Dividir datos entrenamiento y test

***Utilizamos la* función train_test_split para dividir aleatoriamente los datos en conjuntos de entrenamiento y prueba** . Las proporciones de los datos para ambas particiones se pueden especificar usando argumentos de palabra clave. De forma predeterminada, el 25% de los datos se asignan al conjunto de pruebas

Scikit-learn contiene una función que baraja el conjunto de datos y lo divide en subconjuntos de datos: la función **train_test_split**. Esta función extrae el 75% de las filas de los datos como conjunto de entrenamiento, junto con las etiquetas correspondientes para estos datos. El 25% restante de los datos, junto con las etiquetas restantes, se declara como el conjunto de prueba.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(iris_dataset['data'],iris_dataset['target'],random_state=0)
```

Validación cruzada en scikit-learn

La validación cruzada se implementa en scikit-learn utilizando la función **cross_val_score** del módulo model_selection.

Los parámetros de la función cross_val_score son el modelo que queremos evaluar, los datos de entrenamiento y las etiquetas. Vamos a evaluar LogisticRegression en el conjunto de datos iris:

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

iris = load_iris()
logreg = LogisticRegression()

scores = cross_val_score(logreg, iris.data, iris.target)
print("Cross-validation scores: {}".format(scores))
```

De forma predeterminada, **cross_val_score** realiza tres veces la validación cruzada, devolviendo tres valores de precisión. Podemos cambiar el número de validaciones al cambiar el parámetro cv:

```
scores = cross_val_score(logreg, iris.data, iris.target, cv=5)
print("Cross-validation scores: {}".format(scores))
```

En la validación cruzada, con el parámetro se puede decidir sobre el número de validaciones a realizar. Supongamos que usamos cv=3:

```
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(clf, X_train, y_train, cv=3,scoring='accuracy',
n_jobs=-1)
print "ExtraTreesClassifier -> cross validation accuracy: mean = %0.3f std =
%0.3f" % (np.mean(scores), np.std(scores))
```

Obtener la matriz de confusión

Una matriz de confusión es una tabla que utilizamos para entender el desempeño de un modelo de clasificación. Esta nos ayuda a entender cómo clasificamos los datos de prueba en diferentes clases. Cuando queremos afinar nuestro algoritmo, tenemos que entender cómo los datos se clasifican antes de hacer estos cambios en los mismos.

Ahora que tenemos una estimación de base de la precisión en el conjunto de entrenamiento, vamos a ver lo bien que se realiza en el conjunto de pruebas. En este caso generamos la matriz de confusión ya que queremos obtener los falsos positivos y falsos negativos. **Para obtener la matriz de confusión usamos el método confusion_marix dentro del paquete sklearn.metrics.**

```
y_pred=clf.predict(X_test)

from sklearn.metrics import confusion_matrix

confusionMatrix = confusion_matrix(y_test, y_pred)
print confusionMatrix

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)

OUTPUT:
[[6610 448]
[1238 704]]

Our overall test accuracy: 0.81266666666666665
```

from sklearn.metrics import confusion_matrix

Una matriz de confusión es una tabla que utilizamos para entender el desempeño de un modelo de clasificación. Esto nos ayuda a entender cómo clasificamos los datos de prueba en diferentes clases. Cuando queremos afinar nuestros algoritmos, necesitamos entender cómo los datos se clasifican erróneamente antes de realizar estos cambios. Algunas clases son peores que otras, y la matriz de confusión nos ayudará a entender esto.

Por último, nos metemos de lleno al modelo. Éste, en común a la mayoría de modelos en scikit-learn, dispone de una serie de funciones que se ejecutan paso a paso.

- ***nombre-del-modelo*.fit()**
- ***nombre-del-modelo*.predict()**
- ***nombre-del-modelo*.score()**

Con la función fit() entrenamos el modelo para obtener los parámetros que utilizaremos sobre los datos de test con la función predict(). Finalmente, con score() podremos obtener una estimación de la capacidad de acierto de nuestro modelo sobre los datos de trabajo.

Rendimiento sobre los datos de entrenamiento(X_train,y_train)

```
# predict values using the training data
nb_predict_train = nb_model.predict(X_train)

# import the performance metrics library
from sklearn import metrics

# Accuracy
print("Accuracy: {0:.4f}".format(metrics.accuracy_score(y_train, nb_predict_train)))
print()
```

```
Accuracy: 0.7542
```

Rendimiento sobre los datos de test(X_test,y_test)

```
# predict values using the testing data
nb_predict_test = nb_model.predict(X_test)

from sklearn import metrics

# training metrics
print("Accuracy: {0:.4f}".format(metrics.accuracy_score(y_test, nb_predict_test)))
```

```
Accuracy: 0.7359
```

Matriz de confusión

```
print("Confusion Matrix")
# Note the use of labels for set 1=True to upper left and 0=False to lower right
print("{0}".format(metrics.confusion_matrix(y_test, nb_predict_test, labels=[1, 0])))
print("")

print("Classification Report")
print(metrics.classification_report(y_test, nb_predict_test, labels=[1,0]))
```

```
Confusion Matrix
[[ 52  28]
 [ 33 118]]

Classification Report
             precision    recall  f1-score   support

          1       0.61      0.65      0.63        80
          0       0.81      0.78      0.79       151

avg / total       0.74      0.74      0.74       231
```

Entre las **métricas de rendimiento** que se obtienen con el **classification report** podemos destacar:

Precisión se refiere al número de elementos que se clasifican correctamente como un porcentaje del número total de elementos del dataset.

Recall se refiere al número de elementos que se recuperan como un porcentaje del número total de elementos en los datos de entrenamiento