

Actividad 3.7

Predicción de Riesgo de derrumbamiento Terremotos



ÍNDICE

Paso 1: Descargar y cargar los datos	3
Cargar los archivos	4
Paso 2: Preprocesamiento de datos	4
Paso 3: Selección de características con dendrogramas	5
Verificar las características de X:	6
Seleccionar Características y Eliminar en base al dendograma	6
Paso 4: Evaluar los modelos usando Lazy Predict	7
Paso 5: Entrenar los modelos: Random Forest, Decision Tree, Gradient Boosting y SVM	8
5.1: Random Forest Classifier, optimizado con RandomizedSearchCV	8
Primer archivo de submissions generado por el modelo de random forest	10
5.2: DecisionTreeClassifier con optimización RandomizedSearchCV	11
5.3: Gradient Boosting optimizando sus hiperparámetros con RandomizedSearchCV	12
5.4: modelo de clasificación utilizando XGBoost, optimizando sus hiperparámetros con RandomizedSearchCV	13
5.5: SVM con RandomizedSearchCV.	14
Comparaciones de las predicciones de los modelos utilizados:	16
Conclusiones sobre los dos mejores modelos	16
Comparación clave:	16
Últimos resultados de la competición	17
Repositorios y Enlaces	17

Actividad 3.7 - Predicción de Riesgo de derrumbamiento - Terremotos

→ Enlace Competición: <https://www.drivendata.org/competitions/57/nepal-earthquake/>

Paso 1: Descargar y cargar los datos

En este paso, descargamos los datos del reto de Driven Data y los cargaremos en Google Colab para su análisis.

```
1 # IVAN FALCON MONZON
2 import pandas as pd
3
4 # Descargar archivos desde GitHub
5 wget -O train_values.csv "https://raw.githubusercontent.com/IvanFalconMonzon/SNS_ACT3_7_IvanFalconMonzon/main/train_values.csv"
6 wget -O train_labels.csv "https://raw.githubusercontent.com/IvanFalconMonzon/SNS_ACT3_7_IvanFalconMonzon/main/train_labels.csv"
7 wget -O test_values.csv "https://raw.githubusercontent.com/IvanFalconMonzon/SNS_ACT3_7_IvanFalconMonzon/main/test_values.csv"

--2025-03-09 11:56:19-- https://raw.githubusercontent.com/IvanFalconMonzon/SNS_ACT3_7_IvanFalconMonzon/main/train_values.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23442727 (22M) [text/plain]
Saving to: 'train_values.csv'

train_values.csv 100%[=====] 22.36M --KB/s in 0.08s

2025-03-09 11:56:19 (282 MB/s) - 'train_values.csv' saved [23442727/23442727]

--2025-03-09 11:56:19-- https://raw.githubusercontent.com/IvanFalconMonzon/SNS_ACT3_7_IvanFalconMonzon/main/train_labels.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2330792 (2.2M) [text/plain]
Saving to: 'train_labels.csv'

train_labels.csv 100%[=====] 2.22M --KB/s in 0.01s
```

Los archivos se descargan automáticamente del repositorio de github (El enlace al repositorio está en la parte de abajo del pdf: Repositorio y enlaces).

Cargar los archivos

```
[3] 1 # IVAN FALCON MONZON
2 # Cargar los archivos
3 train_data = pd.read_csv('train_values.csv')
4 train_labels = pd.read_csv('train_labels.csv')
5 test_data = pd.read_csv('test_values.csv')
6
7 # Mostrar las primeras filas
8 print("Datos de entrenamiento:")
9 display(train_data.head())
10 print("\nEtiquetas de entrenamiento:")
11 display(train_labels.head())
12 print("\nDatos de prueba:")
13 display(test_data.head())
14
15 # Información general sobre los datos
16 print("\nInformación sobre los datos de entrenamiento:")
17 train_data.info()
```

→ Datos de entrenamiento:

	building_id	geo_level_1_id	geo_level_2_id	geo_level_3_id	count_floors_pre_eq	age	area_percentage	height_percentage	land_surface_sqm
0	802906	6	487	12198	2	30	6	5	
1	28830	8	900	2812	2	10	8	7	
2	94947	21	363	8973	2	10	5	5	
3	590882	22	418	10694	2	10	6	5	
4	201944	11	131	1488	3	30	8	9	

5 rows x 39 columns

Paso 2: Preprocesamiento de datos

En este paso, limpiamos los datos, manejamos valores nulos y transformamos las variables categóricas en numéricas.

```
[4] 1 # IVAN FALCON MONZON
2 from sklearn.preprocessing import LabelEncoder
3
4 # Comprobar valores nulos
5 print("Valores nulos en train_data:\n", train_data.isnull().sum().sum())
6 print("Valores nulos en test_data:\n", test_data.isnull().sum().sum())
7
8 # Fusionamos los datos de entrenamiento y etiquetas
9 train = train_data.merge(train_labels, on="building_id")
10
11 # Convertir variables categóricas a numéricas con LabelEncoder
12 categorical_cols = train.select_dtypes(include=['object']).columns
13
14 label_encoders = {}
15 for col in categorical_cols:
16     le = LabelEncoder()
17     train[col] = le.fit_transform(train[col])
18     test_data[col] = le.transform(test_data[col])
19     label_encoders[col] = le # Guardamos los encoders por si los necesitamos después
20
21 # Separar características y etiquetas
22 X = train.drop(columns=["building_id", "damage_grade"])
23 y = train["damage_grade"]
24
25 # Guardamos el test sin la columna building_id
26 X_test = test_data.drop(columns=["building_id"])
27
28 # Mostrar datos preprocesados
29 print("Datos preprocesados:")
30 display(X.head())
```

→ Valores nulos en train_data:
0
Valores nulos en test_data:
0
Datos preprocesados:

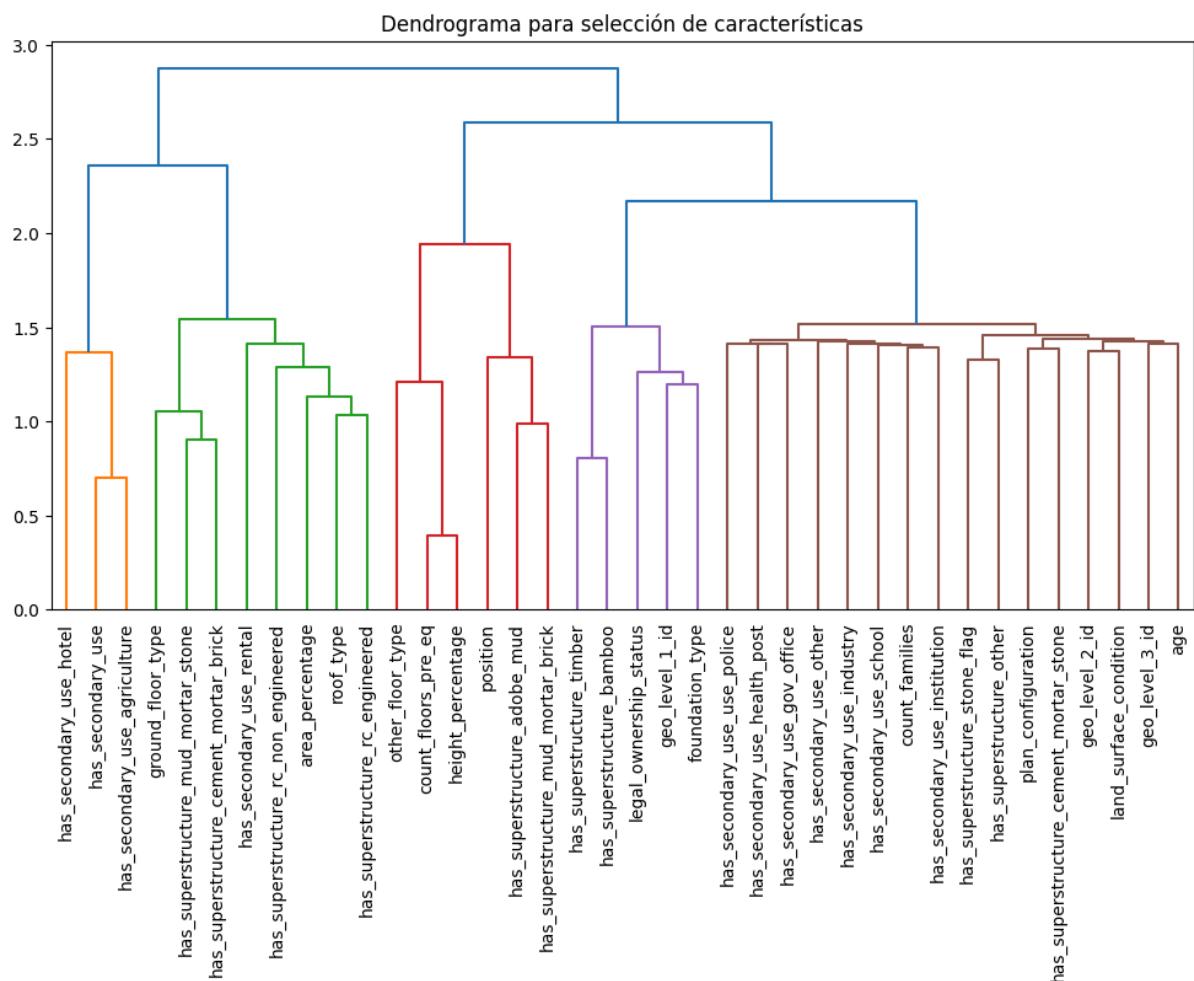
	geo_level_1_id	geo_level_2_id	geo_level_3_id	count_floors_pre_eq	age	area_percentage	height_percentage	land_surface_sqm
0	6	487	12198	2	30	6	5	
1	8	900	2812	2	10	8	7	
2	21	363	8973	2	10	5	5	
3	22	418	10694	2	10	6	5	
4	11	131	1488	3	30	8	9	

5 rows x 38 columns

Paso 3: Selección de características con dendrogramas

Este paso nos ayudará a reducir la dimensionalidad de los datos, eliminando características redundantes o poco informativas.

```
✓ [5] 1 # IVAN FALCON MONZON
2 import scipy.cluster.hierarchy as sch
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6 # Calcular la matriz de correlación
7 corr = X.corr().abs()
8
9 # Aplicar clustering jerárquico
10 linked = sch.linkage(corr, method='ward')
11
12 # Graficar el dendograma
13 plt.figure(figsize=(12, 6))
14 sch.dendrogram(linked, labels=X.columns, leaf_rotation=90, leaf_font_size=10)
15 plt.title("Dendrograma para selección de características")
16 plt.show()
```



Verificar las características de X:

```
[ ] 1 # IVAN FALCON MONZON
2 # Imprime las columnas de X para confirmar los nombres exactos de las características.
3 print("Características en X:")
4 print(X.columns)

→ Características en X:
Index(['geo_level_1_id', 'geo_level_2_id', 'geo_level_3_id',
       'count_floors_pre_eq', 'age', 'area_percentage', 'height_percentage',
       'land_surface_condition', 'foundation_type', 'roof_type',
       'ground_floor_type', 'other_floor_type', 'position',
       'plan_configuration', 'has_superstructure_adobe_mud',
       'has_superstructure_mud_mortar_stone', 'has_superstructure_stone_flag',
       'has_superstructure_cement_mortar_stone',
       'has_superstructure_mud_mortar_brick',
       'has_superstructure_cement_mortar_brick', 'has_superstructure_timber',
       'has_superstructure_bamboo', 'has_superstructure_rc_non_engineered',
       'has_superstructure_rc_engineered', 'has_superstructure_other',
       'legal_ownership_status', 'count_families', 'has_secondary_use',
       'has_secondary_use_agriculture', 'has_secondary_use_hotel',
       'has_secondary_use_rental', 'has_secondary_use_institution',
       'has_secondary_use_school', 'has_secondary_use_industry',
       'has_secondary_use_health_post', 'has_secondary_use_gov_office',
       'has_secondary_use_use_police', 'has_secondary_use_other'],
      dtype='object')
```

Seleccionar Características y Eliminar en base al dendograma

Se ha probado a poner y quitar características que aparecen en el dendograma varias veces, esta es la mejor combinación para que la predicción sea la más alta:

```
✓ 0 s 1 # IVAN FALCON MONZON
2 features_to_drop =
3   'has_superstructure_bamboo',
4   'has_secondary_use_other',
5   'legal_ownership_status',
6   'geo_level_2_id'
7 ]
8
9 # Definir X_reduced
10 X_reduced = X.drop(columns=features_to_drop)
11
12 # Verifica las columnas después de eliminar
13 print("Características en X_reduced:", X_reduced.columns)
14 # Verifica las columnas después de eliminar
15 print("Características en X_reduced:", X_reduced.columns)
16
17 # Verifica que X_reduced tiene las columnas correctas
18 print("Características en X_reduced:")
19 print(X_reduced.columns)
20
21 # Proceder con la división de datos
22 from sklearn.model_selection import train_test_split
23 X_train, X_val, y_train, y_val = train_test_split(X_reduced, y, test_size=0.2, random_state=42)

→ Características en X_reduced: Index(['geo_level_1_id', 'geo_level_3_id', 'count_floors_pre_eq', 'age',
       'area_percentage', 'height_percentage', 'land_surface_condition',
       'foundation_type', 'roof_type', 'ground_floor_type', 'other_floor_type',
       'position', 'plan_configuration', 'has_superstructure_adobe_mud',
       'has_superstructure_mud_mortar_stone', 'has_superstructure_stone_flag',
       'has_superstructure_cement_mortar_stone',
       'has_superstructure_mud_mortar_brick',
       'has_superstructure_cement_mortar_brick', 'has_superstructure_timber',
       'has_superstructure_rc_non_engineered',
       'has_superstructure_rc_engineered', 'has_superstructure_other',
       'count_families', 'has_secondary_use', 'has_secondary_use_agriculture',
```

Paso 4: Evaluar los modelos usando Lazy Predict

Lazy Predict permite evaluar rápidamente diferentes modelos.

```
1 # IVAN FALCON MONZON
2 !pip install lazypredict --quiet # Instalar Lazy Predict si no está instalado
3
4 # =====
5 # Paso 0: Importar Librerías necesarias
6 # =====
7 import pandas as pd
8 import numpy as np
9 from sklearn.model_selection import train_test_split
10 from lazypredict.Supervised import LazyClassifier
11 from sklearn.preprocessing import LabelEncoder
12 from sklearn.tree import DecisionTreeClassifier
13 from sklearn.ensemble import RandomForestClassifier
14
15 # =====
16 # Paso 1: Cargar datos
17 # =====
18 train_data = pd.read_csv('train_values.csv')
19 train_labels = pd.read_csv('train_labels.csv')
20
21 # Fusionar etiquetas con los datos de entrenamiento
22 train = train_data.merge(train_labels, on="building_id")
23
24 # =====
25 # Paso 2: Preprocesamiento
26 # =====
27 # Convertir variables categóricas en numéricas
28 categorical_cols = train.select_dtypes(include=['object']).columns
29 label_encoders = {}
30
31 for col in categorical_cols:
32     le = LabelEncoder()
33     train[col] = le.fit_transform(train[col])
34     label_encoders[col] = le # Guardamos los encoders
35
36 # Definir variables X (features) e y (target)
37 X = train.drop(columns=["building_id", "damage_grade"])
38 y = train["damage_grade"]
39
40 # =====
41 # Paso 3: Reducción de Memoria
42 # =====
43 def reduce_memory_usage(df):
44     for col in df.columns:
45         if df[col].dtype == "int64":
46             df[col] = df[col].astype("int16")
47         elif df[col].dtype == "float64":
48             df[col] = df[col].astype("float16")
49     return df
50
51 X_reduced = reduce_memory_usage(X)
52
53 # =====
54 # Paso 4: División de datos
55 # =====
56 # Usamos solo el 1% del dataset para evitar problemas de RAM
57 X_train_sub, X_test_sub, y_train_sub, y_test_sub = train_test_split(
58     X_reduced, y, test_size=0.99, random_state=42
59 )
60
61 # =====
62 # Paso 5: Lazy Predict - Solo Decision Tree y Random Forest
63 # =====
64 clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=None)
65
66 # Ejecutar Lazy Predict con todos los modelos disponibles
67 models, predictions = clf.fit(X_train_sub, X_test_sub, y_train_sub, y_test_sub)
68
69 # Mostrar los resultados si existen
70 if not models.empty:
71     print("\nResultados de Lazy Predict:")
72     display(models)
73 else:
74     print("Error: No se generaron modelos, reducir el dataset.")
```

Hubo varios problemas en la ejecución de lazy predict porque el google colab se caía por la memoria ram, así que se quitaron datos del dataset y reducir los datos de la memoria.

```
84% |██████████| 27/32 [02:33<01:20, 16.17s/it][LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001265 seconds.  
You can set 'force_row_wise=true' to remove the overhead.  
And if memory is not enough, you can set 'force_col_wise=true'.  
[LightGBM] [Info] Total Bins 713  
[LightGBM] [Info] Number of data points in the train set: 2606, number of used features: 30  
[LightGBM] [Info] Start training from score -2.348119  
[LightGBM] [Info] Start training from score -0.5/6644  
[LightGBM] [Info] Start training from score -1.070985  
100% |██████████| 32/32 [02:41<00:00, 5.04s/it]  
Resultados de Lazy Predict:
```

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
NearestCentroid	0.47	0.54	None	0.45	0.54
BaggingClassifier	0.62	0.54	None	0.62	1.27
LGBMClassifier	0.65	0.54	None	0.64	8.22
BernoulliNB	0.53	0.53	None	0.54	1.04
RandomForestClassifier	0.65	0.51	None	0.63	4.65
ExtraTreesClassifier	0.62	0.51	None	0.61	6.47
DecisionTreeClassifier	0.57	0.50	None	0.57	0.46
LinearDiscriminantAnalysis	0.57	0.49	None	0.54	0.56
LabelSpreading	0.52	0.48	None	0.52	27.18
LabelPropagation	0.52	0.48	None	0.52	27.46
AdaBoostClassifier	0.60	0.48	None	0.57	2.12
KNeighborsClassifier	0.56	0.47	None	0.55	19.83
ExtraTreeClassifier	0.52	0.45	None	0.52	0.47

Paso 5: Entrenar los modelos: Random Forest, Decision Tree, Gradient Boosting y SVM

5.1: Random Forest Classifier, optimizado con RandomizedSearchCV

- Algoritmo: Random Forest (Bosque Aleatorio)
- Optimización: RandomizedSearchCV (búsqueda aleatoria de hiperparámetros)
- Métrica usada: f1_micro (para clasificación multiclas)
- Mejor modelo: Se entrena de nuevo con los mejores hiperparámetros encontrados
- Predicción: Se generan predicciones en X_test y se guardan en un CSV

¿Qué tipo de modelo es?

- Clasificador basado en árboles de decisión (Random Forest)
- Modelo de ensamblado (ensemble) con múltiples árboles
- Optimizado con búsqueda aleatoria de hiperparámetros (RandomizedSearchCV)
- Métricas de evaluación: F1-score micro (para clasificación multiclas)

```

1 # IVAN FALCON MONZON
2 # Librerías necesarias:
3 from sklearn.model_selection import RandomizedSearchCV
4 from sklearn.ensemble import RandomForestClassifier
5 import pandas as pd
6 from sklearn.metrics import f1_score
7
8 # Definición del modelo:
9 rf_model = RandomForestClassifier(random_state=42)
10
11 # Definición del espacio de búsqueda para hiperparámetros:
12 param_grid_rf = {
13     'n_estimators': [50, 100],
14     'max_depth': [10, 20],
15     'min_samples_split': [5, 10]
16 }
17
18 # Optimización con RandomizedSearchCV:
19 random_search_rf = RandomizedSearchCV(
20     estimator=rf_model,
21     param_distributions=param_grid_rf,
22     scoring='f1_micro',
23     n_iter=5, # Número de combinaciones a probar
24     cv=3, # Usar 3 pliegues de validación cruzada
25     verbose=3,
26     n_jobs=-1, # Usa todos los núcleos disponibles
27     random_state=42
28 )
29
30 # Ajustar el modelo con RandomizedSearchCV
31 random_search_rf.fit(X_train, y_train)

33 # Imprimir los mejores hiperparámetros y la mejor puntuación F1
34 print("Mejores hiperparámetros para Random Forest:", random_search_rf.best_params_)
35 print("Mejor puntuación F1:", random_search_rf.best_score_)
36
37 # Entrenar con los mejores hiperparámetros encontrados:
38 best_rf_model = RandomForestClassifier(
39     n_estimators=random_search_rf.best_params_['n_estimators'],
40     max_depth=random_search_rf.best_params_['max_depth'],
41     min_samples_split=random_search_rf.best_params_['min_samples_split'],
42     random_state=42
43 )
44
45 # Entrenar el modelo con los datos de entrenamiento
46 best_rf_model.fit(X_train, y_train)
47
48 # Evaluar el modelo en el conjunto de validación
49 y_pred = best_rf_model.predict(X_val)
50 f1 = f1_score(y_val, y_pred, average="micro")
51 print("F1-score en validación:", f1)
52
53 # Alinear las columnas del conjunto de test con el conjunto de entrenamiento
54 X_test = X_test[X_train.columns]
55
56 # Realizar las predicciones en el conjunto de test
57 final_predictions = best_rf_model.predict(X_test)
58
59 # Crear el DataFrame de submission
60 submission = pd.DataFrame({
61     "building_id": test_data["building_id"], # Asegúrate de que 'test_data' tiene la columna 'building_id'
62     "damage_grade": final_predictions
63 })
64
65 # Guardar el archivo CSV de submission
66 submission.to_csv("submission.csv", index=False)
67 print("Archivo de submission generado con éxito.")

```

Última ejecución del modelo:

Fitting 3 folds for each of 5 candidates, totalling 15 fits

Mejores hiperparámetros para Random Forest: {'n_estimators': 100, 'min_samples_split': 10, 'max_depth': 20}

Mejor puntuación F1: 0.6895865312979094

F1-score en validación: 0.690719671533547

Archivo de submission generado con éxito.

Llegó a alcanzar y en segunda posición como mejor modelo un **0.7084**.

Primer archivo de submissions generado por el modelo de random forest

Richter's Predictor: Modeling Earthquake Damage

Can you predict the level of damage to buildings caused by the 2015 Gorkha earthquake in Nepal based on aspects of building location and construction?



Intermediate practice



7 months left



8,000 joined



Navigation

- [Home](#)
- [Problem description](#)
- [About](#)
- [Official rules](#)
- [Leaderboard](#)
- [Discussion \(3\)](#)
- [Data download](#)
- [Submissions \(1\)](#)
- [Share your work](#)
- [Team](#)

Submissions

- To help you track your progress during the competition, each submission is scored against publically available test data to give a "public score".
- The primary evaluation metric is Micro-averaged F1 score. [Show more.](#)

Best score

0.7084

Current rank

#1821

Submissions used

1 of 3

[Make new submission](#)

You have **2 of 3** submissions left today. Your next submission can be on March 1, 2025 UTC.

Your submissions

Public score	Who	Details
0.7084	IvanFalconMonzon	id-279865 - 0min ago

5.2: DecisionTreeClassifier con optimización RandomizedSearchCV

Este código implementa un modelo de clasificación basado en un árbol de decisión para predecir el grado de daño de los edificios.

Se realiza una optimización de hiperparámetros con RandomizedSearchCV, utilizando validación cruzada de 5 pliegues y búsqueda aleatoria en 10 combinaciones.

Luego, el mejor modelo encontrado se entrena con los datos de entrenamiento y se evalúa con la métrica F1-score en el conjunto de validación. Finalmente, se generan predicciones sobre el conjunto de pruebas y se guardan en un archivo CSV.

```
1 # IVAN FALCON MONZON
2 # Importación de librerías necesarias
3 from sklearn.model_selection import RandomizedSearchCV
4 from sklearn.tree import DecisionTreeClassifier
5 from sklearn.metrics import f1_score
6 import pandas as pd
7 from sklearn.preprocessing import LabelEncoder
8
9 # Cargar los datos de prueba
10 test_data = pd.read_csv('test_values.csv')
11
12 # Definir el modelo de árbol de decisión
13 dt_model = DecisionTreeClassifier(random_state=42)
14 |
15 # Espacio de búsqueda para la optimización de hiperparámetros
16 param_grid_dt = {
17     'max_depth': [10, 20, 30, 40, 50, None], # Profundidad máxima del árbol
18     'min_samples_split': [2, 5, 10, 20, 50], # Mínimo de muestras para dividir un nudo
19     'min_samples_leaf': [1, 2, 4, 6, 10], # Mínimo de muestras en las hojas
20     'criterion': ['gini', 'entropy'] # Criterios de división
21 }
22
23 # Optimización de hiperparámetros con búsqueda aleatoria y validación cruzada
24 random_search_dt = RandomizedSearchCV(
25     estimator=dt_model,
26     param_distributions=param_grid_dt,
27     scoring='f1_micro', # Métrica de evaluación F1-score micro
28     n_iter=10, # Número de combinaciones de hiperparámetros a evaluar
29     cv=5, # Validación cruzada con 5 pliegues
30     verbose=3,
31     n_jobs=-1, # Paralelización usando todos los núcleos disponibles
32     random_state=42
33 )
```

```
51 # Entrenar el modelo optimizado con los datos de entrenamiento
52 best_dt_model.fit(X_train, y_train)
53
54 # Evaluar el modelo en el conjunto de validación
55 y_pred = best_dt_model.predict(X_val)
56 f1 = f1_score(y_val, y_pred, average='micro')
57 print("F1-score en validación:", f1)
58
59 # Generar predicciones en el conjunto de prueba
60 final_predictions = best_dt_model.predict(X_test)
61
62 # Crear un DataFrame con los resultados de las predicciones
63 submission = pd.DataFrame({
64     "building_id": test_data["building_id"],
65     "damage_grade": final_predictions
66 })
67
68 # Guardar las predicciones en un archivo CSV para la competencia
69 submission.to_csv("submission.csv", index=False)
70 print("Archivo de submission generado con éxito.")
```

→ Fitting 5 folds for each of 10 candidates, totalling 50 fits
Mejores hiperparámetros para Decision Tree: {'min_samples_leaf': 10, 'max_depth': 50, 'criterion': 'gini', 'min_samples_split': 10}
Mejor puntuación F1: 0.6825402916346891
F1-score en validación: 0.6866330269948773
Archivo de submission generado con éxito.

5.3: Gradient Boosting optimizando sus hiperparámetros con RandomizedSearchCV

Este código implementa un modelo de clasificación basado en Gradient Boosting, optimizando sus hiperparámetros con RandomizedSearchCV.

Se reduce la carga computacional limitando el número de árboles y la profundidad, además de aplicar validación cruzada con 3 pliegues. Luego, el mejor modelo se entrena con los datos de entrenamiento y se evalúa con la métrica F1-score en el conjunto de validación. Se incluye early stopping para detener el entrenamiento si no hay mejora tras 5 iteraciones.

```
[]
1 # IVAN FALCON MONZON
2 # Librerías necesarias
3 from sklearn.ensemble import GradientBoostingClassifier
4 from sklearn.model_selection import RandomizedSearchCV
5 from sklearn.metrics import f1_score
6
7 # Definir el modelo base de Gradient Boosting
8 gb_model = GradientBoostingClassifier(random_state=42)
9
10 # Espacio de búsqueda de hiperparámetros ajustado para eficiencia
11 param_grid_gb = {
12     'n_estimators': [50, 100], # Número de árboles en el ensamble
13     'learning_rate': [0.05, 0.1], # Tasa de aprendizaje para control de ajuste
14     'max_depth': [3, 5], # Profundidad máxima del árbol
15     'min_samples_split': [2, 5], # Mínimo de muestras para dividir un nodo
16     'min_samples_leaf': [1, 2], # Mínimo de muestras por hoja
17     'subsample': [0.8, 1.0], # Fracción de datos utilizada en cada iteración
18     'max_features': ['sqrt'] # Selección aleatoria de características en cada división
19 }
20
21 # Optimización de hiperparámetros con búsqueda aleatoria y validación cruzada
22 random_search_gb = RandomizedSearchCV(
23     estimator=gb_model,
24     param_distributions=param_grid_gb,
25     scoring='f1_micro', # Métrica de evaluación F1-score micro
26     n_iter=6, # Número limitado de combinaciones para reducir el tiempo de cómputo
27     cv=3, # Validación cruzada con 3 pliegues para menor carga computacional
28     verbose=3,
29     n_jobs=-1, # Paralelización en todos los núcleos disponibles
30     random_state=42
31 )
32
33 # Ajustar el modelo con los datos de entrenamiento
34 random_search_gb.fit(X_train, y_train)

36 # Imprimir los mejores hiperparámetros y la mejor puntuación F1 obtenida
37 print("Mejores hiperparámetros para Gradient Boosting:", random_search_gb.best_params_)
38 print("Mejor puntuación F1 para Gradient Boosting:", random_search_gb.best_score_)
39
40 # Definir el modelo con los mejores hiperparámetros encontrados
41 best_gb_model = GradientBoostingClassifier(
42     **random_search_gb.best_params_,
43     random_state=42,
44     n_iter_no_change=5 # Early stopping: se detiene si no mejora en 5 iteraciones
45 )
46
47 # Entrenar el modelo optimizado con los datos de entrenamiento
48 best_gb_model.fit(X_train, y_train)
49
50 # Evaluar el modelo en el conjunto de validación
51 y_pred_gb = best_gb_model.predict(X_val)
52 f1_gb = f1_score(y_val, y_pred_gb, average="micro")
53 print("F1-score en validación (Gradient Boosting):", f1_gb)

    ▶ Fitting 3 folds for each of 6 candidates, totalling 18 fits
Mejores hiperparámetros para Gradient Boosting: {'subsample': 0.8, 'n_estimators': 100}
Mejor puntuación F1 para Gradient Boosting: 0.669920382919306
F1-score en validación (Gradient Boosting): 0.6677730665182939
```

5.4: modelo de clasificación utilizando XGBoost, optimizando sus hiperparámetros con RandomizedSearchCV

Este código implementa un modelo de clasificación utilizando XGBoost, optimizando sus hiperparámetros con RandomizedSearchCV.

Se realiza un ajuste en las etiquetas de la variable objetivo (`y_train_adj`, `y_val_adj`) para que sean compatibles con el modelo.

Tras la optimización, el mejor modelo se entrena con los datos de entrenamiento y se evalúa utilizando F1-score en el conjunto de validación. Finalmente, se generan predicciones para el conjunto de pruebas y se guardan en un archivo CSV.

```
[ ] 1 # IVAN FALCON MONZON
2 # Librerías necesarias
3 from xgboost import XGBClassifier
4 from sklearn.model_selection import RandomizedSearchCV
5 from sklearn.metrics import f1_score
6 import pandas as pd
7 from sklearn.preprocessing import LabelEncoder
8
9 # Cargar datos de prueba
10 test_data = pd.read_csv('test_values.csv') # Asegurar que el archivo de test sea el correcto
11
12 # Ajustar etiquetas de la variable objetivo si es necesario
13 y_train_adj = y_train - 1 # Ajuste de etiquetas para alinearlas con XGBoost
14 y_val_adj = y_val - 1
15
16 # Definir el modelo base de XGBoost
17 xgb_model = XGBClassifier(
18     objective='multi:softmax', # Clasificación multiclas con etiquetas discretas
19     eval_metric='mlogloss', # Función de evaluación: log-loss multiclas
20     use_label_encoder=False, # Desactivar el codificador de etiquetas de XGBoost
21     random_state=42
22 )
23
24 # Espacio de búsqueda de hiperparámetros
25 param_grid_xgb = {
26     'n_estimators': [100, 200], # Número de árboles en el ensamble
27     'learning_rate': [0.05, 0.1, 0.2], # Tasa de aprendizaje para el ajuste de pesos
28     'max_depth': [3, 5, 7], # Profundidad máxima de los árboles
29     'min_child_weight': [1, 3, 5], # Peso mínimo requerido en un nodo hoja
30     'gamma': [0, 0.1, 0.2], # Reducción mínima de pérdida para realizar un split
31     'subsample': [0.8, 1.0], # Proporción de datos utilizados por cada árbol
32     'colsample_bytree': [0.8, 1.0] # Proporción de características consideradas en cada split
33 }
34
35 # Optimización de hiperparámetros con RandomizedSearchCV
36 random_search_xgb = RandomizedSearchCV(
37     estimator=xgb_model,
38     param_distributions=param_grid_xgb,
39     scoring='f1_micro', # Evaluación con F1-score micro
40     n_iter=6, # Número de combinaciones de hiperparámetros a probar
41     cv=3, # Validación cruzada con 3 pliegues para reducir carga computacional
42     verbose=3,
43     n_jobs=-1, # Paralelización en todos los núcleos disponibles
44     random_state=42
45 )
46
47 # Ajustar el modelo con los datos de entrenamiento y etiquetas ajustadas
48 random_search_xgb.fit(X_train, y_train_adj)
49
50 # Imprimir los mejores hiperparámetros encontrados y la mejor puntuación F1 obtenida
51 print("Mejores hiperparámetros para XGBoost:", random_search_xgb.best_params_)
52 print("Mejor puntuación F1 para XGBoost:", random_search_xgb.best_score_)
53
54 # Definir el modelo final con los mejores hiperparámetros encontrados
55 best_xgb_model = XGBClassifier(
56     **random_search_xgb.best_params_,
57     objective='multi:softmax',
58     eval_metric='mlogloss',
59     use_label_encoder=False,
60     random_state=42
61 )
62
63 # Ajustar el modelo final con los datos de entrenamiento
64 best_xgb_model.fit(X_train, y_train_adj)
65
66 # Evaluar el modelo en el conjunto de validación
67 y_pred_xgb = best_xgb_model.predict(X_val)
68 f1_xgb = f1_score(y_val_adj, y_pred_xgb, average='micro')
69 print("F1-score en validación (XGBoost):", f1_xgb)
70
71 # Realizar predicciones sobre el conjunto de test
72 y_test_pred_xgb = best_xgb_model.predict(X_test)
73
```

Este ha sido el mejor modelo:

```
73
74 # Crear el DataFrame para la entrega de resultados
75 submission = pd.DataFrame([
76     "building_id": test_data["building_id"],
77     "damage_grade": y_test_pred_xgb + 1 # Ajustar etiquetas si es necesario
78 ])
79
80 # Guardar las predicciones en un archivo CSV
81 submission.to_csv("submission.csv", index=False)
82 print("Archivo de submission generado con éxito.")
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Mejores hiperparámetros para XGBoost: {'subsample': 0.8, 'n_estimators': 200, 'n'
Mejor puntuación F1 para XGBoost: 0.7352072179513239
F1-score en validación (XGBoost): 0.7373803265478406
Archivo de submission generado con éxito.
```

5.5: SVM con RandomizedSearchCV.

Este código entrena un modelo SVM (Support Vector Machine) con LinearSVC, optimizando la regularización con RandomizedSearchCV.

Se aplica escalado estándar (StandardScaler) para mejorar la estabilidad del modelo, y se usa una muestra reducida del conjunto de entrenamiento para acelerar el proceso.

Finalmente, se generan predicciones y se guardan en un archivo CSV.

Puntos clave

- **Muestreo:** Se seleccionan 5000 muestras aleatorias del conjunto de entrenamiento (`train_sample`) para optimizar el modelo más rápido.
- **Escalado:** Se normalizan los datos con `StandardScaler()`, lo cual es fundamental para que SVM funcione correctamente.
- **Optimización:** Se ajusta el hiperparámetro `C` mediante `RandomizedSearchCV` con solo 3 iteraciones y 2 folds para reducir la carga computacional.
- **Evaluación:** Se mide el F1-score (micro) en el conjunto de validación.
- **Predicciones finales:** Se generan predicciones sobre el conjunto de test y se guardan en `submission.csv`.

Código completo con los resultados:

```
1 # IVAN FALCON MONZON
2 # Librerías necesarias
3 from sklearn.svm import LinearSVC
4 from sklearn.model_selection import RandomizedSearchCV
5 from sklearn.preprocessing import StandardScaler
6 from sklearn.metrics import f1_score
7 import pandas as pd
8 import numpy as np
9
10 # Definir tamaño de muestra para entrenar más rápido
11 TEST_SAMPLE_SIZE = 5000
12
13 # Seleccionar una muestra aleatoria del conjunto de entrenamiento
14 train_sample = X_train.sample(TEST_SAMPLE_SIZE, random_state=42)
15 train_labels = y_train.loc[train_sample.index]
16
17 # Escalar los datos con StandardScaler
18 scaler = StandardScaler()
19 X_train_scaled = scaler.fit_transform(train_sample) # Ajustar y transformar datos de entrenamiento
20 X_val_scaled = scaler.transform(X_val) # Transformar datos de validación
21 X_test_scaled = scaler.transform(X_test) # Transformar datos de prueba
22
23 # Definir el modelo SVM con LinearSVC
24 svm_model = LinearSVC(random_state=42, dual=False, max_iter=5000)
25
26 # Espacio de búsqueda de hiperparámetros para SVM
27 param_grid_svm = {
28     'C': [0.1, 1, 10] # Parámetro de regularización
29 }
30
31 # Optimización con RandomizedSearchCV
32 random_search_svm = RandomizedSearchCV(
33     estimator=svm_model,
34     param_distributions=param_grid_svm,
35     scoring='f1_micro', # Optimización basada en F1-score micro
36     n_iter=3, # Reducimos las iteraciones para mayor velocidad
37     cv=2, # Usamos solo 2 folds en validación cruzada para reducir tiempo
38     verbose=1,
39     n_jobs=-1, # Utilizar todos los núcleos disponibles
40     random_state=42
41 )
42
43 # Entrenar el modelo con la muestra de datos escalados
44 random_search_svm.fit(X_train_scaled, train_labels)
45
46 # Mostrar los mejores hiperparámetros encontrados
47 print("Mejores hiperparámetros para SVM:", random_search_svm.best_params_)
48 print("Mejor puntuación F1 para SVM:", random_search_svm.best_score_)
49
50 # Modelo final con los mejores hiperparámetros encontrados
51 best_svm_model = LinearSVC(
52     C=random_search_svm.best_params_['C'],
53     random_state=42,
54     dual=False,
55     max_iter=5000
56 )
57
58 # Entrenar el modelo final con los datos escalados
59 best_svm_model.fit(X_train_scaled, train_labels)
60
61 # Evaluar en el conjunto de validación
62 y_pred_svm = best_svm_model.predict(X_val_scaled)
63 f1_svm = f1_score(y_val, y_pred_svm, average="micro")
64 print("F1-score en validación (SVM):", f1_svm)
65
66 # Realizar predicciones sobre el conjunto de test
67 final_predictions = best_svm_model.predict(X_test_scaled)
68
69 # Crear archivo de submission
70 submission = pd.DataFrame({
71     "building_id": test_data["building_id"],
72     "damage_grade": final_predictions
73 })
74 submission.to_csv("submission.csv", index=False)
75 print("Archivo de submission generado con éxito.")

Fitting 2 folds for each of 3 candidates, totalling 6 fits
Mejores hiperparámetros para SVM: {'C': 10}
Mejor puntuación F1 para SVM: 0.5828
F1-score en validación (SVM): 0.5775599086740469
```

Comparaciones de las predicciones de los modelos utilizados:

Modelo	Mejor puntuación F1 (Optimización)	F1-score en Validación
XG Boost	0.7352	0.7374
Random Forest	0.6896	0.6907
Decision Tree	0.6825	0.6866
Gradient Boosting	0.6699	0.6678
SVM	0.5828	0.5776

Conclusiones sobre los dos mejores modelos

1. **XGBoost** es claramente el mejor modelo con la mayor puntuación F1 tanto en optimización (0.7352) como en validación (0.7374). Esto sugiere que es el más efectivo para capturar patrones en los datos y generalizar en el conjunto de validación.
2. **Random Forest** ocupa el segundo lugar con una F1 de 0.6896 en optimización y 0.6907 en validación. Aunque es inferior a XGBoost, sigue siendo un modelo sólido, especialmente si se prioriza interpretabilidad y estabilidad.

Comparación clave:

- XGBoost supera a Random Forest en más de **4.7 puntos porcentuales** en F1-score de validación, lo que indica que maneja mejor las relaciones complejas en los datos.
- Random Forest puede ser preferido si se busca menor tiempo de entrenamiento y mayor facilidad de ajuste.

Últimos resultados de la competición

Submissions

- To help you track your progress during the competition, each submission is scored against publically available test data to give a "public score".
- The primary evaluation metric is Micro-averaged F1 score. [Show more.](#)

Best score 0.7371	Current rank #1120	Submissions used 3 of 3
No submissions remaining		

You have **0 of 3** submissions left today. Your next submission can be on March 11, 2025 UTC.

Your submissions

Public score	Who	Details
0.7084	 IvanFalconMonzon	id-279865 · 1w 1d ago
0.6952	 IvanFalconMonzon	id-279866 · 1w 1d ago
0.6907	 IvanFalconMonzon	id-280053 · 2d ago
0.7035	 IvanFalconMonzon	id-280063 · 1d 21h ago
Error	 IvanFalconMonzon	id-280064 · 1d 18h ago CSV Headers do not match. Submission requires that first line is: "building_id,damage_grade" You submitted: "Id,Predicted"
0.7371	 IvanFalconMonzon	id-280066 · 1d 16h ago
0.6907	 IvanFalconMonzon	id-280107 · 6min ago
0.6835	 IvanFalconMonzon	id-280108 · 4min ago
0.7154	 IvanFalconMonzon	id-280109 · 0min ago

Repositorios y Enlaces

https://github.com/IvanFalconMonzon/SNS_ACT3_7_IvanFalconMonzon.git

<https://colab.research.google.com/drive/1uHCj3qkmKGMteGXN7MqLVAgJ5B1EV2Wv?usp=sharing>