Inteligencia de Negocio (2019-2020)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 3

Andrés López Joya

BEST	CURRENT RANK	# COMPETITORS	SUBS. MADE
0.7389	197	1681	0 of 3

SUBMISSIONS

Score	Submitted by	Timestamp
0.7389	Andres_Lopez_Joya_UGR	2019-12-31 02:01:41 UTC
0.7320	Andres_Lopez_Joya_UGR	2019-12-31 13:24:48 UTC
0.7322	Andres_Lopez_Joya_UGR	2019-12-31 13:40:40 UTC

ÍNDICE

1.INTRODUCCIÓN.

Esta práctica consiste en una competición Basado en aspectos de edificios, el objetivo es predecir el nivel de daño a los edificios causado por el terremoto de Gorkha en 2015 en Nepal. El conjunto de entrenamiento consta de 260.601 instancias y 39 atributos (de los cuales, building id toma valores únicos y solo sirve para identificar cada ejemplo) categóricos, enteros y binarios. Se trata de predecir la variable ordinal damage grade. Que según la documentación que nos han aportado son y significan:

- ***geo_level_1_id, geo_level_2_id, geo_level_3_id (type: int):*** geographic region in which building exists, from largest (level 1) to most specific sub-region (level 3). Possible values: level 1: 0-30, level 2: 0-1427, level 3: 0-12567.
- ***count_floors_pre_eq (type: int):*** number of floors in the building before the earthquake.
- ***age (type: int):*** age of the building in years.
- ***area_percentage (type: int):*** normalized area of the building footprint.
- ***height_percentage (type: int):*** normalized height of the building footprint.
- ***land_surface_condition (type: categorical):*** surface condition of the land where the building was built. Possible values: n, o, t.
- ***foundation_type (type: categorical):*** type of foundation used while building. Possible values: h, i, r, u, w.
- ***roof_type (type: categorical):*** type of roof used while building. Possible values: n, q, x.
- ***ground_floor_type (type: categorical):*** type of the ground floor. Possible values: f, m, v, x, z.
- ***other_floor_type (type: categorical):*** type of constructions used in higher than the ground floors (except of roof). Possible values: j, q, s, x.
- ***position (type: categorical):*** position of the building. Possible values: j, o, s, t.
- ***plan_configuration (type: categorical):*** building plan configuration. Possible values: a, c, d, f, m, n, o, q, s, u.
- ***has_superstructure_adobe_mud (type: binary):*** flag variable that indicates if the superstructure was made of Adobe/Mud.
- ***has_superstructure_mud_mortar_stone (type: binary):*** flag variable that indicates if the superstructure was made of Mud Mortar - Stone.
- ***has_superstructure_stone_flag (type: binary):*** flag variable that indicates if the superstructure was made of Stone.
- ***has_superstructure_cement_mortar_stone (type: binary):*** flag variable that indicates if the superstructure was made of Cement Mortar - Stone.
- ***has_superstructure_mud_mortar_brick (type: binary):*** flag variable that indicates if the superstructure was made of Mud Mortar - Brick.
- ***has_superstructure_cement_mortar_brick (type: binary):*** flag variable that indicates if the superstructure was made of Cement Mortar - Brick.
- ***has_superstructure_timber (type: binary):*** flag variable that indicates if the superstructure was made of Timber.
- ***has_superstructure_bamboo (type: binary):*** flag variable that indicates if the superstructure was made of Bamboo.

- ***has_superstructure_rc_non_engineered (type: binary):*** flag variable that indicates if the superstructure was made of non-engineered reinforced concrete.
- ***has_superstructure_rc_engineered (type: binary):*** flag variable that indicates if the superstructure was made of engineered reinforced concrete.
- ***has_superstructure_other (type: binary):*** flag variable that indicates if the superstructure was made of any other material.
- ***legal_ownership_status (type: categorical):*** legal ownership status of the land where building was built. Possible values: a, r, v, w.
- ***count_families (type: int):*** number of families that live in the building.
- ***has_secondary_use (type: binary):*** flag variable that indicates if the building was used for any secondary purpose.
- ***has_secondary_use_agriculture (type: binary):*** flag variable that indicates if the building was used for agricultural purposes.
- ***has_secondary_use_hotel (type: binary):*** flag variable that indicates if the building was used as a hotel.
- ***has_secondary_use_rental (type: binary):*** flag variable that indicates if the building was used for rental purposes.
- ***has_secondary_use_institution (type: binary):*** flag variable that indicates if the building was used as a location of any institution.
- ***has_secondary_use_school (type: binary):*** flag variable that indicates if the building was used as a school.
- ***has_secondary_use_industry (type: binary):*** flag variable that indicates if the building was used for industrial purposes.
- ***has_secondary_use_health_post (type: binary):*** flag variable that indicates if the building was used as a health post.
- ***has_secondary_use_gov_office (type: binary):*** flag variable that indicates if the building was used fas a government office.
- ***has_secondary_use_use_police (type: binary):*** flag variable that indicates if the building was used as a police station.
- ***has_secondary_use_other (type: binary):*** flag variable that indicates if the building was secondarily used for other purposes.

El lenguaje utilizado para esta práctica ha sido python y los algoritmos probados para la misma han sido KNN , RandomForest , GradientTree, ExtraTree y SVM. De todos los probados el que mejor resultados daba ha sido el RandomForest y por eso es el que he usado y voy a comentar.

2.VISUALIZACIÓN DE DATOS.

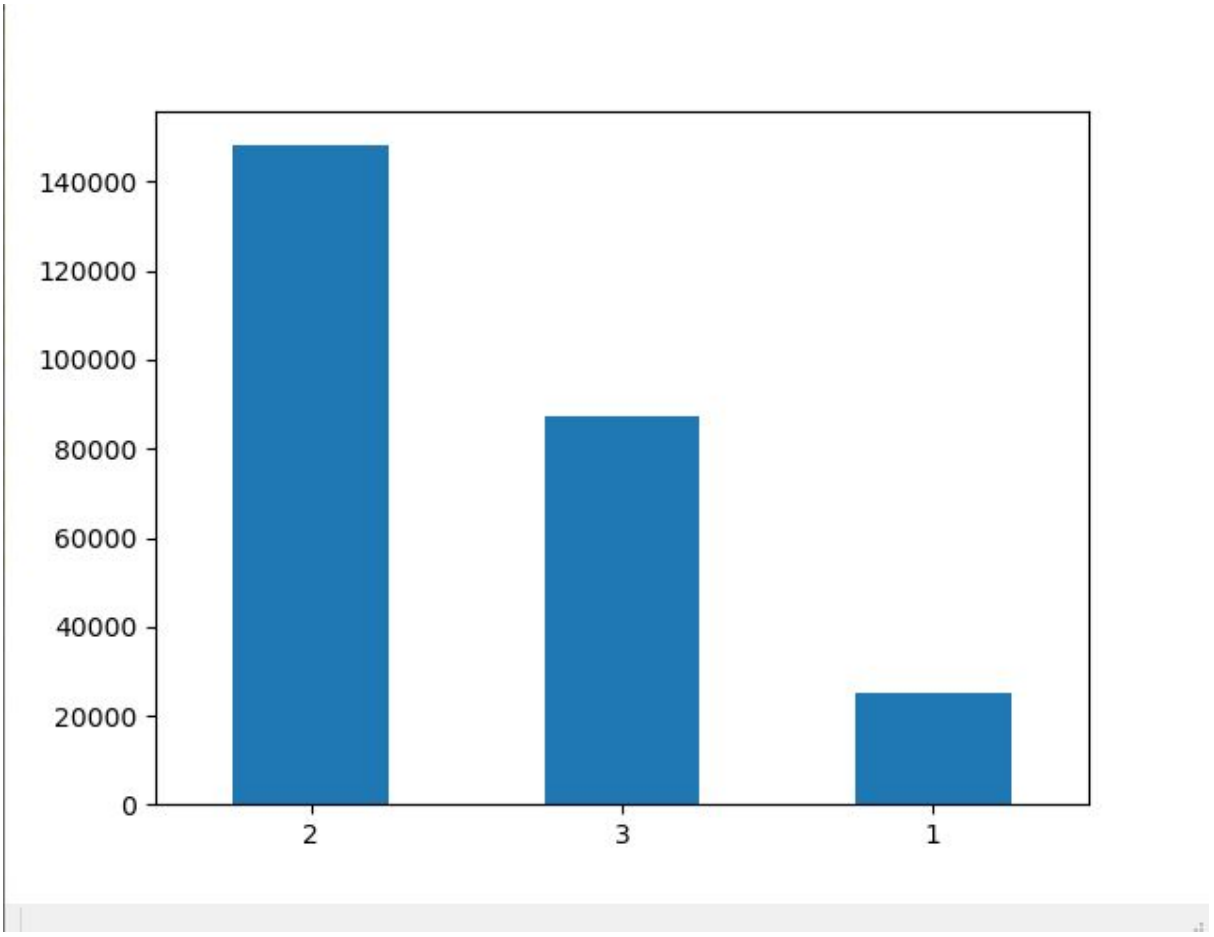
2.1DISTRIBUCIÓN DE LAS CLASES.

Primero vamos a visualizar la distribución que hay en los 3 distintos grados del desastre para ello vamos a usar estas líneas:

```
data_y.damage_grade.value_counts().plot(kind='bar')
```

```
plt.xticks(rotation = 0)
plt.show()
```

Con ellas podremos ver en una gráfica la distribución:



Como vemos hay un claro desnivel de entre el grado dos y los otros dos (3 y 1), el grado dos con más de 140000 mientras que el 3 está entre 80000 y 100000 y el grado 1 entre 20000 y 40000 claramente de grado 1 hay muy pocas muestras.

2.2 VALORES PERDIDOS.

Vamos a comprobar si hay valores perdidos, para ello :

```
print("Valores perdidos en x:")
print(data_x.isnull().sum())
```

Con estas líneas veremos los nulos de x y con :

```
print("Valores perdidos en tst:")
print(data_x_tst.isnull().sum())
```

Los nulos del test.

```
Valores perdidos en tst:
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
```

```
Valores perdidos en x:
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
```

Una vez hecho esto podemos apreciar que no hay valores nulos en estas muestras.

2.3 TIPOS DE ATRIBUTOS.

Después quise ver de qué tipo era cada atributo para ver de qué manera se podría tratar mejor, para ello :

```
print("TIPOS")
print(data_x.dtypes)
```

Esto nos muestra:

```
TIPOS
building_id                              int64
geo_level_1_id                           int64
geo_level_2_id                           int64
geo_level_3_id                           int64
count_floors_pre_eq                      int64
age                                      int64
area_percentage                          int64
height_percentage                        int64
land_surface_condition                  object
foundation_type                         object
roof_type                               object
ground_floor_type                       object
other_floor_type                        object
position                                object
plan_configuration                      object
has_superstructure_adobe_mud             int64
has_superstructure_mud_mortar_stone      int64
has_superstructure_stone_flag            int64
has_superstructure_cement_mortar_stone   int64
has_superstructure_mud_mortar_brick      int64
has_superstructure_cement_mortar_brick   int64
has_superstructure_timber                int64
has_superstructure_bamboo                int64
has_superstructure_rc_non_engineered     int64
has_superstructure_rc_engineered         int64
has_superstructure_other                 int64
```

Como podemos apreciar la gran mayoría son enteros a excepción de 8 objetos , aunque hay que tener en cuenta que gran parte de ellos son booleanos .

2.4 OBSERVACIONES ADICIONALES.

Si nos fijamos en los datos , podemos ver que hay algunas variables que a simple vista parece que no cambian su valor , para comprobar esto tenemos que ver los diferentes valores que toman las variables, para ello:

```
print('geo_level_1_id:\n')
print(data_x['geo_level_1_id'].value_counts()[0:6])
print('\ngeo_level_2_id:\n')
print(data_x['geo_level_2_id'].value_counts()[0:6])
print('\ngeo_level_3_id:\n')
print(data_x['geo_level_3_id'].value_counts()[0:6])
print('\ncount_floors_pre_eq:\n')
```

.
.
.

```
print('has_secondary_use_gov_office:\n')
```

```
print(data_x['has_secondary_use_gov_office'].value_counts()[0:6])
print('\nhas_secondary_use_use_police:\n')
print(data_x['has_secondary_use_use_police'].value_counts()[0:6])
print('\nhas_secondary_use_other:\n')
print(data_x['has_secondary_use_other'].value_counts()[0:6])
```

y podemos ver que hay varias variables que no poseen un claro poder discriminatorio ,con el único fin de producir ruido a la hora de la predecir.

```
has_secondary_use_school:

0    260507
1        94
Name: has_secondary_use_school, dtype: int64

has_secondary_use_industry:

0    260322
1       279
Name: has_secondary_use_industry, dtype: int64

has_secondary_use_health_post:

0    260552
1        49
Name: has_secondary_use_health_post, dtype: int64
has_secondary_use_gov_office:

0    260563
1        38
Name: has_secondary_use_gov_office, dtype: int64

has_secondary_use_use_police:

0    260578
1        23
```

2.PREPROCESADO.

Aquí voy a explicar las distintas acciones que he tomado para el preprocesado.

2.1ELIMINACIÓN DE ATRIBUTOS

Para el preprocesado de mi práctica lo que he hecho primero a sido borrar las variables antes mencionadas que iban a producir algún tipo de ruido , además del id.

```
columns_to_drop = ['building_id', 'has_secondary_use_use_police',
'has_secondary_use_gov_office',

'has_secondary_use_health_post','has_secondary_use_school','has_secondary_use_
institution',
                'has_secondary_use_industry']
data_x.drop(labels=columns_to_drop, axis=1, inplace = True)
```

data_x_tst.drop(labels=columns_to_drop, axis=1,inplace = True)
data_y.drop(labels=['building_id'], axis=1,inplace = True)

2.2 TRANSFORMACIÓN DE CATEGÓRICA A NUMÉRICA

Una vez que he eliminado esas variables lo que he hecho ha sido transformar las variables que son categóricas a numéricas , para ello :

categorical_feature_mask = data_x.dtypes==object // Primero obtengo una máscara de los que son de tipo categórico.
categorical_cols = data_x.columns[categorical_feature_mask].tolist() //Despues obtengo todas las columnas que sean de esa mascara de los datos proporcionados y la convierto a una lista.

data_x[categorical_cols] = data_x[categorical_cols].apply(lambda col: le.fit_transform(col))
data_x_tst[categorical_cols] = data_x_tst[categorical_cols].apply(lambda col: le.fit_transform(col)) // Por último aplico la transformación a las variables de data_x que he obtenido anteriormente , tanto en tst como en training.

2.3 NORMALIZACIÓN.

Como tercer paso del preprocesado he realizado una normalización de las cuales proporciona la librería preprocessing.

data_x = preprocessing.normalize(data_x, norm='l2')
data_x_tst= preprocessing.normalize(data_x_tst, norm='l2')

3.ALGORITMOS E INTENTOS EN DRIVENDATA.

En primera estancia destacar que los mejores resultados los he obtenido con el preprocesado mencionado y con el algoritmo RandomForest con unos parámetros que comentaré posteriormente, pero no solo he realizado las pruebas que aparecen en drivendata , he realizado pruebas con KNN obteniendo unos resultados bastante malos , alrededor de un score en los datos del test del 0.67 , con los parámetros por defecto, también he realizado pruebas con ExtraTree obteniendo resultados mejores que el KNN pero sin llegar al nivel del RandomForest, hice una prueba con el SVM el cual tenía un tiempo de ejecución descabelladamente grande , por lo que decidí eliminarlo y con el GradientBosting el cual obtenía resultados similares al Extra , pero como todos ellos tenían menor score que el RandomForest no lo consideré oportuno subirlo a DrivenData.

3.1 PRIMERA SUBIDA

Una vez dicho esto pasamos a explicar el algoritmo que he usado para el mejor score , el cual fue el primero que subí obteniendo la posición **193** y con un score de **0.7389** el cual daba un **0.9229** de score en los datos del training.

```
-----RANDOM FOREST-----
F1 score (tst): 0.7349, tiempo: 188.74 segundos
F1 score (tst): 0.7337, tiempo: 190.64 segundos
F1 score (tst): 0.7364, tiempo: 187.59 segundos
F1 score (tst): 0.7366, tiempo: 199.96 segundos
F1 score (tst): 0.7333, tiempo: 177.80 segundos

F1 score (tra): 0.9229
```

La configuración de mi algoritmo es:

```
RandomForestClassifier(n_estimators=320,
                       criterion="gini",
                       max_depth=None,
                       min_samples_split=8,
                       min_samples_leaf=1,
                       min_weight_fraction_leaf=0.,
                       max_features="auto",
                       max_leaf_nodes=None,
                       min_impurity_decrease=0.,
                       min_impurity_split=None,
                       bootstrap=True,
                       oob_score=True,
                       n_jobs=-1,
                       random_state=1,
                       verbose=0,
                       warm_start=False,
                       class_weight=None,
                       ccp_alpha=0.0,
                       max_samples=None
                       )
```

Primero le puse un número más alto de nodo que el por defecto, ya que veía que al aumentarle este número el score mejoraba aunque tardará más tiempo , el criterion es el por defecto y el max_depth también. Para el parámetro min_samples_split lo aumenté ya que vi que al aumentar el número por el que se dividía el nodo interno del árbol y con la relación del aumento de los nodos los datos iban mejorando. También puse el parámetro oob_score a true y el n_jobs a -1 para que use todos los procesadores . El random_state lo puse a 1 para controlar la aleatoriedad de las muestras al arrancar ya que el boostrap esta a 1 y para controlar el muestreo de las características a tener en cuenta para buscar la mejor división de nodo , el resto de atributos son por defecto.

Esta configuración junto con el preprocesado me dieron los mejores resultados.

3.2 SEGUNDA SUBIDA Y TERCERA SUBIDA

Ahora voy a pasar a explicar las otras dos subidas que hice a drivendata. Estas subidas tenían un preprocesado similar , solo que usando el código proporcionado por el profesor en el caso de pasar de categórica a numérica :

```
data_x_tmp = data_x.fillna(9999)
data_x_tmp = data_x.astype(str).apply(LabelEncoder().fit_transform)
data_x_nan = data_x_tmp.where(~mask, data_x)
mask = data_x_tst.isnull() #mÃ¡scara para luego recuperar los NaN
data_x_tmp = data_x_tst.fillna(9999) #LabelEncoder no funciona con NaN, se asigna un valor no usado
data_x_tst_tmp = data_x_tmp.astype(str).apply(LabelEncoder().fit_transform) #se convierten categÃ³ricas en numÃ©ricas
data_x_tst_nan = data_x_tst_tmp.where(~mask, data_x_tst)
```

Y en vez de usar la normalización proporcionada por preprocessing use una función propia la cual es:

```
def normalizar(valores):
    c1 = (valores - valores.min()) * 1.0
    c2 = (valores.max() - valores.min())
    return c1 / c2
```

Y con relación al algoritmo el único parámetro que se le ha cambiado ha sido : **n_estimators** de 320 a 500 para probar si aumentando más aún los nodos iba a seguir mejorando o iba a empeorar y claramente a empeorado con un score en drivendata del **0.7320** y con un scorer en los datos de training del **0.8994** , al ser un scorer menor que el primero que subí no me apareció la posición en la que estaría.

```
-----RANDOM FOREST-----
F1 score (tst): 0.7319, tiempo: 214.64 segundos
F1 score (tst): 0.7309, tiempo: 202.86 segundos
F1 score (tst): 0.7334, tiempo: 181.46 segundos
F1 score (tst): 0.7343, tiempo: 153.08 segundos
F1 score (tst): 0.7277, tiempo: 145.12 segundos

F1 score (tra): 0.8994
```

Para la tercera subida el preprocesado fue similar al anterior, pero como había cambiado varios parámetros en el preprocesado respecto a la primera subida quería probar cual era la diferencia entre el primero y el segundo con los mismos parámetros en el algoritmo, para ello volví a poner el **n_estimators** de 500 a 320 y comprobar resultados. Esta configuración me dio un scorer en el drivendata de **0.7322** un poco superior a la anterior pero aun así muy lejos de la primera subida, con un scorer en el training de **0.8989.**

```
-----RANDOM FOREST-----
F1 score (tst): 0.7321, tiempo:  97.94 segundos
F1 score (tst): 0.7299, tiempo:  98.33 segundos
F1 score (tst): 0.7335, tiempo:  99.44 segundos
F1 score (tst): 0.7345, tiempo:  97.62 segundos
F1 score (tst): 0.7277, tiempo:  97.58 segundos

F1 score (tra): 0.8989
```

4.TABLA RESUMEN.

N	FECHA	Posición	Train	DrivenData	Preprocesado	Algoritmo	Parámetro
1	2019-12-31 02:01:41 UTC	193	0.9229	0.7389	-Eliminación de columnas. -De categoricas a numéricas. -normalización de preprocessing l2	Random Forest	n_estimators=320 min_samples_split=8 oob_score=True n_jobs=-1 random_state=1 Los demás por defecto
2	2019-12-31 13:24:48 UTC	193	0.8994	0.7320	-Eliminación de columnas. -De categoricas a numéricas. -normalización con función propia.	Random Forest	n_estimators=500 min_samples_split=8 oob_score=True n_jobs=-1 random_state=1 Los demás por defecto
3	2019-12-31 13:40:40	193	0.8989	0.7322	-igual que la anterior	Random Forest	n_estimators=320 min_samples_split=8 oob_score=True n_jobs=-1 random_state=1 Los demás por defecto

Cabe recalcar que estas son solo las pruebas que subir a drivendata , en mi local y teniendo como referencia los datos que me salian en spyder hice muchas más.

5.BIBLIOGRAFÍA.

https://sci2s.ugr.es/graduateCourses/in
https://www.kaggle.com/mullerismail/richters-predictor-modeling-earthquake-damage/kernels
https://scikit-learn.org/stable/index.html