# APRENDIZAJE AUTOMÁTICO

WAVE ENERGY CONVERTERS

Pedro Gallego López

David Villar Martos

*Universidad de Granada*
*Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones*

28 de junio de 2021

# Índice general

# Wave Energy Converters

## 1.1. Problema asociado y Data Set


**Figura 1.1:** Unidad CETO

Nuestro problema gira entorno a la obtención de energía renovables a través de las olas. El medio marino es una fuente de energía poco explotada y para la que se han ideado múltiples dispositivos para la obtención de energía renovable del mismo. Nuestro problema en concreto usa unos dispositivos WECs (Wave Energy Converters) denominados **CETO**.

Los **CETO** son unos dispositivos situados en el fondo marino que están sujetos a una bomba con la que son capaces de transmitir, a través de unas tuberías, agua a gran presión hasta una central donde se convertirá está agua a presión en energía eléctrica gracias a una turbina. En la Figura1.1 podemos ver una imagen en donde se puede ver la forma de este dispositivo en el fondo marino.

### 1.1.1. Data Set

Vamos a tratar con una base de datos que podemos encontrar en `https://archive.ics.uci.edu/ml/datasets/Wave+Energy+Converters`. Analizaremos el problema correspondiente a la base de datos y conforme a ello aplicaremos técnicas de Aprendizaje Automático para resolverlo. La base de datos contiene información sobre la posición y potencia absorbida por los WECs en cuatro escenarios distintos de la costa australiana: Sydney, Adelaide, Perth y Tasmania. Los WECs serán dispositivos CETO y habrá 16 colocados en un área restringida: $566m^2$.

El conjunto de datos pretende servir para estimar un modelo que sea capaz de predecir la **potencia total (Watt)** absorbida por los 16 CETOs. La base de datos cuenta con 288.000 instancias, donde 72.000 aproximadamente son correspondientes a cada una de las zonas examinadas (Sydney, Adelaide, Perth y Tasmania).

Los atributos que trae de serie cada una de las bases de datos aportadas son: las posiciones de los CETOs en un área de $566m^2$ en el fondo marino de la zona indicada por la base de datos, las potencias absorbidas por cada CETO y la potencia total absorbida entre todos los CETOs, que corresponde con la suma de las potencias individuales. Siendo $(x_i, y_i)$ las coordenadas de posicionamiento del CETO $i - esimo$ y $P_i$ su potencia, las características están ordenadas de

la siguiente forma:

$$(x_1, ..., x_{16}, y_1, ..., y_{16}, P_1, ..., P_{16}, P_T)$$

Donde $P_T = \sum_{i=1}^{16} P_i$ es la potencia total absorbida.

Para nuestro problema de predicción de la **potencia total (Watts)** eliminaremos las potencias $P_i, i \in \{1, ..., 16\}$ para estimar directamente $P_T$. A la hora de predecir la potencia real de una distribución de CETOs, sólo tendríamos sus posiciones. Así, dispondremos de 32 características (16 pares de coordenadas) y de un valor a estimar: **potencia total (Watt)**.

### 1.1.2. Comprensión del problema

Los datos de nuestro problema serán unos entre los datos de: Sydney, Adelaide, Perth y Tasmania. Para resolver nuestro problema no podremos coger todos los conjuntos de datos y unirlos en uno sólo, puesto que en realidad los datos no están idénticamente distribuidos. Indudablemente, las regiones cuadradas de 566 × 566 m no se encuentran en la misma localización submarina. Puede haber muchos factores que puedan afectar a la potencia total generada en cada una de estas localizaciones: profundidas y accidentes geográficos del fondo marino, corrientes marinas en la zona, fauna marina... bajo este fuete indicio, decidimos comprobar las medias de las potencias generadas por cada configuración de CETOs recogidas en cada instancia de cada conjunto de datos, y observamos lo siguiente:

| | *Sydney* | *Adelaide* | *Perth* | *Tasmania* |
|---|---|---|---|---|
| $\overline{P}_T$ | 1.486.224 | 1.410.060 | 1.394.507 | 3.760.088 |

**Cuadro 1.1:** Media de las Potencias Totales $P_T$ en los conjuntos de entrenamiento de cada zona

En el Cuadro 1.1 podemos ver que las diferencias entre las medias de las potencias totales (valor a predecir) que existe entre los conjuntos de datos en función de la zona corroboran nuestra hipótesis. Por tanto, asumiremos que los datos de cada conjunto no siguen una misma distribución y por lo tanto no podemos considerar un único problema de aprendizaje para el conjunto de los 288.000 datos. Debido a ello, utilizaremos para nuestro problema únicamente el conjunto de datos de Sydney, y por lo tanto los resultados obtenidos solo serán aplicables a esta zona.

Habiendo entendido el problema, ahora pasamos a traducirlo en notación de un problema de Aprendizaje Automático. Nuestro espacio de características será $\mathcal{X} = [0, 566]^{32}$ que representan los 16 pares $(x, y)$ para la posición de cada CETO. Nuestra característica a predecir es $\mathcal{Y} = \mathbb{R}$ que hace referencia a la *potencia total en Vatios*.

El objetivo es por tanto conseguir un modelo que sea capaz de encontrar una hipótesis $g$ que se aproxime lo máximo posible a $f : \mathcal{X} \to \mathcal{Y}$ que es la función verdadera que asigna un valor de $y \in \mathcal{Y}$ a cada $x \in \mathcal{X}$.

## 1.2. Modelos y Clases de funciones a utilizar

De cara a encontrar una buena aproximación de la de la función real $f$, vamos a utilizar diferentes clases de funciones que aporten diferentes enfoques. En principio desconocemos la naturaleza de dicha función $f$, por lo que tiene sentido probar con diferentes clases ya que unas pueden proporcionar intrínsecamente mejores aproximaciones puesto que tengan un menor nivel de sesgo.

Vamos a utilizar 6 clases de funciones diferentes, las cuales están asociadas a los modelos: Regresión Lineal, Perceptrón Multicapa, Máquina de Soporte de Vectores (SVM), Boosting, Random Forest y Redes de Funciones de base Radial. Explicaremos cada uno con más detalle a continuación.

### 1.2.1. Clases lineales

La clase de funciones lineales supone una clase de funciones muy simple y con una interpretabilidad natural. Debido a estos factores, si un modelo lineal supone una buena aproximación de la función real $f$, será preferible frente a cualquier otra que posea una bondad equiparable.

Si poseemos una serie de atributos $x = (1, x_1, ..., x_n)$, la clase de funciones lineales sobre dichos atributos se definirá como:

$$\mathcal{H}_{lin} = \left\{\alpha x^T : \alpha \in \mathbb{R}^{n+1}\right\} \tag{1.1}$$

Evidentemente podemos identificar la clase de funciones $\mathcal{H}_{lin} \cong \mathbb{R}^{n+1}$, con lo que podremos hablar indistintamente tanto de funciones lineales como de vectores en $\mathbb{R}^{n+1}$ o equivalentemente hiperplanos en el mismo espacio.

Esta clase de funciones tiene, sin embargo, claras limitaciones. Resultaría bastante presuntuoso suponer que el fenómeno corresponde a una simple separación lineal en base a las características consideradas de los datos, y es bastante probable que el error por sesgo de estos modelos sea bastante grande. Por lo tanto, surge el interés de utilizar otros métodos más sofisticados que a priori sean capaces de ajustarse mejor a la verdadera función $f$.

### 1.2.2. Clases no lineales

Como hemos dicho en la introducción de esta sección, el resto de las clases que utilizaremos se corresponden a los modelos: Perceptrón Multicapa, Máquina de Soporte de Vectores (SVM), Boosting, Random Forest y Redes de Funciones de Base Radial. A continuación se explica cada una.

#### Perceptrón Multicapa

El Perceptrón Multicapa supone un intento de mejora directo sobre los modelos lineales, utilizando una aplicación consecutiva de funciones lineales y no lineales sobre las salidas de las anteriores, mediante una estructura de capas, incrementando así el tamaño de la clase de funciones. La definición general de la clase de funciones definida por el Perceptrón Multicapa sobre un conjunto de atributos $x = (1, x_1, ..., x_n)$ es:

$$\mathcal{H}_{MLP} = \left\{W_L \cdot \theta(W_{L-1} \cdot \theta(..\theta(W_1 \cdot x^T)..)\right\} : W_i \in \mathcal{M}_{n_i,m_i} : m_i = n_{i-1} + 1, i \in \{2, .., L\}\Big\} \tag{1.2}$$

Donde $L \in \mathbb{N}$ y la función $\theta$ será una función no lineal, y tras realizar el producto por cada vector de pesos, se añade a las variables un nuevo término constante 1. En nuestro caso usaremos $L = 3$, y además ocurrirá también que $n_3 = 1$.

El motivo por el que se ha elegido esta clase de funciones es por su gran versatilidad y gran capacidad de adaptación. Es una técnica que promete llegar a muy buenos resultados con una buena definición de la estructura del Perceptrón Multicapa.

Sin embargo, el principal problema de estas funciones es que su potencial de adaptación depende de una forma muy fuerte a la arquitectura empleada (valores $n_i : i \in \{1, ..., L\}$ y $L$

escogidos), y no tenemos un criterio claramente definido con el cual fijarlas para cada problema. Además son modelos que tienden al sobreajuste muy rápidamente en base a estos parámetros y otros relativos al aprendizaje. Por tanto será interesante coger también funciones que no dependan de una arquitectura que nosotros prefijemos.

### Máquinas de soporte de vectores (SVM)

En problemas de regresión como el nuestro utilizaremos la variante de Support Vector Regressor de Scikit Learn utilizada para regresión. De la misma forma que para el problema de clasificación teníamos una especie de pasillo que definía la máxima separación entre las clases, ahora tenemos un pasillo cuyo objetivo es ser lo más fino posible a la vez que contenga el máximo número de puntos posible (Figura 1.2).

Se ha decidido emplear este modelo por las múltiples ventajas que nos ofrece. Es un modelo que es efectivo en espacios de alta dimensionalidad. Es eficiente en términos de que utiliza únicamente un subconjunto de los puntos de entrenamiento para la función, llamados *vectores de soporte*. Tiene una gran versatilidad gracias al uso de kernels. Todo esto hace que sea un modelo a tener en cuenta para nuestro problema.

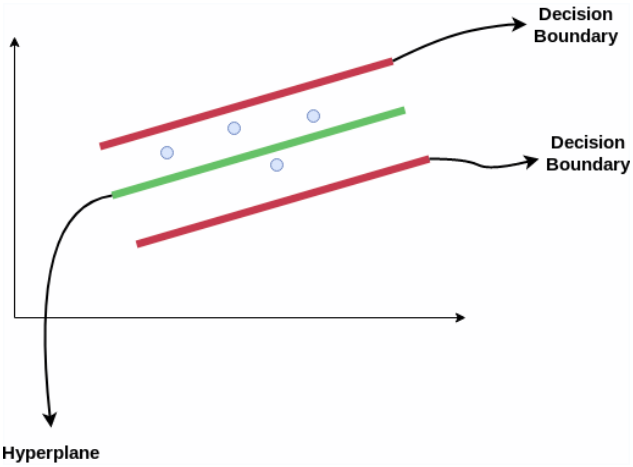



**Figura 1.2:** Support Vector Regressor. En la imagen se puede ver en ROJO la superficie/curva que delimita el pasillo, en VERDE nuestro hiperplano predictor. El objetivo es que el pasillo contenga el máximo de puntos posibles siendo todo lo estrecho que se pueda.

Para el problema utilizaremos un kernel de generación de características cúbicas. Esto nos permite aumentar inicialmente la dimensionalidad de los datos de una forma considerable, pero sin hacerlo demasiado siendo consecuentes con los recursos computacionales que tenemos disponibles, estableciendo un compromiso. Por lo tanto, realmente la clase inicial de funciones que podremos aprender será una clase de funciones cúbicas respecto al vector de características, con lo que, notando

- $x = \left(1, x_1, ..., x_n, x_1^2, ...x_n^2, x_1x_2, ..., x_1x_n, x_2x_3, ...x_{n-1}x_n, x_1^3, ...x_n^3, x_1^2x_2, ..., x_{n-1}x_n^2\right)$

nuestra clase de funciones será:

$$\mathcal{H}_{cub} = \left\{\alpha^T x : \alpha \in \mathbb{R}^{1+3n+\frac{3n(n-1)}{2}+\frac{n(n-1)(n-2)}{3}}\right\} \tag{1.3}$$

Esta clase de funciones nos puede dar el regresor óptimo respecto a las características cúbicas respecto de las iniciales empleadas. Sin embargo, puede que esta aproximación sea excesivamente compleja y poco interpretable, y existan otras aproximaciones o paradigmas de clases de funciones y modelos de aprendizaje mucho más simples, interpretables y con un sesgo menor para la verdadera función $f$, la cual no tiene por qué tener una expresión analítica sencilla respecto a ningún conjunto de características polinómicas de las de partida.

### Boosting

Un enfoque distinto a lo que hemos estado considerando hasta ahora es el del ensemble learning. El ensemble learning sostiene entrenar varios modelos y utilizar como valor predicho

uno que se calcule utilizando los valores individuales devueltos por cada modelo.

Boosting se basa en esta idea, haciendo una combinación adaptativa de modelos simples (distintos entre sí) para dar lugar a un buen regresor. Utilizaremos concretamente el modelo AdaBoost, visto en teoría.

La principal característica de AdaBoost es que ajusta una secuencia de modelos de aprendizaje débiles (modelos que mejoran ligeramente las suposiciones aleatorias, como los árboles de decisión) en versiones de los datos modificados repetidamente. Las predicciones de todos ellos se combinan mediante un voto mayoritario ponderado, es decir, sumando, para producir la predicción final.

Los regresores simples que utilicemos vamos a considerarlos dentro de una clase simple, en nuestro caso una restricción de la clase de los árboles de decisión $\mathcal{H}_{tree}$, donde limitamos de una forma muy acusada su crecimiento para se aprendan reglas muy generales y forzar la simplicidad de los modelos. Por lo tanto, nuestra clase de funciones será:

$$\mathcal{H}_{ada\_boost} = \left\{ h(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) : h_t \in \mathcal{H}_{tree} \right\} \tag{1.4}$$

Uno de los problemas que puede tener AdaBoost es que entrenar modelos muy simples de forma adaptativa pueden verse muy afectado por el ruido aleatorio que tengan los datos. Si usáramos modelos más complejos, el sobreajuste podría aumentar muy rápidamente. Es directamente imposible desde nuestra posición medir estos errores de naturaleza aleatoria, con lo que surge el interés de probar otros ensembles que utilicen modelos más complejos y que sean más robustos ante este tipo de ruido.

**Random Forest**

Los RandomForest también son ensembles basados en árboles de decisión. Sin embargo se usan otro tipo de técnicas distintas a las usadas en AdaBoost. La estructura de los árboles no se limita tan acusadamente como ocurre con AdaBoosting, pero cada uno de los árboles entrenados se entrena sólo con un subconjunto de datos muestreados con reemplazamiento sobre la muestra original (bootstrap). La predicción final, en nuestro caso de regresión, se realiza promediando los valores predichos por cada árbol. Surgieron como forma de disminuir la varianza de la clase de funciones de los árboles de decisión, aunque se pierda en interpretabilidad del modelo obtenido.

La clase de funciones de los Random Forest es la siguiente:

$$\mathcal{H}_{RF} = \left\{ \frac{1}{N} \sum_{i=0}^{N} \sum_{j=0}^{M} \alpha_{i,j} \mathbb{1}_{\mathcal{X}_{i,j}}(x) : \alpha_{i,j}, \mathcal{X}_{i,j} \; cumplen \; \mathbf{C}^* \right\} \tag{1.5}$$

Condiciones $\mathbf{C}^*$:

- $\alpha_{i,j} \in \mathbb{R}$
- $\mathcal{X}_{i,j} = (a_{i,j}, b_{i,j}) \times (c_{i,j}, d_{i,j}) : a_{i,j} < b_{i,j}, c_{i,j} < d_{i,j} \forall i \in \{1, ...N\}, j \in \{1, ...M\}$
- Los conjuntos $\mathcal{X}_{i,j}$ forman una partición de $X$
  - $\forall i \in \{1, ...N\} \cup_{j=0}^{M} \mathcal{X}_{i,j} = X$
  - $\mathcal{X}_{i,j} \cap \mathcal{X}_{i,l} = \emptyset, \forall j, l \in \{1, ..., M\}, i \in \{1, ..., N\} : j \neq l\}$

Esta clase de funciones tiene en general una muy buena capacidad de adaptación y reporta muy buenos resultados. Con una cantidad suficiente de árboles se puede dar una predicción bastante precisa, y puesto que se devuelve un valor promediando los resultados devueltos por los árboles entrenados con subconjuntos de características e instancias no es tan propenso a sobreajustar como otros modelos. Es por ello por lo que lo consideraremos y lo incluiremos entre nuestros modelos a utilizar.

**Funciones de Base Radial**

Las Funciones de Base Radial (RBF) suponen otro paradigma de clases de funciones, basadas en el concepto del aprendizaje por semejanza, pero que de igual manera están muy estrechamente relacionadas con los modelos lineales. En nuestro caso vamos a considerar RBF basadas en un núcleo gaussiano:

$$\phi(x) = e^{\frac{-1}{2}z^2}$$

La idea es fijar una serie de centros $\mu_j$ en los datos de entrada y considerarlos como referencia para aplicar el núcleo utilizando la distancia de cada punto a los centros $\mathbf{v}_j$, reescalada por un parámetro $r_j$. La clase de estas funciones es:

$$\mathcal{H}_{RBF} = \left\{ \sum_{j=1}^{k} w_j \phi\left( \frac{||x - \mathbf{v}_j||}{r_j} \right) : w_i, r_i \in \mathbb{R}, \mathbf{v}_j \in \mathbb{R}^d, i \in \{0, ..., k\} \right\} \tag{1.6}$$

Como los centros son un parámetro del modelo y estos aparecen dentro de una función no lineal, este modelo es no lineal. Este modelo no está implementado en *Scikit-Learn* por lo que se ha tenido que crear una clase

`RBFNetworkRegressor`

basada en el método dado por el libro de [1] en la sección de '*Radial Basis Function Networks*'. El algoritmo es el siguiente:

**Algorithm 1:** Fit::RBFNetworkRegressor

**Parámetros: 1:** Conjunto de datos $X$, **2:** conjunto de etiquetas $y$,
**3:** número de centros $k$, **4:** amplitud radio $\gamma \in [1, 3]$

1 $\mathbf{v} = \{\mathbf{v}_0, ..., \mathbf{v}_{k-1}\} = k\_medias(X, k)$
2 $r_i = \gamma \min_{j \neq i}\{||\mathbf{v}_i - \mathbf{v}_j||_2\}, \qquad i \in \{1, ..., k-1\}$
3 $r = \{r_0, ..., r_{k-1}\}$
4 $Z = KernelRBF\left(\frac{d(X, \mathbf{v})}{r}\right)$
5 $Ridge.fit(Z, y)$

En la referencia de [1] se escoge un radio $r_j$ para cada centro $\mathbf{v}_j$. Donde

$$r_j = \gamma \min_{k \neq j} ||\mathbf{v}_k - \mathbf{v}_j||_2, \ \gamma \in [1, 3]$$

Por último, el modelo lineal escogido ha sido el de Ridge (regresión lineal con regularización L2) ya que es un modelo de regresión lineal y utiliza una regularización de *weight decay* que creemos que es buena para este problema.

Este modelo se ha creído conveniente usarlo ya que es una generalización de la regla NN, y es posible que en este problema concreto tenga sentido el aprendizaje por semejanza, con lo que se esperan buenos resultados.

### 1.2.3. Hipótesis finales empleadas

Las hipótesis finales que utilizaremos son las siguientes:

- Se usará un porcentaje de división de datos de 80 % para entrenamiento y 20 % para test.
- Se usará un esquema de validación cruzada de 5 folds.
- Se supondrán las siguientes clases de funciones asociadas a los respectivos modelos de aprendizaje: Regresión lineal, Perceptrón Multicapa con 3 capas ocultas, SVR, AdaBoost con árboles de decisión, Random Forest y Funciones de Base Radial.
- Se aplicarán varios cauces de preprocesado que nos permitan poder entrenar modelos en las clases recién especificadas, utilizando para ello técnicas de tratamiento de valores atípicos, generación de características relevantes para el problema, y normalización de variables en función de la distribución de los datos del conjunto de entrenamiento.
- Se utilizarán como medidas de error MSE, RMSE y $R^2$, siendo la medida a optimizar por nuestros modelos de aprendizaje MSE.
- Se utilizará la regularización Ridge donde sea pertinente usarla, y en los modelos basados en árboles se controlará la regularización modificando el parámetro de escala.
- Para la búsqueda de las mejores hipótesis en validación utilizaremos diferentes parámetros relativos a regularización en caso de que proceda, así como otras características pertinentes en los casos necesarios como explicaremos con posterioridad.
- La mejor hipótesis para cada modelo será aquella que mejor resultado en error de validación haya obtenido.
- Una vez que se haya encontrado la mejor hipótesis, se reentrenará con todos los datos de entrenamiento para encontrar el mejor modelo posible sobre esos datos.
- Seleccionaremos como mejor modelo general aquel que mejor resultado haya aportado en el conjunto de test.

Iremos explicando cada una de estas hipótesis y decisiones detrás de cada una de ellas a lo largo del análisis.

## 1.3. Partición de los datos: Entrenamiento y Test

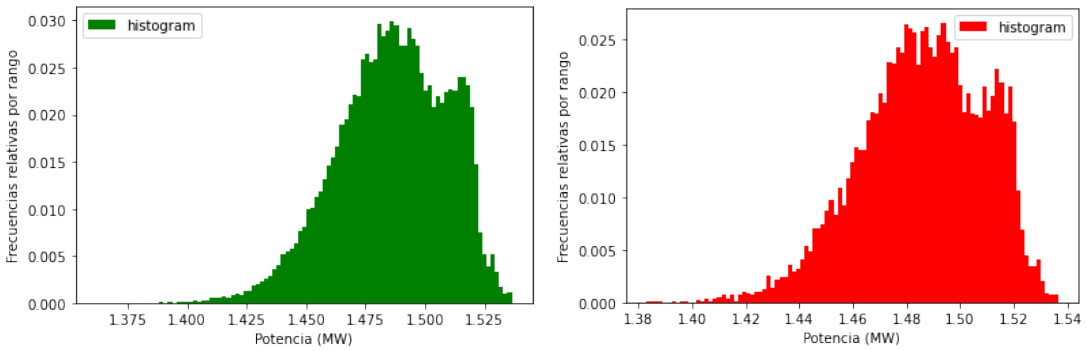
En esta sección hablaremos de cómo se han seleccionado los conjuntos de entrenamiento y test. Hay que recordar que estamos trabajando sobre la base de datos de *Sydney*. En nuestro caso, no poseemos dos conjuntos de datos: uno con datos destinados a entrenamiento y otro con datos destinados a test, sino que sólo poseemos un conjunto de datos con la totalidad de los mismos, por lo que nosotros mismos partiremos dicho conjunto de una forma adecuada para extraer un subconjunto de datos para test.

Como a priori los datos que usaremos no serán demasiado escasos o abundantes, utilizaremos una proporción de partición estándar de los datos. Esta proporción consistirá en reservar un 20 % de datos para test dejar un 80 % de datos para entrenamiento. Esto nos permitirá evaluar nuestros modelos con datos que no han sido vistos durante el proceso de entrenamiento y que son independientes e idénticamente distribuidos frente a éstos, dejando una proporción suficientemente grande de datos para entrenamiento. Como los datos no tienen que seguir ningún tipo de orden, para garantizar el que los datos de entrenamiento y test son independientes,

mezclaremos todos los datos del conjunto antes de realizar la partición, por si acaso éstos tienen algún tipo de orden. La partición la haremos utilizando la función de sklearn

`train_test_split(X,y, test_size)`.

En las gráficas de la Figura 1.3 se puede observar como se distribuyen los datos en ambos conjuntos en términos de frecuencias relativas. Se puede observar que se ha hecho una partición correcta ya que ambos conjuntos siguen una distribución muy similar.




**(a)** Train **(b)** Test

**Figura 1.3:** Distribución de muestras en función de la *potencia total absorbida*. En términos de frecuencias relativas

Con posterioridad utilizaremos un procedimiento de validación cruzada para comparar la calidad de ciertos valores en determinados hiperparámetros en nuestros modelos de aprendizaje. Para realizarlo, volveremos a dividir el conjunto de entrenamiento en 2 subconjuntos: entrenamiento y validación. Utilizaremos un esquema de validación cruzada de 5 folds, puesto que pensamos que es suficiente y supone un compromiso respecto a nuestros recursos computacionales limitados. Por lo tanto, en cada fold de la validación cruzada utilizaremos un total del 80 % de los datos de entrenamiento para entrenar entrenaremos (64 % del total de los datos), y calcularemos el error de validación con el 20 % restante (16 % del total de los datos).

Al encontrar la mejor configuración de hiperparámetros para el modelo, volveremos a entrenarlo, esta vez usando todos los datos de entrenamiento de forma efectiva, para encontrar la mejor aproximación posible a priori puesto que utilizaremos la máxima cantidad de datos e información que disponemos en el conjunto de entrenamiento.

En total poseemos un total de 71999 instancias, con lo que al final tendremos un total de 57599 instancias para entrenamiento y 14400 instancias para test. En el proceso de validación cruzada, para cada fold se entrenarán los modelos con un total de 46079 instancias, dejando un total de 8520 instancias para calcular el error de validación.

## 1.4. Análisis exploratorio de los datos

En esta sección vamos a explorar los datos con el objetivo de intentar extraer conclusiones acerca de los mismos, las cuales puedan ser posteriormente relevantes de cara a preprocesarlos.

### Tipos de datos de las variables

Las 32 variables de las que disponemos son variables reales, y teóricamente acotadas entre 0 y 566, por lo que no tendremos ningún problema a la hora de definir ninguna clase de funciones

o aplicar algoritmos de aprendizaje sobre las mismas.

### Valores perdidos

En este conjunto de datos no existe ningún valor perdido, con lo que no tendremos que realizar ningún tratamiento adicional.

### Visualización de datos

Comenzaremos por hacer una breve resumen estadístico de cada una de las variables. Con este resumen entenderemos mejor su naturaleza y podremos ver si existen algunas variables que se comportan de manera distinta al resto. Puesto que las características están acotadas, y puesto que podremos ver una serie de medidas estadísticas descriptivas básicas, podríamos intuir que alguna variable pudiera tomar valores atípicos.

Consideraremos directamente erróneos los valores de las características referentes a las coordenadas de la posición que sean inferiores a 0 ó superiores a 566, y eliminaremos dichas instancias del conjunto de datos en el preprocesado. Podríamos pensar que se trata de un error a la hora de redondear y podríamos modificar esos valores erróneos poniéndolos directamente al valor extremo del intervalo, pero estos valores inválidos podrían aparecer por otros motivos y es preferible eliminar dichas instancias, en caso evidentemente que no supongan una gran mayoría, en otro caso habría que hacer valoraciones adicionales.

| | **Mean** | **Std** | **Min** | **25 %** | **50 %** | **75 %** | **Max** |
|---|---|---|---|---|---|---|---|
| $X_1$ | 323.987456 | 201.025991 | 0.0000 | 136.99730 | 355.3414 | 523.91030 | 566.0000 |
| $X_2$ | 318.643331 | 200.344355 | 0.0000 | 131.73130 | 354.8975 | 518.47150 | 566.0000 |
| $X_3$ | 248.316824 | 207.938141 | 0.0000 | 42.32020 | 201.7248 | 462.50560 | 566.0000 |
| $X_4$ | 273.297260 | 202.730040 | 0.0000 | 62.50130 | 274.2407 | 468.28500 | 566.0000 |
| $X_5$ | 363.913155 | 165.274775 | 0.0000 | 248.58825 | 388.1354 | 515.60190 | 566.0000 |
| $X_6$ | 270.697487 | 222.065775 | 0.0000 | 38.77060 | 249.7818 | 512.02460 | 566.0000 |
| $X_7$ | 268.204627 | 210.235833 | 0.0000 | 57.37140 | 233.1852 | 485.88890 | 566.0000 |
| $X_8$ | 250.623493 | 191.486786 | 0.0000 | 78.90675 | 212.8147 | 422.45235 | 566.0000 |
| $X_9$ | 257.508938 | 193.844553 | 0.0000 | 71.51360 | 239.2856 | 432.68670 | 566.0000 |
| $X_{10}$ | 263.881558 | 203.195215 | 0.0000 | 48.72300 | 265.5043 | 458.24780 | 566.0000 |
| $X_{11}$ | 314.212835 | 192.640101 | -0.0345 | 163.60065 | 320.2129 | 504.03820 | 566.0000 |
| $X_{12}$ | 276.679784 | 205.585842 | 0.0000 | 79.63880 | 266.1353 | 488.46870 | 566.0000 |
| $X_{13}$ | 273.371300 | 198.447795 | -0.0001 | 75.63440 | 274.7584 | 464.18605 | 566.0000 |
| $X_{14}$ | 340.067221 | 189.676131 | 0.0000 | 180.17210 | 376.1881 | 518.06800 | 566.0000 |
| $X_{15}$ | 280.497450 | 196.855693 | 0.0000 | 87.81280 | 281.9965 | 463.51525 | 566.0000 |
| $X_{16}$ | 273.611649 | 193.364113 | 0.0000 | 85.53535 | 283.8876 | 437.26780 | 566.0000 |
| $Y_1$ | 336.857362 | 191.198836 | 0.0000 | 167.23410 | 372.1299 | 521.08990 | 566.0000 |
| $Y_2$ | 236.064587 | 186.784320 | 0.0000 | 68.43050 | 207.6376 | 397.06790 | 566.0000 |
| $Y_3$ | 256.975686 | 191.431372 | 0.0000 | 64.74345 | 258.4917 | 409.52210 | 566.0000 |
| $Y_4$ | 288.367043 | 190.374024 | 0.0000 | 120.23155 | 285.1156 | 457.98040 | 566.0000 |
| $Y_5$ | 305.646930 | 192.343564 | 0.0000 | 133.89255 | 311.8182 | 485.59550 | 566.0000 |
| $Y_6$ | 336.451657 | 188.880379 | 0.0000 | 194.26720 | 357.5398 | 520.56025 | 566.0000 |
| $Y_7$ | 263.832894 | 189.980429 | 0.0000 | 70.90265 | 276.9588 | 420.00275 | 566.0000 |
| $Y_8$ | 253.856309 | 183.786368 | 0.0000 | 85.09700 | 247.2943 | 404.81400 | 566.0000 |
| $Y_9$ | 285.318026 | 198.003240 | 0.0000 | 102.17995 | 289.8128 | 467.58575 | 566.0000 |
| $Y_{10}$ | 295.238204 | 194.962869 | 0.0000 | 127.09630 | 305.2999 | 475.65815 | 566.0000 |
| $Y_{11}$ | 276.332835 | 208.588042 | 0.0000 | 72.47950 | 249.0457 | 492.56390 | 566.0000 |
| $Y_{12}$ | 306.153740 | 206.819996 | 0.0000 | 88.46680 | 358.4597 | 493.96480 | 566.0008 |
| $Y_{13}$ | 254.431201 | 187.293353 | 0.0000 | 86.92630 | 236.6330 | 417.87880 | 566.0000 |
| $Y_{14}$ | 251.208222 | 186.120696 | 0.0000 | 92.79875 | 222.6482 | 399.97080 | 566.0000 |
| $Y_{15}$ | 303.871908 | 196.030838 | 0.0000 | 120.35730 | 312.2415 | 494.64940 | 566.0000 |
| $Y_{16}$ | 270.284419 | 201.576957 | 0.0000 | 74.63140 | 267.2555 | 469.19160 | 566.0000 |

**Cuadro 1.2:** Resumen estadístico descriptivo de todas las variables originales

Como podemos comprobar en el Cuadro 1.2 donde se puede ver un resumen estadístico descriptivo de las 32 características de los datos originales, efectivamente existen instancias con valores erróneos, con los que directamente las eliminaremos. Por suerte, sólo son 6 instancias las que los poseen, por lo que la proporción respecto del total es ínfima. Asimismo podemos comprobar cómo la distribución que siguen todas las variables es bastante similar.

Puesto que cada instancia posee 32 dimensiones, es imposible realizar una visualización de los datos sobre una superficie 2-dimensional de forma directa. Es por ello por lo que si queremos visualizar los datos, debemos de recurrir a otro tipo de técnicas. Una de estas técnicas es la de reducción de la dimensionalidad, en la que tranformamos nuestras instancias de alta dimensionalidad en otras relacionadas de dimensionalidad mucho más baja de forma que haya características que se conserven entre ambas representaciones. En nuestro caso vamos a utilizar la técnica de reducción de dimensionalidad t-distributed stochastic neighbour embedding (TSNE). Esta técnica modela cada instancia de alta dimensionalidad por un punto con una dimensión nueva y especificada por el usuario, de tal manera que objetos similares son modelados por puntos cercanos y objetos diferentes son modelados por puntos distantes con alta probabilidad. Mostrando por tanto nuestras instancias de forma conjunta con la potencia que proporciona dicha instancia, podremos comprobar la distribución de potencias entre instancias que sean similares en cuanto a su distribución.

En la figura 1.4 podremos ver cual ha sido el resultado de la aplicación de esta técnica 2 veces, especificando que se calculen nuevas coordenadas de los puntos tanto en 2D como en 3D.

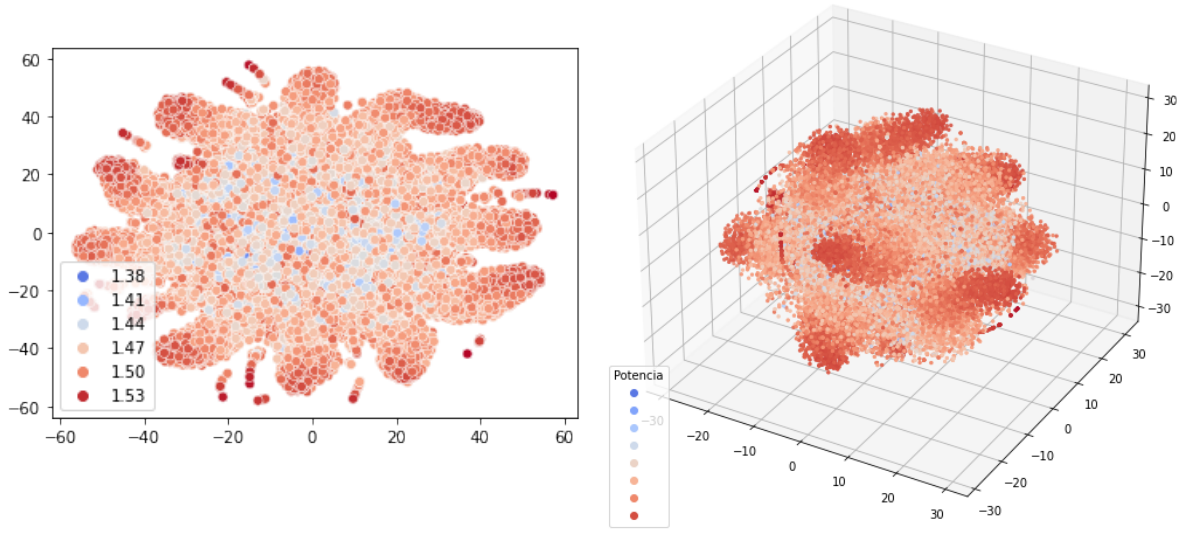



**Figura 1.4:** Visualización de las instancias del conjunto de datos tras la aplicación del algoritmo TSNE para reducción de dimensinalidad a 2 y 3 dimensiones. Instancias cercanas en estas representaciones están asociadas con instancias cercanas en la representación original de las mismas. El color de cada instancia indica la potencia generada por dicha configuración.

La visualización 3D decidimos hacerla puesto que cuanto menor sea la reducción de dimensionalidad, mejor se preservarán las distancias entre pares de instancias. Sin embargo, las conclusiones son muy similares. Como podemos observar, existen unas claras agrupaciones de muchas distribuciones donde la potencia es alta y son muy similares. Estas instancias cercanas entre sí y alejadas de las demás pueden hacer referencia a pequeñas variaciones en las condiciones de los CETOs, o pueden deberse a que existan instancias donde los CETOs se encuentren en las mismas posiciones aunque permutadas. En la visualización 2D podemos identificar 12 grupos bien diferenciados. Además, estos grupos están más alejados de la masa central de instancias,

entre las que se mezclan configuraciones que generan bajas y altas potencias totales, pero desde luego no tan altas como en estos grupos diferenciados. Además se puede ver cómo hay algunas configuraciones con potencias extremadamente altas que están separadas de la masa general, por lo que estos pequeños grupos realmente son instancias muy distantes a cualquier otra (con una configuración distinta).

Sin embargo, esta no es la única forma de visualizar nuestros datos. Nos preguntaremos ahora si los datos siguen algún orden en sus características. Por ejemplo, ¿tienen algo en común los CETOs colocados como primera característica?¿Tienen alguna zona preasignada? Se ha decidido mostrar los datos de cada una de las posicione para todas las instancias. En las Gráficas 1.5 podemos ver como se distribuyen a lo largo de todas las muestras cada CETO según su posición en las características.

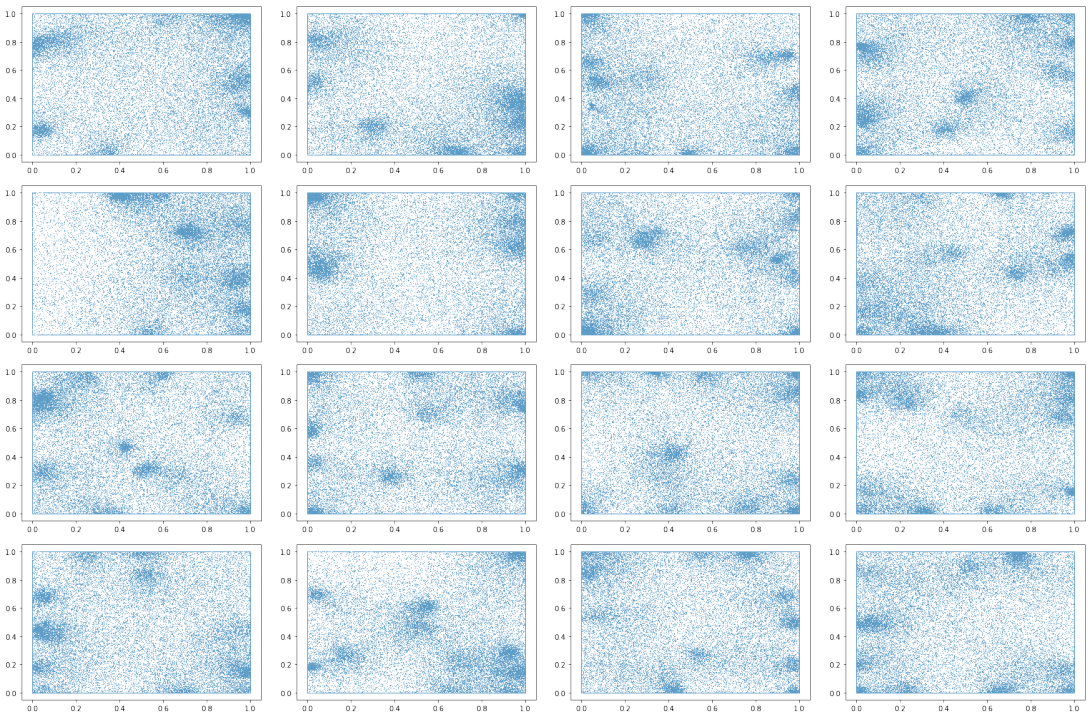


**Figura 1.5:** Posición de los CETOs de todas las muestras. Empezando por la esquina superior izquierda se tiene: CETO 1, CETO 2, CETO 3, CETO 4, siguiente fila: CETO 5, CETO 6,... hasta la esquina inferior derecha CETO 16. El número $i$ en el nombre CETO $i$ hace referencia a que es el $i-esimo$ dato sobre la posición de los CETOs. El rango de los ejes está rescalado del intervalo [0,566] al intervalo [0,1].

Se puede observar como aparentemente cómo los CETOs asociados a cada coordenada tienen una serie de zonas preferentes, aunque podemos observar cómo en general éstos se han distribuido por todo el espacio a lo largo de todas las instancias. No podemos concluir bajo esta evidencia que los CETOs sean completamente iguales entre sí y que no mantengan un determinado orden a la hora de aparecer en nuestras características. Esto podría provocar que no todos ellos tuvieran una misma distribución en la potencia generada.

Vamos a mostrar también una matriz de correlación de las características. Esto nos puede ayudar a ver si existen altas dependencias entre las mismas, ya sea por correlación de forma directa o inversa. Si exisitieran variables altamente correladas, podría empeorar el desempeño de algunos modelos de aprendizaje. Mostramos los resultados en la figura 1.6

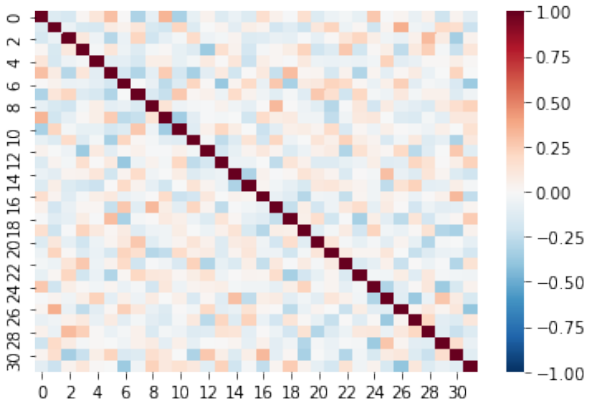



**Figura 1.6:** Matriz de correlaciones: nos indica como de correladas están dos variables entre sí, dando lugar a una matriz simétrica. En la gráfica la correlación 0 está fijada a un color blanco, la correlación 1 (correlación máxima directa) está descrita en color rojo y la correlación -1 (correlación máxima inversa) está descrita en color azul. Se muestra la matriz de correlación de las características originales, **sin preprocesado**.

## 1.5. Preprocesado de datos

### Eliminación de valores atípicos

Como bien se ha dicho en la presentación de los datos, las coordenadas tienen que tomar valores entre $[0, 566]$. Se han encontrado 6 muestras en los datos en donde alguna de sus coordenadas no cumple esta restricción. Debido a que son un número relativamente pequeño de muestras, se ha decidido eliminarlas, asegurándonos no meter ruido al conjunto de datos y quedándonos con un tamaño de los datos casi idéntico.

### Generación de características

Nuestro objetivo es ser capaces de predecir la potencia total generada por una distribución particular de CETOs. Sin embargo, los atributos que nos han dado de la misma son únicamente las posiciones de los CETOs. Es cierto que con estas características tenemos información suficiente para conocer totalmente la distribución particular de los CETOs, pero pensamos que entrenar un modelo con sólo estas variables puede llevarnos a resultados peores que si fuéramos capaces de generar atributos no lineales en base a las que ya tenemos que sean capaces de darnos una mejor información sobre determinadas características relevantes de la distribución de los CETOs, de las que pudiera depender en gran medida la potencia total generada y que fueran comunes a varias distribucions.

Las características que hemos generado las hemos creado en base a medidas relevantes bajo nuestro criterio que nos pueden aportar una información más completa de la distribución de los CETOs. En total, para cada instancia vamos a añadir los siguientes 8 atributos que podemos categorizar en dos subgrupos: atributos de dispersión y atributos de uniformidad.

### Atributos de dispersión

La separación y posición relativa entre los CETOs puede ser una característica a tener en cuenta: los CETOs pueden alterar el funcionamiento de sus vecinos (CETOs cercanos), mientras que si los CETOs están alejados entre sí, no interferirían entre sí y la cantidad de potencia generada por el conjunto podría variar. Por lo tanto, se han decidido incorporar características que nos ayuden un poco a entender cómo de dispersos están los CETOs sobre el recinto.

Siendo $(X_i, Y_i)$ la posición del CETO $i - esimo$, se han cogido los atributos:

- Distancia euclídea mínima entre cualquier par de CETOs distintos:

$$\min_{i,j\in\{1,...,16\},i\neq j} ||(X_i, Y_i) - (X_j, Y_j)||_2$$

- Distancia euclídea máxima entre cualquier par de CETOs distintos.

$$\max_{i,j\in\{1,...,16\}} ||(X_i, Y_i) - (X_j, Y_j)||_2$$

- Media de las distancias euclídeas mínimas entre cada CETO y cualquier otro.

$$\frac{1}{16}\sum_{i=0}^{16} \min_{j\in\{1,...,16\},i\neq j} ||(X_i, Y_i) - (X_j, Y_j)||_2$$

- Media de las distancias euclídeas máximas entre cada CETO y cualquier otro.

$$\frac{1}{16}\sum_{i=0}^{16} \max_{j\in\{1,...,16\}} ||(X_i, Y_i) - (X_j, Y_j)||_2$$

- Distancia euclídea media al centroide de los datos (punto cuya coordenada X es la media de todas las coordenadas X de los CETOs y cuya coordenada Y es la coordenada media de los CETOs):

$$\frac{1}{16}\sum_{i=0}^{16} ||(X_i, Y_i) - (\frac{1}{16}(\sum_{i=0}^{16} X_i, \sum_{i=0}^{16} Y_i))||_2$$

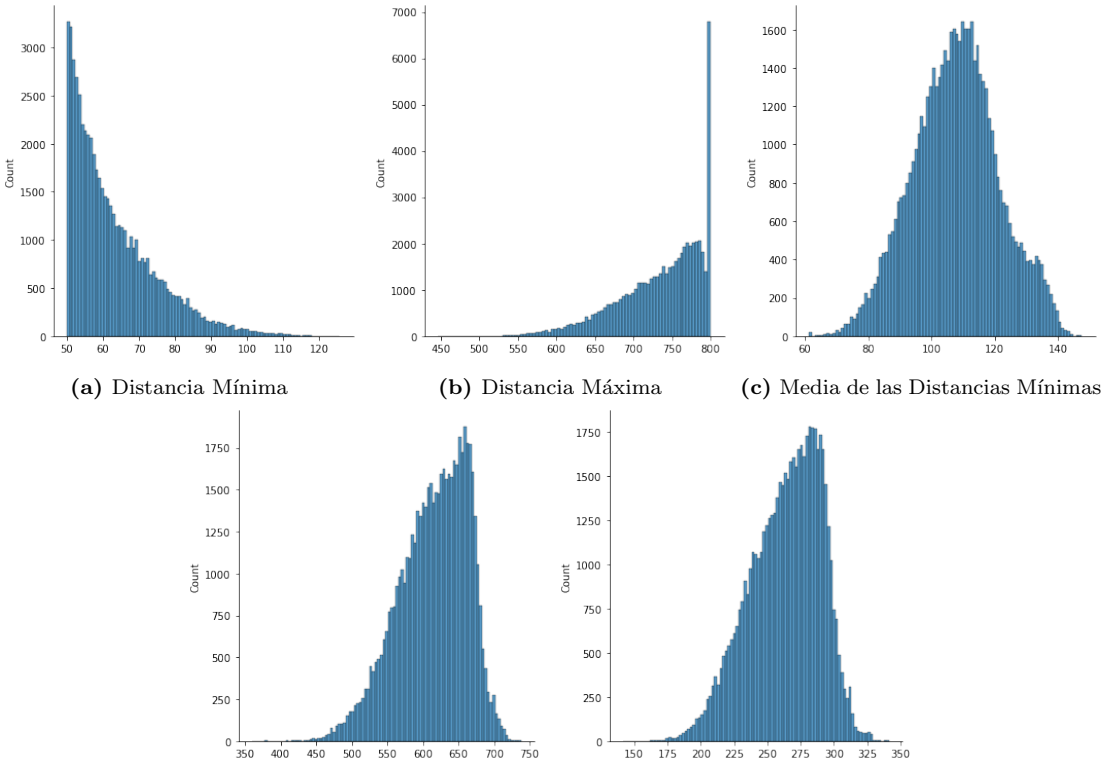
Vamos a mostrar la distribución de cada uno de los resultados para cada variable generada mediante un histograma con ejes autoajustados. Podemos ver los resultados en la figura 1.7

Como se puede comprobar, tanto las características de Distancia Mínima y Distancia Máxima agrupan sus datos alrededor del sus valores extremos correspondientes. Las otras características calculadas como medias tienen unas distribuciones más asemejables a una normal, aunque las dos últimas no son tan simétricas como puede ser la Media de las Distancias Mínimas.

**Atributos de uniformidad**

De gran importancia es también la distribución conjunta de estos CETOs sobre la superficie de 566 $m^2$. A priori, podemos pensar que que los CETOs tengan una distribución uniforme puede ser de gran utilidad para obtener una mayor potencia. Esto se piensa porque en cierto modo, una alta densidad de CETOs en una zona podría aceptar a la capacidad de recepción de energía cinética de las olas por parte de los CETOs. Por otro lado, el que estén uniformemente distribuidos solventaría este supuesto problema, además de que se aproveche de una misma manera toda la superficie. Es por ello que la uniformidad de la distribución de los CETOs va a ser una característica que tenemos que tener en cuenta.

Para comprobar la uniformidad en la distribución de los CETOs hemos decidido aplicar test estadísticos sobre cada instancia para cotejar su uniformidad. El test que hemos decido aplicar es el test $\chi^2$ de uniformidad. Este test es un caso particular del test $\chi^2$. El test estadístico $\chi^2$ es un test de bondad de ajuste, y se aplica a variables aleatorias categóricas. La hipótesis nula $H_0$ sostiene que las probabilidades teóricas de la variable aleatoria tome una categoría concreta




**(a)** Distancia Mínima **(b)** Distancia Máxima **(c)** Media de las Distancias Mínimas

**(d)** Media de las Distancias Mínimas **(e)** Distancia Media al Centroide

**Figura 1.7:** Distribución de las características de uniformidad consideradas. En el eje de abscisas podemos comprobar sus valores, mientras que en el de ordenadas podemos ver sus frecuencias absolutas

son unas probabilidades determinadas conocida. Notando a nuestra variable aleatoria como $X$, y a sus categorías como $A = \{A_1, ..., A_n\}$, tenemos que las hipótesis nula y alternativa son:

$$H_0 : P[X = A_i] = p_i^0, \forall i \in \{1, ..., n\}$$
$$H_1 : P[X = A_i] \neq p_i^0, \text{ para algún } i \in \{1, ..., n\}$$

El test $\chi^2$ de uniformidad es aquel en el que $p_i^0 = \frac{1}{n} \forall i \in \{1, ...n\}$

Para poder aplicar este test estadístico necesitamos hacer una serie de especificaciones y considerar requisitos previos. Para comenzar, debemos suponer que los valores muestreados son una muestra aleatoria simple para poder aplicar el test. La variable aleatoria a la que le vamos a aplicar el test es a la posición que toma en nuestro espacio un determinado CETO. En principio no podremos decir que los CETOs sean diferentes entre sí, ni estructuralmente, ni que sean mejores o peores generando energía... por lo que en realidad, éstos podrían ser intercambiables entre sí, y bajo este pretexto podríamos interpretar que una instancia de nuestra base de datos consiste en muestrear 16 CETOs de forma aleatoria de entre la distribución conjunta de todos ellos. Para mostrar esta distribución, en la Figura 1.8 podremos intuir la distribución bidimensional general de los CETOs, ya que mostramos una nube de puntos donde cada uno es una posición tomada por uno de los 16 CETOs en algunas instancia

Como ya hemos dicho, el test sólo es aplicable a variables categóricas y, como es evidente, la variable aleatoria que describe la posición de un CETO sobre nuestro espacio es una variable aleatoria continua. Es por ello por lo que vamos a hacer una partición de nuestro espacio, con lo que realmente transformaremos la variable posición (x,y) de un determinado CETO en la respectiva región donde cae. Puesto que el conjunto de regiones en las que dividiremos el espacio

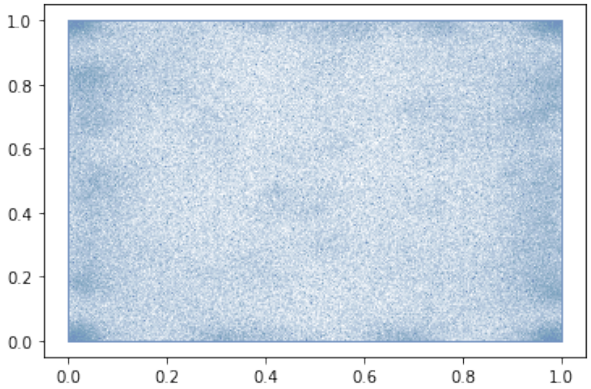



**Figura 1.8:** Distribución de los 16 CETOs de todas las muestras. Aparentemente distribuídos uniformemente. Espacio rescalado $[0, 566]^2 \rightarrow [0, 1]^2$

será finito, hemos transformado nuestra variable continua en una variable categórica.

Como requisito mínimo para aplicar el test, y para que éste tenga un estándar mínimo de significación estadística, es necesario que se cumpla que $Np_i^0 \geq 5, \forall i \in \{1, ..., n\}$, siendo $N$ el tamaño muestral. En nuestro caso, $N = 16$ y $p_i^0 = \frac{1}{n} \forall i \in \{1, ...n\}$, por lo que los únicos valores de $n$, que en nuestro caso que tienen sentido son los valores $n = 2$ y $n = 3$.

Es importante notar que para que las probabilidades teóricas de caer en cada región sean iguales suponiendo la hipótesis de uniformidad, todas las regiones en las que partimos el espacio deben tener el mismo área. Podría generalizarse el test de uniformidad en caso de que quisiéramos coger regiones con áreas distintas y tener en cuenta la proporción en el tamaño de las áreas para calcular las probabilidades teóricas, pero nos parece innecesario.

Como tenemos que dividir el espacio en 3 partes iguales que simbolicen uniformidad en toda la superficie y no ha sido posible hacerlo sobre nuestro espacio, se tendrá que buscar una combinación de división de regiones sobre las que se aplicará el test de uniformidad, de tal forma que con los resultados de todas las divisiones se pueda concluir si el espacio es uniforme o no.

Una forma correcta de dividir el espacio sería en una cuadrícula $3 \times 3$, en donde se recoge información uniforme sobre el área de la superficie, en donde lo esperado de una distribución uniforme es que en todas las casillas hubiese el mismo número de apariciones de nuestra variable. Pero como se ha comentado, esto no es posible porque no podemos dividir el espacio en 9 regiones, solo en 3. Nuestro objetivo es por tanto conseguir un conjunto de particiones que nos den información sobre las 9 casillas siguiendo la restricción de que cada región de cada partición deben esperarse al menos 5 apariciones de un CETO.

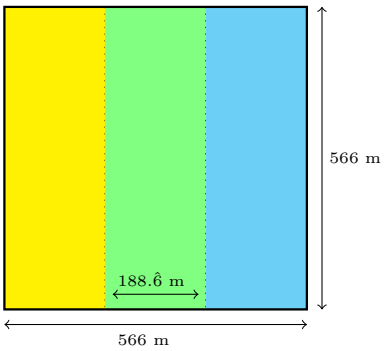
| | | |
|---|---|---|
| $a_{11}$ | $a_{12}$ | $a_{13}$ |
| $a_{21}$ | $a_{22}$ | $a_{23}$ |
| $a_{31}$ | $a_{32}$ | $a_{33}$ |

Graf. 1: Cuadrícula $3 \times 3$

En la Graf. 1 tenemos una representación de nuestra cuadrícula. Nos preguntamos ahora, ¿qué combinación de particiones de 3 nos servirían para que su información equivaliese a la

uniformidad total? La respuesta sería: las suficientes para que se pueda determinar de forma unívoca la cantidad de muestras que caen dentro de cada región $a_{ij}$. Para ello se proponen las particiones de los Gráficos 2,3,4.

¿Son estas particiones válidas? Efectivamente: siendo $P_i$ la partición del Gráfico $i+1$, tenemos que las componentes $Y_i, G_i, B_i$ son los conjuntos Amarillo, Verde y Azul respectivamente de la partición $P_i$. En caso de uniformidad, estas particiones deberían de tener el mismo número de apariciones de la variable: $|Y_i| = |G_i| = |B_i|$. Suponiendo que el número de muestras es $N$, esto nos produce un total de 9 ecuaciones (3 por partición).



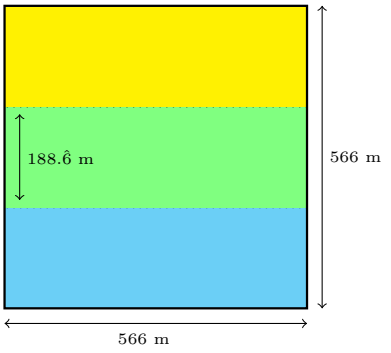
Graf. 2: Distribución respecto el ejeX

Las ecuaciones serían:

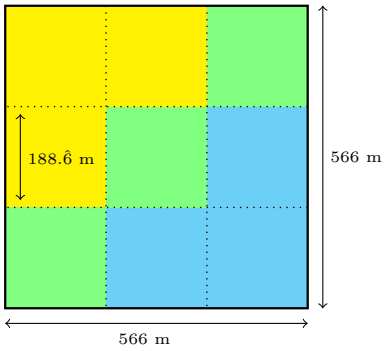
$$P_1 : \begin{cases} Y_1 = \sum_{i=1}^{3} a_{i1} = \frac{N}{3} \\ G_1 = \sum_{i=1}^{3} a_{i2} = \frac{N}{3} \\ B_1 = \sum_{i=1}^{3} a_{i3} = \frac{N}{3} \end{cases} \tag{1.7}$$

$$P_2 : \begin{cases} Y_2 = \sum_{j=1}^{3} a_{1j} = \frac{N}{3} \\ G_2 = \sum_{j=1}^{3} a_{2j} = \frac{N}{3} \\ B_2 = \sum_{j=1}^{3} a_{3j} = \frac{N}{3} \end{cases} \tag{1.8}$$

$$P_3 : \begin{cases} Y_3 = \sum_{i+j\leq 3} a_{ij} = \frac{N}{3} \\ G_3 = \sum_{i+j=4} a_{ij} = \frac{N}{3} \\ B_3 = \sum_{i+j\geq 5} a_{ij} = \frac{N}{3} \end{cases} \tag{1.9}$$



Graf. 3: Distribución respecto el ejeY



Graf. 4: Distribución respecto Diagonal

Nos da un sistema de 9 ecuaciones independientes de donde sacamos un sistema determinado. Por lo tanto podemos obtener una solución que determina unívocamente las instancias que caen dentro de cada $a_{ij}$, en donde claramente tenemos el resultado:

$$P_i \; unif \; \forall i \in \{1,2,3\} \iff C_{3\times 3} \; unif \tag{1.10}$$

Donde $C_{3\times 3}$ es la cuadrícula del Gráfico 1.

Aplicaremos, por tanto, 3 test de uniformidad distintos para cada partición obteniendo la información relevante sobre la uniformidad del conjunto sobre la superficie entera. Al aplicar estos tests, obtenemos dos valores distintos, un valor siendo el del estadístico experimental, y otro siendo el valor del p-nivel. Sólo podemos seleccionar uno de estos valores como la característica generada, y nos decantaremos por el p-nivel puesto que tiene una clara interpretación.

El p-nivel indica la probabilidad de haber obtenido los resultados muestrales que se han dado, supuesto que sea cierta la hipótesis $H_0$. Por tanto, un p-nivel muy bajo indica que la hipótesis nula probablemente sea falsa. Se suele fijar un valor límite a partir del cual si el p-nivel es menor que éste, se considerará que hay evidencia para rechazar la hipótesis nula $H_0$. Este es el valor de significación $\alpha$, que cuando no existe ningún criterio experto por el cual

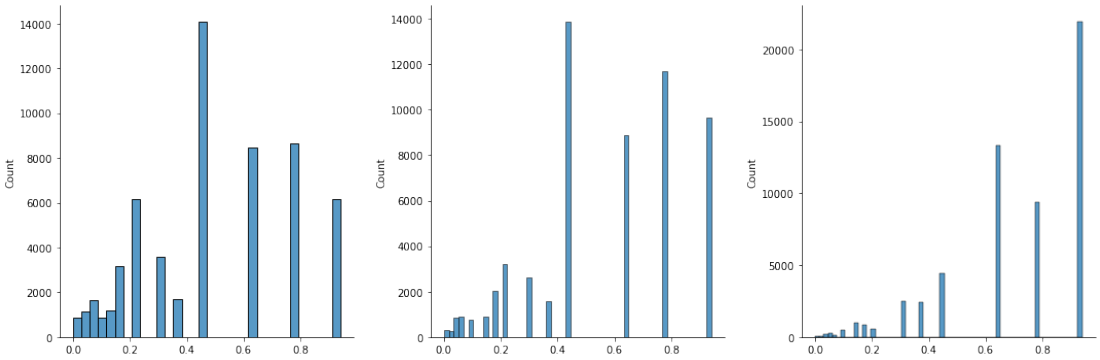
fijarlo a un valor concreto, comúnmente se emplea el valor 0.05. También se habla de forma equivalente de fijar un nivel de confianza del $100(1-\alpha)\,\%$.

En caso de que el p-nivel fuera mayor que el nivel de significación, no tendríamos evidencia suficiente para rechazar la hipótesis nula. El valor del estadístico experimental no tiene una interpretación tan directa ni desde luego es tan empleado como el p-nivel puesto que su rango y escala varía para cada tipo de test estadístico, mientras que el p-nivel siempre toma el mismo rango de valores y se puede obtener e interpretar de forma independiente al test empleado. Además, el p-nivel nos dice cómo de probable es realmente que la distribución subyacente a la muestra sea uniforme frente a la partición del espacio realizada por el test estadístico, lo que dependerá de forma evidente de la estructura de la muestra y que por tanto es una medida directa de su uniformidad. Por tanto será este el valor que utilizaremos como característica a generar mediante la aplicación del test estadístico.

Estos tres tests se han unido al Pipeline a través de las clases que hemos creado:

- `Chi2x`
- `Chi2y`
- `Chi2Diagonal`

Vamos a mostrar la distribución de cada uno de los resultados para cada variable generada mediante un histograma con ejes autoajustados. Podemos ver los resultados en la Figura 1.9




**(a)** Partición $P_1$ (Gráfica 1) **(b)** Partición $P_2$ (Gráfica 2) **(c)** Partición $P_3$ (Gráfica 3)

**Figura 1.9:** Distribución de los valores del p-nivel en las distintas particiones. En el eje de las abscisas se representa el valor del p-nivel y en el eje de las ordenadas su frecuencia.

Hay evidencias de que en cada partición una mayoría de muestras tienen un p-nivel mayor a 0.05, lo cual significa que no tenemos evidencia suficiente para rechazar la hipótesis nula $H_0$. Trasladando el significado de estos hechos a nuestro problema, quiere decir que los CETOs con una alta probabilidad se distribuyen de manera uniforme sobre la superficie de $566m^2$. Luego nos quedamos con la característica del p-valor de cada partición para indicar el grado de uniformidad de una muestra concreta, puesto que puede ser un valor importante a la hora de predecir la potencia.

### Normalización

En la sección de análisis exploratorio de datos hemos podido comprobar cómo todas las variables medidas toman valores de forma consistente en todo el rango de valores, con lo que tiene sentido hacer una normalización simple, cambiando la escala de dichas variables en el rango $[0, 1]$. Sin embargo, la normalización aplicada debería ser la misma para todas las variables, por

lo que debemos de comprobar que para las variables que hemos considerado generar, la normalización al rango $[0, 1]$ no sería una mala opción. Efectivamente, las variables correspondientes a los test $\chi^2$ ya están de por sí normalizadas al rango $[0, 1]$.

Para los atributos de Distancia Mínima y Distancia Máxima es preferible realizar esta normalización al rango $[0, 1]$ ya que en general se recomienda hacerlo cuando las variables no siguen una distribución normal. La discusión surge cuando vemos la distribución de las variables calculadas como medias. Aunque su distribución es bastante equiparable a las de una normal, el hecho de que no posean colas muy pronunciadas o valores extremos, y sean relativamente leptocúrticas, hace que la normalización al rango $[0, 1]$ sea una opción a considerar también. Sumado a que para el resto de variables la normalización al rango $[0, 1]$ es una buena opción, hará que nos decantaremos por este tipo de normalización definitivamente para todos los datos.

Esta normalización la podremos realizar usando la clase de sklearn:

`MixMaxScaler()`.

**Pipeline final**

Tendremos por tanto al final 2 Pipelines distintos.

```
featuregeneration =[("DistanciaMinima", DistanciaMinima() ),
                    ("DistanciaMaxima", DistanciaMaxima() ),
                    ("MediaDistMin", MediaDistanciasMinimas() ),
                    ("MediaDistMax", MediaDistanciaMaximas() ),
                    ("DistMediaCentroide", DistanciaMediaACentroide() ),
                    ("Chi2x", Chi2x() ),
                    ("Chi2y", Chi2y() ),
                    ("Chi2Diagonal", Chi2Diagonal() )]

scaler =[("MinMaxScaler", MinMaxScaler() ),
```

La división en 2 pipelines se hace por el hecho de que el pipeline de generación de características tiene que calcularse y aplicarse respecto a ambos conjunto de entrenamiento y test por separado: la generación de características depende de las propias instancias de cada conjunto. Sin embargo la normalización debe calcularse para el conjunto de entrenamiento y aplicarse a ambos conjuntos de datos, puesto que en teoría no disponemos de la información del conjunto de test para normalizar, sino sólo la de entrenamiento, y la normalización se debe hacer en los dos conjuntos de la misma forma para que las características que tomen el mismo valor tras la transformación también tuvieran el mismo valor antes de la transformación (biyectividad).

**Cambio de unidades de $\mathcal{Y}$**

Hemos decidido cambiar la escala de medida de nuestra variable a predecir $\mathcal{Y}$. Recordemos que la unidad en la que nos viene dada es en **vatios (W)**. Para entender el cambio de unidades tenemos que darnos cuenta de que, tras la normalización de todas las características, éstas están en un rango $[0, 1]$, sin embargo, nuestra variable a predecir está en el orden de millones ($10^6$). Esto implica que habrá modelos de aprendizaje que necesiten aprender unos pesos con norma excesivamente elevada, lo cual puede complicar el proceso de aprendizaje.

Para solventar el problema, se ha creído conveniente pasar las unidades de de medida a **MegaVatios (MW)**: $W \longrightarrow MW$. Esto implicará que nuestra variable a predecir se moverá en el orden de unidades y decimales, estando mucho más acorde con los rangos de valores, haciendo más fácil el trabajo para el modelo de aprendizaje.

### Análisis de la correlación de los nuevos atributos

Vamos a mostrar la matriz de correlación de los atributos, pero a diferencia de la anterior, conteniendo ésta todos los atributos nuevos, con el objetivo de saber si están altamente correlados con alguna variable previamente existente o entre las generadas también.

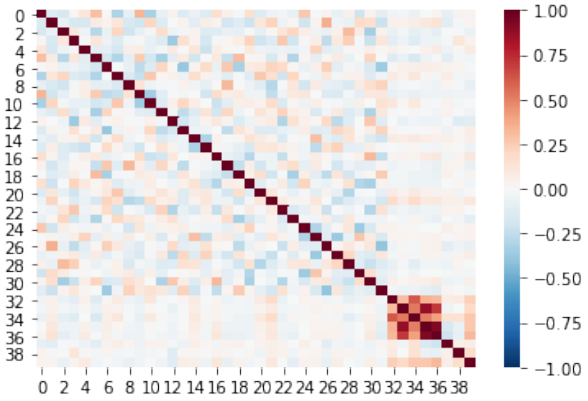



**Figura 1.10:** Matriz de correlaciones: nos indica como de correladas están dos variables entre sí, dando lugar a una matriz simétrica. En la gráfica la correlación 0 está fijada a un color blanco, la correlación 1 (correlación máxima directa) está descrita en color rojo y la correlación -1 (correlación máxima inversa) está descrita en color azul. Se muestra la matriz de correlación de las características **con preprocesado**.

Tal y como se puede comprobar, las nuevas características tienen una correlación muy muy cercana a 0 respecto de las originales, por lo que la información aportadas por estas nueva características tiene una naturaleza muy distinta a cada variable original por separado, lo que tiene sentido. Entre las características de dispersión podemos ver que hay correlaciones más elevadas. Una de ellas podría puede considerarse como extremadamente alta, entre la Media de las Distancias Máximas y la Distancia Media al Centroide. Sin embargo vamos a dejarlas ambas porque consideramos que las informaciones tienen distintas naturalezas. Los p-niveles entre sí tienen una correlación bastante baja, y también respecto al resto de características generadas.

## 1.6. Métricas

Para el problema de regresión utilizaremos dos métricas distintas, siendo una el error cuadrático medio, también conocido por sus siglas en inglés MSE, y la otra el coeficiente de determinación, $R^2$.

### Métrica de error

Se ha usado la métrica de error cuadrático medio **MSE** (1.11). El error cuadrático medio de un estimador mide el promedio de los errores al cuadrado, es decir, la diferencia entre lo estimado y lo que se estima [5].

$$MSE(h) = \frac{1}{N}\sum_{i=0}^{N}(h(x_n) - y_n)^2 \tag{1.11}$$

donde $h$ es el estimador.

El MSE es un error típico utilizado para la regresión lineal en el campo de la estadística, y supone una métrica de error estándar para los problemas de regresión en el campo del Aprendizaje Automático. Esta medida penaliza en gran medida a los valores en los que la predicción ha sido muy errada. Está acotada inferiormente por 0, pero no superiormente. El MSE es el

segundo momento (sobre el origen) del error, y por lo tanto incorpora tanto la varianza del estimador así como su sesgo. Para un estimador insesgado, el MSE es la varianza del estimador.

### Otras métricas

Aquí enumeraremos otras métricas auxiliares que se han usado para medir la calidad de la solución.

1. Al igual que la varianza, el MSE tiene las mismas unidades de medida que el cuadrado de la cantidad que se estima. En una analogía con la desviación estándar, tomando la raíz cuadrada del MSE se obtiene lo que se denomina la desviación de la raíz cuadrada media, **RMSE** (1.12), que tiene las mismas unidades que la cantidad que se estima. Por lo tanto, para un estimador insesgado, el RMSE es la desviación estándar.

$$RMSE(h) = \sqrt{MSE(h)} = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(h(x_n) - y_n)^2} \tag{1.12}$$

2. Utilizaremos además de esta medida, otra medida complementaria que podría ayudarnos a interpretar de otra forma los resultados de error igualmente, esta vez mediante una medida que sí está acotada superiormente en la que cuanto más alto sea el valor obtenido mejor modelo habremos encontrado. Se trata de el **coeficiente de determinación**, también conocido como $R^2$ [4]. Su fórmula es la siguiente:

$$R^2(h) = 1 - \frac{\sum_{i=0}^{N}(y_i - h(x_i))^2}{\sum_{i=0}^{N}(y_i - \overline{y})^2} \tag{1.13}$$

   Esta medida también puede escribirse como:

$$R^2(h) = 1 - \frac{MSE(h)}{Var(y)} \tag{1.14}$$

   En el denominador aparece la varianza total de los datos, y en el numerador aparece la varianza de los residuos, que coincide con el MSE (la división por el número de datos se simplifica). En el mejor de los casos, con una predicción perfecta, la varianza de los residuos será 0 con lo que la medida valdrá 1. De igual forma, esta medida no está acotada inferiormente porque la varianza de los residuos (MSE) podría crecer ilimitadamente. Un valor $R^2(h) = 0$ significaría que nuestra función $h$ ha cometido tanto error como si se hiciera una predicción constante igual a la media de los valores a predecir. Si el valor obtenido es menor que 0, la predicción hecha sería en general peor que este caso. Si el valor obtenido es superior a 0, obtenemos menos error en nuestra regresión de forma relativa a la varianza del modelo.

3. Para este problema en particular podríamos crear métricas de error específicas. Por ejemplo, podría no significar lo mismo un fallo por defecto que por exceso a la hora de generar energía eléctrica a la hora de satisfacer la demanda eléctrica de un proceso o para una determinada población. Por ejemplo, errar por exceso podría conllevar sólo costes económicos de almacenamiento de la energía generada, pero equivocarnos por defecto podría conllevar el tener que recurrir a terceros para generar energía por otro tipo de medios, más caros o no tan económicos, o comprometidos con el medio ambiente. Sin embargo, no somos expertos ni disponemos de dicha información con lo que nos abstendremos de proponer

una métrica de estas características, aunque sería muy interesante y totalmente pertinente consultar con un experto para su debate, discusión y posible implementación.

## 1.7. Parámetros y tipo de regularización

Llegados a este punto. Antes de aplicar *Cross-Validation* tenemos que especificar un grid de parámetros sobre el que la técnica de *Cross-Validation* eligirá los mejores.

```
algs_and_params =[
                  #Regresión lineal
                  [{'Model': [Ridge(max_iter=maxIter,
                                    random_state=randomstate)],
                       "Model__alpha": LinRegSpace}],

                  #Perceptrón multicapa
                  [{'Model':
                  [MLPRegressor(hidden_layer_sizes=(100,100),
                              activation='relu', solver="sgd", max_iter=200,
                              learning_rate='adaptive', learning_rate_init=1e-2,
                              random_state=randomstate, early_stopping=True)],
                       "Model__alpha": MLPSpace}],

                  #SVR, con kernel lineal
                  [{'Model': [SVR(kernel='poly', max_iter=8000)],
                       "Model__C": SVRCSpace,
                       "Model__epsilon": SVRepsilonSpace}],

                  #AdaBoost
                  [{'Model': [AdaBoostRegressor(n_estimators=100,
                                     loss='linear', random_state=randomstate)],
                       "Model__learning_rate": BoostSpace}],

                  #Random Forest
                  [{'Model': [RandomForestRegressor(max_depth=16,
                                     max_features='sqrt', n_jobs=-1)],
                       "Model__n_estimators": RFSpace}],

                  #Funciones de base radial
                  [{'Model': [RBFNetworkRegressor(batch_size=128)],
                       "Model__k": kSpace,
                       "Model__gamma":gammaSpace}]
                       "Model__alpha":alpha}]
                  ]
```

Donde `LinRegSpace`, `MLPSpace`, `SVRSpace`, `SVRepsilonSpace`, `BoostSpace`, `RFSpace`, `kSpace`, `gammaSpace` son los conjuntos de valores que pueden tomar nuestros parámetros. Estos serán explicados en su apartado correspondiente dentro de la subsección de **Parámetros** y **Regularización**.

### 1.7.1. Parámetros

Vamos a proceder a explicar los parámetros usados para cada modelo. Los parámetros dedicados a la regularización se comentarán en la subsección de **Regularización**.

#### Ridge - Regresión Lineal

Empezaremos con Regresión Lineal. En este modelo tenemos como parámetros que determinan la definición del modelo el solver a utilizar, la tolerancia y el número de iteraciones.

- El solver se ha dejado en `auto` puesto que como no conocemos los tipos de solver propuestos y `auto` hace que se seleccione el solver en función al tipo de dato, lo hemos visto lo más conveniente.
- La tolerancia `tol` se ha dejado por defecto, $10^{-3}$ ya que creemos que es un buen nivel de precisión.
- Por último el número de iteraciones máximas se ha dejado en 2000 puesto que creemos que es un número suficientemente algo como para que converja a una solución buena.

#### Perceptrón Multicapa

En el modelo MLP tenemos múltiples parámetros para determinar nuestro modelo. Hemos querido dotar al modelo de dos capas ocultas con 100 neuronas en cada una para darle una mayor capacidad de ajuste que evidentemente tendremos que regular posteriormente con los parámetros de regularización explicados en la subsección de **Regularización**.

Se ha decidido utilizar como función de activación `'relu'` basándonos en [2]. En este estudio se destacan varias características a favor de esta función de activación, entre ellas:

- El cálculo de relu es mucho más eficiente que la sigmoidal y la tangente hiperbólica.
- Proporciona menor error en general.
- En caso de redes profundas salva el problema de desvanecimiento del gradiente.

Para el solver se ha decidido usar `SGD` ya que es el que se ha visto durante el curso. Se ha usado un número de iteraciones bajo: 200, puesto que los Perceptrones Multicapa requieren de mucho cómputo y no queríamos sobrecargar la búsqueda en solo este modelo. Se ha considerado que 200, que además es el valor por defecto en Scikit Learn ([3]) es adecuado.

Para el `learning_rate` se ha elegido empezar en un valor "alto" de 0.01 y poner un `learning_rate` adaptativo, para fomentar la exploración en el espacio de soluciones.

#### Support Vector Regressor

Para el modelo SVR tenemos que especificar distintos parámetros, entre ellos el tipo de kernel. Se ha decidido optar por un kernel polinómico de grado 3 (por defecto), para dar complejidad al modelo ya que podemos permitirnos este grado polinómico debido a la eficiencia de los kernel. Los parámetros `gamma` y `coef0` se han dejado por defecto, ya que en el caso de `gamma` nos valía con que este dependiese del número de características, el por defecto es: $\frac{1}{n\_features}$. El `coef0` por su parte es un término independiente en la función kernel, se ha dejado el valor por defecto en 0 para no complicar el espacio de búsqueda dentro del grid. El número máximo de iteraciones se han puesto en 8000 para dar margen a una buena solución.

**AdaBoost**

Para el modelo de AdaBoost los parámetros esenciales son: la función de pérdida y el número de estimadores. Se ha decidido poner 100 estimadores como compromiso entre cantidad de estimadores y tiempo de ejecución. La función de pérdida se ha dejado lineal para que la actualización de los pesos sea acorde a sus residuos.

Para el learning rate hemos hecho un espacio de valores para el grid, como único hiperparámetro a considerar:

$$\texttt{BoostSpace} = \{0.5, 0.7, 0.9\}$$

El learning rate tiene una contribución inversa al número de estimadores. Como el número de estimadores es 100, se ha decidido bajar el valor del learning rate a unos valores menores de 1.

**Random Forest**

En Random Forest, los parámetros escogidos han sido:

- `criterion`: se ha usado el error cuadrático medio, como se ha venido haciendo para todos los modelos.
- `max_features`: se ha usado `sqrt` ya que en las diapositivas de teoría se ha visto que este compromiso tiene un gran desempeño.

Para la regularización (que se comentará después) se ha usado el parámetro `n_estimators`.

**RBF - Funciones de Base Radial**

En las funciones de base radial el hiperparámetro a controlar es nuestro valor de `gamma` y el número de centros `k`. Que es un multiplicador en el radio. Como se hacía referencia, siguiendo los pasos propuestos por el libro [1], `gamma` tiene que tomar un valor dentro del intervalo [1,3]. Así, el espacio de valores para `gamma` ha sido:

$$\texttt{gammaSpace} = \left\{1 + i\tfrac{2}{7} : i \in \{0, ..., 7\}\right\}$$

y para el valor `k` se ha decidido coger valores para conseguir buen ajuste sin pasarnos al overfitting. Estos valores que hemos considerado son:

$$\texttt{kSpace} = \left\{2^i : i \in \{3, 4, 5, 6, 7\}\right\}$$

### 1.7.2. Regularización

Como sabemos, se han usado varios modelos diferentes y no todos poseen una misma forma de regularización. Por ello, en esta subsección vamos a centrarnos en la regularización de cada modelo, introduciendo el porqué de esa regularización y argumentando el motivo de los distintos valores de nuestro espacio de valores para los parámetros.

**Ridge - Regresión Lineal**

El modelo de aprendizaje `Ridge` hace referencia a un modelo de aprendizaje de Regresión Lineal con una penalización `'l2'`, conocida como *weight decay*. Para entender por qué se ha elegido esta regularización hay que pensar en los datos que tenemos. Tenemos poca cantidad de características, además que se han valorado poco redundantes y por lo tanto representativas. Luego:

1. Una regularización `'l1'` queda totalmente descartada puesto que tiene una función de selector de características, ya que tiende a hacer algunos pesos 0, y no lo creemos necesario.
2. La penalización `'l2'` *weight decay* es totalmente adecuada para nuestro problema ya que su fin es obligar a que los pesos sean bajos, penalizando en mayor medida los pesos muy altos, pero sin hacer una selección implícita de características.

El rango de valores que se le ha dado al parámetro `alpha` han sido:

$$\texttt{LinRegSpace} = \left\{10^i : i \in \{-4, -3, -2, -1, 0, 1\}\right\}$$

Se han creído que estos valores eran necesarios ya que se espera que la regresión no necesite penalizar mucho, es por ello que hay tantos exponentes negativo acompañando a la base 10.

**Perceptrón Multicapa**

En el Perceptrón Multicapa tenemos una regularización por defecto `'l2'`, parámetro que será muy importante en este modelo. Tenemos que darnos cuenta de que tenemos muchas neuronas y que este modelo tiende al *overfitting*. Es por ello que tendremos que aplicar una regularización sustanciosa y el *weight decay* es idónea para regular el *overfitting* y no ajustarnos al ruido. En Scikit Learn, el valor por defecto de `alpha` es $\alpha = 0.0001$. Así que se ha decidido dar valores más sustanciales que ese $\alpha$ por defecto. En concreto se han asignado los valores

$$\texttt{MLPSpace} = \left\{10^i : i \in \{-4, -3, -2, -1, 0, 1\}\right\}$$

Por último, se ha puesto `early_stopping` para ayudar a evitar el sobreajuste. Existen otros métodos para evitar el sobreajuste en perceptrones multicapa como puede ser el `dropout`, pero la implementación de Scikit Learn no nos permite por medio de un parámetro configurarlo.

**Support Vector Regressor**

Ahora cambiamos totalmente de paradigma. En los modelos SVM, y en particular en SVR, la regularización no va asociada a la norma de los pesos si no a un término que está ligado con la anchura del pasillo. En concreto los parámetros libres que usamos son $\varepsilon, \mathbf{C}$. Se recuerda el problema primal que se resuelve en este problema $\varepsilon$-SVR: dados $x_i \in \mathbb{R}^d, i = 1, .., n$, siendo $\phi$ el kernel y las características $y \in \mathbb{R}^n$, el problema primal es:

$$\begin{aligned} \min_{w,b,\zeta,\zeta^*} \frac{1}{2} w^T w + \mathbf{C} \sum_{i=1}^{n} (\zeta_i + \zeta_i^*) \\ subject\ to: y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i, \\ w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ \zeta_i, \zeta_i^* \geq 0,\ i = 1, ..., n \end{aligned}$$

Donde estamos penalizando las muestras tales que sus predicciones están al menos $\varepsilon$ lejos de su etiqueta verdadera. Estas muestras penalizan el objetivo a través de $\zeta_i, \zeta_i^*$, cuya suma está

regulada por nuestro parámetro **C**.

Por otro lado, $\varepsilon$ indica la anchura por defecto del "pasillo". Estos han sido los dos parámetros que hemos usado para regularizar. Cuyo espacio de valores hemos tomado:

$$\begin{aligned} \texttt{SVRCSpace} &= \left\{10^i : i \in \{-4,-3,-2\}\right\} \\ \texttt{SVRepsilonSpace} &= \left\{10^i : i \in \{-4,-3,-2\}\right\} \end{aligned}$$

Se han creído adecuados estos valores para regularizar **C**, se mueven dentro de unos valores habituales en parámetros de regularización. Por otro lado el espacio del epsilón se ha visto adecuado ya que regula la anchura del pasillo en unidades completamente coherentes con respecto a las unidades de las etiquetas, dejando un pasillo muy estrecho en el caso más extremo, que permite que se lleguen a mejores soluciones: en Scikit Learn ([3]) el valor por defecto es 0.1 (bastante elevado en nuestra opinión).

#### AdaBoost

La regularización en AdaBoost va implícita en los estimadores tan simples que usa: al usar árboles de profundidad tres como máximo, tienen una regularización intrínseca muy alta.

#### Random Forest

Para la regularización se ha modificado el valor de `n_estimators`. La cantidad de árboles puede influir bastante en el sobreajuste del modelo, con lo que probaremos con los siguietnes valores:

$$\texttt{RFSpace} = \{50, 100, 200\}\}$$

#### RBF - Funciones de Base Radial

Para la regularización de las redes de Funciones de Base Radial se ha utilizado el parámetro `alpha` que regula el modelo lineal que usa por dentro. La regularización es *weight decay*. Los distintos valores que hemos fijado para probar son los siguientes:

$$\texttt{alpha} = \{0.01, 0.1, 1\}\}$$

## 1.8. Cross-Validation. Selección de la mejor hipótesis

Llegados a este punto toca aplicar *Cross-Validation 5-folds* como se había hablado en la sección **Partición de los datos: Entrenamiento y Test**. Usaremos la función `GridSearchCV` de Scikit Learn.

El motivo de usar *Cross-Validation* viene porque es el mejor *modus operandi* para elegir un modelo. Al final tenemos varias opciones de estimadores que cada uno de ellos conlleva varias opciones de parámetros. Es una forma en la que el error de validación se hace bastante representativo y se podría tomar como una estimación de lo que sería el error fuera de la muestra.

La mejor hipótesis que hemos obtenido mediante *Cross-Validation* ha sido

```
RandomForestRegressor(  max_depth=16,
                        max_features='sqrt',
                        n_estimators=200,
                        n_jobs=-1).
```

Con esta hipótesis se ha obtenido un error de validación cruzada $E_{CV} = 3.249 \times 10^{-5}$ a través del error cuadrático medio, lo que equivale a $0.0057MW$ (RMSE), un error bajísimo, $5700W$ es un valor muy bajo para el error lo cual es símbolo de un buen ajuste en los datos de entrenamiento.

En el Cuadro 1.3 podemos ver los errores obtenidos por cada modelo en el proceso de *Cross-Validation*.

| **Modelo** | **Mejores hiperp.CV** | $RMSE_{val}$ |
|---|---|---|
| Regresión lineal | $\alpha = 0.01$ | 0.0080 |
| MLP | $\alpha = 0.01$ | 0.0081 |
| SVR | $C = 0.001$, $\epsilon = 0.01$ | 0.0077 |
| AdaBoost | $lr = 0.05$ | 0.0098 |
| Random Forest | $n\_estimators = 200$ | 0.0057 |
| RBF | $\alpha = 0.01$, $\gamma = 3$, $k = 32$ | 0.0103 |

**Cuadro 1.3:** Resultados de todos los modelos

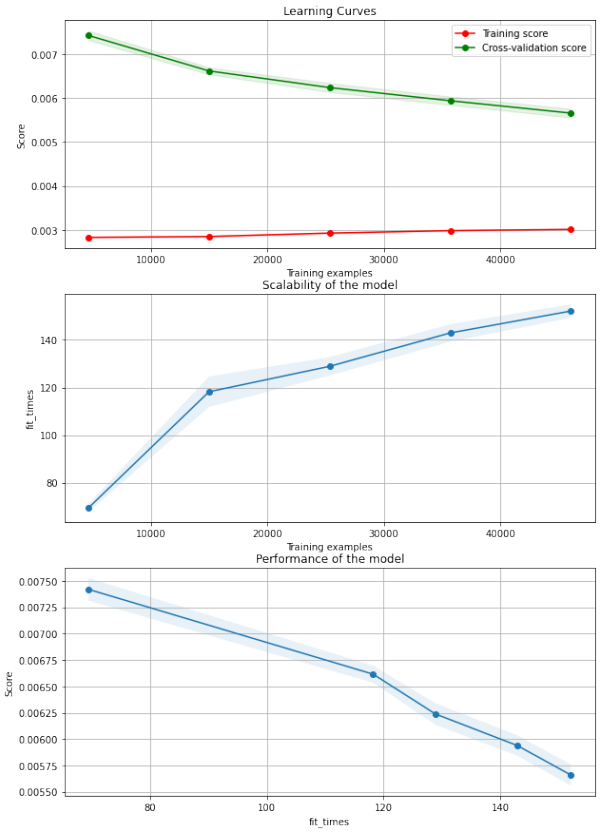



**Figura 1.11:** Estas gráficas se han generado a través de `learning_curve`

Para ver el error de validación del mejor modelo (Random Forest) se han generado las curvas de aprendizaje que podemos ver en la Figura 1.11. El Score se ha transformado: como dijimos en su momento se usó el score$=-ERM$, pero para representar los datos se ha decidido usar $RMSE = \sqrt{ERM}$ ya que tiene una interpretación más amigable.

En la primera gráfica podemos ver la curva de aprendizaje, en donde se muestra el desempeño del modelo en los conjuntos de validación y entrenamiento. Como cabe esperar, con pocos ejemplos de entrenamiento, se obtienen Errores de validación más altos. Se observa como conforme crece el número de muestras el error de validación tiende a disminuir, sacrificando a su vez en baja medida el error de entrenamiento:

$$E_{train} \uparrow \quad E_{val} \downarrow$$

El sombreado verde y roja hacen referencia a los máximos y mínimos que se han obtenido durante cada entrenamiento con cada número de ejemplos, es decir, la desviación.

En la segunda gráfica se ve el número de entrenamientos necesarios en función de las muestras de entrenamiento. Por último, en la tercera gráfica se ve como disminuye muy rápidamente el error conforme aumentamos el número de entrenamientos.

## 1.9. Análisis de resultados

Vamos ahora a analizar los resultados obtenidos para los mejores modelos hallados en validación cruzada. Una vez encontrados los mejores parámetros para cada modelo, se han reentrenado sobre todos los datos de entrenamiento sin hacer validación cruzada, con el objetivo de mejorar la calidad del mismo. Puesto que los hiperparámetros que usaremos son los mejores obtenidos y al tener un tamaño muestral mayor para entrenarlos, en principio tendremos una mejor aproximación. Además, recordemos que hemos reservado un conjunto de datos para test, el cual nos servirá para cotejar el desempeño de los modelos frente a nuevas muestras independientes e idénticamente distribuídas no utilizadas anteriormente para el entrenamiento, lo que nos servirá para proporcionar una estimación bastante aceptable del error fuera de la muestra $E_{out}$.

En el Cuadro 1.4 podemos ver los mismos datos que en el Cuadro 1.3, a los que les hemos añadido los errores en entrenamiento y test, habiéndolos reentrenado con todos los datos como se ha dicho.

| **Modelo** | **Mejores hiperp.CV** | $RMSE_{val}$ | $RMSE_{train}$ | $R^2_{train}$ | $RMSE_{test}$ | $R^2_{test}$ |
|---|---|---|---|---|---|---|
| Regresión lineal | $\alpha = 0.01$ | 0.0080 | 0.0080 | 0.8798 | 0.0080 | 0.8803 |
| MLP | $\alpha = 0.01$ | 0.0081 | 0.0079 | 0.8819 | 0.0080 | 0.8808 |
| SVR | $C = 0.001$, $\epsilon = 0.001$ | 0.0077 | 0.0080 | 0.8810 | 0.0082 | 0.8741 |
| AdaBoost | $lr = 0.05$ | 0.0097 | 0.0098 | 0.8208 | 0.0098 | 0.8219 |
| Random Forest | $n\_estimators = 200$ | 0.0057 | 0.0030 | 0.9829 | 0.054 | 0.9449 |
| RBF | $\alpha = 0.01$, $\gamma = 3$, $k = 32$ | 0.0103 | 0.0104 | 0.7972 | 0.0104 | 0.7973 |

**Cuadro 1.4:** Resultados de todos los modelos

Los resultados obtenidos muestran claras diferencias entre los modelos empleados. Comenzaremos nuestro análisis haciendo notar que ningún modelo tiene flagrantes muestras de sobreajuste. Se puede pensar que hay algo de sobreajuste en Random Forest, el cual ha sido el modelo que más empeora en test con respecto a entrenamiento, pero aun así, la diferencia tampoco es demasiado grande. Nuestro hiperparámetro a ajustar en Random Forest era el número de árboles a emplear, y el modelo ha seleccionado el máximo número posible de entre los propuestos. Este hecho sumado a que no controlamos de forma efectiva la regularización de los árboles en particular pueden ser dos causas por las que se produce este fenómeno. Se podría hacer un análisis incorporando otro parámetro que controlara el crecimiento de los árboles, pero se deja para posteriores estudios. Hay por el contrario otros modelos en donde el error en test ha sido algo mejor que el error en entrenamiento, como puede ocurrir con regresión lineal. De todas formas, en general y para todos los modelos, los valores en entrenamiento y validación han sido muy parejos.

El modelo que mejores resultados ha conseguido dar tanto en entrenamiento, validación cruzada y test ha sido el Random Forest. Como ya hemos dicho es igualmente el modelo en el que más sube el error de test, pero aun así sigue siendo el mejor de todos los modelos en test. Puesto que ha dado el mejor resultado sobre el conjunto de test, que como hemos dicho supone una muestra tomada de forma independiente e idénticamente distribuida con respecto a los datos de entrenamiento, $E_{test}$, será una buena estimación de $E_{out}$. Debido a que no estamos tratando con un problema de clasificación, no podemos utilizar las cotas de la desigualdad de Hoeffding ni las de la dimensión de Vapnik-Chervonenkis para poder acotar el error, por lo que utilizaremos este error exclusivamente como estimación de $E_{out}$

Un hecho a apreciar es que todos los modelos en general también han dado buenos resultados. Cogiendo el modelo de regresión lineal como base, el cual tiene un RMSE de partida bastante bajo tanto en términos absolutos como relativos respecto a la varianza de las etiquetas tal y como se puede ver en las medidas de $R^2$, los modelos de MLP y Random Forest han dado unos resultados mejores, que se podrían considerar como muy buenos. Los modelos SVR, AdaBoost y RBF aunque reporten resultados ligeramente peores que la regresión lineal, aportan resultados bastante decentes.

Puesto que la regresión lineal da un valor predicho bastante bueno, quiere decir que las características que utilizamos son muy informativas, y un hiperplano sobre el espacio de las mismas es capaz de hacer muy buena clasificación. Para entender mejor la importancia de las variables que hemos creado frente a la variables originales, vamos a estudiar el valor de los pesos asignados por la regresión lineal. Cuanto mayores sean los pesos, mayor será la importancia relativa de cada una de esas variables. Mostrándolos en un diagrama de barras, podemos ver los resultados en la figura 1.12

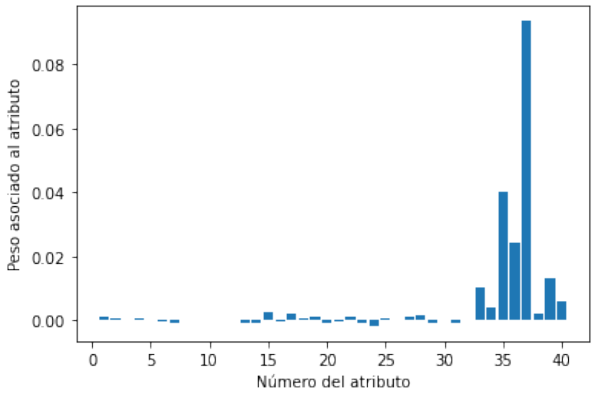



**Figura 1.12:** Pesos obtenidos por nuestra mejor hipótesis de regresión lineal

Como podemos comprobar, las características que hemos generado tienen pesos muchísimos más grandes en valor absoluto que los asociados a las variables originales. Además estos pesos nuevos son positivos, lo que quiere decir que a mayor dispersión y a mayor uniformidad, mayor será la potencia generada.

El que los pesos asociados a nuestras variables sean tan grandes nos hace pensar que un modelo de regresión lineal entrenado sin estas características proporcionaría unos resultados mucho peores. Tiene por tanto interés estudiar qué pasaría en dicho caso, y para ello hemos ejecutado adicionalmente y meramente a título comparativo una regresión lineal con el mismo procedimiento de validación cruzada que el explicado en el apartado anterior, pero utilizando sólo las características originales. Los resultados los podemos ver en el Cuadro 1.5

| **Modelo** | **Mejores hiperp. CV** | $RMSE_{val}$ | $RMSE_{train}$ | $R^2_{train}$ | $RMSE_{test}$ | $R^2_{test}$ |
|---|---|---|---|---|---|---|
| Regresión Lineal originales | $\alpha = 10$ | 0.0215 | 0.0215 | 0.1328 | 0.0217 | 0.1222 |
| Regresión Lineal | $\alpha = 0.01$ | 0.0080 | 0.0080 | 0.8798 | 0.0080 | 0.8803 |

**Cuadro 1.5:** Resultados de ambos modelos de regresión lineales ejecutados, siendo el de la primera línea un modelo de regresión lineal sobre las caracterísicas originales, y el segundo el modelo de regresión lineal con las características ampliadas

Tal y como se esperaba, la diferencia es muy muy significativa. Tanto el mejor valor del hiperparámetro a ajustar como los propios valores de errores son abismalmente distintos, y es

evidente que las características que hemos empleado han sido relevantes muy muy relevantes de cara a obtener unos modelos de tal alta calidad.

Por otro lado, el modelo SVR da unos resultados similares a la regresión lineal puesto que hemos limitado el número de iteraciones a 8000. En caso de no haberlo hecho probablemente hubiera sido capaz de superar a la regresión lineal. Sin embargo es el modelo que más ha tardado en ejecutar en validación cruzada con muchísima diferencia, por lo que en nuestro caso no nos ha resultado factible entrenarlo utilizando más iteraciones.

Tanto AdaBoost como RBF nos proporcionan unos resultados peores que Regresión Lineal. Respecto a AdaBoost, probablemente el número de modelos a ajustar haya sido pequeño, o puede que cada uno de ellos no haya sido lo suficientemente complejos. En el caso de RBF, pese a la comprobación de 27 posibles combianciones de parámetros, puede que haya alguna mejor combinación de los tres que no haya sido probada. Cada uno de los parámetros a a justar sólo podía tomar 3 posibles valores, lo que en sí es un valor pequeño. También puede ocurrir que otra forma de calcular $r$ proporcione mejores resultados.

Sin embargo, también es posible simplemente que ambos modelos no sean tan buenos modelos como los demás para este contexto concreto. Por el teorema de No Free Lunch siempre existirán problemas para los que un modelo será mejor que otros. Por tanto puede ocurrir que simplemente estas dos aproximaciones no sean tan buenas ni se presten a tener tan buenos resultados respecto a los demás por la propia estructura del problema.

## 1.10. Análisis del mejor modelo

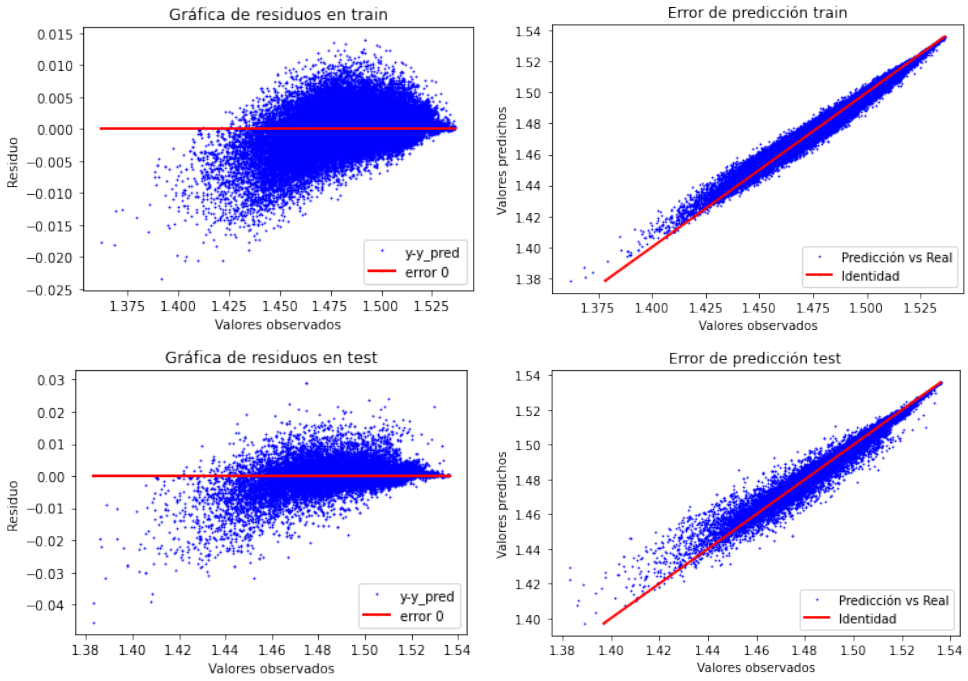



**Figura 1.13:** Gráficas que muestran la bondad de la hipótesis final. Las gráficas de la izquierda vemos los residuos. Muestra la diferencia entre todos los $h(x_i) - y_i$. Por otro lado en la gráfica de la derecha vemos la misma información pero con otro enfoque: comparamos el valor estimado frente al real. Lo ideal sería que en las gráficas de la izquierda los puntos estuviesen en la recta $ejeY = 0$ definida por la línea roja y que en las gráficas de la derecha los puntos se acumulasen en torno a la recta identidad marcada en rojo, que implicaría que $h(x_i) \approx y_i$

Como ya hemos dicho, nuestro mejor modelo ha sido Random Forest. Vamos a realizar dos gráficas muy pertinentes que nos van a permitir ver cual es la bondad de nuestro mejor modelo. En estas imágenes vamos a mostrar cuáles son los valores observados frente a los residuos y a los valores predichos. Los resultados los podemos observar en la Figura 1.13

En base a la Figura 1.13 vemos que la estimación sigue la tendencia de lo que sería el estimador ideal: viene muy bien reflejado en las gráficas de error de predicción. Por otra parte vemos que los residuos se distribuyen entorno a unas cotas claras, estando todos los residuos en un rango de valores bastante pequeño.

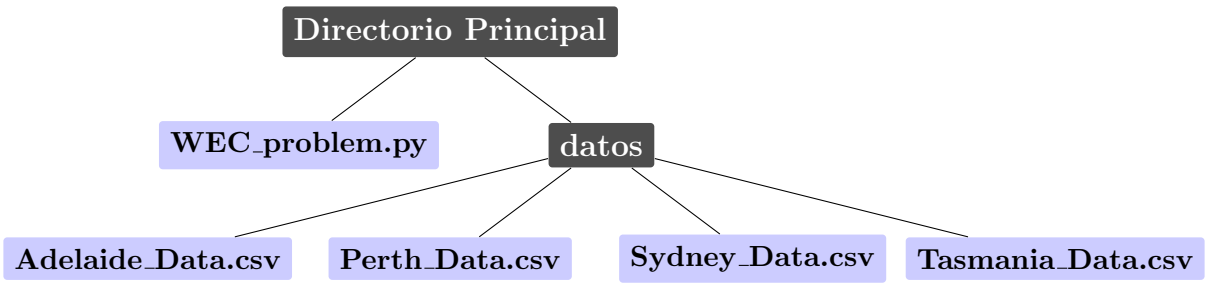
# Anexo I

# Código: Guía de usuario

En este apéndice se tratará todo lo referente al código de los ficheros Python de cada uno de los problemas. Ambos problemas comparten muchas similitudes, lo que hará más simple la compresión en su conjunto.

## 2.1. Estructura de directorios

Tener la siguiente estructura de ficheros




En caso de no tener esta estructura y no querer adaptarla, puedes adaptar el `path` de los archivos de datos modificando la variable global:

`path`

que se encuentra después de la importación de librerías.

## 2.2. Estructura del código

Ambos códigos poseen una estructura similar. El código está dividido por apartados, en cada apartado se implementan una serie de funciones que tienen una funcionalidad común. Estos apartados son:

1. **Variables Globales**: aquí tendremos variables que determinan la ejecución del problema, en concreto tenemos las siguientes variables en ambos problemas:

   *a*) `SEED`: Semilla utilizada para funciones aleatorias. Incrustada al inicio dentro de la función `np.random.seed`.

   *b*) `MOSTRAR_GRAFICOS_TSNE`: variable booleana que determinará si se quiere ejecutar la técnica de **t-SNE** o no.

   *c*) `MOSTRAR_CURVA_DE_APRENDIZAJE`: variable booleana que determinará si se quiere ejecutar la función de **learning_curve** o no.

2. **Funciones y Clases auxiliares**: Estas funciones serán de utilidad en el desarrollo del preprocesado para los datos. En particular, todas las funciones auxiliares que aparecen más abajo están dedicadas a la generación de características. Además se incluirán funciones auxiliares como wait() para pausar la ejecución

3. **Visualizacion de datos**: contiene todas las funciones que muestran resultados y gráficas.

4. **Lectura de Datos**: alberga a la función `read_data` de donde se leen los datos.

5. **Tratado de Datos**: Funciones para añadir características y preprocesar datos

6. **Ejecución**: En esta parte se incluyen las órdenes que ejecutan toda el código

## 2.3. Variables de modificación de ejecución

Las variables siguientes están por defecto en `False`, pero se pueden cambiar a `True` si así se desea con el correspondiente pago en tiempo de ejecución:

1. `MOSTRAR_GRAFICOS_TSNE`: variable booleana que determinará si se quiere ejecutar la técnica de **t-SNE** o no.

2. `MOSTRAR_CURVA_DE_APRENDIZAJE`: variable booleana que determinará si se quiere ejecutar la función de **learning_curve** o no.

son importantes ya que de ellas depende un gran cambio en el tiempo de ejecución del programa: pueden conllevar horas.

# Bibliografía


[1] V. Cherkassky and F. M. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley-IEEE Press, 2007.

[2] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[3] Scikit-Learn. scikit-learn: machine learning in python. `https://scikit-learn.org/`. Accessed: 2021-06-12.

[4] Wikipedia. Coeficiente de determinación lineal $r^2$. `https://es.wikipedia.org/wiki/Coeficiente_de_determinaci%C3%B3n`. Accessed: 2021-06-12.

[5] Wikipedia. Error cuadrático medio. `https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio`. Accessed: 2021-06-12.