Regresión Logística

Se dice que un proceso es binomial cuando sólo tiene dos posibles resultados: "éxito" y "fracaso", siendo la probabilidad de cada uno de ellos constante en una serie de repeticiones.

Un proceso binomial está caracterizado por la probabilidad de éxito, representada por p, la probabilidad de fracaso se representa por q y, evidentemente, ambas probabilidades están relacionadas por p+q=1. Se usa el cociente p/q, denominado "odds", y que indica cuánto más probable es el éxito que el fracaso, como parámetro característico de la distribución binomial aunque, evidentemente, ambas representaciones son totalmente equivalentes.

Los modelos de regresión logística nos permiten estudiar si una variable binomial depende, o no, de otra u otras variables (no necesariamente binomiales).

Ejemplo 1: Se quiere comparar la eficacia de dos tratamientos alternativos para una misma enfermedad. Asumiendo que el proceso "curar" sólo tiene dos resultados: sí o no y que la probabilidad de curación es la misma para todos los enfermos, se trata de un proceso binomial.

Se trata de ver si este proceso está asociado, o no, con el tratamiento, es decir, si la probabilidad de curación dado el tratamiento A es igual, o distinta, a la probabilidad de curación dado el tratamiento B. Supóngase que sobre una muestra aleatoria de 40 enfermos, dividida aleatoriamente en dos grupos de 20, a cada uno de los cuales se le suministra un tratamiento, se obtienen los siguientes resultados:

	tratamiento. A (X=1)	tratamiento. B (X=0)
curación	18	13
no	2	7
Total	**20**	**20**

Si se define la variable tratamiento como X=1 para el tratamiento A y X=0 para el B, a partir de la tabla podemos estimar la probabilidad de curación para el tratamiento B: p|(X=0)=13/20 y para el tratamiento A: p|(X=1)=18/20 Como ambas probabilidades son distintas, "parece" que la probabilidad de curación depende del tratamiento.

De aquí podemos sacar las medidas de asociación: diferencia de riesgo (DR), riesgo relativo (RR) y "odds ratio" (OR). En el ejemplo:

DR: 18/20 - 13/20 = 5/20 =0,25
RR: (18/20)/(13/20) = 18/13 = 1,38
OR: ((18/20)/(2/20))/(13/20)/(7/20) =(18x7)/(13x2) = 4,85

DR es 0 en caso de no diferencia, mientras que RR y OR son ambos 1.

Recordemos que el OR, aunque es la medida menos intuitiva es la más extendida por diversas razones y que es conveniente que a estas estimaciones puntuales las acompañemos de su intervalo de confianza que nos indica la precisión de la estimación.

En este caso significa que 4.85 personas se van a curar con el tratamiento A por cada una que se cure con el B.

Ejemplo 2: Sean dos juegos, en uno (X=0) se apuesta sobre la salida de una cierta cara en una tirada de un dado, y en otro (X=1) sobre la salida de una cara en la tirada de una moneda.

Evidentemente, la probabilidad de ganar es para el caso del dado p|(X=0)=1/6 y para la moneda p|(X=1)=1/2

El riesgo relativo es:

$$RR = \frac{p|(X=1)}{p|(X=0)} = \frac{1/2}{1/6} = 3$$

que, como es distinto de 1, quiere decir que la probabilidad de ganar está asociada al tipo de juego, y que es 3 veces más probable ganar con la moneda que con el dado. El odds ratio para este ejemplo es:

$$OR = \frac{p/q|(X=1)}{p/q|(X=0)} = \frac{\frac{1/2}{1/2}}{\frac{1/6}{5/6}} = 5$$

el odds para la moneda es 5 veces el odds del dado, es decir, que cuando se habrá ganado 5 veces con la moneda cuando se gane 1 vez en el dado. El OR está siempre más alejado de 1 que el RR, aunque cuando las probabilidades son muy pequeñas la diferencia (entre el OR y el RR) es pequeña.

Ejemplo 3: Se trata, ahora, de comparar el juego de la lotería nacional (X=1) en el que el premio es para un número extraído de entre 100.000, con el de la lotería tipo Melate (X=0) en que se premia una combinación de 6 números de entre las que se pueden formar con 49 números. Resulta p|(X=1)=1/100.000. El número de combinaciones de 6 números que se pueden formar con 49 es C49;6=13.983.816 por lo tanto p|(X=0)=1/13.983.816 y:

$$RR = \frac{p|(X=1)}{p|(X=0)} = \frac{1/100.000}{1/13.983.816} = 139.83816$$

Es aproximadamente 140 veces más probable ganar en el juego de la lotería que en el de la lotería tipo Melate. El odds ratio para este ejemplo es:

$$OR = \frac{p/q|X=1}{p/q|X=0} = \frac{\frac{1/100.000}{99.999/100.000}}{\frac{1/13.983.816}{13.983.815/13.983.816}} = \frac{13.983.815}{99.999} = 139.83954$$

Que, como era de esperar, debido a los pequeños valores de p|X=1 y p|X=0 es prácticamente igual que el riesgo relativo. Esto significa que por cada 140 personas que ganan la lotería 1 gana la lotería del tipo Melate.

Algunos Ejemplos de Regresión Logística son:

- Clasificar si el correo que llega es Spam o No es Spam
- Dados unos resultados clínicos de un tumor clasificar en “Benigno” o “Maligno”.
- El texto de un artículo a analizar es: Entretenimiento, Deportes, Política ó Ciencia
- A partir de historial bancario conceder un crédito o no
- Confiaremos en la implementación del paquete sklearn en Python para ponerlo en práctica.

Para el ejercicio usamos un CSV con datos de entrada a modo de ejemplo para clasificar si el usuario que visita un sitio web usa como sistema operativo Windows, Macintosh o Linux. Nuestra información de entrada son 4 características:

- Duración de la visita en Segundos
- Cantidad de Páginas Vistas durante la Sesión
- Cantidad de Acciones del usuario (click, scroll, uso de checkbox, sliders,etc)
- Suma del Valor de las acciones (cada acción lleva asociada una valoración de importancia)

Asignaremos los siguientes valores a las etiquetas:

0 – Windows
1 – Macintosh
2 -Linux

La muestra es pequeña: son 170 registros para poder comprender el ejercicio, pero recordemos que para conseguir buenos resultados siempre es mejor contar con un número abundante de datos que darán mayor exactitud a las predicciones y evitarán problemas de overfitting u underfitting. (Por decir algo, de mil a 5 mil registros no estaría mal).

Código:

Primero haremos los import necesarios:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sb
```

Leemos el archivo csv (por sencillez, se considera que estará en el mismo directorio que el archivo de notebook .ipynb) y lo asignamos mediante Pandas a la variable dataframe. Mediante el método dataframe.head() vemos en pantalla los 5 primeros registros.

```
dataframe = pd.read_csv(r"usuarios_win_mac_lin.csv")
dataframe.head()
```

	duracion	paginas	acciones	valor	clase
0	7.0	2	4	8	2
1	21.0	2	6	6	2
2	57.0	2	4	4	2
3	101.0	3	6	12	2
4	109.0	2	6	12	2

A continuación llamamos al método dataframe.describe() que nos dará algo de información estadística básica de nuestro set de datos. La Media, el desvío estándar, valores mínimo y máximo de cada característica.

```
dataframe.describe()
```

	duracion	paginas	acciones	valor	clase
count	170.000000	170.000000	170.000000	170.000000	170.000000
mean	111.075729	2.041176	8.723529	32.676471	0.752941
std	202.453200	1.500911	9.136054	44.751993	0.841327
min	1.000000	1.000000	1.000000	1.000000	0.000000
25%	11.000000	1.000000	3.000000	8.000000	0.000000
50%	13.000000	2.000000	6.000000	20.000000	0.000000
75%	108.000000	2.000000	10.000000	36.000000	2.000000
max	898.000000	9.000000	63.000000	378.000000	2.000000

Luego analizaremos cuantos resultados tenemos de cada tipo usando la función groupby y vemos que tenemos 86 usuarios “Clase 0”, es decir Windows, 40 usuarios Mac y 44 de Linux.

```
print(dataframe.groupby('clase').size())
```

```
clase
0    86
1    40
2    44
dtype: int64
```

Ahora cargamos las variables de las 4 columnas de entrada en X excluyendo la columna “clase” con el método drop(). En cambio agregamos la columna “clase” en la variable y. Ejecutamos X.shape para comprobar la dimensión de nuestra matriz con datos de entrada de 170 registros por 4 columnas.

```
X = np.array(dataframe.drop(['clase'],1))
y = np.array(dataframe['clase'])
X.shape
```

```
(170, 4)
```

Y creamos nuestro modelo y hacemos que se ajuste (fit) a nuestro conjunto de entradas X y salidas ‘y’.

```
model = linear_model.LogisticRegression()
model.fit(X,y)
```

Una vez compilado nuestro modelo, le hacemos clasificar todo nuestro conjunto de entradas X utilizando el método “predict(X)” y revisamos algunas de sus salidas y vemos que coincide con las salidas reales de nuestro archivo csv.

```
predictions = model.predict(X)
print(predictions)[0:5]
```

```
[2 2 2 2 2]
```

Y confirmamos si bueno fue nuestro modelo utilizando model.score() que nos devuelve la precisión media de las predicciones, en nuestro caso del 77%.

```
model.score(X,y)
```

```
0.77647058823529413
```

Una buena práctica en Machine Learning es la de subdividir nuestro conjunto de datos de entrada en un set de entrenamiento y otro para validar el modelo (que no se utiliza durante el entrenamiento y por lo tnto la máquina desconoce). Esto evitará problemas en los que nuestro algoritmo pueda falalar por “sobregeneralizar” el conocimiento.

Para ello, subdividimos nuestros datos de entrada en forma aleatoria (mezclados) utilizando 80% de registros para entrenamiento y 20% para validar.

```
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation =
model_selection.train_test_split(X, y, test_size=validation_size,
random_state=seed)
```

Volvemos a compilar nuestro modelo de Regresión Logística pero esta vez sólo con 80% de los datos de entrada y calculamos el nuevo scoring que ahora nos da 74%.

```
name='Logistic Regression'
kfold = model_selection.KFold(n_splits=10, random_state=seed)
cv_results = model_selection.cross_val_score(model, X_train,
Y_train, cv=kfold, scoring='accuracy')
msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(msg)

Logistic Regression: 0.743407 (0.115752)
```

Y ahora hacemos las predicciones -en realidad clasificación- utilizando nuestro “cross validation set”, es decir del subconjunto que habíamos apartado. En este caso vemos que los aciertos fueron del 85% pero hay que tener en cuenta que el tamaño de datos era pequeño.

```
predictions = model.predict(X_validation)
print(accuracy_score(Y_validation, predictions))

0.852941176471
```

Finalmente vemos en pantalla la “matriz de confusión” donde muestra cuántos resultados equivocados tuvo de cada clase (los que no están en la diagonal), por ejemplo predijo 3 usuarios que eran Mac como usuarios de Windows y predijo a 2 usuarios Linux que realmente eran de Windows.

```
print(confusion_matrix(Y_validation, predictions))
```

```
[[16  0  2]
 [ 3  3  0]
 [ 0  0 10]]
```

También podemos ver el reporte de clasificación con nuestro conjunto de Validación. En nuestro caso vemos que se utilizaron como “soporte” 18 registros windows, 6 de mac y 10 de Linux (total de 34 registros). Podemos ver la precisión con que se acertaron cada una de las clases y vemos que por ejemplo de Macintosh tuvo 3 aciertos y 3 fallos (0.5 recall). La valoración que de aquí nos conviene tener en cuenta es la de F1-score, que tiene en cuenta la precisión y recall. El promedio de F1 es de 84% lo cual no está nada mal.

```
print(classification_report(Y_validation, predictions))
```

	precision	recall	f1-score	support
0	0.84	0.89	0.86	18
1	1.00	0.50	0.67	6
2	0.83	1.00	0.91	10
avg / total	0.87	0.85	0.84	34

Como último ejercicio, vamos a inventar los datos de entrada de navegación de un usuario ficticio que tiene estos valores:

- Tiempo Duración: 10
- Páginas visitadas: 3
- Acciones al navegar: 5
- Valoración: 9

Lo probamos en nuestro modelo y vemos que lo clasifica como un usuario tipo 2, es decir, de Linux.

```
X_new = pd.DataFrame({'duracion': [10], 'paginas': [3],
'acciones': [5], 'valor': [9]})
model.predict(X_new)
```

```
array([2])
```