Ingeniería de características y selección de modelos © EDICIONES ROBLE, S.L.

Indice

Ingeniería de características y selección de modelos

I. Introducción

En las unidades anteriores se han visto los principales tipos de modelos utilizados en aprendizaje automático. Para todos estos modelos, es necesario contar con un conjunto de datos con los que se pueda explicar el problema planteado. En los conjuntos de datos utilizados en los problemas reales, salvo escasas excepciones, no se conocen *a priori* las características que son más adecuadas para el modelo empleado. Además, durante el proceso de entrenamiento, en la mayoría de los modelos es necesario indicar parámetros, los cuales determinarán un mejor o peor resultado, por ejemplo, el número de clústeres en k-means o el tamaño del vecindario en k-nn.

En esta unidad, se verán algunas técnicas que permiten seleccionar las características más adecuadas para los conjuntos de datos y la identificación de los parámetros óptimos a la hora de crear un modelo.

La unidad comienza haciendo un repaso por los diferentes tipos de características que se pueden encontrar en un conjunto de datos utilizado para el entrenamiento de modelos. Los tipos son básicamente tres: cuantitativas, ordinales y categóricas. Conocer y poder identificar los diferentes tipos de características es importante para poder seleccionarlos posteriormente, en función del problema planteado. Una vez que se conozcan los tipos de características se estudiarán las principales transformaciones que se pueden realizar en las características existentes originalmente en los conjuntos de datos. Mediante estas transformaciones es posible adaptar la información disponible a las necesidades de los diferentes modelos.

Posteriormente, se estudiará cómo seleccionar las características candidatas a ser empleadas en los modelos. La adición de más características durante el proceso de entrenamiento de los modelos suele llevar a mejores resultados mientras se realiza el entrenamiento, pero una vez que estos modelos son puestos en producción, sus resultados son muy inferiores a los esperados. Esto es debido a que se trata de un modelo sobreajustado que ha memorizado los resultados. Para evitar esto es necesario seleccionar cuidadosamente las características que se van a utilizar, reduciendo estas a la cantidad mínima necesaria.

La unidad finaliza presentando un método a partir del cual se pueden seleccionar los parámetros más adecuados para el entrenamiento de un modelo.

Esta unidad se complementa con un notebook Python en el que se incluye todo el código utilizado y algunos ejemplos adicionales a los que se hace referencia en el texto. Es aconsejable seguir el texto con este notebook U5_Códigos

II. Objetivos

Los objetivos que los alumnos alcanzarán tras el estudio de esta unidad son:

- Conocer los diferentes tipos de características existentes en un conjunto de datos.
- Utilizar la discretización y normalización para la transformación de características.
- Poder realizar una selección previa de las características candidatas de un modelo.
- Comprender los efectos de la multicolinealidad y su evaluación con el VIF.
- Conocer los algoritmos Stepwise para la selección de características de forma iterativa.
- Aplicar la validación cruzada para la selección de los parámetros óptimos para un modelo.

III. Diferentes tipos de características y transformación

Las características son un elemento clave en los procesos de construcción de los modelos de aprendizaje automático. Comprender la tipología existente y la forma en la que se pueden transformar es fundamental para su posterior uso.

3.1. Tipos de características

En las unidades anteriores se han visto diferentes técnicas disponibles para la construcción de modelos que permitan contestar preguntas en función de unas características. A la hora de crear los modelos se ha visto que existen diferentes tipos de características con diferentes peculiaridades que han de ser tomadas en cuenta antes de utilizarlas.

Las características de los conjuntos de datos utilizados en los modelos de aprendizaje automático se pueden clasificar en tres categorías principales:

- Cuantitativas.
- Ordinales.
- Categóricas.

Se podría pensar que existe una categoría más, las características booleanas, pero estas no son más que un caso particular de las características categóricas en las que solamente existen dos categorías posibles: verdadero y falso.

3.1.1. Características cuantitativas

Las características cuantitativas son aquellas que toman valores continuos asociados a unas unidades que se vinculan a una escala. Estas características se encuentran generalmente representadas mediante valores reales (altura, ingresos, etc.), aunque en algunas ocasiones pueden estar representadas por valores enteros (edad en años, número de hijos, etc.). Habitualmente, los valores de las características continuas toman solamente un subconjunto de valores —por ejemplo, las edades de los clientes en muchos conjuntos de datos comienzan con edad mínima requerida para ser cliente y también pueden finalizar con valor máximo—, aunque se debe tener cuidado con utilizar la escala completa en el cálculo de estadísticos como la media, con la que se puede ver la tendencia o la desviación estándar, gracias a lo cual se puede comprender la dispersión, ya que estas dependen de las unidades de la característica.

Uno de los principales usos de estas características es emplearlas en los modelos geométricos, como se ha visto en los algoritmos de análisis de clúster, o en los problemas de regresión, debido a que las características cuantitativas tienen una escala numérica significativa. En estas aplicaciones se han de tener en cuenta las unidades. No es lo mismo medir la atura de una persona en metros, centímetros o pies y relacionarlo posteriormente con el peso en kilogramos, gramos u onzas.

En otras aplicaciones, como pueden ser los árboles de decisión, los valores se utilizan para seleccionar el camino en división binaria, en función de que el valor tomado supere o no uno de corte. Esto hace que las características cuantitativas realmente sean tratadas como características categóricas en este tipo de aplicaciones. Las categorías son los intervalos de los puntos de corte. En este caso las unidades no afectan a los resultados, simplemente se ha de cambiar el valor de corte al de las unidades en la que se encuentre la característica —por ejemplo, cambiando las distancias de metros a pies—.

3.1.2. Características ordinales

En segundo lugar, se encuentran las características ordinales, las cuales contienen valores numéricos que únicamente asignan un orden a los registros, pero, a diferencia de las continuas, no contienen unidades que permitan asociarlas con una escala. Estas características generalmente se suelen representar mediante número enteros. Por ejemplo, la posición final de los corredores en una carrera indica el orden en el que llegan a la meta: el primero ha llegado antes que el segundo y este que el tercero, pero contando únicamente con esta característica no es posible saber lo cerca que han llegado entre sí, como se vería con una característica continua como el tiempo empleado o la velocidad media.

Debido a que las características ordinales carecen de una escala, en ellas no tienen sentido transformaciones como la suma, la resta o la multiplicación. Por ejemplo, no tiene sentido sumar o restar la posición de llegada de los corredores. A causa de esto, estadísticos como la media o la desviación estándar no aportan ninguna información relevante acerca de la forma de estos datos. En este caso, la tendencia se observa utilizando la mediana —el valor que supera al de la mitad de los datos y se ve superado por la otra mitad— y la dispersión mediante los percentiles —los valores a partir de los cuales se encuentra un porcentaje de los datos—.

La utilización de este tipo de características en modelos basados en distancia obliga a asumir que la distancia entre ellos es simplemente su diferencia, una distancia a la que se la suele conocer como distancia de Hamming. En otros supuestos, su utilización sin aplicar una transformación previa no suele ser adecuada, ya que generalmente las distancias utilizadas son euclídeas.

Por otro lado, en las aplicaciones tipo árbol de decisión, la utilización de características ordinales funciona de una forma similar a la de las cuantitativas: seleccionando los caminos en función de que la posición supere o no un cierto valor umbral.

3.1.3. Características categóricas

El último tipo de características son las categóricas o nominales. En este grupo de características no existe orden ni escala y suelen estar representadas por los nombres de las categorías. Al no tener orden, los estadísticos como la media o la desviación estándar tienen sentido en ellas. Únicamente otros estadísticos como la moda —el valor que más frecuentemente aparece en los registros— pueden aportar información acerca de la tendencia.

A pesar de no tener valores numéricos, estas características pueden ser utilizadas en los modelos probabilísticos. Una de las formas es mediante la creación de distancias que son cero, cuando dos registros muestran el mismo valor, y uno, en caso contrario. Otra de las formas en la que pueden ser utilizadas es mediante la creación de variables dummies a partir de las categorías originales.

Como se ha comentado anteriormente, las características booleanas son un caso particular de característica categórica en la que únicamente existen dos niveles.

3.1.4. Resumen de clases de características

En la tabla 4.1. se hace un resumen de las principales propiedades estudiadas previamente para los tipos de características existentes. En ella se puede consultar si estas indican orden y escala junto a los estadísticos adecuados para medir la tendencia y la dispersión de cada una de las familias.

Categoría	Ordinal	Escala	Tendencia	Dispersión
Cuantitativas	Sí	Sí	Media	Desviación estándar
Ordinales	Sí	No	Mediana	Percentiles
Categóricas	No	No	Moda	N/A

Tabla 4.1. Resumen de los diferentes tipos de características.

3.2. Transformación de características

En muchas ocasiones, las características disponibles para la construcción de un modelo no son las más adecuadas. En estas situaciones se pueden plantear dos soluciones: buscar nuevas —lo que no suele ser factible tanto por coste, tiempo o disponibilidad— o transformar las características originales, con la idea de conseguir que estas sean más útiles. La transformación de una característica se puede llevar a cabo mediante la combinación de operaciones matemáticas como la suma, la resta o la multiplicación o mediante la aplicación de operaciones no lineales como puede ser la discretización.

3.2.1. Discretización

Una de las transformaciones más comúnmente utilizadas durante el preprocesado de datos en un modelo de aprendizaje automático es la discretización. En la discretización se parte de una característica, generalmente cuantitativa, y se buscan valores umbrales que definan los niveles de una característica ordinal. A partir de esta característica ordinal se puede crear un conjunto de características dummies, como se ha visto en la unidad 2. Las características así definidas son adecuadas para modelos en los que se utilicen valores discretos, como pueden ser los modelos basados en reglas.

La discretización de las características continuas permite mejorar el rendimiento de los modelos de regresión, donde la relación entre una característica independiente y la característica dependiente es no lineal. Por ejemplo, se puede pensar en la relación que existe entre la siniestralidad y la edad de los conductores en una compañía de seguros. Es sabido que, inicialmente, la siniestralidad se reduce con la edad para estancarse y volver a subir, lo que es representado por una gráfica de tipo U. En este caso, si se utiliza la edad como característica independiente en una regresión lineal, posiblemente no se obtengan los mejores resultados, ya que en estos modelos solamente se recogen relaciones lineales y una de las dos tendencias; la reducción inicial o el aumento final no podrá ser tenida en cuenta.

En la figura 4.1. se muestra un ejemplo de discretización. En esta se puede apreciar la curva que relaciona los valores de una característica independiente con otra dependiente, con forma de U. La utilización de esta característica en un modelo lineal requiere crear una nueva característica con cuatro niveles que se han representado en la gráfica como áreas de diferente color.

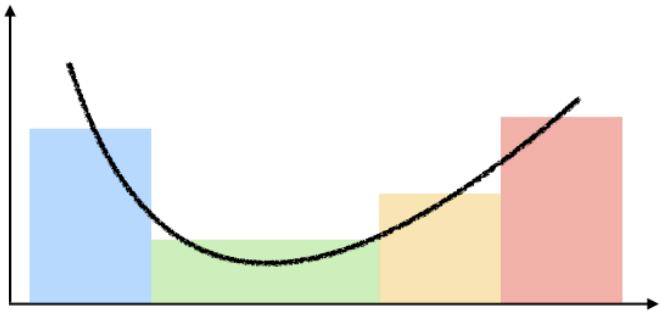

Figura 4.1. Ejemplo de discretización de una característica continua.

Un caso particular de la discretización es la binarización, en donde la característica se transforma en otra que solamente tiene dos niveles, en la cual se asigna un estado diferente a los valores por debajo de un umbral y otro al resto.

En Pandas la discretización de una característica se realiza utilizando la función cut. A esta función se le ha de pasar un vector y el número de tramos en los que se desea dividir el conjunto. Adicionalmente, el comportamiento se configura utilizando algunas de las siguientes opciones:

- right: valor lógico en el que se indica si el valor más elevado ha de incluir el último intervalo. Por defecto, es verdadero.
- labels: array con el nombre de los niveles.
- retbins: valor lógico que indica si la función ha de devolver la definición de los niveles.
- include_lowest: valor lógico en el que se indica si el valor más bajo ha de incluir el primer intervalo. Por defecto, es falso.

En el código de la figura 4.2. se muestra un ejemplo de la utilización de esta función, en la cual se observa la división de un vector en tres niveles a los que se les asigna una etiqueta diferente.

```
import pandas as pd

print pd.cut(np.array([.2, 1.4, 2.5, 6.2, 9.7, 2.1]), 3,
             labels = ['good', 'medium', 'bad'],
             retbins = True)
```

Figura 4.2. Ejemplo de la función cut de Pandas.

Al ejecutar el código de la figura 4.2. se obtiene un vector con los valores [good, good, good, medium, bad, good] y, al seleccionar la opción retbins, los niveles en los que se divide la muestra: [0.1905, 3.36666667, 6.53333333, 9.7]. Esto indica que el nivel good se asocia a los valores por debajo de 3,36, el nivel bad a los valores por encima de 9,7 y el valor medium a los intermedios.

Existen diferentes métodos para seleccionar la división más adecuada de una característica continua. Por ejemplo, se pueden utilizar estadísticas de tendencia central, como la media o la mediana o la optimización de una función objetivo, como es el peso de la evidencia.

El peso de la evidencia (WoE, *Weight of Evidence*) es un valor que indica la capacidad predictiva de cada uno de los niveles de una característica categórica respecto a una característica binaria. La definición matemática del WoE es:

$$WoE = ln \left| \frac{R_i(T)}{R_i(F)} \right|$$

$R_i(T)$ es el porcentaje de valores positivos en la categoría i de la característica y$R_i(F)$ es el porcentaje de valores negativos. En caso de que no se use la notación positivo-negativo, para indicar los niveles se ha de asignar uno a cada estado. El cambio de positivo por negativo en una característica solamente se traduce en un cambio de signo del WoE, pero no de su valor absoluto. El WoE puede tomar valores entre menos infinito —cuando el porcentaje de valores positivos es 0, es decir, el tramo solamente tiene valores negativos— y más infinito —cuando el porcentaje de valores negativos es cero: el tramo solamente tiene valores positivos—.

No existe una función en las librerías Python que se utilizan en el curso para obtener el WoE de una característica, por lo que es necesario crearla desde cero. En la figura 4.3. se muestra una implementación de ejemplo. A esta función se le ha de pasar un dataframe (data), el nombre de la característica a estimar (var) y la variable objetivo. Esta función se prueba con el código de la figura 4.4., donde se ha definido un conjunto de datos para la tarea.

```
def get_WoE(data, var, target):
    crosstab = pd.crosstab(data[target], data[var])

    print "Obteniendo el Woe para la variable", var, ":"

    for col in crosstab.columns:
        if crosstab[col][1] == 0:
            print "  El WoE para", col, "[", sum(crosstab[col]), "] es infinito"
        else:
            WoE = np.log(float(crosstab[col][0]) / float(crosstab[col][1]))
            print "  El WoE para", col, "[", sum(crosstab[col]), "] es", WoE
```

Figura 4.3. Implementación del cálculo del WoE.

```
data = pd.DataFrame({'Value': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                     'Target': [True, True, False, True, True, False, True, False, False, False]})

data['Cat 1'] = data['Value'] > 3
data['Cat 2'] = data['Value'] > 6

get_WoE(data, 'Cat 1', 'Target')
get_WoE(data, 'Cat 2', 'Target')
```

Figura 4.4. Ejemplo de utilización de la función get_WoE.

En el código de la figura 4.4. se han creado dos características categóricas a partir de una característica continua y se han calculado los valores del WoE en ambos casos. Se puede observar que en los dos el valor del WoE por debajo del umbral es -0,69, mientras que el valor por encima es de 0,28, en la primera, y 1,09, en la segunda, por lo que esta segunda configuración es mejor que la primera.

El WoE permite identificar la capacidad de discretización de los niveles de una característica categórica, pero no dice nada del conjunto. Por ejemplo, en el caso externo de un identificador, se obtiene un valor infinito para todos los niveles, ya que en cada registro el valor es diferente, pero estas características no tienen capacidad para identificar tendencias. En una sección posterior de esta unidad se estudiará el IV, que es un indicador derivado del WoE que permite identificar la capacidad de discretización de la característica en su conjunto.

3.2.3. Normalización

Uno de los motivos por los que se emplea la discretización en conjuntos de características cuantitativos es para eliminar las dimensiones en las que se han medido, es decir, para evitar los sesgos que puede introducir utilizar unidades diferentes, como metros o pies en la medida. Pero, en muchas ocasiones puede que no sea aceptable la pérdida de información que conlleva la discretización de una característica. En estas ocasiones se puede utilizar la normalización para eliminar las dimensiones de las características y, al mismo tiempo, conservar toda la información disponible en las mismas.

A la hora de normalizar una característica se pueden utilizar diferentes aproximaciones que en Python se encuentran en sklearn.preprocessing. Entre las que existen, tres de las más utilizadas son las que se describen a continuación.

En la primera aproximación se asume que la distribución de los valores que toma una característica es homogénea. La normalización se lleva a cabo restando el valor mínimo a cada uno de los valores y se divide por la diferencia entre el valor máximo y el mínimo. Se utiliza la siguiente expresión para normalizar la característica:

$$x_{mm} = \frac{x - l}{h + l}$$

x es la característica original, l el valor mínimo, h el máximo y x_{mm} es el resultado de la característica normalizada. De esta forma se consigue una nueva característica en la que los datos se encuentran distribuidos de la misma forma que los originales, pero con valores entre 0 y 1 e independientes de las unidades.

El objeto que implementa esta transformación es MinMaxScaler. La principal opción del constructor es feature_range, con la que se puede fijar el rango al que se desea transformar los datos. En la figura 4.5. se muestra un ejemplo de utilización.

```
from sklearn.preprocessing import MinMaxScaler

data = [35.6, -26.4, 54.9, -63.4, 37.9, 45.8, 44.3, 9.2, 35.5, -12.9]
data = np.array(data).reshape(-1, 1)

scaler = MinMaxScaler().fit(data)
scaler.transform(data)
```

Figura 4.5. Ejemplo de utilización de MinMaxScaler.

En este código se puede ver que se crea un conjunto de datos con 10 valores que se utilizará en los ejemplos de normalización. El objeto scaler se ha de ajustar con el conjunto de datos, posteriormente, se utiliza el método transform para llevar a cabo la normalización. El hecho de ajustar el objeto permite que este se pueda usar sobre otros conjuntos de datos y que se consiga la misma transformación. En los modelos en producción en los que se utilice la normalización no se ha de aplicar la normalización con base en los datos usados en producción, sino en los datos utilizados en el entrenamiento, ya que la escala de los segundos puede ser diferente.

Una segunda aproximación es asumir que los datos se distribuyen de forma normal, lo que es aceptable en una gran cantidad de escenarios. En este caso, la normalización se puede llevar a cabo mediante la resta de la media a cada registro y dividir el resultado por la desviación estándar, es decir:

$$x_{sd} = \frac{x - \mu}{\sigma}$$

μ es la media, σ es la desviación estándar y x_{sd} es el resultado de la característica normalizada. De esta forma se consigue una nueva característica que, a diferencia del caso anterior, no está acotada y la media es cero.

El objeto que implementa esta transformación es StandardScaler. El uso es similar al anterior, se ha de ajustar con un conjunto de datos y posteriormente se puede utilizar para realizar las transformaciones. Entre las principales opciones del constructor destacan:

- with_mean: es un valor lógico con el que se indica si se resta la media; el valor, por defecto, es cierto.
- with_std: es un valor lógico con el que se indica si se divide por la desviación estándar; el valor, por defecto, es cierto.

En la figura 4.6. se muestra un ejemplo de utilización de este objeto.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(data)
scaler.transform(data)
```

Figura 4.6. Ejemplo de utilización de StandardScaler.

Finalmente, una tercera aproximación es utilizar el rango intercuartil. El rango intercuartil es la distancia entre el primer y tercer cuartil de un conjunto de datos, siendo el primer cuartil el valor por debajo del cual se encuentra el 25% de los registros, el segundo cuartil o mediana el que divide los datos en dos conjuntos con la misma cantidad de registros y el tercer cuartil el valor por debajo del cual se encuentra el 75% de los registros.

$$x_r = \frac{x - m}{iqr}$$

iqr es la distancia intercuartil, m es la mediana y x_r es el resultado de la característica normalizada. Al igual que con la normalización basada en la desviación estándar, el valor de la nueva característica no se encuentra acotada entre cero y la unidad.

La implementación de esta transformación se puede encontrar en el constructor RobustScaler, que se utiliza de manera análoga a los anteriores. Entre las principales opciones se pueden destacar:

- with_centering: es un valor lógico con el que se indica si se resta la mediana. El valor, por defecto, es cierto.
- with_scaling: es un valor lógico con el que se indica si se divide por el rango intercuartil. El valor, por defecto, es cierto.
- quantile_range: es una tupla en la que se indican los porcentajes que definen el rango intercuartil. Por defecto, los valores de la tupla son (25.0, 75.0).

En la figura 4.7. se muestra un ejemplo de utilización de este objeto.

```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler().fit(data)
scaler.transform(data)
```

Figura 4.7. Ejemplo de utilización de RobustScaler.

IV. Selección de características

El proceso de selección de las características que han de ser incluidas en un modelo es fundamental. Antes de la creación de un modelo no se puede saber cuáles son las más significativas, las menos significativas ni tampoco las que simplemente son ruido. Una selección poco exigente puede terminar en un modelo con demasiadas características que puede acabar en sobreajuste. Por otro lado, una eliminación excesiva puede llevar a que no se consigan obtener modelos útiles.

Una vez cargado el conjunto de datos, se pueden ir eliminando características en función de un conjunto de métodos que se pueden enumerar en base a su complejidad, de menor a mayor:

- Eliminar características con poca capacidad predictiva.
- Características con baja varianza: constantes.
- Identificadores: nombres o ID.
- Análisis univariante.
- Eliminación recursiva de características.

El primer paso consiste en conocer qué significa cada uno de los datos disponibles. Las constantes —como puede ser el país en modelos que se van a ejecutar solamente en una ubicación— o los identificadores —como el número de clientes— son características que se pueden eliminar fácilmente.

Otro factor a tener en cuenta es que hay que evitar la inclusión de características que no estén disponibles cuando el modelo se encuentre en producción, ya que su obtención es posterior al momento en el que se desea realizar la predicción. La inclusión de estas características, aunque sean estadísticamente significativas, lleva a la construcción de modelos que no pueden ser puestos en marcha y, por lo tanto, utilizados. Desafortunadamente, la única forma de identificar estas características es conociendo el proceso de captura de datos.

En esta sección se va a volver a utilizar el conjunto de datos del archivo"winequality-white.csv" separando, por un lado, la calidad en la característica y el resto de características en la característica x. Se muestra el proceso de carga en la figura 4.8.

```
wine = pd.read_csv('winequality-white.csv', sep = ';')

target = 'quality'
features = list(wine.columns)
features.remove('quality')

x = wine[features]
y = wine[target]
```

Figura 4.8. Carga de los datos del archivo "winequality-white.csv".

4.1. Selección univariante de características

Uno de los métodos disponibles en scikit-learn para la selección de características es VarianceThreshold. Este constructor, que se puede encontrar en sklearn.feature_selection, permite seleccionar las características de un conjunto que supere un valor dado de varianza. Las características que no muestren variación se pueden eliminar del conjunto de datos utilizados porque no pueden explicar la variabilidad de la característica dependiente. Se puede ver un ejemplo en el que se utiliza este método en la figura 4.9.

```
from sklearn.feature_selection import VarianceThreshold

var_th = VarianceThreshold(threshold = 0.60)
x_var = var_th.fit_transform(x)

print "Variables originales ", x.shape[1]
print "Variables finales ", x_var.shape[1]

print "Listado de variables ", np.asarray(list(x))[var_th.get_support()]
```

Figura 4.9. Selección de características con VarianceThreshold.

En la figura 4.9. se importa el constructor, se crea un nuevo objeto con la opción del umbral a 0,6 y se ejecuta la transformación. En este caso, el método fit_transform devuelve un nuevo conjunto de datos con menos características, concretamente, 5 de las 11 originales.

Otro método para la selección de características es SelectKBest. Este método permite seleccionar las k características que mejor resultado obtengan con una función. Las funciones que se utilizan habitualmente son f_regression, para modelos de regresión lineal, y chi2, para modelos de clasificación. La figura 4.10. muestra un ejemplo de la utilización de este procedimiento.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import chi2

var_sk = SelectKBest(f_regression, k = 5)
x_sk = var_sk.fit_transform(x, y)

print "Variables finales ", x_sk.shape[1]

print "Listado de variables ", np.asarray(list(x))[var_sk.get_support()]
```

Figura 4.10. Selección de características con SelectKBest.

El código de la figura 4.10. es similar al de la figura 4.9., pero se pueden observar dos diferencias importantes. La primera es que SelectKBest necesita, además del número de características a seleccionar, una función de score para ordenar las características. El método fit_transform requiere también de la característica dependiente. En este caso, el método devuelve únicamente la cantidad de características que se ha indicado en el constructor.

Una alternativa a SelectKBest es SelectPercentile, donde se selecciona un porcentaje de características en lugar de una cantidad fija. En la figura 4.11. se muestra un ejemplo de su uso. En este código se seleccionan cinco características, dado que se ha indicado que se seleccione el 50% de la muestra y esta tiene un tamaño de 11 registros.

```
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import f_regression

var_sp = SelectPercentile(f_regression, percentile = 50)
x_sp = var_sp.fit_transform(x, y)

print "Variables finales ", x_sp.shape[1]

print "Listado de variables ", np.asarray(list(x))[var_sp.get_support()]
```

Figura 4.11. Selección de características con SelectPercentile.

4.2. Selección de características con IV

El WoE puede ser utilizado para seleccionar las categorías de una característica, de forma que esta maximice la capacidad de discernir en un problema de clasificación. Para esto se han de seleccionar y diseñar categorías, de manera que el valor absoluto del WoE sea máximo. Esta aproximación presenta un problema: en caso de que no se defina el número de grupos, se pueden crear tantos como valores, de modo que todos tengan un valor para el WoE infinito. A primera vista, esto parecería una buena idea, pero no lo es, ya que de esta forma se obtendría sobreajuste. Además, el WoE tampoco permite comparar características entre sí, ya que informa a nivel de categoría.

Para solucionar estos problemas se utiliza el Valor de la Información (IV, *Information Value*):

$$IV = \sum_{i=1}^{N}(R_i(T) - R_i(F) ln \left|\frac{R_i(T)}{R_i(F)}\right| = \sum_{i=1}^{N}(R_i(T) - R_i(F))WoE_i$$

El IV mide la capacidad de clasificación de las diferentes variables independientes en problemas de clasificación. Como criterio orientativo se pueden utilizar los valores de la tabla 4.2. para seleccionar las variables candidatas en los modelos de clasificación. Estos valores se han de tomar como referencia, ya que dependiendo del problema y la cantidad de características disponibles será necesario un mayor o menor nivel de exigencia para este indicador.

Valor de IV	Capacidad de clasificación
<0,02	Muy débil
0,02 a 0,1	Débil
0,1 a 0,3	Promedio
0,3 a 0,5	Fuerte
>0,5	Muy fuerte

Tabla 4.2. Capacidad de clasificación en función del IV.

Para utilizar el IV es necesario disponer de características categóricas. Se pondrá en práctica cargando el archivo "crx_data.txt" con la valoración crediticia de los clientes de una entidad. En este conjunto de datos existen 10 características categóricas y 5 numéricas. Los detalles de este conjunto de datos se pueden encontrar en el repositorio de datos para Aprendizaje Automático existente en la página web de la Universidad de California (https://archive.ics.uci.edu/ml/datasets/Credit+Approval). El código para la carga y la separación de las mismas se muestra en la figura 4.12.

```
# Importación de los datos
credit_approval = pd.read_csv('crx_data.txt', sep = ',')
credit_approval.head()
mask = credit_approval.applymap(lambda x: x in ['?'])
credit_not_null = credit_approval[-mask.any(axis=1)]

# Separación de las variables
var_categoricas = ['A1', 'A4', 'A5', 'A6', 'A7', 'A9', 'A10', 'A11', 'A12', 'A13']
var_numericas   = ['A2', 'A3', 'A8', 'A14', 'A15']
var_target      = credit_not_null['A16'] == '+'
```

Figura 4.12. Importación del archivo crx_data.txt.

En este código se cargan los datos utilizando Pandas en la variable credit_aproval. Primero se imprimirán los registros del conjunto de datos y se aplicará un filtro para eliminar los registros con campos vacíos, que en el archivo utilizado se identificarán con el símbolo de interrogación.

Posteriormente, las características disponibles en el conjunto de datos se dividen en tres: las categóricas, las numéricas y la variable independiente, a la que se le asigna el valor cierto cuando la columna A16 tenga el símbolo de suma.

```
result_IV = []

for v_cat in var_categoricas:
    var_target = array(var_target)
    var_values = array(credit_not_null[v_cat])
    var_levels = unique(var_values)

    mat_values = numpy.zeros(shape=(len(var_levels),2))

    for i in range(len(var_target)):
        # Obtención de la posición en los niveles del valor
        for j in range(len(var_levels)):
            if var_levels[j] == var_values[i]:
                pos = j
                break

        # Estimación del número valores en cada nivel
        if var_target[i]:
            mat_values[pos][0] += 1
        else:
            mat_values[pos][1] += 1

        # Obtención del IV
        IV = 0
        for j in range(len(var_levels)):
            if mat_values[j][0] > 0 and mat_values[j][1] > 0:
                rt = mat_values[j][0] / (mat_values[j][0] + mat_values[j][1])
                rf = mat_values[j][1] / (mat_values[j][0] + mat_values[j][1])
                IV += (rt - rf) * np.log(rt / rf)

    # Se agrega el IV al listado
    result_IV.append(IV)

for i in range(len(var_categoricas)):
    print "La característica", var_categoricas[i], "el IV es", result_IV[i]
```

Figura 4.13. Selección de características mediante IV.

Una vez importados los archivos se puede ejecutar el código de la figura 4.13. Este listado contiene el proceso para la obtención del WoE de cada una de las categorías de la característica y posteriormente el IV. El proceso de cálculo comienza iterando sobre las características categóricas, en cada paso se selecciona una única característica para la que se obtiene el listado de categorías. Tras conseguir este listado, se calcula el número de casos positivos y negativos para cada una de las categorías, obteniéndose el ratio de positivos y el ratio de negativos, con los que se calcula en sumatorio del IV. Finalmente, se presenta el valor para cada una de las características.

Al ejecutar el código anterior se obtienen los siguientes resultados:

- Para la característica A1, el IV es 0,0310557839282
- Para la característica A4, el IV es 0,353312725336
- Para la característica A5, el IV es 0,353312725336
- Para la característica A6, el IV es 5,30149147134
- Para la característica A7, el IV es 2,34839349885
- Para la característica A9, el IV es 3,24769074169
- Para la característica A10, el IV es 0,895478013341
- Para la característica A11, el IV es 11,0569990916
- Para la característica A12, el IV es 0,0423925791404
- Para la característica A13, el IV es 0,411470398153

Atendiendo a la tabla 4.2., se puede asumir que las características A6, A7, A9, A10 y A11 muestran una capacidad de clasificación muy fuerte y las A4, A5 y A13 fuerte. Si se construye un modelo utilizando un conjunto de estas características, tendrá altas probabilidades de ofrecer buenos resultados.

4.3. Eliminación de Características multicolinealidad (VIF)

En la unidad 2 se ha visto que la existencia de multicolinealidad en el conjunto de características independientes afecta de forma negativa a los modelos construidos con estas. En las ocasiones en las que existe multicolinealidad, un conjunto de las características independientes puede ser calculado como combinación lineal del resto, por lo que en el proceso de entrenamiento se estará utilizando información duplicada. Debido a esta duplicidad en la información, los procesos de entrenamiento no pueden encontrar los parámetros adecuados para la correcta construcción de los modelos.

Una solución a este problema es utilizar el Factor de Inflación de la Varianza (VIF, *Variance Inflation Factor*), que permite cuantificar la intensidad de la multicolinealidad. La definición de este factor es:

$$VIF_i = \frac{1}{1 - R_i^2}$$

VIF_i es el valor del factor para la característica i y R_i^2 es coeficiente de determinación para la característica i. El valor del VIF es siempre positivo y crece en un conjunto de datos a medida que aumenta la multicolinealidad entre las características. En caso de que en un conjunto de características exista multicolinealidad perfecta, es decir, que una característica se pueda explicar perfectamente mediante la combinación lineal del resto, el valor será infinito, ya que, en tal caso, el R^2 de la regresión será igual la unidad.

Se consideran elevados a los valores del VIF por encima de 5 y muy elevados a los que estén por encima de 10. Al igual que para el IV, el criterio para eliminar o incluir una característica se deberá ajustar en función de la cantidad disponible de las mismas.

El cálculo del VIF se puede implementar fácilmente en Python, solamente es necesario importar el constructor LinealRegression, que ya se ha utilizado y explicado en la unidad 2. En la figura 4.14. se muestra una implementación de un método que permite el cálculo del VIF para un dataframe. En este código se puede ver que inicialmente se extrae el listado de características existentes en el conjunto de datos y se crea un dataframe vacío (result) para almacenar los resultados. En este punto se procede a iterar sobre el listado de características realizando una regresión lineal de una característica frente al resto. El valor de R^2 de la regresión se utiliza para calcular el VIF y guardar el resultado en el dataframe, que devolverá el método al terminar.

```
from sklearn.linear_model import LinearRegression

def calculateVIF(data):
    features = list(data.columns)
    num_features = len(features)

    model = LinearRegression()

    result = pd.DataFrame(index = ['VIF'], columns = features)
    result = result.fillna(0)

    for ite in range(num_features):
        x_features = features[:]
        y_featue = features[ite]
        x_features.remove(y_featue)

        x = data[x_features]
        y = data[y_featue]

        model.fit(data[x_features], data[y_featue])

        result[y_featue] = 1/(1 - model.score(data[x_features], data[y_featue]))

    return result
```

Figura 4.14. Función que implementa el cálculo del VIF.

Al ejecutar el código de la figura 4.14. sobre el conjunto de datos de vinos blancos importado previamente, se obtienen los siguientes resultados:

- fixed acidity: 2,691435
- volatile acidity: 1,141156
- citric acid: 1,165215
- residual sugar: 12,644064
- chlorides: 1,236822
- free sulfur dioxide: 1,78788
- total sulfur dioxide: 2,239233
- density: 28,232546
- pH: 2,196362
- sulphates: 1,13854
- alcohol: 7,706957

Al revisar el listado, se observa que hay dos características con VIF muy elevado (residual sugar y density) y una con VIF elevado (alcohol).

En el listado también se puede ver que no se ha incluido la característica dependiente, lo cual sería un error, ya que, en la construcción del modelo, lo que se busca es justamente una relación entre la característica dependiente y las independientes.

Una vez obtenido el VIF, hay que definir un proceso para la eliminación de características. El proceso se puede resumir en los siguientes pasos:

1. Seleccionar un valor umbral del VIF para la eliminación.
2. Obtener el valor del VIF de todas las características.
3. Identificar la característica con VIF más elevado.

4. Si este valor supera el umbral definido, la característica se elimina del conjunto y se vuelve al punto 2, en caso contrario, el proceso termina.

En el punto 4 del proceso solamente se elimina una característica cada vez, esto es así porque puede que todas las características con VIF elevado sean colineales y al eliminar una del conjunto esta relación desaparecería. Eliminar más de una característica en cada paso puede llevar a despreciar características que no son redundantes.

Este proceso se ha implementado en el código la figura 4.15., que utiliza el método calculateVIF, definido en el código de la figura 4.14. El método selectDataUsingVIF tiene dos parámetros de entrada: el conjunto de datos (data) y una opción con el criterio de eliminación (max_VIF). El proceso copia los datos de entrada en un nuevo conjunto de datos y calcula el VIF. A partir de aquí, se entra en un bucle que termina cuando el valor máximo del VIF es inferior al criterio. En el bucle se elimina la columna de dataframe con el valor de VIF más elevado en cada momento y se vuelve a calcular el VIF.

```
def selectDataUsingVIF(data, max_VIF = 5):
    result = data.copy(deep = True)

    VIF = calculateVIF(result)

    while VIF.as_matrix().max() > max_VIF:
        col_max = np.where(VIF == VIF.as_matrix().max())[1][0]
        features = list(result.columns)
        features.remove(features[col_max])
        result = result[features]

        VIF = calculateVIF(result)

    return result
```

Figura 4.15. Función que implementa la selección de características con VIF.

Si se ejecuta el código la figura 4.15. se obtiene un nuevo conjunto de datos con los siguientes valores de VIF:

- fixed acidity: 1,356128
- volatile acidity: 1,128298
- citric acid: 1,159884
- residual sugar: 1,435215
- chlorides: 1,203645
- free sulfur dioxide: 1,744627
- total sulfur dioxide: 2,15317
- pH: 1,330912
- sulphates: 1,056637
- alcohol: 1,647117

Nótese que solamente ha sido necesario eliminar una característica (density) para que el valor máximo del VIF pase a ser 2,15. Esto es así porque todas las características que tenían VIF elevado estaban relacionadas con la densidad y, al desaparecer estas, desaparece el valor del VIF del resto y se ve reducido.

4.4. Stepwise

Stepwise es una familia de métodos para la selección de características de forma iterativa. En estas técnicas se parte de un modelo con una cantidad de características y se prueba a agregar una a una el resto de características o a eliminar una a una las utilizadas. Una vez hecho esto, se comprueba la calidad de los modelos y se selecciona el mejor, en caso de que sea el original, el proceso termina, en caso de que sea otro, el proceso vuelve a comenzar.

Dentro de los métodos Stepwise existen tres enfoques, que son:

- Adición de características (*forward selection*), lo que implica comenzar sin características en el modelo, poner a prueba la adición de cada característica utilizando un criterio para comparar el nuevo modelo con el anterior, añadir una característica que permita mejorar el modelo y repetir este proceso hasta que añadir una característica no mejore el modelo.
- Eliminación de características (*backward elimination*), lo que implica comenzar con todas las características candidatas, comprobar el efecto de eliminar una característica utilizando un criterio para comparar el nuevo modelo con el anterior, eliminar en cada paso una característica con la que se mejore el modelo y repetir este proceso hasta que eliminar una característica no mejore el modelo.
- Selección bidireccional (*bidirectional elimination*), una combinación de lo anterior: las pruebas en cada paso implican la eliminación y adición de características.

Actualmente, a pesar de ser un método bastante popular en algunos entornos, no está imprentando en scikit-learn. Por eso, para ponerlo en práctica se ha implementado en la figura 4.16. una versión pedagógica del enfoque *forward selection*.

```
from sklearn.linear_model import LinearRegression

# Modelo para realizar los ajustes
model = LinearRegression()

# Variable para almecena los índices de la lista de atributos usados
feature_order =  []
feature_error = []

# Iteración sobre todas las variables
for i in range(len(features)):
    idx_try = [val for val in range(len(features)) if val not in feature_order]
    iter_error = []

    for i_try in idx_try:
        useRow = feature_order[:]
        useRow.append(i_try)

        use = x[x.columns[useRow]]

        model.fit(use, y)
        rmsError = numpy.linalg.norm((y - model.predict(use)), 2)/sqrt(len(y))
        iter_error.append(rmsError)

    pos_best = numpy.argmin(iter_error)
    feature_order.append(idx_try[pos_best])
    feature_error.append(iter_error[pos_best])

for i in range(len(features)):
    print ("En el paso", i, "se ha insertado la variable",
        features[feature_order[i]], "con un error", feature_error[i])
```

Figura 4.16. Implementación de la regresión Stepwise con selección.

En la figura 4.16. se importa la clase para la creación de los modelos lineales y se crea un objeto con ella. Además, se inicializan dos características con el listado de características que el algoritmo incluirá en el modelo y su error.

El proceso de selección de características se inicia iterando sobre todas las características independientes. En esta iteración, lo primero que se hace es seleccionar únicamente las características que no están seleccionadas (idx_try) y crear un vector para guardar el error cometido al añadir una característica al modelo (iter_error). Obviamente, en la primera iteración serán todas. En este punto se inicia una iteración sobre las características seleccionadas.

Posteriormente, se inicia un segundo bucle sobre las características seleccionadas en el paso anterior (idx_try). Aquí, a las características ya seleccionadas (feature_order) se añade una de las disponibles en idx_try (i_try). Con este nuevo conjunto de datos se crea un modelo y se obtiene su error (rms_error), en este caso, el error cuadrático medio que se agrega a array de errores.

Cuando se termina esta última iteración, se selecciona el mejor de los modelos (pos_best), aquel con menor error, y se añade al listado de características seleccionadas. En este punto es en el que un algoritmo completo debería decidir si continuar con los modelos o seleccionar el anterior.

Al ejecutar este código, se puede observar que el orden de adición de las características es:

1. alcohol
2. volatile acidity
3. residual sugar
4. free sulfur dioxide
5. density
6. pH
7. sulphates
8. fixed acidity
9. total sulfur dioxide
10. chlorides
11. citric acid

En la figura 4.17. se muestra la evolución del error en función del número de características añadidas. En este caso se observa que un modelo con más de 6 características no es mejor que uno con menos características.

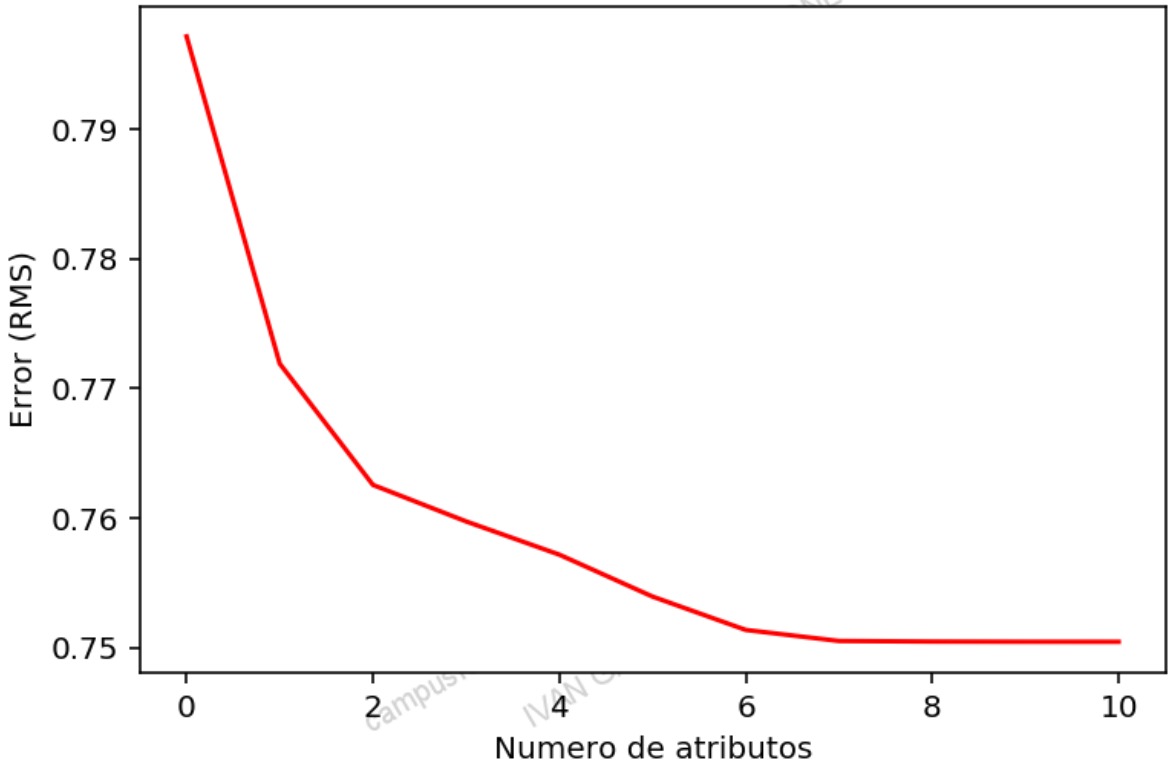

Figura 4.17. Error en función del número de características en Stepwise.

4.5. LASSO Regression

En la segunda unidad se ha visto la definición de función de esfuerzo. Esta es la función que indica el error cometido por el modelo en función de los parámetros y es la que se minimiza utilizando el algoritmo del gradiente descendente para la obtención de los parámetros del modelo. En aquel momento, la función de esfuerzo para un modelo lineal se había definido como:

$$J(w_o, w_1) = \frac{1}{2m}\sum_{i=1}^{m}(y(x_i; w_o, w_1) - y_i)^2$$

J representa la función de esfuerzo, m es el número de datos en el conjunto de entrenamiento, x_i es la característica dependiente para el registro i e y_i es la característica independiente para el registro i.

Las funciones de esfuerzo se pueden modificar mediante la regularización. La regularización consiste en añadir una penalización a los coeficientes de regresión que sean demasiado grandes para reducir el sobreajuste. Los dos métodos más utilizados son el Ridge, en el que se penaliza añadiendo el cuadrado del parámetro del modelo multiplicado por una constante:

$$J(w_o, w_1) = \frac{1}{2m}\sum_{i=1}^{m}(y(x_i; w_o, w_1) - y_i)^2 + \lambda\sum_{j=0}^{1} w_j^2$$

y el LASSO (*least absolute shrinkage and selection operator*), en el que la penalización consiste en añadir el parámetro del modelo multiplicado por una constante:

$$J(w_o, w_1) = \frac{1}{2m}\sum_{i=1}^{m}(y(x_i; w_o, w_1) - y_i)^2 + \lambda\sum_{j=0}^{1} w_j$$

En ambos casos, λ es una constante positiva. λ se conoce como parámetro de regularización y su valor se acota entre 0 y 1. Se puede observar que cuando $\lambda = 0$, la función de esfuerzo es justamente la que se había definido para la regresión lineal en la segunda unidad.

La regresión LASSO, al penalizar los parámetros que requieren mayor peso, permite seleccionar las características eliminado aquellas que necesitan valores de los parámetros más grandes para poder aportar información al modelo.

La implantación de la regresión LASSO en scikit-learn es similar a la regresión lineal y comparte una gran cantidad de opciones. El constructor para este tipo de modelos es Lasso, que se encuentra dentro de sklearn.linear_model. Entre las opciones más interesantes, se pueden enumerar:

- alpha: es el constante que multiplica al parámetro del modelo para regularizarlo. El valor por defecto es 1 y, en caso de indicar 0, el modelo es uno lineal simple.
- fit_intercept: es un valor lógico que indica si el modelo tiene término independiente, por defecto, su valor es cierto.
- normalize: es un valor lógico que indica si las características se normalizan antes de la regresión. En caso de que se seleccione esta opción, los resultados suelen ser más robustos. Por defecto, este valor es falso.

En la figura 4.18. se muestra la creación de un modelo.

```
from sklearn.linear_model import Lasso

model_ridge = Lasso(alpha = 0.1)
model_ridge.fit(x, y)

print model_ridge.coef_
```

Figura 4.18. Implementación de una regresión LASSO sobre el conjunto de datos del vino.

Al ejecutar el código de la figura 4.18. se observa que solamente cuatro de las características incluidas en el modelo son distintas de cero, el resto de las características han sido eliminadas.

El parámetro de regularización determina la cantidad de características que serán utilizadas en el modelo final. A medida que el valor se acerca a la unidad, la penalización en la regresión se hace más estricta. Por otro lado, a medida que se reduce la penalización, empieza a ser menos importante hasta llegar a una regresión lineal cuando este se hace cero.

La correcta determinación del valor del parámetro de regularización es clave para la obtención de modelos con una buena capacidad predictiva y que, a su vez, no presenten sobreajuste.

En la siguiente sección se estudiarán algunas técnicas disponibles para determinar este valor.

V. Selección de modelos

En las unidades anteriores se han visto modelos tanto supervisados como no supervisados que necesitan una serie de parámetros para su construcción. En esta unidad también se han visto extensiones de estos modelos —como LASSO— que requiere la utilización de parámetros para su construcción. La selección del parámetro óptimo no es trivial en la mayoría de las ocasiones. Para solucionar este problema, en scikit-learn se encuentra disponible el constructor GridSearchCV, con el cual se pueden crear modelos para la selección de parámetros. El constructor admite múltiples opciones, entre ellas destacan:

- estimator: el modelo que se va a probar.
- param_grid: un diccionario con los parámetros que se desean evaluar.
- scoring: una cadena de texto con la función utilizada para medir la calidad del modelo. Por defecto, utilizará el score del modelo.
- cv: un valor entero con el número de grupos con los que se realiza la validación cruzada de los resultados.

La validación cruzada es una técnica para detectar la existencia de sobreajuste en los modelos. Su utilización en la selección de los modelos garantiza que los resultados sean más estables. En la validación cruzada, el conjunto de datos es dividido en una cantidad de grupos y se utilizan todos menos uno para la construcción de un modelo y los restantes para su validación. Se repite el proceso tantas veces como grupos se hubiesen definido. Al repetir el proceso con diferentes datos se puede obtener un promedio de los modelos construidos e identificar si estos son estables o no, debido a la existencia de sobreajuste. En la figura 4.19. se muestra un esquema de esta técnica.

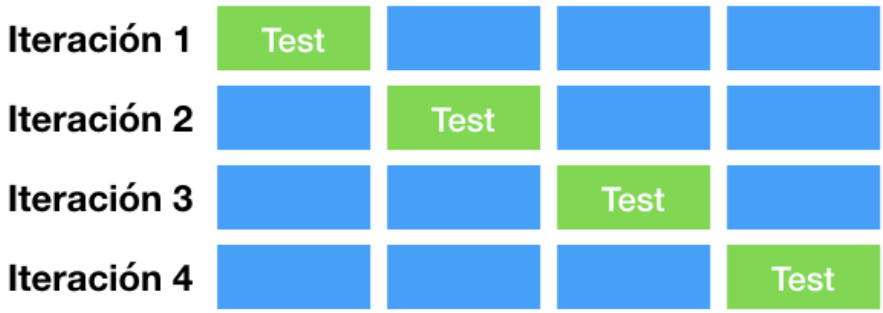

Figura 4.19. Validación cruzada.

La validación cruzada se puede utilizar para la selección de los parámetros con la que se ejecutan los algoritmos de aprendizaje, por ejemplo, el valor del alpha en una regresión LASSO. En scikit-learn, esta tarea se puede realizar empleando los objetos del constructor GridSearchCV. A estos objetos se les ha de indicar el modelo y los parámetros a probar. Las opciones más habituales son:

- estimator: un objeto con un modelo.
- param_grid: un diccionario en el que se indican los parámetros a probar como clave y el conjunto de elementos como valor.
- cv: el número de conjuntos en los que se dividen los datos para la validación cruzada. Al aumentar este valor aumenta el tiempo necesario para validar los modelos.

Una vez construido un objeto, ha de ser entrenado como si fuese un modelo, utilizando el método fit.

En la figura 4.20. se ha utilizado el constructor GridSearchCV para seleccionar el parámetro alpha que permite obtener el mejor modelo LASSO. En este código se ha creado un vector con los valores de los alpha a probar, se ha creado un modelo LASSO y un objeto con GridSearchCV. A este objeto se le ha pasado como parámetro el modelo, los alphas y se le ha indicado que realice validación cruzada de los resultados en 10 grupos.

```
from sklearn.model_selection import GridSearchCV

# Listado de alphas para ser evaluados
alphas = np.array([1, 0.1, 0.01, 0.001, 0.0001])

# Creación del modelo
model = Lasso()

# Selección del modelo
grid = GridSearchCV(estimator = model,
                    param_grid = dict(alpha = alphas),
                    cv = 10)
grid.fit(x, y)

# Los mejores parametros para el modelo
print('El mejor parametro es', grid.best_params_)
print('El mejor score es', grid.best_score_)
```

Figura 4.20. Selección de alpha para los modelos tipo LASSO.

Al ejecutar el código de la figura 4.20. se puede observar que el mejor resultado se obtiene con un valor de alpha de 0.001. Para decidir cuál es el mejor modelo, se utiliza el score, un valor que es diferente para cada tipo de modelo, como el R^2 en la regresión lineal, y la precisión en la regresión logística.

Una de las ventajas de GridSearchCV es que ofrece la posibilidad de probar múltiples parámetros de los modelos a la vez. Por ejemplo, se puede probar si es mejor un modelo con término independiente o sin él. Este caso se implementa en la figura 4.21.

```
fit_intercept = np.array([True, False])

# Selección del modelo
grid = GridSearchCV(estimator = model,
                    param_grid = dict(alpha = alphas,
                                      fit_intercept = fit_intercept),
                    cv = 10)
grid.fit(x, y)

# Los mejores parametros para el modelo
print('El mejor parametro es', grid.best_params_)
print('El mejor score es', grid.best_score_)
```

Figura 4.21. Selección de alpha y fit_intercept para los modelos tipo LASSO.

Una vez ejecutado el código de la figura 4.21. se obtienen los mismos resultados que en el caso anterior, indicando que la mejor opción es utilizar un modelo lineal con término independiente.

Hasta ahora ha sido necesario indicar al algoritmo los valores a probar de los parámetros, lo que no se conoce en muchas ocasiones. Una alternativa es utilizar el constructor RandomizedSearchCV junto a una distribución que genere números aleatorios, como puede ser sp_rand, para seleccionar aleatoriamente el valor de los parámetros. El constructor RandomizedSearchCV, además de las opciones utilizadas en GridSearchCV, admite otras entre las que destacan:

- param_dsitribution: es un diccionario en el que se indican las funciones utilizadas con la distribución de los parámetros.
- n_iter: es un entero con el número de veces que se repite el proceso.
- random_stat: es la semilla utilizada en el generador de números aleatorios.

En la figura 4.22. se muestra un código similar al de la figura 4.20., en donde se han reemplazado los parámetros escritos a mano por una distribución que genera números aleatorios y GridSearchCV por RandomizedSearchCV. En esta ocasión se ha fijado el número de iteraciones a 100 y la semilla a 1 para reproducir los resultados.

```
from scipy.stats import uniform as sp_rand
from sklearn.model_selection import RandomizedSearchCV

param_grid = dict(alpha = sp_rand())

rsearch = RandomizedSearchCV(estimator = model,
                             param_distributions = param_grid,
                             n_iter = 100,
                             cv = 10,
                             random_state = 1)
rsearch.fit(x, y)

# Los mejores parametros para el modelo
print('El mejor parametro es', rsearch.best_params_)
print('El mejor score es', rsearch.best_score_)
```

Figura 4.22. Selección de alpha para los modelos tipo LASSO utilizando valores aleatorios.

Al ejecutar el código de la figura 4.22., se obtiene como resultado un alpha de 0,000114, un valor que no se encontraba en la lista inicial.

VI. Resumen

En esta unidad se ha estudiado la importancia que tienen las características en el proceso de construcción de modelos. Una incorrecta elección de estas puede llevar a sobreajuste, en caso de que se empleen demasiadas, o a la falta de resultados, cuando no se seleccionan las suficientes o las más adecuadas. La unidad ha comenzado repasando los tipos de características que se pueden encontrar en los conjuntos de datos y algunas transformaciones que permiten mejorar la utilización de estas, como son la discretización y la normalización.

Posteriormente, se han estudiado varias técnicas para identificar las características que mejor se adaptan a cada modelo, comenzando por la selección univariante, donde simplemente se comprueba si las características son adecuadas para el objetivo buscado.

Otras técnicas útiles en los problemas de clasificación son el WoE y el IV. El primero permite identificar los niveles con mayor capacidad de discretización, mientras que el segundo facilita hacer lo mismo a nivel de característica.

La multicolinealidad es un problema muy habitual, ya que en muchos conjuntos de datos algunas características suelen estar relacionadas entre sí. Para identificar el problema se puede usar el VIF y mediante la utilización de un proceso iterativo se pueden eliminar las causas.

Una familia de técnicas muy populares en algunos entornos es Stepwise. Aunque no se encuentre disponible entre el conjunto de herramientas que provee scikit-learn, en ella la selección de las características se realiza de forma iterativa, probando a quitar o a añadir nuevas características en modelos ya existentes.

Otro método, el cual sí que se encuentra entre los disponibles en scikit-learn, es la regresión LASSO. Este es una regresión en la cual se penalizan las características que necesitan un gran esfuerzo para aportar información, de este modo, se eliminan, al mismo tiempo que se realiza la regresión, las menos interesantes.

La unidad ha terminado con el repaso a un conjunto de técnicas para seleccionar los parámetros utilizados durante la construcción de los modelos. En el ejemplo se estudió cómo identificar el valor más adecuado del parámetro de regularización en la regresión LASSO.

Ejercicios

Caso práctico

El archivo "myopia.csv" contiene los datos de un estudio realizado durante 5 años en el que se siguió la salud ocular de una población. Los registros se corresponden con los valores tomados inicialmente en el estudio y existe una variable "MYOPIC" en la que se registra si al sujeto se le diagnosticó miopía durante el estudio. Las variables son:

- ID: Identificador ID.
- STUDYYEAR: Año en el que se inició el estudio.
- MYOPIC: Desarrollo de la miopía durante los primeros cinco años.
- AGE: Edad en la primera visita.
- GENDER: Género.
- SPHEQ: Refracción esférica equivalente.
- AL: Longitud Axial.
- ACD: Profundidad de cámara anterior.
- LT: Grosor de la lente.
- VCD: Profundidad de cámara vítrea.
- SPORTHR: ¿Cuántas horas a la semana, fuera de la escuela, participa el niño en deportes o actividades al aire libre?
- READHR: ¿Cuántas horas a la semana, fuera de la escuela, lee el niño por placer?
- COMPHR: ¿Cuántas horas a la semana, fuera de la escuela, pasa el niño jugando a videojuegos de ordenador o frente a la pantalla?
- STUDYHR: ¿Cuántas horas a la semana, fuera de escuela, dedica el niño a las tareas escolares, leyendo o estudiando?
- TVHR: ¿Cuántas horas a la semana, fuera de la escuela, ve la televisión el niño?
- DIOPTERHR: Compendio de horas de actividades de trabajo cercano (actividad en la que el niño fija la vista en objetos que se sitúan a poca distancia) que se define como: DIOPTERHR = 3 (READHR + STUDYHR) + 2 COMPHR + TVHR
- MOMMY: ¿La madre del sujeto es miope?
- DADMY: ¿El padre del sujeto es miope?

A partir de este conjunto de datos se debe crear un modelo para predecir la aparición de miopía en el conjunto de estudio.

Solución

Para resolver este problema inicialmente se han de cargar los datos del archivo. Una vez hecho esto, se pueden analizar los tipos de características disponibles, para lo que se debe contar el número de registros en cada nivel. Esto se puede realizar utilizando el código de la figura 4.23.

```
import pandas as pd

myopia = pd.read_csv('myopia.csv', sep = ';')

# Separación de la variable objetivo y las explicativas
target = 'MYOPIC'
features = list(myopia.columns)
features.remove('MYOPIC')

# Listado de variables disponibles para hacer un modelo.
for var in features:
    print var , ':' , len(set(myopia[var]))
```

Figura 4.23. Carga de los datos de miopía.

El resultado que se obtiene al ejecutar el código es:

- ID: 618
- STUDYYEAR: 6
- AGE: 5
- GENDER: 2
- SPHEQ: 511
- AL: 254
- ACD: 206
- LT: 128
- VCD: 226
- SPORTHR: 40
- READHR: 16
- COMPHR: 18
- STUDYHR: 14
- TVHR: 28
- DIOPTERHR: 75
- MOMMY: 2
- DADMY: 2

Se puede comprobar que ID, como su nombre y la descripción de los datos indican, es un identificador. Por otro lado, STUDYYEAR es una característica que no puede ser utilizada, ya que, en caso de repetirse el estudio, esta no se encontrará disponible, al ser necesariamente posterior al estudio.

En las características discretas se puede medir el valor de WoE e IV para, en caso de que sea necesario, reagrupar los niveles. Las variables categóricas son:

- GENDER
- MOMMY
- DADMY

En esta parte también se puede estudiar la edad (AGE), aunque sea ordinal, ya que existen pocos valores.

La edad se puede discretizar y también se puede calcular el IV para comprobar que todos tienen un valor por encima de 0,5.

La variable continua SPHEQ también se puede discretizar, de lo que se obtendrá un valor de IV también elevado.

En este punto, una vez discretizadas algunas variables, se puede utilizar el VIF para eliminar aquellas que muestren multicolinealidad. En este paso hay que comprobar que las características READHR, COMPHR, STUDYHR y TVHR son perfectamente colineales y por lo tanto hay que eliminar una de ellas. El algoritmo de la figura 4.15. elimina READHR además de AL.

Finalmente, se crea un modelo y se utiliza la validación cruzada para comprobar que no existe sobreajuste.

En el notebook U5_Caso se incluye todo el código utilizado para la ejecución de este caso práctico.

Recursos

Bibliografía

- ***Applied Logistic Regression***. : Hosmer, D. W., Lemeshow, S. y Sturdivant R. X. Wiley; 2103. Third Edition.
- ***Bootstrapping Machine Learning***. : Dorard, L. Createspace Independent Publishing Platform (11 de agosto de 2014).
- ***Designing Machine Learning Systems with Python***. : Julian, D. Birmingham: Packt Publishing; 2016.
- ***Mastering Machine Learning With scikit-learn***. : Hackeling, G. Birmingham: Packt Publishing; 2014.
- ***scikit-learn Cookbook***. : Hauck, T. Birmingham: Packt Publishing; 2014.
- ***The Elements of Statistical Learning: Data Mining, Inference, and Prediction*** : Hastie, T., Tibshirani R., Friedman J. Springer Series in Statistics; 2017. Second Edition.

Glosario.

- **Binarización**: Transformación de una característica en otra de tipo booleana que solamente tiene dos niveles.
- **Característica cualitativa**: Aquella en la que los valores son niveles.
- **Característica cuantitativa**: Aquella en la que sus valores son continuos.
- **Característica ordinal**: Aquella en la que sus valores representan un orden, pero no informa de las dimensiones.
- **Cuartil**: Son los tres valores que dividen a los conjuntos de datos en cuatro partes porcentualmente iguales.
- **Discretización**: Transformación de una característica cualitativa u ordinal en otra categórica que solamente varios niveles.
- **Distancia de Hamming**: Distancia que se calcula como la diferencia entre dos valores ordinales.
- **Dummy**: Es la variable binaria que toma valor verdadero para indicar la presencia de una categoría y falso para su ausencia.
- **Multicolinealidad**: Fuerte correlación entre las variables independientes de un modelo.
- **Rango intercuartil**: La diferencia entre el tercer y el primer cuartil de una distribución.
- **Stepwise**: Familia de métodos para la selección de características en modelos basados en probar nuevos, en los que se agrega o elimina una y se comprueba si existe o no mejora.
- **Validación cruzada**: Método consistente en dividir el conjunto de datos en varios grupos para entrenar los modelos. Se hace en todos menos uno, el cual servirá para validar el resto.