



ugr

Universidad
de Granada

Inteligencia de negocio

Práctica 3: Competición en DrivenData

Realizado por:
Francisco Solano López Rodríguez
DNI: 20100444P
Email: fransol0728@correo.ugr.es

















DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS.
QUINTO CURSO

ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



BEST	CURRENT RANK	# COMPETITORS	SUBS. TODAY
0.8246	98	6175	0 / 3

0.7183	Francisco_Lopez_UGR 	2018-12-28 21:35:48 UTC
0.7720	Francisco_Lopez_UGR 	2018-12-28 22:06:34 UTC
0.7678	Francisco_Lopez_UGR 	2018-12-28 23:07:54 UTC
0.7644	Francisco_Lopez_UGR 	2018-12-29 17:43:30 UTC
0.7728	Francisco_Lopez_UGR 	2018-12-29 20:56:11 UTC
0.7714	Francisco_Lopez_UGR 	2018-12-30 01:48:27 UTC
0.7705	Francisco_Lopez_UGR 	2018-12-30 22:07:26 UTC
0.7711	Francisco_Lopez_UGR 	2018-12-30 22:19:59 UTC
0.7642	Francisco_Lopez_UGR 	2018-12-31 13:36:39 UTC
0.7719	Francisco_Lopez_UGR 	2018-12-31 14:36:04 UTC
0.8215	Francisco_Lopez_UGR 	2018-12-31 19:59:13 UTC
0.8222	Francisco_Lopez_UGR 	2019-01-01 00:03:55 UTC
0.8232	Francisco_Lopez_UGR 	2019-01-01 00:11:52 UTC
0.8228	Francisco_Lopez_UGR 	2019-01-01 00:28:19 UTC
0.8224	Francisco_Lopez_UGR 	2019-01-02 13:09:17 UTC
0.8196	Francisco_Lopez_UGR 	2019-01-02 13:14:10 UTC
0.8208	Francisco_Lopez_UGR 	2019-01-02 13:19:06 UTC
0.8222	Francisco_Lopez_UGR 	2019-01-03 00:16:58 UTC
0.8215	Francisco_Lopez_UGR 	2019-01-03 00:28:10 UTC
0.7669	Francisco_Lopez_UGR 	2019-01-03 11:58:37 UTC
0.8213	Francisco_Lopez_UGR 	2019-01-04 11:42:37 UTC
0.8226	Francisco_Lopez_UGR 	2019-01-04 11:54:52 UTC
0.8215	Francisco_Lopez_UGR 	2019-01-04 15:09:13 UTC
0.8242	Francisco_Lopez_UGR 	2019-01-05 00:01:09 UTC
0.8239	Francisco_Lopez_UGR 	2019-01-05 00:14:26 UTC
0.8246	Francisco_Lopez_UGR 	2019-01-05 00:49:11 UTC

Índice

1. Introducción	3
2. Visualización de los datos	4
2.1. Distribución de las etiquetas	5
2.2. Valores perdidos	6
2.3. Correlación entre variables numéricas	8
2.4. Observaciones adicionales	9
3. Preprocesamiento	10
3.1. Eliminación de columnas	10
3.2. Tratamiento de valores perdidos	10
3.3. Valores raros	11
3.4. Coordenadas cartesianas	11
3.5. Fechas	13
3.6. Categóricas a numéricas	13
4. Validación cruzada	13
5. Progreso seguido	20
6. Bibliografía	21

1. Introducción

En esta práctica debemos de participar en una competición en DrivenData, en dicha competición tenemos que obtener soluciones para un problema de clasificación. Dicho problema consiste en predecir la condición operativa de un punto de agua para cada registro en el conjunto de datos. Como datos aportados por DrivenData disponemos de un conjunto de datos de entrenamiento el cual consta de 59400 ejemplos, y además del conjunto de datos para el test, de los cuales no disponemos de las etiquetas, el cual consta de 14850 ejemplo.

El número de características de las que disponemos es 39 (sin contar el 'id'). A continuación podemos ver cuales son estas características:

- **amount_tsh** - Cabezal estático total (cantidad de agua disponible para el punto de agua)
- **date_recorded** - La fecha en que se ingresó la fila.
- **funder** - ¿Quién financió el pozo?
- **gps_height** - Altitud del pozo.
- **installer** - Organización que instaló el pozo.
- **longitude** - coordenadas GPS
- **latitude** - coordenadas GPS
- **wpt_name** - Nombre del punto de agua si hay uno.
- **num_private** -
- **basin** - Cuenca geográfica del agua.
- **subvillage** - Ubicación geográfica
- **region** - Ubicación geográfica
- **region_code** - Ubicación geográfica (codificada)
- **district_code** - Ubicación geográfica (codificada)
- **lga** - Ubicación geográfica
- **ward** - Ubicación geográfica
- **population** - Población alrededor del pozo.
- **public_meeting** - Verdadero Falso
- **recorded_by** - Grupo entrando a esta fila de datos.
- **scheme_management** - Quien opera el punto de agua.
- **scheme_name** - Quien opera el punto de agua.

- **permit** - Si el punto de agua está permitido.
- **construction_year** - Año de construcción del punto de agua.
- **extraction_type** - El tipo de extracción que utiliza el punto de agua.
- **extraction_type_group** - El tipo de extracción que utiliza el punto de agua.
- **extraction_type_class** - El tipo de extracción que utiliza el punto de agua.
- **management** - Cómo se gestiona el punto de agua.
- **management_group** - Cómo se gestiona el punto de agua.
- **payment** - Lo que cuesta el agua.
- **payment_type** - Lo que cuesta el agua.
- **water_quality** - La calidad del agua.
- **quality_group** - La calidad del agua.
- **quantity** - La cantidad de agua.
- **quantity_group** - La cantidad de agua.
- **source** - La fuente del agua.
- **source_type** - La fuente del agua.
- **source_class** - La fuente del agua.
- **waterpoint_type** - El tipo de punto de agua.
- **waterpoint_type_group** - El tipo de punto de agua.

El lenguaje de programación utilizado ha sido Python. Los algoritmos utilizados en su mayoría han sido de la biblioteca sklearn. Algunos de los algoritmos probados han sido LightGBM, XGB, RandomForest o KNN. De todos los probados el que mejor resultados ha dado en la validación cruzada ha sido random forest, debido a ello ha sido el utilizado para la subida de soluciones a DrivenData.

2. Visualización de los datos

En esta sección verán diferentes gráficas e impresiones por pantalla para entender mejor los datos de los que disponemos.

2.1. Distribución de las etiquetas

En primer lugar vamos a ver la distribución de las etiquetas en el conjunto de entrenamiento. Los valores posibles se corresponde con

```
data_y = pd.read_csv('data/water_pump_tra-target.csv')
data_y.status_group.value_counts().plot(kind='bar')
plt.xticks(rotation = 0)
plt.show()
```

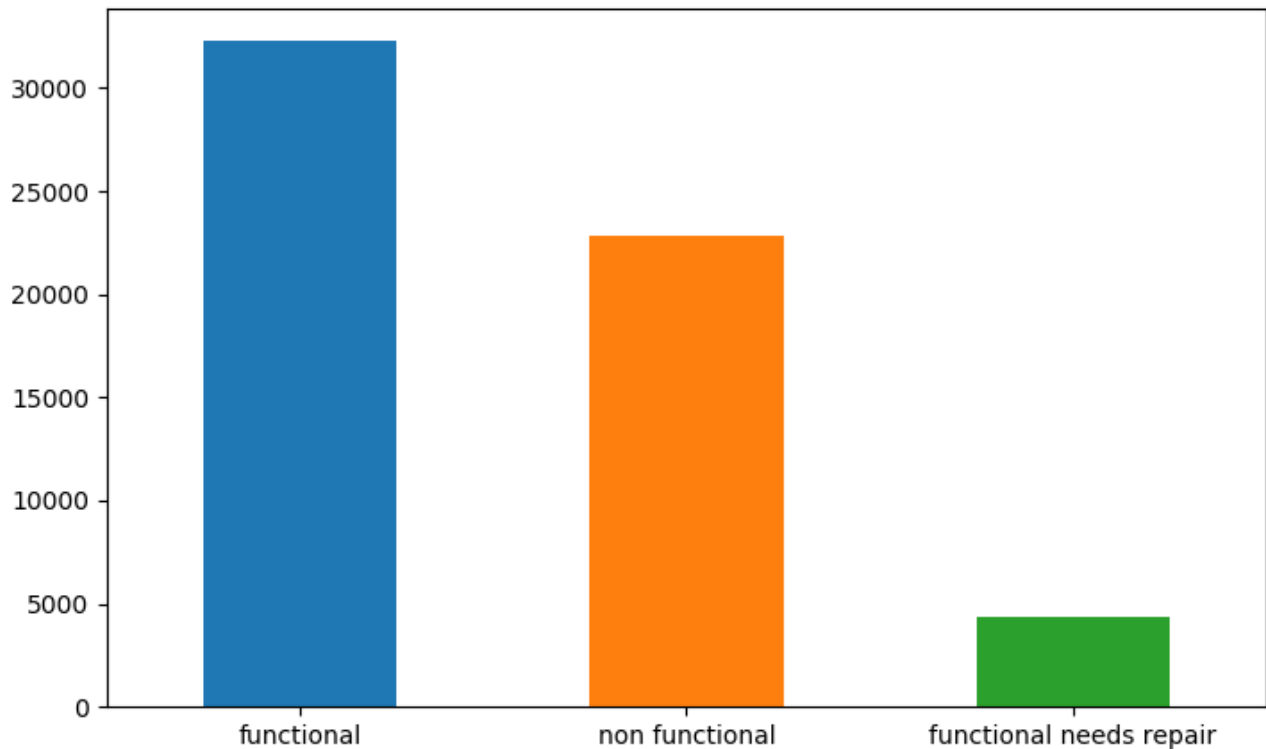


Figura 1: Distribución de las etiquetas

Como podemos ver hay una clara desproporción siendo la mayoría de los casos 'functional', mientras que 'functional needs repair' representa a una minoría. Veamos a que porcentajes se corresponde cada uno.

```
print(data_y.status_group.value_counts()/len(data_y))
```

functional	0.543081
non functional	0.384242
functional needs repair	0.072677

Podemos ver que más de la mitad se corresponde con 'functional', casi un 40 % con 'non functional' y solamente un 7 % con 'functional needs repair'.

2.2. Valores perdidos

Veamos ahora que características poseen valores perdidos.

```
data_x.isnull().sum().plot.bar()
plt.show()
```

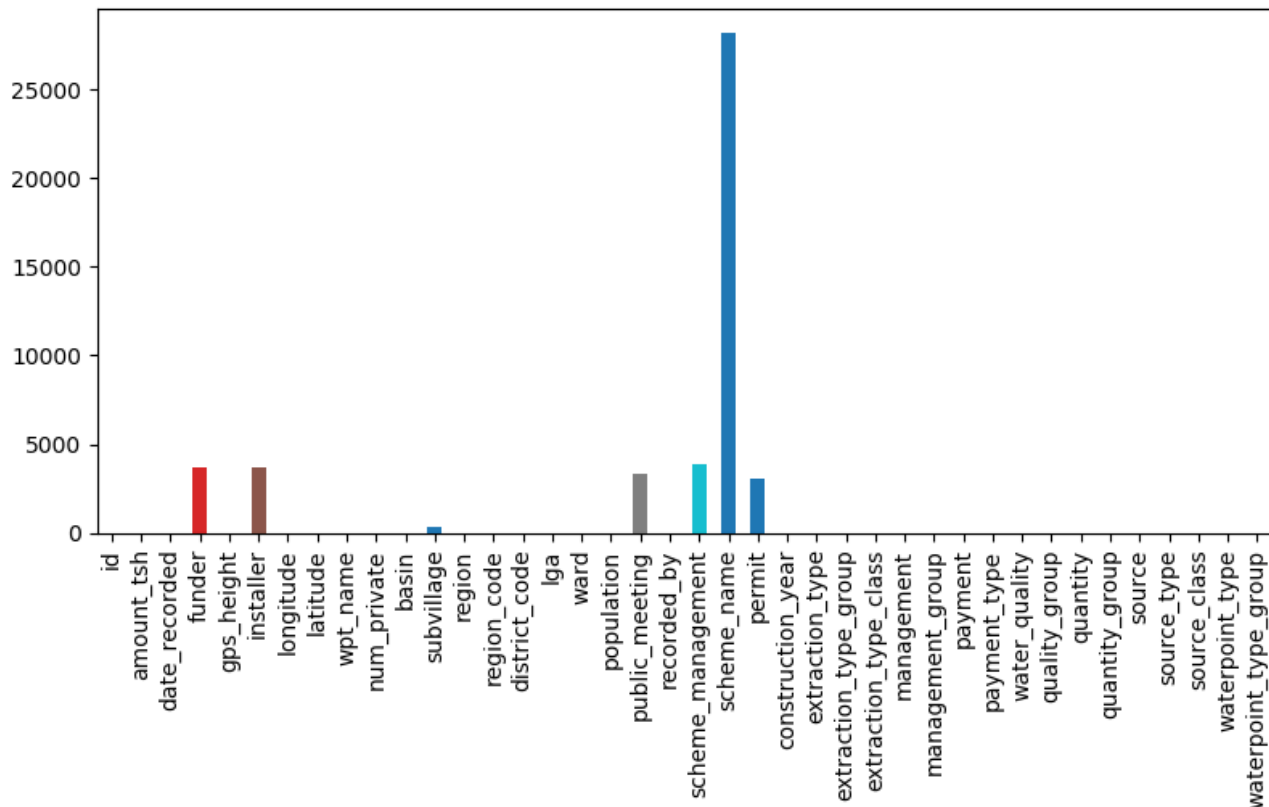


Figura 2: Valores perdidos

Podemos ver que las características que poseen valores perdidos con valor 'nan' son: funder, installer, subvillage, public_meeting, scheme_management, scheme_name, permit y aunque no se aprecia en el gráfico también la wpt_name con solo 20 valores perdidos. Entre estas características, hay una de ellas que destaca por tener demasiados valores perdidos, esta es scheme_name, lo cual sugiere que podría ser buena idea no tenerla en cuenta para la clasificación.

Ahora veamos cuales son los valores más repetidos en las características nombradas anteriormente, (a excepción de scheme_name) para en un futuro tener como referencia estos valores para sustituirlos por los valores nan. Solo se mostrarán los 6 valores más repetidos.

```
print('funder:\n')
print(data_x['funder'].value_counts()[0:6])
print('\ninstaller:\n')
print(data_x['installer'].value_counts()[0:6])
print('\npublic_meeting:\n')
print(data_x['public_meeting'].value_counts()[0:6])
```

```

print( '\nscheme_management:\n' )
print( data_x[ 'scheme_management' ].value_counts() [0:6] )
print( '\npermit:\n' )
print( data_x[ 'permit' ].value_counts() [0:6] )
print( '\nsubvillage:\n' )
print( data_x[ 'subvillage' ].value_counts() [0:6] )
print( '\nwpt_name:\n' )
print( data_x[ 'wpt_name' ].value_counts() [0:6] )

```

La salida de este código es la siguiente:

```

funder :
Government Of Tanzania    9084
Danida                    3114
Hesawa                    2202
Rwssp                     1374
World Bank                1349
Kkkt                      1287

```

```

installer :
DWE          17402
Government   1825
RWE          1206
Commu        1060
DANIDA       1050
KKKT         898

```

```

public_meeting :
True          51011
False         5055

```

```

scheme_management :
VWC          36793
WUG           5206
Water authority  3153
WUA           2883
Water Board    2748
Parastatal     1680

```

```

permit :
True          38852
False         17492

```

```

subvillage :
Shuleni       506
Majengo       502
Kati          373
Mtakuja       262
Sokoni        232

```

```

wpt_name :
none          3563
Shuleni       1748
Zahanati      830
Msikitini     535
Kanisani      323

```


Podemos ver que los valores más repetidos para cada una de estas características son:

- **funder:** Government Of Tanzania
- **installer:** DWE
- **public_meeting:** True
- **scheme_management:** VWC
- **permit:** True
- **subvillage:** Shuleni
- **wpt_name:** none

Además de estos valores perdidos, podemos ver que hay más, solo que están algo camuflados. Respecto al año de construcción si miramos a los datos podemos ver que hay muchos en los que el valor es 0, lo cual no tiene mucho sentido, y que no se corresponde con un año válido. Veamos cuantas de los ejemplos del conjunto de entrenamiento tiene como año de construcción el valor 0.

```
print(len(data_x.ix[data_x['construction_year']==0,'construction_year']))  
20709
```

Como podemos ver hay más de 20000 ejemplos con valor 0 en esta característica.

A las características de longitud y latitud les pasa lo mismo y además siempre les ocurre a la vez, cuando la longitud tiene el valor 0.0, en ese mismo ejemplo podemos comprobar que la latitud tiene el valor -0,00000002. Veamos cuantos ejemplos hay con este problema.

```
print(len(data_x.ix[data_x['longitude']==0,'longitude']))  
1812
```

No son tantos ejemplos como en el año de construcción, aún así el número supera los 1800, y puede ser un problema ya que la latitud y la longitud pueden ser características muy buenas a la hora de realizar las predicciones.

2.3. Correlación entre variables numéricas

Veamos ahora la correlación existente entre las variables numéricas:

```
corr = data_x.corr()  
sns.heatmap(corr)  
plt.xticks(rotation=45)  
plt.show()
```

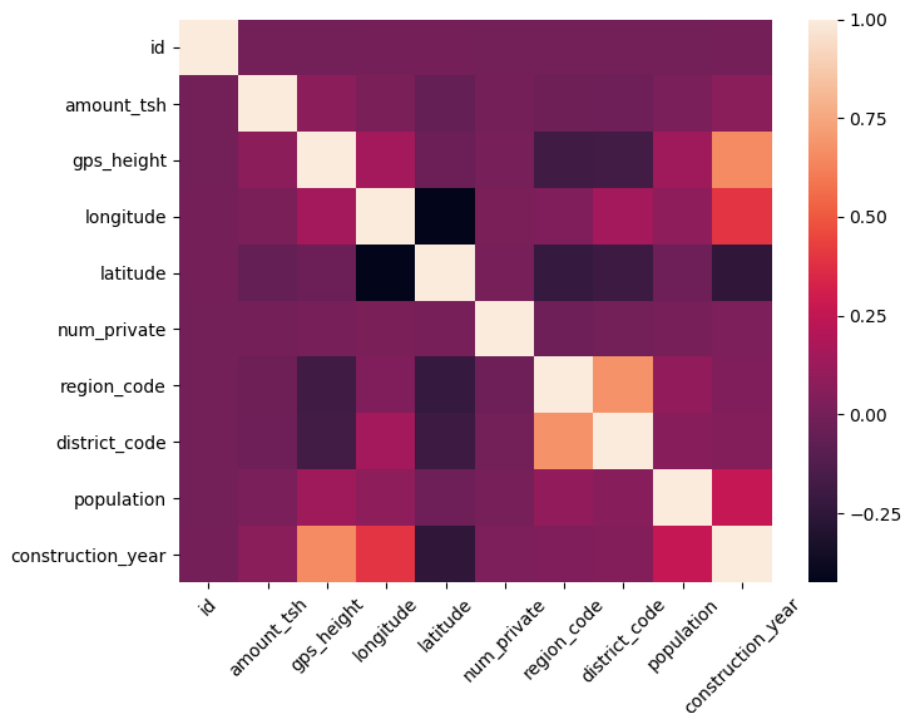


Figura 3: Correlación

2.4. Observaciones adicionales

Otra cosa que podemos observar si miramos la tabla de datos, es que hay dos columnas que parece que siempre tienen el mismo valor, estas son 'num_private' y 'recorder_by'. Veamos que valores tiene estas dos características.

```
print('num_private:\n')
print(data_x['num_private'].value_counts())

num_private:
0          58643
6           81
1           73
5           46
8           46
32          40
...

print('recorded_by:\n')
print(data_x['recorded_by'].value_counts())

recorded_by:
GeoData Consultants Ltd    59400
```

Podemos comprobar que el único valor que presenta a característica 'recorder_by' es 'GeoData Consultants Ltd' y respecto a la característica 'num_private' prácticamente todos tiene como

valor el 0. Esto nos indica que estas dos columnas no tienen ningún poder predictivo, ya que siempre presentan el mismo valor.

3. Preprocesamiento

Después de haber visto los datos, podemos ver que preprocesamiento puede ser más adecuado para este problema.

3.1. Eliminación de columnas

En primer lugar vamos a eliminar aquellas columnas que no vamos a utilizar para la predicción.

```
columns_to_drop = ['id', 'num_private', 'recorded_by', 'scheme_name']
data_x.drop(labels=columns_to_drop, axis=1, inplace = True)
```

La columna 'id' es eliminada ya que esta se corresponde con el identificador del pozo y no aporta nada a la hora de la clasificación. 'num_private' y 'recorder_by' son eliminadas debido a que siempre tienen el mismo valor. 'scheme_name' es eliminada debido a que como vimos en el apartado anterior la mayoría de las filas tienen valores perdidos para esta característica.

3.2. Tratamiento de valores perdidos

Comenzamos sustituyendo los valores perdidos por los valores más repetidos en las columnas correspondientes. Dichos valores más repetidos fueron vistos en el apartado 2.2.

```
print(" Modificando valores nan...")
data_x['funder'] = data_x['funder'].fillna('Government Of Tanzania')
data_x['installer'] = data_x['installer'].fillna('DWE')
data_x['public_meeting'] = data_x['public_meeting'].fillna(True)
data_x['scheme_management'] = data_x['scheme_management'].fillna('VWC')
data_x['permit'] = data_x['permit'].fillna(True)
data_x['subvillage'] = data_x['subvillage'].fillna('Unknown')
data_x['wpt_name'] = data_x['wpt_name'].fillna('none')
```

Vimos también que encontramos valores perdidos en el año de construcción, donde había filas con el valor 0 para esta característica. Sustituiremos el valor 0, por un valor con sentido para esta característica.

```
data_x.construction_year=pd.to_numeric(data_x.construction_year)
data_x.loc[data_x.construction_year <= 0, data_x.columns=='construction_year'] =
1950
```

Vimos también valores extraños en la latitud y en la longitud. Estos valores se correspondían con la tupla (latitud, longitud) = (-0.00000002, 0.0), así que convertimos dichos valores en valores perdidos y posteriormente completamos dichos valores perdidos. Para rellenar estos valores perdidos agrupamos los ejemplos por el código de la región y después modificamos cada valor perdido por media de la longitud y latitud de dichos grupos. Así los valores erróneos de longitud

y latitud serán intercambiados por valores que son cercanos las verdaderas coordenadas, ya que se han calculado con la media de aquellos ejemplos pertenecientes a la misma región.

```
data_x.ix[data_x['latitude']==-0.00000002,'latitude']=None
data_x.ix[data_x['longitude']==0,'longitude']=None

data_x["longitude"] = data_x.groupby("region_code").transform(lambda x: x.fillna(
    (x.median()))).longitude
data_x["latitude"] = data_x.groupby("region_code").transform(lambda x: x.fillna(
    x.median()))).latitude
```

3.3. Valores raros

En las variables categóricas podemos encontrar características en las que hay ciertos valores que son muy poco frecuentes, los cuales serán considerados como raros, dichos valores serán modificados por el valor 'Others'. Para ello en cada columna, en la que queramos añadir estos cambios, contaremos cuantas veces se repite cada valor. Si un valor se repite menos de 20 veces será considerado como raro y se modificará su valor.

A continuación se muestra el código para conseguir lo que acabamos de explicar:

```
columns_other = [x for x in data_x.columns if x not in ['latitude', 'longitude', '
    gps_height', 'age', 'population', 'construction_year', 'month_recorder']]

for col in columns_other:
    value_counts = data_x[col].value_counts()
    lessthen = value_counts[value_counts < 20]
    listnow = data_x.installer.isin(list(lessthen.keys()))
    data_x.loc[listnow, col] = 'Others'
```

3.4. Coordenadas cartesianas

Para ampliar la información espacial, vamos a pasar la latitud y longitud a coordenadas cartesianas y además vamos a incluir también la distancia de cada pozo al punto (0, 0). Para conseguirlo vamos a ayudarnos de las siguiente funciones:

```
def distancia(lon1, lat1, lon2, lat2):
    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    R = 6371

    return R * c

def cartesian_x(lon, lat):
    lat=radians(lat)
    lon=radians(lon)
    R=6371.0
```

```

x = R * cos(lat) * cos(lon)
return x

def cartesian_y(lon, lat):
    lat=radians(lat)
    lon=radians(lon)
    R=6371.0
    y = R * cos(lat) * sin(lon)
    return y

```

Ahora utilizamos las funciones anteriores para añadir las características 'cartesian_x', 'cartesian_y' y 'dist'.

```

print(" Preprocesando coordenadas y distancias...")
data_x['dist'] = data_x.apply(lambda row: distancia(row['longitude'], row['latitude'], 0, 0), axis=1)
data_x['cartesian_x'] = data_x.apply(lambda row: cartesian_x(row['longitude'], row['latitude']), axis=1)
data_x['cartesian_y'] = data_x.apply(lambda row: cartesian_y(row['longitude'], row['latitude']), axis=1)

```

Veamos la representación de las coordenadas cartesianas 'x' e 'y' coloreados según la etiqueta.

```

y = np.ravel(data_y.values)
y = le.fit(y).transform(y)
plt.scatter(data_x['longitude'].values, data_x['latitude'].values, c=y)
plt.show()

```

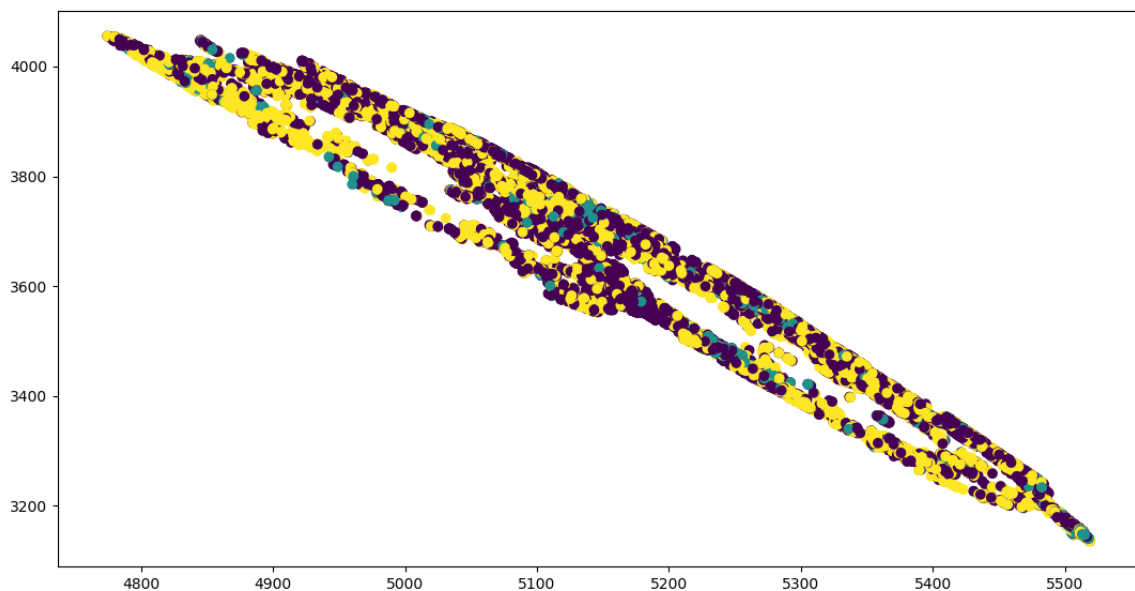


Figura 4: Coordenadas cartesianas

3.5. Fechas

Obtenemos características adicionales a partir de 'date_recorded' y 'construction_year'. De 'date_recorded' obtenemos el año, el mes, el día de la semana y la semana del año. Después calculando la diferencia entre 'year_recorded' y 'construction_year' obtenemos la edad del pozo.

```
def date_parser(df):
    date_recorder = list(map(lambda x: datetime.datetime.strptime(str(x), '%Y-%m-%d'),
                             df['date_recorded'].values))
    df['year_recorder'] = list(map(lambda x: int(x.strftime('%Y')),
                                   date_recorder))
    df['weekday_recorder'] = list(map(lambda x: int(x.strftime('%w')),
                                      date_recorder))
    df['yearly_week_recorder'] = list(map(lambda x: int(x.strftime('%W')),
                                           date_recorder))
    df['month_recorder'] = list(map(lambda x: int(x.strftime('%m')),
                                    date_recorder))
    df['age'] = df['year_recorder'].values - df['construction_year'].values
    del df['date_recorded']

    return df

print(" Preprocesando fechas...")
data_x = date_parser(data_x)
```

3.6. Categóricas a numéricas

Por último pasamos las variables categóricas en numéricas, para ello utilizamos LabelEncoder(), con el cual no obtenemos problemas ya que ya no hay valores nan.

```
print(" Convirtiendo categóricas a numéricas...")
data_x = data_x.astype(str).apply(LabelEncoder().fit_transform)
```

4. Validación cruzada

Para realizar la validación cruzada he utilizado la siguiente función, a la cual se le pasa como parámetros el modelo del clasificador a utilizar, el conjunto de datos, las etiquetas asociadas a los datos y el modo de validación cruzada, si no se indica ninguna validación cruzada se usará una validación cruzada estratificada con k=5. Además hay parámetros para indicar si se quiere realizar algún preprocesamiento de los datos como puede ser un escalado, normalización. También puede indicarse si se quieren añadir características polinómicas.

```
def cross_validation(clf, X, y, cv = None, min_max_scaler = False, scaled =
    False, standard_scaler = False, normalizer = False, poly = False, m_confusion
    = False):

    if cv == None:
        cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=123456)
```

```

iteration = 0

for train, test in cv.split(X, y):

    X_train, X_test = X[train], X[test]
    y_train, y_test = y[train], y[test]

    if min_max_scaler:
        X_train = MinMaxScaler().fit_transform(X_train)
        X_test = MinMaxScaler().fit_transform(X_test)

    if scaled:
        X_train = scale(X_train)
        X_test = scale(X_test)

    if poly:
        X_train = PolynomialFeatures(degree = 2, interaction_only=True).
            fit_transform(X_train)
        X_test = PolynomialFeatures(degree = 2, interaction_only=True).
            fit_transform(X_test)

    if standard_scaler:
        transformer = StandardScaler().fit(X_train)
        X_train = transformer.transform(X_train)
        X_test = transformer.transform(X_test)

    if normalizer:
        transformer = Normalizer().fit(X_train)
        X_train = transformer.transform(X_train)
        X_test = transformer.transform(X_test)

    t = time.time()
    clf = clf.fit(X_train, y_train)
    training_time = time.time() - t

    predictions_train = clf.predict(X_train)
    predictions = clf.predict(X_test)

    print("———— Iteración ", iteration, " ———— ")
    print("Tiempo :: ", training_time)
    print("Train Accuracy :: ", accuracy_score(y_train, predictions_train))
    print("Test Accuracy :: ", accuracy_score(y_test, predictions))
    print("")

    if m_confusion:
        plot_confusion_matrix(y_test, predictions)

    iteration += 1

```

Podemos apreciar que hay otro parámetro llamado 'm_confusion' en el cual podemos especificar si deseamos ver la representación de la matriz de confusión o no.

Para la representación de la matriz de confusión se ha utilizado la siguiente función.

```
def plot_confusion_matrix(y_test, predictions):
    cm = metrics.confusion_matrix(y_test, predictions)
    plt.figure(figsize=(9,9))
    sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()
```

Veamos un ejemplo de ejecución de esta función para la validación cruzada.

```
clf = RandomForestClassifier(n_estimators=125, max_depth = 20, random_state =
10)
cross_validation(clf, X, y, m_confusion=)
```

Los resultados obtenidos tras realizar la validación cruzada anterior han sido los siguientes:

```
----- Iteración 0 -----
Tiempo :: 19.252601146697998
Train Accuracy :: 0.9586481196994886
Test Accuracy :: 0.8112111775103106

----- Iteración 1 -----
Tiempo :: 18.67781901359558
Train Accuracy :: 0.9590269155495696
Test Accuracy :: 0.8141570574867435

----- Iteración 2 -----
Tiempo :: 18.465768098831177
Train Accuracy :: 0.9583964646464647
Test Accuracy :: 0.8127104377104377

----- Iteración 3 -----
Tiempo :: 18.78512740135193
Train Accuracy :: 0.960563973063973
Test Accuracy :: 0.8138888888888889

----- Iteración 4 -----
Tiempo :: 19.4857075214386
Train Accuracy :: 0.9598922604267497
Test Accuracy :: 0.8110793062805186
```

A continuación vemos la matriz de confusión de la primera iteración de la validación cruzada.

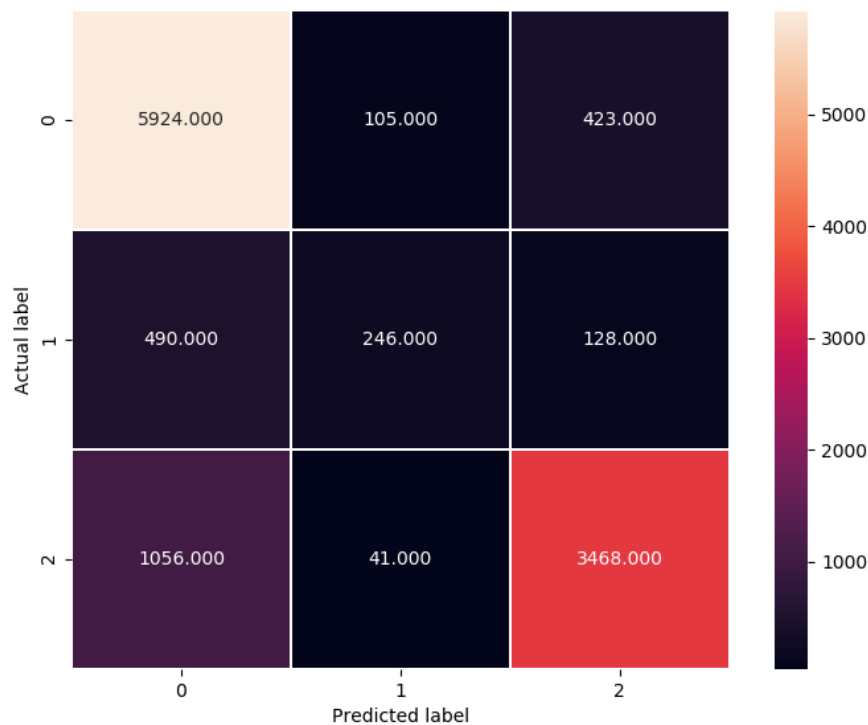


Figura 5: Matriz de confusión

Antes de subir soluciones a DrivenData realicé experimentos con diferentes algoritmos, en el siguiente código podemos ver algunos de los algoritmos que utilicé para la validación cruzada. En el siguiente código podemos ver algunos de los experimentos con diferentes algoritmos que se han llevado a cabo. Los algoritmos probados a continuación se corresponden con KNN, XGB, LGB, RandomForest y ExtraTreesClassifier.

```
print('\nKNN\n')
knn = KNeighborsClassifier(n_neighbors=5)
# Para el KNN es necesario normalizar los datos ya que KNN está basado en
# distancias
cross_validation(clf=knn, X = X, y = y, cv = None, min_max_scaler = True)

print('\nXGB\n')
clf = xgb.XGBClassifier(n_estimators = 200)
cross_validation(clf, X, y)

print('\nLGB\n')
clf = lgb.LGBMClassifier(objective='binary', n_estimators=200, num_leaves=31)
cross_validation(clf, X, y)

print('\nRandomForest\n')
clf = RandomForestClassifier(n_estimators=125, max_depth = 20, random_state =
10)
cross_validation(clf, X, y)

print('\nExtraTreesClassifier\n')
clf = ExtraTreesClassifier(n_estimators = 125, max_depth = 20)
cross_validation(clf, X, y)
```

KNN

Iteración 0

Tiempo :: 3.2514231204986572
Train Accuracy :: 0.824154548706833
Test Accuracy :: 0.7546502819627977

Iteración 1

Tiempo :: 3.5892577171325684
Train Accuracy :: 0.8242176813485133
Test Accuracy :: 0.7495160340038717

Iteración 2

Tiempo :: 3.676161527633667
Train Accuracy :: 0.8223905723905723
Test Accuracy :: 0.7547979797979798

Iteración 3

Tiempo :: 3.156574010848999
Train Accuracy :: 0.8231902356902356
Test Accuracy :: 0.7437710437710437

Iteración 4

Tiempo :: 3.174323797225952
Train Accuracy :: 0.8258280375405076
Test Accuracy :: 0.7470112813604984

XGB

Iteración 0

Tiempo :: 22.008957147598267
Train Accuracy :: 0.7659041646499295
Test Accuracy :: 0.760962881912297

Iteración 1

Tiempo :: 21.464710235595703
Train Accuracy :: 0.7645573349607525
Test Accuracy :: 0.7614678899082569

Iteración 2

Tiempo :: 21.207271099090576
Train Accuracy :: 0.766077441077441
Test Accuracy :: 0.7593434343434343

Iteración 3

Tiempo :: 22.781185626983643
Train Accuracy :: 0.7641203703703704
Test Accuracy :: 0.7571548821548821

Iteración 4

Tiempo :: 23.101938247680664
Train Accuracy :: 0.7651613989310214

Test Accuracy :: 0.7555985856204748

LGB

Iteración 0

Tiempo :: 7.352467775344849
Train Accuracy :: 0.8314568909278394
Test Accuracy :: 0.7958084336335325

Iteración 1

Tiempo :: 7.281744956970215
Train Accuracy :: 0.8305940781582104
Test Accuracy :: 0.7923575456611396

Iteración 2

Tiempo :: 7.10748553276062
Train Accuracy :: 0.830787037037037
Test Accuracy :: 0.7932659932659932

Iteración 3

Tiempo :: 5.973698139190674
Train Accuracy :: 0.8291245791245792
Test Accuracy :: 0.7962121212121213

Iteración 4

Tiempo :: 6.158836364746094
Train Accuracy :: 0.8319304743066369
Test Accuracy :: 0.7942414547903688

RandomForest

Iteración 0

Tiempo :: 18.89332938194275
Train Accuracy :: 0.9586481196994886
Test Accuracy :: 0.8112111775103106

Iteración 1

Tiempo :: 18.439769983291626
Train Accuracy :: 0.9590269155495696
Test Accuracy :: 0.8141570574867435

Iteración 2

Tiempo :: 18.55832815170288
Train Accuracy :: 0.9583964646464647
Test Accuracy :: 0.8127104377104377

Iteración 3

Tiempo :: 18.501311540603638
Train Accuracy :: 0.960563973063973
Test Accuracy :: 0.8138888888888889

Iteración 4

```
Tiempo :: 18.28575587272644
Train Accuracy :: 0.9598922604267497
Test Accuracy :: 0.8110793062805186
```

ExtraTreesClassifier

```
----- Iteración 0 -----
Tiempo :: 10.710731029510498
Train Accuracy :: 0.9533870662261411
Test Accuracy :: 0.808433633532531
```

```
----- Iteración 1 -----
Tiempo :: 10.65065312385559
Train Accuracy :: 0.9537237736484354
Test Accuracy :: 0.809864489521084
```

```
----- Iteración 2 -----
Tiempo :: 10.722788095474243
Train Accuracy :: 0.9528198653198653
Test Accuracy :: 0.8083333333333333
```

```
----- Iteración 3 -----
Tiempo :: 10.63897705078125
Train Accuracy :: 0.9554713804713805
Test Accuracy :: 0.812037037037037
```

```
----- Iteración 4 -----
Tiempo :: 10.686703443527222
Train Accuracy :: 0.9538529523168217
Test Accuracy :: 0.8090587641017006
```

Podemos ver que Random forest ha sido el que ha dado mejores resultados y debido a ello es el que he decido usar para la competición.

5. Progreso seguido

N	Fecha y hora	Posición	Train	DrivenData	Preprocesado	Algoritmos	Parámetros
1	2018-12-28 21:35	< 1500	0.7640	0.7183	Convertir categóricas a numéricas	XGB	Por defecto
2	2018-12-28 22:06	1335	0.9514	0.7720	Anterior + Rellenar Nan con la mediana	Random Forest	n_estimators=125
3	2018-12-28 23:07	1335	0.9849	0.7678	Anterior	Random Forest	Por defecto
4	2018-12-29 17:43	1335	1.0	0.7644	Anterior	RandomForest	n_estimators=1000
5	2018-12-29 20:56	1330	0.9501	0.7728	Anterior	Random Forest	n_estimators=125, max_depth = 20
6	2018-12-30 01:48	1330	1.0	0.7714	Anterior + Borrar 'num_private', 'recorded_by', 'scheme_name'	RandomForest	max_features = 'sqrt', n_estimators = 500
7	2018-12-30 22:07	1330	1.0	0.7705	Anterior	Random Forest	max_features = 'sqrt', n_estimators = 200
8	2018-12-30 22:19	1330	0.9489	0.7711	Anterior	Random Forest	max_features = 'sqrt', n_estimators = 200, max_depth = 20
9	2018-12-31 13:36	1330	0.9856	0.7642	Anterior	Random Forest	Por defecto
10	2018-12-31 14:36	1330	0.9483	0.7719	Anterior	Random Forest	n_estimators=125, max_depth = 20
11	2018-12-31 19:59 0.9537	290	0.9537	0.8215	Modificar los 0's de longitud y latitud por la media de sus regiones, marcar como 'others' valores con menos de 20 ocurrencias, obtener la edad del pozo y extraer de 'date_recorded', el año, el mes y día de la semana, obtener la distancia euclídea del pozo al punto (0,0)	RandomForest	n_estimators=200, max_depth = 20
12	2019-01-01 00:03	250	0.9530	0.8222	Anterior	RandomForest	n_estimators=125, max_depth = 20
13	2019-01-01 00:11	170	0.9398	0.8232	Anterior solo que ahora en vez de rellenar Nan con mediana, relleno con los valores más repetidos y modifíco el valor de las filas con año de construcción igual a 0 por 1950	RandomForest	n_estimators=10, max_depth = 20
14	2019-01-01 00:28	170	0.9512	0.8228	Anterior	RandomForest	n_estimators=125, max_depth = 20
15	2019-01-02 13:09	170	0.9528	0.8224	Anterior	RandomForest	n_estimators=125, max_depth = 22

N	Fecha y hora	Posición	Train	DrivenData	Preprocesado	Algoritmos	Parámetros
16	2019-01-02 13:14	170	0.9514	0.8196	Anterior	RandomForest	n_estimators=500, max_depth = None
17	2019-01-02 13:19	170	0.9518	0.8208	Anterior	RandomForest	n_estimators=100, max_depth = 20
18	2019-01-03 00:16	170	0.9563	0.8222	Anterior	RandomForest	n_estimators=160, max_depth = 20
19	2019-01-03 00:28	170	0.9556	0.8215	Anterior	RandomForest	n_estimators=150, max_depth = 20
20	2019-01-03 11:58	170	—	0.7669	Subido por error	Subido por error	Subido por error
21	2019-01-04 11:42	170	0.9543	0.8213	Anterior (19)	RandomForest	n_estimators=140, max_depth = 20
22	2019-01-04 11:54	170	0.9560	0.8226	Anterior	RandomForest	n_estimators=175, max_depth = 20
23	2019-01-04 15:09	170	0.9448	0.8215	Anterior	RandomForest	n_estimators=10, max_depth = 25
24	2019-01-05 00:01	115	0.9537	0.8242	Anterior + Pasar longitud y latitud a coordenadas cartesianas	RandomForest	n_estimators=100, max_depth = 20
25	2019-01-05 00:14	115	0.9535	0.8239	Anterior	RandomForest	n_estimators=150, max_depth = 20
26	2019-01-05 00:49	98	0.9538	0.8246	Anterior	RandomForest	n_estimators=125, max_depth = 20

El preprocesamiento final utilizado para las últimas submissions es el explicado en el apartado 3 sobre preprocesamiento.

6. Bibliografía

- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- https://scikit-learn.org/stable/model_selection.html#model-selection
- <https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing>
- Transparencias de clase.