Curso de especialización en Inteligencia Artificial y Big Data

UD04. DESARROLLO DE APLICACIONES DE IA: ALGORITMOS

EVALUACIÓN DE MODELOS

Carlos Sáenz Adán

1. Evaluación de los resultados

La evaluación de un modelo de aprendizaje en Inteligencia Artificial forma parte de su ciclo de vida. Gracias a la evaluación podemos cuantificar la calidad del modelo y su capacidad de predicción. En la mayoría de las ocasiones existe una realimentación entre la fase de evaluación y la de diseño, tal que una evaluación con resultados insatisfactorios da pie a modificaciones sobre el diseño para mejorar la calidad.

Los métodos de evaluación dependen del tipo de modelo; de esta forma, los modelos de aprendizaje automático como, por ejemplo, métodos de regresión lineal y métodos de clustering emplean unos métodos de evaluación distintos a los empleados en modelos de aprendizaje supervisado, principalmente métodos de clasificación a partir de un conjunto de ejemplos etiquetados previamente.

La biblioteca Scikit-learn es una biblioteca, libre, para el lenguaje de programación Python e incluye un amplio repertorio de herramientas con métodos de Machine Learning e Inteligencia Artificial. Dicha biblioteca facilita la evaluación de modelos en todas sus variantes y será la que emplearemos.

En la documentación oficial de Scikit-learn y, en particular, en el apartado de modelos de evaluación (https://scikit-learn.org/stable/modules/model_evaluation.html) se ofrece una descripción detallada de las diferentes métricas soportadas para cuantificar la calidad de las predicciones de un modelo. Dichas métricas están especialmente divididas en tres categorías:

- Métricas de clasificación
- Métricas de clustering
- Métricas de regresión

En concreto las métricas soportadas por Scikit-learn para problemas de clasificación son:

- ‘accuracy’ metrics.accuracy_score
- ‘balanced_accuracy’ metrics.balanced_accuracy_score
- ‘top_k_accuracy’ metrics.top_k_accuracy_score
- ‘average_precision’ metrics.average_precision_score
- ‘neg_brier_score’ metrics.brier_score_loss
- ‘f1’ metrics.f1_score
- ‘f1_micro’ metrics.f1_score
- ‘f1_macro’ metrics.f1_score
- ‘f1_weighted’ metrics.f1_score
- ‘f1_samples’ metrics.f1_score
- ‘neg_log_loss’ metrics.log_loss
- ‘precision’ etc. metrics.precision_score
- ‘recall’ etc. metrics.recall_score
- ‘jaccard’ etc. metrics.jaccard_score
- ‘roc_auc’ metrics.roc_auc_score
- ‘roc_auc_ovr’ metrics.roc_auc_score
- ‘roc_auc_ovo’ metrics.roc_auc_score
- ‘roc_auc_ovr_weighted’ metrics.roc_auc_score
- ‘roc_auc_ovo_weighted’ metrics.roc_auc_score

Con el listado anterior, puedes hacerse una idea general de un conjunto representativo de métricas para la evaluación de modelos de clasificación en aprendizaje supervisado. En Scikit-learn podemos

encontrar todas estas métricas dentro de sklearn.metrics. Puedes encontrar más información en el siguiente enlace.

https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics

Los modelos de aprendizaje supervisado parten de un conjunto de muestras (Dataset), siendo cada muestra una tupla formada por un conjunto de valores correspondientes a las variables de entrada, y una salida o clase a la que pertenece la muestra. Es decir, el modelo aprende a partir de un conjunto de ejemplos catalogados correctamente a priori. Por ejemplo, un Dataset muy popular (ver Figura 1) con el que los diseñadores prueban sus modelos es Iris, disponible en la mayor parte de plataformas de IA (Scikit learn, no es una excepción) y contiene 50 muestras de cada una de tres especies (150 ejemplos en total) de la flor Iris (iris setosa, iris virginica, iris versicolor).

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

Figura 1. Dataset de Iris como DataFrame

En este conjunto de datos las variables de entrada son: longitud del sépalo, anchura del sépalo, longitud del pétalo y anchura del pétalo. Por tanto, cada tupla está formado por cuatro valores de entrada, uno para cada variable de las mencionadas anteriormente, y la clase a la que pertenece (setosa, virginica, versicolor). A partir de los 150 ejemplos del Dataset, el modelo aplicado debe ser capaz de clasificar una nueva flor a partir de los cuatro valores de entrada. Precisamente, los métodos de evaluación nos ayudarán a medir la calidad de esa predicción.

```
from sklearn import datasets
iris = datasets.load_iris(as_frame=True)
```

1.1. Evaluación de modelos de clasificación

Matriz de confusión

La matriz de confusión no puede ser considerada una métrica como tal, pero es una herramienta fundamental para evaluar y optimizar los modelos de clasificación, ya que ayuda a profundizar en el tipo de error que el modelo está cometiendo y a comprender otras métricas que se emplean.

Una matriz de confusión se representa mediante una tabla en la que las columnas corresponden a los valores reales y las filas a los valores predichos. En el caso particular de un modelo de clasificación binaria, esta tabla se confeccionará de la siguiente forma:

		Valores Reales	
Valores Predichos	Positivo (A)	Positivo (A)	Negativo (B)
		TP	FP
	Negativo (B)	FN	TN

Supongamos un modelo de clasificación binaria, es decir, aquel en el que tan solo se pueden asignar dos clases diferentes (A y B) para cada ejemplo de entrada. Supongamos también un modelo cuyo objetivo es la clasificación de un tipo de cáncer como maligno (A) o Benigno (B). En función del valor real y el valor predicho por el modelo, podemos encontrarnos cuatro situaciones posibles (ver tabla anterior):

- Verdadero Positivo (True Positive, TP): el cáncer realmente es maligno (clase A) y ha sido clasificado como tal.
- Falso Positivo (False Positive, FP): el cáncer realmente es benigno (B) y el modelo lo ha clasificado como maligno (A).
- Falso Negativo (False Negative, FN): el cáncer realmente es maligno (A) y el modelo lo clasifica como benigno (B).
- Verdadero Negativo (True Negative, TN): el cáncer es benigno (B) y el modelo lo clasifica como benigno (B).

Por tanto, los Verdaderos (positivos y negativos) representan casos que han sido correctamente clasificados, mientras que los Falsos (positivos y negativos) representan errores de clasificación. De esta forma, nos interesa que la cantidad de ejemplos contabilizados en la diagonal principal (marcada en color verde en la anterior tabla) sea mucho mayor a los contabilizados en diagonal secundaria (marcados en color rojo en la anterior tabla), lo que querrá decir que nuestro modelo se aproxima a un diagnóstico fiable.

En función de los valores obtenidos, se pueden realizar modificaciones sobre el modelo con la intención de reducir el número de falsos positivos o falsos negativos (o incluso ambos). Dependiendo del contexto en el que esté enmarcado el modelo, será más crítico reducir uno u otro.

Por ejemplo, en el contexto médico que planteábamos anteriormente, los casos especialmente críticos corresponderían con los Falsos Negativos (FN), es decir, casos de tumores malignos diagnosticados como benignos, lo cual repercutiría gravemente en la salud del paciente.

Por otro lado, en un contexto jurídico en el que un sospechoso puede ser clasificado como culpable (A) o inocente (B), los casos más críticos los representan los Falsos Positivos (FP), es decir, una persona inocente que ha sido catalogada como culpable, con el riesgo de terminar en prisión de forma injusta.

El siguiente fragmento de código muestra un ejemplo escrito en Python con la librería Scikit-learn en donde se define un vector básico de clases con valores reales (y_real) y otro con los valores predichos (y_pred). Ambos vectores son empleados para calcular la matriz de confusión.

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

import matplotlib.pyplot as plt
import seaborn as sns

y_real =  [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
y_pred = [1, 1, 1, 1, 0, 1, 1, 0, 0, 0]

cm = confusion_matrix(y_real, y_pred)
print (cm)
```

Si has ejecutado el anterior código verás que no es muy ilustrativo, aunque contiene toda la información necesaria. Si queremos generar una matriz de confusión más visual se puede crear un heatmap con esta matriz de confusión.

```
#Creación de Figura
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax=ax, cmap='Greens');

# labels, title and ticks
ax.set_xlabel('Valores Predichos');ax.set_ylabel('Valores Reales');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels([1, 0]);
ax.yaxis.set_ticklabels([1, 0]);
plt.show()
```

Por otro lado, para aquellos modelos de clasificación no binaria, en los que un ejemplo de entrada puede ser clasificado en n clases diferentes, también es posible la representación de un matriz de confusión. En este caso, las columnas representan el valor verdadero o la clase correcta a la que corresponde el ejemplo, y las filas las clases predichas. La siguiente tabla muestra una representación general de este tipo de matrices.

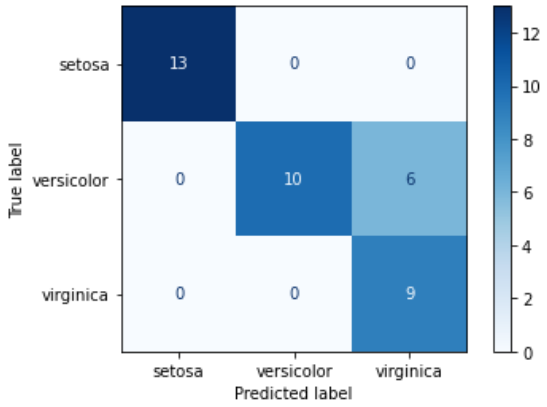
		Clases verdaderas			
		A	B	...	X
Clases Predichas	A				
	B				
	...				
	X				

La diagonal principal representa los casos que han sido correctamente clasificados, es decir, la clase predicha corresponde con la clase real del ejemplo de entrada. El resto de las celdas representan casos de errores en la clasificación.

Un fragmento de código en el que se lleva a cabo una clasificación de estas características es el que se muestra a continuación.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay

#Importación de Iris Dataset
iris = datasets.load_iris()
#Conjunto de ejemplos
X = iris.data
#Clases
y = iris.target
#nombre de las clases
class_names = iris.target_names

# División del DataSet en dos cojuntos: entrenamiento y test
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Usamos como clasificador Support Vector Classification
classifier = svm.SVC(kernel='linear', C=0.01).fit(X_train, y_train)

disp = ConfusionMatrixDisplay.from_estimator(
    classifier,
    X_test,
    y_test,
    display_labels=class_names,
    cmap=plt.cm.Blues
)
plt.show()
```

Como resultado de esta ejecución obtenemos una matriz de confusión con el siguiente aspecto.



Métricas para la evaluación de los resultados

Teniendo en cuenta los valores de una matriz de confusión, se pueden establecer diferentes métricas de evaluación.

		Valores Reales	
Valores Predichos	Positivo (A)	Positivo (A)	Negativo (B)
		TP (VP)	FP (FP)
	Negativo (B)	FN (FN)	TN (VN)

Recordamos el significado de las siglas:

- TP/VP: True Positive/Verdadero Positivo
- FP/FP: False Positive/Falso Positivo
- FN/FN: False Negative/Falso Negativo
- TN/VN: True Negative/Verdadero Negativo

Exactitud (accuracy)

Documentación oficial:

https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score

La exactitud es el ratio entre las predicciones correctas: suma de verdaderos (positivos y negativos) y las predicciones totales. Dicho de otra forma, de todos los ejemplos ¿cuántos se predijeron correctamente? En el caso de una clasificación binomial el cálculo es tan sencillo como:

$$\frac{VP + VN}{TOTAL}$$

En el caso planteado en la siguiente tabla, el cálculo de la exactitud del modelo, sería el siguiente:

		Valores Reales	
Valores Predichos	Positivo (A)	Positivo (A)	Negativo (B)
		8	2
	Positivo (B)	1	9

$$\text{Accuracy} = \frac{8+9}{8+9+2+1} = 0{,}85$$

De forma general, la exactitud de un modelo multiclase podría representarse de la siguiente forma:

$$accuracy\_score(y, \hat{y}) = \frac{1}{n}\Sigma_{i=0}^{n-1}(\hat{y} = \hat{y}_i)$$

Donde n representa el número de muestras (o ejemplos) totales, y la clase real e ŷ la clase predicha. Veamos cómo representar el ejemplo anterior en Python y realizar el cálculo de la exactitud en mediante sklearn:

```
from sklearn.metrics import accuracy_score

y_pred=['A','A','A','A','A','A','A','A','B','A','A','B','B','B','B','B','B','B','B']

y_true=['A','A','A','A','A','A','A','A','A','B','B','B','B','B','B','B','B','B','B']

print(accuracy_score(y_true, y_pred))
```

Precisión

Documentación oficial:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html#sklearn.metrics.precision_score

La precisión determina el grado de acierto en la clase relevante. Suponiendo que la clase relevante sea A (tumor maligno en los ejemplos planteados anteriormente), la precisión sería la proporción entre los verdaderos positivos (TP) y el total de casos positivos (primera fila de la tabla). Con ello podemos saber cuántas de las muestras que fueron catalogadas como casos positivos eran realmente positivas:

$$\frac{VP}{VP+FP}$$

El cálculo para el caso planteado en la tabla de ejemplo sería:

$$\text{precission} = \frac{8}{8+2} = 0{,}8$$

```
from sklearn.metrics import precision_score

y_pred =['A','A','A','A','A','A','A','A','B','A','A','B','B','B','B','B','B','B','B']

y_true =['A','A','A','A','A','A','A','A','A','B','B','B','B','B','B','B','B','B','B']

print(precision_score(y_true, y_pred, labels=['A', 'B'], pos_label='A', average="binary"))
```

La precisión también podría calcularse para la clase secundaria mediante la relación entre verdaderos negativos (VN) y la suma de VN + FN:

$$\frac{VN}{VN + FN}$$

La interpretación del dato en este caso es similar al anterior: cuántas de las muestras negativas reales han sido realmente detectadas. En el ejemplo de la clasificación de tumores, este valor serviría para conocer la proporción de tumores benignos detectados.

Exhaustividad (Recall)

Documentación oficial:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html#sklearn.metrics.recall_score

Métrica también conocida como sensibilidad (sensitivity) o Ratio de Verdaderos Positivos (TPR, True Positive Rate), es un ratio entre los verdaderos positivos y la suma de los verdaderos positivos y los falsos negativos. El valor obtenido nos ofrece una idea clara sobre cuántos de los casos catalogados como positivos fueron predichos correctamente. De vuelta al ejemplo anterior, la exhaustividad determinaría cuántos de los tumores clasificados como malignos, son realmente malignos. ¿Cuál sería el número real de tumores malignos? La suma de TP (tumores malignos detectados correctamente) y los FN (tumores malignos no detectados).

$$\frac{VP}{VP + FN}$$

El cálculo para el caso planteado en la tabla de ejemplo sería:

$$\text{recall} = \frac{8}{8 + 1} = \ 0{,}88$$

```
from sklearn.metrics import recall_score

y_pred=['A','A','A','A','A','A','A','A','B','A','A','B','B','B','B','B','B','B','B']

y_true=['A','A','A','A','A','A','A','A','A','B','B','B','B','B','B','B','B','B','B']

print(recall_score(y_true, y_pred, labels=['A', 'B'], pos_label='A', average="binary"))
```

Pérdida Logarítmica (Log Loss)

Documentación oficial:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html#sklearn.metrics.log_loss

La pérdida logarítmica es una puntuación única que representa la ventaja del clasificador sobre una predicción aleatoria. La pérdida logarítmica mide la incertidumbre del modelo comparando las probabilidades de sus resultados con los valores conocidos (verdad de base). El objetivo es minimizar la pérdida logarítmica para el modelo en su conjunto, cuanto menor sea el valor, menor será la incertidumbre y mejor será la capacidad del modelo para clasificar correctamente las muestras de entrada. El cálculo de la pérdida logarítmica sería el siguiente:

$$LogLoss = -\frac{1}{n}\Sigma_{i=1}^{n}[y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)]$$

Donde n representa el número de muestras (o ejemplos) totales, y la clase real e ŷ la clase predicha. En Python y la librería sklearn el cálculo sería tan sencillo como se muestra a continuación:

```
from sklearn.metrics import log_loss
y_pred =[0,0,0,0,0,0,0,0,1,0,0,1,1,1,1,1,1,1,1,1]
y_true =[0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1]
print(log_loss(y_true, y_pred))
```

Es importante recalcar que la función log_loss trabaja únicamente con clases representadas numéricamente. Por ello, en el caso del ejemplo anterior cuyas clases se representan mediante las letras A y B, sería necesario una conversión numérica previa.

Valor-F o Media-F (F1-Score)

Documentación oficial:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score

El Valor-F es la media ponderada de la precisión y la exhaustividad (recall). Dicho valor varía entre 0 y 1, siendo 1 el valor ideal y 0 el caso opuesto. El Valor-F puede interpretarse como una media armónica de la precisión y la exhaustividad. La contribución relativa de la precisión y la exhaustividad al Valor-F son iguales.

$$F - score = \frac{2}{\frac{1}{Precisin} + \frac{1}{Exhaustividad}} = \frac{TP}{TP + \frac{FP+FN}{2}}$$

El siguiente fragmento de código muestra cómo hacer uso de esta métrica en Python y sklearn.

```
from sklearn.metrics import f1_score
y_pred=['A','A','A','A','A','A','A','A','A','B','A','A','B','B','B','B','B','B','B','B']
y_true=['A','A','A','A','A','A','A','A','A','B','B','B','B','B','B','B','B','B','B','B']

print(f1_score(y_true, y_pred, average="weighted"))
```

Entre los atributos de la función f1_score merece especial mención average, cuyo valor repercute directamente en el cálculo del Valor-F. Veamos las alternativas que ofrece dicho atributo:

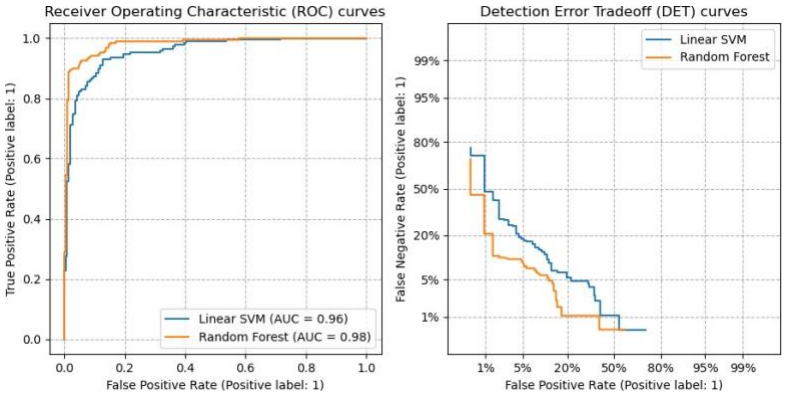
- None: se devuelven las puntuaciones de cada clase. En caso contrario, el tipo de promedio viene determinado por alguna de las siguientes opciones:
  - binary: sólo informa de los resultados de la clase especificada por el atributo pos_label. Esto es aplicable sólo si los objetivos (y_true, y_pred) son binarios.
  - micro: calcula las métricas globalmente contando el total de verdaderos positivos, falsos negativos y falsos positivos.
  - macro: calcula las métricas de cada etiqueta y encuentra su media no ponderada. Esto no tiene en cuenta el desequilibrio de las etiquetas.
  - weighted: calcula las métricas para cada etiqueta y encuentra su media ponderada por soporte (el número de instancias verdaderas para cada etiqueta). Esto altera la "macro" para tener en cuenta el desequilibrio de etiquetas; puede dar lugar a una puntuación F que no esté entre la precisión y la recuperación.
  - samples: calcula las métricas para cada instancia, y encuentra su promedio (sólo tiene sentido para la clasificación multietiqueta donde esto difiere de accuracy_score).

Curvas ROC

Una curva ROC (curva de característica operativa del recepto) es un gráfico que muestra el rendimiento de un modelo en base a la tasa de verdaderos positivos, y la tasa de falsos positivos. La forma de calcular dichas tasas es la siguiente:

- Tasa de verdaderos positivos (TPR) = VP / VP + FN (misma fórmula de la Exhaustividad)
- Tasa de falsos positivos (FPR) = FP / FP + VN

No solo es un tipo de gráfico con el que se puede apreciar a simple vista el rendimiento de un modelo, sino también la comparación entre modelos. Por ejemplo, modelos de aprendizaje y clasificación diferentes aplicados sobre un mismo Dataset, como muestra la siguiente figura.



Cuanto más cerca esté la curva al área de verdaderos positivos (cuanto más alta sea) mejor clasificará el modelo los datos de entrada.

AUC: área bajo la curva ROC

Documentación oficial:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve

El área bajo la curva ROC toma un valor entre 0 y 1, en donde 1 representa una clasificación perfecta por parte del modelo. En tal caso, hay que prestar especial atención a que dicho resultado no sea consecuencia de un sobreajuste del modelo (overfitting). El sobreajuste de un modelo se produce cuando éste se ajusta casi a la perfección a los ejemplos del conjunto de entrenamiento (tiene capacidad para clasificarlos correctamente), pero no tiene capacidad para clasificar la mayoría de nuevos ejemplos (de los que no ha aprendido). En otras palabras, la capacidad de generalización del modelo es baja.

A medida que la curva ROC se acerque al área de verdaderos positivos, el área bajo la curva aumentará. El valor AUC de área representa la probabilidad de que el modelo clasifique correctamente un modelo de entrada. Por ejemplo, si AUC = 0.7, significa que el modelo ofrece un 70% de probabilidad de clasificar correctamente una muestra de entrada. Si queremos ver la curva ROC de una determinada predicción podemos utilizar la función RocCurveDisplay.

```
#Curva ROC y área AUC
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics
y = y_test
pred = y_pred_1

#calculo el ratio de verdaderos positivos y falsos positivos.
fpr, tpr, _ = metrics.roc_curve(y, pred)

#con estos valores obtengo el valor para la AUC
roc_auc = metrics.auc(fpr, tpr)

#genero el gráfico
display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc,
                                  estimator_name='nombre clasificador')
display.plot()

plt.show()
```

Si queremos mostrar varios clasificadores en un mismo gráfico podemos hacerlo de la siguiente forma:

```
#Curva ROC y área AUC
import matplotlib.pyplot as plt
import numpy as np
from sklearn import metrics

#primer clasificador
y = y_test
pred = y_pred_1
fpr, tpr, _ = metrics.roc_curve(y, pred)
roc_auc = round(metrics.auc(fpr, tpr),2)
plt.plot(fpr,tpr,label="SVC(C= 0.1, gamma= 1, kernel= 'linear') ="+str(roc_auc))

#segundo clasificador
y = y_test
```

```
pred = y_pred_2
fpr, tpr, _ = metrics.roc_curve(y, pred)
roc_auc = round(metrics.auc(fpr, tpr),2)
plt.plot(fpr,tpr,label="SVC(kernel='rbf', gamma=0.5, C=0.1) ="+str(roc_auc))

#tercer clasificador
y = y_test
pred = y_pred_3
fpr, tpr, _ = metrics.roc_curve(y, pred)
roc_auc = round(metrics.auc(fpr, tpr),2)
plt.plot(fpr,tpr,label="LGBMClassifier() ="+str(roc_auc))

plt.legend()
```

Informe global con Sklearn

Mediante la métrica classification_report de sklearn existe la posibilidad de elaborar un informe con información relevante, sin la necesidad de cálculo de métricas independientes. Con este informe podríamos obtener una primera idea general de la efectividad de nuestro modelo.

```
from sklearn.metrics import classification_report
y_pred=['A','A','A','A','A','A','A','A','B','A','A','B','B','B','B','B','B','B','B']
y_true=['A','A','A','A','A','A','A','A','A','B','B','B','B','B','B','B','B','B','B']
print(classification_report(y_true, y_pred, labels=['A', 'B']))
```

La salida obtenida será:

```
              precision    recall  f1-score   support

           A       0.80      0.89      0.84         9
           B       0.90      0.82      0.86        11

    accuracy                           0.85        20
   macro avg       0.85      0.85      0.85        20
weighted avg       0.86      0.85      0.85        20
```

Los promedios indicados incluyen la macro media (que promedia la media no ponderada por etiqueta), la media ponderada (que promedia la media ponderada por soporte por etiqueta) y la media de la muestra (sólo para la clasificación multietiqueta). El micromedio (que promedia el total de verdaderos positivos, falsos negativos y falsos positivos) sólo se muestra para la clasificación multietiqueta o multiclase con un subconjunto de clases, porque de lo contrario corresponde a la exactitud y sería la misma para todas las métricas. Por último, support representa el número de ocurrencias de cada clase en y_true.

Una función que podemos definir para mostrar los diferentes valores obtenidos para la clasificación utilizando un clasificador concreto puede ser:

```
#También podemos utilizar una función como la siguiente
def print_score(clf, X_train, y_train, X_test, y_test):
 pred = clf.predict(X_test)
 clf_report = pd.DataFrame(classification_report(y_test,
                                                 pred,
                                                 output_dict=True))
 print("Test Result:\n================================================")
```

```
print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
print("_____")
print(f"CLASSIFICATION REPORT:\n{clf_report}")
print("_____")
print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
print('\n')
```

Podemos utilizar esta función por ejemplo si queremos obtener las métricas de un determinado modelo. En el siguiente código se muestra cómo se llamaría para evaluar la clasificación de flores del dataset Iris.

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

#cargo dataset iris
iris = datasets.load_iris()
X = iris.data
y = iris.target

#creo conjunto de entrenamiento y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify = y)

#normalizo
sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)

# Training a SVM classifier using SVC class
svm = SVC(kernel= 'linear', random_state=1, C=0.1)
svm.fit(X_train, y_train)

print_score(svm, X_train, y_train, X_test, y_test, iris.target_names)
```

Un resultado posible es el siguiente:

```
Test Result:
================================================
Accuracy Score: 97.78%

_____
CLASSIFICATION REPORT:
                0          1          2  accuracy  macro avg  weighted avg
precision     1.0   0.937500   1.000000  0.977778   0.979167      0.979167
recall        1.0   1.000000   0.933333  0.977778   0.977778      0.977778
f1-score      1.0   0.967742   0.965517  0.977778   0.977753      0.977753
support      15.0  15.000000  15.000000  0.977778  45.000000     45.000000
_____
Confusion Matrix:
 [[15  0  0]
 [ 0 15  0]
 [ 0  1 14]]
```

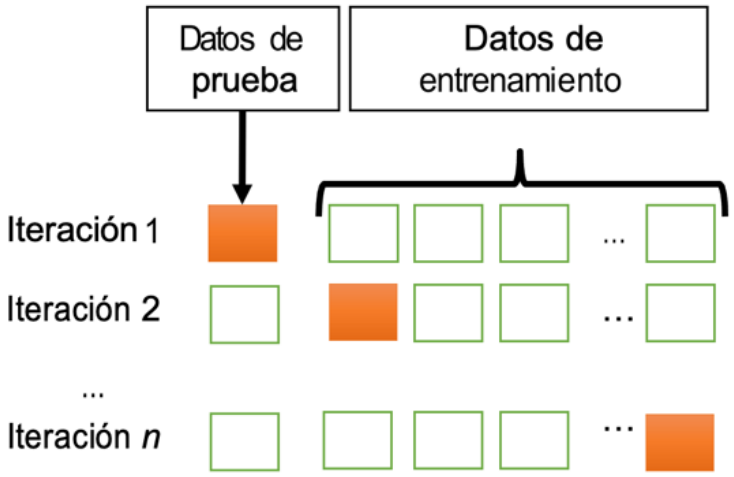
1.2. Validación cruzada

A la hora de validar un modelo es muy habitual emplear una validación cruzada y, sobre ésta, aplicar las diferentes métricas que hemos nombrado anteriormente y que veremos en detalle en las próximas secciones.

En primer lugar, será necesario dividir en conjunto de datos inicial o Dataset en dos subconjuntos: conjunto de datos de entrenamiento (aprendizaje) y conjunto de datos para las pruebas (clasificación). En cuanto al porcentaje de ejemplos que corresponde a cada uno del conjunto inicial hay diversidad de opiniones, pero en la mayoría de los casos, los autores optan por un 70% de ejemplos para el entrenamiento y un 30 % para las pruebas, u 80%-20%.

Los algoritmos de aprendizaje aplicados sobre el conjunto de entrenamiento generan como salida un modelo con capacidad para clasificar los ejemplos del conjunto de pruebas. La evaluación del modelo se lleva a cabo en esta segunda fase, con la aplicación de métricas que determinan la exactitud, precisión o sensibilidad entre otros factores.

La proporción de ejemplos para el entrenamiento y las pruebas es importante, pero también lo es cuáles de ellos son seleccionados. Para evaluar correctamente un modelo es necesario un banco de pruebas con varias iteraciones. En cada una de esas iteraciones se hace una selección diferente de ejemplos para los conjuntos de entrenamiento y pruebas, y se aplican las métricas para obtener los marcadores. Los marcadores finales pueden obtenerse mediante la media aritmética de los obtenidos en cada una de las iteraciones.



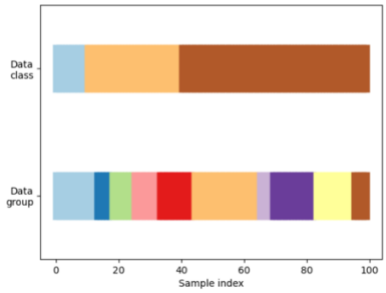
La validación cruzada se preocupa por la selección de ejemplos en cada una de estas iteraciones, y hay diversas variantes (se verán más adelante). Una de ellas, de las más sencillas, es elegir al azar en cada iteración los ejemplos que forman parte de los conjuntos de entrenamiento y prueba. Sin embargo, en otras ocasiones se opta por la no repetición entre iteraciones. Para ello, se trocea el conjunto de datos en tantas partes como iteraciones o pruebas se pretendan realizar y, en cada iteración, se elige un fragmento diferente para el conjunto de pruebas. La figura anterior se muestra gráficamente un ejemplo de validación cruzada, en donde el conjunto de ejemplos empleado para la clasificación varía en cada iteración y sin repetición.

Validación Cruzada con Python

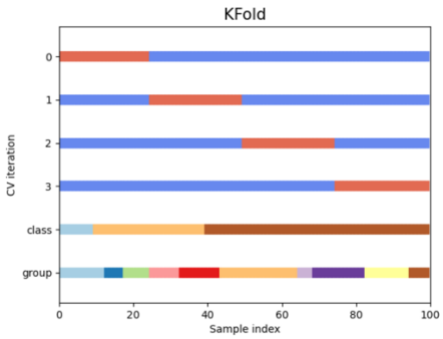
Scikit-learn también incluye herramientas para dividir el conjunto de datos mediante la técnica de validación cruzada.

```
from sklearn.model_selection import cross_val_score
clf = svm.SVC(kernel='linear', C=1, random_state=42)
#cv será el número de subgrupos que se definirán
scores = cross_val_score(clf, X, y, cv=5)

scores
```

Con el fin de ilustrar uno de los problemas de la Validación Cruzada podemos ver un dataset que tiene 100 puntos de datos de entrada generados aleatoriamente, 3 clases divididas de manera desigual en los puntos de datos y 10 "grupos" divididos de manera uniforme en los puntos de datos. Los grupos corresponden por ejemplo al valor de una variable, en este caso habrá diez valores diferentes

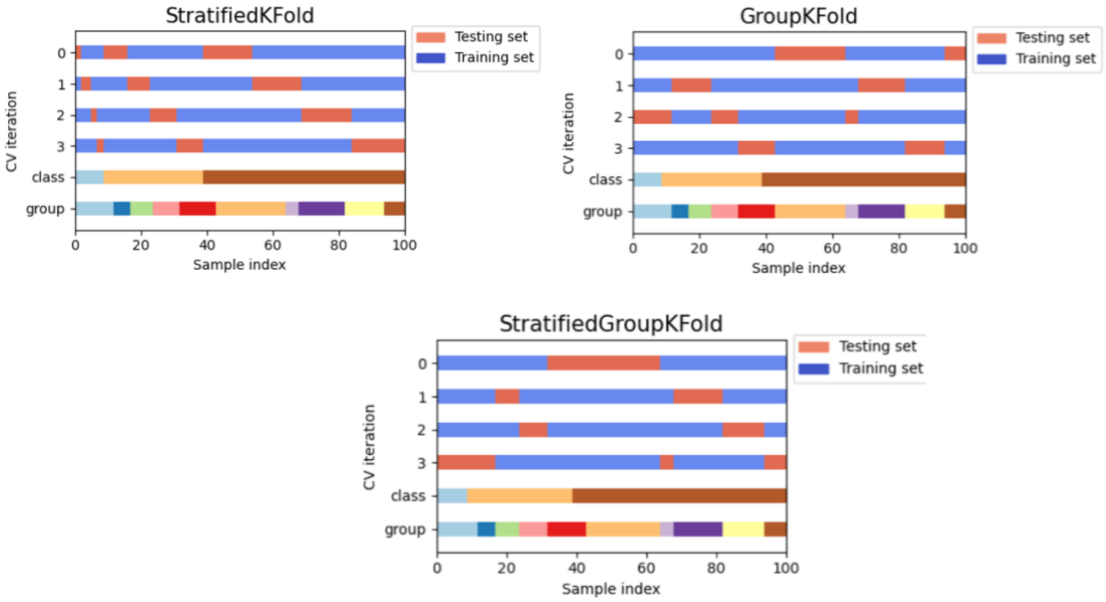


Para la ilustración realizaremos una Validación Cruzada con 4 iteraciones. En cada iteración, se visualizarán los índices elegidos para entrenar (azul) y para testear (rojo).



Como se puede ver, de forma predeterminada con KFold, el iterador de validación cruzada no tiene en cuenta ni la clase ni el grupo de puntos de datos. Podemos cambiar esto usando:

- StratifiedKFold para preservar el porcentaje de muestras para cada clase.
- GroupKFold para asegurarse de que el mismo grupo no aparecerá en dos subconjuntos de test diferentes.
- StratifiedGroupKFold para mantener la restricción de GroupKFold al intentar devolver subconjuntos de test estratificados.



En Python podemos utilizar las diferentes configuraciones del iterador de la siguiente forma.

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import KFold

#KFold
print("KFold")
clf = svm.SVC(kernel='linear', C=1, random_state=42)
k_fold = KFold(n_splits=5) #defino un KFold con 5 iteraciones
scores = cross_val_score(clf, X, y, cv=k_fold)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))

#StratifiedKFold
print("StratifiedKFold")
stratified_k_fold = StratifiedKFold(n_splits=5) #defino un StratifiedKFold con 5 iteraciones
scores = cross_val_score(clf, X, y, cv=stratified_k_fold)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))

#GroupKFold
print("GroupKFold")
#si tuvieramos un dataframe se podría obtener de la siguiente forma, en nuestro caso será groups (columna añadida artificialmente)
#groups = X['your_group_name'] # or pass your group another way
groups =  groups # or pass your group another way
group_k_fold = GroupKFold(n_splits=5) #defino un StratifiedKFold con 5 iteraciones
scores = cross_val_score(clf, X, y, cv=group_k_fold, groups= groups)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))
```