Universidad de Granada

Trabajo 3: Programación

Aprendizaje Automático

Federico Rafael García García

Curso 2018/2019

20 de mayo de 2019

Índice

1. Dígitos manuscritos **2**

1.1. Particiones training y test 2

1.2. Visualización y preprocesado de datos 2

1.3. Funciones a usar 6

1.4. Entrenamiento y test 6

1.5. Conclusiones 7

2. Airfoil Self-Noise Data Set **7**

2.1. Particiones training y test 7

2.2. Visualización y preprocesado de datos 7

2.3. Funciones a usar 15

2.4. Entrenamiento y test 16

2.5. Conclusiones 16

2.6. EXTRA: Ajuste no lineal 18

3. Bibliografía **20**

1. Dígitos manuscritos

1.1. Particiones training y test

La base de datos de dígitos manuscritos consta de dos ficheros de texto, uno para *training* (optdigits.tra) y otro para *test* (optdigits.tes), por lo que no hará falta particionar los datos al ya venir preparados. Los datos fueron obtenidos a través de los siguientes enlaces:

`https://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/optdigits.tes`

`https://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/optdigits.tra`

1.2. Visualización y preprocesado de datos

Cada línea de cada fichero consta de 65 valores enteros separados por comas. Los 64 primeros son las características, mientras que el último es la etiqueta: $0, 1, ..., 9$. Las 64 características representan una matriz 8x8 con valores en [0-16]: son imágenes de los dígitos manuscritos en escala de gris.

Con el fin de entender mejor los datos con los que se trabajará, se ha decidido seleccionar 10 elementos de cada dígito y visualizarlos. Para ello se ha transformado cada vector de 64 características en una matriz 8x8, y cada uno se ha copiado a una región de una matriz 80x80. En la figura 1 se pueden ver 100 dígitos manuscritos.

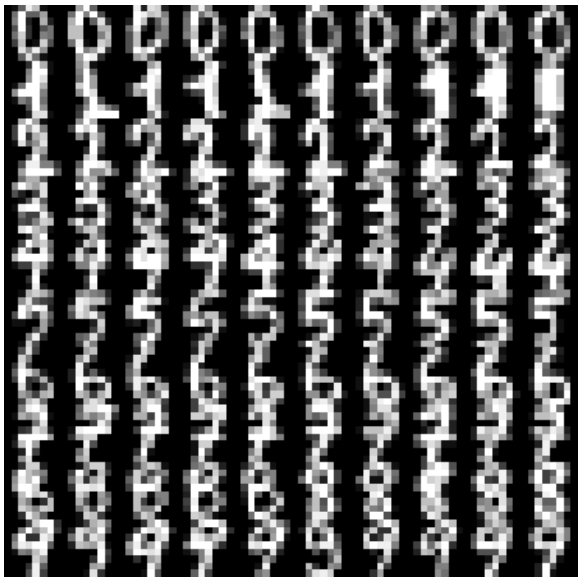

Figura 1: Representación gráfica de 100 dígitos.

Dado que el espacio es de 64 dimensiones, se ha decidido buscar alguna transformación que permita reducirlo, con las ventajas que ello significa: entrenamiento más rápido, mejor representación de los datos y un modelo más simplificado y claro de entender. Al poder

ser los datos representados como imágenes, se ha decidido utilizar una transformación que utilice las siguientes características:

- *Simetría vertical*: Se divide la matriz en dos mitades de 8x4, se restan, se obtiene el absoluto y se suman todos su valores. Cuanto menor sea el resultado, mayor es la simetría.
- *Intensidad superior*: Se divide la matriz en la primera mitad de 4x8 y se suman todos los valores.
- *Intensidad inferior*: Se divide la matriz en la segunda mitad de 4x8 y se suman todos los valores.

Se han elegido estas características ya que pueden diferenciar números simétricos (0, 1, 8, ...) de aquellos que no lo son (2, 3, 4 , ...) y en el caso de que haya el mismo nivel de simetría vertical, diferenciarlos según qué número tiene más intensidad en la parte superior o inferior (como en el caso del 6 y el 9). Se ha realizado por tanto la siguiente transformación de dimensión: $\mathbb{Z}^{64} \rightarrow \mathbb{Z}^3$.

Usando la función `Plot3D`, se pueden representar puntos 3D con diferente color según su etiqueta. En la figura 2 se pueden ver los datos desde diferentes perspectivas.

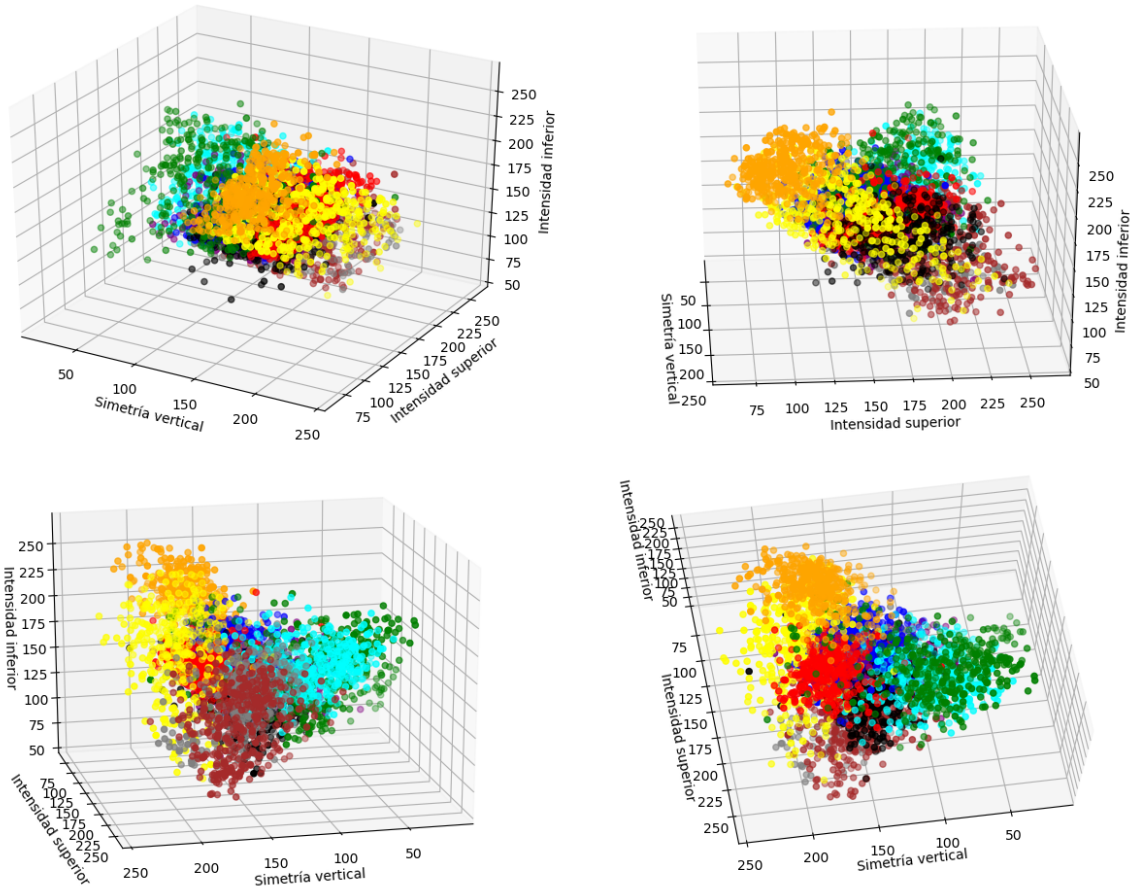

Figura 2: Plot 3D de la transformación a simetría, intensidad superior e inferior.

Se aprecian diferentes *clusters*, pero algunos no están bien diferenciados del resto.

Para comprobar que la transformación es de calidad y no se ha reducido la caracterización de los dígitos que permite diferenciarlos, se ha utilizado el algoritmo *Knn Leave One Out* con $k = 1$. En este algoritmo de clasificación, dado un elemento del conjunto se compara con el resto, buscando el elemento más similar, dado que probablemente su etiqueta sea igual al del más parecido. Una vez se ha encontrado, se comparan sus etiquetas; si son iguales, se tiene un acierto, y en caso contrario un fallo. El número de aciertos indicará si se es posible identificar un dígito del resto. No utilizamos el conjunto de test, con el fin de evitar influenciar el entrenamiento de alguna manera (*bias*), razón por la cual usamos la técnica *Leave One Out* para poder usar el propio conjunto de entrenamiento. Para acelerar los cálculos, se ha escogido el 10 % (en proporción para cada etiqueta) de los datos para ser comparado con todos los datos.

Para la comparación de un dígito con otro, se resta una a una las características de uno con el otro, se obtiene sus valores absolutos y finalmente se suman; cuanto menor sea este valor, existe mayor probabilidad de que los elementos tengan la misma etiqueta.

La ejecución de *Knn* con el conjunto sin transformar, $\mathbb{Z}^{64}$, ha resultado en un 97.08 % de aciertos; con el conjunto transformado a $\mathbb{Z}^3$ se ha obtenido una tasa de acierto del 42.18 %. Claramente, la transformación es pésima y no debería ser usada.

Se ha decidido utilizar otra transformación, de dimensión 16: cada característica es la suma de todos los valores de una fila o de una columna. De esta manera, se está comprimiendo toda la intensidad en una única fila o en una única columna. La ejecución de *Knn* con este nuevo conjunto transformado en $\mathbb{Z}^{16}$ ha resultado en un 95.23 % de aciertos, un 1.85 % menos que los datos sin transformar. A cambio de un pequeño incremento del error, se ha reducido considerablemente la complejidad del problema.

Los datos pueden ser visualizados con dos histogramas. En la figura 3 se puede ver la media de las características para todos los elementos de cada dígito: en azul se ve la suma de intensidad por fila y en rojo por columna. Por ejemplo, en el caso del dígito 1, puesto que se trata de un trazo vertical, la suma de intensidad de cada fila será similar; por ello las barras en azul tienen alturas similares. Al ser un único trazo vertical, las columnas centrales tendrán la mayor suma de intensidad, como se puede ver reflejado en las barras rojas.

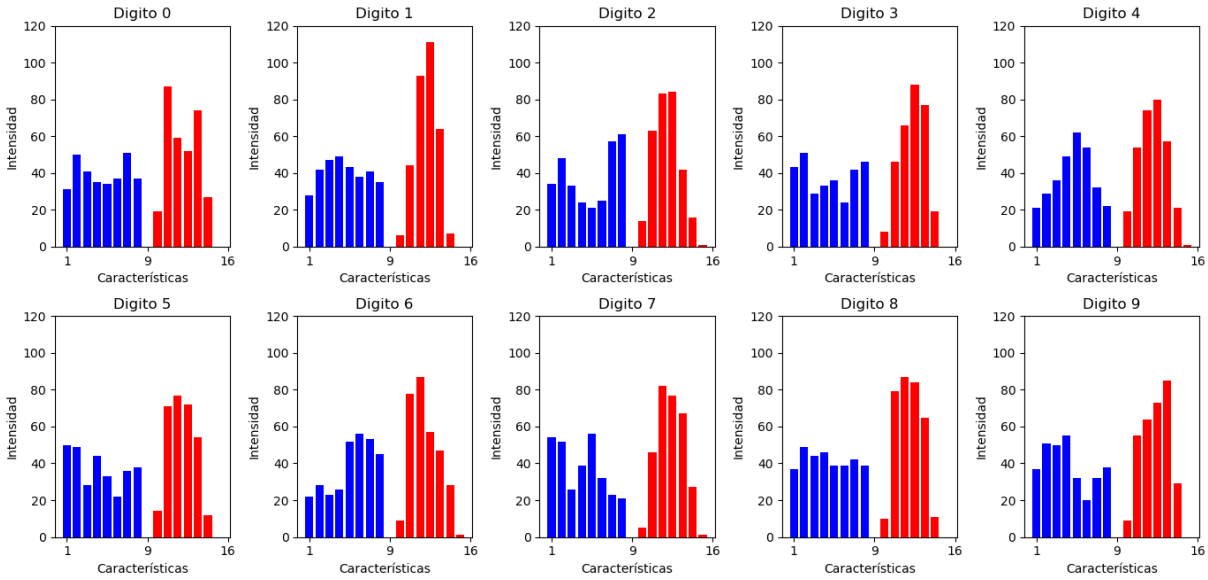

Figura 3: Media de características en $\mathbb{Z}^{16}$ de todos los elementos de cada dígito. Las barras en azul representan las primeras 8 características, la suma de intensidades de cada fila. Las barras en rojo representan las últimas 8 características, la suma de intensidades de cada columna.

No se normalizaron las características, puesto que todas ellas tienen el mismo rango de valores.

Finalmente, se ha decidido reducir la dimensionalidad todavía más mediante eliminación, quitando características que no reduzcan la tasa de aciertos de *Knn*; se pretende eliminar las características que no sean "importantes". En nuestro caso, serían las casillas de la fila o columna suma de intensidad que siempre tienen los mismos valores para todos los dígitos.

Para determinar las características importantes de las que no, se ha utilizado la *varianza de la media de cada característica de todos los dígitos*. El proceso es el siguiente:

1. Para cada conjunto transformado de cada dígito, hacer la media de cada característica. Por ejemplo, para el conjunto de todos los elementos etiquetados como 0, obtener la media de la primera característica, de la segunda, etc.

2. Hacer la varianza de todas las medias de una característica. Por ejemplo, dada la media de la primera característica del conjunto 0, y el conjunto 1, etc. obtener la varianza.

Cada varianza nos permite saber qué tan diferente son las medias de una característica. Si la varianza de una característica es baja, significa que todos los dígitos tienen valores casi iguales para esa característica. Esto significa que no ayuda a distinguir un dígito de otro. Estas características pueden ser eliminadas.

Dado que no existe una clara definición de lo que es una *varianza baja*, esta se ha hallado mediante experimentación. Se ha determinado que eliminar características con

varianza menor a 10 no afecta la tasa de aciertos de *Knn*. De esta manera, se ha conseguido reducir la dimensión de 16 a 14. Se eliminaron las características 9 y 16: la primera y última característica del histograma de columnas. Se puede ver en el gráfico que los valores de las características 9 y 16 son cero o extremadamente bajos.

1.3. Funciones a usar

Al tener clústers claramente diferenciables (según los resultados obtenidos con *KNN-LOU*), se utilizarán funciones lineales para clasificar los dígitos: se obtendrán hiperplanos que permitan separar clústers entre sí. Para ello se utilizará un clasificador de Regresión Logística Multinomial. Usamos Regresión Logística para obtener la probabilidad de que un elemento pertenezca a una clase u otra, es decir, que sea un dígito u otro, y Multinomial al haber múltiples clases. Cada elemento será clasificado con la clase que tenga mayor probabilidad.

En el clasificador se utilizará el algoritmo de *newton-cg*, basado en el "Método de Newton".

No se ha utilizado regularización, por lo que el parámetro `C`, que define la inversa de la fuerza de regularización, se ha dejado en su valor por defecto de 1. Se ha decidido así puesto que se asume una gran variabilidad en los manuscritos digitalizados: la forma de escritura es muy personal, y será muy diferente de un individuo a otro. Por ello asumimos que no ocurrirá *overfitting*.

1.4. Entrenamiento y test

Se ha entrenado ejecutando las funciones de *SK Learn* con el conjunto de entrenamiento transformado y reducido (*xf_train*) y el conjunto de etiquetas (*y_train*):

```
softReg = LogisticRegression(multi_class='multinomial',
          solver='newton-cg', max_iter=200)

softReg.fit(xf_train, y_train)
```

Se ha entrenado con 200 iteraciones, puesto que con una cantidad mayor no se han obtenido mejor resultados (no se ha reducido E_{in}).

Luego se ha transformado y reducido el conjunto de test y se han obtenido las predicciones: un array con la clase predecida de cada elemento del conjunto de test *xf_test*:

```
predicciones = softReg.predict(xf_test)
```

Al tener predicciones de etiquetas que se pueden clasificar como "correcta" o "incorrecta", el error es MAE; en un bucle se ha comparado cada clase predecida con la etiqueta del elemento. Con ello se ha obtenido el error; la tasa de acierto es por tanto $(1 - E) \times 100$.

En el caso de E_{out}, este ha sido 0.09961 (tasa de acierto de casi 90.04 %).

En el caso de E_{in}, este ha sido 0.08553 (tasa de acierto de casi 91.45 %).

1.5. Conclusiones

Se puede ver el gran efecto que puede tener el preprocesado en la clasificación: se ha de ser cuidadoso con las transformaciones, ya que se puede perder información relevante que permita diferenciar unas clases de otras. En caso de éxito, se puede reducir enormemente el número de características, con las ventajas que ello conlleva. Se ha conseguido obtener un modelo simple con clústers muy diferenciables, lo cual ha permitido utilizar un modelo lineal para clasificar.

Los errores E_{in} y E_{out} son bajos, y con poca diferencia: esto da a entender que se ha obtenido un clasificador de calidad. Se ha conseguido entender la importancia de las intensidades de la imagen de los dígitos manuscritos, lo cual ha llevado a estos resultados positivos.

2. Airfoil Self-Noise Data Set

2.1. Particiones training y test

La base de datos de "Airfoil Self-Noise" consta de un único fichero de texto (airfoil_self_noise.dat). Por ello se particionarán los datos en dos conjuntos de *training* y *test*. El primero constará del 80 % de los datos y el segundo del 20 %, porcentajes bastante comunes; se prefiere un tamaño de *training* mayor para poder ajustar los datos lo mejor posible. Los datos serán barajados antes de realizar las particiones.

Los datos fueron obtenidos a través del siguiente enlace:

`https://archive.ics.uci.edu/ml/machine-learning-databases/00291/airfoil_self_noise.dat`

2.2. Visualización y preprocesado de datos

Cada línea de cada fichero consta de 6 valores reales separados por espacios. Los primeros 5 representan la entrada, y fueron obtenidos de experimentaciones con alas de aeronaves en túneles de viento para diferentes ángulos de ataque y velocidad del viento. Cada uno de estos valores representa:

1. Frecuencia (Hz).
2. Ángulo de ataque (grados).
3. Longitud de cuerda (metros).
4. Velocidad de corriente libre (metros por segundo).

5. Espesor de desplazamiento lateral de succión (metros).

El sexto valor, la salida, representa el nivel de presión del sonido (en decibelios).

Se pretende por tanto predecir el nivel de presión del sonido que se producirá según el ángulo del ala y la velocidad del aire, entre otras características.

Las características tienen diferentes rangos de valores, por lo que se normalizarán los datos entre 0 y 1.

Al no tener conocimientos de aeronáutica, no se transformarán los datos, y además al haber pocas características no se reducirá la dimensión.

Debido a la dificultad para visualizar los datos (6 características de las que no conocemos ningún tipo de relación entre ellas o su interpretación), graficamos en 3D cada pareja de características junto con la salida (los datos se grafican sin normalizar, la normalización se aplicará en el ajuste). Tenemos por tanto 10 combinaciones distintas. Desde la figura 4 hasta la 13 se ven las gráficas.

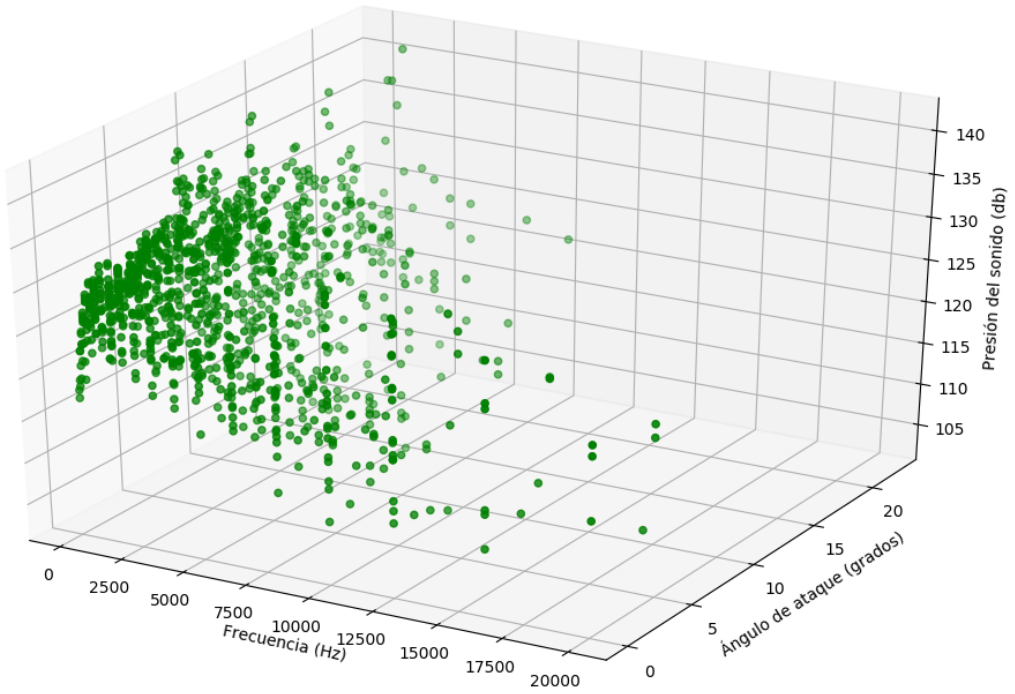

Figura 4: Frecuencia Vs. Ángulo de ataque

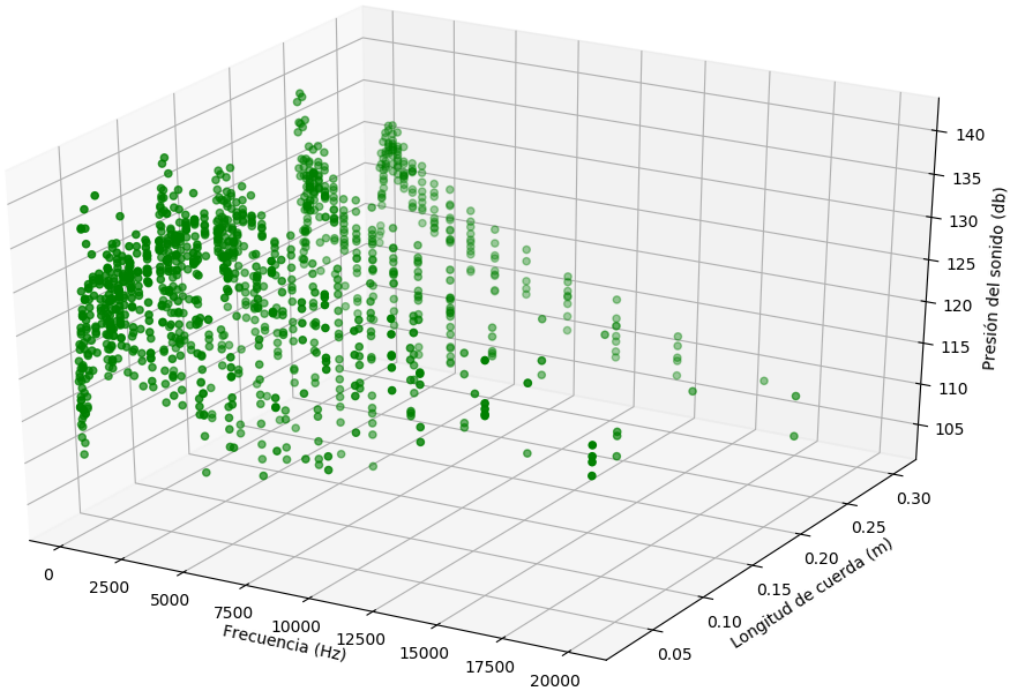

Figura 5: Frecuencia Vs. Longitud de cuerda

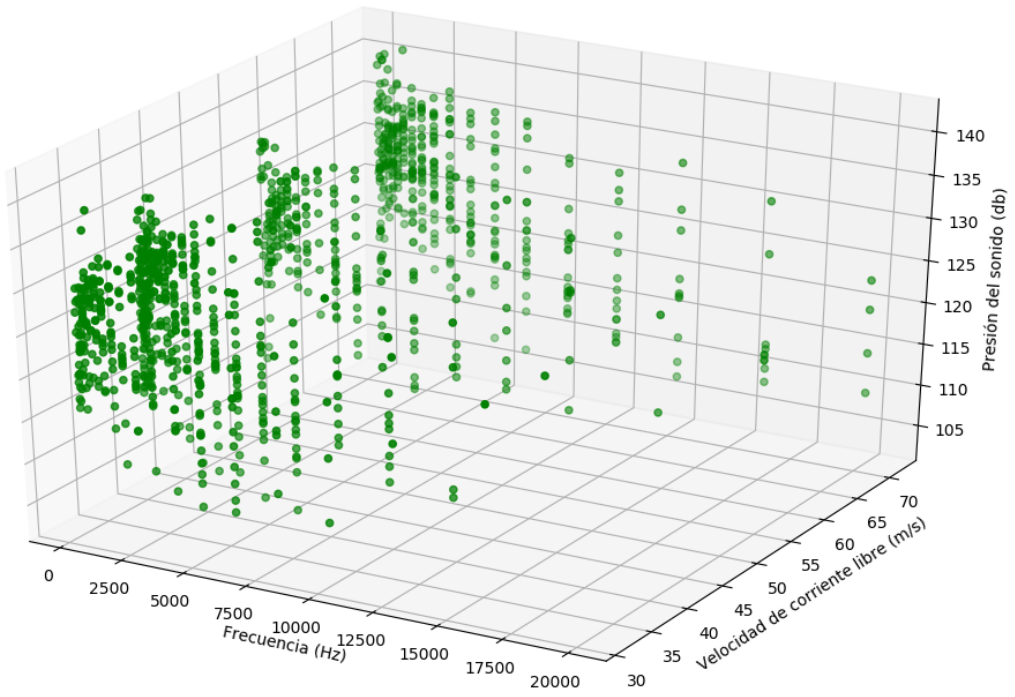

Figura 6: Frecuencia Vs. Velocidad de corriente libre

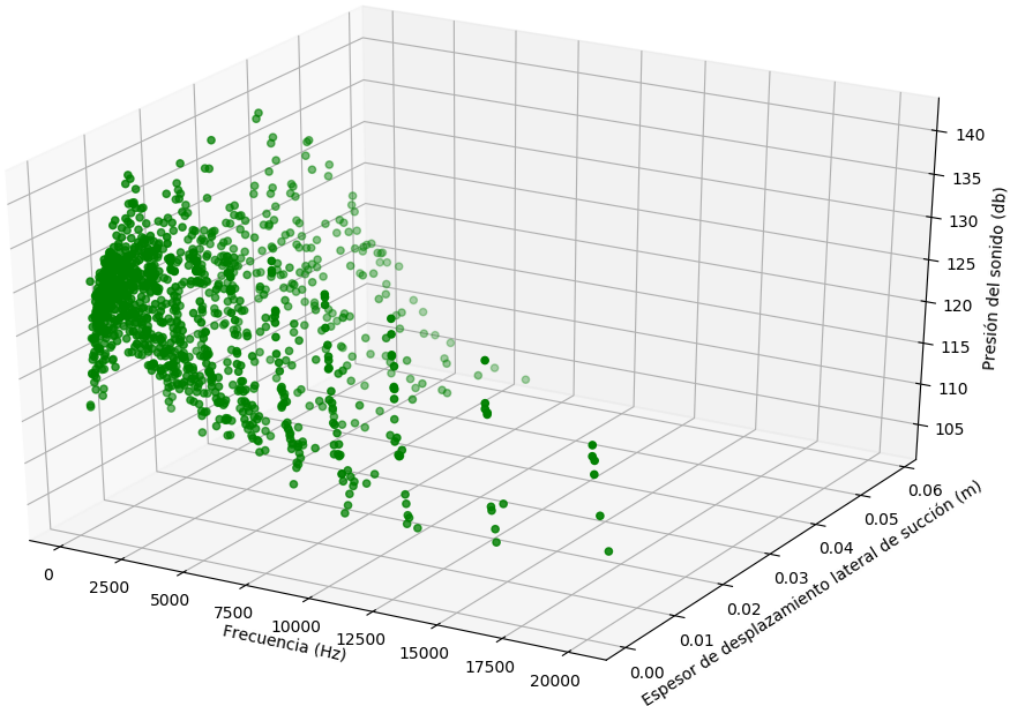

Figura 7: Frecuencia Vs. Espesor de desplazamiento lateral de succión

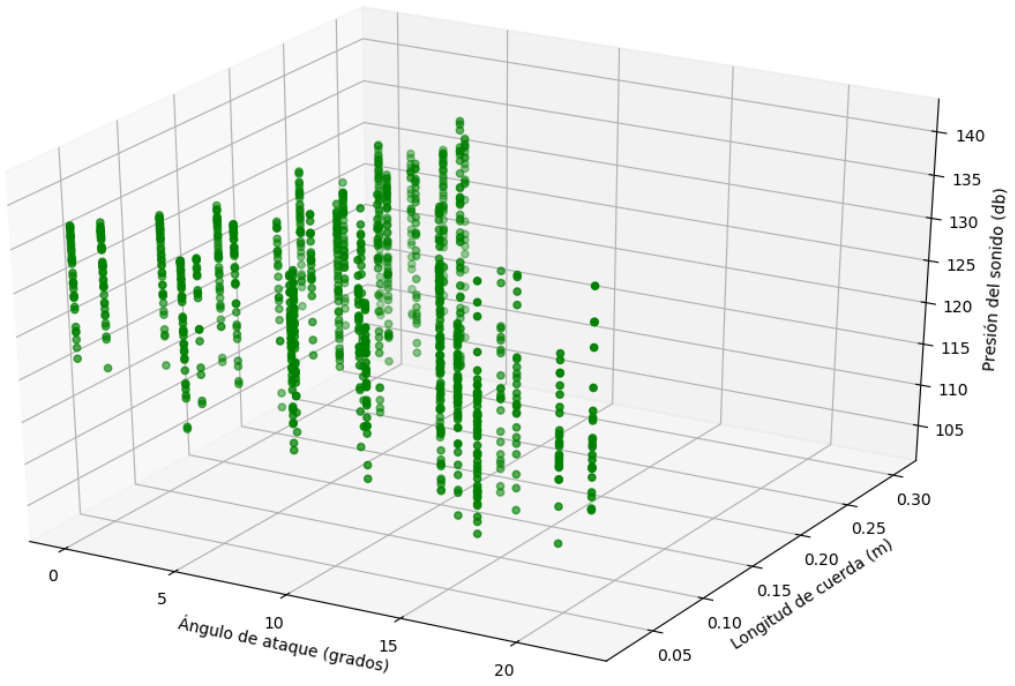

Figura 8: Ángulo de ataque Vs. Longitud de cuerda

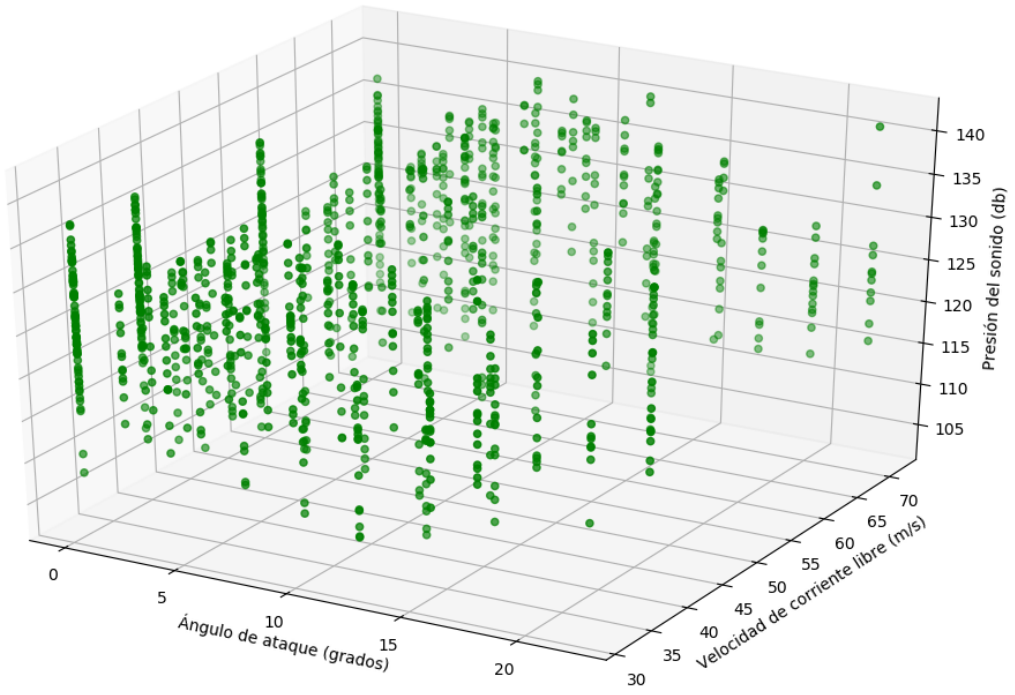

Figura 9: Ángulo de ataque Vs. Velocidad de corriente libre

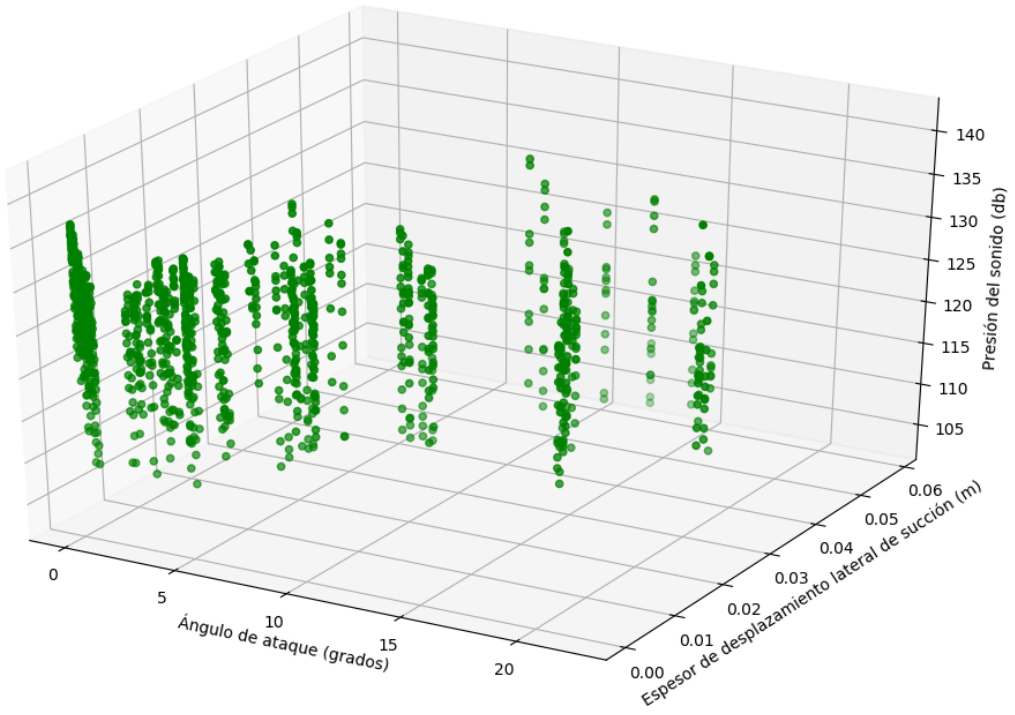

Figura 10: Ángulo de ataque Vs. Espesor de desplazamiento lateral de succión

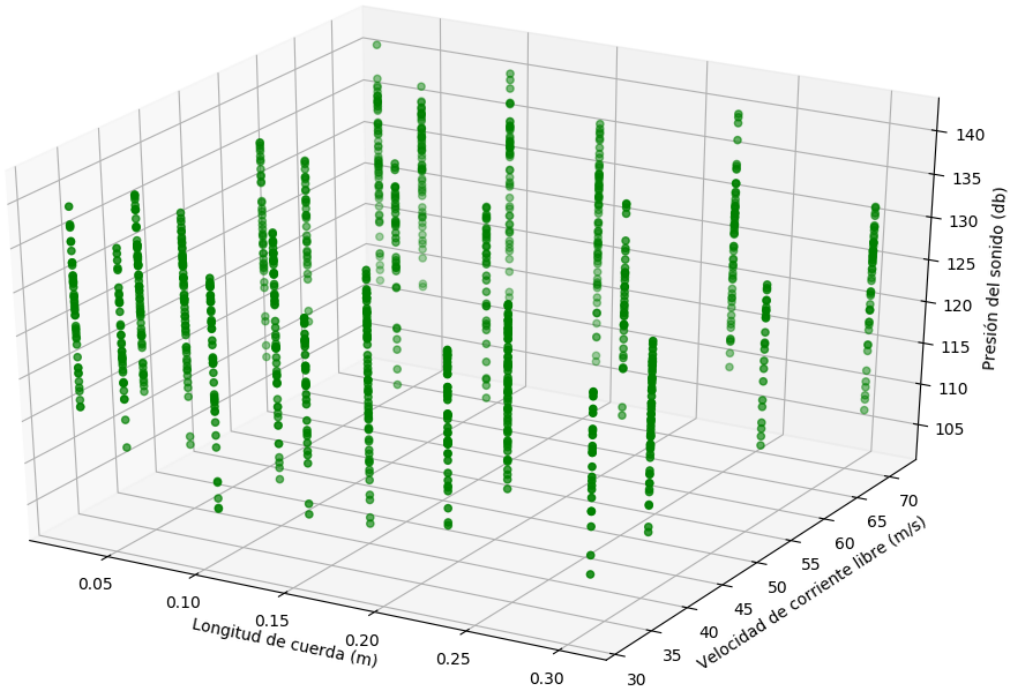

Figura 11: Longitud de Cuerda Vs. Velocidad de corriente libre

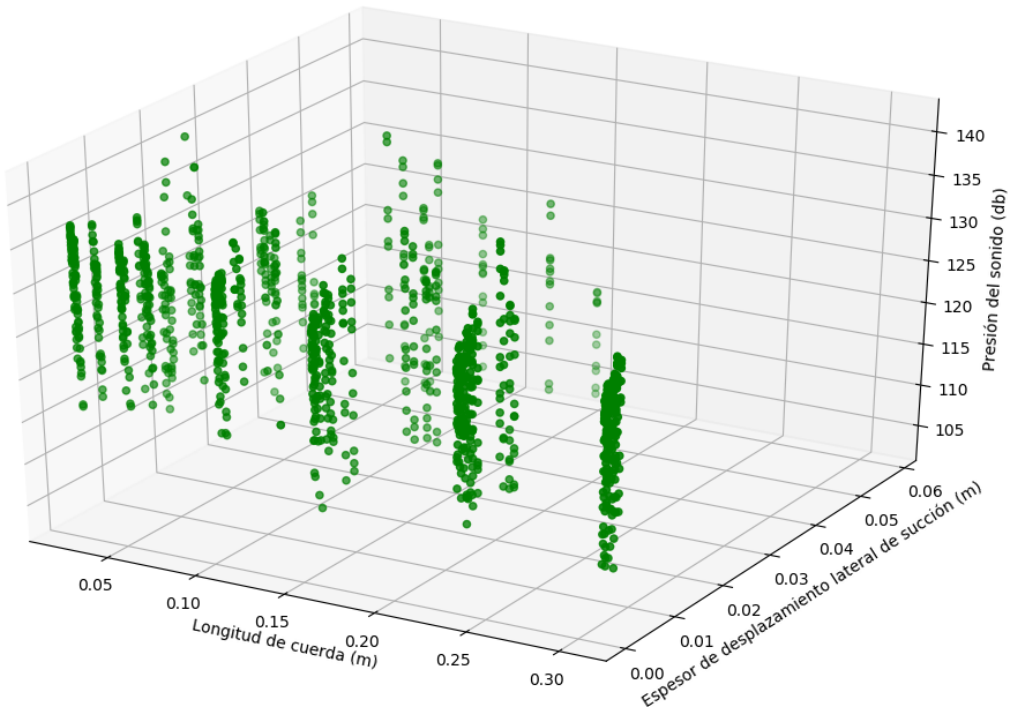

Figura 12: Longitud de cuerda Vs. Espesor de desplazamiento lateral de succión

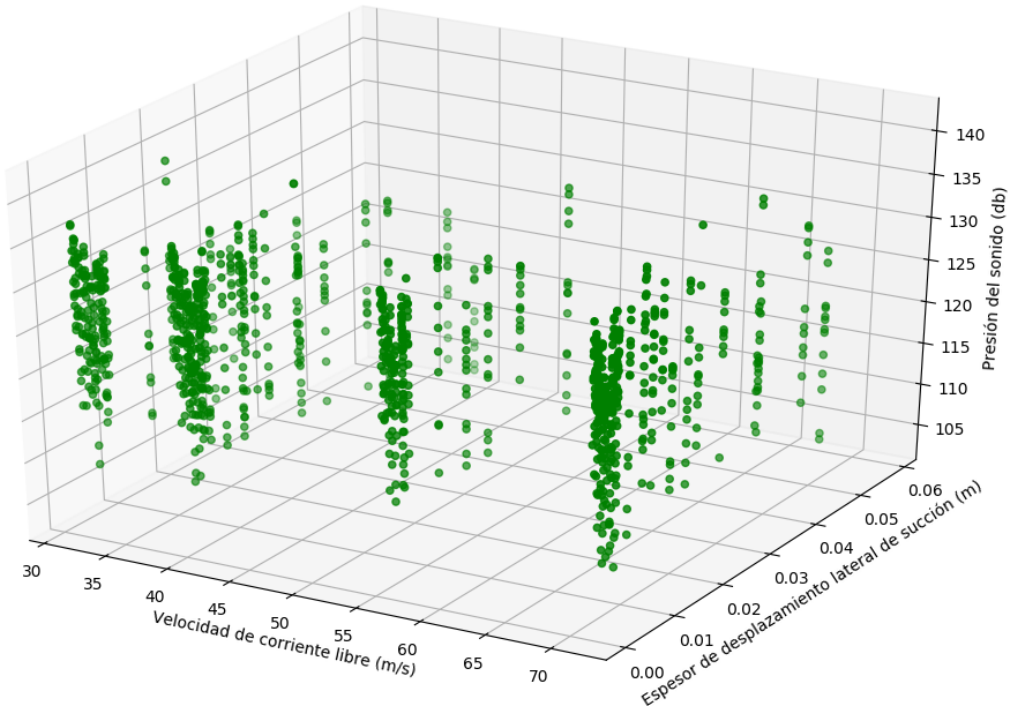

Figura 13: Velocidad de corriente libre Vs. Espesor de desplazamiento lateral de succión

Se pueden apreciar algunas distribuciones y tendencias a formar ciertos cúmulos, como es el caso de la figura 7 (Frecuencia Vs. Espesor de desplazamiento lateral de succión), pero los datos no son fáciles de interpretar. En algunas de las características los valores usados no se distribuyen a lo largo de un intervalo, si no que son pocos valores de un pequeño conjunto que se repiten, como es el caso de la "Velocidad de corriente libre".

Graficamos únicamente la salida: en el eje X cada elemento y en el eje Y su salida.

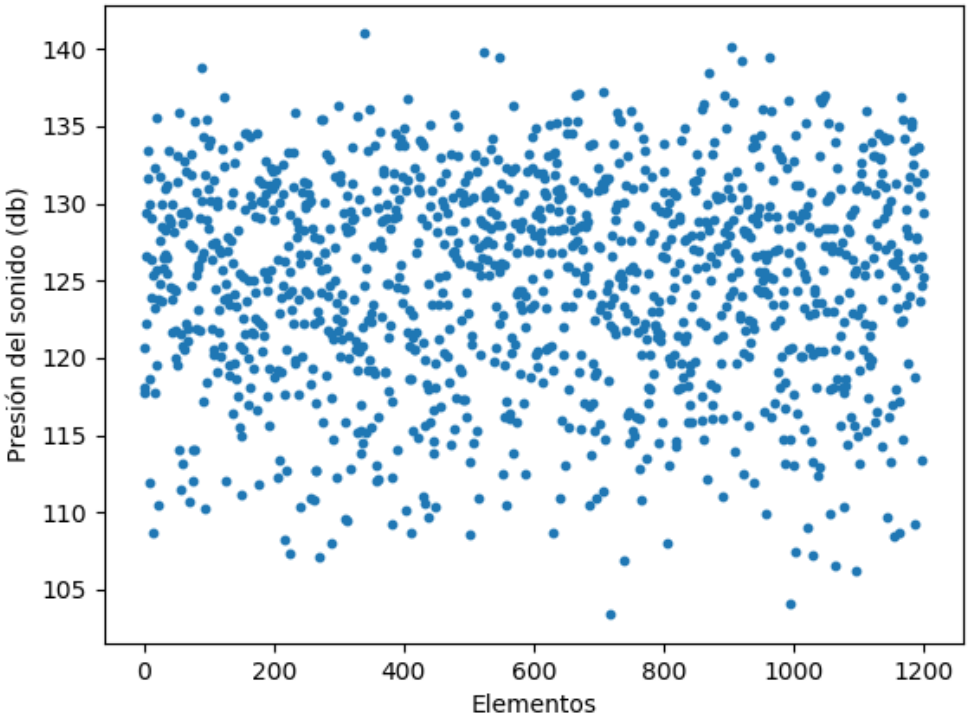

Figura 14: Salida de datos de training.

Parece ser una distribución uniforme sin tendencias ni patrones. Para detectar tendencias, ordenamos las salidas de menor a mayor.

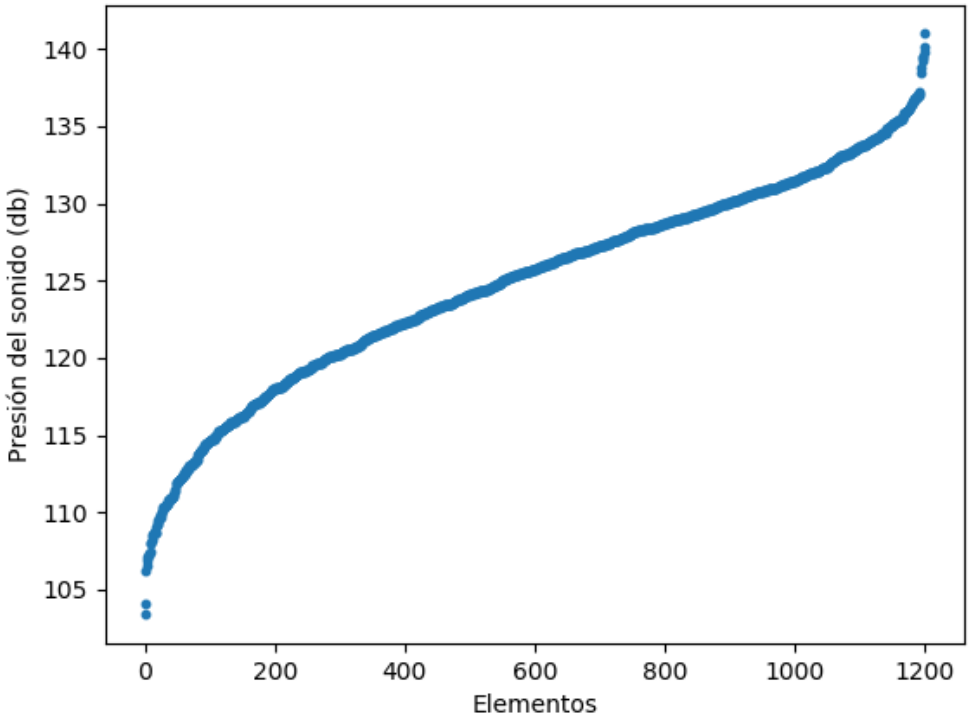

Figura 15: Salida de datos de training, ordenados de menor a mayor.

Se puede apreciar claramente que la salida sigue una función "logit"; no podemos esperar los mejores resultados si ajustamos con un modelo lineal a no ser que se haga una transformación. Debido a la dificultad de ello (y la falta de entendimiento del problema como se comentó con anterioridad), no se hará.

2.3. Funciones a usar

Dado que debemos predecir un valor continuo, se utilizará Regresión Lineal. Aprovechamos el trabajo realizado en la primera práctica y utilizamos SGD: el algoritmo hallará los pesos o coeficientes W de una función lineal que predecirá la salida, donde la función de coste es MSE; buscamos minimizar lo mayor posible la distancia entre nuestra predicción y la salida verdadera. Utilizamos un tamaño de *minibatch* bajo, 4, para acelerar los cálculos, y utilizamos un *learning rate* de 0.01, igual que en prácticas anteriores debido a sus buenos resultados. Se ejecutará el algoritmo miles de iteraciones (10.000) para conseguir convergencia y obtener un ajuste lo más cercano posible. No se aplicará regularización: sabemos de hecho que el ajuste lineal no será el mejor para este tipo de problema, por lo que aplicar regularización no ayudará mucho.

Antes de ejecutar SGD, se debe añadir a cada tupla del conjunto el valor 1.0, para el coeficiente independiente de W.

2.4. Entrenamiento y test

Se ha ejecutado SGD con el junto de training, y se han obtenido los siguientes pesos:

$W = [0{,}8345524 - 0{,}63795127 - 0{,}27559551 - 0{,}267574490{,}10771084 - 0{,}1976129]$

Posteriormente se obtuvieron las predicciones con el conjunto de salida.

Los errores han sido:

$E_{in} = 0{,}01616$

$E_{out} = 0{,}01868$

2.5. Conclusiones

Los errores son bajos y casi similares, por lo que se puede considerar como un buen ajuste. El error de salida es ligeramente superior al de entrada, por lo que podemos asumir que se ha producido algo de sobreajuste. Puede ser debido a que no se ha utilizado *Cross-validation*, y el conjunto de entrenamiento fue mayor al de salida.

Graficamos los resultados, dado que no se esperaba un ajuste decente al ser la salida una sigmoide. En las figuras 16 y 17 se ven, normalizadas, las salidas de los conjuntos de entrada y salida con sus predicciones.

Aunque los puntos siguen la tendencia de la sigmoide, se puede ver claramente una gran dispersión existente entre las predicciones (naranja) y las salidas verdaderas (azul). Como se comentó con anterioridad, un modelo lineal no parece ser el mejor para este tipo de problema, donde la salida claramente no es lineal y las entradas siguen tendencias difíciles de interpretar. La gran dispresión de las predicciones respecto a la salida es prueba de ello.

Aún así, se han conseguido errores bastante bajos y similares: el entrenamiento permite realizar predicciones con el mismo nivel de error.

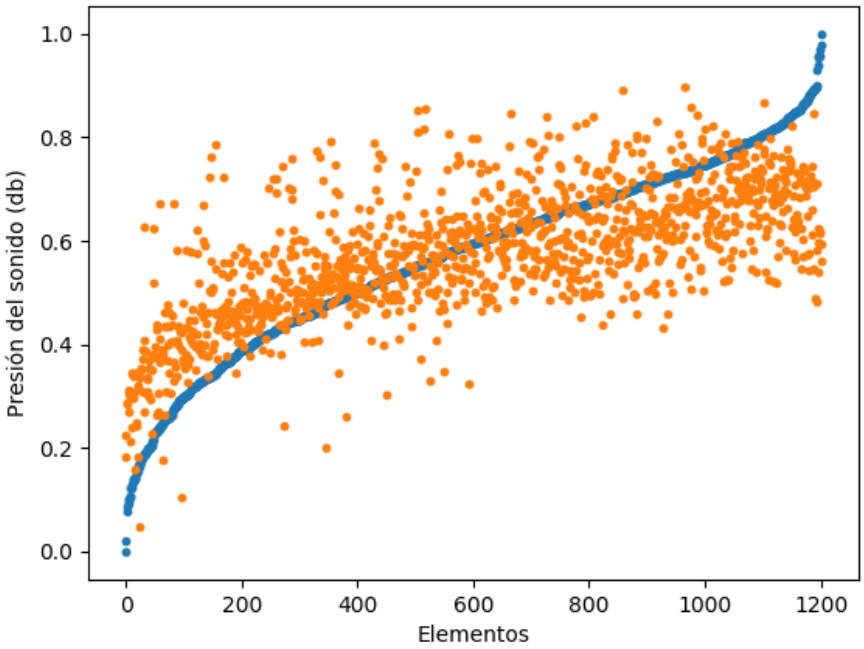

Figura 16: Salida de datos de training (azul) junto con las predicciones (naranja) para ajuste con SGD.

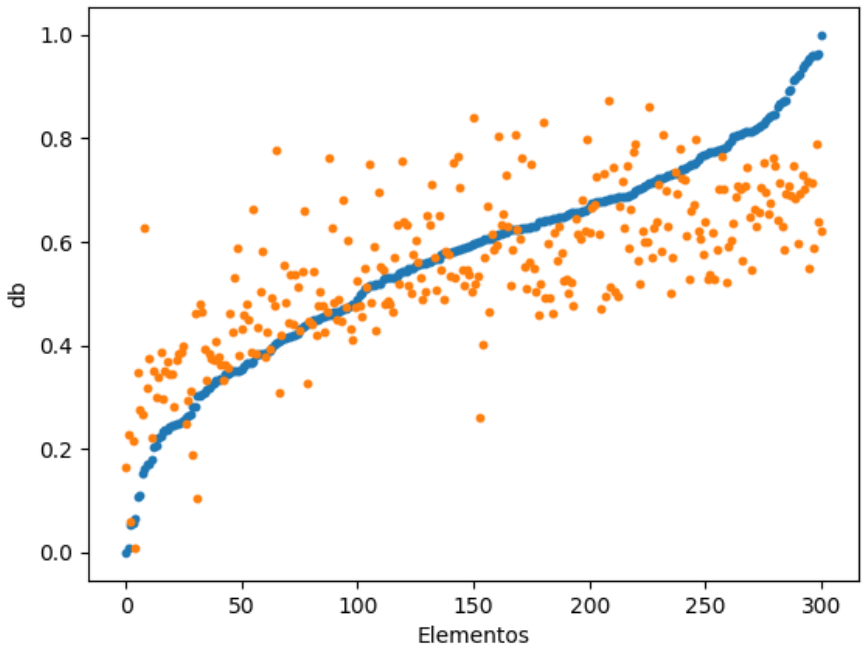

Figura 17: Salida de datos de test (azul) junto con las predicciones (naranja) para ajuste con SGD.

2.6. EXTRA: Ajuste no lineal

Se ha decidido probar un ajuste no lineal para compararlo con el lineal. Se ha utilizado Random Forest, ejecutándose con los parámetros por defecto de SKLearn pero con un mayor número de árboles, 100.

En las figuras 18 y 19 se ven los resultados.

Se puede ver un mejor ajuste que en el caso lineal, como era de esperar.

Los errores han sido:

$E_{in} = 0{,}00031$

$E_{out} = 0{,}00489$

El error de entrada ha sido mucho menor que el de entrada; en las figuras se puede ver claramente que se ha producido *overfitting*. Aún así, el error de salida es mucho menor que en el modelo lineal.

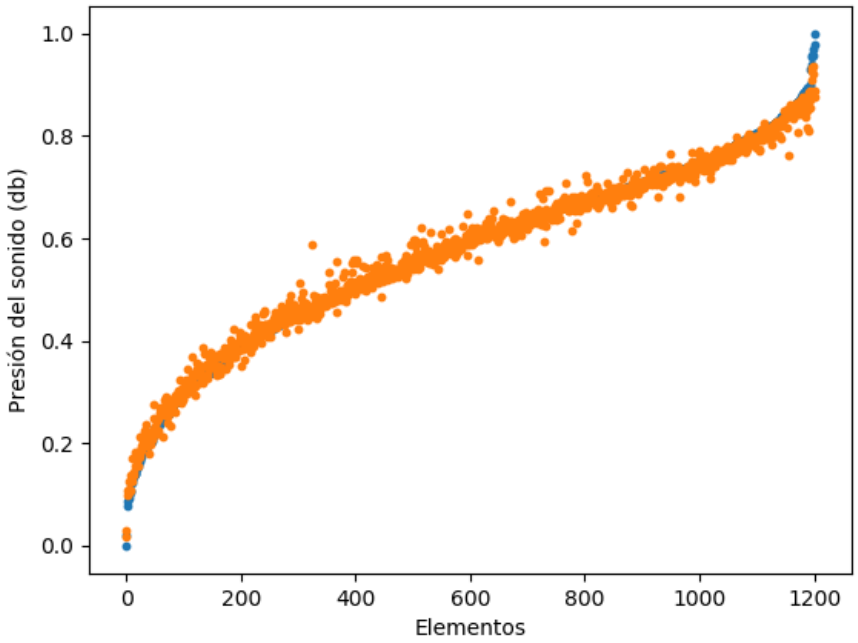

Figura 18: Salida de datos de training (azul) junto con las predicciones (naranja) para ajuste con Random Forest.

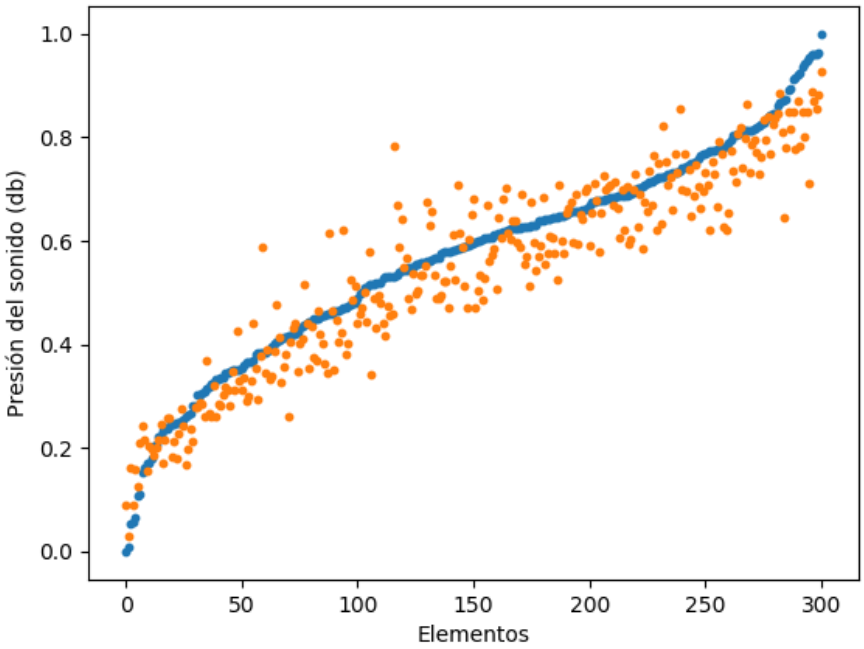

Figura 19: Salida de datos de test (azul) junto con las predicciones (naranja) para ajuste con Random Forest.

3. Bibliografía

Vladimir Cherkassky, Filip Mulier
LEARNING FROM DATA: Concepts, Theory, and Methods.

Se ha utilizado como bibliografia adicional PDFs ofrecidos por el profesorado y el guión de prácticas.