



EU LAND COVER CLASSIFICATION

POR:

ALCIDES ANTONIO ZAPATA MENDEZ
LEONEL ALFONSO ZAPATA MENDEZ
CARLOS ENRIQUE MORALES RODRÍGUEZ

MATERIA:

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL
MODELOS DE SISTEMAS 1

PROFESOR:

RAÚL RAMOS POLLÁN

UNIVERSIDAD DE ANTIOQUIA FACULTAD DE INGENIERÍA ORIENTE - BAJO CAUCA
2023

CONTENIDO

INTRODUCCIÓN

1. Planteamiento del problema

1.1. Dataset

1.2. Métrica

1.3. Variable Objetivo

2. Exploración de variables

2.1. Análisis de la variable objetivo

2.2. Exploración de variables

2.3. Análisis de la variable objetivo

2.4. Datos faltantes

2.5. Correlación de variables

3. Separación de la base de datos en variable de entrenamiento y prueba

3.1. Ajuste de la métrica:

3.2. Clasificación del conjunto de datos

3.3. Código selección del mejor modelo

4. Métodos de predicción escogidos.

4.1. DecisionTreeClassifier

4.2. RandomForestClassifier

5. Conclusiones

Bibliografía

INTRODUCCIÓN

En la actualidad, la inteligencia artificial (IA) se ha convertido en un catalizador poderoso para abordar desafíos complejos en diversas disciplinas. En particular, su aplicación en proyectos como este, destinados a la clasificación de la cobertura del suelo en Europa, representa un ejemplo significativo de cómo la IA puede transformar nuestra comprensión y manejo de datos a gran escala.

El uso de técnicas de inteligencia artificial y aprendizaje automático no solo amplía los límites de nuestro conocimiento actual, sino que también permite un análisis más profundo y detallado de conjuntos de datos complejos. En este proyecto específico, la IA se utiliza para predecir la clasificación de 72 clases de cobertura del suelo en un conjunto de datos masivos que abarcan cinco años distintos, aprovechando más de 42,000 muestras in situ y 416 características derivadas de datos de teledetección e imágenes satelitales.

La IA facilita la identificación y comprensión de patrones complejos en datos multidimensionales, permitiendo una clasificación más precisa y detallada de la cubierta terrestre. La capacidad de estos modelos para evaluar y procesar múltiples clases y características proporciona una visión más completa y detallada de la dinámica del paisaje europeo a lo largo del tiempo.

Además, la inteligencia artificial no solo se destaca por su capacidad para procesar grandes volúmenes de datos, sino también por su capacidad de adaptación. En este contexto, la capacidad de los modelos para mantener una alta precisión a lo largo de diferentes períodos de tiempo es fundamental. La IA ofrece la flexibilidad necesaria para ajustarse a cambios temporales y proporcionar resultados precisos en entornos cambiantes.

En resumen, este proyecto no solo destaca el poder y la utilidad de la inteligencia artificial en la clasificación de la cobertura del suelo en Europa, sino que también ilustra cómo la IA puede ser una herramienta vital para abordar desafíos complejos en la comprensión y el análisis de grandes conjuntos de datos, ofreciendo nuevas perspectivas y aplicaciones en el ámbito medioambiental y geoespacial.

1. Planteamiento del problema

En Europa, la clasificación de la cobertura del suelo es fundamental para comprender y monitorear los cambios en el paisaje. Este desafío involucra la predicción de 72 clases de cobertura del suelo a lo largo de cinco años usando datos de teledetección e imágenes satelitales. El objetivo es desarrollar modelos de aprendizaje automático que puedan predecir con precisión la cobertura del suelo en un conjunto de pruebas.

1.1. Dataset

Vamos a usar el dataset de kaggle esta competición (<https://www.kaggle.com/competitions/oemc-hackathon-eu-land-cover-classification/data>). El desafío de mapear la cobertura terrestre en Europa se aborda con un conjunto de datos detallado. Este dataset contiene 72 clases de cobertura, como terrenos despejados, praderas sin árboles, bosques de coníferas, viñedos, entre otros. Cubre cinco años (2006, 2009, 2012, 2015 y 2018) con más de 42,000 muestras en el terreno y 416 características derivadas de teledetección. Un conjunto de prueba adicional de 42,000 muestras se usa para evaluar modelos de aprendizaje automático. Este conjunto ofrece una base sólida y desafiante para el desarrollo de modelos de clasificación de cobertura terrestre en Europa a lo largo del tiempo, con datos de entrenamiento que contienen 42,237 filas y 420 columnas con metadatos detallados sobre las variables

1.2. Métrica

La métrica de evaluación principal para el modelo será el error logarítmico medio cuadrático (RMSLE), el cual se calcula mediante la siguiente expresión:

$$\epsilon = \sqrt{\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i + 1))^2}$$

Donde ϵ es el valor del RMSLE, n es el número total de observaciones en el dataset, p_i es la predicción de la variable objetivo y a_i es el valor real de la variable objetivo.

La métrica para medir el desempeño de los modelos es el RMSLE (Root Mean Squared Logarithmic Error), sin embargo, ya que aplicamos una transformación logarítmica sobre la variable objetivo (land_cover_label), podemos usar directamente el RMSE (Root Mean Squared Error) para obtener el valor de la métrica. Para evaluar la precisión de los modelos de aprendizaje automático en la clasificación de 72 clases de cobertura del suelo en Europa, se empleará la puntuación F1 ponderada. Esta métrica equilibra precisión y recall, esencial al manejar múltiples clases. La fórmula se basa en precisión y exhaustividad, promediada

según el peso de cada clase en el conjunto de prueba. Una alta puntuación F1 ponderada indicará un rendimiento sólido en la clasificación de la cobertura del suelo. El énfasis está en obtener una alta puntuación F1 ponderada para todas las clases, lo que reflejará un desempeño óptimo en este desafío de mapeo de la cobertura terrestre en Europa.

1.3. Variable Objetivo

La variable objetivo es la clasificación de las 72 clases de cobertura del suelo, representando diversos paisajes y vegetación en Europa.

2. Exploración de variables

Se analiza la distribución y relevancia de la variable objetivo, tipos de medidas utilizadas, el uso primario de la edificación, consumo de energía, datos faltantes, correlaciones y distribuciones de variables numéricas.

2.1. Análisis de la variable objetivo

El análisis de la variable objetivo en este proyecto implica comprender la distribución, relaciones y cambios temporales de las 72 clases de cobertura del suelo en Europa. Este análisis es fundamental para el desarrollo preciso de modelos que puedan predecir con exactitud la dinámica de la cubierta terrestre a lo largo del tiempo.

2.2. Exploración de variables

Para iniciar con la exploración de variables lo primero que se hace es juntar la información que esta distribuida en los dataset “oemc-hackathon-eu-land-cover-classification”, “land_cover” y “train.csv”, de forma de que se obtiene un dataset con todas las variables disponibles, dicho dataset se deja con el nombre de “train” ya que será con el cual se trabajará. Luego de hacer este procedimiento, se analizan algunas variables que se consideran de interés.

2.3. Análisis de la variable objetivo

Para empezar se carga la base de datos se lee el archivo train.csv y se pide que se impriman las 10 primeras filas, con el fin de tener una aproximación a la estructura de la base de datos.

sample_id	land_cover	land_cover_label	year	evi_mod13q1.stl.trend_p90_250m_jan01_feb28	lst_mod11a2.nighttime.trend_p50_1km_jan01_jan31	lst_mod11a2.nighttime.trend_p50_1km_feb01_feb28	lst_mod11a2.nighttime.trend_p
0	0	16	Grassland without tree/shrub cover	2006	3630.0	13868.0	13877.0
1	1	10	Coniferous woodland	2006	2851.0	13810.0	13815.0
2	2	15	Grassland with sparse tree/shrub cover	2018	3973.0	13958.0	13961.0
3	3	16	Grassland without tree/shrub cover	2006	4401.0	13793.0	13796.0
4	4	58	Shrubland with sparse tree cover	2012	4330.0	13868.0	13868.0
5	5	12	Dry pulses	2018	3124.0	14110.0	14106.0
6	6	16	Grassland without tree/shrub cover	2015	5640.0	14111.0	14109.0
7	7	16	Grassland without tree/shrub cover	2018	5006.0	14010.0	14011.0

figura1. Información dataset train.csv

Con el método `## KEEPOUTPUT print (train.shape)print (test.shape)` se verifica cuántas filas y columnas tiene la base de datos.

En la siguiente imagen se puede apreciar como están distribuidos los datos en la variable objetivo.

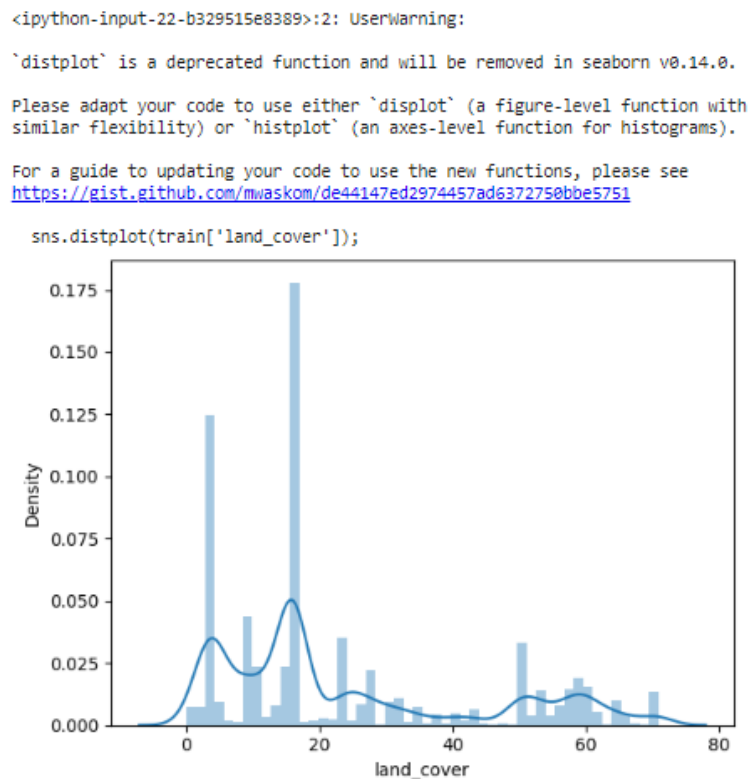


Figura2. Distribución de los datos variable objetivo.

2.4 Datos faltantes

Nuestro dataset no tenía datos faltantes por lo cual se eliminan el 10% de los datos aleatoriamente a través del siguiente código esto para cumplir con los criterios de la entrega.

```
#import pandas as pd
#import numpy as np # Importa NumPy para generar valores aleatorios
```

```

# Supongamos que 'train' es tu DataFrame cargado previamente
# Define la fracción de celdas que deseas eliminar (por ejemplo, 10%)
fraccion_a_eliminar = 0.1
# Calcula el número total de celdas en el DataFrame
total_celdas = train.size
# Calcula la cantidad de celdas a eliminar
celdas_a_eliminar = int(total_celdas * fraccion_a_eliminar)
# Genera una lista de índices aleatorios para seleccionar celdas al azar
indices_aleatorios = np.random.choice(train.index, size=celdas_a_eliminar, replace=True)
# Itera sobre los índices aleatorios y establece el valor en esas celdas como NaN (o cualquier otro valor que elijas)
for indice in indices_aleatorios:
    columna = np.random.choice(train.columns) # Elige una columna aleatoriamente
    train.at[indice, columna] = np.nan # Establece el valor como NaN
# Ahora, algunas celdas aleatorias en 'train' contienen valores NaN
train.head(10)

```

sample_id	land_cover	land_cover_label	year	evi_mod13q1.st1.trend_p90_250m_jan01_feb28	lst_mod11a2.nighttime.trend_p50_1km_jan01_jan31	lst_mod11a2.nighttime.trend_p50_1km_feb01_feb28	lst_mod11a2.nighttime.trend
0	0.0	16.0	Grassland without tree/shrub cover	2006.0	3630.0	13868.0	13877.0
1	1.0	10.0	Coniferous woodland	NaN	2851.0	13810.0	13815.0
2	2.0	15.0	Grassland with sparse tree/shrub cover	2018.0	3973.0	13958.0	13961.0
3	3.0	16.0	Grassland without tree/shrub cover	2006.0	4401.0	13793.0	13796.0
4	4.0	NaN	Shrubland with sparse tree cover	2012.0	4330.0	13868.0	13868.0
5	5.0	12.0	NaN	NaN	3124.0	14110.0	14106.0
6	NaN	16.0	Grassland without tree/shrub cover	2015.0	5640.0	14111.0	14109.0
7	7.0	16.0	Grassland without tree/shrub cover	2018.0	5006.0	14010.0	14011.0
8	8.0	51.0	Pine dominated mixed woodland	2018.0	3845.0	14002.0	14002.0
9	9.0	NaN	Grassland with sparse tree/shrub cover	2012.0	3889.0	14054.0	14053.0

10 rows x 420 columns

figura3. tabla con datos despues de la eliminacion.

Luego se realiza el llenado de los datos faltantes NAN teniendo en cuenta el tipo de datos de las variables a través del siguiente código utilizando la media para los datos numéricos y la moda para los categórico numericos.

```

# Rellenar los valores NaN con la media de cada columna
# Fill NaNs for numerical columns with the mean or median
numerical_cols = train.select_dtypes(include=['float64', 'int64']).columns
train[numerical_cols] = train[numerical_cols].apply(lambda x: x.fillna(x.median()))

# Fill NaNs for categorical columns with the mode
categorical_cols = train.select_dtypes(include=['object']).columns
train[categorical_cols] = train[categorical_cols].apply(lambda x: x.fillna(x.mode()[0]))
train.head(10)

```

sample_id	land_cover	land_cover_label	year	evi_mod13q1.stl.trend_p50_250m_jan01_feb28	lst_mod11a2.nighttime.trend_p50_1km_jan01_jan31	lst_mod11a2.nighttime.trend_p50_1km_feb01_feb28	lst_mod11a2.nighttime.trend
0	0.0	16.0	Grassland without tree/shrub cover	2006.0	3630.0	13066.0	13077.0
1	1.0	10.0	Coniferous woodland	2012.0	2051.0	13010.0	13015.0
2	2.0	15.0	Grassland with sparse tree/shrub cover	2018.0	3973.0	13958.0	13961.0
3	3.0	16.0	Grassland without tree/shrub cover	2006.0	4401.0	13793.0	13796.0
4	4.0	16.0	Shrubland with sparse tree cover	2012.0	4330.0	13068.0	13068.0
5	5.0	12.0	Grassland without tree/shrub cover	2012.0	3124.0	14110.0	14106.0
6	21104.0	16.0	Grassland without tree/shrub cover	2015.0	5640.0	14111.0	14109.0
7	7.0	16.0	Grassland without tree/shrub cover	2018.0	5006.0	14010.0	14011.0
8	8.0	51.0	Pine dominated mixed woodland	2018.0	3845.0	14002.0	14002.0
9	9.0	16.0	Grassland with sparse tree/shrub cover	2012.0	3889.0	14054.0	14053.0

10 rows x 420 columns

Figura4. tabla con datos llenos.

Posteriormente se hace una conversión de variables numéricas a categóricas para cumplir con los requisitos de la entrega final para tal fin utilizamos el siguiente código.

Convertir columnas a categóricas

```
columnas_a_convertir = ['lst_mod11a2.nighttime.trend_p50_1km_jan01_jan31',
'slst_mod11a2.nighttime.trend_p50_1km_feb01_feb28','slope.percent_gedi.eml_m_30m_2000_2018']
```

```
for columna in columnas_a_convertir:
    train[columna] = train[columna].astype('category')
```

Mostrar el DataFrame después de la conversión

```
print("DataFrame después de convertir columnas a categóricas:")
train.head(10)
```

Con el siguiente código nos muestra que variables quedaron categorías

```
[19] ## KEEPOUTPUT
ccols = [i for i in train.columns if not i in train._get_numeric_data()]
print(ccols)

['land_cover_label', 'lst_mod11a2.nighttime.trend_p50_1km_jan01_jan31', 'lst_mod11a2.nighttime.trend_p50_1km_feb01_feb28', 'slope.percent_gedi.eml_m_30m_2000_2018']
```

Figura5. variables categóricas

Para un mejor procesamiento por parte de los modelos que utilizamos convertimos todas las variables categóricas excepto la variable land cover label a variables categóricas numéricas a través del siguiente código.

Identificar columnas categóricas

```
ccols = [i for i in train.columns if not i in train._get_numeric_data()]
```

Asignar categorías numéricas a las variables categóricas

```
for columna in ccols:
    if columna != 'land_cover_label':
```



```

train[columna] = train[columna].astype('category').cat.codes

# Mostrar el DataFrame después de la conversión
print("DataFrame después de asignar categorías numéricas a variables categóricas:")
train.head(10)

```

2.5 Correlación de variables

La importancia también radica en entender la correlación entre las distintas variables y la variable principal o objetivo. Además, se puede notar que, en términos generales, las variables relacionadas muestran una correlación tan baja que prácticamente se puede concluir que no están vinculadas con la variable objetivo.

Este código genera la matriz de correlación para el DataFrame train y muestra las primeras 10 filas de esa matriz.

```

## KEEPOUTPUT
#correlation matrix
corrmat = train.corr()
f, ax = plt.subplots(figsize=(12, 9))
# sns.heatmap(corrmat, vmax=.8, square=True);
corrmat

```

Como resultado final sería la visualización de las primeras 10 filas de la matriz de correlación en formato tabular, mostrando las correlaciones entre las variables del conjunto de datos.

nir_mod13q1_sd_250m_2000_2022	-0.006620	-0.144077	-0.013879	0.302692	-0.289918	-0.2883
pop.count_ghs.jrc_m_100m	-0.002247	-0.041573	0.032384	-0.134226	0.068705	0.0661
ost.distance.to.coast_gedi.grass.gis_30m_2000_2018	0.006186	-0.003635	0.054496	-0.233076	-0.187420	-0.1898
slope.percent_gedi.eml_m_30m_2000_2018	-0.004166	0.017729	0.111673	-0.060800	0.066626	0.0659
elev.lowestmode_gedi.eml_m_30m_2000_2018	0.001202	0.038034	0.109487	-0.275437	-0.118398	-0.1188

9 rows x 419 columns

Figura6. Matriz de correlación formato tabular

Con el siguiente código calcula la matriz de correlación absoluta para identificar relaciones fuertes entre variables en el DataFrame train. Se establece un umbral de 0.8 para determinar correlaciones altas. Luego, se selecciona la parte superior del triángulo de la matriz para evitar duplicaciones. Se identifican las columnas con correlaciones altas, las cuales podrían tener multicolinealidad. Estas columnas se imprimen como candidatas para eliminación. es decir, se analiza y muestra las columnas que podrían eliminarse debido a correlaciones altas, ayudando a reducir la redundancia en el conjunto de datos.

```

import pandas as pd
import numpy as np

```

```
# Assuming df is your DataFrame
corr_matrix = train.corr().abs()
# Set the threshold for high correlation
high_corr_threshold = 0.8
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# Find index of feature columns with correlation greater than threshold
to_drop = [column for column in upper.columns if any(upper[column] > high_corr_threshold)]
print("Columns to drop:", to_drop)
```

Al final de la depuración con el código `len(to_drop)` se muestran 258 columnas que no de gran interés para las predicciones. Hacemos una copia de la base de datos a través de `train_copy= train.copy(deep=True)` para conservar la información original y trabajar sobre la copia. Luego se utilizó el código `train_copy.drop(columns=to_drop)` el resultado fue la impresión de las columnas que tienen una alta correlación (mayor que 0.8) con al menos una otra columna en el conjunto de datos. Estas son las columnas que podrían eliminarse para reducir la multicolinealidad en el conjunto de datos. con el código

```
## KEEPOUTPUT
#correlation matrix
corrmat = train_copy.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
# corrmat
```

Como resultado nos muestra un mapa de calor que visualiza la matriz de correlación del DataFrame `train_copy`. Este tipo de visualización es útil para identificar patrones de correlación entre las variables del conjunto de datos. Las áreas más oscuras representan correlaciones más fuertes, mientras que las áreas más claras representan correlaciones más débiles o nulas.

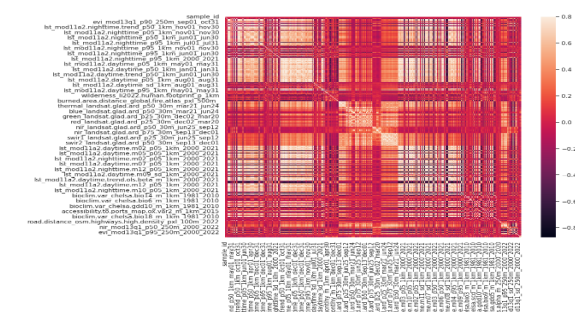


Figura7. mapa de calor correlación

```
corr_flat = corrmat.unstack()
corr_flat = corr_flat[corr_flat.index.get_level_values(0) != corr_flat.index.get_level_values(1)]
```

```
sort_by_pairs = corr_flat.sort_values()
sort_by_pairs[:10]
```

Con el código anterior se muestra una lista de los 10 pares de variables que tienen las correlaciones más bajas en el conjunto de datos. Cada entrada de la lista contendrá dos variables y su valor de correlación. Esto puede proporcionar información valiosa sobre las relaciones más débiles o insignificantes entre las variables del conjunto de datos.

3. Separación de la base de datos en variable de entrenamiento y prueba

El siguiente código realiza la preparación de datos típica para el entrenamiento y la evaluación de modelos de aprendizaje automático. Separar los datos en conjuntos de entrenamiento y prueba es esencial para evaluar la capacidad de generalización de un modelo en datos no vistos durante el entrenamiento.

```
from sklearn.model_selection import train_test_split

# Supongamos que 'X' son tus características y 'y' son tus etiquetas en tu DataFrame 'train'
# Asegúrate de tener 'X' e 'y' correctamente definidos según tus datos
# Por ejemplo, si tus columnas son 'feature1', 'feature2', ..., 'target'
X = train_copy.drop('land_cover_label', axis=1) # 'X' son todas las columnas excepto la
columna 'target'
y = train_copy['land_cover_label'] # 'y' es la columna 'target'
# Dividir los datos en conjuntos de entrenamiento y prueba
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3.1 Ajuste de la métrica:

Como se mencionó anteriormente, la métrica para medir el desempeño de los modelos será el RMSLE (Root Mean Squared Logarithmic Error), con el siguiente código define una función llamada `report_cv_score` que toma un diccionario `z` (presumiblemente los resultados de una validación cruzada) y calcula y muestra el Root Mean Squared Logarithmic Error (RMSLE) tanto en los conjuntos de prueba como en los de entrenamiento. La función `report_cv_score` facilita la presentación del rendimiento de un modelo mediante la impresión del RMSLE promedio y la desviación estándar para los conjuntos de prueba y entrenamiento durante la validación cruzada.

```
from sklearn.model_selection import train_test_split

# Supongamos que 'X' son tus características y 'y' son tus etiquetas en tu DataFrame 'train'
# Asegúrate de tener 'X' e 'y' correctamente definidos según tus datos
# Por ejemplo, si tus columnas son 'feature1', 'feature2', ..., 'target'
X = train_copy.drop('land_cover_label', axis=1) # 'X' son todas las columnas excepto la
columna 'target'
y = train_copy['land_cover_label'] # 'y' es la columna 'target'
```

```
# Dividir los datos en conjuntos de entrenamiento y prueba
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Luego la función RMSLE proporciona una manera conveniente de calcular el RMSLE entre las etiquetas reales y las predicciones del modelo. Este tipo de métrica es comúnmente utilizada en problemas de regresión, especialmente cuando las variables objetivo tienen una escala logarítmica, como en problemas de predicción de precios.

```
#Función para calcular el RMSLE de los modelos implementados
```

```
def RMSLE(y_actual, y_pred):
```

```
    return np.sqrt(mean_squared_error(y_actual, y_pred))
```

con el siguiente código se inicializan tres modelos de regresión con diferentes algoritmos y configuraciones. Estos modelos pueden ser utilizados posteriormente para entrenar y realizar predicciones en conjuntos de datos específicos.

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
# Ahora puedes crear instancias de los estimadores
```

```
estimator1 = LinearRegression()
```

```
estimator2 = DecisionTreeRegressor(max_depth=5)
```

```
estimator3 = RandomForestRegressor(n_estimators=2, max_depth=5)
```

Utilizamos OneHotEncoder para convertir las etiquetas categóricas en una representación one-hot, que es comúnmente utilizada en problemas de clasificación para manejar variables objetivo categóricas.

```
from sklearn.preprocessing import OneHotEncoder
```

```
onehot_encoder = OneHotEncoder(sparse=False)
```

```
y_train_encoded = onehot_encoder.fit_transform(y_train.values.reshape(-1, 1))
```

3.2 Clasificación del conjunto de datos

Realizamos la clasificación de un conjunto de datos utilizando un clasificador de regresión logística y evaluamos su rendimiento mediante validación cruzada. Realiza la clasificación de un conjunto de datos utilizando un clasificador de regresión logística y evaluamos su rendimiento mediante validación cruzada a través del siguiente código.

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.preprocessing import LabelEncoder
```

```

from sklearn.model_selection import cross_validate, ShuffleSplit
from sklearn.metrics import mean_squared_error
# Codificar las etiquetas
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
# Definir el clasificador (puedes ajustar los parámetros según sea necesario)
classifier = LogisticRegression()
# Realizar la validación cruzada
z = cross_validate(classifier, x_train, y_train_encoded, return_train_score=True,
                  return_estimator=False, scoring="neg_mean_squared_error",
                  cv=ShuffleSplit(n_splits=10, test_size=val_size))
# Informar sobre el resultado
report_cv_score(z)

```

3.3. Código selección del mejor modelo

Para entrenar los modelos se usará el dataset que se generó, llamado *train*. De todo el conjunto de estos datos es necesario hacer una partición para entrenar y otra para prueba. El siguiente código realiza la comparación de varios clasificadores (regresión logística, bosques aleatorios y árboles de decisión) en términos de su rendimiento utilizando validación cruzada. realiza una comparación de diferentes clasificadores utilizando validación cruzada y selecciona el modelo con el mejor rendimiento basado en el error cuadrático medio negativo. El rendimiento se mide en términos de la raíz cuadrada del error cuadrático medio (RMSLE).

```

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_validate, ShuffleSplit
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
# Codificar las etiquetas
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)

# Definir los clasificadores (puedes ajustar los parámetros según sea necesario)
classifiers = [LogisticRegression(), RandomForestClassifier(), DecisionTreeClassifier()]
best_model = None
best_score = 0 # Inicializamos con un valor bajo para la mejor precisión

```

```

for classifier in classifiers:
    print("-----")
    z = cross_validate(classifier, x_train, y_train_encoded, return_train_score=True,
                      return_estimator=False, scoring="accuracy",
                      cv=ShuffleSplit(n_splits=10, test_size=val_size))

    # Calcular la precisión
    accuracy = np.mean(z['test_score'])
    print("Accuracy: {:.5f}".format(accuracy))

    # Actualizar el mejor modelo si es necesario
    if accuracy > best_score:
        best_score = accuracy
        best_model = classifier

# Imprimir información sobre el mejor modelo
print("\nMejor modelo:")
print(best_model)

```

Como resultado arrojó que el mejor modelo es DecisionTreeClassifier()

```

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
options={"iprint": iprint, "gtol": tol, "maxiter": max_iter},
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=2):
ABNORMAL_TERMINATION_IN_LNSRCH.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
options={"iprint": iprint, "gtol": tol, "maxiter": max_iter},
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=2):
ABNORMAL_TERMINATION_IN_LNSRCH.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
options={"iprint": iprint, "gtol": tol, "maxiter": max_iter},
Accuracy: 0.24967
-----
Accuracy: 0.52462
-----
Accuracy: 0.75867

Mejor modelo:
DecisionTreeClassifier()

```

Figura7. Mejor Modelo

4. Métodos de predicción escogidos.

Utilizamos el clasificador de árbol de decisión (DecisionTreeClassifier) de scikit-learn para entrenar un modelo de clasificación. Entrena el modelo utilizando el conjunto de entrenamiento (X_train e y_train). Durante el entrenamiento, el clasificador aprenderá a

realizar predicciones basadas en las características proporcionadas. Este código se utiliza para entrenar un modelo de clasificación de árbol de decisión en un conjunto de datos dividido en conjuntos de entrenamiento y prueba. Después de ejecutar este código, el modelo entrenado se almacenará en la variable `tree_classifier` y estará listo para hacer predicciones sobre nuevos datos.

4.1 DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Supongamos que X_train, X_test, y_train, y_test son tus conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(x_train, y_train_encoded, test_size=0.2, random_state=42)

# Crea una instancia del DecisionTreeClassifier
tree_classifier = DecisionTreeClassifier()

# Entrenar el modelo
tree_classifier.fit(X_train, y_train)
```

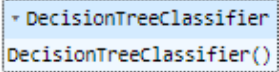


Figura8. DecisionTreeClassifier

con el siguiente código

```
# Realizar predicciones en el conjunto de prueba
y_pred = tree_classifier.predict(x_test)
```

Este código utiliza el clasificador de árbol de decisión (`DecisionTreeClassifier`) previamente entrenado para realizar predicciones en el conjunto de prueba (`x_test`). Utiliza el modelo de árbol de decisión (`tree_classifier`) para predecir las etiquetas en el conjunto de prueba (`x_test`). Las predicciones se almacenan en la variable `y_pred`. Después de ejecutar este código, `y_pred` contendrá las predicciones del modelo para el conjunto de prueba. Estas predicciones pueden compararse con las etiquetas reales (`y_test`) para evaluar el rendimiento del modelo. Puedes usar métricas de evaluación, como precisión, recall, matriz de confusión, etc., para analizar la calidad de las predicciones.

por último se calculó la predicción del modelo con el siguiente código

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

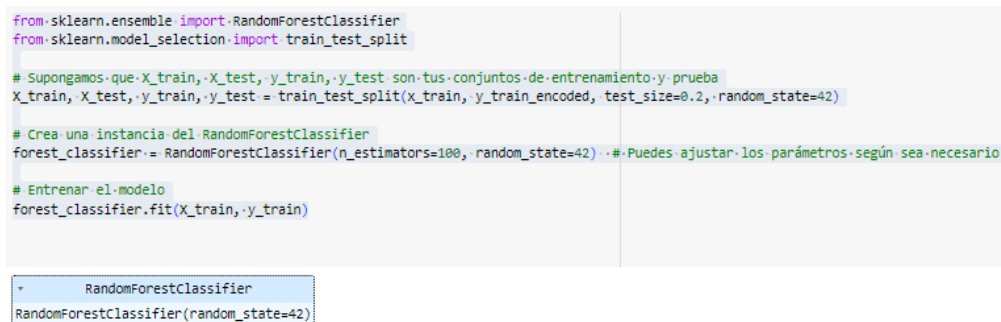
# Calcular la precisión
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
# Otros informes de clasificación
print(classification_report(y_test, y_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

Este código evalúa el rendimiento de un modelo de clasificación y proporciona varias métricas de evaluación: Calcula y muestra la matriz de confusión, que es una tabla que describe el rendimiento del modelo al clasificar instancias en términos de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos. Estas métricas son útiles para evaluar cómo el modelo de clasificación se desempeña en diferentes aspectos, como la precisión general y su capacidad para clasificar correctamente instancias positivas y negativas.

4.2 RandomForestClassifier



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Supongamos que X_train, X_test, y_train, y_test son tus conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_train_encoded, test_size=0.2, random_state=42)

# Crea una instancia del RandomForestClassifier
forest_classifier = RandomForestClassifier(n_estimators=100, random_state=42) # Puedes ajustar los parámetros según sea necesario

# Entrenar el modelo
forest_classifier.fit(X_train, y_train)
```

RandomForestClassifier
RandomForestClassifier(random_state=42)

Figura9. RandomForestClassifier

En la figura anterior se crea una instancia del clasificador Random Forest con 100 árboles (`n_estimators=100`) y una semilla (`random_state=42`) para reproducibilidad. Puedes ajustar estos parámetros según tus necesidades. Entrena el modelo Random Forest utilizando los datos de entrenamiento (`X_train` e `y_train`). este código prepara conjuntos de entrenamiento y prueba, crea un clasificador Random Forest y lo entrena con los datos de entrenamiento.

luego se realizan predicciones en el conjunto de prueba a traves del siguiente codigo

```
# Realizar predicciones en el conjunto de prueba
y_pred = forest_classifier.predict(x_test)
```

Finalmente se utiliza el clasificador Random Forest previamente entrenado (`forest_classifier`) para realizar predicciones en el conjunto de prueba (`x_test`). Almacena las predicciones en la variable `y_pred`. El resultado final es un conjunto de predicciones (`y_pred`) basado en el modelo de clasificación Random Forest aplicado a los datos del conjunto de prueba. Estas predicciones pueden ser comparadas con las etiquetas reales (`y_test`) para evaluar el

rendimiento del modelo.

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Calcular la precisión
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Otros informes de clasificación
print(classification_report(y_test, y_pred))
# Matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

5. Conclusiones

El uso de la inteligencia artificial en la clasificación de la cobertura del suelo en Europa ha representado un avance significativo en la comprensión y el análisis de datos a gran escala. Este proyecto, apoyado en técnicas de IA y aprendizaje automático, ha demostrado cómo la capacidad de procesar grandes volúmenes de datos, identificar patrones complejos y adaptarse a cambios temporales ha sido fundamental para obtener predicciones precisas a lo largo del tiempo.

La flexibilidad y el poder de adaptación de estos modelos de IA han permitido mantener altos niveles de precisión a lo largo de cinco años distintos, utilizando datos de teledetección e imágenes satelitales. Esto no solo ha proporcionado una visión detallada de la dinámica del paisaje europeo, sino que también resalta el potencial de la IA como una herramienta crucial en la comprensión de entornos medioambientales y geoespaciales complejos.

Este proyecto destaca cómo la inteligencia artificial es fundamental en el análisis de grandes conjuntos de datos, abriendo nuevas perspectivas y aplicaciones valiosas en la ciencia medioambiental y geoespacial

Bibliografía

ASHRAE - Great Energy Predictor III | Kaggle. (2021). Retrieved 16 December 2021, from <https://www.kaggle.com/c/ashrae-energy-prediction/overview/description>