ANEXO 1

- *Manual de Usuario para el Empleo de Scripts Desarrollados en Python para la Identificación de Variaciones Genéticas Relevantes* -

1. JUSTIFICACIÓN Y PLANTEAMIENTO

Como se comenta a lo largo del desarrollo de la memoria, para el tratamiento de una parte de la información analizada, principalmente de la información extraída de la base de datos de LOVD, se han desarrollado programas o *scripts* escritos en el lenguaje de programación Python. La razón por la que se hizo uso de este lenguaje fue su versatilidad a la hora de analizar conjuntos de datos de naturaleza diversa, así como también por la precisión con la que permitía cumplir con los tipos de análisis que se llevaron a cabo (Joshi, 2021).

Este manual pretende explicar la función que realiza cada *script*, así como mostrar una parte del trabajo desempeñado durante el presente proyecto final de grado, con objeto de proporcionar una justificación fundamentada de los resultados obtenidos. Esto también permitirá que cualquier usuario que pretenda extraer información de variaciones haciendo uso de la base de datos de LOVD y de la plataforma OraGenDel, disponga de la información necesaria para utilizar el soporte arquitectónico de *software* requerido para cumplir este objetivo.

Si bien el objetivo principal de estos *scripts* consistió en analizar las variaciones de la BBDD de LOVD, también se desarrollaron otros que permitieron gestionar la información de las BBDD de ClinVar y Ensembl. El motivo por el cual se crearon *scripts* para extraer una información que ya había sido extraída previamente mediante la plataforma OraGenDel fue validar el funcionamiento de los mismos. Por ello, en este manual se muestran los *scripts* realizados para el manejo de estas tres BBDD (ClinVar, Ensembl y LOVD).

Ahora bien, antes de profundizar en la explicación de los *scripts*, a continuación, se muestra cómo se extrajeron los archivos de las respectivas BBDD para, posteriormente, ser analizados.

2. OBTENCIÓN DE LOS FICHEROS CRUDOS

La extracción manual se realiza por medio de la interfaz web de cada repositorio de datos genómicos. Esta interfaz confiere una serie de herramientas de filtrado que permiten al usuario acceder a la información de manera intuitiva, para posteriormente descargar los resultados de la búsqueda en distintos formatos (p.ej., CSV, TSV, JSON, etc.). Esta forma de acceso está orientada a profesionales clínicos o usuarios en general, para consultas específicas o descargas de cantidades pequeñas de datos (Landrum *et al*., 2018; Howe *et al.*, 2021).

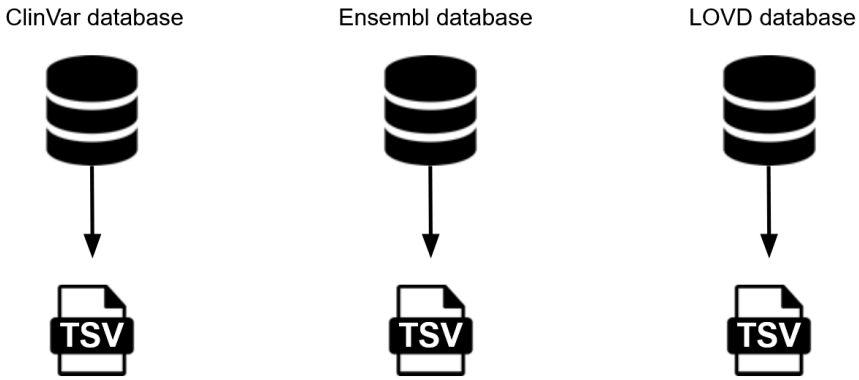

Figura 1A. Representación de la extracción de los ficheros crudos de las BBDD de ClinVar, Ensembl y LOVD.

2.1. ClinVar

En la barra de búsqueda de la interfaz web (*https://www.ncbi.nlm.nih.gov/clinvar/*) se selecciona "Búsqueda avanzada" y se elige la opción "[Disease/Phenotype]" de las opciones del desplegable. Los términos de búsqueda, al ser más de una palabra, se escriben separados por una barra baja: "Duchenne_muscular_dystrophy" y "Becker_muscular_dystrophy".

A continuación se procede a la descarga del archivo con las variaciones, mediante la opción "*Download*", que se encuentra en la parte superior derecha de la página. Al pulsar, se abre un desplegable y se selecciona la opción de formato "*Tabular (text)*" para guardarse el archivo en formato TSV.

2.2. Ensembl

En las opciones de la parte superior de la página (*https://www.ensembl.org*), se selecciona "BioMart", una herramienta web única de esta BBDD que permite la extracción de los datos de manera personalizada mediante la selección de atributos y filtros (Howe *et al.*, 2021).

Se selecciona, en primer lugar, el *data set* del cual se va a obtener la información: *"Ensembl Variation* 104" y *"Human Short Variants* (SNPs *and indels excluding flagged variants*) (GRCh38.p13)".

Del total de filtros, únicamente se emplea uno, para seleccionar el fenotipo "Duchenne Muscular Dystrophy" o "Becker Muscular Dystrophy" En cuanto a los atributos, los cuales hacen referencia a las columnas que se quiere que tenga el fichero,

se seleccionan los siguientes: "*Variant name*", "*Variant source*", "*Chromosome/scaffold name*", "*Chromosome/scaffold position start (bp)*", "*Chromosome/scaffold position end (bp)*", "*Strand*", "*Variant alleles*", "*Associated variant risk allele*", "*Clinical significance*", "*Phenotype name*", "*Phenotype description*", "*P value*", "*Authors*" y "*Year*".

Una vez seleccionados los filtros y los atributos, se exporta en formato TSV.

2.3. LOVD

Esta BBDD no tiene opción de descargar las variaciones. Únicamente se pueden ver en la propia interfaz web de la página (*https://www.lovd.nl/*).

Tras tratar de ponerse en contacto con el responsable de la BBDD para solicitar un archivo con las variaciones, no se obtuvo respuesta. Por esta razón, en base a las recomendaciones de los expertos clínicos, se procedió a copiar las variaciones manualmente en una página de Excel.

El archivo Excel tiene formato XLSX, por lo que, para proceder a su tratamiento posterior, se exporta como archivo con formato TSV.

3. EXPLICACIÓN DE LOS *SCRIPTS*

3.1. Filtrado de los ficheros crudos

Lo primero que se realiza tras la extracción de los archivos, a partir de cada repositorio, es el filtrado de los ficheros crudos o sin tratar. Con este filtrado, se seleccionan las columnas de interés y se eliminan las redundancias que puedan haber. Los resultados de este filtrado se muestran en la **Tabla 1A**.

Tabla 1A. Ficheros de las tres BBDD, antes y después del proceso de filtrado.

BBDD	*Input* / Nº variaciones	Nombre *script*	*Output* / Nº variaciones
Ensembl	Duchenne_Ensembl.tsv / 1796	001_ensembl_D.py	001_Ensembl_filtered_D.tsv / 1425
	Becker_Ensembl.tsv / 137	001_ensembl_B.py	001_Ensembl_filtered_B.tsv / 71
ClinVar	Duchenne_ClinVar.tsv / 2894	002_clinvar_D.py	002_ClinVar_filtered_D.tsv / 2894
	Becker_ClinVar.tsv / 122	002_clinvar_B.py	002_ClinVar_filtered_B.tsv / 122
LOVD	Duchenne_Becker_LOVD.tsv / 30875	003_LOVD_D&B.py	003_LOVD_filtered_D&B.tsv / 7689

En la tabla anterior, se observan cuatro columnas. En la primera, se indica la BBDD de la que proceden los archivos; en la segunda, el nombre del archivo de entrada (o *input*) y el número de variaciones que incluye; en la tercera, el *script* por el que se hace pasar el *input*; y, en la cuarta, el fichero de salida *(output)*.

A continuación, se muestra el *script* “003_LOVD_D&B.py”, dado que se trata del que permite extraer información nueva que no se puede obtener haciendo uso de la plataforma OraGenDel. Los *scripts* “001_ensembl_D.py”, “001_ensembl_B.py”, “002_clinvar_D.py”, “002_clinvar_B.py” no se muestran en el presente manual, dado que realizan la misma función que el que se muestra a continuación pero para las BBDD de Ensembl y ClinVar.

```
# capturar el fichero tsv (Duchenne_Becker_LOVD.tsv)
import sys
tsvfile = sys.argv[1]

# definir la función main
def main(fichero):
    # seleccionar campos y crear diccionario con las significancias clini
cas
    diccionario, diccionario2, outfilee2 = funcion_seleccion(fichero)
    # seleccionar significancia asociada a la variacion
    outfilee3 = funcion_signific(diccionario, diccionario2, outfilee2)
    # eliminar redundancias en base al ID de la BBDD de LOVD
    outfilee4 = funcion_filtrado(outfilee3)

# crear fichero con los campos de interes
def funcion_seleccion(file):
    # crear string para quedarme con los nombres solo una vez
    unique = ''
    # crear diccionario para guardar ID de la BBDD y las distintas signif
icancias clinicas
    # key: identificador
    # value: significancias clinicas
    dic_sign = {}
    # crear diccionario para guardar toda la bibliografia asociada a cada
 variacion
    # key: identificador
    # value: referencias
    dic_biblio = {}
    # abrir fichero nuevo
    outfile2 = open('leiden_split.txt','w')
    # recorrer fichero linea a linea
    n = 0
    firstline = 'a'
    for line in open(file):
        line = line.strip('\n').split('\t')
        # primera linea
        if firstline:
            # escribir en el output las columnas: nombre_dbSNP - Leiden_I
D - Chr - Change - Location - Clinical significance - Effect - Exon - Ref
erence(s)
            outfile2.write('dbSNPname' + '\t' + 'DB-
ID' + '\t' + 'Chr' + '\t' + 'Change' + '\t' + 'Location_start' + '\t' + '
```

```
Location_end' + '\t' + 'Genome (hg38)' + '\t' + 'cDNA (NM_004006.2)' + '\
t' + 'Clinical significance' + '\t' + 'Effect' + '\t' + 'Exon' + '\t' + '
Reference(s)' + '\t' + 'Version' + '\t' + 'Proteina' + '\n')
            firstline = ''
            continue
        # definir conceptos
        identif = line[11]
        if '(recessive)' in line[6] and 'likely' in line[6]:
            signif = 'likely pathogenic'
        elif '(recessive)' in line[6] and 'likely' not in line[6]:
            signif = 'pathogenic'
        else:
            signif = line[6]
        fecha_ref = line[13]
        # añadir significancias clinicas y fecha a cada identificador
        if identif not in dic_sign.keys():
            dic_sign[identif] = []
            dic_biblio[identif] = []
        # escribir en el diccionario
        dic_sign[identif].append(signif)
        dic_biblio[identif].append(fecha_ref)

        # eliminar las repetidas
        if identif not in unique:
            n += 1
            unique = identif
            # de esa linea crear un fichero con nombre - chr (X todas por
que gen = DMD) - cambio - posicion - significancia clinica - referencia -
 version
            # definir campos
            dbSNPname = line[15]
            cDNA = line[2]
            effect = line[0]
            exon = line[1]
            location_change38 = line[8]
            location_change37 = line[7]
            change = ''
            location = ''
            proteina = line[4]


            # extraer la localizacion y el cambio version GRCh38
            if 'g.?' not in location_change38 or '-
' not in location_change38:

                # modificar el campo de posicion (GRCh38)
                for i in range(len(location_change38)):
                    # si es un "indel"
                    u = 0
```

```
                    if 'delins' in location_change38 and '"' not in locat
ion_change38:
                        u += 1
                        if u == 1:
                            # cuando aparezca la "d" de "delins"
                            d = 0
                            if 'd' in location_change38:
                                d += 1
                                if d == 1:
                                    if location_change38[i] == 'd': #or l
ocation_change[i+1] != '[':
                                        change = location_change38[i:]
                                        location = location_change38[2:i]
                    # si hay comillas significa que hay anotados más de u
n tipo de cambio
                    elif 'delins' in location_change38 and '"' in locatio
n_change38:
                        u += 1
                        if u == 1:
                            change = 'delins + (...)'
                    # pero pese a que haya comillas y más de un tipo de c
ambio, este no tiene porqué ser un "indel"
                    elif ']"' in location_change38:
                        u += 1
                        if u == 1:
                            change = 'varios cambios (...)'
                    # si es una insercion
                    elif 'delins' not in location_change38:
                        m = 0
                        if 'ins' in location_change38:
                            m += 1
                            if m == 1:
                                if location_change38[i] == 'i':
                                    change = location_change38[i:]
                                    location = location_change38[2:i]
                        else:
                            change = location_change38[-3:]
                            location = location_change38[2:-3]
                    # si no hay informacion, poner un guion
                    elif '-' in location_change38:
                        change = '-'
                        location = '-'
                    else:
                        change = location_change38[-3:]
                        location = location_change38[2:-3]
                #print(location_change + '   &   ' + change + '   &   ' +
 location) # it works

                # editar campo "location"
```

```
                location_ini = ''
                location_end = ''
                b = 0
                for i in range(len(location)):
                    # si no hay parentesis ni barra baja
                    if '(' not in location and '_' not in location:
                        location_ini = location
                        location_end = location
                    # si no hay parentesis pero si hay barra baja
                    elif '(' not in location and '_' in location:
                        if location[i] == '_':
                            location_ini = location[:i]
                            location_end = location[i+1:]
                    # si hay parentesis
                    if '(' in location and '_' in location:
                        if location[i] == '_':
                            b += 1
                            if b == 1:
                                if location[1:i] == '?':
                                    location_ini = location[i+1:i+9]
                                else:
                                    location_ini = location[1:i]
                            elif b == 3:
                                if location[i+1] == '?':
                                    location_end = location[i-8:i]
                                else:
                                    location_end = location[i+1:-1]
                #print(location + '   &   ' + location_ini + '   &   ' +
location_end) # it works
                # escribir en el output
                outfile2.write(dbSNPname + '\t' + identif + '\t' + 'X' +
'\t' + change + '\t' + location_ini + '\t' + location_end + '\t' + locati
on_change38 + '\t' + cDNA + '\t' + signif + '\t' + effect + '\t' + exon +
 '\t' + fecha_ref + '\tGRCh38\t' + proteina + '\n')

            ######## extraer la localizacion y el cambio version GRCh37
            if ('g.?' in location_change38 or '-
' in location_change38) and ('g.?' not in location_change37 or '-
' not in location_change37):
                # modificar el campo de posicion (GRCh37)
                for i in range(len(location_change37)):
                    # si es un "indel"
                    u = 0
                    if 'delins' in location_change37 and '"' not in locat
ion_change37:
                        u += 1
                        if u == 1:
                            # cuando aparezca la "d" de "delins"
                            d = 0
```

```
                        if 'd' in location_change37:
                            d += 1
                            if d == 1:
                                if location_change37[i] == 'd': #or l
ocation_change[i+1] != '[':
                                    change = location_change37[i:]
                                    location = location_change37[2:i]
                                #else:
                                #    change = 'delins'
                                #    location = location_change[2:]
                    # si hay comillas significa que hay anotados más de u
n tipo de cambio
                    elif 'delins' in location_change37 and '"' in locatio
n_change37:
                        u += 1
                        if u == 1:
                            change = 'delins + (...)'
                    # pero pese a que haya comillas y más de un tipo de c
ambio, este no tiene porqué ser un "indel"
                    elif ']"' in location_change37:
                        u += 1
                        if u == 1:
                            change = 'varios cambios (...)'
                    # si es una insercion
                    elif 'delins' not in location_change37:
                        m = 0
                        if 'ins' in location_change37:
                            m += 1
                            if m == 1:
                                if location_change37[i] == 'i':
                                    change = location_change37[i:]
                                    location = location_change37[2:i]
                        else:
                            change = location_change37[-3:]
                            location = location_change37[2:-3]
                    # si no hay informacion, poner un guion
                    elif '-' in location_change37:
                        change = '-'
                        location = '-'
                    else:
                        change = location_change37[-3:]
                        location = location_change37[2:-3]
                #print(location_change + '   &   ' + change + '   &   ' +
 location) # it works

                # editar campo "location"
                location_ini = ''
                location_end = ''
                b = 0
```

```
                for i in range(len(location)):
                    # si no hay parentesis ni barra baja
                    if '(' not in location and '_' not in location:
                        location_ini = location
                        location_end = location
                    # si no hay parentesis pero si hay barra baja
                    elif '(' not in location and '_' in location:
                        if location[i] == '_':
                            location_ini = location[:i]
                            location_end = location[i+1:]
                    # si hay parentesis
                    if '(' in location and '_' in location:
                        if location[i] == '_':
                            b += 1
                            if b == 1:
                                if location[1:i] == '?':
                                    location_ini = location[i+1:i+9]
                                else:
                                    location_ini = location[1:i]
                            elif b == 3:
                                if location[i+1] == '?':
                                    location_end = location[i-8:i]
                                else:
                                    location_end = location[i+1:-1]
                #print(location + '   &   ' + location_ini + '   &   ' +
location_end) # it works
                # escribir en el output
                outfile2.write(dbSNPname + '\t' + identif + '\t' + 'X' +
'\t' + change + '\t' + location_ini + '\t' + location_end + '\t' + locati
on_change37 + '\t' + cDNA + '\t' + signif + '\t' + effect + '\t' + exon +
 '\t' + fecha_ref + '\tGRCh37\t' + proteina + '\n')

            if ('g.?' in location_change38 or '-
' in location_change38) and ('g.?' in location_change37 or '-
' in location_change37):
                change = '-'
                location_ini = '-'
                location_end = '-'
                outfile2.write(dbSNPname + '\t' + identif + '\t' + 'X' +
'\t' + change + '\t' + location_ini + '\t' + location_end + '\t' + locati
on_change37 + '\t' + cDNA + '\t' + signif + '\t' + effect + '\t' + exon +
 '\t' + fecha_ref + '\tNA\t' + proteina + '\n')
    outfile2.close()

    print(str(n) + ' variaciones unicas')

    return(dic_sign,dic_biblio,outfile2)
```

```
def funcion_signific(dicc,dicc2, output2):
    #print(dic_sign)
    mas = 0
    # diccionario para guardar 1 vez las significancias clinicas distinta
s que hay asociadas a cada variacion
    # key: identificador de la BBDD
    # value: significancias
    dic_uniq = {}
    # diccionario para guardar toda la bibliografia asociada a 1 variacio
n
    # key: identificador de la BBDD
    # value: refs
    dic_bib = {}
    # diccionario para contar cuantas veces ha sido cargada cada variacio
n en la BBDD
    # key: identificador de la BBDD
    # value: nº de veces que aparece
    dic_num = {}
    for key in dicc:
        # contar cuantas veces ha sido cargada en la BBDD la misma variac
ion
        u = 0
        uniq = []
        dic_uniq[key] = ''
        dic_num[key] = ''
        for signifi in dicc[key]:
            u += 1
            #dic_num[signifi] == ''
            # si no se repite la significancia
            if signifi not in uniq:
                uniq.append(signifi)
                # añadir al diccionario cada tipo de significancia solo 1
 vez
                if dic_uniq[key] == '':
                    #dic_num[signifi] == str(m)
                    dic_uniq[key] += signifi
                elif dic_uniq[key] != '':
                    #dic_num[signifi] == str(m)
                    dic_uniq[key] += ', ' + signifi
        dic_num[key] = str(u)
        #print(str(u) + ' elementos, ' + str(len(uniq)) + ' significancia
(s) distinta(s), que son: ')
        if u != 0:
            mas += 1
        #print(key, str(u), dic_sign[key])
    #print(str(mas) + ' variaciones con >1 significancia clinica')
    #print(dic_uniq)
    #print(dic_num)
```

```
    for key in dicc2:
        dic_bib[key] = ''
        unique = []
        for element in dicc2[key]:
            # si no se repite la referencia bibliografica
            if element not in unique:
                unique.append(element)
                # añadir al diccionario cada tipo de significancia solo 1
 vez
                if dic_bib[key] == '':
                    dic_bib[key] += element
                elif dic_bib[key] != '':
                    dic_bib[key] += ';; ' + element
    #print(dic_bib)

    # generar un nuevo output
    outfile3 = open('leiden_split_2.txt','w')
    # abrir el output2
    firstline = 'a'
    # diccionario
    # key: id
    # value: 1er campo localizacion/ version cromosoma?
    dic_version = {}
    for line in open('leiden_split.txt'):
        line = line.strip('\n').split('\t')
        if firstline:
            firstline = ''
            outfile3.write(line[0] + '\t' + line[1] + ' (veces)\t' + line
[2] + '\t' + line[3] + '\t' + line[4] + '\t' + line[5] + '\t' + line[6] +
 '\t' + line[7] + '\t' + line[8] + '\t' + line[9] + '\t' + line[10] + '\t
' + line[11] + '\t' + line[12] + '\t' + line[13] + '\n')
            continue
        # modificar el campo de significancias clinicas, para incluir tod
as las asociadas a una misma variacion
        if '37' in line[12] and len(line[4]) > 5:
            outfile3.write(line[0] + '\t' + line[1] + ' (' + dic_num[line
[1]] + ')\t' + line[2] + '\t' + line[3] + '\t' + line[4] + '\t' + line[5]
 + '\t' + line[6] + '\t' + line[7] + '\t' + dic_uniq[line[1]] + '\t' + li
ne[9] + '\t' + line[10] + '\t' + dic_bib[line[1]] + '\t' + line[12] + '\t
' + line[13] + '\n')
        #elif '38' in line[12] and len(line[4]) < 4:
        #    continue
        elif '38' in line[12] and len(line[4]) > 5:
            outfile3.write(line[0] + '\t' + line[1] + ' (' + dic_num[line
[1]] + ')\t' + line[2] + '\t' + line[3] + '\t' + line[4] + '\t' + line[5]
 + '\t' + line[6] + '\t' + line[7] + '\t' + dic_uniq[line[1]] + '\t' + li
ne[9] + '\t' + line[10] + '\t' + dic_bib[line[1]] + '\t' + line[12] + '\t
' + line[13] + '\n')
        else:
```

```
            print(line[1])

    outfile3.close()
    return(outfile3)

def funcion_filtrado(output3):
    # generar un nuevo output
    outfile4 = open('003_LOVD_filtered_D&B.tsv','w')
    # abrir el output2
    firstline = 'a'
    unico = []
    for line in open('leiden_split_2.txt'):
        line = line.strip('\n').split('\t')
        if firstline:
            firstline = ''
            outfile4.write(line[0] + '\t' + line[1] + '\t' + line[2] + '\
t' + line[3] + '\t' + line[4] + '\t' + line[5] + '\t' + line[6] + '\t' +
line[7] + '\t' + line[8] + '\t' + line[9] + '\t' + line[10] + '\t' + line
[11] + '\t' + line[12] + '\t' + line[13] + '\n')
            continue
        if line[1] not in unico:
            unico.append(line[1])
            outfile4.write(line[0] + '\t' + line[1] + '\t' + line[2] + '\
t' + line[3] + '\t' + line[4] + '\t' + line[5] + '\t' + line[6] + '\t' +
line[7] + '\t' + line[8] + '\t' + line[9] + '\t' + line[10] + '\t' + line
[11] + '\t' + line[12] + '\t' + line[13] + '\n')
    outfile4.close()
    return(outfile4)

main(tsvfile)
```

3.2. Uniformización del nombre

Una vez generados los ficheros que contienen las columnas de interés, y a los cuales se les han eliminado las posibles redundancias que pudiesen haber, se crea una nueva columna para nombrar las variaciones con el nombre que sigue la estructura propuesta por el Oráculo Genómico de Delfos:

'Localización : Cambio (Ensamblaje)'

Los *scripts* desarrollados para ello se denominan:

- 011_OraGenDel name_ensembl.py
- 012_OraGenDel name_clinvar.py
- 013_OraGenDel name_LOVD.py

A estos últimos, se les hizo pasar el *output* generado en el paso anterior, pero esta vez como *input*. El resultado consistió en la obtención de los ficheros con la misma información que antes de pasar por los *scripts*, pero con una nueva columna añadida en la cual se mostraba el nombre de cada variación en formato OraGenDel.

A continuación, se muestra el *script* "013_OraGenDel name_LOVD.py" que se desarrolló con tal fin.

```
# capturar fichero input (003_LOVD_filtered_D&B.tsv)
import sys
tsvfile = sys.argv[1]

def main(fichero):
    # escribir nombre de las variaciones como en OraGenDe (Chr:position:a
lleles)
    delfos_name = funcion_nombre(fichero)

def funcion_nombre(file):
    firstline = 'a'
    # abrir output
    out = open('013_LOVD_name_D&B.tsv','w')
    d = 0
    g = 0
    f = 0
    dif = ''
    for line in open(file):
        line = line.strip('\n').split('\t')
        cambio = ''
        ref = ''
        alt = ''
        oragende = ''
        # escribir linea header
        if firstline:
            out.write('OraGenDe\t' + line[0].replace('dbSNPname','dbSNP_I
D') + '\t' + line[1] + '\t' + line[2] + '\t' + line[3] + '\t' + line[4] +
 '\t' + line[5] + '\t' + line[6] + '\t' + line[7] + '\t' + line[8] + '\t'
```

```
 + line[9] + '\t' + line[10] + '\t' + line[11] + '\t' + line[12] + '\t' +
 line[13] + '\n')
            firstline = ''
            continue
        # campo localización
        if len(line[4]) > 3 and len(line[5]) > 3 and 'AC[15_25]' not in l
ine[5]:
            inicio = line[4]
            final = line[5]
            # calcular diferencia entre localizacion final e inicial
            #print(inicio, final)
            dif = int(final) - int(inicio)
            #print(dif)
            # si la diferencia es > 10 ntds
            if dif > 10 or dif < -10:
                cambio = 'longchange'
                oragende = 'ChrX:g.' + line[4] + '-
' + line[5] + ':' + cambio + '(' + line[12] + ')'
                f += 1
            else:
                cambio = line[3]
                # si hay dos alelos
                if '>' in line[3]:
                    g += 1
                    oragende = 'ChrX:g.' + line[4] + '-
' + line[5] + ':' + cambio + '(' + line[12] + ')'
                elif '>' not in line[3] and '?' not in line[3] and '[' no
t in line[3]:
                    #print(line[3])
                    d += 1
                    oragende = 'ChrX:g.' + line[4] + '-
' + line[5] + ':?>?(' + line[12] + ')'
                else:
                    #f += 1
                    print(line[1], line[3])
                    oragende = '_ChrX:g.' + line[4] + '-
' + line[5] + ':' + cambio + '(' + line[12] + ')'

        out.write(oragende + '\t' + line[0] + '\t' + line[1] + '\t' + lin
e[2] + '\t' + line[3] + '\t' + line[4] + '\t' + line[5] + '\t' + line[6]
+ '\t' + line[7] + '\t' + line[8] + '\t' + line[9] + '\t' + line[10] + '\
t' + line[11] + '\t' + line[12] + '\t' + line[13] + '\n')

    print(g)
    print(d)
    print(f)
    out.close()
    return(out)
main(tsvfile)
```

3.3. Clasificación de las variaciones

La clasificación de las variaciones se llevó a cabo tanto para las variaciones extraídas de Ensembl y ClinVar como para las de LOVD. Dicha clasificación se basó en seleccionar aquellas cuya significancia clínica era "*pathogenic*" o "*likely pathogenic*", debido a que este era uno de los requisitos necesarios para ser aceptadas por el algoritmo de clasificación de la plataforma OraGenDel (lo cual se explica en el **Subapartado 4.3.1**), entre otros requisitos. Cabe destacar que las variaciones que albergaban la significancia "*VUS*" y "*pathogenic*", o "NA" y "*pathogenic*" también fueron incluidas dentro de la clasificación, para no descartar ninguna posibilidad.

Los *scripts* desarrollados para extraer las variaciones relevantes de ClinVar y Ensembl consiguieron obtener un 96.72% de cobertura al compararlas con las extraídas por OraGenDel para BMD y un 99.75% para las de DMD, lo que permitió asegurar la fiabilidad de los *scripts* a la hora de extraer las variaciones para la BBDD de LOVD.

A continuación, se muestra el *script* que se elaboró para realizar dicha clasificación para la BBDD de LOVD, seguido del resultado de la misma, el cual se puede observar en la **Figura 2A**.

```
# capturar fichero input (013_LOVD_name_D&B_.tsv)
import sys
file = sys.argv[1]

# contadores para la clasificacion
n = 0
p = 0
b = 0
g = 0
v = 0
a = 0
u = 0
s = 0
o = 0

# abrir fichero output
out = open('113_LOVD_D&B.tsv','w')
firstline = 'a'
for line in open(file):
    line = line.strip('\n')
    # si es la primera linea
    if firstline:
        firstline = ''
        # escribir la línea header del output
        out.write(line + '\n')
        continue
    # definir conceptos
    line = line.split('\t')
    sign = line[9]
    name = line[0]
```

```
    # escribir en el output aquellas variaciones que sean patogénicas o
probablemente patogenicas
    if ',' in sign:
        o += 1
        if 'pathogenic, likely pathogenic' in sign or 'likely pathogenic,
 pathogenic' in sign:
            s += 1
            out.write(line[0] + '\t' + line[1] + '\t' + line[2] + '\t' +
line[3] + '\t' + line[4] + '\t' + line[5] + '\t' + line[6] + '\t' + line[
7] + '\t' + line[8] + '\t' + line[9] + '\t' + line[10] + '\t' + line[11]
+ '\t' + line[12] + '\t' + line[13] + '\t' + line[14] + '\n')
        if 'pathogenic' in sign and 'benign' not in sign:
            s += 1

    if ',' not in sign:
        if 'pathogenic' in sign and 'likely pathogenic' not in sign:
            n += 1
            out.write(line[0] + '\t' + line[1] + '\t' + line[2] + '\t' +
line[3] + '\t' + line[4] + '\t' + line[5] + '\t' + line[6] + '\t' + line[
7] + '\t' + line[8] + '\t' + line[9] + '\t' + line[10] + '\t' + line[11]
+ '\t' + line[12] + '\t' + line[13] + '\t' + line[14] + '\n')
        if 'likely pathogenic' in sign:
            p += 1
            out.write(line[0] + '\t' + line[1] + '\t' + line[2] + '\t' +
line[3] + '\t' + line[4] + '\t' + line[5] + '\t' + line[6] + '\t' + line[
7] + '\t' + line[8] + '\t' + line[9] + '\t' + line[10] + '\t' + line[11]
+ '\t' + line[12] + '\t' + line[13] + '\t' + line[14] + '\n')
        if 'benign' in sign and 'likely benign' not in sign:
            b += 1
        if 'likely benign' in sign:
            g += 1
        if 'VUS' in sign:
            v += 1
        if 'NA' in sign:
            a += 1
        if 'unclassified' in sign:
            u += 1
# sacar por pantalla el numero total de cada tipo de variaciones atendiendo
a su clasificación
print(str(o) + ' variaciones con más de una interpretación')
print(str(n) + ' pathogenic')
print(str(p) + ' likely pathogenic')
print(str(b) + ' benign')
print(str(g) + ' likely benign')
print(str(v) + ' VUS')
print(str(a) + ' NA')
print(str(u) + ' unclassified')
print(str(s) + ' con >1 significancia pero patogénicas principalmente')
```

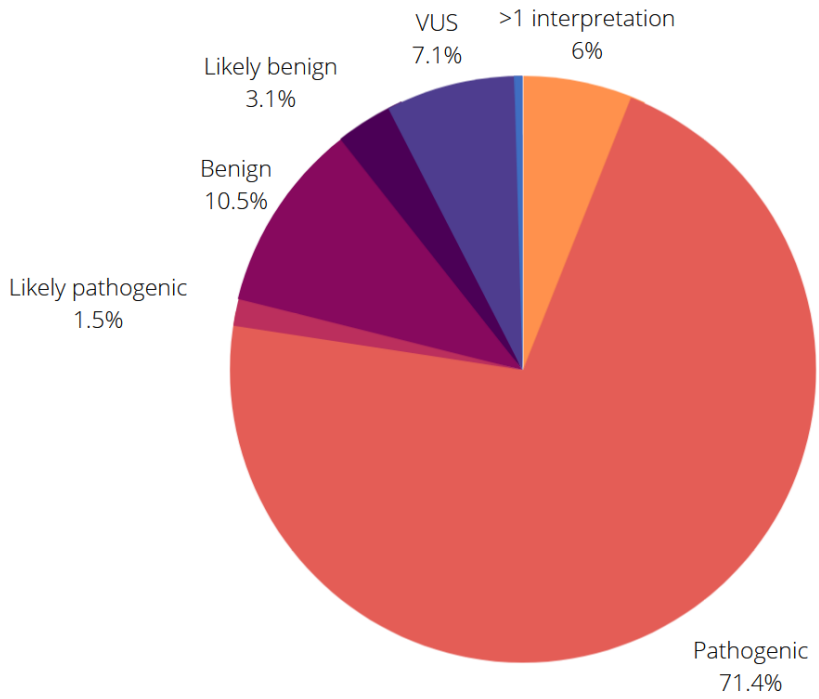

Figura 2A. Clasificación de las variaciones de LOVD según su significancia clínica.

El resultado de la clasificación que se muestra en la Figura 2A consistió en 4970 variaciones "Pathogenic" (71.4%), 103 variaciones "Likely pahogenic" (1.5%), 714 variaciones "Benign" (10.5%), 484 variaciones "VUS" (7.1%), 410 variaciones con más de una interpretación (6%), 210 variaciones "Likely benign" (3.1%) y 32 variaciones "NA"/"Unclassified" (<1%).

3.4. Otros análisis

Por último, una vez analizadas las características y el tipo de variaciones de la BBDD de LOVD, y habiendo determinado su importancia en base al bajo ratio de cobertura que presentaba esta última frente a las BBDD de Ensembl y ClinVar, se plantearon una serie de estudios con tal de determinar el significado biológico que podía extraerse del estudio de las variaciones.

Para llevar a cabo dichas determinaciones, se estudiaron las variaciones atendiendo al tipo de cambio y a la localización. El siguiente *script* analiza los tipos de cambios de las variaciones clasificadas como relevantes para la DMD y para la BMD. Los resultados de este estudio se muestran en el **Subapartado 5.4.1**.

```
# capturar archivo input (variaciones relevantes para DMD/BMD)
import sys
file = sys.argv[1]
# contador del tipo de cambio asociado a cada variacion
totalsnv = 0
total = 0
snv = 0
ter = 0
miss = 0
```

```
fs = 0
dup = 0
dele = 0
microsat = 0
indel = 0
loss = 0
ins = 0
inv = 0
interr = 0
cor = 0

firstline = 'a'
for line in open(file):
    line = line.strip('\n').split('\t')
    # definir las variables
    cambio = line[1]
    proteina = line[17]
    if firstline:
        firstline = ''
        continue
    total += 1
    if 'Indel' in cambio:
        indel += 1
    if 'single nucleotide variant' in cambio:
        snv += 1
        totalsnv += 1
        # analizar si se trata de una alteracion de tipo nonsense o
missense
        if 'Ter' in proteina:
            ter += 1
        if '=' in proteina:
            miss += 1
        if '=' not in proteina and 'fs' not in proteina and 'Ter' not in
proteina:
            miss += 1
    if 'Deletion' in cambio:
        dele += 1
    if 'Duplication' in cambio:
        dup += 1
    if 'Microsatellite' in cambio:
        microsat += 1
    if 'loss' in cambio:
        loss += 1
    if 'Insertion' in cambio:
        ins += 1
    if 'inv' in cambio:
        inv += 1
    if cambio == '?':
        interr += 1
```

```
    if '[' in cambio or ']' in cambio:
        cor += 1
    # ver cuantas variaciones del total afectan a la pauta de lectura
    if 'fs' in proteina:
        fs += 1

print(totalsnv)

# sacar por pantalla los resultados de la clasificación según el tipo de
cambio, también su porcentaje con respecto al total de variaciones
print(str(total) + ' variaciones en total (' + str(round(total/total*100,
2)) + ')')
print(str(snv) + ' SNPs (' + str(round(snv/total*100,2)) + '), que se cla
sifican en:')
print(str(round(totalsnv/totalsnv*100,2)))
print(str(ter) + ' nonsense (' + str(round(ter/totalsnv*100,2)) + ')')
print(str(miss) + ' missense (' + str(round(miss/totalsnv*100,2)) + ')')

print(str(dup) + ' duplications (' + str(round(dup/total*100,2)) + ')')
print(str(dele) + ' deletions (' + str(round(dele/total*100,2)) + ')')
print(str(microsat) + ' microsatellites (' + str(round(microsat/total*100
,2)) + ')')
print(str(indel) + ' indels (' + str(round(indel/total*100,2)) + ')')
print(str(loss) + ' copy number loss (' + str(round(loss/total*100,2)) +
')')
print(str(ins) + ' insertions (' + str(round(ins/total*100,2)) + ')')
print(str(inv) + ' inversions (' + str(round(inv/total*100,2)) + ')')
print(str(interr) + ' ? (' + str(round(interr/total*100,2)) + ')')
print(str(cor) + ' corchete (' + str(round(cor/total*100,2)) + ')\n')
print(str(fs) + ' variaciones que desplazan el marco de lectura o framesh
ift variations (' + str(round(fs/total*100,2)) + ')')
```

El siguiente *script*, por último, se centra en determinar la localización de las variaciones a lo largo de la secuencia genómica del gen. Los resultados de dicho análisis se muestran en las gráficas del **Subapartado 5.4.2**.

```
# capturar fichero input
import sys
exonslocation = sys.argv[1] # archivo con las localizaciones de los
exones
variationsfile = sys.argv[2] # fichero con variaciones de DMD/BMD

def main(exones,fichero):
    # crear diccionario con los exones y su localización
    diccionario = funcion_dicc(exones)
    # asociar cada variacion con su localizacion en el genoma
    dic1,dic2 = funcion_ubicar(diccionario,fichero)
```

```
    # contar cuantas variaciones hay asociadas a cada localización
    dic_contador = funcion_contar(dic1,dic2)

def funcion_dicc(exonsfile):
    # diccionario
    # key: exon
    # value: localizacion inicial y final
    dic_exons = {}
    # abrir el fichero con los exones y sus posiciones genómicas
    firstline = 'a'
    for line in open(exonsfile):
        line = line.strip('\n').split('\t')
        if firstline:
            firstline = ''
            continue
        nexon = line[0]
        ini = line[3]
        fin = line[4]
        dic_exons[nexon] = ini, fin
    #print(dic_exons)
    return(dic_exons)

def funcion_ubicar(diccio,variaciones):
    # crear diccionario con los exones y las variaciones que se asocian a
 cada uno
    # key: exones
    # value: variaciones
    dic_exones = {}
    # asignar las llaves del diccionario
    for i in range(79):
        dic_exones[i+1] = []
    # crear diccionario con los intrones y las variaciones que se asocian
 a cada uno
    # key: intrones
    # value: variaciones
    dic_intrones = {}
    # asignar las llaves del diccionario
    for j in range(78):
        dic_intrones[j+1] = []

    firstline = 'a'
    # abrir fichero con las variaciones
    for line in open(variaciones):
        line = line.strip('\n').split('\t')
        if firstline:
            firstline = ''
            continue
        inicial = line[0][7:15]
        final = line[0][16:24]
```

```
        #print(final)
        # abrir diccionario
        stop = 'a'
        stop2 = 'a'
        n = 0
        for exon in diccio:
            #print(exon)
            n += 1
            posiciones = diccio[str(n)]
            # guardar
            pos = ''
            # recorrer las 2 posiciones de cada exon
            t = 0
            p = 0
            for posicion in posiciones:
                p += 1
                # si es la posicion inicial
                if p == 1:
                    # si la localización inicial de la variacion es mayor
 que la posición donde empieza el exón
                    if inicial > posicion:
                        pos = inicial
                        #print(exon,loc)
                        continue
                    if inicial < posicion and stop:
                        #print(dic_intrones)
                        t = 1
                        stop = ''
                        dic_intrones[n].append(line[0])
                if p == 2:
                    # si la localización inicial de la variacion es menor
 que la posicion final
                    if pos > posicion:
                        continue
                    if pos < posicion and stop2:
                        stop2 = ''
                        if line[0] not in dic_intrones.values():
                            dic_exones[n].append(line[0])
    print(dic_intrones)
    #print(dic_exones)
    return(dic_intrones,dic_exones)

def funcion_contar(intrones,exones):
    # crear diccionario para contar el numero de variaciones asociado a c
ada intron
    # key: intron
    # value: numero de variaciones asociadas
    dic_counti = {}
```

```
    # crear diccionario para contar el numero de variaciones asociado a c
ada intron
    # key: intron
    # value: numero de variaciones asociadas
    dic_counte = {}
    # asociar variaciones con intrones
    intr = 0
    for key in introns:
        dic_counti[key] = ''
        n = 0
        for variacion in introns[key]:
            intr += 1
            n += 1
            dic_counti[key] = n
    print(dic_counti)
    print(intr)
    # asociar variaciones con exones
    exo = 0
    for key in exones:
        dic_counte[key] = ''
        m = 0
        for variacion in exones[key]:
            exo += 1
            m += 1
            dic_counte[key] = m
    print(dic_counte)
    print(exo)

main(exonslocation,variationsfile)
```

ANEXO 2

- MCGH v.3 -

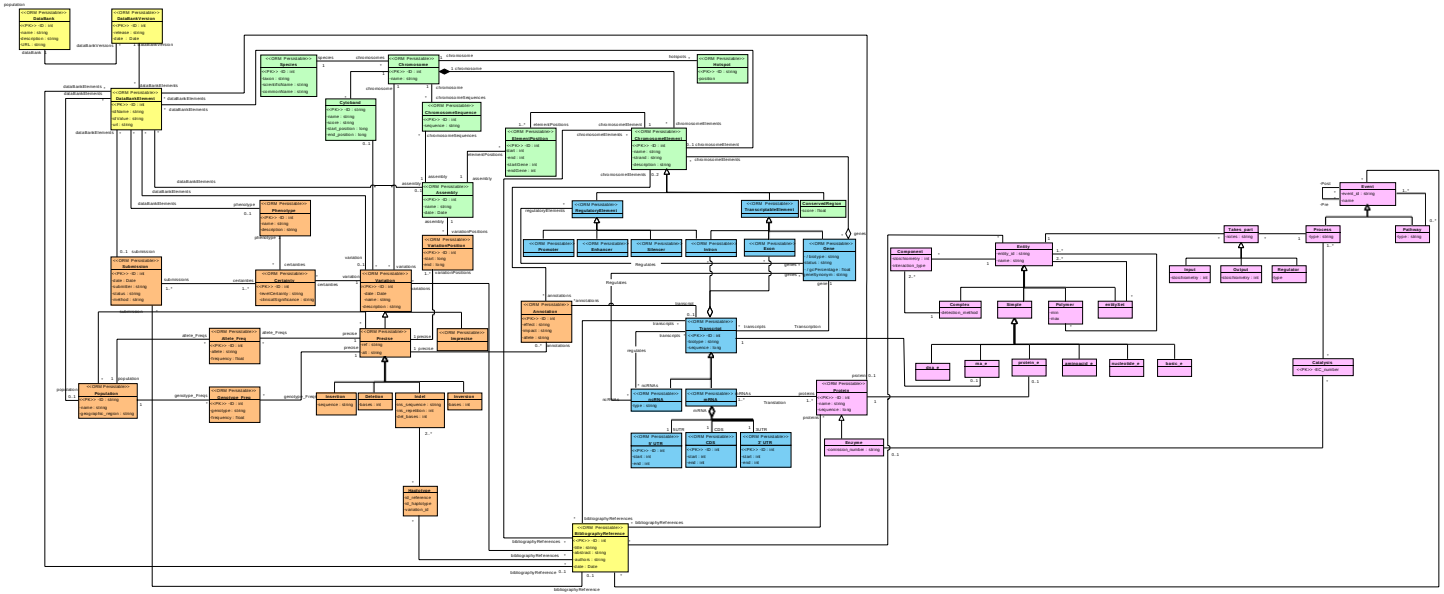

Figura 1B. Modelo Conceptual del Genoma Humano. Fuente: Reyes, 2018.