forma del círculo o de un reloj, denotando así su carácter periódico.



* *past\_minutes*: el número de minutos transcurridos desde el comienzo del día.
* *remaining\_minutes*: el número de minutos que quedan para terminar el día.

Como se mostró en el análisis del dataset, las características temporales pueden llegar a tener un rol muy importante al momento de predecir el comportamiento sedentario de un usuario, debido a que, por lo general, estos siguen patrones temporales. Estos patrones ya fueron explorados en otros trabajos relacionados y es posible que la redes neuronales puedan sacar provecho de eso.

### Características de actividad física:

* *stationary\_count*: la cantidad de instancias de actividad física clasificadas como ‘estacionario/a’ en cada bucket.
* *walking\_count*: la cantidad de instancias de actividad física clasificadas como ‘caminando’ en cada bucket.
* *running\_count*: la cantidad de instancias de actividad física clasificadas como de ‘corriendo’ en cada bucket.
* *total\_activity\_count*: la cantidad de instancias en cada bucket.
* *activity\_major*: el tipo de actividad física con más instancias en cada intervalo.

### Características de audio:

* *silence\_count*: la cantidad de instancias de audio clasificadas como ‘silencio’ en cada intervalo.
* *voice\_count*: la cantidad de instancias de audio clasificadas como ‘voz’ en cada intervalo.
* *noise\_count*: la cantidad de instancias de audio clasificadas como ‘ruido’ en cada intervalo.
* *total\_audio\_count*: la cantidad de instancias de actividad en cada bucket.
* *number\_of\_conversations*: el número de conversaciones que ese estudiante tuvo en cada intervalo.

### Otras características

* *wifi\_changes*: el número de cambio de conexiones wifi en cada intervalo.

## Características por intervalo

* *is\_charging*: si el *smartphone* se estaba cargando.
* *charging\_ratio*: representa la proporción del *bucket* en la cual el *smartphone* se estaba cargando.

Para realizar este cálculo, se divide, iterativamente, cada intervalo que supere el tamaño de bucket en nuevos intervalos más pequeños, hasta que todos los intervalos tengan un tamaño menor o igual al tamaño de *bucket*. Una vez hecho esto, para cada intervalo se calcula la proporción que ocupa del *bucket* al que pertenece. Luego, se dividen todos los intervalos en grupos de acuerdo al id del usuario y al *bucket* al que pertenecen (por ejemplo: si los buckets son de una hora y el intervalo comienza en 14:34:20, se utiliza la función piso para obtener el *bucket* correspondiente, que da como resultado 14:00:00). Finalmente, se suman las proporciones de cada intervalo perteneciente a cada grupo en particular.

Al calcular esta categoría se descubrieron nuevas inconsistencias en los datos (sobretodo en el caso del usuario 49), ya que el proceso de realizar los cálculos existían buckets para los cuales la proporción en la cual el smartphone se estaba cargando era mayor a 1. Realizando un análisis se descubrió que había intervalos de tiempo superpuestos. Por ejemplo, para el usuario 49 existe un intervalo que va desde 00:34 hasta 08:53, mientras que otro intervalo va desde 00:13 hasta 03:22.

En la descripción del dataset StudentLife se afirma que todos los intervalos tienen una duración mayor a 1 hora. Si se analiza el caso de los buckets de 1 hora, al momento de la agrupación debería existir solo un intervalo por grupo. Sin embargo, se hallaron en total 355 grupos con más de un intervalo, lo que quiere decir que existen 355 buckets de una hora que presentan inconsistencias debido a la superposición de intervalos. La cantidad de buckets que superan la proporción de 1 son 45, por lo que la mayoría de buckets con inconsistencias no superan la proporción total aunque los valores calculados resultan ser igualmente erróneos.

No es posible decidir cuál de los intervalos superpuestos es el correcto, por lo que la única forma de solucionar este problema sería eliminando los intervalos inconsistentes. Esta acción implicaría eliminar los buckets involucrados, lo cual reduciría el tamaño del dataset. Por lo tanto, se decidió no descartar estos buckets inconsistentes, y en lugar de ello, reemplazar por 1 (ocupación total del bucket) aquellos buckets cuya proporción era superior a 1, y dejando igual los demás buckets inconsistentes.

Para el caso de las demás características generadas a partir de sensores TSI se da la misma inconsistencia, en mayor o menor grado. El proceso que se utiliza es el mismo para todas ellas.

Para complementar esta característica de proporción se agrega también una característica booleana que indica si en un bucket determinado se halló uno o más intervalos.

* *is\_locked*: si el smartphone está bloqueado;
* *locked\_ratio*: representa la proporción del *bucket* en la cual el *smartphone* estuvobloqueado.
* *is\_in\_dark*: si el smartphone estaba en la oscuridad;
* *dark\_ratio*: representa la proporción del *bucket* en la cual el *smartphone estaba* en un ambiente oscuro (por ejemplo, en un bolsillo).
* *is\_in\_conversation*: si el usuario mantuvo o no una conversación.
* *conversation\_ratio*: representa la proporción del *bucket* en la cual el usuario se encontraba manteniendo una conversación.
* *conversation\_number*: cantidad de conversaciones diferentes manatenidas en el *bucket.*

### Variable objetivo

* *sLevel*: el valor de MET para el *bucket*.

Esta variable es llamada variable objetivo. Es decir, es la variables que los modelos de *Deep Learning* tratarán de predecir a partir de todas las demás características. A pesar de que *sLevel* es la variable objetivo, también es usada como característica. Es decir, como la tarea de predicción que se busca llevar a cabo en esta tesis es la de predecir el comportamiento sedentario futuro, el nivel de sedentarismo del momento actual y pasado es ya conocido, por lo que puede formar parte de los *datasets* de entrenamiento para ser usado por los modelos para aprender.

***Salida de esta etapa de preprocesamiento:*** Como salida de esta etapa del preprocesamiento se tiene un dataset producido a partir del dataset StudentLife. El dataset está compuesto por registros. Cada registro representa a una combinación usuario/bucket única. A su vez, cada registro está compuesto por 32 campos (uno por cada una de las características listadas en esta sección, incluyendo la variable objetivo).

***Representación matricial de la salida:*** 76.032filas y 32 columnas para una granularidad de una hora y 152.064 filas y 32 columnas para una granularidad de 30 minutos.

## Eliminación de valores nulos

En esta etapa de preprocesamiento se buscan y eliminan aquellos registros que presenten por lo menos un valor nulo en alguno de sus campos. Esto es necesario ya que los modelos utilizados en esta tesis necesitarán la presencia de todas las características para entrenarse y realizar predicciones. Es importante notar que la cantidad de registros que se tendrán para una granularidad de 1 hora será siempre mayor o igual a la mitad de la cantidad de registros que se tendrán para una granularidad de 30 minutos. Esto se debe a que la cantidad de registros que se pierdan al llevar a cabo la eliminación de valores nulos siempre será mayor o igual en una granularidad de 30 minutos. Esto se debe a que, dado un bucket válido de una hora, si se lo separa en dos buckets de 30 minutos, existe la posibilidad de que uno de ellos no sea válido y deba descartarse.

***Salida de esta etapa de preprocesamiento:*** Como salida de esta etapa del preprocesamiento se tiene un dataset con una estructura similar a aquella de la salida de la etapa anterior, solo que con menos registros.

***Representación matricial de la salida:*** 61.319 filas y 32 columnas para una granularidad de una hora y filas y 32 columnas para una granularidad de 30 minutos. Obsérvese que y .

## Tratamiento de variables categóricas

Como las redes neuronales no pueden recibir como datos de entrada variables de tipo categórica (sobretodo si estas no están representadas en forma numérica), es necesario utilizar un método para convertirlas o codificarlas en un tipo de variable que sea aceptable por el modelo. Además, no solo debe ser aceptable por el modelo, sino que tiene que representar de alguna manera la mayor información posible de tal manera que el modelo pueda utilizarla para realizar las predicciones. Una vez generadas las características, el dataset resultante posee dos características de tipo categóricas: *day\_of\_week* y *activity\_major*. Es importante notar que dichas variables son nominales y no ordinales, ya que no es posible determinar un orden entre ellas. Si estas variables fuesen nominales sería más fácil convertirlas, ya que compartían una característica con el resto de las variables numéricas, un orden entre ellas.

Hay muchos métodos que permiten codificar variables categóricas. Algunos de los más comunes son: Label Encoding, One-Hot Encoding, Dummy Encoding, Target Encoding, etc. La utilización de uno por sobre otro depende del dataset que se esté utilizando. A continuación se describe brevemente cada uno de estos métodos.

* One-Hot Encoding: consiste en generar tantas nuevas variables como cantidad de categorías diferentes tenga la variable categórica en cuestión. Si la variable categórica tiene valores distintos, las nuevas variables serán vectores pertenecientes donde el componente , con , del vector tomará el valor de 1 si la variable categórica tiene el valor para el caso . Se recomienda no usar One-Hot Encoding para casos donde la variable categórica tenga muchas categorías diferentes (ejemplo, >20) ya que se incurre en un problema llamado *curse of dimensionallity*. Es claro observar que cuantas más categorías haya, cada variable generada por este método aportará menos información.
* Dummy Encoding: muy similar a One-Hot Encoding, con la diferencia de que se generan nuevas variables, en lugar de nuevas variables en el proceso de codificación. La presencia de la última categoría se representa dejando en 0 el valor de todas las otras variables. Este método de codificación de variables categóricas tiene la ventaja de generar menos nuevas variables, aunque no es recomendable para métodos de aprendizaje basados en árboles de decisión, ya que estos no tienen forma de deducir la existencia de la última variable.
* Label Encoding: en este método, quizás el más simple, se reemplaza cada categoría por un valor numérico. Por ejemplo, cada categoría puede ser reemplazada por un número natural, o cualquier otro a elección. Este método de codificación es recomendable siempre y cuando la variable categórica sea ordinal. La ventaja de este método de codificación es que no aumenta el número de variables en el dataset.
* Target Encoding: en síntesis, este método consiste en reemplazar la ocurrencia de cada categoría por el valor promedio de la variable objetivo. Por ejemplo, si la tarea es de clasificación binaria, el valor por el cual se reemplace cada categoría de una variable categórica será un número real entre 0 y 1. Este método tiene varias implementaciones diferentes que buscan mejorarlo en términos de generalización, es decir, de reducir el *overfitting.* No es recomendable utilizar este método en el caso de que los datos de entrada de los modelos sean series de tiempo, ya que la naturaleza de este tipo de datos tiende a cambiar mucho entre los datos de entrenamiento y de testeo.

Finalmente, se decidió aplicar Dummy Encoding sobre las 2 variables categóricas presentes en el dataset procesado. Con esta transformación, se añadieron 6 nuevas características y se descartaron 2. Esta es la última transformación que se les hace a cada *bucket* en particular.

***Salida de esta etapa de preprocesamiento:*** Como salida de esta etapa del preprocesamiento se tiene un dataset donde cada registro está compuesto por campos. La cantidad de registros no se ve modificada.

***Representación matricial de la salida:*** 61.319 filas y 36 columnas para una granularidad de una hora y filas y 36 columnas para una granularidad de 30 minutos.

## Generación de datasets con retrasos

El siguiente paso del procesamiento consiste en generar los datasets con retrasos. En esta etapa del procesamiento se prepara el dataset para que pueda ser utilizado para entrenar y testear los modelos. Dos parámetros intervienen en este paso: la cantidad de lags y el valor de los períodos. Cuando se realicen los experimentos, diferentes valores para estos parámetros serán evaluados en pos de estudiar cuál de ellos es el que alcanza un mejor desempeño y cuáles funcionan mejor con cada arquitectura.

Para llevar a cabo esta etapa del procesamiento, se toma cada uno de los buckets disponibles como si representara el momento presente. A continuación, se prepara la lista de buckets anteriores a que serán utilizados a partir de su ubicación temporal relativa de . Para preparar dicha lista, se tienen en cuenta la cantidad de lags y el valor del periodo. Esta lista va a definir de qué información se valdrán los modelos para predecir el nivel de MET del momento presente, es decir, el valor de MET de .

A continuación, se concatenan los buckets seleccionados al buckets actual, al mismo tiempo que se eliminan todas las características del bucket actual salvo la variable objetivo. La cantidad de unidades de tiempo que cada registro del dataset abarca es de . También es posible calcular el intervalo de tiempo que abarca cada registro de un dataset determinado. Dicho intervalo está dado por . Por lo tanto el intervalo de tiempo que abarca cada registro es mayor o igual a la cantidad de unidades de tiempo de la cual posee información. Otro punto importante a notar es que la cantidad de característica que posee cada registro del dataset es , por lo que la cantidad de características crece directamente proporcional al número de lags. Esto puede traer problemas si la cantidad de casos de entrenamiento es baja ya que puede presentarse el problema conocido como *curse of dimensionality* o la maldición de la dimensionalidad, donde el espacio dimensional es tan grande que no se dispone de la cantidad suficiente de casos de entrenamiento como para que se posean ejemplos regularmente espaciados de forma que dicho espacio dimensional quede relativamente ocupado. Gracias a la introducción del período, los modelos pueden obtener información de un pasado lejano sin aumentar el número de características.

En la figura X pueden observarse 3 ejemplos de registros de diferentes datasets con retrasos. Las celdas verdes representan buckets utilizados por el modelo, mientras que el bucket rojo es el que representa el momento presente y del cual el modelo va a intentar predecir el nivel de MET. En el dataset tiene 8 lags y un periodo de 1. En el dataset tiene 2 lags y un periodo de 2. En el dataset tiene 4 lags y un periodo de 4. Suponiendo una granularidad de 1 hora, puede apreciarse el efecto de la utilización de diferentes valores del período. Por ejemplo, aunque en el dataset tiene el doble de lags que en , el intervalo de tiempo que es abarcado en es de 16 horas, la mitad que en . Los buckets seleccionados en cada caso son concatenados y dados de entrada al modelo para realizar una predicción del MET , llámese . Luego, el MET de , llámeselo , es utilizado para estudiar cuán acertada fue la predicción . Esto se lleva a cabo a partir de una función de error, como MSE.

Es importante recordar que muchos de los *buckets* se encuentran ausentes ya que una o más de las características no están presentes. De esta forma, si alguno de los buckets que forman parte de la lista de *buckets* a concatenar no está disponible, el registro entero se descarta. Por lo tanto, cuanto mayor sea la cantidad de *buckets* pertenecientes a dicha lista, mayor la probabilidad de que la lista entera deba ser descartada por la ausencia de uno de los *buckets* pertenecientes a ella.

En la práctica, el proceso de generar los datasets con retrasos se lleva a cabo como la traslación del dataset completo en una sola operación y no de manera iterativa, ya que de esta manera el proceso paralelizable y mucho más rápido. Entonces, se genera una lista con los tiempos - por ejemplo: [16, 12, 8, 4] en el caso de de la Fig. X- relativos a partir del nro. de lags y el valor del período. Luego, para cada elemento de esa lista se realiza la traslación del dataset. Luego, todos los datasets son concatenados.

***Salida de esta etapa de preprocesamiento:*** Como salida de esta etapa del pre-procesamiento se obtiene un dataset donde cada registro está compuesto por la variable objetivo que representa al momento presente y las características de uno o más buckets del pasado.

***Representación matricial de la salida:*** En este punto del preprocesamiento, la cantidad de filas variará de acuerdo al número de lags y del valor del periodo. Diferentes combinaciones tendrán diferencias en cuanto a la cantidad de filas disponibles. Además, el número de lags determinará la cantidad de columnas de la matriz. En este punto, el dataset está representado por una matriz de dimensiones , donde es el número de filas. Por ejemplo, si el número de lags es de 4 y el periodo es 2 se tiene una matriz de para una granularidad de un hora y de para una granularidad de 30 minutos. Por dar otro ejemplo, si el número de lags es de 2 y el periodo es 1 se tiene una matriz de para una granularidad de un hora y de para una granularidad de 30 minutos. Nótese cómo al aumentar el número de lags disminuye la cantidad de filas y aumenta la cantidad de columnas.

## Separación entre X e y

En este punto, se procede a separar al dataset entre las características -que serán utilizadas por los modelos para predecir- y la variable objetivo. Respetando la convención en el área de Machine Learning, se llama al conjunto resultante de características e al vector cuyos componentes representan el valor de la función objetivo para cada caso.

***Salida de esta etapa de preprocesamiento:*** Como salida de esta etapa del preprocesamiento se tiene un dataset de características y un vector de la variable objetivo .

***Representación matricial de la salida:***  tendrá una dimensión de e tendrá una dimensión de .

## Separación entre entrenamiento y testeo

Esta etapa del preprocesamiento consiste en separar el dataset en 2 conjuntos. A diferencia de la etapa anterior, donde se separó al dataset en su dimensión horizontal (es decir, a nivel de columnas), en esta etapa se lo separa en su dimensión vertical (a nivel de filas). El primero de los conjuntos es llamado conjunto de entrenamiento y, el segundo, conjunto de testeo. El primero de ellos será utilizado para entrenar al modelo de Deep Learning, mientras que el segundo será utilizado para evaluarlo, es decir, para medir el desempeño alcanzado en términos de su capacidad de generalización. De esta forma, el dataset original queda dividido en 4 partes, ya que tanto el conjunto de características, conocido como , y el vector de la variable objetivo, conocido como , son divididos en conjuntos de entrenamiento y de testeo. Estos 4 conjuntos serán llamados e .

Como será explicado más adelante, la evaluación de los modelos de *Deep Learning* fue realizada mediante una técnica especialmente aplicada para el problema que se ataca en esta tesis. Por un lado, se tuvo que tener en cuenta que el *dataset* es de tipo *time-series,* y por otro, que se probaran modelos personales e impersonales. Estos dos últimos puntos impactan en la decisión correcta sobre cómo evaluar los modelos. De esta forma, la evaluación de los modelos que se llevó a cabo en esta tesis escapa de la simple separación entre casos de entrenamiento y de testeo. En la sección X se verá que los datasets de entrenamiento y de testeo pueden provenir de conjuntos de datos diferentes.

***Salida de esta etapa de preprocesamiento:*** Como salida de esta etapa del preprocesamiento se tiene un total de 4 conjuntos de datos diferentes: un *dataset* de características que posee los casos de entrenamiento, un dataset de características que posee los casos de testeo, un vector con los valores de la variable objetivo para los casos de entrenamiento y un vector con los valores de la variable objetivo para los casos de testeo.

***Representación matricial de la salida:*** Las dimensiones de las matrices serán las mismas que en la sección anterior.

## Normalización

Esta etapa del preprocesamiento consiste en normalizar los datos de los datasets que contienen los casos de entrenamiento y testeo -esto es, y-. El proceso de normalización tiene la finalidad última de agilizar el proceso de optimización de los parámetros del modelos. Esto se logra normalizando los valores de cada una de las características, es decir, haciendo que todas las características numéricas se encuentren entre el mismo rango de valores.

Para este paso del preprocesamiento se utilizan normalizadores provistos por la librería Scikit-learn5. El proceso de normalización se basa en que un normalizador aprenda la transformación a realizar a partir de los datos de entrenamiento y, luego, con la transformación aprendida, aplicarla a los *datasets* de entrenamiento y testeo. El motivo por el cual se aprende la transformación de los datos sólo a partir del conjunto de entrenamiento es evitar la fuga de datos (*data leakage*). El fenómeno *data-leakage* debe ser siempre tenido en cuenta para evaluar la capacidad de generalización de los predictores. En esta tesis se usa un normalizador conocido como *StandardScaler[[14]](#footnote-13).* *StandardScaler* consiste en calcular la media y la desviación estándar de cada característica. Luego, a todos los valores de una característica dada se le resta la media y se lo divide por la desviación estándar.

***Salida de esta etapa de preprocesamiento:*** Como salida de esta etapa del preprocesamiento se obtienen los datasets e normalizados. Los vectores de la variable objetivo no son utilizados en esta etapa.

***Representación matricial de la salida:*** Las dimensiones de las matrices serán las mismas que en la sección anterior.

## Ajuste de las dimensiones de X

Esta etapa del preprocesamiento es aplicada sobre los datasets e solo en el caso en que ciertos modelos de *Deep Learning* los vayan a utilizar. Algunos modelos esperan que su entrada sean vectores y otros esperan matrices. Dicho de otra forma, en el primer caso, el *dataset* deberá estar representado por una matriz, mientras que en el segundo, el dataset deberá estar representado por un tensor. Un tensor es la extensión del concepto de matriz a un espacio de dimensiones. Por diversas características, esta representación de un alto número de dimensiones puede beneficiar a algunos tipos de modelos, ya que estos pueden aprovechar mejor la estructura espacial de los datos de entrada.

En el caso de las redes Perceptrón Multicapa, se espera como entrada un vector, por lo que el dataset estará representado por una matriz donde las filas representan los casos y las columnas las características. Esta última representación es la que se ha mantenido a lo largo de todo el preprocesamiento.

Otros tipos de modelos, como las CNN o las RNN pueden recibir como entrada matrices o incluso tensores. En lo que respecta a esta tesis, los modelos que no sean MLP esperan como entrada una matriz, por lo que el dataset deberá estar representado por un tensor de 3 dimensiones (una más que las matrices). Estas 3 dimensiones representan: los casos, las características y los lags de dichas características. Con esta organización de los datos, los modelos pueden sacar provecho de la característica temporal de dichos datos. Esto es así porque gracias a esta disposición los datos de una misma característica son contiguos entre dos lags contiguos, hecho que no se da en la representación de matrices. De esta forma, es más fácil para los modelos -aquellos que puedan utilizar datos organizados de acuerdo a esta disposición- aprender características relacionadas a la dimensión temporal. Por ejemplo, la relación entre las características relacionadas a la actividad en dos o más lags consecutivos.

***Salida de esta etapa de preprocesamiento:*** Como salida de esta etapa del preprocesamiento se obtiene una versión remodelada de los datasets e en el caso en que no vayan a ser utilizados por una arquitectura MLP. Los vectores de la variable objetivo no son utilizados en esta etapa.

***Representación matricial de la salida:*** Las dimensiones de las matrices e serán o bien .

## Validación de los modelos

Como ya fue mostrado en el marco teórico, existen muchos métodos utilizados para validar modelos de *Machine Learning*. El objetivo de la validación es verificar el desempeño del modelo. Para validar los modelos debe discutirse cuál de los métodos existentes es el más adecuado para ser utilizado en el problema que motiva esta tesis. Como los datos que serán utilizados para entrenar y testear los modelos propuestos son de tipo *time-series,* no todos los métodos de validación son aplicables, ya que deben cumplir con ciertas características.

En la gran mayoría de los métodos de evaluación se debe dar por supuesto que las variables del *dataset* son independientes e idénticamente distribuidas, lo cual no puede darse por supuesto en el caso de *time-series,* ya que claramente no pueden intercambiarse casos entre sí sin modificar la distribución del *dataset.*

Además, el método de validación a utilizar debe cumplir con la característica de adaptarse a la forma real en la cual los datos son registrados y puestos a disposición de los modelos. Es decir, todos los datos que se usan para entrenar un modelo deben ser anteriores a aquellos datos utilizados para testearlo. De esta forma, se simula una situación real en la cual los datos del futuro no están aún disponibles.

Por último, el método a utilizar debe adaptarse para los diferentes experimentos propuestos en esta tesis. Sobre todo, debe poder adaptarse tanto a los modelos personales como a los impersonales. Esto es necesario ya que los datos utilizados por los modelos para ser entrenados y testeados pueden o no pertenecer al mismo *dataset*.

Normalmente, para evaluar modelos entrenados a partir de *datasets* de tipo *time-series* se utiliza un método de validación cruzada para series de tiempo llamado *TimeSeriesSplit*, disponible en la librería *Scikit-Learn*. *TimeSeriesSplit* consiste en dividir el *dataset* en *k* particiones, respetando el orden temporal, donde cada partición tiene la misma cantidad de casos. Luego, se llevan a cabo iteraciones en las cuales se re-entrena el modelo, con la particularidad de que en cada iteración el conjunto de casos de entrenamientos es un superconjunto del conjunto de casos de la iteración anterior. Más precisamente, el conjunto de casos de entrenamiento de la iteración es igual a la unión entre los conjuntos de entrenamiento y de testeo de la iteración Entonces, en la iteración se utilizan las primeras particiones para entrenar al modelo y la partición para testearlo. El resultado final de este método de validación es la media del desempeño alcanzado por el modelo sobre el *dataset* de testeo en cada iteración. En algunos casos, la desviación estándar puede ser un estadístico útil.

El método de validación descrito en el párrafo anterior no es adecuado para ser utilizado en esta tesis por dos razones:

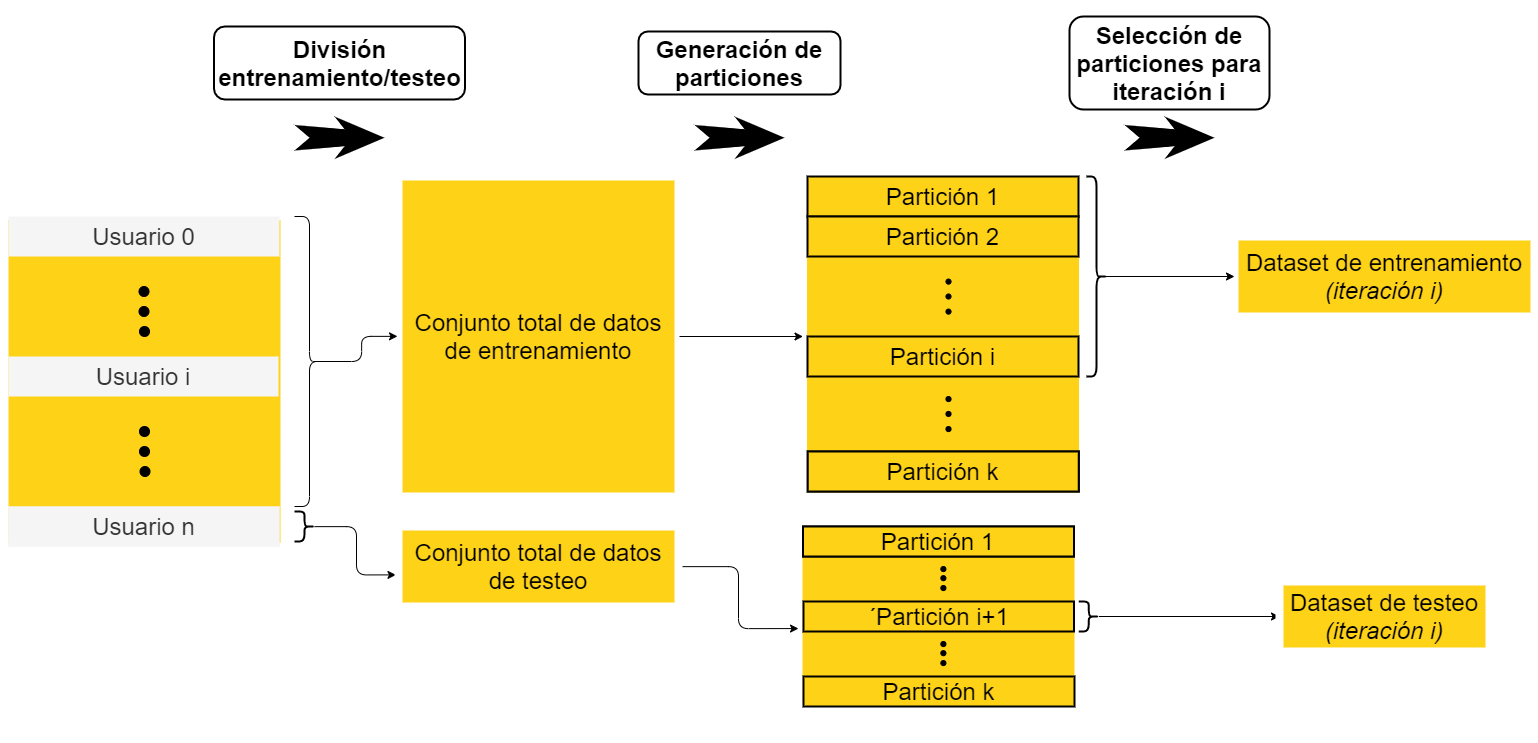
* Da por supuesto que los casos son observaciones en intervalos fijos de tiempo, lo cual no es aplicable a los *datasets* obtenidos a partir del *pipeline* de preprocesamiento, donde en dos intervalos de tiempo de igual longitud puede contener un cantidad diferente de casos.
* No permite recibir dos *datasets* (uno de entrenamiento y otro de testeo) a partir de los cuales se podrían modelar los modelos personales e impersonales.
* Para incluir tanto modelos personales como impersonales, es necesario que el método de evaluación pueda realizar el entrenamiento y testeo de los modelos a partir de dos *datasets* diferentes. Este requerimiento se da en el caso de los modelos impersonales, donde el *dataset* de entrenamiento pertenece a un *dataset* que incluye datos sobre muchos usuarios, mientras que el *dataset* de testeo contiene información sobre un único usuario.

Por lo tanto, se propone un nuevo método basado en *TimeSeriesSplit* que resuelve los problemas por los cuales este último no es aplicable en el problema de la predicción del comportamiento sedentario futuro utilizando el *dataset StudentLife*. El método de evaluación propuesto difiere en lo siguiente de método *TimeSeriesSplit:*

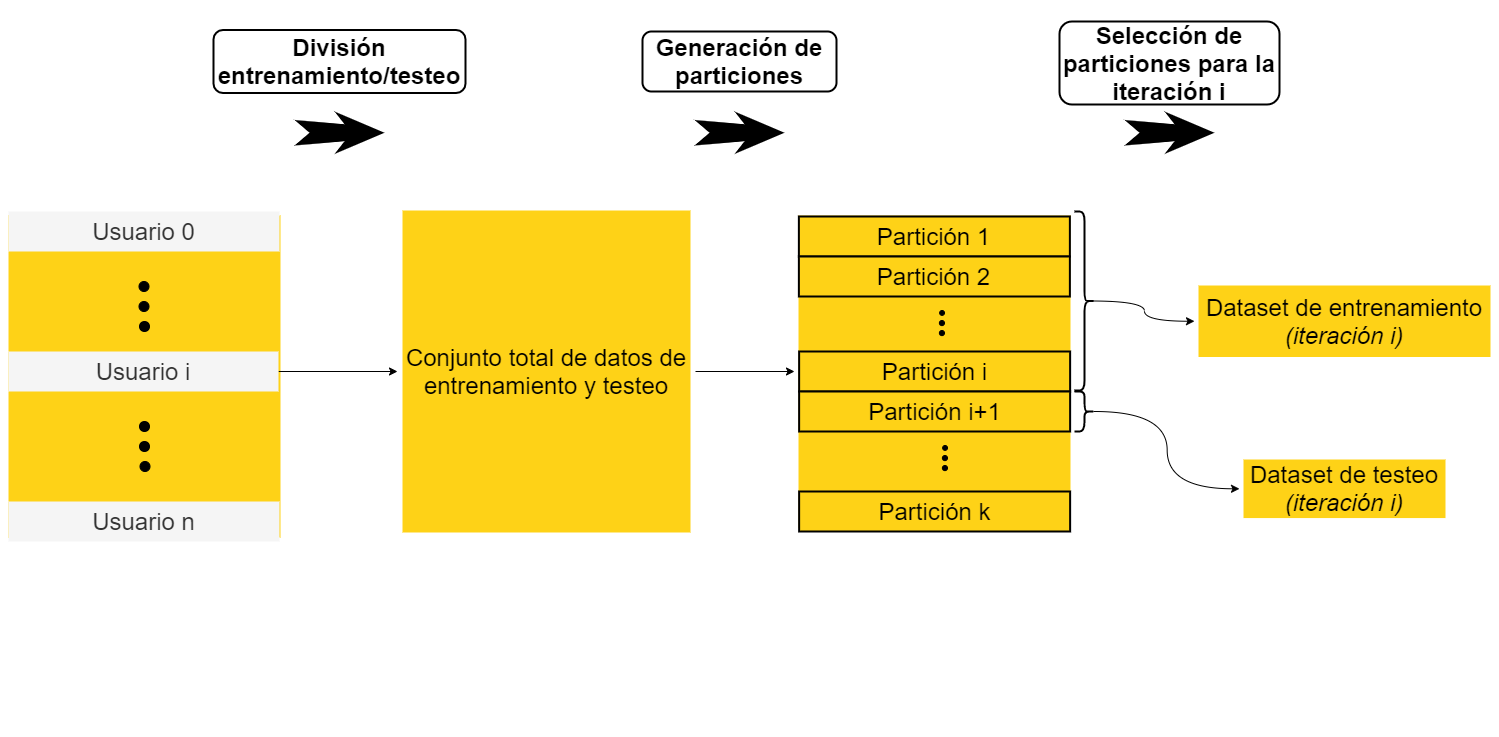
* El criterio para generar las particiones es que estas deben representar el mismo intervalo de tiempo y no la misma cantidad de casos. Para esto, se calculan las fechas mínimas y máximas del *dataset.* Luego, se calcula la longitud del intervalo temporal comprendido entre las dos fechas calculadas (por ejemplo 65 días y 4 horas) y se lo divide por la cantidad de iteraciones , obteniendo así la longitud de tiempo que aborda cada partición. Una vez obtenidas estas particiones, se prosigue de la misma manera que en *TimesSeriesSplit.*
* Para permitir que el método se adapte a los modelos personales e impersonales, éste recibe como parámetro dos *datasets* diferentes, uno para el entrenamiento y otro de testeo. De esta forma, en el caso de los modelos personales, los *datasets* de entrenamiento y de testeo pertenecen al mismo usuario, mientras que en el caso de los modelos impersonales el *dataset* de entrenamiento contiene los datos de todos los usuarios salvo el del usuario objetivo mientras que el *dataset* de testeo contiene los datos del usuario objetivo.

Los pasos a seguir por el algoritmo de validación anteriormente enunciado varían de acuerdo a si el modelo es personal o impersonal, ya que esto determina de qué usuarios provienen los datos de entrenamiento y de testeo.

En las figura X se muestra el caso general de validación para un modelo impersonal. En éste, se parte del *dataset* entero, que posee los datos de todos los usuarios. El primer paso consiste en seleccionar los *datasets* de entrenamiento y de testeo, donde el segundo corresponde a los datos de un usuario, y el primero a los datos de todos los demás usuarios. En este caso, el *dataset* de testeo corresponde a los datos del usuario y el de entrenamiento a los datos del resto de los usuarios. A continuación, se generan particiones de cada uno de los *datasets*. Las particiones con el mismo índice de ambos *datasets* corresponden al mismo intervalo temporal. Por último, se determinan los *datasets* de entrenamiento y de testeo para la iteración . El *dataset* de entrenamiento de la iteración estará compuesto por la unión de todas las particiones con índice menor o igual a , de la forma , del *dataset* de entrenamiento completo. El *dataset* de testeo de la iteración estará compuesto por la partición del *dataset* de testeo completo.



Por su parte, en las figura Y se muestra el caso general de validación para un modelo personal. En éste, al igual que en los modelos impersonales. se parte del dataset entero. Luego, se selecciona el conjunto del dataset del usuario en cuestión que corresponderá al mismo tiempo al al dataset de entrenamiento y de testeo. He aquí la mayor diferencia entre ambos tipos de modelos, ya que en el caso de los modelos personales tanto el dataset de entrenamiento como el de testeo forman parte del mismo conjunto de datos. A continuación, se generan las particiones. Para la iteración se seleccionan las primeras particiones para entrenar el modelos y la particiones para entrenar.



## Métricas de evaluación

La métrica utilizada para evaluar el desempeño de los modelos propuestos es el Error Cuadrático Medio o MSE (Mean Squared Error). El MSE es una medida de desempeño cuantitativa utilizada comúnmente para medir el error que hay entre el valor real y el valor estimado. En este contexto MSE es igual a la sumatoria de los errores cuadráticos. En comparación con el Error Medio Absoluto o MAE, MSE amplifica y penaliza con mayor fuerza aquellos errores de mayor magnitud. La fórmula de cálculo del MSE es:

,

donde es el resultado en el tiempo t y es el pronóstico de valor en el tiempo t.

Es interesante remarcar que la métrica seleccionada para evaluar los modelos coincide con la función de pérdida que es optimizada al entrenar dichos modelos.

## Tuning de los modelos

En el entrenamiento de un modelo de *Deep Learning* existen diferentes hiperparámetros que deben ser ajustados para lograr que el modelo alcance el mayor desempeño posible. Para esto, se deben probar diferentes valores para cada hiperparámetro, comparando el desempeño obtenido entrenando el modelo con cada uno de ellos. Existen diferentes formas de realizar la comparación de los diferentes valores de los hiperparámetros. Para realizar el *tuning* o “búsqueda de los hiperparámetros óptimos” en los modelos que serán comparados en la tarea de PCSF a partir del *dataset StudentLife* se utilizo la Optimizacion de Bayes [(Agnihotri and Batra 2020)](https://paperpile.com/c/FQGDmo/cUq8).

## Optimización de Bayes

La optimización de Bayes es utilizada para buscar el máximo global en funciones analizadas como “cajas negras”, es decir aquellas que no tienen una forma especial y en las que no es posible utilizar la optimización por el gradiente. Las Redes Neuronales Profundas son un caso claro de este tipo de funciones, en las que por muchos años la mejor forma de encontrar el valor adecuado de los hiperparámetros de un modelo de *Deep Learning* fue el de utilizar una búsqueda aleatoria o simplemente evaluar todos los hiperparámetros posibles. El problema con este enfoque es que si el tiempo que se tarda en entrenar un modelo es elevado es inviable realizar una búsqueda exhaustiva de los mejores hiperparámetros.

La Optimización de Bayes puede solucionar este problema decidiendo de forma inteligente qué valores de los hiperparámetros evaluar en la siguiente iteración, encontrando un punto medio entre los siguientes dos objetivos:

* Obtener información sobre la función con el fin de tener más información a la hora de realizar las siguientes evaluaciones. Esto se logra evaluando en los puntos para los cuales la incertidumbre es mayor (en el caso de usar un Proceso Gausiano como modelo sustituto, donde la varianza sea la mayor).
* Encontrar el máximo de la función basándose en la información ya conocida por el modelo sustituto.

.

Más formalmente, la Optimización de Bayes consiste en hallar correspondiente al maximo o minimo global de una función . La Optimización de Bayes tiene las siguientes restricciones:

* El conjunto factible de es simple.
* es continuo pero no es fácil de optimizar.
* La evaluación de no da información sobre el gradiente.
* es costosa de evaluar.
* es ruidosa.

Teniendo en cuenta estas restricciones, la Optimización de Bayes sigue los siguientes pasos:

1. Se elige un modelo sustituto para modelar la verdadera función y definir sus probabilidades a priori.
2. A partir de las evaluaciones de , se utiliza la regla de Bayes (de ahí el nombre del proceso de optimización) para obtener probabilidades a posteriori.
3. Se utiliza una función de adquisición , que es una función de la probabilidad a posteriori, para decidir cuál será la siguiente evaluación de . se selecciona de tal forma que .
4. Añadir la nueva observación al conjunto de observaciones e ir al paso 2. Repetir hasta que haya convergencia o hasta que se acabe el presupuesto.

En el caso donde la función representa un modelo de *Deep Learning*, cada evaluación se basa en obtener el desempeño del modelo para un conjunto de hiperparámetros determinado, y en base al valor obtenido, actualizar el modelo sustituto para luego seleccionar otro valor para los hiperparámetros basándose en la función de adquisición.

En esta tesis, se utiliza un Proceso Gaussiano [(Görtler et al. 2019)](https://paperpile.com/c/FQGDmo/EKJz) como modelo sustituto. Además, se decidió utilizar la función de adquisición llamada Probabilidad de Mejora. Se utiliza la implementación de la Optimización de Bayes provista por la librería *Scikit-Optimize*[[15]](#footnote-14). Esta librería es un proyecto paralelo al de la librería *Scikit-Learn* especializada en la optimización de modelos secuenciales. A cada proceso de tuning se le dan hasta 100 evaluaciones de diferentes configuraciones de hiperparámetros. Estos procesos pueden finalizar antes de las 100 evaluaciones en caso de que el modelo sustituto converja. Finalmente, el resultado del proceso de *tuning* será la configuración de hiperparámetros que haya alcanzado el mejor desempeño.

## Selección de experimentos modelo

Resultaría ideal que se llevase a cabo el proceso de *tuning* para cada *dataset* diferente, lo cual resulta inviable debido a la alta cantidad de *datasets* que serán evaluados. Por esta razón, se decidió tomar ciertos experimentos como representantes de los demás. Las decisiones que se tomaron son las siguientes:

* Todos los *datasets* utilizados en el proceso de tuning comparten las siguientes características: granularidad de 1 hora, 4 *lags* y períodos de valor 1.
* Se exploran tanto modelos personales como impersonales, debido a que son la categoría que mayor diferencia implica en los *datasets*.
* Se realiza el proceso de ajuste de hiperparámetros para cada tipo de arquitectura propuesta. Esto es: MLP, CNN, TCN y RNN.
* También, se seleccionan dos usuarios representantes. Esta selección es más compleja que las demás por lo que se le dedica su propia subsección.

De esta forma, la cantidad de arquitecturas totales resultantes del proceso de *tuning* será de .

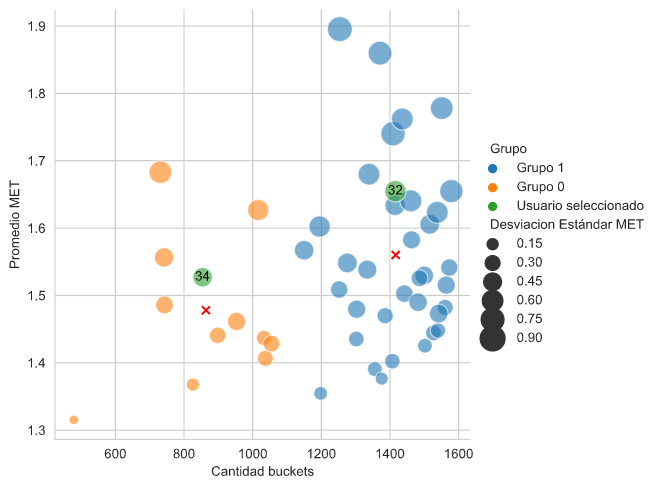
Se decidió darle 100 iteraciones a cada uno de los procesos de tuning. Una vez realizadas las 100 iteraciones se toma el conjunto de hiperparámetros para los cuales el modelo alcanzó el mayor desempeño.

La función que el algoritmo de optimización de bayes intenta minimizar en este caso es el error del modelo para un experimento (ver pipeline del modelo). Es decir, se lleva a cabo el pipeline completo de un experimento normal y se toma la media del desempeño alcanzado en cada una de sus iteraciones.

## Selección de usuarios modelo

Para decidir cuáles serán los usuarios representantes se utiliza el método de *Machine Learning* no supervisado *K-Means*, que se encarga de encontrar centroides a partir de ciertas características computadas de cada usuario. Dichas características son: promedio de MET, desviación estándar de MET, cantidad de *buckets* de 1 hora disponibles. El procedimiento funciona generando dos grupos de usuario a partir de las características de cada uno de ellos. Se toma como el usuario modelo aquel más cercano al centroide calculado por el algoritmo.

En la figura X se muestra un eje cartesiano donde el eje corresponde a la cantidad de buckets de una hora disponibles, mientras que el eje corresponde al promedio de MET. Además se introduce la tercera característica utilizada por el algoritmo *K-Means* representada por el radio de cada punto del eje cartesiano. Cuanto más grande el punto, mayor es la desviación estándar del nivel de MET. Además, se muestran en verde los usuarios seleccionados: el 34, que representa a los usuarios con bajo MET promedio y baja cantidad de buckets disponibles, y el 32, que representa a los usuarios con alta cantidad de buckets y alto MET promedio. Por último, los cruces rojas muestran la ubicación exacta de los centroides determinados por K-Means. Cabe aclarar que solo se muestran las primeras 2 dimensiones de dichos centroides, de manera que parece que los usuarios 32 y 34 no son los más cercanos a ellos, cuando se está obviando la última dimensión.



*Figura X*

La idea detrás de seleccionar usuarios modelo es la de representar patrones de gasto energético diferentes a la hora de encontrar los mejores hiperparámetros para las diferentes arquitecturas y datasets, a la vez que se disminuye enormemente el tiempo requerido por el proceso de *tuning*.

## Diseño de los experimentos

Como se introdujo en el Capítulo 4, en esta tesis se proponen diferentes aspectos a estudiar a la hora de abordar la tarea de PCSF. Estos son: el tipo de red neuronal, las características del *dataset* (número de lags, granularidad y valor del periodo) y la naturaleza los modelos (personal/impersonal). En esta sección, se presentan los experimentos que serán llevados a cabo a partir de la evaluación de diferentes valores para cada uno de los aspectos anteriores, conformando así diferentes categorías en las cuales los experimentos pueden ser clasificados. Para ello se propone una lista de valores para cada aspecto. Cada uno de los valores será puesto a prueba para cada uno de los usuarios. La tabla X muestra todas las categorías en las que pueden ser clasificados los experimentos, así como también se muestran los posibles valores que puede tomar cada una de esas características.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Categoria | **Algoritmo de aprendizaje** | **Naturaleza del modelo** | **Características del dataset** | | | **Usuario** |
| **Cantidad de lags** | **Períodos** | **Granularidad del dataset** |
| **Posibles valores** | MLP, RNN, CNN, TCN | Personal o Impersonal | 1,2,4,8 | 1,2,4 | 30 minutos, 1 hora | Conjunto de usuarios |

Una vez descritos los posibles valores para cada una de las categorías que caracterizan a los experimentos, es posible calcular la cantidad de experimentos a llevar a cabo. Dicha cantidad queda definida por la cardinalidad del producto cartesiano de los conjuntos de posibles valores de cada categoría en la que pueden clasificarse los experimentos. De esta forma, la cantidad de experimento a realizar es:

Por lo tanto, deben llevarse a cabo 9.216 experimentos diferentes. Puede observarse aquí porque se tomaron las medidas de la sección de Selección de los Experimentos Modelos (5.5.2): la cantidad de experimentos a realizar es muy grande como para llevar a cabo el proceso de *tuning* para cada uno de ellos. Además, también puede notarse cómo aumentar la cantidad de valores de alguna categoría aumenta considerablemente la cantidad de experimentos a realizar. Por ejemplo, si se agregase un valor al conjunto granularidad la cantidad de experimentos a realizar aumentaría a 13.824.

## Resultados

Como presentar los resultados:

* Mostrar en cuantas salió primero cada arquitectura
* Mostrar graficos de las predicciones para varias arquitecturas dentro del mismo dataset
* Mostrar predicciones por grupo (centroids)
* Primero que nada, habría que separar los modelos personales de los impersonales porque evaluan cosas diferentes. Los impersonales te permiten ver si la info colectiva se puede usar para predecir el comportamiento general de la población y los otros son personalizados para cada usuario y ahi lo que ves es qué variables influyen en los resutlados para cada caso.
* Ahora, si tu hipótesis es ver si la info del pasado mejora los resultados, tendrías que mostrar, para cada usuario + impersonal, el mejor MSE de cada arquitectura y aclarar cuales son las otras variables que condujeron a ese mejor resultado. Por ejemplo para los impersonales, la mejor performance fue TCN, 2 lags, 2 periodos, buckets de 1 hora, para CNN, x lags, x periodos, x buckets, y así con cada tipo de red. Para el caso de los personales, yo mostraría a modo de resumen acá los usuarios prototipo y después pondría el detalle de cada usuario en un apéndice (o sea, para cada usuario particular, mostrar lo mismo que mostraste para los impersonales). En este caso habría que discutir, si las configuraciones varían para cada usuario, cuál podría ser la causa de esa variación (las particularidades de cada usuario o grupo de usuarios)

En esta sección se procede a presentar los resultados obtenidos, tanto en el proceso de *tuning* como en el de experimentación. Para los resultados de los experimentos, se proponen diferentes maneras de ser presentados.

## Resultados del proceso de tuning

En esta sección se muestran los resultados obtenidos en el proceso de *tuning*. A modo de recapitulación: el proceso de *tuning* consistió en realizar ciertos experimentos seleccionados repetidas veces para poder encontrar las mejores combinaciones de hiperparámetros para cada una de las arquitecturas propuestas. Este proceso fue llevado a cabo a través de la Optimización de Bayes, que utiliza un Proceso Gaussiano para decidir cuál combinación de los hiperparámetros es conveniente ajustar en cada iteración.

Cada una de las arquitecturas propuestas posee diferentes hiperparámetros a ajustar, por lo que se incluye una sección para cada una de ellas. Igualmente, existen ciertos hiperparámetros comunes a todas ellas: número de épocas y el tamaño de bache, ya que no están relacionados a la arquitectura del modelo sino que a la cantidad de veces se itera sobre el *dataset* de entrenamiento y cada cuantos casos de entrenamiento se actualizan los parámetros del modelo. En todos los casos se utilizó *Adam* como función de optimización y *MSE* como función de pérdida.

En total, fueron seleccionados 16 experimentos modelo. En esta sección, se los presenta categorizados por el tipo de red neuronal, mientras que para cada tipo de red neuronal se muestran los resultados de los experimentos modelos asociados. Las otras categorías que varían son la naturaleza del modelo (personal o impersonal) y el usuario modelo (usario 32 o 34).

## MLP

Las arquitecturas MLP son las más simples de todas, por lo que comparten algunos hiperparámetros con otras arquitecturas. Los hiperparámetros que se ajustaron para las MLP fueron: el número de capas densas, el número de unidades por capa, si usar o no *Batch Normalization* y el valor del dropout. En la tabla X se muestran los resultados del proceso de tuning asociados a las MLP.

Personal, 32

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Usuario** | **Personal/Impersonal** | **Nr. nodos** | **Nr. capas** | **Usar Batch Normalization** | **Dropout** | **Nr. épocas** | **Tam. bache** | **MSE** |
| 34 | per | 256 | 16 | 1 | 0.8 | 64 | 8 | 0.264 |
| 32 | per | 4 | 4 | 0 | 0.386 | 64 | 8 | 0.375 |
| 34 | imp | 32 | 2 | 0 | 0.623 | 64 | 64 | 0.213 |
| 32 | imp | 32 | 4 | 1 | 0.346 | 64 | 32 | 0.313 |
| *Tabla X.* | | | | | | | | |

## CNN

En el caso de las arquitecturas CNN los hiperparámetros que se ajustaron fueron: el número de filtros, el tamaño del *kernel* y el *dropout* aplicado a la capa de convolución. Por lo general, esta arquitectura es utilizada en combinación con una o más capas densas. Por lo tanto se decidió agregar una capa densa opcional como la última capa de esta arquitectura, por lo que el número de unidades densas también debió ser tuneado, así como también el valor del *dropout* para esta capa. En la tabla X se muestran los resultados del proceso de tuning asociados a las CNN.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Usuario** | **Personal/Impersonal** | **Nr. filtros** | **Tam. kernel** | **Dropout (conv)** | **Nr. nodos densos** | **Dropout (denso)** | **Nr. épocas** | **Tam. bache** | **MSE** |
| 34 | per | 256 | 4 | 0 | 16 | 0.8 | 64 | 8 | 0.27 |
| 32 | per | 128 | 16 | 0.8 | 32 | 0.8 | 64 | 8 | 0.384 |
| 34 | imp | 128 | 16 | 0.8 | 64 | 0.763 | 64 | 128 | 0.212 |
| 32 | imp | 64 | 4 | 0.8 | 1 | 0.8 | 32 | 256 | 0.308 |

## TCN

Como las TCN son similares a las CNN, comparten ciertos hiperparámetros. En esta arquitectura, los hiperparámetros que se ajustaron fueron: el número de filtros, el tamaño del kernel, el valor del dropout, si usar o no *Batch Normalization* y si usar o no la omisión de conexiones (*skip connections*). En la tabla X se muestran los resultados del proceso de tuning asociados a las TCN.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Usuario** | **Personal/Impersonal** | **Nr. filtros** | **Tam. kernel** | **Dropout** | **Omitir conexiones** | **Usar Batch Normalization** | **Nr. épocas** | **Tam. bache** | **MSE** |
| 34 | per | 32 | 16 | 0.8 | 2 | 1 | 256 | 8 | 0.263 |
| 32 | per | 4 | 4 | 0.246 | 1 | 1 | 128 | 16 | 0.396 |
| 34 | imp | 4 | 4 | 0.462 | 2 | 1 | 32 | 16 | 0.217 |
| 32 | imp | 8 | 16 | 0.33 | 1 | 2 | 64 | 16 | 0.308 |

Esta arquitectura presenta una diferencia importante al ser comparada con las otras. Esto es: es necesario que el campo receptivo sea igual o mayor a la cantidad de *lags* que posea el datasets que se está utilizando, ya que de esta forma es seguro que el modelo esté utilizando efectivamente todos los lags. De esta forma, y como se decidió que el proceso de tuning sea llevado a cabo sólo para algunos casos -más particularmente, para 4 *lags*-*,* el hiperparámetro ‘número de dilaciones’ no fue tuneado por el proceso de tuning. En cambio, este hiperparámetro es ajustado en cada experimento a partir del número de lags y el tamaño de kernels. Como se reporta en [(Bai et al. 2018)](https://paperpile.com/c/FQGDmo/XcA5), las arquitectura no cambian considerablemente su desempeño siempre y cuando su campo receptivo haya sido determinado correctamente. La fórmula que determina el número de dilaciones es:

Al ser utilizada la fórmula anterior, se asegura que la cantidad de dilaciones sea determinada de forma que el campo receptivo siempre abarque la cantidad variable de *lags*.

Cabe aclarar que aunque las TCN permiten la concatenación de más de un bloque residual, en esta tesis se utiliza uno solo. Esta decisión fue tomada teniendo en cuenta que no se justifica la utilización de más de un bloque residual en una experimentación donde la mayor cantidad de *lags* es 8.

## RNN

Por último, en las arquitecturas RNN se ajustaron los siguientes hiperparámetros: número de capas recurrentes, cantidad de unidades LSTM por capa y el valor del *dropout* aplicado en cada capa. Al igual que en las arquitecturas CNN, se permitió que la última capa sea una capa densa opcional, por lo que también deben ser tuneados la cantidad de unidades y el valor del *dropout* de esta última capa. En la tabla X se muestran los resultados del proceso de tuning asociados a las RNN.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Usuario** | **Personal/Impersonal** | **Nr. capas** | **Nr. unidades LSTM** | **Dropout (lstm)** | **Nr nodos densos** | **Dropout (denso)** | **Nr. épocas** | **Tam. bache** | **MSE** |
| 34 | per | 4 | 4 | 0.446 | 64 | 0 | 32 | 16 | 0.246 |
| 32 | per | 2 | 512 | 0 | 64 | 0.8 | 16 | 8 | 0.344 |
| 34 | imp | 2 | 128 | 0.748 | 1 | 0 | 8 | 8 | 0.215 |
| 32 | imp | 2 | 4 | 0 | 32 | 0.214 | 8 | 8 | 0.302 |

## Resultados de los experimentos

Como ya ha sido mostrado en la sección de experimentos, son muchas las categorías por las cuales puede ser clasificado cada experimento. Dichas categorías son: la arquitectura utilizada, el usuario objetivo, aquellas relacionadas a la conformación del dataset (número de lags, periodo y granularidad) y por último, la naturaleza del modelo -es decir, personal o impersonal-. Además, se suman otras categorías a partir de las cuales los experimentos pueden ser comparados: como el MSE obtenido en cada una de las iteraciones del algoritmo de evaluación, el MSE promedio, el tiempo requerido para el entrenamiento para cada una de las interacciones y el número de parámetros del modelo como resultado del proceso de tuning. A continuación, se discuten las formas existentes de mostrar y comparar los resultados.

Una primera forma de mostrar los resultados de los experimentos es realizar rankings en los que se muestran los experimentos con mayor desempeño para cada una de las categorías o conjunto de ellas. Esto es, tomar los posibles valores de una categoría o conjunto de ellas y compararlos entre sí. Por ejemplo, dadas las categorias arquitectura y granularidad, realizar un ranking para comparar cuales obtuvieron obtuvieron el mayor desempeño. En este último caso: los experimentos a comparar son: TCN/1h, MLP/1h , CNN/1h, RNN/1h, TCN/30min , MLP/30min, CNN/30min , RNN/30min. Cada una de estas combinaciones de categorías representa a un conjunto de experimentos. Por ejemplo, en el caso anterior cada una de las combinaciones representa a 1152 experimentos. Se obtiene un solo experimento en el único caso en que se determine el valor de cada una de las categorías. Es debido entonces discutir cómo comparar estas categorías si representan a más de un experimento. Una opción es aplicar alguna función de agregación para obtener un valor que represente a dicho conjunto de experimentos. Diferentes alternativas son posibles y se listan a continuación:

* Tomar del conjunto de experimentos el que mayor MSE obtuvo.
* Tomar del conjunto de experimentos el promedio del MSE de todos ellos.

Sin embargo, esta forma de comparación supone un problema: los diferentes usuarios pueden llegar a obtener valores muy diferentes entre sí debido a la idiosincrasia de cada uno de ellos. Esto puede afectar en cuan predecibles sean sus patrones a partir de los modelos entrenados. Por esta razón, es razonable estudiar a cada usuario en particular. Por lo tanto, se propone como una segunda forma de mostrar los resultados, aplicar la primera estrategia para cada uno de los usuarios. Por ejemplo, una tabla para cada uno de ellos donde se muestre cuál arquitectura alcanzó el mejor desempeño o una comparación por cualquier otra categoría o conjunto de ellas. La desventaja de este enfoque es que supondría tener diversas tablas para cada uno de los 48 usuarios.

Por último, se propone una tercera forma de mostrar los resultados que busca obtener las ventajas de las dos primeras. Esta estrategia consiste en realizar un ranking para cada uno de los usuarios, pero resumirlos en una única tabla. La generación de esta tabla puede llevarse a cabo siguiendo los siguientes pasos:

1. Seleccionar la/s categoría/s a que se quieren comparar;
2. Obtener todos los posibles valores resultantes de la combinación de estas categorías;
3. Para cada usuario, aplicar una función de agregación a los experimentos que agrupa cada posible valor y realizar un ranking a partir de ellos.
4. A partir de todos los rankings obtenidos, generar un ranking global. Esto es, para cada posible valor, calcular en cuantos rankings quedó primero, en cuantos quedó segundo, etc.

A pesar de que este proceso se mostró solo para realizar cada ranking individual sobre la categoría ‘usuario’, que opino que es el más útil, podría ser llevado a cabo para cualquier otra categoría, siempre y cuando no sea una categoría seleccionada para hacer la comparación.

Otra variable interesante a introducir es que los experimentos incluidos en el proceso anterior pueden ser personalizados. Es decir, la entrada al proceso de generación del ranking global puede no ser el conjunto total de experimentos, sino que un subconjunto de ellos. De esta forma, pueden realizarse análisis con mayor granularidad. Es decir, puede seleccionarse un valor específico para algunas de las categorías. Lo que motiva a tomar un subconjunto del total de los experimentos es que hay aspectos tanto cualitativos como cuantitativos que pueden ser tomados en cuenta al momento de comparar los experimentos y que no son el MSE obtenido. Estos aspectos pueden ser (pero no se limitan a): el tamaño del dataset, la cantidad de registros disponibles para cada bucket, el hecho de poder predecir el comportamiento sedentario futuro con una mayor o menor granularidad, el tiempo que llevó entrenar al modelo, su número de parámetros (a partir del cual puede aproximarse el tamaño ocupado por él). Por la existencia de estos aspectos, se justifica que puede ser útil estudiar diferentes particiones del conjunto de experimentos por separado.

A modo de resumen, la estrategia para mostrar los resultados se compone de los siguientes aspectos:

1. La categoría del modelo que será comparada (por ejemplo, la arquitectura).
2. La categoría de la cual serán tomados los valores posibles con los cuales se realizará cada ranking individual (por ejemplo, el usuario) .
3. La categoría del modelo que será utilizada para realizar cada ranking individual (por ejemplo, el MSE obtenido).
4. Valores específicos para ciertas categorías de los modelos que funcionaran como un filtro sobre el total del conjunto de experimentos (por ejemplo, una determinada cantidad de lags).

A continuación se muestran diferentes configuración de la estrategia anteriormente detallada para mostrar los rankings. Con respecto al aspecto número 2, se decidió darle prioridad al MSE en el análisis ya que es el que mayor importancia tiene en cuanto al desempeño de los modelos, aunque la cantidad de parámetros del modelo y el tiempo requerido para el entrenamiento también serán abordados. Finalmente, en la ultima seccion se muestran los resultados especificos para los usuarios modelos.

### Comparación entre las redes neuronales a partir del MSE

Para comenzar con los resultados, en la Tabla X se muestra el ranking de los experimentos por el tipo de red neuronal utilizada comparados por el MSE utilizando los usuarios para los rankings individuales sobre el conjunto total de los experimentos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Puesto 1** | **Puesto 2** | **Puesto 3** | **Puesto 4** |
| **rnn** | 6 | 5 | 20 | 17 |
| **cnn** | 11 | 9 | 9 | 19 |
| **tcn** | 21 | 21 | 3 | 3 |
| **mlp** | 10 | 13 | 16 | 9 |

En la tabla anterior, cada celda muestra la cantidad de veces en las cuales un determinado tipo de red neuronal terminó en un determinado puesto. La suma de cada fila o columna es 48, el número total de usuarios, ya que sobre ellos se llevó a cabo cada ranking individual.

Como puede observarse, los modelos TCN obtuvieron el mejor desempeño para 21 de los 48 usuarios, mientras que las CNN y las MLP se desempeñaron mejor en 11 y 10 casos respectivamente. Por último, las RNN se desempeñaron mejor que las demás redes en solo 6 casos. Es interesante notar que las TCN obtuvieron el primer o segundo mejor desempeño para 42 de los 48 usuarios.

En las siguientes dos tablas se muestra el mismo ranking que el de la tabla anterior con la diferencia de que se filtran por las diferentes granularidades posibles. Como puede observarse, los resultados no cambian en gran medida entre la granularidad de 1 hora y las de 30 minutos. Específicamente, la ventaja de las TCN se sigue manteniendo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Puesto 1** | **Puesto 2** | **Puesto 3** | **Puesto 4** |
| **rnn** | 6 | 6 | 18 | 18 |
| **cnn** | 11 | 8 | 11 | 18 |
| **tcn** | 22 | 20 | 3 | 3 |
| **mlp** | 9 | 14 | 16 | 9 |

*Tabla x. Granularidad de 1 hora*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Puesto 1** | **Puesto 2** | **Puesto 3** | **Puesto 4** |
| **rnn** | 5 | 5 | 19 | 19 |
| **cnn** | 10 | 12 | 11 | 15 |
| **tcn** | 22 | 15 | 6 | 5 |
| **mlp** | 11 | 16 | 12 | 9 |

*Tabla x. Granularidad de 30 minutos*

A continuación, se realizan otros rankings a partir de los diferentes tipos de modelos según su naturaleza, esto es, modelos personales e impersonales. En la primera tabla (Tabla X) se muestra el ranking de los modelos personales.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Puesto 1 | Puesto 2 | Puesto 3 | Puesto 4 |
| rnn | 21 | 15 | 7 | 5 |
| cnn | 5 | 15 | 15 | 13 |
| tcn | 17 | 7 | 10 | 14 |
| mlp | 5 | 11 | 16 | 16 |

*Tabla x. Modelos personales*

En la segunda tabla, podemos observar el ranking para los modelos impersonales...

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Puesto 1** | **Puesto 2** | **Puesto 3** | **Puesto 4** |
| **rnn** | 3 | 6 | 18 | 21 |
| **cnn** | 11 | 8 | 12 | 17 |
| **tcn** | 23 | 20 | 2 | 3 |
| **mlp** | 11 | 14 | 16 | 7 |

*Tabla x. Modelos impersonales*

### Comparación entre las redes neuronales a partir del tiempo requerido para ser entrenadas

Ranking por redes neuronales comparados por el tiempo requerido para entrenar

Primero, modelos personales

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Puesto 1** | **Puesto 2** | **Puesto 3** | **Puesto 4** |
| **rnn** | 48 | 0 | 0 | 0 |
| **cnn** | 0 | 12 | 36 | 0 |
| **tcn** | 0 | 0 | 0 | 48 |
| **mlp** | 0 | 36 | 12 | 0 |

Modelos impersonales

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Puesto 1** | **Puesto 2** | **Puesto 3** | **Puesto 4** |
| **rnn** | 0 | 0 | 30 | 18 |
| **cnn** | 48 | 0 | 0 | 0 |
| **tcn** | 0 | 0 | 18 | 30 |
| **mlp** | 0 | 48 | 0 | 0 |

### Comparación entre las redes neuronales a partir del número de parámetros de los modelos

Comparacion sobre numero de parametros

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Puesto 1** | **Puesto 2** | **Puesto 3** | **Puesto 4** |
| **rnn** | 12 | 0 | 36 | 0 |
| **cnn** | 0 | 0 | 0 | 48 |
| **tcn** | 0 | 36 | 12 | 0 |
| **mlp** | 36 | 12 | 0 | 0 |

### Resultados detallados de los usuarios modelo

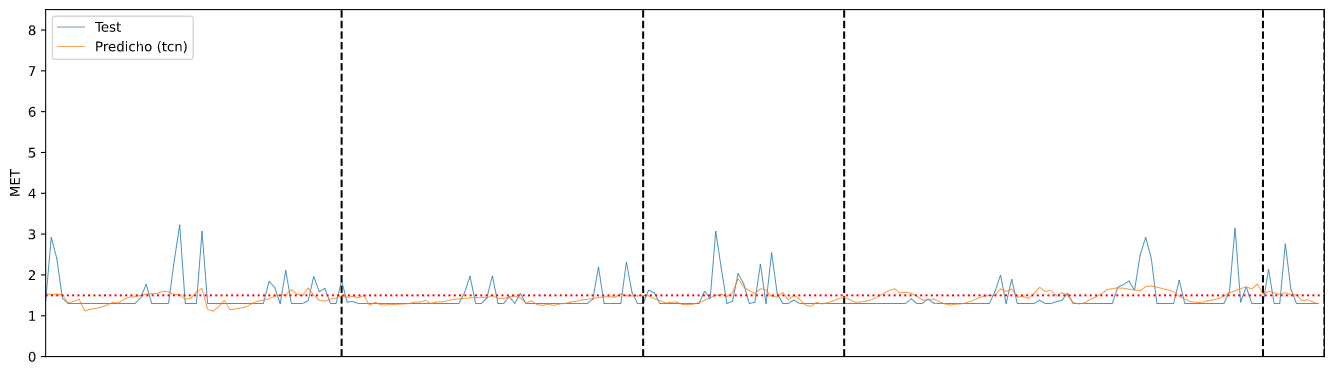
En esta sección se muestran más detalladamente los resultados obtenidos para los usuarios modelos. El usuario 34 representa al grupo de usuarios con menor gasto energético y menor cantidad de *buckets* disponibles, mientras que el usuario 32 representa al grupo de usuarios con mayor gasto energético y mayot cantidad de *buckets* disponibles.

A continuación, se muestran las configuraciones de los experimentos que obtuvieron el menor MSE para cada uno de los usuarios modelo así como también el MSE y el tiempo requerido para el entrenamiento para cada una de las iteraciones. Además, se muestran graficadas las predicciones realizadas.

En el caso del usuario 34, el modelo que menor MSE obtuvo fue de naturaleza impersonal, con una granularidad de 1 hora, usando 8 lags, y un periodo de 4. La arquitectura correspondió a una TCN. El promedio del MSE obtenido fue de 0.1328 mientras que el tiempo de entrenamiento promedio fue de de aproximadamente 7 minutos. En la Tabla X se muestran los valores de MSE y tiempo de entrenamiento de todas las iteraciones.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **It. 1** | **It. 2** | **It. 3** | **It. 4** | **It. 5** |
| **MSE** | 0.195 | 0.044 | 0.14 | 0.107 | 0.178 |
| **Tiempo** | 0.713 | 1.393 | 2.029 | 2.614 | 2.844 |

En la Figura X pueden observarse las predicciones realizadas por el modelo que menor desempeño obtuvo para el usuario 34. Las diferentes iteraciones están separadas por líneas horizontales intercaladas. Puede observarse la poca cantidad de *buckets* disponibles y la poca cantidad de registros con comportamiento no sedentario disponible. Debido a esto, el modelo es incapaz de identificar correctamente donde el usuario tuvo un alto nivel de MET. En estos casos donde la gran mayoría de los *buckets* corresponden a un comportamiento sedentario se ve la importancia de la utilización de MET como función de error, ya que los *buckets* de comportamiento no sedentario deben ser fuertemente penalizados.



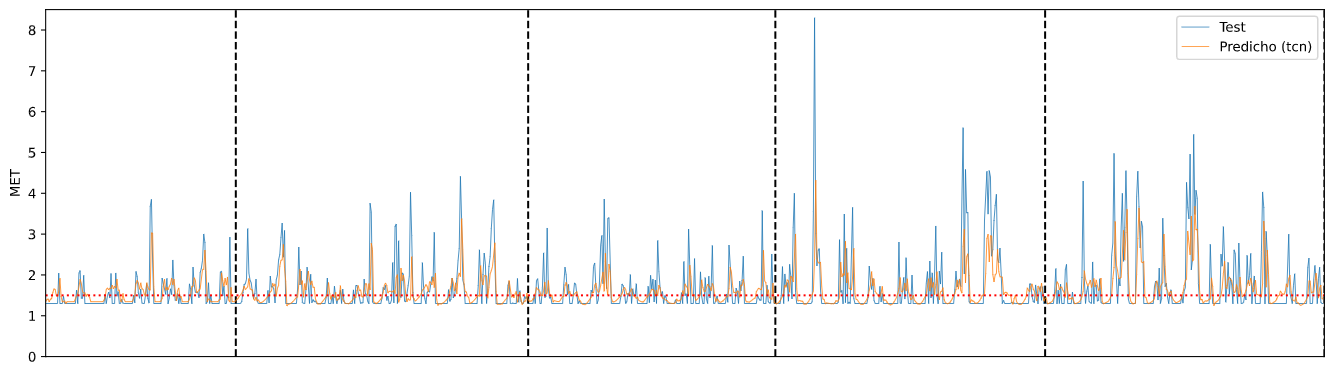
*Tabla X*

En el caso del usuario 32, el modelo que menor MSE obtuvo fue de naturaleza impersonal, con una granularidad de 1 hora, usando 8 lags, y un periodo de 1. La arquitectura correspondió a una TCN. El promedio del MSE obtenido fue de 0.3 mientras que el tiempo de entrenamiento promedio fue de aproximadamente 5 minutos. En la Tabla X se muestran los valores de MSE y tiempo de entrenamiento de todas las iteraciones.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **It. 1** | **It. 2** | **It. 3** | **It. 4** | **It. 5** |
| **MSE** | 0.11 | 0.221 | 0.186 | 0.569 | 0.414 |
| **Tiempo** | 0.938 | 0.973 | 2.934 | 2.391 | 3.892 |

*Tabla X*

En la Figura Y pueden observarse las predicciones realizadas por el modelo que menor desempeño obtuvo para el usuario 32. En comparación con la Figura X, en este caso pueden observarse muchos más *buckets* disponibles de comportamiento no sedentario, por lo que el modelo puede predecir mucho mejor cuando habrá un pico de alto gasto energético. A pesar de esto, el MSE obtenido es mayor al obtenido para el usuario 34, ya que para este último el modelo aprendió a predecir siempre un bajo nivel de MET y no ser penalizado haciéndolo.



able 1. Valores de MAE conseguidos para cada arquitectura propuesta

## Discusión

Ideas:

* Mostrar el desempeño de acuerdo al promedio de met de los usuarios
* Averiguar si impacto el tunner los modelos de acuerdo a solo dos usuarios. Ej: los usuarios mas alejados del cluster obtuvieron un desempeño menor?
* Anomalias: algunos experimentos tienen un mse altisimo en la primer iteracion
* Rnn con numero variable de lags
* Como va variando el tiempo y MSE con respecto a las iteraciones

El usuario que presentó mayor desafío para los predictores resultó ser el número 50, ya que fue el que contaba con más casos de comportamiento sedentario. Por lo tanto, resultó difícil que las arquitecturas propuestas aprendieran, a partir de las características del pasado, cuándo iba a tener un comportamiento no sedentario en el futuro. El modelo correspondiente a la arquitectura 3, con 2 lags, fue el que mejor desempeño presentó, con un MAE de 0.086. La Fig. 3 muestra las predicciones de prueba para esta arquitectura.

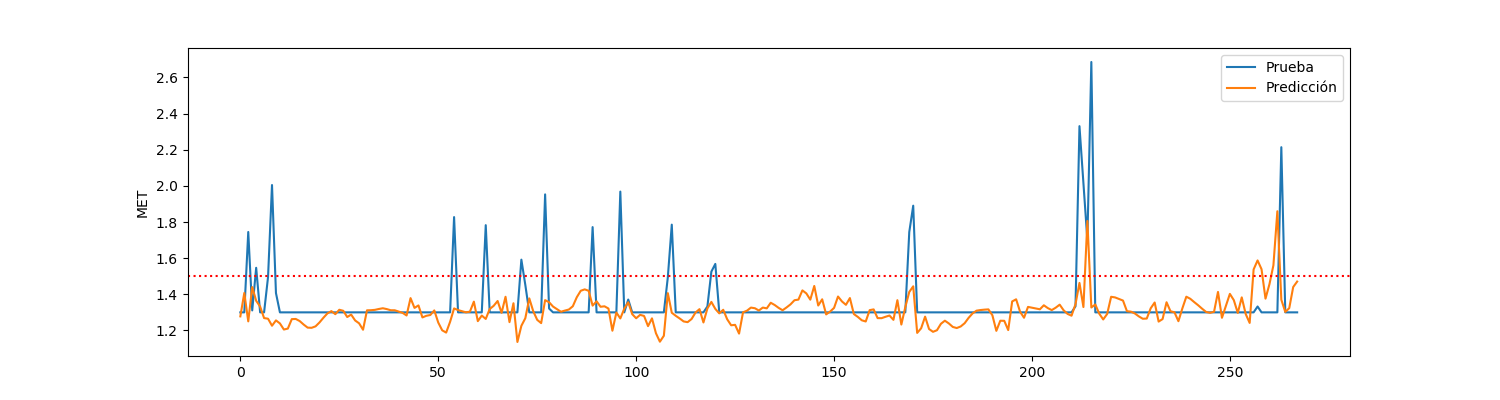


Fig. 1. Predicciones con datos de prueba del usuario 50 con la arquitectura 3 y 2 lags

En el caso del usuario 31, se puede observar un comportamiento más rutinario. Teniendo en cuenta que en el pre-procesamiento se le dio especial atención a las características de tiempo, es entendible que los regresores se desempeñen mejor en estos casos. La arquitectura 3, con 2 lags, fue la que mejor performance tuvo en este caso, con un MAE de prueba de 0.282. La Fig. 4 muestra las predicciones de prueba para esta arquitectura.

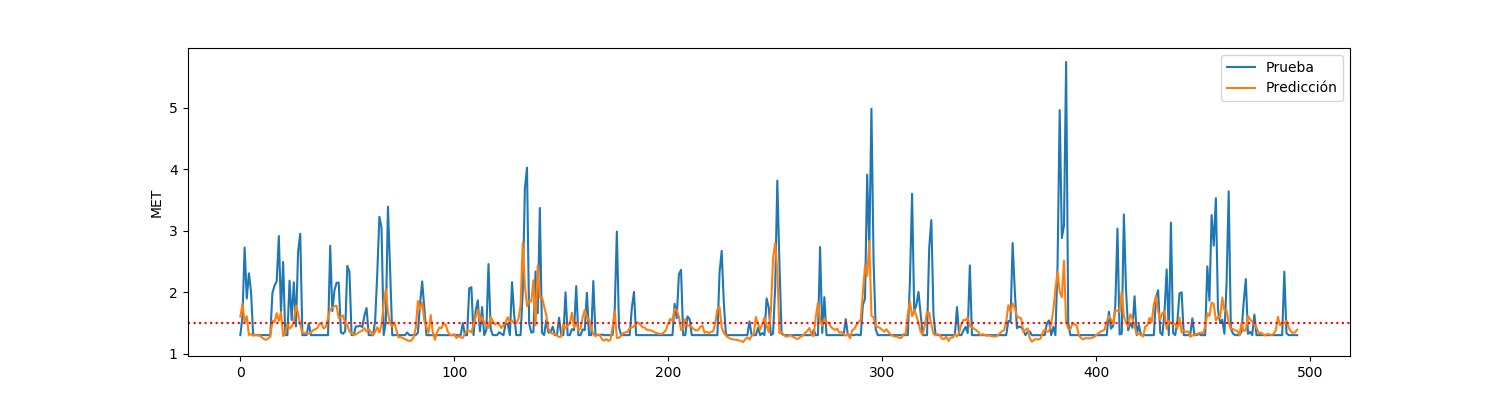


Fig. 1. Predicciones con datos de prueba del usuario 31 con la arquitectura 3

Por último, los modelos sobre el usuario 4 fueron los que mejor desempeño tuvieron, a pesar de tener un mayor MAE que los modelos sobre los otros usuarios. Esto se debe a que este usuario posee muchos más casos de comportamiento no sedentario, por lo que para que el regresor se desempeñe bien debe aprender los patrones de comportamiento del usuario a partir de las características provistas. En el caso del usuario 50, el predictor podría solo aprender a predecir un valor cercano a 1.3 METs para obtener un bajo MAE. La arquitectura que mejor se desempeñó en este caso fue, nuevamente, la número 3, con 2 lags, con un MAE de 0.479. La Fig. 5 muestra las predicciones de entrenamiento y testeo para esta arquitectura.

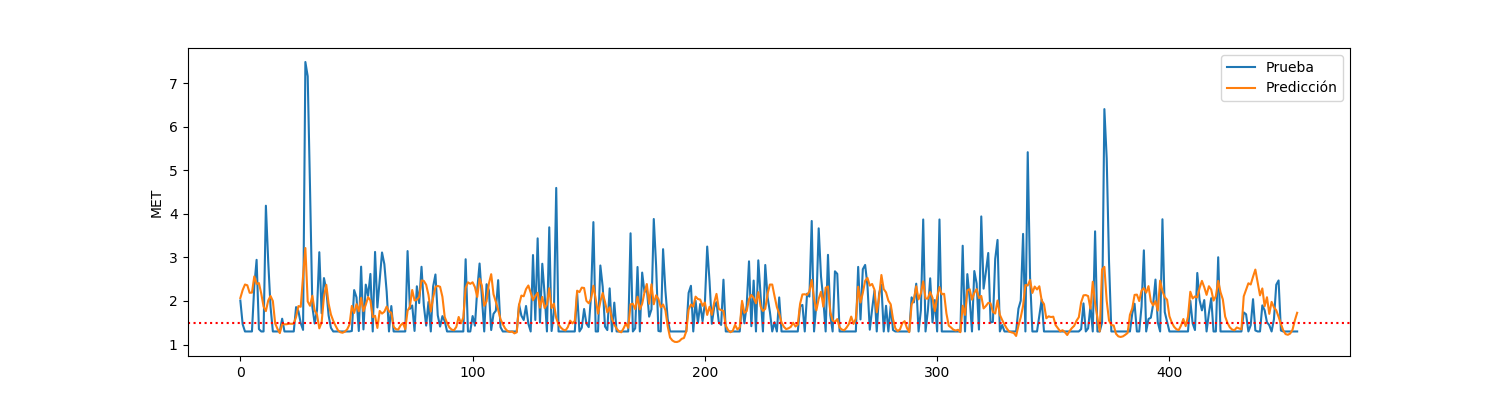


Fig. 1. Predicciones con datos de prueba del usuario 31 con la arquitectura 4

El hecho de que, para todos los modelos, las CNN tuvieron una mejor performance que las RNN, confirma nuestra hipótesis de que las CNN son más adecuadas para problemas de predicción que deban aprender de una secuencia de datos.

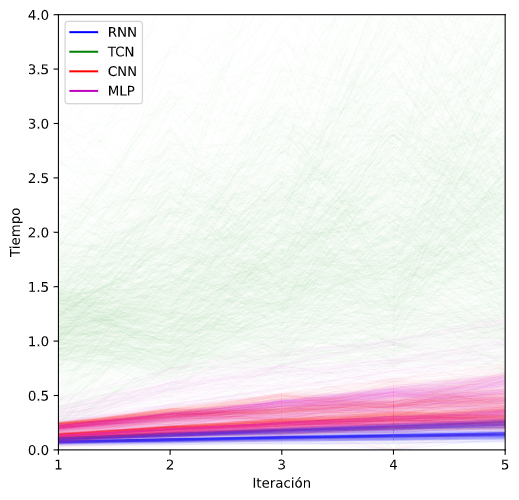
## Discusion resultados

## Tiempo de entrenamiento a lo largo de las iteraciones

En las discusiones anteriores se tomó al MSE como el aspecto más importante, ya que un menor MSE permite predecir con mayor precisión el comportamiento sedentario futuro. Aunque es cierto, es importante analizar el tiempo que lleva entrenar los modelos, ya que es un aspecto importante a tener en cuenta. El tiempo en que el modelo es entrenado es también el tiempo que el usuario deberá esperar para poder obtener predicciones. Las diferentes características de los modelos tienen implicancias en cuanto a cuánto lleva a entrenar el modelo, así como también las tiene el ambiente de cómputo donde sea dicho modelo entrenado.

Tras observar los resultados mostrados en las Tablas X e Y puede verse que existe una gran diferencia entre el tiempo requerido para entrenar los modelos dependiendo de la naturaleza del modelo (personal o impersonal). Esto era esperable, ya que los modelos impersonales poseen muchos más datos de entrenamiento que los modelos personales. En el caso de los modelos personales la arquitectura RNN fue la que menos tiempo requiere para ser entrenada para cada uno de los usuarios, mientras que en el caso de los modelos impersonales la arquitectura CNN fue la que menos tiempo requiere para ser entrenada para cada uno de los usuarios.

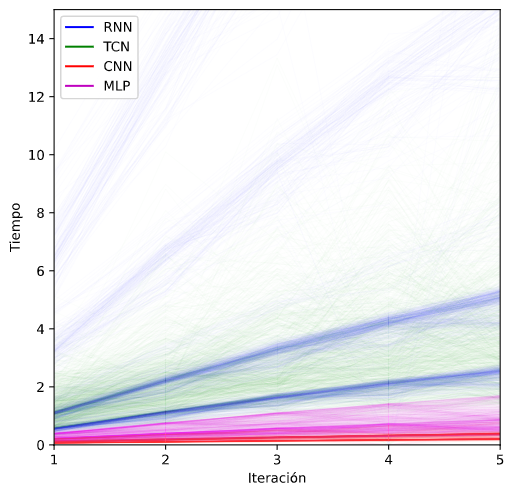
En las Figuras X e Y pueden observarse graficados el tiempo requerido para entrenar el modelo en cada uno de los experimentos. Para cada experimento, se grafican los 5 puntos correspondientes a cada una de las iteraciones y cada par de puntos consecutivos se une por una línea. Los experimentos de cada arquitectura poseen un color especial para que pueda observarse a grandes rasgos si existen patrones.



*Figura X. Modelos personales*

Con respecto a los modelos personales puede observarse que todas las arquitecturas menos la TCN se llevaron a cabo en general, en menos de medio minuto, mientras que el tiempo requerido para entrenar las TCN varía mucho ya que puede observarse una bruma en la parte superior de la imagen. Esto puede deberse en parte a que las TCN fueron la única arquitectura cuyo conjunto de hiperparametros no fue completamente determinado al finalizar el proceso de *tunning,* ya que debió calcularse la cantidad de dilaciones para cada experimento en particular.

En el caso de los modelos impersonales, los modelos CNN y las MLP fueron entrenadas en una cantidad substancial de tiempo menor que las otras dos arquitecturas. Para los modelos TCN, puede observarse un fenómeno similar que en la figura anterior, donde no se identifica un claro patrón, sino que todas las líneas verdes parecen formar una nube, significando así que existe una gran diferencia en el tiempo requerido para entrenar este tipo de modelos. Por último, los modelos RNN pasan a ser, junto a los TCN, los más lentos para ser entrenados en el caso en que la naturaleza del modelo sea impersonal. En el caso de los RNN se debe a su baja capacidad de paralelización, en comparación con otros modelos como los CNN que son altamente paralelizables. Otro fenómeno que puede observarse en el caso de los modelos RNN es la existencia de 4 líneas diferentes claramente distinguibles. Esto se debe, posiblemente, a las 4 configuraciones de hiperparametros que fueron determinadas para los modelos RNN al finalizar el proceso de *tunning.* Este fenómeno seguramente se de en el caso de los CNN y MLP pero al ser tan bajo el tiempo de entrenamiento no llega a ser visible*.*

**

*Figura X. Modelos impersonales*

## MSE a lo largo de las iteraciones

## 

## Conclusiones

En este trabajo nos propusimos comparar diferentes tipos de redes neuronales para evaluar su performance en el problema de la predicción del comportamiento sedentario. Este trabajo es el primero en evaluar el rendimiento de redes neuronales que intentan sacar provecho al carácter secuencial del problema tratado. Se presentaron usuarios con diferentes niveles de gasto energético, lo que hace que sea muy difícil diseñar una arquitectura que tenga una buena performance para todos ellos.

A pesar de haber utilizado un alto grado de regularización, las NN más extensas son propensas a producir overfitting. En consecuencia, tendrán un mal rendimiento en la etapa de testing. Este es un trade-off que fue considerado fue el nivel de la complejidad de las redes. Al momento de diseñar cada red, se buscó que tuvieran, cada una, una performance aceptable para cada uno de los usuarios seleccionados. Por lo tanto, es importante que la complejidad de cada red sea la menor posible para que posean la capacidad necesaria de generalización.

Por otra parte, como solo se toman los datos de un usuario en particular para entrenar las redes, la cantidad de datos es muy baja y no permite llegar a entrenar apropiadamente a redes más complejas y con mayor cantidad de parámetros. Este fue un trade-off al momento de diseñar las arquitecturas propuestas y podría llegar a ser resuelto aumentando la granularidad en el pre-procesamiento y tomar, por ejemplo, buckets de tiempo de 30 minutos o 15 minutos, resultando así en más casos de entrenamiento.

Otro de los trade-off que tuvimos que enfrentar fue la cantidad de time-lags a tomar. Es lógico pensar que cuanta más información del pasado se le dé a la arquitectura, más información tendrá y así podrá hacer mejores predicciones. Esto no es cierto por dos razones. La primera de ellas es que el MET que tendrá un usuario en el tiempo t+1 puede no estar relacionado con las features que poseemos del mismo usuario en el tiempo t-n, siento n un entero arbitrario. Esto, al tener pocos casos de entrenamiento, puede generar ruido en el modelo. La segunda razón es que, como en muchos casos no se poseen datos sobre todas las horas que duró el experimento, cuantos más time-lags se tengan, mayor es la probabilidad de que un caso de entrenamiento sea descartado. Por ejemplo, si la cantidad de time-lags es n y si se tienen los datos del usuario x desde el tiempo t al t-n+1 pero no del tiempo t-n, el caso de entrenamiento se desecha.

Un punto importante a destacar es que, como los estudiantes mantuvieron, por lo general, un comportamiento totalmente sedentario, los clasificadores que intentaban siempre predecir valores cercanos a cero MET fueron descartados, ya que conseguían un MSE bajo pero no eran útiles. Este problema fue tratado en el trabajo realizado por M. Santillán Cooper y M. Armentano con las técnicas de over-sampling y asignación de peso a las clases.

Con respecto a la hipótesis planteada, todas ellas pudieron ser evaluadas y comprobadas. Se lograron diseñar arquitecturas con un mejor desempeño que aquellas arquitecturas que no utilizan información del pasado. Además, se comprobó que las CNN pueden llegar a desempeñarse de una mejor manera que las RNN.

## Bibliografía

[Activity Prediction. (2015). In *Activity Learning* (pp. 127–147).](http://paperpile.com/b/FQGDmo/khD0k)

[Ainsworth, B. E., Haskell, W. L., Herrmann, S. D., Meckes, N., Bassett, D. R., Jr, Tudor-Locke, C., Greer, J. L., Vezina, J., Whitt-Glover, M. C., & Leon, A. S. (2011). 2011 Compendium of Physical Activities: a second update of codes and MET values. *Medicine and Science in Sports and Exercise*, *43*(8), 1575–1581.](http://paperpile.com/b/FQGDmo/zOAD7)

[*Alert me: Enhancing active lifestyle via observing sedentary behavior using mobile sensing systems*. (n.d.). Retrieved February 2, 2021, from](http://paperpile.com/b/FQGDmo/6ewn) <https://ieeexplore.ieee.org/document/8210838>

[Ali, S., Khusro, S., Hassan, L., & Khan, A. A. (2014). A Survey of Mobile Phones Context-Awareness Using Sensing Computing Research. *Journal of Engineering and Applied Sciences*, *33*(2), 75–93.](http://paperpile.com/b/FQGDmo/TYzA)

[Atkin, A. J., Gorely, T., Clemes, S. A., Yates, T., Edwardson, C., Brage, S., Salmon, J., Marshall, S. J., & Biddle, S. J. H. (2012). Methods of Measurement in epidemiology: sedentary Behaviour. *International Journal of Epidemiology*, *41*(5), 1460–1471.](http://paperpile.com/b/FQGDmo/45GF)

[Bai, S., Zico Kolter, J., & Koltun, V. (2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*.](http://paperpile.com/b/FQGDmo/XcA5) <http://dx.doi.org/>

[Benatti, F. B., & Ried-Larsen, M. (2015). The Effects of Breaking up Prolonged Sitting Time. In *Medicine & Science in Sports & Exercise* (Vol. 47, Issue 10, pp. 2053–2061). https://doi.org/](http://paperpile.com/b/FQGDmo/Nwq4)[10.1249/mss.0000000000000654](http://dx.doi.org/10.1249/mss.0000000000000654)

[Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks / a Publication of the IEEE Neural Networks Council*, *5*(2), 157–166.](http://paperpile.com/b/FQGDmo/NwkN)

[Berry, M. W., Mohamed, A., & Yap, B. W. (2019). *Supervised and Unsupervised Learning for Data Science*. Springer Nature.](http://paperpile.com/b/FQGDmo/GuPX)

[Carter, S., Hartman, Y., Holder, S., Thijssen, D. H., & Hopkins, N. D. (2017). Sedentary Behavior and Cardiovascular Disease Risk: Mediating Mechanisms. *Exercise and Sport Sciences Reviews*, *45*(2), 80–86.](http://paperpile.com/b/FQGDmo/lXon)

[Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 103–111.](http://paperpile.com/b/FQGDmo/aBhN)

[Chollet, F. (2017). *Deep Learning with Python*. Manning Publications Company.](http://paperpile.com/b/FQGDmo/KYgf)

[Cook, D. J., & Krishnan, N. C. (2015). *Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data*. John Wiley & Sons.](http://paperpile.com/b/FQGDmo/DZqSa)

[Dauphin, Y. N., Fan, A., Auli, M., & Grangier, D. (2017). Language Modeling with Gated Convolutional Networks. *International Conference on Machine Learning*, 933–941.](http://paperpile.com/b/FQGDmo/RfyA)

[Dutta, M. J., Kaur-Gill, S., Tan, N., & Lam, C. (2018). mHealth, Health, and Mobility: A Culture-Centered Interrogation. In *mHealth Innovation in Asia* (pp. 91–107). Springer, Dordrecht.](http://paperpile.com/b/FQGDmo/QgUc)

[Felez-Nobrega, M., Hillman, C. H., Dowd, K. P., Cirera, E., & Puig-Ribera, A. (2018). ActivPALTM determined sedentary behaviour, physical activity and academic achievement in college students. In *Journal of Sports Sciences* (Vol. 36, Issue 20, pp. 2311–2316). https://doi.org/](http://paperpile.com/b/FQGDmo/fa6g)[10.1080/02640414.2018.1451212](http://dx.doi.org/10.1080/02640414.2018.1451212)

[Gong, J., Huang, Y., Chow, P. I., Fua, K., Gerber, M. S., Teachman, B. A., & Barnes, L. E. (2019). Understanding behavioral dynamics of social anxiety among college students through smartphone sensors. In *Information Fusion* (Vol. 49, pp. 57–68). https://doi.org/](http://paperpile.com/b/FQGDmo/TxFr)[10.1016/j.inffus.2018.09.002](http://dx.doi.org/10.1016/j.inffus.2018.09.002)

[Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.](http://paperpile.com/b/FQGDmo/ZFoL)

[Grundgeiger, T., Pichen, J., Häfner, J., & Huber, S. (2017). Combating Sedentary Behavior: An App Based on a Distributed Prospective Memory Approach. *The 2017 CHI Conference Extended Abstracts*, 1632–1639.](http://paperpile.com/b/FQGDmo/V35l)

[Harari, G. M., Gosling, S. D., Wang, R., Chen, F., Chen, Z., & Campbell, A. T. (2017). Patterns of behavior change in students over an academic term: A preliminary study of activity and sociability behaviors using smartphone sensing methods. In *Computers in Human Behavior* (Vol. 67, pp. 129–138). https://doi.org/](http://paperpile.com/b/FQGDmo/VGeQ)[10.1016/j.chb.2016.10.027](http://dx.doi.org/10.1016/j.chb.2016.10.027)

[He, Q., & Agu, E. (2014). *On11: An activity recommendation application to mitigate sedentary lifestyle*. https://doi.org/](http://paperpile.com/b/FQGDmo/F6qP)[10.1145/2611264.2611268](http://dx.doi.org/10.1145/2611264.2611268)

[He, Q., & Agu, E. (2017). A Rhythm Analysis-Based Model to Predict Sedentary Behaviors. *2017 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, 383–391.](http://paperpile.com/b/FQGDmo/oye2)

[He, Q., & Agu, E. O. (2016a). A frequency domain algorithm to identify recurrent sedentary behaviors from activity time-series data. *2016 IEEE-EMBS International Conference on Biomedical and Health Informatics (BHI)*. https://doi.org/](http://paperpile.com/b/FQGDmo/axGRS)[10.1109/bhi.2016.7455831](http://dx.doi.org/10.1109/bhi.2016.7455831)

[He, Q., & Agu, E. O. (2016b). Smartphone usage contexts and sensable patterns as predictors of future sedentary behaviors. *2016 IEEE Healthcare Innovation Point-Of-Care Technologies Conference (HI-POCT)*. https://doi.org/](http://paperpile.com/b/FQGDmo/GSfPw)[10.1109/hic.2016.7797695](http://dx.doi.org/10.1109/hic.2016.7797695)

[He, Q., & Agu, E. O. (2016c). Towards sedentary lifestyle prevention: An autoregressive model for predicting sedentary behaviors. *2016 10th International Symposium on Medical Information and Communication Technology (ISMICT)*. https://doi.org/](http://paperpile.com/b/FQGDmo/2NLm5)[10.1109/ismict.2016.7498879](http://dx.doi.org/10.1109/ismict.2016.7498879)

[Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.](http://paperpile.com/b/FQGDmo/XGNz)

[Kalchbrenner, N., Espeholt, L., Simonyan, K., van den Oord, A., Graves, A., & Kavukcuoglu, K. (2016). *Neural Machine Translation in Linear Time*.](http://paperpile.com/b/FQGDmo/pq2Q) <https://research.google/pubs/pub48560/>

[Kanjo, E., Younis, E. M. G., & Ang, C. S. (2019). Deep learning analysis of mobile physiological, environmental and location sensor data for emotion detection. In *Information Fusion* (Vol. 49, pp. 46–56). https://doi.org/](http://paperpile.com/b/FQGDmo/zeAi)[10.1016/j.inffus.2018.09.001](http://dx.doi.org/10.1016/j.inffus.2018.09.001)

[Koster, A., Caserotti, P., Patel, K. V., Matthews, C. E., Berrigan, D., Van Domelen, D. R., Brychta, R. J., Chen, K. Y., & Harris, T. B. (2012). Association of sedentary time with mortality independent of moderate to vigorous physical activity. *PloS One*, *7*(6), e37696.](http://paperpile.com/b/FQGDmo/8ipe)

[Lea, C., Flynn, M. D., Vidal, R., Reiter, A., & Hager, G. D. (2017). Temporal Convolutional Networks for Action Segmentation and Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 156–165.](http://paperpile.com/b/FQGDmo/VXWI)

[Lecun, Y., & Bengio, Y. (1995). Convolutional Networks for Images, Speech, and Time-Series. *The Handbook of Brain Theory and Neural Networks*.](http://paperpile.com/b/FQGDmo/BoTl) <http://dx.doi.org/>

[Magnon, V., Vallet, G. T., & Auxiette, C. (2018). Sedentary Behavior at Work and Cognitive Functioning: A Systematic Review. *Frontiers in Public Health*, *6*, 239.](http://paperpile.com/b/FQGDmo/Hfl9)

[Mahata, S. K., Das, D., & Bandyopadhyay, S. (2019). MTIL2017: Machine Translation Using Recurrent Neural Network on Statistical Machine Translation. *Journal of Intelligent Systems*, *28*(3), 447–453.](http://paperpile.com/b/FQGDmo/OEcq)

[Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Education.](http://paperpile.com/b/FQGDmo/tsbr)

[Paing, A. C., McMillan, K. A., Kirk, A. F., Collier, A., Hewitt, A., & Chastin, S. F. M. (2018). The associations of sedentary time and breaks in sedentary time with 24-hour glycaemic control in type 2 diabetes. *Preventive Medicine Reports*, *12*, 94–100.](http://paperpile.com/b/FQGDmo/xvIZ)

[Pelletier, C., Webb, G., & Petitjean, F. (2019). Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series. In *Remote Sensing* (Vol. 11, Issue 5, p. 523). https://doi.org/](http://paperpile.com/b/FQGDmo/qKiY)[10.3390/rs11050523](http://dx.doi.org/10.3390/rs11050523)

[Steinhubl, S. R., Muse, E. D., & Topol, E. J. (2015). The emerging field of mobile health. *Science Translational Medicine*, *7*(283), 283rv3.](http://paperpile.com/b/FQGDmo/QxBt)

[Time series prediction for the epidemic trends of COVID-19 using the improved LSTM deep learning method: Case studies in Russia, Peru and Iran. (2020). *Chaos, Solitons & Fractals*, *140*, 110214.](http://paperpile.com/b/FQGDmo/C6fF)

[van den Oord, A., & Dieleman, S. (n.d.). *WaveNet: A Generative Model for Raw Audio*. Retrieved February 20, 2021, from](http://paperpile.com/b/FQGDmo/tNCm) <https://deepmind.com/blog/article/wavenet-generative-model-raw-audio>

[Wilmot, E. G., Edwardson, C. L., Achana, F. A., Davies, M. J., Gorely, T., Gray, L. J., Khunti, K., Yates, T., & Biddle, S. J. H. (2012). Sedentary time in adults and the association with diabetes, cardiovascular disease and death: systematic review and meta-analysis. *Diabetologia*, *55*(11), 2895–2905.](http://paperpile.com/b/FQGDmo/P5fD)

[Wu, C., Boukhechba, M., Cai, L., Barnes, L. E., & Gerber, M. S. (2018). Improving momentary stress measurement and prediction with bluetooth encounter networks. In *Smart Health* (Vols. 9–10, pp. 219–231). https://doi.org/](http://paperpile.com/b/FQGDmo/H0Qg)[10.1016/j.smhl.2018.07.017](http://dx.doi.org/10.1016/j.smhl.2018.07.017)

[Yan, H., & Ouyang, H. (2017). Financial Time Series Prediction Based on Deep Learning. *Wireless Personal Communications*, *102*(2), 683–700.](http://paperpile.com/b/FQGDmo/FOiI)

[Yerrakalva, D., Yerrakalva, D., Hajna, S., & Griffin, S. (2019). Effects of Mobile Health App Interventions on Sedentary Time, Physical Activity, and Fitness in Older Adults: Systematic Review and Meta-Analysis. *Journal of Medical Internet Research*, *21*(11). https://doi.org/](http://paperpile.com/b/FQGDmo/Kpxj)[10.2196/14343](http://dx.doi.org/10.2196/14343)

[Zia, J., Tadayon, A., McDaniel, T., & Panchanathan, S. (2016). Utilizing Neural Networks to Predict Freezing of Gait in Parkinson’s Patients. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS ’16*. https://doi.org/](http://paperpile.com/b/FQGDmo/SRc7)[10.1145/2982142.2982194](http://dx.doi.org/10.1145/2982142.2982194)

1. https://www.sedentarybehaviour.org/ [↑](#footnote-ref-0)
2. https://www.pewresearch.org/internet/fact-sheet/mobile/ [↑](#footnote-ref-1)
3. https://github.com/martinscooper/Making-it-linearly-separable [↑](#footnote-ref-2)
4. https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/ [↑](#footnote-ref-3)
5. https://enterprise.foursquare.com/products/places [↑](#footnote-ref-4)
6. https://ieeexplore.ieee.org/Xplore/home.jsp [↑](#footnote-ref-5)
7. https://dl.acm.org/ [↑](#footnote-ref-6)
8. https://www.wiley.com/en-ar [↑](#footnote-ref-7)
9. https://www.sciencedirect.com/ [↑](#footnote-ref-8)
10. https://www.scopus.com/home.uri [↑](#footnote-ref-9)
11. https://keras.io/ [↑](#footnote-ref-10)
12. https://github.com/martinscooper/tesis-project [↑](#footnote-ref-11)
13. <https://sites.google.com/site/compendiumofphysicalactivities/home> [↑](#footnote-ref-12)
14. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html> [↑](#footnote-ref-13)
15. <https://scikit-optimize.github.io/stable/> [↑](#footnote-ref-14)