Programación para *Data Science*

Unidad 6: Preprocesamiento de datos en Python

Instrucciones de uso

A continuación se presentarán explicaciones y ejemplos de preprocesamiento de datos en Python. Recordad que podéis ir ejecutando los ejemplos para obtener sus resultados.

Introducción

En este módulo trabajaremos con la librería pandas (http://pandas.pydata.org/), que ya hemos introducido en módulos anteriores, y scikit-learn (http://scikit-learn.org), una nueva librería que presentamos en este módulo. Scikit-learn es una librería de aprendizaje automático de Python que nos ofrece herramientas y implementaciones de algoritmos para minería y análisis de datos. En la propia web de scikit-learn podéis encontrar la documentación completa (http://scikit-learn.org/stable/documentation.html) de la librería.

Este Notebook contiene ejemplos concretos de técnicas que pueden aplicarse para preprocesar datos para cada uno de los grupos de técnicas descritos en la introducción del módulo (en la xwiki asociada). Es importante destacar que se han seleccionado únicamente algunas técnicas dentro de cada grupo para presentar ejemplos del tipo de transformaciones que se realizan pero, en la práctica, el conjunto de técnicas que se aplican en el preprocesamiento de los datos es mucho más amplio.

En este Notebook veremos cómo aplicar diferentes técnicas de preprocesamiento de datos sobre un conjunto de datos metereológicos de la ciudad de Beijing. El *dataset* original puede encontrarse en el siguiente repositorio de Machine Learning de la UC Irvine (http://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data#), aunque para las actividades utilizaremos una variante modificada del mismo que nos permitirá practicar un conjunto más amplio de técnicas. Podéis encontrar una pequeña descripción de los atributos del conjunto de datos siguiendo el enlace anterior.

Primeros pasos

En primer lugar, cargamos el conjunto de datos:

In [2]:
```
# Importamos la librería pandas
import pandas as pd

# Cargamos los datos del fichero "weather_dataset_edited.csv" en un dataframe
data = pd.read_csv("data/weather_dataset_edited.csv")

# Mostramos una descripción básica de los datos cargados
print type(data)
print len(data)
data.head(n=5)
```

```
<class 'pandas.core.frame.DataFrame'>
43824
```

Out[2]:

	No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	cbwd	Iws	Is	Ir
0	1	2010	jan	1	0	NaN	-21	-11	1021	Nw	1.79	0	0
1	2	2010	jan	1	1	NaN	-21	-12	1020	nw	4.92	0	0
2	3	2010	jan	1	2	NaN	-21	-11	1019	nw	6.71	0	0
3	4	2010	jan	1	3	NaN	-21	-14	1019	NW	9.84	0	0
4	5	2010	jan	1	4	NaN	-20	-12	1018	nW	12.97	0	0

Integración de datos

El conjunto de datos ha sido creado con la colaboración de diferentes personas. Aunque todas ellas anotaban la misma información, lo cierto es que utilizaron una nomenclatura distinta para describir la dirección del viento. Veamos cómo podemos unificar la nomenclatura usada por todos ellos.

In [3]:
```
# Visualizamos las diferentes abreviaturas utilizadas
set(data["cbwd"])
```

Out[3]: {nan, 'NE', 'NW', 'Nw', 'SE', 'Se', 'nW', 'ne', 'nw', 'sE', 'se'}

In [4]:
```
# Unificamos la nomenclatura para usar únicamente mayúsculas
data.loc[data.cbwd == "ne", "cbwd"] = "NE"
data.loc[(data.cbwd == "Nw") | (data.cbwd == "nW") | (data.cbwd == "nw"), "cbwd"] = "NW"
data.loc[(data.cbwd == "Se") | (data.cbwd == "sE") | (data.cbwd == "se"), "cbwd"] = "SE"
```

Notad que usamos el operador `.loc`, que habíamos visto en el módulo 4 (en las explicaciones sobre la librería pandas) para filtrar las filas que cumplen una característica concreta (por ejemplo, para la primera sentencia, que tienen el valor "ne" en el campo `cbwd`) y luego seleccionamos únicamente la columna "cbwd" para poder asignarle el nuevo valor (en este caso, "NE").

In [5]:
```
# Comprobamos que la sustitución se haya realizado correctamente
set(data["cbwd"])
```

Out[5]: {nan, 'NE', 'NW', 'SE'}

Además, sabemos que normalmente la temperatura se tomaba con un termómetro configurado para usar el sistema métrico internacional, por lo que esta se encuentra expresada en grados Celsius. Sin embargo, durante el año 2011 se estuvieron tomando las mediciones con otro termómetro configurado con grados Farenheit, por lo que las muestras de ese año se encuentran expresadas en °F. Veamos cómo podemos unificar las mediciones de temperatura.

In [6]:
```
# Importamos la librería numpy
import numpy as np

# Visualizamos la media anual de las temperaturas
grouped = data.groupby("year")
grouped.aggregate({"TEMP": np.mean})
```

Out[6]:

	TEMP
year	
2010	11.632420
2011	54.617534
2012	11.967441
2013	12.399201
2014	13.679566

Fijaos como, efectivamente, la media del año 2011 es mucho más alta que la del resto de años.

In [7]:
```
# Definimos una función que convierte grados farenheit en grados celsius
def farenheit_to_celsius(x):
    return (x-32)*5/9

# Sustituimos los valores de las temperaturas del año 2011 por el resultado de aplicar la función
# farenheit_to_celsius al valor actual
data.loc[data.year == 2011, "TEMP"] = data[data.year == 2011]["TEMP"].apply(farenheit_to_celsius)
```

In [8]:
```
# Comprobamos que los cambios realizados han tenido efecto
grouped.aggregate({"TEMP": np.mean})
```

Out[8]:

	TEMP
year	
2010	11.632420
2011	12.565297
2012	11.967441
2013	12.399201
2014	13.679566

Transformación de datos

Los atributos `month` y `cbwd` contienen cadenas de caracteres como valores y representan variables categóricas, por lo que algunos tipos de algoritmos de minería de datos no podrán trabajar con ellas. Por ello, las transformaremos en un conjunto de atributos binarios (un atributo para cada categoría posible).

In [9]:
```
# Mostramos el conjunto de atributos original
print list(data)
```

```
['No', 'year', 'month', 'day', 'hour', 'pm2.5', 'DEWP', 'TEMP', 'PRES', 'cbwd', 'Iws', 'Is', 'Ir']
```

In [10]:
```
# Creamos nuevos atributos binarios para las categorías utilizadas en las columnas "month" y "cbwd"
data_trans = pd.get_dummies(data, columns=["month", "cbwd"], dummy_na=True)
```

In [11]:
```
# Mostramos el conjunto de atributos después de la transformación
print list(data_trans)
```

```
['No', 'year', 'day', 'hour', 'pm2.5', 'DEWP', 'TEMP', 'PRES', 'Iws', 'Is', 'Ir', 'month_apr', 'month_aug', 'month_de
c', 'month_feb', 'month_jan', 'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep
t', 'month_nan', 'cbwd_NE', 'cbwd_NW', 'cbwd_SE', 'cbwd_nan']
```

Podemos ver un ejemplo de cómo se han transformado los valores observando algunas muestras concretas. Así, para las muestras entre las posiciones 10 y 20 y la columna `cbwd`:

In [12]:
```
# Mostramos el valor de la columna "cbwd" original para las muestras entre las posiciones 10 y 20
print data.loc[10:20, ["cbwd"]]

# Mostramos los valores de las nuevas columnas "cbwd_NE", "cbwd_NW", "cbwd_SE", "cbwd_nan"
# para las muestras entre las posiciones 10 y 20
data_trans.loc[10:20, ["cbwd_NE", "cbwd_NW", "cbwd_SE", "cbwd_nan"]]
```

```
   cbwd
10   NW
11   NW
12   NW
13   NW
14   NW
15  NaN
16   NW
17   NW
18   NE
19   NW
20  NaN
```

Out[12]:

	cbwd_NE	cbwd_NW	cbwd_SE	cbwd_nan
10	0	1	0	0
11	0	1	0	0
12	0	1	0	0
13	0	1	0	0
14	0	1	0	0
15	0	0	0	1
16	0	1	0	0
17	0	1	0	0
18	1	0	0	0
19	0	1	0	0
20	0	0	0	1

Limpieza de datos

Uno de los problemas que se tratan en la limpieza de datos es el tratamiento de valores perdidos. Existen múltiples estrategias para tratar con estos valores, desde directamente eliminar las muestras que contienen algún valor perdido hasta sustituir los valores perdidos por algún otro valor (por ejemplo, para atributos numéricos, la media del atributo en el resto de muestras). Veamos un ejemplo de sustitución de valores perdidos por la media del atributo.

En primer lugar, identificamos los atributos que tienen algún valor NaN:

In [13]:
```
# Definimos una función que nos retorna un valor booleano indicando si alguno de los valores
# de la serie es nan
def any_is_null(x):
    return any(pd.isnull(x))

# Aplicamos la función any_is_null a cada columna del dataframe
print data_trans.apply(any_is_null)
```

```
No            False
year          False
day           False
hour          False
pm2.5          True
DEWP          False
TEMP          False
PRES          False
Iws           False
Is            False
Ir            False
month_apr     False
month_aug     False
month_dec     False
month_feb     False
month_jan     False
month_jul     False
month_jun     False
month_mar     False
month_may     False
month_nov     False
month_oct     False
month_sept    False
month_nan     False
cbwd_NE       False
cbwd_NW       False
cbwd_SE       False
cbwd_nan      False
dtype: bool
```

Notad que aunque la columna `cbwd` original contenía valores perdidos, después de la transformación ya no los tenemos ya que estos se encuentran representados con valores binarios en la columna `cbwd_nan`. Así, únicamente será necesario tratar los valores perdidos de la columna `pm2.5`.

Procedemos a sustituir los valores perdidos de la columa `pm2.5` por la media de la columna utilizando la librería sklearn (aunque también podríamos utilizar las funciones de indexación de pandas para conseguir el mismo objetivo).

In [14]:
```
# Importamos Imputer del módulo de preprocesamiento de la librería sklearn
from sklearn.preprocessing import Imputer

# Sustituiremos los valores perdidos por la media de la columna (el parámetro axis=0 indica que calcularemos
# la media sobre la columna)
imp = Imputer(strategy='mean', axis=0)

# Aplicamos la transformación a la columna pm2.5
data_trans["pm2.5"] = imp.fit_transform(data_trans[["pm2.5"]]).ravel()
```

In [15]:
```
# Comprobamos que se han eliminado los valores perdidos
print data_trans.apply(any_is_null)
```

```
No            False
year          False
day           False
hour          False
pm2.5         False
DEWP          False
TEMP          False
PRES          False
Iws           False
Is            False
Ir            False
month_apr     False
month_aug     False
month_dec     False
month_feb     False
month_jan     False
month_jul     False
month_jun     False
month_mar     False
month_may     False
month_nov     False
month_oct     False
month_sept    False
month_nan     False
cbwd_NE       False
cbwd_NW       False
cbwd_SE       False
cbwd_nan      False
dtype: bool
```

Normalización de datos

Una de las alternativas para normalizar los datos consiste en centrar los valores para que la media del atributo se encuentre cercana a cero y escalarlos para que la varianza sea 1. Veamos cómo realizar este proceso sobre el atributo que contiene la presión atmosférica.

In [16]:
```
# Observamos los estadísticos básicos originales del atributo "PRES"
data_trans["PRES"].describe()
```

Out[16]:
```
count    43824.000000
mean      1016.447654
std         10.268698
min        991.000000
25%       1008.000000
50%       1016.000000
75%       1025.000000
max       1046.000000
Name: PRES, dtype: float64
```

In [17]:
```
# Importamos StandardScaler del módulo de preprocesamiento de la librería sklearn
from sklearn.preprocessing import StandardScaler

# Utilizamos el StandardScaler de sklearn para normalizar los valores del atributo "PRES"
data_trans.loc[:, ["PRES"]] = StandardScaler().fit_transform(data_trans.loc[:, ["PRES"]])
```

In [18]:
```
# Observamos los estadísticos básicos del atributo "PRES" después de la transformación
data_trans["PRES"].describe()
```

Out[18]:
```
count    4.382400e+04
mean     4.851095e-15
std      1.000011e+00
min     -2.478206e+00
25%     -8.226701e-01
50%     -4.359456e-02
75%      8.328654e-01
max      2.877939e+00
Name: PRES, dtype: float64
```

Notad como, efectivamente, la media se aproxima ahora al valor 0 y la desviación a 1.

Reducción de dimensiones

Una opción sencilla para reducir dimensiones consiste en seleccionar un conjunto de características de interés. Podemos realizar esta selección de manera sencilla gracias a las funciones que disponemos sobre los *dataframes* de pandas.

In [19]:
```
# Mostramos los atributos actuales
print list(data_trans)
```

```
['No', 'year', 'day', 'hour', 'pm2.5', 'DEWP', 'TEMP', 'PRES', 'Iws', 'Is', 'Ir', 'month_apr', 'month_aug', 'month_de
c', 'month_feb', 'month_jan', 'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep
t', 'month_nan', 'cbwd_NE', 'cbwd_NW', 'cbwd_SE', 'cbwd_nan']
```

In [20]:
```
# Eliminamos el atributo "DEWP"
data_trans = data_trans.drop("DEWP", axis=1)
```

In [21]:
```
# Mostramos los atributos después del cambio
print list(data_trans)
```

```
['No', 'year', 'day', 'hour', 'pm2.5', 'TEMP', 'PRES', 'Iws', 'Is', 'Ir', 'month_apr', 'month_aug', 'month_dec', 'mont
h_feb', 'month_jan', 'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sept', 'mont
h_nan', 'cbwd_NE', 'cbwd_NW', 'cbwd_SE', 'cbwd_nan']
```

Un grupo de técnicas de reducción de dimensiones muy desarrollado se centra en la extracción de características. Aunque conceptualmente estos procesos se escapan de este curso introductorio, lo cierto es que es fácil aplicar estas técnicas con sklearn. El lector interesado puede consultar los ejemplos (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#examples-using-sklearn-decomposition-pca) de la propia documentación de sklearn.

Reducción de muestras

Una alternativa sencilla para realizar una reducción de las muestras disponibles consiste en seleccionar de manera aleatoria uniforme un subconjunto de muestras del *dataset*.

In [22]:
```
# Mostramos el número de muestras original
print len(data_trans)
```

```
43824
```

In [23]:
```
# Seleccionamos un 25% de las muestras de manera aleatoria
sampled_data = data_trans.sample(frac=0.25)
```

In [24]:
```
# Mostramos el número de muestras seleccionado
print len(sampled_data)

# Mostramos las 5 primeras muestras seleccionadas
sampled_data.head(n=5)
```

```
10956
```

Out[24]:

	No	year	day	hour	pm2.5	TEMP	PRES	Iws	Is	Ir	...	month_mar	month_may	month_nov	month_oct	month_sept	month_nan	cb
6045	6046	2010	9	21	144	21	-0.627901	31.74	0	0	...	0	0	0	0	1	0	0
37715	37716	2014	21	11	39	22	0.151174	1.79	0	0	...	0	0	0	0	0	0	1
4802	4803	2010	20	2	53	22	-1.309592	1.79	0	4	...	0	0	0	0	0	0	1
3072	3073	2010	9	0	139	15	-0.920055	1.79	0	1	...	0	1	0	0	0	0	0
41377	41378	2014	21	1	200	20	-0.238363	15.21	0	0	...	0	0	0	0	1	0	0

5 rows × 27 columns

Notad que el *dataframe* conserva el número de atributos original, pero solo contiene un 25% de las muestras originales.

Discretización

En ocasiones nos interesará convertir un atributo continuo en uno de discreto. Una manera de hacerlo es divir el espacio de posibles valores que toma el atributo en n *bins* o intervalos del mismo tamaño y asignar cada muestra al intervalo al que pertenece. Veamos un ejemplo discretizando el atributo `Iws` en 5 intervalos del mismo tamaño.

In [25]:
```
# Observamos los estadísticos básicos del atributo "Iws"
data_trans["Iws"].describe()
```

Out[25]:
```
count    43824.000000
mean        23.889140
std         50.010635
min          0.450000
25%          1.790000
50%          5.370000
75%         21.910000
max        585.600000
Name: Iws, dtype: float64
```

In [26]:
```
# Creamos un nuevo atributo "Iws_disc" que contiene la discretización de "Iws"
data_trans["Iws_disc"] = pd.cut(data_trans["Iws"], 5)
```

In [27]:
```
# Visualizamos el contenido de los atributos "Iws" y "Iws_disc" para un subconjunto de muestras
# para observar el resultado
data_trans.loc[80:90, ["Iws", "Iws_disc"]]
```

Out[27]:

	Iws	Iws_disc
80	80.90	(-0.135, 117.48]
81	90.73	(-0.135, 117.48]
82	100.56	(-0.135, 117.48]
83	108.61	(-0.135, 117.48]
84	117.55	(117.48, 234.51]
85	127.38	(117.48, 234.51]
86	136.32	(117.48, 234.51]
87	145.26	(117.48, 234.51]
88	152.41	(117.48, 234.51]
89	159.56	(117.48, 234.51]
90	165.37	(117.48, 234.51]

Por defecto la función `cut` utiliza el intervalo como valor del nuevo atributo. Podemos asignar valores arbitrarios al nuevo atributo, por ejemplo:

In [28]:
```
# Designamos 5 nombres para los intervalos
group_names = ['Very Low', 'Low', 'Medium', 'High', 'Very High']
```

In [29]:
```
# Creamos un nuevo atributo "Iws_disc_named" discretizando de nuevo "Iws" con 5 intervalos
# del mismo tamaño pero usando ahora las etiquetas definidas
data_trans["Iws_disc_named"] = pd.cut(data_trans["Iws"], 5, labels = group_names)
```

In [30]:
```
# Visualizamos el contenido de los atributos "Iws", "Iws_disc" y "Iws_disc_named"
# para un subconjunto de muestras para observar el resultado
data_trans.loc[80:90, ["Iws", "Iws_disc", "Iws_disc_named"]]
```

Out[30]:

	Iws	Iws_disc	Iws_disc_named
80	80.90	(-0.135, 117.48]	Very Low
81	90.73	(-0.135, 117.48]	Very Low
82	100.56	(-0.135, 117.48]	Very Low
83	108.61	(-0.135, 117.48]	Very Low
84	117.55	(117.48, 234.51]	Low
85	127.38	(117.48, 234.51]	Low
86	136.32	(117.48, 234.51]	Low
87	145.26	(117.48, 234.51]	Low
88	152.41	(117.48, 234.51]	Low
89	159.56	(117.48, 234.51]	Low
90	165.37	(117.48, 234.51]	Low