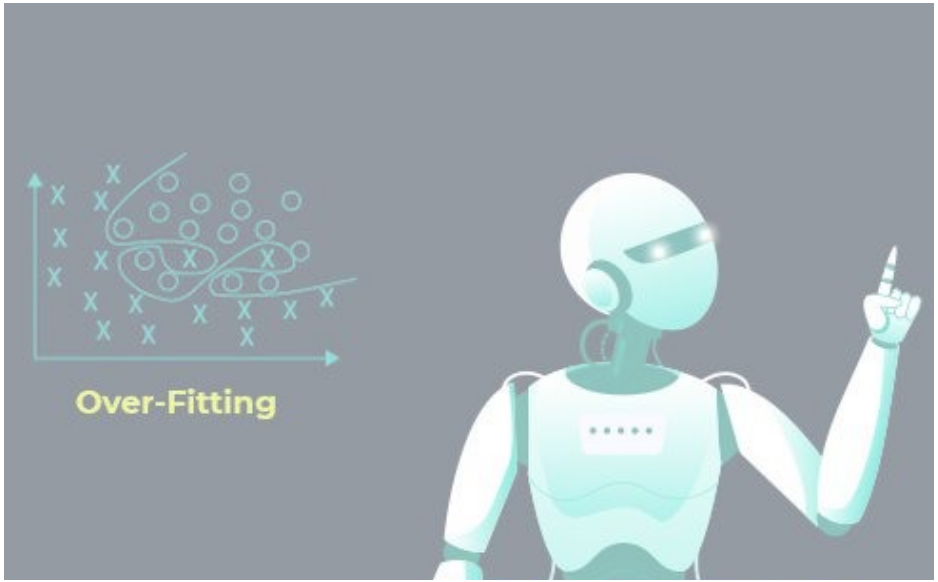
Don't Overfit! II

Álvaro Prieto Cano
Juan Manuel Ruiz
Manuel Cabello Casas
María García Zambrano

Introducción

Modelar una matriz de 20,000 x 200

250 muestras de entrenamiento



Enfoque

1. Carga y Verificación de Datos
2. Análisis Exploratorio de Datos
3. Estandarización de datos
4. Preparación de Datos
5. Entrenamiento de Modelos
6. Evaluación y Análisis de Resultados

Procedimientos Previos - Carga de datos

Especificamos las rutas en donde se encuentran las bases de datos en el Google Drive. Se usa la función de pandas .read_csv para que lea el archivo CSV y lo convierte en un DataFrame.

```
# Rutas a los archivos CSV en Google Drive (si están allí)
train_path = '/content/drive/My Drive/train.csv'
test_path = '/content/drive/My Drive/test.csv'
sample_submission_path = '/content/drive/My Drive/sample_submission.csv'

# Leer los archivos CSV
train = pd.read_csv(train_path)
test = pd.read_csv(test_path)
sample_submission_data = pd.read_csv(sample_submission_path)
```

Procedimientos Previos - Verificación de Valores Nulos

Verificamos que no haya valores nulos en las columnas de las distintas bases de datos, para ello usamos el .isnull:

```
nul = 0
for col in train.columns:
    if(train[col].isnull().any()):
        print(col,'has null values')
        nul = 1
if(nul==0):
    print('There is no Null value present')
```

```
There is no Null value present
```

```
nul = 0
for col in sample_submission_data.columns:
    if(sample_submission_data[col].isnull().any()):
        print(col,'has null values')
        nul = 1
if(nul==0):
    print('There is no Null value present')
```

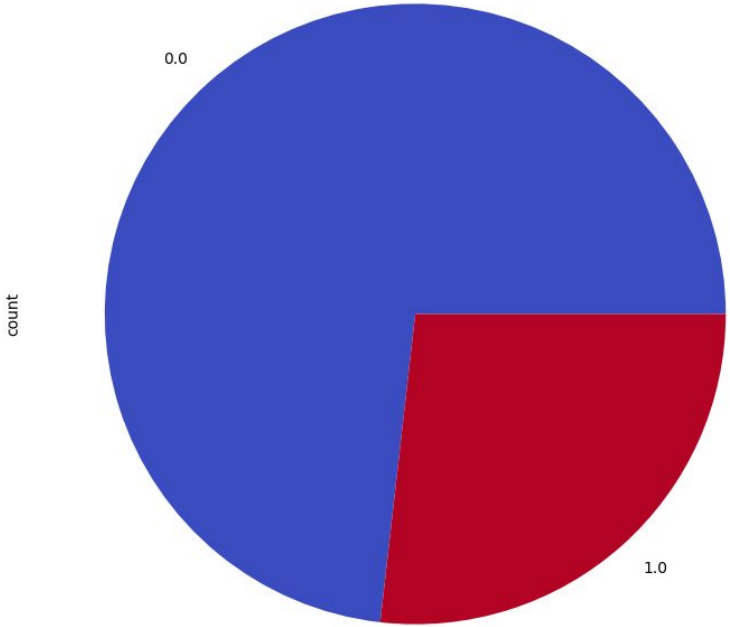
```
There is no Null value present
```

```
nul = 0
for col in test.columns:
    if(test[col].isnull().any()):
        print(col,'has null values')
        nul = 1
if(nul==0):
    print('There is no Null value present')
```

```
There is no Null value present
```

Procedimientos Previos - Análisis Preliminar de Datos

La clase 0.0 es mucho más común que la clase 1.0 en el conjunto de datos train, esto demuestra que hay desbalanceo de clases.




```
train.shape, test.shape
```

```
((250, 302), (19750, 301))
```

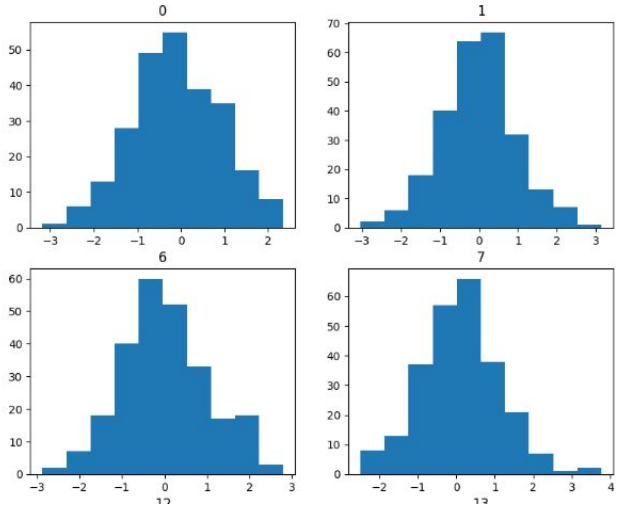
Procedimientos Previos - Selección de Características

Seleccionamos las características numéricas del DataFrame 'train'.

La librería “gc” expone el mecanismo de administración de memoria subyacente de Python, el recolector de basura automático.

```
numerical_features = train.columns[2:]
```

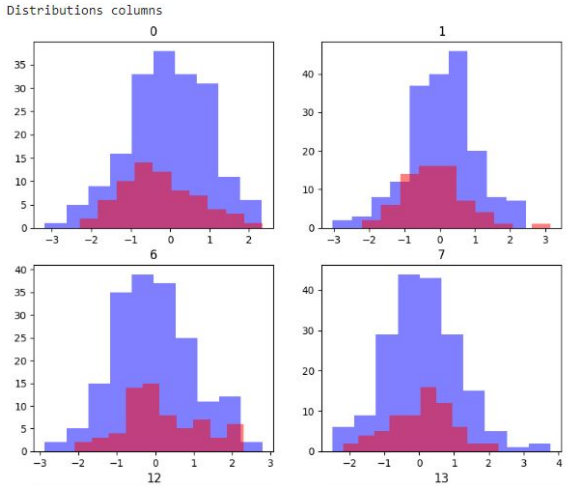
```
print('Distributions - Histograms columns')
plt.figure(figsize=(30, 200))
for i, col in enumerate(numerical_features):
    plt.subplot(50, 6, i + 1)
    plt.hist(train[col])
    plt.title(col)
gc.collect();
```



Procedimientos Previos - Selección de Características

Los histogramas de las características numéricas, segmentadas por la variable objetivo target, nos permite identificar diferentes patrones entre las etiquetas.

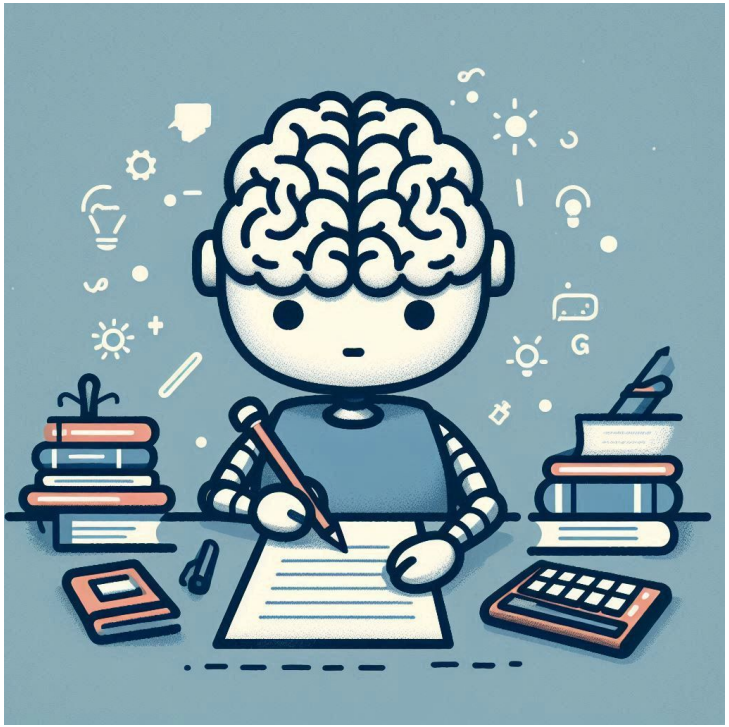
```
print('Distributions columns')
plt.figure(figsize=(30, 200))
for i, col in enumerate(numerical_features):
    plt.subplot(50, 6, i + 1)
    plt.hist(train[train["target"] == 0][col], alpha=0.5, label='0', color='b')
    plt.hist(train[train["target"] == 1][col], alpha=0.5, label='1', color='r')
    plt.title(col)
gc.collect();
```



Entrenamiento y test

Objetivos:

- Explicar el proceso de entrenamiento de los modelos.
- Presentar los resultados obtenidos durante la fase de test.



División del Conjunto de Datos

Se dividió el conjunto de datos en un 70% para entrenamiento y un 30% para prueba de forma estratificada.

```
xtrain, xvalid, ytrain, yvalid = train_test_split(X, y, random_state=42, test_size=0.3)
```

Balanceo de Clases: Procedimiento SMOTE

1. Identificar las muestras de la clase minoritaria.
2. Selecciona a los vecinos más cercanos.
3. Generación de Muestras Sintéticas.
4. Adición de Muestras Sintéticas:
    a. Se añaden al conjunto de datos
    b. Balancean el conjunto de datos

```
smote = SMOTE(random_state=42)
xtrain_balanced, ytrain_balanced = smote.fit_resample(xtrain, ytrain)
```

Clasificadores Evaluados

- Logistic Regression
- Random Forest
- Extra Trees.

```
classifiers = [
    LogisticRegression(C=0.1, class_weight='balanced', penalty='l1', solver='liblinear', random_state=42),
    RandomForestClassifier(max_depth=2, random_state=42),
    ExtraTreesClassifier(max_depth=2, random_state=42)
]
clf_names = ['log_clf', 'rand_clf', 'extra_clf']
```

Entrenamiento y Evaluación

Entrenamos

Evaluamos:

- F1-score
- área bajo la curva ROC (AUC).

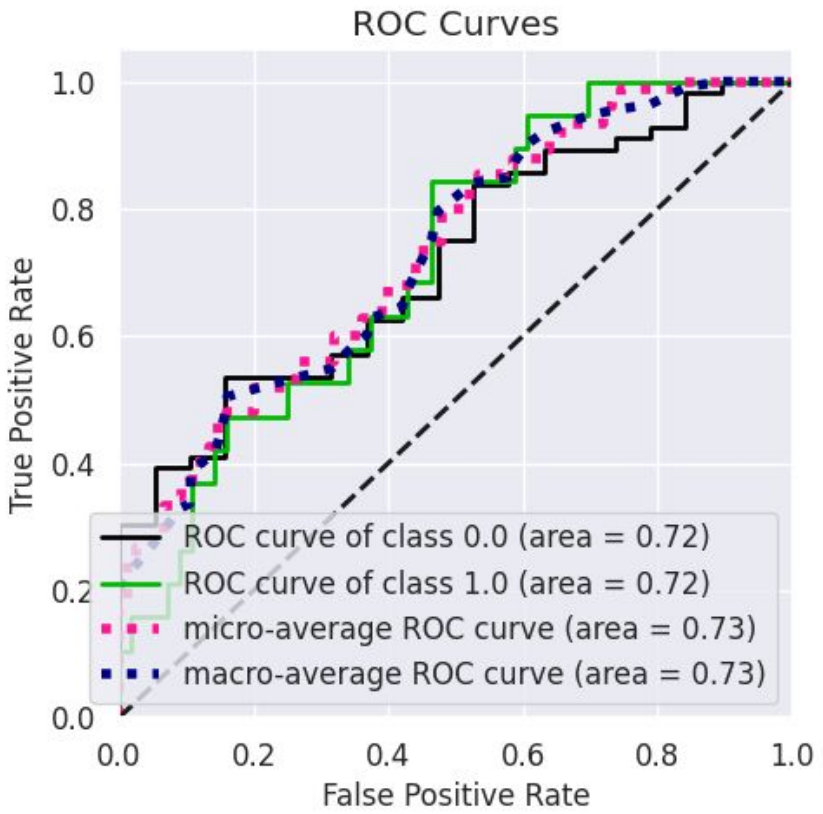
```
for clf, name in zip(classifiers, clf_names):
    start_time = time.time()
    clf.fit(xtrain_balanced, ytrain_balanced)
    predictions = clf.predict(xvalid)
    predictions_probas = clf.predict_proba(xvalid)
```

Resultados

Clasificador Logistic Regression:

f1_score: 0.5800000000000001

roc_auc_score:
0.7171052631578947

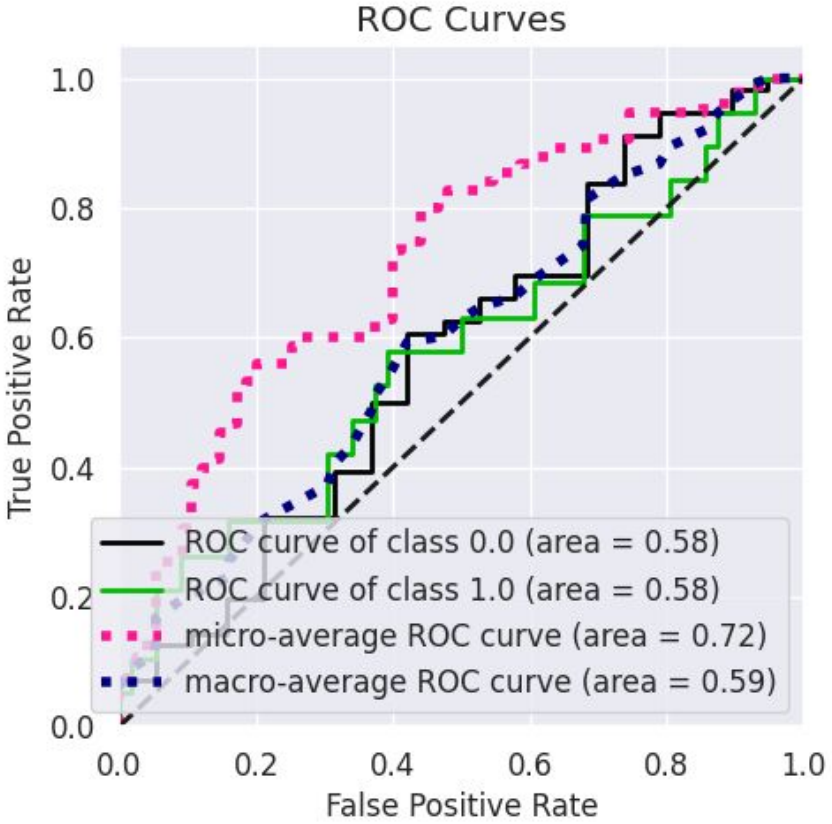


Resultados

Clasificador Random Forest

CV f1_score: 0.5227272727272727

roc_auc_score: 0.5780075187969925

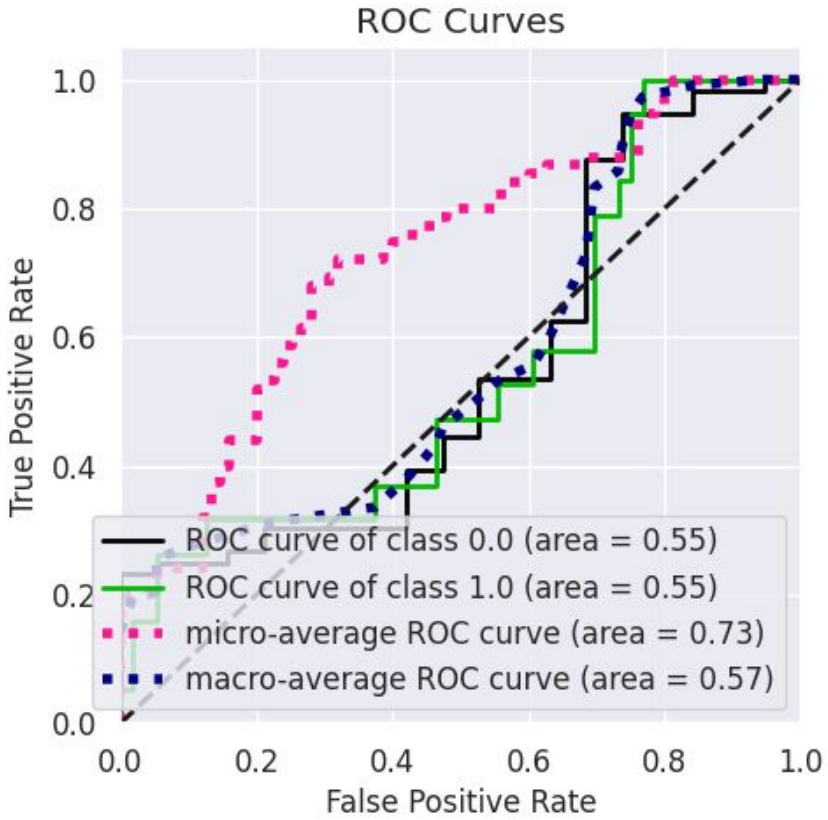


Resultados

Clasificador Extra Trees

f1_score: 0.5714285714285714

roc_auc_score:
0.5516917293233083



Conclusiones

- El clasificador Logistic Regression ha mostrado el mejor rendimiento.
- Se realizaron las predicciones utilizando el clasificador entrenado.
- Se aseguró que la longitud de las predicciones coincida con la longitud del DataFrame de envío.
- Se añadieron las predicciones al DataFrame de envío.
- Se guardó el DataFrame de envío en un archivo CSV en la ubicación especificada.