Ingeniería en Inteligencia Artificial,
Machine Learning
Sem: 2025-1, 5BM1, Complejidad de Datos ,
Fecha: 29 de Septiembre 2024

Complejidad de Datos

Machine Learning

Grupo: 5BM1

Profesor: Andrés Floriano García

Integrantes:
Juan Manuel Alvarado Sandoval
Alexander Iain Crombie Esquinca
Herrera Saavedra Jorge Luis
Quiñones Mayorga Rodrigo

Contents

1 Introducción

El desequilibrio de clases es un problema común en el ámbito del aprendizaje automático, en particular en tareas de clasificación. Se produce cuando una o más clases están subrepresentadas en comparación con otras, lo que puede llevar a que el modelo se sesgue hacia la clase mayoritaria. Como resultado, la precisión del modelo en las clases minoritarias tiende a ser baja, lo que afecta la capacidad del modelo para generalizar correctamente.

En estos casos, es importante aplicar técnicas que permitan mitigar este problema. Entre ellas, se encuentra la generación sintética de muestras, como es el caso del algoritmo *Synthetic Minority Over-sampling Technique* (SMOTE).

2 Algoritmo SMOTE

El algoritmo SMOTE es una técnica de sobremuestreo que genera instancias sintéticas para las clases minoritarias. En lugar de simplemente replicar las instancias existentes, SMOTE genera nuevas instancias interpolando entre las instancias minoritarias existentes. Dado un punto minoritario x_i y uno de sus vecinos más cercanos x_{nn}, el algoritmo genera una nueva instancia como:

$$x_{\text{new}} = x_i + \lambda \times (x_{nn} - x_i)$$

donde λ es un número aleatorio en el rango $[0, 1]$. Esta interpolación crea nuevas instancias que están distribuidas de manera más uniforme en el espacio de las características.

3 Limpieza del Dataset

Antes de proceder a aplicar cualquier técnica de clasificación o balanceo, es esencial realizar una limpieza adecuada del *dataset*. Este proceso incluye la eliminación de valores duplicados, la corrección de errores y, sobre todo, la eliminación de valores perdidos (*missing values*).

La limpieza de datos asegura que el modelo no esté influenciado por información incorrecta o incompleta, lo que puede afectar negativamente su rendimiento. Además, un conjunto de datos limpio facilita un entrenamiento

más eficiente y reduce el riesgo de introducir sesgos involuntarios en el modelo.

4 Importancia de Eliminar Valores Perdidos

Los valores perdidos en un *dataset* pueden representar un desafío considerable en el preprocesamiento de datos. Si no se manejan adecuadamente, estos valores pueden causar que los algoritmos de aprendizaje automático produzcan resultados inexactos o se comporten de manera errática. Las principales técnicas para manejar los valores perdidos incluyen:

- Eliminación de filas o columnas con valores perdidos.
- Imputación de valores utilizando la media, la mediana o la moda.
- Uso de algoritmos más avanzados como la imputación múltiple.

Al eliminar o imputar correctamente los valores perdidos, el *dataset* se convierte en una base más sólida para entrenar el modelo de clasificación, lo que mejora tanto su rendimiento como su capacidad de generalización.

5 Evidencia de la Práctica (pantallazos)

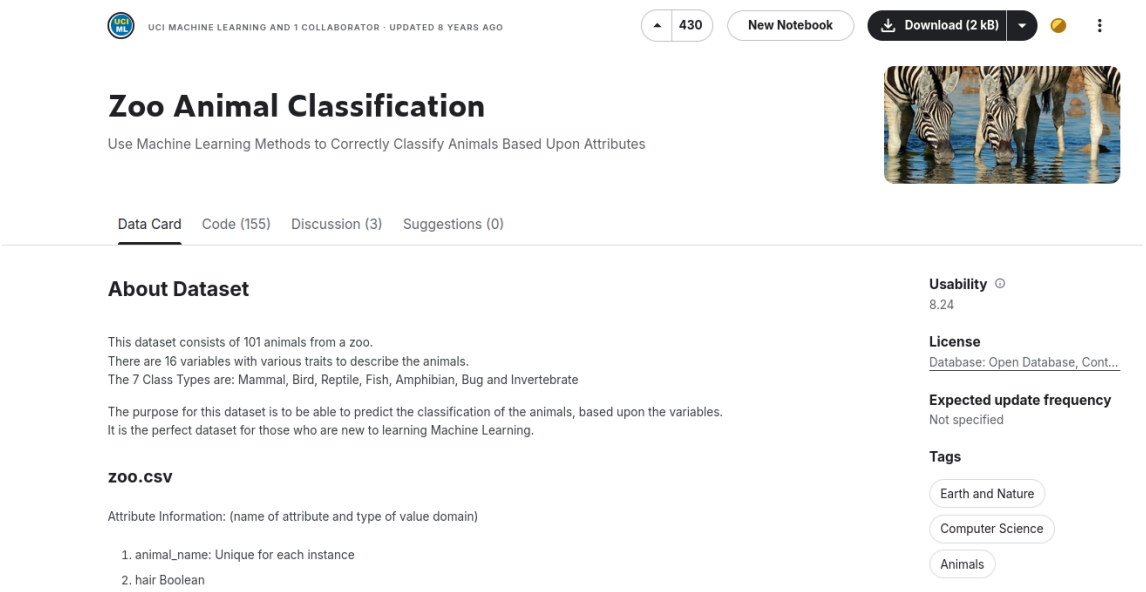


Figure 1: Descarga del dataset de clasificación de especies nombrado "Zoo" disponible en Kaggle o en UCI

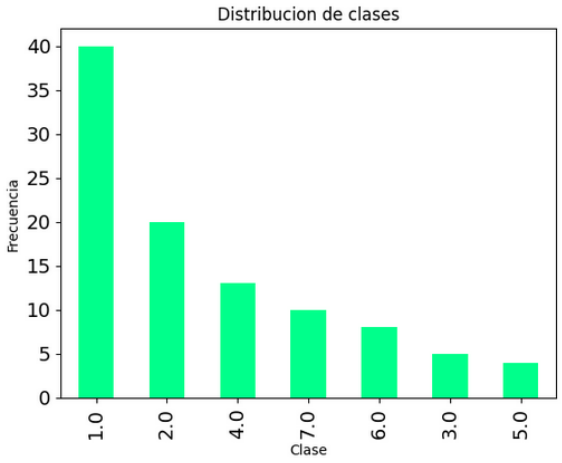
```
In [3]: df.head()
```

Out[3]:

	animal_name	hair	feathers	eggs	milk	airbone	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	type
0	antelope	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	4.0	1.0	0.0	1.0	1.0
1	bass	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	4.0
2	bear	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	4.0	0.0	0.0	1.0	1.0
3	boar	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	4.0	1.0	0.0	1.0	1.0
4	buffalo	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	4.0	1.0	0.0	1.0	1.0

Figure 2: Carga del dataset en CSV y visualización de su cabecera/columnas.

```
In [4]: class_counts = df['type'].value_counts()
        class_counts.plot(kind= 'bar',fontsize=14,color='#00ff8b')
        plt.title('Distribucion de clases')
        plt.xlabel('Clase')
        plt.ylabel('Frecuencia')
        plt.show()
        print(class_counts)
```




```
type
1.0    40
2.0    20
4.0    13
7.0    10
6.0     8
3.0     5
5.0     4
Name: count, dtype: int64
```

Figure 3: Gráfico de la distribución de clases del conjunto de datos.

In [5]:
```
#Eliminaremos animal_name puesto que no será necesaria
df.drop(['animal_name'],axis=1, inplace= True)
df.head()
```

Out[5]:

	hair	feathers	eggs	milk	airbone	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	type
0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	4.0	1.0	0.0	1.0	1.0
1	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	4.0
2	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	4.0	0.0	0.0	1.0	1.0
3	1.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	0.0	0.0	4.0	1.0	0.0	1.0	1.0
4	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	4.0	1.0	0.0	1.0	1.0

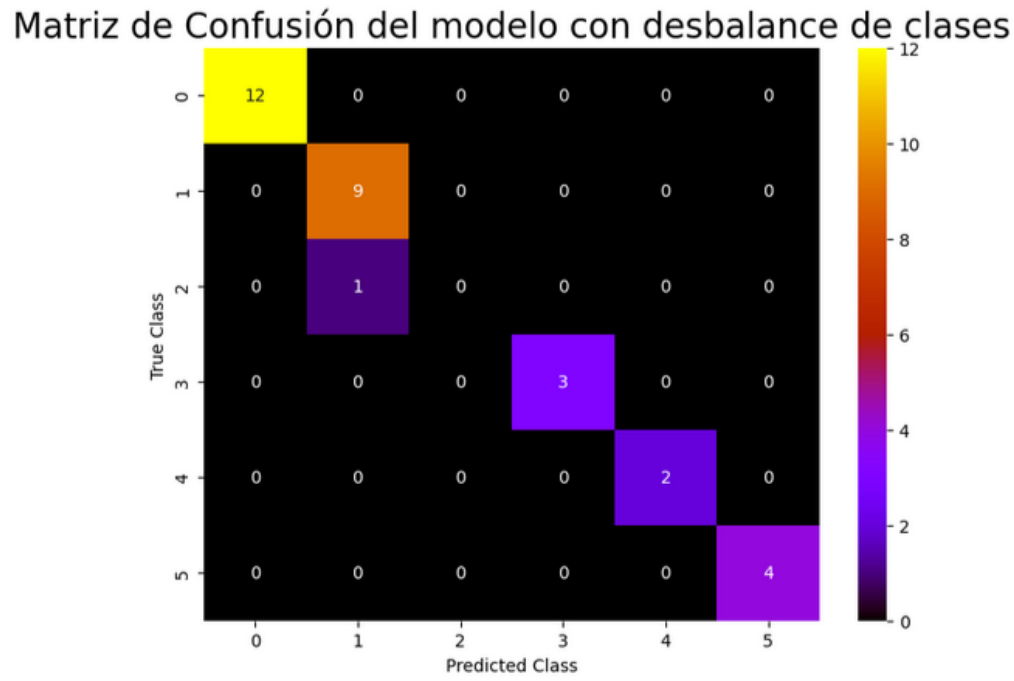
In [6]:
```
from sklearn.model_selection import train_test_split
import numpy as np

X = df.drop('type', axis=1)  # Variables independientes
y = df['type']  # Variable dependiente

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

# Imprimir porcentajes de clase positiva en el conjunto de entrenamiento y prueba
print(f'''% Positive class in Train = {np.round(y_train.value_counts(normalize=True)[1] * 100, 2)}
% Positive class in Test = {np.round(y_test.value_counts(normalize=True)[1] * 100, 2)}''')
```

```
% Positive class in Train = 40.58
% Positive class in Test = 38.71
```

Figure 4: Ajuste de los valores y descarte de columnas que no aportan información.

```python
# Imprimir la cantidad de NaN en y_train
print(f'Valores nulos en y_train: {y_train.isnull().sum()}')

# Manejo de valores nulos en y_train
if y_train.isnull().any():
    imputer = SimpleImputer(strategy='most_frequent')  # Imputación con la moda
    y_train_imputed = imputer.fit_transform(y_train.values.reshape(-1, 1)).flatten()
else:
    y_train_imputed = y_train

# Verificar si hay NaN en X_train
if X_train.isnull().any().any():
    imputer_X = SimpleImputer(strategy='mean')  # Imputación con la media
    X_train_imputed = imputer_X.fit_transform(X_train)
else:
    X_train_imputed = X_train

# Entrenamos el modelo
model = RandomForestClassifier(random_state=42)
model.fit(X_train_imputed, y_train_imputed)

# Realizamos predicciones
predictions = model.predict(X_test)



# Matriz de confusión
c_matrix = confusion_matrix(y_test, predictions)

# Visualizar la matriz de confusión
plt.figure(figsize=(8, 6))
plt.title("Matriz de Confusión del modelo con desbalance de clases", size=20)
sns.heatmap(c_matrix, annot=True, cmap="gnuplot", fmt='g')
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()


print("Clases en y_test:", y_test.unique())

print("Distribución en y_train:")
print(y_train.value_counts())

print("Distribución en y_test:")
print(y_test.value_counts())
# Evaluar el modelo
print("\n")
```

Figure 5: Manejo de valores nulos y división del conjunto de entrenamiento y pruebas para posteriormente aplicar el algorítmo de Random Forest.




```
Clases en y_test: [7. 2. 1. 6. 4. 3.]
Distribución en y_train:
type
1.0    28
2.0    11
4.0    10
7.0     6
6.0     6
5.0     4
3.0     4
Name: count, dtype: int64
Distribución en y_test:
type
1.0    12
2.0     9
7.0     4
4.0     3
6.0     2
3.0     1
Name: count, dtype: int64


Accuracy = 0.97
Recall (macro) = 0.83
```

Figure 6: Matríz de confusión para el dataset con desbalance de clases. Se observa un accuracy de 0.97 y un recall de 0.83.

```
Valores nulos en y: 1
Cambio de X antes de SMOTE: (101, 16)
Cambio de X después de SMOTE: (287, 16)


 (+/-) Balance de las clases (%):
Clase 1.0: 14.29%
Clase 2.0: 14.29%
Clase 3.0: 14.29%
Clase 4.0: 14.29%
Clase 5.0: 14.29%
Clase 6.0: 14.29%
Clase 7.0: 14.29%
Dataset transformado guardado como 'zoo_smote.csv'
```

Figure 7: Resultados del balance de clases del dataset después de aplicar SMOTE para oversampling.

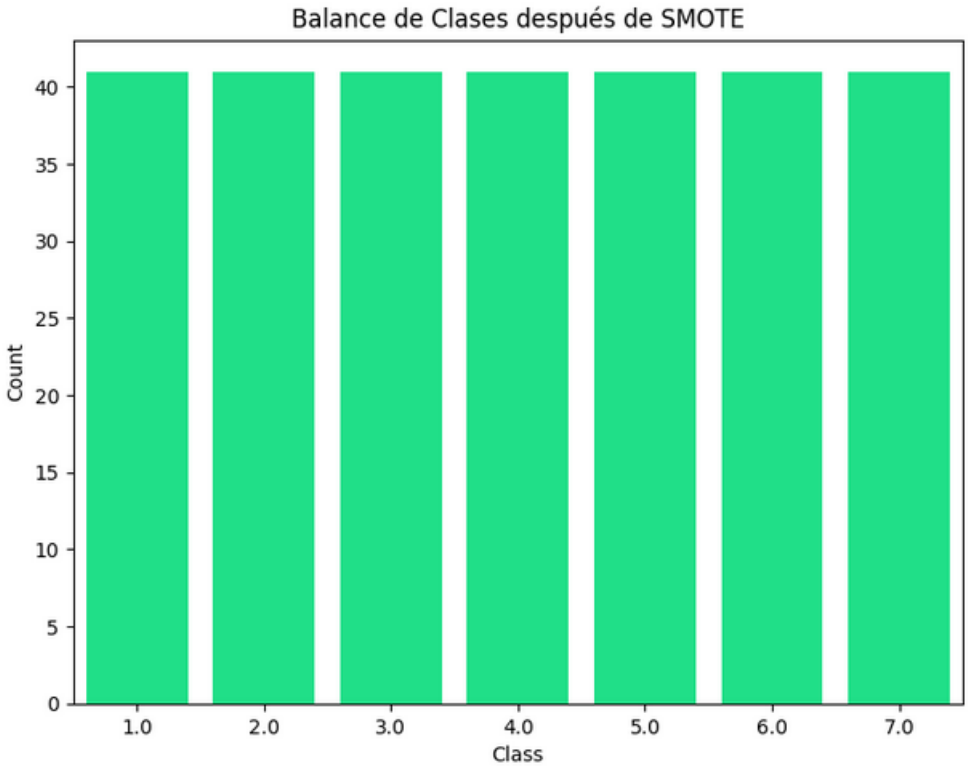


Figure 8: Equilibrio de clases del dataset después de aplicar SMOTE y reducir el desbalance de clases.

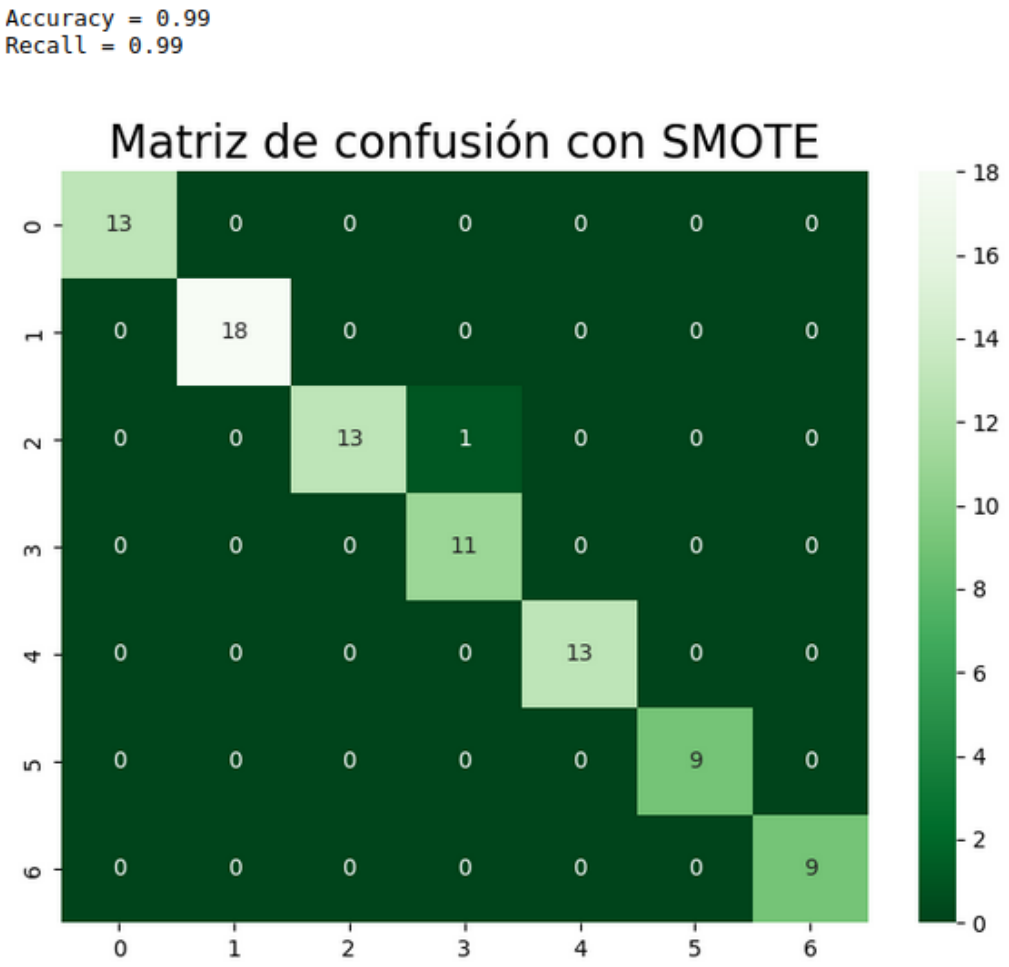


Figure 9: Matríz de confusión luego de aplicar Random Forest al dataset con reducción de desbalance de clases. Se observa como el accuracy y el recall del modelo han incrementado directamente, lo cual contribuye significativamente a este problema de clasificación.

6 Agradecimientos

Agradecemos al UCI Machine Learning Repository por proporcionar acceso al dataset *Zoo*, así como a Richard Forsyth por la creación del mismo. También agradecemos el apoyo de la comunidad de aprendizaje automático y a todos aquellos que han contribuido a la evolución de este repositorio.

Source Information – Creator: Richard Forsyth – Donor: Richard S. Forsyth 8 Grosvenor Avenue Mapperley Park Nottingham NG3 5DX 0602-621676 – Date: 5/15/1990

7 Referencias

- Forsyth, R. (1990). *Zoo* [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5R59V