Módulo 5.1 Clasificación ¿Qué, para qué y cómo?

Autor:

Por Prof. Alberto Fernández Hilario

Profesor Catedrático de Universidad de Granada. Instituto Andaluz Interuniversitario en Data Science and Computational Intelligence (DasCI)

Breves Instrucciones

Recordatorio: Introducción a NoteBook

El cuaderno de *Jupyter* (Python) es un enfoque que combina bloques de texto (como éste) junto con bloques o celdas de código. La gran ventaja de este tipo de celdas, es su interactividad, ya que pueden ser ejecutadas para comprobar los resultados directamente sobre las mismas.

Muy importante: el orden de las instrucciones (bloques de código) es fundamental, por lo que cada celda de este cuaderno debe ser ejecutada secuencialmente. En caso de omitir alguna, puede que el programa lance un error (se mostrará un bloque salida con un mensaje en inglés de color rojo), así que se deberá comenzar desde el principio en caso de duda. Para hacer este paso más sencillo, se puede acceder al menú "Entorno de Ejecución" y pulsar sobre "Ejecutar anteriores".

¡Ánimo!

Haga clic en el botón "play" en la parte izquierda de cada celda de código. Las líneas que comienzan con un hashtag (#) son comentarios y no afectan a la ejecución del programa.

También puede pinchar sobre cada celda y hacer "*ctrl+enter*" (*cmd+enter* en Mac).

Cuando se ejecute el primero de los bloques, aparecerá el siguiente mensaje:

"*Advertencia: Este cuaderno no lo ha creado Google.*

El creador de este cuaderno es \<autor>@go.ugr.es. Puede que solicite acceso a tus datos almacenados en Google o que lea datos y credenciales de otras sesiones. Revisa el código

fuente antes de ejecutar este cuaderno. Si tienes alguna pregunta, ponte en contacto con el creador de este cuaderno enviando un correo electrónico a \<autor>@go.ugr.es."

No se preocupe, deberá confiar en el contenido del cuaderno (Notebook) y pulsar en "*Ejecutar de todos modos*". Todo el código se ejecuta en un servidor de cálculo externo y no afectará en absoluto a su equipo informático. No se pedirá ningún tipo de información o credencial, y por tanto podrá seguir con el curso de forma segura.

Cada vez que ejecute un bloque, verá la salida justo debajo del mismo. La información suele ser siempre la relativa a la última instrucción, junto con todos los `print()` (orden para imprimir) que haya en el código.

ÍNDICE

En este *notebook*:

1. Se presentará la tarea de Clasificación en Machine Learning.
2. Se describirá cómo medir la calidad de los modelos de clasificación, y cuáles son las métricas más importantes.
3. Se mostrará un ejemplo sobre cómo aplicar el proceso de aprendizaje y validación sobre un conjunto de datos.

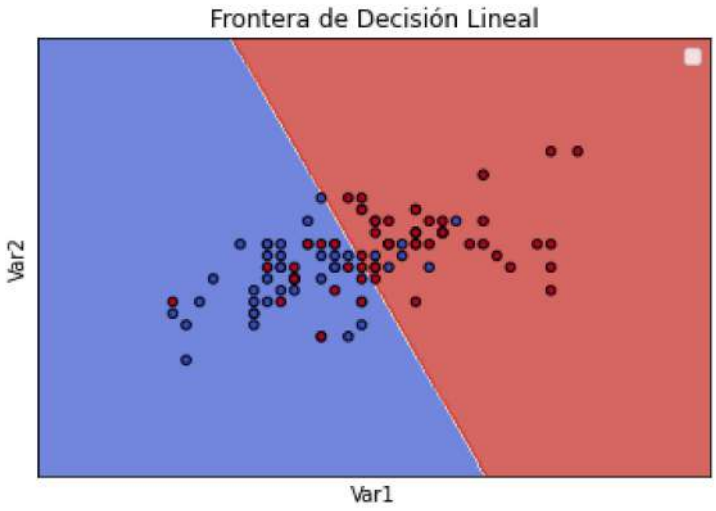
Contenidos:

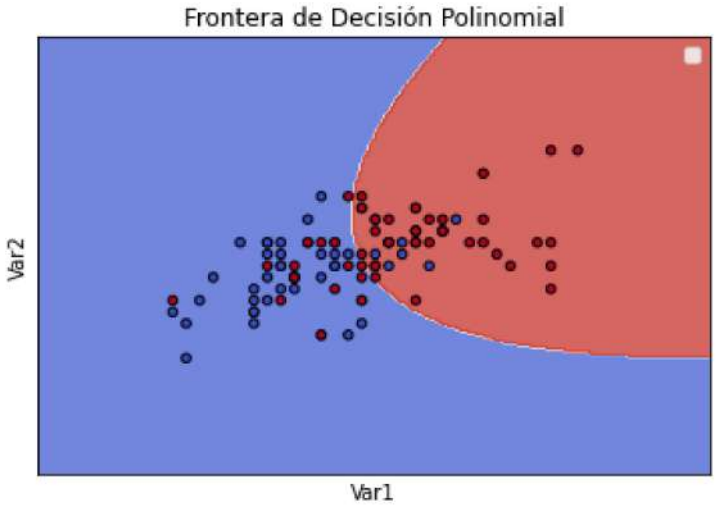
1. Introducción a clasificación
2. Midiendo la calidad de los modelos de clasificación en Machine Learning
3. Ejemplo completo sobre un caso de estudio
4. Referencias bibliográficas

1. INTRODUCCIÓN A CLASIFICACIÓN

La tarea de aprendizaje en clasificación se centra en crear lo que se conoce como "*discriminante*". Este término se refiere a una función que sea capaz de distinguir entre las clases o etiquetas de salida representadas en el problema. Esta función discriminante utilizará los valores de las variables de entrada que definen el caso de estudio o problema con el que se está trabajando. De este modo se genera una división (por ejemplo una simple línea recta) que parte el conjunto de datos en dos o más secciones, creando zonas que identifican a cada una de las clases.

En la siguiente figura, se observan dos tipos de funciones discriminantes en clasificación, una sencilla de tipo lineal (una recta), y otra más compleja (un función polinómica). Ambas son utilizadas para determinar las zonas del espacio que pertenecen a cada una de las clases de un mismo problema de dos variables.





Es importante destacar que las funciones discriminantes que componen el modelo de clasificación tienen una doble misión. Como tarea principal, se utilizan como herramientas predictivas frente a nuevos ejemplos. Adicionalmente, una ventaja añadida es permitir la extracción de conocimiento subyacente en los datos mediante la propia representación del modelo.

Esto dependerá de la denominada *interpretabilidad* del modelo, es decir, si su representación es simple o cercana a la cognición y semántica humana, y por tanto explica los datos de una manera parecida a como lo haría una persona. Por ejemplo, si se aprende un modelo para distinguir pacientes con o sin una patología concreta a través de un panel genético reducido, tenemos el conocimiento de los principales genes a estudiar para pacientes de riesgo. Esta información permitiría trabajar en inhibidores de dichos genes para buscar un tratamiento útil y efectivo.

Existen diferentes tipos de *funciones discriminantes*, lo que identificará distintos paradigmas o modelos de clasificación. Podemos usar discriminantes lineales como en la figura anterior (una simple función lineal entre las variables de entrada), un conjunto de reglas (un listado de conjunciones atributo-valor), o un entorno o vecindario sobre el conjunto de datos de entrenamiento (similitud entre muestras). También existen sistemas más complejos como las llamadas redes neuronales, o las máquinas de vectores soporte.

Al primer grupo de modelos presentado (lineales y reglas) se los denomina "modelos de caja blanca" o "sistemas transparentes" porque permiten una interpretación directa por parte del usuario, y por tanto suelen preferirse cuando se precisa un alto nivel de confianza en el modelo. El segundo tipo de modelos (redes neuronales) son los conocidos como de "caja negra" y, aunque suelen alcanzar un mayor ratio de acierto, no es trivial explicar el porqué producen una salida concreta.

A lo largo del presente módulo se analizarán algunos de los paradigmas de clasificación más conocidos y representativos de ambos paradigmas.

2. MIDIENDO LA CALIDAD DE LOS MODELOS DE CLASIFICACIÓN EN MACHINE LEARNING

Todas las metodologías o técnicas de Machine Learning necesitan una función de evaluación o métrica para estimar cuantitativamente la capacidad de generalización del modelo, es decir, cómo de bien realizará predicciones sobre nuevos datos.

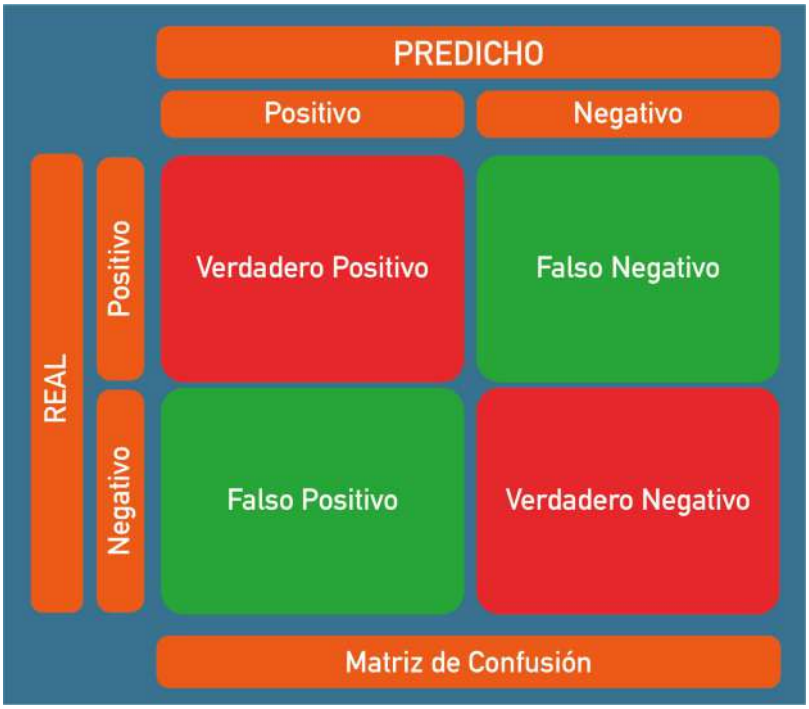
En el contexto de clasificación, la más común de todas estas medidas es conocida como la tasa de aciertos en la predicción (en inglés *accuracy*). Sin embargo, se debe tener en cuenta que hay muchas otras, y que no necesariamente tiene que estar identificado por un simple valor numérico.

Lo fundamental en este caso es que la elección de una medida concreta debe estar guiada por el objetivo final. Este objetivo puede ser simplemente acertar en el máximo número de instancias, o puede que haya alguna clase en particular que sea más interesante de cara al estudio. Por ejemplo, en un sistema automático de diagnóstico, existe más riesgo en dar una falsa predicción negativa (dar el alta a un paciente enfermo), que la contraria (identificar un paciente sano como enfermo).

2.1 Matriz de confusión

Para la definición de cualquiera de las métricas de evaluación en clasificación, se parte de la llamada "*matriz de confusión*". Ésta no es más que una tabla que cruza las predicciones con la clasificación real (en inglés *ground truth*). Así, permite separar los aciertos y fallos por parejas de clases, lo que proporciona un mayor conocimiento sobre qué clases son más difíciles de clasificar, o cuáles son aquéllas que tienden a "mezclarse".

Considerese un problema de clasificación binaria, con una clase positiva (`pos`) y otra negativa (`neg`), así como un clasificador aprendido que se quiere evaluar. Según la siguiente figura, los tipos de predicciones serían:



Es momento de plantear un ejemplo muy sencillo de matriz de confusión. Para ello, se utilizarán dos listas de valores que corresponderán a la salida real, y la predicción del modelo. Ambas se notarán como `y_real` e `y_pred` respectivamente. Al ejectuar el siguiente trozo de código se mostrará la matriz.

In [1]:
```
from sklearn.metrics import confusion_matrix

y_real = ["pos", "neg", "pos", "neg", "neg", "pos"] #etiquetas reales
y_pred = ["pos", "pos", "pos", "neg", "neg", "pos"] #predicciones de ML


confusion_matrix(y_real, y_pred, labels=["pos", "neg"])
```

Out[1]:
```
array([[3, 0],
       [1, 2]])
```

Obsérvese que se ha cometido un único "fallo", en este caso asociado a la segunda clase "*neg*" (negativo).

2.2 Porcentaje de acierto (accuracy)

Si se identifica cada casilla en la matriz anterior como:

- Verdadero Positivo (TP): instancia positiva, predicha positiva.
- Falso Positivo (FP): instancia negativa, predicha positiva.
- Verdadero Negativo (TN): instancia negativa, predicha negativa.
- Falso Negativo (FN): instancia positiva, predicha negativa.

La primera de las métricas mencionadas, la tasa de acierto o *accuracy* se obtiene mediante la siguiente fórmula:

$$accuracy = \frac{TP+TN}{(TP+FP+TN+FN)}$$

que, como se puede observar, indica la proporción de instancias correctamente clasificadas con respecto al total.

A continuación, se chequea qué tal se comporta el modelo obtenido de acuerdo al ejemplo anterior que distingue entre `neg` y `pos`. Para ello, se usa la función `accuracy_score` que toma como parámetros las listas o arrays de valores reales de salida y predichos, respectivamente. El valor que se obtiene está indicado en forma de "ratio" (tanto por uno), si bien es común expresar el *accuracy* en formato porcentual. Para ello, bastará con multiplicar el valor obtenido por 100.0.

In [2]:
```
from sklearn.metrics import accuracy_score

accuracy_score(y_real, y_pred)
```

Out[2]:
```
0.8333333333333334
```

Se ha obtenido un valor bastante alto, casi un 85% de acierto. Esto es debido a únicamente se falló en una de 6 instancias, como se indicó previamente. Para observar el comportamiento de esta métrica, se procede a comprobar las diferencias con otra predicción no tan acertada (`y_pred2`):

In [3]:
```
#Se falla en la mitad de las ocasiones...
y_pred2 = ["neg", "pos", "pos", "neg", "neg", "neg"]
#Por comodidad, se recuerda la lista de valores inicial:
y_real  = ["pos", "neg", "pos", "neg", "neg", "pos"] #etiquetas reales

accuracy_score(y_real, y_pred2)
```

Out[3]:
```
0.5
```

En este caso, la calidad del modelo ha bajado hasta el 50% de acierto. Como se ha podido determinar, la métrica *accuracy* cuantifica el ratio de instancias bien clasificadas con respecto al total, independientemente de la clase a la que pertenecen.

2.3 Métricas individuales por clase

En ocasiones, es mucho más interesante evaluar el acierto de manera independiente para cada una de las clases. Esto es necesario cuando una de dichas clases tiene mayor relevancia, lo que sucede en muchos casos de estudio dentro de la biomedicina, como en la detección de enfermedades neurológicas donde interesa poner el foco en aquéllas que necesiten una atención temprana.

Por ese motivo, existen métricas individuales como las siguientes:

- Sensibilidad (*Sensitivity*, *recall*) o ratio de aciertos positivos (TP_{rate}). Se obtiene como $sensitivity = \frac{TP}{TP+FN}$
- Especificidad (*Specificity*) o ratio de aciertos negativos (TN_{rate}). Se obtiene como $specificity = \frac{TN}{TN+FP}$
- Precisión. Se refiere al ratio de ejemplos positivos correctamente identificados con respecto al total, buscando reducir el número de "falsas alarmas". Se obtienen como $precision = \frac{TP}{TP+FP}$

Puesto que el uso de valores individuales no permite una visión global de la calidad del modelo, resulta muy común resumir o agregar estas medidas, utilizando por ejemplo la métrica conocida como *F-measure*, que agrega *precision* y *recall*:

$$F1 = 2 * \frac{precision * recall}{precision + recall} \quad (1)$$

De acuerdo a lo anterior, el siguiente paso será analizar qué tal rendimiento se alcanza para los dos ejemplos anteriores de predicción de "neg" versus "pos" de acuerdo a las medidas individuales. Para ello, es importante indicar cuál es la clase que se considera como "positiva", que en este caso particular se decide sea "pos"; para lo cual se usa el parámetro `pos_label`.

In [4]:
```
from sklearn.metrics import recall_score, precision_score, f1_score

# Primera predicción
recall = recall_score(y_real, y_pred, pos_label="pos")
precision = precision_score(y_real, y_pred,pos_label="pos")
f1 = f1_score(y_real, y_pred,pos_label="pos")

print("Recall {} | Precision {} | F1 {}".format(recall,precision,f1))

# Segunda predicción
recall = recall_score(y_real, y_pred2,pos_label="pos")
precision = precision_score(y_real, y_pred2,pos_label="pos")
f1 = f1_score(y_real, y_pred2,pos_label="pos")

print("Recall {} | Precision {} | F1 {}".format(recall,precision,f1))
```

```
Recall 1.0 | Precision 0.75 | F1 0.8571428571428571
Recall 0.3333333333333333 | Precision 0.5 | F1 0.4
```

Resulta un poco tedioso tener que preguntar por todas y cada una de las medidas de manera individual. ¿No existe una función para realizar un resumen de las principales de ellas?

La respuesta es por supuesto que sí, y se denomina `classification_report`. En el siguiente bloque se enseña la sintaxis de su uso.

In [5]:
```
from sklearn.metrics import classification_report

print("Analizando modelo #1 (mejor comportamiento)")
print(classification_report(y_real, y_pred))

print("Analizando modelo #2 (peor comportamiento)")
print(classification_report(y_real, y_pred2))
```

```
Analizando modelo #1 (mejor comportamiento)
              precision    recall  f1-score   support

         neg       1.00      0.67      0.80         3
         pos       0.75      1.00      0.86         3

    accuracy                           0.83         6
   macro avg       0.88      0.83      0.83         6
weighted avg       0.88      0.83      0.83         6

Analizando modelo #2 (peor comportamiento)
              precision    recall  f1-score   support

         neg       0.50      0.67      0.57         3
         pos       0.50      0.33      0.40         3

    accuracy                           0.50         6
   macro avg       0.50      0.50      0.49         6
weighted avg       0.50      0.50      0.49         6
```

Para cada una de los dos resúmenes que han sido generados arriba, la información que se observa es la siguiente:

- En primer lugar (filas 1 y 2), para cada una de las clases que contiene el problema ("pos" y "neg" en nuestro caso) se muestra la *precisión, recall, f1* y número de muestras (`support`).
- En segundo lugar, tras un espacio en blanco, se indican las métricas de accuracy:
  - Accuracy global (bajo la columna f1).
  - Accuracy "macro promedio", promediando la media no ponderada por etiqueta de clase.
  - Accuracy "promedio ponderado", promediando la media ponderada de soporte por etiqueta.

Obsérvese que, en la clasificación binaria, el "recall" de la clase positiva se conoce también como "sensibilidad"; mientras que el "recall" de la clase negativa es "especificidad".

2.4 Métricas gráficas: Área bajo la Curva ROC

Por último, una métrica que resume numérica y visualmente la calidad de los modelos es la denominada como Área bajo la Curva ROC (en inglés *Area Under the ROC Curve – AUC*).

Para ello, en lugar de la matriz de confusión, se utiliza el grado de confianza de salida de cada predicción, es decir, cómo de "seguro" está el clasificador de la decisión que ha tomado. Este valor se puede utilizar como "umbral" para forzar al clasificador a realizar la predicción como "clase positiva" siempre que su confianza sea mayor a dicho valor. Por defecto el valor de este umbral se establece como 0.5, es decir, que el clasificador esté al menos un 50% seguro que la clase real sea positiva. Este valor base de 0.5 se puede disminuir, con lo que habría más instancias predichas como positivas (y posiblemente más falsos positivos) o aumentar (el caso totalmente contrario).

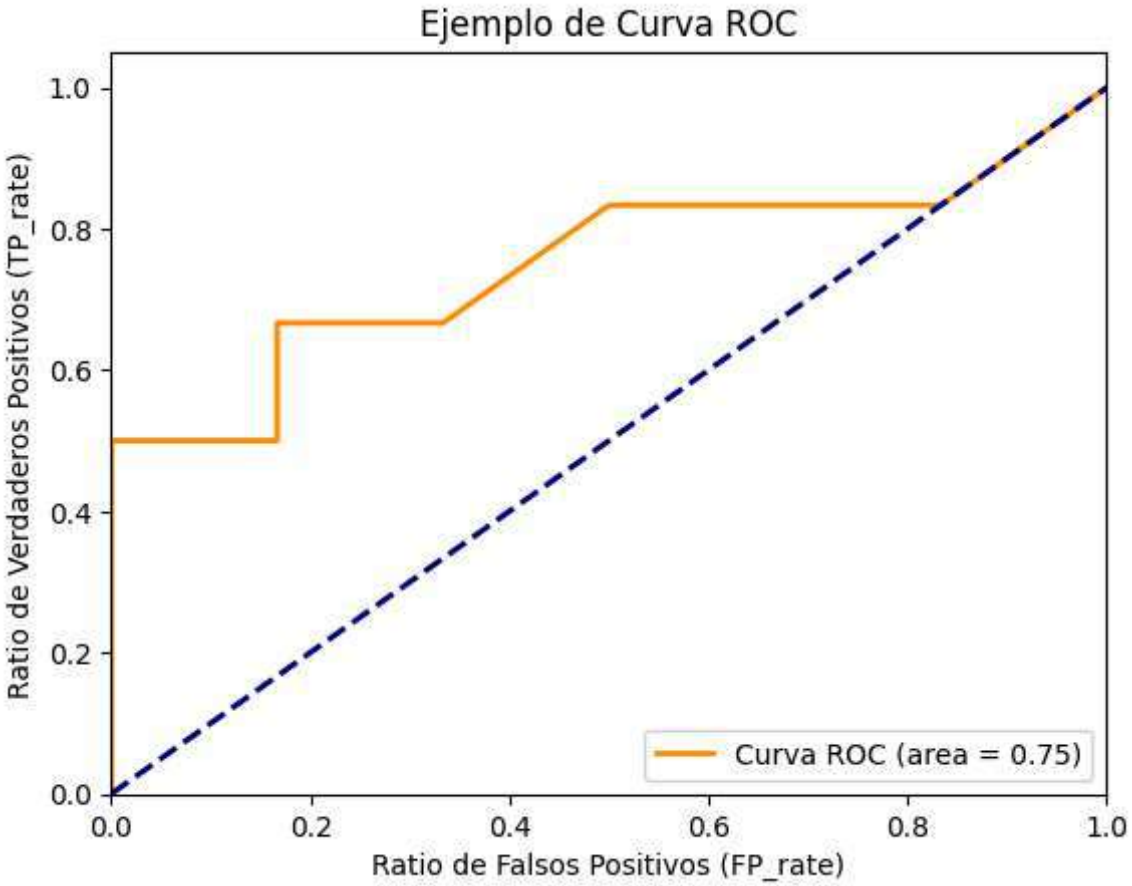
La curva ROC representa cómo varían los valores de sensibilidad TP_{rate} y ratio de falsos positivos (FP_{rate} = $1 - TN_{rate}$) si se modificase el anterior umbral. Lógicamente, si el umbral se fija a 0.0, entonces todas las instancias se clasificarán directamente como positivas

(esquina superior derecha de la gráfica). Si fuese 1.0, entonces solamente cuando haya una certeza absoluta (100% de confianza) las instancias se clasificarían como positivas (corte con el eje de abscisas). Cada posible valor intermedio representa un valor en la curva, e implica un equilibrio diferente entre TP_{rate} y FP_{rate}.

Debido al uso de la probabilidad de la clase positiva, esta aproximación gráfica funciona únicamente para clasificación de tipo binaria (dos clases máximo).

En el siguiente ejemplo se ha utilizado un código relativamente complicado para representar esta gráfica o curva ROC. Por el momento, se deben ignorar los detalles técnicos de programación, puesto que más adelante se obtendrá el mismo resultado mediante una única línea de código.

In [6]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve,roc_auc_score

#Salidas
y_real = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
y_probs = np.array([0.1, 0.2, 0.5, 0.4, 0.7, 0.35, 0.8, 0.65, 1.0, 0.4, 0.75, 0.1])
#para "y_pred" (ahora llamado y_probs) en lugar de etiquetas, utilizamos probabilid

#Cómputo de las métricas de calidad
roc_auc = roc_auc_score(y_real, y_probs)
fpr, tpr, _ = roc_curve(y_real, y_probs)

#Pintamos la figura
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='Curva ROC (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Ratio de Falsos Positivos (FP_rate)')
plt.ylabel('Ratio de Verdaderos Positivos (TP_rate)')
plt.title('Ejemplo de Curva ROC')
plt.legend(loc="lower right")
plt.show()
```



En toda curva ROC, lo ideal es que la curva esté cercana a la esquina superior izquierda (alta sensibilidad (`TP_rate`), baja tasa de falsos positivos). Estar sobre la línea diagonal implica que las predicciones son cercanas a la aleatoriedad.

Además de la propia gráfica, se calcula un valor numérico asociado a la integral (área) de la curva mostrada. Como se puede deducir fácilmente, este cálculo de la métrica AUC no es trivial, pero afortunadamente la mayoría de los Software de Machine Learning lo incorpora. En cualquier caso, si se ignoran los valores de "predicciones de probabilidad" y por tanto se contase con un único punto en la curva ROC (el asociado al umbral por defectdo de 0.5), la fórmula que calcula AUC sería la siguiente:

$$AUC = \frac{1 + TP_{rate} - FP_{rate}}{2} \tag{2}$$

Como curiosidad, es necesario comentar que el AUC en un único punto y la medida accuracy son exactamente iguales cuando se dispone del mismo número de instancias de las clases positiva y negativa.

3. EJEMPLO COMPLETO SOBRE EL CASO DE ESTUDIO DE DETECCIÓN DE MELANOMA MEDIANTE DATOS ÓMICOS.

En esta sección, se presenta una visión general del proceso de creación de modelos para clasificación con algoritmos de Machine Learning.

Para ello, se comienza recuperando el conjunto de datos con información ómica para el caso de estudio sobre melanoma. El resto de secciones están dedicadas al proceso general de

extracción de conocimiento en Ciencia de Datos, a saber, carga del conjunto de datos, particionamiento en entrenamiento y test, aprendizaje de modelos, y validación de la calidad de los mismos.

3.1 Recuperando los datos de la matriz de expresión genética

Como ya se indicó al comienzo de este curso, las alteraciones genómicas resultan de vital importancia en muchos campos de la Bioinformática. De este modo, el procedimiento suele partir de la toma de diferentes muestras de tejido o sangre para obtener información celular. A partir de un procedimiento de secuenciación (técnicas de *Next Generation Sequencing* o NGS), se extraen los valores de la expresión de los diferentes genes que componen el ADN de las muestras.

De esta manera, se suele obtener un conjunto de datos un número de filas equivalentes al número de genes que se haya conseguido secuenciar (variables de entrada), y un número de columnas igual a las muestras que se hayan recogido (instancias). Por último, en estudios clínicos se realizan anotaciones externas sobre cada muestra. Esta serie de variables adicionales resultan útiles y relevantes tanto a nivel predictivo (como variables de salida) como descriptivo (variables de entrada), o incluso permiten una interpretación más sencilla de los resultados obtenidos.

En este apartado del curso, el caso de estudio seleccionado versa sobre predicción de **melanoma cutáneo**. En concreto, se tendrán en cuenta dos clases relevantes a nivel biológico. Por un lado, muestras cuyos genes más expresados pueden considerarse asociados a células, moléculas o proteínas aparentemente 'inhibidoras' del melanoma; se denominarán de manera genérica '*immune*'. Por otro lado, muestras cuyos genes más expresados se pueden asociar con factores de transcripción asociados a microftalmia (*microphthalmia-associated transcription factor*) o simplemente '*MITF*'.

Durante todo el **Módulo de Clasificación** se va a utilizar una modificación del conjunto de datos original. En concreto, se ha realizado un preprocesamiento para simplificar el problema con respecto tanto al número de variables de entrada, como a la información contenida en las instancias. También se debe hacer hincapié en que la estructura del fichero de entrada de datos (donde se almacena la matriz de expresión genética) se ha traspuesto, para seguir la notación estándar de Machine Learning en el que cada fila es una instancia, y cada columna una variable.

3.2 Cargar los datos y análisis exploratorio

Este procedimiento inicial se ha utilizado anteriormente, por lo que solo es necesario repetir lo que ya es conocido (ver Módulos 2 y 4). Al final del trozo de código se imprimen las 5 primeras muestras para confirmar que se ha cargado correctamente.

In [7]:
```
import pandas as pd

#Cargamos los datos ómicos de la matriz de expresión desde un fichero compartido en
gene_exp_inmune = pd.read_csv('https://drive.google.com/uc?id=1PYzEIdmnfjOnBpPDIFBE
#Cargamos la variable clínica correspondiente a las etiquetas "inmune" vs. "MITF-lo
```

```
clinical_info_inmune = pd.read_csv('https://drive.google.com/uc?id=1hHQfcvrFa5Jds-9

X, y = gene_exp_inmune, clinical_info_inmune

#Imprimimos las 5 primeras muestras del conjunto de datos
X.head()
```

Out[7]:

	COL2A1	RXRG	CCL19	SSX1	CST2	PRSS33	CDH2	SCUBE2	TMPRSS1
0	-1.431141	-7.845756	0.665118	-1.409304	-2.537396	-1.676281	1.529957	-0.895042	-0.29877
1	-0.424374	-8.352423	0.386055	-2.846138	-0.685105	0.339787	-3.488043	-0.584982	5.67981
2	11.014251	0.415549	-1.633781	0.315442	-0.662332	-0.498761	0.535811	-0.467456	-2.81873
3	-1.180446	-8.187415	-1.958023	5.061146	-2.603744	-0.666706	0.456460	-4.609624	-1.71316
4	0.816312	-1.189303	4.837235	4.972176	-2.963715	-2.665721	-0.268042	-1.740607	0.01161

5 rows × 50 columns

Siempre que se comienza a trabajar con un nuevo conjunto de datos, conviene comprender la estructura del mismo, es decir, número de instancias, número de variables de entrada, estadísticos descriptivos, distribución de las clases, entre otros. Esta es la etapa de exploración y análisis de datos.

En primer lugar, se observa tanto la estructura del conjunto de datos, como las clases (variable de salida) y su distribución.

In [8]:
```
print("Número de instancias y número de variables:",X.shape)
print("Valores de clase:",pd.unique(y['RNASEQ-CLUSTER_CONSENHIER']))
print("Número de instancias para cada clase:\n",y.value_counts())
```

```
Número de instancias y número de variables: (336, 50)
Valores de clase: ['immune' 'MITF-low']
Número de instancias para cada clase:
 RNASEQ-CLUSTER_CONSENHIER
MITF-low                    168
immune                      168
dtype: int64
```

Se ha comprobado que se está trabajando con un conjunto de datos reducido con respecto al original, en concreto con un número de genes o variables de entrada igual a 50. Asímismo, la proporción de instancias por clase se ha forzado para que se mantenga en el ratio 1:1.

Por último, se pueden observar algunos de los principales estadísticos descriptivos, por ejemplo sobre los 10 primeros genes.

In [9]:
```
#La orden "describe" muestra los principales estadísticos (solo 10 primeras variabl
X[X.columns[0:10]].describe()
```

Out[9]:

	COL2A1	RXRG	CCL19	SSX1	CST2	PRSS33	CDH2	SCU
count	336.000000	336.000000	336.000000	336.000000	336.000000	336.000000	336.000000	336.00(
mean	0.851833	-0.719944	1.136670	0.455719	0.562422	0.243546	0.173262	-0.14
std	3.225967	3.171022	3.112834	3.059409	3.024017	2.848285	2.612762	2.61
min	-2.453544	-9.460513	-6.670597	-3.316233	-2.963715	-3.826060	-6.937990	-6.29
25%	-0.996501	-1.466714	-0.157164	-2.445496	-1.696510	-2.353053	-1.664596	-2.23
50%	-0.366554	0.408615	1.615818	0.019864	-0.429073	0.293924	0.576447	0.03
75%	1.433984	1.352582	3.148308	2.615407	2.031879	2.039151	2.096088	1.90
max	13.077674	3.452661	7.279227	8.045333	12.186303	9.439966	5.199115	5.52

3.3 Crear la partición de los datos en los conjuntos de entrenamiento y test

Por simplicidad, se utilizará una validación tipo "*hold-out*" por defecto. Para más detalles consúltese el **Módulo 3** sobre Aprendizaje Supervisado.

In [10]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

print("Numero de instancias en entrenamiento: {}; y test: {}".format(len(X_train),l
```

```
Numero de instancias en entrenamiento: 252; y test: 84
```

3.4 Entrenamiento y predicciones de un modelo de Machine Learning

Entre las diferentes técnicas disponibles, un clasificador popular y fácil de entender es el conocido como de los "*k vecinos más cercanos*" (en inglés *k-nearest neighbors* o *kNN*).

Esta técnica es una de las estrategias más simples de aprendizaje (de hecho, en realidad no aprende): dada una nueva instancia desconocida, buscar en el conjunto de datos de entrenamiento aquellas instancias cuyas variables sean más parecidas, y asignarle la clase más frecuente entre todos estos "vecinos".

En los próximos apartados del presente módulo, se describirán más detalles sobre el funcionamiento específico de este algoritmo. Por ahora, basta con tener una noción básica y conocer su interfaz (cómo interactuar con él), ya que es común a todos los algoritmos de aprendizaje supervisado implementados en *Scikit-Learn*.

Para generar y aprender nuestro modelo se debe instanciar (crear un objeto de tipo `estimator`) y posteriormente entrenar (llamar al método `fit()`). Se recuerda la necesidad de incluir tanto las instancias de entrada `X` como sus valores de salida `y` para construir el modelo.

Los pasos indicados se muestran en el siguiente trozo de código:

In [11]:
```
from sklearn.neighbors import KNeighborsClassifier # cargamos la función desde la b
import warnings
warnings.filterwarnings("ignore") #ignorar esta linea

knn = KNeighborsClassifier() # instanciamos el modelo (parámetros por defecto)
knn.fit(X_train, y_train) # lo más importante: entrenamos el modelo
print(knn.get_params())
```

```
{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': Non
e, 'n_jobs': None, 'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}
```

La salida mostrada en el bloque anterior es la configuración del clasificador que ha sido creado, con los parámetros que se han asignado de acuerdo a la configuración por defecto de Scikit-Learn: número de vecinos (`n_neighbors`), distancia (`metric`), pesos (`weights`), etc.

3.5 Cálculo de la calidad del modelo generado

Una de las etapas durante el uso de Machine Learning es contrastar si se han alcanzado los resultados de calidad deseados, para lo que se utilizará la partición de test. En Scikit-Learn, para todos los estimadores de tipo supervisados, se cuenta con los siguientes métodos por defecto para realizar la predicción u obtener las métricas de calidad directamente:

- `model.predict()` : dado un modelo que ya está entrenado, predice la clase sobre un nuevo conjunto de datos. Este método acepta un único parámetro: los nuevos datos de test `X_test` (p.ej. `model.predict(X_test)`) y devuelve la clase predicha para instancia del conjunto de datos.
- `model.predict_proba()` : para problemas de clasificación, algunos algoritmos también tienen este método. Devuelve la probabilidad de que cada instancia pertenezca a cada una de las clases. La clase con máxima probabilidad coincidiría con la etiqueta predicha por `model.predict()` . Se utilizará fundamentalmente para generar la curva ROC y la medida AUC.
- `model.score()` : casi todos los clasificadoreas implementan un método denominado "puntuación". Para clasificadores, el método `score()` está asociado al porcentaje de ejemplos bien clasificados (accuracy).

El siguiente bloque de código utiliza el algoritmo *kNN* aprendido en el bloque anterior para generar predicciones sobre el conjunto de test (variable `X_test`).

In [12]:
```
y_pred = knn.predict(X_test) # predicción de cada etiqueta

# imprimimos la etiqueta calculada para los 10 primeros datos de X_test
print(y_pred[:10])
```

```
['MITF-low' 'MITF-low' 'MITF-low' 'MITF-low' 'MITF-low' 'MITF-low'
 'MITF-low' 'MITF-low' 'MITF-low' 'MITF-low']
```

Ahora, la puntuación mediante accuracy puede calcularse utilizando la función correspondiente de la biblioteca de *Scikit-Learn*, tal como se indicó en la Sección 2 del presente NoteBook. Para ello, basta comparar la salida real `y_test` con la predicción realizada por *kNN* `y_pred` .

In [13]:
```
from sklearn.metrics import accuracy_score

acc_score = accuracy_score(y_test, y_pred)
print(acc_score)
```

```
0.6428571428571429
```

El acierto no es muy alto. Podemos ver cuántas muestras han sido mal clasificadas creando una matriz de confusión, del mismo modo que se estudió en la pasada Sección 2.

En esta matriz, las filas representan la clase real, y las columnas representan la clase predicha. Cuantas más instancias estén en la diagonal, mejor puntuación será alcanazada.

In [14]:
```
from sklearn.metrics import confusion_matrix
m = confusion_matrix(y_test, y_pred)
print(m)
```

```
[[43  1]
 [29 11]]
```

3.6 Validación cruzada hecha simple

Es imprescindible realizar una validación del modelo obtenido en la fase de aprendizaje. Entre las alternativas para este proceso, la más apropiada es la validación cruzada de k particiones. Como dicho procedimiento es muy común en Machine Learning, hay funciones para hacerlo de forma flexible y con poco código.

El paquete `sklearn.model_selection` contiene todas las funciones relacionadas con validación de modelos. Aunque en el pasado **Módulo 3** se explicó cómo generar las particiones "a mano", la forma más sencilla es utilizar la función `cross_val_score` que recibe un estimador (clasificador) y un conjunto de datos y hace todo el proceso de forma automática. En otras palabras, devuelve un `array` (un tipo de dato similar a una lista) con las predicciones obtenidas para cada partición.

In [15]:
```
import numpy as np
from sklearn.model_selection import cross_val_score

scores = cross_val_score(knn, X, y) #por defecto 5 particiones

print(scores) #los valores individuales de cada partición
print(np.mean(scores)) #la media de acierto global
```

```
[0.61764706 0.59701493 0.59701493 0.62686567 0.68656716]
0.6250219490781387
```

Por defecto, `cross_val_score` utiliza `StratifiedKFold` para clasificación. Tal como se explicó en el **Modulo 3**, el tipo de particionamiento estratificado asegura que la proporción de ejemplos por clase se respeta en cada partición.

A modo de recordatorio, en el caso de un conjunto de datos binario para clasificación que esté no balanceado, es decir, si se tiene un 90% de instancias de la clase "neg", significaría que en cada partición debería haber un 90% de instancias en la clase "neg".

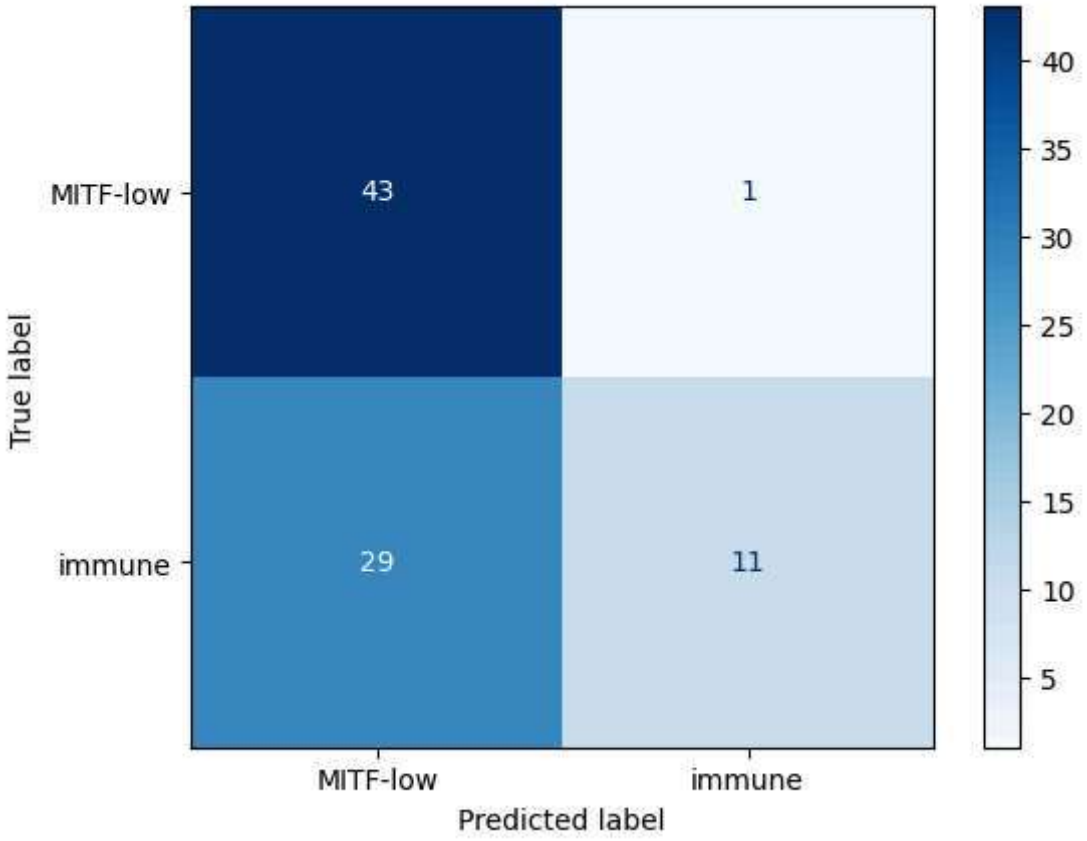
3.7 Visualización de la calidad del modelo

En primer lugar, resulta muy importante chequear la salida del modelo en cada una de las clases mediante la matriz de confusión. Anteriormente ya se mostró este información, pudiendo así contrastar si existe algún tipo de sesgo hacia una de las clases. En este caso, es preferible generar gráficos que sean más cómodos de visualizar en un informe.

Para esta tarea, se utilizará el método `plot_confusion_matrix()` de acuerdo al siguiente ejemplo:

In [16]:
```
from sklearn import metrics
import matplotlib.pyplot as plt

#el parámetro cmap utilizar un mapa de color en azules para ser cómodo a la vista
metrics.ConfusionMatrixDisplay.from_estimator(knn, X_test, y_test,cmap=plt.cm.Blues
```

Out[16]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b7af9234ee0>
```



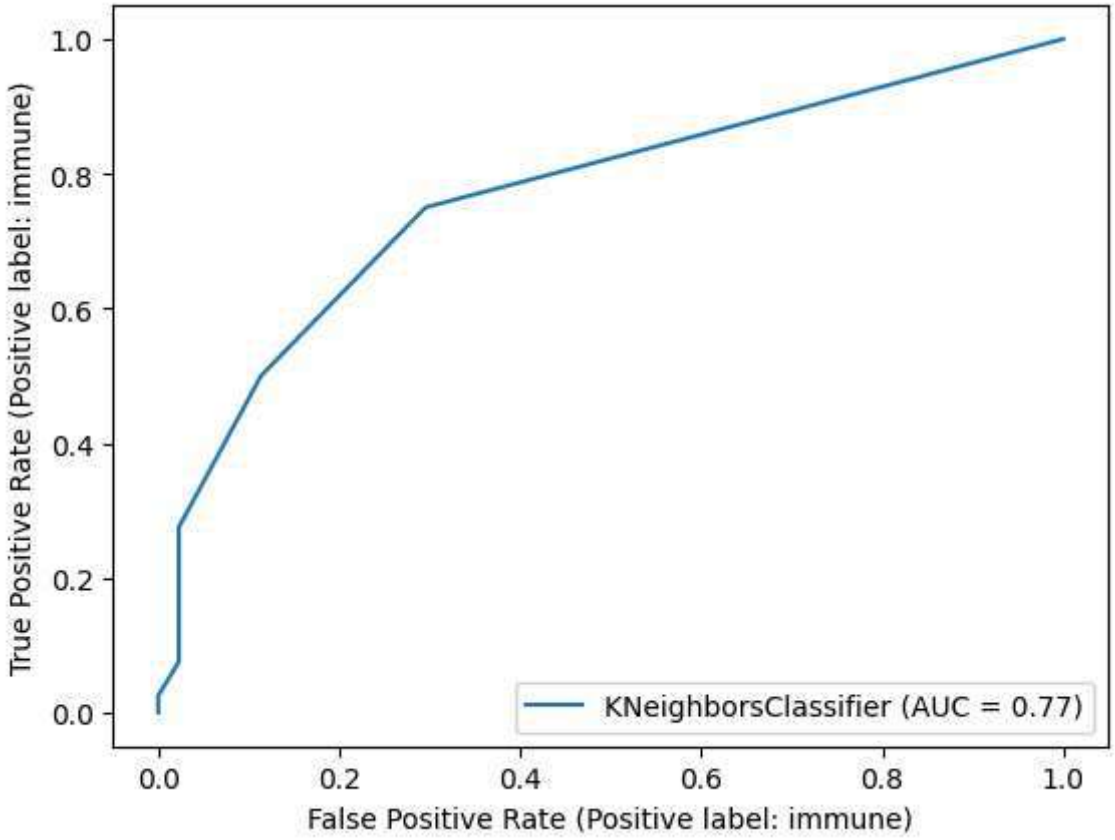
También se pueden calcular el valor AUC mediante el uso de `roc_auc_score`. Para ello, es imprescindible utilizar el método `predict_proba()` del clasificador, ya que se necesitan los valores de probabilidad, y no las clases de salida.

Por último, se puede pinta la curvas ROC con la función `plot_roc_curve`. En este caso el método recibe tres parámetros: el clasificador, el conjunto de instancias, y la lista de valores de salida.

In [17]:
```
from sklearn import metrics

y_prob = knn.predict_proba(X_test)

auc_knn = metrics.roc_auc_score(y_test, y_prob[:,1])
print("El valor de AUC para kNN es", auc_knn)

metrics.RocCurveDisplay.from_estimator(knn, X_test, y_test) #pintamos la curva
```

```
El valor de AUC para kNN es 0.7664772727272727
```

Out[17]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7b7aeb21f5e0>



REFERENCIAS BIBLIOGRÁFICAS

- Han, J., Kamber, M., Pei, J. (2011). Data Mining: Concepts and Techniques. San Francisco, CA, USA: Morgan Kaufmann Publishers. ISBN: 0123814790, 9780123814791
- Witten, I. H., Frank, E., Hall, M. A., Pal, C. J. (2017). Data mining: practical machine learning tools and techniques. Amsterdam; London: Morgan Kaufmann. ISBN: 9780128042915 0128042915
- Scikit-Learn: Metrics and scoring: quantifying the quality of predictions https://scikit-learn.org/stable/modules/model_evaluation.html (visitado el 25 de Junio de 2020).

Referencias adicionales

- Alpaydin, E. (2016). Machine Learning: The New AI. MIT Press. ISBN: 9780262529518
- Towards Data Science: Various ways to evaluate a machine learning model's performance https://towardsdatascience.com/various-ways-to-evaluate-a-machine-learning-models-performance-230449055f15 (visitado el 25 de Junio de 2020).