**Escalamiento de datos**

Diversos algoritmos son sensibles a la escala en la que viene cada feature. **Re-escalarlos** puede traer significativas mejoras de rendimiento.

Existen distintas estrategias de escalamiento de los features, pero **la más común es la “Estandarización ”** donde convertimos la variable para que la distribución de esta **siga una distribución que es Gaussiana**: de media ” 0 “ y de desv. estándar “ 1 ”

**from** **sklearn.preprocessing** **import** StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train)

scaler.transform(X\_train) #Procederemos a transformar los datos

**Dato:** No muy importante, pero nos puede servir.

Cuando queremos borrar los warnings de Jupyter (Los que nos indican de futuros cambios, etc), escribimos lo siguiente.

import warnings

warnings.simplefilter("ignore")

*#Ahora compararemos los scores*

print(model.score(X\_test, Y\_test))

print(model\_scaled.score(X\_test\_scaled, Y\_test))

0.5343309013968608

0.5343309296578249

Vemos que los resultados son muy cercanos, por lo que podemos ver que no hay mejora de nuestros scores.

#### **Recuerda el reescalamiento es útil para algoritmos que no sean de regresión lineal**

## **Simplificar las transformaciones con pipelines**

La manera en cómo hicimos el re-escalamiento en lo anterior, fue un poco largo. Sin embargo, hay una manera más corta y más fácil de realizarlo .

De esta manera podemos juntar el escalador y el estimador y realizar el entrenamiento con solo **“ fit ”** y no estar haciendo primero la transformación.

Es el “**Pipeline “**

from sklearn.pipeline import make\_pipeline

**Ahora vamos a ver cómo crear nuevos features**

**from sklearn.preprocessing import PolynomialFeatures**

Es un método de Sklearn para crear nuevos features polinomiales

Ejemplo con una matriz simple

from sklearn.preprocessing import PolynomialFeatures

transformer = PolynomialFeatures(2)#Instanciamos el método y le indicamos que será un polinomio de grado 2

transformer.fit(A)

transformer.fit\_transform(A) #Aquí le creamos los nuevos features

En lo anterior, hicimos la transformación de nuestros features a un polinomio de grado 2 .Sin embargo, lo hicimos en 2 pasos debido a que primero estamos fiteando

model\_poly = make\_pipeline(PolynomialFeatures(2),

Lasso())

#Hacemos un pipeline para hacer la transformación en un solo paso

transformer = PolynomialFeatures(2)#Instanciamos el método y le indicamos que será un polinomio de grado 2

transformer.fit\_transform(A) #Aquí le creamos los nuevos features en un

#solo paso

Hay que tener mucho cuidado cuando aplicamos este método, ya que si lo aplicamos a un dataset que tiene muchos features, vamos a obtener un número muy grande de features y eso no es muy re**comendable para nuestro entrenamiento.**

**Variables Categóricas**

En terminos de Machine Learning a las features que pueden tomar un número finito de valores se les llama categóricas. Ejemplos para esto són: género, páis, grado académico, etc.

Para no introducir información falsa o erronéa en nuestro modelos existen formas más inteligentes de encodear nuestros datos.

***Encoding one-hot***

Este encoding consiste en asignarle una columna a cada categoría y rellenarla con 0 y 1 de la forma siguiente:

Ejm:

d = pd.DataFrame([['Chile','Colombia','Colombia','Venezuela'],['hombre' ,'mujer','hombre','mujer']])

d = d.T

d.columns = pd.Index(['pais','genero']) #Ojo con este método es para poner los índices a nuestro dataframe

d

Luego aplicamos un método de pandas que nos ayuda a separar los features categóricos y crear una columna para cada uno.

pd.get\_dummies(d)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **pais\_Chile** | **pais\_Colombia** | **pais\_Venezuela** | **genero\_hombre** | **genero\_mujer** |
| **0** | 1 | 0 | 0 | 1 | 0 |
| **1** | 0 | 1 | 0 | 0 | 1 |
| **2** | 0 | 1 | 0 | 1 | 0 |
| **3** | 0 | 0 | 1 | 0 | 1 |

Pero Recuerda que debes de tener mucho cuidado al aplicar, es útil cuando no tienes muchos valores únicos en tus features categóricos.

**Aquí una explicación**

Primero: ¿Cómo obtenemos cuántos valores únicos tiene cada feature categórico?

movies\_obj = pd.read\_csv('movies\_obj.csv') #Imp. nuestro dataset

movies\_obj.head()

Luego

movies\_obj.apply(pd.Series.nunique).sort\_values() #El sort\_values() es para que lo muestre de manera ordenada

color 2

content\_rating 18

language 47

country 65

genres 914

actor\_1\_name 2097

director\_name 2398

actor\_2\_name 3032

actor\_3\_name 3521

plot\_keywords 4760

movie\_title 4917

dtype: int64

Aquí vemos que “director\_name” tiene 2398 valores únicos. Si aplicamos one hot encoding, nuestro datos se van a distorsionar un montón y no es recomendable.

Un tip sería solo tomar el top 20 de esa variable y hacer un One Hot Encodding de esos 20, pero esto sería útil si nuestros 20 mejores se encuentran en más del 50% de nuestros datos (Osea que en más del 50% de películas han trabajadoesos 20 mejores)

**Encoding Binario**

Esta técnica no es canónica por lo que tendremos que buscarla en otra librería. Sin embargo el autor tuvo la buena idea de hacer su API compatible con la de sklearn, así que no tendremos ninguna dificultad en usarla.

!pip install category\_encoders #Es una librería que ha sido aportada por la comunidad de machine learning de Sklearn, pero no es oficial

categoricals = pd.read\_csv('categoricals.csv')

categoricals= categoricals.drop(['Unnamed: 0'], axis=1).set\_index('Unnamed: 0') #definimos al índice de nuestra tabla a Unnamed: 0

categoricals.head(2)

|  |  |  |
| --- | --- | --- |
|  | **actor\_1\_name** | **director\_name** |
| **Unnamed: 0** |  |  |
| **0** | CCH Pounder | James Cameron |

Hacemos unos ajustes a categoricals: quitamos la columna la columna de índices “unnamed: 0” para crear una con los índices desde el 0 hasta el 4101. De esta manera, podemos concatenarlo a nuestro dataset “X”, el cual tiene índices del 0 al 4101 también

categoricals = categoricals.reset\_index(drop=True).fillna(0)

categoricals.head(3)

X\_binenc = pd.concat([X, categoricals], axis =1)#lo concat. con ”X”

Ahora, importamos nuestra librería

import category\_encoders as ce #importamos la librería

encoder = ce.BinaryEncoder(cols= ['actor\_1\_name', 'director\_name']) # con "cols= " le decímos qué columnas queremos que nos transforme

X\_binenc = encoder.fit\_transform(X\_binenc) #hacemos la transformación y la guardamos en una variable

Preparamos datos de entrenamiento y prueba

xb\_train, xb\_test, y\_train, y\_test = train\_test\_split(X\_binenc, Y)

X\_train, X\_test = (xb\_train[X.columns], xb\_test[X.columns]) # Aquí solo seleccionamos las columnas de nuestro dataset "X" para poder utilizarlo como entrenamiento y test de nuestro modelo sin encoding binario

#Se hizo esto con fines de comparación: Comparar los scores

Creamos nuestros modelos

model\_binec = Lasso()

model = Lasso()

model\_binec.fit(xb\_train, y\_train)

model.fit(X\_train, Y\_train)

**“y\_train” e ”y\_test” que utilizamos son los mismos que generamos en nuestro train\_test\_split(X\_binenc, Y)**

#imprimimos nuestros scores

print(model\_binec.score(xb\_test, y\_test))

print(model.score(X\_test, y\_test))

0.5939433773958589

0.5948891057433912