UNIVERSITAT JAUME I

Grado en Matemática Computacional

Trabajo Final de Grado

Modelos de predicción en series temporales: Un estudio comparativo entre métodos estadísticos y machine learning

Autor:
Francisco Peris Tena

Tutor académico:
Jorge Mateu Mahiques

Fecha de lectura: 15 de octubre de 2022
Curso académico 2021/2022

Resumen

Este trabajo de fin de grado recoge y organiza datos de la enfermedad COVID-19 desde mayo del año 2020 hasta marzo 2022 de los 20 municipios más poblados de la provincia de Castellón. Por cada municipio se tiene una serie temporal univariante, positivos COVID-19 en los últimos 14 días, a la cual se aplican 7 modelos para estimar futuros valores de la serie. Estos modelos son Naive, SARIMA, Holt Winter's, MLP, LSTM, ConvLSTM y CNN-LSTM.

Por cada municipio y modelo se aplica *Walk-forward optimization* para obtener la mejor configuración de hiperparámetros para la serie temporal en concreto. Finalmente, se suman los errores cometidos al estimar por la mejor configuración de cada modelo en los 20 municipios para poder realizar una comparación de los modelos.

Palabras clave

COVID-19, Walk-forward optimization, SARIMA, Holt Winter's, MLP, LSTM, ConvLSTM, CNN-LSTM.

Keywords

COVID-19, Walk-forward optimization, SARIMA, Holt Winter's, MLP, LSTM, ConvLSTM, CNN-LSTM.

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto se desarrolla tras la pandemia mundial causada por la enfermedad COVID-19 que ha afectado a día de hoy más de 620 millones de personas y provocando la muerte de más de 6 millones de personas a nivel mundial (COVID Live - Coronavirus Statistics - Worldometer, s. f.). Sin lugar a dudas, la enfermedad ha sido el tema central de los últimos años con innumerables esfuerzos para limitar sus efectos. Por ello, predecir el número de infectados en los siguientes días puede ser fundamental a la hora de tomar decisiones y mitigar la expansión de la enfermedad (Remuzzi, 2020).

El conjunto de datos estudio fue inicialmente la Comunidad Valenciana a nivel municipal, posteriormente se redujo a los 20 municipios más poblados de la provincia de Castellón debido al tiempo necesario para entrenar los modelos. Sin embargo, las conclusiones obtenidas en la muestra esperan poder aportar un grano de arena al conocimiento y estudio de los efectos de la enfermedad en la provincia de Castellón. La procedencia del autor y la localización de la *Universitat Jaume I* ha sido determinante para la elección de los datos.

Las nuevas técnicas y avances para analizar y predecir de series de tiempo permiten un estudio muy amplio del conjunto de datos seleccionado. Los modelos utilizados en el proyecto son bastante utilizados por la comunidad científica en la actualidad y realizar este estudio ha permitido al autor conocer y aprender a utilizar los modelos así como las características más importantes de cada uno. Se han utilizado 7 modelos, un modelo base *Naive*, modelos estadísticos clásicos (SARIMA y Holts Winter's) y redes neuronales (MLP, LSTM, ConvLSTM y CNN-LSTM).

Por último, el hecho de comparar los modelos entre sí nos aporta mucha información y también nos permite concluir que modelos debemos aplicar y nos aportan más valor al análisis y a la hora de predecir nuevos valores. Aunque las conclusiones están estrechamente relacionadas con el conjunto de datos utilizado, el autor espera que los resultados obtenidos en el proyecto sean de utilidad para otros proyectos.

Capítulo 2

Objetivos

Hoy en día existen nuevos métodos para realizar predicciones en series temporales, algunos de ellos se pueden englobar dentro del *Machine Learning* que desde 2015 ha crecido su popularidad hasta máximos históricos (Google Trends, s. f.). Además, el *Machine Learning* es especialmente bueno encontrando características y patrones para ser extrapolados a nuevas observaciones (Alameda, 2022). Los métodos estadísticos hasta hace poco habían sido los elegidos dado que tratan mejor la incertidumbre que las redes neuronales, aunque su popularidad parece un tanto oscurecida desde la introducción de la inteligencia artificial y el *Machine Learning* (Google Trends, s. f.). Veamos si esta justificado este cambio de paradigma comparando ambos métodos analizando los resultados obtenidos, ventajas y desventajas.

Un primer objetivo es comprobar si los modelos aportan valor a la hora de predecir nuevas observaciones. Para poder concluir esto el trabajo incluye un modelo base llamado *Naive* el cual se trata de un método que escogerá un valor persistente, una media móvil o una mediana de los datos de entrenamiento para predecir nuevos valores. Es decir, de un método que no sea capaz de batir el resultado de este método podemos concluir que no aporta valor, porque además de obtener peores resultados será computacionalmente mucho más costoso.

Un segundo objetivo será comparar los resultados obtenidos de los métodos estadísticos frente a los métodos basados en aprendizaje estadístico (*Statistical Learning*). Tanto los resultados obtenidos, tiempo de entrenamiento y complejidad se tendrán en cuenta a la hora de obtener conclusiones sobre si debemos utilizar métodos estadístico como se venía haciendo hasta ahora o adoptar los nuevos métodos basados en *Machine Learning*.

Por último, un tercer objetivo es averiguar cual es el mejor método en términos de predicción y tratar de explicar las causas de ello, teniendo en cuenta los resultados, tiempo de entrenamiento, complejidad y configuración. La configuración es muy importante porque realizamos

una búsqueda de configuraciones, es decir, no decidimos cual es la configuración de los métodos sino que nos quedaremos con la óptima. Por tanto, analizar cuales han sido los hiperparámetros con mejores resultados nos puede dar *insights* de los métodos y como configurarlos de forma óptima. Aunque como se ha mencionado anteriormente los resultados no son extrapolables de forma generalista a otros problemas, si nos puede dar indicios y pistas para nuevos estudios.

Capítulo 3

Conjunto de datos

3.1. Recogida de datos

Los datos han sido obtenidos de la Generalidad Valenciana en su página web de datos abiertos[1] desde el principio de la pandemia mayo de 2020 hasta marzo de 2022 se publicaba periódicamente una tabla con los datos de la enfermedad COVID-19 a nivel municipal.

Los datos eran publicados aproximadamente cada 3 o 4 días con una fila por municipio y las siguientes columnas:

- CodMunicipio (entero), código de municipio según la codificación de la INE.
- Municipio (texto), denominación del municipio.
- Casos PCR+ (entero), número de casos acumulados desde el 31 de enero de 2020.
- Incidencia acumulada PCR+ (número real)

$$Indicencia\ acumulada\ PCR+ = \frac{Casos\ PCR+}{Número\ de\ habitantes} \times 100000$$

- Casos PCR+ 14 días (entero), número de casos acumulados los últimos 14 días.
- Indicencia acumulada PCR+ 14 días (número real)

$$Indicencia\ acumulada\ PCR+\ 14 : d\acute{i}as = \frac{Casos\ PCR+\ 14\ d\acute{i}as}{Número\ de\ habitantes} \times 100000$$

- Defunciones (entero), número de defunciones causadas por la enfermedad COVID-19 desde el 31 de enero de 2020.

[1] `https://dadesobertes.gva.es/es/dataset?tags=COVID-19`

- Tasa de defunción (número real)

$$Tasa\ de\ defunción = \frac{Defunciones}{Número\ de\ habitantes} \times 100000$$

Para poder crear la serie temporal de cada municipio fue necesario un procesamiento de los datos realizado mediante Python para crear un *DataFrame* (estructura de datos de la librería Pandas) por municipio seleccionando de cada archivo la fila correspondiente al municipio. Inicialmente se realizó este proceso para todos los municipios y columnas, pero debido al coste computacional de entrenar a las redes neuronales se decidió solo seleccionar los 20 pueblos más poblados de la provincia de Castellón y solo utilizar la columna Casos PCR+ 14 días. Debido a la forma de reporte de los datos se ha aplicado una media móvil de 4 periodos para eliminar de los datos los picos que se producían en las publicaciones posteriores al fin de semana.

3.2. Análisis descriptivo de los datos

Seleccionamos como muestra los municipios de Castellón, Alcora y Borriol para realizar un análisis descriptivo de los datos. Los cuales intenta ser representativos del conjunto de municipios total dado que Castellón es el municipio más grande, Alcora uno mediano y Borriol el más pequeño.

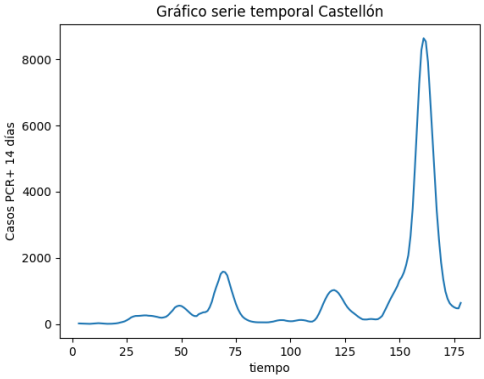
El análisis estadístico del subconjunto de municipios seleccionado es:

Casos PCR+ 14 días	Castellón	Alcora	Borriol
count	176	176	176
mean	833.01	55.68	22.63
std	1627.85	114.81	62.76
min	4	0	0
25 %	104.5	4	2
50 %	257	14	6
75 %	753.75	40.25	22
max	8637	571	358

Cuadro 3.1: Análisis estadístico el subconjunto de municipios de la columna.

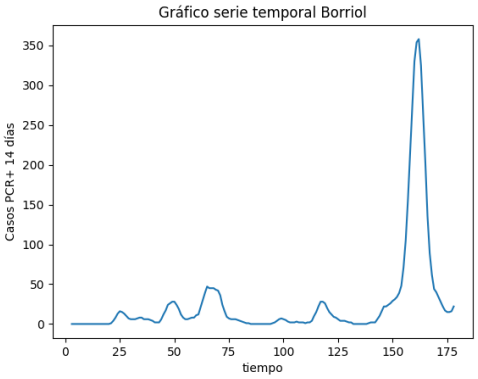
Se puede ver que el tamaño de la población del municipio es determinante en el número de casos producidos por la enfermedad. Por otro lado, se puede ver que la estructura de los cuartiles es la misma. Los gráficos de la serie temporal:

En las series temporales podemos apreciar que están relacionadas y tienen un patrón similar. Se puede apreciar claramente tres subidas y bajadas pronunciadas:



(a) Castellón

(b) Alcora



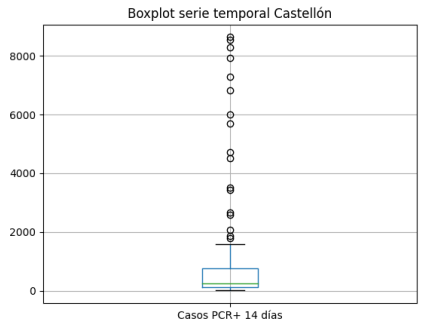
(c) Borriol

Figura 3.1: Series temporales subconjunto municipios

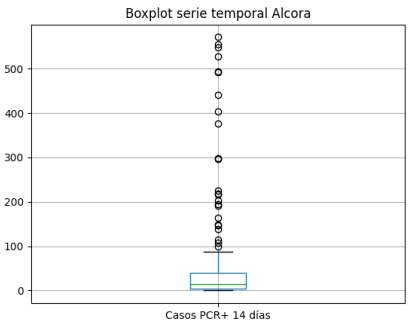
1. $t = [60, 75]$, este periodo se corresponde con principios del año 2021.
2. $t = [110, 120]$, este periodo se corresponde con el verano del año 2021.
3. $t = [150, 170]$, este periodo se corresponde con principios del año 2022.

Teniendo en cuenta lo anterior podemos entender que los datos tienen una componente estacional. Para visualizar la distribución de los datos utilizamos los gráficos *boxplots*.

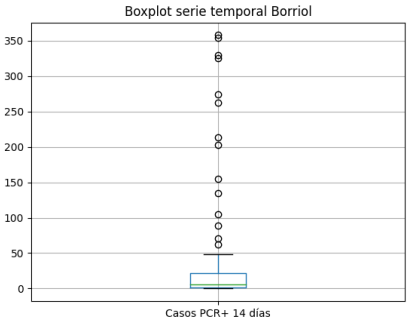
Salvo los periodos estacionales de subidas y bajadas pronunciadas las series temporales mantienen un rango estable sin tendencia dado que podemos ver que el rango intercuartílico es bastante estrecho a comparación de los valores máximos atípicos. Estos valores atípicos podemos identificarlos con los periodos de crecimiento y decrecimiento de casos estacionales.



(a) Castellón



(b) Alcora



(c) Borriol

Figura 3.2: *Boxplots* subconjunto municipios

Capítulo 4

Modelos

4.1. Modelo base

4.1.1. Naive

La principal razón de utilizar este modelo es para utilizarlo como base de comparación con los siguientes modelos. Este modelo analiza tres estrategias distintas:

- Persistente, el nuevo valor estimado es un valor anterior de la serie temporal.
- Media, el nuevo valor estimado es la media de n valores anteriores de la serie temporal.
- Mediana, el nuevo valor estimado es la media de n valores anteriores de la serie temporal.

Para cada serie temporal se probarán todas las configuraciones posibles para obtener el mejor modelo para el conjunto de datos. El modelo se ha llamado *Naive* por la falta de complejidad del mismo.

4.2. Modelos estadísticos

4.2.1. Introducción

Una serie temporal es una sucesión de un conjunto de datos medidos sobre una característica (univariante) o sobre varias características (multivariante) ordenados cronológicamente. Las series temporales utilizadas en el trabajo son univariantes y discretas (las observaciones se miden

cada cierto tiempo discreto). Podemos representar matemática la serie temporal univariante como

$$(y_t)_{t=1}^{N},\ (y_t : t = 1, \ldots, N)$$

donde la y_t es la observación número t de la serie y N el número total de observaciones.

Las series temporales se ven afectadas por 4 componentes:

- Tendencia (*trend*) T, representa la tendencia general de la serie temporal a crecer, decrecer o estar en un rango a lo largo de un período de tiempo.
- Variación de temporada (*Seasonal variation*) S, representa las variaciones que se suelen producir dentro de una temporada.
- Variaciones cíclicas (*Cyclical variation*) C, representa las variaciones que se producen cada cierto tiempo, que se repiten en ciclos.
- Variaciones irregulares (*Irregular variation*) I, representa las variaciones que no podemos explicar y no se repiten ningún patrón.

Existen generalmente dos tipos modelos usados:

- Modelo aditivo (*Additive Model*), se supone que las componentes son independientes entre ellas.
$$Y(t) = T(t) + S(t) + C(t) + I(t)$$
- Modelo multiplicativo, se supone que las componentes no son necesariamente independientes entre ellas. (*Multiplicative Model*)
$$Y(t) = T(t) \times S(t) \times C(t) \times I(t)$$

Podemos construir modelos de predicción de series temporales asumiendo que la serie temporal sigue un proceso estocástico. Un proceso estocástico es una sucesión de variables aleatorias indexadas en orden cronológico y equidistantes sobre una característica (proceso univariante) o varias características (proceso multivariante). Podemos representar matemáticamente los procesos estocásticos univariantes como

$$\ldots, Y_{-1}, Y_0, Y_1, \ldots\ (Y_t : t = 0, \pm 1, \pm 2, \ldots)$$

donde Y_t es la variable aleatoria escalar en el momento t. Por tanto, tenemos que la procedencia de una serie temporal es en realidad las observaciones realizadas de un proceso estocástico.

$$\ldots, Y_{-1}, Y_0, Y_1, \cdots \rightarrow y_1, y_2, \ldots, y_N$$

Así, la serie temporal será un periodo muestral que pertenece a una parte del proceso estocástico de donde la serie proviene.

Existen dos tipos de procesos estocásticos:

- Procesos estocásticos estacionarios, su distribución de probabilidad varía a lo largo del tiempo de forma más o menos constante. Se cumple cuando las propiedades estadísticas son semejantes entre dos secuencias finitas de componentes Y_t separadas por un número entero h cualquiera.
- Procesos estocásticos no estacionarios, su distribución de probabilidad varía a lo largo del tiempo de forma no constante. Se cumple cuando las propiedades estadísticas son diferentes entre dos secuencias finitas de componentes Y_t para al menos un número entero $h > 0$.

4.2.2. SARIMA

El modelo SARIMA es un acrónimo de *Seasonal Autoregressive Integrated Moving-Average* utiliza variaciones y regresiones para calcular nuevos valores de las series temporales asumiendo que los datos futuros pueden ser explicados por valores anteriores y por variables no independientes. El modelo se compone de las siguientes componentes para una serie temporal $\{x_t\}$:

- Componente temporal (*Seasonal component*) S(P, D, Q, M)
  - P, indica el orden de la componente autoregresiva.
  - D, indica el orden de la componente de integración.
  - Q, indica el orden de la componente media móvil.
  - M, indica número de periodos por temporada.
- Componente autorregresivo (*Autoregressive component*) AR(p), el modelo supone que el valor a estimar esta relacionado con un número p de observaciones anteriores (*lagged observations*). Ejemplo, si AR(p=1) se llama *Random Walk* que significa que el valor x_t solo depende del valor anterior x_{t-1} y una variable aleatoria w_t (*white noise term*).

  $$x_t = x_{t-1} + w_t$$

  Para un modelo con orden p tenemos

  $$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + w_t = \sum_{i=1}^{p} \phi_i x_{t-i} + w_t$$

  donde $w_t \sim wn(0, \sigma_w^2)$ es ruido blanco, y $\phi_1, \phi_2, ...\phi_p$ ($\phi_p \neq 0$) son parámetros del modelo.
- Componente integrado (*Integrated component*) I(d), d es el número de veces que diferencias la serie de tiempo para hacerla estacionaria, se llama *degree of difference*. El ecuación para obtener la primera diferencia es

  $$\nabla x_t = x_t - x_{t-1}$$

- Componente media móvil (*Moving Average component*) MA(q), el modelo supone que el valor a estimar esta relacionado con un número q de términos de ruido blanco anteriores (*past white noise terms*). Para un modelo con orden q tenemos

$$x_t = w_t + \theta_1 w_{t-1} + \cdots + \theta_q w_{t-q} = w_t + \sum_{j=1}^{q} \theta_j w_{t-j}$$

donde $w_t \sim wn(0, \sigma_w^2)$ es ruido blanco, y $\theta_1, \theta_2, ...\theta_p$ ($\theta_p \neq 0$) son parámetros del modelo.

El modelo SARIMA se obtiene al combinar todos los componentes. Para simplificar la notación utilizaremos el operador *backshift* que se define como $BX_t = Xt - 1$ que se puede extender a $B^k X_t = X_{t-k}$. Ahora podemos simplificar la ecuaciones de las componentes:

- S(P, D, Q, M)
  - AR(P), definimos el operador autorregresivo estacional (*seasonal autoregresive operator*) como
$$\Phi_P(B^s) = 1 - \Phi_1 B^s - \Phi_2 B^{s2} - \cdots - \Phi_P B^{sP}$$
Por lo que, la componente AR(p) puede ser escrita como
$$\Phi_P(B^s) x_t = w_t$$
  - I(D), definimos el operador de diferencias estacional (*seasonal differences operator*) como
$$\nabla_s^D = (1 - B^s)^D$$
Por lo que, la componente I(D) puede ser escrita como
$$\nabla_s^D x_t = (1 - B^s)^D x_t$$
  - MA(Q), definimos el operador de media móvil estacional (*seasonal moving average operator*) como
$$\Theta_Q(B^s) = 1 + \theta_1 B^s + \theta_2 B^{s2} + \cdots + \theta_Q B^{sQ}$$
Por lo que, la componente MA(Q) puede ser escrita como
$$x_t = \theta_Q(B^s) w_t$$
- AR(p)

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + w_t \Longrightarrow x_t - \phi_1 x_{t-1} - \phi_2 x_{t-2} - \cdots - \phi_p x_{t-p} = w_t$$
$$(1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p) x_t = w_t$$

Definimos el operador autorregresivo (*autoregresive operator*) como

$$\phi_p(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p$$

Por lo que, la componente AR(p) puede ser escrita como

$$\phi_p(B) x_t = w_t$$

- I(d)

$$\nabla x_t = x_t - x_{t-1} = x_t - Bx_t = (1 - B)x_t$$

Se puede generalizar para orden d, y obtenemos el operador de diferencias (*differences operator*) que se define como

$$\nabla^d = (1 - B)^d$$

Por lo que, la componente I(d) puede ser escrita como

$$\nabla^d x_t = (1 - B)^d x_t$$

- MA(q)

$$x_t = w_t + \theta_1 w_{t-1} + \cdots + \theta_q w_{t-q} \Longrightarrow x_t = (1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q) w_t$$

Definimos el operador media móvil (*moving average operator*) como

$$\theta_q(B) = 1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q$$

Por lo que, la componente MA(q) puede ser escrita como

$$x_t = \theta_q(B) w_t$$

Así un modelo multiplicativo SARIMA puede escribirse como

$$\Phi_P(B^s)\phi_p(B)\nabla_s^D \nabla^d x_t = \theta_q(B)\Theta_Q(B^s)w_t$$

4.2.3. Holts Winter's

Para introducir este modelo veamos en primer lugar algunos conceptos:

- Media móvil (*moving average*), podemos utilizarla para predecir los futuros valores de la serie temporal como la media móvil de los k valores previos.

$$\hat{y}_t = \frac{1}{k} \sum_{n=1}^{k} y_{t-n}$$

- Média móvil ponderada (*Weighted average*), similar a la media móvil pero dando mayor importancia a los últimos valores.

$$\hat{y}_t = \sum_{n=1}^{k} w_n y_{t-n}$$

- Nivel (*level*), es la media en la serie temporal en un intervalo de tiempo concreto.

El método Holts Winter's es un modelo basado métodos de suavizado (*Smoothing Methods*) y es una mejora de modelos más sencillos. Las componentes de los métodos de suavizado puede combinarse de forma aditiva y multiplicativa dando lugar a muchas combinaciones. Veamos las características más importantes de cada uno hasta llegar al modelo utilizado en el trabajo:

- Suavizado exponencial simple (*Simple Exponential Smoothing*) SES, este método funciona con series de tiempo estacionarias, sin tendencia y sin estacionalidad. La formula de este método es

$$\hat{y}_t = \alpha \cdot y_t t - 1 + (1 - \alpha) \cdot \hat{y}_{t-1}$$

  donde $\hat{y}_t$ es el valor a estimar, y_{t-1} es el valor anterior en la serie, $\hat{y}_{t-1}$ es el valor previo estimado y α es el factor de suavidad ($0 \leq \alpha \leq 1$).

- Suavizado exponencial doble (*Double Exponential Smoothing*) DES o (*Holt's Exponential Smoothing*) HES, este método funciona con series de tiempo estacionarias, con tendencia y sin estacionalidad. Este método permite a la serie tener tendencia y nivel. La formula de este método es

$$l_t = \alpha y_t + (1 - \alpha)(l_{t-1} + b_{t-1})$$
$$b_t = \beta(l_t - l_{t-1} + (1 - \beta)b_{t-1}$$
$$\hat{y}_{t+1} = l_t + b_t$$

  donde l_t es el nivel de la serie en el momento t, l_{t-1} es el nivel de la serie en el momento $t - 1$, b_t es la tendencia de la serie en el momento t, b_{t-1} es la tendencia de la serie en el momento $t - 1$, $\hat{y}_{t+1}$ es el valor a estimar, y_t es el valor de la serie en el momento t, α es el factor de suavidad del nivel ($0 \leq \alpha \leq 1$) y β es el factor de suavidad de la tendencia ($0 \leq \beta \leq 1$).

- Suavizado exponencial triple (*Triple Exponential Smoothing*) TES o (*Holt-Winter's*, este método funciona con series de tiempo estacionarias, con tendencia y con estacionalidad. La formula de este método es

$$l_t = \alpha(y_t - s_{t-p}) + (1 - \alpha)(l_{t-1} + b_{t-1})$$
$$b_t = \beta(l_t - l_{t-1}) + (1 - \beta)b_{t-1}$$
$$s_t = \gamma(y_t - l_t) + (1 - \gamma)s_{t-p}$$
$$\hat{y}_{t+m} = l_t + mb_t + s_{t-p+1+(m-1)mod\,p}$$

  donde l_t es el nivel de la serie en el momento t, l_{t-1} es el nivel de la serie en el momento $t - 1$, b_t es la tendencia de la serie en el momento t, b_{t-1} es la tendencia de la serie en el momento $t-1$, s_t es la estacionalidad de la serie en el momento t, s_{t-p} es la estacionalidad de la serie en el momento $t - p$, $s_{t-p+1+(m-1)mod\,p}$ es la estacionalidad de la serie en el momento $t - p + 1 + (m - 1)mod\,p$, $\hat{y}_{t+m}$ es el valor a estimar en el momento $t + m$, y_t es el valor de la serie en el momento t, α es el factor de suavidad del nivel ($0 \leq \alpha \leq 1$), β es el factor de suavidad de la tendencia ($0 \leq \beta \leq 1$) y γ es el factor de suavidad de la estacionalidad ($0 \leq \gamma \leq 1$).

4.3. Redes neuronales

4.3.1. Introducción

Para introducir el funcionamiento de las redes neuronales utilizaremos la neurona *threshold logic unit* y la arquitectura *Perceptron* que son la base para la red neuronal *Multilayer Perceptron* (MLP). Un *Perceptron* es una arquitectura de la redes neuronales basada en una neurona llamada *threshold logic unit* (TLU) inventada por Frank Rosenblatt en el año 1957. Las entradas (*inputs*) y salidas (*outputs*) de la neurona TLU son números reales y cada entrada tiene un peso asociado.

La neurona TLU realiza la suma ponderada de las conexiones de las entradas y el *bias*, el resultado es introducido como entrada de la función de activación (*step function*) cuya salida es la salida de la neurona. La composición de varias neuronas TLU en una capa se llama *Perceptron*. En la Figura 4.1 podemos ver un ejemplo de un *Perceptron*.

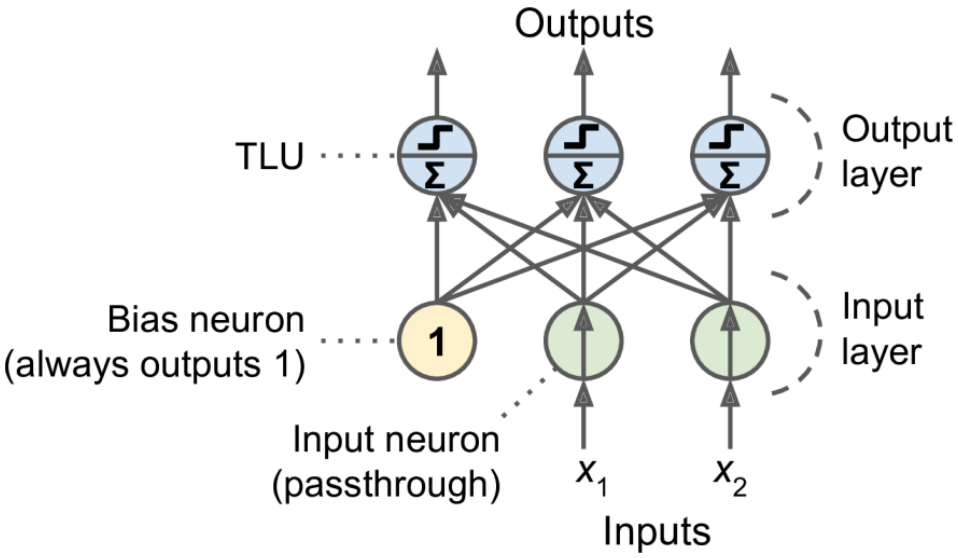


Figura 4.1: Ejemplo de arquitectura *Perceptron*. Fuente: *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow* (p. 286), por Géron, 2022.

Teniendo en cuenta el funcionamiento de las neuronas, podemos expresar matemáticamente esto con la ecuación (4.1) para calcular la salida la red de una capa densa h (cada neurona esta conectada con todas las neuronas de la capa anterior y siguiente).

$$h_{W,b}(X) = \phi(XW + b) \tag{4.1}$$

Donde X representa la matriz de entradas tendrá un fila por instancia y una columna por cada variable, W representa la matriz de pesos de las conexiones tendrá una fila por neurona de entrada y una columna por neurona de salida, b representa el vector de pesos de la variable *bias* tendrá un valor para cada neurona y ϕ representa la función de activación.

Para que las redes neuronales devuelvan buenos resultados han de ser entrenadas con los datos de entrenamiento para ajustar progresivamente los pesos de las conexiones y *bias* de cada neurona. Esto fue posible gracias a que en el año 1986 David Rumelhart, Geoffrey Hinton y Ronald Williams publicaron un trabajo sobre *backpropagation* que nos permite entrenar a la redes neuronales usando el algoritmo *gradient descent* el cual nos permite encontrar rápidamente el mínimo local de una función diferenciable. Calcular de forma automática los gradientes se llama *automatic differentiation* o *autodiff*, en el algoritmo de *backpropagation* utilizamos la técnica *reverse-mode autodiff* que especialmente apropiada para muchas variables pero con pocas salidas.

La idea fundamental del entrenamiento de redes neuronales puede expresarse matemáticamente con la ecuación (4.2) que se utiliza para calcular los pesos en cada iteración de entrenamiento.

$$w_{i,j}^{(next\ step)} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \tag{4.2}$$

En la ecuación (4.2) $w_{i,j}$ representa el peso de la conexión entre las neurona i-ésima y la j-ésima, x_i representa el valor de entrada i-ésimo de la instancia actual, $\hat{y}_j$ es el valor j-ésimo de la salida de la neurona para la instancia actual, y_j el valor j-ésimo de salida objetivo de la neurona para la instancia actual y η es el ratio de aprendizaje (*learning rate*).

Teniendo en cuenta todo lo anterior el algoritmo para entrenar la redes neuronales sigue generalmente los siguientes pasos:

- La red neuronal tiene como entrada una instancia o instancias (*mini-batch* o *batch*) de los datos de entrenamiento. Para entrenar el modelo suele ser necesario varios *epochs*[1].
- Las capas ocultas obtienen las salidas de las capa anterior (la primera capa oculta recibe las entradas de la capa de entrada) cada neurona de la capa calcula su salida sumando los pesos de las conexiones y el *bias*. El resultado es la entrada de la función de activación y la salida de la función de activación será la salida de la neurona.
- Los valores obtenidos en la capa salida son comparados con los valores esperados de la muestra mediante una función de pérdida (*loss function*) que nos devolverá el error cometido por la red al estimar el valor esperado.
- Ahora tenemos que modificar aquellos pesos en las conexiones entre las neuronas y *bias* que han contribuido a generar error al estimar los valores esperados. Para ello, el algoritmo

[1]Cada vez que la red entrena con todos los datos de la muestra se llama *epoch*.

de *backpropagation* ajusta los pesos de las conexiones y *bias* calculando el gradiente (la derivada) utilizando generalmente *gradient descent* para propagar los errores hacia atrás.

En el trabajo se ha utilizado *adaptative moment estimation* ADAM como optimizador para todas las redes neuronales. Este variante tiene en cuenta la decadencia exponencial media de los pasado gradientes (*momentum optimization*) y la decadencia exponencial media de las raíces cuadradas de los pasados gradientes (RMSProp).

Por último, cuando entrenamos una red neuronal podemos configurar algunos parámetros que determinan como es la red neuronal y como esta entrena. Los llamamos hiperparámetros de la red neuronal:

- Número de capas ocultas de la red neuronal.
- *Learning rate*, determina como de rápido es el avance hacia el mínimo de la función de pérdida. Un valor muy bajo provoca que necesitemos muchas iteraciones para alcanzar el mínimo pero un un valor muy alto provocará saltos en el acercamiento al mínimo.
- *Batch size*, número de instancias propagadas en la red antes de reajustar los pesos mediante el algoritmo *backpropagation*. A mayor número de instancias se acumulará mayor error en las estimaciones y necesitaremos menos iteraciones para realizar un *epoch* pero la red puedes ajustarse peor al propagar los errores acumulados.
- Función de activación (*activation function*), permite a las capas sean capaces de aprender relaciones no lineales (dado que con multiplicaciones de los pesos y sumas (transformaciones afines) el modelo solo seria capaz de representar relaciones lineales). En este trabajo se ha utilizado la función de activación *exponential linear unit* ELU.

$$ELU_\alpha(z) = \begin{cases} \alpha(e^z - 1) & z < 0 \\ z & z \geq 0 \end{cases}$$

  Se ha escogido esta función de activación porque para valores negativos la función tiene un valor de salida cercano a 0 y eso alivia los problemas de desvanecimiento de los gradientes (*vanishing gradients*). Además, evita tener neuronas muertas (salida 0) dado que la función de activación permite tener gradientes no cero para valores negativos. Por último, si el valor de $\alpha = 1$ la función es suave alrededor del 0 y eso evita saltos cuando aplicamos el algoritmo de *gradient descent*.
- Función de pérdida (*step function loss*), mide el error entre el valor esperado y el valor estimado por la red. La función de pérdida utilizada en el trabajo ha sido *mean squared error* MSE.

$$E[(X - \hat{X})^2]$$

- Optimizador, determina como la red se actualizará basándose en la función de pérdida implementa alguna variante de *gradient descent*, en este trabajo utilizamos *adaptive moment estimation* ADAM. Este optimizador combina las estrategias de *momentum* y *RMSProp*. Las ecuaciones del algoritmo ADAM son:

$$m = \beta_1 m - (1 - \beta_1)\nabla_\Theta J(\Theta) \tag{4.3}$$

$$s = \beta_2 s - (1 - \beta_2)\nabla_\Theta J(\Theta) \otimes \nabla_\Theta J(\Theta) \tag{4.4}$$

$$\hat{m} = \frac{m}{1 - \beta_1^t} \tag{4.5}$$

$$\hat{s} = \frac{s}{1 - \beta_2^t} \tag{4.6}$$

$$\Theta = \Theta + \eta \hat{m} \oslash \sqrt{\hat{s} + \epsilon} \tag{4.7}$$

En la ecuación (4.3), m representa un vector de momento (*momentum*), $\nabla_\Theta J(\Theta)$ representa el gradiente de la función de coste J evaluada con los pesos Θ y β_1 es un hiperparámetro que indica la decaimiento del momento (generalmente $\beta_1 = 0{,}9$). El vector de momento va decayendo progresivamente pero hace que los pasos tomados hacia el mínimo local sean más rápido al principio, al cual vamos restando los gradientes calculados en cada iteración y ajustados por el hiperparámetro β_1.

En la ecuación (4.4), s representa un vector que acumula el cuadrado de los gradientes en cada dimensión, $\nabla_\Theta J(\Theta)$ representa el gradiente de la función de coste J evaluada con los pesos Θ, β_2 es un hiperparámetro que indica el decaimiento del escalado de los gradientes (generalmente $\beta_2 = 0{,}999$) y $\otimes$ indica multiplicación de matrices elemento a elemento. Esta ecuación es equivalente a $s_i = s_i + (\partial J(\Theta)/\partial\theta_i)^2$. Esta ecuación permite que aquellas dimensiones del gradiente que más crezcan sean agrandadas para que los pasos en las direcciones que nos acercan al mínimo local mucho más rápido.

En la ecuación (4.5), $\hat{m}$ representa el vector de momento ajustado, m representa el vector de momento, β_1^t representa el valor del hiperparámetro que indica la decaimiento del momento en la iteración t. Esta ecuación realiza la función de ajustar rápidamente el valor de m dado que se suele inicializar a 0.

En la ecuación (4.6), $\hat{s}$ representa el vector que acumula el cuadrado de los gradientes en cada dimensión ajustado, s representa el vector que acumula el cuadrado de los gradientes en cada dimensión, β_2^t es un hiperparámetro que indica el decaimiento del escalado de los gradientes en la iteración t. Esta ecuación realiza la función de ajustar rápidamente el valor de s dado que se suele inicializar a 0.

En la ecuación (4.7), Θ representa el vector peso, η el tasa de aprendizaje (*learning rate*), $\hat{m}$, $\hat{s}$, ϵ es un término de suavizado (generalmente inicializado $\epsilon = 10^-7$ y $\oslash$ representa la división de matrices elemento a elemento. Esta ecuación suma al vector peso de la iteración anterior el vector de momento ajustado $\hat{m}$ multiplicado por la tasa de aprendizaje η pero escalada hacia abajo por el factor $\sqrt{\hat{s} + \epsilon}$ para corregir las direcciones donde exista mayor acumulación del vector s, es decir, a medida que nos acercamos al mínimo local queremos

dejar de dar grandes pasos incluso en las direcciones donde en iteraciones anteriores el gradiente era elevado.

- *Dropout rate*, indica la probabilidad de que la salida de una capa se elimine. Esto se utiliza para evitar el *overfitting* (la red no aprende solo las patrones extrapolables de la muestra sino que también aprende características especificas de los datos de entrenamiento) de la red neuronal con los datos de entrenamiento.

4.3.2. Multilayer Perceptron (MLP)

Al conectar varias capas *Perceptron* obtenemos una red neuronal llamada *Multilayer Perceptron* (MLP). Las capas se dividen en *input layer*, *hidden layer* y *output layer*. Generalmente si en la red hay dos o más capas ocultas se puede considerar una red neuronal profunda (*deep neural network* (DNN)).

La red MLP es un ejemplo de las arquitecturas de redes neuronales *feedforward neural network* (FNN), es decir, el flujo de las entradas es unidireccional. En la Figura 4.2 podemos ver un ejemplo de una red MLP con una capa de entrada, una capa oculta y una capa de entrada:

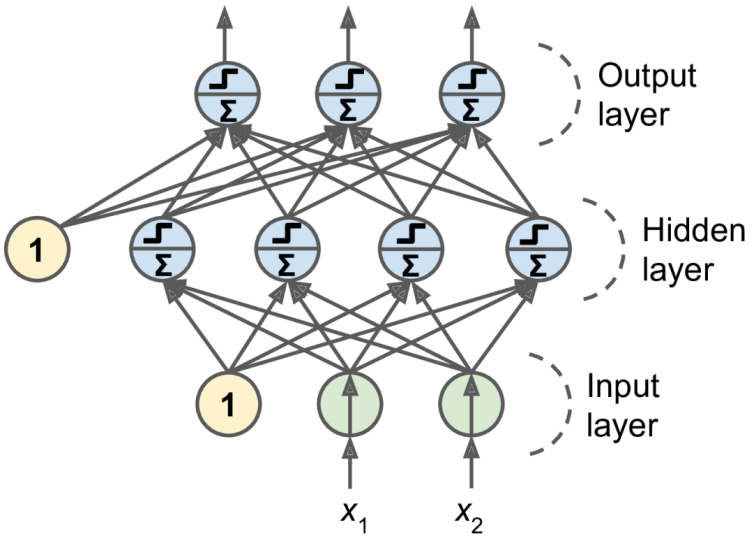


Figura 4.2: Ejemplo de arquitectura MLP. Fuente: *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow* (p. 289), por Géron, 2022.

4.3.3. LSTM

La red neuronal *Long Short-Term Memory* LSTM es una red neuronal recurrente (*Recurrent Neural Network*) RNN que toma el nombre de la neurona LSTM propuesta en el año 1997 por Sepp Hochreiter y Jürgen Schmidhuber. El funcionamiento de las redes neuronales recurrentes son similares a las *Feedforward Neural Network* FNN pero añaden conexiones también a capas anteriores para que la neurona reciba como entrada, la salida de la iteración anterior. En la Figura 4.3 podemos ver el funcionamiento de una neurona en una RNN.

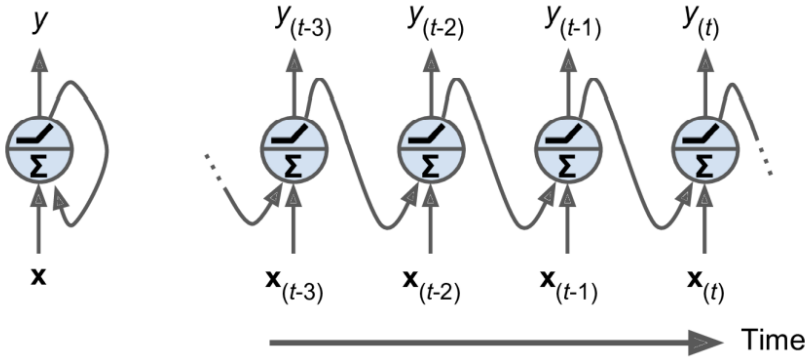


Figura 4.3: Ejemplo de una neurona recurrente. Fuente: *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow* (p. 498), por Géron, 2022.

En la Figura 4.3 podemos apreciar cierta similitud cuando en un RNN se realizan varias iteraciones con la forma de estimar nuevos valores en una serie temporal, es decir, para estimar el valor y_t la red tiene en cuenta el valor anterior y_{t-1} que a su vez tiene en cuenta el valor y_{t-2} y sucesivamente. Por ello, las correlaciones entre valores puede ser tenidas en cuenta e incluso ajustando los pesos se puede dar más peso a los últimos valores como se realiza con las métodos de alisado. Consecuentemente, este tipo de redes neuronales son apropiadas para series temporales.

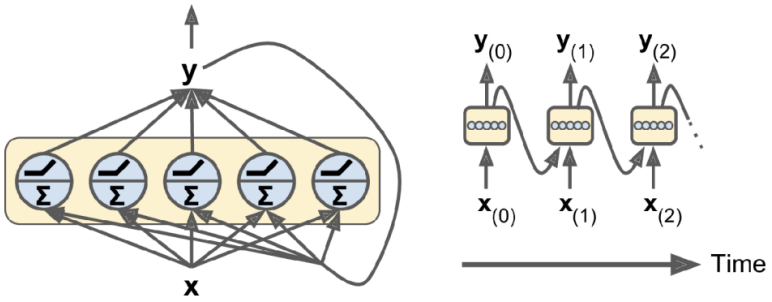


Figura 4.4: Ejemplo de una capa con neuronas recurrentes. Fuente: *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow* (p. 499), por Géron, 2022.

En la Figura 4.4 podemos ver el funcionamiento de una capa con neuronas recurrentes. A partir de ella podemos ver que la salida de una capa con neuronas recurrentes puede ser expresada matemáticamente para una instancia de entrada como

$$y_{(t)} = \phi(W_x^T x_{(t)} + W_y^T y_{(y-1)} + b)$$

donde $y_{(t)}$ es el vector salida de la neurona en la iteración t, $y_{(t-1)}$ es el vector salida de la neurona en la iteración $t-1$, $x_{(t)}$ es el vector entrada de la neurona en la iteración t, W_x^T es la matriz de pesos de las conexiones de entrada, W_y^T es la matriz de pesos de las conexiones de salida de la iteración anterior $t-1$ y b es el vector con el *bias* de cada neurona.

Como vemos las salidas de las neuronas son entradas para la mismas neuronas en la iteración siguiente esto se puede ver como si las neuronas tuvieran memoria de sus resultados pasados y eso mismo da el nombre a las neuronas *Long Short Term Memory*. En la Figura 4.5 podemos ver como la neurona funciona realmente y es capaz de retener memoria a largo plazo (salidas de las iteraciones anteriores) y corto plazo (salidas de las iteraciones más recientes).

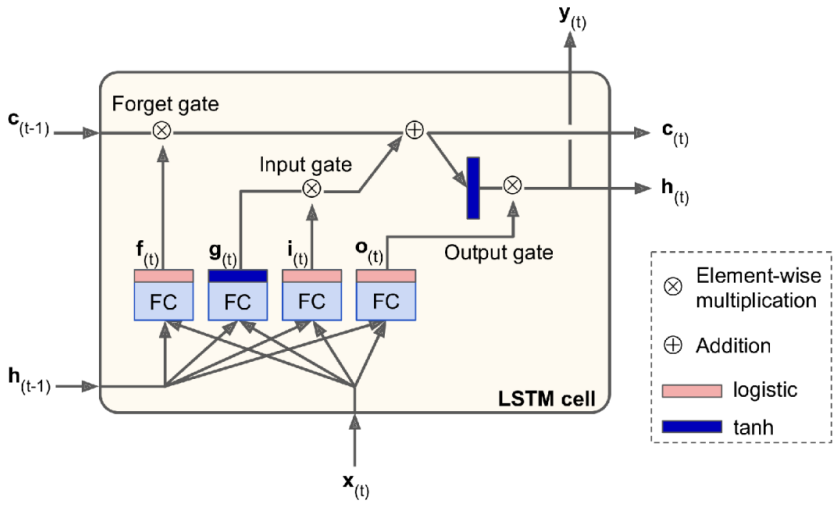


Figura 4.5: Ejemplo de una neurona LSTM. Fuente: *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow* (p. 516), por Géron, 2022.

En la Figura 4.5 podemos ver que la neurona tiene 3 vectores de entrada y 3 vectores de salida. Donde $x_{(t)}$ es el vector de entradas, $y_{(t)}$ es el vector de salida, $h_{(t-1)}$ es el vector entrada de estado a corto plazo, $h_{(t)}$ es el vector salida de estado a corto plazo, $c_{(t-1)}$ es el vector entrada de estado a largo plazo y $c_{(t)}$ es el vector salida de estado a largo plazo. Veamos que transformaciones tienen cada una de las entradas.

El estado a largo plazo c_{t-1}, pasa por una puerta *forget gate* lo cual eliminará parte de la memoria acumulada y posteriormente añadirá nueva memoria con la operación de adición $\oplus$

con los datos de los vectores de estado a corto plazo y de entrada transformados. Tras esto el resultado es la salida del estado a largo plazo c_t. Esta salida también se utiliza para calcular el vector estado a corto plazo.

Las entradas de los vectores x_t y $h_{(t-1)}$ se pasan como entrada a 4 capas densamente conectadas. Veamos la función capa por cada:

- La capa con la función de activación (logistica) $f_{(t)}$ (*forget gate*) controla que parte del estado a largo plazo ha de ser borrado.
- La capa con la función de activación (tanh) $g_{(t)}$ es la principal y de su salida algunas partes se guardan en el vector estado a largo plazo y otras partes son olvidadas.
- La capa con la función de activación (logistica) $i_{(t)}$ (*input gate*) controla que parte de la salida de la función $g_{(t)}$ es añadida al estado a largo plazo.
- La capa con la función de activación (logistica)$o_{(t)}$ (*output gate*) controla que parte del estado a largo plazo ha de incluirse en la salida.

Por último, podemos expresar matemáticamente la neurona LSTM con las siguientes ecuaciones.

$$
\begin{aligned}
i_{(t)} &= \sigma(W_{xi}^T x_{(t)} + W_{hi}^T h_{(t-1)} + b_i) \\
f_{(t)} &= \sigma(W_{xf}^T x_{(t)} + W_{hf}^T h_{(t-1)} + b_f) \\
o_{(t)} &= \sigma(W_{xo}^T x_{(t)} + W_{ho}^T h_{(t-1)} + b_o) \\
g_{(t)} &= \tanh(W_{xg}^T x_{(t)} + W_{hg}^T h_{(t-1)} + b_g) \\
c_{(t)} &= f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)} \\
y_{(t)} &= h_{(t)} = o_{(t)} \otimes \tanh(c_{(t)})
\end{aligned}
$$

Donde W_{xi}, W_{xf}, W_{xo}, W_{xg} son las matrices de pesos para cada conexión de las 4 capas para el vector de entrada x_t, W_{hi}, W_{hf}, W_{ho}, W_{hg} son las matrices de pesos para cada conexión de las 4 capas para el vector de estado a corto plazo h_{t-1} y b_i, b_f, b_o, b_g son los términos *bias* para las 4 capas.

4.3.4. CNN-LSTM

La red neuronal CNN-LSTM hace referencia a combinar capas de neuronas *Convolutional Neural Network* CNN y neuronas *Long Short Term Memory* LSTM. Supongamos que la red neuronal tiene dos capas ocultas, la primera siendo una capa con neuronas CNN que nos permite captar patrones en los datos al tratar la serie temporal como si fueran imágenes y pasar su resultado a una capa de neuronas LSTM que nos permite tener todas las ventajas de las RNN

a la hora de trabajar con series temporales. Las neuronas LSTM ya han sido explicadas en el modelo anterior, veamos ahora como funcionan las neuronas CNN.

Las neuronas CNN se componen:

- Entrada, suele ser los píxeles de una imagen y la profundidad de la imagen.
- Capa de convolución (*convolutional layer*), las neuronas en esta capa no están conectadas a todos los píxeles de la entrada sino a un subconjunto pequeño y tratan de encontrar características que pasarán como salida.
- Filtros (*filters*), los filtros están compuestos de *kernels* que son matrices que utilizaremos para realizar multiplicaciones escalares con la entrada para obtener una nueva ïmagen"filtrada por el kernel. Por cada filtro que utilicemos obtendremos una matriz como salida la cual se llama *feature mapping*.
- *Pooling layers*, esta capa se utiliza para reducir el tamaño de las matrices (*feature mappings*) existen varias técnicas pero la más utiliza se llama *Max-Pooling* que consiste en seleccionar el valor máximo de una matriz, es decir, si tenemos una matriz 9x9 y por bloques 3x3 seleccionamos el mayor valor 1 reducimos considerable el tamaño de la matriz. Al tomar el valor máximo intentamos retener las características más importantes que cada *kernel* haya detectado.
- *Flatten layers*, esta capa se utiliza para transformar las matrices resultantes en vectores para que puedan ser la entrada de neuronas MLP o LSTM.

En la Figura 4.6 podemos ver un ejemplo de como sería una neurona CNN.

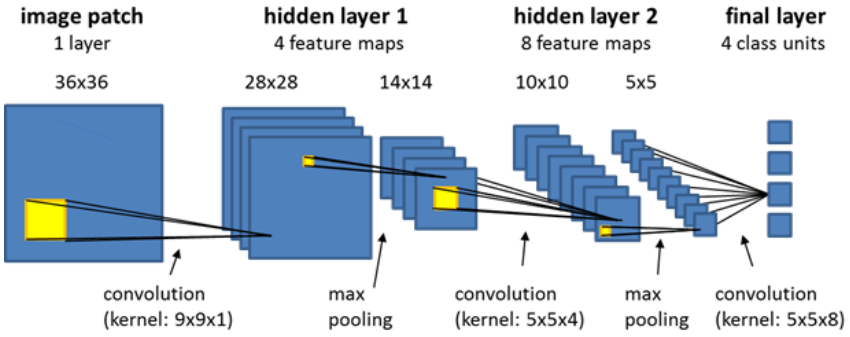


Figura 4.6: Ejemplo de una neurona CNN. Fuente: *Convolutional Neural Network Algorithms*, 2019 `https://docs.ecognition.com/v9.5.0/eCognition_documentation/Reference%20Book/23%20Convolutional%20Neural%20Network%20Algorithms/Convolutional%20Neural%20Network%20Algorithms.htm`

4.3.5. ConvLSTM

Las neuronas ConvLSTM son una combinación de las neuronas *Convolutional Neural Network* CNN y las neuronas *Long Short Term Memory* LSTM. La neurona ConvLSTM es recurrente como LSTM pero nos permite utilizar datos en 3 dimensiones, es decir, imágenes (en nuestro caso formaremos una ïmagen"(matriz) combinando varias secuencias de valores de la serie temporal).

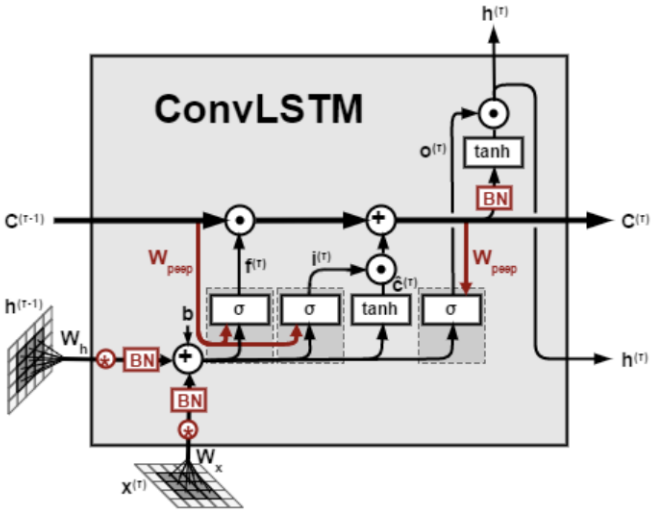


Figura 4.7: Ejemplo de una neurona LSTM. Fuente: *An introduction to ConvLSTM*, por Xavier, 2021, `https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7`.

En la Figura 4.7 podemos ver como la estructura y funcionamiento de la neurona ConvLSTM es similar a la neurona LSTM, algunas diferencia entre ambas neuronas son el tipo de entrada en ConvLSTM son tensores tridimensionales, las entradas como en la neurona CNN pasa por una capa de convolución y las operaciones entre los tensores. Podemos expresar matemáticamente el funcionamiento de la neurona con las siguientes ecuaciones.

$$
\begin{aligned}
i_t &= \sigma(W_{xi}X_t + W_{hi}H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}X_t + W_{hf}H_{t-1} + W_{cf} \circ C_{t-1} + b_f) \\
C_t &= f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}H_{t-1} + b_o) \\
o_t &= \sigma(W_{xo}X_{(t)} + W_{ho}H_{(t-1)} + W_{co} \circ C_t + b_o) \\
H_t &= o_t \circ \tanh(C_t)
\end{aligned}
$$

Donde X, C, H son tensores tridimensionales, W son los tensores de peso de las conexiones de los tres tensores de entrada a las capas de la neurona, b es el *bias* de cada capa, $\circ$ es el producto Hadamart (multiplicación de matrices elemento a elemento).

Capítulo 5

Resultados

5.1. Introducción

Hemos utilizado los 7 modelos descritos en el capítulo 4 para predecir las series temporales de la enfermedad COVID-19 de los 20 municipios más poblados de la provincia de Castellón. Para decidir que modelo se comporta mejor hemos utilizado *Walk-forward optimization* y para encontrar la mejor configuración de hiperparámetros posibles para cada modelo se ha realizado un (*grid search*).

Antes de pasar a mostrar los resultados veamos en que consiste el *Walk-forward optimization* y *grid search*:

- *Walk-forward optimization*, consiste en entrenar un modelo con los datos de entrenamiento y posteriormente evaluar su desempeño con los datos de evaluación, este proceso se puede repetir volviendo a entrenando los modelos y evaluando el modelo con nuevos datos de evaluación. Por lo que, para nuevas observaciones el proceso se puede repetir para optimizar los hiperparámetros de los modelos.
- *Grid search*, se trata de probar un rango de combinaciones de hiperparámetros en un modelo y quedarnos con la combinación de hiperparámetros que tenga menor error en las estimaciones. Esto es apropiado para encontrar la mejor combinación de hiperparámetros de forma automática y además se pueden obtener información de la combinación de hiperparámetros obtenida.

En los siguiente apartado 5.2 veremos cual ha sido el mejor modelo comparando los errores que han cometido al predecir las series temporales de los 20 municipios y posteriormente en los apartadas 5.3 en adelante veremos los resultados de cada método al ser aplicados sobre

un subconjunto de municipios Castellón, Alcora y Borriol. Veremos las gráficas de el valor estimado contra el real (el valor estimado es como se llama a las predicciones hechas con métodos estadísticos y no para métodos de *Machine Learning* por simplicidad llamaremos a ambos valor estimado) y las gráficas de un mapa de la provincia de Castellón[1] que tendrán el valor estimado contra el valor real en el último periodo de tiempo (aquí no se tienen en cuenta ningún componente espacial, es tan solo una visualización de los datos que podemos realizar dado que tenemos los datos a nivel municipal).

[1]Solo tienen valores en la gráfica los 20 pueblos estudiados

5.2. Mejor modelo

Los modelos estudiados han sido *Naive*, SARIMA, Holt Winter's, MLP, LSTM, CNN-LSTM y ConvLSTM. Para obtener el mejor modelo se ha sumado los errores que ha cometido cada modelo en las 20 series temporales estudiadas. Los resultados obtenidos están directamente relacionados a los datos utilizados y por ello no podremos inferir el desempeño de los modelos de este trabajo a otros.

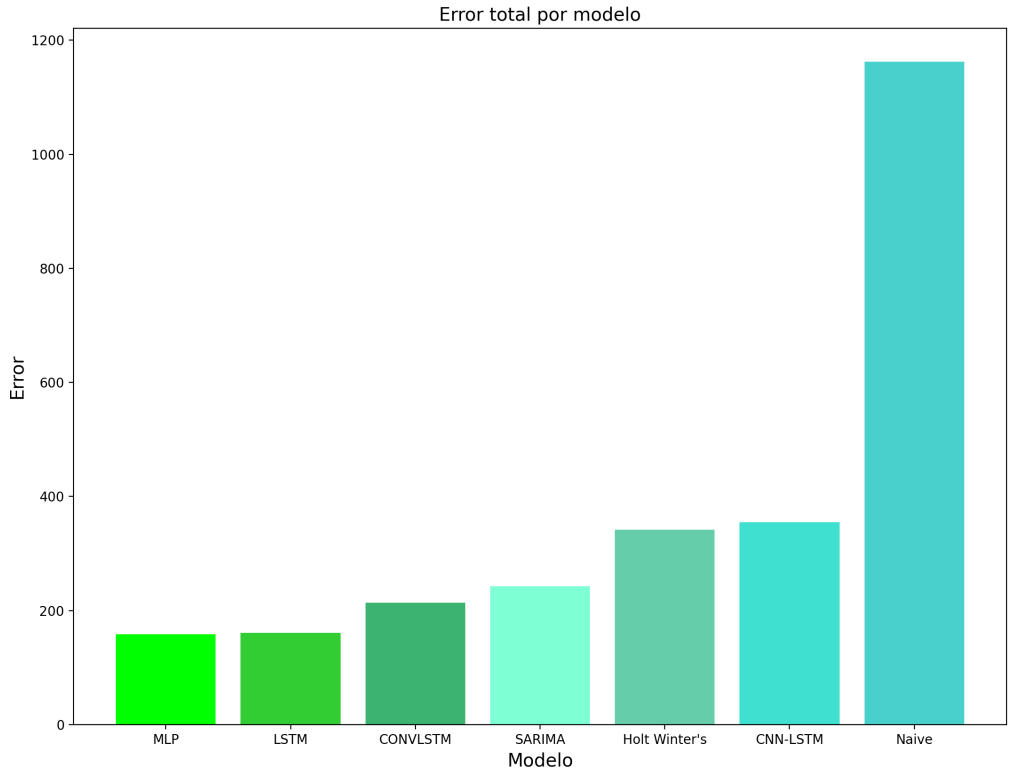


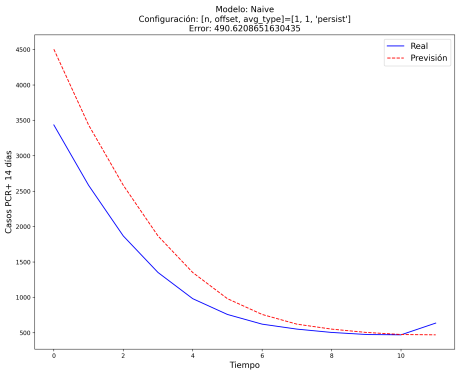
Figura 5.1: Errores cometidos al predecir nuevas observaciones

De los resultados obtenidos podemos observar:

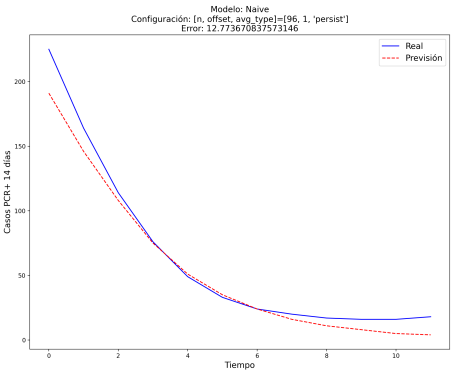
- El modelo *Naive* nos sirve como base para evaluar al resto de modelos y podemos concluir claramente que todos los modelos estudiados aportan valor a la hora de predecir la serie temporal porque reducen considerablemente el error cometido si lo comparamos con el modelo base.

- El modelo CNN-LSTM no obtiene buenos resultados en comparación a otros modelos y además se trata de un modelo más complejo.
- Ambos modelos estadísticos SARIMA y Holt Winter's obtienen buenos tienen en cuenta que son computacionalmente mucho más rápidos y obtienen errores similares a las redes neuronales.
- El modelo ConvLSTM no consigue justificar su complejidad porque no consigue mejores resultados que modelos menos complejos como MLP o LSTM pero obtiene mejores resultados que CNN-LSTM. Por lo que, podemos concluir que combinar las neuronas CNN y LSTM en una neurona da mejores resultados y que combinar capas con neuronas CNN y LSTM.
- Los modelos MLP y LSTM son los mejores modelos para nuestros datos, ambos cometen un error similar.
- Las redes neuronales menos complejas han obtenido mejores resultados que las complejas, esto podría deberse a que dada una la serie temporal univariante añadir complejidad a los modelos no esta directamente relacionado con un mejor desempeño.

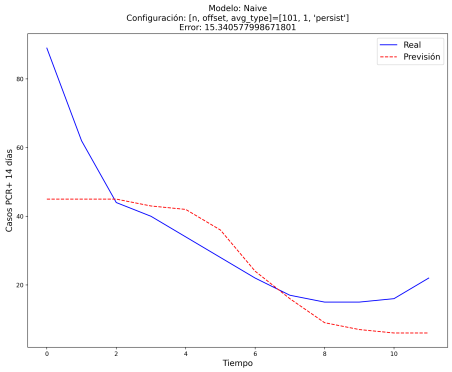
5.3. Naive



(a) Castellón

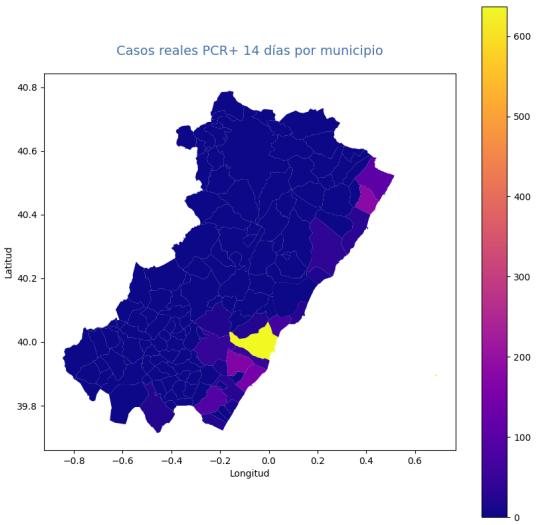


(b) Alcora

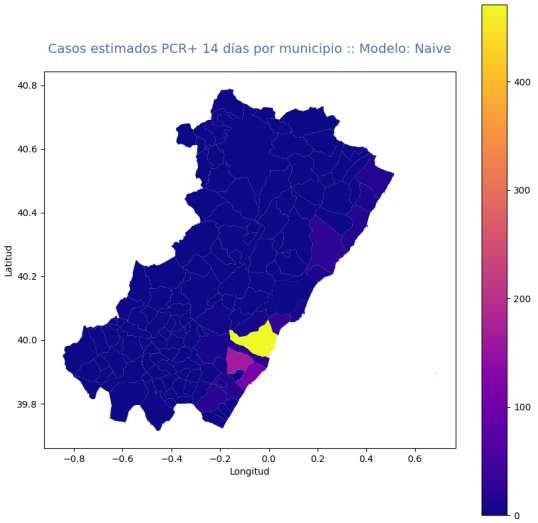


(c) Borriol

Figura 5.2: Modelo Naive Estimación vs Real subconjunto de municipios



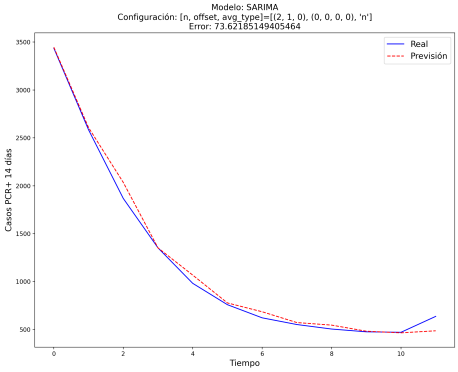
(a) Real



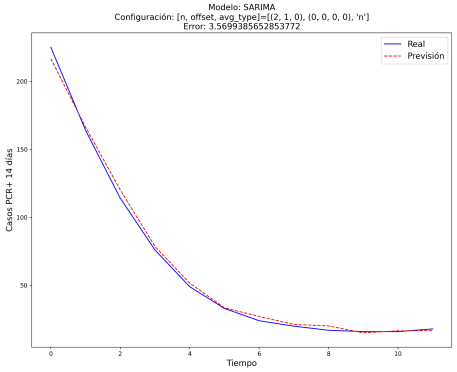
(b) Estimado

Figura 5.3: Modelo Naive Estimación vs Real

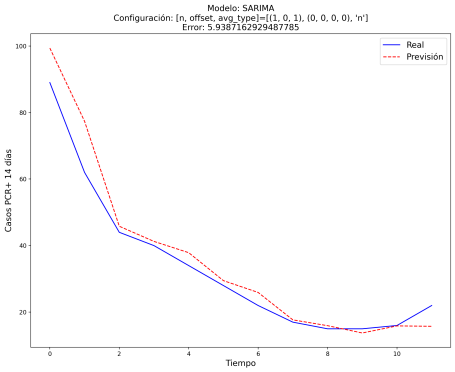
5.4. SARIMA



(a) Castellón

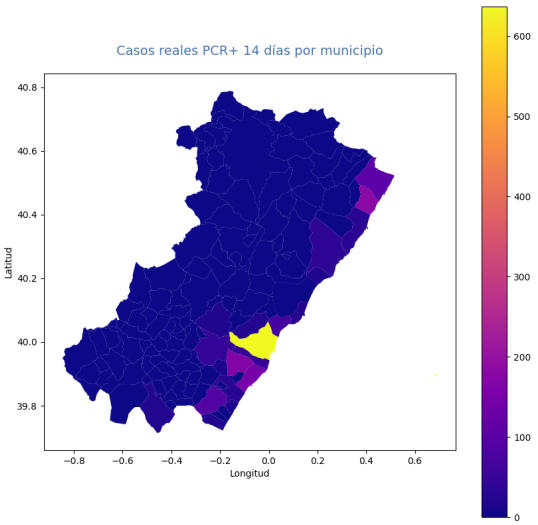


(b) Alcora

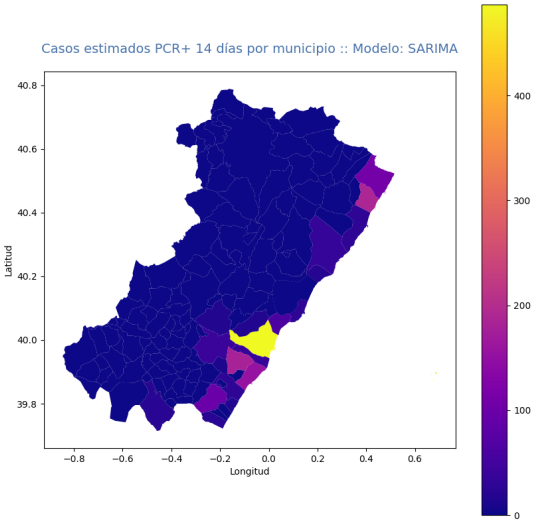


(c) Borriol

Figura 5.4: Modelo SARIMA Estimación vs Real subconjunto de municipios



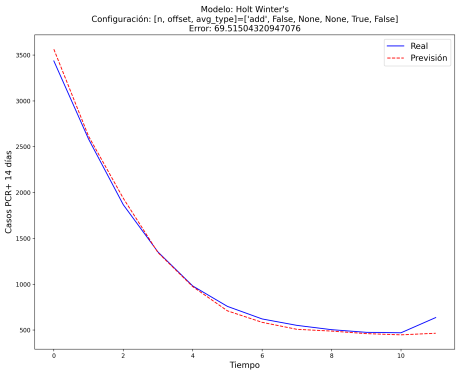
(a) Real



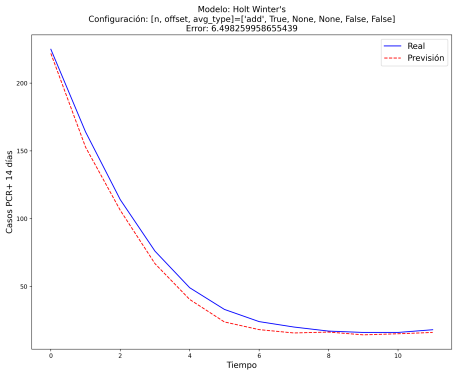
(b) Estimado

Figura 5.5: Modelo SARIMA Estimación vs Real

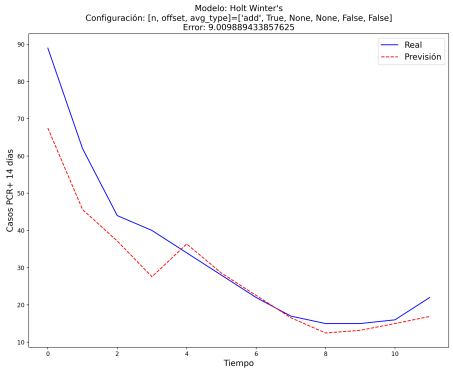
5.5. Holt Winter's



(a) Castellón

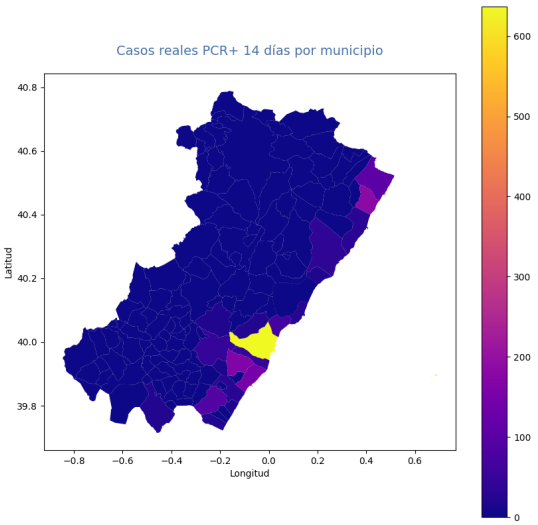


(b) Alcora

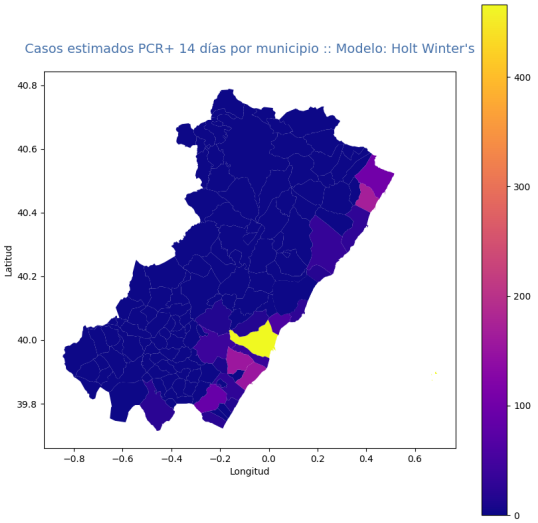


(c) Borriol

Figura 5.6: Modelo Holts Winter's Estimación vs Real subconjunto de municipios



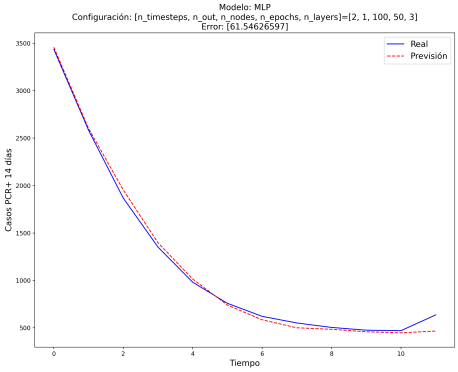
(a) Real



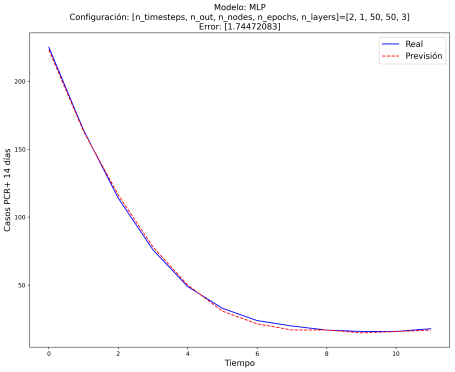
(b) Estimado

Figura 5.7: Modelo Holts Winter's Estimación vs Real

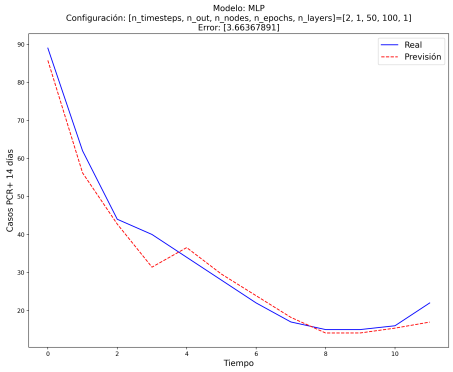
5.6. MLP



(a) Castellón

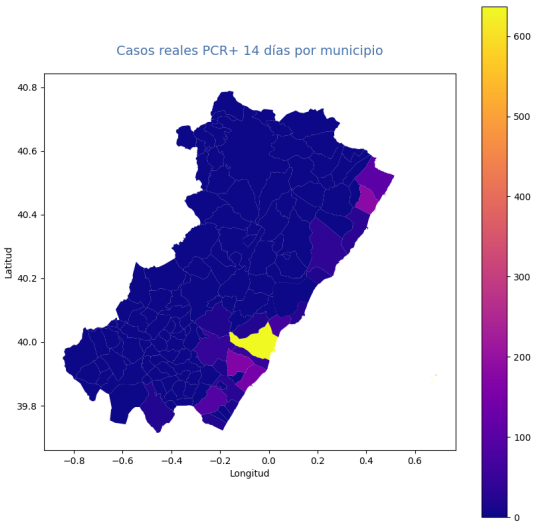


(b) Alcora

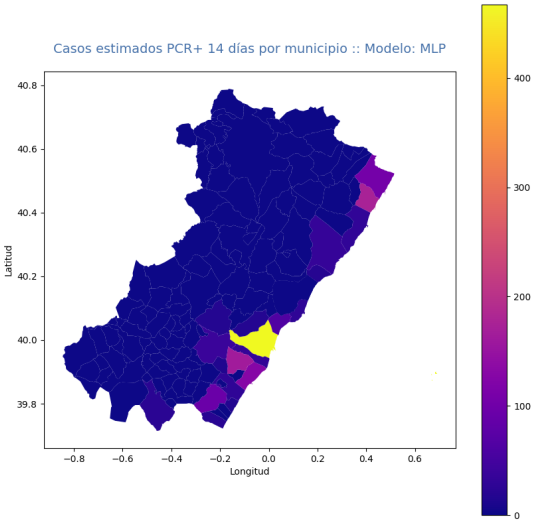


(c) Borriol

Figura 5.8: Modelo MLP Estimación vs Real subconjunto de municipios



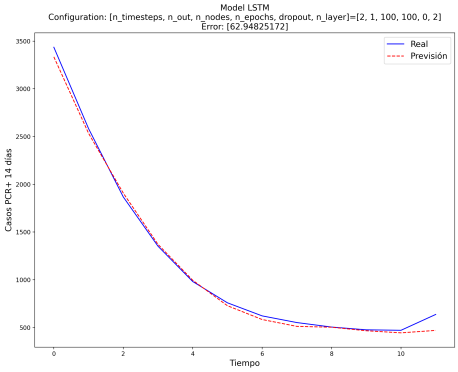
(a) Real



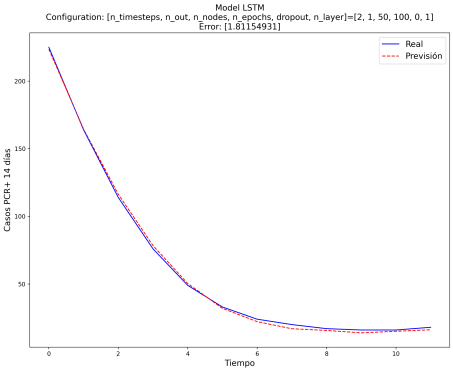
(b) Estimado

Figura 5.9: Modelo MLP Estimación vs Real

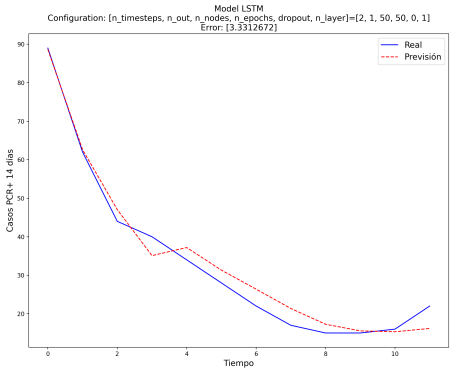
5.7. LSTM



(a) Castellón

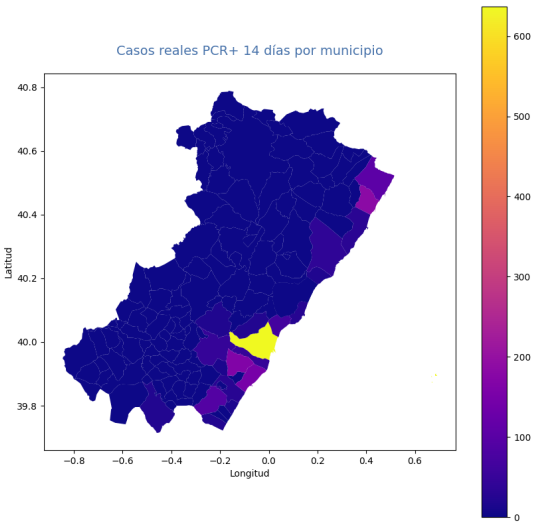


(b) Alcora

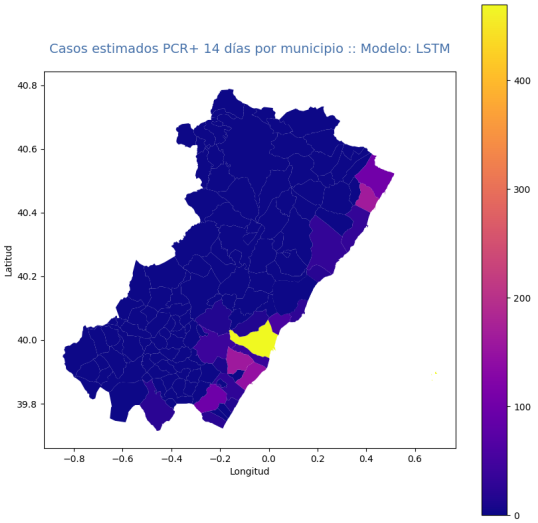


(c) Borriol

Figura 5.10: Modelo LSTM Estimación vs Real subconjunto de municipios



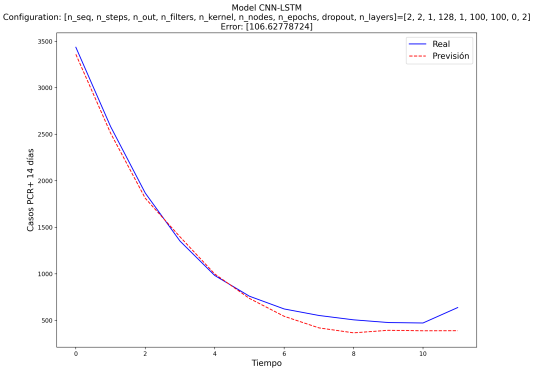
(a) Real



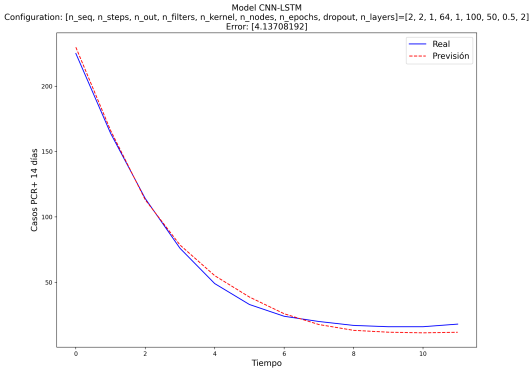
(b) Estimado

Figura 5.11: Modelo LSTM Estimación vs Real

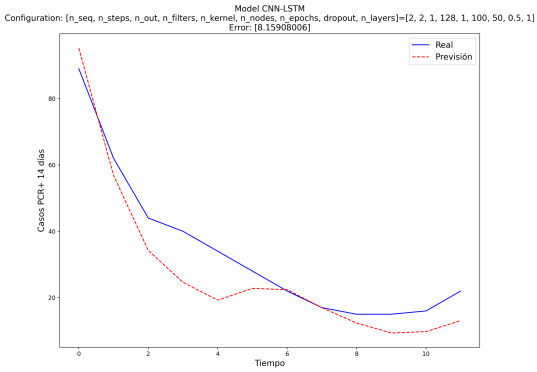
5.8. CNN-LSTM



(a) Castellón

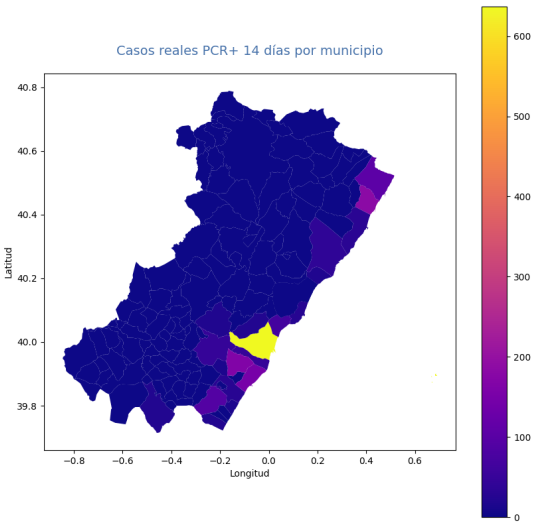


(b) Alcora

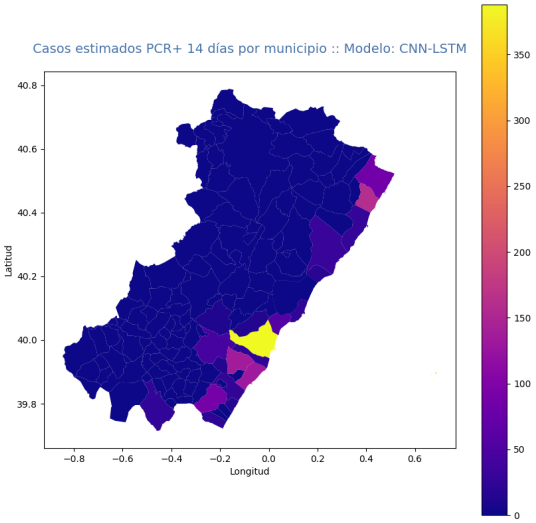


(c) Borriol

Figura 5.12: Modelo CNN-LSTM Estimación vs Real subconjunto de municipios



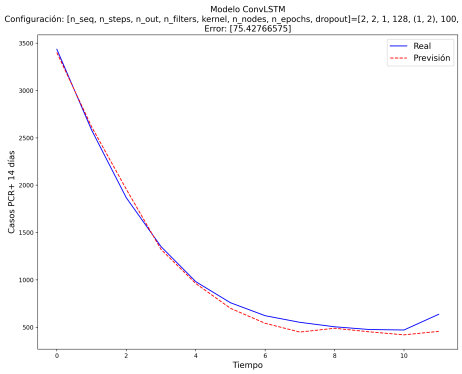
(a) Real



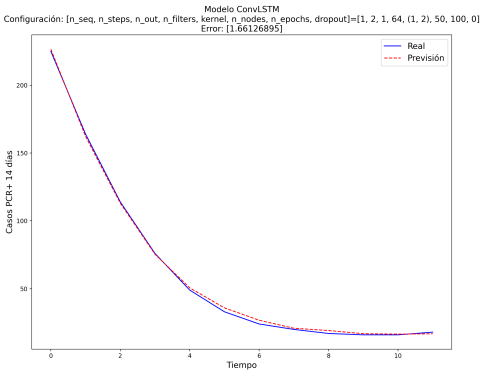
(b) Estimado

Figura 5.13: Modelo CNN-LSTM Estimación vs Real

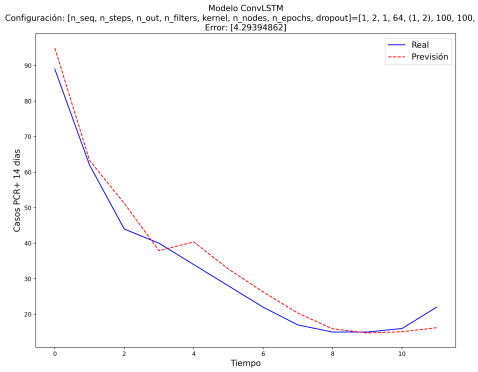
5.9. ConvLSTM



(a) Castellón

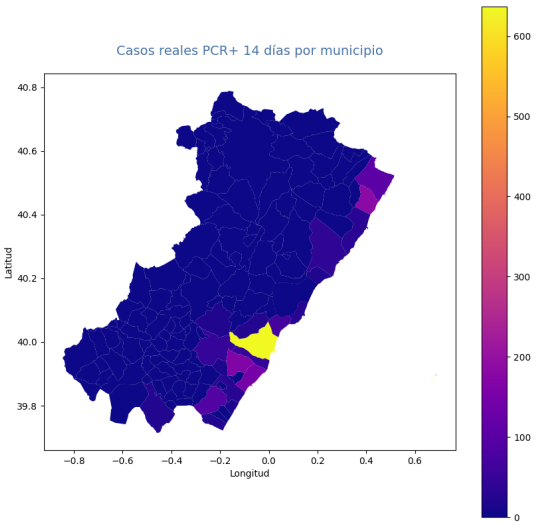


(b) Alcora

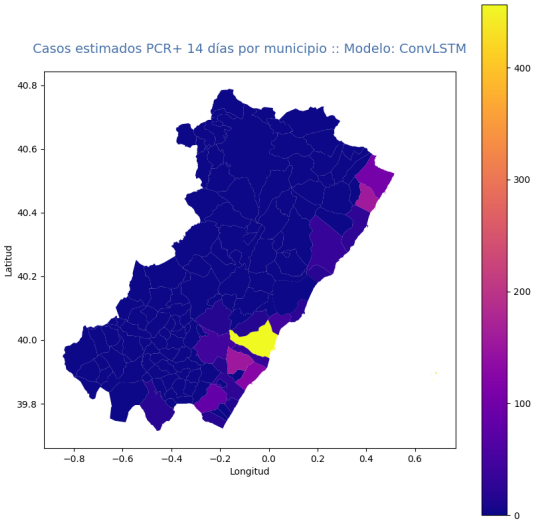


(c) Borriol

Figura 5.14: Modelo ConvLSTM Estimación vs Real subconjunto de municipios



(a) Real



(b) Estimado

Figura 5.15: Modelo ConvLSTM Estimación vs Real

Capítulo 6

Conclusiones

El primer objetivo del trabajo es comprobar que los modelos aportan valor a la hora de estimar nuevas observaciones de las series temporales comparándolos con el modelo base *naive*. Este objetivo lo cumplen todos los modelos utilizados en el trabajo reduciendo los errores cometidos considerablemente. El peor modelo es capaz de reducir los errores aproximadamente a una tercera parte y el mejor modelo es capaz de reducirlos aproximadamente hasta una séptima parte. Por lo tanto, podemos concluir que tanto los modelos estadísticos como los modelos basados en *Machine Learning* aportan valor al predecir en series temporales.

El segundo objetivo del trabajo es comparar los resultados obtenidos de los métodos estadísticos frente a los métodos basados en aprendizaje estadístico (*Statistical Learning*). Los resultados obtenidos muestran que en general los resultados obtenidos por los métodos basados en aprendizaje estadístico fueron mejores reduciendo los errores cometidos por los métodos estadísticos aproximadamente a la mitad. Por otro lado, hemos de tener en cuenta que las redes neuronales tienen un coste computacional más elevado y que si se trata de modelos que han de ser entrenados constantemente con nuevas observaciones puede que la reducción en los errores cometidos no compense el coste computacional.

Además, las redes neuronales no nos aportan *insights* sobre como realiza las predicciones pero los métodos estadísticos si lo hacen. Por ejemplo, el modelo SARIMA nos da información sobre la estacionalidad y el orden de los componentes que nos indica exactamente que valores anteriores esta teniendo en cuenta para realizar las predicciones. Teniendo en cuenta todo lo anterior podemos concluir que los mejores resultados los vamos a obtener con redes neuronales aunque si necesitamos velocidad de computación o información sobre como se realizan las estimaciones los métodos estadísticos puede ser una mejor opción.

El tercer objetivo del trabajo es averiguar que modelo tiene un mejor desempeño prediciendo

nuevos valores. Los resultados obtenidos en orden son MLP, LSTM, ConvLSTM, SARIMA, Holt Winter's, CNN-LSTM y Naive. Los modelos MLP y LSTM cometen aproximadamente los mismos errores por lo que no podemos concluir que uno es mejor que el otro aunque sí que podemos concluir que baten al resto con un margen considerable.

Adicionalmente, podemos concluir que añadir complejidad a la red neuronal no ha repercutido en un mejor desempeño del modelo sino que ha obtenido peores resultados y tiempos de entrenamiento más largos. Esto puede deberse a que los datos utilizados sean una serie temporal univariante y que las redes convoluciones no sean capaces de extraer patrones y características sobre los datos como lo harían si se tratase de una serie temporal multivariante, imágenes, etc.

Bibliografía

[1] Niraula, P., Mateu, J., & Chaudhuri, S. (2022). A Bayesian machine learning approach for spatio-temporal prediction of COVID-19 cases. Stochastic environmental research and risk assessment : research journal, 36(8), 2265–2283. https://doi.org/10.1007/s00477-021-02168-w

[2] Chollet, F. (2021, 21 diciembre). Deep Learning with Python, Second Edition. Manning Publications

[3] Brownlee, J. (2019). Deep learning for time series forecasting (v1.6). Machine Learning Mastery

[4] Géron, A. (2022, 15 noviembre). Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems (3rd ed.). O'Reilly Media.

[5] Brockwell, P. J. & Davis, R. A. (2016, 31 agosto). Introduction to Time Series and Forecasting (3rd 2016 ed.). Springer

[6] Peña, D. (2010). Análisis de series temporales / Time Series Analysis. Alianza Editorial SA

[7] Brownlee, J. (2017). Long short-term memory networks with Python (v1.0). Machine Learning Mastery

[8] Team, K. (s. f.). Keras documentation: Keras API reference. Recuperado 24 de septiembre de 2022, de https://keras.io/api/

[9] Library Reference. Python 3.10.7 documentation. Recuperado 24 de septiembre de 2022, de https://docs.python.org/3/library/index.html

[10] API reference — pandas 1.5.0 documentation. (s. f.). Recuperado 24 de septiembre de 2022, de https://pandas.pydata.org/docs/reference/index.html#api

[11] NumPy Reference — NumPy v1.23 Manual. (s. f.). Recuperado 24 de septiembre de 2022, de https://numpy.org/doc/stable/reference/index.html#reference

[12] Plotting commands summary — Matplotlib 2.0.2 documentation. (s. f.). Recuperado 24 de septiembre de 2022, de https://matplotlib.org/2.0.2/api/pyplot_summary.html

[13] API Reference — GeoPandas 0.11.0+0.g1977b50.documentation. (s. f.). Recuperado 24 de septiembre de 2022, de https://geopandas.org/en/stable/docs/reference.html

[14] API Reference - Statsmodel v0.13.2.documentation (s. f.). Recuperado 24 de septiembre de 2022, de https://www.statsmodels.org/stable/api.html

[15] Ariton, L. (2021, 31 diciembre). A Thorough Introduction to Holt-Winters Forecasting. Medium. Recuperado 24 de septiembre de 2022, de https://medium.com/analytics-vidhya/a-thorough-introduction-to-holt-winters-forecasting-c21810b8c0e6

[16] Artley, B. (2022, 27 junio). Time Series Forecasting with ARIMA , SARIMA and SARIMAX. Medium. Recuperado 24 de septiembre de 2022, de https://towardsdatascience.com/time-series-forecasting-with-arima-sarima-and-sarimax-ee61099e78f6

[17] Sosna, M. (2022, 6 enero). A Deep Dive on ARIMA Models - Towards Data Science. Medium. Recuperado 24 de septiembre de 2022, de https://towardsdatascience.com/a-deep-dive-on-arima-models-8900c199ccf

[18] Understanding of Convolutional Neural Network (CNN) — Deep Learning. (2020, 5 febrero). Medium. Recuperado 24 de septiembre de 2022, de https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

[19] Kang, E. (2018, 18 junio). Long Short-Term Memory (LSTM): Concept - Eugine Kang. Medium. Recuperado 24 de septiembre de 2022, de https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359

[20] Xavier, A. (2021, 9 diciembre). An introduction to ConvLSTM - Neuronio. Medium. Recuperado 24 de septiembre de 2022, de https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

[21] COVID Live - Coronavirus Statistics - Worldometer. (s. f.). Recuperado 25 de septiembre de 2022, de https://www.worldometers.info/coronavirus/

[22] Remuzzi, A. (2020, 11 abril). COVID-19 and Italy: what next? The Lancet. Recuperado 25 de septiembre de 2022, de https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(20)30627-9/fulltext

[23] Shetty, C. (2022, 30 marzo). Time Series Models - Towards Data Science. Medium. Recuperado 25 de septiembre de 2022, de https://towardsdatascience.com/time-series-models-d9266f8ac7b0

[24] Autoregressive Integrated Moving Average ARIMA(p, d, q) Models for Time Series Analysis — QuantStart. (s. f.). Recuperado 25 de septiembre de 2022, de https://www.quantstart.com/articles/Autoregressive-Integrated-Moving-Average-ARIMA-p-d-q-Models-for-Time-Series-Analysis/

[25] Autoregressive Moving Average ARMA(p, q) Models for Time Series Analysis - Part 1 — QuantStart. (s. f.). Recuperado 25 de septiembre de 2022, de https://www.quantstart.com/articles/Autoregressive-Moving-Average-ARMA-p-q-Models-for-Time-Series-Analysis-Part-1/

[26] Autoregressive Moving Average ARMA(p, q) Models for Time Series Analysis - Part 2 — QuantStart. (s. f.). Recuperado 25 de septiembre de 2022, de https://www.quantstart.com/articles/Autoregressive-Moving-Average-ARMA-p-q-Models-for-Time-Series-Analysis-Part-2/

[27] Autoregressive Moving Average ARMA(p, q) Models for Time Series Analysis - Part 3 — QuantStart. (s. f.). Recuperado 25 de septiembre de 2022, de https://www.quantstart.com/articles/Autoregressive-Moving-Average-ARMA-p-q-Models-for-Time-Series-Analysis-Part-3/

[28] Zhang, M. (2018, 23 octubre). Time Series: Autoregressive models AR, MA, ARMA, ARIMA. University of Pittsburg.Recuperado 25 de septiembre de 2022, de https://people.cs.pitt.edu/~milos/courses/cs3750/lectures/class16.pdf

[29] Mauricio, J. A. (2007, marzo). Análisis de Series Temporales. Universidad Complutense de Madrid. Recuperado 25 de septiembre de 2022, de https://www.ucm.es/data/cont/docs/518-2013-11-11-JAM-IAST-Libro.pdf

[30] Alonso, A. M. (s. f.). 6. Seasonal ARIMA processes. Universidad Carlos III de Madrid. Recuperado 25 de septiembre de 2022, de http://halweb.uc3m.es/esp/Personal/personas/amalonso/esp/TSAtema6.pdf

[31] Baysan, M. (2022, 5 marzo). Introduction to Time Series Forecasting: Smoothing Methods. Medium. Recuperado 25 de septiembre de 2022, de https://medium.com/codex/introduction-to-time-series-forecasting-smoothing-methods-9a904c00d0fd

[32] Xavier, A. (2021b, diciembre 9). An introduction to ConvLSTM - Neuronio. Medium. Recuperado 25 de septiembre de 2022, de https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7

[33] Convolutional Neural Network Algorithms. (s. f.). Recuperado 25 de septiembre de 2022, de https://docs.ecognition.com/v9.5.0/eCognition_documentation/Reference+Book/23+Convolutional+Neural+Network+Algorithms/Convolutional+Neural+Network+Algorithms.htm

[34] Shi, X. (2015, 13 junio). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. arXiv.org. Recuperado 25 de septiembre de 2022, de https://arxiv.org/abs/1506.04214

[35] Google Trends. (s. f.). Recuperado 9 de octubre de 2022, de https://trends.google.es/trends/explore?date=all&q=machine%20learning

[36] Alameda, T. (2022, 4 marzo). «Machine learning»: ¿qué es y cómo funciona? BBVA NOTICIAS. Recuperado 9 de octubre de 2022, de https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/