Universidad de Granada

Ingeniería Informática

Practica 3: Competicion Driven Data.

Autor: José Antonio Ruiz Millán
email: jantonioruiz@correo.ugr.es
Curso: 2018-2019
Asignatura: Inteligencia de Negocio
10 de enero de 2019

Score	Submitted by	Timestamp
0.7589	JoseAntonio_Ruiz_UGR	2018-12-31 00:31:12 UTC
0.7557	JoseAntonio_Ruiz_UGR	2018-12-31 02:47:50 UTC
0.7888	JoseAntonio_Ruiz_UGR	2018-12-31 19:32:46 UTC
0.7889	JoseAntonio_Ruiz_UGR	2019-01-01 12:20:36 UTC
0.7785	JoseAntonio_Ruiz_UGR	2019-01-01 12:20:58 UTC
0.7645	JoseAntonio_Ruiz_UGR	2019-01-01 18:32:17 UTC
0.7906	JoseAntonio_Ruiz_UGR	2019-01-02 04:22:39 UTC
0.7504	JoseAntonio_Ruiz_UGR	2019-01-02 12:53:49 UTC
0.7886	JoseAntonio_Ruiz_UGR	2019-01-02 16:08:36 UTC
0.7769	JoseAntonio_Ruiz_UGR	2019-01-03 02:14:09 UTC
0.6882	JoseAntonio_Ruiz_UGR	2019-01-03 02:26:51 UTC
0.7170	JoseAntonio_Ruiz_UGR	2019-01-03 22:37:55 UTC
0.8173	JoseAntonio_Ruiz_UGR	2019-01-04 01:22:13 UTC
0.8151	JoseAntonio_Ruiz_UGR	2019-01-04 15:07:39 UTC
0.8116	JoseAntonio_Ruiz_UGR	2019-01-04 19:41:06 UTC
0.8214	JoseAntonio_Ruiz_UGR	2019-01-05 19:17:53 UTC
0.8203	JoseAntonio_Ruiz_UGR	2019-01-05 20:44:20 UTC
0.8200	JoseAntonio_Ruiz_UGR	2019-01-05 22:57:10 UTC

Índice

1. Introduccion

La competición será la Pump it Up: Data Mining the Water Table disponible en `https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/`. Con los datos de Taarifa y el Ministerio de Agua de Tanzania, el objetivo es predecir qué bombas de extracción de agua funcionan, cuáles necesitan algunas reparaciones y cuáles no funcionan. Para predecir una de estas tres clases se dispone de 39 variables (numéricas y categóricas) sobre qué tipo de bomba se está usando, cuándo se instaló, cómo se administra, su ubicación, datos sobre la cuenca geográfica, tipo de extracción, coste del agua, etc. Una comprensión inteligente de qué puntos de agua fallarán puede mejorar las operaciones de mantenimiento y garantizar que el agua potable y limpia esté disponible para las comunidades de Tanzania. El conjunto de entrenamiento consta de 59.400 instancias.

En esta competición, se emplea el porcentaje de acierto como score para valorar la calidad de la solución aportada. En esta competición se permite el uso de cualquier software, algoritmo o lenguaje que el alumno considere útil. Está terminantemente prohibido usar la clase en los datos de test, en caso de conocerse, para entrenar, configurar o mejorar el modelo predictivo. Cualquier indicio de esta conducta supondrá la anulación de la práctica.

La siguiente web sirve de ayuda para resolver esta competición: `https://community.drivendata.org/c/pump-it-up-data-mining-the-water-table`

2. Tabla resumen

Score Tst	Fecha-Hora	Posicion actual	Score Train	Score CV	Preprocesado	Algoritmos	Conf-Algoritmos
0.7589	31/12/2018 - 00:31:12	1486	0.8402	0.79	Solo se elimina 'id' y 'date recorded. Se pasan las categoricas a numericas utilizando LabelEncoder() y se le aplica tambien a las numericas	LGM	n_estimator=200 objetive='multiclass'
0.7557	31/12/2018 - 02:47:50	1486	0.92	0.79	Lo mismo del caso anterio mas eliminacion de variables con todos los valores iguales como 'recorder_by'. Utilizacion de correlation matrix para eliminar variables. Ajustes de parametros utilizando RandomizedSearch	LGM	n_estimator=1000 objetive='multiclass'
XGB - 0.7888	31/12/2018 - 19:32:46	1202	LGB - 0.9182 XGB - 0.9769	LGB - 0.803 XGB - 0.806	Lo anterior y ademas imputo los valores de las variables categoricas con la mas comun, y las de las variables numericas con la media de la variable.	LGM XGB	LGB - anterior XGB - n_estimator = 700, max_depth=12, num_class=4, objective=multi:sofmax
LGM - 0.7785 XGB - 0.7889	01/01/2019 - 12:20:58	1187	LGB - 0.9183 XGB - 0.9759	LGB - 0.802 XGB - 0.806	Lo anterior pero ahora en vez de tener las variables categoricas codificadas a un numero por categoria, se utiliza la funcion get_dummies para crear varias caracteristicas sobre una variable. Para ello, la variable debe tener menos de 10 niveles	LGB XGB	Los mismos que anteriormente
0.7645	01/01/2019 - 18:32:17	1187	0.9645	0.8	Esta subida fue una prueba rápida traduciendo el código anterior a R. El resultado fue basante malo, por lo que deseché lo que hice y seguí con el fichero ded Python	XGB	Los mismos que anteriormente
0.7906	02/01/2019 - 04:22:39	1124	0.9985	0.85	Siguiendo con el fichero de Python, ahora aplico SMOTE para hacer balanceo de clases y poder tener una mejor distribucion de cada una de las clases	XGB	Los mismos
0.7504	02/01/2019 - 12:53:49	1124	0.9976	0.86	Lo anterior pero ahora bajamos el umbral de la matriz de correlacion a 0.6 (anteriormente 0.9) para que elimine aun mas variables con correlacion	XGB	Los mismos
0.7886	02/01/2019 - 16:08:36	1124	0.9153	0.804	Ahora cambio la matriz de correlacion de posición ya que me di cuenta que no eliminaba variables categoricas porque no estaban transformadas, vuelvo a poner el umbral 0.9. Elimino Smote (claramente me mejoraba el score falsamente)	XGB	Los mismos
0.7769	02/01/2019 - 16:08:36	1124	0.9564	0.802	Igual que en anterior pero utilizando Random Forest	RF	n_estimator = 1000, max_depth = 20
0.6882	03/01/2019 - 02:26:51	1124	0.87	0.76	En este caso, estaba un poco experimentando y haciendo cosas distintas eliminando variables que a simple vista parecian que no afectaban, modificando parametros de los algoritmos. En resumen, modifique en exceso el fichero porque estaba atascado y no me mejoraba, con la clara valoracion negativa que se obtuvo ya que hacia cambios sin una base real.	XGB	Los mismos
0.7170	03/01/2019 - 02:26:51	1124	0.89	0.801	Lo mismo que el Random Forest anterior pero ahora con LGB	LGM	Los mismos
0.8173	04/01/2019 - 01:22:13	512	0.993	0.809	Llegado a este punto, decidi comenzar desde un punto de partida estable y con sentido ya que anteriormente no estaba consiguiendo unos resultados favorables. Para este caso hice un nuevo fichero que lo que hace es eliminar las variables que contienen algun valor Nan. Las variables numericas toma los 0 como 0 y no como Nan, y como en el comienzo, elimino algunas variables sin importacia como 'recorded_by' y las de la matriz de correlacion. Para las variables categoricas sin Nan y con menos de 10 niveles hago dummy. La variable date_recorded se desglosa en otras variables.	RF	Los mismos
0.8151	04/01/2019 - 15:07:39	512	0.986	0.809	Una vez aqui, ahora decidi imputar todas las variables numericas donde el valor sea 0 utilizando KNN con K=5	RF	Los mismos
0.8116	04/01/2019 - 19:41:06	512	0.996	0.808	Hago un modedlo por votos, utilizando LGM y RF con el fichero anterior	LGM RF	Los mismos
0.8214	**05/01/2019 - 19:17:53**	**303**	**0.94**	**0.813**	**En vez de eliminar las variables que tienen valores nulos (o imputar), tomamos estos valores como una categoria distinta, imputamos unicamente las variables numericas de posicion (latitude,longitude,gps_heigt). No utilizo correlacion para eliminar variables, se hace un estudio previo de todas las variables y se eliminan las que no van a aportar nada. Esto podra verse en los siguientes apartados.**	**RF**	**Los mismos**
0.8203	05/01/2019 - 20:44:20	303	0.94	0.811	Vuelvo a hacer el sistema por votos pero ahora con el fichero mejorado.	LGM RF	Los mismos
0.8200	05/01/2019 - 22:27:10	303	0.93	0.811	Imputamos ademas amount_tsh	RF	Los mismos

Tabla 1: Tabla comparativa modelos y subidas a DrivenData

3. Proceso seguido

3.1. Descripcion

Para esta práctica he tenido dos fases principales. **La primera de ellas** basada mas en un aspecto práctico, utilizando funciones de Python y viendo los resultados que obtenía. La segunda parte la realicé por el mal funcionamiento de ésta primera ya que no conseguía unos resultados muy finos. **La segunda fase** está basada en un estudio más profundo de los datos y así una mayor comprensión del problema. Con este método conseguí mejorar el modelo y obtener el resultado final conseguido. No obstante, **me gustaría comentar que por cuenta propia, en mi cuenta personal de DrivenData, he seguido con el problema y he de remarcar que utilizando el mismo fichero con el que obtuve el resultado de 0.8214 pero ejecutandolo de nuevo y con ello asumo que otra semilla, conseguí un 0.8239**. Una lástima que cuando lo subí con la cuenta de la UGR me hizo ese resultado ya que subí 200 puestos ejecutando el mismo fichero sin hacer ninguna modificación, con la cual conseguí la posicion 140.

Aunque explicaré ambas fases, me centraré más en la segunda ya que es realmente la interesante ya que como he comentado la primera fué mas para relacionarme con este sistema de DrivenData y comprobar en qué consistía el problema mientras que la segunda fase se centra concretamente en el problema.

3.2. Primera fase: funciones de python

1. En primer lugar utilicé el fichero de ejemplo que encontramos en la web de la asignatura[1] el cual realiza los siguientes procesos.
   - En primer lugar elimina las variables 'id' y 'date_recorded' por lo que estas variables no serán utilizadas.
   - En segundo lugar convirte todas las variables tanto numericas como categoricas a ordinales con lo siguiente:

```
'''
Se convierten las variables a variables numéricas (ordinales)
'''
from sklearn.preprocessing import LabelEncoder
mask = data_x.isnull()
data_x_tmp = data_x.fillna(9999)
data_x_tmp = data_x_tmp.astype(str).apply(LabelEncoder().fit_transform)
data_x_nan = data_x_tmp.where(~mask, data_x)

#máscara para luego recuperar los NaN
mask = data_x_tst.isnull()
#LabelEncoder no funciona con NaN, se asigna un valor no usado
data_x_tmp = data_x_tst.fillna(9999)
#se convierten categóricas en numéricas
```

```
data_x_tmp = data_x_tmp.astype(str).apply(LabelEncoder().fit_transform)
#se recuperan los NaN
data_x_tst_nan = data_x_tmp.where(~mask, data_x_tst)
```

- Con esto, ejecutamos el modelo con los datos obtenidos.

2. Partiendo de esto, quise ajustar más e intentar eliminar variables que tienen un alto nivel de correlación con otras variables. También eliminé variables que no aportan nada debido a sus valores ya que son iguales para cualquier elemento del conjunto de datos.

   - Elimino a parte de las anteriores, 'recorded_by' y 'num_private'.
   - Pasamos todas las variables a numéricas ordinales como en el caso anterior.
   - Eliminamos variables por correlación utilizando:

```
                '''
Eliminacion de variables por matriz de correlacion
'''
corr_matrix = data_x_nan.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
 ↪  k=1).astype(np.bool))
umbral = 0.9
to_drop = [column for column in upper.columns if any(upper[column] >
 ↪  umbral) or any(upper[column] < -1.0 * umbral)]
data_x_nan = data_x_nan.drop(to_drop, axis=1)
data_x_tst_nan = data_x_tst_nan.drop(to_drop, axis=1)
```

   - Por último se ejecuta el modelo utilizando los datos generados nuevos.

   Este cambio me hizo bajar un poco, lo que consideré que las variables que estaba elimando podrían ser de suma importancia, aunque también podría ser un cambio minúsculo por la semilla del algoritmo. Por ello, decidí seguir utilizando este fichero.

3. Una de las cosas que me fije que no estaba correcta era que pasaba tambien las variables numericas a ordinales cuando esto no es esctictamente necesario. En este paso lo que hice fué separar las variables categoricas por un lado y las numéricas por otro. También imputo las variables categóricas a la mas comun y las numericas con la media.

   - Al igual que en los ejemplos anteriores eliminamos las mismas variables.
   - En este caso, en vez de pasar a ordinales como en el caso anterior, primero separamos entre los string y los numericos y aplicamos una imputacion muy simple:

```
'''
Imputacion
'''
#Train
strings = [col for col in data_x.columns if data_x[col].dtype=='object']
```

```
df_strings =
    ↪ pd.DataFrame(data_x[strings],index=data_x.index,columns=strings)
numbers = np.setdiff1d(data_x.columns.values,np.array(strings))
df_numbers = pd.DataFrame(data_x[numbers], index=data_x.index,
    ↪ columns=numbers)
#Creamos los modelos de imputacion y los entrenamos
imps = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
impn = SimpleImputer(missing_values=0, strategy='mean')
imps.fit(df_strings)
impn.fit(df_numbers)
df_strings =
    ↪ pd.DataFrame(imps.transform(df_strings),index=df_strings.index,columns=df_stri
df_numbers =
    ↪ pd.DataFrame(impn.transform(df_numbers),index=df_numbers.index,columns=df_numb
#Transoformamos las categoricas a ordinales
df_strings = df_strings.apply(LabelEncoder().fit_transform)

#Creamos los datos volviendo a unir las categoricas con las numericas
data_x = pd.DataFrame.join(df_strings,df_numbers)

#Test
strings = [col for col in data_x_tst.columns if
    ↪ data_x_tst[col].dtype=='object']
df_strings =
    ↪ pd.DataFrame(data_x_tst[strings],index=data_x_tst.index,columns=strings)
numbers = np.setdiff1d(data_x_tst.columns.values,np.array(strings))
df_numbers =
    ↪ pd.DataFrame(data_x_tst[numbers],index=data_x_tst.index,columns=numbers)
#Imputamos en test utilizando los valores de Train para así no mirar
    ↪ Test
df_strings =
    ↪ pd.DataFrame(imps.transform(df_strings),index=df_strings.index,columns=df_stri
df_numbers =
    ↪ pd.DataFrame(impn.transform(df_numbers),index=df_numbers.index,columns=df_numb
#Transoformamos las categoricas a ordinales
df_strings = df_strings.apply(LabelEncoder().fit_transform)

#Creamos los datos volviendo a unir las categoricas con las numericas
data_x_tst = pd.DataFrame.join(df_strings,df_numbers)
```

- Una vez tenemos estos datos, volvemos a ejecutar el modelo

Con esta modificación conseguí una mejora significativa, lo que nos dice que imputando los valores hemos conseguido hacer un ajuste más fino y por lo tanto un mejor modelo.

4. Llegado a este punto, pensé que la forma de trbajar con las variables categóricas no era la correcta, ya que les está dando una numeración que tiene un orden cuando esto en las

variables categóricas no tiene porque serlo. Asique decidí utilizar la funcion "get_dummy()" para pasar algunas de las variables categóricas a varias variables. Esto lo hago a las variables categóricas que tienen menos de 10 niveles.

- Se sigue el mismo proceso que en el caso anterior.
- Lo que cambia en la imputación respecto al anterior es que ya no convertimos las categóricas a ordinales porque esto se encargará el nuevo método que he creado. Por lo que una vez imputadas y eliminadas las variables por correlacion, se llama al método para que de las variables que quedan, haga "dummy" a las que sea posible.

```
def categoricas_dummy(data_x_p,data_x_tst_p,delete=True):
    data_x = data_x_p.copy()
    data_x_tst = data_x_tst_p.copy()
    # Train
    # Separamos los string de los numeros
    strings = [col for col in data_x.columns if data_x[col].dtype ==
    'object']
    df_strings = pd.DataFrame(data_x[strings], index=data_x.index,
    columns=strings)
    numbers = np.setdiff1d(data_x.columns.values, np.array(strings))
    df_numbers = pd.DataFrame(data_x[numbers], index=data_x.index,
    columns=numbers)

    # Si delete esta activo, se eliminaran las variables que tengan
    NaN.
    if delete:
        drop_columns = []
        for col in df_strings:
            if df_strings[col].isna().sum() > 0:
                df_strings = df_strings.drop(col, axis=1)
                drop_columns.append(col)
        # '''

    # Se aplica dummy a las variables con menos de 10 niveles. Al
    resto se aplica una codificacion ordinal
    df_strings_c = df_strings.copy()
    n = 0
    mod_columns = []
    for col in df_strings.columns:
        if df_strings[col].nunique() < 10:
            aux = pd.get_dummies(df_strings[col])
            index = [str(name) + str(i + n) for i, name in
    enumerate(aux.columns.values)]
            n += len(index)
            aux.columns = index
```

```
30            df_strings_c = pd.DataFrame.join(df_strings_c.drop(col,
    axis=1), aux)
31            mod_columns.append(col)
32        else:
33            mask = df_strings[col].to_frame().isnull()
34            data_x_tmp = df_strings[col].to_frame().fillna(9999)
35            data_x_tmp =
    data_x_tmp.astype(str).apply(LabelEncoder().fit_transform)
36            aux = data_x_tmp.where(~mask, data_x_tmp)
37            df_strings_c[col] = aux.copy()
38
39    df_strings = df_strings_c.copy()
40    # '''
41
42    #Creamos de nuevo el conjunto de datos
43    data_x = pd.DataFrame.join(df_strings, df_numbers)
44
45    # Test
46    # Ahora en test modificamos las mismas columnas que en train
47    strings = [col for col in data_x_tst.columns if
    data_x_tst[col].dtype == 'object']
48    df_strings = pd.DataFrame(data_x_tst[strings],
    index=data_x_tst.index, columns=strings)
49    numbers = np.setdiff1d(data_x_tst.columns.values, np.array(strings))
50    df_numbers = pd.DataFrame(data_x_tst[numbers],
    index=data_x_tst.index, columns=numbers)
51
52    if delete:
53        for col in drop_columns:
54            df_strings = df_strings.drop(col, axis=1)
55
56    df_strings_c = df_strings.copy()
57    n = 0
58    not_columns = np.setdiff1d(df_strings_c.columns.values,
    np.array(mod_columns))
59    for col in mod_columns:
60        aux = pd.get_dummies(df_strings[col])
61        index = [str(name) + str(i + n) for i, name in
    enumerate(aux.columns.values)]
62        n += len(index)
63        aux.columns = index
64        df_strings_c = pd.DataFrame.join(df_strings_c.drop(col, axis=1),
    aux)
65
66    for col in not_columns:
67        mask = df_strings[col].to_frame().isnull()
68        data_x_tmp = df_strings[col].to_frame().fillna(9999)
```

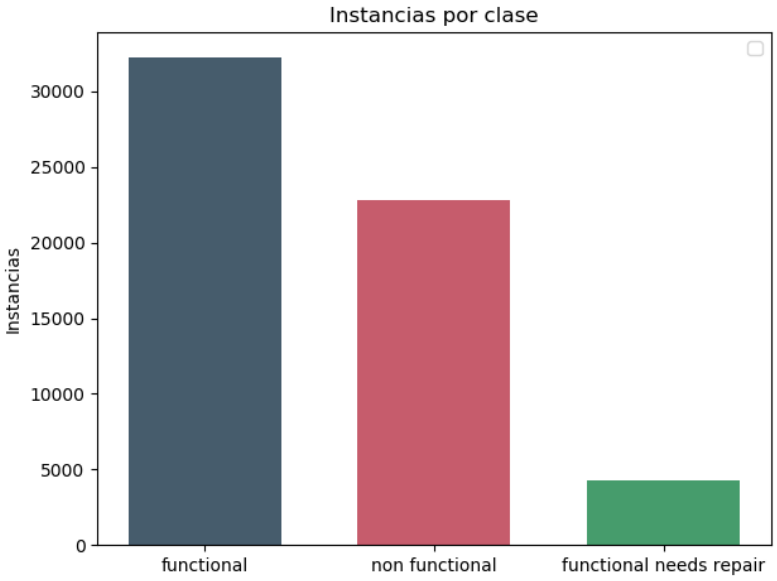
```
        data_x_tmp =
↪ data_x_tmp.astype(str).apply(LabelEncoder().fit_transform)
        aux = data_x_tmp.where(~mask, data_x_tmp)
        df_strings_c[col] = aux.copy()

    df_strings = df_strings_c.copy()

    data_x_tst = pd.DataFrame.join(df_strings, df_numbers)

    return data_x,data_x_tst
```

- Una vez echo esto, se ejecuta el modelo y listo.

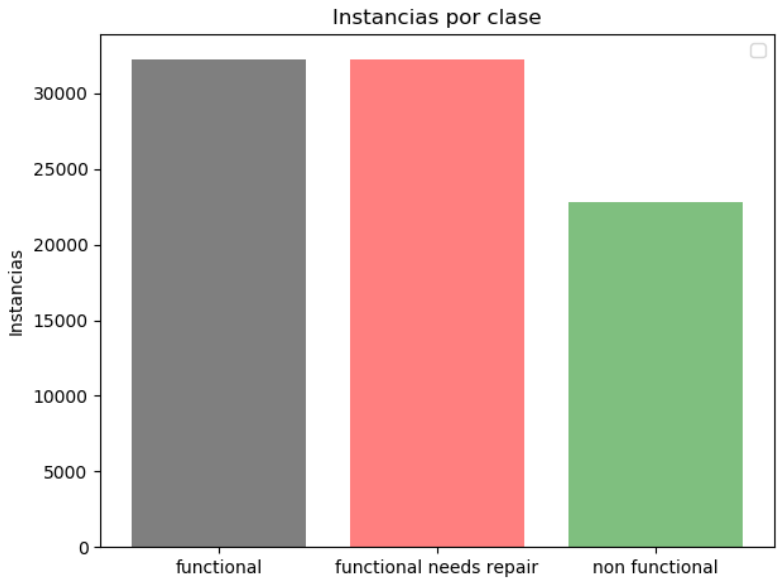
Con esta nueva configuración conseguí una mejora mínima que pudo ser debida incluso a la semilla del algoritmo y no a la modificación del fichero. No obstante, como mejoraba mínimamente, seguí con esta condfiguración

5. Despúes de esto estube cambiando distintas configuraciones, parámetros, eliminando más variables, menos... pero no conseguía mejorar los valores en train ni en validación. Por lo que llegado a este punto decidí pasar el código descrito en el caso anterior, de Python a R. Este cambio fué un poco desastre y los resultados fueron bastante peores, por lo que decidí olvidar este paso y seguir por donde iva, intentando buscar nuevas soluciones.

6. Para este paso, comprobé el balanceo de clases que tenemos en este problema, obteniendo lo siguiente:



Por lo que decidí utilizar el algoritmo SMOTE para hacer balanceo de clases. Realicé un oversample aunque únicamente a la clase minoritaria, obteniendo:



- En primer lugar se sigue el mismo proceso que en los casos anteriores.
- Ahora cuando tenemos ya lo datos procesados hacemos SMOTE para el balanceo de clases
- Por último se ejecuta el algorimo con los datos y listo.

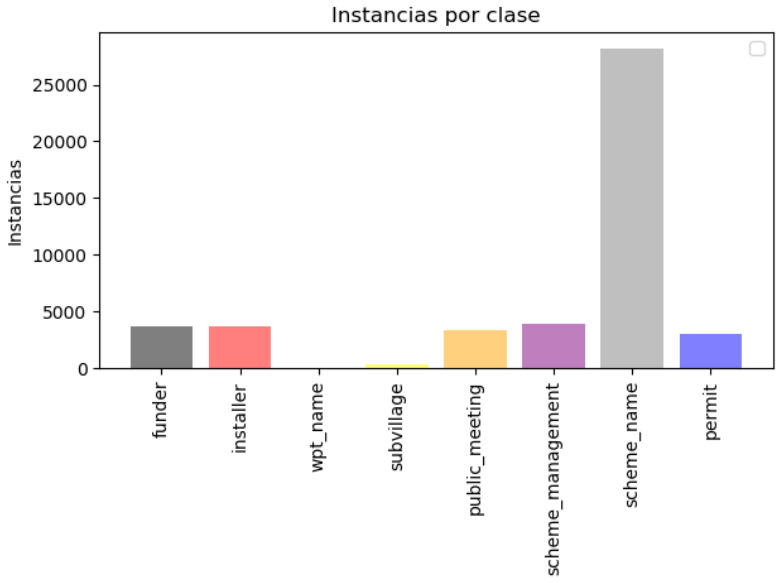
Con esto conseguí mejorar hasta lo que ahora había llegado, mínimamente es cierto, pero conseguí mejorar algo. Este cambio me subió mucho el porcentaje de acierto tanto en train como en validación haciendome pensar al principio que era un cambio muy favorable. No obstante, este cambio se debe a la "copia de elementos". Esta claro que si el algoritmo es capaz de predecir un elemento, si ese elemento esta 50 veces acierta 50 veces ese elemento y hace que el porcentaje de acierto suba cuando es un poco engañoso.

7. Una vez llegado a este punto, despues de probar muchisimas cosas distintas, no conseguí mejorar para nada, es más sólo empeoraba y la cosa iva cuesta abajo. Realicé modificaciones en los algoritmos, en los datos, imputaba con imputaciones simples e incluso llegué en este apartado a imputar con KNN. Si es cierto que el algoritmo que diseñé para imputar con KNN, en pasos posteriores me dí cuenta que no estaba bien programado y por ello supongo que obtuve tan malos resultados.

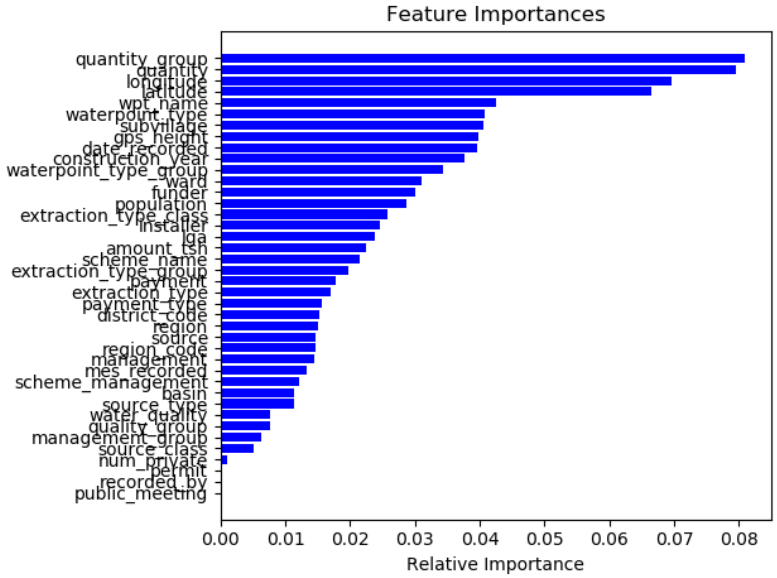
   Como ya no sabía que más hacer y no conseguía nada mejor, estaba un poco frustrado y no sabía cual podía ser el siguiente paso. Mi intuición me decía que el fallo estaba principalmente el los métodos que programé y no en el proceso seguido, por lo que despues de un tiempo, partí desde 0, estudiando el problema con profundidad y intentando entender que estaba pasando. Aquí es donde realmente empecé a competir y a entender el problema.

3.3. Segunda fase: Estudio de datos y comprensión

Uno de los principales problemas en la fase anterior era qué hacía con los valores nulos, si imputarlos, si tomarlos como NaN, si eliminarlos... , para el comienzo de esta fase, estudié cada variable, viendo cuantos elementos NaN tenía, obteniendo lo siguiente:



1. Por lo que viendo esto, en primer lugar decidí tomar los 0 en las variables numéricas como 0 y no como valor nulo. Respecto a las variables categóricas, elimino todas las variables que tienen NaN, por lo que todas las variables que se ven el gráfico serán eliminadas. Las elimino a pesar de que algunas de ellas tienen muy pocos NaN e incluso son buenas para el modelo como se puede ver en el siguiente gráfico donde se usan todas las variables para entrenar a Random Forest.



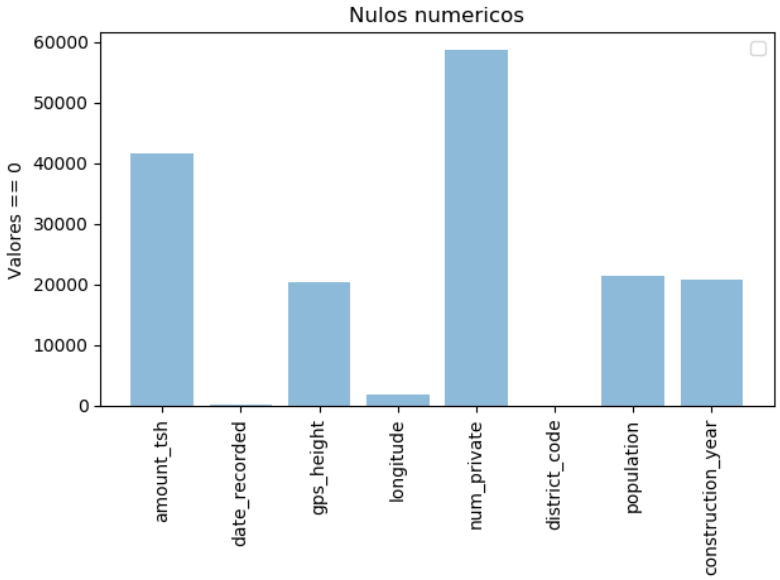
No obstante, como primer acercamiento no era mala idea, ya que ahora todas las instancias que tenemos, tenemos los valores reales de éstas aunque como he comentado tomamos los 0

como un elemento más dentro de las variables numéricas.

También hasta ahora habíamos obviado la variable 'date_recorded' sin ningún tipo de motivo, lo que hice en este caso es cambiar esta variable y desglosarla en 2 variables que sería, una variable que indica únicamente el mes, y otra que indica la diferencia que hay desde el día que se tomó los datos hasta el máximo de estos días, es decir, el más reciente tendrá el valor 0 y el resto de elementos tendrán un valor mas grande a medida que se alejen de tiempo.

Este cambio me hizo un mejora bastante elevada ya que conseguí el 0.8173, que está muy alejado de hasta lo que había conseguido. Esto me hizo ver que eligiendo correctamente las variables y dando los datos que yo pensaba que eran nulos como buenos, se pueden conseguir grandes resultados.

2. Lo que hice ahora es cambiar y creer que los 0 en las variables numericas son nulos. Por lo que utilizando una imputación algo más avanzada como es KNN (arreglado), imputo todos los valores numéricos donde tengamos un 0. Las variables que se van a ver afectadas serían las siguientes:



Este cambio me empeoró un poco, lo que quiere decir que no todos los 0 tienen porque ser nulos, sino que algunos de los 0 que tenemos, son 0 en valores reales y no nulos.

3. En este paso se me ocurrió para mejorar la predicción, utilizar un sistema de votos, utilizar Random Forest como hasta ahora y meterle LGM. Entre los dos se elegiría la etiqueta final dependiendo de el porcentaje que asigna cada algoritmo a cada una de las clases:

```
def prediccion_conjunta_doble(y_pred1,y_pred2):
    d = {0: 'functional', 1: 'functional needs repair', 2: 'non functional'}

    y_pred_tst = []
    for elem1, elem2 in zip(y_pred1, y_pred2):
            valores = elem1 + elem2
            y_pred_tst.append(d[valores.argmax()])
```

```
8
9       return y_pred_tst
```

Aunque pensaba que esta modificación me iva a mejorar el resultado, el LGM no es tan potente como el Random Forest al menos como yo lo tenía configurado y se ve que lo que consigo con esto es bajar el porcentaje de acierto.

4. Viendo los resultados que estaba obteniendo en validacion con los diferentes cambios que iva realizando, me puse a estudiar cada una de las variables independientemente. Con esto, vi la relacion que hay entre las variables ya que tenemos variables que son exactamente iguales para todos los datos, por lo que a la hora de entrenar, estas variables son insignificantes para nosotros. Dicho esto, estube investigandolas todas y llegué a la conclusion de eliminar las siguientes variables:

   - 'recorded_by': Sólo hay que ver los distintos valores que tiene esta variable. Para todos los elementos el valor de la variable es el mismo, por lo que no nos aporta absolutamente nada al modelo. Se puede comprobar en la imagen de las importancias para Random Forest que no aporta nada.
   - 'num_private': Como se puede ver en la imagen anterior, pŕacticamente todos los valores de esta variable tienen un único valor (en este caso 0), por lo que tampoco aportará mucho al modelo. Se puede ver en la imagen de las importancias para Random Forest que no aporta nada.
   - 'public_meeting': Se elimina porque no aporta nada al modelo.
   - 'permit': Se elimina porque no aporta nada al modelo.
   - 'extraction_type_group': En este caso tenemos tres variables ['extraction_type', 'extraction_type_group', 'extraction_type_class'] que son muy similares, sin embargo, una explica mucho, otra se resume un poco más y finalmente la última se resume al máximo. Por lo que decidí dejar la que más explica y la que agrupa más y elimnar la intermedia. Se podría probar a no eliminar esta variable.
   - 'payment_type': Esta variable es exactamente igual a la variable 'payment', por lo que la eliminamos.
   - 'water_quality': Es idéntica a la variable 'quality', por lo que la elimino.
   - 'quantity_group': Esta variable es idéntica a la variable quantity, por lo que la elimino.
   - 'source_type': Esta variable es prácticamente igual a la variable 'source', por lo que la elimino. Al no ser exactamente iguales se podría dejar, pero son tan parecidas que no creo que haya un cambio significativo.
   - 'waterpoint_type_group': Esta variable es muy similar a la variable 'waterpoint_type', solo que ésta está más agrupada, por lo que decidí eliminarla y dejar la que más explica. Como en el caso anterior, podría probarse a dejarla aunque no creo que haya unos cambios muy significativos.
   - 'region': Elimino esta porque tenemos 'region_code' que nos dice lo mismo.

Una vez eliminadas estas variables, (en mi caso antes) se imputan los valores de posición, es decir, 'latitude', 'longitude' y 'gps_height' utilizando KNN:

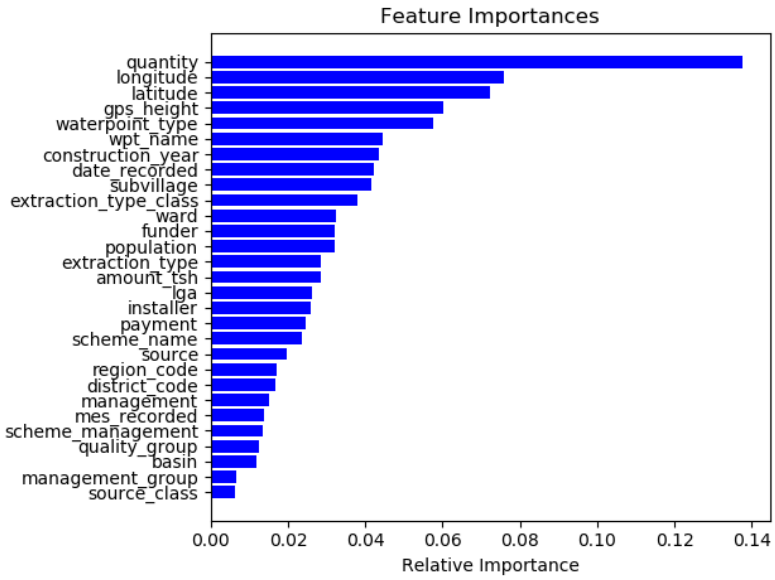
```
def knn_imputation(data_x_p,data_x_tst_p,columns_p,col,K=5,clasificacion=False):
    data_x = data_x_p.copy()
    data_x_tst = data_x_tst_p.copy()
    columns = columns_p.copy()
    # TRAIN

    #COnjunto de entrenamiento
    if clasificacion:
        nulos = data_x.isna()
        index_no_null = nulos.index[nulos[col]==False]
        data_x_n = data_x.loc[index_no_null]
    else:
        data_x_n = data_x.loc[data_x[col] != 0]
    columns.append(col)
    data_x_n = data_x_n[columns]

    aux = data_x_n.isna()
    for colum in aux.columns:
        index = aux.index[aux[colum] == True]
        if len(index) > 0:
            data_x_n = data_x_n.drop(index, axis=0)
            aux = aux.drop(index, axis=0)

    data_y_n = data_x_n[col].values.ravel()
    data_x_n = data_x_n.drop(col, axis=1)

    la = create_labelEncoder(data_x,columns)

    data_x_n = categoricas_ordinales(data_x_n,la)

    norm = MinMaxScaler()
    norm.fit(data_x_n)
    data_x_n = pd.DataFrame(norm.transform(data_x_n), index=data_x_n.index,
    columns=data_x_n.columns)

    # Calculamos los nulos porque los necesitamos
    if clasificacion:
        nulos = data_x.isna()
        index_null = nulos.index[nulos[col] == True]
        nulos = data_x.loc[index_null]
    else:
        nulos = data_x.loc[data_x[col] == 0]
```

```
    nulos = nulos[columns]
    nulos = nulos.drop(col, axis=1)

    #Imputamos los nulos

    nulos = nulos.fillna(9999)

    nulos = categoricas_ordinales(nulos,la)

    if clasificacion:
        neight = KNeighborsClassifier(n_neighbors=K,
    n_jobs=-1,weights='distance',algorithm='brute')
        neight.fit(data_x_n, data_y_n)
    else:
        neight = KNeighborsRegressor(n_neighbors=K,
    n_jobs=-1,weights='distance',algorithm='brute')
        neight.fit(data_x_n, data_y_n)

    nulos = pd.DataFrame(norm.transform(nulos), index=nulos.index,
    columns=nulos.columns)

    labels = neight.predict(nulos)

    data_x.iloc[nulos.index, data_x.columns.get_loc(col)] = labels

    # Test

    if clasificacion:
        nulos = data_x_tst.isna()
        index_null = nulos.index[nulos[col] == True]
        nulos = data_x_tst.loc[index_null]
    else:
        nulos = data_x_tst.loc[data_x_tst[col] == 0]
    nulos = nulos[columns]
    nulos = nulos.drop(col, axis=1)


    if len(nulos > 0):

        nulos = nulos.fillna(9999)

        nulos = categoricas_ordinales(nulos,la)

        nulos = pd.DataFrame(norm.transform(nulos), index=nulos.index,
    columns=nulos.columns)

        labels = neight.predict(nulos)
```

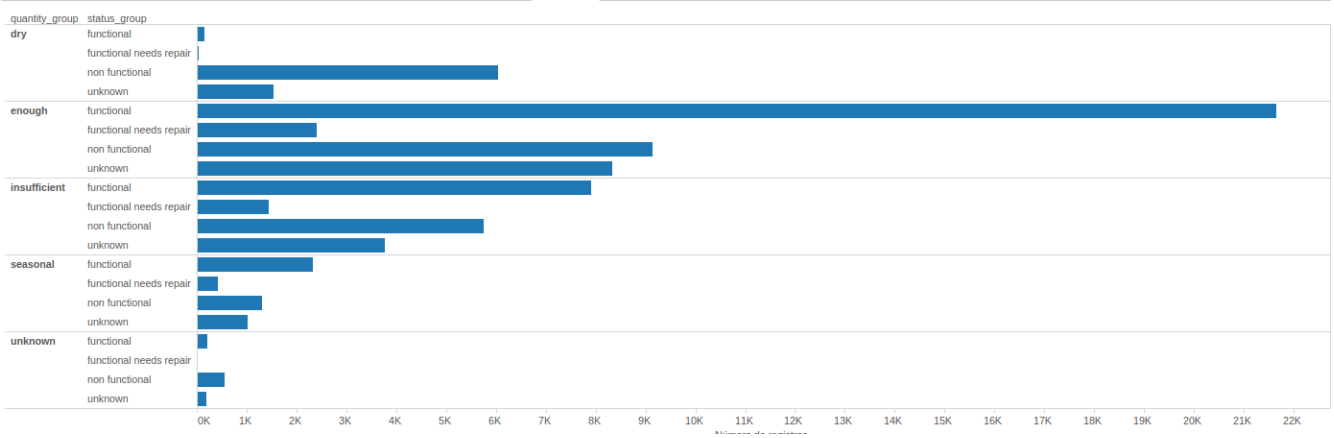
```
85
86          data_x_tst.iloc[nulos.index, data_x_tst.columns.get_loc(col)] =
    ↪ labels
87
88      return data_x,data_x_tst
```

Por último, se pasan las variables categoricas a variables ordinales como he hecho anteriormente, donde tomabamos los NaN como una categoria nueva. Una vez echo esto se ejecuta Random Forest con los datos nuevos y obtenemos el mejor resultado obtenido para mi en este tramo que es **0.8214**. Como comenté al principio, en mi cuenta personal volví a subir este fichero ejecutando de nuevo el algoritmo con los mismos parámetros y debido a la semilla conseguí hacer un 0.8240, un lástima que no pasó cuando realmente importaba para la asignatura.

Finalmente ahora Random Forest utiliza las variables que quedan que realmente son las necesarias, obteniendo:

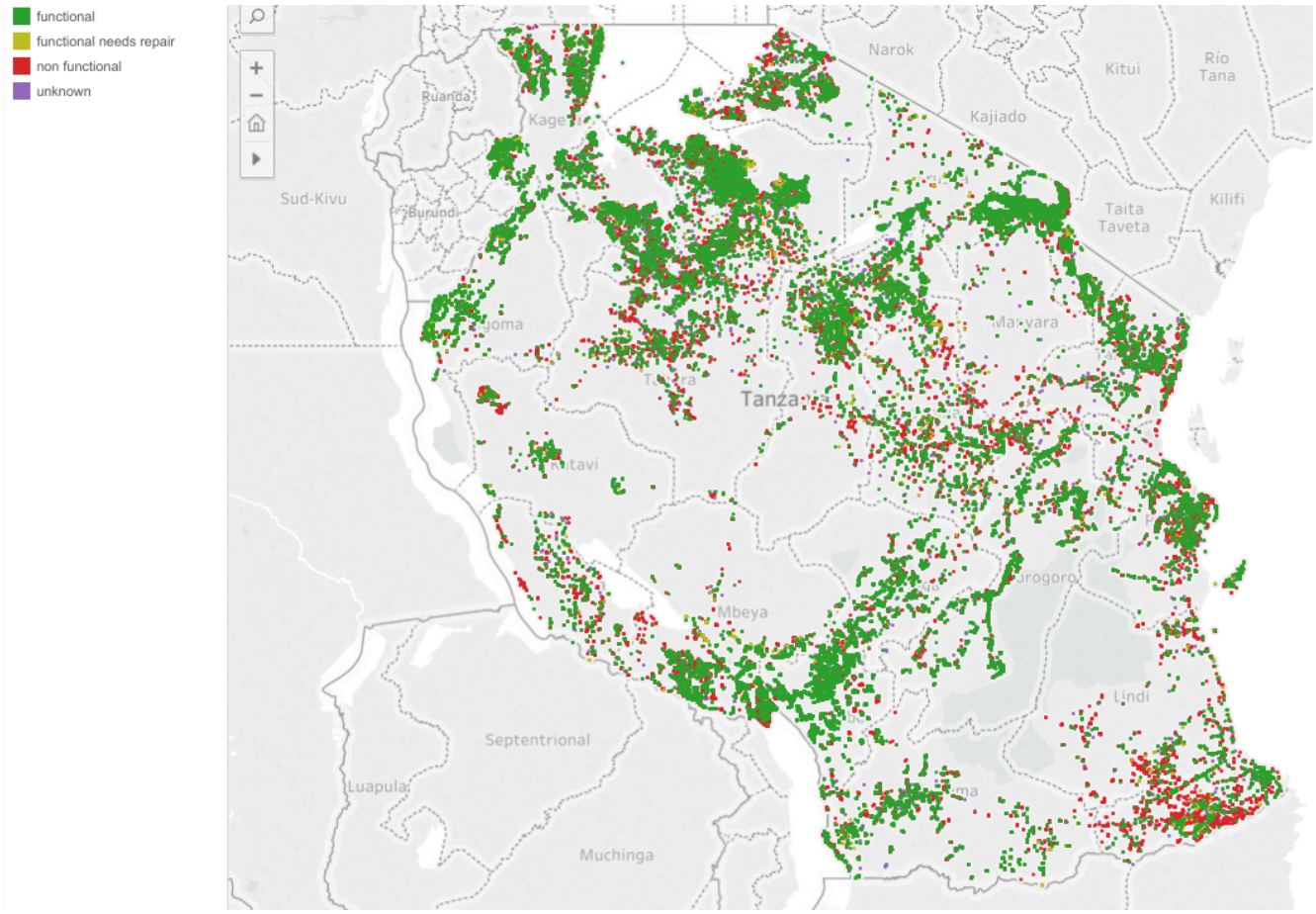


Podemos ver las variables más importantes en la zona superior, esto se puede comprobar, por ejemplo la variable 'quantity', es determinante ya que tenemos:



Podemos apreciar como por ejemplo cuando esta variable tiene el valor “dry”, la mayoría de los elementos son **no funcionales**, mientras que si la variable tiene el valor de “enough”, la mayoría de los elementos son **funcionales**. Es por ello que para Random Forest es tan impotante esta variable.

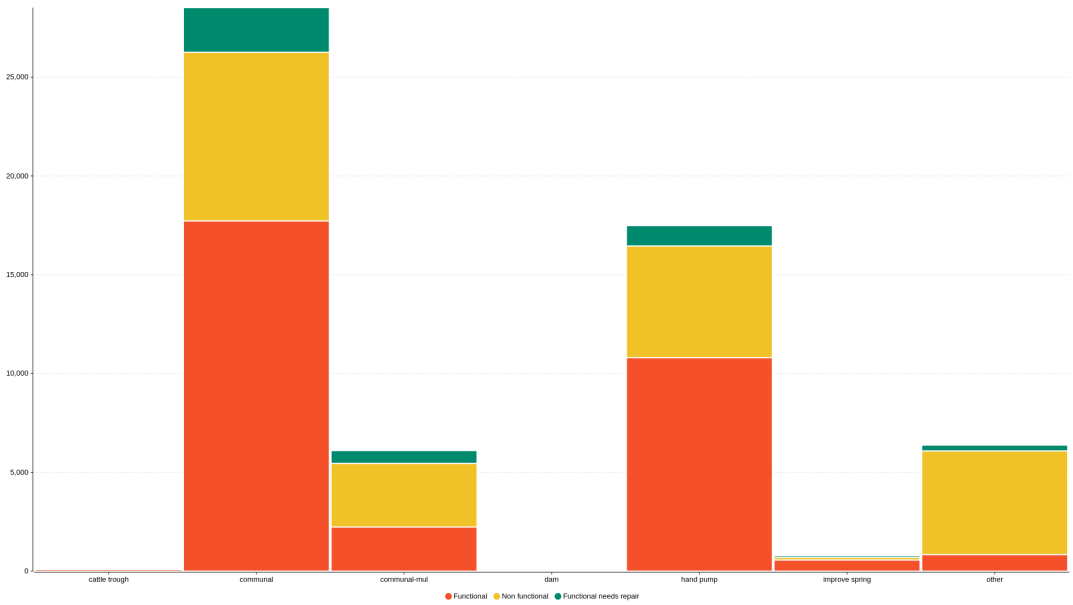
Por otra parte, esta claro y es natural que la posición del pozo es muy interesante a la hora de decidir si la clase a la que pertenece.



Por ello, la latitud y la longitud son variables tan importantes para el algoritmo. Es por ello que una correcta imputación de estas variables puede ayudar a mejorar el resultado considerablemente.

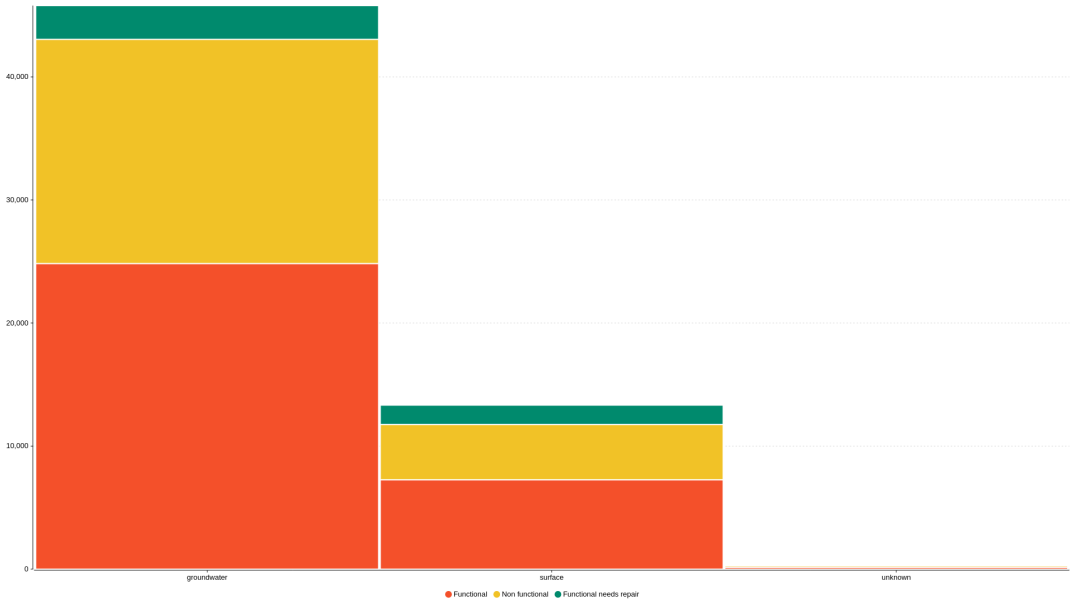
Comentar que la forma de imputar de KNN normal utilizando la distancia no sería realmente la correcta ya que estamos trabajando con variables categóricas y al codificarlas, tenemos por ejemplo Albacete como 1 y Barcelona como 2, sin embargo si queremos sacar las coordenadas de Madrid, por empezar por M no tiene porque estar mas cerca de Barcelona que de Albacete y perdemos un poco la lógica real. Esto hace que el KNN haga una aproximación bastante buena pero que no es realmente la correcta. Si definiesemos por ejemplo la distancia como el número de elementos que coindicen en todas las variables de posición, podría ser una forma de mejorar esta predicción.

Otro ejemplo puede ser la vriable ‘waterpoint_type’ que tambien es bastante decisiva para el algoritmo:



Como podemos ver, tenemos características que nos pueden ayudar a definir la clase bastante bien, por ejemplo en el caso de que la variable tenga el valor de 'improve_spring', generalmente será funcional, o que la variable tenga el valor de 'other', generalmente será 'non functional'.

Por otra parte, al contrario del resto, tenemos variables como 'source_class' que no tiene mucha importancia para el algoritmo y esto lo podemos ver claramente en la siguiente imagen:



Que podemos ver como tenemos una proporcion similar para cada tipo lo que no nos permite por sí misma ayudar al modelo a decidir la clase. Esto no quiere decir que en combinación con otras variables no sea potente, pero sí individualmente podemos decir que esta variable no aporta mucho al modelo.

5. Por último para las últimas subidas, estube únicamente modificando este fichero, cambiando qué elementos imputar cuales no, obteniendo unos resultados simpre similares pero nunca logré superarlos. También probé como anteriormente a ejecutar una votación entre Random

Forest y LGM para ver si al hacer un consenso entre los dos obtenía mejor predicción pero tampoco tuvo éxito este cambio. Probé incluso a hacer una votación entre Random Forest, LGM y XGB, los 3 a la vez pero tampoco obtuve una mejora a lo que únicamente Random Forest era capaz de predecir.

4. Conclusiones

Finalmente como conclusión clara después de todo el proceso realizado, es que los problemas reales, los problemas de verdad, el 75 % del tiempo se tiene que dedicar al estudio del problema, el estudio de las variables, como se comporta cada una, que nos dice cada una, cuales son las importantes, etc... .Si nos perdemos intentando buscar un modelo a prueba y error no obtendremos ningún buen resultado como es mi caso en la primera fase.

Tampoco es necesario hacer tantas transformaciones a los datos (dependiendo del problema), sólo hay que transformar, imputar o modificar las variables que lo necesiten, por ejemplo en nuestro caso teníamos 'latitude' y 'longitude' que enra (0,0), ese dato sabemos que no es correcto y por lo tanto podemos imputar pero otras variables como 'amount_tsh' en la que muchos de los valores son 0, pero 0 puede ser un valor real y por lo tanto tomar los 0 como NaN puede transformarnos valores reales y modificarlos resultados para mal (lo comprobé que pasaba).

Referencias

[1] Web asignatura, sci2s.ugr.es, `https://sci2s.ugr.es/graduateCourses/in`, Accedido el 10 de enero de 2019.

[2] Sklearn references, scikit-learn.org, `https://scikit-learn.org/stable/modules/classes.html#`, Accedido el 10 de enero de 2019.

[3] Some plots, Calin Uioreanu, public.tableau.com, `https://public.tableau.com/profile/calin.uioreanu#!/vizhome/DataMiningtheWaterTableDrivenData_com/Dashboard1`, Accedido el 10 de enero de 2019.

[4] Driven Data, drivendata.org, `https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/23/`, Accedido el 10 de enero de 2019.