# **Capítulo 3. Clasificación**

* **Introducción**

Como se mencionó anteriormente, entre los algoritmos de aprendizaje supervisados más comunes se encuentra las regresiones y la clasificación. En el capítulo anterior exploramos un proyecto end-to-end acerca de las casas en Londres y pudimos poner en práctica algoritmos como árboles de decisión, bosque aleatorio y regresión lineal que se explicarán más a fondo en los siguientes capítulos.

Por ahora, con respecto al capítulo 3, profundizaremos en el tema de clasificación y para ello nos centraremos en un set de datos que contiene información médica y las siguientes especificaciones:

* Se tienen 5 tipos de medicamentos, el de tipo A, B, C, X y el de tipo Y.
* Cada tipo de medicamento reacciona mejor dependiendo el tipo de paciente.
* Para definir que tipo de medicamento requiere cada paciente se cuenta con su edad, su género, su nivel de presión sanguínea, su colesterol y su nivel de sodio a potasio en la sangre.
* Los datos contienen 200 ejemplos de pacientes con características diferentes y el tipo de medicamento que deberían de consumir, hay que mencionar que un set de datos de 200 eventos es bastante pequeño en el mundo de ML, aunque en este caso nos servirá perfectamente para practicar en un proyecto.

El propósito general de este proyecto es implementar un modelo de ML que se entrene con estos datos y al ingresar un paciente nuevo, el modelo le pueda asignar un tipo de medicamento, basándose en sus características.

* **Extracción de los datos**

Primero que nada, hay que descargar los datos con Python. Recuerda que estos están disponibles en [GitHub](https://github.com/a2Proyectos/MachineLearning_Data).

def extraer\_datos(root,database):

csv\_path = root + database

return pd.read\_csv(csv\_path)

df = extraer\_datos(DOWNLOAD\_ROOT,MEDICAMENTOS)

df.head()

Al hacerlos les saldrá una data frame como el que se muestra, en donde aparecen características como la edad, el género, la presión sanguínea, el colesterol y el índice de sodio a potasio de cada paciente.

Table

Description automatically generated

En la tabla, la columna de “Drug” corresponde a la variable categórica. Está, según nuestros objetivos, es la variable a predecir cuándo se nos den datos de un paciente nuevo. Recuerden qué pueden verificar qué tipo de variables tienen con el comando data.info()

Text

Description automatically generated

Como podemos observar 4 de las 6 variables son de tipo categórico. Debido a que los algoritmos de Machine Learning trabajan con variables numéricas, tendremos que transformar las cuatro variables categóricas a variables numéricas. Tal como lo hicimos, con los municipios del capítulo pasado.

Antes de hacer modificaciones a nuestro data set, vamos a hacer un análisis gráfico de cada una de las variables para tener una mejor idea de lo que nos vamos a encontrar.

* **Análisis del Set de Datos** 
  + **Edad**

Para empezar, utilizaremos las funciones de pandas *min y max*, las cuales nos regresan el valor mínimo y el valor máximo de nuestros datos respectivamente.

print("Max Age:", df.Age.max())

print("Min Age:", df.Age.min())



Como podemos ver, el rango de edades de nuestros pacientes va desde los 15 hasta los 74 años.

Para ver si la distribución entre las edades, es decir, si tenemos pacientes más jóvenes o adultos, podemos hacer una gráfica de distribución. Esto se puede realizar con función *displot* de la librería *seaborn*.

Esta función es bastante fácil de utilizar, solo ingresamos los datos y el parámetro *kde* (marca la distribución) y los graficamos.

import seaborn as sns

plt.figure(figsize = (9,5))

sns.displot(df.Age,kde=True)

Chart, histogram

Description automatically generated

Como se puede observar, la distribución de edades está bastante equilibrada.

* + **Género**

Para analizar la variable de género, empezaremos por observar el número de mujeres y el número de hombres que hay.

df.Sex.value\_counts()

Text, Word

Description automatically generated with medium confidence

Como podemos ver, la variable “género” está distribuida casi por mitad, aunque hay un poco más de hombres en el set de datos.

* + **Presión Sanguínea**

Ahora vamos a hacer una gráfica de barras con la información de la variable de presión sanguínea, la cual se divide en alta, normal y baja. Para esto usamos la función de *seaborn histplot.*

plt.figure(figsize = (9,5))

sns.histplot(data=df,x="BP",hue="BP")

Con respecto a los parámetros del código, *data* es el set de datos completo, *x* es la variable que queremos graficar y *hue* es la variable con la que queremos asignar la distribución de colores.

Chart, bar chart

Description automatically generated

En este caso tenemos más valores en la categoría “*alta*” y “*baja*” que en la categoría “*normal*”. Se podría inferir que probablemente gran parte de estas personas tenga alguna enfermedad o tenga malos hábitos de salud.

* + **Colesterol**

Para la siguiente variable que es el índice de colesterol vamos a hacer el mismo tipo de gráfica.

plt.figure(figsize = (9,5))

sns.histplot(data=df,x="Cholesterol",hue="Cholesterol")Chart, bar chart

Description automatically generated

En este caso la cantidad de personas que tienen colesterol “*alto*”, es muy parecida con respecto a la cantidad de personas que tienen el colesterol “*normal*”.

* + **Equilibrio Sodio Potasio**

La siguiente variable se llama “*equilibrio sodio potasio*”. Que este índice está equilibrado es fundamental para evitar enfermedades como la hipertensión.

Como esta variable es numérica vamos a usar la función *displot*, para ver una gráfica de su distribución.

plt.figure(figsize = (9,5))

sns.displot(df.Na\_to\_K,kde=True)

Chart, histogram

Description automatically generated

Como podemos ver los valores se concentran en valores que van desde los 10 hasta los 20. Aunque sí hay pacientes alejados del medio.

* + **Medicamentos**

Por último, vamos a hacer una grafica de barras para ver cual es la cantidad de medicamentos en el set de datos.

plt.figure(figsize = (9,5))

sns.histplot(data=df,x="Drug",hue="Drug")

df.Drug.value\_counts()

Chart, bar chart, histogram

Description automatically generated

Text

Description automatically generated

Obeservamos que el medicamento Y es el que aparece con mayor frecuencia en comparación con los demás medicamentos.

* + **Comparación entre Variables**

Con el objetivo de conocer más información con respecto al set de datos, graficamos todas las variables con respecto a la variable a predecir.

Por ejemplo, vamos a graficar la variable edad en comparación con la de medicamentos, esto nos ayudará a saber si hay medicamentos para una edad en específico o no.

Para hacer esto vamos a usar la función de *seaborn swarmplot*, la cual gráfica puntos en la gráfica con un valor x y un valor y.

* + **Medicamentos v.s. Edad**

Para esta gráfica los valores en *x* serán los medicamentos y los valores en *y serán* la edad.

plt.figure(figsize = (9,5))

sns.swarmplot(x = "Drug", y = "Age",data = df)

plt.legend(df.Drug.value\_counts().index)

plt.title("Edad/Medicamento")

Chart, scatter chart

Description automatically generated

Aquí ya nos podemos fijar en algo muy importante, el medicamento A solo se administra en menores de >50 años mientras que el medicamento B solo se administra en mayores de <50 años.

* + **Medicamentos v.s. Género**

Vamos ahora a hacer una gráfica de barras para saber de cada uno de los medicamentos, cuántos se administraron en hombres y cuántos en mujeres.

Para esto primero tenemos que crear un dataframe que cuente estas incidencias.

Utilizaremos la función de *pandas groupby*, la cual nos permite seleccionar un parámetro y ejecutar funciones sobre esa selección.

En este caso tenemos 2 parámetros (*Sex, Drug*) y de esos 2 parámetros queremos contar cuantas incidencias hay, para lo que utilizaremos la función size, esta regresa el número de veces que se repite.

Cuando usamos la función groupby es importante recordar que crea un dataframe, esto significa que hay que asignar nombres a las columnas y un índice.

En este caso las columnas de los parámetros que tomamos del set original ya están definidas, aunque todavía hay que nombrar la columna que contendrá los valores contados.

Para darle formato al índice podemos usar la función *reset* la cual nos dará un índice predeterminado de 0 a la cantidad de datos que tengamos.

df\_Sex\_Drug = df.groupby(["Drug","Sex"]).size().reset\_index(name = "Count")

df\_Sex\_Drug

A picture containing table

Description automatically generated

Una vez que tenemos nuestros datos organizados como los necesitamos, podemos graficar con la función *barplot*, está creará una gráfica de barras donde los parámetros *x* son los *medicamentos* y los parámetros *y* son el número de incidencias(*Count*), el parámetro *hue* que asigna la distribución de colores será nuestra variable de *género*.

plt.figure(figsize = (9,5))

sns.barplot(x = "Drug",y="Count", hue = "Sex",data = df\_Sex\_Drug)

plt.title("Género/Medicamento")

Chart, bar chart

Description automatically generated

En general, los medicamentos no están repartidos en partes iguales entre hombre y mujer, normalmente a través de las “drogas” hay un género en particular que consume más que el otro.

* + **Medicamentos vs Presión Sanguínea**

Para analizar la comparación entre los medicamentos y el nivel de presión sanguínea vamos a repetir exactamente el mismo proceso que con la variable de género.

Con la función groupby contamos cada incidencia y luego las graficamos con la función *barplot*.

df\_BP\_Drug = df.groupby(["Drug","BP"]).size().reset\_index(name = "Count")

Graphical user interface, application, table

Description automatically generated

plt.figure(figsize = (9,5))

sns.barplot(x = "Drug",y="Count", hue = "BP",data = df\_BP\_Drug)

plt.title("Presión Sanguinea/Medicamentos")

Chart, bar chart

Description automatically generated

El medicamento Y lo pueden utilizar personas con cualquier tipo de presión sanguínea, el medicamento A solo es para personas con presión alta al igual que el medicamento B, el medicamento C solo es para personas con la presión baja y el medicamento X no es apto para personas con la presión alta.

* + **Medicamentos vs. Colesterol**

Para el colesterol, también utilizaremos la función *groupby* para contar las incidencias y representarlo con una gráfica de barras.

df\_CH\_Drug = df.groupby(["Drug","Cholesterol"]).size().reset\_index(name = "Count")

df\_CH\_Drug

Table

Description automatically generated

plt.figure(figsize = (9,5))

sns.barplot(x = "Drug",y="Count", hue = "Cholesterol",data = df\_CH\_Drug)

plt.title("Cholesterol -- Drug")

Chart, bar chart

Description automatically generated

Otro indicador importante que se puede observar es que el medicamento C es únicamente utilizado por personas con el colesterol alto.

* + **Medicamentos vs. Sodio/Potasio**

Al ser una variable numérica, haremos el mismo procedimiento que con la variable de edad.

Utilizamos la función *swarmplot* con el parámetro *x* en los medicamentos y el parámetro *y* en el sodio-potasio.

plt.figure(figsize = (9,5))

sns.swarmplot(x = "Drug", y = "Na\_to\_K",data = df)

plt.title("Sodio-Potasio/Medicamentos")

Chart, scatter chart

Description automatically generated

La gente que tiene un índice de equilibrio de sodio y potasio arriba de 15 solo usa el medicamento Y.

* + **Importancia de analizar y conocer tu set de datos**

El conocer el set de datos con el que trabajarás puede ayudarte a mejorar y planear el modelo que predecirá la variable de interés. Con estos conocimientos se pueden crear nuevas variables con características que ayuden a optimizar el rendimiento en los modelos, tal y como lo veremos más adelante.

* + **Ajustar set de datos**

Antes de pasar a nuestro primer modelo de clasificación, vamos a ajustar nuestros datos.

Hay que recordar que los modelos de ML trabajan con números, por lo que vamos a convertir nuestras variables de clase (éenero, presión sanguínea, colesterol, sodio-potasio y medicamentos) a variables numéricas.

Esto lo haremos con la función de *scikit-learn label encoder,* la cual asigna un valor numérico a las opciones de cada variable. Por ejemplo, si en la variable género tenemos hombre y mujer, estos pasaran a ser 0 y 1.

Importamos y definimos nuestra función:

from sklearn.preprocessing import LabelEncoder

def label\_encoder(datos\_categoria):

le = LabelEncoder()

df[datos\_categoria] = le.fit\_transform(df[datos\_categoria])

Una vez creada la función que transforma nuestros datos, seleccionamos las variables que queremos transformar y les aplicamos la función.

variables = ["Sex","BP","Cholesterol","Na\_to\_K","Drug"]

for l in variables:

label\_encoder(l)

df.head()

A screenshot of a computer

Description automatically generated with medium confidence

Una vez teniendo nuestro data set solo con variables numéricas, pasamos a crear nuestro set de prueba y nuestro set de entrenamiento.

Para hacer esto utilizaremos la función de scikit-learn que vimos el capitulo pasado *train\_test\_split.*

Antes de dividir nuestro set de datos vamos a separar nuestros predictores “x” de la variable a predecir, que en este caso es la de medicamentos “y”

x = df.drop(["Drug"],axis=1)

y = df.Drug.

Ahora si procedemos a dividir los datos, recordando que los parámetros para esta función son:

* Nuestros set de datos que en este caso son *x, y*.
* Test\_size es el tamaño en porcentaje que le queremos dar a nuestro set de prueba.
* Random\_state es el método de números aleatorios que queremos utilizar.
* *Shuffle*, es un parámetro booleano, con el cual indicamos si queremos una partición aleatoria (True) o no aleatoria (False).

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 = 42, shuffle = True)

Listo, una vez tenemos nuestros datos divididos en 20% set de prueba y 80% set de entrenamiento, podemos proseguir a probar los modelos de clasificación.

## **Modelo De Clasificación Binario**

Para poner en práctica el modelo binario, escogeremos que el modelo solo identifique uno de nuestros medicamentos. El modelo solo será capaz de identificar 2 clases: es ese tipo de medicamento o no es ese tipo de medicamento el que le corresponde.

Para este ejemplo vamos a verificar que el medicamento que se asigne sea el medicamento C. Entonces vamos a tener dos opciones – O es el medicamento Y, o NO ES el medicamento Y. Se me olvidan de los A, B, D, X, Y

Al convertir las variables de clase a numéricas nos quedamos con que el medicamento Y ahora es el número 3.

Para esto vamos a crear un set de datos que tenga todos los ejemplos de nuestras variables a predecir y, con el medicamento Y y que regrese una lista con True si es correcto o False si no es el medicamento Y.

y\_train\_c = (y\_train == 0)

y\_test\_c = (y\_test == 0)

Table

Description automatically generated

En este ejemplo podemos ver que solo el índice 102 en los mostrados es el medicamento Y y todos los demás que nos muestran son otros medicamentos.

* + **Función de Pérdida (SGD)**

Para nuestro primer modelo vamos a usar el conocido SGD que significa Stochastic Gradient Descent o descenso de gradientes estocásticos en español.

El SGD utiliza un sistema de aleatoriedad durante su entrenamiento, es por eso que se llama estocástico y que como muchos modelos estocásticos intenta predecir la realidad con ayuda de la aleatoriedad.

Este modelo funciona con funciones de pérdida y penalizaciones para realizar la clasificación y es ideal para trabajar aprendizaje a gran escala y de escasa máquina de aprendizaje como clasificación de textos.

Para utilizarlo tenemos que importar la clase de scikit-learn *SGDClassifier.*

Tenemos que señalar que semilla de aleatoriedad queremos con el parámetro *random\_state* y creamos el objeto.

from sklearn.linear\_model import SGDClassifier

sgd = SGDClassifier(random\_state=42)

Después de crear nuestro objeto, tenemos que entrenar nuestros datos, para esto vamos a usar la función *fit* y vamos a asignar nuestro set de entrenamiento de predictores y nuestro set de entrenamiento de la variable a predecir adaptada solo al medicamento Y.

sgd.fit(x\_train,y\_train\_y)

Con la función predict Podemos probarlo con algunos valores que se nos ocurran. si el objeto regresa True, entonces lo clasifica como Medicamento Y, si dice False lo clasifica como otro medicamento.

sgd.predict([[47,1,1,0,8]])

Sabemos que una característica importante para que sea Y, es que se tenga una presión sanguínea baja o en este caso 1, por lo que asignamos una edad, género, presión baja y los otros predictores.



Vemos que regresa True, por lo que está clasificado como paciente que necesita el Medicamento Y.

## **Medidas De Desempeño**

Evaluar a un modelo clasificador puede ser un poco más complejo que evaluar a uno de regresión. Es por eso que invertiremos una buena parte de este capítulo en esto, ya que el objetivo es que no quede duda alguna.

### **Medir la Precisión Con Cross-Validation**

Como vimos en el capítulo 2, podemos usar el método **de cross-validation o validación cruzada** para medir el desempeño de nuestro modelo.

Una forma de hacerlo es con la misma función que utilizamos el capítulo pasado *cross\_val\_score.*

Hay que recordar que esta función divide nuestros datos en diferentes partes llamadas folds y válida de forma diferente cada una de esas partes, podemos dividir nuestros datos en un número k, es por eso que también es conocida como **validación K-folds.**

En este caso vamos a dividir nuestro folds en 3.

from sklearn.model\_selection import cross\_val\_score

cross\_val\_score(sgd,x\_train,y\_train\_y,cv=3,scoring="accuracy")

Vamos a repasar los parámetros de esta función uno a uno.

* Primero esta *sgd*, el cual es nuestro modelo definido anteriormente.
* Segundo, nuestros predictores en el set de entrenamiento.
* Tercero nuestra variable a predecir, manipulada para que de True solo cuando se trata del medicamento Y.
* El parámetro *cv* es para definir cuántos folds queremos, en este caso queremos 3.
* El parámetro *scoring* define el tipo de medida que queremos ejecutar, en este caso deseamos ver el nivel de precisión o accuracy.



En este caso tenemos una precisión del 85% al 92% en nuestros folds, parece muy bueno pero la realidad es un poco diferente.

La precisión por sí sola no es muy buena para evaluar modelos clasificadores.

* Por Ejemplo: Se tiene un set de datos con 100 personas, de estas personas se sabe que 90 no necesitan la medicina Y, y 10 necesitan la medicina Y. El modelo dijo que 99 personas no necesitaban la medicina Y, y 1 si la necesitaba. Aunque sí clasificó de manera correcta los 90 que no necesitaban la medicina Y, solo acertó en 1 persona que si la necesitaba

Si este es el caso, nuestro modelo realmente no sirve para saber qué paciente necesita el medicamento Y, solo sirve para saber cuál no lo necesita.

### **Matriz De Confusión**

Una mejor manera de evaluar el performance de un modelo clasificador es con la llamada matriz de confusión. La idea es contar el número de veces que un ejemplo A es clasificado como un ejemplo B.

Para crear una matriz de confusión primero se debe tener un conjunto de predicciones para después compararlas con el objetivo en específico. Para esto vamos a utilizar nuestro set de entrenamiento de nuevo, ya que como sabemos, lo más recomendable es dejar el set de prueba hasta el final del proyecto.

Para crear nuestra matriz de confusión primero vamos a usar la función de scikit-learn *cross\_val\_predict.*

Como la función que utilizamos antes *cross\_val\_score*, esta nueva función trabaja con folds, solo que en vez de regresar valores que evalúan el desempeño, regresa las predicciones hechas por el modelo

from sklearn.model\_selection import cross\_val\_predict

y\_train\_pred = cross\_val\_predict(sgd,x\_train,y\_train\_y,cv=3)

Como podemos observar los parámetros son exactamente los mismos a excepción de el de score, ya que esta función no trabaja con medidas de desempeño.

Al ejecutar nuestra función esta nos regresa un conjunto de predicciones de la siguiente manera.

Text, table

Description automatically generated with medium confidence

Una vez tenemos las predicciones hechas podemos crear nuestra matriz de confusión, para esto solo necesitamos importar la función de scikit-learn, *confusión\_matrix.*

from sklearn.metrics import confusion\_matrix

confusion\_matrix(y\_train\_y,y\_train\_pred)

Como podemos observar, los parámetros que se necesitan son nuestro set de datos con las instancias del medicamento Y, y nuestro set de datos con las predicciones.



Y esa es nuestra matriz de confusión, para entender cómo funciona pongamos atención en la siguiente imagen.

Table

Description automatically generated

Como podemos ver del lado izquierdo de la matriz están los negativos por lo que, en nuestra matriz, 140 de los casos que no son medicamento Y fueron clasificados como otro medicamento mientras que 10 casos que no eran medicamento Y fueron clasificados como este.

Del lado derecho tenemos los positivos, por lo que en 9 de los casos que tenían que ser medicamento Y nuestro modelo lo marcó como otro medicamento, y solo en una ocasión lo clasificó como medicamento Y.

Esto nos habla de que hasta el momento el modelo se está desempeñando mal a la hora de tener que clasificar cuando si es el medicamento Y.

Una matriz que representa 100% de las clasificaciones correctas se vería de la siguiente manera:

Calendar

Description automatically generated with medium confidence

### **Precisión y Recall**

La matriz de confusión puede darte mucha información importante y se puede complementar con la medida de precisión que se saca como en la fórmula.



Una forma de tener una precisión del 100% sería tener la parte de los positivos sin errores, aunque esto sería algo trivial ya que se ignorará toda la parte de los negativos.

Para resolver esto tenemos otra métrica que se llama recall, también conocida como la sensibilidad o la proporción de True Positives: esto es la proporción de ocasiones positivas que han sido correctamente reconocidas por el clasificador.



Para medir la precisión y el recall de nuestro modelo, simplemente tenemos que importar las funciones de scikit-learn y utilizar los mismos parámetros que nuestra matriz de confusión.

from sklearn.metrics import precision\_score, recall\_score

p = precision\_score(y\_train\_y,y\_train\_pred)

r = recall\_score(y\_train\_y,y\_train\_pred)

p,r



Como podemos ver nuestro modelo ya no parece tan prometedor como cuando solo medimos la precisión con **k folds.**

De tener un 90% de efectividad pasamos a tener valores de 10%.

Probablemente la razón de que esta efectividad sea tan baja es que el modelo SGD trabaja con cantidades de datos bastante grandes, por lo que sería buena idea cambiar nuestro modelo de clasificación.

Vamos a probar otro modelo de clasificación bastante conocido, que se llama **Random Forest.** Este como vimos el capítulo pasado trabaja con múltiples árboles de decisión.

Vamos a profundizar acerca de este modelo y su funcionalidad en el capítulo 7, pero vamos a experimentar cómo se ajusta a los datos con este clasificador.

Para esto vamos a importarlo de scikit-learn como *RandomForestClassifier.*

Después repetimos los mismos pasos que con el SGD para crear nuestras predicciones.

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(random\_state = 42)

rfc.fit(x\_train,y\_train\_y)

y\_train\_pred = cross\_val\_predict(rfc,x\_train,y\_train\_y,cv=3)

Ahora podemos calcular nuestra matriz de confusión de nuevo.

confusion\_matrix(y\_train\_y,y\_train\_pred)



Como podemos observar ahora en la parte de los true positives tenemos un 100% de efectividad.

Vamos a calcular nuestra precisión y recall.

p = precision\_score(y\_train\_y,y\_train\_pred)

r = recall\_score(y\_train\_y,y\_train\_pred)

p,r



Podemos observar que nuestro recall sigue fallando, aunque la precisión es mejor.

Para combinar la precisión y el recall, existe una métrica que se llama F1, es bastante útil especialmente si queremos comparar 2 clasificadores. F1 es una media armónica de la precisión y el recall. Esto porque la media normal evalúa cada valor de igual manera. Por lo que valores pequeños pesan mucho más en la ecuación.

A picture containing diagram

Description automatically generated

La ventaja de esto es que solo obtendrás un valor F1 alto si precisión y recall son altos. Para calcularlo tenemos que importar la función *f1\_score.*

from sklearn.metrics import f1\_score

Tenemos que ingresar los mismos parámetros que con la matriz de confusión, la precisión y el recall.

Vamos a calcular el f1 de ambos clasificadores.

SGD:



Random Forest:



Como podemos ver el *f1* de Random Forest es 6 veces mayor que el de SGD por lo que de momento seguiremos utilizando este modelo.

La métrica *f1* tiende a favorecer clasificadores que tienen una precisión y un recall similar.

En algunas ocasiones eso no es lo que buscamos, ya que dependiendo del contexto se priorizará la medida de recall y la medida de precisión.

Por ejemplo, si se quiere tener un clasificador que separé imágenes que son aptas para niños, probablemente sea más conveniente que se rechacen buenos videos (recall bajo) y que se conserven las imágenes seguras (precisión alta).

O tal vez quieres detectar intrusos en un área de seguridad, no importará tanto que la precisión sea alta, mientras que tu recall sea casi 100% seguro.

Esto ya que el recall se enfoca también en el lado negativo de la matriz de confusión, por lo que se quiere que toda persona que no sea alguien con autorización (área positiva) esté identificada.

Desafortunadamente la mayoría de las veces un incremento en la precisión reduce el recall y un incremento en el recall reduce la precisión.

A esto se le llama la compensación precisión/recall.

### **Compensación Precisión/Recall**

Para entender esta compensación vamos a volver al clasificador SGD y cómo toma decisiones.

Para cada evento, asigna un valor basado en una función de decisión. Si ese valor es mayor que cierto umbral, este le asigna ese evento a una clase positiva, si es menor le asignará el evento a una clase negativa.

Un ejemplo de esto es la siguiente imagen:

Chart

Description automatically generated

Como podemos ver entre más movemos el umbral, los resultados cambian.

Si movemos el umbral hacia la izquierda, tenemos un recall de 100% ya que todas sus predicciones serán correctas al clasificar todos los verdaderos negativos.

Si movemos el umbral hacia la derecha, tendremos una precisión perfecta con 3 clasificaciones correctas, pero sacrificaremos el recall con varios errores.

Si centramos el umbral, tendremos una buena estimación en ambos, aunque ninguna será perfecta.

Es por eso que es importante revisar el contexto en el que se esta trabajando, para saber donde será más conveniente colocar el umbral.

Scikit-learn no nos deja seleccionar el umbral directamente, pero si nos da acceso a los valores de decisión que usa para hacer sus predicciones

Podemos utilizar el método decision\_function, el cual nos devuelve un valor para cada instancia, y después de ver el valor, seleccionar el umbral que nosotros deseemos.

Ejemplo:

y\_score = sgd.decision\_function([[47,1,1,0,8,0]])

y\_score



En este caso vemos que nuestro umbral tiene un valor de 1706.

Esto significa que a la derecha de ese umbral el modelo clasificará estos valores como modelo C, pero a la izquierda de este umbral, se clasificará como otro medicamento.

Pero qué tal si cambiamos el umbral a 0.

threshold = 0

y\_some\_pred = (y\_score > threshold)

Esta función regresa lo mismo que la función *predict*, por lo que como el umbral esta a la derecha de 1706 debería de regresar True.



Y por ejemplo si ubicamos el umbral en 2000.

threshold = 2000

y\_some\_pred = (y\_score > threshold)



Esto nos confirma que disminuir el umbral disminuye la precisión y viceversa.

Pero la pregunta es ¿cómo decidimos qué umbral utilizar?

Para esto utilizaremos de nuevo la función *cross\_val\_predict*, pero esta vez en lugar de que nos diga que tan preciso es el modelo, vamos a pedirle que nos regrese el umbral de decisión de todos los folds en los que dividimos los datos.

Esto se hace cambiando el parámetro *score* por el parámetro method e indicando *decision\_function* a diferencia de antes que teníamos accuracy.

y\_scores = cross\_val\_predict(sgd,x\_train,y\_train\_y,cv=3,method="decision\_function")

Después utilizaremos la función de scikit-learn *precisión\_recall\_curve*, la cual regresa cada uno de los valores de precisión y recall de todos los umbrales que calculamos.

precisions, recalls, umbrales = precision\_recall\_curve(y\_train\_y,y\_scores)

Es importante crear una variable para cada uno de los parámetros que buscamos ya que el objetivo principal de hacer esto es graficarlo, ya que solo así podremos ver claramente que umbral es mejor.

La función requiere las variables a predecir y el conjunto de umbrales como parámetros.

Y en orden regresa las precisiones, los recalls y por último los umbrales, es por eso que acomodamos así las variables.

Procedemos a graficar la precisión contra los umbrales y los recalls contra los umbrales.

plt.plot(umbrales, precisions[:-1],"b--",label="Precisión")

plt.plot(umbrales, recalls[:-1],"g-",label="Recall")

plt.show()

Chart, line chart

Description automatically generated

Otra forma de graficarlo es directamente precisión contra recall.

plt.plot(precisions[:-1], recalls[:-1],"g-",label="Precisión Vs Recall")

plt.xlabel("Recall")

plt.ylabel("Precisión")

plt.legend()

plt.show()

Chart, line chart

Description automatically generated

Podemos ver que la precisión sube cuando el recall baja y el recall baja cuando la precisión sube.

Supongamos que queremos que nuestro modelo esté más cargado del lado de la precisión, por lo que queremos una precisión de 90%

Para esto podemos usar la función de *numpy argmax*, la cual nos deja buscar un elemento con el valor máximo especificando nuestras condiciones, en este caso, una precisión del 90%.

Para esto solo tenemos que ir a nuestro set de datos con los umbrales, que están en forma de lista, y utilizar nuestra función.

umbral\_90 = umbrales[np.argmax(precisions >= 0.90)]

umbral\_90



El umbral que necesitamos para esta operación es el valor 42147.72.

Por ahora ya que tenemos nuestro umbral seleccionado y un data set con los valores de cada umbral, podemos definir nuestra variable de predicción de esta manera.

y\_train\_90 = (y\_scores >= umbral\_90)

Esto ya que estamos seleccionando las predicciones a partir del umbral seleccionado.

Después de esto podemos verificar cómo quedaría la precisión y el recall de nuestro modelo.



Listo ya tenemos un modelo con 90% de precisión y 60% de recall.

Como pudiste observar es algo fácil tener un modelo con una precisión alta sacrificando el recall, aunque hay que tener en cuenta que un modelo que tiene un recall muy bajo no es muy útil, ni recomendable.

### **La Curva ROC**

La curva conocida como **receiving operating characteristic** o en español característica operativa del receptor, pero comúnmente conocida como **ROC**, es una herramienta bastante utilizada con clasificadores binarios.

Es parecida a la curva anterior de precisión contra recall, pero, la curva ROC gráfica el porcentaje de Verdaderos positivos (TPR o recall) contra el porcentaje de falsos positivos (FPR).

Esta medida se suele graficar con valores de 0 a 1. Para graficar la curva ROC, podemos utilizar la función de scikit roc\_curve. Esta función nos regresa la lista de tpr, fpr y umbrales, justo como lo hicimos con las precisiones los recalls y los umbrales.

from sklearn.metrics import roc\_curve

fpr, tpr, umbrales = roc\_curve(y\_train\_y,y\_scores)

Ahora procedemos a graficar el FPR contra el TPR.

plt.plot(fpr, tpr, label="ROC Curve")

plt.plot([0, 1],[0, 1], 'k--')

plt.xlabel("True Positive Rate")

plt.ylabel("False Positive Rate")

plt.grid()

plt.show()

Chart, scatter chart

Description automatically generated

Aquí podemos observar esa compensación otra vez, entre mayor es el porcentaje de verdaderos positivos, mayor es el porcentaje de falsos positivos, por lo que se dice que el umbral óptimo esta justo en el “codo” donde se crea la curva arriba de la diagonal.

Una forma muy común de comparar modelos de clasificación es observar el área bajo la curva de ROC, también conocida como **ROC AUC, área under the curve.**

Entre más grande sea el área, mejor es el modelo, ya que se puede tener un porcentaje de TPR Y FPR más equilibrado.

Esto en la gráfica se vería como la curva más pegada a la esquina superior izquierda.

Para conocer su valor podemos importar la funcion de scikit roc\_auc\_score e ingresarle los parámetros de nuestra variable a predecir y los valores del set de entrenamiento.

from sklearn.metrics import roc\_auc\_score

roc\_auc\_score(y\_train\_y,y\_scores)



El valor máximo de esta métrica es 1, por lo que 0.89 no es tan malo, pero como se había mencionado antes el objetivo principal de esta medida es la comparación, por lo que vamos a utilizar nuestro modelo de random forest que habíamos definido antes.

Hay que tomar en cuenta el modelo random forest no utiliza el método de decision\_function sino uno llamado predict\_proba el cual regresa un set de datos conteniendo una hilera por evento y una columna por clase, donde cada valor es la probabilidad de que ese evento pertenezca a esa clase.

Casi todos los clasificadores tienen alguna de estas 2 propiedades o las 2.

Por ejemplo, un evento puede tener una probabilidad del 90% de ser medicamento Y, y 10% de ser medicamento C.

En este caso como es un modelo binario, solo tenemos las clases de probabilidad de que si pertenece al medicamento Y o probabilidad de que no pertenezca al medicamento Y.

Vamos a guardar esas probabilidades de la siguiente manera.

y\_forest = cross\_val\_predict(rfc,x\_train,y\_train\_y,cv=3,method="predict\_proba")

Ahora vamos a guardar solo la columna con las probabilidades de que si sea medicamento Y.

y\_scores\_forests = y\_forest[:,1]

Una vez teniendo esto creamos nuestros TPR, FPR y umbrales con la función *roc\_curve.*

fpr\_forest, tpr\_forest, umbral\_forest = roc\_curve(y\_train\_y, y\_scores\_forest)

Ahora podemos graficar nuestras 2 curvas ROC.

plt.plot(fpr, tpr, label=" SGD ROC Curve")

plt.plot(fpr\_forest, tpr\_forest, label=" RF ROC Curve")

plt.plot([0, 1],[0, 1], 'k--')

plt.legend()

plt.ylabel("True Positive Rate")

plt.xlabel("False Positive Rate")

plt.grid()

plt.show()

Chart, line chart

Description automatically generated

Aquí podemos ver que nuestro modelo random forest se ajusta al 100% por lo que es bastante mejor que el modelo SGD para este set de datos

Vamos a calcular el ROC AUC, para verificarlo:

roc\_auc\_score(y\_train\_y,y\_scores\_forest)



Como podemos ver tiene un resultado de 1, por lo que es un modelo bastante acertado.

Ahora sabemos porque es importante saber cómo entrenar clasificadores, elegir la métrica apropiada para estos y saber si el modelo de verdad funciona o no.

## Clasificación Multiclase

Mientras que los clasificadores binarios solo pueden distinguir entre 2 clases. Los clasificadores multiclase pueden distinguir entre 2 o más clases.

Hay algunos modelos de clasificación como el de regresión logística, random forest y bayes que pueden hacer clasificación multiclase automáticamente.

Hay otros como **SGD o SVM support vector machine classifier**, que son estrictamente binarios por naturaleza. Aun así existen algunas técnicas que se pueden emplear a los clasificadores binarios para que hagan tareas multiclase.

Una forma es crear un sistema que pueda clasificar por ejemplo, nuestros 5 medicamentos, donde se entrenaron 5 diferentes clasificadores, y cada uno de estos regresará algún puntaje de cuánto se asemeja al medicamento correspondiente y ya con los puntajes tomar una decisión.

A esta estrategia se le llama uno contra el resto, ya que los clasificadores compiten para ver cual genera el mayor puntaje.

Otra estrategia consiste en entrenar cada clase contra cada clase respectivamente, por ejemplo un clasificador para medicamento Y contra medicamento B, otro para medicamento Y contra medicamento C, otro para medicamento Y contra medicamento X, y así sucesivamente hasta tener un clasificador contra cada comparación posible. A este método se le conoce como 1 contra 1 y consiste en ver cual clase o en este caso medicamento gana más duelos contra los otros medicamentos.

Algunos modelos como SVM no escalan bien con el tamaño del set de entrenamiento por lo que para este tipo de situaciones, es recomendable optar por el 1 contra 1. Ya que será más efectivo a la hora de entrenar con sets de datos pequeños.

Aunque en la mayoría de los casos y con sets de datos bien estructurados, se usará el uno contra todos.

Afortunadamente scikit-learn decide automáticamente si usar la estrategia uno contra todos o 1 contra 1 dependiendo del set de datos.

Vamos a crear un clasificador SVM pero esta vez con todo nuestro set de entrenamiento y los diferentes 5 tipos de medicamentos, no solo con el set de datos para el medicamento Y.

Para esto utilizaremos la clase de *scikit SVC.*

from sklearn.svm import SVC

svm = SVC()

svm.fit(x\_train,y\_train)

svm.predict([[25,0,1,0,167,1]])

En este caso a la función predict le pasamos unos valores que corresponderían a un paciente que tiene que tomar medicamento Y.



Y el modelo lo clasificó correctamente ya que la clase 0, es igual a la clase Y.

Esto lo podemos observar de manera mas detallada, utilizando la función decision\_function ya que esta nos regresara los puntajes que se dieron a cada clase o en este caso medicamento.

some\_scores = svm.decision\_function([[25,0,1,0,167,1]])

some\_scores



Como podemos observar el índice con mayor puntaje es el 0, ya que es el correspondiente al medicamento Y. El que tiene el segundo mejor puntaje es el índice 6 que corresponde al medicamento A.

Si tenemos muchos puntajes, la mejor forma para saber cual es el mejor es usar la función de *numpy argmax*, la cual nos regresara el índice de donde se encuentra el mayor puntaje.

np.argmax(some\_scores)



Si en algún momento realmente deseamos decidir que tipo de estrategia usar para la clasificación multiclase con las clases de scikit, OneVsOneClassifier y OneVsRestClassifier.

Vamos a hacer un ejemplo para decirle a nuestro modelo SVM, que utilice la estrategia 1 contra todos.

from sklearn.multiclass import OneVsRestClassifier

svm = OneVsRestClassifier(SVC())

svm.fit(x\_train,y\_train)

svm.predict([[25,0,1,0,167,1]])



Listo, como puedes ver lo único que hicimos diferente fue ingresar el modelo dentro de la clase de 1 contra el resto, seguimos los mismos pasos y el modelo clasificó correctamente los datos.

Vamos a verificar los puntajes que se asignaron esta vez con esta estrategia.

some\_scores = svm.decision\_function([[25,0,1,0,167,1]])

some\_scores



De nuevo tenemos al índice 0 con el mayor puntaje, y contundentemente ya que las otras clases tienen valores negativos.

## **Análisis de Errores**

Si este fuera un proyecto de ML real, tendríamos que seguir los pasos del capítulo pasado como, explorar más opciones de preparación de datos, intentar usar más modelo para compararlos, hacer un grid search para ver si mejora con el uso de hiperparámetros.

Supongamos que tienes un modelo que te convence lo suficiente y quieres encontrar alguna manera de mejorarlo, una de las opciones es revisar los errores que comete.

Para esto podemos ver la matriz de confusión, pero esta vez de un modelo multiclase.

Vamos a usar la validación cruzada para crear nuestro set de predicciones.

El modelo random forest fue el que mejor se adaptó en el ejemplo binario por lo que lo utilizaremos de nuevo y después utilizaremos la función de matriz de confusión.

y\_train\_pred = cross\_val\_predict(rfc, x\_train, y\_train, cv=3)

conf\_mz = confusion\_matrix(y\_train,y\_train\_pred)

conf\_mz

A picture containing table

Description automatically generated

A simple vista la diagonal se ve bastante bien, recordemos que la diagonal marca los aciertos en la matriz de confusión mientras que los valores fuera de la diagonal son valores falsos.

Por ejemplo, podemos ver que en la clase 3 que en este caso es el medicamento C, 12 si se clasificaron como medicamento C, pero 1 se clasificó como medicamento clase 2 o medicamento A.

Mientras que en la clase 4 o medicamento X, 4 eventos se clasificaron bien mientras que 2 eventos se clasificaron como clase 5 o medicamento B.

En la clase 0 que es el medicamento Y, uno de los medicamentos se clasificó mal, se clasificó como clase 4 o X.

Esta matriz nos ayuda a ver este tipo de errores. Con estos resultados se podría pensar en enfocarse en el falso A y en los falsos B para mejorar el modelo.

Una forma de hacer esto sería recaudando más datos de pacientes con medicamentos A y B para que el modelo aprenda de ellos.

Otra forma de resolver esto sería creando nuevas variables que ayuden al algoritmo a saber cuándo un medicamento corresponde a cierto evento.

Vamos a crear una variable que nos arrojé el valor de 1 cuando el equilibrio sodio potasio es menor a 15 y 0 cuando sea menor a 15, ya que esto ayudará a saber si el medicamento es Y o no es Y.

Recordemos que en las gráficas vimos que si este es mayor a 15 entonces será medicamento Y, y si es menor, es otro medicamento.

df['Na\_to\_K\_Bigger\_Than\_15'] = [1 if i >=15.015 else 0 for i in df.Na\_to\_K]

df.head()

Table

Description automatically generated

Una vez creadas las variables nuevas, corremos todo el modelo nuevo para revisar cual es la matriz de correlación.

A picture containing text

Description automatically generated

Podemos observar que, con esta variable de ayuda, nuestro modelo ahora funciona bastante mejor.

## **Clasificación Multinivel**

Hasta ahora cada instancia ha sido asignada a una sola clase, o nos ha dado un solo resultado, pero ¿qué pasa si queremos que un resultado se asigna a varias clases? Por ejemplo, en este proyecto a cada persona se le asigna un solo tipo de medicamento, pero que tal si se le pudieran asignar varios, que una persona pudiera recibir tanto el medicamento A, el Y y el X.

Por ejemplo, conscientes de los 5 medicamentos disponibles, nos podría arrojar un vector como este: [1,0,0,1,1], diciéndonos que los medicamentos o clases Y, A, X o 0,4,5 son correctos.

A este tipo de vectores binarios se les conoce como sistema de clasificación multinivel.

Vamos a crear un ejemplo rápido solo para ver cómo funciona y para ello utilizaremos el método método de clasificación ***K-neighbors.***

De este método vamos a profundizar más adelante en el curso, ahora sólo lo usaremos para practicar.

Vamos a crear nuestro set de datos que acepte los medicamentos Y y A.

y\_0 = (y\_train == 0)

y\_5 = (y\_train == 5)

y\_multi = np.c\_[y\_0,y\_5]

A picture containing text

Description automatically generated

Ahora tenemos un conjunto de datos que dicen si un evento pertenece tanto al medicamento C como al medicamento A.

Vamos a predecir con nuestro nuevo modelo.

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

knn.fit(x\_train,y\_multi)

knn.predict([[45,0,1,0,89,1]])



En este caso este evento ficticio no pertenece a ninguno de los dos medicamentos.

Para medir el desempeño de este tipo de modelos una medida bastante utilizada es el valor f1, que vimos antes.

Lo que se hace es crear un conjunto de predicciones con validación cruzada y utilizar nuestra función f1\_score.

Utilizamos el parámetro *average*, para decirle que nos devuelva un porcentaje, en este caso con la palabra macro.

y\_train\_pred = cross\_val\_predict(knn, x\_train, y\_multi, cv=3)

f1\_score(y\_multi,y\_train\_pred, average="macro")



En este caso si tenemos muchos más ejemplo en los datos de un evento que de otro, podemos decirle a la función que les de su peso correspondiente con la palabra weighted en average.

y\_train\_pred = cross\_val\_predict(knn, x\_train, y\_multi, cv=3)

f1\_score(y\_multi,y\_train\_pred, average="weighted")



De esta manera como nuestro set de datos tiene muchos más casos con el medicamento Y, el modelo nos da mejores resultados.

## **Clasificación Multisalida**

El último tipo de clasificación que vamos a revisar es la llamada clasificación multisalida o multioutput.

Este tipo de clasificación es una rama de la clasificación multiclase donde cada clase puede tener múltiples valores.

Hasta ahora solo hemos visto True y False o 0 y 1, pero no hemos visto alguna que cada clase pueda tener más de 2 valores.

O también puede regresar algún tipo de objeto.

Por ejemplo, si tenemos un clasificador de imágenes, podemos, para que al clasificar la imagen nos regrese la misma imagen, pero con el nombre de la persona.

Podremos ver algún ejemplo de clasificación multisalida cuando veamos clasificación de imágenes.

Con esto concluimos el capítulo de clasificación. Ahora debes de ser capaz de seleccionar métricas, utilizar umbrales, comparar clasificadores y utilizar estos para diferentes tareas.