Inteligencia de Negocio
Práctica 3

Guillermo Gómez Trenado | 77820354-S
guillermogotre@correo.ugr.es

10 de enero de 2019

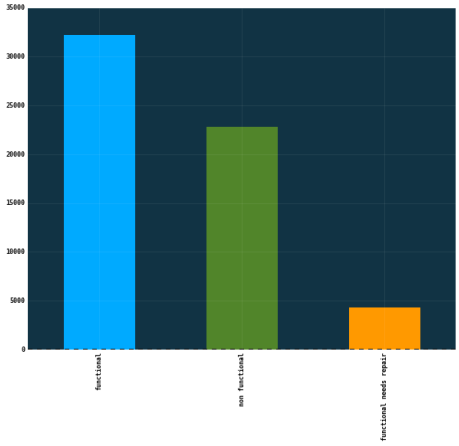
Índice

1. Descripción del problema

El problema que se nos plantea es el etiquetamiento del estado de pozos de agua en Tanzania a partir de diferentes características que se nos presentan, binarias, numéricas y categóricas. Tenemos un conjunto de entrenamiento con sus etiquetas correspondientes —sin separación entre train y validación— y un conjunto de test que se evalúa en la página web, para no hacer sobreajuste con los datos de test sólo se permiten tres subidas al día. La métrica utilizada en la página web es la porción de instancias correctamente etiquetadas.

2. Análisis de datos

Lo primero que hice fue leer la decripción del problema[1]. Algunas características están mejor comentadas que otras, pero aun así nos permiten hacernos una idea general del problema, es un problema desbalanceado, hay tres etiquetas desigualmente distribuidas



Después de ver la descripción busqué información sobre el problema en google para un acercamiento preliminar a su análisis, ya que no era la primera vez que se realiza el concurso, y por lo tanto hay muy buenos análisis en internet, especialmente los citados abajo. [2] [3] [4] [5]

[1] DrivenData. Pump it Up: Data Mining the Water Table. Problem Description. `https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/25/`

[2] Pump it Up: Data Mining the Water Table. Análisis `https://rstudio-pubs-static.s3.amazonaws.com/187095_fa3184e3f1244ba9b98f351570699732.html`

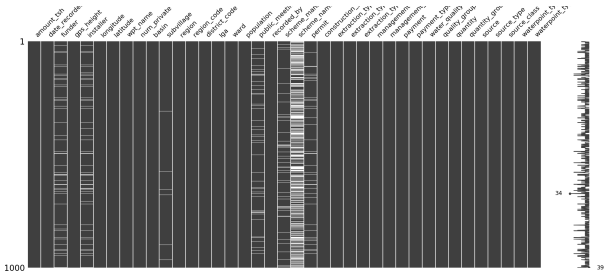
[3] Vaibhav Shukla. Medimum. Pump it Up: Data Mining the Water Table `https://medium.com/@vaibhavshukla182/pump-it-up-data-mining-the-water-table-f903d4cfc7a8`

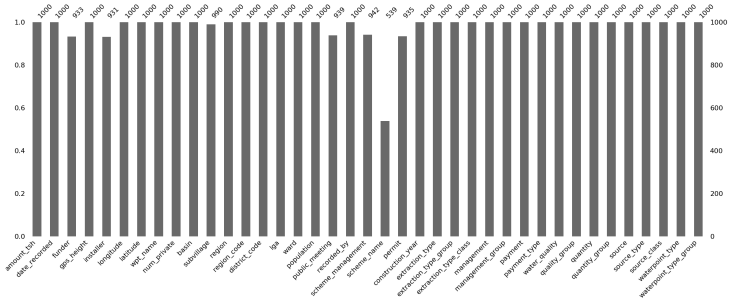
[4] David Currie. Pump it Up: Data Mining the Water Table. `https://currie32.github.io/water_pumps.html`

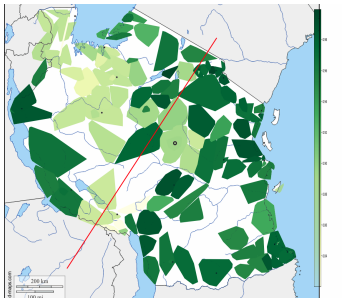
[5] Calin Uioreanu. Community Driven Data. `https://community.drivendata.org/t/pumps-visualization-dashboard-with-tableau/186`. Tableau. `https://public.tableau.com/profile/calin.uioreanu#!/vizhome/DataMiningtheWaterTableDrivenData_com/Bubblestatusquantity`

2.1. Valores nulos

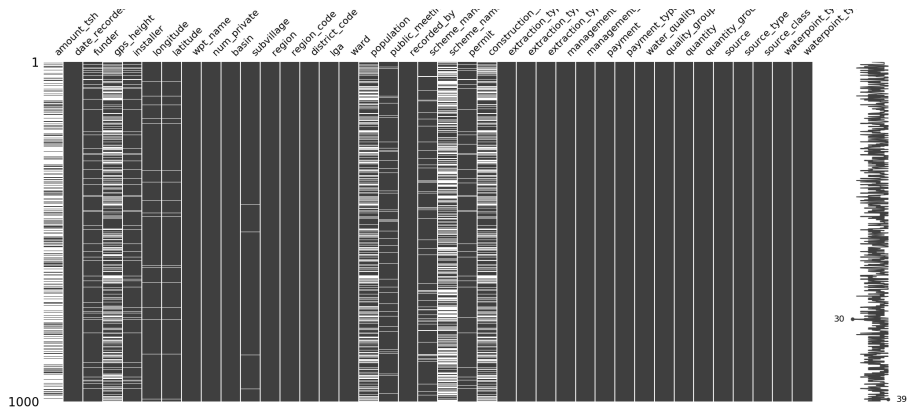
Tras este primer acercamiento al problema, el primer asunto que quiero tratar es la presencia de valores nulos. Para las variables categóricas es inmediato porque vienen etiquetadas con *NaN*. Veamos tres representaciones de la proporción de valores nulos, el primero una representación de una muestra de mil instancias, la segunda imagen un histograma, y en tercer lugar la distribución por código de región —vemos cómo no sólo cómo las variables están irregularmente rellenas entre ellas sino con distinta distribución en distintas zonas del mapa.



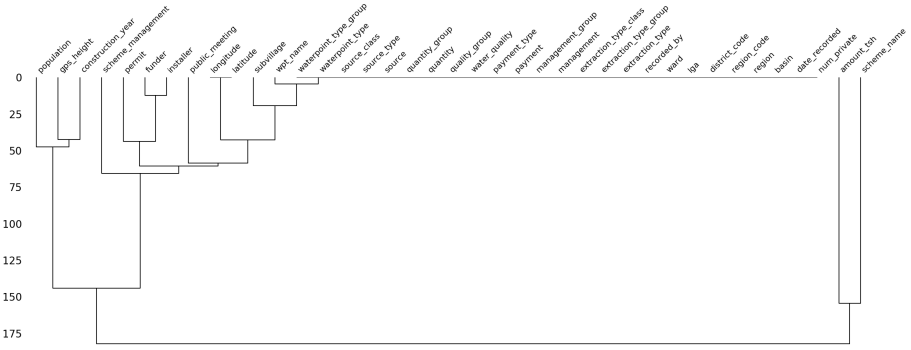




Sin embargo, para las numéricas no queda claro cuando el cero significa cero y cuándo es una característica no cumplimentada. A la luz de la descripción de la página web, las tres variables a 0 que con seguridad suponen valores nulos son *amount_tsh* —cantidad de agua disponible al pozo—, *longitud* y *latitud* —la coordenada 0,0 está fuera de tanzania— y *gps_height* —altura 0—.



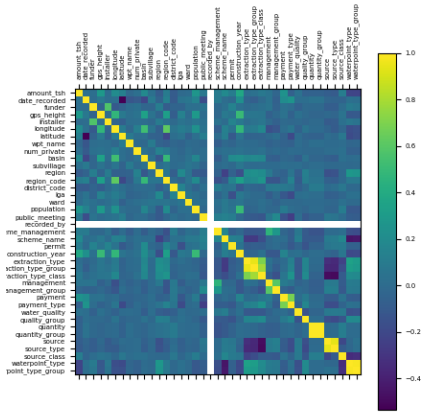
Lo primero que observo es que las variables con mayor porcentaje de valores nulos son las relativas a las coordenadas y altura geográfica, la cantidad de agua disponible, la cantidad de población que utiliza el pozo, el año de construcción y quién opera el pozo. Sobre las geográficas tendremos oportunidad de centrarnos con mayor detenimiento en el preprocesado.



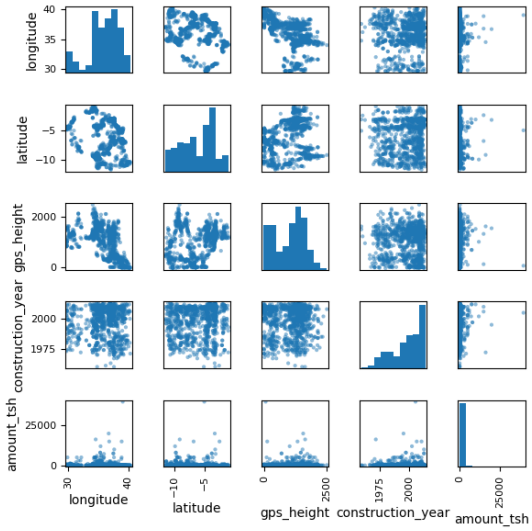
Vemos como las variables agrupadas por su tasa de valores nulos estás por lo general semánticamente relacionadas, esta intuición nos será útil para la imputación.

2.2. Correlación

Ahora me interesa ver la naturaleza de las características y su relación, aplico *LabelEncoder* para poder hacer una matriz de correlación y la imprimo



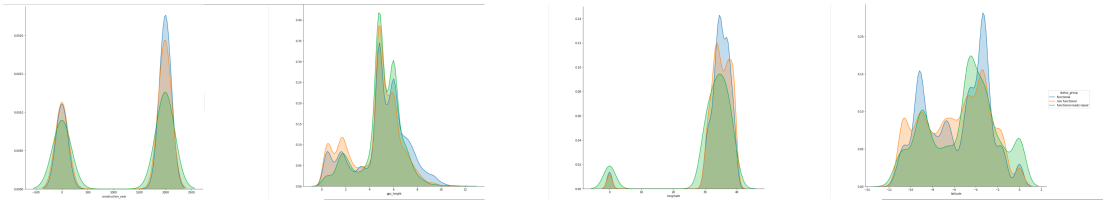
Lo que analizamos es que hay variables altamente correlacionadas, al analizarlas en detalle observo que explican el mismo fenómeno, como sucede con *extraction_type*, *extraction_type_group* y *extraction_type_class*, o aquellas que diferencia entre *type* y *class* o entre *quantity* y *quantity_group*, este tipo de variables serán las primeras que eliminemos en el preprocesado.



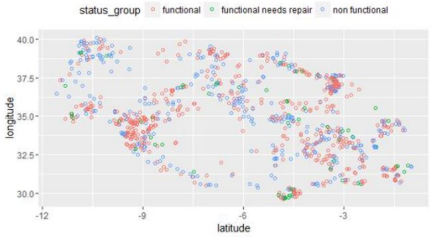
Arriba ĺas gráficas de correlación e histogramas de las variables geográficas — después de eliminar los ceros— porque serán sobre las que centre principalemten el preprocesado por ser altamente inteligibles y fáciles de imputar con relativa confianza.

2.3. Relación con la etiqueta de la salida

Vaibhav Shukla en Medium [6] hace un análisis interesantísimo en su descripción del problema de las distribuciones superpuestas según la clase, adjunto algunas como referencia



Analizo las cuatro variables numéricas que veo más fácil imputar, vemos que el año de construcción no parece tener una capacidad predictiva relevante, pero la longitud, latitud y altura parecen ser unos predictores de relativa calidad como podemos confirmar en la siguiente gráfica de Jithin Paul[7]. Por eso nos centraremos en estos valores para la imputación.



2.4. Métrica y balanceo

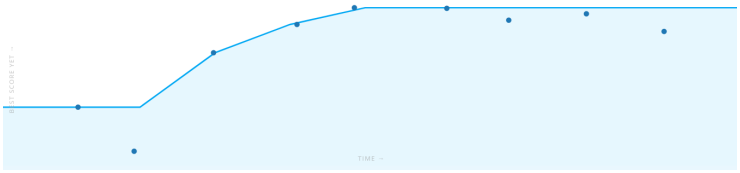
La métrica con la que se nos evalua en DrivenData es la tasa de acierto, que como ya hemos visto no es la métrica más robusta ni más descriptiva de la calidad del clasificador, el clasificador constante arroja una tasa de acierto de $\sim 0{,}56$ —con la salida constante *functional*— aun siendo un problema con tres clases, debido al desbalanceo. Sin embargo, debido a la métrica usada no voy a centrar mis esfuerzos en balancear las clases, pues si bien pudiera esto generar un clasificador más robusto y que sirviera mejor a los propósitos últimos de esta competición, lo más posible es que penalizara el rendimiento en DrivenData pues ellos no atajan en su métrica el problema de la falta de balanceo, de hecho si hicieramos un clasificador binario, ignorando la clase *functional need repair* podríamos optar a clasificar correctamtene el 93 % de la muestra.

[6] Vaibhav Shukla. Medimum. Pump it Up: Data Mining the Water Table `https://medium.com/@vaibhavshukla182/pump-it-up-data-mining-the-water-table-f903d4cfc7a8`

[7] Jithin Paul. PUMP IT UP – Data Mining the Water Table. `https://jitpaul.blog/2017/07/12/pump-it-up/`

3. Resultados obtenidos

Descripcion	CV 5fold	All train	DrivenData	DD Posicion	Preprocesado	Algoritmo	Parámetros
Base	0.7954	0.825			*LE*	*LGBM*	Estimators=200
Fechas	0.7966	0.8262			*LE+Date*	*LGBM*	Estimators=200
Geodist	0.7971	0.8262			*LE+Date+GeoDist*	*LGBM*	Estimators=200
KNN GPS	0.7969	0.8267	0.8034	833	*LE+Date+GeoDist+GPS*	*LGBM*	Estimators=1000
W2V	0.7963	0.8364	0.7937	833	*W2V+Date+GeoDist+GPS*	*LGBM*	Estimators=1000
Height KNN	0.7978	0.826	0.8155	641	*LE+Date+GeoDist+GPS+Height*	*LGBM*	Estimators=1000
Ordenadas	0.8099	0.8999	0.8218	295	*Sorted LE+Date+GeoDist+GPS+Height*	*LGBM*	Estimators=1000
Dual	0.8146	0.9059			*Sorted LE+Date+GeoDist+GPS+Height*	*LGBM+RF*	Es=1000+Es=500,Maxdepth=20
Triple	0.816	0.9503	0.8255	57	*Sorted LE+Date+GeoDist+GPS+Height*	*XGBOOST+LGBM+RF*	Es=500, NClass=4, MaxDepth=14 | Es=1000 | Es=500,Maxdepth=20
Genetico Hiperparámetros	0.8165	0.9411	0.8253	57	*Sorted LE+Date+GeoDist+GPS+Height*	*XGBOOST+LGBM+RF*	*Descripción en el apartado respectivo*
Ruido	0.8191	0.9152	0.8241	57	*Sorted LE+Date+GeoDist+GPS+Height+Noise*	*XGBOOST+LGBM+RF*	
No overfit	0.8172	0.8841	0.8202	57	*Sorted LE+Date+GeoDist+GPS+Height*	*XGBOOST+LGBM+RF*	
*Differential Evolution Pesos**	*0.8179*	*0.9312*	*0.8258*	*48*	*Sorted LE+Date+GeoDist+GPS+Height*	*XGBOOST+LGBM+RF*	



Submissions

BEST	CURRENT RANK	# COMPETITORS	SUBS. TODAY
0.8255	58	6184	0 / 3

SUBMISSIONS

Score	Submitted by	Timestamp
0.8034	Guillermo_Gomez_UGR	2019-01-02 21:12:52 UTC
0.7937	Guillermo_Gomez_UGR	2019-01-02 22:13:48 UTC
0.8155	Guillermo_Gomez_UGR	2019-01-03 22:30:24 UTC
0.8218	Guillermo_Gomez_UGR	2019-01-03 23:48:25 UTC
0.8255	Guillermo_Gomez_UGR	2019-01-04 00:39:57 UTC
0.8253	Guillermo_Gomez_UGR	2019-01-04 21:30:45 UTC
0.8227	Guillermo_Gomez_UGR	2019-01-05 03:03:18 UTC
0.8241	Guillermo_Gomez_UGR	2019-01-05 15:51:18 UTC
0.8202	Guillermo_Gomez_UGR	2019-01-05 22:43:52 UTC

Al final del plazo para participar en la competición, el resultado obtenido en DrivenData fue de 82.55, estando en la posición 57. Sin embargo, un día después, fuera de plazo, con otro usuario que creé al finalizar éste y con una estrategia que seguía desarrollando la línea de las anteriores conseguí colocarme en el puesto 48, desplazándome a mí mismo, en la cuenta creada *ex profeso* para la UGR hasta el 58, adjunto también la información relativa a esta última subida como extra.

Además de la documentación adjunto una carpeta con el código y los csv con el mismo título que en la tabla de arriba. Como el proceso que seguí fue progresivo, primero centrándome en el preprocesado y posteriormente en los clasificadores, vamos a seguir esta misma sucesión en la descripción.

Por otro lado, no tengo el resultado en DrivenData para las primeras modificaciones porque como le comenté por correo no recibí el correo de confirmación hasta 3 días después de que comenzara el desarrollo de la práctica, y como el número de subidas estaba limitado decidí invertirlo en mis progresos.

4. Preprocesado

Mi estrategia se centró en ir transformando todas las variables que tuvieran un significado fuertemente ordinal —categóricas y evidentemente numéricas— en variables numéricas con transformaciones personalizadas a sus características y por último atajar el problema de las categóricas puras. Tanto el preprocesado como la imputación, por no basarse en la etiqueta de salida y debido al enorme tiempo que consumen algunas lo he realizado fuera de la validación cruzda, como tenemos una tercera etapa de test, aunque sea incorrecto, podemos comprobar la calidad última de estas trasnformaciones.

4.1. Código original (Base)

La primera y única modificación que realicé sobre el código que se nos facilitó fue modificar *LabelEncoder* ya que de la forma en la que estaba definido, aunque gracias al orden alfabético estarían próximas las variables entre ellas, no teníamos garantía de que una instancia con la misma etiqueta en train y test tuvieran el mísmo valor numérico tras el preprocesado. Sencillamente uní las dos tablas, apliqué LabelEncoder sobre las variables y las volví a separar, como el orden es alfabético sin observar la etiqueta de salida, no hay por qué entrenar la operación de transformación sobre el train y aplicarla sobre el test —reservando una etiqueta para las variables desconocidas— porque no se puedo producir *data leaking*.

4.2. Transformaciones

4.2.1. Fechas

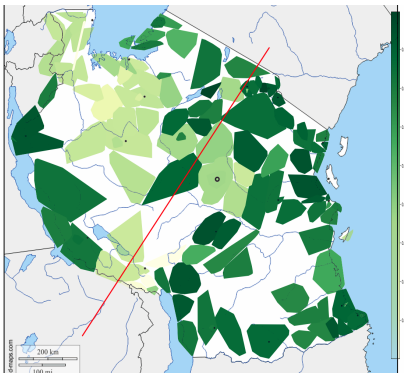
La primera variable que modifiqué fue la fecha en que se registró la instancia, como el formato de fecha no es inteligible por los algoritmos disponibles en las librerías de clasificadores con las que trabajamos —scikit-learn, xgboost y lgbm— la dividí en dos nuevas características, el tiempo sucedido desde que se registró la primera de la que tenemos constancia en el conjunto de train en días por un lado, y una segunda categoría, el mes en el que se registró, por si tuviera alguna relevancia que desconocemos —luego descubriremos que no la tiene y la eliminaré—, la idea era no perder información ni del tiempo transcurrido ni de la periodicidad inherente en el tiempo.

```python
# String to Python datetime
def parsed(x):
  TIME_STR = '%Y-%m-%d'
  return datetime.datetime.strptime(x, TIME_STR)

# Days from 2002-10-14 (Precomputed)
def getDays(date_c,min_date=parsed('2002-10-14')):
  # Si queremos calular la fecha en vivo
  if min_date is None:
      min_date = parsed(date_c.min())
  days = [(parsed(d) - min_date).days for d in date_c]
  return days

# Month of the year (1-12)
def getMonth(date_c):
  months = [parsed(d).month for d in date_c]
  return months

# Add new columns
def addDatesPrep(data):
  # Obtenemos la columna de fecha
  days = getDays(data['date_recorded'])
  # Aniadimos dias desde la referencia
  data['days_ellapsed'] = days
  # Aniadimos mes del año
  data['month_n'] = getMonth(data['date_recorded'])
```

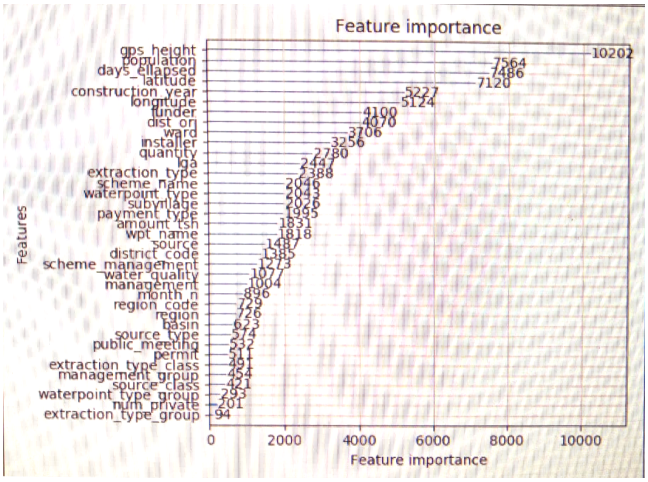
Normalmente el criterio que seguí fue evaluar sobre 5-fold CV y si no empeoraba al aplicar el cambio conservaba la transformación, a fin de ir eliminando las variables no inteligibles por los algoritmos. En este caso pasamos de 0.7954 a 0.7966, que si bien no tenemos confianza para indicar que sea estadísticamente mejor, parece que al menos no es peor.

4.2.2. Distancia a un punto

Esta transformación se volverá a realizar después de la imputación, pero la describimos aquí porque la desarrollé en primer lugar. La idea era introducir una tercera coordenada no ortogonal a las dos ya presentes, probé con varios orígenes de referencia y no encontré una mejora significativa de ninguno respecto al otro, así que elegí la coordenada 0,0. El resultado fue una nueva coordenada relativamente redundante perpendicular la linea roja dibujada, además añadí una segunda variable booleana que reflejaba si la coordenada que imputaría en un paso posterior era la original o una imputada.



A pesar de ser redundante vi que los resultado mejoraban ligeramente sobre CV —de 0.7966 a 0.7971— y en la gráfica de importancia sobre las características utilizadas por lgbm estaba entre las primeras. Siempre estamos a tiempo de eliminarla posteriormente para reducir la dimensionalidad. En un exabrupto de genialidad tras pasarle una foto a un compañero de esta gráfica la eliminé , por eso tiene una calidad tan pobre, porque he tenido que reutilizar una fotografía de la gráfica.




```python
# Librería para la distancia esférica
from geopy.distance import geodesic

def addGeoDist(data):
  # Obtenemos tupla long, lat
  coords = list(zip(data['longitude'],data['latitude']))

  # Eliminamos los valores nulos
  msk = np.abs(np.sum(coords,axis=1)) < 1e-5

```

```
    # Booleano coordenada imputada
    data['reliable_gps'] = ['T' if m else 'F' for m in msk]

    # Calculo distancia
    kms = [geodesic(p1,(0,0)).km for p1 in coords]
    data['dist_ori'] = kms

    return msk
```

4.2.3. Word2Vector

La siguiente estrategia utilizada fue transformar las variables categóricas en numéricas con el mismo algoritmo word2vec que ha demostrado buenos resultados con variables categóricas [8], para ello utilicé la librería de *gensim* [9].

El código es demasiado extenso para copiarlo aquí, el procedimiento es el siguiente. Elegimos las columnas que queremos transformar, creamos una bolsa de palabras con el conjunto completo —unión de todas las columnas— con la forma "Feature {value} {column_name}", definimos un tamaño de ventana y un tamaño para los vectores —definido experimentalmente—. Aunque este procedimiento —idealmente— es capaz de separa espacialmente las frases según su parecido dadas sus características semánticas, el aumento de la dimensionalidad es tremendo, número de características categóricas por tamaño de los vectores—.

```
CAT_KEYS =
  ↪  ["funder","installer","wpt_name","basin","subvillage","region",
"lga","ward","recorded_by","scheme_management","scheme_name",
"extraction_type","extraction_type_group","extraction_type_class",
"management","management_group","payment","payment_type","source",
"source_type","source_class","water_quality","quality_group",
"quantity","quantity_group","waterpoint_type","waterpoint_type_group"]
BIG_CAT_KEYS = ['funder', 'installer', 'wpt_name', 'subvillage',
  ↪  'ward', 'scheme_name']
```

Probé en primer lugar con *CAT_KEYS* y la explosión de la dimensionalidad era inasumible, y posteriormente con *BIG_CAT_KEYS* y el rendimiento no mejoraba en train —de 0.7969 a 0.7963—, empeoraba en DrivenData —0.8034 a 0.7937—, y era costosísimo a nivel computacional, así que lo descarté.

[8] MyYellowRoad. Yonatan Hadar. Using categorical data in machine learning with python: from dummy variables to Deep category embedding and Cat2vec -Part 2https://blog.myyellowroad.com/using-categorical-data-in-machine-learning-with-python-from-dummy-variables-to-deep-category-42fd0a43b009

[9] Gensim. Word2Vec https://radimrehurek.com/gensim/models/word2vec.html

4.2.4. Orden por clasificación de las variables categóricas

Tras descartar la intentona anterior probé otra estrategia también decrita por Yonatan Hadar[10]. La idea es que aquellas etiquetas que clasifiquen de una forma similar estén próximas entre ellas, este procedimiento, al realizarlo fuera de la validación cruzada sí que produce *data leaking*, y encontramos información sobre las etiquetas de salida de las instancias de validación en el conjunto de entrenamiento —a través de las estadísticas sobre el conjunto completo—, sin embargo tomé dos precauciones, esta fue la última modificación antes de pasar a modificar los clasficadores y lo validé sobre el test en la página de DrivenData, mal hecho lo anterior pero decidí priorizar el desarrollo rápido y poder realizar muchas pruebas por encima del rigor —y a la luz de los resultados en DataDriven parece que no fue una estrategia equivocada—. Como tenemos tres categorías podía o crear dos columnas de salida por cada columna de entrada para reflejar toda la información o elegir la mayoritaria —opté por esta última ópción—.

```
def ordenateCategories(tr,tst,y):
  # Obtenemos las columnas categóricas
  keys = list(filter(lambda x: tr[x].dtype == np.dtype('O'),
  ↪  tr.columns))
  # Por cada columna
  for k in keys:
      # Obtenemos los valores distintos sobre el train
      vals = list(Counter(tr[k].values.reshape(-1)))
      # Los ordenamos por su porcentaje de clasificacion como
      ↪  funcionales
      impV = sorted(map(
          lambda x: (np.sum((y[tr[k] == x] ==
          ↪  'functional').values) / np.sum((tr[k] ==
          ↪  x).values), x),vals),reverse=True)
      # Creamos un diccionario asignando un índice según su
      ↪  posición
      d = dict(map(lambda x: (x[1][1], x[0]), enumerate(impV)))

      # Modificamos train
      tr[k] = list(map(lambda x: d.get(x,-1),tr[k].values))
      # Modificamos test
      tst[k] = list(map(lambda x: d.get(x,-1),tst[k].values))
```

[10] MyYellowRoad. Yonatan Hadar. Using categorical data in machine learning with python: from dummy variables to Deep category embedding and Cat2vec -Part 2https://blog.myyellowroad.com/using-categorical-data-in-machine-learning-with-python-from-dummy-variables-to-deep-category-42fd0a43b009

4.3. Reducción de la dimensionalidad

Las siguientes transformaciones no se encuadran temporalmente en este momento concreto sino que forman parte de otrasde modificaciones que veremos más adelante, sin embargo, materialmente tiene sentido agruparalas aquí.

4.3.1. Eliminación de características

El siguiente paso fue utilizar el análisis realizado previamente para eliminar las columnas estrictamente redundantes o sin significancia para el problema, estas son *recorded_by* ,*payment*, *quality_group*, *quantity_group*, excepto *recorded_by* las otras estaban duplicadas en el dataset, y eran explicadas de la misma manera o de forma más precisa por otra variable. De esta modificación no registré la evaluación porque entendí que sólo podía beneficiar a los algoritmos y evitar la *maldición de la dimensionalidad*.

4.3.2. Reducción de características

Al substituir *LabelEncoder* por la transformación personalizada que veremos a continuación añadí una transformación preliminar para reducir la dimensionalidad del problema, tomé aquellas variables categóricas que restaban por transformar y por cada una de ellas transformé los valores nulos a una nueva categoría llamada *"NaN"*, conté las apariciones de cada valor y aquellos con menos de $MIN = 50$ apariciones los agrupé en una nueva etiquetada llamada *"Others"*. El valor de MIN lo definí experimentalmente, a medida que aumentaba el valor el resultado sobre la validación cruzada mejoraba pero llegaba un punto en el que penalizaba el rendimiento, 50 estaba próximo a ese punto dulce.

```
def reduceBigCat(tr,tst):
  MIN = 50
  gen_label = 'Other'
  # Unimos train y test
  X = pd.concat([tr,tst],axis=0)

  # Por cada columna de tipo no numérico
  for k in list(filter(lambda x: tr[x].dtype == np.dtype('O'),
  ↪  tr.columns)):
      # Sustituimos valores nules por nueva etiqueta nula
      col = X[k].fillna('NaN')
      # Contamos ocurrencias
      c = Counter(col.values.reshape(-1))
      # Creamos una máscara con instancias a reducir
      msk = col.isin(list(filter(lambda x: c[x] < MIN, c)))
      # Cambiamos la etiqueta
      col[msk] = gen_label
      X[k] = col
```

```
18
19    # Devolvemos los dos subconjuntos de instancias separados
20    return X.iloc[:tr.shape[0],:], X.iloc[tr.shape[0]:,:]
```

4.3.3. Eliminación de instancias (Ruido)

En este caso, la reducción del ruido la realicé después de la imputación de las coordenadas y altura que veremos a continuación porque esta imputación no se basaba en la etiqueta de salida, y por lo tanto el ruido no se vería amplificado, además, las coordenadas nulas ya las había detectado y eliminado, al comprobar que entraran dentro de valores razonables para la posición geográfica de Tanzania. Sin embargo la analizo aquí por seguir un orden lógico en el tratamiento de los datos.

Para reducir el ruido utilicé la técnica basada en CV donde entreno con un subconjunto pero predigo la totalidad de los datos, tras las cinco particiones, aquellas instancias que no se hayan clasificado correctamente ninguna vez son eliminadas. Para la eliminación creo una máscara para que a la hora de la validación cruzada se entrene con el nuevo subconjunto pero se valida con la totalidad de los datos.

```
1  def cleanNoise(modelo,X,y):
2    # 5fold CV
3    cv = StratifiedKFold(n_splits=5, shuffle=True,
     ↪  random_state=123456)
4    # Matriz de clasificación
5    corr = np.zeros((X.shape[0],5),dtype=np.bool)
6
7    i = 0
8    for train, test in cv.split(X, y):
9        # Entrenamos con subconjunto
10       modelo = modelo.fit(X[train], y[train])
11       # Evaluamos todos los datos
12       y_pred = modelo.predict(X)
13       # Añadimos la columna con los aciertos
14       corr[:,i] = y_pred == y
15       i += 1
16   # Reduzco a una sola columna de booleanos
17   # donde los 0 deben ser eliminados
18   m = np.apply_along_axis(np.sum,axis=1,arr=corr).astype(np.bool)
19
20   return m
```

El resultado parecía prometedor, pues conseguí pasar en train de 0.8165 a 0.8191, sin embargo en DrivenData, bajó de 0.8253 a 0.8241. Esto me hace

pensar que si bien estos casos no pueden ser clasificados y esta estrategia ayuda a la clasificación más robustadel conjunto de entrenamiento, el desplazamiento de los *centroides* —en el sentido más poético posible— de las hojas de los árboles de decisión ayuda a una mejor clasificación sobre el test. Por lo tanto asumo que no es efectivamente ruido sino instancias que representan a un grupo muy reducido y que quedan absorvidas por subconjuntos mayores, sin embargo el incremento del tamaño de los árboles, para crear grupos más especializados repercutía negativamente sobre la validación cruzada, pues sobreajustaba los datos, con los parámetros que definí y veremos luego intenté equilibrar este fenómeno para obtener el mejor balance de esta situación indeseable.

4.4. Imputación

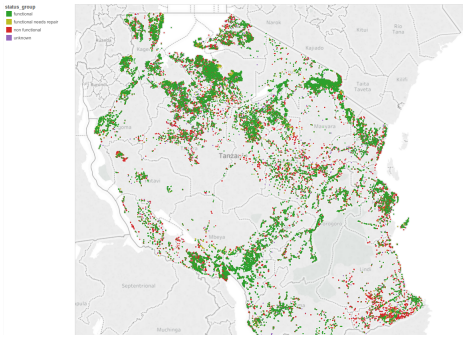
A la luz de la gráfica de importancia del LGBM decidí imputar longitud, latitud y altura, éstas tres por distintos motivos, el primero es que tenemos variables categóricas que hacen referencia a la posición geográfica bien cumplimentadas y en segundo lugar, tenía confianza en que la nulidad de estos datos no refleja ninguna condición del pozo sino de la toma de los datos, sin embargo el resto de variables —a excepción de *amount_tsh* que también intenté imputar pero el resultado no fue bueno— no está claro, por su pobre definición o por su propia naturaleza, que su nulidad sea por una pobre cumplimentación de los datos o por las propias características de los objetos que describen. También probé otras estrategias de imputación menos robustas como la mediana o KNN sobre variables categóricas transformadas pero en primer lugar no daban buenos resultados y en segundo lugar no estaba seguro que semánticamente reflejaran la naturaleza de las instancias.

4.4.1. Imputación de longitud y latitud

Esta era una de las variables que tenía más interés en imputar, por la importancia de éstas en las gráficas antes vistas —importancia de variables y distribución de clases en el mapa—. Podía hacer una imputación con suficiente confianza sobre su robustez basándome en las etiquetas categóricas que describen la posición en el espacio, y además es evidente que todos los punto de extracción de agua tienen una posición en el espacio.

El mapa creado por Calin Uioreanu [11] fue un gran motivador para esta decisión, al ver que las distintas etiquetas se distribuían con distintas densidades dependiendo de la zona.

[11] Calin Uioreanu. Community Driven Data. `https://community.drivendata.org/t/pumps-visualization-dashboard-with-tableau/186`. Tableau. `https://public.tableau.com/profile/calin.uioreanu#!/vizhome/DataMiningtheWaterTableDrivenData_com/Bubblestatusquantity`



El problema que enfrentaba era que las variables categóricas no se llevan bien con los algoritmos de regresión disponibles en las librerías que estábamos usando —LGBM y XGBooster sí que lo admiten—, por lo tanto decidí implementar mi propio KNN con mi propia métrica de distancia, ya que el orden alfabético o el orden estadístico en principio no debería reflejar ninguna relación espacial. La función de similitud es el número de etiquetas en las características de interés (*GPS_KEYS*) que comparten.

```
GPS_KEYS = ["basin", "subvillage", "region", "region_code",
 ↪  "district_code", "lga", "ward"]

# En preprocess
def preprocess(data_x, data_y, data_x_tst):
  ...
  data_x_copy = data_x.copy()
  # Máscara para las etiquetas de interés de referencia
  label_mask = [x in GPS_KEYS for x in data_x.columns]
  # Máscara para longitud y latitud
  coord_mask = [x in ['longitude','latitude'] for x in
   ↪  data_x.columns]

  # Coordenadas predichas
  coords = replaceGps(
      data_x_copy.iloc[np.logical_not(msk),label_mask].values,
      data_x_copy.iloc[np.logical_not(msk),coord_mask].values,
      data_x_copy.iloc[msk,label_mask].values)

  # Sustituimos en los datos
  data_x.iloc[msk, coord_mask] = coords
  ...

# Función con el KNN
def replaceGps(ref,ref_y,dst):
  # Etiquetas de interés
  gGps = ref
```

```
    # Coordenadas de entrenamiento
    gY = ref_y
    # Instancias a predecir
    bGps = dst

    # Función de distancia
    dist = lambda x1,x2: np.sum([l1==l2 for l1,l2 in zip(x1,x2)])

  Tres vecinos
    KNN = 3

    res = []
    # Definición del KNN por columna
    def knn(row):
        # Calculamos todas las distancias
        d = np.apply_along_axis(lambda x: dist(x, row), 1, gGps)
        # Apilamos distancias y coordenadas
        m = np.hstack((d.reshape((-1, 1)), gY))
        # Ordenamos por distancia
        m = sorted(m, key=lambda x: x[0], reverse=True)
        # Elegimos los vecinos
        mnn = np.array(m[:KNN])
        # Ponderamos por la similitud
        mnn[:, 1] *= mnn[:, 0]
        mnn[:, 2] *= mnn[:, 0]
        # Devolvemos la media para longitud y latitud dividido por
        ↪  la media de los pesos
        return np.mean(mnn, axis=0)[1:] / np.mean(mnn, axis=0)[0]

    # Multithreading
    p = mp.Pool(4)
    res = p.map(knn, bGps)
    p.close()
    p.join()

    return res
```

El resultado no lo pude contrastar en DrivenData porque fue la primera subida que pude hacer —hasta entonces no tuve acceso— pero sobre el entrenamiento el resultado no empeoraba, 0.7971 a 0.7969, y me permitiría después imputar la altura con mayor seguridad. Cogí tres casos aleatorios y comprobé en Google Maps que las coordenadas coincidían con la descripción de las etiquetas de interés, así que quedé muy satisfecho.

4.4.2. Imputación de altura

El siguiente paso fue imputar la altura, una vez que tenía todas las coordenadas imputadas, podía entrenar un regresor para la altura, en este caso probé con la implementación de KNNRegressor de Scikit-Learn y con LGBMRegressor y el segundo me dio mejor resultado, así que este fue el que conservé

```
GPS_KEYS = ["basin", "subvillage", "region", "region_code",
↪  "district_code", "lga", "ward"]

# En preprocess
def preprocess(data_x, data_y, data_x_tst):
  ...
  # Imputamos train consigo mismo
  impZeroHeight(data_x,data_x)
  # Imputamos el test con el modelo del train
  impZeroHeight(data_x,data_x_tst)
  ...


def impZeroHeight(src,dst):
  # Obtenemos X(long,lat) e y(height) origen
  coords_tr = src[['longitude','latitude']]
  height_tr = src[['gps_height']]
 Obtenemos X(long,lat) e y(height) destino
  coords_tst = dst[['longitude','latitude']]
  height_tst = dst[['gps_height']]

  # Máscara para el test
  tst_msk = (height_tst == 0).values.reshape(-1)
  coords_tst = coords_tst.values[tst_msk]

  # Máscara para el train
  msk = (height_tr != 0).values.reshape(-1)
  # X(long,lat) no nulo
  nonzero_coords = coords_tr.values[msk]
  # y(height) no nula
  nonzero_height = height_tr.values[msk]

  # Creamos el regresor y lo entrenamos
  rg = lgb.LGBMRegressor(n_estimators=500)
  rg.fit(nonzero_coords, nonzero_height)

  #Predecimos el destino
  lbs = rg.predict(coords_tst)

```

```
39  # Sustituimos en el destino
40  dst.iloc[tst_msk, list(map(lambda x: x == 'gps_height',
    ↪  dst.columns))] = lbs.reshape((-1,1))
```

Este cambio fue significativo, me hizo pasar sobre CV de 0.7969 a 0.7978 y en DD de 0.8030 a 0.8155, un salto de casi 200 posiciones, ya estaba más cerca de mi objetivo.

4.4.3. Otras imputaciones

Intenté imputar otras variables, como *amount_tsh* pero los modelos arrojaban peor rendimiento, asimismo intenté sustituir los valores nulos en aquellas variables que no pudiera imputar de forma razonable con la mediana y con el knn sobre las variables ordenadas, pero seguía sin mejorar el valor de *accuracy*. Estas funciones pueden encontrarse en el código fuente pero no las comento porque las descarté rápidamente.

5. Clasificadores

Hasta el momento había estado trabajando con una configuración inicial del LGBM con 1000 estimadores, que era el punto dulce de estimadores que obtuve nada más afrontar el problema, más o menos estimadores repercutía negativamente, sin embargo desde entonces no había modificado estos valores. Los clasificadores más populares en el foro del problema eran Random Forest de Scikit-Learn que me daba un resultado ligeramente peor que LGBM y XGBoost, que arrojaba un resultado similar pero era insufriblemente lento y obstaculizaba la dinámica de desarrollo que llevaba hasta el momento.

5.1. Uso de dos clasificadores (Dual)

Tras aplicar la ordenación de etiquetas por sus estadísticas había conseguido colocarme por encima de los 300 mejores, y no conseguía mejorar el rendimiento con LGBM, así que resolví utilizar la decisión combinada de LGBM y Random Forest, que eran los más rápidos. Leí que Random Forest trabajaba mejor con *OneHot Encoding* así que sin eliminar las columnas anteriores, expandí 5 columnas que según la gráfica de importancia parecían ayudar a clasificar el problema y tenían un número reducido de etiquetas posibles por característica, estas son *lga*, *payment_type*, *source*, *waterpoint_type* y *basin*. En este momento tenía unas 140 columnas en total. Tras esto intenté podar características que se presentaban en la gráfica como poco útiles a la clasificación pero empeoraban ligeramente el resultado.

```
1 INTERSTING_SMALL = ['basin','lga','payment_type', 'source',
  ↪  'waterpoint_type']
2 from sklearn.preprocessing import OneHotEncoder
```

```
def oneHot(tr,tst):
  keys = INTERSTING_SMALL
  # Generamos el encoder
  oh = OneHotEncoder()
  # Lo entrenamos sobre train (no había variables en test que no
  ↪  estuvieran en train)
  oh.fit(tr)

  # Transformamos train y test y lo devolvemenos añadido al resto
  ↪  de columnas
  mini_tr = oh.transform(tr).toarray()
  mini_tst = oh.transform(tr).toarray()

  return \
      pd.concat([tr,pd.DataFrame(mini_tr)],axis=1),\
      pd.concat([tst,pd.DataFrame(mini_tst)],axis=1)
```

Ahora definí mi clasificador, posteriormente añadí la opción de devolver la matriz tridimensional de probabilidades para optimizar la ponderación de cada uno de ellos. El procedimiento es sencillo, entreno los clasificadores de forma individual y devuelvo la clase con mayor suma.

```
class CustomClassifier:
  clfs = []
  def __init__(self,*clf):
      # Creamos la lista de clasificadores
      self.clfs = clf

  # fit: Entrenamos cada clasificador por separado
  def fit(self,X,y):
      for clf in self.clfs:
          clf.fit(X,y)
      return self
  # predict: Obtenemos el máximo de la suma de las probabilidades
  def predict(self,X,return_prob=False):
      # Obtenemos las predicciones individuales
      res = [clf.predict_proba(X) for clf in self.clfs]
      # Obtenemos la lista de etiquetas
      k = self.clfs[0].classes_

      # Definimos una matriz vacía
      z = np.zeros(res[0].shape)
      # Sumamos cada probabilidad
      for r in res:
          z += r
```

```
24
25      # Tomamos el índice de la etiqueta más probable
26      pos = np.apply_along_axis(lambda x: np.where(x == x.max()),
        ↪  axis=1, arr=z).reshape(-1)
27
28      # Esta es la opción original
29      if not return_prob:
30          return list(map(lambda i: k[i], pos))
31      # Esto se añadió posteriormente (predicción + volumen de
        ↪  probabilidades)
32      else:
33          return list(map(lambda i: k[i], pos)),
            ↪  np.stack(res,axis=2)
34
35  # predict_proba: Devolvemos el volumen de probabilidades
    ↪  individuales
36  def predict_proba(self, X):
37      # Obtenemos las predicciones individuales
38      res = [clf.predict_proba(X) for clf in self.clfs]
39
40      # Definimos una matriz vacía
41      z = np.zeros(res[0].shape)
42      # Sumamos cada probabilidad
43      for r in res:
44          z += r
45
46      return z
```

Con este cambio, conseguí crear un clasificador más robusto que utilizaba las confianzas individuales para clasificar las instancias, el resultado fue pasar en train de 0.8099 a 0.8146. A la luz de estos resultados decidí añadir un tercer clasificador.

5.2. Uso de tres clasficadores (Triple)

Añadí un tercer clasificador, XGBooster, que estaba muy referenciado en el foro del problema, lo evalué y el resultado fue extraordinario, pasé en train de 0.8146 a 0.816 y en DD desde 0.8218 (Variables categóricas ordenadas) a 0.8255, esto me puso en la posición 57 en el momento de la subida y hasta el final del plazo para subir nuestras predicciones.

Viendo lo prometedor de esta solución vi que XGBooster tenía una opción sobre GPU, lo cual me permitiría acelerar el proceso de entrenamiento —en ese momento tardaba entre 500 y 800 segundos por entrenamiento y predicción debido al alto número de columnas—. Así que instalé CUDA, conseguí bajar cada etapa de fit/predict a unos 200 segundos y seguí avanzando. LGBM tam-

bién tiene opción sobre GPU pero utiliza OpenCL, no CUDA, y el tiempo de ejecución de este algoritmo era el menor de los tres, así que no me preocupé por el pues desde luego no era el cuello de botella.

6. Ajuste de hiperparámetros

6.1. Algoritmo genético para los hiperparámetros

Como el número de parámetros era de 10, era inasumible utilizar una rejilla de hiperparámetros, así que definí un algoritmo genético personalizado, muy orientado a la intensificación y poca exploración, ya que cada ejecución era larga y sólo necesitaba resultados ligeramente mejores para colocarme entre los cincuenta mejores. Definí un vector de 10 elementos limitado entre 0 y 1, que representaban los parámetros, y funciones personalizadas que los convertían a la horquilla —enteros o coma flotante— que definiera.

Con una población inicial de 6 —5 aleatorios y la configuración utilizada hasta el momento—, en cada iteración se eligen dos padres, se crean dos hijos cruzándolos con un valor comprendido entre los dos padres más un factor multiplicador de la horquilla de 1.15, mutan con una probabilidad del 0.1 —como son diez características siempre muta un valor con un número aleatorio entre 0 y 1— y se evaluan. Se añaden los dos a la población —en vez de añadir sólo el mejor—, se ordenan y se seleccinan de nuevo los seis mejores. Como se puede ver este algoritmo tiene una capacidad exploratoria muy reducida, casi determinada por la población aleatoria inicial, pero como las evaluaciones son muy costosas no quería encontrar el óptimo global sino un óptimo en la localidad de la solución actual. La condición de salida la puse a 200 evaluaciones pero es irrelevante porque nunca llega a evaluar 400 configuraciones, lo interrumpo manualmente.

Probé también con differential evolution, de la librería *scipy*, pero se estabilizaba en la primera evaluación y salía demasiado rápido. No adjunto el código del algoritmo porque es relativamente extenso pero se puede encontrar en el código fuente adjunto. En cada evaluación de una configuración utilizaba como estimador sólo las dos primeras evaluaciones del 5-fold CV, que eran un indicador pesimista —comprobado experimentalmente— del resultado de las cinco evaluaciones. Las horquillas de los parámetros las definí experimentalmente considerando la calidad de los resultados y el tiempo de ejecución. En el algoritmo cacheo las respuestas para cada configuración una vez transformada, pues valores decimales próximos arrojan la misma configuración.

```
1 def intbetween(min,max):
2   return lambda x: int(x*(max-min) + min)
3
4 def floatbetween(min,max):
5   return lambda x: x*(max-min) + min
6
```

```
# Dentro del train
def train(X,y):
  ...
  # XGBooster: max_depth, n_estimators
  p1 = [intbetween(3,21),intbetween(50,800)]
  # LGBM: boosting_type, num_leaves, max_depth, n_estimators
  p2 =
  ↪  [oneof(['gbdt','dart','goss']),intbetween(20,101),intbetween(5,100),intbetween(200,150
  # Random Forest: n_estimators, max_depth, min_samples_split,
  ↪  min_samples_leaf=
  p3 =
  ↪  [intbetween(100,1200),intbetween(10,31),intbetween(2,20),intbetween(1,20)]

  p_transf = p1+p2+p3
  ...

def buildCls(c, c_transf):
  p = getParams(c, c_transf)
  clf1 = xgb.XGBClassifier(n_estimators=p[1], max_depth=p[0],
  ↪  n_jobs=NPROC,
  ↪  objective='multi:softmax',tree_method='gpu_hist')
  clf2 = lgb.LGBMClassifier(boosting_type=p[2], num_leaves=p[3],
  ↪  max_depth=p[4], n_estimators=p[5],
                            objective='multiclass', n_jobs=NPROC)
  clf3 = RandomForestClassifier(n_estimators=p[6],
  ↪  max_depth=p[7], min_samples_split=p[8],
  ↪  min_samples_leaf=p[9],
                                n_jobs=NPROC)

  clf = CustomClassifier(clf1, clf2, clf3)

  return clf
```

Sin embargo, pese a todo lo anterior no conseguí mejorar el resultado obtenido, aunque sobre el train sí, que conseguía pasar a 0.817 e incluso 0.818, a la hora de evaluarlo en DrivenData el resultado no era superior al obtenido anteriormente.

6.2. Evitar sobreajuste

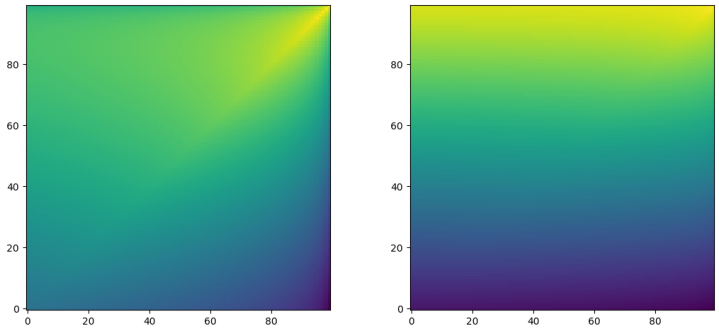
El siguiente paso fue reducir el ruido como ya comenté con anterioridad y el resultado no era prometedor. Con todo esto, al ver que el problema mejoraba sobre el train pero no en DD, sospeché que podía estar sobreajustando algunas características presentes en el train que no aparecían en el test, por eso definí una nueva fórmula para calcular el *fitness* del algoritmo genético, en la que

penalizaba el sobreajuste, pero seguía reflejando la medida de *accuracy* que utilizaban en la web, para ella utilizaba la media sobre la validación cruzada y la media de la evaluación sobre el total de los datos en cada paso de 5-fold —menos costoso que entrenar con el conjunto de datos para el algoritmo genético—. La gráfica de la fórmula tiene el siguiente aspecto, a la izquierda con $\alpha = 0{,}5$ y a la derecha 0,9 —el valor que mejor reflejaba el comportamiento que esperaba—. Siendo x_v —eje y en la gráfica— el valor sobre la validación y x_a —eje x en la gráfica— sobre el total de los datos.

$$x'_a = 1 - (x_a - x_v)/(1 - x_v)$$

$$x'_a = \frac{1}{x'_a} \quad if \quad x'_a > 1 \quad else \quad x'_a$$

$$f(x_v, x_a) = x_v * \alpha + \sqrt[5]{x'_a} * (1 - \alpha)$$



El resultado es una fórmula basada en la tasa de acierto que maximiza el valor en validación y penaliza el sobreajuste, ponderado experimentalemnte. El resultado sin embargo, aunque parecía prometedor, pasaba de 0.816/0.9503 en el clasificador triple a 0.0.8172/0.8841 lo que me hizo estimar que había conseguido el objetivo, sin embargo, pasé en DD de 0.8255 a 0.8202.

Esta fue la última subida con el usuario creado para la competición con el nombre de *Guillermo_Gomez_UGR*.

6.3. Extra. Differential evolution para ponderación de clasificadores individuales

Como veía tan cerca aparecer entre los cincuenta primeros y seguía obsesionado con que el ajuste de hiperparámetros me permitiría subir esas décimas que me separaban del primero —0.0031— a la mañana siguiente me desperté pensando que había ajutado los hiperparámetros de los clasificadores individuales sin atender a las características de las predicciones de cada uno, LGBM era el más robusto, pero era posible que hubiera situaciones que RF y XGBoost se aliaran en su contra o que fueran sobreoptimistas. Así que se me ocurrió otra optimización sencilla para ajustar los clasificadores individuales, basada en dos parámetros —posteriormente probé con tres pero no suponían una mejora sobre dos—.

1. Una potencia que modificara los valores de probabilidad de cada clasificador de forma individual —elimina las probabilidades de suma uno pero sigue conservando la ordenación—, potencias por encima de 1 penalizan las confianzas pobres, y potencias entre 0 y 1 hacen a los clasificadores más optimistas.

2. Una suma ponderada de las distintas probabilidades para cada clase, hasta ahora la ponderación era simétrica ($\frac{1}{3}$ en nuestro caso), sin embargo es posible que no todos los clasificadores sean igual de interesantes para la predicción.

Modifiqué la función de validación cruzada para que devolviera el volumen de probabilidades, como el algoritmo de *Differential Evolution* de la librería *SciPy* permite definir las horquillas no tuve que codificar cada individuo de la población entre 0 e 1 y el código se simplifico bastante —llegados a este punto el código fuente es un campo de minas de modificaciones y comentarios—.

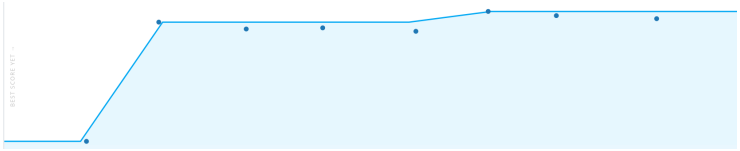
En cada evaluación del algoritmo genético, tras hacer la validación cruzada, obtengo la matriz de de probabilidades y busco los pesos óptimos para las características. Como esta parte quedaba fuera de la práctica el código es menos legible, lo adjunto por consistencia con el resto de la documentación. El exponente finalmente lo acoté entre 1 y 10, porque nunca elegía valores negativos y ralentizaba la ejecución; y el factor de ponderaciónn entre 0 y 10, aunque igual habría dado entre 0 y 1 siempre que sea la misma horquilla para las tres.

```
# Obtener Accuracy ponderada
getAcc = lambda x,y: np.sum(np.where(x ==
  ↪  np.repeat(np.max(x,axis=1),3,axis=0).reshape((-1,3)))[1] ==
  ↪  y)/y.size
# Obtener lista de índices
getYIdx = lambda x: np.where(x ==
  ↪  np.repeat(np.max(x,axis=1),3,axis=0).reshape((-1,3)))[1]
# Ponderar las probabilidades para cada matriz n_clasificadores x
  ↪  n_instancias
pond = lambda y,pw,fc: np.power(y, np.repeat(pw,
  ↪  y.shape[0]).reshape((len(pw), -1)).transpose())*np.repeat(fc,
  ↪  y.shape[0]).reshape((len(fc), -1)).transpose()


# Accuracy ponderado con etiquetas codificadas entre 0 y 2
def getPondAcc(prob,y,pw,fc):
  z = np.zeros(prob.shape)
  # Lo aplico a cada matriz n_clasificadores x n_instancias por
    ↪  no hacer más ilegible el código
  for i in range(prob.shape[1]):
      z[:, i, :] = pond(prob[:, i, :], pw, fc)
  return getAcc(np.sum(z,axis=2),y)
```

```python

# Accuracy ponderado con etiquetas de texto
def getPondAcc_text(prob,y,v):
  pred_y = getPondRes(prob,v)
  return np.sum(pred_y == y)/len(y)

# Predicción ponderada para aplicar al test
def getPondRes(prob,v):
  k = ['functional', 'functional needs repair', 'non functional']

  # Obtenemos las probabilidades modificadas
  z = np.zeros(prob.shape)
  for i in range(prob.shape[1]):
      z[:, i, :] = pond(prob[:, i, :], v[:3], v[3:])
  # Obtenemos los índices de las etiquetas más votadas
  y_pred = getYIdx(np.sum(z, axis=2))
  # Transformamos a etiquetas de texto
  y_pred_lab = [k[i] for i in y_pred]

  return y_pred_lab


from scipy.optimize import differential_evolution

# Aplica DE para obtener la mejor configuración
def bestWeight(prob_all, y):
  # Transformamos la clase a numérica
  k = ['functional', 'functional needs repair', 'non functional']
  y_n = np.zeros(y.shape)
  for i, ke in enumerate(k):
      y_n[y == ke] = i

  # Evaluación candidato
  def eval_cad(v):
      nonlocal prob_all, y_n
      #v[:3] = potencias, v[3:] = ponderación
      return getPondAcc(prob_all, y_n, v[:3], v[3:])

  de = differential_evolution(
      # Como el algoritmo minimiza una función multiplicamos por
      ↪  -1 el valor de ajuste
      lambda x: -eval_cad(x),
      [(1,10)]*3+[(0,10)]*3,
      # El que mejor resultado arroja
      strategy='rand1exp',
```

```
60        # Defino las iteraciones máximas a 100 aunque he comprobado
          ↪  que no las consume
61        maxiter=100
62    )
63    #Devolvemos el ajuste y la configuración que lo logra
64    return -de.fun, de.x
```

Y finalmente, tras ejecutarlo durante una noche, conseguí una configuración que me permitió obtener 0.8179/0.9312 frente a los 0.816/0.9503 de la mejor solución hasta el momento, y conseguir en DD una precisión de 0.8258, sólo 0.0005 mejor que mi anterior solución pero que me permitió colarme en el puesto 48 con el usuario *guillermogotre* —adjunto una captura para comprobar que no utilicé esta cuenta para realizar subidas mientras duró la práctica—.



Submissions

BEST	CURRENT RANK	# COMPETITORS	SUBS. TODAY
0.8258	48	6188	0 / 3

SUBMISSIONS

Score	Submitted by	Timestamp
0.8029	guillermogotre	2019-01-06 19:51:01 UTC
0.8239	guillermogotre	2019-01-06 23:58:48 UTC
0.8227	guillermogotre	2019-01-06 23:59:17 UTC
0.8229	guillermogotre	2019-01-07 16:56:05 UTC
0.8223	guillermogotre	2019-01-07 17:18:00 UTC
0.8258	guillermogotre	2019-01-08 00:05:14 UTC
0.8251	guillermogotre	2019-01-08 22:34:13 UTC
0.8245	guillermogotre	2019-01-08 23:26:31 UTC

7. Reflexiones finales

La práctica ha sido un ejercicio divertidísimo aunque me ha robado más tiempo del que me podía permitir. El principal elemento de conflicto ha sido el sobreajuste, llegados a este punto sigo sin poder utilizar esta información para estimar la calidad de un modelo, y es evidente que el resultado obtenido en la validación cruzada no es suficiente para explicar el rendimiento sobre el

conjunto de test. Se observa que es interesante cierta cantidad de sobreajuste para explicar las instancias del test —no he comprobado si hubiera instancias repetidas en ambos conjuntos—, aunque de ninguna manera es indicador de la calidad del modelo. Por otro lado, la aleatoriedad juega un papel crítico en los resultados, tanto en la formación de los conjuntos de validación como en el funcionamiento estocástico de los algoritmos de clasificación, y dos ejecuciones con dos semillas distintas con los mismos hiperparámetros pueden suponer una puntuación de 0.8258 o de 0.8230, que no son dramáticamente distintos pero en una competición con tantos candidatos puede suponer varios cientos de puestos de diferencia.

Por otro lado, es posible que mi discurrir durante la práctica no haya sido el correcto, cuando llegué a la evaluación del modelo con tres clasificadores distintos y conseguí llegar a la posición 57 asumí que una mejor configuración de los hiperparámetros me reservaría una plaza entre los mejores resultados, y ha sido así sólo relativamente, conseguí con esto escalar diez posiciones pero sigo estando muy lejos de la primera posición —después de dos días de pruebas sólo conseguir mejorar 0.0005, mientras que el primero está a una distancia de 0.0028, 6 veces más—. Se me presenta ahora evidente que la limitación está en los datos, y que quizás si hubiera invertido ese tiempo en mejor imputación de datos, o en analizar mejor las instancias que eliminaba para discriminar la que efectivamente eran ruido de aquellas que eran casos raros pero de interés predictivo podría haber obtenido una mejor puntuación, por ejemplo, al hacer la documentación y revisar la etapa de visualización, he visto que hay valores raros en *amount_tsh* que ignoré al principio porque no conocía aún el problema, y es posible que esta variable tenga interés predictivo suficiente para tratarla con mayor profundidad pese al primer acercamiento frustrado que tuve.

En definitiva, cuando te enfrentas a un problema a ciegas, donde las referencias no tienen contenido material y sólo vemos los resultados de otros, es difícil tener claro donde focalizar el esfuerzo, más aún cuando el tiempo con el que contamos es limitado, y nos vemos obligados a tomar decisiones intuitivas sobre la marcha, sin embargo estoy muy satisfecho, y no hay mejor manera de entrenar la intuición que con la practica. Dentro de plazo conseguí quedar en la posición 57 y dos días después en la 48, dentro del podio que era donde me había propuesto llegar, todo esto con una experiencia formativa de un gran interés —y mucho más ameno que KNIME :) —.