Práctica 6: Aprendizaje automático en la práctica

Datos de entrenamiento

En esta práctica los datos de entrenamiento serán artificiales, generados con esta función:

```
def gen_data(m, seed=1, scale=0.7):
    """ generate a data set based on a x^2 with added noise """
    c = 0
    x_train = np.linspace(0, 49, m)
    np.random.seed(seed)
    y_ideal = x_train**2 + c
    y_train = y_ideal + scale * y_ideal*(np.random.sample((m,))-0.5)
    x_ideal = x_train
    return x_train, y_train, x_ideal, y_ideal
```

De esta forma podremos generar tantos ejemplos de entrenamiento como sea necesario y con tanto ruido como nos interese. Así, por ejemplo, con la llamada `gen_data(64)` obtenemos unos datos de entrenamiento como los que se muestran en la Figura 1.1.

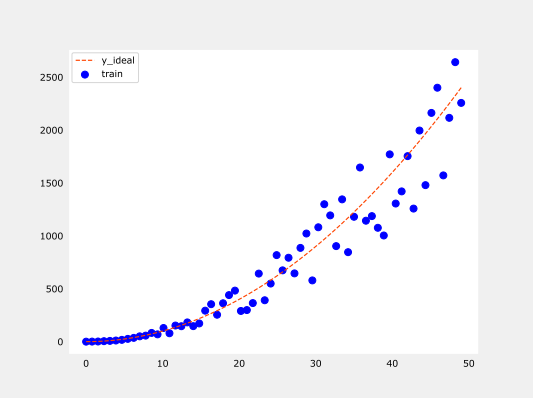

Figure 1.1: Datos de entrenamiento artificiales

El array que devuelve `gen_data` es unidimensional mientras que las funciones de scikit-learn que usaremos a continuación necesitan arrays bidimensionales, por lo que deberemos añadir una nueva dimensión cuando sea necesario:

```
1  X = X[:, None]
```

Sobreajuste a los ejemplos de entrenamiento

Empezaremos por entrenar un modelo que se sobreajuste a los ejemplos de entrenamiento, ajustando los datos con un polinomio de grado 15. Para ello, en primer lugar, dividimos los datos generados por `gen_data` en datos de entrenamiento y datos de prueba utilizando la función `sklearn.model_selection.train_test_split`[1], asignando un 33% a los datos de entrenamiento y estableciendo el parámetro `random_state` a 1 para que los resultados sean reproducibles.

A continuación, transformamos los datos de entrenamiento X instanciando la clase `sklearn.preprocessing.PolynomialFeatures`[2] (inicializada con el parámetro include_bias=False?) y aplicando el método `fit_transform` de esta clase para generar las combinaciones polinómicas de los atributos hasta grado 15.

Antes de ejecutar la regresión lineal sobre los datos polinómicos será necesario normalizar-los instanciando la clase `sklearn.preprocessing.StandardScaler`[3] y aplicando el método `fit_transform` de esta clase.

Finalmente, tendremos que crear una instancia de la clase `sklearn.linear_model.LinearRegression`[4] y entrenarla aplicando el método `fit` de esta clase sobre los datos de entrenamiento.

Ahora ya podemos comparar el error del modelo sobre los datos de entrenamiento y sobre los datos de prueba. Para ello tendremos que implementar una función que calcule el error de las predicciones del modelo:

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2$$

[1] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[4] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

y comparar el error aplicando esa función sobre los datos de entrenamiento y los datos de test.

Las predicciones del modelo instanciado de `sklearn.linear_model.LinearRegression` y previamente entrenado con el método `fit` se generan con el método `predict` de esta misma clase aplicado a la matriz X para la que queremos obtener las predicciones y. Como el modelo ha sido entrenado sobre datos polinómicos normalizados, habrá que transformar la matriz X con las mismas instancias de `PolynomialFeatures` y de `StandardScaler` que se utilizaron para transformar los datos de entrenamiento.

Si la implementación es correcta deberías obtener un valor aproximado de 11855 para $J_{train}(\vec{w}, b)$ y de 48579 para $J_{test}(\vec{w}, b)$. Además, puedes generar una gráfica donde se muestre que efectivamente el modelo está sobre-ajustado a los ejemplos de entrenamiento, como en la Figura 1.2.

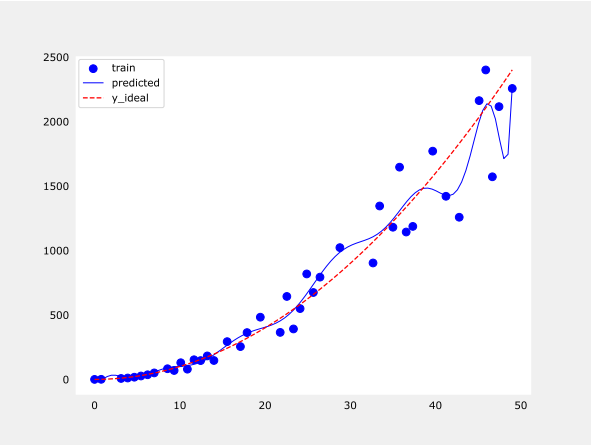

Figure 1.2: Sobre-ajuste a los datos de entrenamiento

Elección del grado del polinomio usando un conjunto de validación

Para obtener el grado óptimo de la transformación polinomial aplicada al modelo dividiremos el conjunto de datos en entrenamiento, validación y prueba, y evaluaremos sobre los datos de validación los modelos obtenidos sobre los datos de entrenamiento, como se explica en teoría.

Generando los datos de la misma forma que en el apartado anterior (`gen_data(64)`) esta vez usaremos el 60% de los datos para entrenamiento, el 20% para validación y el restante 20% para prueba.

Entrenando modelos con transformaciones polinómicas de grado 1 a 10 deberíamos encontrar que el de menor error genera predicciones como las de la Figura 1.3.

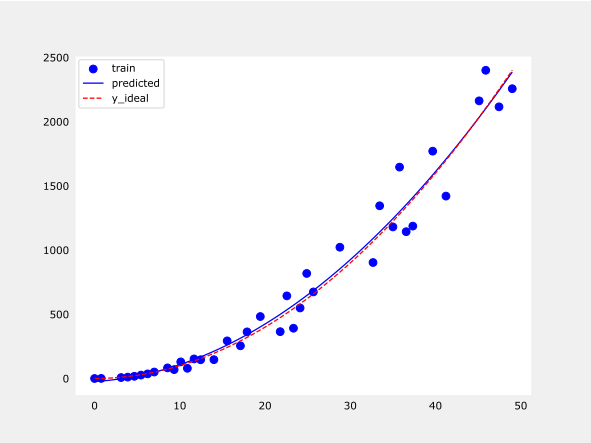

Figure 1.3: Modelo con el grado óptimo del polinomio

Elección del parámetro λ

En este apartado repetiremos el proceso del anterior para seleccionar el mejor valor para el parámetro de regularización λ. Utilizaremos una transformación polinómica de grado 15 y evaluaremos sobre el conjunto de validación cruzada los valores de $\lambda = 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300, 600, 900$. Al tratarse de regresión lineal regularizada, en lugar de instanciar la clase `sklearn.linear_model.LinearRegression` como hasta ahora, instanciaremos la clase `sklearn.linear_model.Ridge`[5] inicializada con el correspondiente valor de λ.

El mínimo error sobre el conjunto de validación lo deberíamos encontrar para un modelo que genera predicciones como las de la Figura 1.4.

[5]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

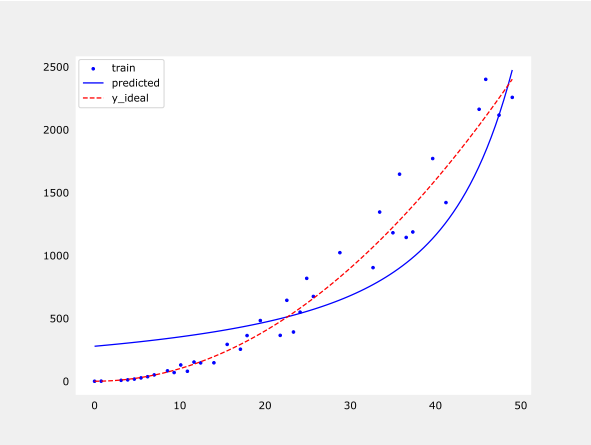

Figure 1.4: Modelo con el mejor parámetro de regularización

Elección de hiper-parámetros

A continuación implementaremos una búsqueda exhaustiva de la mejor combinación de grado del polinomio y valor de λ para un conjunto de 750 valores (`gen_data(750)`), usando el 60% de los datos para entrenamiento, el 20% para validación y el restante 20% para prueba.

Probando con polinomios hasta grado 15 y valores de $\lambda = 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300, 600, 900$ encontrar un modelo que genera predicciones como las de la Figura 1.5

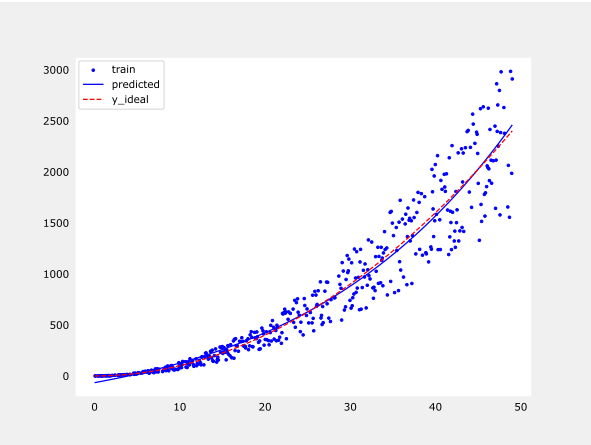

Figure 1.5: Modelo con el mejor combinación de grado del polinomio y parámetro de regularización

El modelo resultante debe tener un error sobre el conjunto de prueba entre 20.000 y 30.000, dependiendo de la división concreta de los datos generados.

Curvas de aprendizaje

Por último, veremos el efecto que tiene aumentar el número de ejemplos de entrenamiento para un modelo que está sobre-ajustado a los ejemplos de entrenamiento.

Entrenaremos un modelo con polinomios de grado 16 sin regularizar con conjuntos crecientes de datos de entrenamiento, generando entre 50 y 1000 ejemplos y usando en cada caso el 60% de los datos para entrenamiento, el 20% para validación y calculando en cada caso el error de entrenamiento y el de validación para generar una gráfica similar a la Figura 1.6

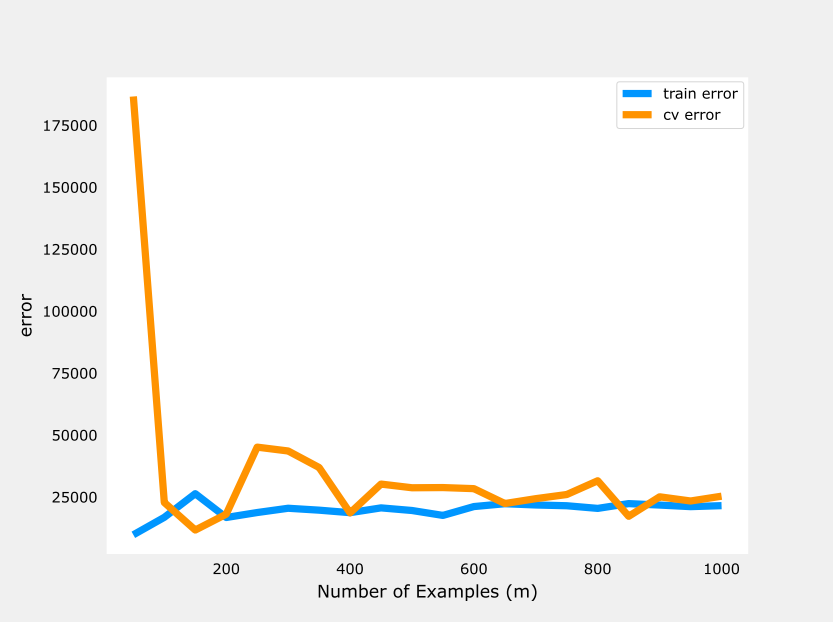

Figure 1.6: Curvas de aprendizaje