Regresión Logística

Objetivos

- Utilizar scikit Logistic Regression para clasificar.
- Entender la matriz de confusión.

Creará un modelo para una compañía de telecomunicaciones, para predecir si sus clientes la cambiarán por la competencia y así tomar acciones para retenerlos.

Tabla de contenido

Cuál es la diferencia entre la regresión lineal y la logística?

Mientras que la regresión lineal es apropiada para estimar valores continuos (por ejemplo el precio de una casa) no es la mejor herramienta para predecir la clase de un punto de datos observado. Para estimar la clase de un punto de datos necesitamos una guía acerca de cuál sería **clase más probable** de ese punto. Para esto podemos usar la regresión logística.

Recordemos la regresión lineal:

La regresión lineal encuentra una función que relaciona una variable dependiente continua y con alguos predictores (variables independientes x_1, x_2, etc.). Por ejemplo, la regresión lineal simple asume una función de la forma:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots$$

y encuentra los valores de los parámetros $\theta_0, \theta_1, \theta_2$, etc, donde el término θ_0 es la "intercepción". Puede formularse generalmente como:

$$h_\theta(x) = \theta^T X$$

La regresión logística es una variación de la regresión lineal, útil cuando la variable dependiente y es categórica. Produce una fórmula que predice la probabilidad de la clase de la etiqueta como función de las variables independientes.

La regresión logística ajusta una curva especial con forma de s, tomando la regresión lineal y transformando el estimado numérico en una probabilidad con la siguiente función, que es llamada sigmoide σ:

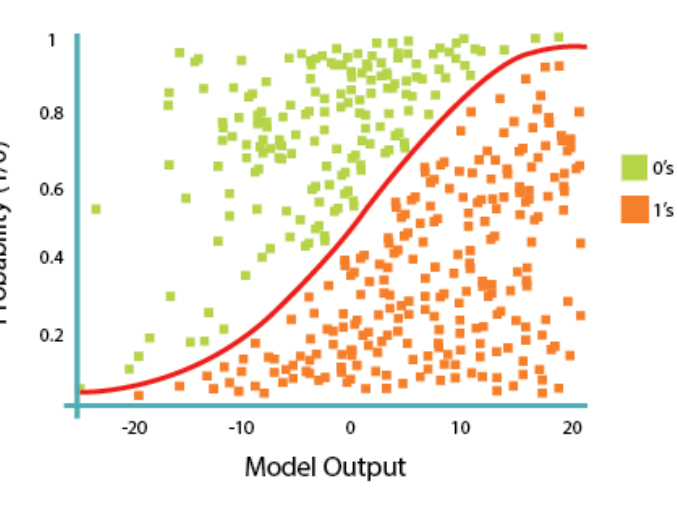
$$h_\theta(x) = \sigma(\theta^T X) = \frac{e^{(\theta_0+\theta_1 x_1+\theta_2 x_2+\ldots)}}{1+e^{(\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots)}}$$

o:

$$ProbabilityOfaClass_1 = P(Y=1|X) = \sigma(\theta^T X) = \frac{e^{\theta^T X}}{1+e^{\theta^T X}}$$

En esta ecuación, $\theta^T X$ es el resultado de la regresión (la suma de las variables ponderadas por los coeficientes), `exp` es la función exponencial y $\sigma(\theta^T X)$ es la sigmoide función logística

En resumen, la regresión logística pasa la entrada a través de la sigmoide pero luego trata el resultado como una probabilidad.



El objetivo de la regresión logística es encontrar los mejores parámetros θ, para $h_\theta(x) = \sigma(\theta^T X)$, de tal forma que el modelo prediga de la mejor forma posible la clase para cada caso.

Churn de clientes con regresión logística

Una compañía de telecomunicaciones está preocupada acerca del número de clientes que dejan su negocio por los competidores. Tienen que entender quiénes se van. Imagine es una anlista para esta compañía y debe encontrar quiénes se van y por qué.

Importemos las librerías necesarias:

In [2]:
```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
import matplotlib.pyplot as plt
```

Acerca del dataset

Utilizaremos un dataset de telecomunicaciones para predecir el churn. Es un dataset histórico de clientes, donde cada fila representa un cliente. Típicamente es más barato mantener clientes que conseguir nuevos, por lo que nos centraremos en predecir los clientes que se quedarán o no en la compañía.

Los datos brindan información para ayudar a predecir qué comportamiento posibilitará se retengan los clientes. Puede analizar los datos y desarrollar programas de retención de clientes especializados.

El dataset incluye información de:

- Clientes que se fueron el último mes - la columna se llama Churn
- Servicios que ha solicitado cada cliente: teléfono, líneas múltiples, internet, seguridad online, backup online, protección de dispositivos, soporte tecnológico, streaming y películas de TV.
- Información de la cuenta del cliente - por cuánto tiempo ha sido cliente, contrato, método de pago, facturación electrónica, cargos mensuales y totales
- Información demográfica del cliente - género, rango de edad y si tienen compañeros y dependientes.

Cargando los datos de Churn

Cada caso corresponde a un cliente y registra cierta información del mismo.

In [3]:
```
#Click here and press Shift+Enter
#!wget -O ChurnData.csv https://cf-courses-data.s3.us.cloud-object-storage.appdomain.c

import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
filename = 'ChurnData.csv'
urllib.request.urlretrieve(url, filename)
```

Out[3]: `('ChurnData.csv', <http.client.HTTPMessage at 0x220ce0490d0>)`

Cargando los datos desde el CSV

In [4]:
```
churn_df = pd.read_csv("ChurnData.csv")
churn_df.head()
```

Out[4]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	longmon	...	pager	internet	callw
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	4.40	...	1.0	0.0	
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	9.45	...	0.0	0.0	
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	6.30	...	0.0	0.0	
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	6.05	...	1.0	1.0	
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10	...	0.0	0.0	

5 rows × 28 columns

Pre-procesamiento y selección

Seleccionemos algunas características para el modelado. También cambiemos el datatype a entero, ya que es requerimiento por el algoritmo de skitlearn.

In [4]:
```
churn_df = churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip',
churn_df['churn'] = churn_df['churn'].astype('int')
churn_df.head()
```

Out[4]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0

Veamos la cantidad de filas y columnas.

In [6]:
```
churn_df.shape
```

Out[6]: `(200, 28)`

Definamos X e y

In [7]:
```
X = np.asarray(churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip
X[0:5]
```

Out[7]:
```
array([[ 11.,  33.,   7., 136.,   5.,   5.,   0.],
       [ 33.,  33.,  12.,  33.,   2.,   0.,   0.],
       [ 23.,  30.,   9.,  30.,   1.,   2.,   0.],
       [ 38.,  35.,   5.,  76.,   2.,  10.,   1.],
       [  7.,  35.,  14.,  80.,   2.,  15.,   0.]])
```

In [8]:
```
y = np.asarray(churn_df['churn'])
y [0:5]
```

Out[8]: `array([1., 1., 0., 0., 0.])`

Normalizemos:

In [9]:
```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[9]:
```
array([[-1.13518441, -0.62595491, -0.4588971 ,  0.4751423 ,  1.6961288 ,
        -0.58477841, -0.85972695],
       [-0.11604313, -0.62595491,  0.03454064, -0.32886061, -0.6433592 ,
        -1.14437497, -0.85972695],
       [-0.57928917, -0.85594447, -0.261522  , -0.35227817, -1.42318853,
        -0.92053635, -0.85972695],
       [ 0.11557989, -0.47262854, -0.65627219,  0.00679109, -0.6433592 ,
        -0.02518185,  1.16316   ],
       [-1.32048283, -0.47262854,  0.23191574,  0.03801451, -0.6433592 ,
         0.53441472, -0.85972695]])
```

Train/Test dataset

Dividimos en entrenamiento y test:

In [10]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (160, 7) (160,)
Test set: (40, 7) (40,)
```

Modelando

Construyamos el modelo utilizando **LogisticRegression** del paquete Scikit-Learn. Esta función implementa la regresión logística y puede utilizar diferentes optimizadores numéricos para encontrar los parámetros, incluyendo 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' solvers.

La versión de Scikit-Learn soporta la regularización, que es una técnica utilizada para resolver el problema del overfitting. El parámetro **C** indica el **inverso de la fuerza de regularización** que debe ser un float positivo. Valores más pequeños especifican regularizaciones más fuertes.

Ahora ajustemos y entrenemos nuestro modelo:

In [11]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR
```

Out[11]: `LogisticRegression(C=0.01, solver='liblinear')`

Ahora predecimos usando el conjunto de test:

In [12]:
```
yhat = LR.predict(X_test)
yhat
```

Out[12]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
       0., 0., 1., 0., 0., 0.])
```

predict_proba retorna estimados para todas las clases, ordenados por la etiqueta. Así, la primer columna es la probabilidad de la clase 1, P(Y=1|X) y la segunda de la clase 0, P(Y=0|X)

In [13]:
```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

Out[13]:
```
array([[0.54132919, 0.45867081],
       [0.60593357, 0.39406643],
       [0.56277713, 0.43722287],
       [0.63432489, 0.36567511],
       [0.56431839, 0.43568161],
       [0.55386646, 0.44613354],
       [0.52237207, 0.47762793],
       [0.60514349, 0.39485651],
       [0.41069572, 0.58930428],
       [0.6333873 , 0.3666127 ],
       [0.58068791, 0.41931209],
       [0.62768628, 0.37231372],
       [0.47559883, 0.52440117],
       [0.4267593 , 0.5732407 ],
       [0.66172417, 0.33827583],
       [0.55092315, 0.44907685],
       [0.51749946, 0.48250054],
       [0.485743  , 0.514257  ],
       [0.49011451, 0.50988549],
       [0.52423349, 0.47576651],
       [0.61619519, 0.38380481],
       [0.52696302, 0.47303698],
       [0.63957168, 0.36042832],
       [0.52205164, 0.47794836],
       [0.50572852, 0.49427148],
       [0.70706202, 0.29293798],
       [0.55266286, 0.44733714],
       [0.52271594, 0.47728406],
       [0.51638863, 0.48361137],
       [0.71331391, 0.28668609],
       [0.67862111, 0.32137889],
       [0.50896403, 0.49103597],
       [0.42348082, 0.57651918],
       [0.71495838, 0.28504162],
       [0.59711064, 0.40288936],
       [0.63808839, 0.36191161],
       [0.39957895, 0.60042105],
       [0.52127638, 0.47872362],
       [0.65975464, 0.34024536],
       [0.5114172 , 0.4885828 ]])
```

Evaluación

Índice de Jaccard

Podemos definirlo como el tamaño de la intersección dividido por el tamaño de la unión de las 2 clases de etiquetas. Si el conjunto completo de etiquetas predecidas para una muestra coincide estrictamente con las verdaderas la precisión del subconjunto es 1; de otro modo es 0.

In [15]:
```
#from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import jaccard_score
jaccard_score(y_test, yhat)
```

Out[15]: `0.375`

Matriz de confusión

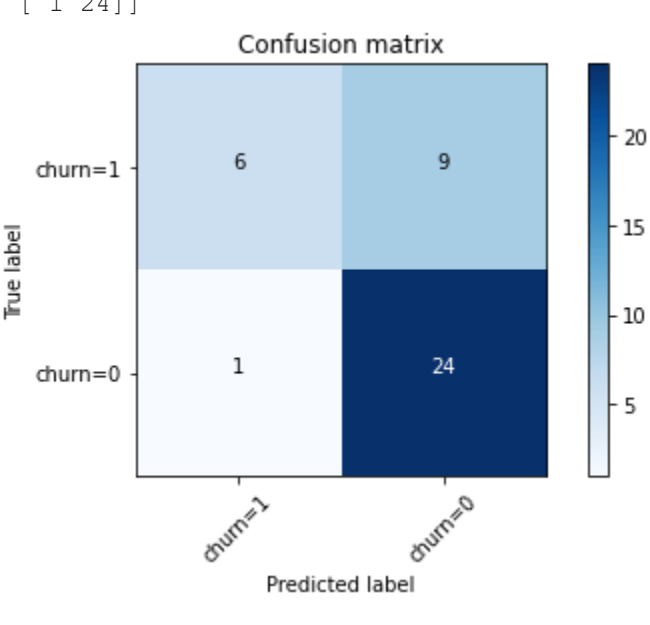
Es otra forma de ver la precisión del clasificador.

In [16]:
```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    Esta función imprime y grafica la matriz de confusión.
    La normalización puede aplicarse estableciendo "normalize=True"
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

```
[[ 6  9]
 [ 1 24]]
```

In [17]:
```
# Computamos la matriz de confusión
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Graficamos la matriz de confusión no normalizada
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1','churn=0'],normalize= False,  ti
```

```
Confusion matrix, without normalization
[[ 6  9]
 [ 1 24]]
```



Mire la primer fila. Corresponde a los clientes cuyo valor verdadero de churn en el conjunto de test es 1. De los 40 clientes, el valor de churn de 15 de ellos es 1. Y de esos 15, el clasificador predijo correctamente 6 de ellos como 1, y 9 como 0.

Signifca que para 6 clientes, el valor verdadero de churn es 1 en el conjunto de test y el clasificador los predijo correctamente. Los otros 9 los predijo mal; esto puede considerarse como un error del modelo para la primera columna.

Para los clientes con valor 0 de churn miramos la segunda fila. Hay 25 clientes para los cuales el valor de churn fue 0.

El clasificador predijo correctamente 24 de ellos como 0 y 1 erróneamente como 1; por lo que ha tenido una buena labor.

Algo bueno acerca de la matriz de confusión es que muestra la habilidad del modelo de predecir correctamente o separar las clases.

En el caso del clasificador binario, como en este ejemplo, podemos interpretar estos números como la cuenta de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos.

In [18]:
```
print (classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

         0.0       0.73      0.96      0.83        25
         1.0       0.86      0.40      0.55        15

    accuracy                           0.75        40
   macro avg       0.79      0.68      0.69        40
weighted avg       0.78      0.75      0.72        40
```

Basados en la cuenta de cada sección podemos calcular la precisión y el recall de cada etiqueta:

- **Precisión** es una medidad de la precisión siempre que se haya provisto una etiqueta de clase. Está definida por : precision = TP / (TP + FP)
- **Recall** es la tasa de verdaderos positivos. se define como: Recall = TP / (TP + FN)

Entonces, podemos calcular la precisión y recall de cada clase.

F1 score: Ahora podemos calcular el F1 score para cada etiqueta basados en la precisión y recall para esa etiqueta.

F1 score es el promedio armónico de la precisión y el recall, siendo su mejor valor 1 (precisión y recall perfectos) y su peor valor 0. Es una buena forma de mostrar que un clasificador tiene buenos valores tanto de precisión como de recall.

Finalmente, podemos decir la precisión promedio para el clasificador, que es el promedio del F1 score para ambas etiquetas; 0.72 en nuestro caso.

log loss

Ahora probemos **log loss** para la evaluación.En la regresión logística, la salida puede ser la probabilidad de que el churn del cliente sea "si" (o igual a 1). Log loss (Logarithmic loss) mide la performance de un clasificador donde la salida predecida es una probabilidad.

In [19]:
```
from sklearn.metrics import log_loss
log_loss(y_test, yhat_prob)
```

Out[19]: `0.6017092478101187`

Práctica

Construya el modelo de regresión otra vez, pero esta vez con diferentes valores de solver y regularization. Calcula la nueva log loss.

In [20]:
```
LR2 = LogisticRegression(C=0.01, solver='sag').fit(X_train,y_train)
yhat_prob2 = LR2.predict_proba(X_test)
print ("LogLoss: : %.2f" % log_loss(y_test, yhat_prob2))
```

```
LogLoss: : 0.61
```