

Predicción de Riesgo de derrumbamiento Terremotos



Néstor Batista Díaz

ÍNDICE

INTRODUCCIÓN	2
PROCESAMIENTO DE DATOS	2
VISUALIZACIÓN DE DATOS	3
PROCESAMIENTO DE DATOS	6
NORMALIZACIÓN	6
DIVIDIR LOS DATOS	6
SELECCIÓN DE CARACTERÍSTICAS	7
MATRIZ DE CORRELACIÓN	7
DENDOGRAMA	8
SELECCIONAR ESTRATEGIA	10
SIN ESTRATEGIA	10
ESTRATEGIA: Penalización	12
ESTRATEGIA: Subsampling de las clases mayoritarias	14
ESTRATEGIA: Oversampling de la clase minoritaria	16
ESTRATEGIA: Oversampling de la clase minoritaria	18
ESTRATEGIA: Ensamble de Modelos con Balanceo	20
CONCLUSIÓN	21
ENTRENAMIENTO	22
Logistic Regression	22
LAZY PREDICT	25
ExtraTreeClasifier	26
CONCLUSIÓN	28
PREDICCIÓN CON LOS DATOS DE LA COMPETICIÓN	29
POSICIÓN EN LA COMPETICIÓN	30
REFERENCIAS	30

INTRODUCCIÓN

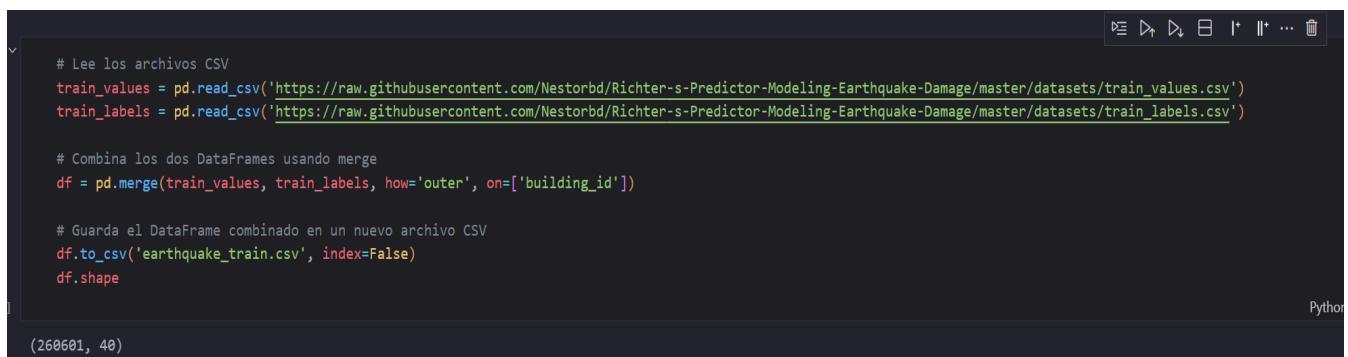
El terremoto de Gorkha de 2015 en Nepal dejó una profunda huella en la infraestructura del país, causando daños significativos a miles de edificios y afectando a comunidades enteras. En este contexto, la competición de Driven Data ofrece una oportunidad única para aplicar técnicas de aprendizaje automático y análisis de datos con el objetivo de predecir el nivel de daño a los edificios afectados por este desastre natural.

El conjunto de datos recopilado a través de encuestas exhaustivas proporciona una rica fuente de información sobre la ubicación, la construcción y otras variables relevantes que influyen en la vulnerabilidad de los edificios ante eventos sísmicos. Con la colaboración de Kathmandu Living Labs y la Oficina Central de Estadísticas de Nepal, se dispone de una base de datos sólida que permite explorar patrones, identificar factores críticos y desarrollar modelos predictivos precisos para evaluar el riesgo de daño estructural.

En este trabajo, se abordará el desafío de predecir el nivel de daño a los edificios mediante técnicas avanzadas de aprendizaje automático, con el objetivo de contribuir a la planificación de medidas de mitigación y respuesta en caso de futuros eventos sísmicos. La combinación de la expertise en ciencia de datos y la relevancia social de este problema hacen de esta competición una oportunidad invaluable para generar conocimiento y aplicarlo en la protección de comunidades vulnerables.

PROCESAMIENTO DE DATOS

Importamos los datasets y los juntamos.



```
# Lee los archivos CSV
train_values = pd.read_csv('https://raw.githubusercontent.com/Nestorbd/Richter-s-Predictor-Modeling-Earthquake-Damage/master/datasets/train_values.csv')
train_labels = pd.read_csv('https://raw.githubusercontent.com/Nestorbd/Richter-s-Predictor-Modeling-Earthquake-Damage/master/datasets/train_labels.csv')

# Combina los dos DataFrames usando merge
df = pd.merge(train_values, train_labels, how='outer', on=['building_id'])

# Guarda el DataFrame combinado en un nuevo archivo CSV
df.to_csv('earthquake_train.csv', index=False)
df.shape
```

(260601, 40)

Transformamos los parámetros de texto en números.

```
1 df['land_surface_condition'].replace(['n', 'o', 't'],[0, 1, 2], inplace=True)
2 df['foundation_type'].replace(['h', 'i', 'r', 'u', 'w'],[0, 1, 2, 3, 4], inplace=True)
3 df['roof_type'].replace(['n', 'q', 'x'],[0, 1, 2], inplace=True)
4 df['ground_floor_type'].replace(['f', 'm', 'v', 'x', 'z'],[0, 1, 2, 3, 4], inplace=True)
5 df['other_floor_type'].replace(['j', 'q', 's', 'x'],[0, 1, 2, 3], inplace=True)
6 df['position'].replace(['j', 'o', 's', 't'],[0, 1, 2, 3], inplace=True)
7 df['plan_configuration'].replace(['a', 'c', 'd', 'f', 'm','n', 'o', 'q', 's','u'],[0,
8 df['legal_ownership_status'].replace(['a', 'r', 'v', 'w'],[0, 1, 2, 3], inplace=True)
```

✓ 0.5s

Python

VISUALIZACIÓN DE DATOS

```
1 df.describe()
```

✓ 0.2s

Python

	building_id	geo_level_1_id	geo_level_2_id	geo_level_3_id	count_floors_pre_eq	age	year_renovated
count	260601.00	260601.00	260601.00	260601.00	260601.00	260601.00	260601.00
mean	525675.48	13.90	701.07	6257.88	2.13	26.54	2000.00
std	304545.00	8.03	412.71	3646.37	0.73	73.57	1900.00
min	4.00	0.00	0.00	0.00	1.00	0.00	1900.00
25%	261190.00	7.00	350.00	3073.00	2.00	10.00	2000.00
50%	525757.00	12.00	702.00	6270.00	2.00	15.00	2000.00
75%	789762.00	21.00	1050.00	9412.00	2.00	30.00	2000.00
max	1052934.00	30.00	1427.00	12567.00	9.00	995.00	2000.00

8 rows × 40 columns

```
1 df.isnull().sum()
```

✓ 0.0s

```
building_id          0  
geo_level_1_id      0  
geo_level_2_id      0  
geo_level_3_id      0  
count_floors_pre_eq 0  
age                  0  
area_percentage      0  
height_percentage    0  
land_surface_condition 0  
foundation_type      0  
roof_type            0  
ground_floor_type    0  
other_floor_type     0  
position              0  
plan_configuration    0  
has_superstructure_adobe_mud 0  
has_superstructure_mud_mortar_stone 0  
has_superstructure_stone_flag 0  
has_superstructure_cement_mortar_stone 0  
has_superstructure_mud_mortar_brick 0  
has_superstructure_cement_mortar_brick 0  
has_superstructure_timber    0  
has_superstructure_bamboo   0  
has_superstructure_rc_non_engineered 0  
has_superstructure_rc_engineered 0  
...  
has_secondary_use_gov_office 0  
has_secondary_use_use_police 0  
has_secondary_use_other     0  
damage_grade             0  
dtype: int64
```

```
1 df.info()
```

✓ 0.0s

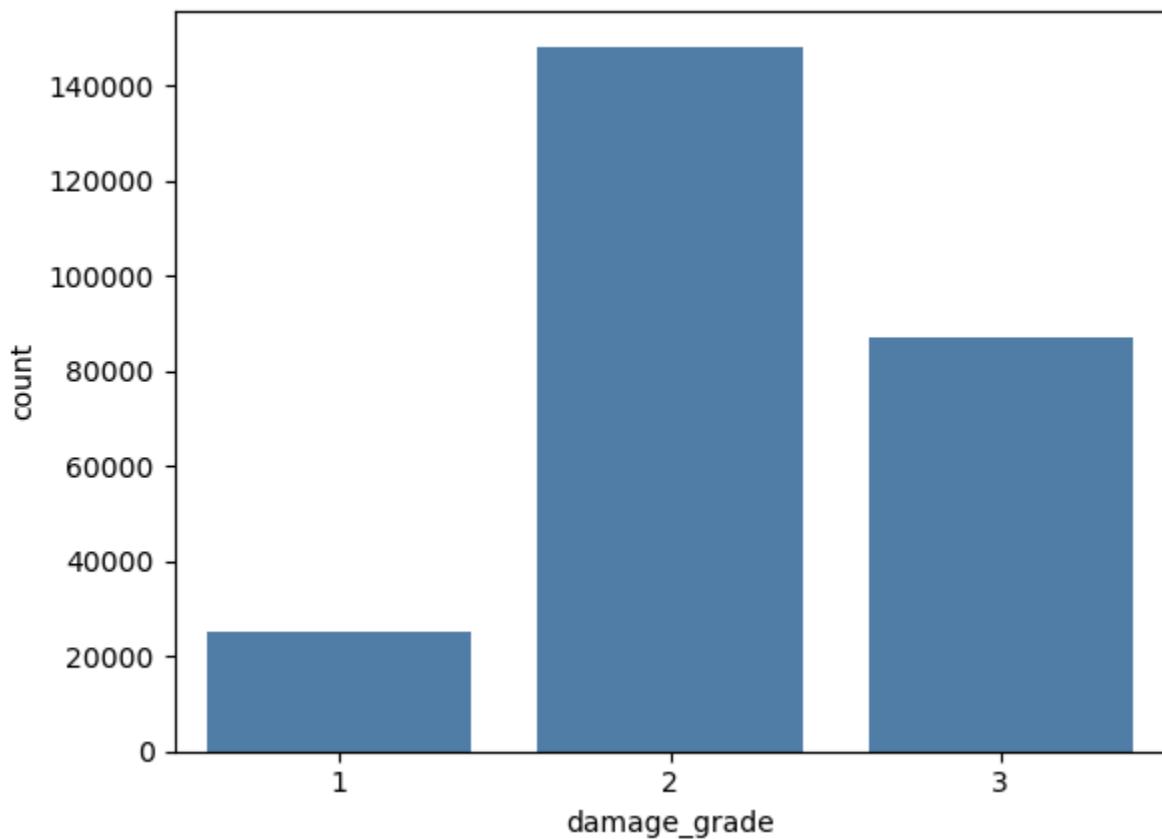
```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 260601 entries, 0 to 260600
```

```
Data columns (total 40 columns):
```

#	Column	Non-Null Count	Dtype
0	building_id	260601	non-null int64
1	geo_level_1_id	260601	non-null int64
2	geo_level_2_id	260601	non-null int64
3	geo_level_3_id	260601	non-null int64
4	count_floors_pre_eq	260601	non-null int64
5	age	260601	non-null int64
6	area_percentage	260601	non-null int64
7	height_percentage	260601	non-null int64
8	land_surface_condition	260601	non-null int64
9	foundation_type	260601	non-null int64
10	roof_type	260601	non-null int64
11	ground_floor_type	260601	non-null int64
12	other_floor_type	260601	non-null int64
13	position	260601	non-null int64
14	plan_configuration	260601	non-null int64
15	has_superstructure_adobe_mud	260601	non-null int64
16	has_superstructure_mud_mortar_stone	260601	non-null int64
17	has_superstructure_stone_flag	260601	non-null int64
18	has_superstructure_cement_mortar_stone	260601	non-null int64
19	has_superstructure_mud_mortar_brick	260601	non-null int64
...			

Aquí podemos comprobar que no hay datos nulos en el datasets.



Con este gráfico podemos observar que los datos están desbalanceados. Tenemos muy pocos datos de la clase 1 y muchos de la clase 2.

Para abordar este problema voy a aplicar diferentes estrategias y usaré la que de mejor resultado para entrenar el modelo que será enviado a la valoración de la competición.

PROCESAMIENTO DE DATOS

NORMALIZACIÓN

```
1 column_names = df.columns.values
2 index_to_remove = np.where(column_names == 'damage_grade')[0]
3 column_names = np.delete(column_names, index_to_remove)
4 (variable) scaled_data: ndarray
5
6 scaled_data = scaler.fit_transform(df.drop('damage_grade', axis=1))
● 7 scaled_df = pd.DataFrame(scaled_data, columns=column_names)
8
9 df_scaled = scaled_df.copy()
10 df_scaled["damage_grade"] = df["damage_grade"]
✓ 0.4s
```

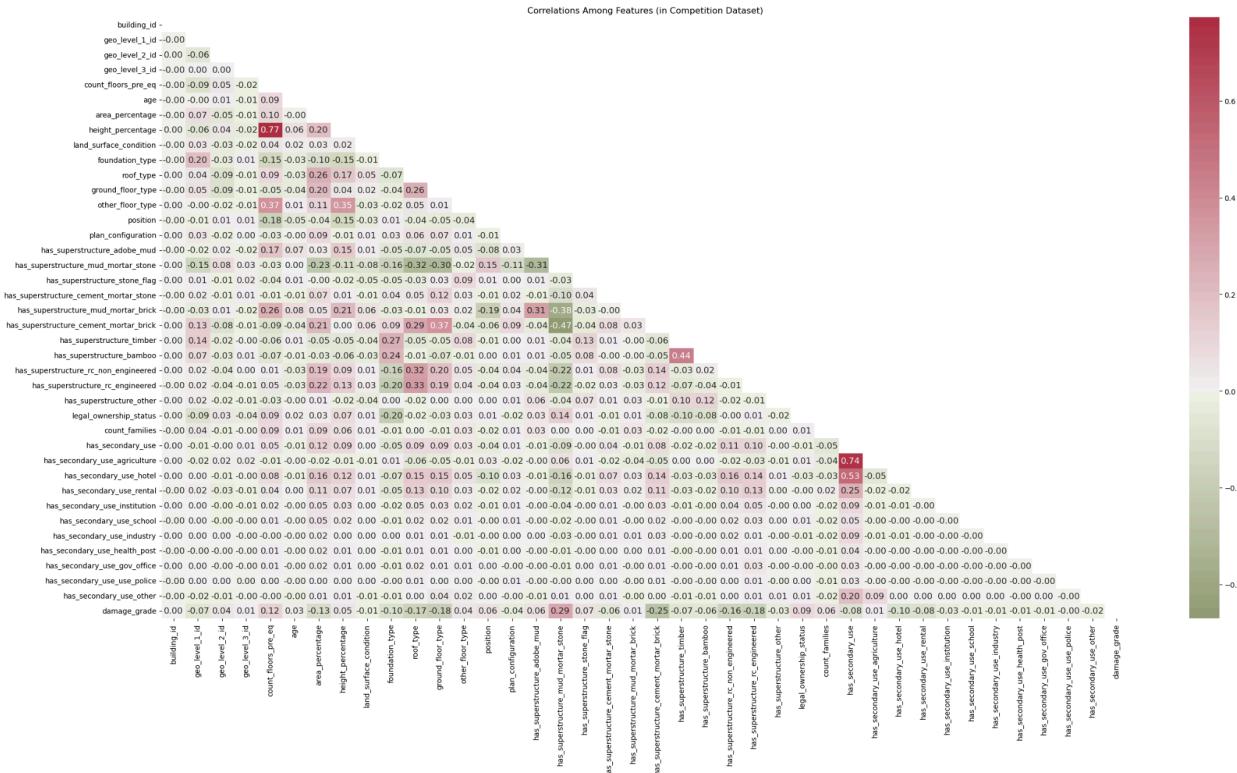
DIVIDIR LOS DATOS

```
1 X = df_scaled.drop('damage_grade', axis=1)
2 y = df_scaled['damage_grade']
3
4 # Dividir los datos en conjuntos de entrenamiento y prueba
5 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
6 X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, stratify=y_train, random_state=42)
✓ 0.2s
```

Python

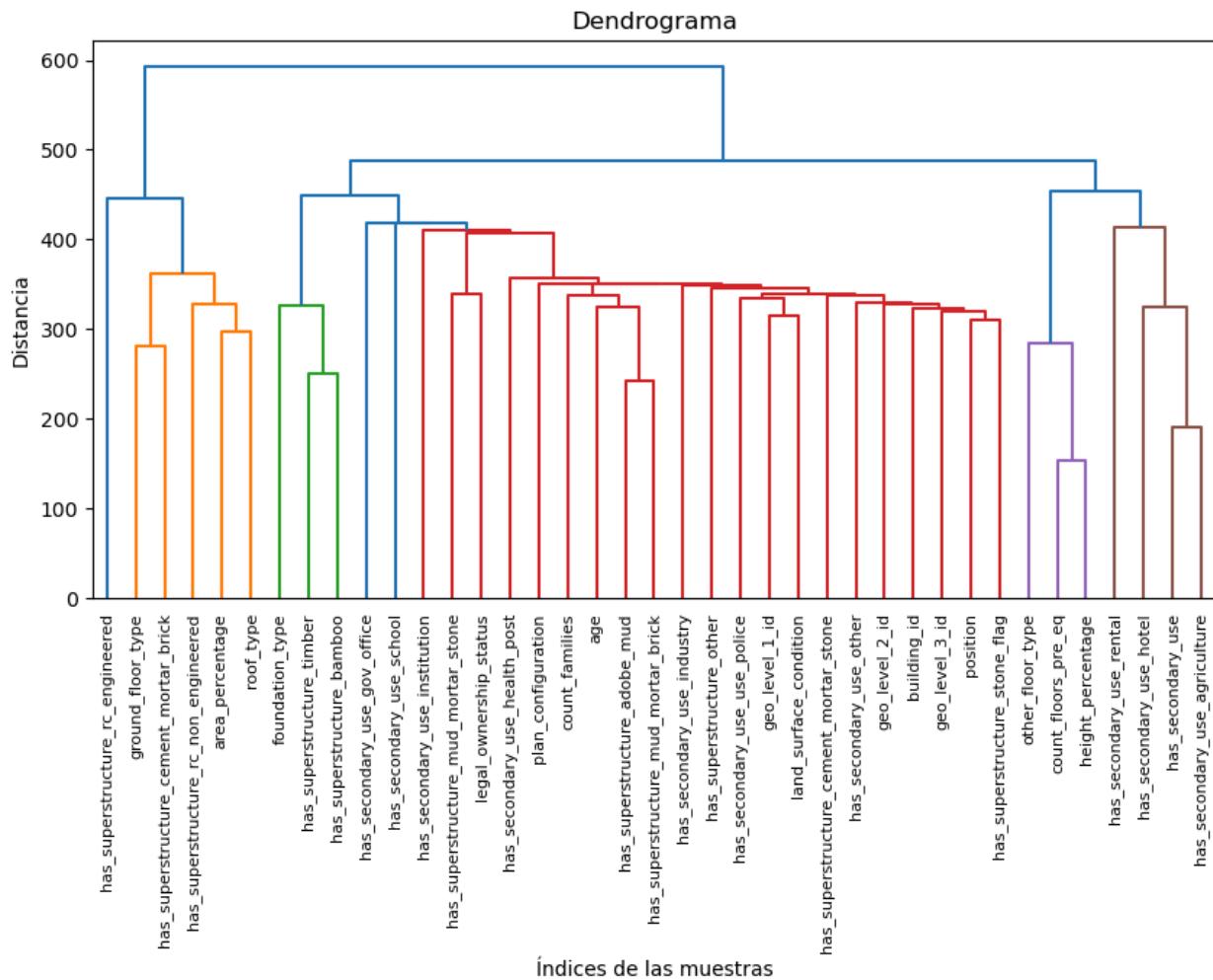
SELECCIÓN DE CARACTERÍSTICAS

MATRIZ DE CORRELACIÓN



Es tan grande que casi no se puede sacar nada en claro, así que la acompañe de otros métodos de selección de características.

DENDROGRAMA



Se divide el dataset en 8 Clusters, para abordar este problema use una característica por cluster a excepción del cluster más grande marcado en rojo del que use 3.

```

1 from sklearn.feature_selection import SelectKBest, f_classif
2
3 # Crear un objeto SelectKBest para cada cluster
4 select_k_best_list = []
5 selected_features = []
6
7 for cluster in cluster_features:
8     # Obtener indices de las características del cluster en el DataFrame original
9     feature_indices = [muestra_balanceada.columns.get_loc(feature) for feature in cluster]
10
11     K = 1
12     if(len(feature_indices) > 5):
13         K=3
14
15     # Seleccionar las k mejores características para el cluster actual
16     select_k_best = SelectKBest(score_func=f_classif, k=K) # Seleccionar 2 características por cluster
17     select_k_best.fit(muestra_balanceada.iloc[:, feature_indices], y_muestra) # Ajustar SelectKBest con las características
18
19     # Guardar el objeto SelectKBest en la lista
20     select_k_best_list.append(select_k_best)
21
22     # Obtener la máscara de booleanos de las características seleccionadas
23     mask = select_k_best.get_support()
24
25     # Obtener los nombres de las características seleccionadas
26     features = [cluster[i] for i, selected in enumerate(mask) if selected]
27
28     selected_features.append(features)
29
30 # Imprimir las características seleccionadas por cada cluster
31 for i, features in enumerate(selected_features):
32     print(f"Cluster {i+1} - Características seleccionadas: {features}")

```

Para seleccionar las características use selectKBest por cada cluster y este fue el resultado:

```

Cluster 1 - Características seleccionadas: ['has_superstructure_cement_mortar_brick']
Cluster 2 - Características seleccionadas: ['has_superstructure_rc_engineered']
Cluster 3 - Características seleccionadas: ['foundation_type']
Cluster 4 - Características seleccionadas: ['geo_level_1_id', 'has_superstructure_adobe_mud', 'has_superstructure_mud_mortar_stone']
Cluster 5 - Características seleccionadas: ['has_secondary_use_school']
Cluster 6 - Características seleccionadas: ['has_secondary_use_gov_office']
Cluster 7 - Características seleccionadas: ['count_floors_pre_eq']
Cluster 8 - Características seleccionadas: ['has_secondary_use_hotel']

```

```

1 # Unir todas las características seleccionadas en un solo array
2 all_selected_features = np.concatenate(selected_features)
3
4 # Imprimir el array con todas las características seleccionadas
5 print("Todas las características seleccionadas:")
6 print(all_selected_features)

```

✓ 0.0s Open 'all_selected_features'

```

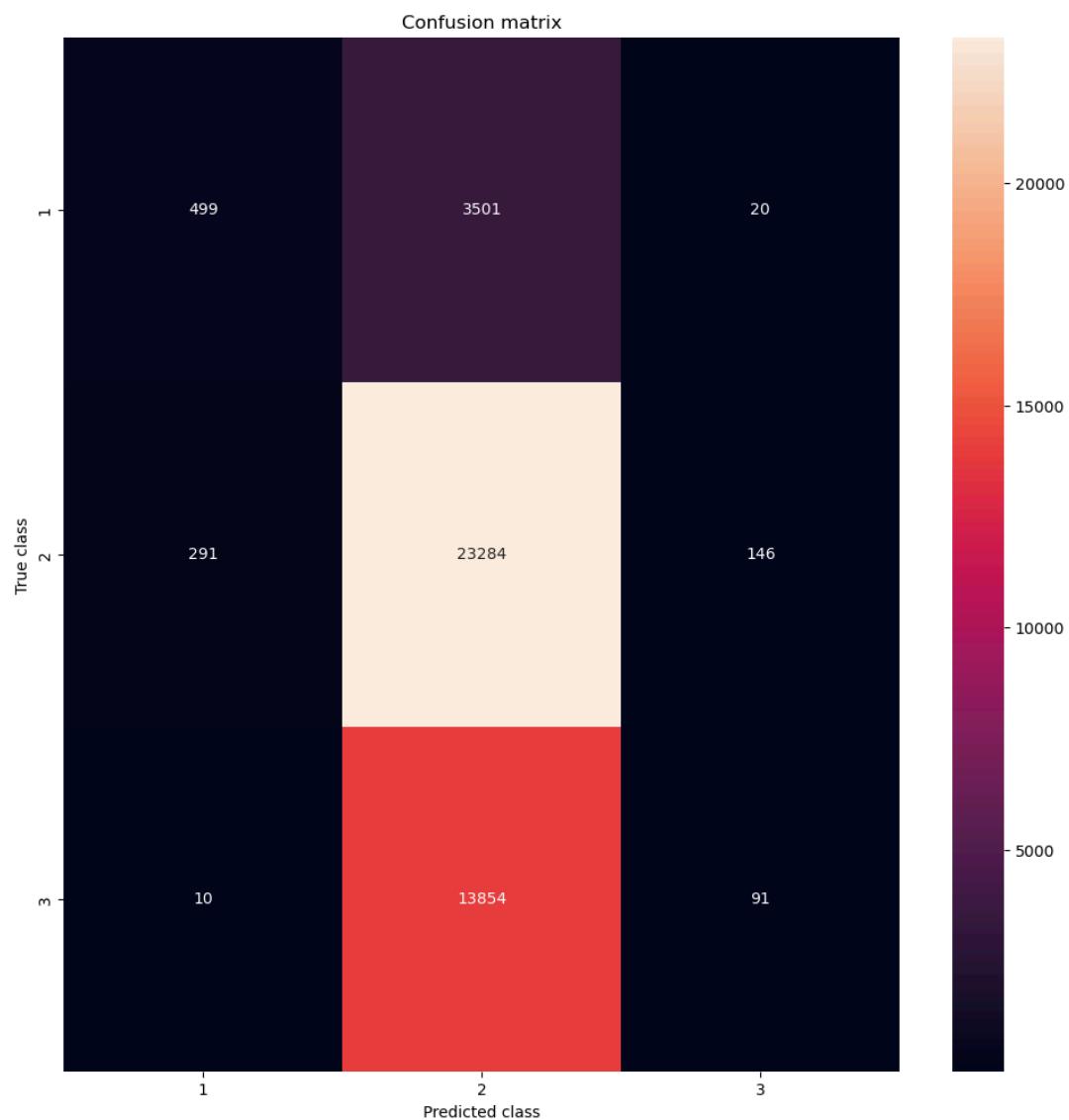
Todas las características seleccionadas:
['has_superstructure_cement_mortar_brick',
 'has_superstructure_rc_engineered', 'foundation_type', 'geo_level_1_id',
 'has_superstructure_adobe_mud', 'has_superstructure_mud_mortar_stone',
 'has_secondary_use_school', 'has_secondary_use_gov_office',
 'count_floors_pre_eq', 'has_secondary_use_hotel']

```

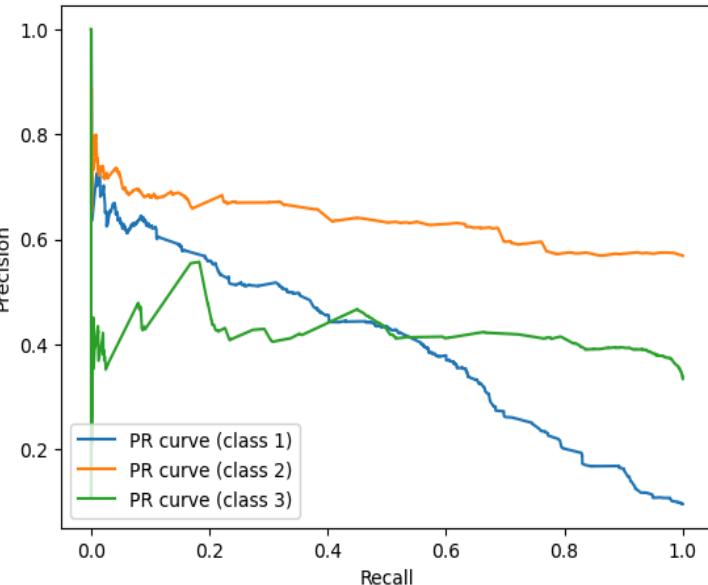
SELECCIONAR ESTRATEGIA

SIN ESTRATEGIA

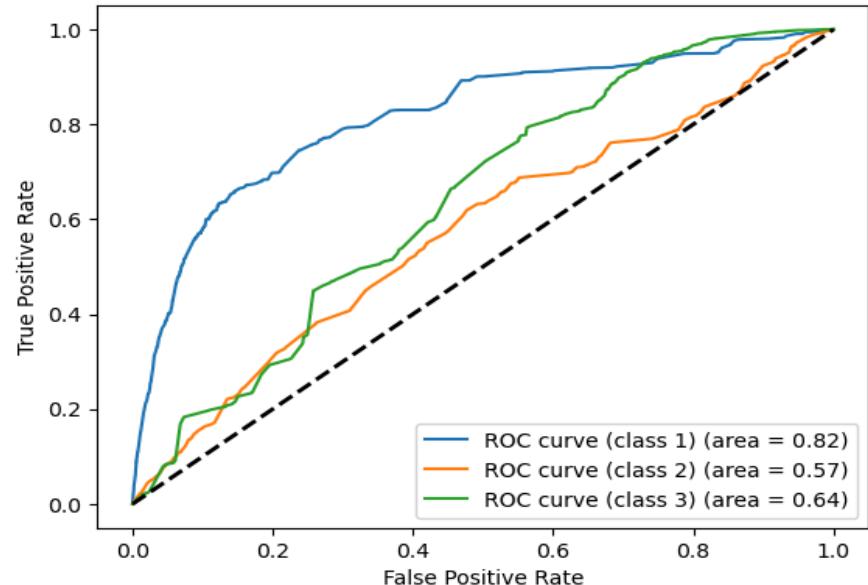
```
1 model = LogisticRegression(C=1.0,penalty='l2',random_state=1,solver="liblinear")
2 model.fit(X_train,y_train)
3 pred_y = model.predict(X_val)
4 mostrar_resultados(model, X_val, y_val, pred_y)
✓ 1.8s
```



Precision-Recall Curve for Multiclass



Receiver Operating Characteristic (ROC) Curve for Multiclass



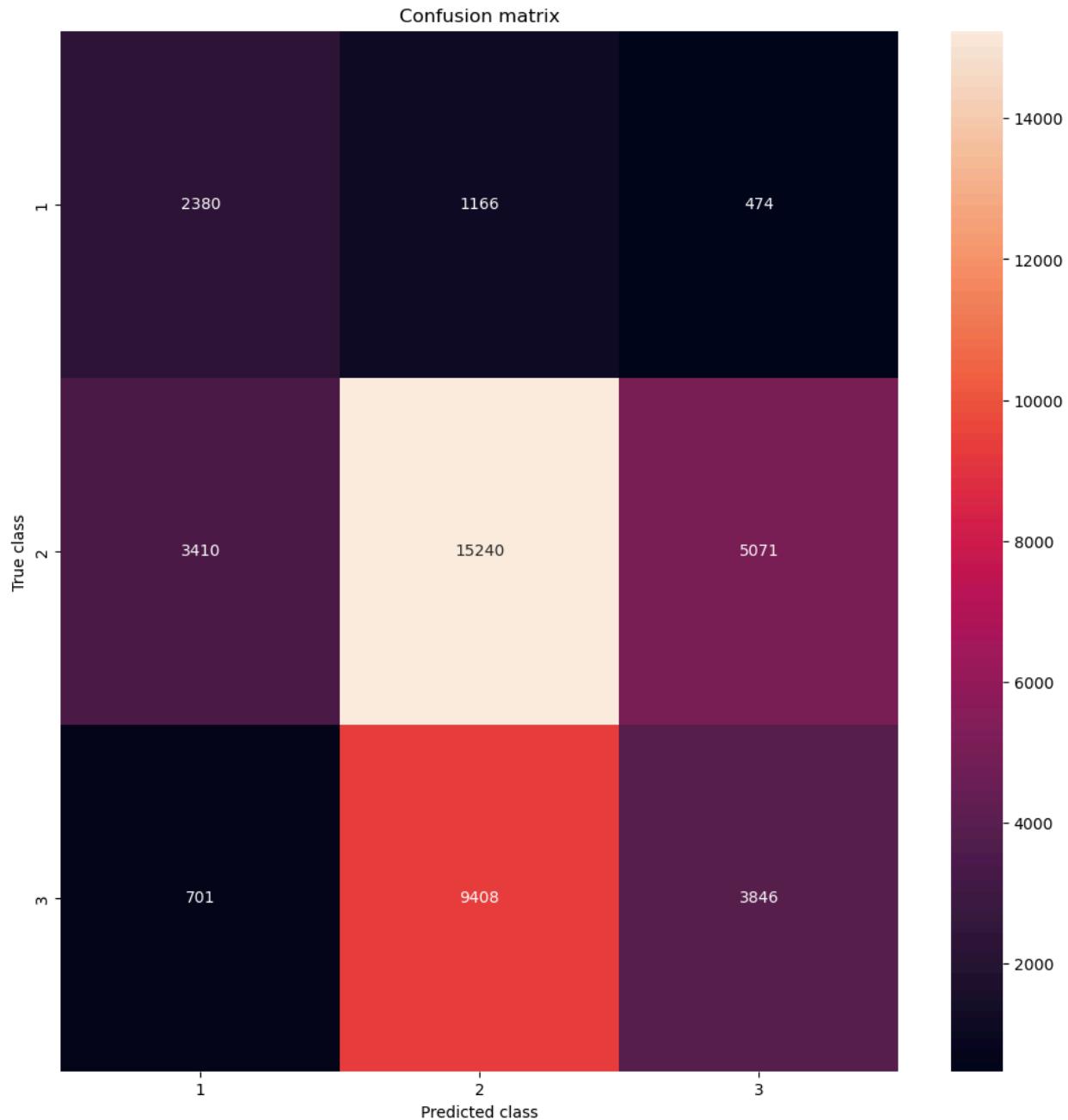
	precision	recall	f1-score	support
1	0.62	0.12	0.21	4020
2	0.57	0.98	0.72	23721
3	0.35	0.01	0.01	13955
accuracy			0.57	41696
macro avg	0.52	0.37	0.31	41696
weighted avg	0.50	0.57	0.44	41696

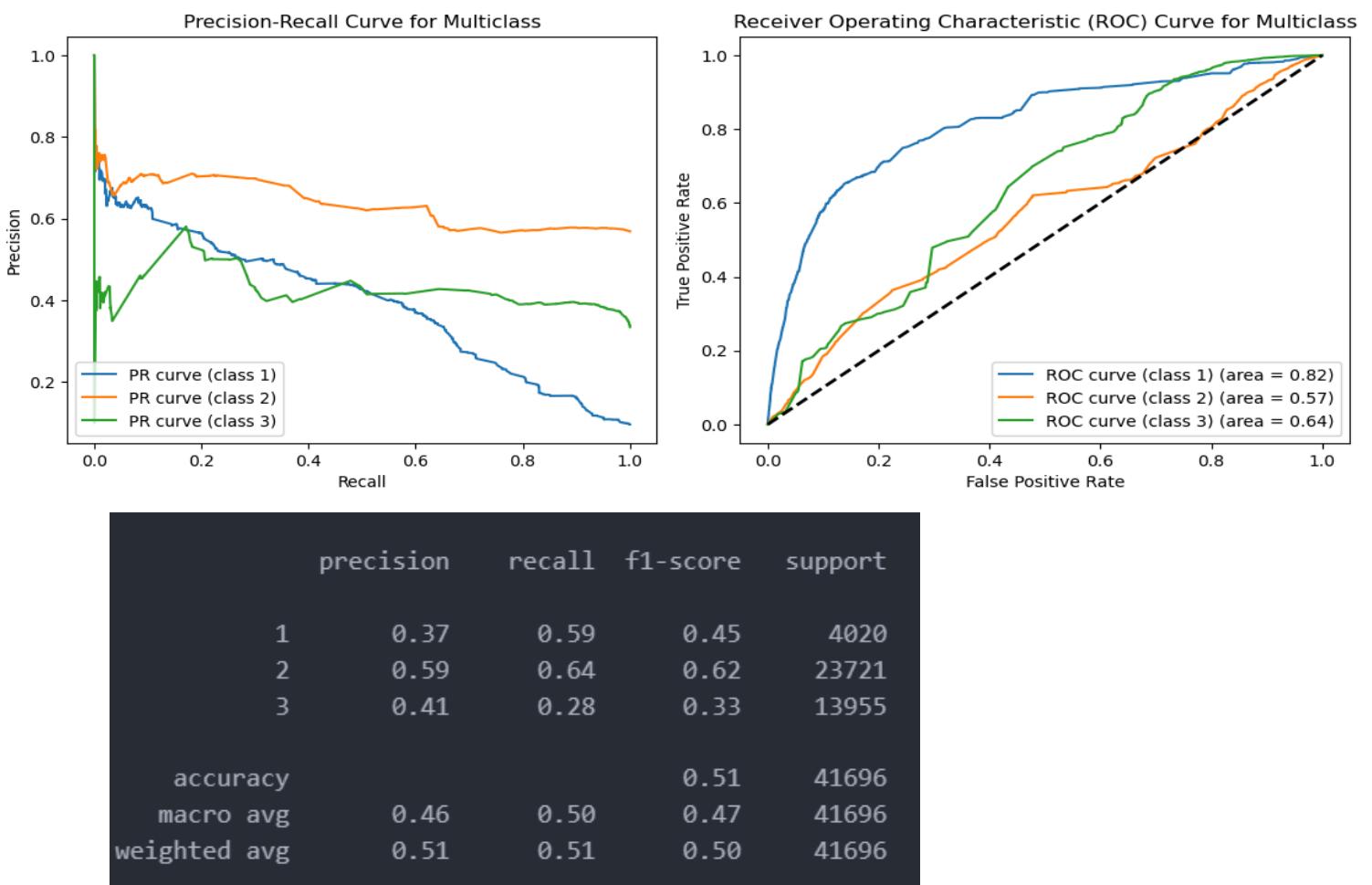
Como podemos comprobar el accuracy está bastante bien, 0.57, pero es porque la mayoría de los datos predichos correctamente pertenecen a la clase 2 que es la que tiene más representación en el dataset. Si nos fijamos en la matriz de confusión, nos podemos fijar en que las clases 1 y 3 se confunden con la clase 2. Por lo que este método no nos sirve para lo que queremos.

ESTRATEGIA: Penalización

```
1 model = LogisticRegression(C=1.0,penalty='l2',random_state=1,solver="liblinear",class_weight="balanced")
2 model.fit(X_train, y_train)
3 pred_y = model.predict(X_val)
4 mostrar_resultados(model, X_val, y_val, pred_y)
```

1.7s





En este caso vemos una mejoría, en la matriz de confusión vemos como está mejor distribuido que el anterior. El problema en este caso es solo la clase 3, que la confunde con la 2.

También nos podemos fijar en la gráfica de la curva de ROC en donde la clase dos está muy cerca de la diagonal ya que las clases 1 y 3 se confunden mucho con ella.

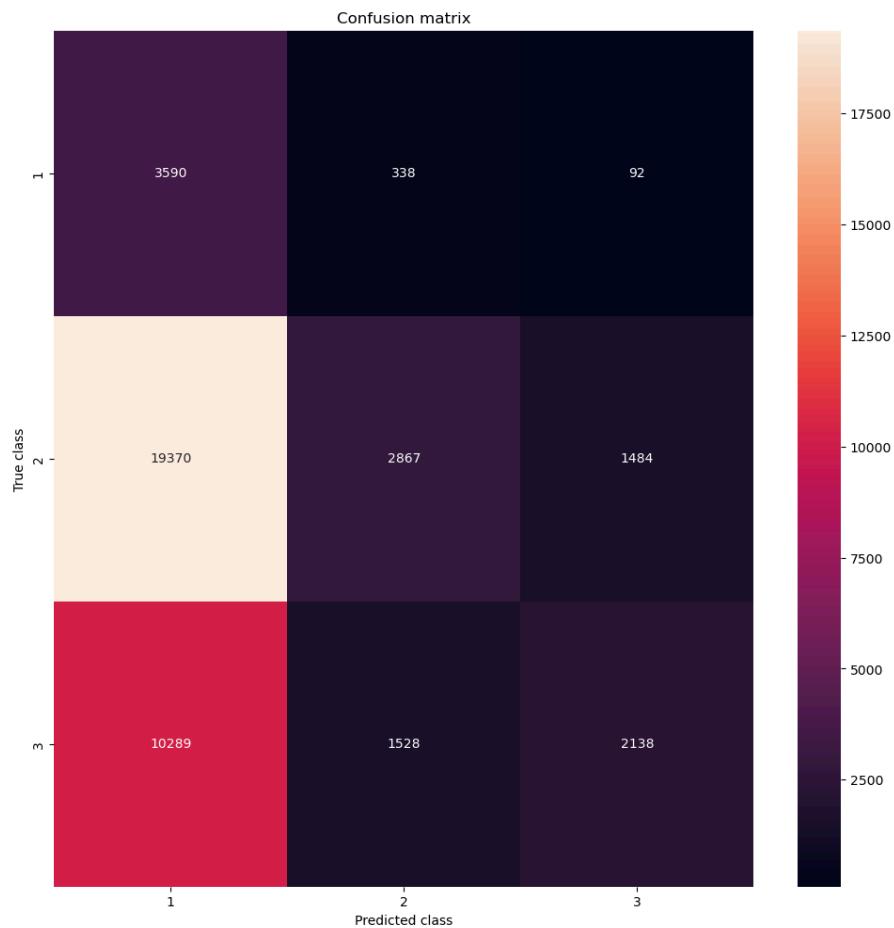
Por otro lado, aunque la distribución de aciertos ha mejorado, el accuracy ha empeorado bajando hasta un 0.51.

ESTRATEGIA: Subsampling de las clases mayoritarias

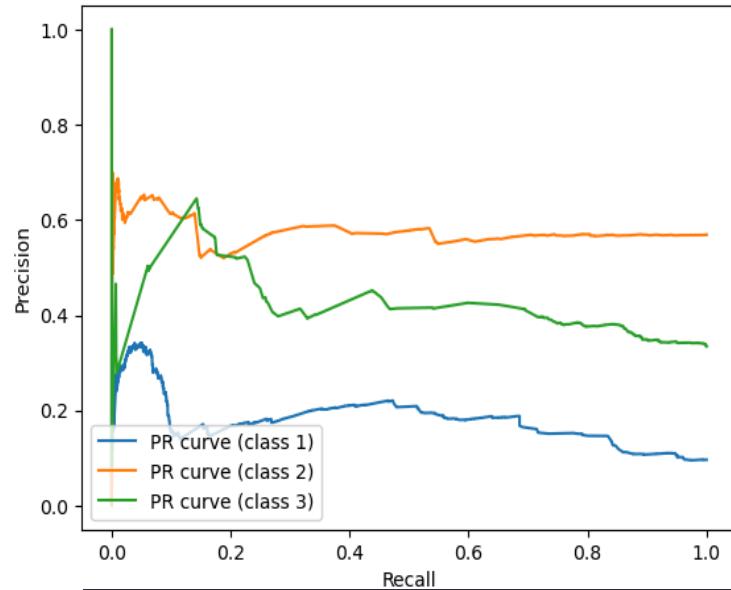
```
1 def subsampling_strategy(model, X_train, y_train, X_test, y_test):
2     us = NearMiss(n_neighbors=3, version=2)
3     X_train_res, y_train_res = us.fit_resample(X_train, y_train)
4
5     print ("Distribution before resampling {}".format(Counter(y_train)))
6     print ("Distribution after resampling {}".format(Counter(y_train_res)))
7
8     model.fit(X_train_res, y_train_res)
9     pred_y = model.predict(X_test)
10    mostrar_resultados(model, X_test, y_test, pred_y)
11
12    ✓ 0.0s
```

```
1 model = LogisticRegression(C=1.0,penalty='l2',random_state=1,solver="liblinear")
2 subsampling_strategy(model, X_train, y_train, X_val, y_val)
3 ✓ 14m 7.2s
```

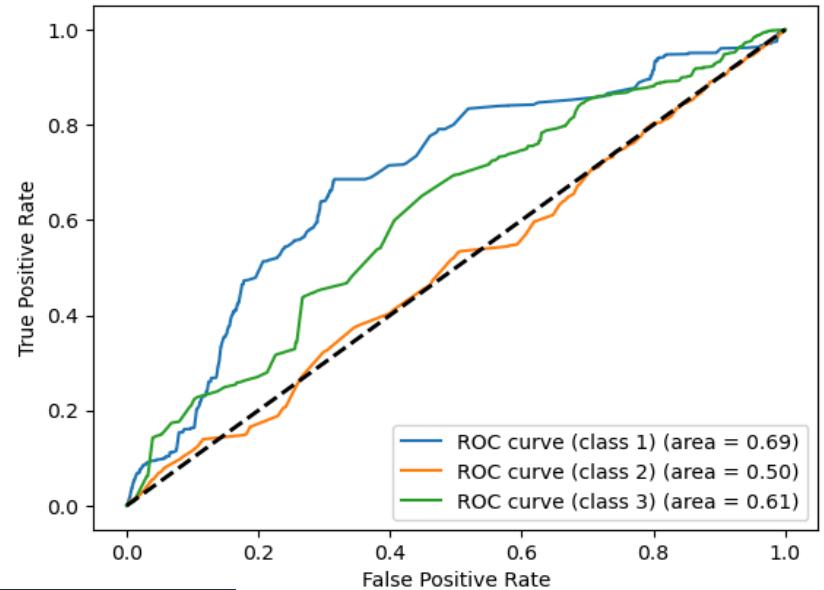
```
Distribution before resampling Counter({2: 94886, 3: 55819, 1: 16079})
Distribution after resampling Counter({1: 16079, 2: 16079, 3: 16079})
```



Precision-Recall Curve for Multiclass



Receiver Operating Characteristic (ROC) Curve for Multiclass



	precision	recall	f1-score	support
1	0.11	0.89	0.19	4020
2	0.61	0.12	0.20	23721
3	0.58	0.15	0.24	13955
accuracy			0.21	41696
macro avg	0.43	0.39	0.21	41696
weighted avg	0.55	0.21	0.21	41696

En este caso, vemos que esta vez se confunden las clases 2 y 3 con la clase 1. También podemos ver que en la curva de ROC la clase 2 está prácticamente encima de la línea diagonal y que la accuracy ha empeorado muchísimo, hasta el 0.21.

ESTRATEGIA: Oversampling de la clase minoritaria

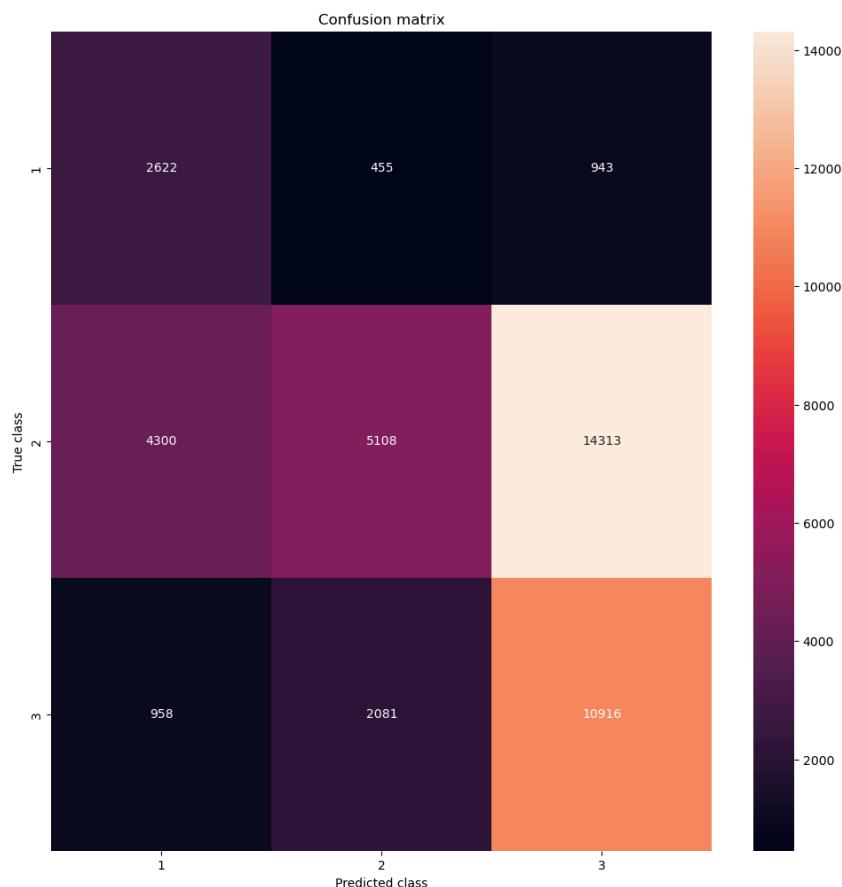
```
1 def oversampling_strategy(model, X_train, y_train, X_test, y_test):
2     os = RandomOverSampler()
3     X_train_res, y_train_res = os.fit_resample(X_train, y_train)
4
5     print ("Distribution before resampling {}".format(Counter(y_train)))
6     print ("Distribution labels after resampling {}".format(Counter(y_train_res)))
7
8     model.fit(X_train_res, y_train_res)
9     pred_y = model.predict(X_test)
10    mostrar_resultados(model, X_test, y_test, pred_y)
```

✓ 0.0s

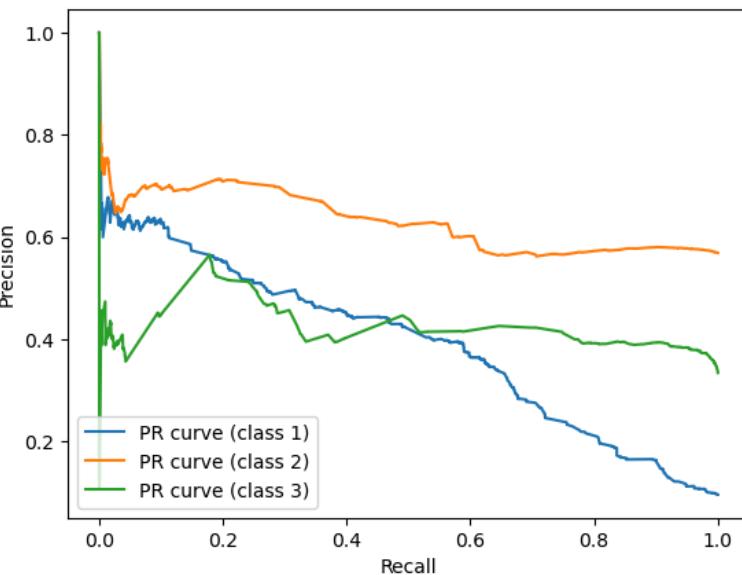
```
1 model = LogisticRegression(C=1.0,penalty='l2',random_state=1,solver="liblinear")
2 oversampling_strategy(model, X_train, y_train, X_val, y_val)
```

✓ 2.0s

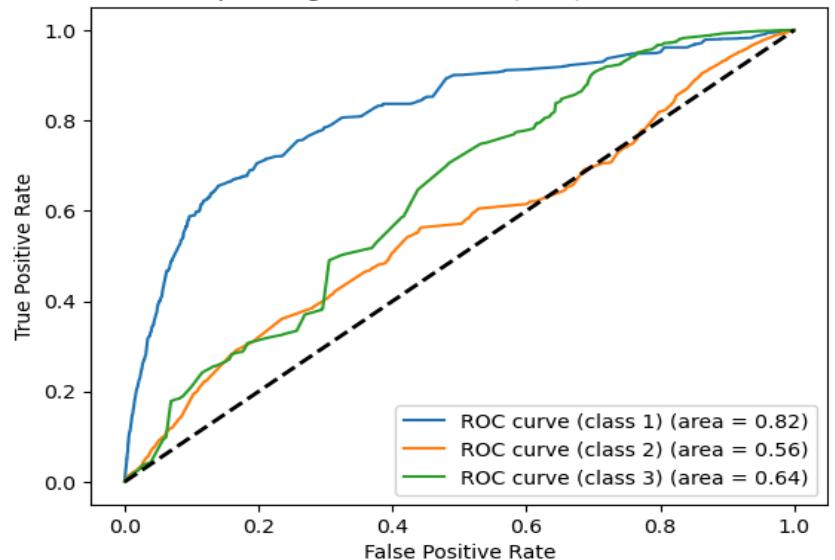
```
Distribution before resampling Counter({2: 94886, 3: 55819, 1: 16079})
Distribution labels after resampling Counter({3: 94886, 2: 94886, 1: 94886})
```



Precision-Recall Curve for Multiclass



Receiver Operating Characteristic (ROC) Curve for Multiclass

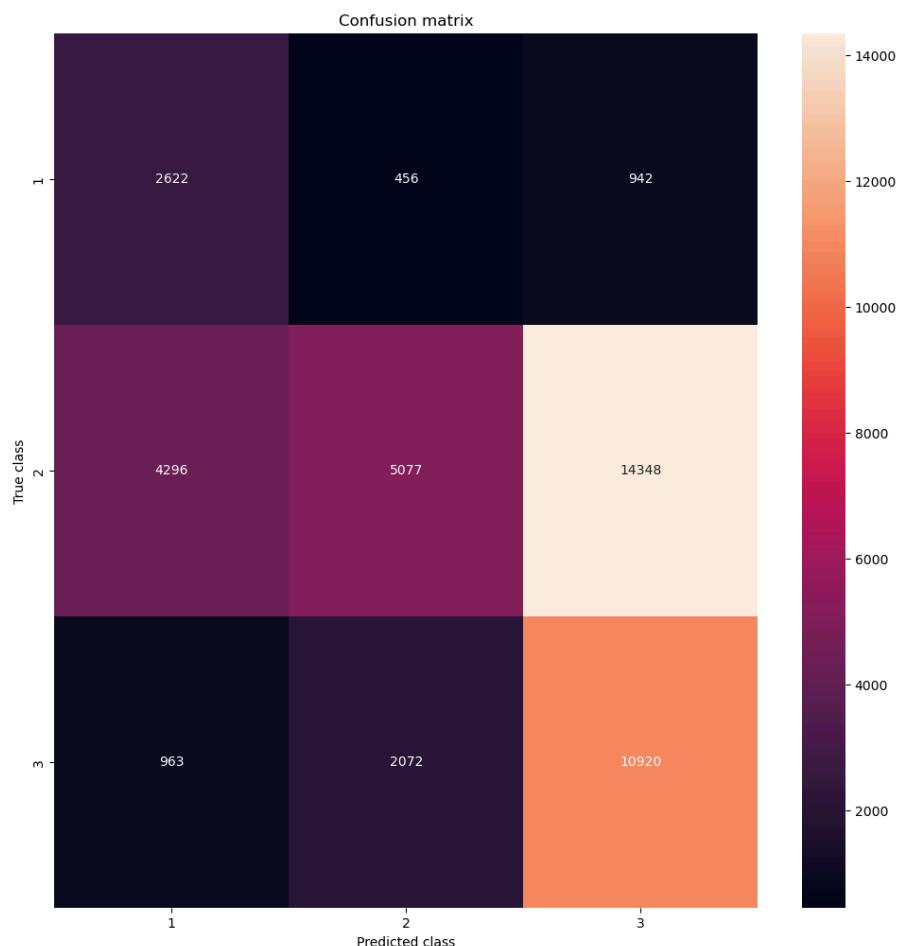


	precision	recall	f1-score	support
1	0.33	0.65	0.44	4020
2	0.67	0.22	0.33	23721
3	0.42	0.78	0.54	13955
accuracy			0.45	41696
macro avg	0.47	0.55	0.44	41696
weighted avg	0.55	0.45	0.41	41696

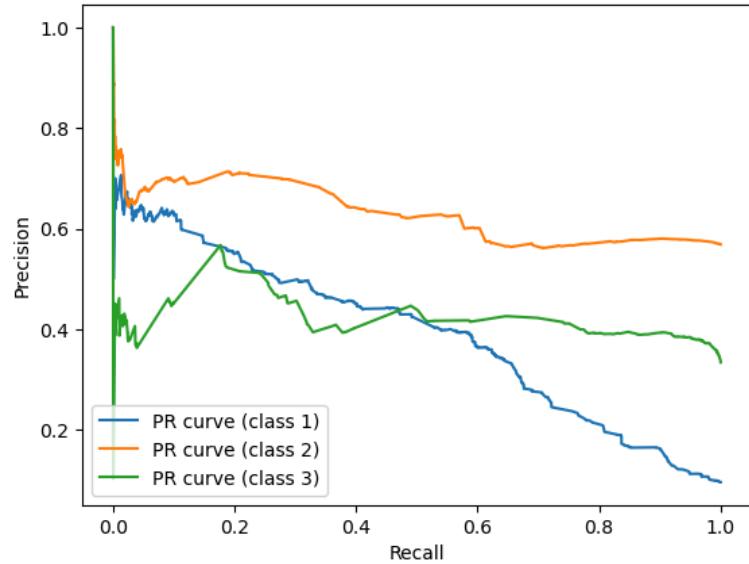
En este caso, se confunden las clase 2 con la 3 y la precisión descendió hasta el 0.45.

ESTRATEGIA: Oversampling de la clase minoritaria

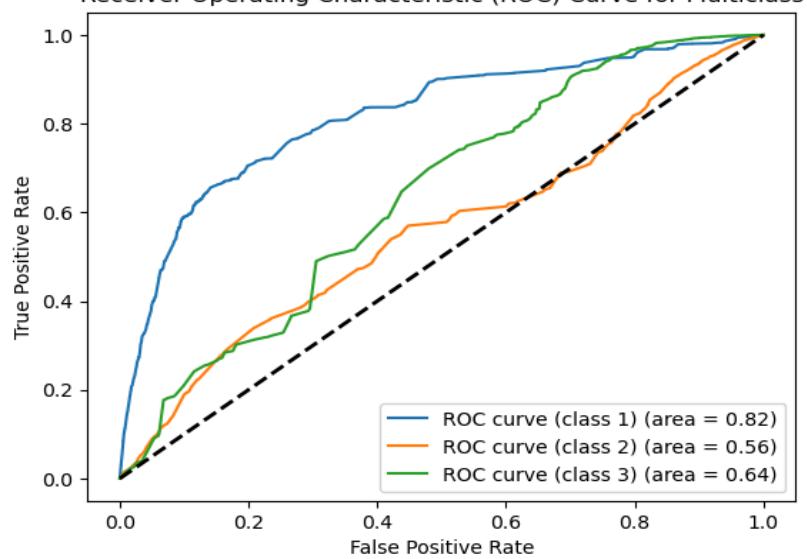
```
1 def resampling_with_SmoteTomek_strategy(model, X_train, y_train, X_test, y_test):
2     os_us = SMOTETomek()
3     X_train_res, y_train_res = os_us.fit_resample(X_train, y_train)
4
5     print ("Distribution before resampling {}".format(Counter(y_train)))
6     print ("Distribution after (parameter) X_test: Any Counter(y_train_res))")
7
8     model.fit(X_train_res, y_train_res)
9     pred_y = model.predict(X_test)
10    mostrar_resultados(model, X_test, y_test, pred_y)
11
12] ✓ 0.0s
13
14 1 model = LogisticRegression(C=1.0,penalty='l2',random_state=1,solver="liblinear")
15 2 resampling_with_SmoteTomek_strategy(model, X_train, y_train, X_val, y_val)
16] ✓ 11m 45.4s
17 Distribution before resampling Counter({2: 94886, 3: 55819, 1: 16079})
18 Distribution after resampling Counter({2: 94792, 3: 94737, 1: 94695})
```



Precision-Recall Curve for Multiclass



Receiver Operating Characteristic (ROC) Curve for Multiclass



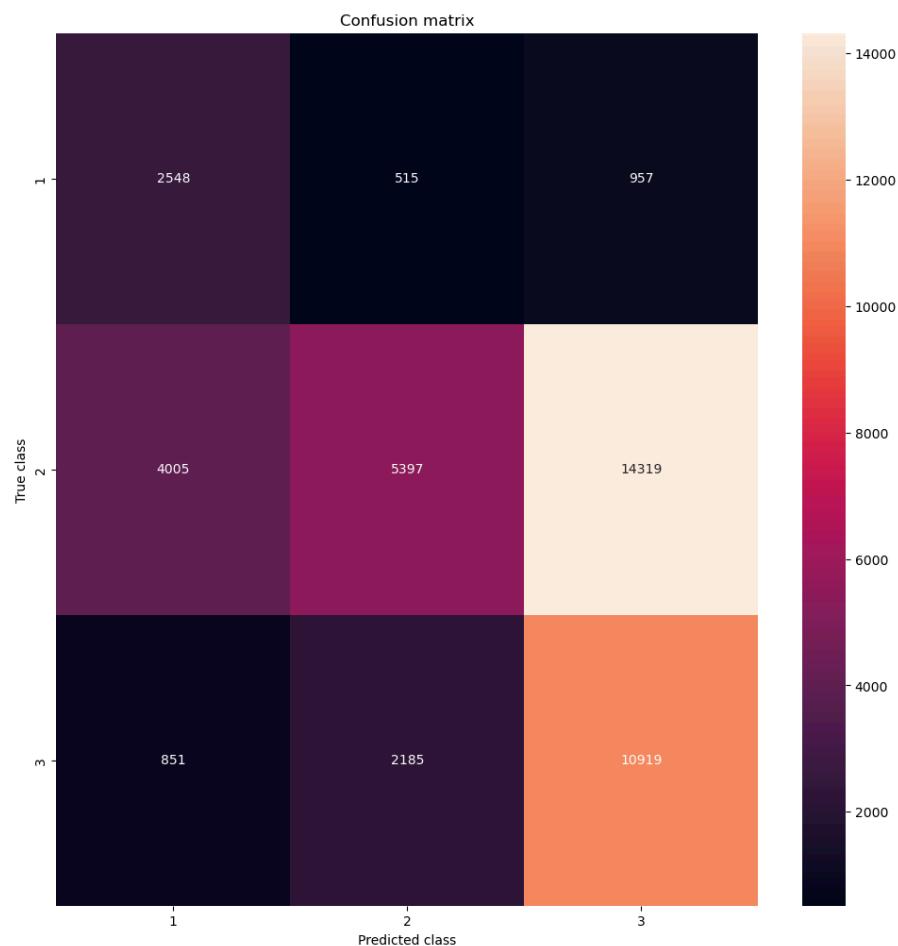
	precision	recall	f1-score	support
1	0.33	0.65	0.44	4020
2	0.67	0.21	0.32	23721
3	0.42	0.78	0.54	13955
accuracy			0.45	41696
macro avg	0.47	0.55	0.44	41696
weighted avg	0.55	0.45	0.41	41696

Este caso es muy parecido al anterior, donde se confunden la clase 2 con la 3.

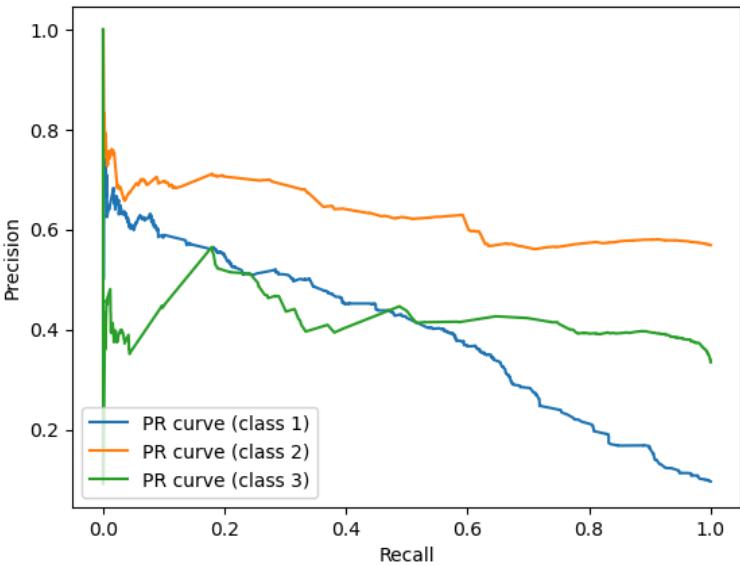
ESTRATEGIA: Ensamble de Modelos con Balanceo

```
1 def balanced_ensable_models_strategy(model, X_train, y_train, X_test, y_test):
2     bbc = BalancedBaggingClassifier(estimator=model,
3                                     sampling_strategy='auto',
4                                     replacement=False,
5                                     random_state=0)
6
7     #Train the classifier.
8     bbc.fit(X_train, y_train)
9     pred_y = bbc.predict(X_test)
10    mostrar_resultados(model, X_test, y_test, pred_y)
```

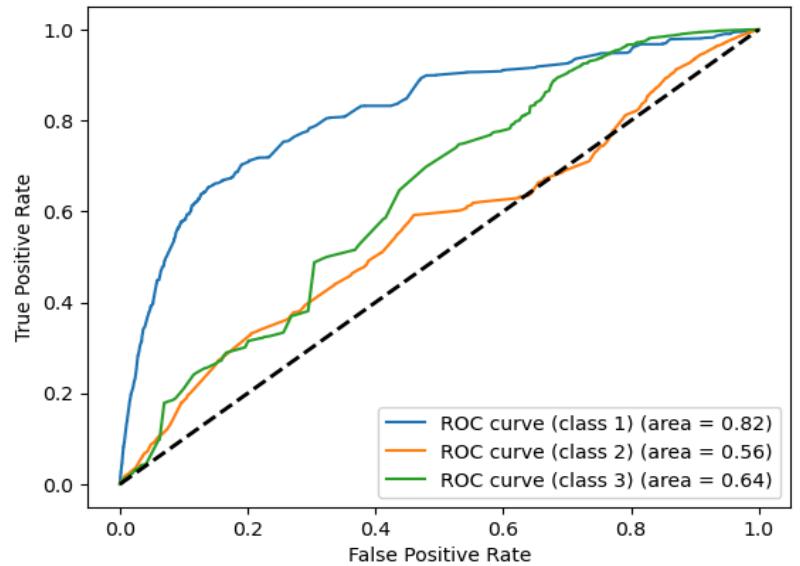
✓ 0.0s



Precision-Recall Curve for Multiclass



Receiver Operating Characteristic (ROC) Curve for Multiclass



	precision	recall	f1-score	support
1	0.34	0.63	0.45	4020
2	0.67	0.23	0.34	23721
3	0.42	0.78	0.54	13955
accuracy			0.45	41696
macro avg	0.48	0.55	0.44	41696
weighted avg	0.55	0.45	0.42	41696

Aquí volvemos a tener el mismo resultado que en los dos casos anteriores.

CONCLUSIÓN

La mejor estrategia es la de penalización, es donde más aciertos hay por clase y mejor accuracy en proporción a los aciertos.

ENTRENAMIENTO

Para este caso elegí el modelo de regresión logística LogisticRegression para resolver este problema debido a su eficiencia computacional, regularización incorporada para prevenir el sobreajuste, interpretación sencilla de coeficientes y buen desempeño en conjuntos de datos linealmente separables. Su simplicidad y capacidad para manejar conjuntos de datos grandes lo hacen una opción sólida en este contexto.

Logistic Regression

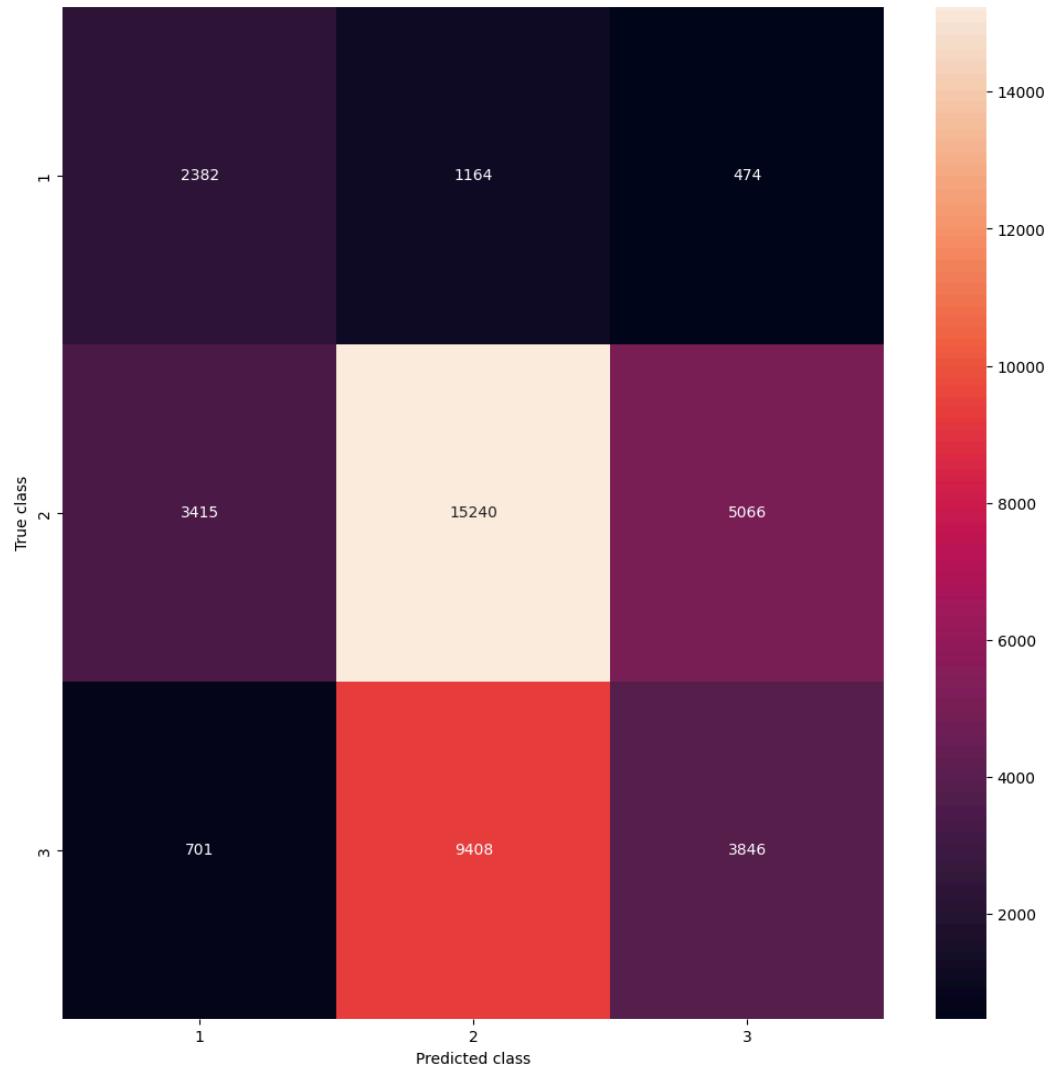
```
1 # Definir el espacio de búsqueda de hiperparámetros
2 params_lr = {
3     'C': [1.0, 5.0, 10.0],
4     'penalty': ['l2'],
5     'random_state': [1, 42, 100],
6     'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
7     'class_weight': ['balanced']
8 }
9
10 lr = LogisticRegression(C=1.0,penalty='l2',random_state=1,solver="liblinear",class_weight="balanced")
11
```

```
best_params(model=lr, params=params_lr, CV_technique="all", max_splits=15)
```

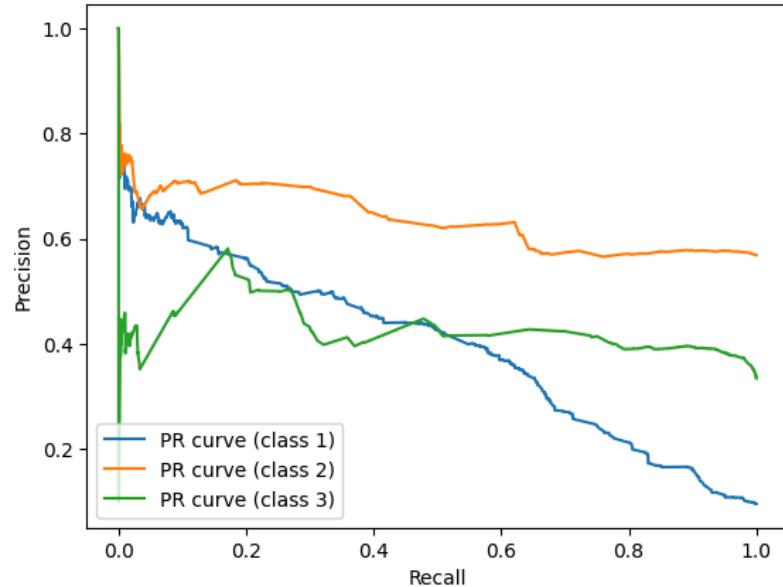
```
min: 2 max: 15 actual: 12
search: 7 - 15 actual: 14
gridSearchCV Finish
randomizeSearchCV Finish
Best accuracy: 0.5162245778971604 , Best params: {'C': 1.0, 'class_weight': 'balanced', 'penalty': 'l2', 'random_state': 1, 'solver': 'liblinear'}
```

La función best_params es una función creada por mi para que entrene con varios splits el modelo que le pase con GridSearchCV y RandomizedSearchCV.

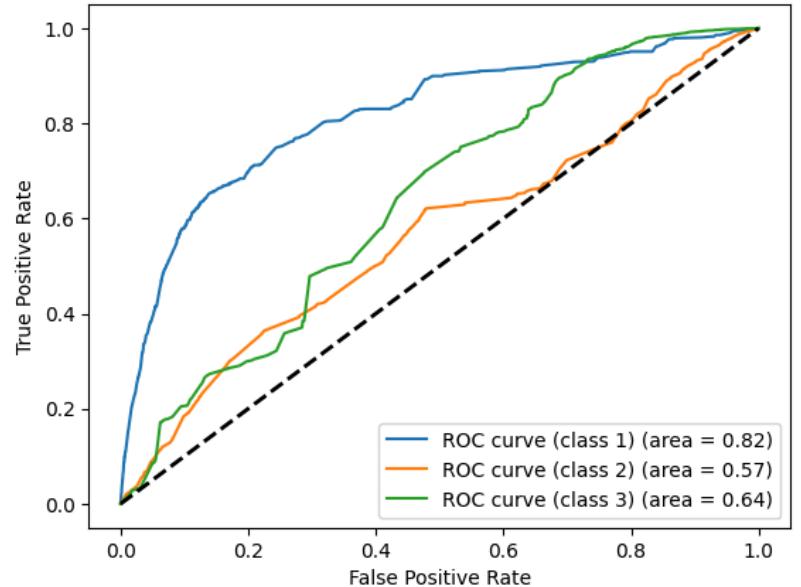
Confusion matrix



Precision-Recall Curve for Multiclass



Receiver Operating Characteristic (ROC) Curve for Multiclass



	precision	recall	f1-score	support
1	0.37	0.59	0.45	4020
2	0.59	0.64	0.62	23721
3	0.41	0.28	0.33	13955
accuracy			0.51	41696
macro avg	0.46	0.50	0.47	41696
weighted avg	0.51	0.51	0.50	41696

Como podemos comprobar, el modelo no mejora con la validación cruzada, queda prácticamente igual.

LAZY PREDICT

Usamos LazyPredict para que nos recomiende otro modelo para compararlo con LogisticRegression.

```
1 # Crea un clasificador LazyPredict
2 clf = LazyClassifier(predictions=True)
3 models, predictions = clf.fit(X_train, X_val, y_train, y_val)
4
5 # Muestra los modelos y sus puntajes
6 print(models)
```

Model				
ExtraTreeClassifier	0.67	0.57	None	0.66
ExtraTreesClassifier	0.67	0.57	None	0.66
DecisionTreeClassifier	0.67	0.57	None	0.65
BaggingClassifier	0.67	0.56	None	0.65
KNeighborsClassifier	0.64	0.56	None	0.63
LGBMClassifier	0.67	0.56	None	0.66
RandomForestClassifier	0.67	0.56	None	0.65
AdaBoostClassifier	0.64	0.52	None	0.61
NearestCentroid	0.41	0.52	None	0.36
QuadraticDiscriminantAnalysis	0.40	0.52	None	0.28
GaussianNB	0.40	0.51	None	0.28
BernoulliNB	0.58	0.48	None	0.52
PassiveAggressiveClassifier	0.40	0.46	None	0.34
SVC	0.60	0.44	None	0.53
LinearDiscriminantAnalysis	0.56	0.44	None	0.45
Perceptron	0.45	0.39	None	0.46
LogisticRegression	0.57	0.39	None	0.45
CalibratedClassifierCV	0.57	0.37	None	0.44
SGDClassifier	0.57	0.36	None	0.43
RidgeClassifier	0.57	0.36	None	0.43
RidgeClassifierCV	0.57	0.36	None	0.43
LinearSVC	0.57	0.36	None	0.43
DummyClassifier	0.57	0.33	None	0.41

Model	Time Taken
ExtraTreeClassifier	0.14
ExtraTreesClassifier	4.71
DecisionTreeClassifier	0.20

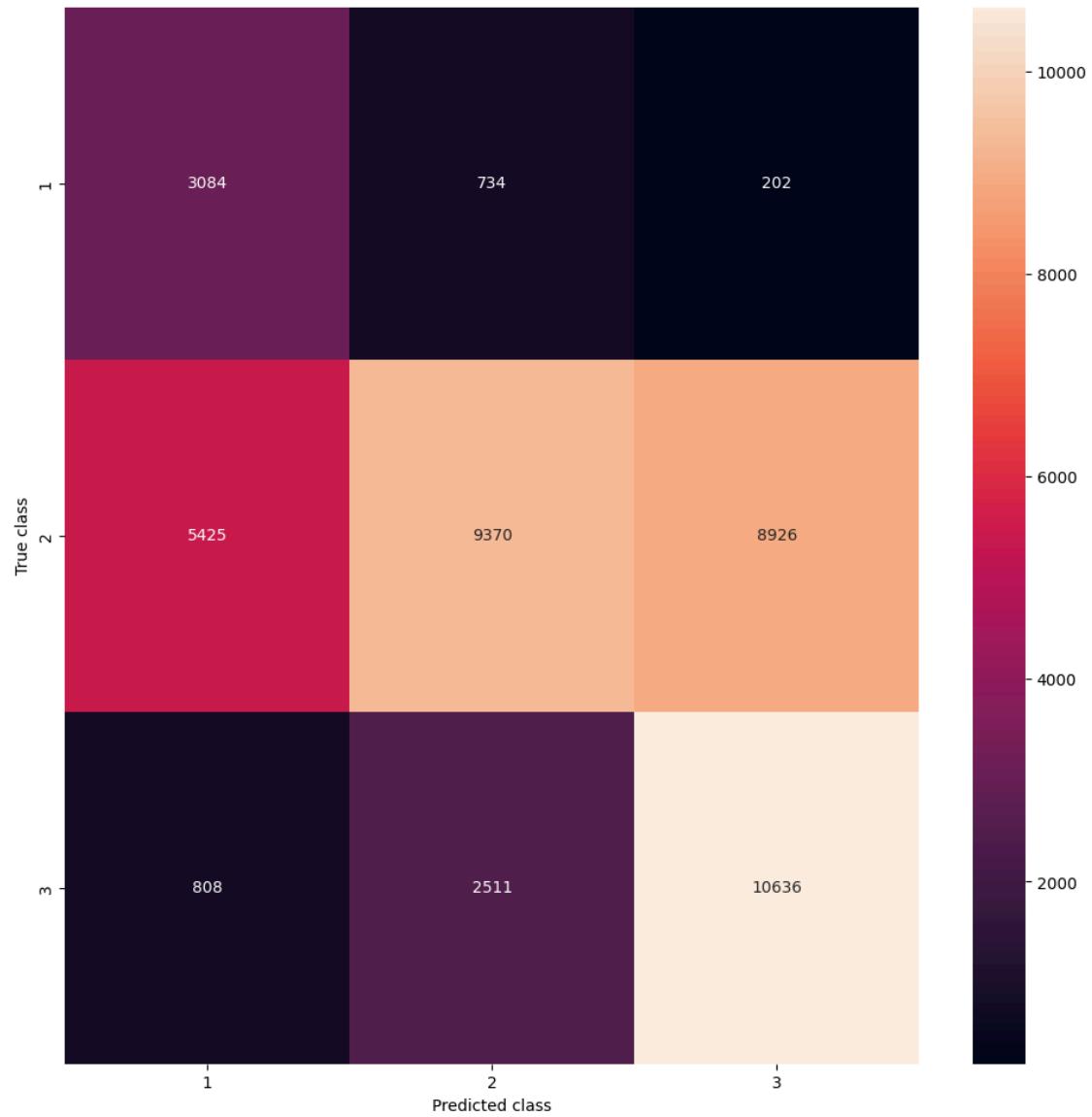
En este caso vamos a elegir el ExtraTreeClassifier porque tiene buenos resultados y tarda menos en entrenar.

ExtraTreeClasifier

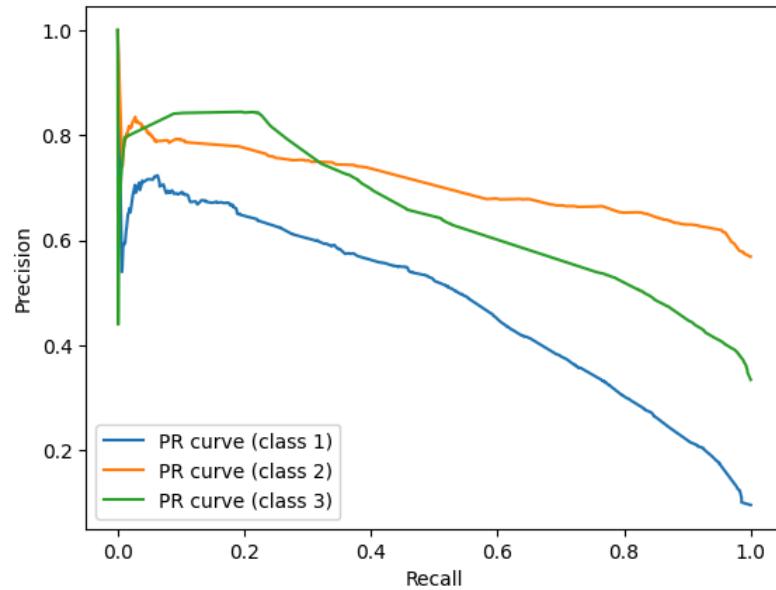
```
2
3 # Definir los parámetros a buscar
4 params_clf = {
5     'criterion': ['gini', 'entropy'],
6     'splitter': ['best', 'random'],
7     'max_depth': [None, 5, 10, 15],
8     'min_samples_split': [2, 5, 10],
9     'min_samples_leaf': [1, 2, 4],
10    'max_features': ['auto', 'sqrt', 'log2'],
11    'class_weight': ['balanced']
12 }
13
14 # Crear el modelo ExtraTreeClassifier
15 clf = ExtraTreeClassifier(class_weight='balanced')
```

```
best_params(model=clf, params=params_clf, CV_technique="all", max_splits=15)
```

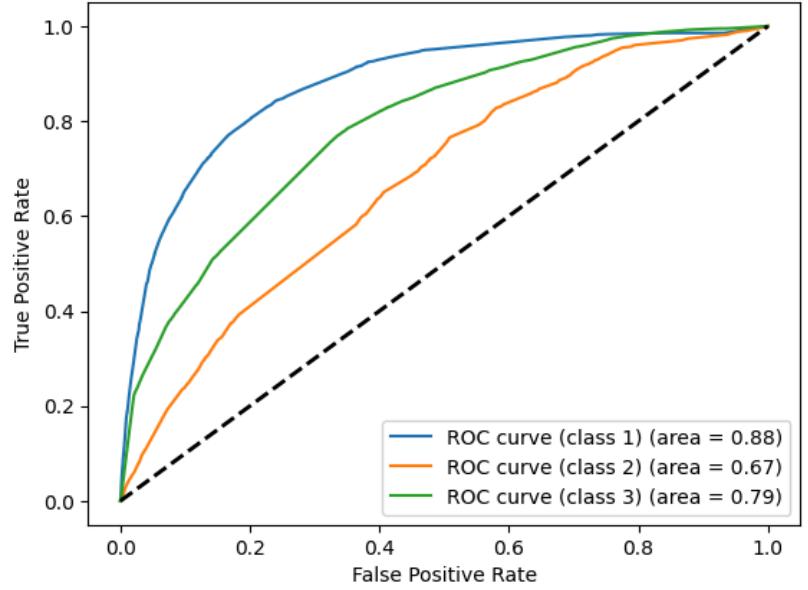
Confusion matrix



Precision-Recall Curve for Multiclass



Receiver Operating Characteristic (ROC) Curve for Multiclass



	precision	recall	f1-score	support
1	0.33	0.77	0.46	4020
2	0.74	0.40	0.52	23721
3	0.54	0.76	0.63	13955
accuracy			0.55	41696
macro avg	0.54	0.64	0.54	41696
weighted avg	0.63	0.55	0.55	41696

Este modelo confunde muy poco las clases 1 y 3, pero la clase 2 acierta la mitad y la otra la confunde con la clase 3. La matriz de ROC está muy bien distribuida, las tres clases están alejadas de la línea diagonal, por otro lado la curva de precisión-recall no es muy estable.

Por otra parte, la precisión ha mejorado bastante, hasta el 0.55.

CONCLUSIÓN

Para este problema es mejor entrenar con ExtraTreeClassifier recomendado por LazyPredict, ya que da mejores resultados en todos los sentidos en comparación a LogisticRegression.

PREDICCIÓN CON LOS DATOS DE LA COMPETICIÓN

Importamos los datos facilitados por la competición.

```
# Cargar el archivo CSV con los datos a predecir  
data_to_predict = pd.read_csv('https://raw.githubusercontent.com/Nestorbd/Richter-s-Predictor-Modeling-Earthquake-Damage/master/datasets/test_values.csv')
```

+ Código + Markdown

Tratemos los datos igual que hicimos con el dataset de entrenamiento.

```
1 data_to_predict['land_surface_condition'].replace(['n', 'o', 't'],[0, 1, 2], inplace=True)  
2 data_to_predict['foundation_type'].replace(['h', 'i', 'r', 'u', 'w'],[0, 1, 2, 3, 4], inplace=True)  
3 data_to_predict['roof_type'].replace(['n', 'q', 'x'],[0, 1, 2], inplace=True)  
4 data_to_predict['ground_floor_type'].replace(['f', 'm', 'v', 'x', 'z'],[0, 1, 2, 3, 4], inplace=True)  
5 data_to_predict['other_floor_type'].replace(['j', 'q', 's', 'x'],[0, 1, 2, 3], inplace=True)  
6 data_to_predict['position'].replace(['j', 'o', 's', 't'],[0, 1, 2, 3], inplace=True)  
7 data_to_predict['plan_configuration'].replace(['a', 'c', 'd', 'f', 'm','n', 'o', 'q', 's','u'],[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],  
8 data_to_predict['legal_ownership_status'].replace(['a', 'r', 'v', 'w'],[0, 1, 2, 3], inplace=True)
```

Normalizamos los datos.

```
1 column_names_test = data_to_predict.columns.values  
2  
3 scaler_test = StandardScaler()  
4 scaled_data_test = scaler_test.fit_transform(data_to_predict)  
5 scaled_df_test = pd.DataFrame(scaled_data_test, columns=column_names_test)
```

Seleccionamos las características de entrenamiento.

```
1 scaled_df_test = scaled_df_test[all_selected_features]
```

Predecimos con el modelo cargado y descargamos la tabla en el formato requerido por la competición.

POSICIÓN EN LA COMPETICIÓN

Best score

0.5590

Current rank

#2202

Submissions used

0 of 3

Make new submission

You have **3 of 3** submissions left today. Your next submission can be on April 12, 2024 UTC.

REFERENCIAS

- ❖ [COMPETICIÓN](#)
- ❖ [GITHUB](#)
- ❖ [CLASIFICACIÓN CON DATOS DESBALANCEADOS](#)
- ❖ [MULTICLASS RECEIVER OPERATING CHARACTERISTIC \(ROC\)](#)