In [1]:

```
%load_ext watermark
%watermark
```

```
2019-05-30T21:31:53+02:00

CPython 3.6.5
IPython 6.4.0

compiler   : GCC 7.2.0
system     : Linux
release    : 5.1.5-arch1-2-ARCH
machine    : x86_64
processor  :
CPU cores  : 4
interpreter: 64bit
```

Regularización.

La regularización es un método usado ampliamente para tratar el sobreajuste (overfitting) de los modelos. Se realiza principalmente con las siguientes técnicas:

1. Reducir el tamaño del modelo: a través de la reducción del número de parámetros de aprendizaje del modelo, y con ello su capacidad de aprendizaje. El objetivo es llegar a un punto de equilibrio entre una capacidad de aprendizaje excesiva y una capacidad de aprendizaje insuficiente. Desafortunadamente, no hay fórmulas mágicas para determinar este equilibrio, debe ser probado y evaluado estableciendo un número diferente de parámetros y observando su rendimiento.
2. Añadir la regularización de pesos: En general, cuanto más sencillo es un modelo, mejor es. Mientras pueda “aprender bien”, un modelo simple es menos susceptible de caer en el sobreajuste. Una forma normal de conseguir esto es restringir la complejidad de la red forzando a sus pesos a tomar únicmente valores pequeños. Esto se realiza añadiendo a la función de pérdida de la red “loss function” un coste asociado al uso de grandes pesos. El coste se puede obtener de dos formas:
    - Regularización L1: El coste es proporcional al valor absoluto de los coeficientes de peso (norma del peso L1).
    - Regularización L2: El coste es proporcional al cuadrado de los coeficientes de peso (norma de peso L2).

Para decidir cuál de las dos aplicar a nuestro modelo, se recomienda tener en cuenta la siguiente información en relación con la naturaleza de nuestro problema:

In [2]:

```
import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.model_selection import cross_validate
```

In [3]:

```
#importamos un dataset de vehiculos
vehiculos = pd.read_csv("../../RESOURCES/vehiculos_procesado.csv")
datos_entrenamiento = vehiculos[["desplazamiento","cilindros","consumo"]]

objetivo = vehiculos["co2"]
datos_entrenamiento.head()
```

Out[3]:

	desplazamiento	cilindros	consumo
0	2.5	4.0	17
1	4.2	6.0	13
2	2.5	4.0	16
3	4.2	6.0	13
4	3.8	6.0	16

In [4]:

```
from sklearn.linear_model import(LinearRegression,Lasso,Ridge,ElasticNet)
```

In [5]:

```
modelo_ols = LinearRegression()
modelo_ols.fit(datos_entrenamiento,objetivo)
modelo_ols.coef_
```

Out[5]:

```
array([ 11.76787991,   1.23791071, -19.80355606])
```

In [6]:

```
"""Mediremos la complejidad del modelo con la norma L1 y la norma L2"""
def norma_l1(coeficientes):
    return np.abs(coeficientes).sum()

def norma_l2(coeficientes):
    return np.sqrt(np.power(coeficientes,2).sum())



"""Podemos utilizar también los métodos de numpy
que proporciona para estas normas"""
def norma_l1(coeficientes):
    return np.linalg.norm(coeficientes,ord=1)

def norma_l2(coeficientes):
    return np.linalg.norm(coeficientes,ord=2)

print(norma_l1(modelo_ols.coef_))
print(norma_l2(modelo_ols.coef_))
```

```
32.80934668020822
23.069379124497573
```

Veremos como regularizar la complejidad de los modelos en cuanto a número de parámetros

In [7]:

```
from sklearn.preprocessing import PolynomialFeatures
```

In [8]:

```
transformador_polinomial = PolynomialFeatures(5)
```

In [9]:

```
transformador_polinomial.fit(datos_entrenamiento)
```

Out[9]:

```
PolynomialFeatures(degree=5, include_bias=True, interaction_only=False)
```

In [10]:

```
#igual que utilizabamos el metodo predict utilizamos el método transform
variables_polinomiales=transformador_polinomial.transform(
    datos_entrenamiento)
```

In [11]:

```
variables_polinomiales.shape
```

Out[11]:

```
(35539, 56)
```

In [12]:

```
datos_entrenamiento.loc[0]
```

Out[12]:

```
desplazamiento     2.5
cilindros          4.0
consumo           17.0
Name: 0, dtype: float64
```

In [13]:

```
variables_polinomiales[0]
```

Out[13]:

```
array([1.000000e+00, 2.500000e+00, 4.000000e+00, 1.700000e+01,
       6.250000e+00, 1.000000e+01, 4.250000e+01, 1.600000e+01,
       6.800000e+01, 2.890000e+02, 1.562500e+01, 2.500000e+01,
       1.062500e+02, 4.000000e+01, 1.700000e+02, 7.225000e+02,
       6.400000e+01, 2.720000e+02, 1.156000e+03, 4.913000e+03,
       3.906250e+01, 6.250000e+01, 2.656250e+02, 1.000000e+02,
       4.250000e+02, 1.806250e+03, 1.600000e+02, 6.800000e+02,
       2.890000e+03, 1.228250e+04, 2.560000e+02, 1.088000e+03,
       4.624000e+03, 1.965200e+04, 8.352100e+04, 9.765625e+01,
       1.562500e+02, 6.640625e+02, 2.500000e+02, 1.062500e+03,
       4.515625e+03, 4.000000e+02, 1.700000e+03, 7.225000e+03,
       3.070625e+04, 6.400000e+02, 2.720000e+03, 1.156000e+04,
       4.913000e+04, 2.088025e+05, 1.024000e+03, 4.352000e+03,
       1.849600e+04, 7.860800e+04, 3.340840e+05, 1.419857e+06])
```

In [14]:

```
#Estos últimos pasos se suelen agrupar por convenio
variables_polinomiales = PolynomialFeatures(5).fit_transform(datos_entrenamiento)
```

In [15]:

```
variables_polinomiales.shape
```

Out[15]:

```
(35539, 56)
```

Modelo OLS con variables polinomiales

Comenzamos realizando una regresion lineal con todas esas variables sin ningun tipo de regularizacion

In [16]:

```
RESULTADOS = {}

modelo_ols = LinearRegression()
modelo_ols.fit(variables_polinomiales,objetivo)
print(modelo_ols.coef_)
RESULTADOS["ols"]={
    "norma_l1":norma_l1(modelo_ols.coef_),
    "norma_l2":norma_l2(modelo_ols.coef_)
}
```

```
[-1.33581691e-04 -2.16202007e+03 -4.74743025e+02 -9.33379425e+02
  2.06179368e+03 -1.98652141e+03  4.07183956e+02  6.98188922e+02
 -8.60206830e+00  5.68565581e+01 -5.04043554e+02  5.97572364e+02
 -2.39915727e+02 -1.93155159e+02  2.06936511e+02 -2.46366997e+01
 -7.53120039e+00 -6.16587972e+01  2.77167162e+00 -1.72213964e+00
  3.22404684e+01 -2.32860787e+01  3.67724866e+01 -1.28564531e+01
 -4.24157145e+01  9.08423002e+00  1.17166991e+01  1.52755499e+01
 -7.40121867e+00  6.04111957e-01 -9.27527797e-01 -8.19881989e-01
  1.96304324e+00 -8.55528671e-02  2.44388052e-02 -2.44193545e+00
  6.17851513e+00 -1.41371913e+00 -7.50158521e+00  1.59153882e+00
 -6.92569799e-01  5.18941968e+00 -3.95368960e-01  8.24178212e-01
 -1.15475754e-01 -1.85224537e+00 -2.10933021e-02 -3.68950001e-01
  9.99115993e-02 -5.66794738e-03  2.41619836e-01 -2.05795973e-02
  5.80537284e-02 -2.77634650e-02  1.11091113e-03 -1.26148048e-04]
```

Modelo Regularizacion L1 (de Lasso) con variables polinomiales

Como podemos ver en la documentacion, la clase lasso admite un parametro alpha (coeficiente de regularizacion) que indica la complejidad del modelo, donde 0 seria una regresion lineal normal y 1 regulariza al maximo. Tambien tenemos que tener en cuenta el *max_iter* que seria el maximo de iteraciones que necesita para considerar que ya ha aprendido suficiente. ademas tiene un parametro *tol* que indica la tolerancia.

In [17]:

```
Lasso?
```

In [18]:

```
modelo_l1 = Lasso(alpha=1.0,tol=0.01,max_iter=5000)
modelo_l1.fit(variables_polinomiales,objetivo)
print(modelo_l1.coef_)

RESULTADOS["regularizacion_l1"]={
    "norma_l1":norma_l1(modelo_l1.coef_),
    "norma_l2":norma_l2(modelo_l1.coef_)
}
```

```
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -3.30237023e+01
  5.12409278e+00  1.44708364e+00 -1.47969971e+00  6.52491862e-01
 -5.04276126e-01  5.26431228e-02 -1.48062992e-01 -0.00000000e+00
 -1.32137016e-01  1.02289253e-01 -9.17626300e-02 -1.30192905e-02
  6.15573125e-02 -3.01413839e-02  2.31053922e-02  5.17315962e-03
  1.91792946e-02 -1.51220456e-03 -1.88077866e-03 -1.09782669e-03
 -6.50140180e-03 -2.91026927e-04  1.73571147e-03 -3.33999083e-03
 -1.64104930e-03  1.59784747e-03  1.31929178e-03  1.69629743e-04
 -1.64562586e-03  2.03369130e-04  1.99415033e-05 -2.81072727e-03
  7.11467485e-04  2.46531750e-03 -2.20515928e-03  2.38235311e-04
  6.38712931e-04 -1.09015174e-03 -3.89575283e-04 -8.09552788e-05
  2.70933679e-04 -1.56154368e-04 -1.88340282e-04 -2.27097112e-04
  3.81560605e-05  2.29005451e-06 -5.56615246e-05  1.44114807e-04
  4.13806087e-06  5.51970343e-05  7.62940670e-07 -1.14293565e-06]
```

observamos que la regularizacion ha convertido muchos de ls parametros originales a 0.

Modelo Regularizacion L2 (Ridge) con variables polinomiales

Aunque la regularizacion l2 va mas rapido ponemos el mismo numero de iteraciones maximas y la misma tolerancia

In [19]:

```
modelo_l2 = Ridge(alpha=1.0,tol=0.01,max_iter=5000)
modelo_l2.fit(variables_polinomiales,objetivo)

print(modelo_l2.coef_)
RESULTADOS["regularizacion_l2"]={
    "norma_l1":norma_l1(modelo_l2.coef_),
    "norma_l2":norma_l2(modelo_l2.coef_)
}
```

```
[ 0.00000000e+00  1.51700145e+01  1.42697225e+00 -9.87021958e+00
  4.84917097e+01  3.40056811e+01  4.08617962e+01  1.87320285e+01
 -3.38485904e+01 -4.32030250e+00 -2.38651378e+01  2.49480682e+01
 -1.68754953e+01 -1.28017569e+01 -6.76114933e+00 -2.53957421e+00
 -4.79464960e+00  6.20674303e+00  2.06740524e+00  2.74904588e-01
  1.53996032e+01 -3.67044578e+01  3.58929869e+00  3.45188723e+01
 -4.38964155e+00  9.62648142e-01 -1.53176933e+01  4.03068321e+00
 -3.81950229e-02  4.61124419e-02  3.15167406e+00 -1.12387047e+00
 -2.48505114e-01 -3.05623638e-02 -7.73541022e-03 -2.33673851e+00
  6.80494202e+00 -8.69748260e-01 -7.95078796e+00  2.20022716e+00
 -1.37697480e-01  4.60214517e+00 -2.17510167e+00  2.25462479e-01
 -1.86520095e-02 -1.28785706e+00  9.40202486e-01 -1.93271866e-01
  1.34228047e-02 -4.68455498e-04  1.24956055e-01 -1.58111335e-01
  5.90665314e-02 -2.26320625e-03  2.49827232e-04  8.13219567e-05]
```

```
/opt/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/ridge.py:125: LinAlgWarning:
scipy.linalg.solve
Ill-conditioned matrix detected. Result is not guaranteed to be accurate.
Reciprocal condition number4.797447e-19
  overwrite_a=True).T
```

Modelo Regularizacion ElasticNet con variables polinomiales

Tiene un parametro l1_ratio que es "elastico" donde 1 sera regularizacion l1 y si pones 0 sera regularizacion l2. En nuestro caso pondremos el mismo peso entre regularizacion de lasso y de ridge

In [20]:

```
modelo_elasticnet = ElasticNet(l1_ratio=0.5,tol=0.01,max_iter=5000)
modelo_elasticnet.fit(variables_polinomiales,objetivo)

print(modelo_elasticnet.coef_)
```

```
print(modelo_elasticnet.coef_)
RESULTADOS["regularizacion_elasticnet"]={
    "norma_l1":norma_l1(modelo_elasticnet.coef_),
    "norma_l2":norma_l2(modelo_elasticnet.coef_)
}
```

```
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.33310440e+00
  4.89645007e+00  6.09439149e+00 -1.31816563e+00  2.37723778e+00
 -3.10050434e+00 -6.41934303e-01 -7.84374220e-01 -9.59081564e-02
 -7.18954629e-02  7.19478059e-03 -4.57252233e-02 -6.37787774e-02
  1.22426563e-02  3.46550752e-02  4.26955780e-02  2.66673958e-03
  1.39812144e-02 -8.09891082e-03  8.76409316e-03 -1.69690074e-02
 -2.05177215e-03  1.54020194e-03 -6.29758134e-03 -2.92855315e-03
 -6.86058449e-04  1.42133650e-03 -9.92879254e-04  1.28704557e-04
  5.03220762e-04  6.61024458e-04  5.80200778e-05  3.41020839e-03
  2.27563082e-03  2.71125911e-03 -1.90350591e-03 -7.82250851e-05
  6.56665428e-04 -7.34759017e-04 -9.12167373e-04 -2.43120629e-04
  1.60164558e-04  3.89147755e-04 -5.45731423e-04 -3.49391006e-04
  3.45240438e-05  3.46406797e-05  2.59823852e-04 -6.35175029e-05
 -9.14762151e-05  5.08095098e-05  1.14384625e-05  7.36969809e-07]
```

Visualizacion de los Resultados

In [21]:

```
pd.set_option("display.float_format",lambda x:str(round(x,6)))
```

In [22]:

```
resultados_df = pd.DataFrame(RESULTADOS).T
l1_ols = resultados_df.loc["ols","norma_l1"]
l2_ols = resultados_df.loc["ols","norma_l2"]

resultados_df["pct_reduccion_l1"]=1-resultados_df.norma_l1/l1_ols
resultados_df["pct_reduccion_l2"]=1-resultados_df.norma_l2/l2_ols

resultados_df
```

Out[22]:

	norma_l1	norma_l2	pct_reduccion_l1	pct_reduccion_l2
ols	10853.747955	3922.194711	0.0	0.0
regularizacion_l1	42.945147	33.494056	0.996043	0.99146
regularizacion_l2	457.523205	109.55068	0.957847	0.972069
regularizacion_elasticnet	21.002924	8.997096	0.998065	0.997706

In [23]:

```
#exportamos el dataset a un csv
resultados_df.to_csv("resultados_regularizacion.csv")
```