Comité de Ética de la Investigación

Hospital Universitario Torrecárdenas

Validación de Modelos

Autor:

Luis Téllez Ramírez

Noviembre 2021

Versión 1.3

Índice

*

1 Aclaraciones del Texto

El código de Python vendrá dado de la siguiente manera:

```
# comentario de prueba
import libreria as lib

class MyClass(Yourclass):
    def __init__(self, my, yours):
        bla = '5 1 2 3 4'
        print('bla')

def Saludar(nombre):
    print(f'hola ',nombre)
```

Ejemplo 1.1 Se puede importar un archivo de python de la siguiente manera:

```
"""Simulate ARFIMA process.
This module allows simulation of the ARFIMA(p, d, q) processes:
    AR(B) (1-B)^d x = MA(B) epsi .
If d = 1, the process becomes ARMA(p, q) process:
    x[t] = ar[1]*x[t-1] + ar[2]*x[t-2] + ... + ar[p]*x[t-p] +
           + ma[1]*epsi[t-1] + ma[2]*epsi[t-2] + ... + ma[q]*
              epsi[t-q] +
           + epsi[t]
Fractional differencing is done by using Fourier space, as
   proposed by Jensen
& Nielsen (2014).
Typical usage example:
    import arfima
    series = arfima.arfima([], 0.3, [], 2**16, warmup=2**16)
"""
from ._arfima import arfima
```

Se puede usar código Python en línea como: `class MyClass` means ...

2 Validación de Modelos

2.1 ¿Qué es la Validación de Modelos?

Se entiende por validación de modelos, un conjunto de procesos y técnicas cuyo objetivo es verificar que el modelo se comporta como esperamos.

- **Generalización:** Es la habilidad de un modelo de mantener su capacidad de predicción en entrenamiento sobre nuevos datos.
- Un modelo de clasificación/regresión es **válido** en la medida en la que es capaz de **generalizar** a nuevos datos.
- Navaja de Ockham

2.2 ¿Por qué es importante?

El objetivo de un modelo no es otro que entender la naturaleza de los datos y hacer predicciones sobre ellos. La validación determina si el modelo entrenado es fiable. Además, la validación de modelos permite reducir los costes, descubrir más errores, aumentar la escalabilidad y la flexibilidad y mejorar la calidad del modelo.

2.3 Enunciado:

Dado un conjunto de datos $\mathcal{D}$ y un algoritmo de aprendizaje $\mathcal{A}$, si aprendemos un modelo $\mathcal{M}$:

1. ¿Qué garantía tenemos para ese modelo cuando se aplique a nuevos datos?
2. Dados dos algoritmos distintos ($\mathcal{A}_1$ y $\mathcal{A}_2$) y un mismo conjunto de datos $\mathcal{D}$, ¿Cuál es mejor?
3. Dado un algoritmo $\mathcal{A}$ con una serie de parámetros $p_1,\ldots,p_n$ ¿Cuál es la mejor parametrización dado un conjunto de datos $\mathcal{D}$? ¿Y para generalizar?

Bien, la validación de modelos viene a responder a estas preguntas. Para ello, lo primero que se necesita es definir una medida f de rendimiento que nos diga cómo de bueno es un modelo respecto a los datos. Y lo más importante, necesitamos estimar ese valor de forma honesta. El rendimiento de un modelo de aprendizaje supervisado refleja la diferencia entre las predicciones de éste para un conjunto de entradas **X**, y los valores **Y** esperados.

2.4 Complejidad del Modelo

- Aumentar la complejidad del modelo:
  - Puede disminuir el error sobre los datos de entrenamiento y nuevos datos.
  - Depende el tamaño y naturaleza del conjunto de datos.
- Aumento **excesivo** de la complejidad del modelo:
  - Puede disminuir el error sobre los datos de entrenamiento pero aumentarlo sobre nuevos datos.
  - A veces de manera dramática. (Podemos caer en sobreajuste).
- **Sobreajuste:** Se produce cuando el modelo se ajusta **demasiado** al conjunto de datos de entrenamiento, y por ello, no generaliza bien en la predicción de nuevos casos.

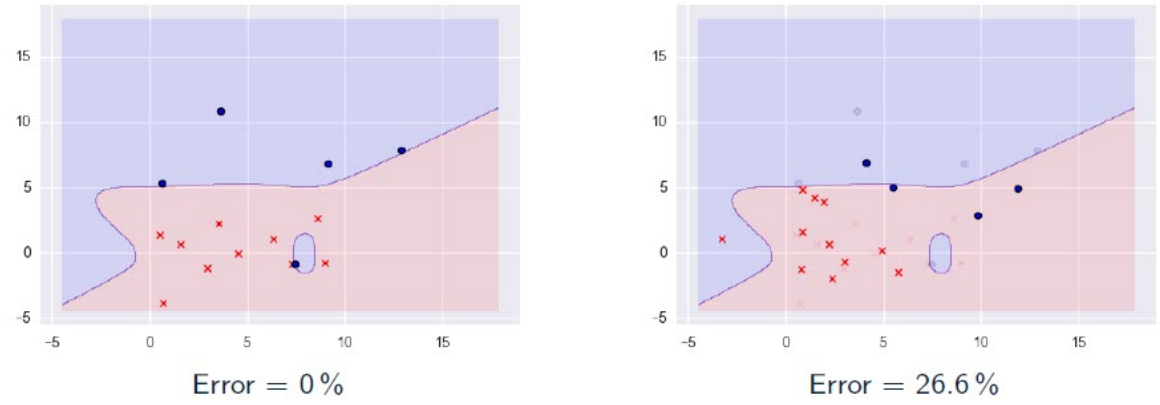


Figura 1: Sobreajuste

2.5 Sesgo o bias

El sesgo o (bias) es el error que se produce debido a la incapacidad del modelo para ajustar los datos. Puede deberse a:

- Asunciones erróneas del algoritmo de aprendizaje.
- Falta de expresividad.

Un sesgo alto hace que el algoritmo no pueda capturar la relación entre las variables de entrada y las variables objetivo (**subajuste** o **underfitting**).

- Ej. Uso de modelos de regresión lineal en la predicción de funciones no lineales.

2.6 Varianza

La varianza es el error que se produce debido a la sensibilidad ante pequeñas fluctuaciones o particularidades específicas en el conjunto de datos de entrenamiento.

- Esas particularidades no se presentan en los nuevos datos.

Una varianza alta puede producir sobreajuste u overfitting.

- Ej: Regresión logística con características polinómicas (grado 5). 1

2.7 Visualizar sesgo-varianza

Figura 2: Sesgo-Varianza

2.7.1 Descomposición del error en sesgo y varianza

$$Error = Sesgo + varianza + error\ irreducible$$

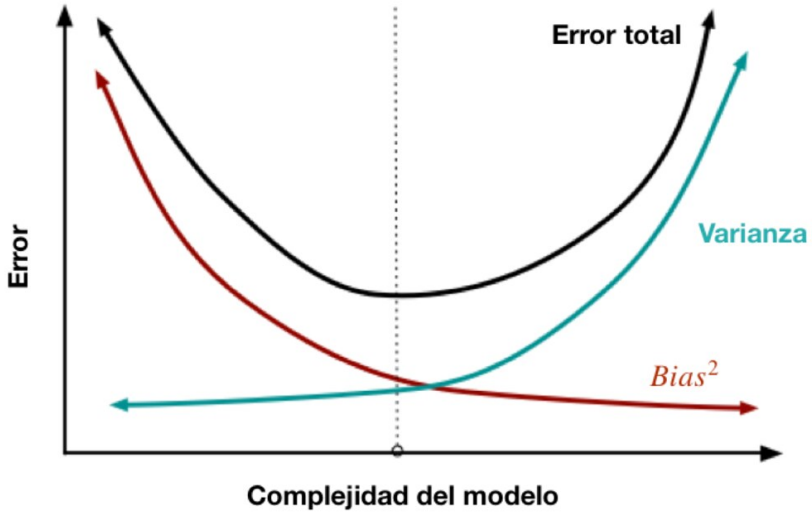


Figura 3: Compensación Sesgo-Varianza

2.8 Definición del rendimiento

El rendimiento de un modelo de aprendizaje supervisado refleja la diferencia entre las predicciones de éste para un conjunto de entradas **X**, y los valores **Y** esperados.

2.8.1 Notas importantes

- Rendimiento se corresponde al que presenta en la predicción de nuevos casos.
- La evaluación del modelo ha de hacerse sobre datos que no hayan sido utilizados en **ninguna fase** del proceso de entrenamiento. (Data Leakage).
- Es necesario desordenar aleatoriamente los ejemplos del conjunto original antes de hacer la partición (de cara a evitar sesgo de ordenación).
- A veces se trabaja con particiones estratificadas. Mantienen la proporción de casos de cada clase en entrenamiento y test. **Especialmente necesario cuando el conjunto de datos es pequeño o muy desbalanceado**.

2.8.2 Problemas de Regresión

El error cuadrático medio, entre una de las muchas métricas que existen para los problemas de regresión, se formula como:

$$ECM(h_\theta, X) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

2.8.3 Problemas de Clasificación

La tasa de acierto/error:

$$\text{ERROR}(c_\theta, X) = \frac{1}{m}\sum_{i=1}^{m}\text{error}\left(c_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

$$\text{error}\left(c_\theta\left(x^{(i)}\right), y^{(i)}\right) = \begin{cases} 0 & \text{si } c_\theta\left(x^{(i)}\right) = y^{(i)} \\ 1 & \text{si } c_\theta\left(x^{(i)}\right) \neq y^{(i)} \end{cases}$$

$$ACIERTO(c_\theta, X) = 1 - \text{ERROR}(\text{c}_\theta, \text{X})$$

Donde **X** representa un conjunto de datos de tamaño m; h_θ y c_θ, respectivamente, los modelos de regresión y clasificación; y θ representa los parámetros de los modelos.

Cuando realizamos el **entrenamiento** de un modelo sobre un conjunto de datos, se nos devuelve la **configuración de parámetros óptima**, es decir, aquella que **minimiza el coste con respecto a los datos utilizados en el aprendizaje**.

Evidentemente, el error del modelo aumentará en la predicción de nuevos casos. [1]

[1] Puede no ocurrir en algunos casos, pero lo habitual es que aumente.

2.9 Aspectos a tener en cuenta: Análisis de curvas de aprendizaje.

- El tamaño del conjunto de datos de entrenamiento es determinante en el rendimiento del algoritmo.
- El aumento en datos de entrenamiento disminuye la varianza.
- Un conjunto suficientemente grande de datos impide el sobreajuste.
- Una manera de analizar el comportamiento del algoritmo (si el error se debe al sesgo o a la varianza):
    - Dibujar los errores de entrenamiento y validación en función del número de ejemplos utilizados en el entrenamiento, y compararlos.

2.10 Subajuste

- El aumento del tamaño del conjunto de datos no ayuda demasiado cuando el problema es que el modelo no tiene capacidad de ajustar (sesgo alto).
- En este caso, tanto el error de entrenamiento como el de validación/test se estancan en un valor alto.

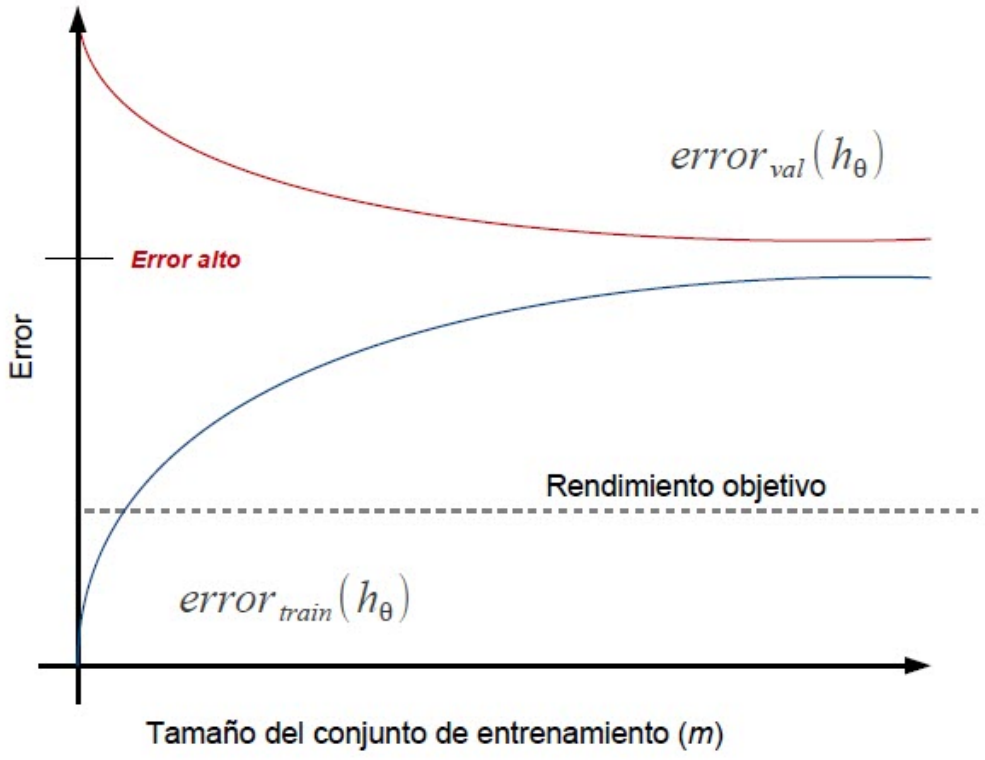


Figura 4: Subajuste

2.11 Sobreajuste

- Cuando el modelo presenta sobreajuste, un aumento en el tamaño del conjunto de datos repercute en una mejor capacidad de generalización (se disminuye la varianza).
- El error de entrenamiento puede aumentar, pero el de validación disminuye, y ambos tienen a acercarse.

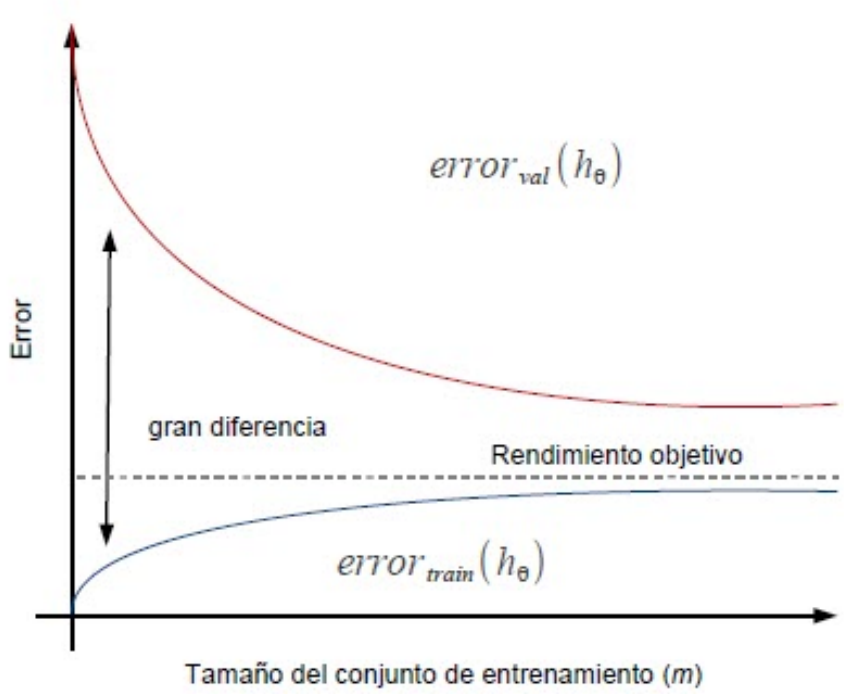


Figura 5: Sobreajuste

Cuando se aborda un problema de aprendizaje, no solamente hay que elegir el algoritmo adecuado. Es necesario establecer diferentes parámetros que tendrían un impacto en el rendimiento final. Por ejemplo:

- Variables (número) utilizadas.
- Creación de características polinomiales.
- Regularización en regresión lineal y logística.
- Topología y número de neuronas en redes neuronales.
- Limitar el número de dependencias en clasificadores bayesianos.
- Umbral de poda en árboles de clasificación.
- etc.

Cuando los modelos son muy expresivos pueden, dependiendo del volumen de datos, sobreajustar. Por eso se busca un compromiso.

En la siguiente imagen, se puede apreciar cómo a medida que aumenta la capacidad de captar detalles, el error en Train se reduce, pero llega un punto en el que el modelo empieza a «memorizar» patrones no generales y empieza a aumentar el error en Test.

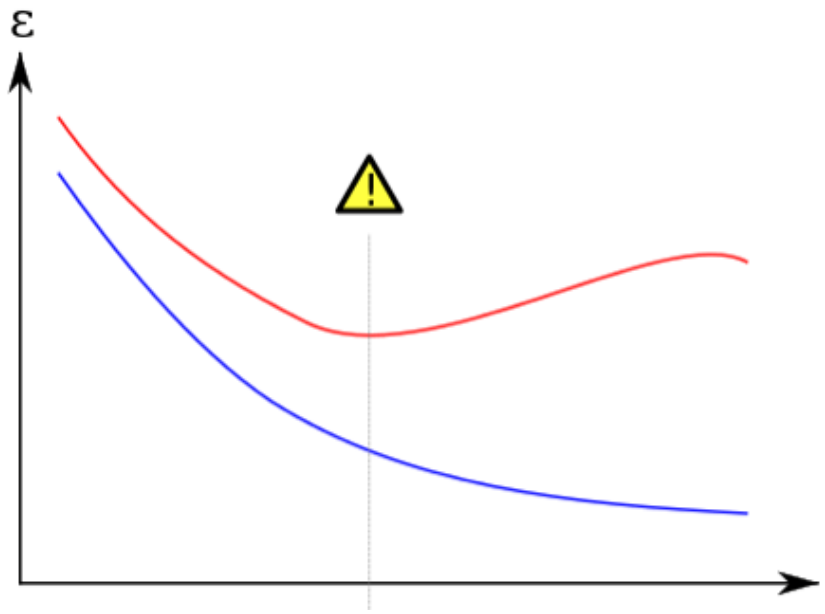


Figura 6: Sobreajuste

2.12 Técnicas de Validación

- Train/test split
- k-Fold Cross-Validation
- Leave-one-out Cross-Validation
- Leave-one-group-out Cross-Validation
- Nested Cross-Validation
- Time-series Cross-Validation
- Wilcoxon signed-rank test
- McNemar's test
- 5x2CV paired t-test
- 5x2CV combined F test

2.13 Técnicas más utilizadas

En primer lugar, hablaremos de las técnicas más utilizadas en este proceso de validación, y si es pertinente, iremos ampliando esta lista progresivamente.

2.13.1 *Train/Test Split*

La técnica más básica de validación de modelos es realizar una división de entrenamiento/validación/prueba en los datos. Una proporción típica para esto podría ser 80/10/10 para asegurarnos de que tenemos suficientes datos de entrenamiento. Después de entrenar el modelo con el conjunto de entrenamiento, pasaremos a validar los resultados y a ajustar los hiperparámetros con el conjunto de validación hasta alcanzar una métrica de rendimiento satisfactoria. Una vez completada esta etapa, pasaremos a probar el modelo con el conjunto de pruebas para predecir y evaluar el rendimiento.

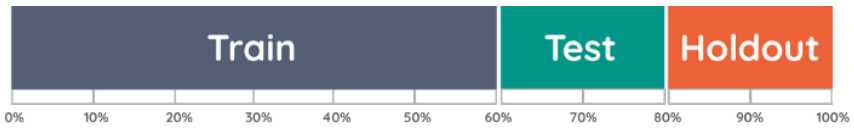


Figura 7: División en train - test

La evaluación entrenamiento/test no es adecuada cuando las bases de datos son pequeñas. Gran diferencia entre las dos particiones (depende de la semilla).

Solución:

- Hold-out repetido/iterado. Repetir k veces.
    1. Aleatorizar conjunto de datos. Dividir train/test.
    2. Aprender sobre el conjunto de entrenamiento y validar con el de test.
    3. Devolver como medida la media obtenido sobre los k conjuntos de test.
- Bootstraping. Repetir k veces:
    1. Obtener una muestra de tamaño m a partir del conjunto de datos mediante **muestreo con reemplazo.**
    2. Aprender el modelo a partir de la muestra obtenida.
    3. Validar el modelo sobre los datos OOB (out of Bag) no seleccionados en el muestreo.
    4. Devolver como medida la media obtenido sobre los k conjuntos de test.

¿Y si seguimos teniendo buena/mala suerte? ¿Se puede hacer más sistemáticamente?

La respuesta es afirmativa y viene dada por la **Validación Cruzada.**

2.13.2 *Validación cruzada K-fold con un conjunto de datos de prueba independiente*

En el caso de que quisiéramos conservar la mayor cantidad de datos posible para la etapa de entrenamiento y no arriesgarnos a perder datos valiosos para el conjunto de validación. Esta técnica no requerirá que los datos de entrenamiento cedan ninguna porción para un conjunto de validación. En este caso, el conjunto de datos se divide en un número k de pliegues, uno de los cuales se utilizará como conjunto de prueba y el resto se utilizará como conjunto de datos de entrenamiento, y esto se repetirá n veces, según especifique el usuario. En una regresión, la media de los resultados se utilizará como resultado final. En una clasificación, se tomará como resultado final la media de los resultados (es decir, exactitud, tasa de verdaderos positivos, F1, etc.).

Figura 8: Validación K-fold

- En clasificación se suele hacer validación cruzada estratificada.
- *Leave One Out:* Cuando k es igual al número de registros de **X** (se verá más adelante).
- **Validación Cruzada Iterada:** r x k-CrossVal. Se realizan r iteraciones de una validación cruzada de k pliegues o divisiones y se devuelve la media.

2.13.3 *Leave-one-out cross-validation with independent test data set. (Validación cruzada con un conjunto de datos de prueba independiente)*

La validación Leave-One-Out es similar a la validación cruzada k-fold. La iteración se lleva a cabo unas veces determinadas y el conjunto de datos se dividirá en $n-1$ conjuntos de datos y el que se elimine será el dato de prueba.

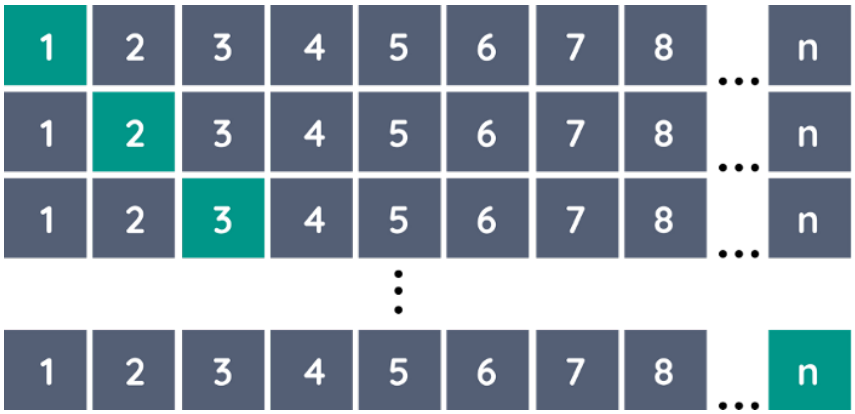


Figura 9: Leave One-out

Cuidado porque este tipo de validación es único para conjuntos de entrenamiento relativamente pequeños debido a su alto coste computacional.

3 Metodología personal para entrenar y validar a la vez

En esta sección, voy a contar cómo suelo entrenar la mayoría de modelos. Ya que es la forma a la que estoy acostumbrado y para mí es buena, rápida y sencilla. Está basada en una combinación de las técnicas que han sido explicadas hasta ahora.

- En primer lugar, realizo una división en **Train** y **Test**, ya que es importante probar el modelo sobre datos que nunca ha visto, para comprobar el rendimiento. Además, procuro que la división en clases de ambos conjuntos sea estratificada (se conserve la proporción).

Una vez que ya tengo la división en conjunto de entrenamiento y test, nos olvidaremos completamente del conjunto de test durante un rato, y se procede al siguiente paso:

- Ahora, centrándome únicamente en el conjunto de entrenamiento, realizo una **validación cruzada exhaustiva** (adjunto el enlace al objeto que utilizo de la librería **Scikit-Learn (Sklearn)**, en **Python**.

Definición 3.1 (Validación Cruzada Exhaustiva)**.** *Entenderemos por* ***Validación Cruzada Exhaustiva*** *una validación cruzada K-fold (2.13.2) para cada combinación de hiperparámetros. Devolviendo el mejor modelo (con la mejor combinación de hiperparámetros, y validado con la validación cruzada). La* ***consecuencia*** *será la obtención de un modelo validado internamente, y por tanto a la hora de probarlo en el conjunto test, no debería variar mucho de los resultados obtenidos en la validación cruzada.*

Esto suena un poco lioso al decirlo de esta manera, pero vamos a proceder a desmenuzarlo poco a poco, y lo haremos con un ejemplo.

Los **modelos** como bien sabemos, se cargan como **objetos** (que tienen propiedades, parámetros, métodos, ...) (todo esto se presupone que el lector lo conoce), y al entrenarlos, obtenemos esos parámetros y entonces podemos usar sus métodos, como la realización de predicciones, etc.

- Por ejemplo, si consideramos el modelo **Regresión Logística** (LogisticRegression en Sklearn), tenemos los siguientes parámetros:
  - penalty (penalización/ regularización) (l1, l2, elasticnet, ninguna), por defecto, l2.
  - dual (opción para regularización) (booleano), por defecto, false.
  - C (parámetro de regularización)
  - Class Weight (peso de las clases) (balanceado, personalizado o ninguna), por defecto ninguna.
  - etc...

Como hemos podido observar, cada modelo tiene distintos parámetros y cada parámetro tiene distintas opciones disponibles. Los parámetros más importantes de la regresión logística son: **penalty, C y Class Weight**.

```
class sklearn.linear_model.LogisticRegression(penalty='l2',
    *, dual=False, tol=0.0001, C=1.0, fit_intercept=True,
   intercept_scaling=1, class_weight=None, random_state=
   None, solver='lbfgs', max_iter=100, multi_class='auto',
   verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)
```

Se puede apreciar la cantidad de parámetros que tienen estos modelos, y no es necesario tampoco conocer cómo funcionan la mayoría de ellos, pero sí los más importantes. Ya que es vital conocer cómo funciona el modelo y sus parámetros (cómo influyen) para poder ajustarlo correctamente.

Imaginemos por comodidad que queremos obtener el modelo con los hiperparámetros **C** y **Class Weight** mejor ajustados a los datos. Entonces el objeto GridSearchCV, probará todas las combinaciones de parámetros posibles y hará una validación cruzada con cada una, devolviéndonos la mejor.

Es decir, si tenemos las siguientes listas de candidatos a hiperparámetros:

```
from itertools import product

C, class_weight, = [0.1, 1, 10, 100], [None, 'balanced']

list(product(C, class_weight))
```

Lo que nos devuelve el producto cartesiano de dichas listas, o lo que es lo mismo, todas las posibles combinaciones:

```
>>> (0.1, None)
>>> (0.1, 'balanced')
>>> (1, None)
>>> (1, 'balanced')
>>> (10, None)
>>> (10, 'balanced')
>>> (100, None)
>>> (100, 'balanced')
```

El problema está en que para cada una de esas combinaciones, el modelo tendrá que realizar una validación cruzada sobre los datos de training. La validación cruzada tiene un coste computacional bastante alto, aquí, tendremos que hacer un mínimo de 8 validaciones cruzadas. Pero se puede intuir el peligro que tienen estas combinaciones para un modelo más complejo.

Lo bueno es que el modelo devuelto, a parte de ser óptimo en parámetros e hiperparámetros (entre los aportados), está validado internamente, y se puede comprobar ello en el conjunto test. Si fuese necesario, se puede repetir todo el entrenamiento, usando toda la muestra como si fuese entrenamiento, una vez de ha hecho este estudio.

Veamos esto de manera práctica, lo primero serán las librerías que debemos utilizar:

```
    # Para usar los Nucleos de la CPU
    # ================================
    import multiprocessing

    # Validacion Modelo
    from sklearn.model_selection import GridSearchCV,
       train_test_split, StratifiedKFold
```

Ahora, procedemos a hacer la división en los conjuntos de train y test.

```
    X_train, X_test, y_train, y_test = train_test_split(
    data.drop('EXITUS', axis='columns'),
    data['EXITUS'],
    train_size=0.85,
    random_state=44,
    shuffle=True,
    stratify=data['EXITUS']
)
```

La función train_test_split es una función esencial en la vida de un Data Scientist. Es importante estudiar su documentación, ya que tiene una gran cantidad de parámetros.

Según el código indicado previamente, vamos a obtener 4 conjuntos de datos, **Train y sus etiquetas**, **Test y sus etiquetas**. Realizando esta división en 4 conjuntos, nos aseguramos de que el modelo no ve la variable objetivo.

Esta división será de un 85% - 15%, la variable objetivo es **EXITUS**, la semilla de aleatoriedad es 44, le decimos que realice un shuffle para evitar sesgos de ordenación y que nos devuelva particiones estratificadas por la variable objetivo.

```
param_grid = {
        'C': [10e-3, 10e-2, 10e-1, 1, 10, 100, 1000],
        'class_weight'   : ['balanced']
             }

grid = GridSearchCV(
        estimator  = LogisticRegression(), # modelo a entrenar
        param_grid = param_grid, # combinaciones de parametros
        scoring    = 'f1', # metrica a optimizar
        n_jobs     = multiprocessing.cpu_count() - 1, # CPU
        cv         = 5, # numero de la K en K-fold CV
        refit      = True, # Volver a reentrenar una vez hecho
           todo
        verbose    = 0, # No devuelva informacion
        return_train_score = True # Devuelva metricas de
           entrenamiento
       )
```

Una vez que tenemos el objeto **grid** creado, procedemos a su entrenamiento.[2]

```
grid.fit(X = X_train, y = y_train) # Entrenamiento del modelo.

modelo_final = grid.best_estimator_ # Para quedarnos con el
   mejor modelo dentro de modelo_final
```

Una vez obtenido el mejor modelo, queremos realizar ciertas comprobaciones, veamos cómo se comporta el modelo en el conjunto de entrenamiento frente a test:

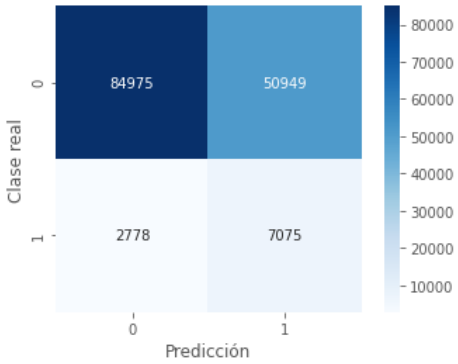


Figura 10: Matriz confusión entrenamiento

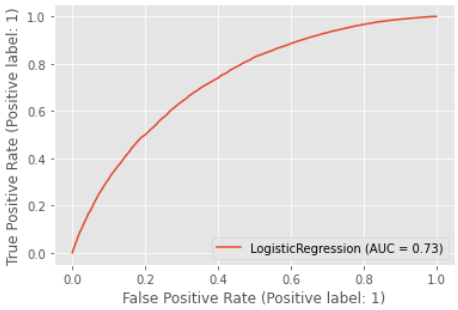


Figura 11: Curva ROC entrenamiento

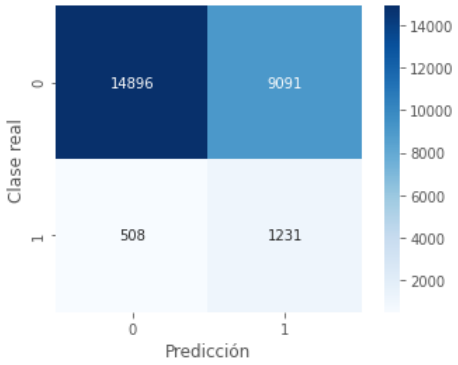


Figura 12: Matriz confusión test

Figura 13: Curva ROC test

Los resultados son los que esperábamos en parte, pero ahora toca ver cómo se comporta el modelo con algunas métricas. Veamos cómo podemos hacer esto.

[2]Nótese que este entrenamiento es sobre el conjunto Train, y ya nos olvidamos del conjunto test.

```
scoring = ["accuracy", "balanced_accuracy", "f1", "roc_auc", '
   precision','recall']          #creamos una lista con todas
   las metricas a comprobar

index = []
scores = {"Accuracy": [], "Balanced accuracy": [], "F1-Score":
   [], "AUROC":[], "Precision":[], "Recall":[]}

#La idea es crear un diccionario de Python e ir rellenandolo
   con el objeto cross_validate, de esta manera podremos
   hacernos una idea exacta de como se esta comportando el
   modelo en el conjunto de test.

index += ['Balanced Logistic Regression']

cv_result = cross_validate(modelo_final, X_test, y_test,
   scoring=scoring)

scores["Accuracy"].append(cv_result["test_accuracy"].mean())
scores["Balanced accuracy"].append(cv_result["
   test_balanced_accuracy"].mean())
scores["F1-Score"].append(cv_result["test_f1"].mean())
scores["AUROC"].append(cv_result["test_roc_auc"].mean())
scores["Precision"].append(cv_result["test_precision"].mean())
scores["Recall"].append(cv_result["test_recall"].mean())

df_scores = pd.DataFrame(scores, index=index)
```

Obteniendo la siguiente tabla:

	Acc	Balanced acc	F1-Score	AUROC	Precision	Recall
Balanced Log Reg	0.6242	0.6646	0.2038	0.7245	0.1189	0.7113

Y si vemos las comparaciones directamente en una tabla:

	Accuracy	Recall	Precision	F1
Train	0.631444	0.718055	0.121932	0.208465
Test	0.626876	0.707878	0.119260	0.204129

Por tanto, se puede afirmar que el modelo está entrenado y validado correctamente.

4 Guardar un Modelo

Para guardar un modelo, aprenderemos a hacer uso de dos cosas:

- Contextos.
- Serializado a binario de Python (Pickle).

4.1 Contextos

Un contexto permite asignar o liberar recursos de una forma expresa. El ejemplo más usado es el **with**. Imagínate que tienes dos operaciones relacionadas que te gustaría ejecutar con un determinado código de por medio.

Cuando abrimos un archivo, el archivo se queda abierto, y es necesario realizar una operación que lo cierre:

```
file = open('fichero', 'w')

try:
    file.write('Hola!')

finally:
    file.close()
```

Los gestores de contexto te permiten hacer precisamente esto, pero de manera mucho más compacta. Veamos un ejemplo:

```
with open('fichero', 'w') as opened_file: #abrir en modo write
   (escritura)
opened_file.write('Hola!')
```

El binario de Python, es el formato Pickle. Permite ahorrar espacio y acaba haciendo el código mucho más limpio, con el tiempo ahorraremos también mucho tiempo.

```
import pickle

todo = ['write blog post', 'reply to email', 'read in a book']
pickle_file = file('todo.pickle', 'w') #abro un archivo pickle
   vacio
pickle.dump(todo, pickle_file) #meto la lista "todo" en el
   archivo pickle
```

Ahora, unificamos estos dos conceptos que acabamos de ver, para entender realmente el interés del asunto.

```
import os
import pickle

# Si no se ha ejecutado el entrenamiento todavia, realiza
   entrenamiento y guarda modelo.
if not os.path.isfile(os.getcwd() + '\\modelo_final.pkl'):

    from sklearn.svm import SVC

    svm_model = SVC(probability=True);

    parameters = {}
    parameters['C'] = [10e-2, 1, 100]
    parameters['kernel'] = ['linear', 'rbf']

    GS = GridSearchCV(svm_model, param_grid=parameters, scoring
       ='f1', cv=5 ,refit = True)
    GS.fit(X_train, y_train)

    print("Mejor score: ", GS.best_score_)
    print("Mejore configuracion de parametros: ", GS.
       best_params_)

    modelo_final = GS.best_estimator_

    y_new_pred = modelo_final.predict(X_test)
    show_results(y_test, y_new_pred)

    with open(os.getcwd() + '\\modelo_final.pkl',"wb") as
       model_data_file:
        pickle.dump(modelo_final, model_data_file)

# Si se habia entrenado previamente, y el archivo esta
   disponible, lo lee sin entrenar.
else:
    with open(os.getcwd() + '\\modelo_final.pkl','rb') as
       model_data_file:
        modelo_final = pickle.load(model_data_file)
```

Este snippet de código viene a decirnos lo siguiente:

- Si se ha entrenado el modelo previamente, léelo.
- En caso de que no se encuentre el archivo precisado (el binario del modelo), entrena el modelo y guárdalo en un archivo.

Esto es muy importante, ya que modelos como los Support Vector Machine, requieren una gran capacidad de computación, y eso conlleva una demora considerable. No querríamos entrenar el modelo por error más de una vez.

5 Evaluación de clasificadores

5.1 ¿Es adecuado usar siempre la tasa de acierto?

La evaluación es sensible a la distribución de las clases:

- A veces la distribución de las clases en los datos no está equilibrada. (imbalanced data). Esto quiere decir que, si el tamaño del conjunto de datos es m, y el número de clases es c, no tiene por qué haber $\frac{m}{c}$ instancias de cada clase.
- Esto puede suponer un problema en la evaluación.

5.1.1 Ejemplo

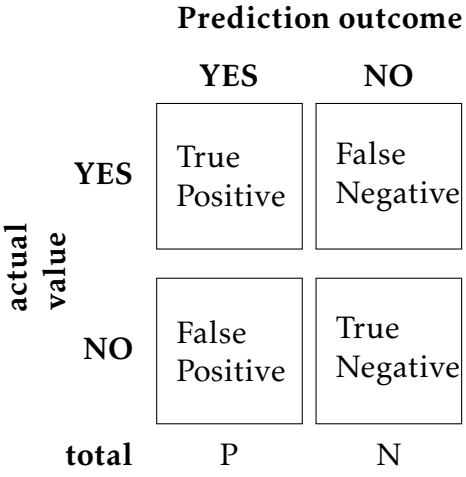
Se han estudiado 10000 pacientes, de los cuales 9900 están sanos y 100 tienen una enfermedad grave.

Un clasificador C_1 predice que todos los pacientes están sanos.

La tasa de acierto de dicho clasificador es de $\frac{9900}{10000} = 99\,\%$

¿Es C_1 un buen clasificador?

Tenemos que echar mano de la matriz de confusión:



- Tasa de acierto (Accuracy): $accuracy = \frac{TP+TN}{P+N}$
- Tasa de verdaderos positivos TPR (sensitivity o recall): $TPR = \frac{TP}{TP+FN} = \frac{TP}{P}$
- Tasa de verdaderos negativos TNR (specificity): $TNR = \frac{TN}{TN+FP} = \frac{TN}{N}$
- Tasa de falsos positivos FPR (negative error): $FPR = \frac{FP}{TN+FP} = \frac{FP}{N}$
- Valor positivo predictivo (precision): $PPV = \frac{TP}{TP+FP}$
- Macromedia: $MM = \frac{TPR+TNR}{2}$

Se cumple que: $FPR = (1 - specificity)$

Cuando una clase aparece con poca frecuencia, se suelen usar los índices *precision* y *recall* para resumir los resultados del modelo.

- **Precision** ($\frac{TP}{TP+FP}$): De todos los casos clasificados como positivos, qué fracción de ellos lo son realmente.
  - Qué fracción de los pacientes diagnosticados como enfermos lo están realmente.
- **Recall** ($\frac{TP}{TP+FN}$): Qué fracción de casos positivos se ha detectado.
  - Qué fracción de pacientes enfermos se han clasificado como tales.

5.2 Balance entre precision y recall

La regresión logística devuelve $h_\theta(x) \in [0,1]$, que es la probabilidad de que un caso sea positivo.

Por defecto (salvo que se modifique), se clasifica como positivo el caso con probabilidad > 0.5, lo que se llama $umbral = 0.5$, pero se puede modificar.

Por tanto, si solamente se quiere predecir $y = 1$ cuando se está muy seguro:

- Umbral muy alto.
- Más precision.
- Menor recall.

Si lo que se busca es evitar falsos negativos:

- Se baja el umbral.
- El recall sería más alto.
- La precision sería más baja

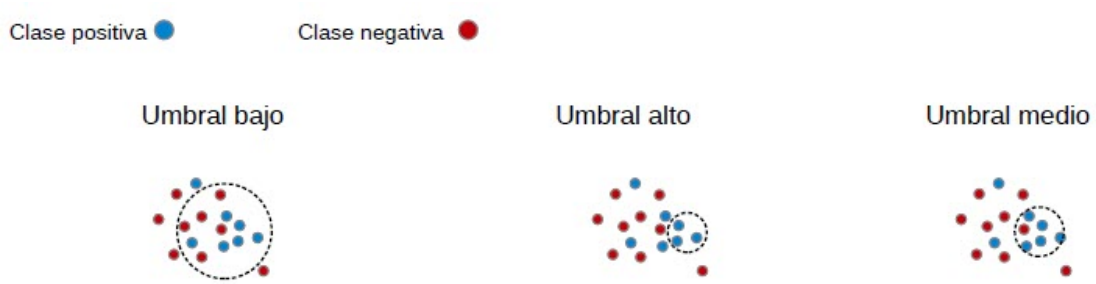


Figura 14: Umbrales

¿Cómo comparamos entonces los índices de *precision* y *recall*? Veamos qué ocurre en el siguiente ejemplo:

	Precision	Recall	Media	F_1 Score
Algoritmo 1	0.5	0.4	0.45	0.44
Algoritmo 2	0.7	0.1	0.4	0.175
Algoritmo 3	0.02	1.0	0.51	0.0392

En el ejemplo, la mejor media se da para un algoritmo con recall = 1 y precision $\approx$ 0, que no funciona bien porque predice casi todo positivo.

$$\boxed{F_1 = 2\frac{P.R}{P+R}}$$

F_1 - Score Oscila entre 0 (si $P = 0$ o $R = 0$ y 1 (si $P = 1$ y $R = 1$).

5.3 *Análisis ROC*

En el Análisis ROC (Receiver Operating Characteristics) se normaliza la matriz de confusión por columnas.

El clasificador se representa como un punto en el espacio bidimensional dado por la tasa de verdaderos y falsos positivos. De la siguiente tabla:

	sí	no
sí	50	1000
no	50	8900
Total	100	9900

Normalizamos y pasamos a la siguiente:

	sí	no
sí	TPR = 0.5	FPR = 0.101
no	FNR = 0.5	TNR = 0.899
Total	1	1

Si para esta tabla el umbral es de 0.5, ya hemos obtenido el primer punto de la curva ROC, que se corresponde con:

$$(FPR,\ TPR) = (0.101,\ 0.5)$$

Si hacemos variar el umbral entre todos los valores posibles y repetimos esta operación, obteniendo todos los puntos de la curva, nos queda lo siguiente:

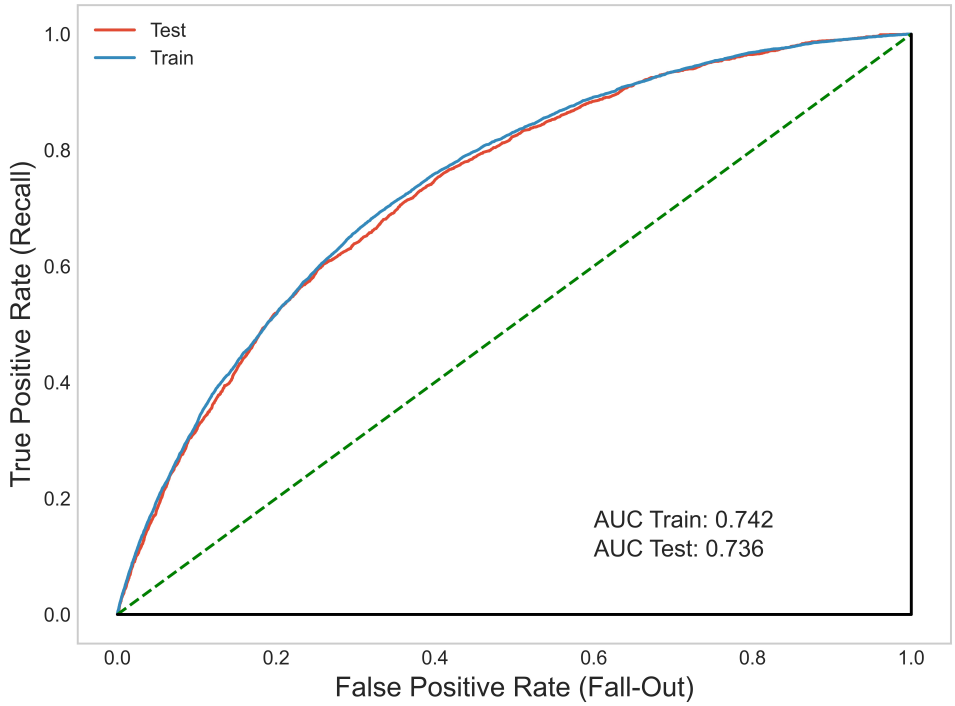


Figura 15: Curvas ROC

La curva ROC representa el rendimiento de clasificadores que utilizan score+umbral (Ej. regresión logística). Cada punto representa el rendimiento con un umbral. Se utilizan como umbrales los score correspondientes a cada caso del conjunto de entrenamiento. El área bajo la curva ROC (AUC) de un modelo de clasificación corresponde al área debajo de la curva ROC que va desde los puntos (0,0) a (1,1).

- Puede interpretarse como la probabilidad de que el score que el modelo asigna a un ejemplo positivo sea superior al asignado a un ejemplo un negativo.

6 *Evaluación sensible al coste*

¿De los siguientes clasificadores, cuál es mejor? (Real(R) vs Predicción(P))

Array 1:

C_1	sí (R)	no (R)
sí (p)	0	0
no (p)	100	9900

Array 2:

C_2	sí (R)	no (R)
sí (p)	50	1000
no (p)	50	8900

Clasificador	C1	C2
Métrica	-	-
ACC	99%	89.5%
TPR	0%	50%
TNR	100%	89.5%
FPR	0%	10.1%
PPV	indefinido	9.52%
MM	50%	94.6%

En muchas ocasiones, los errores producidos no tienen la misma importancia, es más grave clasificar como sano a un paciente enfermo que clasificar como enfermo a un paciente sano.

Cada uno de los tipos de errores (FP y FN) puede llevar un coste asociado:

Coste	sí	no
sí	0	100$
no	5000$	0

Aquí el objetivo será obtener el clasificador que minimice el coste.

7 Conclusiones

- El verdadero rendimiento de un modelo de aprendizaje supervisado se evalúa sobre datos no utilizados sobre el aprendizaje.
- Si no es posible reservar datos para la evaluación, se puede utilizar validación cruzada.
- Un mayor número de datos para el entrenamiento disminuye el error.
- El error puede deberse a subajuste (demasiado sesgo) o sobreajuste (demasiada varianza).
- Un aumento de la complejidad del modelo puede repercutir positivamente(eliminación de sesgo/subajuste) o negativamente (aumento de la varianza/sobreajuste).
- Las curvas de aprendizaje y validación permiten determinar el origen del error y cómo mejorar el rendimiento del modelo.