

# **Evaluaciones Agropecuarias de los municipios de Cundinamarca y Boyacá**

Andrea Nataly Hernández Fuentes, Edgar Felipe Torres Ortegón,  
Helmy Andrea Moreno Navarro, y Samuel Mantilla Velásquez

Universidad Sergio Arboleda

Talento Tech

## Índice

<b>Objetivos</b>	<b>3</b>
Objetivo General . . . . .	3
Objetivos Específicos . . . . .	3
<b>Preprocesamiento</b>	<b>3</b>
Descripción General . . . . .	3
Técnicas de Preprocesamiento . . . . .	4
Estandarización . . . . .	4
Reducción de Dimensionalidad usando PCA . . . . .	5
PCA con 6 Componentes . . . . .	5
PCA con 16 Componentes . . . . .	5
Análisis Visual y Modelado . . . . .	5
Conclusión . . . . .	6
<b>Modelos Predictivos de Regresión</b>	<b>6</b>
Árboles de decisión con datos de frecuencia . . . . .	7
Árboles de decisión con One hot Encoding . . . . .	11
Bosques aleatorios de Regresión . . . . .	13
SVM de regresión . . . . .	18
Validación Cruzada . . . . .	20
<b>Modelos Predictivos de Clasificación</b>	<b>24</b>
Arboles de Clasificación . . . . .	26
Bosques Aleatorios de Clasificación . . . . .	27
SVM de Clasificación . . . . .	28
Validación Cruzada . . . . .	30
Conclusiones . . . . .	31

## Objetivos

### Objetivo General

Realizar un análisis integral de la biodiversidad, conservación ambiental y los ecosistemas en las regiones de Cundinamarca y Boyacá, utilizando técnicas avanzadas de análisis de datos y modelos predictivos. A través de la exploración, análisis y modelado de datos, buscamos identificar patrones significativos, áreas de interés prioritarias y proponer recomendaciones fundamentadas para la gestión sostenible y la conservación eficaz de la biodiversidad en estas regiones.

### Objetivos Específicos

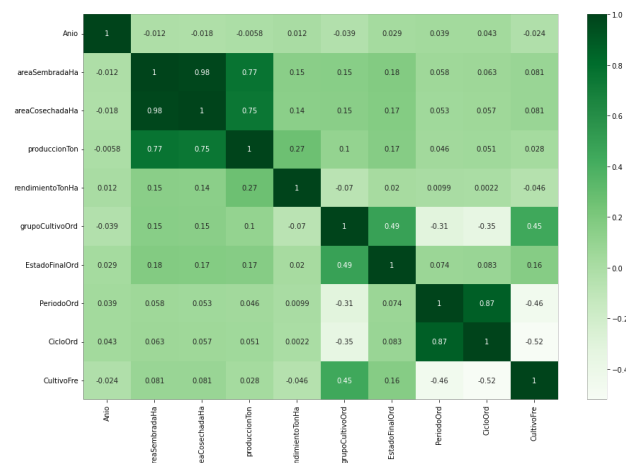
1. Seleccionar y aplicar modelos predictivos adecuados para los datos, como modelos de regresión o clasificación.
2. Evaluar el rendimiento de los modelos y ajustar según sea necesario.
3. Generar recomendaciones prácticas y accionables para la conservación ambiental y la gestión de ecosistemas.

## Preprocesamiento

### Descripción General

En esta etapa del proyecto, se llevaron a cabo diversas técnicas de preprocesamiento de datos con el objetivo de preparar los datos para la construcción de modelos predictivos. La variable objetivo seleccionada fue el **rendimiento**, específicamente el rendimiento en toneladas por hectárea (*rendimientoTonHa*). Las técnicas implementadas incluyen la estandarización de los datos y la reducción de componentes principales utilizando el método de Análisis de Componentes Principales (PCA).

Figura 1

*Depuración de Variables*

*Note.* Tabla de correlación de variables con correlación significativa

**Técnicas de Preprocesamiento***Estandarización*

La estandarización es una técnica común en el preprocesamiento de datos que implica escalar las características para que tengan una media de 0 y una varianza de 1. Esto es esencial cuando los modelos de aprendizaje automático son sensibles a las escalas de las características.

```

1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3 X = df.drop(columns='rendimientoTonHa').iloc[:, 1:]
4 y = df['rendimientoTonHa']
5 X_scaler = scaler.fit_transform(X)

```

En el código anterior, se realiza la estandarización de todas las características excepto la variable objetivo.

## ***Reducción de Dimensionalidad usando PCA***

El Análisis de Componentes Principales (PCA) se utiliza para reducir la dimensionalidad de los datos mientras se conserva la mayor parte de la variabilidad presente en el conjunto de datos original. Esto ayuda a mitigar problemas de multicolinealidad y mejora la eficiencia computacional del modelo.

Para este proyecto, se utilizaron dos configuraciones de PCA: una con 6 componentes y otra con 16 componentes.

### ***PCA con 6 Componentes***

```
1 from sklearn.decomposition import PCA
2 pca = PCA(n_components=6)
3 X_Pca = pca.fit_transform(X_scaler)
4 print(X_Pca)
```

En este caso, se redujo la dimensionalidad de los datos a 6 componentes principales.

### ***PCA con 16 Componentes***

```
1 pca = PCA(n_components=16)
2 X_Pca_one = pca.fit_transform(X_scaler)
3 varianza = pca.explained_variance_ratio_
4 print(varianza.sum())
```

Aquí, se utilizó PCA para reducir la dimensionalidad a 16 componentes principales y se verificó la varianza explicada para asegurarse de que se mantiene una alta proporción de la variabilidad original.

## **Análisis Visual y Modelado**

Se visualizó la distribución de las nuevas componentes principales y se implementó un modelo de regresión utilizando árboles de decisión para predecir el rendimiento basado en las componentes principales obtenidas.

```

1 from sklearn.tree import DecisionTreeRegressor
2 X_train, X_test, y_train, y_test = train_test_split(X_Pca, y, test_size
    =0.2, random_state=42)
3 modelo_tree = DecisionTreeRegressor()
4 modelo_tree.fit(X_train, y_train)
5 y_predict = modelo_tree.predict(X_test)
6 print(y_predict[0:10])
7 print(y_test[0:10].values)

```

Se utilizó la técnica de validación cruzada para evaluar el rendimiento del modelo y se obtuvieron las predicciones para el conjunto de prueba.

## Conclusión

En la etapa de preprocesamiento, se aplicaron técnicas cruciales para preparar los datos para el modelado predictivo. La estandarización aseguró que todas las características estuvieran en la misma escala, y el PCA ayudó a reducir la dimensionalidad del conjunto de datos, lo que facilitó un mejor rendimiento y eficiencia del modelo predictivo. Estos pasos son fundamentales para obtener modelos precisos y eficientes en proyectos de análisis de datos.

## Modelos Predictivos de Regresión

Se desarrolló modelos de regresión para la predicción del rendimiento el cual representa la medida de la eficiencia de la producción agrícola y se expresa como la cantidad de producto agrícola (en toneladas) obtenida por unidad de área de tierra sembrada (hectáreas). Es decir, es la producción por hectárea de tierra sembrada. Lo anterior con el fin de poder predecir el rendimiento de los cultivos, para ello se establecieron varios modelos predictivos de regresión tales como: Árboles de decisión con datos de frecuencia, Árboles de decisión con One hot Encoding, Bosques aleatorios de Regresión y SVM de regresión.

## Árboles de decisión con datos de frecuencia

Los árboles de decisión son modelos predictivos que utilizan estructuras de árbol para mapear características a conclusiones sobre un valor objetivo. Cada nodo interno representa una prueba en una característica, cada rama el resultado de la prueba, y cada hoja el resultado final. Son interpretables y adecuados para datos complejos y no lineales, pero pueden sobreajustarse. Su rendimiento mejora con técnicas como la poda y el uso de conjuntos de árboles, como Bosques Aleatorios. Un árbol de decisión es un modelo de predicción que se puede usar tanto para clasificación como para regresión. En este caso, se está utilizando para regresión, donde se predicen valores continuos. Este árbol se usa para mejorar la precisión y la generalización del modelo.

```
1 from sklearn.tree import DecisionTreeRegressor
2 modelo_tree = DecisionTreeRegressor(random_state=42,max_depth=20)
3 modelo_tree.fit(X_train,y_train)
4 # modelo_tree.get_depth()
5 y_predict_tree = modelo_tree.predict(X_test)
6 # print(y_predict[0:10])
7 # print(y_test[0:10].values)
8 })
```

En el código anterior se está utilizando un árbol de decisión para realizar una regresión con el uso de la biblioteca “scikit-learn”. Se está construyendo, entrenando y utilizando un modelo de regresión de árboles de decisión para predecir valores basados en un conjunto de datos de prueba, con parámetros específicos para controlar la profundidad del árbol y asegurar resultados reproducibles.

**Evaluación del modelo:** Se usaron métricas de evaluación para regresión como:

- **Coeficiente de determinación ( $R^2$ ):**

Indica la proporción de la varianza en la variable dependiente que es predecible a partir de

las variables independientes. Un valor cercano a 1 indica un buen ajuste.

$$R^2 = 1 - \frac{\sum_i^n (y_{\text{test}} - \hat{y}_{\text{predict}})^2}{\sum_i^n (y_{\text{test}} - \bar{y}_{\text{predict}})^2}$$

- **Error Cuadrático Medio (MSE):**

Mide la media de los cuadrados de las diferencias entre las predicciones y los valores reales. A menores valores, mejor rendimiento.

$$MSE = \frac{\sum_i^n (y_{\text{test}} - \hat{y}_{\text{predict}})^2}{n}$$

- **Raíz del Error Cuadrático Medio (RMSE):**

Es una métrica de evaluación utilizada en problemas de regresión para medir la diferencia entre los valores predichos por un modelo y los valores observados (reales). Un valor bajo indica que el modelo tiene un buen desempeño, ya que las predicciones están cerca de los valores reales.

$$RMSE = \sqrt{\frac{\sum_i^n (y_{\text{test}} - \hat{y}_{\text{predict}})^2}{n}}$$

Por lo anterior se ejecutó el siguiente código:

```
1 # error cuadrático medio
2 from sklearn.metrics import mean_squared_error
3 import math
4 mse = mean_squared_error(y_test, y_predict_tree)
5 print(f'R^2 = {modelo_tree.score(X_test, y_test):.3f}')
```



```
6 print(f'MSE = {mse:.3f}')
7 print(f'RMSE = {math.sqrt(mse):.3f}')
```

Como resultado se obtuvieron los siguientes valores:

**$R^2= 0.855$**

El coeficiente de determinación  $R^2$  es 0.855, lo que significa que aproximadamente el 85.5% de la variabilidad en los datos de prueba puede ser explicada por el modelo de regresión de árboles de decisión.

**MSE=36.455**

El Error Cuadrático Medio es 36.455. Este valor representa la media de los errores al cuadrado entre los valores predichos y los valores reales. Es una medida de la precisión del modelo; un valor más bajo indica un mejor ajuste.

**RMSE=6.038**

La Raíz del Error Cuadrático Medio es 6.038. Al ser la raíz cuadrada del MSE, proporciona una medida de la magnitud promedio de los errores de predicción, en las mismas unidades que la variable objetivo. Un RMSE más bajo indica un mejor desempeño del modelo.

```
1 print(y_predict_tree[0:10])
2 print(y_test[0:10].values)
```

```
[11.92857143 25.          14.70245902  9.          4.91         20.47222222
 8.88833333 15.          6.          27.          ]
[10. 22. 10.  9.  4. 22. 11.  5.  7. 25.]
```

En este fragmento de código, se están comparando los primeros 10 valores predichos por el modelo con los primeros 10 valores reales del conjunto de prueba. Por lo anterior de las 10 predicciones, una es exacta, varias están razonablemente cerca del valor real, y algunas muestran una desviación considerable. El modelo tiende a estar cerca de los valores reales en la mayoría de los casos, pero hay algunas predicciones que son notablemente diferentes,

como la octava predicción. Las diferencias entre las predicciones y los valores reales varían, con algunas diferencias pequeñas (menores de 2 unidades) y otras grandes (10 unidades). Finalmente se crea un Dataframe con 6 componentes principales con el siguiente código:

```
1 X_grafica =pd.DataFrame(data=X_Pca ,columns=['PCA1 ','PCA2 ','PCA3 ','PCA4 '
      , 'PCA5 ','PCA6 '])
2 X_grafica
```

	PCA1	PCA2	PCA3	PCA4	PCA5	PCA6
0	-2.275591	-0.115370	0.101015	-0.361575	0.087427	0.085633
1	1.016299	-0.274357	0.372488	1.490733	-1.126539	-0.228583
2	1.073383	-0.521254	0.220306	1.359295	-1.068824	0.031890
3	2.343979	-0.203588	0.191657	-0.862682	-0.647423	-0.081042
4	0.653769	-0.627791	1.096255	0.575230	0.373366	-0.157758
...	...	...	...	...	...	...
17738	1.093528	0.256630	-1.038081	-0.215296	1.131111	0.007898
17739	0.877356	-0.098794	-0.346472	-0.350371	0.494716	0.046586
17740	2.459962	-0.109862	-0.634084	-0.580094	-0.007806	0.141346
17741	1.717899	0.015166	-0.724313	0.209352	1.348752	-0.104218
17742	1.092608	0.248373	-1.042658	-0.214544	1.132154	0.017523
17743 rows × 6 columns						

Con el siguiente código, se muestra el árbol de decisión entrenado con una profundidad de 2 niveles, esto permite una comprensión rápida de cómo el árbol toma decisiones basadas en las componentes principales obtenidas mediante PCA.

```
1 from sklearn import tree
2 import matplotlib.pyplot as plt
3 # for arbol in (modelo_tree_clas.tree_):
```

```

4 plt.figure(figsize=(20,10))
5 print(tree.export_text(modelo_tree,feature_names=['PCA1','PCA2','PCA3',
    'PCA4','PCA5','PCA6'],max_depth=2))
6 # plt.show()

```

Como resultado se muestra lo siguiente, una vista simplificada del árbol de decisión:

```

|--- PCA3 <= 0.58
|   |--- PCA3 <= -3.23
|   |   |--- PCA6 <= 0.44
|   |   |   |--- truncated branch of depth 18
|   |   |   |--- PCA6 > 0.44
|   |   |   |   |--- truncated branch of depth 4
|   |   |--- PCA3 > -3.23
|   |   |   |--- PCA3 <= -0.03
|   |   |   |   |--- truncated branch of depth 18
|   |   |   |   |--- PCA3 > -0.03
|   |   |   |   |   |--- truncated branch of depth 18
|   |--- PCA3 > 0.58
|   |   |--- PCA6 <= 0.10
|   |   |   |--- PCA5 <= 0.48
|   |   |   |   |--- truncated branch of depth 18
|   |   |   |   |--- PCA5 > 0.48
|   |   |   |   |   |--- truncated branch of depth 11
|   |   |   |--- PCA6 > 0.10
|   |   |   |   |--- PCA5 <= 0.27
|   |   |   |   |   |--- truncated branch of depth 9
|   |   |   |   |   |--- PCA5 > 0.27
|   |   |   |   |   |   |--- truncated branch of depth 15

```

## Árboles de decisión con One hot Encoding

En esta sección se tomaron árboles de decisión como modelo de machine learning en combinación con el preprocesamiento de datos categóricos utilizando la técnica de One Hot Encoding.

```

1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3
4 X_one = df_agro_One.drop(columns='rendimientoTonHa').iloc[:,1:]
5 y_one = df_agro_One['rendimientoTonHa']

```

```

6
7 X_scaler_one = scaler.fit_transform(X_one)
8 # print(X_scaler_one.mean())
9 # print(X_scaler_one.var())
10 from sklearn.decomposition import PCA
11 pca= PCA(n_components=16)
12 X_Pca_one=pca.fit_transform(X_scaler_one)
13 # print(X_Pca_one)
14 varianza=pca.explained_variance_ratio_
15 # print(varianza)
16 # print(varianza.sum())
17 from sklearn.model_selection import train_test_split
18 X_train_one,X_test_one,y_train_one,y_test_one = train_test_split(
    X_Pca_one,y_one,test_size=0.2,random_state=42)
19
20 # Modelo de regresion con arboles de decision
21
22 from sklearn.tree import DecisionTreeRegressor
23 modelo_tree = DecisionTreeRegressor()
24 modelo_tree.fit(X_train_one,y_train_one)
25 y_predict_one = modelo_tree.predict(X_test_one)
26
27 print(y_predict_one[0:10])
28 print(y_test_one[0:10].values)

```

```

[ 3.  30.   6.   8.   3.  27.   5.4  2.   1.2 22. ]
[10. 22. 10.   9.   4. 22. 11.   5.   7. 25.]

```

Con el código anterior lo primero que se hace es una estandarización de los datos con el fin de que estén en la misma escala.

Con la variable X\_one después de haber realizado un tratamiento al conjunto de datos me-

diante la técnica One hot Encoding, se indica hasta donde van las columnas correspondientes a los registros de las características.

Por otro lado, la variable `y_one` indica la variable objetivo que en este caso es el rendimiento. Luego se entrena el modelo `X_scaler_one = scaler.fit_transform(X_one)`, con el fin de realizar una transformación, es decir, reducir el conjunto de datos y transformarlos, mantener la escala.

Una vez teniendo el valor de `X_scaler_one`, se toman los valores transformados y se ingresan dentro del PCA para reducir los componentes a 16 y la varianza explicada es mayor al 95 % y con esos valores de `X_Pca_one` se realiza el proceso de división del conjunto de datos entre prueba y test y se realiza el modelo de árboles de decisión, se toman los valores estandarizados se reemplaza y se presenta el proceso típico.

Finalmente se imprime las métricas de evaluación para regresión como se muestra a continuación:

$$R^2 = 0,71$$

$$MSE = 73,00$$

$$RMSE = 8,54$$

El modelo de árboles de decisión está funcionando de manera moderada, explicando el 71 % de la varianza con un error promedio de 8.54 unidades.

## Bosques aleatorios de Regresión

Funcionan combinando las predicciones de múltiples árboles de decisión para mejorar la precisión y controlar el sobreajuste.

```
1 from sklearn.ensemble import RandomForestRegressor
2
3 X_train_forest, X_test_forest, y_train_forest, y_test_forest =
   train_test_split(X_scaler, y, test_size=0.3, random_state=42)
4
5 modelo_Forest_Regressor = RandomForestRegressor(n_estimators=100,
```

```

6                                     max_depth=8,
7                                     random_state=42)
8 modelo_Forest_Regressor.fit(X_train_forest,y_train_forest)
9 # modelo_Forest_Regressor.get_params()

```

Con el anterior código se divide el conjunto de datos, se crea el modelo RandomForestRegressor y se entrena el modelo, como resultado se obtiene lo siguiente:

```

▼ RandomForestRegressor
RandomForestRegressor(max_depth=8, random_state=42)

```

```

1 y_predict_rf = modelo_Forest_Regressor.predict(X_test_forest)
2
3 print(f'R^2 = {modelo_Forest_Regressor.score(X_test_forest,
4       y_test_forest):.3f}')
5 mse = mean_squared_error(y_test_forest,y_predict_rf)
6 print(f'MSE = {mse:.3f}')
7 print(f'RMSE = {math.sqrt(mse):.3f}')

```

Con el código anterior se realiza la predicción y se evalúa el modelo, se muestran los siguientes resultados:

$$R^2 = 0,888$$

$$MSE = 28,842$$

$$RMSE = 5,370$$

**$R^2$ :** El coeficiente de determinación para evaluar qué tan bien las predicciones del modelo se ajustan a los valores reales. Un  $R^2=0.888$  indica un buen ajuste.

**MSE:** Error cuadrático medio. Un MSE de 28.842 sugiere que, en promedio, el cuadrado de las diferencias entre las predicciones y los valores reales es 28.842.

**RMSE:** Raíz del error cuadrático medio, es de 5.370, proporcionando una métrica más interpretable que el MSE al estar en la misma escala que los datos originales. Por último se imprimen las primeras 5 predicciones realizadas por el modelo de Bosques Aleatorios y comparándolas con los valores reales correspondientes del conjunto de prueba.

```
1 print(y_predict_rf[0:5])
2 print(y_test_forest[0:5].values)
```

Como resultado se obtiene lo siguiente:

```
[12.00688292 20.75858672 13.67774182  8.36807423  6.17702077]
[10.  22.  10.   9.   4.]
```

En general, el modelo de Bosques Aleatorios realiza predicciones razonablemente cercanas a los valores reales, con algunas sobreestimaciones y subestimaciones. Esto sugiere que el modelo captura bien las tendencias generales del conjunto de datos, pero hay margen de mejora en la precisión de ciertas predicciones individuales. El modelo podría beneficiarse de ajustes adicionales para mejorar la precisión en casos específicos.

### ■ Ajuste de los HiperParametros

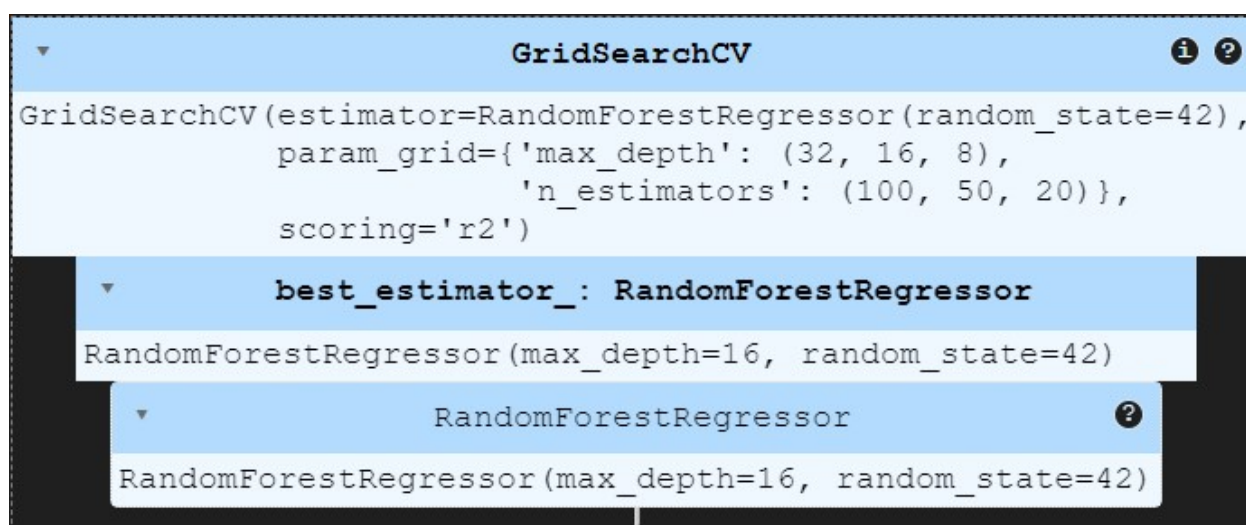
Se evaluaron los parámetros y se realizó mediante técnicas de Tuning De Hiperparámetros para mejorar el rendimiento como el método GridSearchCV, el cual consiste en definir un conjunto de valores posibles para cada hiperparámetro y evaluar el rendimiento del modelo para todas las combinaciones posibles. Sklearn proporciona la clase GridSearchCV para realizar esta búsqueda.

```
1 from sklearn.model_selection import GridSearchCV
2 parametros_regresion = {'max_depth':(32,16,8),
3                           'n_estimators':(100,50,20)
4                           }
5 modelo_Forest_Regressor = RandomForestRegressor(random_state=42)
```

Con el código anterior usamos GridSearchCV con el fin de realizar una búsqueda exhaustiva de los valores de hiperparámetros y así encontrar la mejor combinación de parámetros para el modelo

```
1 hiperParametros= GridSearchCV(modelo_Forest_Regressor,
2                               parametros_regresion,
3                               scoring='r2')
4 hiperParametros.fit(X_train_forest,y_train_forest)
```

Como resultado de su ejecución muestra lo siguiente:



```
GridSearchCV(estimator=RandomForestRegressor(random_state=42),
              param_grid={'max_depth': (32, 16, 8),
                           'n_estimators': (100, 50, 20)},
              scoring='r2')

best_estimator_: RandomForestRegressor
RandomForestRegressor(max_depth=16, random_state=42)
RandomForestRegressor(max_depth=16, random_state=42)
```

Con el código anterior se toma el modelo de Bosques Aleatorios y el diccionario de parámetros. Se utiliza el coeficiente de determinación R2 como métrica para evaluar el rendimiento del modelo, éste mide qué tan bien se ajustan los valores predichos a los valores observados, luego se entrena el modelo para cada combinación de hiperparámetros especificada, utilizando los datos de entrenamiento X\_train\_forest y y\_train\_forest.

```
1 print(hiperParametros.best_score_)
2 print(hiperParametros.best_params_)
3 mejor_A=hiperParametros.best_estimator_
```



Con el anterior código primero se imprime la mejor puntuación de validación cruzada  $R^2$  obtenida durante la búsqueda en cuadrícula, el cual como resultado se obtiene lo siguiente:

0.8780727546622085

El mejor modelo encontrado por el GridSearchCV tiene un coeficiente de determinación  $R^2$  de aproximadamente 0.878. Esto significa que el modelo es capaz de explicar aproximadamente el 87.8 % de la varianza en los datos de validación.

Luego se imprime el conjunto de hiperparámetros que proporcionó la mejor puntuación de validación cruzada, el cual muestra el siguiente resultado:

'max\_depth': 16, 'n\_estimators': 100

La mejor configuración de hiperparámetros encontrada es un modelo de Bosques Aleatorios con una profundidad máxima de 16 para los árboles y 100 árboles en el bosque.

Por último se asigna el mejor estimador (modelo entrenado) encontrado por GridSearchCV a la variable mejor\_A. Almacena el modelo de Bosques Aleatorios que tiene la mejor combinación de hiperparámetros y el mejor rendimiento según la validación cruzada.

Este modelo puede ser utilizado posteriormente para hacer predicciones en nuevos datos.

Finalmente se ejecuta el siguiente código:

```
1 y_predict_rfh= mejor_A.predict(X_test_forest)
2 print(f'R^2 = {mejor_A.score(X_test_forest,y_test_forest):.3f}')
3 mse = mean_squared_error(y_test_forest,y_predict_rfh)
4 print(f'MSE = {mse:.3f}')
5 print(f'RMSE = {math.sqrt(mse):.3f}')
```

Con el código anterior se predicen los valores de  $y$  que es la variable objetivo en el conjunto de datos de prueba  $X\_test\_forest$ , luego se calcula el coeficiente de determinación  $R^2$ , el error cuadrático medio MSE y la raíz del error cuadrático medio RMSE obteniendo así los siguientes resultados:

$$R^2 = 0,942$$

$$MSE = 14,981$$

$$RMSE = 3,871$$

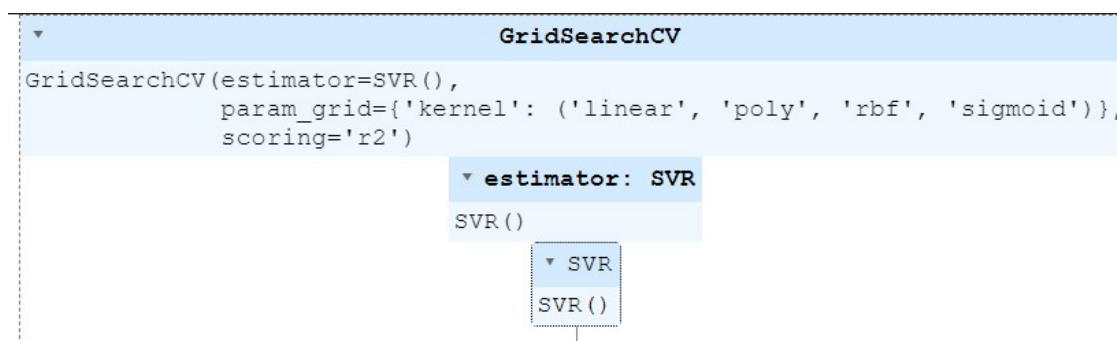
El coeficiente de determinación  $R^2$  de 0.942 indica que el modelo es capaz de explicar el 94.2% de la variabilidad observada en los datos de prueba. Un  $R^2$  tan alto sugiere que el modelo captura muy bien la relación entre las variables independientes y la variable dependiente. El MSE de 14.981 significa que, en promedio, el cuadrado de la diferencia entre los valores observados y los predichos es de aproximadamente 15 unidades al cuadrado. El RMSE de 3.871 sugiere que, en promedio, las predicciones del modelo están a 3.871 unidades de distancia de los valores reales.

## SVM de regresión

Las Máquinas de Soporte Vectorial (SVM) son un poderoso método de aprendizaje supervisado utilizado para la clasificación y la regresión. En este caso para la regresión se usó el siguiente código:

```
1 from sklearn.svm import SVR
2 model_SVM = SVR()
3 parametros_SVR = {'kernel': ('linear', 'poly', 'rbf', 'sigmoid')}
4 hiperParametros_SVR = GridSearchCV(model_SVM,
5                                     parametros_SVR,
6                                     scoring='r2')
7 hiperParametros_SVR.fit(X_train, y_train)
```

Se obtiene el siguiente resultado:



Con el código anterior se importa el modelo SVR para regresión de la biblioteca sklearn. Se define un diccionario de hiperparámetros para el SVR, especificando diferentes tipos de funciones como lo son Núcleo lineal, Núcleo polinómico, Núcleo radial y Núcleo sigmoide.

Con `hyperParametros_SVR` se configura una búsqueda en cuadrícula (`GridSearchCV`) para encontrar los mejores hiperparámetros de SVR basándose en el puntaje de  $R^2$ . `GridSearchCV` es una técnica que evalúa exhaustivamente todas las combinaciones posibles de los hiperparámetros especificados.

Con `hyperParametros_SVR.fit(X_train,y_train)` se realiza la búsqueda en cuadrícula ajustando el modelo SVR a los datos de entrenamiento `X_train` y `y_train` para cada combinación de hiperparámetros.

Con `hyperParametros_SVR.get_params()` listamos todos los parámetros del objeto `GridSearchCV` configurado para SVR como se muestra a continuación:

```
{'cv': None,
 'error_score': nan,
 'estimator__C': 1.0,
 'estimator__cache_size': 200,
 'estimator__coef0': 0.0,
 'estimator__degree': 3,
 'estimator__epsilon': 0.1,
 'estimator__gamma': 'scale',
 'estimator__kernel': 'rbf',
 'estimator__max_iter': -1,
 'estimator__shrinking': True,
 'estimator__tol': 0.001,
 'estimator__verbose': False,
 'estimator': SVR(),
 'n_jobs': None,
 'param_grid': {'kernel': ('linear', 'poly', 'rbf', 'sigmoid')},
 'pre_dispatch': '2*n_jobs',
 'refit': True,
 'return_train_score': False,
 'scoring': 'r2',
 'verbose': 0}
```

```

1 y_predict_SVR=hiperParametros_SVR.predict(X_test)
2 print(f'R^2 = {hiperParametros_SVR.score(X_test,y_test):.3f}')
3 mse_SVR = mean_squared_error(y_test,y_predict_SVR)
4 print(f'MSE = {mse_SVR:.3f}')
5 print(f'RMSE = {math.sqrt(mse_SVR):.3f}')

```

Con el anterior código imprimimos el coeficiente de determinación  $R^2$ , error cuadrático medio (MSE) y la Raíz del Error Cuadrático Medio (RMSE)

$$R^2 = 0,462$$

$$MSE = 135,287$$

$$RMSE = 11,631$$

**R2:** El modelo explica el 46.2 % de la variabilidad en los datos de prueba, lo cual sugiere que hay un margen significativo para mejorar el ajuste.

**MSE:** Un valor relativamente alto del error cuadrático medio indica que las predicciones del modelo están, en promedio, bastante alejadas de los valores verdaderos.

**RMSE:** La raíz del error cuadrático medio también es alta, sugiriendo que el modelo tiene un error significativo en sus predicciones.

## Validación Cruzada

La Validación Cruzada es una técnica esencial en el campo del aprendizaje automático y la modelización estadística que aborda ciertos problemas asociados con la división tradicional de datos en conjuntos de entrenamiento y prueba.

Se divide el conjunto de datos en 3 secciones:

- **Arboles de regresión**

```

1 from sklearn.model_selection import KFold,cross_validate
2 kf= KFold(n_splits=5,shuffle=True,random_state=42)
3 metrics_tree = cross_validate(modelo_tree,

```

```

4             X_Pca,y,
5             cv=kf,
6             scoring='r2')
7 metrics_tree
8 print(f'R^2 = {metrics_tree["test_score"].mean():.3f}')
```

Con el código anterior se utiliza la técnica de validación cruzada para evaluar el rendimiento de un modelo de árbol de decisión en un conjunto de datos, en este caso obtenemos como resultado lo siguiente:

$$R^2 = 0,863$$

Lo cual indica que, en promedio, el modelo de árbol de decisión explica el 86.3% de la variación en los datos de salida durante el proceso de validación cruzada. Este valor sugiere un buen ajuste del modelo a los datos, pero también es importante revisar otras métricas y validaciones para confirmar su rendimiento.

### ■ Bosque de regresión

```

1 from sklearn.model_selection import KFold,cross_validate
2 # kf= KFold(n_splits=5,shuffle=True,random_state=42)
3 metrics_forest = cross_validate(modelo_Forest_Regressor,
4             X_scaler,y,
5             cv=kf,
6             scoring='r2')
7 metrics_forest
8 print(f'R^2 = {metrics_forest["test_score"].mean():.3f}')
```

Con el código anterior se utiliza la validación cruzada para evaluar el rendimiento de un modelo de Bosque Aleatorio en un conjunto de datos, en este caso obtenemos como resultado lo siguiente:

$$R^2 = 0,905$$

Este resultado indica que el modelo de Bosque Aleatorio explica el 90.5 % de la variabilidad en los datos de salida durante la validación cruzada. Este valor sugiere un ajuste muy bueno del modelo a los datos, lo que significa que el modelo está capturando la mayoría de las relaciones entre las características y la variable objetivo.

### ■ SVM de regresión

```
1 from sklearn.model_selection import KFold, cross_validate
2
3 # kf= KFold(n_splits=5, shuffle=True, random_state=42)
4 metrics_SVM = cross_validate(model_SVM,
5                               X_Pca, y,
6                               cv=kf,
7                               scoring='r2')
8 metrics_SVM
```

Con el código anterior se utiliza la validación cruzada para evaluar el rendimiento del modelo de Máquina de Soporte Vectorial (SVM) en un conjunto de datos.

Como resultado se obtiene lo siguiente:

```
{'fit_time': array([7.59962368, 7.76415324, 7.38061666, 7.75188017, 7.21615267]),
 'score_time': array([5.32437611, 4.1803844 , 4.36303329, 4.24266863, 4.42119575]),
 'test_score': array([0.46176924, 0.44877029, 0.41582849, 0.45410069, 0.33128277])}
```

**fit\_time:** Esta lista indica el tiempo en segundos que tomó ajustar el modelo en cada uno de los cinco folds. Los tiempos varían ligeramente pero están en el rango de 7 a 8 segundos por fold.

**score\_time:** Esta lista indica el tiempo en segundos que tomó evaluar el modelo en los datos de prueba en cada fold. Estos tiempos están alrededor de 4 a 5 segundos.

**test\_score:** Estos son los valores  $R^2$  para cada uno de los 5 folds. Los puntajes están en el rango de 0.331 a 0.462.

Los valores  $R^2$  indican qué tan bien el modelo explica la variabilidad de los datos de prueba en cada fold.

Un  $R^2$  cercano a 1 indica un ajuste excelente, mientras que valores cercanos a 0 indican un ajuste pobre. Aquí, los valores son relativamente bajos, lo que sugiere que el modelo SVM no está capturando adecuadamente la variabilidad en los datos para estos folds específicos.

En conclusión los resultados indican que el modelo SVM no está proporcionando un buen ajuste a los datos, con puntajes  $R^2$  que sugieren que no está explicando bien la variabilidad de la variable objetivo.

Luego se calcula y se muestra el promedio de los valores  $R^2$  obtenidos en los cinco folds durante la validación cruzada.

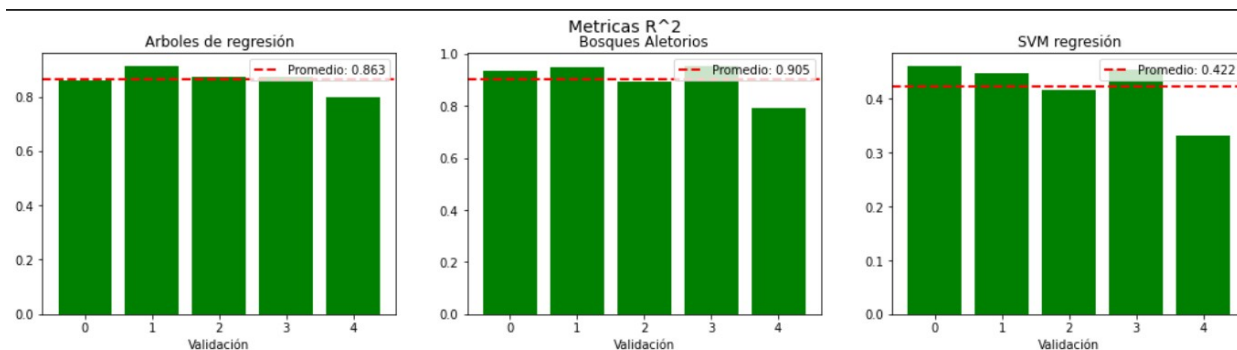
Obteniendo el siguiente resultado

$$R^2 = 0.422$$

Lo cual indica que, en promedio, el modelo SVM está explicando el 42.2% de la variabilidad en la variable objetivo en los datos de prueba. Aunque mejor que un valor de  $R^2$  cercano a cero, 0.422 aún sugiere que el modelo podría no estar capturando de manera óptima la relación entre las características y la variable objetivo. indica un rendimiento moderado del modelo SVM en este conjunto de datos.

Luego se visualiza y compara los resultados de la validación cruzada de diferentes modelos de regresión.

Las barras verdes representan las puntuaciones individuales de cada fold, y la línea roja punteada representa el promedio de esas puntuaciones para el modelo como se muestra a continuación:



La visualización permite comparar visualmente el desempeño de los tres modelos en términos de sus  $R^2$  scores en cada una de las particiones de la validación cruzada.

La línea roja punteada ayuda a identificar rápidamente cuál de los modelos tiene el promedio de  $R^2$  más alto, proporcionando una forma clara de evaluar cuál modelo podría estar funcionando mejor en general.

Por lo anterior según se observa, en la gráfica de árboles de regresión tiene un promedio de 0.863, en la gráfica de bosques aleatorios tiene un promedio de 0.905 y en la gráfica de SVM (Máquinas de Vectores de Soporte) tiene un promedio de 0.422, por lo tanto la gráfica de Bosques Aleatorios representa el mejor modelo, ya que tiene el mayor promedio de  $R^2$ , indicando que tiene el mejor rendimiento general en la validación cruzada entre los tres modelos.

## Modelos Predictivos de Clasificación

Con el fin de aplicar técnicas de predicción para datos categóricos, se decidió construir modelos de predicción para el estado final del cultivo. Para facilitar la elaboración de estos modelos, se optó por establecer una codificación basada en la frecuencia, asignando una etiqueta numérica que indique un orden. Esta codificación se estableció de forma descendente según la frecuencia.

```
1 df_agro['EstadoFinalOrd'] = df_agro['estadoFisicoCultivo'].map({
2     'En fresco':0
3     , 'Grano seco':1
4     , 'Grano':2
5     , 'Pergamino o seco de trilla':3
```

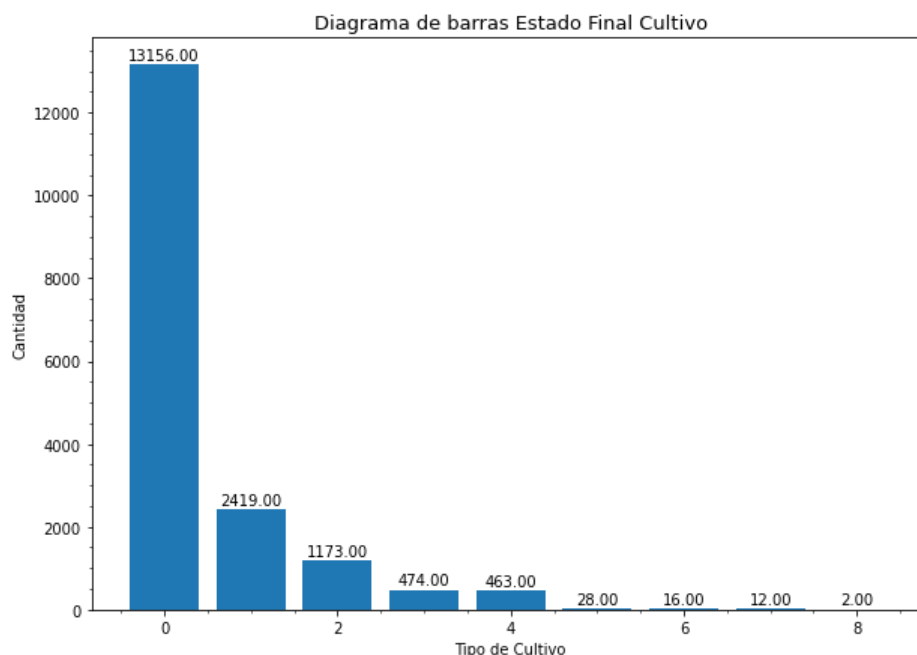


```

6      , 'Cana o verde':4
7      , 'Paddy o cascara verde':5
8      , 'Fibra o cabuya':6
9      , 'Aceite crudo':7
10     , 'Algodon semilla':8
11 })

```

Inicialmente, se elaboró un diagrama de barras para observar la distribución de los datos. El resultado evidenció un desequilibrio en la distribución.



Dado el desbalanceo en los datos, consideramos que las métricas más apropiadas para este caso debían ser la matriz de confusión, para observar los aciertos en cada una de las predicciones por clase, así como las métricas F1 Macro y Weighted. Estas métricas otorgan igual importancia a cada clase que se predice, de modo que un valor cercano a 1 implicaría que el modelo de predicción es capaz de predecir cada una de las clases con alta precisión.

```

1 def evaluacion(y_test,y_predict):
2     print('Evaluacion del Modelo'.center(75,' '),'\n')
3     print(f'\tF1_Score Macro: {f1_score(y_test,y_predict,average= "
4         macro"):.3f} ')
5     print(f'\tF1_Score Weighted: {f1_score(y_test,y_predict,average= "
6         weighted"):.3f} ')
7     print(classification_report(y_test,y_predict))
8
9 def Matrix(y_test,y_predict):
10    print(' '.center(75,' '))

```

```

9     cm = confusion_matrix(y_test,y_predict)
10    plt.figure(figsize=(9,6))
11    plt.title('Matriz de confusion multiclase',fontsize=17)
12    sns.heatmap(cm,annot=True,cmap='Greens',fmt='d')
13    plt.ylabel('Predicciones',fontsize = 14)
14    plt.xlabel('Reales',fontsize = 14)
15    plt.show()

```

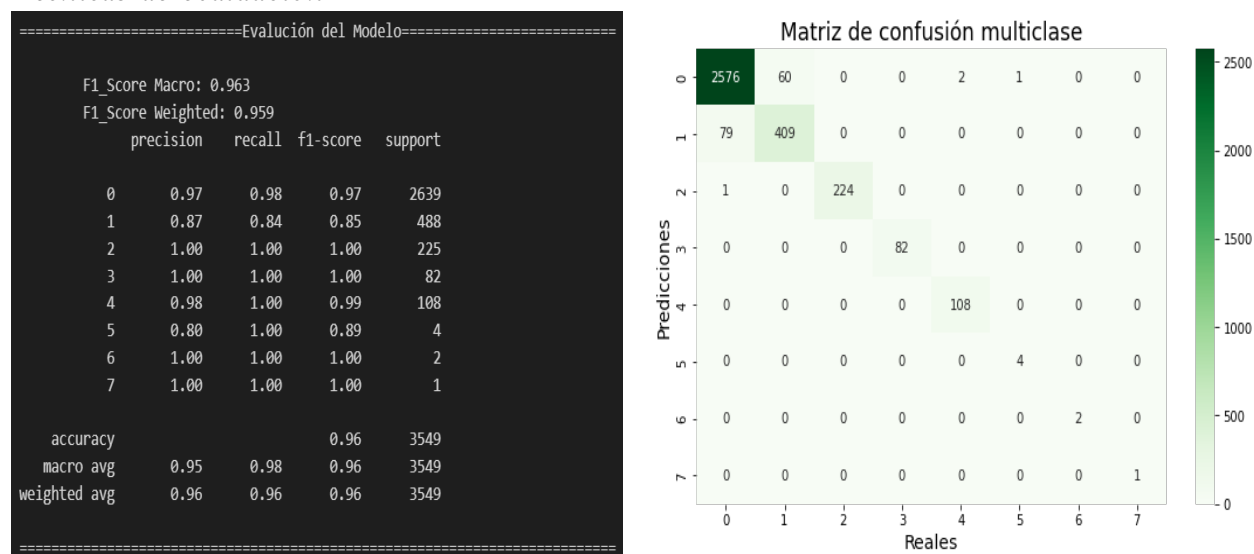
Los modelos de predicción que se seleccionaron fueron, *árboles de clasificación*, *Bosques aleatorios de clasificación* y *SVM de clasificación*

## Árboles de Clasificación

Para este modelo, se utilizaron técnicas de PCA (Análisis de Componentes Principales) para reducir la dimensionalidad de los atributos, así como técnicas de estandarización para establecer una escala uniforme para cada dato. Los resultados obtenidos fueron los siguientes:

**Figura 2**

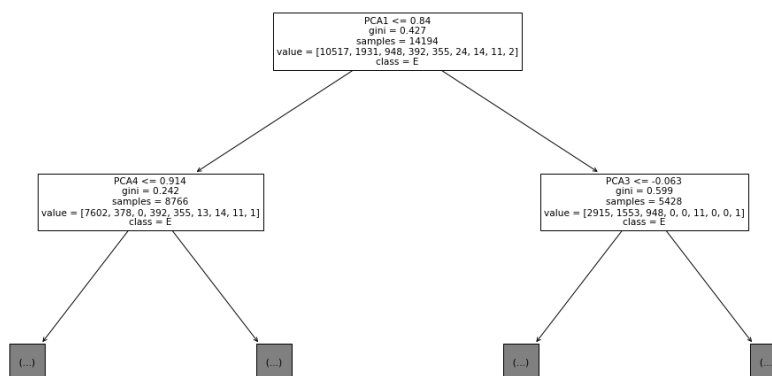
### Técnicas de evaluación



Para comprender un poco los parámetros de decisión tomados para cada nodo del árbol, se presenta una aproximación a 2 nodos en la siguiente figura.

Figura 3

Diagrama de árbol de decisión para dos nodos



## Bosques Aleatorios de Clasificación

Para este modelo de predicción de usaron técnicas de hiperparametros con el fin de mejorar los rendimiento de los modelos, cuyos parámetros mas apropiados fueron los que se representan a continuación.

Listing 1: Hiperparametros modelo de Bosques Aleatorios

```

1 from sklearn.model_selection import GridSearchCV
2
3 paraemtros_FC = {'criterion':('gini','entropy'),
4                  'max_depth':(8,16),
5                  'n_estimators':(100,150)}
6 hiperParametros_Model_FC = GridSearchCV(model_Forest_classifier,
7     paraemtros_FC,scoring=['f1_macro','f1_weighted'],refit='f1_macro')

```

modelo:

```

1 from sklearn.ensemble import RandomForestClassifier
2 from sklearn.model_selection import train_test_split
3 #creacion del modelo de bosques aleatorios
4 model_Forest_classifier = RandomForestClassifier(n_estimators=100
5     ,criterion='gini'
6     ,max_depth=16,
7     random_state=42)
8 #Division del conjunto de datos en train y test del 0.2
9 X_train_FC,X_test_FC,y_train_FC,y_test_FC = train_test_split(X_clas_FC,
10     y_clas_FC,test_size=0.2,random_state=42)
11
12 model_Forest_classifier.fit(X_train_FC,y_train_FC)
13 y_predict_clas_FC = model_Forest_classifier.predict(X_test_FC)

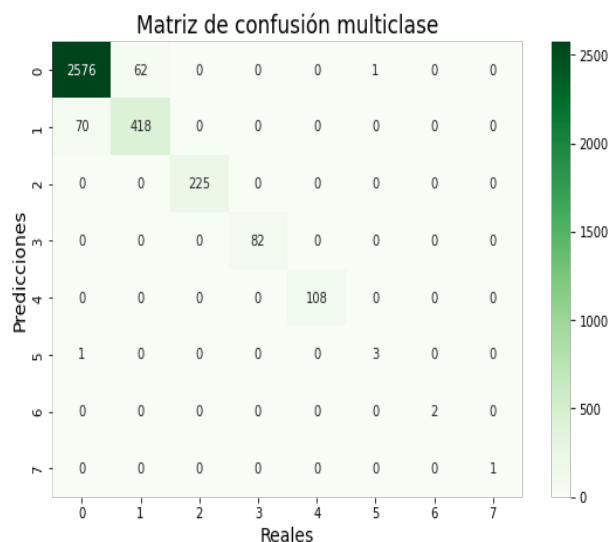
```

Resultados:

**Figura 4**

### *Técnicas de evaluación*

=====Evaluación del Modelo=====				
F1_Score Macro: 0.949				
F1_Score Weighted: 0.962				
	precision	recall	f1-score	support
0	0.97	0.98	0.97	2639
1	0.87	0.86	0.86	488
2	1.00	1.00	1.00	225
3	1.00	1.00	1.00	82
4	1.00	1.00	1.00	108
5	0.75	0.75	0.75	4
6	1.00	1.00	1.00	2
7	1.00	1.00	1.00	1
accuracy			0.96	3549
macro avg	0.95	0.95	0.95	3549
weighted avg	0.96	0.96	0.96	3549
=====				



Mediante la matriz de confusión se evidencia que la clase llamada 5, presenta falencias y su precisión a la hora de predecir es de tan solo 75 %.

## SVM de Clasificación

Se utilizó el método de Support Vector Machines (SVM) para la clasificación. Los parámetros del modelo se ajustaron utilizando diferentes kernels, y el más óptimo, que obtuvo las mejores métricas para F1\_macro y Weighted, fue el kernel **linear**.

```
1 from sklearn.svm import SVC
2
3 Model_SVC = SVC(kernel='linear')
4 Model_SVC.fit(X_train,y_train)
5 y_predict_clas_SVC = Model_SVC.predict(X_test)
6 evaluacion(y_test,y_predict_clas_SVC)
7 Matrix(y_test,y_predict_clas_SVC)
```

Hiperparametros:

```
1 parametros_SVC = {'kernel':('linear','poly','rbf')}
2 hiperparametros_model_SVC = GridSearchCV(Model_SVC,
3                                           parametros_SVC,
4                                           scoring=['f1_macro','f1_weighted'],refit='f1_macro')
```

```
5 hiperparametros_model_SVC.fit(X_train,y_train)
```

Cuyos resultados fueron:

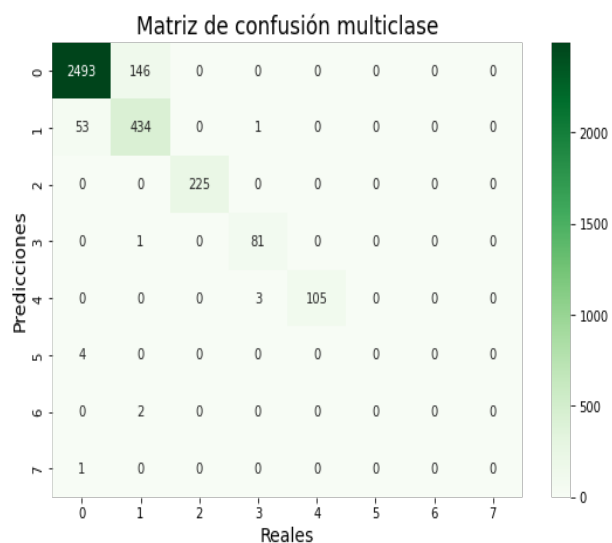
**Figura 5**

### *Técnicas de evaluación*

```
=====Evaluación del Modelo=====
```

F1_Score Macro: 0.591				
F1_Score Weighted: 0.942				
	precision	recall	f1-score	support
0	0.98	0.94	0.96	2639
1	0.74	0.89	0.81	488
2	1.00	1.00	1.00	225
3	0.95	0.99	0.97	82
4	1.00	0.97	0.99	108
5	0.00	0.00	0.00	4
6	0.00	0.00	0.00	2
7	0.00	0.00	0.00	1
accuracy			0.94	3549
macro avg	0.58	0.60	0.59	3549
weighted avg	0.94	0.94	0.94	3549

```
=====
```



Este modelo mostró la menor capacidad para predecir las distintas clases, teniendo una gran dificultad con las clases 5, 6 y 7, resultando en un valor de **F1\_macro de 0.58**. Por tal motivo, este modelo no cumple con los estándares de rendimiento esperados.

## Validación Cruzada

Dado que los datos presentan un desbalanceo se opta por trabajar con una validación cruzada estratificada y tomando como métrica principal de la evaluación `f1_macro`

**Figura 6**

*Consolidado de métricas para conjuntos de datos con validación cruzada*



*Note.* Gráfico de barras de `F1_macro` y `Weighted` para 5 divisiones del conjunto de datos, aplicado para los modelos de árboles de decisión, bosques aleatorios y SVM. La línea puntea de color rojo representa el valor promedio entre los 5 resultados de acá validación

## Conclusiones

- El desbalanceo en la distribución de los datos afectó negativamente el rendimiento de los modelos, especialmente en la predicción de clases menos representadas
- El mejor modelo implementado, que mostró el mejor rendimiento para la clasificación a pesar del desbalanceo de las clases, fue el ***Bosques Aleatorios de Clasificación***. Este modelo logró un valor mínimo en la métrica F1\_macro de 93 %, un valor máximo de 98 % y un valor promedio de 96 %