Table of Contents

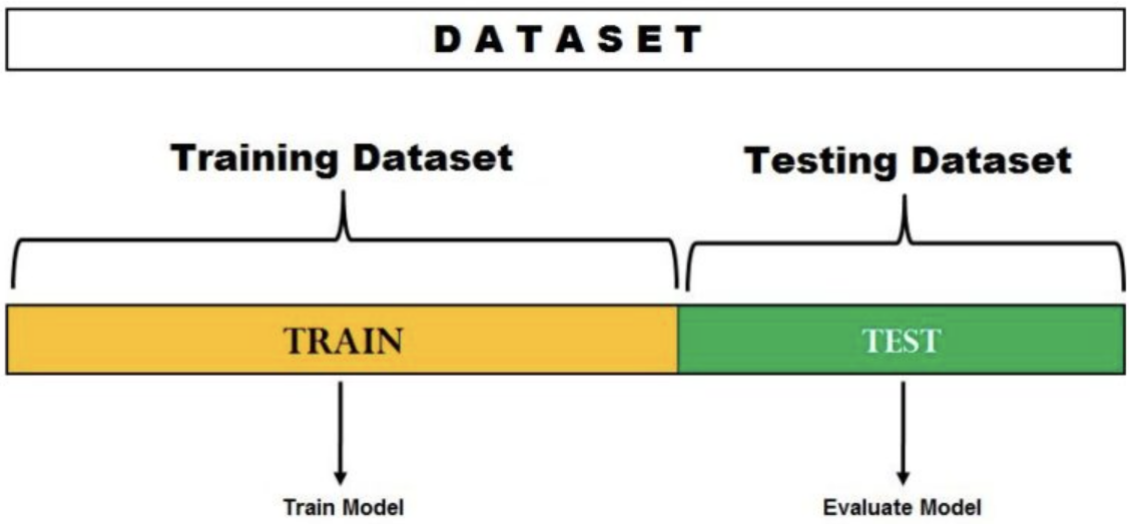
Introducción

Conjunto de entrenamiento, validación y test.

En aprendizaje supervisado, para entrenar un modelo de Machine Learning y poder saber si está funcionando bien, separemos el conjunto de datos inicial en 2:

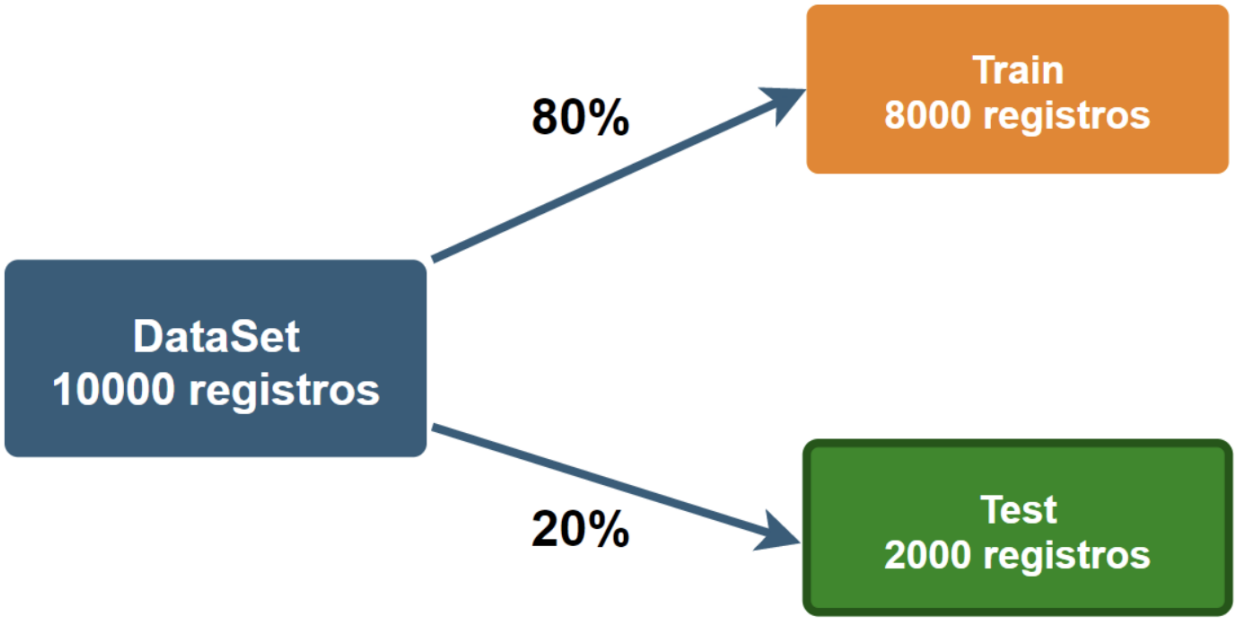
- **Conjunto de entrenamiento (train).**
- **Conjunto de pruebas (test).**

Generalmente, se divide aplicando la regla "80-20". Y se toman muestras aleatorias -no en secuencia, si no, mezclado.



Para hacer el ejemplo sencillo, supongamos que queremos hacer clasificación usando un algoritmo supervisado, con lo cual tendremos:

- **X_train** con 8.000 registros para entrenar
- **y_train** con las "etiquetas" de los resultados esperados de X_train
- **X_test** con 2.000 registros para test
- **y_test** con las "etiquetas" de los resultados de X_test



Para ello se utiliza el método **`train_test_split`** que nos permite fácilmente dividir un conjunto de datos de una matriz o DataFrame en dos aleatorios con un tamaño dado. Esta función se puede llamar de la siguiente manera:

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Se han pasado como parámetros a la función las variables "X" e "y" las cuales representan los datos y las etiquetas de los mismos. Como resultado se obtiene un conjunto de datos para entrenamiento (X_train e y_train) y otro para test (X_test e y_test). En este caso, por defecto, el 75% de todos los registros estarán en el conjunto de entrenamiento y el 25 restante en el de test.

La función admite diferentes propiedades interesantes con las adaptar el funcionamiento del método, entre las que se puede destacar:

- `test_size` : el tamaño del conjunto de datos que se asigna para test que tiene que ser un valor entre 0 y 1. Una opción complementaria a esta es train_size la que se puede usar en lugar de esta para indicar el tamaño del conjunto de entrenamiento. Usar una u otra es solamente una cuestión de preferencia personal.
- `random_state` : un entero con el que se indica la semilla utilizada para la selección de datos. Parámetro que es clave cuando necesitamos que los resultados sean repetibles, por lo que es aconsejable usarlos siempre.
- `stratify` : una variable con la que se puede indicar como hacer una estratificación de los datos.

Entrenamiento.

Una vez dividido el dataset inicial (Ej. 10000 muestras) en los dos conjuntos mencionados anteriormente, aplicando la regla 80-20, se utilizarán 8.000 registros para entrenar (train) y 2.000 para probar (test). Para aplicar el entrenamiento, una vez inicializado el modelo se utiliza el siguiente método:

```
modelo.fit(X_train, y_train)
```

Evaluación

Después de entrenar nuestro modelo y habiendo decidido como métrica el Accuracy (el % de aciertos. Luego veremos si esta metrica es la más adecuada) imaginemos que obtenemos un 75% sobre el set de entrenamiento (y asumimos que ese porcentaje nos sirve para nuestro objetivo de negocio).

Los 2.000 registros que separamos en X_test aún nunca han pasado por el modelo de ML. Lo utilizaremos ahora:

```
modelo.predict(X_test)
```

Como verás, no estamos usando `fit()` , sólo pasaremos los datos sin la columna de "y_test" que contiene las etiquetas (la solución). Además remarco que estamos haciendo predicción; me refiero a que el modelo NO se está entrenando ni "incorporando conocimiento". El modelo se limita a "ver la entrada y mostrar una salida".

Cuando hacemos el `predict()` sobre el conjunto de test y obtenemos las predicciones, las podemos comprobar y contrastar con los valores reales almacenados en y_test y hallar así la métrica que usamos. Los resultados que nos puede dar serán:

- Si el accuracy en Test es cercano al de Entrenamiento (dijimos 75%) por ejemplo en este caso si estuviera entre 65 ú 85% quiere decir que nuestro modelo entrenado está generalizando bien y lo podemos dar por bueno (siempre y cuando estemos conformes con las métricas obtenidas).
- Si el Accuracy en Test es muy distinto al de Entrenamiento tanto por encima como por debajo, nos da un 99% ó un 25% (lejano al 75%) entonces es un indicador de que nuestro modelo no ha entrenado bien y no nos sirve. De hecho este podría ser un indicador de Overfitting o sobre ajuste.

Para evaluar mejor el segundo caso, es donde aparece el **"Conjunto de Validación"**.

Overfitting (sobreajuste) y underfitting (subajuste)

Las principales causas al obtener malos resultados en Machine Learning son el overfitting o el underfitting de los datos. Cuando entrenamos nuestro modelo intentamos “hacer encajar” (fit en inglés) los datos de entrada entre ellos con la salida. Tal vez se pueda traducir overfitting como “sobreajuste” y underfitting como “subajuste” y hacen referencia al fallo de nuestro modelo al generalizar el conocimiento que pretendemos que adquieran.

Generalización del conocimiento

Como si se tratase de un ser humano, las máquinas de aprendizaje deberán ser capaces de generalizar conceptos. Supongamos que vemos un perro Labrador por primera vez en la vida y nos dicen "eso es un perro". Luego nos enseñan un Caniche y nos preguntan: ¿eso es un perro? Diremos "No", pues no se parece en nada a lo que aprendimos anteriormente. Ahora imaginemos que nuestro tutor nos muestra un libro con fotos de 10 razas de perros distintas. Cuando veamos una raza de perro que desconocíamos seguramente seremos capaces de reconocer al cuadrúpedo canino al tiempo de poder discernir en que un gato no es un perro, aunque sea peludo y tenga 4 patas.

Cuando entrenamos nuestros modelos computacionales con un conjunto de datos de entrada estamos haciendo que **el algoritmo sea capaz de generalizar un concepto** para que al consultarle por un nuevo conjunto de datos desconocido éste sea capaz de sintetizarlo, comprenderlo y devolvernos un resultado fiable dada su capacidad de generalización.

El problema de la máquina al generalizar

Si nuestros datos de entrenamiento son muy pocos nuestra máquina no será capaz de generalizar el conocimiento y estará incurriendo en underfitting. Este es el caso en el que le enseñamos sólo una raza de perros y pretendemos que pueda reconocer a otras 10 razas de perros distintas. El algoritmo no será capaz de darnos un resultado bueno por falta de "materia prima" para hacer sólido su conocimiento. También es ejemplo de subajuste cuando la máquina reconoce todo lo que "ve" como un perro, tanto una foto de un gato o un coche.

Por el contrario, si entrenamos a nuestra máquina con 10 razas de perros sólo de color marrón de manera rigurosa y luego enseñamos una foto de un perro blanco, nuestro modelo no podrá reconocerlo cómo perro por no cumplir exactamente con las características que aprendió (el color forzosamente debía ser marrón). Aquí se trata de un problema de overfitting.

Tanto el problema del ajuste "por debajo" como "por encima" de los datos son malos porque no permiten que nuestra máquina generalice el conocimiento y no nos darán buenas predicciones (o clasificación, o agrupación, etc.)

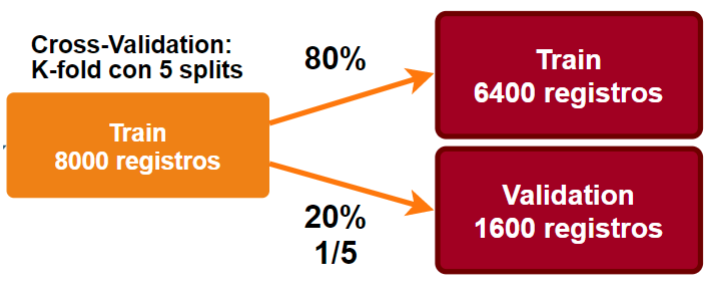
Overfitting en Machine Learning

Es muy común que al comenzar a aprender machine learning caigamos en el problema del Overfitting. Lo que ocurrirá es que nuestra máquina sólo se ajustará a aprender los casos particulares que le enseñamos y será incapaz de reconocer nuevos datos de entrada. En nuestro conjunto de datos de entrada muchas veces introducimos muestras atípicas (ó anomalas) o con "ruido/distorsión" en alguna de sus dimensiones, o muestras que pueden no ser del todo representativas. Cuando "sobre-entrenamos" nuestro modelo y caemos en el overfitting, nuestro algoritmo estará considerando como válidos sólo los datos idénticos a los de nuestro conjunto de entrenamiento –incluidos sus defectos– y siendo incapaz de distinguir entradas buenas como fiables si se salen un poco de los rangos ya prestablecidos.

El equilibro del aprendizaje

Deberemos encontrar un punto medio en el aprendizaje de nuestro modelo en el que no estemos incurriendo en underfitting y tampoco en overfitting. A veces esto puede resultar una tarea muy difícil.

Para reconocer este problema deberemos subvididir nuestro conjunto de datos de entrada para entrenamiento (train) en dos: uno para entrenamiento (train) y otro para la validación (validation) que el modelo no conocerá de antemano. Esta división se suele hacer del 80% para entrenar y 20% para validar.



Cuando entrenamos nuestro modelo solemos parametrizar y limitar el algoritmo, por ejemplo la cantidad de iteraciones que tendrá o un valor de "tasa de aprendizaje" (learning-rate) por iteración y muchos otros. Para lograr que nuestro modelo dé buenos resultados iremos revisando y contrastando nuestro entrenamiento (train) con el conjunto de validación (validation) y su tasa de errores, utilizando más o menos iteraciones, etc. hasta dar con buenas predicciones y sin tener los problemas de over-under-fitting.

Buenas prácticas para prevenir el sobreajuste de datos

Para intentar que estos problemas nos afecten lo menos posible, podemos llevar a cabo diversas acciones.

- **Cantidad mínima de muestras** tanto para entrenar el modelo como para validarlo.
- **Clases variadas y equilibradas en cantidad**: En caso de aprendizaje supervisado y suponiendo que tenemos que clasificar diversas clases o categorías, es importante que los datos de entrenamiento estén balanceados. Supongamos que tenemos que diferenciar entre manzanas, peras y bananas, debemos tener muchas fotos de las 3 frutas y en cantidades similares. Si tenemos muy pocas fotos de peras, esto afectará en el aprendizaje de nuestro algoritmo para identificar esa fruta.
- **Conjunto de Test de datos**. Siempre subdividir nuestro conjunto de datos y mantener una porción del mismo "oculto" a nuestra máquina entrenada. Esto nos permitirá obtener una valoración de aciertos/fallos real del modelo y también nos permitirá detectar fácilmente efectos del overfitting /underfitting.
- **Parameter Tunning o Ajuste de Parámetros**: deberemos experimentar sobre todo dando más/menos "tiempo/iteraciones" al entrenamiento y su aprendizaje hasta encontrar el equilibrio.
- **Cantidad excesiva de Dimensiones (features)**, con muchas variantes distintas, sin suficientes muestras. A veces conviene eliminar o reducir la cantidad de características que utilizaremos para entrenar el modelo. Una herramienta útil para hacerlo es PCA.
- Quiero notar que si nuestro modelo es una red neuronal artificial –deep learning-, podemos caer en overfitting si usamos capas ocultas en exceso, ya que haríamos que el modelo memorice las posibles salidas, en vez de ser flexible y adecuar las activaciones a las entradas nuevas.
- **El sobreajuste o overfitting suele darse con mayor probabilidad en modelos no lineales**, por ello muchos de estos algoritmos de aprendizaje automático también incluyen parámetros o técnicas para limitar y restringir la cantidad de detalles que aprende. Algunos ejemplos de algoritmos no lineales son los siguientes:
  - Decision Trees
  - Naive Bayes
  - Support Vector Machines
  - Neural Networks
- **El subajuste o underfitting suele darse con mayor probabilidad en modelos lineales**, como por ejemplo:
  - Logistic Regression
  - Linear Discriminant Analysis
  - Perceptron

Si el modelo entrenado con el conjunto de train tiene un 90% de aciertos y con el conjunto de test tiene un porcentaje muy bajo, esto señala claramente un problema de overfitting.

Si en el conjunto de Test sólo se acierta un tipo de clase (por ejemplo "peras") o el único resultado que se obtiene es siempre el mismo valor será que se produjo un problema de underfitting.

Técnicas de evaluación

Las técnicas de evaluación que se tratarán y que son muy utilizadasa son:

- **Matriz de confusión** (confusion_matrix)
- **Reporte de clasificación** (classification_report)
- **Métricas** (accuracy, precision, recall, f1-score)
- **Curva ROC (área AUC)**

Evaluación del rendimiento en algoritmos de aprendizaje supervisado (Clasificación/Regresión). Matriz de confusión

En el campo de la inteligencia artificial y el aprendizaje automático una matriz de confusión es una herramienta que permite visualizar el desempeño o rendimiento de un algoritmo de aprendizaje supervisado (algoritmos de clasificación sobre todo y en algún caso de regresión).

Terminología

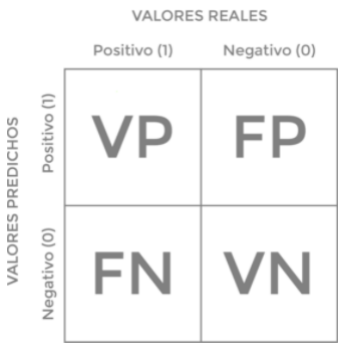
Estas métricas también tienen su correspondiente nombre en español, pero es importante que sepas su nombre en inglés porque muchas librerías (scikit-learn), las tienen ya implementadas. En esta tabla puedes encontrar la correspondencia.

Inglés	Español
Precision	Precisión
Recall	Exhaustividad
F1-score	Valor-F o Medida F
Accuracy	Exactitud
Confusion Matrix	Matriz de Confusión
True Positive	Positivos Verdaderos
True Negative	Negativos Verdaderos
False Positive	Positivos Falsos
False Negative	Negativos Falsos

Matriz de confusión

La matriz de confusión es un método de evaluación de rendimiento de un modelo de clasificación (a veces se usa en regresión). La matriz compara los valores reales con los predichos por el modelo de aprendizaje. Esto nos ayuda a ver que tan bien está funcionando nuestro modelo

Así se ve de forma general



- La variable de destino tiene dos valores: **positivo** o **negativo**
- Las **columnas** representan los **valores reales** de la variable objetivo.
- Las **filas** representan los **valores predichos** de la variable objetivo.

Para tener claro el significado de cada elemento dentro de la matriz de confusión (TP, FP, FN y TN) se define:

- **Verdadero Positivo (VP) o (TP: True Positive)**
  - El valor predicho coincide con el valor real

    - El valor real fue positivo y el modelo predijo un valor positivo
- **Verdadero Negativo (VN) o (TN: True Negative)**
    - El valor predicho coincide con el valor real
    - El valor real fue negativo y el modelo predijo un valor negativo
- **Falso Positivo (FP) o (FP: False Positive): error de tipo 1**
    - El valor predicho fue predicho falsamente
    - El valor real fue negativo pero el modelo predijo un valor positivo
    - También conocido como error tipo 1
- **Falso Negativo (FN) o (FN: False Negative): error de tipo 2**
    - El valor predicho fue predicho falsamente
    - El valor real fue positivo pero el modelo predijo un valor negativo
    - También conocido como error tipo 2

Resumiendo:

VP (TP)	Verdadero positivo
Interpretación	Predijo positivo y es verdad

VN (TN)	Verdadero negativo
Interpretación	Predijo negativo y es verdad

FP (FP)	Falso positivo
Interpretación	Predijo positivo pero es falso

FN (FN)	Falso negativo
Interpretación	Predijo negativo pero es positivo

Truco para recordar fácilmente la matriz de confusión:

- **Positivo (Positive) o Negativo (Negative)**: se refiere a la predicción. Si el modelo predice 1 entonces será positivo, y si predice 0 será negativo.
- **Verdadero (True) o Falso (False)**: se refiere si la predicción es correcta o no.

Ejemplo de perros y gatos.

En este ejemplo, tenemos:

- Verdadero Positivo (VP) : El modelo dijo que era un gato y efectivamente era un gato 😀 (buena predicción)
- Falso Positivo (FP): Predijo que era un gato cuando en realidad era un perro ☹ (mala predicción)
- Falso Negativo (FN) Predijo que no era un gato cuando en realidad si lo era ☹ (mala predicción)
- Verdadero Negativo (VN): el modelo dijo que no era un gato y efectivamente no lo era 😀 (buena predicción)

Métricas de la matriz de confusión

Básicamente las métricas nos sirven para evaluar mejor la matriz. Y minimizar los falsos negativos vs los falsos positivos y aplicaremos cada una dependiendo del caso.

Las metricas son:

- Recall o exhaustividad
- Presicion o precisión
- Accuracy o exactitud
- F1-Score o Medida F

Vamos a explicar esto con el ejemplo de los perros y los gatos.

De forma simple **queremos saber si un modelo logra predecir si el animal que le estoy mostrando es un gato o un perro** (los dos valores para nuestra variable objetivo serían: Gato y No Gato)

Ahora, debemos preguntarnos: **¿Por qué necesitamos una matriz de confusión cuando tenemos simplemente la accuracy (de todas las clases cuantas se predijeron correctamente)?** Bueno, veamos dónde falla esto.

Digamos que tenemos los siguientes datos y hacemos calculos:

- TP o VP: 30
- TN o VN: 930
- FP: 30
- FN: 10

$$Accuracy = \frac{30 + 930}{30 + 30 + 930 + 10} = 0.96$$

Precision (Precisión)

Con la métrica de precisión podemos medir la **calidad** del modelo de machine learning en tareas de clasificación. Nos dice cuántos de los casos predichos correctamente resultaron realmente positivos. En el ejemplo, la precisión es la respuesta a la pregunta **¿qué porcentaje hay de gatos?**

$$Precision = \frac{VP}{VP + FP} = \frac{30}{30 + 30} = 0.5$$

Recall (Exhaustividad)

La métrica de exhaustividad nos va a informar sobre la **cantidad** que el modelo de machine learning es capaz de identificar. En el ejemplo, la exhaustividad (recall) es la respuesta a la pregunta **¿qué porcentaje de los gatos somos capaces de identificar?** Es decir, nos dice cuántos de los casos positivos reales pudimos predecir correctamente con nuestro modelo.

$$Recall = \frac{VP}{VP + FN} = \frac{30}{30 + 10} = 0.75$$

El 50% de los casos predichos correctamente resultaron ser casos positivos. Mientras que nuestro modelo predijo con éxito el 75% de los positivos. ¿Ves la diferencia?

La precisión es una métrica útil en los casos en los que los falsos positivos son una preocupación mayor que los falsos negativos.

El recall es una métrica útil en los casos en que el falso negativo triunfa sobre el falso positivo.

Pero habrá casos en los que no haya una distinción clara entre si la precisión es más importante o el recall. ¿Qué debemos hacer en esos casos? ¡Los combinamos y entra en juego F1-Score o Medida F!

F1-Score (Medida F)

El valor F1 se utiliza para **combinar las medidas de precision y recall en un sólo valor**. Esto es práctico porque hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones.

F1 se calcula haciendo la media armónica entre la precisión y la exhaustividad:

$$F1 = 2 * \frac{precision * recall}{precision + recall} = 2 * \frac{0.5 * 0.75}{0.5 + 0.75} = 0.6$$

El valor F1 asume que nos importa de igual forma la precisión y la exhaustividad. Esto no tiene que ser así en todos los problemas. Por ejemplo, cuando necesitamos predecir si hay riesgo de que un trozo de basura espacial se choque con un satélite, podemos valorar más la exhaustividad (% de choques que somos capaces a detectar) a riesgo de tener una peor precisión.

Accuracy (Exactitud)

La exactitud (accuracy) mide el porcentaje de casos que el modelo ha acertado. Esta es una de las métricas más usadas y favoritas ... que te recomiendo evitar! El problema con la exactitud es que nos puede llevar al engaño, es decir, puede hacer que un modelo malo (como el del ejemplo) parezca que es mucho mejor de lo que es.

El accuracy (exactitud) se calcula con la siguiente fórmula:

$$accuracy = \frac{VP + VN}{VP + VN + FP + FN} = \frac{30 + 930}{30 + 930 + 30 + 10} = 0.96$$

Es decir, según esta metrica el modelo acierta el 96% de las veces. Como ves, la exactitud es una métrica muy engañosa. De hecho, si tuviésemos un modelo que siempre predijera a los gatos como perros (pasamos los 30 de VP a que se conviertan en falsos negativos en FN), **¡su accuracy sería del 93% a pesar de no haber predicho correctamente ningun gato!**.

- VP = 0
- VN = 930
- FP = 30
- FN = 40

$$accuracy = \frac{VP + VN}{VP + VN + FP + FN} = \frac{0 + 930}{0 + 930 + 30 + 40} = 0.93$$

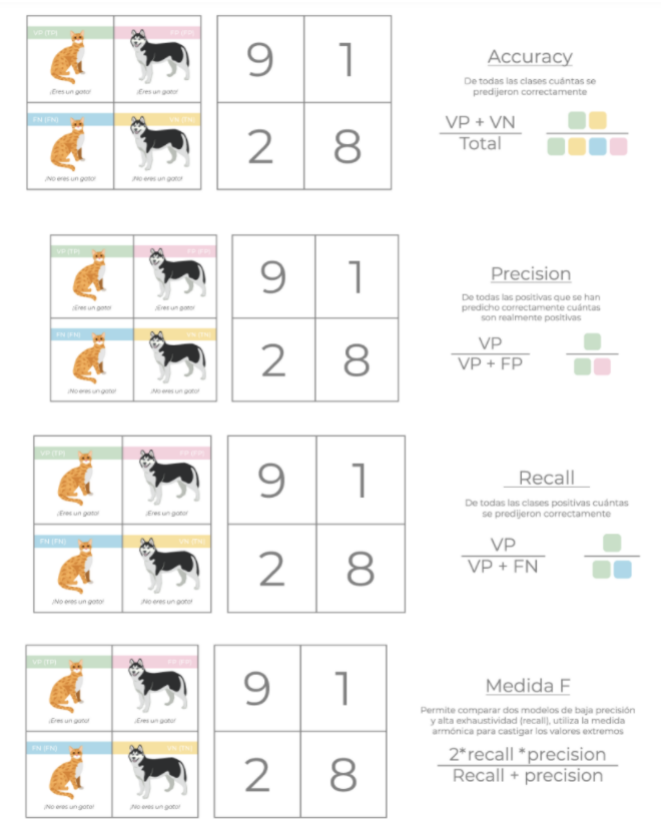
CUIDADO: La métrica accuracy (exactitud) no funciona bien cuando las clases están desbalanceadas como es en este caso. **La mayoría de los animales no son gatos, así que es muy fácil acertar diciendo que es un perro**. Para problemas con clases desbalanceadas es mucho mejor usar precision, recall y F1. Estas métricas dan una mejor idea de la calidad del modelo.

Ejemplo perros y gatos

En este caso, tenemos que el modelo está definido como:

- Caracteristicas:
  - 1: Gato
  - 0: No Gato o lo que es lo mismo, Perro
- Datos del ejemplo
  - Real ..... [1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0]
  - Predicho .. [1,1,1,1,1,1,1,1,1,0,0,1,0,0,0,0,0,0,0,0]

De esta forma, las métricas de la matriz de confusión quedan definidas como:



CUIDADO: La matriz de confusión que hemos visto hasta ahora, es la clásica (la teórica), pero scikit learn nos mostrará otras dos opciones, veamos las diferencias:

Ejemplo de código

In [ ]:
```
from sklearn import metrics

real =     [1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0]
predicho = [1,1,1,1,1,1,1,1,1,0,0,1,0,0,0,0,0,0,0,0]
```

In [ ]:
```
# Introducir manualmente el resultado
vp = 9
vn = 8
fp = 1
fn = 2

print(f'Verdaderos positivos (VP) = {vp}')
print(f'Verdaderos negativos (VN) = {vn}')
print(f'Falsos positivos (FP) = {fp}')
print(f'Falsos positivos (FN) = {fn}')
```

In [ ]:
```
matriz = metrics.confusion_matrix(real, predicho, labels=[0,1])
print("Matriz de confusión:\n",matriz,end=2*'\n')

# Seleccionar elementos de la matriz de confusión
vp = matriz[1][1]
vn = matriz[0][0]
fp = matriz[0][1]
fn = matriz[1][0]

print(f'Verdaderos positivos (VP) = {vp}')
print(f'Verdaderos negativos (VN) = {vn}')
print(f'Falsos positivos (FP) = {fp}')
print(f'Falsos positivos (FN) = {fn}')
```

In [ ]:
```
# En este caso hemos decidido que lo que queremos predecir son los gatos (1)
matriz = metrics.confusion_matrix(real, predicho, labels=[1,0])
print("Matriz de confusión:\n",matriz,end=2*'\n')

# Seleccionar elementos de la matriz de confusión
vp = matriz[0][0]
vn = matriz[1][1]
fp = matriz[1][0]
fn = matriz[0][1]
```

```
print(f'Verdaderos positivos (VP) = {vp}')
print(f'Verdaderos negativos (VN) = {vn}')
print(f'Falsos positivos (FP) = {fp}')
print(f'Falsos positivos (FN) = {fn}')
```

In [ ]:
```
vp, fn, fp, vn = confusion_matrix(real, predicho, labels=[1,0]).reshape(-1)
print("Verdadero Positivo (VP o TP):", vp) # Predijo positivo y es verdad: 9
print("Verdadero Negativo (VN o TN):", vn) # Predijo negativo y es verdad: 8
print("Falso Positivo (FP):", fp) # Predijo positivo pero es falso: 2
print("Falso Negativo (FN):", fn) # Predijo negativo pero es positivo: 1
```

In [ ]:
```
# Calcular manualmente métricas de la matriz de confusión
accuracy = (vp + vn) / (vp + vn + fp + fn)
precision = vp / (vp + fp)
recall = vp / (vp + fn)
f1 = (2 * recall * precision) / (recall + precision)
print("Accuracy o exactitud:", round(accuracy,2))
print("Precision:", round(precision,2))
print("Recall o exhaustividad:", round(recall,2))
print("F1-Score o Medida F:", round(f1,2))
```

In [ ]:
```
# Mostramos el informe de la clasificación.
report = metrics.classification_report(real, predicho)
print(report)
```

Ejemplo de perros y gatos

Recordemos el ejemplo de los perros y los gatos explicados anteriormente:

De forma simple **queremos saber si un modelo logra predecir si el animal que le estoy mostrando es un gato o un perro** (los dos valores para nuestra variable objetivo serían: Gato y No Gato)

Ahora, debemos preguntarnos: **¿Por qué necesitamos una matriz de confusión cuando tenemos simplemente la accuracy (de todas las clases cuantas se predijeron correctamente)?** Bueno, veamos dónde falla esto.

Digamos que tenemos los siguientes datos y hacemos calculos:

- TP o VP: 30
- TN o VN: 930
- FP: 30
- FN: 10

In [ ]:
```
vp = 30
vn = 930
fp = 30
fn = 10

accuracy = (vp + vn) / (vp + vn + fp + fn)
precision = vp / (vp + fp)
recall = vp / (vp + fn)
f1 = (2 * recall * precision) / (recall + precision)
print("Precision:", round(precision,2))
print("Recall o exhaustividad:", round(recall,2))
print("F1-Score o Medida F:", round(f1,2))
print("Accuracy o exactitud:", round(accuracy,2))
```

Nos da como resultado un 96% de accuracy, esto a priori puede parecer un resultado excelente .... pero no lo es ... Analicemos todas las métricas.

Curvas ROC

En Machine Learning, la medición del rendimiento es una tarea esencial. Entonces, cuando se trata de un problema de clasificación, podemos contar con una curva AUC-ROC. Esta es una de las métricas de evaluación más importante para verificar el rendimiento de cualquier modelo de clasificación.

Una **curva ROC (curva de característica operativa del recepto)** es un gráfico que muestra el rendimiento de un modelo de clasificación en todos los umbrales de clasificación. Esta curva representa dos parámetros:

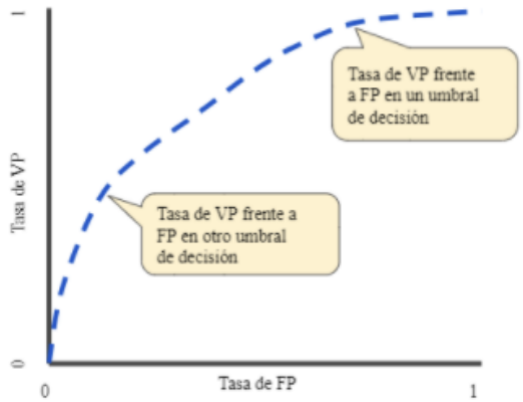
- **Tasa de verdaderos positivos (TPR)**: es sinónimo de exhaustividad y, por lo tanto, se define de la siguiente manera:

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

- **Tasa de falsos positivos (FPR)** se define de la siguiente manera:

$$FPR = \frac{FP}{FP + VN}$$

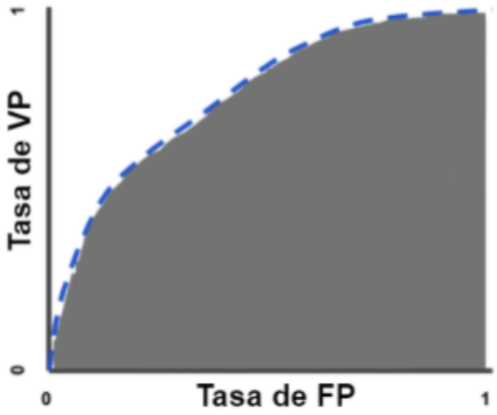
Una curva ROC representa TPR frente a FPR en diferentes umbrales de clasificación. Reducir el umbral de clasificación clasifica más elementos como positivos, por lo que aumentarán tanto los falsos positivos como los verdaderos positivos. En la siguiente figura, se muestra una curva ROC típica.



¿Qué es la curva AUC?

AUC significa "área bajo la curva ROC". Esto significa que el **AUC mide toda el área bidimensional por debajo de la curva ROC** completa.

Un modelo cuyas predicciones son un 100% incorrectas tiene un AUC de 0.0; otro cuyas predicciones son un 100% correctas tiene un AUC de 1.0.



In [ ]:

```
# Curva ROC
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

print("El modelo obtuvo un AUC (en base a la curva ROC) de:", roc_auc_score(real, predicho))

fpr, tpr, _ = roc_curve(real, predicho)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.ylim([0.0, 1.05])
plt.xlabel("Tasa de Falsos Positivos")
plt.ylabel("Tasa de Verdaderos Positivos")
plt.title("Curva ROC para nuestro Modelo")
plt.legend(loc="lower right")
plt.show()
```