UNIVERSIDAD DE GRANADA

Inteligencia de Negocio

Práctica 3

Competición en Driven Data

David Carrasco Chicharro

Grupo de prácticas 2 (Jueves)

davidcch@correo.ugr.es

Score	Submitted by	Timestamp
0.6883	David_Carrasco_UGR	2019-12-05 10:05:33 UTC
0.6923	David_Carrasco_UGR	2019-12-05 12:05:20 UTC
0.6282	David_Carrasco_UGR	2019-12-22 13:33:31 UTC
0.6788	David_Carrasco_UGR	2019-12-22 14:31:36 UTC
0.6755	David_Carrasco_UGR	2019-12-23 04:30:28 UTC
0.6580	David_Carrasco_UGR	2019-12-23 12:10:19 UTC
0.6861	David_Carrasco_UGR	2019-12-23 18:15:17 UTC
0.6884	David_Carrasco_UGR	2019-12-24 00:00:46 UTC
0.6874	David_Carrasco_UGR	2019-12-24 14:12:40 UTC
0.7323	David_Carrasco_UGR	2019-12-27 00:26:24 UTC
0.7366	David_Carrasco_UGR	2019-12-28 07:35:20 UTC
0.7379	David_Carrasco_UGR	2019-12-28 16:35:13 UTC
0.7387	David_Carrasco_UGR	2019-12-28 16:41:15 UTC
0.7390	David_Carrasco_UGR	2019-12-29 16:23:10 UTC
0.7379	David_Carrasco_UGR	2019-12-29 19:30:25 UTC
0.7407	David_Carrasco_UGR	2019-12-29 19:40:40 UTC
0.7399	David_Carrasco_UGR	2019-12-30 15:45:27 UTC
0.7380	David_Carrasco_UGR	2019-12-30 23:32:54 UTC
0.7293	David_Carrasco_UGR	2019-12-31 10:07:58 UTC

Índice

1. Introducción

El problema que se trata en esta última práctica de la asignatura es utilizar métodos avanzados para aprendizaje supervisado en clasificación sobre una competición en *Driven Data*: `https://www.drivendata.org/competitions/57/nepal-earthquake/`. En ella el objetivo es predecir el nivel de daño, representado en una variable categórica, que sufre una edificación tras un terremoto, basándonos en un conjunto de datos extraídos tras el terremoto "Gorkha" en Nepal ocurrido en 2015.

El conjunto de datos proporcionado por la web de la competición consiste en información acerca de 260601 edificios. Cada uno presenta 38 variables (numéricas, categóricas y binarias) con las que trabajar para predecir la variable `damage_grade`, que representa el daño del edificio afectado por el terremoto. Para medir el rendimiento de los algoritmos se utilizará la medida F1, que equilibra la precisión y el *recall* de un clasificador, utilizando en este caso la variante *F1 micropromedida* al tratarse de un problema de clasificación con tres valores posibles.

$$F_{micro} = \frac{2 \cdot P_{micro} \cdot R_{micro}}{P_{micro} + R_{micro}}$$

donde

$$P_{micro} = \frac{\sum_{k=1}^{3} TP_k}{\sum_{k=1}^{3}(TP_k + FP_k)}, R_{micro} = \frac{\sum_{k=1}^{3} TP_k}{\sum_{k=1}^{3}(TP_k + FN_k)}$$

siendo TP Verdaderos Positivos, FP Falsos Positivos, FN Falsos Negativos y k las clases 1, 2 y 3 de `damage_grade`.

Para el desarrollo de la práctica he hecho uso de *scripts* y de cuadernos de *Jupyter Notebook* –para pruebas y visualización de datos– en el lenguaje de programación *Python*.

2. Tabla de subidas

#	Fecha	Hora	Pos.	Score tra	Score Driven Data	Descripción proceso realizado	Descripción algoritmos empleados	Configuración de parámetros de algoritmos
0	05/12/19	11:05:33	299	0,7264	0,6883	Prueba script de partida	LightGBM	por defecto
1	05/12/19	13:05:20	309	0,8229	0,6923	Prueba con XGB	XGB	n_estimators=300, max_depth=10
2	22/12/19	14:33:31	355	0,6471	0,6282	Pondero cada clase con un peso inversamente proporcional a la aparición de cada clase	XGB con ponderación	n_estimators=500
3	22/12/19	15:32:36	356	0,7617	0,6788	Cambio parámetros de XGB	XGB	n_estimators=1000, max_depth=5
4	23/12/19	05:30:28	355	0,9785	0,6755	Elijo un conjunto de datos de entrenamiento más grande	XGB sin CV	n_estimators=1000, max_depth=15
5	23/12/19	13:10:19	356	0,6756	0,6580	Utilizo 8 algoritmos para ver qué filas se clasifican más veces de manera incorrecta y así no utilizarlas en el conjunto de entrenamiento	XGB sin CV	n_estimators=1000, max_depth=15
6	23/12/19	19:15:17	360	0,8378	0,6861	Igual que el anterior	XGB	n_estimators=1000, max_depth=15
7	24/12/19	00:01:00	362	0,8253	0,6884	Utilizo 3 algoritmos que votan por el resultado	XGB, LGBM y Random Forest	XGB y RF: n_estimators=1000, max_depth=15; LGBM: n_estimators=1000
8	24/12/19	15:12:40	362	0,6812	0,6874	Prueba con Random Forest	Random Forest	n_estimators=2500, max_depth=18
9	27/12/19	01:26:24	211	1	0,7323	Limpieza de datos eliminando dos características y algunas instancias	XGB	n_estimators=1500, max_depth=15
10	28/12/19	08:35:20	198	0,9238	0,7366	Limpieza de datos con rango intercuartil	XGB	n_estimators=2000, max_depth=15, learning_rate=0,01
11	28/12/19	17:35:13	190	0,8527	0,7379	Igual que el anterior	XGB	n_estimators=1500, max_depth=10, learning_rate=0,03
12	28/12/19	17:41:15	182	0,8610	0,7387	Igual que el anterior	XGB	n_estimators=1000, max_depth=10, learning_rate=0,05
13	29/12/19	17:23:10	180	0,8656	0,7390	Limpieza de datos con rango intercuartil y eliminación de dos características	XGB	n_estimators=1000, max_depth=10, learning_rate=0,05
14	29/12/19	20:30:25	183	0,8644	0,7379	Igual que el anterior, elimino outlayers y quito dos características más	XGB	n_estimators=1000, max_depth=10, learning_rate=0,05
15	29/12/19	20:40:40	171	-	0,7407	Votación de los 5 resultados anteriores	XGB	cinco anteriores
16	30/12/19	16:45:27	177	-	0,7399	Votación de 10,12,13,14 y Random Forest	XGB (4 primeros) y RF (último)	igual que el anterior
17	31/12/19	00:32:54	180	0,8595	0,7380	Limpieza de datos con cuartiles y selección de instancias y características	XGB	n_estimators=1000, max_depth=10, learning_rate=0,05
18	31/12/19	11:07:58	186	0,8838	0,7293	Igual que el anterior	XGB	n_estimators=2000, max_depth=15, learning_rate=0,05

Tabla 1: Tabla de subidas

3. Análisis exploratorio de datos

El conjunto de datos se presenta sin valores perdidos ni nulos, lo cual facilita en gran medida el trabajo a realizar. El comando de *Python* `data_x.isna().sum()` devuelve la lista de variables del *dataset* con el número de valores perdidos, el cual es cero en toda la lista. Lo mismo ocurre al comprobar los valores nulos con `data_x.isnull().sum()`. Con `data_x.describe()` obtenemos algo más de información acerca de cada variable:

	count	mean	std	min	25 %	50 %	75 %	max
building_id	260601	525675	304545	4	261190	525757	789762	1.05e+06
geo_level_1_id	260601	13.9004	8.03362	0	7	12	21	30
geo_level_2_id	260601	701.075	412.711	0	350	702	1050	1427
geo_level_3_id	260601	6257.88	3646.37	0	3073	6270	9412	12567
count_floors_pre_eq	260601	2.12972	0.727665	1	2	2	2	9
age	260601	26.535	73.5659	0	10	15	30	995
area_percentage	260601	8.01805	4.39223	1	5	7	9	100
height_percentage	260601	5.43437	1.91842	2	4	5	6	32
has_superstructure_adobe_mud	260601	0.0886451	0.284231	0	0	0	0	1
has_superstructure_mud_mortar_stone	260601	0.761935	0.4259	0	1	1	1	1
has_superstructure_stone_flag	260601	0.0343322	0.182081	0	0	0	0	1
has_superstructure_cement_mortar_stone	260601	0.0182348	0.1338	0	0	0	0	1
has_superstructure_mud_mortar_brick	260601	0.068154	0.25201	0	0	0	0	1
has_superstructure_cement_mortar_brick	260601	0.0752683	0.263824	0	0	0	0	1
has_superstructure_timber	260601	0.254988	0.435855	0	0	0	1	1
has_superstructure_bamboo	260601	0.0850112	0.278899	0	0	0	0	1
has_superstructure_rc_non_engineered	260601	0.04259	0.201931	0	0	0	0	1
has_superstructure_rc_engineered	260601	0.0158595	0.124932	0	0	0	0	1
has_superstructure_other	260601	0.0149846	0.121491	0	0	0	0	1
count_families	260601	0.983949	0.418389	0	1	1	1	9
has_secondary_use	260601	0.11188	0.315219	0	0	0	0	1
has_secondary_use_agriculture	260601	0.0643781	0.245426	0	0	0	0	1
has_secondary_use_hotel	260601	0.0336261	0.180265	0	0	0	0	1
has_secondary_use_rental	260601	0.00810051	0.0896377	0	0	0	0	1
has_secondary_use_institution	260601	0.000940135	0.0306473	0	0	0	0	1
has_secondary_use_school	260601	0.000360705	0.0189888	0	0	0	0	1
has_secondary_use_industry	260601	0.0010706	0.0327026	0	0	0	0	1
has_secondary_use_health_post	260601	0.000188027	0.013711	0	0	0	0	1
has_secondary_use_gov_office	260601	0.000145817	0.0120746	0	0	0	0	1
has_secondary_use_use_police	260601	8.82575e-05	0.00939415	0	0	0	0	1
has_secondary_use_other	260601	0.00511894	0.0713635	0	0	0	0	1

Tabla 2: Descripción de las variables

Se puede apreciar en la tabla 2 que no hay valores extraños.

En la siguiente gráfica se observa la distribución de las clases. Claramente existe desbalanceo, ocupando la clase 1 un 9,64 %, la clase 2 un 56,89 % y la 3 un 33,47 %.

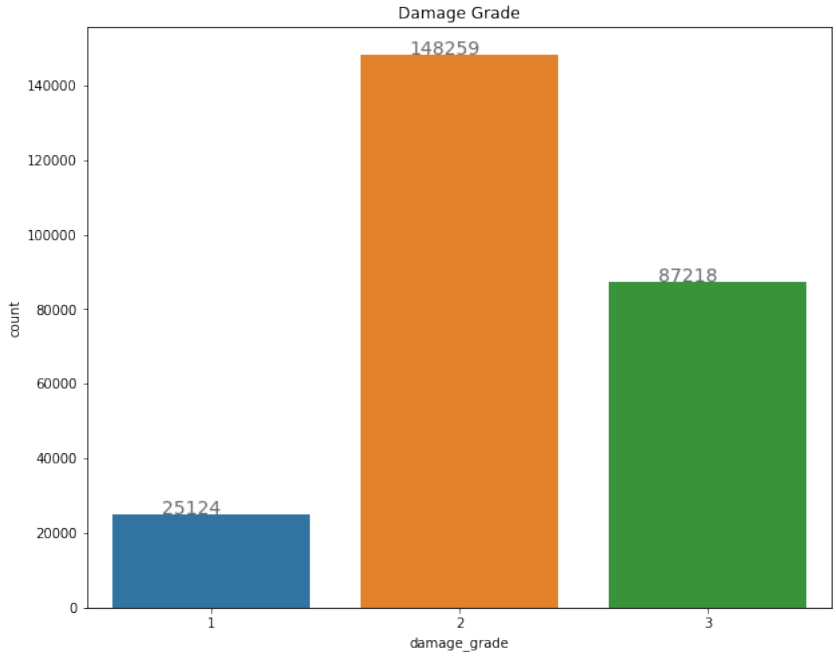

Figura 1: Distribución de las clases

La correlación entre variables numéricas no muestra nada especialmente revelador de cara a escoger, de manera visual, variables que influyan directamente en el resultado.

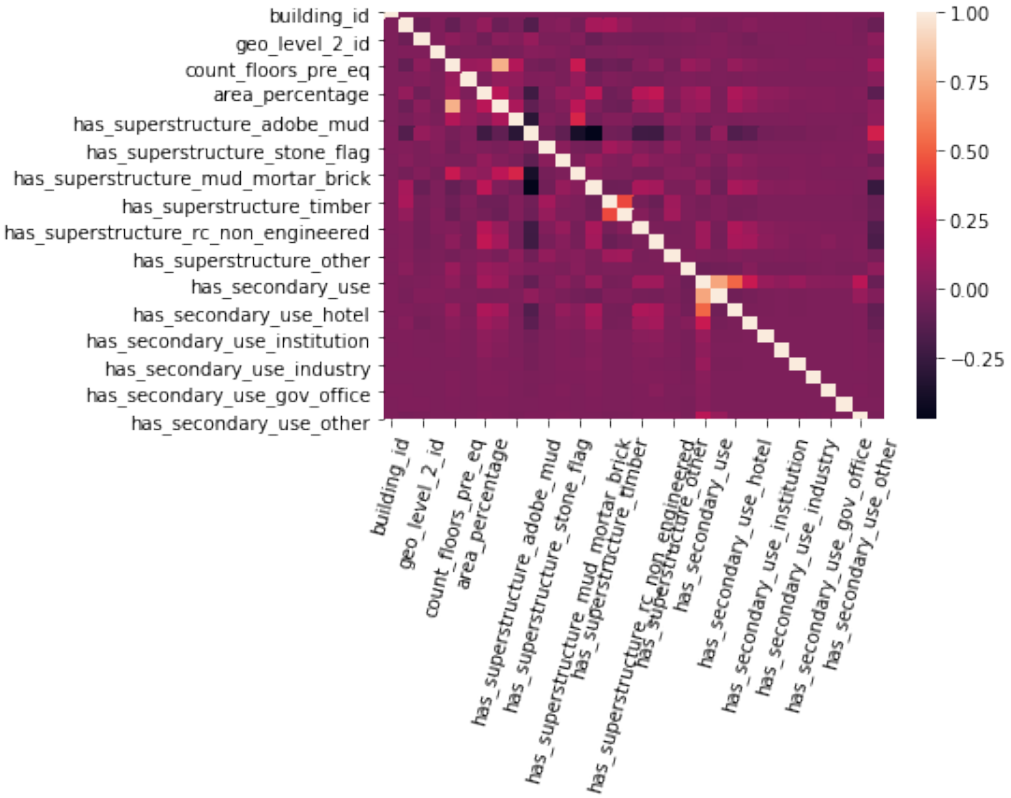

Figura 2: Correlación entre variables

4. Subidas

4.1. Subida 1

Tras la ejecución de prueba con el script de partida descargado de la web de la asignatura, cuyo resultado aparece en la tabla 1 como subida 0, ejecuté, también como prueba, el algoritmo *eXtreme Gradient Boosting* (XGB) con 300 estimadores y una profundidad máxima del árbol de 10 niveles.

```
clf = xgb.XGBClassifier(max_depth=10, n_estimators = 300, n_jobs=-1)
clf, y_test_clf = validacion_cruzada(clf,X,y,skf)
```

El resultado obtenido en *Driven Data* con este script (0,6923) no lo superé hasta la subida 9, puesto que en las seis subidas intermedias realicé diversas pruebas que se explicarán a continuación, todas ellas sin preprocesamiento, lo cual es clave en este problema.

4.2. Subida 2

En vista del desbalanceo de las clases asigno pesos –inversamente proporcionales a la aparición de `damage_grade` en el conjunto de datos– a cada clase para intentar balancearlas, pero el resultado arrojado fue el peor de todas las subidas (0,6282).

```
pesos = []
for i in data_y['damage_grade']:
        if i==1:
                pesos.append(float(0.675))
        elif i==2:
                pesos.append(float(0.125))
        else:
                pesos.append(float(0.2))

clf = xgb.XGBClassifier(n_estimators = 500)
clf, y_test_clf = validacion_cruzada(clf,X,y,skf)

clf = clf.fit(X,y, sample_weight=pesos)
```

4.3. Subida 3

Tras el fracaso de la subida anterior decidí seguir probando la configuración del algoritmo XGB. Intenté utilizar *Grid Search*, pero no me funcionaba con XGB al probar con más de 6-8 combinaciones, según el caso, por la potencia de mi ordenador, así que tuve que probar manualmente algunas configuraciones. En esta caso establecí el número de estimadores en 1000 y la profundidad máxima en 5. El resultado (0,6788) no fue el esperado, ya que creía que el aumento del número de estimadores ofrecería una mejorar sustancial.

4.4. Subida 4

En esta subida probé a no realizar la validación cruzada para comprobar los efectos que esto producía sobre el resultado final, el cual fue un *score* de 0,6755 (una puntuación similar, pero inferior, a la subida anterior) pero con un *score* sobre los datos de entrenamiento de 0,9785, lo cual indica un claro sobreajuste al no haber realizado la validación cruzada.

4.5. Subida 5

En esta ocasión decido implementar un nuevo algoritmo consistente en ejecutar previamente ocho configuraciones distintas de *Light GBM* y de *Random Forest* (2 y 6 respectivamente) de modo que una vez hechas las ocho predicciones se seleccionan aquellas filas del conjunto de entrenamiento que se clasifican mal un mayor número de veces y se eliminan del conjunto de entrenamiento que se ejecuta posteriormente con XGB.

```
lista_predicciones = [y_pred_tra1, y_pred_tra2, y_pred_tra3, y_pred_tra4,
↪  y_pred_tra5, y_pred_tra6, y_pred_tra7, y_pred_tra8]
sumas = []
for i in range(0,len(y_pred_tra1)):
        suma = 0
        for pred in lista_predicciones:
                suma += abs(y[i]-pred[i])
        sumas.append(suma)

total = len(sumas)
mantener = []
for i in range(total):
    if sumas[i]<5:
        mantener.append(i)

X_nuevo = np.take(X,mantener,axis=0)
y_nuevo = np.take(y,mantener,axis=0)

clf = xgb.XGBClassifier(max_depth=15, n_estimators = 1000, n_jobs=-1)
clf = clf.fit(X_nuevo,y_nuevo)
```

La variable `suma` almacena la diferencia entre la clase predicha por cada algoritmo y la clase correcta; `sumas` es una lista con la suma de diferencias (errores), de donde se mantendrán aquellas filas donde el valor almacenado sea menor que 5.

A pesar de que este método parecía ser una buena opción para subir la puntuación, no fue así, lo cual se podría deber a que, al igual que en el caso anterior, no utilizo validación cruzada.

4.6. Subida 6

En vista de que la subida anterior no arrojó los resultados esperados, decidí ejecutar el mismo script pero con validación cruzada, de modo que mejoró el *score* en casi 0,03 (0,6861), pero aún así el resultado no superaba ni siquiera al resultado de partida de la subida 0.

4.7. Subida 7

El método de los dos casos anteriores fue descartado, y probé uno nuevo, consistente en ejecutar tres algoritmos (XGB, *Light GBM* y *Random Forest*), de modo que la tres predicciones realizadas eligen para cada fila el valor predicho en mayor número de ocasiones. En caso de empate elijo por defecto la clase 2 por ser la mayoritaria.

```
# XGB
clf = xgb.XGBClassifier(max_depth=15, n_estimators=1000, n_jobs=-1)
clf, y_test_clf = validacion_cruzada(clf,X,y,skf)
clf = clf.fit(X,y)
y_pred_tst_xgb = clf.predict(X_tst)

# LightGBM
lgbm = lgb.LGBMClassifier(objective='regression_l1', n_estimators=1000,
  ↪  n_jobs=-1)
lgbm, y_test_lgbm = validacion_cruzada(lgbm,X,y,skf)
lgbm = lgbm.fit(X,y)
y_pred_tst_lgbm = lgbm.predict(X_tst)

# Random Forest
rf = RandomForestClassifier(n_estimators=1000, max_depth=15, n_jobs=-1)
rf, y_test_lgbm = validacion_cruzada(rf,X,y,skf)
rf = rf.fit(X,y)
y_pred_tst_rf = rf.predict(X_tst)

total = len(X_tst)
lista_preds = [y_pred_tst_xgb, y_pred_tst_lgbm, y_pred_tst_rf]
result = []

for i in range(total):
  valores = [0,0,0]
  for pred in lista_preds:
    if pred[i]==1:
      valores[0]+=1
    elif pred[i]==2:
      valores[1]+=1
    else:
      valores[2]+=1

  if 3 in valores:
    result.append(valores.index(3)+1)
```

```
  elif 2 in valores:
    result.append(valores.index(2)+1)
  else:
    result.append(2)
```

Con este sistema de votación no mejoré mucho el resultado, pero al menos no fue tan malo como en casos anteriores.

4.8. Subida 8

En esta subida tan sólo realizo una prueba ajustando los parámetros de Random Forest al límite máximo que alcanzaba la memoria de mi ordenador, pues cuanto más aumento el número de estimadores y la profundidad máxima mejor es el *score*, pero ni siquiera ajustando los parámetros al máximo posible consigo igualar a XGB.

4.9. Subida 9

A partir de este instante decido empezar con el preprocesado de los datos en vista de que la configuración de los algoritmos no es primordial en este problema. Para ello, en primer lugar, elimino un par de columnas que a priori parecen poco relevantes y me quedo sólo con aquellas instancias donde el año de construcción es inferior o igual a 250 años.

```
building_damage1 = data_x.merge(data_y, how = 'inner', on = 'building_id')
building_damage1 = building_damage1.drop(columns ="has_secondary_use")
building_damage1 = building_damage1.drop(columns
  ↪  ="has_secondary_use_agriculture")
building_damage = building_damage1[building_damage1['age'] <= 250]

cat_feats = ['land_surface_condition', 'foundation_type', 'roof_type',
  ↪  'ground_floor_type', 'other_floor_type', 'position',
  ↪  'legal_ownership_status', 'plan_configuration']

# Se convierten las variables categóricas en numéricas
train_final = pd.get_dummies(building_damage,columns=cat_feats,drop_first=True)
test_final = pd.get_dummies(data_x_tst,columns=cat_feats,drop_first=True)

y_train=train_final.damage_grade
train=train_final.drop('damage_grade',axis=1)
X = train_final.drop('damage_grade',axis=1)
y = train_final['damage_grade']
```

Una vez hecha la limpieza de datos ejecuto XGB con 1500 estimadores y una profundidad del árbol máxima de 15 niveles. Es aquí donde se produce un punto de inflexión, obteniendo en *Driven Data* un *score* de 0,7323 y subiendo 151 puestos en la clasificación.

4.10. Subida 10

En lugar de seleccionar dos características de manera casi aleatoria decido calcular el rango intercuartílico del conjunto de datos, de modo que aquellos datos que queden fuera del primer y tercer cuartil no se tienen en cuenta para el conjunto de entrenamiento.

```
# División entre entidades numéricas y categórcias
building_damage_num_train = building_damage.select_dtypes(include=["number"])
building_damage_cat_train = building_damage.select_dtypes(exclude=["number"])

# Binarización entre V/F para los valores numéricos en el intervalo 3*sigma de una distribución normal
idx = np.all(stats.zscore(building_damage_num_train) < 3, axis=1)
# Rango intercuartil
Q1 = building_damage_num_train.quantile(0.02)
Q3 = building_damage_num_train.quantile(0.98)
IQR = Q3 - Q1
idx = ~((building_damage_num_train < (Q1 - 1.5 * IQR)) | (building_damage_num_train > (Q3 + 1.5 * IQR))).any(axis=1)
building_damage_cleaned = pd.concat([building_damage_num_train.loc[idx], building_damage_cat_train.loc[idx]], axis=1)

cat_feats = ['land_surface_condition', 'foundation_type', 'roof_type', 'ground_floor_type', 'other_floor_type', 'position', 'legal_ownership_status', 'plan_configuration']

# Se convierten las variables categóricas en numéricas
train_final = pd.get_dummies(building_damage_cleaned,columns=cat_feats,drop_first=True)
test_final = pd.get_dummies(data_x_tst,columns=cat_feats,drop_first=True)
```

Tras esta nueva limpieza ejecuto XGB con 2000 estimadores, 15 niveles de profundidad máxima y un valor de `learning_rate` igual a 0,01. El *score* mejora en 0,0043 respecto al caso anterior, siendo de 0,7366.

4.11. Subida 11

Como el uso de *Grid Search* no era posible con demasiados valores para la búsqueda óptima de hiperparámetros, hago una pequeña ejecución en un *notebook* con pocos valores y en varias tandas, de manera que encuentro que utilizando 1500 estimadores, una profundidad máxima del árbol de 10 niveles y un valor de tasa de aprendizaje de 0,03 en XGB los resultados pueden ser algo mejores. Utilizando pues el mismo script que en el caso anterior pero con esta nueva configuración consigo un resultado de 0,7379 (mejora de 0,0013) y reduzco significativamente el *score* sobre los datos de entrenamiento, que era demasiado elevado en las ejecuciones anteriores.

4.12. Subida 12

Exactamente igual que el caso anterior, pero variando el valor de `learning_rate` a 0,05. Consigo una leve mejora, obteniendo un *score* de 0,7387.

4.13. Subida 13

Hay algunas características que no son relevantes para elegir la clase a la que pertenece cada instancia. Para ello, una vez ajustado el modelo, con la orden `plot_importance(clf)` se puede ver por orden cómo de importante es cada característica, de donde obtengo una lista de 62 variables (pues las categóricas aparecen descompuestas de manera binaria por cada valor posible que pueden tomar) junto con la importancia que tienen respecto a la clase `damage_grade`. En dicha lista destacan especialmente todas la variantes de `plan_configuration` y `legal_ownership_status`, que aparecen en su mayoría en las últimas posiciones (algunas ni aparecen). En la imagen que se muestra a continuación se puede observar que la mayoría de valores de cada característica tiene una baja frecuencia de aparición.

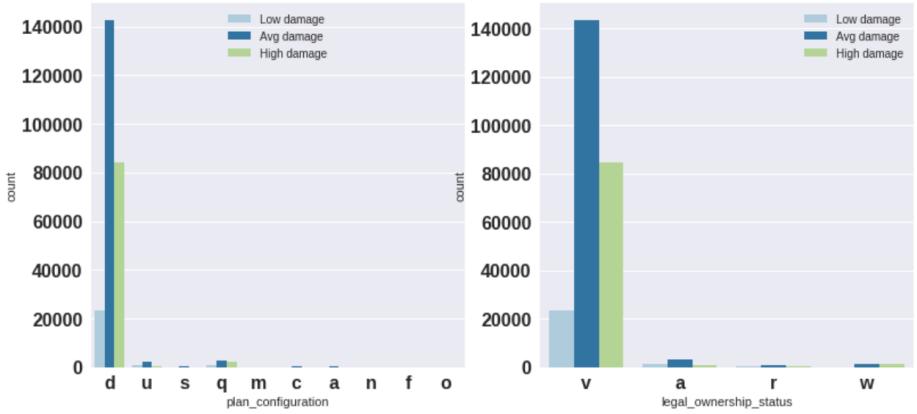

Figura 3: Apariciones de cada valor de `plan_configuration` y `legal_ownership_status`

Es por ello que decido omitir del conjunto de características `plan_configuration` y `legal_ownership_status`.

```
# Quedan fuera 'plan_configuration' y 'legal_ownership_status'
cat_feats = ['land_surface_condition', 'foundation_type', 'roof_type',
 ↪  'ground_floor_type', 'other_floor_type', 'position']
```

Aunque ínfima, pero vuelve a haber mejora.

4.14. Subida 14

El *dataset* presenta *outliers* en las variables `area_percentage`, `height_percentage` y `age`, así que elimino del conjunto aquellas instancias donde los valores se encuentran muy alejados del rango intercuartil. Elimino también cuatro características poco relevantes.

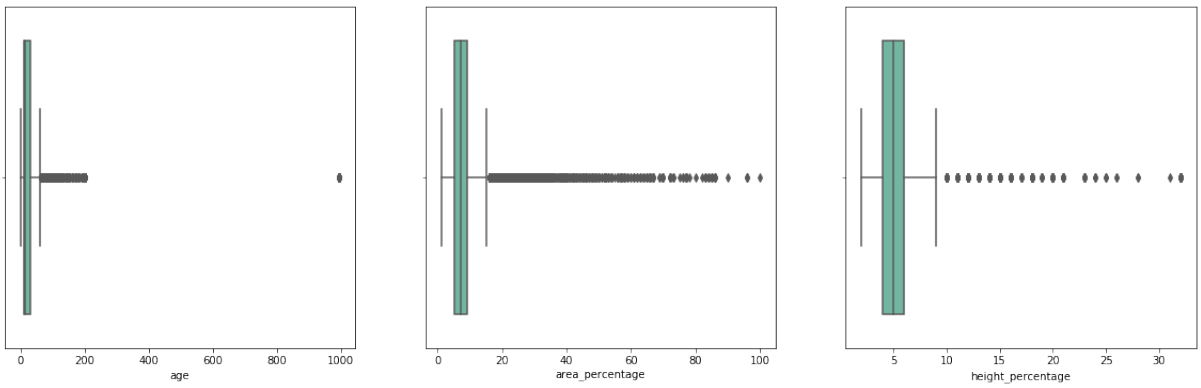

Figura 4: Outliers

```
# Eliminación de algunos outliers
bd_no_outly = building_damage_cleaned[building_damage_cleaned['age'] <= 250]
bd_no_outly = bd_no_outly[bd_no_outly['area_percentage'] <= 80]
bd_no_outly = bd_no_outly[bd_no_outly['height_percentage'] <= 25]

# Selección de características
building_damage_2 = bd_no_outly.drop(columns ="has_secondary_use")
building_damage_2 = building_damage_2.drop(columns
  ↪  ="has_secondary_use_agriculture")
building_damage_2 = building_damage_2.drop(columns ="plan_configuration")
building_damage_2 = building_damage_2.drop(columns ="legal_ownership_status")

# Quedan fuera 'plan_configuration' y 'legal_ownership_status'
cat_feats = ['land_surface_condition', 'foundation_type', 'roof_type',
       'ground_floor_type', 'other_floor_type', 'position']

# Se convierten las variables categóricas en numéricas
train_final =
  ↪  pd.get_dummies(building_damage_2,columns=cat_feats,drop_first=True)
test_final = pd.get_dummies(data_x_tst,columns=cat_feats,drop_first=True)
```

En esta ocasión, por primera vez desde que comencé a subir predicciones a la web con limpieza de datos, no conseguí mejorar el *score*, obteniendo el mismo resultado que en la subida 11.

4.15. Subida 15

Tras la subida del apartado anterior no sabía qué más podía hacer en cuanto a limpieza del conjunto de datos, así que implemento un algoritmo similar al de la subida 7 (4.7), que realiza un sistema de votación con los cinco mejores conjuntos de predicciones obtenidos hasta el momento, es decir, con las últimas cinco subidas (10 a 14), con la salvedad de que, en lugar de ejecutar los mismos algoritmos en el script, utilizo los ficheros `.csv` de salida de las subidas anteriores como entrada de las cinco predicciones a utilizar.

Para cada fila selecciona la clase de `damage_grade` que más veces aparezca. En caso de empate entre dos de los tres valores posibles se selecciona la clase 2 por defecto por ser la que más veces aparece, o la 3 en caso de que haya empate entre las clases 1 y 3.

```
lista_preds = [pred1, pred2, pred3, pred4, pred5]
result = []

for i in range(total):
  valores = [0,0,0]  #veces que aparecen los valores 1,2,3
  for pred in lista_preds:
    if pred[i]==1:
      valores[0]+=1
    elif pred[i]==2:
      valores[1]+=1
    else:
      valores[2]+=1

  if 5 in valores:
    result.append(valores.index(5)+1)
  elif 4 in valores:
    result.append(valores.index(4)+1)
  elif 3 in valores:
    result.append(valores.index(3)+1)
  else:
    if valores[1] == 2:
        result.append(2)
    else:
        result.append(3)
```

Es con esta ejecución con la que consigo el mejor resultado de todos, consiguiendo sobre pasar el 0,74.

4.16. Subida 16

La subida anterior podría haber realizado alguna predicción errónea teniendo en cuenta que los cinco algoritmos son del mismo tipo, cambiando tan solo algunos parámetros, por lo que cambio uno de los peores de los resultados escogidos, que es el de la subida 11 por otro de un *Random Forest*. Aunque el resultado obtenido es mejor que todos lo anteriores a la subida 15, no supera a esta última.

4.17. Subida 17

Intento mejorar el resultado de los scripts 10 a 14 (los previos a la votación). En este caso elimino, además de los outliers, la variable `position`, que era también una de las poco relevantes, además de las variables `plan_configuration` y `legal_ownership_status`. El resultado es similar a los casos anteriores.

4.18. Subida 18

Con el mismo script que en el caso anterior hago una última prueba cambiando parámetros, estableciendo el número de estimadores en el doble que solía utilizar (2000) y aumentando la profundidad máxima a 15. Disminuye de 0,73 el resultado obtenido.

5. Reflexiones

Esta práctica, a diferencia de las dos anteriores, me ha permitido conocer mucho más cómo funciona el aprendizaje supervisado, con mucha más profundidad que en la primera.

En el transcurso de la práctica me he dado cuenta de que quizás debería haber empezado antes a realizar el preprocesado de los datos, que es la parte más importante del problema, en lugar de probar las distintas configuraciones de los algoritmos. Este posible error se debe a que es la primera vez que me enfrento a un problema de estas características.

Me hubiera gustado probar otras técnicas de preprocesamiento y algún que otro algoritmo más para la competición, pero el tiempo que tarda en ejecutarse cada script, el límite de subidas diarias y las fechas en las que se desarrolla la práctica acotan mucho las posibilidades de realizar todo aquello que uno quiere desarrollar. Aún así, el hecho de que el formato se enmarque en el de una competición, hace que en todo momento se busque obtener un resultado mejor, lo cual es muy positivo para el aprendizaje, la competitividad y el aprendizaje. No obstante, aunque estoy contento con el resultado obtenido, no lo estoy tanto con mi posición, lo cual puede ser frustrante de cara a la nota que consiga, pues depende de lo que haga el resto aunque yo haya cumplido con mis objetivos.

6. Bibliografía

[1] *scikit-learn: Machine Learning in Python* (https://scikit-learn.org/)

[2] *XGBoost Python Package* (https://xgboost.readthedocs.io/)

[3] *Kaggle notebook: Rihcter's Prediction – Ajay Gorkar* (https://www.kaggle.com/ajaygorkar/rihcter-s-prediction-modeling-earthquake-damage)

[4] *GitHub notebook: Modeling Earthquake Damage – Rekha Remadevi* (https://github.com/rekharchandran/Modeling-Earthquake-Damage)

[5] *SciPy* (https://docs.scipy.org/)

[6] *Stack Exchange - Data Science* (https://datascience.stackexchange.com)

[7] *Efficient Majority Vote Algorithm* (https://medium.com/stephen-rambles/)

[8] *Preprocesamiento y calidad de datos* (https://sci2s.ugr.es/sites/default/files/ficherosPublicaciones/2133_Nv237-Digital-sramirez.pdf)