Normalizar los valores de las características

Trasnformación a un intervalo

Cuando tenemos datos como los de 'delta_elevators' cada característica se mueve en unos rangos diferentes y eso puede ser problemático para algunos de los algoritmos de aprendizaje que nos puede interesar aplicar después.

In [53]:
```
# Importar los paquetes que usaremos
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn import preprocessing
```

In [54]:
```
# Cargar el conjunto de datos
dataset = datasets.fetch_openml(name='delta_elevators', version=1, as_frame=True)
tabla = dataset.frame
tabla
```

Out[54]:

	climbRate	Altitude	RollRate	curRoll	diffClb	diffDiffClb	Se
0	2.0	-50.0	-0.0048	-0.001	0.2	0.00	-0.001
1	6.5	-40.0	-0.0010	-0.009	0.2	0.00	0.003
2	-5.9	-10.0	-0.0033	-0.004	-0.1	0.00	-0.001
3	-6.2	-30.0	-0.0022	-0.011	0.1	0.00	-0.002
4	-0.2	-40.0	0.0059	-0.005	0.1	0.00	0.001
...	...	...	...	...	...	...	...
9512	5.0	-30.0	0.0013	-0.004	0.2	0.00	0.004
9513	1.4	0.0	0.0024	0.019	-0.2	-0.01	-0.001
9514	-3.5	-10.0	-0.0082	0.004	-0.1	0.00	-0.003
9515	-2.4	-10.0	-0.0065	-0.012	0.2	-0.02	-0.001
9516	4.7	-10.0	0.0018	-0.020	0.3	0.00	0.001

9517 rows × 7 columns

In [55]:
```
tabla.describe()
```

Out[55]:

	climbRate	Altitude	RollRate	curRoll	diffClb	diffDiffClb	Se
count	9517.000000	9517.000000	9517.000000	9517.000000	9517.000000	9517.000000	9517.000000
mean	-0.289881	-12.304298	0.000045	0.001632	-0.000389	-0.000196	-0.000133
std	4.912074	25.496266	0.005295	0.015916	0.176858	0.006078	0.002374
min	-15.000000	-100.000000	-0.023700	-0.051000	-0.800000	-0.030000	-0.014000
25%	-3.900000	-30.000000	-0.004000	-0.009000	-0.100000	0.000000	-0.002000
50%	-0.900000	-10.000000	0.000100	0.002000	0.000000	0.000000	0.001000
75%	3.500000	0.000000	0.004100	0.012000	0.100000	0.000000	0.001000
max	15.100000	90.000000	0.018400	0.049000	0.700000	0.020000	0.013000

Si vamos a trabajar con scikit-learn o cualquiera que use como entrada ndarrays podemos usar el módulo de preprocesado de scikit-learn (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing). Scikit-learn trabaja con objetos transformadores. Se crea el objeto con su configuración y luego se llama a sus métodos para que hagan la transformación o aprendizaje que corresponda. En este caso creamos un objeto `scaler` configurado para el intervalo [-1, 1] y luego lo llamamos, primero a su método `fit`para que ajuste sus parámetros internos (mejora la eficiencia en algunos casos) y luego a `transform` para que nos devuelva los datos escalados:

In [56]:
```
scaler = preprocessing.MinMaxScaler(feature_range=(-1,1))
scaler.fit(tabla)
scaler.transform(tabla)
```

Out[56]:
```
array([[ 0.12956811, -0.47368421, -0.10213777, ...,  0.33333333,
         0.2       , -0.03703704],
       [ 0.42857143, -0.36842105,  0.0783848 , ...,  0.33333333,
         0.2       ,  0.25925926],
       [-0.39534884, -0.05263158, -0.03087886, ..., -0.06666667,
         0.2       , -0.03703704],
       ...,
       [-0.2358804 , -0.05263158, -0.26365796, ..., -0.06666667,
         0.2       , -0.18518519],
       [-0.1627907 , -0.05263158, -0.18289786, ...,  0.33333333,
        -0.6       , -0.03703704],
       [ 0.3089701 , -0.05263158,  0.21140143, ...,  0.46666667,
         0.2       ,  0.11111111]])
```

Si queremos conservar la estructura de *DataFrame* de Pandas, habrá que hacerlo programando pero es una transformación muy sencilla:

In [57]:
```
def escalado_intervalo(a, b, columna):
    """Escala la columna al intervalo [a,b]"""
    cero_uno = (columna - columna.min()) / (columna.max() - columna.min())
    return cero_uno * (b-a) + a

for col in tabla.columns:
    tabla[col] = escalado_intervalo(-2, 2, tabla[col])

tabla
```

Out[57]:

	climbRate	Altitude	RollRate	curRoll	diffClb	diffDiffClb	Se
0	0.259136	-0.947368	-0.204276	-2.220446e-16	0.666667	0.4	-0.074074
1	0.857143	-0.736842	0.156770	-3.200000e-01	0.666667	0.4	0.518519
2	-0.790698	-0.105263	-0.061758	-1.200000e-01	-0.133333	0.4	-0.074074
3	-0.830565	-0.526316	0.042755	-4.000000e-01	0.400000	0.4	-0.222222
4	-0.033223	-0.736842	0.812352	-1.600000e-01	0.400000	0.4	0.222222
...	...	...	...	...	...	...	...
9512	0.657807	-0.526316	0.375297	-1.200000e-01	0.666667	0.4	0.666667
9513	0.179402	0.105263	0.479810	8.000000e-01	-0.400000	-0.4	-0.074074
9514	-0.471761	-0.105263	-0.527316	2.000000e-01	-0.133333	0.4	-0.370370
9515	-0.325581	-0.105263	-0.365796	-4.400000e-01	0.666667	-1.2	-0.074074
9516	0.617940	-0.105263	0.422803	-7.600000e-01	0.933333	0.4	0.222222

9517 rows × 7 columns

Transformación a valores normales

Hay quien se refiere a las transformaciones anteriores como normalizaciones. Sin embargo, es mejor identificarlas como transformación a un intervalo y reservar 'normalizar' para las transformaciones en las que se intenta que los datos estén distribuidos con una distribución Normal de media 0 y desviación típica 1.

La transformación más simple es un escalado, como los de intervalo que acabamos de hacer pero usando como factor el inverso de la desviación típica muestral y como desplazamiento el opuesto de la media:

In [58]:
```
def escalado_intervalo(columna):
    """Escala la columna al intervalo [a,b]"""
    return columna / columna.std() - columna.mean()

for col in tabla.columns:
    tabla[col] = escalado_intervalo(tabla[col])

tabla
```

Out[58]:

	climbRate	Altitude	RollRate	curRoll	diffClb	diffDiffClb	Se
0	0.442148	-1.611190	-0.662171	-0.105277	1.281267	0.438368	-0.264889
1	1.358258	-1.218975	0.055543	-0.607914	1.281267	0.438368	1.419787
2	-1.166134	-0.042332	-0.378863	-0.293766	-0.415009	0.438368	-0.264889
3	-1.227208	-0.826761	-0.171104	-0.733573	0.715842	0.438368	-0.686058
4	-0.005728	-1.218975	1.358760	-0.356595	0.715842	0.438368	0.577449
...	...	...	...	...	...	...	...
9512	1.052888	-0.826761	0.489948	-0.293766	1.281267	0.438368	1.840956
9513	0.320000	0.349882	0.697707	1.151316	-0.980435	-1.206930	-0.264889
9514	-0.677542	-0.042332	-1.304336	0.208871	-0.415009	0.438368	-1.107227
9515	-0.453604	-0.042332	-0.983254	-0.796403	1.281267	-2.852227	-0.264889
9516	0.991814	-0.042332	0.584384	-1.299040	1.846692	0.438368	0.577449

9517 rows × 7 columns

Esta es la misma transforamción que aplica `sklearn.preprocessing.StandardScaler` pero retornando el ndarray sin los metadatos de Pandas (los nombres de las columnas). Sin embargo, esta transformación de escalado no tiene en cuenta la forma de la distribución y, por tanto, los valores no estarán distribuidos normalmente salvo que ya lo estuviesen originalmente.

Scikit-learn incluye dentro de su módulo de preprocesado de scikit-learn (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing) dos clases que permiten ajustar los datos a una distribución normal (manteniendo el orden de los valores para no perder la información):

- PowerTransformer
- QuantileTransformer

QuantileTransformer también permite ajustar los datos a una distribución uniforme.

Ejercicio: Mirando su documentación, usa ambas clases de trasnformadores para normalizar alguna de las variables del conjunto de datos con el que estamos trabajando. Pásale luego el test que vimos en el tema 1.5 para comprobar si los datos siguen una distribución normal.

Ejercicio: El código que hemos utilizado para el escalado en Pandas no nos sirve si hay variables categóricas. Revisando los ejemplos del tema 1.3 y la documentación sobre los tipos de columnas en Pandas, adaptalo para que sirva en cualquier objeto Pandas `DataFrame` (ignorando las columnas categoricas).