U01 PRÁCTICA 1

EL PROCESO DE DESARROLLO DE UN SISTEMA DE MACHINE LEARNING

SISTEMAS DE APRENDIZAJE AUTOMÁTICO

IES SERRA PERENXISA
TORRENT (VALENCIA)

ANALIZAR DATOS DE DATASETS

OBJETIVOS: Aplicar el uso de las siguientes técnicas:

1. Poner en práctica técnicas de todas las etapas del proceso de desarrollo.
2. Usar información estructurada (Cargar, analizar, manipular).
3. Reducción Univariada de Características (eliminar bajas correlaciones con el target).
4. Reducción de Características Basada en la Colinealidad (por parejas de características).
5. Implementar diferentes modelos.
6. Afinar sus hiperparámetros.
7. Comparar Modelos de Aprendizaje Supervisado (Regresión Logística y GBM), usando datos de test y realizando un ranking de importancia de características.
8. Calcular percentiles bins de cada modelo y calcular el ratio de clases positivas por bin percentil.
9. Introducción a las curvas ROC y Precision-vs-Recall como herramientas de validación.

ELIGE UNA DE ESTAS DOS FORMAS DE ENTREGA (elige la más te interese) :

- **PRIMERA**: Crea un **jupyter notebook** donde respondas a las preguntas y realices los ejercicios de código. Yo te aconsejo esta porque
- **SEGUNDA**: Usa un entorno de ejecución y entrega las respuestas en un documento .pdf y el código en sus propios ficheros. Comprime todo en una sola carpeta antes de subir la actividad.

ACTIVIDAD 1: INVESTIGAR DATASETS DISPONIBLES EN SCIKIT-LEARN.

Ve a la página oficial de la librería *scikit-learn* (clic en este enlace) y en la opción de menú ***"User Guide"*** aparece otro menú en un panel a la izquierda, haz clic en el apartado ***"7. DatasetLoading Utilities."*** y explora y lee un poco el contenido que aparece y responde a estas cuestiones.

ENTREGA 1: Responde a estas preguntas:

a) ¿Cómo se llama el paquete donde se encuentran los datasets que tiene la librería?

b) ¿Qué características tienen los ***Toy dataset***?

c) ¿Qué otros tipos de datasets tiene *scikit-learn*?

d) Busca información del dataset de tipo *Toy* llamado Iris. Responde a estas preguntas sobre él:

- Escribe las sentencias con las que se cargaría el dataset Iris en un programa en Python.
- ¿Cuántos ejemplos tiene?
- ¿Para qué tipo de problemas se puede utilizar?
- ¿Cuántas clases diferentes tiene y cómo se llaman?
- ¿Qué porcentaje de datos o ejemplos de cada clase hay?

e) Escribe las sentencias con las que:

- Descargarías un dataset de *openml.org* (mira el apartado 7.4.1).
- Haz un programa en Python que descargue uno e imprima:
  - Cuántos ejemplos tiene.
  - Cuántas características tiene cada ejemplo
  - Imprime los datos de los primeros 5 ejemplos.

ACTIVIDAD 2: ANALIZAR DATASET "BREAST CANCER WISCONSIN".

Las características de cada ejemplo se han calculado a partir de una imagen digitalizada de una masa tumoral de cáncer de mama. Describen características de la zona tumoral de la imagen. De cada característica se añade la media *'mean … '*, el error estándar *'standard error …'* y el `'worst …'` o media de los 3 valores mayores, generando un total de 30 características. Por ejemplo las mediciones del radio de la masa son: '*mean radius*' (radio medio), '*standard error of the radius*' (error estándar del radio) y '*worst radius*' (mayor tri-media). Todas las características se registran con 4 dígitos significativos. La columna target tiene dos clases (dos posibles valores) que corresponden a las etiquetas negativa (*"Benigno"*) y positivo (*"Maligno"*). Recordaros que maligno en el contexto médico equivale a que conduce a la muerte. Para realizar las tareas necesitarás todos estos paquetes que puedes importar ya o a medida que los vayas necesitando, como prefieras:

```
import os
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from collections import OrderedDict

from sklearn import datasets
from sklearn.preprocessing import label_binarize, LabelBinarizer
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import precision_recall_curve
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc

DISPLAY_PRECISION = 4
pd.set_option("display.precision", DISPLAY_PRECISION)
```

Paso 2.1. Cargamos el dataset de *sklearn* y describimos las características. Ten en cuenta que el dataset es un diccionario Python con varias claves y en cada clave tendrá algo. Primero miramos las claves.

```
# Imprimimos las descripciones
datos = datasets.load_breast_cancer()
print(datos.DESCR)    # detailed description
print("---- keys del dataset \"breast cancer\":", datos.keys() )
```

Y a continuación miramos las proporciones de cada clase, creamos un *DataFrame* de *Pandas* y se mapea target a valores 0 y 1. Una vez construido el *DataFrame* lo describimos.

```
# Debemos Remapear el target para que aparezcan valores numéricos:
#   Benigno = 0 (clase negativa)
#   Maligno = 1 (clase positiva)
li_clases = [datos.target_names[1], datos.target_names[0]]
li_target = [1 if x==0 else 0 for x in list(datos.target)]
li_predictoras = list(datos.feature_names)
print("---- El target tuene 2 clases:", li_clases)
print(f"---- Su distribución en un total de {len(li_target)} valores es:")
print(pd.Series(li_target).value_counts())
df_todas = pd.DataFrame(datos.data[:,:], columns=li_predictoras)
print("Describimos las 6 primeras columnas del dataframe:")
print(df_todas.iloc[:,:6].describe().to_string())
```

ENTREGA 2: Responde a estas preguntas:

a) ¿Qué hace la sentencia? **`df_todas.iloc[:, :6]`**

b) ¿Hay valores ausentes en alguna columna?

c) ¿Hay el mismo número de casos benignos que malignos? ¿Cuál es el % de cada uno?

Paso 2.2. Dividimos el dataset en datos de train y datos de test:

```
RATIO_PARA_TEST = 0.5          # dos partes iguales
X = df_todas
y = pd.Series(li_target)
X_train_0, X_test_0, y_train, y_test = train_test_split(X, y, test_size=RATIO_PARA_TEST,
                                                        random_state=0)
print(f"X_train_0.shape {X_train_0.shape} e y_train.shape {y_train.shape}")
print(f"X_test_0.shape  {X_test_0.shape} e  y_test.shape  {y_test.shape}")
```

ACTIVIDAD 3: REDUCIR CARACTERÍSTICAS.

Usamos dos técnicas del método ***selección de características***:

1. Reducción Univariada (Eliminar bajas correlaciones con el target).
2. Reducción basada colinearidad (para cada pareja de predictoras, si tienen mucha correlación, nos quedamos con la más correlacionada con el target y quitamos la otra).

Paso 3.1. Primero hacemos un mapa de calor de las correlaciones:

```
def matriz_correlaciones(y, X, se_dibuja=False):
  # Calcula y dibuja la matriz de correlaciones
  # Devuelve:
  # yX - datos concatenado
  # yX_corr - matriz de correlaciones, correlación de Person entre [-1, +1]
  # yX_abs_corr - matriz de correlaciones en valor absoluto
  yX = pd.concat([y, X], axis=1)
  yX = yX.rename(columns={0: 'TARGET'}) # cambiar nombre de primera columna
  # Calcula la matriz y convierte a Dataframe para visualizar mejor
  yX_corr = yX.corr(method='pearson')
  yX_abs_corr = np.abs(yX_corr)  # Convierte a valor absoluto
  if se_dibuja:
      plt.figure(figsize=(10, 10))
      plt.imshow(yX_abs_corr, cmap='RdYlGn', interpolation='none', aspect='auto')
      plt.colorbar()
      plt.xticks(range(len(yX_abs_corr)), yX_abs_corr.columns, rotation='vertical')
      plt.yticks(range(len(yX_abs_corr)), yX_abs_corr.columns);
      plt.suptitle('Mapa de Calor de Correlación (valor absoluto)', fontsize=13,
                   fontweight='bold')
      plt.show()
  return yX, yX_corr, yX_abs_corr

# Calcular y visualizar la matriz de correlación de los datos de entrenamiento
yX, yX_corr, yX_abs_corr = matriz_correlaciones(y_train, X_train_0, se_dibuja=True)
```

ENTREGA 3: Responde a estas preguntas:

1. Entrega el gráfico de la matriz de correlaciones

2. Una columna influye en el target si su correlación está cerca del +1 o del -1. Como hemos dibujado el valor absoluto en los datos, solo puede estar cera del +1 si tiene influencia o del 0 si no la tiene. Si una columna tiene baja correlación con el target, significa que tienen poca relación y por tanto no tendrá apenas influencia sobre él, no le afecta. Por tanto eliminarla no supone una gran pérdida de información para los algoritmos de aprendizaje. Examinando visualmente la matriz ¿Podrías indicar los nombres de las características que están en ese caso, es decir, tienen una correlación cercana a cero con el target.

3. Examinando visualmente la matriz, ¿es simétrica respecto a la diagonal? Es decir (*caracteristica1* influye en *característica2*) lo mismo que (*característica2* en *característica1*)?

4. ¿Qué característica guarda una fuerte correlación con *"mean radius"* y *"mean perimeter"*?

Paso 3.2. Aplicamos reducción univariada:

```
# Reducción univariada
CORRELACION_MIN = 0.1
# Ordenamos características por su correlación con el target
s_corr_target = yX_abs_corr['TARGET']
s_corr_target_sort = s_corr_target.sort_values(ascending=False)
# Solo usamos columnas con correlación > 0.1
s_low_correlation_ftrs = s_corr_target_sort[s_corr_target_sort <= CORRELACION_MIN]
print(f"Eliminadas{ len(s_low_correlation_ftrs)} predictoras:")
for i,v in enumerate(s_low_correlation_ftrs):
    print(f"  {i:3d}, {np.round(v, DISPLAY_PRECISION)}, {s_low_correlation_ftrs.inde
s_corr_target_sort = s_corr_target_sort[s_corr_target_sort > CORRELACION_MIN]
print(f"Quedan {len(s_corr_target_sort)-1} características correlacionadas:")
for i,v in enumerate(s_corr_target_sort):
    ftr = s_corr_target_sort.index[i]
    if ftr == 'TARGET':
        continue
    print(f"  {i:3d}, {np.round(v, DISPLAY_PRECISION)}, {ftr}")
```

Paso 3.3. Reducción basada en colinealidad:

```
# Eliminar a los de baja correlacióndel anterior
li_X1_cols = list(set(s_corr_target_sort.index) - set(s_low_correlation_ftrs.index)
li_X1_cols.remove('TARGET')
# Colinearidad
CORRELACION_MAX = 0.8
X1 = X_train_0[li_X1_cols]
yX1, yX_corr1, yX_abs_corr1 = matriz_correlaciones(y_train, X1, se_dibuja=False)
# Obtener todas las parejas de predictoras
Xcorr1 = yX_abs_corr1.iloc[1:,1:]
s_parejas = Xcorr1.unstack()
print("s_parejas.shape", s_parejas.shape)
s_parejas = np.round(s_parejas, decimals=DISPLAY_PRECISION)
s_parejas_sorted = s_parejas.sort_values(ascending=False)
s_parejas_sorted = s_parejas_sorted[(s_parejas_sorted != 1) &
                                    (s_parejas_sorted > CORRELACION_MAX)]
# Convertir una lista de nombres
li_corr_parejas = s_parejas_sorted.index.tolist()
print("len(li_corr_parejas):", len(li_corr_parejas))
print("li_corr_parejas[:10]", li_corr_parejas[:10])
```

Paso 3.4. Para cada pareja con alta colinealidad, eliminar la de menos correlación con el target:

```
# Calcular lista de predictoras a eliminar
li_remove_parejas = []
li_remove_puntuaciones = []
for tup in li_corr_parejas:
    s0 = s_corr_target_sort.loc[tup[0]]
    s1 = s_corr_target_sort.loc[tup[1]]
    remove_predictora = tup[1] if s1 < s0 else tup[0] # obtener la < con target
    if remove_predictora not in li_remove_parejas:
        li_remove_parejas.append(remove_predictora)
        di = {'var_0':tup[0], 'var_1':tup[1], 'coef_0':s0, 'coef_1':s1,
              'PREDICTORA_A_QUITAR':remove_predictora}
        li_remove_puntuaciones.append(OrderedDict(di))
df_remove_puntuaciones = pd.DataFrame(li_remove_puntuaciones)
```

```
# Eliminar predictoras encotradas
print("Eliminando %d predictoras (ver última columna):" % len(li_remove_parejas))
print(df_remove_puntuaciones.to_string())
li_X2_cols = list(set(li_X1_cols) - set(li_remove_parejas))
li_X2_cols.sort()
print("==== Quedan %d predictoras:" % (len(li_X2_cols)))
for i,v in enumerate(s_corr_target_sort):
    ftr = s_corr_target_sort.index[i]
    if ftr in li_X2_cols:
        print(i, np.round(v, DISPLAY_PRECISION), ftr)
```

ENTREGA 4: Responde a esta pregunta (puedes ejecutar el código para ayudarte):

a) Cuando tenemos dos columnas predictoras con una capacidad predictiva muy parecida, por ejemplo *"mean radius"* se correlaciona con el target en 0.7108 y la predictora *"perimeter"* se correlaciona con target 0.7240. ¿Estas dos columnas están correlacionadas entre sí?

b) Si es el caso y queremos eliminar una ¿Cuál se debe eliminar?

Paso 3.5. Dibujar de nuevo la matriz de correlación para las columnas que quedan:

```
# Calcula la matriz de correlaciones
X2 = X1[li_X2_cols]
print("==== Tras la reducción con parejas X2.shape:", X2.shape)
yX2, yX_corr2, yX_abs_corr2 = matriz_correlaciones(y_train, X2)
s_X3_cols = yX_abs_corr2['TARGET'].sort_values(ascending=False)
li_X3_cols = s_X3_cols.index.tolist()
print("Las Remaining features:")
print(s_X3_cols)
print("---")
li_X3_cols.remove('TARGET')
X3 = X2[li_X3_cols]
print("Tras la reducción con parejas X3.shape:", X3.shape)
yX3, yX_corr3, yX_abs_corr3 = matriz_correlaciones(y_train, X3, se_dibuja=True)
X_train = X3
X_test = X_test_0[li_X3_cols]
```

Paso 3.6. Comprobar con un gráfico una de las predictoras que quedan con el target. La predictora '*worst concave points*' tiene una correlación con el target de 0.81 y en el gráfico que generamos vemos como hay una buena separación de puntos entre la clase negativa del target (abajo *TARGET=0*) y la clase positiva (arriba *TARGET=1*).

```
sns.jointplot(yX3, x='worst concave points', y='TARGET', kind='scatter',
              marginal_kws=dict(bins=12, rug=True))
plt.suptitle('Ejemplo de matriz de parejas')
```

Paso 3.7. Dibujar la matriz de gráficos *scatter* de todas las parejas. La matriz de gráficos *scatter* ayuda a comprender las relaciones que mantienen ente ellas las características. Los puntos azules representan target negativos y los verdes target positivos. Los gráficos de la diagonal contienen los histogramas de cada característica.

```
mapa_colores = {0: '#0392cf', 1: '#7bc043'} # 0(clase negativa):blue, 1(positiva):green
colores = yX3['TARGET'].map(lambda x: mapa_colores.get(x))
pd.plotting.scatter_matrix(yX3, alpha=0.5, color=colores, figsize=(12,14),
                           diagonal='hist', hist_kwds={'bins':12})
plt.suptitle('Matriz Scatter de target y predictoras')
plt.show()
```

ENTREGA 5: Entrega la matriz de gráficos scatter y responde:

a) Si estas son las predictoras más influyentes en el target ¿Qué porcentaje son de las 30 que teníamos inicialmente?

b) Entrega el gráfico.

ACTIVIDAD 4. CREAR DOS MODELOS DE CLASIFICACIÓN.

Clasificar los datos usando dos diferentes tipos de modelos:

1. Modelo de Regresión Logística.
2. GBM – Modelo Boosting por Gradiente

Paso 4.1. Comprobar el balanceo de los datos. Algunos modelos no se entrenan bien si los datos no están balanceados, es decir, si no hay aproximadamente la misma cantidad de valores positivos y negativos en el target. Tampoco puede medirse bien el funcionamiento del modelo si los datos de test no están balanceados. Pero incluso si al modelo y a sus algoritmos de entrenamiento no les afecta, en los datos de entrenamiento debería haber unas proporciones similares de cada clase tanto en los datos de train como de test. Haz la comprobación con este código:

```
print(f"==== X_train.shape: {X_train.shape}, y_train.shape: {y_train.shape}")
## Distribución de clases de TARGET en train y test
val_cnts = y_train.value_counts()
print("Distribución de ejemplos positivos y negativos en train:")
print(val_cnts)
print("Porcentaje de positivos: %s" % "%2f%%" % (100 * val_cnts[1] / len(y_train)))
print(f"==== X_test.shape: {X_test.shape} y_test.shape: {y_test.shape}")
val_cnts = y_test.value_counts()
print("Distribución de ejemplos positivos y negativos en test:")
print(val_cnts)
print("Porcentaje de positivos: %s" % "%2f%%" % (100 * val_cnts[1] / len(y_test)))
```

ENTREGA 6: A la vista de los resultados:

a) ¿Los datos están balanceados?

b) ¿Es lógico porque se corresponde con lo que puedes encontrar en la vida real?

c) Las proporciones de cada clase en train y test ¿Son similares?

d) Si necesitamos datasets muy balanceados se lo podemos pedir a las funciones que particionan los datos. ¿Cómo se llama la técnica que lo realiza? (marca la opción correcta):

( ) Estratificación ( ) Validación cruzada. ( ) Ingeniería de características ( ) ADAM

Paso 4.2. Dibujar un mapa de calor para el proceso *gridsearch* de un modelo. Cada modelo tiene sus propios hiperparámetros, ajustarlos es otra tarea del proceso de desarrollo de un sistema ML. Vamos a usar *gridsearch* para hacerlo y creamos una función para dibujar como funciona el modelo bajo cada posible juego de hiperparámetros.

```
def plot_2d_grid_search_heatmap(grid_search, grid_params, x_param, y_param, is_verbose=True):
    '''
    Dibuja un mapa de calor 2D del proceso gridsearch
    Parámetros:
        grid_search: instancia de un objeto sklearn.GridSearchCV
        grid_params: diccionario de parámetros grid search
        x_param: nombre del parámetro eje-x del grid_params
        y_param: nombre del parámetro eje-y del grid_params
        is_verbose (opcional): imprime resultados
    Return:
        grid_search.best_score_: mejor puntuación encontrada
        grid_search.best_estimator_: mejor estimador encontrado
    '''
    grid_params_x = grid_params[x_param]
    grid_params_y = grid_params[y_param]
    df_resul = pd.DataFrame(grid_search.cv_results_)
    ar_scores = np.array(df_resul.mean_test_score).reshape(len(grid_params_y),
                                                           len(grid_params_x))
    sns.heatmap(ar_scores, annot=True, fmt='.3f', xticklabels=grid_params_x,
                yticklabels=grid_params_y)
    plt.suptitle('Mapa de Calor Grid Search')
    plt.xlabel(x_param)
    plt.ylabel(y_param)
    if is_verbose:
        print("\n==== grid_search.best_score_:")
        print(grid_search.best_score_)
        print()
        print("grid_search.best_estimator_:")
        print(grid_search.best_estimator_)
    plt.show()
    return grid_search.best_score_, grid_search.best_estimator_
```

Paso 4.3. Implementar Clasificador con Regresión Logística. Este modelo es sensible a la presencia de *outliers* y a las escalas diferentes de cada predictora.

Un paso previo al entrenamiento del modelo sería solucionar previamente estos problemas de los datos si es que los tienen. En el caso de las escalas deberíamos normalizar os datos antes de entrenar, aunque esto tiene el inconveniente de que pierdes significado con la realidad aunque mantengas las propiedades estadísticas.

```
grid_rl = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000], 'penalty': ['l1', 'l2']}
cla_rl = LogisticRegression(class_weight='balanced', dual=False,
                            fit_intercept=True, intercept_scaling=1, max_iter=200,
                            n_jobs=1, random_state=0, tol=0.0001, verbose=0,
                            warm_start=False)
gs_rl = GridSearchCV(cla_rl, grid_rl, return_train_score=True)
# Se realizan por defecto validacion cruzada de 3-fold
gs_rl.fit(X_train, y_train)  # Entrenar el moslo
best_score_rl, cla_rl = plot_2d_grid_search_heatmap(gs_rl, grid_rl, 'C', 'penalty')
print("==== Atributode de Clasificador de Regresión Logística:")
print("  intercept_:", cla_rl.intercept_ )
print("  coef_:", cla_rl.coef_ )
```

En el *heatmap* que genera, el eje ***x*** es del parámetro ***'C'*** de los modelos de Regresión Logística: representa la inversa de la fuerza de la regularización y debe ser un valor flotante positivo. Como es la inversa, a valores más pequeños, mayor fuerza tiene la regularización.

Este modelo usa dos penalizaciones: una de tipo ***ℓ2*** (para los errores al cuadrado) y una ***ℓ1*** (para el valor absoluto de los errores). Solo cuando el hiperparámetro ***'C'*** tiene un alto valor, el ***ℓ1*** y el ***ℓ2*** se vuelven parecidos.

ENTREGA 7: A la vista del mapa de calor de *grid-search* y de los datos mostrados por consola:

a) ¿Qué valor de hace que se obtenga una puntuación más alta (el modelo funciona mejor)?

b) ¿Significa que no se debe usar regularización?

c) Cuando aplicamos restricciones al algoritmo que configura un modelo, elige:

- ( ) Le damos libertad
- ( ) Le quitamos libertad
- ( ) Hacemos que el modelo generalice mejor
- ( ) Hacemos que el modelo generalice peor
- ( ) Es mejor hacerlo cuando el modelo es demasiado complejo y tiene overfitting.
- ( ) Es mejor hacerlo cuando el modelo es demasiado simple y tiene underfitting.

d) ¿Qué parámetros configuran al modelo?

e) ¿Con qué hiperparámetros se han calculado esos parámetros?

Paso 4.3. Implementar el Clasificador *GBM*. A estos clasificadores no les afectan que los datos estén en diferentes escalas porque se basan en árboles a los que las transformaciones lineales no les afecta. En el *grid search* vamos a usar solamente dos hiperparámetros, aunque en la vida real deberíamos probar más.

```
grid_gb = {'min_samples_leaf':[2, 4, 8, 16], 'learning_rate':[0.001, 0.01, 0.1]}
cla_gbm = GradientBoostingClassifier(criterion='friedman_mse',
                                     init=None, loss='log_loss', max_features='sqrt',
                                     max_leaf_nodes=None, max_depth=3, warm_start=False,
                                     min_impurity_decrease=0.0, min_samples_split=2,
                                     min_weight_fraction_leaf=0.0, n_estimators=800,
                                     random_state=0, subsample=1.0, verbose=0)
# por defecto usa 3-fold CV
gs_gb = GridSearchCV(cla_gbm, grid_gb, verbose=0, return_train_score=True)
gs_gb.fit(X_train, y_train)  # Entrenar modelo
best_score_gb, cla_gb = plot_2d_grid_search_heatmap(gs_gb, grid_gb, 'min_samples_leaf',
                                                    'learning_rate')
```

ENTREGA 8: A la vista del mapa de calor de *grid-search*:

a) ¿La mejor puntuación es mejor o peor que la mejor alcanzada por la Regresión Lineal?

b) ¿Y el rango de [mínima puntuación, máxima_puntuación]?

c) Si cambiamos el valor de la constante RATIO_PARA_TEST de 0.5 a 0.2, particionamos el 80% de los datos para train. Hazlo y repite la ejecución. ¿Cómo queda ahora la máxima puntuación? ¿Mejora a la Regresión Lineal? ¿El intervalo [peor, mejor]?

c) ¿Alguno de los dos parámetros no influye cuando se alcanza la mejor puntuación? O dicho de otro modo ¿Hay algún valor óptimo de uno de los dos parámetros?

d) Cuando usamos `GridSearchCV()` también debemos usar la función `Pipeline()` para preprocesar los datos antes de hacer la validación cruzada para entrenar el modelo, algo así como esto: `tuberia_de_operaciones = Pipeline([('nombre_preproceso1', preproceso1), ('nombre_modelo', modelo)])` ¿Usamos esto en el código?

ACTIVIDAD 5. VALIDAR EL MODELO.

La importancia que tiene cada característica (la información que aporta sobre el problema o la influencia que tiene para que el modelo tome sus decisiones) varía de un modelo a otro.

Paso 5.1. Calcular la matriz de confusión de los dos modelos.

```
li_clas = ['cla_rl', 'cla_gb']
dfp = pd.DataFrame(index=['TARGET'], data=[y_test]).T
for s_cla in li_clas:
    print("MODELO: " + s_cla)
    print("--------------")
    cla = eval(s_cla)
    y_pred = cla.predict(X_test).astype(int) # devuelve una clase decisión
    y_score = cla.predict_proba(X_test)      # devuelve el valor de probabilidad
    s_class = "%s_class" % s_cla
    s_proba = "%s_proba" % s_cla
    s_rank = "%s_rank" % s_cla
    dfp[s_class] = y_pred
    dfp[s_proba] = y_score[:,1]
    dfp[s_rank] = dfp[s_proba].rank(ascending=1).astype(int)
    # Imprime la matriz de confusión y el informe de clasificación
    # desde sklearn.metrics importa la matriz de confusión
    # cm = confusion_matrix(y_test, y_pred)
    # Pandas 'crosstab' muestra mejor esta matriz
    cm = pd.crosstab(y_test, y_pred, rownames=['Reality'], colnames=['Predicted'],
                     margins=True)
    print(cm)
```

ENTREGA 9: Una vez ejecutado el código:

a) Entrega una captura de la matriz de confusión del modelo de Regresión Lineal.

b) Lo mismo para el modelo *GBM*.

Paso 5.2. Calcular el informe de importancia de las características de los dos modelos. (Ten en cuenta que el código continúa dentro del bucle y se realizará para cada modelo).

```
    print("\nInforme de Clasificadores:")
    print(classification_report(y_test, y_pred))
    if s_cla == 'cla_rl':
        y_score_lr = y_score.copy()
    elif s_cla == 'cla_gb':
        y_score_gb = y_score.copy()
    else:
        print('Error')
        break
    print(dfp.tail(10).to_string())
```

ENTREGA 10: Una vez ejecutado el código:

a) Entrega una captura del informe del modelo de Regresión Lineal.

b) Lo mismo para el modelo *GBM*.

Paso 5.3. Dividir los datos en Bins percentiles, es decir, definir 10 grupos (bins) uno para cada posible intervalo de scores, y ver cuantos datos caen dentro de cada bin. El rango de puntuaciones estará limitada por '*score_min*' a '*score_max*'.

```
RANGO_BINS = 10
rango_cols = ['i_bin', 'i_min_bin', 'i_max_bin', 'score_min', 'score_max', 'bin_cnt', 'pos_cnt', 'pos_rate']
rango_cols2 = ['i_bin', 'score_min', 'score_max', 'tnr', 'fpr', 'fnr', 'tpr', 'tn', 'fp', 'fn', 'tp']
print(dfp.shape)
len_test = dfp.shape[0]
len_bin = int(len_test / RANGO_BINS)
for s_cla in li_clas:
    i_min_bin = 0
    i_max_bin = 0
    dfr = pd.DataFrame(columns=rango_cols)
    dfr2 = pd.DataFrame(columns=rango_cols2)
    s_class = "%s_class" % s_cla
    s_proba = "%s_proba" % s_cla
    s_rank = "%s_rank" % s_cla
    for i in range(RANGO_BINS):
```

Ahora el código que viene está dentro del bucle interno (**`for i in`** `range(RANGO_BINS)`):

```
        if i == RANGO_BINS - 1:
            i_max_bin = len_test
        else:
            i_max_bin += len_bin
        # Rango usado para cada bin de puntuaciones
        df_rng = dfp[(dfp[s_rank] >= i_min_bin) & (dfp[s_rank] < i_max_bin)]
        score_min = np.min(df_rng[s_proba])
        score_max = np.max(df_rng[s_proba])
        bin_cnt = df_rng.shape[0]
        pos_cnt = len(df_rng[df_rng['TARGET'] == 1])
        pos_rate = pos_cnt / bin_cnt
        # Rango usado para todos los bins hasta i_max_bin
        df_rng_0 = dfp[dfp[s_rank] < i_max_bin]
        df_rng_1 = dfp[dfp[s_rank] >= i_max_bin]
        # targets positivos (tp=true positivos, fn=false negativos)
        tp = len(df_rng_1[df_rng_1['TARGET'] == 1])
        fn = len(df_rng_0[df_rng_0['TARGET'] == 1])
        pos = tp + fn
        tpr = tp / pos
        fnr = 1 - tpr
        # Targets Negativos
        tn = len(df_rng_0[df_rng_0['TARGET'] == 0])
        fp = len(df_rng_1[df_rng_1['TARGET'] == 0])
        neg = tn + fp
        fpr = fp / neg
        tnr = 1 - fpr
        # Hacer el dataframe para resumen de estadísticas por bin
        row = [i, i_min_bin, i_max_bin, score_min, score_max, bin_cnt, pos_cnt, pos_rate]
        dfr.loc[i] = row
        # Hacer el dataframe para las estadísticas ROC por bin
        row2 = [i, score_min, score_max, tnr, fpr, fnr, tpr, tn, fp, fn, tp]
        dfr2.loc[i] = row2
        i_min_bin = i_max_bin # Preparar la siguiente iteración
```

Y este estará dentro del bucle externo, pero fuera del bucle interno:

```
    if s_cla == 'cla_rl':
        dfr_lr = dfr.copy()
        dfr2_lr = dfr2.copy()
    elif s_cla == 'cla_gb':
        dfr_gb = dfr.copy()
        dfr2_gb = dfr2.copy()
    else:
        print('Error')
        break
```

Por último, fuera de los bucles está el código que imprime los datos calculados:

```
print("==== Resulados de contadores estadísticos positivos por bin")
print("Estadísticas de Regresión Logística:")
print(dfr_lr.to_string())
print()
print("Estadísticas de GBM:")
print(dfr_gb.to_string())
print()
print("---")
print("Estadísticas globales en el test set:")
pos_cnt = len(dfp[dfp['TARGET'] == 1])
pos_rate = pos_cnt / len_test
print("pos_cnt", pos_cnt)
print("total filas", len_test)
print("pos_rate", pos_rate)
```

ENTREGA 11: Ejecuta el código y:

a) Entrega una captura de los datos que genera.

Paso 5.4. Dibujar el ratio de clases positivas por cada bin. Con frecuencia necesitamos que los modelos den resultados consistentes cuando los valores de los scores aumentan. Para este tipo de análisis, preferimos modelos que tengan bins de scores que se incrementen de manera monótona (que el siguiente de la derecha sea mayor que el anterior de su izquierda).

```
s_titulos = ['Regresión Logística', 'GBM']
plt.figure(figsize=(12, 4))
for i, s_cla in enumerate(li_clas):
    s_titulo = s_titulos[i]
    if s_cla == 'cla_rl':
        dfr = dfr_lr
        dfr2 = dfr2_lr
    elif s_cla == 'cla_gb':
        dfr = dfr_gb
        dfr2 = dfr2_gb
    else:
        print('Error')
        break
    plt.subplot(1, 2, i+1)
    plt.bar(dfr['i_bin'], dfr['pos_rate'])
    plt.title('Percentiles para ' + s_titulo)
    plt.grid()
    plt.xlabel('Score bin (de bins con bajos a altos scores)')
    plt.ylabel('Ratio de clases positivas por bin')
    plt.legend(loc="lower right")
    plt.xlim([0, RANGO_BINS])
    plt.ylim([0.0, 1])
plt.show()
```

ENTREGA 12: Ejecuta el código y:

a) Entrega una captura de los gráficos datos que imprime.

b) ¿Hay algún modelo de los que hemos usado que tenga un comportamiento monótono?

Paso 5.5. Imprimir datos estadísticos de cada bin y dibujar las curvas ROC y precisión contra recall. Aunque todos estos términos a la mayoría no os digan nada, son herramientas que veremos en próximas unidades y que se utilizan para saber como se comporta de bien o mal un modelo.

```
print("==== Resultados de estadísticas ROC por bin")
print("Estadísticas de Regresión Logística:")
print(dfr2_lr.to_string())
print()
print("Estadísticas de GBM:")
print(dfr2_gb.to_string())
```

```
def plot_roc_y_precision_recall(y_true, y_score, modelo=None):
    fpr, tpr, _ = roc_curve(y_true, y_score)
    roc_auc = auc(fpr, tpr) # Calcula ROC y AUC usando FPR y TPR
    precision, recall, _ = precision_recall_curve(y_true, y_score)
    plt.figure(figsize=(8, 3))
    plt.subplot(1,2,1)
    lw = 2
    plt.plot(fpr, tpr, color='darkorange', lw=lw, label='Curva ROC (AUC=%0.4f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('Ratio Falsos Positivos')
    plt.ylabel('Ratio True Positivos')
    titulo = 'Curva ROC'
    if modelo != None:
        titulo = titulo + ' de ' + modelo.__class__.__name__
    plt.title(titulo)
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.subplot(1,2,2)
    plt.step(recall, precision, color='orange', where='post')
    plt.xlabel('Recall')
    plt.ylabel('Precisión')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('Curva Precisión-Recall')
    plt.grid(True)
    plt.subplots_adjust(0.125, 0.1, 0.9, 0.9, 0.5, 0.2)
    plt.show()
```

Y por último, llamar al método que dibuja las gráficas:

```
for s_cla in li_clas:
    print("MODELO: " + s_cla)
    print("--------------")
    cla = eval(s_cla)
    y_pred = cla.predict(X_test).astype(int)
    y_score = cla.predict_proba(X_test)
    plot_roc_y_precision_recall(y_test, y_score[:,1], cla)
    print()
```

Por resumir lo que hemos hecho en esta práctica que ha consistido en realizar algunas tareas típicas en la preparación de un modelo de ML:

- Usando técnicas de ***selección de características*** al final hemos usado 9 de las 30 iniciales.
- Hemos creado sos modelos de clasificación (Regresión logística y *GBM*) y hemos encontrado sus hiperparámetros óptimos usando **`GridSearchCV`**.
- Hemos calculado un ranking de importancia de las características para cada modelo.
- Hemos medido y comparado ambos con la curva *ROC* y la de *precisión* vs *recall*.
- Hemos cambiado **`TEST_SIZE_RATIO=0.2`** y eso mejora a *GBM* porque lo vuelve más estable.

ACTIVIDAD 6. HAZ ALGO PARECIDO CON OTRO DATASET

Añade una entrada en el fichero ***alumnos*** que está accesible desde este enlace con tus credenciales de Conselleria. El fichero tiene una serie de características que te explico a continuación:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	NOMBRE	EDAD	PESO_KG	ALTURA_CM	NUM_HERMANOS	HS_DEPORTE	HS_VIDEOJUEGOS	TALLA_PIE	GENERO_MASCULINO	MIOPÍA	LLEVA_GAFAS	TIENE_ALERGIAS	PIEZAS_FRUTA_DÍA	CAFÉS_DIA
	JOROSROD	54	86	155	0	5	0	42	1	5,25	1	0	3	5

- NOMBRE: las primeras 3 letras de tu nombre y las primeras 3 de tus apellidos.
- EDAD: los años que tienes.
- PESO_KG: tu peso aproximado.
- ALTURA_CM: la altura aproximada en centímetros.
- NUM_HERMANOS: el número de hermanos o hermanas que tienes.
- HS_DEPORTE: horas semanales que dedicas a actividades deportivas de cualquier tipo.
- HS_VIDEOJUEGOS: horas semanales que dedicas a jugar videojuegos.
- TALLA_PIE: número de pie que usas cuando compras calzado.
- GENERO_MASCULINO: un 1 si es tu género biológico o 0 en otro caso.
- MIOPÍA: la cantidad de diotrías de miopía que tienes. Si no tienes, un 0.
- LLEVAS_GAFAS: Un 1 si utilizas gafas o lentillas y 0 en otro caso.
- TIENE_ALERGIAS: un 1 si eres alérgico a alguna cosa o un 0 en otro caso.
- PIEZAS_FRUTA_DÍA: la cantidad aproximada de piezas de fruta que sueles comer cada día.
- CAFÉS_DÍA: la cantidad de cafés aproximada que sueles tomar cada día.

- Cuando todos hayamos rellenado el fichero, puedes descargarlo como un *.csv* y cargarlo.
- Haz algún procesamiento como quitar la columna nombre al dataset. Escoge una columna que te sirva para clasificar a los alumnos usando el resto de columnas. Por ejemplo: tiene_alergias, lleva_gafas, o incluso una numérica podrías transformarla en dos clases CAFÉS_DÍA si es mayor de 0 es SI (1) y si es cero es No (0). O tomas poco entre 0 y 2 y muchos entre 3 o más. Lo que se te ocurra.
- Indica el target que has elegido y transforma los datos si necesitas hacerlo.
- Haz un procesamiento similar al que hemos realizado al dataset original:
  - Estudia si puedes eliminar características para el clasificador.
  - Crea dos modelos clasificadores (pueden ser los mismos).
  - Configura sus hiperparámetros (al tener muy pocos datos posiblemente tengan overfitting).
  - Estudia el resultado de compararlos y sus curvas *ROC*, su *AUC* y *precisión-recall*.

ENTREGA 13: Entrega un informe con los gráficos que generes y los resultados que obtengas, el código de las anteriores actividades lo podrás reutilizar casi en su totalidad haciendo pocos cambios para que funcione con los nuevos datos. Llama al archivo ***`saa_u01_p01_e13_<tusIniciales>.py`***