Regresión Logística

La regresión logística es una de las formas más básicas de modelización de regresión. Es parte de una familia de "modelos lineales generalizados".Esto significa básicamente que la fórmula es muy similar a la de una regresión lineal. Sin embargo, como la variable de destino es binaria, en lugar de una variable numérica continua, la variable de destino tiene que ser modificada para ajustarse a esta fórmula.

Los modelos lineales, también pueden ser utilizados para clasificaciones; es decir, que primero ajustamos el modelo lineal a la probabilidad de que una cierta clase o categoría ocurra y, a luego, utilizamos una función para crear un umbral en el cual especificamos el resultado de una de estas clases o categorías. La función que utiliza este modelo, no es ni más ni menos que la función logística.

A grandes rasgos,hay 2 formas de clasificar datos,una asignando categorías y otra asignando la probabilidad de pertenecer a una categoría.

Podemos clasificar de dos formas, mediante discriminación o asignando probabilidades. Discriminando, asignamos a cada x una de las k clases Ck. Desde un punto de vista probabilístico, lo que haríamos es asignar a cada x la probabilidad de pertenecer a la clase Ck.

Con la función make_classification de scikit-learn, creamos un conjunto de datos para clasificar.

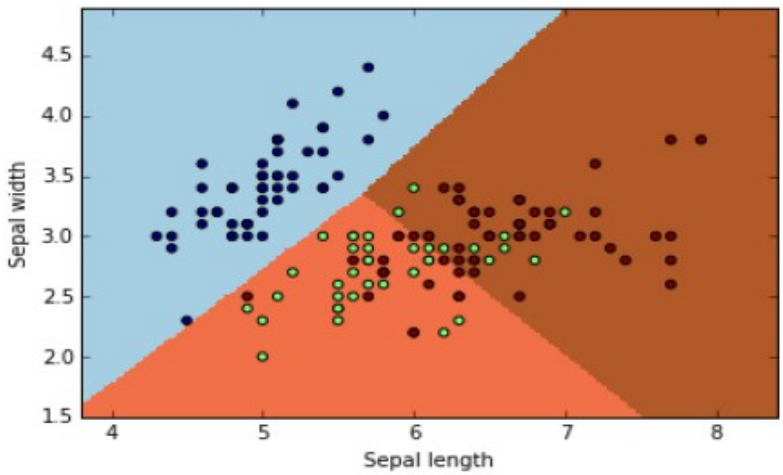
En scikit-learn podemos obtener clasificaciones de ambas maneras una vez entrenado el modelo.

- **modelo.predict()**, para asignar una categoría.
- **modelo.predict_proba()**, para determinar la probabilidad de pertenencia.

Logistic Regression como algoritmo de regresión logística

La implementación de la regresión logística en scikit-learn se puede acceder desde la clase **LogisticRegression**.En el siguiente ejemplo clasificaremos las flores del iris dataset de acuerdo a la longitud y anchura del sépalo:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model, datasets

#importar iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]
Y = iris.target

#Entrenemos los datos con  LogisticRegression
model = linear_model.LogisticRegression(C=10000) # C = 1/alpha
model.fit(X, Y)

x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5

h = .02
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

#Pintar el gráfico

Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.show()
```



También podríamos aplicar cross_validation para dividir los datos entre datos de entrenamiento y datos de test,para posteriormente realizar una predicción sobre los datos y obtener los coeficientes de rendimiento y precisión

```
import numpy as np
from sklearn import linear_model, datasets, cross_validation, metrics

#importar iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]
Y = iris.target

#Definimos el modelo LogisticRegression
model = linear_model.LogisticRegression(C=10000)

#Dividimos en dataset
x_train, x_test, y_train, y_test = cross_validation.train_test_split(X, Y)

#Entrenemos los datos con  el modelo
model.fit(x_train, y_train)

#Realizamos una prediccion
y_pred = model.predict(x_test)

#Obtenemos rendimiento
print("Accuracy: %2f" % metrics.accuracy_score(y_test, y_pred))
print("Precision: %2f" % metrics.precision_score(y_test, y_pred,average="macro"))
print("F1: %2f" % metrics.f1_score(y_test, y_pred, average="macro"))

# Accuracy: 0.815789
# Precision: 0.817460
# F1: 0.813789
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression

lr_model =LogisticRegression(C=0.7, random_state=42)
lr_model.fit(X_train, y_train.ravel())
lr_predict_test = lr_model.predict(X_test)

# training metrics
print("Accuracy: {0:.4f}".format(metrics.accuracy_score(y_test, lr_predict_test)))
print(metrics.confusion_matrix(y_test, lr_predict_test, labels=[1, 0]) )
print("")
print("Classification Report")
print(metrics.classification_report(y_test, lr_predict_test, labels=[1,0]))
```

```
Accuracy: 0.7446
[[ 44  36]
 [ 23 128]]

Classification Report
             precision    recall  f1-score   support

          1       0.66      0.55      0.60        80
          0       0.78      0.85      0.81       151

avg / total       0.74      0.74      0.74       231
```