Discretización de características

En algunos casos podemos necesitar que una caracteristica (o todas) sea discreta o categórica. Podemos convertir la variables continuas en discretas estableciendo varios intervalos y asignando valores a la variable según al intervalo que perteneciese el valor original. Por ejemplo, trabajando con el mismo conjunto de datos que en el apartado anterior:

In [18]:
```
# Importar los paquetes que usaremos
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn import preprocessing
```

In [19]:
```
# Cargar el conjunto de datos
dataset = datasets.fetch_openml(name='delta_elevators', version=1, as_frame=True)
tabla = dataset.frame
tabla
```

Out[19]:

	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

Podemos discretizar la columna 'climbRate' usando Scikit-learn con el siguiente código. Nota, como Scikit-learn siempre trabaja con ndarrays de numeros en coma flotante, la salida del discretizador, aunque sólo contiene valores enteros están representados en números flotantes. Por esta razón, lo convertimos con `astype` antes de reasignar la columna al DataFrame de Pandas.

In [20]:
```
discretizador = preprocessing.KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='uniform')
tabla['climbRate'] = discretizador.fit_transform(tabla[['climbRate']]).astype('int')
tabla
```

Out[20]:

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

9517 rows × 7 columns

Los parámetros son: `n_bins` que establece el número de intervalos, `encode` que establece posibles codificaciones de las variables categoricas (lo veremos en el apartado siguiente, Tema 2.3) y `strategy` que establece el tipo de discretización entre los siguientes, que son los más comunmente usados:

- `'uniform'` los intervalos son todos del mismo tamaño, simplemente parte en trozos iguales el intervalo entre los valores mínimo y máximo de la característica.
- `'quantile'` es un método que deja aproximadamente (salvo restos) el mismo número de muestras en cada intervalo, también se le denomina discretización por frecuencias y puede ser útil cuando los valores están lejos de una distribución uniforme. Por ejemplo, si hay muchos valores concentrados y otros dispersos, puede que la discretización uniforme haga perder información importante.
- `'kmeans'` utiliza un algoritmo de agrupamiento (o *clustering*), que se verá en otras asignaturas, para agrupar los valores por semejanza y fijar así los intervalos para no partir muestras que estén muy juntas

Ejercicio: Leer este artículo sobre discretización en Pandas (http://exponentis.es/discretizacion-de-datos-en-python-manteniendo-el-nombre-de-las-columnas) y programar como discretizar la tabla anterior completa.

Ejercicio: Explorar los diferentes métodos de discretización y usar el método `.describe()` sobre la tabla para ver como afectan a la distribución de los valores discretizados.