Práctica 3

Aprendizaje Automático

David Gil Bautista

45925324M

Grupo 1

Clasificación

1. **Comprender el problema a resolver**

   Dicha práctica nos planteaba resolver un problema de clasificación de una base de datos de dígitos escritos a mano usando varios modelos y presentando aquél que nos ofreciera un mejor resultado, siempre y cuando lo justificaramos.

   A la hora de afrontar un problema de clasificación debemos comprender el problema y estudiar la base de datos que tenemos.

   En este caso nos encontramos con una base de datos mediana, con 3823 datos en el conjunto de train y con 64 características.

```
In [9]: X_train = np.load("./datos/optdigits_tra_X.npy")

In [10]: X_train.shape
Out[10]: (3823, 64)
```

   En el caso de las etiquetas podemos ver que nos encontramos ante un problema multiclase con las siguientes etiquetas.

```
In [11]: y_train = np.load("./datos/optdigits_tra_y.npy")

In [12]: np.unique(y_train)
Out[12]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

   Sabiendo esto lo trataremos más detenidamente en otro apartado.

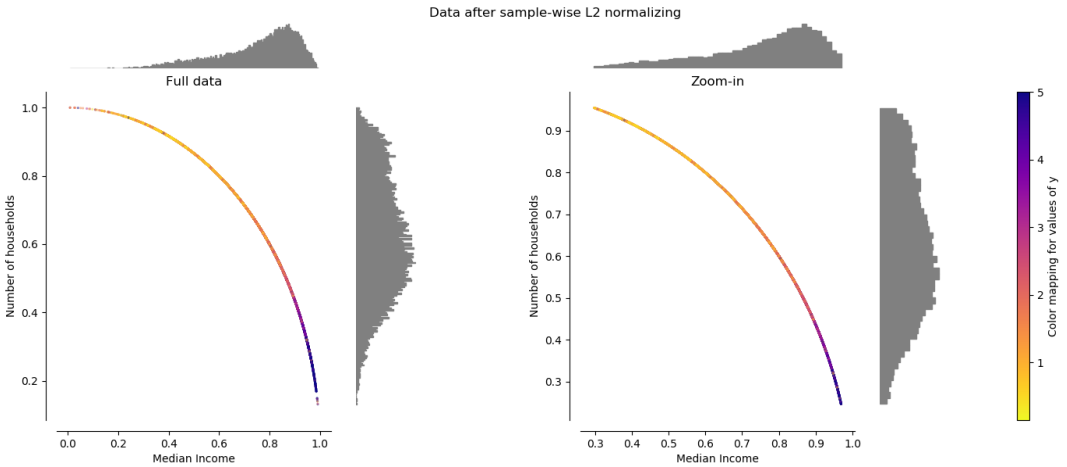
2. **Preprocesado de los datos**

Una vez sabemos algo de nuestra base de datos debemos tratar con ella para comprobar si podemos usar aprendizaje automático para aproximar una solución o no, para que podamos usar aprendizaje automático debemos tener una muestra de datos iid (independiente e identicamente distribuido).

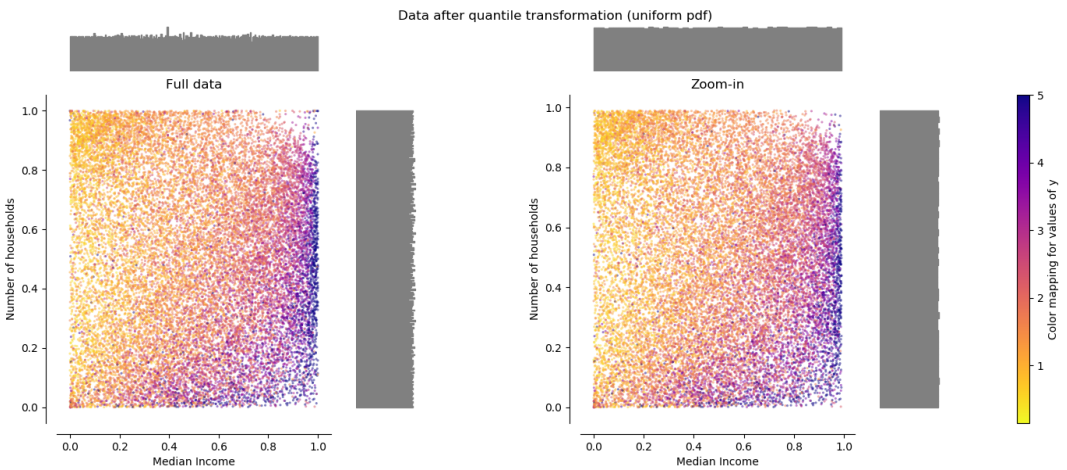
Sabiendo que tenemos una muestra correcta de datos debemos prepararlos para poder aplicar un modelo que nos ajuste correctamente, si los datos son correctos pero no los hemos preparado puede que nuestro modelo no ajuste bien o que tarde bastante más tiempo en obtener una buena solución.

Al preprocesar los datos usamos la muestra de entrenamiento para ajustar los mejores parámetros de preprocesado y una vez ajustados los parámetros los aplicamos a los conjuntos de entrenamiento y de testeo.

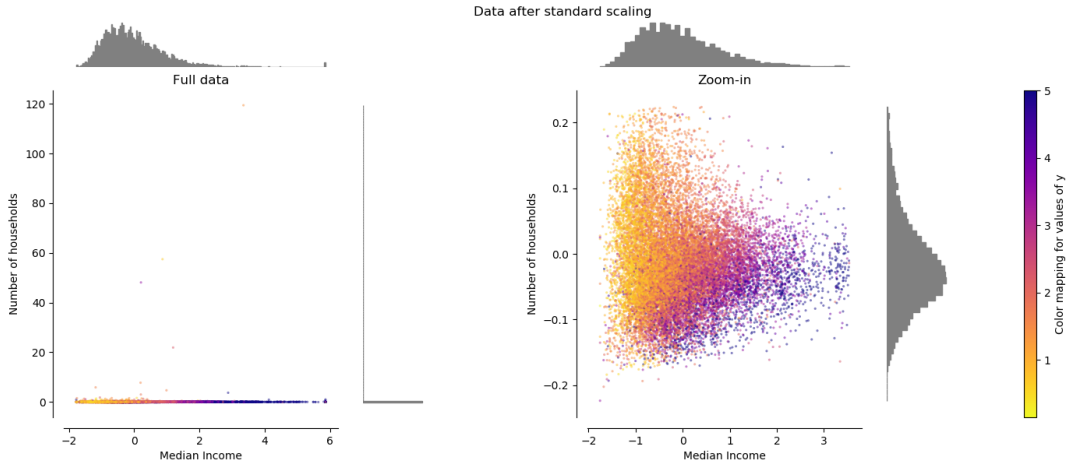
La primera medida que he tomado para preprocesar los datos ha sido normalizarlos. Al normalizar usando *Normalizer* reescalamos cada muestra (fila), que al menos tenga una característica distinta de 0, a la norma de 1.



Transformar las características para que sigan una distribución normal o uniforme usando *QuantileTransformer*. Al usar esto hacemos que las muestras que presentan datos extraños (ya sea por ruido o por que no están cerca de la media) sean menos relevantes a la hora de tratar con ellos. Una vez hecho este proceso obtenemos muestra uniforme o normal que nos permite tomar otra medida más.



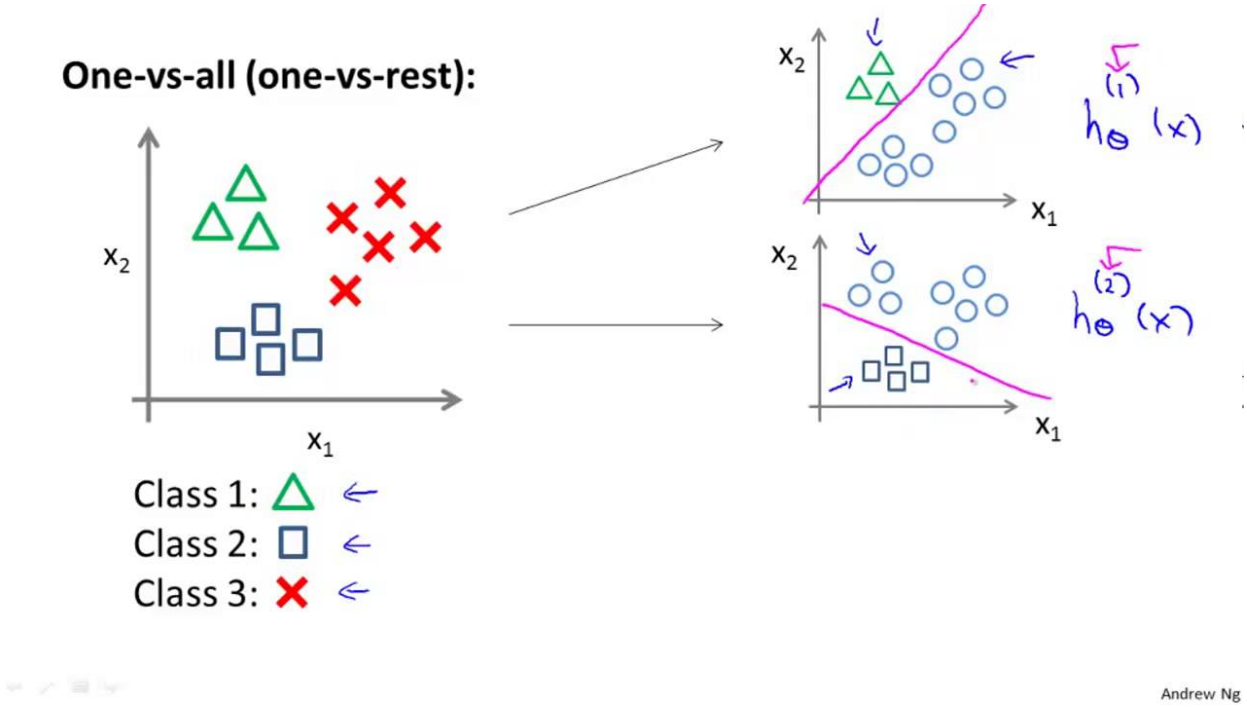
La última medida a tomar ha sido estandarizar los datos, para hacer esto debemos tener una muestra perteneciente a una distribución probabilística . Usando *StandardScaler* estandarizamos las características (columnas) eliminando la media y escalando por la varianza.



Sabiendo esto he ejecutado el programa obteniendo un modelo que nos aporta un 93% de accuracy score, sin embargo, sin procesaro los datos he obtenido un accuracy score del 94%. Por lo que he decidido no preprocesar los datos.

3. **Selección de clases de funciones a usar**

   Para este problema he usado la clase de funciones '*ovr*', dicha clase hace que en un problema de clasificación en el que las muestras de datos tienen etiquetas multiclase, resuelva el modelo ajustando una función que permita separar una clase de todas las demás. Se puede ver mejor en el siguiente ejemplo.



4. **Definición de los conjuntos de training, validación y test usados en su caso**

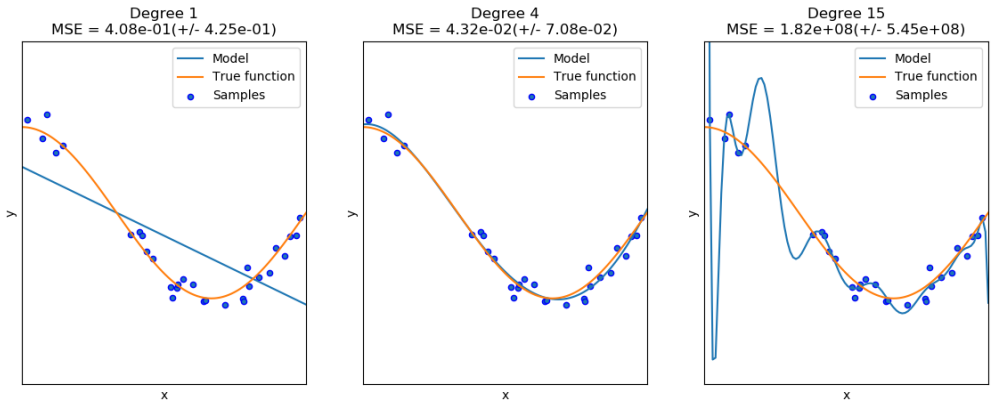
   En este caso se nos daban divididos los conjuntos de train y set, por lo que no ha sido necesario dividirlos.

   Para el conjunto usado en la validación he decidido usar una división en 5 partes, ya que con un tamaño muestral de casi 4000 ejemplos, al dividirlo en 5 partes nos aseguramos que cada parte sea completa y mantenga una varianza estable. El dividir un conjunto en más partes nos puede hacer conseguir un mejor modelo en una parte (train más grande), pero que disminuya la varianza usada para validar. Dividir el conjunto en menos partes aumenta la varianza pero le quita datos al training, por lo que puede obtener un peor modelo.

   Creo que usando una división del 20% para la validación y el training obtenemos un modelo correcto en el que no nos arriesgamos a que los datos estén muy dispersosy que nos ofrezca un score en la validación bastante aceptable.

5. **Discutir la necesidad de regularización y en su caso la función usada para ello**

¿Por qué penalizamos siempre en los modelos? Al penalizar un modelo hacemos que se "ajuste un poco peor", la regularización nos ayuda a evitar el sobreajuste o que un modelo no se ajuste suficiente. Es por esto que siempre lo usamos a la hora de obtener un modelo.



Como podemos ver en la imagen superior, con un grado de 1, obtenemos una modelo lineal que no se ajusta a la función de los datos, sin embargo, usando un grado de 4, podemos apreciar que obtenemos un modelo prácticamente igual a la función original. Conforme subimos el grado estamos haciendo que nuestro modelo sobreaprenda sobre esos datos, lo cual lo lleva a crear un modelo que divida basado exactamente en los datos que se han usado para entrenar, lo cual nos lleva a obtener un E_{in} perfecto pero arriesgandonos a que el E_{out} sea desastroso.

Con la regularización buscamos obtener un modelo que nos ofrezca una función capaz de obtener un buen E_{out} sacrificando que el E_{in} un poco peor al entrenar nuestro modelo.

He optado por usar lasso, ridge y elasticnet. De los cuáles después, al comentar el modelo escogido, mostraré el usado.

- *Ridge*: aproxima a cero los coeficientes de los predictores pero sin llegar a excluir ninguno.
- *Lasso*: aproxima a cero los coeficientes, llegando a excluir predictores.

6. **Definir los modelos a usar y estimar sus parámetros e hyperparámetros**

   Para resolver este problema he decidido usar regresión logística, debido al buen resultado que me ha ofrecido en otras prácticas, y un clasificador basado en gradiente descendente, debido a su poco complejo funcionamiento y alta eficiencia.

   Para regresión logística he usado los siguientes hiperparámetros.

```
    #Logistic Regression

hyper= [{'penalty': ['l1','l2'], 'C': [1, 10, 100, 1000, 10000, 100000],'solver':
['liblinear'], 'max_iter':[15]},
        {'penalty': ['l2'],'C': [1, 10, 100, 1000, 10000, 100000],'solver':['newton-
cg'],'max_iter':[15]}] #Al final descarté el uso de newton-cg
```

   Y para gradiente descendente:

```
# SGD Classifier

hyper= [{'penalty': ['l1','l2','elasticnet'], 'alpha': [0.00001, 0.0001, 0.001, 0.01, 0.1,
1], 'max_iter':[15]}]
```

   Ambos modelos han sido testeados con la misma penalización, añadiendo en sgd elasticnet, que es una combinación de lasso y ridge, y aparte 6 valores distintos para el parámetro que establece la relevancia del parámetro de regularización.

   Para la regresión logística se ha probado con una clase lineal '*liblinear*' basada en ovr y otra clase, '*newton-cg*' , que soporta la pérdida multinomial.

   Ambos modelos han sido entrenados con un máximo de iteraciones, aunque en el caso de regresión logística con *'newton-cg'*, con tan solo un máximo de 15 iteraciones no converge. A pesar de ello obtiene parámetros que obtienen un buen score.

7. **Selección y ajuste modelo final**

Tras seleccionar el mejor modelo de la validación cruzada calculamos su E_{in}, comparamos los dos modelos elegidos y el que tenga un mejor score será el elegido para estimar el E_{out}.

En la primera prueba que hice usando tan sólo '*liblinear*' en regresión logística y un máximo de 5 iteraciones en los dos modelos obtuve el siguiente resultado.

```
LOGISTIC REGRESSION

Estimando parámetros para regresión logística...

Execution time :  6.442292928695679 s
Best parameters:  {'C': 10, 'max_iter': 5, 'penalty': 'l2'}
Accuracy train:  0.9542244310750719
Accuracy test:  0.9254312743461325

SGD CLASSIFIER

Estimando parámetros para clasificador sgd...

Execution time :  4.187949895858765 s
Best parameters:  {'alpha': 0.0001, 'max_iter': 5, 'penalty': 'l1'}
Accuracy train:  0.9649489929374836
Accuracy test:  0.9332220367278798

Best classifier: sgd
Confusion matrix:
 [[176   0   0   0   1   1   0   0   0   0]
 [  0 161   3   2   0   0   2   0   6   8]
 [  0   1 171   3   0   0   0   0   2   0]
 [  1   0   1 170   0   2   0   1   2   6]
 [  0   3   0   0 172   0   1   1   3   1]
 [  0   0   1   1   0 176   1   0   0   3]
 [  0   1   0   0   2   0 175   0   3   0]
 [  0   1   0   1   2   9   0 160   2   4]
 [  0   7   0   1   0   1   4   0 150  11]
 [  0   1   0   2   2   3   0   0   6 166]]
```

En el cual obtenemos un buen resultado en ambos modelos, siendo un poco mejor el gradiente descendente a parte de ser más rápido ajustando.

Para otra prueba aumenté el número de iteraciones a 20 en ambos modelos, y en este caso obtuve lo siguiente:

```
LOGISTIC REGRESSION

Estimando parámetros para regresión logística...

Execution time :  113.9072470664978 s
Best parameters:  {'C': 10, 'max_iter': 20, 'penalty': 'l2'}
Accuracy train:  0.9761967041590374
Accuracy test:  0.9393433500278241

SGD CLASSIFIER

Estimando parámetros para clasificador sgd...

Execution time :  15.030266284942627 s
Best parameters:  {'alpha': 1e-05, 'max_iter': 20, 'penalty': 'l1'}
Accuracy train:  0.9680878890923359
Accuracy test:  0.9293266555370061

Best classifier: logistic regression
Confusion matrix:
 [[176   0   0   0   1   1   0   0   0   0]
 [  0 170   2   0   0   0   0   0   3   7]
 [  0   1 171   2   0   0   0   0   3   0]
 [  1   0   2 169   0   2   0   2   4   3]
 [  0   3   0   0 176   0   0   1   1   0]
 [  0   0   1   1   0 176   2   0   0   2]
 [  0   3   0   0   2   0 174   0   2   0]
 [  0   0   0   0   2  12   0 162   2   1]
 [  0  10   0   1   0   1   2   0 153   7]
 [  1   2   0   3   3   3   0   0   7 161]]
```

Ahora podemos ver que el mejor clasificador es la regresión logística, pero dado que el tiempo que tarda para conseguir un 1% más de porcentaje de aciertos en el train es 10 veces el de gradiente descendente no vería factible usar este modelo.

Para finalizar, tras varias pruebas usando distintos solver decidí utilizar el que venía por defecto puesto que era mucho más rápido para regresión logística y para el cual obtenía el siguiente resultado:

```
sgd 0.9754119801203244
lr 0.9832592205074548
Best classifier: logistic regression
Params:  {'C': 1, 'max_iter': 15, 'penalty': 'l1'}
Eout:  0.9499165275459098
Confusion matrix:
 [[176   0   0   0   0   2   0   0   0   0]
 [  0 170   0   0   0   0   0   0   6   6]
 [  0   1 172   1   0   0   0   1   2   0]
 [  0   0   1 170   0   3   0   2   2   5]
 [  0   1   0   0 176   0   0   1   2   1]
 [  0   0   1   1   0 178   1   0   0   1]
 [  0   1   0   0   2   0 177   0   1   0]
 [  0   0   0   0   1   5   0 165   1   7]
 [  0  10   0   3   0   2   0   0 154   5]
 [  0   1   0   3   2   2   0   0   3 169]]
```

En este caso, tras cambiar los parámetros de la regresión logística podemos ver que volvemos a obtener un mejor resultado, sin preprocesar los datos y con un máximo de 15 iteraciones obtenemos un modelo que nos da un porcentaje de aciertos de un 98% en el train.

En caso de elegir un modelo para clasificar este conjunto de datos usaría el clasificador basado en gradiente descendente puesto que el tiempo que ocupa en resolver el problema es muy pequeño para el score que obtenemos.

8. **Discutir la idoneidad de la métrica usada en el ajuste**

   Para la medida del error se ha usado accuracy, que nos da la tasa de aciertos que obtenemos al predecir *X_test* y comparar con el *y_test*.

   A pesar de no ser una clasificación binaria, en la que es más fácil representar la matriz de confusión, la hemos usado para ver cómo se comporta en un problema con multiclase.

   Como podemos ver en el apartado anterior, obtenemos una matriz en la que la mayoría de las posiciones están a cero exceptuando la diagonal principal. Las columnas nos dicen los dígitos que ha predicho nuestro modelo y las filas los dígitos que aparecen en el *y_test*, en caso de tener una matriz diagonal hubiéramos obtenido un modelo capaz de predecir correctamente el 100% de la muestra.

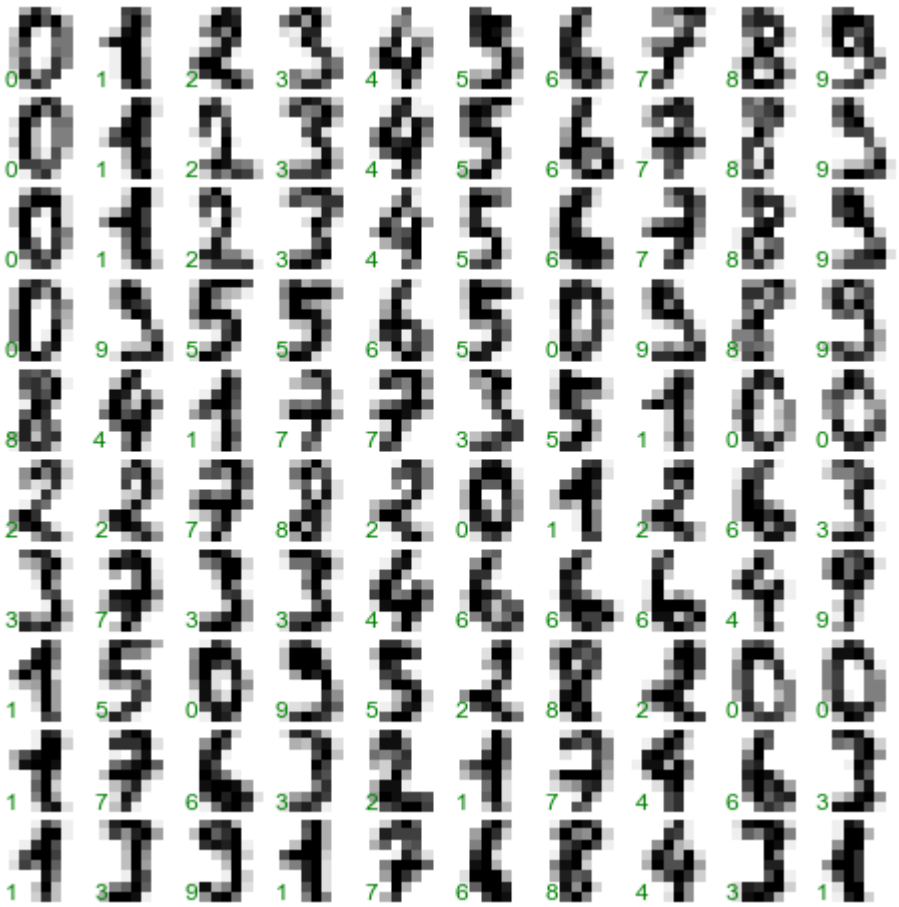
9. **Estimacion del error E_{out} del modelo lo más ajustada posible**

   Como hemos visto antes, obtenemos errores muy parecidos usando los distintos clasificadores, pero que en caso de querer ajustar el E_{out} de la mejor forma posible podríamos usar Regresión Logística y dar bastantes iteraciones hasta conseguir un modelo capaz de predecir casi el 100% de la muestra (si suponemos que la muestra no tiene reuido).

   La tasa de aciertos de la regresión es de orden logarítmico y para llegar a conseguir ese 100% deberíamos iterar muchas veces cambiando poco a poco los parámetros hasta conseguir los coeficientes que más aproximen las características a la función objetivo.

10. **Discutir y justificar la calidad del modelo encontrado y las razones por las que considera que dicho modelo es un buen ajuste que representa adecuadamente los datos muestrales**

    Para responder esta pregunta comenzaré mostrando la base de datos que estudiamos en el ejemplo.



Dado que una persona normal puede que llegue a fallar a la hora de intentar reconocer uno de estos dígitos, un modelo que consigue un 94% de aciertos me parece más que bueno.

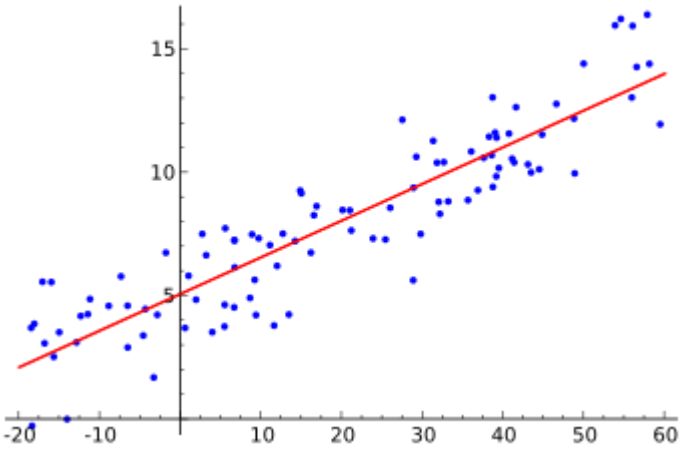
Para justificarme en el modelo que he escogido, SGD, he de decir que este modelo ha ajustado un conjunto de casi 4000 muestras con varios hiperparámetros en tan sólo 12 segundos. En 4 segundos ha sido capaz de aproximar un modelo que nos proporcione una tasa de aciertos de un 93%, por lo que lo veo un muy buen clasificador para el problema con el que nos encontramos.

En caso de encontrarnos ante otro problema en el que tuviéramos la necesidad de obtener una predicción con el 100% de precisión no lo usaría. Para problemas médicos en los que tuviéramos la necesidad de saber con certeza si un paciente tiene o no una enfermedad valdría más arriesgar un poco el tiempo y obtener una mejor predicción, a tan sólo obtener una predicción buena en muy poco tiempo. En otros problemas en los que necesitaramos una buena respuesta sin necesidad de ser la mejor, este modelo presenta de los mejores resultados.

Regresión

1. **Comprender el problema a resolver**

   Por definición, la regresión lineal es un modelo usado para aproximar la relación de dependencia entre una variable dependiente Y, y las variables independientes X. Es decir, nuestro modelo de regresión va a ser una función que reciba X y produzca una Y, $f : X \rightarrow Y$.

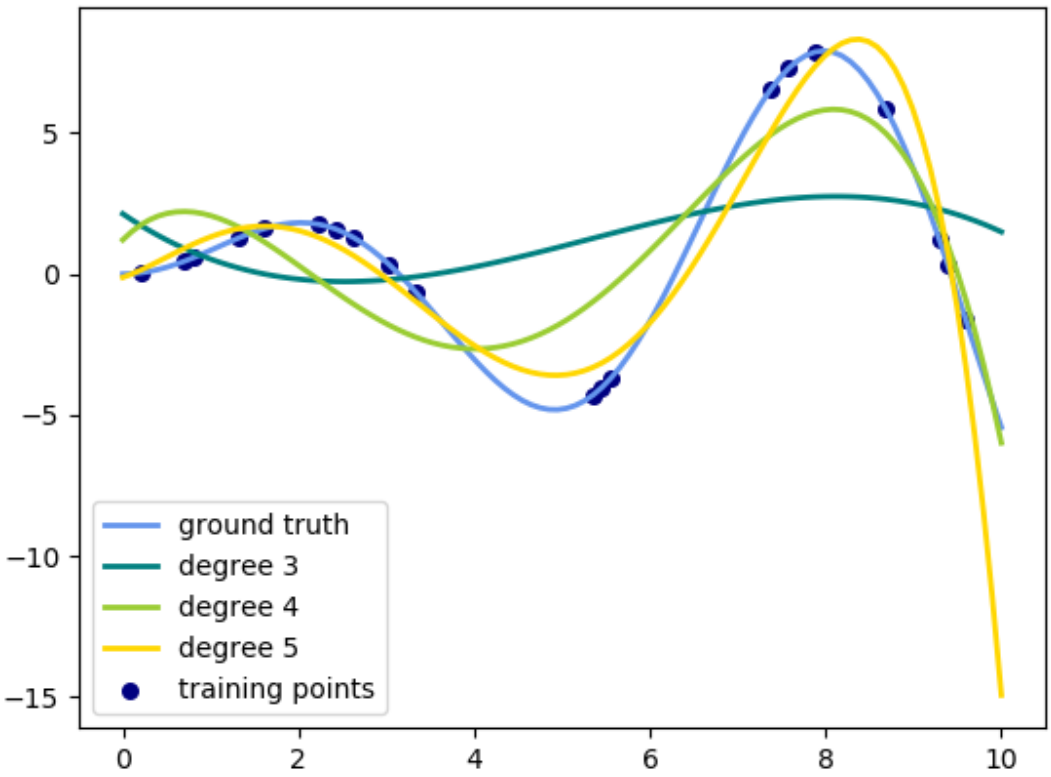
   Para una regresión lineal la función que nos da un Y, es una función lineal, es por esto que para problemas más complejos usar regresión lineal no es lo óptimo y debemos probar con distintas dimensiones.

2. **Preprocesado de los datos**

   En este apartado, al igual que en el anterior, hemos normalizado, ajustado a una distribución y escalado los datos.

   Haciendo este preprocesado he llegado a obtener un score de un 30%.

   Para solucionar esto he optado por añadir una conversión de los datos de la muestra a polinomios de distinto grado.



Como se puede observar en la imagen, el cambiar el grado del polinomio hace que explore unos puntos distintos que los que se le han ofrecido.

Suponiendo que tenemos un conjunto X que tiene dos características {a,b}, al buscar un modelo que permita operar con esas características para predecir un resultado, tan sólo podriamos hacer operaciones sobre a y b. Al convertirlo a un polinomio de mayor grado tendríamos lo siguiente:

Grado 1: $X(a, b)$

Grado 2: $X^2(a^2, ab, b^2)$

Ahora exploramos más opciones con las que podemos operar para obtener unos coeficientes que nos permitan predecir un resultado distinto.

3. **Selección de clases de funciones a usar**

   Con el parámetro de solver fijado en automático es el propio modelo el que escoge las mejores clases dependiendo del tipo de datos que presente la muestra.

4. **Definición de los conjuntos de training, validación y test usados en su caso**

   Con esta base de datos nos hemos encontrado con dos conjuntos, uno de datos y otro de etiquetas. Tras haber comprobado el tamaño del conjunto he optado por dividirlo en un 80% para el train y un 20% para el test.

```
In [97]: X = np.load("./datos/airfoil_self_noise_X.npy")

In [98]: X.shape
Out[98]: (1503, 5)
```

   Una vez tenemos dividido el conjunto de datos lo siguiente es escoger el tamaño que tendrá el conjunto de validación. Como he explicado antes, al aumentar el porcentaje de división, entrenamos con más datos, lo que nos ayuda a obtener un mejor modelo.

   Al tratarse de una regresión, en la que buscamos llegar a una función objetivo, el testear después con menos datos no influye mucho en el E_{out}.

5. **Discutir la necesidad de regularización y en su caso la función usada para ello**

   Al igual que en el ejemplo anterior, se ha usado regularización. En este caso he optado por usar Ridge. Como ya he explicado antes, Ridge aproxima a 0 los coeficientes, sin llegar a excluirlos. Con la combinación de Ridge y los polinomios de grado n en la muestra nos encontramos con un espacio muestral mayor en el que podemos explorar más posibilidades las cuales acercan nuestro modelo a una solución óptima evitando el sobreajuste.

6. **Definir los modelos a usar y estimar sus parámetros e hyperparámetros**

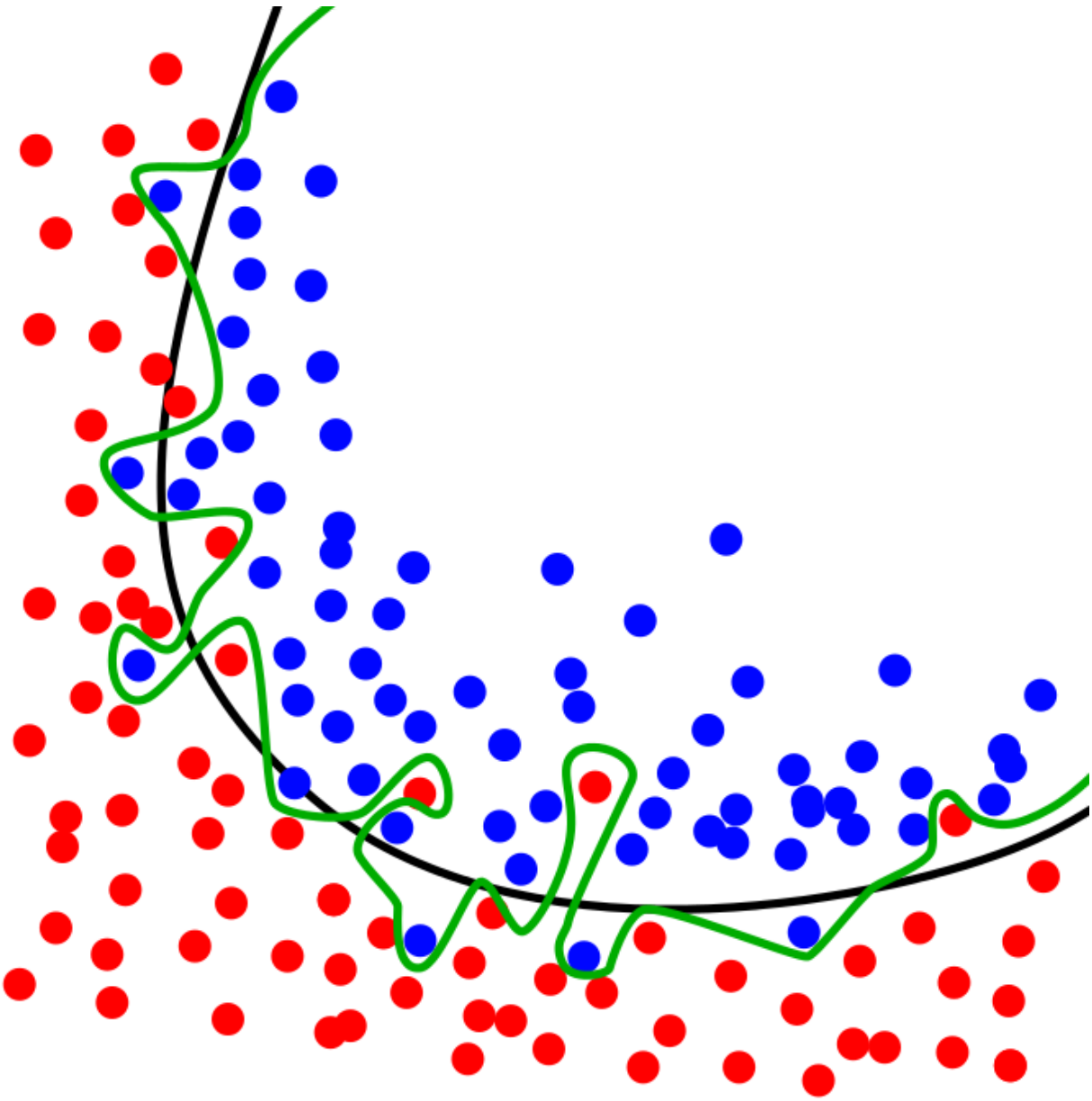
   En este caso he usado Ridge, puesto que es el único modelo que se nos permitía usar y que tuviera hiperparámetros. También probé con Regresión Lineal, pero dado que no usaba hiperparámetros lo descarté, a pesar de que ofrecía una mejor solución que Ridge.

   Los hiperparámetros usados han sido los siguientes:

```
# Ridge

hyper= [{'alpha': [0.0001, 0.001, 0.01, 0.1, 1]}]
```

   Dado que hemos convertido la muestra de datos a diferentes polinimios, lo normal sería producir un sobreajuste:



En la foto superior podemos ver una linea negra que divide perfectamente al conjunto de datos, sin embargo al aumentar la complejidad de este se pueden dar modelos que ajusten como el de la línea verde. Usando Ridge penalizamos este ajuste, es por esto que conforme aumentan los grados del polinomio nuestro modelo ajusta peor, cuando debería ser lo contrario, dado que a mayor complejidad mejor ajustaría el train.

De esta forma, usando Ridge, obtenemos un modelo fiable que produzca un buen resultado en el E_{out}.

7. **Selección y ajuste modelo final**

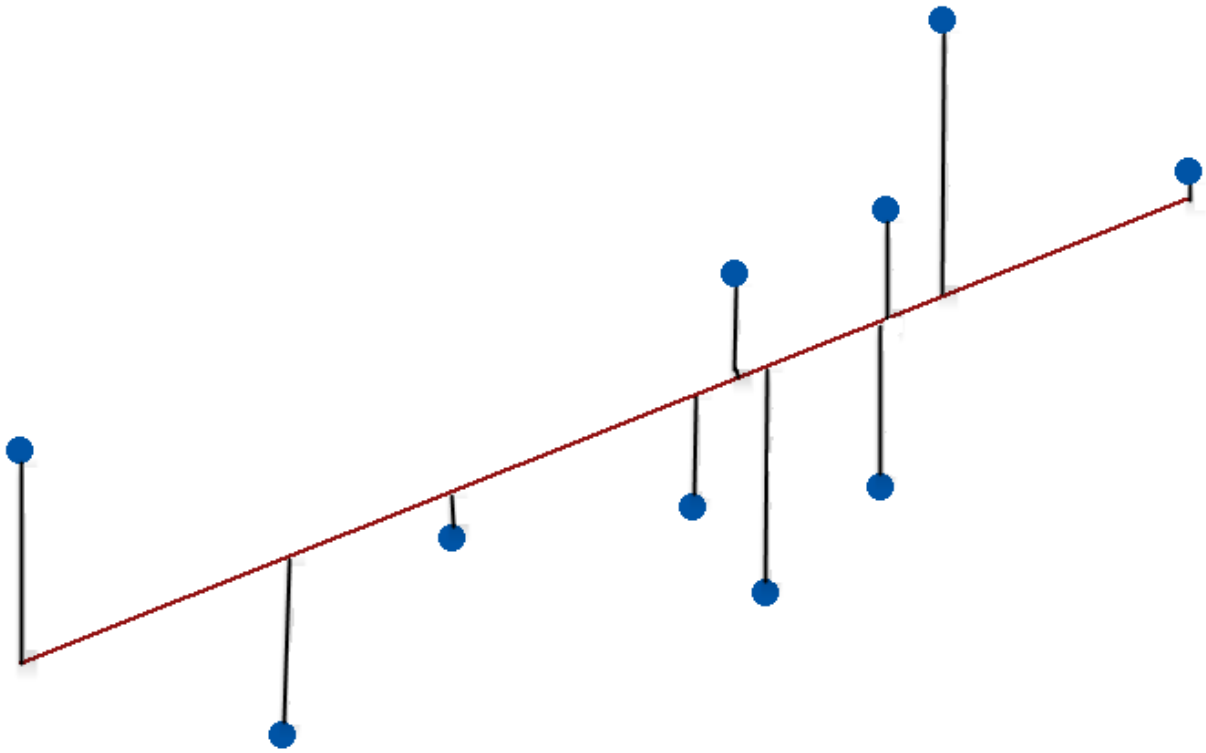
   Para el apartado he decidido usar Ridge con distintos parámetros para controlar la regularización. El mejor modelo obtenido ha sido para una validación cruzada de 8 partes, con un alpha de 1.

   Al tener un alpha de 1, sabemos que la regularización no es muy fuerte, por lo que deducimos que aumentando el grado del polinomio a 8, obtenemos un modelo fiable para el que no tiene que regularizar.

```
Best polynomial degree:  8
Best classifier: GridSearchCV(cv=8, error_score='raise',
       estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'alpha': [0.0001, 0.001, 0.01, 0.1, 1]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='r2', verbose=0)
Eout:
 0.8850481106626378
```

8. **Discutir la idoneidad de la métrica usada en el ajuste**

   Hemos usado R2 para la métrica del ajuste, cuando usamos R2 como score, buscamos maximizarlo. Maximizar R2 equivale a minimizar SSR (sum of squared residuals), donde los residuos son la distancia de los puntos al modelo ajustado:



Minimizar esto es lo mismo que buscar aquel modelo cuya distancia a los puntos observados sea mínima, es decir, el modelo que ajusta los datos y tiene una varianza mínima.

9. **Estimacion del error E_{out} del modelo lo más ajustada posible**

   Usando los parámetros que hemos obtenido en la validación cruzada obtenemos un score del 88% para un grado 8 de polinomio. Para distintos valores en la validación cruzada he alcanzado valores de 83%, 85%... pero el más alto obtenido con el código que tengo ha sido del 88%.

10. **Discutir y justificar la calidad del modelo encontrado y las razones por las que considera que dicho modelo es un buen ajuste que representa adecuadamente los datos muestrales**

    Obtenemos un modelo que nos proporciona un score del 88%, dado que la muestra de datos no es muy grande y que el mejor score obtenido en el conjunto de entrenamiento es del 94.8% podemos afirmar que nos encontramos ante un modelo que clasifica fielmente los datos muestrales.