Transformaciones sobre las características

Algunos algoritmos de aprendizaje están diseñados para trabajar sólo con características continuas. Por tanto, puede interesarnos cambiar el tipo de todas las características a continuas. De hecho, scikit-learn funciona siempre con ndarrays de Numpy con valores flotantes (representación de los números reales).

Continuización de características ordinales

Cuando tenemos una caracteristica con valores que respresentan un orden. Nos interesa que esa información no se pierda. Por ejemplo, en la siguiente tabla:

In [43]:
```
import pandas as pd

tabla = pd.DataFrame(
    {'altura': ["alto", "bajo", "alto", "medio"],
     'peso'  : [81.3, 76.4, 73.2, 82.9],
     'puntos': [90, 40, 80, 98]}
)
tabla
```

Out[43]:

	altura	peso	puntos
0	alto	81.3	90
1	bajo	76.4	40
2	alto	73.2	80
3	medio	82.9	98

No tendría mucho sentido asignar: alto -> 0, bajo -> 1, medio -> 2. Probablemente nos interesará asignar un valor numérico mayor a 'alto' que a 'bajo' por semantica pero eso no representa un problema para los algoritmos de aprendizaje, que simplemente captarían un orden inverso. Lo que sí es muy importante es que el orden se mantenga y 'medio' sea un valor entre ambos porque, aunque un algoritmo de aprendizaje podría llegar a aprender esta distorsión, se lo estaríamos poniendo mucho más difícil.

Entonces, podemos convertir esa columna así:

In [44]:
```
def continuizar(traduccion, columna):
    """Cambia los valores de columna segun la tabla de traduccion dada como dicci
onario"""
    return [traduccion[value] for value in columna]

tabla['altura'] = continuizar({'bajo' : 0.0, 'medio' : 1.0, 'alto' : 2.0}, tabla
['altura'])

tabla
```

Out[44]:

	altura	peso	puntos
0	2.0	81.3	90
1	0.0	76.4	40
2	2.0	73.2	80
3	1.0	82.9	98

Ejercicio: Crea el código para traducir todas las columnas de la siguiente tabla de ejemplo a valores reales que sean razonables. Utiliza una función a la que se le pase un diccionario con los diccionarios de traducción de todas las columnas.

Ejemplo:

```
{'presion' : {'nula' : 0.0, 'baja' : 1.0, 'media' : 2.0, ...},
 'voltaje' : { ... },
 ...
}
```

In [45]:
```
import pandas as pd

control = pd.DataFrame(
    {'presion'  : ['nula', 'baja', 'media', 'nula', 'alta', 'muy alta', 'baja', '
baja'],
     'voltaje'  : [8.3, 6.4, 3.1, 8.2, 2.8, 11.1, 0.2, 0.1],
     'velocidad': ['rapida', 'optima', 'lenta', 'lenta', 'rapida', 'optima', 'len
ta', 'rapida'],
     'distancia': ['distante', 'cercano', 'cercano', 'distante', 'muy distante',
'cercano', 'cercano', 'distante']}
)
control
```

Out[45]:

	presion	voltaje	velocidad	distancia
0	nula	8.3	rapida	distante
1	baja	6.4	optima	cercano
2	media	3.1	lenta	cercano
3	nula	8.2	lenta	distante
4	alta	2.8	rapida	muy distante
5	muy alta	11.1	optima	cercano
6	baja	0.2	lenta	cercano
7	baja	0.1	rapida	distante

Binarización de características categóricas

Algunos algoritmos de aprendizaje no pueden trabajar con características categóricas y otros dan peores resultados. Mientras que las caracteristicas ordinales representan un orden y tiene sentido convertirla a valores de un intervalo continuo. Si una característica categórica se transforma de la misma forma, se está estableciendo un orden irreal entre valores que no tienen por qué tenerlo. Por ejemplo, una categoría profesión como la de la siguiente tabla no tiene un orden definido.

En estos caso se puede utilizar lo que se denomina como *one hot encoding* o convertir los valores de la categoría en varias características binarias.

In [46]:
```
import pandas as pd

transporte = pd.DataFrame(
    {'profesion': ['albañil', 'profesor', 'electricista', 'administrativo', 'elec
tricista', 'electricista'],
     'ratio_rb' : [0.3, 0.4, 0.1, 0.2, 0.3, 0.1],
     'distancia': [5.4, 10.8, 2.7, 4.3, 1.2, 7.5]}
)
transporte
```

Out[46]:

	profesion	ratio_rb	distancia
0	albañil	0.3	5.4
1	profesor	0.4	10.8
2	electricista	0.1	2.7
3	administrativo	0.2	4.3
4	electricista	0.3	1.2
5	electricista	0.1	7.5

Podríamos hacerlo programandolo a mano y puede ser conveniente para ello conocer las formas de crear nuevas columnas en Pandas (https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/05_add_columns.html#min-tut-05-columns) pero también tenemos la función `get_dummies()` de Pandas que hace justo eso:

In [47]:
```
pd.get_dummies(transporte)
```

Out[47]:

	ratio_rb	distancia	profesion_administrativo	profesion_albañil	profesion_electricista	profesion_p
0	0.3	5.4	0	1	0	0
1	0.4	10.8	0	0	0	1
2	0.1	2.7	0	0	1	0
3	0.2	4.3	1	0	0	0
4	0.3	1.2	0	0	1	0
5	0.1	7.5	0	0	1	0

Ejercicio: Localiza la función que tiene esta misma función en el módulo de preprocesado (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing) de Scikit-learn y aplicala sobre estos datos para obtener un ndarray equivalente a la tabla anterior.