Cross validation
&
Evaluating metric and scoring
...

Crispin Alvarado
Camila Silva

Evaluación de modelos y selección óptima de parámetros.

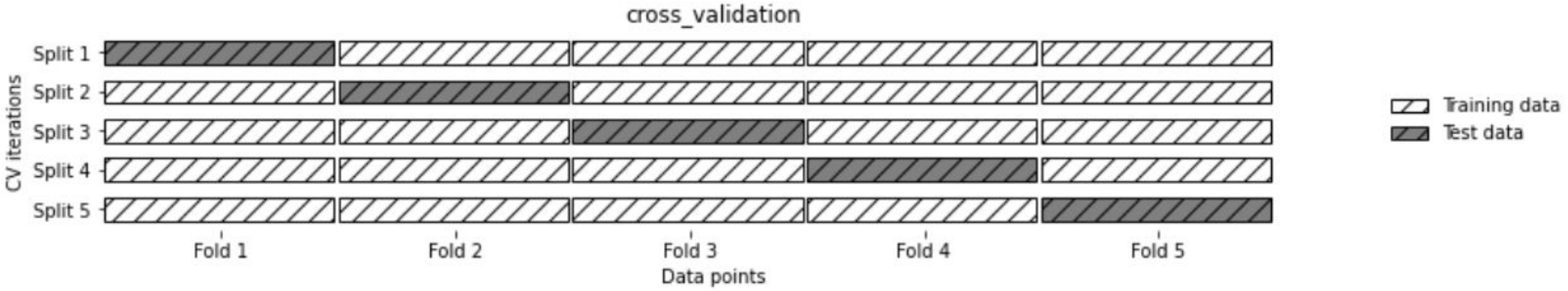
Cross-Validation (Validación Cruzada)

Método estadístico que evalúa el rendimiento de los algoritmos de clasificación. Evaluando la generalización de sus resultados.

En la validación cruzada, los datos se dividen repetidamente y se entrenan varios modelos.

La versión de validación cruzada más utilizada, es la **k-fold cross-validation**. Donde k es generalmente 5 o 10.

1. Se divide el modelo en k=5 partes
2. Se entrenan varios modelos
3. Para cada uno de estos se calcula la precisión.



Cross-validation está implementado en *scikit-learn* usando la función *cross_val_score* del módulo *model_selection*

cross_val_score toma k=3 (en mi caso personal toma k=5) de forma predeterminada

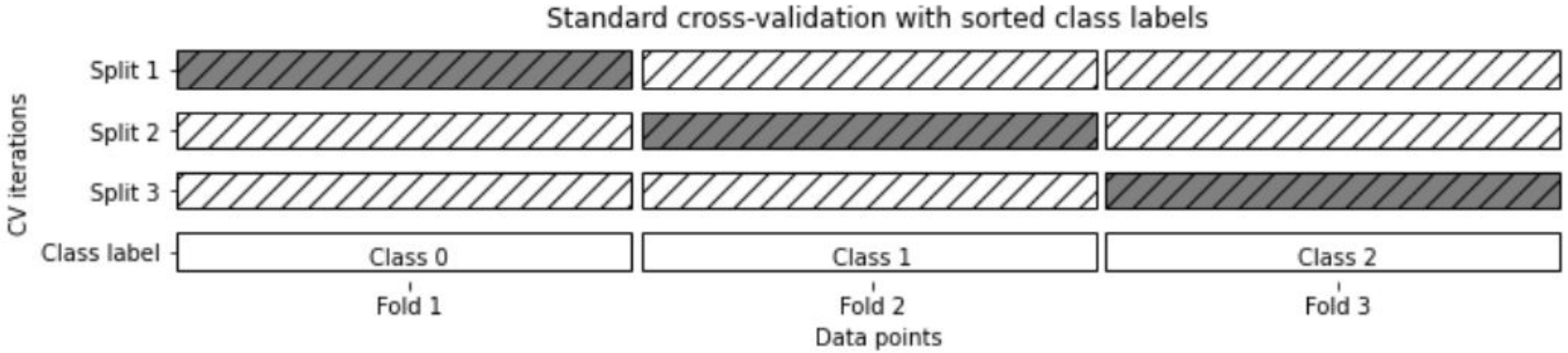
Beneficios del Cross-Validation

- Se generaliza mejor a los datos, por la rotación de los conjuntos.
- La división en folds, proporciona información sobre la sensibilidad a la escogencia de los datos

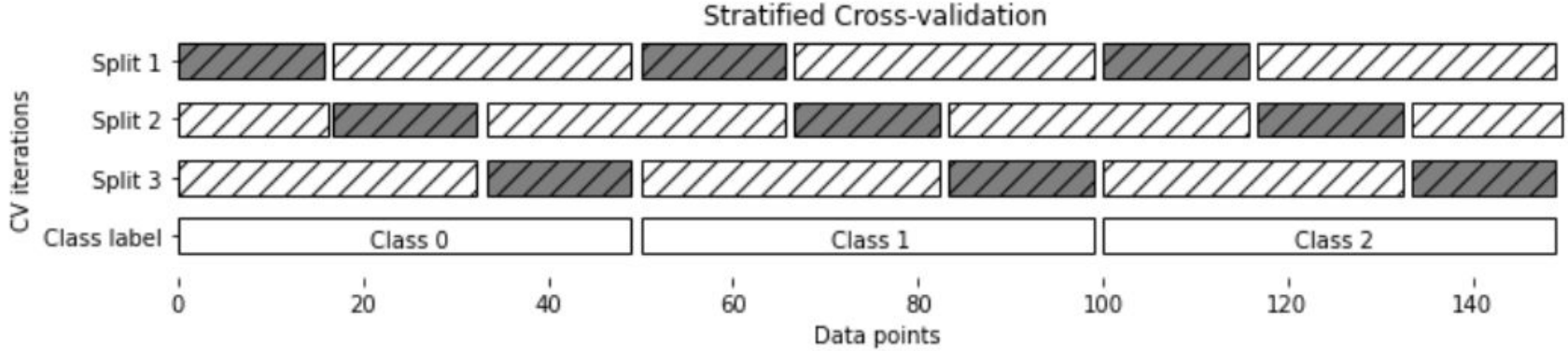
Desventajas del Cross-Validation

Tiene un mayor costo computacional, ya que se entrenan k modelos, en lugar de uno.

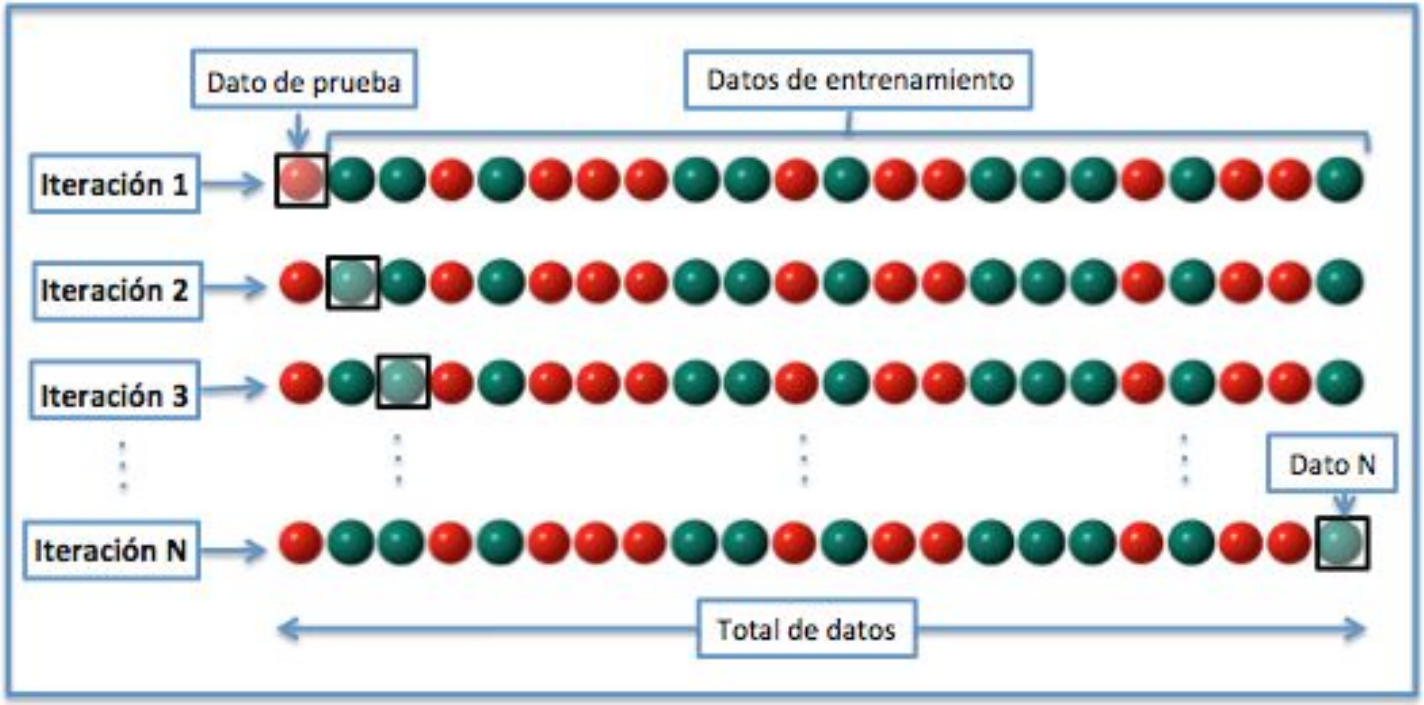
Existen varias formas de usar cross-validation



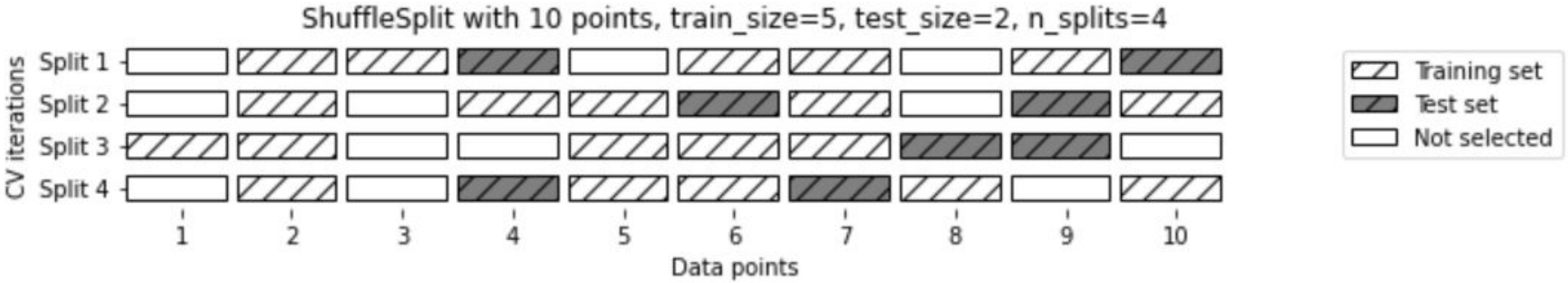
En **Stratified k-Fold Cross-Validation**, se dividen los datos de manera que las proporciones entre clases sean las mismas en cada pliegue que en todo el conjunto de datos.



En **Leave-one-out cross-validation**, se toma un solo dato como conjunto de prueba y todos los demás como entrenamiento.

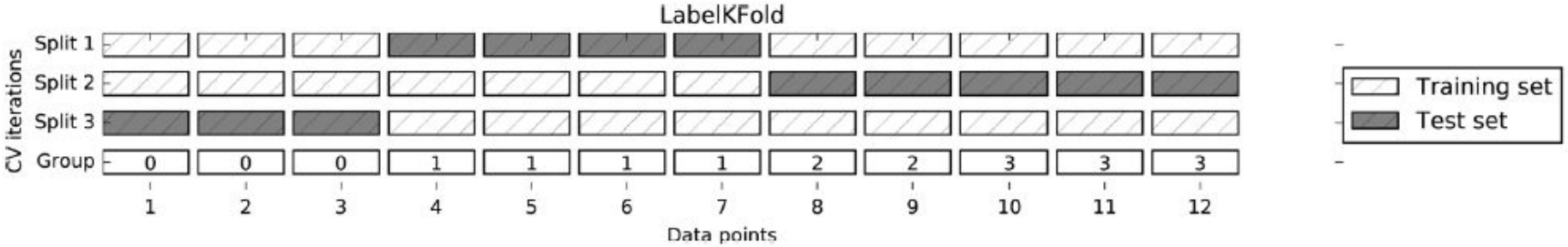


En **Shuffle-split cross-validation**, se divide el conjunto de datos aleatoriamente, tomando conjunto de entrenamiento y de prueba disjuntos (no necesariamente toma la totalidad de los datos)



ShuffleSplit con 10 puntos, train_size = 5, test_size = 2 y n_iter = 4

En **Cross-validation with groups**, se tiene grupos de datos relacionados entre sí antes de dividir en conjunto de entrenamiento y prueba, por lo que estos grupos se deben tener en cuenta y no dividirlos.



Evaluation Metrics and Scoring (Métricas de evaluación y puntuación)

Existen muchas formas posibles de resumir qué tan bien se desempeña un modelo supervisado en un conjunto de datos determinado.

Ejemplo de aplicación: predicción de cáncer

Se tiene una aplicación para la detección de cáncer usando un test automatizado.

Si el test es negativo, el paciente se supone como sano.

Si el test es positivo, el paciente es sometido a más estudios.

Tipos de errores

Se tienen dos tipos de errores:

1. El paciente está sano y el test sale positivo
2. El paciente está enfermo y el test sale negativo

   A estos errores se les conoce como **falso positivo** y **falso negativo** respectivamente.

Ejemplo: desbalance de clases

- La tarea es predecir cuando un usuario de internet dará click a un ítem publicitario, dará click (mostrando interés) o no.
- Podría pasar que para que un usuario de click a un anuncio, a éste se le sean mostrados 100 anuncios. En otras palabras el dataset consiste de 1% (click) y 99% (no-click).

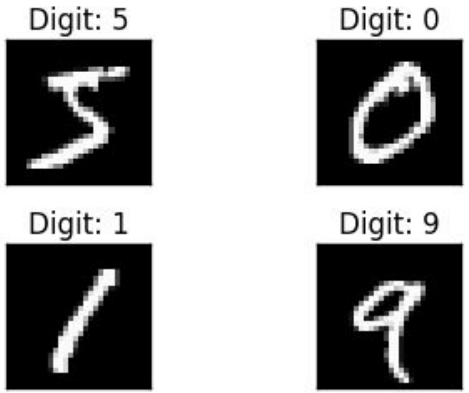
Podemos crear un clasificador con 99% de accuracy, pero eso no toma en cuenta el desbalance en las clases.

Clasificador del dígito 9

Para ilustrar accuracy puede no desempeñarse bien cuando existe el desbalance de clases, creemos un dataset.

```
[1]  from sklearn.datasets import load_digits
     from sklearn.model_selection import train_test_split

     digits = load_digits()
     ## 1 a todos los que son 9 y 0 a los que no
     y = (digits.target == 9)
     X_train, X_test, y_train, y_test = train_test_split(
     digits.data, y, random_state=0)

     print(X_train.shape)
     print(X_test.shape)
```

```
(1347, 64)
(450, 64)
```



Clasificador que siempre predice la clase mayoritaria

```
[2]  import numpy as np
     from sklearn.dummy import DummyClassifier

     dummy_majority = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
     pred_most_frequent = dummy_majority.predict(X_test)
     print("Unique predicted labels: {}".format(np.unique(pred_most_frequent)))
     print("Test score: {:.2f}".format(dummy_majority.score(X_test, y_test)))
```

```
Unique predicted labels: [False]
Test score: 0.90
```

Se obtiene un 90% de accuracy, sin haber aprendido nada

Clasificador con árboles de decisión

```
[3]  from sklearn.tree import DecisionTreeClassifier
     tree = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
     pred_tree = tree.predict(X_test)
     print("Test score: {:.2f}".format(tree.score(X_test, y_test)))
```

```
Test score: 0.92
```

Se obtiene un 2% por encima del predictor anterior

Clasificador que hace predicciones aleatorias

```
[4]  import warnings
     warnings.filterwarnings('ignore')

     dummy = DummyClassifier().fit(X_train, y_train)
     pred_dummy = dummy.predict(X_test)
     print("dummy score: {:.2f}".format(dummy.score(X_test, y_test)))
```

```
dummy score: 0.83
```

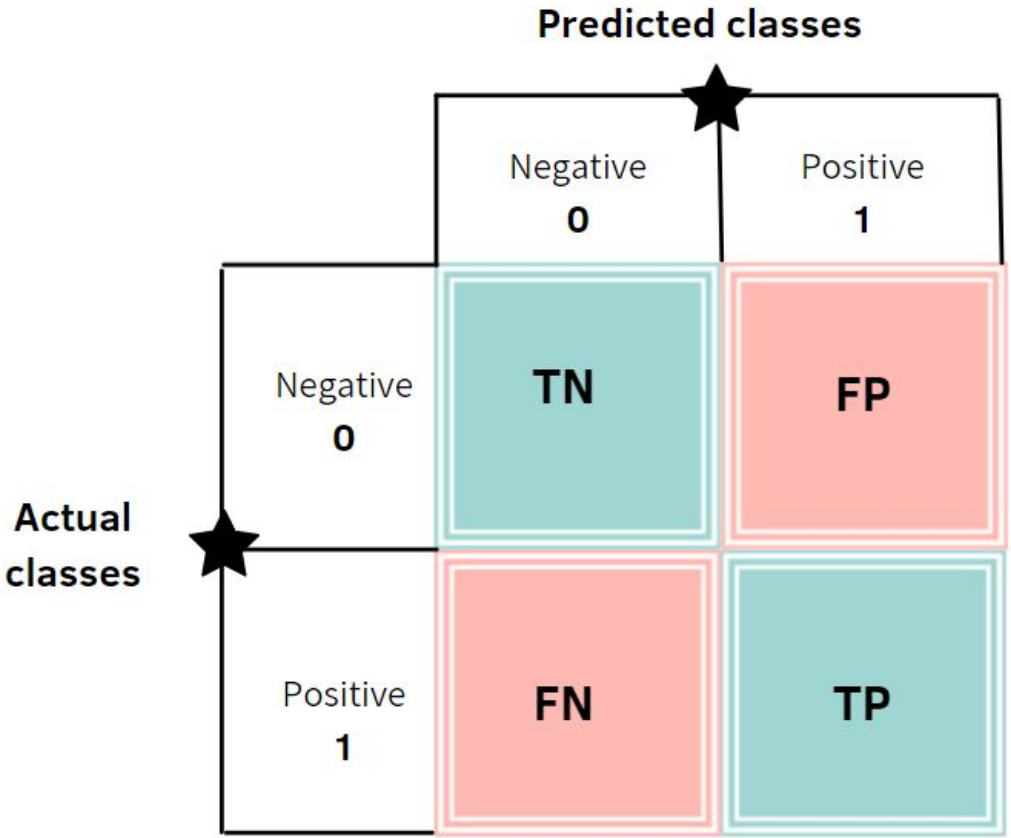
Clasificador con regresión logística

```
[5]  from sklearn.linear_model import LogisticRegression

     logreg = LogisticRegression(C=0.1).fit(X_train, y_train)
     pred_logreg = logreg.predict(X_test)
     print("logreg score: {:.2f}".format(logreg.score(X_test, y_test)))
```

```
logreg score: 0.98
```

Se vuelve difícil de juzgar, cuál de todos los resultados es realmente útil, al tener un clasificador aleatorio con 80% de accuracy

Matriz de confusión



Provee una forma de representar las evaluaciones hechas por un clasificador binario

Matriz de confusión en sklearn

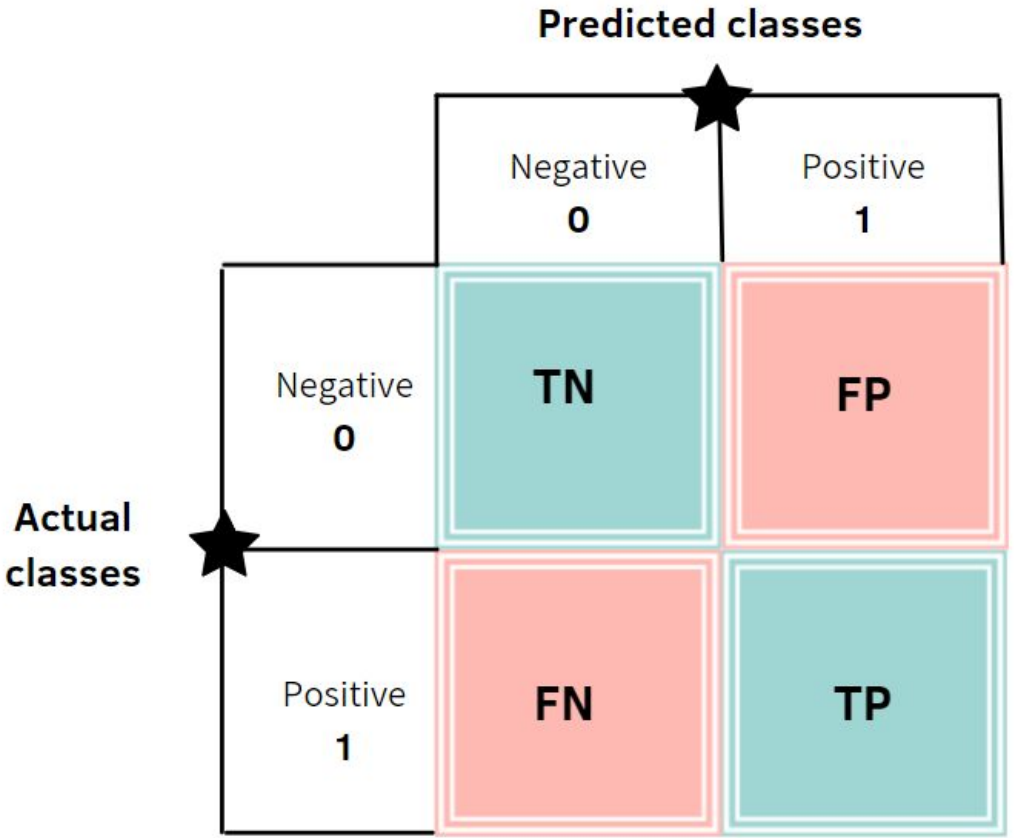
```
[6]  from sklearn.metrics import confusion_matrix

     print("Most frequent class:")
     print(confusion_matrix(y_test, pred_most_frequent))
     print("\nDummy model:")
     print(confusion_matrix(y_test, pred_dummy))
     print("\nDummy model:")
     print(confusion_matrix(y_test, pred_tree))
     print("\nLogistic Regression")
     print(confusion_matrix(y_test, pred_logreg))
```

Matriz de confusión de nuestros clasificadores del dígito 9

```
Most frequent class:
[[403   0]
 [ 47   0]]

Dummy model:
[[373  30]
 [ 44   3]]

Dummy model:
[[390  13]
 [ 24  23]]

Logistic Regression
[[402   1]
 [  6  41]]
```

1. En el primer caso solamente predice una clase
2. En el *Dummy model* tiene un número pequeño de *true positives* y hay más *false positives* que *true positives*
3. Las predicciones hechas con un árbol de decisión parecen ser mejores que las anteriores, pero el accuracy es parecido
4. Tiene menos falsos positivos y falsos negativos

Relación de la matriz de confusión con *accuracy*



accuracy es el número de predicciones correctas dividido entre el número de muestras

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

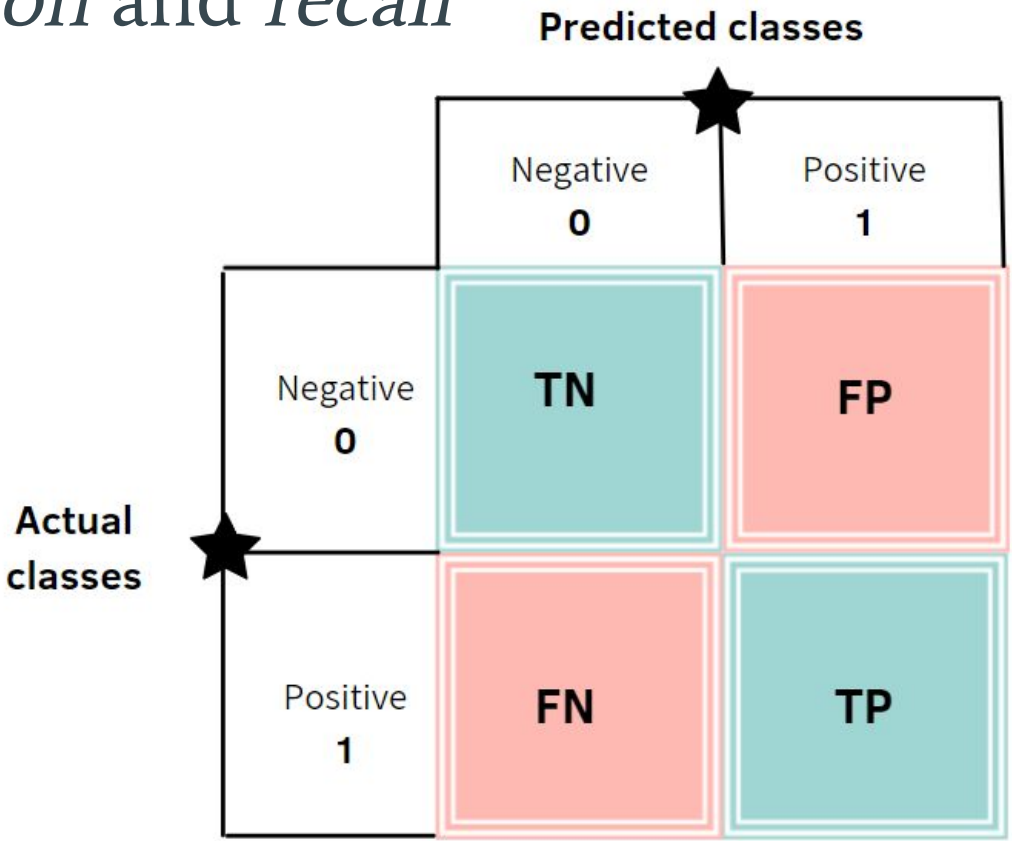
Precision and *recall*

precision (PPV) mide cuantas de las muestras predichas como positivas, son realmente positivas

$$precision = \frac{TP}{TP + FP}$$

recall (TPR) mide cuántas de las muestras positivas son capturadas por predicciones positivas

$$recall = \frac{TP}{TP + FN}$$



F-score o F1-score

Una forma de resumir dos medidas tan importantes como *precisión* y *recall* es con su media armónica, que penaliza los valores extremos

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

Una desventaja del F1-score, es que es más difícil de interpretar que *accuracy*

F1-score en el clasificador del dígito 9

```
[7]  from sklearn.metrics import f1_score

     print("f1 score most frequent: {:.2f}".format( f1_score(y_test, pred_most_frequent)))
     print("f1 score dummy: {:.2f}".format(f1_score(y_test, pred_dummy)))
     print("f1 score tree: {:.2f}".format(f1_score(y_test, pred_tree)))
     print("f1 score logistic regression: {:.2f}".format(f1_score(y_test, pred_logreg)))

     f1 score most frequent: 0.00
     f1 score dummy: 0.07
     f1 score tree: 0.55
     f1 score logistic regression: 0.92
```

Reporte de clasificación

```
[8]  from sklearn.metrics import classification_report

     print(classification_report(y_test, pred_tree,
                                 target_names=["not nine", "nine"]))
```

	precision	recall	f1-score	support
not nine	0.94	0.97	0.95	403
nine	0.64	0.49	0.55	47
accuracy			0.92	450
macro avg	0.79	0.73	0.75	450
weighted avg	0.91	0.92	0.91	450

```
[9]
     print(classification_report(y_test, pred_dummy,
                                 target_names=["not nine", "nine"]))
```

	precision	recall	f1-score	support
not nine	0.89	0.90	0.90	403
nine	0.07	0.06	0.07	47
accuracy			0.82	450
macro avg	0.48	0.48	0.48	450
weighted avg	0.81	0.82	0.81	450

```
[10] print(classification_report(y_test, pred_logreg,
                                 target_names=["not nine", "nine"]))
```

	precision	recall	f1-score	support
not nine	0.99	1.00	0.99	403
nine	0.98	0.87	0.92	47
accuracy			0.98	450
macro avg	0.98	0.93	0.96	450
weighted avg	0.98	0.98	0.98	450

Receiver operating characteristics curve (ROC curve)

La curva ROC considera todos los posibles umbrales para un clasificador dado, mostrando la false positive rate (FPR) contra la true positive rate (TPR).

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

$$TPR = \frac{FP}{FP + TN}$$

TPR es otro nombre para recall y FPR es la fracción de falsos positivos sobre todos los ejemplos negativos.

Receiver operating characteristics curve (ROC curve)