VCFWeb: Aplicación web para el filtrado de variantes en formato VCF y diseño de estrategias para su priorización e integración con distintos tipos de datos.

Alba Martín Lázaro
Máster Universitario Bioinformática y Bioestadística
Programación para la Bioinformática.

Consultor UOC: Brian Jiménez García
Consultores externos: Jorge de la Barrera Martínez, Fátima Sánchez Cabo
Profesor/a responsable de la asignatura: Maria Jesús Marco Galindo

Fecha de entrega: 5 de junio de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*VCFWeb: Aplicación web para el filtrado de variantes en formato VCF y diseño de estrategias para su priorización e integración con distintos tipos de datos.*
Nombre del autor:	*Alba Martín Lázaro*
Nombre del consultor/a:	*UOC: Brian Jiménez García* *Consultores externos: Jorge de la Barrera Martínez, Fátima Sánchez Cabo*
Nombre del PRA:	*María Jesús Marco Galindo*
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	Máster Universitario Bioinformática y Bioestadística
Área del Trabajo Final:	*TFM "Ad-hoc"* *Tema propuesto por el alumno.*
Idioma del trabajo:	*Castellano*
Palabras clave	*VCF (Variant Call Format); variant filtering; Web-based tool*

Resumen del Trabajo (máximo 250 palabras):

Las nuevas estrategias de secuenciación del ADN, como (NGS, Next-Generation Sequencing [1]) permiten a los genetistas identificar rápidamente las variaciones genéticas en muchos genomas humanos. Sin embargo, ser capaz de aislar unas pocas variantes que puedan ser causales de enfermedad, sigue siendo un desafío importante para la genética médica.

El formato de archivo estándar para contener llamadas de variante es el archivo VCF (variant call format [2]). Este formato, no es fácilmente manejable, ya que contiene información muy diversa y resulta complicado representar las variantes de forma plana. Además se trabaja con grandes volúmenes de datos, y los recursos son limitados.

Mediante la elaboración de un parseador de ficheros en formato VCF, obtenido a partir de las lecturas en crudo de diversas secuencias, se genera una estructura de datos que es explotada desde una herramienta web, que se desarrolla también como parte del proyecto.

Como resultado se obtiene una aplicación web intuitiva y eficiente para los investigadores del CNIC (Centro Nacional de Investigaciones Cardiovasculares),

para intentar reducir al máximo posible el número de variantes tras pasar por los procesos de filtrado y priorización.

Dependiendo del tipo de enfermedad, se llegará a unas pocas variantes candidatas potencialmente causales de enfermedad, y esto permitirá, en ocasiones, tomar medidas de prevención antes de que aparezcan los primeros síntomas, pudiendo en algunos casos evitar o reducir considerablemente la enfermedad.

Abstract (in English, 250 words or less):

New DNA sequencing strategies, such as (NGS, Next-Generation Sequencing [1]) allow geneticists to quickly identify genetic variations in many human genomes. However, being able to isolate a few variants that may be causative of disease continues to be a major challenge for medical genetics.

The standard file format for containing variant calls is the VCF file (variant call format [2]). This format is not easily manageable, since it contains very diverse information and it is difficult to represent the variants in a flat way. Furthermore, we work with large volumes of data, and resources are limited.

Through the elaboration of a VCF file parser, obtained from the raw reads of sequences of different characteristics, a data structure is generated and it is exploited from a web tool, which is also developed as part of the project.

As a result, an intuitive and efficient web application for CNIC researchers (National Center for Cardiovascular Research) has been obtained, as an attempt to reduce the number of variants as much as possible after passing through the filtering and prioritizing processes.

The researcher will obtain, depending on the type of disease, a few candidate variants potentially causative of illness, and this will allow, on occasions, to take preventive measures before the first symptoms appear, being able in some cases to avoid or considerably reduce the disease.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

1.1.1. Contexto

Gracias a las últimas tecnologías de secuenciación del ADN, hoy en día es posible identificar rápidamente las variantes genéticas presentes en muchos genomas humanos.

La técnica de secuenciación del genoma entero (whole-genome sequencing, WGS) ha permitido identificar variantes estructurales que no eran detectables mediante otras técnicas, como biochips CGH/SNP o secuenciación del exoma (whole-exome sequencing, WES), por su reducido tamaño o por su localización en regiones no codificantes.

Sin embargo, el sistema desarrollado en este proyecto no funcionaria muy bien con datos WGS: tanto por el volumen (millones de lecturas) como porque fuera del exoma la interpretación es difirente. Este sistema funciona de manera óptima con exoma (WES) o paneles.

A pesar de ello, conseguir aislar variantes genéticas que puedan causar alguna enfermedad, sigue siendo todo un desafío para el mundo de la genética.

El procesamiento de los datos de secuenciación implica Preprocesado, llamada a variante (genotipado), anotación y filtrado.

1.1.2. Problema a resolver

Una de las áreas de investigación de la Unidad de Bioinformática del CNIC (Centro Nacional de Investigaciones Cardiovasculares) es el análisis de datos de ADN en humanos, con especial énfasis en cardiopatías familiares.

En esta área se ha implementado un pipeline de análisis basado en GATK y una batería de QC, para determinar la calidad de las muestras de exoma o genoma completo. La principal complicación de este tipo de análisis es la identificación de variantes con un posible efecto causal.

En este contexto se propone a la alumna la creación de un front-end para el filtrado de variantes con la intención de priorizar las variantes causales de enfermedad.

Además del desarrollo de la parte web, el proyecto recoge todas las etapas previas de obtención de los datos, desde las lecturas en crudo, hasta la generación del fichero con formato VCF.

El formato VCF (variant call format), utilizado para almacenar la información de las variantes, no es fácilmente manejable, ya que se intenta codificar de manera plana información muy diversa y representar las variantes de esta forma no resulta sencillo. Además, se trabaja con grandes volúmenes de datos y los recursos de tiempo son limitados.

Actualmente, en el CNIC, el filtrado de variantes se realiza mediante scripts y se da al usuario como fichero CSV.

A partir del fichero VCF empieza la parte de desarrollo del proyecto, que consiste en parsearlo y almacenar y procesar correctamente la información encontrada en el mismo. Posteriormente, desde la aplicación web desarrollada, se diseñan y se aplican una serie de estrategias para el filtrado y la priorización de variantes. Se analizan muestras reales de pacientes tanto de fuentes de datos públicas como privadas.

1.1.3. Resultados esperados

Se estudian diferentes muestras procedentes de distintas lecturas en crudo de secuencias de ADN. Posteriormente se muestra a través del front-end, cómo mediante la aplicación de distintos filtros y ordenaciones de campos, se llega a unas pocas variantes que son causantes de enfermedad.

Más concretamente, se divide el análisis bioinformático en 2 partes:

- Variant-calling: ejecución del pipeline de análisis bioinformático para la llamada a variantes germinales, utilizando como datos de entrada lecturas correspondientes al *pilot genome Reference Material 8398* (muestra NA12878) obtenidos desde IGSR (The International Genome Sample Resource[3]).
- Priorización de variantes usando el interfaz web. Los datos utilizados son los de un paciente que padece el síndrome de Schinzel Giedion[4], una enfermedad genética “de novo” pontencialmente mortal, provocada por una mutación en las células sexuales de los padres. A pesar de ser una enfermedad hereditaria, no se ve en la población porque los individuos que la padecen no llegan a la edad adulta.

  A pesar de que este trabajo se desarrolla dentro del área de análisis de datos de ADN en humanos, con especial énfasis en cardiopatías familiares, al ser datos privados, no podían incluirse en esta memoria, y se ha tenido que buscar un caso público.

1.1.4. Motivación

La motivación para la realización de este trabajo reside en el interés de la alumna tanto en el análisis de datos de ADN, como en la utilización de tecnologías desconocidas para ella hasta el momento, como Web2py[5], para el desarrollo de la aplicación web. Ambos campos tanto combinados como por separado pueden ser, además, de mucha utilidad en el ámbito laboral.

1.2 Objetivos del Trabajo

Los objetivos generales de este trabajo se pueden dividir en tres grandes bloques, y dentro de cada uno de ellos se proponen una serie de objetivos específicos:

1. Entender el proceso de obtención del fichero de variantes en formato VCF, partiendo de las lecturas en crudo de ADN.
    a. Documentarse sobre el estado del arte de llamada a variantes y familiarizarse con el formato el formato VCF

2. Saber tratar ficheros VCF y almacenar su información en base de datos.
    a. Parsear el fichero VCF
    b. Diseñar base de datos relacional (diagrama entidad-relación)
    c. Crear base de datos relacional
    d. Insertar datos en base de datos para almacenar la información que se ha parseado previamente a partir del fichero VCF
    e. Hacer pruebas funcionales con varios ficheros

3. Desarrollar una aplicación web que permita filtrar y priorizar variantes candidatas a partir de un fichero VCF almacenado en base de datos.
    a. Diseñar estrategias para la priorización de variantes y la integración con diversas bases de datos públicas.
    b. Instalar los paquetes necesarios para el desarrollo de la aplicación web.
    c. Diseñar y crear una base de datos relacional para almacenar la información a partir de un fichero VCF
    d. Diseñar y desarrollar plantillas para visualización de datos en el front-end.
    e. Integrar la creación e inserción de datos en la base de datos (modo batch) con la aplicación web (modo online).
    f. Implementar e integrar en la aplicación las estrategias de filtrado y priorización de variantes.
    g. Exportar los datos a fichero .csv
    h. Probar el correcto funcionamiento de la aplicación (que no se cuelgue, que no se pierda la sesión del usuario, etc)
    i. Comprobar el correcto funcionamiento de todo el proceso (desde los datos iniales, pasando por la generación del fichero

VCF, el filtrado y priorización de variantes y la exportación de resultados en formato CSV).

1.3 Enfoque y método seguido

En este proyecto se realizará un análisis “in silico” de datos genómicos procedentes de las lecturas en crudo obtenidas mediante NGS aplicado sobre WES (Whole-Exome Sequencing[6]).

Se describirán de forma teórica los pasos para obtener un fichero VCF con anotaciones, que será el objeto de partida del desarrollo.

Este fichero recogerá la información de las variantes de diferentes muestras. Se almacenará la información de este fichero en una estructura de datos creada específicamente para ello, de forma que resulte rápido y sencillo acceder a los datos.

La idea inicial era almacenar todo en una misma estructura o tabla pero se descartó porque se pensó que podía no ser eficiente y escalable ya que 100K variantes podían convertirse fácilmente en 600K registros. Para un exoma de pocos pacientes podía ser viable (si se optimizaba muy bien), pero para el genoma completo (unos 3,5 millones de variantes por persona) de 300 personas (un proyecto actual del CNIC) no sería viable.

Además, para no perjudicar a los tiempos de espera de la aplicación web, y para que cada usuario solo tuviera acceso a sus datos, el proceso de almacenamiento del fichero VCF se decidió hacer en modo Batch, integrándose posteriormente con la aplicación. Esto permitiría mostrar al usuario los datos de una forma inmediata y con un formato visual e intuitivo, ahorrándose el parseo del VCF en tiempo real, que es un proceso costoso, ya que posee mucha casuística.

Respecto al diseño de la aplicación web se optó por utilizar Web2py, por ser una herramienta moderna y sencilla para desarrollar aplicaciones web.

La lógica de la aplicación se ha desarrollado principalmente en dos ventanas: “variants” y “filters”. En ambas se muestran los datos en forma de tabla utilizando jQuery Datatables[7]. En un primer momento se pensó en hacer una única ventana principal, “variants”, que contendría toda la lógica de la aplicación, pero finalmente se decidió hacer dos ventanas principales, accesibles una desde la otra: la primera para visualizar los datos más relevantes y hacer búsquedas sencillas o exportar los datos, y la segunda donde poder elegir qué datos mostrar y donde poder aplicar las técnicas de filtrado y priorización de las variantes.

En los siguientes capítulos de la memoria se dará más detalle sobre las diferentes pantallas de la aplicación.

El diseño ha ido sufiendo alteraciones a medida que se ha ido desarrollando el proyecto, pero la funcionalidad no se ha visto afectada.

Para comprobar la correcta lectura y almacenamiento de los datos, se han hecho pruebas sencillas con ficheros VCF generados de ejemplo, y con ficheros grandes tanto privados como de dominio público.

En la aplicación se ha comprobado que los datos almacenados se muestran correctamente, paginan bien y se va reduciendo correctamente el número de variantes a medida que se aplican filtros sobre ellos. También se ha comprobado que la aplicación responde bien ante una gran cantidad de datos.

De cara a probar el correcto rendimiento y funcionamiento global de la aplicación, se ha ejecutado todo el proceso indicado en la Figura 1 y se ha probado con diferentes conjuntos de datos.

La aplicación desarrollada tendrá la calidad suficiente para que pueda ser utilizada y, si fuera necesario, ampliada por la Unidad de Bioinformática del CNIC, que será la encargada de mantenerla a posteriori.

Para el desarrollo del front-end se utilizará el lenguaje de programación Python[8], la base de datos PostgreSql [9] y el Framework Web2py, por su sencillez y versatilidad, a la par que potencia.

La Figura 1 muestra el flujo de trabajo desde que se realizan las lecturas en crudo hasta que se priorizan las variantes.

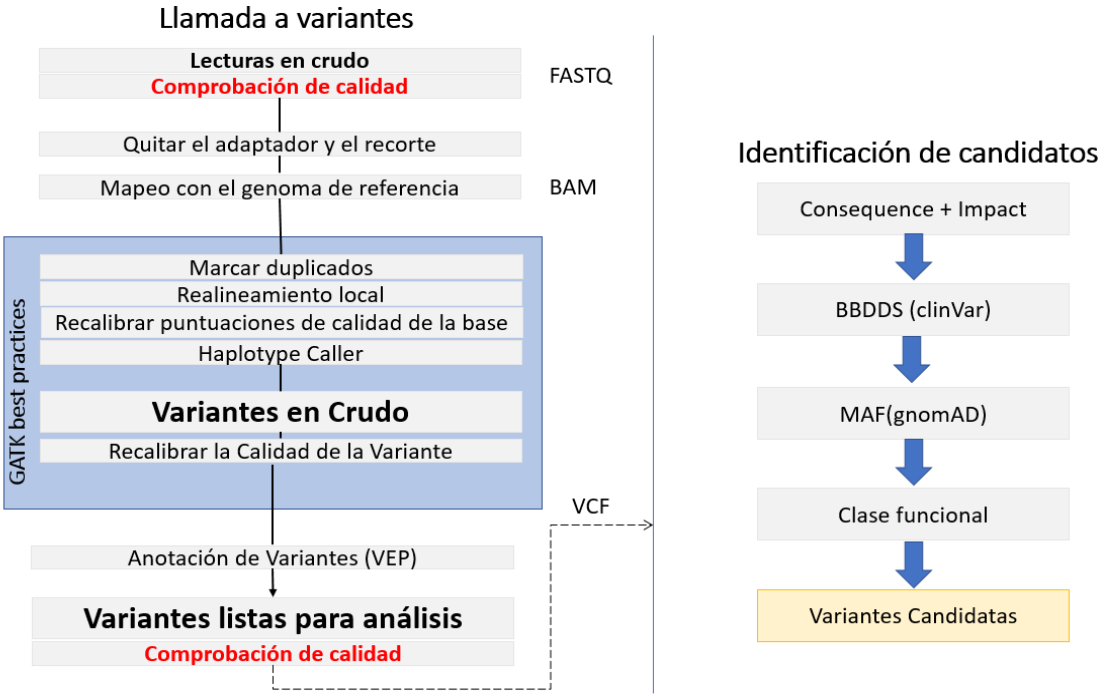


Figura 1. Proceso completo desde la lectura de datos en crudo hasta la identificación de variantes candidatas.

1.4 Planificación del Trabajo

El proyecto se divide en tres partes principales: comprensión del proceso de lectura de variantes y etapas hasta la generación del fichero VCF, almacenamiento de los datos contenidos con el fichero VCF y diseño de estrategias y desarrollo de herramienta web para seleccionar las variantes candidatas. El trabajo concluirá con la presentación de la memoria final, su defensa pública y la autoevaluación del alumno.

El tiempo asignado a cada objetivo se asignó dejando un margen para posibles imprevistos que pudieran provocar retrasos o incumplimientos en el plan de trabajo.

La planificación no ha sufrido grandes cambios de acuerdo a lo que se planteó, pero tampoco se ha seguido de forma meticulosa, ya que se han ido introcuciendo algunas modificaciones, y a veces se ha cambiado el orden de realización de algunas tareas.

1.4.1. Tareas

1. Comprensión del proceso de lectura de variantes y etapas hasta la generación del fichero VCF
    a. Documentarse sobre el estado del arte de llamada a variantes y familiarizarse con el formato VCF -> 15 días
    b. Elección de las distintas herramientas de trabajo (framework, base de datos, lenguaje de programación) -> 2 días
2. Almacenamiento de los datos contenidos con el fichero VCF
    a. Parsear el fichero VCF mediante un script en python -> 1 semana
    b. Diseñar base de datos relacional (diagrama entidad-relación) -> 4 días
    c. Crear base de datos relacional -> 2 días
    d. Insertar datos en base de datos para almacenar la información que se ha parseado previamente a partir del fichero VCF -> 1 día
    e. Hacer pruebas funcionales con varios ficheros -> 3 días

3. Diseño de estrategias y desarrollo de herramienta web para seleccionar las variantes candidatas.
    a. Instalar los paquetes necesarios para el desarrollo de la aplicación web. -> 1 día
    b. Diseñar y desarrollar plantillas para visualización de datos en la aplicación -> 1 semana
    c. Integrar la base de datos ya creada (modo batch) con la aplicación web (modo online) -> 6 días
    d. Empezar a diseñar estrategias de filtrado y priorización de variantes -> 2días

e. Implementar la pantalla de login y registro de nuevos usuarios -> 5 días
f. Implementar e integrar en la aplicación las estrategias de filtrado y priorización de variantes. -> 6 días
g. Mostrar contador de variantes total y parcial -> 1 día
h. Exportar los datos a fichero .csv -> 2 días
i. Probar el correcto funcionamiento de la aplicación -> 2 días
j. Comprobar el correcto funcionamiento de todas las fases del proyecto (desde la lectura de datos inicial, pasando por la generación del fichero VCF, el filtrado y priorización de variantes y la exportación de resultados en formato CSV) con datos de prueba -> 3 días
k. Volver a testear el pipeline ya desarrollado y la aplicación web con datos reales de diferentes individuos y bases de datos (se deja margen por si hubiera que rehacer o corregir alguna parte) -> 1 semana

1.4.2. Calendario

Propuesta TFM “Ad-hoc” y matriculación. 20/02/2018

PEC 0: Definición de los contenidos del trabajo 21/02/2018- 05/03/2018

Inicio de la etapa de formación 21/02/2018

Inicio PEC 1: Plan de trabajo 06/03/2018

Fin PEC 1: Plan de trabajo 19/03/2018

Inicio PEC 2: Desarrollo del trabajo. Fase 1 20/03/2018

Inicio de la etapa de almacenamiento de datos 20/03/2018

Inicio de la etapa de desarrollo web 06/04/2018

Fin PEC 2: Desarrollo del trabajo. Fase 1 23/04/2018

Inicio PEC 3: Desarrollo del trabajo. Fase 2 24/04/2018

Fin de la etapa de desarrollo web 21/05/2018

Fin PEC 3: Desarrollo del trabajo. Fase 2 21/05/2018

PEC 4: Cierre de la Memoria 22/05/2018-05/06/2018

PEC 5: Elaboración de la presentación 06/06/2018-13/06/2018

PEC 6: Defensa pública y autoevaluación 14/06/2018-25/06/2018

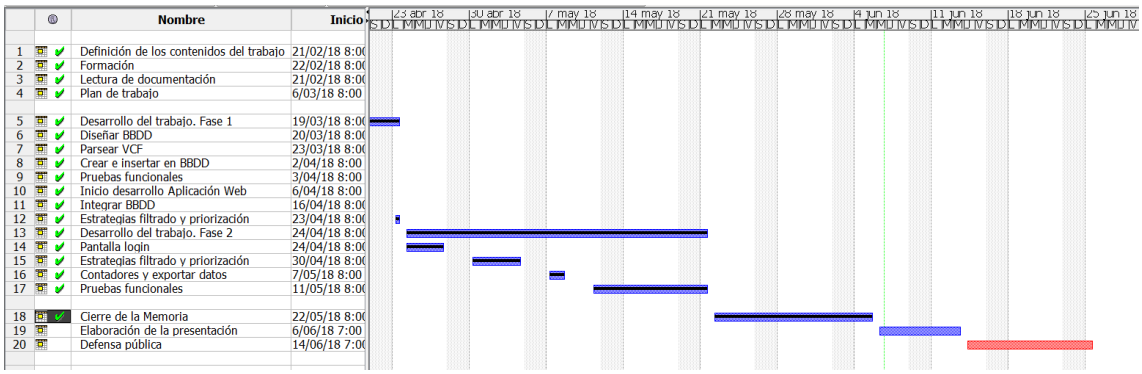


Figura 2. Diagrama que muestra la planificación global de las tareas del proyecto con el tiempo de dedicación previsto para cada una.

1.4.3. Hitos

Propuesta TFM “Ad-hoc” y matriculación 20/02/2018

Inicio PEC 0: Definición de los contenidos del trabajo 21/02/2018-05/03/2018

Inicio de la etapa de formación 21/02/2018

- Lectura de documentación y estado del arte de llamada a variantes [10][11][12][13][14][15][16]
- Planificación global del proyecto

Fin PEC 0: Definición de los contenidos del trabajo 05/03/2018

Fin de la etapa de formación 06/03/2018

Inicio PEC 1: Plan de trabajo 06/03/2018

- Diseño a gran escala del proyecto y consenso con los consultores externos y de la UOC
- Elección de las distintas herramientas de trabajo (framework, base de datos, lenguaje de programación)[14]

Fin PEC 1: Plan de trabajo 19/03/2018

Inicio PEC 2: Desarrollo del trabajo. Fase 1 20/03/2018

Inicio de la etapa de almacenamiento de datos 20/03/2018

- Diseñar base de datos relacional 20/03/2018-22/03/2018
- Parsear el fichero VCF 23/03/2018-30/03/2018
- Crear base de datos relacional 31/03/2018-01/04/2018
- Insertar datos en base de datos para almacenar la información que se ha parseado previamente a partir del fichero VCF 02/04/2018

- Hacer pruebas funcionales con varios ficheros 03/04/2018-05/04/2018

Inicio de la etapa de desarrollo web 06/04/2018

- Instalar los paquetes necesarios para el desarrollo de la aplicación web. 06/04/2018
- Diseñar y desarrollar plantillas para visualización de datos en la aplicación. 07/04/2018-14/04/2018
- Integrar la base de datos ya creada (modo batch) con la aplicación web (modo online). 15/04/2018-21/04/2018
- Empezar a diseñar estrategias de filtrado y priorización de variantes 22/04/2018-23/04/2018

Fin PEC 2: Desarrollo del trabajo. Fase 1 23/04/2018

Inicio PEC 3: Desarrollo del trabajo. Fase 2 24/04/2018

- Implementar la pantalla de login y registro de nuevos usuarios 24/04/2018-29/04/2018
- Implementar e integrar en la aplicación las estrategias de filtrado y priorización de variantes. 30/04/2018-05/05/2018
- Mostrar contador de variantes total y parcial 06/05/2018
- Exportar los datos a fichero .csv 07/05/2018-08/05/2018
- Probar el correcto funcionamiento de la aplicación 09/05/2018-10/05/2018
- Comprobar el correcto funcionamiento de todas las fases del proyecto (desde la lectura de datos inicial, pasando por la generación del fichero VCF, el filtrado y priorización de variantes y la exportación de resultados en formato CSV) con datos de prueba. 11/05/2018-13/05/2018
- Volver a testear el pipeline ya desarrollado y la aplicación web con datos reales de diferentes individuos y bases de datos (se deja margen por si hubiera que rehacer o corregir
- alguna parte). 14/05/2018-21/05/2018

Fin de la etapa de desarrollo web 21/05/2018

Fin PEC 3: Desarrollo del trabajo. Fase 2 21/05/2018

PEC 4: Cierre de la Memoria 22/05/2018-05/06/2018

PEC 5: Elaboración de la presentación 06/06/2018-13/06/2018

PEC 6: Defensa pública y autoevaluación 14/06/2018-25/06/2018

1.4 Breve sumario de productos obtenidos

a. Fichero .txt con las instrucciones a seguir para ejecutar los scripts

b. Ficheros VCF de ejemplo para probar con scripts (se han probado con varios ficheros, pero al contener datos sensibles de pacientes solo se han podido adjuntar aquellos de fuentes públicas o con datos falseados).
c. Scripts en Python (parseador VCF, que genera a su vez un fichero .sql para cargar la base de datos).
d. Plantillas, scripts y capturas de pantalla para la visualización de datos en Web2py.

Estos son los productos que se obtendrán tras la finalización del proyecto:

1.4.1. Memoria Final del trabajo (recoge y amplía los documentos entregados desde la PEC0 a la PEC3).

1.4.2. Productos de desarrollo

- Pipeline para obtener el fichero VCF
- Ficheros VCF de ejemplo para probar con scripts (se han probado con varios ficheros, pero al contener datos sensibles de pacientes solo se han podido adjuntar aquellos de fuentes públicas o con datos falseados).
- Script desarrollado para parsear el fichero VCF
- Script para cargar la base de datos creada a partir de un fichero VCF
- Software desarrollado para crear la aplicación Web (Plantillas, scripts y capturas de pantalla para la visualización de datos en Web2py).
- Fichero CSV para entregárselo al investigador, con las variantes candidatas
- Breve manual de usuario de la aplicación

Todos los ficheros mencionados anteriormente se encuentran subidos en el repositorio GitHub, al que se puede acceder mediante el siguiente enlace:

https://github.com/amartinlla/TFM

1.4.3. Presentación virtual

1.4.4. Autoevaluación del proyecto

Todos estos entregables se elaborarán de acuerdo a las indicaciones y plantillas referidas en el aula.

1.5. Breve descripción de los otros capítulos de la memoria

1.5.1. Datos de partida y justificación

En este apartado se selecciona la enfermedad que va a ser estudiada y que tratará de predecirse mediante la reducción de las variantes de partida a unas pocas candidatas.

1.5.2. Formato de los ficheros de datos iniciales

Se describirán los formatos iniciales de ficheros tratados en el proyecto: FASTQ, BAM.

1.5.3. Proceso de llamada a variante

Se detallarán los pasos desde las lecturas en crudo de las secuencias de ADN hasta la obtención del fichero de variantes con anotaciones, que será el punto de partida del parseador, descrito en el apartado siguiente.

1.5.4. Parser de ficheros VCF

Justificación del lenguaje de programación elegido para desarrollar el parseador, descripción del script paso a paso y llamada al script para ejecutarlo.

1.5.5. Testeo del parser de ficheros VCF

Descripción de las pruebas realizadas al paser de ficheros VCF desarrollado y conjunto de datos utilizados para las mismas.

1.5.6. Estructura de datos

Descripción y justificación de la estructura de datos utilizada para almacenar la información de las variantes.

1.5.7. Diseño, desarrollo y visualización de la aplicación web

Descripción de los pasos y las tecnologías utilizadas para desarrollar la aplicación web, descripción de las distintas pantallas de la aplicación y visualización de los datos en la misma.

1.5.8. Filtrado, priorización y obtención de variantes cantidatas

Se detallan los procedimientos y estrategias de filtrado y priorización utilizados para descartar a la mayoría de variantes.

1.5.9. Exportación de variantes candidatas

Se muestran las variantes potencialmente causales de la enfermedad propuesta en el caso de estudio y se exportan los resultados obtenidos en formato csv para utilizar por el investigador.

1.5.10. Conclusiones

Conclusiones, reflexiones y líneas futuras de trabajo.

1.5.11. Glosario

Definición de los términos y acrónimos más relevantes dentro de la memoria.

1.5.12. Bibliografía

Lista numerada de fuentes utilizadas para el desarrollo de todo el proyecto (libros, artículos de revista y páginas web).

1.5.13. Anexos

Scripts necesarios para la generación del fichero VCF, parseador de ficheros VCF en python y código de las principales pantallas de la aplicación web.

2. Resto de capítulos

2.1. Datos de partida y justificación

Los datos con los que se trabajó inicialmente correspondían a datos privados de pacientes pertenecientes al CNIC, que no pueden ser publicados en esta memoria.
Por ello, se han analizado además otros dos conjuntos de datos públicos: el primero corresponde a datos de Genome in a Bottle[17], un consorcio académico organizado por NIST[18] para desarrollar la infraestructura técnica (estándares de referencia, métodos de referencia y datos de referencia) para permitir la traducción de la secuenciación completa del genoma humano a la práctica clínica. El segundo conjunto de datos pertenece al caso de un paciente que padece el sindrome de Schinzel Giedion.

Utilizando estos dos conjuntos de datos, el análisis bioinformático se dividirá en dos partes:

1) Variant-calling: ejecución del pipeline de análisis bioinformático para la llamada a variantes germinales a partir de datos NGS (Whole Genome Sequence).
Este pipeline se compone de una serie de scripts bash que se ejecutan en un cluster SGE (Son of Grid[19]) siguiendo las recomendaciones de GATK para la llamada a variantes germinales[20].

Como datos de entrada se han utilizado lecturas correspondientes al *pilot genome Reference Material 8398* (muestra NA12878) obtenidos desde IGSR. En concreto, las correspondientes al dataset SRR1517848.

2) Priorización de variantes usando el interfaz web.

Se analiza la muestra de un paciente que padece el sindrome de Schinzel-Giedion. Se trata de una enfermedad mendeliana provocada por una mutación “de novo”. Se caracteriza por dismorfia cráneo-facial específica, anomalías congénitas múltiples y discapacidad intelectual grave.

Los datos utilizados provienen de secuenciación NGS (Whole Exome Sequencing) de ADN.

Esta enfermedad se eligió por varios motivos: el primero, como ya se ha indicado, fue por la necesidad de buscar un conjunto de datos público, ya que los datos que se utilizan internamente en el CNIC no se podían hacer públicos al ser de pacientes.

Por otra parte, atendiendo al tipo de enfermedad, existen varios tipos.
En un extremo se encuentran las enfermedades compleixes, que son aquellas en las que intervienen muchos factores genéticos (variantes) con poco impacto cada uno y factores ambientales. Por sus características, estas enfermedades

son difíciles de estudiar, ya que los investigadores deben identificar y estimar el peso de cada factor. Lo cual no es nada fácil. El filtrado que se puede hacer con la aplicación desarrollada no sirve para este tipo de enfermedades, porque son muchas variantes comunes en la población. Es la combinación de muchas variantes y el ambiente lo que hace que se tenga la enfermedad. Algunos ejemplos de este tipo de enfermedades son la diabetes, la esquizofrenia, el alzhéimer o el asma.
En el otro extremo se encuentran las enfermedades mendelianas, que son causadas por la mutación (presumiblemente una mutación si es dominante o de Novo; o dos mutaciones, si es recesiva) o alteración en la secuencia de ADN de un solo gen. Es decir, las mutaciones tienen mucho efecto y el ambiente prácticamente no tiene influencia. Algunos ejemplos de estas enfermedades son: Fibrosis quística (autosómica recessiva), síndrome de Schinzel-Giedion ("de novo"), Enfermedad de Huntington (autosómica dominante).

Frente a estos extremos: Complejas vs. Mendelianas, hay un espectro intermedio. Algunas cardiopatías familiares están es este punto (pero no podemos usar los datos porque son de pacientes del CNIC).

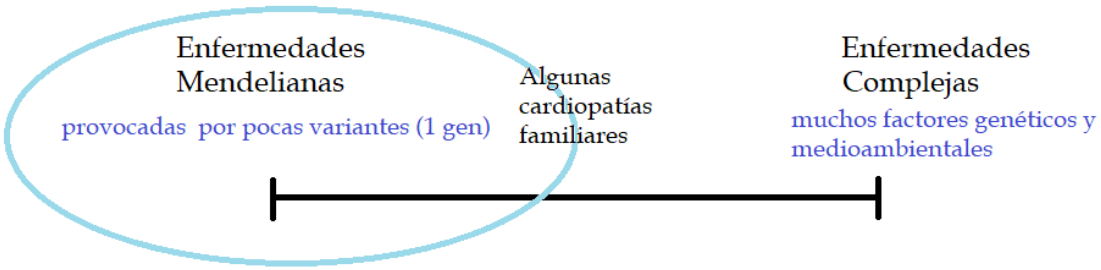


Figura 3. Enfermedades Mendelianas vs. Enfermedades Complejas

En la Figura 3 se ha representado mediante una elipse azul el espectro de enfermedades sobre el que se ha trabajado en el proyecto.

Otro de los motivos por el que se eligió el sindrome de Schinzel-Giedion fue porque, desgraciadamente, los pacientes (niños) no sobreviven por lo que la variante (mutación) tiene mucho efecto (se dice que es muy deletérea) y no esperamos encontrarla en la población (porque los niños no han tenido descendencia y la variante es "de Novo"). Que no esté en la población es, en esencia, la razón por la cual podemos identificarla con facilidad.

Aunque se trate de un caso "real", son datos ilustrativos, ya que se sabe que el gen afectado en este síndrome es SETBP1. Por lo que únicamente habría que ir a mirar ese gen y ya se tendría identificada la enfermedad. Pero la idea es visualizar el procedimiento de priorización en la aplicación web.

En los siguientes apartados se describirá el formato de los datos de partida así como las diferentes etapas hasta la obtención del fichero VCF con anotaciones.

2.2. Formato de los ficheros de datos iniciales

Se parte de un fichero con formato FASTQ[21]. Se trata de un fichero de texto plano que contiene los datos crudos obtenidos por el secuenciador. Dependiendo del secuenciador que se utilice se llegará al fichero FASTQ de diferentes maneras (por ejemplo, con Illumina el proceso de secuenciación es óptica mientras que en Ion Torrent es electroquímica).

En nuestro caso, el fichero origen (público) que se va a utilizar en el proyecto para describir las diferentes etapas hasta conseguir el fichero VCF anotado, proviene del proyecto 1000 genomas (1000 Genomes Project).
Este proyecto se llevó a cabo entre 2008 y 2015, creando el catálogo público más grande de variación humana y datos genotípicos. Cuando el proyecto finalizó, el Centro de Coordinación de Datos de EMBL-EBI recibió fondos continuados del Wellcome Trust para mantener y expandir el recurso. El Recurso de muestra del genoma internacional (IGSR) se creó para hacer esto y tiene como objetivos: asegurar el acceso futuro y la usabilidad de los datos de referencia de 1000 genomas, incorporar datos genómicos adicionales publicados en las muestras de 1000 genomas y ampliar la recopilación de datos para incluir nuevas poblaciones no representadas en el Proyecto 1000 Genomas.

El fichero FASTQ está formado por cuatro tipos de líneas:

1. Título: encabezado siempre por el carácter @. Normalmente es el identificador de la secuencia. Sin límite de longitud.

2. Secuencia biológica, compuesta por la combinación de los cuatro nucleótidos: Adenina (A), Guanina (G), Citosina (C) y Timina (T) y ‘N’, cuando el secuenciador no es capaz de asignar un nucleótido. Los espacios en blanco o tabuladores no están permitidos.

3. Indicador de fin de secuencia e inicio de calidades asociadas: carácter ‘+’. El contenido de este campo es opcional, la línea puede constar únicamente de dicho carácter.

4. Calidades, codificadas por un conjunto de caracteres ASCII (la mayoría ASCII 33-126. Este campo debe tener la misma longitud que la secuencia biológica.

El fichero FASTQ es el fichero de entrada estándar que reconocen las herramientas bioinformáticas encargadas del alineamiento.
Dependiendo de si el tipo de secuenciación es *single end* o *pair end* se tendrá uno o dos ficheros FASTQ respectivamente, uno por cada lectura.

Las lecturas *single end* son más fáciles de alinear con el genoma de referencia, debido al precio, pero en general siempre es preferible tener paired-end.

La secuenciación *paired-end* permite a los usuarios secuenciar ambos extremos de un fragmento y generar datos de secuencia alineables de alta calidad. Esta secuenciación facilita la detección de reordenamientos genómicos y elementos de secuencias repetitivas, así como fusiones de genes y nuevas transcripciones.

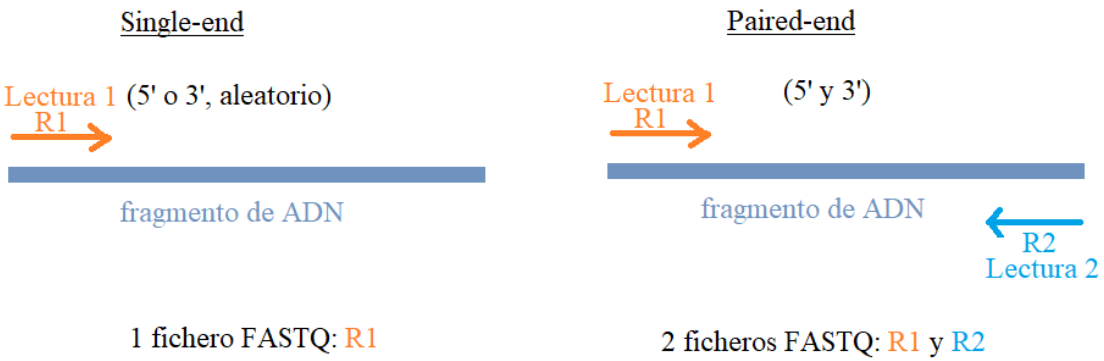


Figura 4. Lecturas Single end vs. Paired-end

En nuestro caso trabajaremos con lecturas *paired-end* y, por lo tanto, partiremos de dos ficheros FASTQ: R1 y R2.

El siguiente tipo de fichero que se verá es BAM[22]. Un fichero BAM (Binary Alignment/Map format) es la versión comprimida del formato SAM que permite realizar un indexado para tener acceso directo a las posiciones genómicas.

Un fichero SAM (Sequence Alignment/Map format) es un fichero de texto tabulado para representar alineamientos de secuencias contra un genoma o secuencia de referencia. Está compuesto por una sección de cabecera (opcional) y una sección de alineamiento.
Las líneas de la cabecera empiezan con el carácter '@' mientras que las del alineamiento no.

En la sección de alineamiento, hay 11 campos obligatorios de información esencial acerca del alineamiento y un número variable de campos opcionales de información más específica.

2.3. Proceso de llamada a variante

En la siguiente figura se muestra el proceso bioinformático que se ha seguido desde las lecturas en crudo de ADN hasta la obtención del fichero VCF con anotaciones.

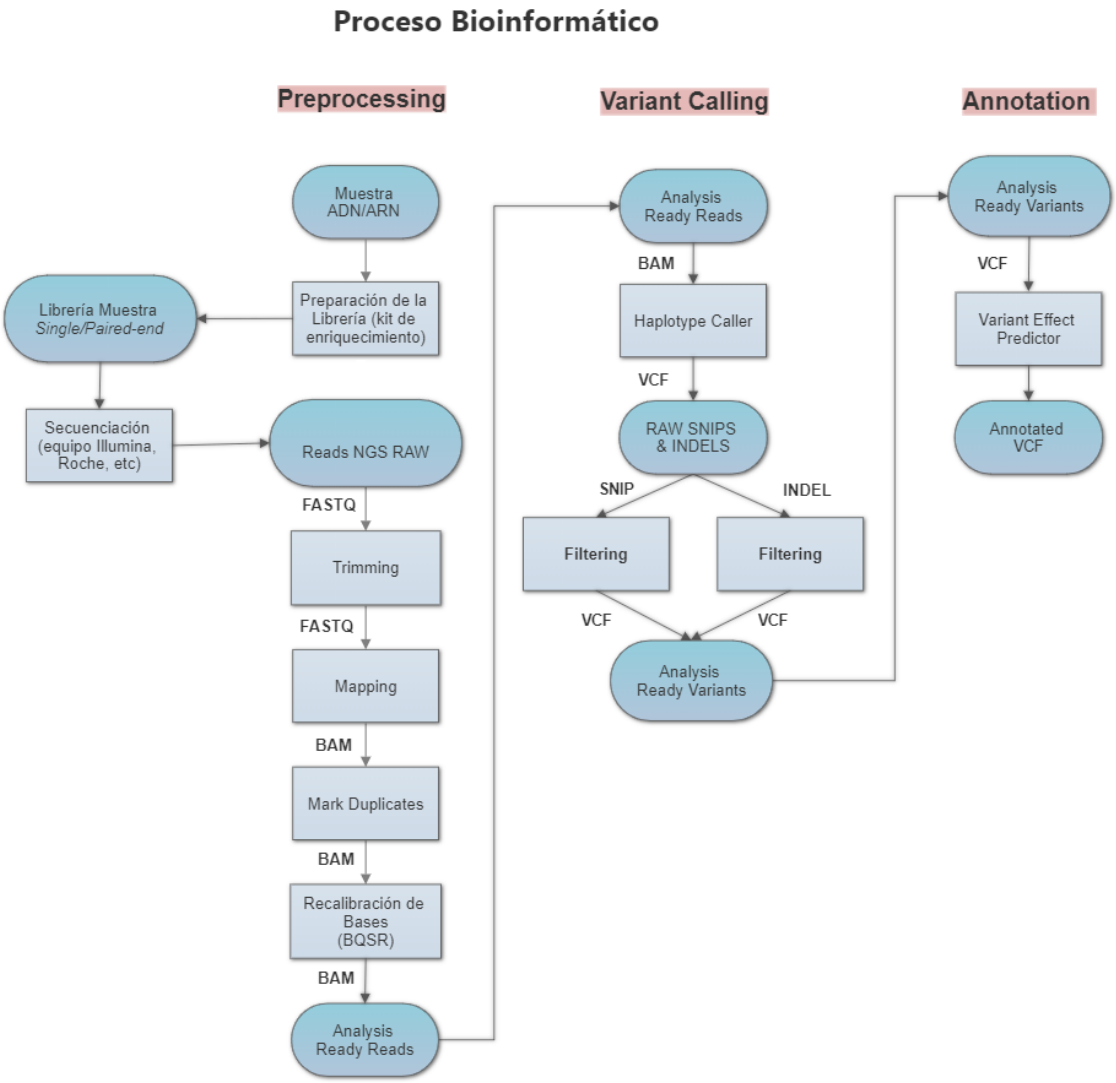


Figura 5. Proceso bioinformático desde las lecturas en crudo de ADN hasta la obtención del fichero VCF con anotaciones.

El pipeline se divide en tres bloques principales: preprocesado, *variant calling* y anotación VEP.
Se parte de los ficheros de secuencias (FASTQ) obtenidos de la secuenciación de exoma WES (Whole Exome Sequencing) paired-end (por tanto tenemos dos, R1 y R2).

Los scripts se lanzan en el clúster SGE mendiante el comando *qsub -t 1* <nombre_del_script> (el 1 es porque solo tenemos una muestra).

Por jemplo,

qsub -t 1 _runStep_0000_trimmomatic.sh

lanza el scrip de trimming

2.3.1. Preprocesado

Implica el procesamiento previo de los datos de secuencia en crudo (proporcionados en formato FASTQ) para producir archivos BAM listos para el análisis. Esto implica la alineación con un genoma de referencia, así como algunas operaciones de limpieza de datos para corregir los sesgos técnicos y hacer que los datos sean adecuados para el análisis.

i. Trimming

Se utiliza la herramienta Trimmomatic[23], una herramienta flexible de recorte de lectura para datos de Illumina NGS.

En esta etapa se van quitando los adaptadores que se habían introducido al preparar la librería.

Para ejecutar esta herramienta, se ejecuta el siguiente código:

```
item=$(awk "NR==$SGE_TASK_ID" ../0000_dataset/FASTQFilename.lst)
/usr/lib/jvm/java-1.8.0/jre/bin/java -Xmx90G -jar
/home/jdelabarrera/Apps/Trimmomatic/trimmomatic.jar PE -threads
44 \
_in/${item}_1.fastq.gz _in/${item}_2.fastq.gz \
_out/${item}.R1.fastq.gz _out/${item}_unpaired.fastq.gz
_out/${item}.R2.fastq.gz _out/${item}_unpaired.fastq.gz \
ILLUMINACLIP:/home/jdelabarrera/Apps/Trimmomatic/adapters/TruSeq
3-PE.fa:2:30:10 LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

NOTA: Los adaptadores especificados se corresponden con el kit de preparacion de libreria utilizado (TruSeq3[24]).

Se han señalado en naranja y azul los dos ficheros origen correspondientes a las lecturas *paired-end*.

ii. Mapping

Se mapean las lecturas al genoma de referencia (GRCh37).

Se utiliza el paquete de software BWA (Burrows-Wheeler Aligner[25]). Este software es útil para mapear secuencias poco divergentes frente a un gran genoma de referencia, como el genoma humano. Consta de tres algoritmos: BWA-backtrack, BWA-SW y BWA-MEM. El primer algoritmo está diseñado para lecturas de secuencia de Illumina de hasta 100 pb, mientras que el resto para secuencias más largas varía de 70 pb a 1 Mb. BWA-MEM y BWA-SW comparten características similares, como soporte de lectura prolongada y alineación de división, pero BWA-MEM, que es el último, generalmente se recomienda para consultas de alta calidad ya que es más rápido y preciso. BWA-MEM también tiene un mejor rendimiento que BWA-backtrack para lecturas Illumina de 70-100 pb. Como se están utilizando lecturas de Illumina se utiliza el algoritmo BWA-MEM, que es el recomendado por los desarrolladores.

Para ejecutar esta herramienta, se ejecuta el siguiente código:

```
item=$(awk "NR==$SGE_TASK_ID" ../0000_dataset/FASTQFilename.lst)
bwa mem -t 45 -M -R "$rgString"
../../REFERENCES/bwa/human_g1k_v37.fasta _in/${item}.R1.fastq.gz
_in/${item}.R2.fastq.gz > _tmp/${item}.sam
```

```
/usr/lib/jvm/java-1.8.0/jre/bin/java         -jar         -Xmx16G
/programs/prod/apps/picard-tools/MergeSamFiles.jar \
INPUT=_tmp/${item}.sam \
OUTPUT=_out/${item}.bam  \
SORT_ORDER=coordinate VALIDATION_STRINGENCY=SILENT

samtools index _out/${item}.bam
```

El parámetro -t indica el número de hilos que se van a utilizar para llevar a cabo el trabajo.
-M adapta el formato del archivo para que sea compatible con el programa Picard. Es un parámetro opcional.
-R para indicar una etiqueta de READ GROUP. Son metadatos sobre las lecturas del BAM que necesita GATK.
_out/${item}.bam es el fichero .bam donde se encuentran los datos obtenidos por BWA-MEM, con la información sobre el alineamiento de las lecturas en el genoma de referencia seleccionado.

iii. Mark duplicates

Identifica los duplicados de PCR [26] que pueden surgir durante la preparación de la librería y los marca con un flag. [27]
Esta herramienta localiza y etiqueta las lecturas duplicadas dando como resultado un nuevo archivo BAM o SAM, donde las lecturas duplicadas se definen como provenientes de un único fragmento de ADN. Los duplicados están marcados con el valor hexadecimal de 0x0400, que corresponde a un valor decimal de 1024.

Código para la ejecución de la herramienta:

```
item=$(awk "NR==$SGE_TASK_ID" ../0000_dataset/Samples.lst)
/usr/lib/jvm/java-1.8.0/jre/bin/java         -jar         -Xmx16G
/programs/prod/apps/picard-tools/MarkDuplicates.jar \
I=_in/${item}.bam \
O=_out/${item}.bam  \
VALIDATION_STRINGENCY=LENIENT                    CREATE_INDEX=true
ASSUME_SORTED=true \
M=_log/${item}.output.metrics
```

iv. Recalibración de Bases (BQSR)

Es una etapa de procesamiento previo de datos que detecta errores sistemáticos hechos por el secuenciador cuando estima la puntuación de calidad de cada llamada base. [28][29]
Intenta homogeneizar los distintos criterios de calidad de los ficheros FASTQ.

Funciona solo en sitios que no están en dbSNP; suponemos que todos los desajustes de referencia que vemos son por lo tanto errores e indicativos de mala calidad de base. Esta herramienta genera tablas basadas en diversas covariables especificadas por el usuario (como grupo de lectura, puntuación de calidad reportada, ciclo y contexto). Suponiendo que estamos trabajando con una gran cantidad de datos, podemos calcular una

probabilidad empírica de error dadas las covariables particulares vistas en este sitio, donde p (error) = num desajustes / num observaciones.

El archivo de salida es una tabla (de los distintos valores de covariables, número de observaciones, número de desajustes, puntuación de calidad empírica).

Esta etapa consta de tres pasos.

Código para la ejecución de la herramienta:

Paso 1:

```
/usr/lib/jvm/java-1.8.0/jre/bin/java -Xmx100G -jar
/home/jdelabarrera/Apps/GATK/GenomeAnalysisTK.jar -T
BaseRecalibrator -nct 40 \
-R ../../REFERENCES/GATK_37/human_g1k_v37.fasta \
-I _in/${item}.bam \
-o _tmp/${item}.recal_data.table \
-knownSites
../../REFERENCES/GATK_37/1000G_phase1.snps.high_confidence.b37.v
cf.gz \
-knownSites
../../REFERENCES/GATK_37/Mills_and_1000G_gold_standard.indels.b3
7.vcf.gz \
-knownSites ../../REFERENCES/GATK_37/dbsnp_137.b37.vcf.gz
```

Paso 2:

```
/usr/lib/jvm/java-1.8.0/jre/bin/java -Xmx100G -jar
/home/jdelabarrera/Apps/GATK/GenomeAnalysisTK.jar -T
BaseRecalibrator -nct 40 \
-R ../../REFERENCES/GATK_37/human_g1k_v37.fasta \
-I _in/${item}.bam -BQSR _tmp/${item}.recal_data.table \
-o _tmp/${item}.post_recal_data.table \
-knownSites
../../REFERENCES/GATK_37/1000G_phase1.snps.high_confidence.b37.v
cf.gz \
-knownSites
../../REFERENCES/GATK_37/Mills_and_1000G_gold_standard.indels.b3
7.vcf.gz \
-knownSites ../../REFERENCES/GATK_37/dbsnp_137.b37.vcf.gz
```

Paso 3:

```
/usr/bin/time --verbose -a -o _log/runtime.${item}.log \
/usr/lib/jvm/java-1.8.0/jre/bin/java -Xmx100G -jar
/home/jdelabarrera/Apps/GATK/GenomeAnalysisTK.jar -T PrintReads
-nct 40 \
-R ../../REFERENCES/GATK_37/human_g1k_v37.fasta \
-I _in/${item}.bam -BQSR _tmp/${item}.post_recal_data.table \
-o _out/${item}.bam
```

2.3.2. Variant Calling

i. Haplotype Caller

HaplotypeCaller [30] es capaz de invocar SNPs e indels simultáneamente a través de ensamblaje de novo local de haplotipos en una región activa. En otras palabras, cada vez que el programa encuentra una región que muestra signos de variación, descarta la información de mapeo existente y vuelve a montar completamente las lecturas en esa región. Esto permite que HaplotypeCaller sea más preciso cuando se llaman regiones que tradicionalmente son difíciles de llamar, por ejemplo, cuando contienen diferentes tipos de variantes cercanas entre sí. También hace que HaplotypeCaller sea mucho mejor al invocar indels que las llamadas basadas en posición como UnifiedGenotyper.

Código para la ejecución de la herramienta:

```
/etc/alternatives/java_sdk_1.8.0/bin/java      -jar      -Xmx100G
/home/jdelabarrera/Apps/GATK/GenomeAnalysisTK.jar              -T
HaplotypeCaller -nct 44 \
-R ../../REFERENCES/GATK_37/human_g1k_v37.fasta \
--dbsnp ../../REFERENCES/GATK_37/dbsnp_137.b37.vcf.gz \
-L ../../REFERENCES/Regions.interval_list \
-I _in/${item}.bam -o _out/${item}.vcf \
-G Standard -G AS_Standard
```

ii. Filtering

Como solo estamos procesando una muestra no podemos utilizar el procedimiento de filtrado de variantes basado en técnicas de machine learning (se necesitan alrededor de 30 muestras); en su lugar realizamos un filtrado “a mano” basado en las recomendaciones de GATK. Estos filtros son independientes para SNPs e INDELs, por ello tenemos que hacerlo por separado y luego combinarlos.[31]

Código para seleccionar los SNPs:

```
item=$(awk "NR==$SGE_TASK_ID" ../0000_dataset/Samples.lst)
/etc/alternatives/java_sdk_1.8.0/bin/java                    -jar
/home/jdelabarrera/Apps/GATK/GenomeAnalysisTK.jar              -T
SelectVariants \
-R ../../REFERENCES/GATK_37/human_g1k_v37.fasta \
-V _in/${item}.vcf  -selectType SNP \
-o _tmp/${item}.vcf.snp.vcf
```

Aplicar los filtros recomendados:

```
/etc/alternatives/java_sdk_1.8.0/bin/java                    -jar
/home/jdelabarrera/Apps/GATK/GenomeAnalysisTK.jar              -T
VariantFiltration \
-R ../../REFERENCES/GATK_37/human_g1k_v37.fasta \
-V _in/${item}.vcf  --filterName "standard_SNP_hard_filter" \
--filterExpression "QD < 2.0 || FS > 60.0 || MQ < 40.0 || MQRankSum
< -12.5 || ReadPosRankSum < -8.0" \
-o _tmp/${item}.snp.vcf
```

Seleccionar los INDEL:

```
/etc/alternatives/java_sdk_1.8.0/bin/java                    -jar
/home/jdelabarrera/Apps/GATK/GenomeAnalysisTK.jar             -T
SelectVariants \
-R ../../REFERENCES/GATK_37/human_g1k_v37.fasta \
-V _in/${item}.vcf  -selectType INDEL \
-o _tmp/${item}.vcf.indel.vcf
```

Aplicar los filtros recomendados:

```
/etc/alternatives/java_sdk_1.8.0/bin/java                    -jar
/home/jdelabarrera/Apps/GATK/GenomeAnalysisTK.jar             -T
VariantFiltration \
-R ../../REFERENCES/GATK_37/human_g1k_v37.fasta \
-V _in/${item}.vcf --filterName "standard INDEL hard filter" \
--filterExpression "QD < 2.0 || FS > 200.0 || ReadPosRankSum < -
20.0" \
-o _tmp/${item}.indel.vcf
```

Combinar los VCF correspondiente a SNPs e INDELs:

```
bcftools concat _tmp/${item}.snp.vcf _tmp/${item}.indel.vcf -o
_tmp/${item}.vcf
```

Nos quedamos solo con los PASS (campo FILTER):

```
bcftools view -f "PASS" -O z -o _out/${item}.vcf.gz
_tmp/${item}.vcf
```

Indexamos el fichero VCF:

```
tabix -p vcf _out/${item}.vcf.gz
```

2.3.3. Anotación VEP

VEP (Variant Effect Predictor[32]) determina el efecto de las variantes (SNP, inserciones, deleciones, CNV o variantes estructurales) en genes, transcripciones y secuencias de proteínas, así como en regiones reguladoras.

A partir de las coordenadas de las variantes y de los cambios de nucleótidos se puede descubrir [33]:

- Genes y Transcripciones afectadas por las variantes
- Ubicación de las variantes (por ejemplo, cadena arriba de una transcripción, en secuencia codificante, en ARN no codificante, en regiones reguladoras)
- Consecuencia de sus variantes en la secuencia de la proteína (por ejemplo, stop ganado, missense, stop perdido, frameshift)
- Variantes conocidas que coinciden con las que se tenían y frecuencias alélicas menores asociadas del Proyecto 1000 Genomas
  Puntajes SIFT y PolyPhen para cambios en la secuencia de proteínas

Código para llamar a la herramienta:

```
item=$(awk "NR==$SGE_TASK_ID" ../0000_dataset/Samples.lst)

/programs/prod/apps/ensembl-tools/scripts/variant_effect_predictor/vep
\
--offline --cache -species homo_sapiens --cache_version 90 --dir
../../REFERENCES/VEP \
--fasta ../../REFERENCES/GATK_37/human_g1k_v37.fasta \
-i _in/SRR1517848.vcf.gz \
-o _out/SRR1517848.vcf \
--vcf \
--variant_class \
--sift b \
--polyphen b \
-gene_phenotype ClinVar dbGaP DDG2P GIANT GOA HGMD-PUBLIC HGMD-PUBLIC
NHGRI-EBI GWAS catalog OMIM Teslovich \
--domains \
--hgvs \
--protein \
--symbol \
--ccds \
--uniprot \
--biotype \
--pubmed \
--plugin LoFtool \
--plugin CADD,../../REFERENCES/CADD/whole_genome_SNVs.tsv.gz
```

2.3.4. Estructura del fichero VCF

Como salida de la última fase del pipeline, se obtiene un fichero VCF con anotaciones.
Esta es la estructura de un fichero VCF[34] básico:

El fichero VCF consta de dos partes claramente diferenciables: la primera parte o cabecera, es un conjunto de líneas que comienzan con "##" y dan información sobre el formato de la segunda parte del fichero, el cuerpo, que contiene la información de cada variante.

Además, la primera parte o cabecera, incluye la versión del formato VCF utilizada.

Para cada variante se tienen los siguientes campos:

CHROM: cromosoma en el que se encuentra la variante.

POS: posición o locus en la que se encuentra la variante dentro del cromosoma anteriormente especificado.

ID: Lista de identificadores únicos separados por puntos y coma, si están disponibles. Si es una variante dbSNP se suele usar el número de rs. Se utiliza en la fase de HaplotypeCaller. Si no hay identificador disponible, entonces se rellena con '.'.

REF: Indica la base o bases del genoma de referencia, en la posición *pos*, pudiendo ser A, C, G, T o N (desconocido).

ALT: Indica la base o bases del genoma alternativo, en la posición *pos*, pudiendo ser A, C, G, T o N (desconocido). Si no se encuentra alelo alternativo (deleción) se marca con el carácter '*'.

QUAL: valor de calidad de la variante de acuerdo a la escala Phred (explicada en el siguiente apartado). A mayor valor de este campo, mayor calidad. Como este valor se ve modificado por otras variables, no es el único campo que se utiliza para medir la calidad de la variante.

FILTER: especifica si la variante ha pasado o no el control de calidad. Este campo se rellena en la fase *Filtering*, que acabamos de ver en el pipeline de llamada a variante y que genera como salida un fichero VCF con las variantes marcadas en función de si se consideraban verdaderas o no. El valor PASS en este campo indica que la variante se ha marcado como verdadera.

INFO: incluye información adicional sobre las variantes, que pueden ser de utilidad al investigador. Debido al amplio abanico de estudios que se pueden realizar los desarrolladores de las herramientas para generar el fichero VCF incluyen todo tipo de información para satisfacer las demandas de los investigadores. Para este proyecto se va a utilizar la información contenida dentro de la etiqueta CSQ. Cada propiedad está separada por ';'.

Ejemplo del campo info:

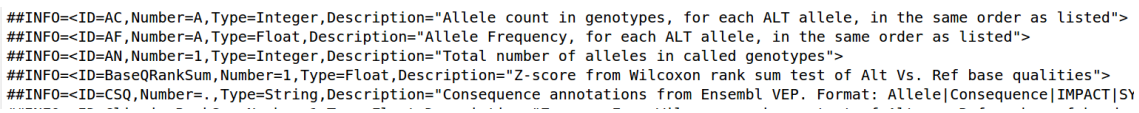
DP=154;MQ=52;H2.

Se permiten claves sin valores correspondientes para indicar pertenencia a un grupo (por ejemplo, H2 indica que el SNP se encuentra en HapMap 2).

FORMAT: Especifica el formato que tendrán las siguientes columnas (correspondientes a cada muestra). La descripción de los elementos que contendrá este campo se indica en la cabecera del fichero VCF.

Ejemplo:

- Cabecera



- Cuerpo

```
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  NA12878
1   57061178    .   C   T   247.77  .   AC=1;AF=0.500;AN=2;BaseQRankSum=1.85;CSQ=T|intrc
1   64806671    .   T   G   896.77  .   AC=2;AF=1.00;AN=2;CSQ=G|intergenic_variant|MODIF
1   64667260    .   C   A   16.86   .   AC=1;AF=0.500;AN=2;BaseQRankSum=-1.471e+00;CSQ=A
1   42729673    .   G   A   480.77  .   AC=2;AF=1.00;AN=2;CSQ=A|intron_variant|MODIFIER|
```

NOMBRES DE LAS MUESTRAS: a continuación de la columna FORMAT se incluyen tantes columnas como muestras, encabezadas con el nombre de la muestra.

2.3.5. Fichero VCF con anotaciones

Una vez finaliza la fase de Anotación VEP, se obtiene como resultado un fichero VCF con las siguientes anotaciones, incluidas bajo el campo INFO, con el ID “CSQ”[35]:

```
##INFO=<ID=CSQ,Number=.,Type=String,Description="Consequence annotations from
Ensembl VEP. Format:
Allele|Consequence|IMPACT|SYMBOL|Gene|Feature_type|Feature|BIOTYPE|EXON|INTRON
|HGVSc|HGVSp|cDNA_position|CDS_position|Protein_position|Amino_acids|Codons|Ex
isting_variation|DISTANCE|STRAND|FLAGS|VARIANT_CLASS|SYMBOL_SOURCE|HGNC_ID|CCD
S|ENSP|SWISSPROT|TREMBL|UNIPARC|GENE_PHENO|SIFT|PolyPhen|DOMAINS|HGVS_OFFSET|g
nomAD_AF|gnomAD_AFR_AF|gnomAD_AMR_AF|gnomAD_ASJ_AF|gnomAD_EAS_AF|gnomAD_FIN_AF
|gnomAD_NFE_AF|gnomAD_OTH_AF|gnomAD_SAS_AF|CLIN_SIG|SOMATIC|PHENO|PUBMED|LoFto
ol|CADD_PHRED|CADD_RAW">
```

En la etiqueta description se encuentran todos los campos de información generados.

Para este proyecto se trabajará con los campos marcados en negrita, no obstante se ha procesado el resto de campos mediante el parseador de ficheros VCF por si en un futuro se quisiera trabajar con ellos.

A continuación se describen los campos utilizados:
Allele: el alelo de la variante utilizado para calcular la consecuencia
Consequence: tipo de consecuencia de esta variante.
Ejemplos:
missense variant: una secuencia de la variante que cambia una o más bases, resultando en un aminoácido diferente pero que mantiene la misma longitud.
downstream gene variant: variante localizada en el extremo 3’ del gen.

IMPACT: el modificador de impacto para el tipo de consecuencia
SYMBOL: el símbolo del gen
Gene: ID Ensembl estable del gen afectado
Feature type: tipo de característica que está en el siguiente campo (por ejemplo, tránscrito, motivo, miRNA, etc.).
Feature: Dependiendo de la anotación, puede ser: ID del tránscrito, ID de motivo, miRNA, pico de ChipSeq, marca de Histona, etc.
BIOTYPE: Biotipo de transcripción o característica reguladora
CLIN_SIG: Importancia clínica de ClinVar de la variante dbSNP
Existing_variation: identificador/es de las variantes conocidas co-ubicadas
SIFT: predicción y/o puntuación SIFT[36], con ambos dados como predicción

PolyPhen: la predicción y / o puntuación de PolyPhen [37]
gnomAD_AF - Frecuencia de variante existente en exomas gnomAD[1] de la población combinada
gnomAD_AFR_AF - Frecuencia de variante existente en exomas gnomAD de la población afroamericana
gnomAD_AMR_AF - Frecuencia de variante existente en exomas gnomAD de la población estadounidense
gnomAD_ASJ_AF - Frecuencia de variante existente en exomas gnomAD de la población judía Ashkenazi
gnomAD_EAS_AF - Frecuencia de variante existente en exomas gnomAD de la población de Asia oriental
gnomAD_FIN_AF - Frecuencia de variante existente en exomas gnomAD de la población finlandesa
gnomAD_NFE_AF - Frecuencia de variante existente en exomas gnomAD de la población no finlandesa europea
gnomAD_OTH_AF - Frecuencia de variante existente en exomas gnomAD de otras poblaciones combinadas
gnomAD_SAS_AF - Frecuencia de variante existente en exomas gnomAD de la población de Asia meridional
LoFtool: una puntuación de intolerancia genética basada a variantes de pérdida de función [38]. Presenta una ventaja sobre PolyPhen y SIFT ya que aquellas no extrapolan sus prediccions a nivel de gen.
CADD_PHRED: valor de puntuación CADD (Combined Annotation Dependent Depletion [39]) con escala Phred.
La fórmula para calcular la puntuación de calidad Phred (Q) es la siguiente:
Q= -10(Log10P)
Siendo P la probabilidad de que una base sea deletérea.

CADD_RAW: puntuaciones CADD (Combined Annotation Dependent Depletion) en bruto, es decir, vienen directamente del modelo, y son interpretables en la medida en que el perfil de anotación para una variante dada sugiere que esa variante es probable que sea "observada" (valores negativos) frente a "simulada" (valores positivos).

Ejemplo de una línea de un fichero VCF con estos campos:

1 64806671 . T G 896.77 . AC=2;AF=1.00;AN=2;CSQ=G|intergenic_variant|MODIFIER||||||||||||||rs10889479||||SNV|||||||||||||||||||||||||||3.779|0.115999;DP=21;ExcessHet=3.0103;FS=0.000;MLEAC=2;MLEAF=1.00;MQ=60.00;QD=29.46;SOR=1.828 GT:AD:DP:GQ:PGT:PID:PL 1/1:0,21:21:63:1|1:64806671_T_G:925,63,0

Los campos están separados por pipes (“|”).

[1] La Base de datos de agregación de genomas (gnomAD) es un recurso desarrollado por una coalición internacional de investigadores con el objetivo de agregar y armonizar datos de secuenciación de exoma y genoma de una amplia variedad de proyectos de secuenciación a gran escala y hacer que los datos de resumen estén disponibles para el resto Comunidad Científica[40].

2.4. Parser de ficheros VCF

El lenguaje de programación elegido para desarrollar el parseador de ficheros VCF es Python, por ser un lenguaje de ágil y sencillo, que ya se había visto en alguna asignatura del máster.

Se ha utilizado como base PyVCF [41], que se ha modificado para adaptarlo a las necesidades del proyecto.

El código se ha incluido dentro del repositorio GitHub indicado en el apartado *1.5.2. Productos desarrollados*.
El enlace para accedir directamente a este script es:

```
https://github.com/amartinlla/TFM/blob/master/vcf_parser.py
```

Algunas de las modificaciones principales han consistido en crear una estructura de diccionarios para almacenar la información más relevante en una misma variable y poder distinguirla mediante etiquetas para que resulte más fácil su extracción cuando sea necesario.

Además, se ha añadido la funcionalidad de volcar los datos leídos en un fichero llamado out.sql con la idea de ser ejecutados directamente desde la base de datos PostgreSql mediante el siguiente comando:

```
psql -U user1 -d estudio1 -f out.sql -h localhost
```

En un primer momento se desarrolló el parseador haciendo los accesos a la base de datos directamente desde el script de Python, pero se vio que cuando el fichero VCF era grande los tiempos de ejecución eran demasiado elevados.

Además, de esta forma se obtienen dos módulos independientes: el parseador de ficheros VCF y el fichero .sql para crear e insertar en las tablas de la base de datos. De esta forma, si el investigador ya posee un fichero .sql con los datos, podría cargarlos sin necesidad de utilizar el parseador.

El script está formado por las siguientes funciones principales, sombreadas en gris:

```
read_vcf(args.file,f_out)
        abrir el fichero VCF para lectura
        descomprimir el fichero en caso de que estuviera comprimido
        para cada línea:
                si la línea empieza por ‘##’:
                        si empieza por '##INFO':
                                read_info(línea)
                        si empieza por ‘##FILTER’:
                                read_filter(línea)
                        si empieza por ‘##ALT’:
                                read_alt(línea)
```

```
                        si empieza por ‘##FORMAT’:
                                read_format(línea)
                        si empieza por ‘##contig:
                                read_contig(línea)
                        sino: #otras etiquetas no contempladas antes
                                read_meta(línea)
                #Comienza la lectura de los datos
                record = read_data(line,row_pattern,
dic,samples,sample_indexes,formats)
                si record[info] contiene la etiqueta CSQ:
                        para cada CSQ separado por “,”:
                                almacena la info en un array
                                escribe en fichero de salida (SQL)
```

Cada una de las funciones *read_...* a las que se llama si la línea es de la cabecera (es decir, si empieza por ‘##’) devuelve un par [clave,valor], para guardar más adelante en un diccionario de datos.

record es la estructura (diccionario de datos) que contendrá la información de cada campo de datos (chrom, pos, ID, ref, alt, qual, filt, CSQ) así como la información de cada muestra.

Ejemplo del contenido de record para una línea del fichero VCF con los datos del paciente con el síndrome de Schinzel Giedion:

{'info': {'CSQ':
['A|upstream_gene_variant|MODIFIER|CMTM1|ENSG00000089505|Transcript|ENST00000379500|protein_coding|||||||||rs2232727|640|1||SNV|HGNC|19172|CCDS10812.2|ENSP00000368814|Q8IZ96||UPI000056D4A6|||||||||||||||||||0.975|7.492|0.508054',
'A|intron_variant|MODIFIER|CKLF|ENSG00000217555|Transcript|ENST00000264001|protein_coding||3/3|ENST00000264001.4:c.334-79G>A|||||||rs2232727||1||SNV|HGNC|13253|CCDS10807.1|ENSP00000264001|Q9UBR5||UPI0000036209|||||||||||||||||||0.413|7.492|0.508054',
'A|intron_variant|MODIFIER|CKLF-CMTM1|ENSG00000254788|Transcript|ENST00000527729|protein_coding||1/2|ENST00000527729.1:c.79-11297G>A|||||||rs2232727||1||SNV|HGNC|39977|CCDS56000.1|ENSP00000433998||E9PSG2|UPI00001FF344|||||||||||||||||||7.492|0.508054']}, 'filt': [], 'chrom': '16', 'fmt': 'GT:AD:DP:GQ:PL', 'pos': 66599710, 'qual': 272.08, 'sample_indexes': {'manuel': 0}, 'samples': {'manuel': {'GT': '0/1', 'GQ': 99.0, 'AD': [17, 12], 'DP': 29, 'PL': [302, 0, 496]}}, 'alt': 'A', 'ref': 'G', 'ID': 'rs2232727'}

La función *read_data* almacena la información de los siguientes campos en *record*: chrom, pos, ID, ref, alt, qual, filt, info, fmt, sample_indexes

A continuación, si se encuentra la etiqueta CSQ dentro de la clave ‘info’ de la variable record se empieza a leer cada bloque de CSQ. Es decir, para una misma variante puede haber varios bloques de información CSQ:

Variante1 CSQ=valor11|valor21|valor31,valor12|valor22|valor32

Cada bloque CSQ está separado de otro por “,”.

Mientras se va leyendo la información de cada variante (o, lo que es lo mismo, de cada línea del fichero), se van escribiendo en un fichero de salida las sentencias SQL necesarias para insertar los datos en cada una de las tablas que se describirán en el siguiente apartado.

Ejemplo de fragmento de código en *VCF_parser.py*:

```
f_out.write('INSERT                                    INTO
PREDICTORS(fk_trans,sift,polyphen,loftool,cadd_phred,cad
d_raw) VALUES (')
f_out.write(str(ind+1))
f_out.write(',')
f_out.write(preds['sift'])
f_out.write(',')
f_out.write(preds['poly'])
f_out.write(',')
f_out.write(preds['lof'])
f_out.write(',')
f_out.write(preds['phred'])
f_out.write(',')
f_out.write(preds['raw'])
f_out.write(');\n')
```

Que generaría en el fichero .sql una línea como esta:

```
INSERT INTO PREDICTORS(fk_trans,sift,polyphen,loftool,cadd_phred,cadd_raw)
VALUES (0,'','','0.847','-0.252552');
```

El fichero de salida se llamará *out.sql* y también incluirá algunos cruces entre tablas, que se almacenarán en unas tablas auxiliares, de las que se dará más detalle en el siguiente apartado.

2.5. Testeo del parser de ficheros VCF

Una vez finalizado el desarrollo del parser lo probamos con diferentes conjuntos de datos: en primer lugar utilizamos datos privados de pacientes del CNIC. Lanzamos el parser y vemos si hay líneas problemáticas. Si las hay las separamos en un fichero aparte y lanzamos de nuevo el pipeline solo con esas líneas. Tratamos los errores y volvemos a lanzar el pipeline con el fichero entero.

Repetimos el procedimiento con algunos ficheros VCF que encontramos por internet de ejemplo, aunque no correspondan a datos reales y hacemos lo mismo con los dos conjuntos de datos públicos mencionados previamente: NA12878.vcf y schinzel_giedion.vcf.

En el caso de NA12878.vcf hacemos el proceso bioinformatico (desde el FASTQ al VCF), pero al tratarse de un genoma control [42], es decir, de un individuo sano, no lo usamos posteriormente en la aplicación web para

el filtrado. Solo nos sirve como ensayo de proceso de llamada a variantes a partir de las secuencias.

Para el caso del paciente con el síndrome de schinzel_giedion volvemos a repetir todo el proceso bioinformático y generamos el fichero schinzel_giedion.vcf. En este caso, como los datos corresponden a un paciente real (niño) con este síndrome, podemos utilizar estos datos además para el filtrado y priorización de variantes en la web.

2.6. Estructura de datos

La idea inicial desde la que se diseñó la estructura de la base de datos partió de identificar la cardinalidad de cada campo, tomando como referencia la variante. Es decir, para una variante cuántos cromosomas, posiciones, ids, alelos de referencia, alelos alternativos, etc tiene.

De ahí surgió un primer diseño, que a medida que se fue abanzando en el desarrollo se fue modificando y sobre todo ampliando, para incluir nueva información. Este diseño inicial estaba formado por cuatro tablas principales: VARIANTS, INFO, SAMPLE y GT_FIELDS.

Derivado de la necesidad de meter más campos y como consecuencia de que estábamos duplicando mucha información se decidió separar la entidad INFO en cuatro entidades: TRANSCRIPT, PREDICTORS, POP_MAFS y EXISTING_VARIATION. De esta manera se contemplaba que una variante podia tener más de un tránscrito, un predictor y, a su vez, más de un predictor, frecuencia, etc.

La última modificación, que se indicó en la PEC3 consistió en crear tablas auxiliares que almacenaran los resultados parciales de cruces entre tablas más grandes.

De esta forma, se liberaba a la aplicación web de realizar cruces entre las tablas en tiempo real y se agilizaba la navegación.
Así, los datos mostrados en el front provienen de una de estas tablas auxiliares: AUX2, que contiene todos los campos necesarios para realizar los filtrados y la priorización de variantes y que se cruza solamente con la tabla de frecuencias de la población (POP_MAFS).

El esquema final resultante de la Base de Datos del proyecto es el siguiente:

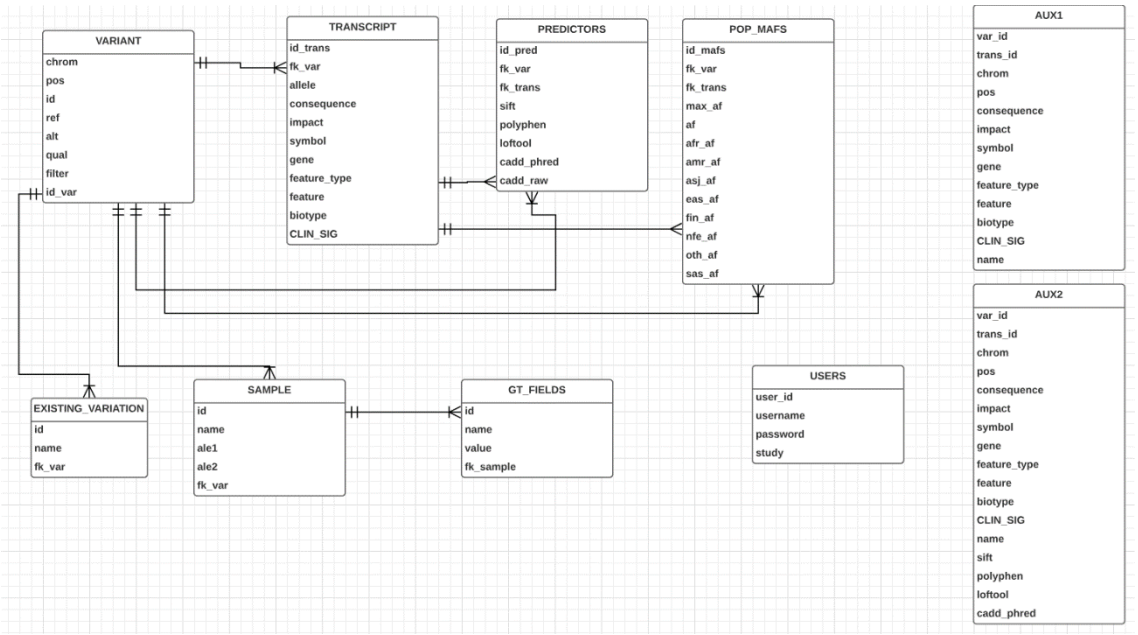


Figura 6. Diagrama Entidad-Relación de la Base de Datos

VARIANT: contiene los campos más relevantes para identificar una variante, que ya se vieron en el apartado 2.3.4.: chrom, pos, id, ref, alt, qual y filter. Los campos FORMAT se encuentran separados en las tablas SAMPLE y GT_FIELDS.

id_var: es la clave primaria de la tabla, es un identificador único.

TRANSCRIPT: contiene los campos descritos en el apartado 2.3.4. (Fichero VCF con anotaciones), correspondientes a la información de un tránscrito. En esencia, un tránscrito contiene la información de la consecuencia, el alelo de la variante utilizado para calcular la consecuencia, el modificador de impacto para el tipo de consecuencia, el símbolo del gen, el identificador Ensembl del gen, el tipo de función, la importancia clínica de ClinVar de la variante dbSNP y algunos campos de puntuación de calidad.

PREDICTORS: tabla con la información de los predictores. El significado de los campos se puede encontrar en el apartado 2.3.4.

POP_MAFS: contiene las frecuencias alélicas en la población. También se describió el significado de cada una de ellas en el apartado 2.3.4. Además contiene el campo “max_af”, que es el máximo de todas las frecuencias, excluyendo a la frecuencia de la población total “af”.

EXISTING_VARIATION: El campo “name” de esta tabla contiene el identificador dbSNP de la variante. Por ejemplo: rs4888735. Más adelante en la aplicación web se verá cómo este campo ayuda en la selección de las variantes candidatas.

SAMPLE: tabla que contiene el nombre y los alelos de cada muestra.

GT_FIELDS: contiene la información del genotipo de las muestras. Se corresponde con el campo FORMAT del fichero VCF, ya descrito en el apartado 2.3.4.

AUX1: tabla auxiliar que contiene la información resultante de los cruces entre las tablas TRANSCRIPT, VARIANT y EXISTING_VARIATION.

AUX2: tabla auxiliar utilizada en el front-end que contiene el cruce entre las tablas AUX1, PREDICTORS y SAMPLE.

USERS: contiene los usuarios dados de alta en la base de datos y el nombre del estudio (campo “study”), que se corresponde con el nombre de la base de datos. Será el administrador de base de datos el que dé de alta un usuario y habrá una base de datos para cada estudio y para cada usuario.

2.7. Diseño, desarrollo y visualización de la aplicación web

Para el desarrollo de la aplicación se ha utilizado Web2py[5], como ya se ha indicado.

Web2py es un framework de código abierto pensado para facilitar el desarrollo de aplicaciones web siguiendo buenas prácticas.

Posee las herramientas necesarias para crear, modificar y administrar aplicaciones web seguras conectadas a servicios de bases de datos.
Está programado en Python[8] y es programable en Python.

Está basado en la idea de MVC (Modelo, vista, controlador). Esto quiere decir que se separa la representación de los datos (vista), de la interacción con la base de datos (modelo) y de la lògica de la aplicación (controlador).

Siguiendo esta idea, el desarrollo de la aplicación web se ha agrupado en tres grandes niveles, como se puede apreciar en el repositorio de GitHub:
- models: que contiene los ficheros .py necesarios para crear la conexión a la base de datos PostgreSql y las tablas de la base de datos. Como los ficheros dentro de “models” son los primeros en cargarse, se puede incluir en ellos todo aquello que se desee visualizar en todas las pantallas de la aplicación. Por ejemplo, en el fichero menu.py, que viene por defecto en la instalación, se puede incluir un menú para que se muestre en todas las ventanas de la aplicación. O utilizar variables globales para ser vistas desde distintos controladores.
Para el desarrollo de la aplicación VCFWeb se han desarrollado cuatro ficheros:

- db.py: fichero que viene por defecto
  - establece la conexión con la base de datos por defecto.
  - Elige un formato para los formularios por defecto.
    ```
    response.formstyle = 'bootstrap4_inline'
    response.form_label_separator = ''
    ```
  - Inicializa la variable Auth, necesaria para el login de los usuarios
    ```
    auth = Auth(db, host_names=configuration.get('host.names'))
    ```

- define las acciones de autenticación que se van a utilizar (en este caso se deshabilita el registro, el cambio de contraseña, la actualización del perfil del usuario y solo se le da la opción de hacer login.

```
auth = Auth(db, host_names=configuration.get('host.names'))
```

- se fuerza a hacer login utilizando un nombre de usuario en lugar de un email

```
auth.define_tables(username=True)
```

- menu.py: fichero que viene por defecto. Contiene los items que se quieren mostrar en la aplicación. En este caso solo la plantilla index.html sin menús adicionales.
- variants.py: se crea de cero para:
  - crear una conexión a la base de datos de PostgreSQL que se describió en el apartado 2.6., indicando el usuario, la contraseña, el nombre de la base de datos y el host.

```
db =
DAL('postgres://user1:VCFW3b@localhost/estudio1',pool_size=0,lazy_tables=True)
```

  - definir todas las tablas de la base de datos que se describieron en el apartado 2.6.
- z_init.py: añade el usuario deseado a la tabla `auth_user` de Web2py, que es la tabla que contiene los usuarios que han hecho login en la aplicación. Se hace de esta forma manual ya que no se permite a los usuarios registrarse en el sistema.

- controllers: contiene los ficheros necesarios para conectar la base de datos con la parte de visualización.

- appadmin.py: este fichero viene por defecto y no se ha modificado. Contiene la configuración por defecto para administrar la aplicación.
- default.py: una vez se han definido y creado las tablas, web2py además genera una interfaz de administración de la base de datos completamente funcional, llamada **appadmin**, para poder acceder a esa base de datos y a sus tablas. "default.py" es el controlador. Este fichero que viene por defecto se ha modificado para incluir:
  - la lógica de la página inicial (default/index.html) y hacia donde se redirigirá al usuario desde esa página, una vez haya hecho login. En este caso se le redirigirá a *variants*.
  - contiene los datos que serán utilizados en la plantilla default/variants.html. Se cruza la tabla AUX2 (de la que se habló en el apartado anterior) con la tabla POP_MAFS y se guardan los datos en un formato serializable (JSON[43]). Posteriormente se convierten a XML para utilizarlos con DataTables[44]

```
@auth.requires_login()
def variants():
	data = json.dumps(db.executesql('SELECT * FROM AUX2 INNER JOIN POP_MAFS ON POP_MAFS.fk_trans=AUX2.trans_id;',as_dict=True))
	return dict(all=XML(data))
```

  - contiene los datos que serán utilizados en la plantilla default/filters.html

```
@auth.requires_login()
def filters():
```

@auth.requires_login() se utiliza para restringir el acceso a funciones a aquellos usuarios que se hayan autenticado únicamente.

- views: contiene las plantillas en formato HTML para visualizar los datos en la aplicación.
Se utiliza el plugin DataTables para mostrar los datos de una forma clara y sencilla y permitir la interacción con los mismos.
Web2py trae por defecto bastantes plantillas, solo se describirán las que se han modificado:

- layout.html: el resto de vistas extenderán de esta. Para ello, llevaran en la primera línea: `{{extend 'layout.html'}}`. Por ello, todo lo que se incluya en esta vista aparecerá en las demás. En este caso se han modificado las opciones de la lista desplegable para dejar solo la posibilidad de hacer login o logout al usuario.

```
{{if auth.user:}}
<a class="dropdown-item"
href="{{=URL('default','user/profile')}}">{{=T('Profile')}}</a>
              <a class="dropdown-item"
href="{{=URL('default','user/change_password')}}">{{=T('Change
Password')}}</a>
              <a class="dropdown-item"
href="{{=URL('default','user/logout')}}">{{=T('Logout')}}</a>
              {{else:}}
              <a class="dropdown-item"
href="{{=URL('default','user/login')}}">{{=T('Login')}}</a>
              <a class="dropdown-item"
href="{{=URL('default','user/register')}}">{{=T('Sign up')}}</a>
              <a class="dropdown-item"
href="{{=URL('default','user/request_password')}}">{{=T('Lost
Password')}}</a>
              {{pass}}
```

- index.html: contiene una página inicial sencilla para acceder a la aplicación desde la cual el usuario puede hacer LOGIN.

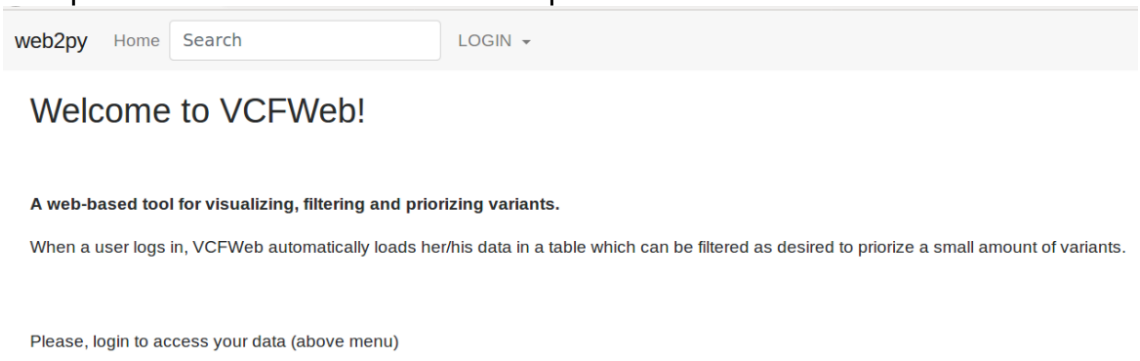


Figura 7. Página principal de la aplicación web

- variants.html: muestra una tabla con algunos datos de la base de datos y ofrece al usuario la posibilidad de exportar los datos a distintos formatos. Pulsando el botón “Go Filtering!” se accede a la ventana “default/filters.html”.

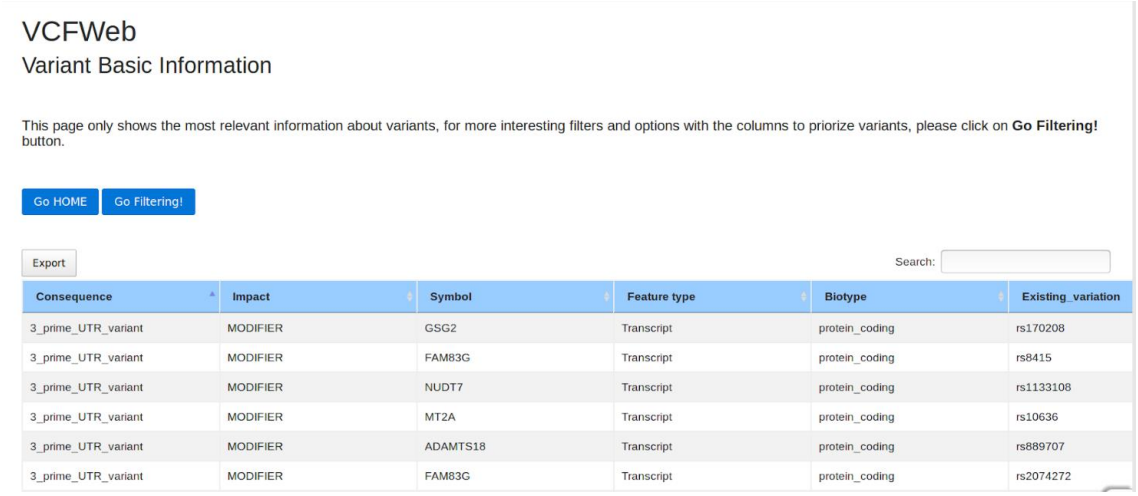


Figura 8. Ventana de variants

- filters.html: contiene más información que la ventana de variants y más opciones como el número de filas de la tabla que se quieren mostrar o las columnas que se quieren ocultar. Se pueden aplicar filtros individuales para cada uno de los campos.

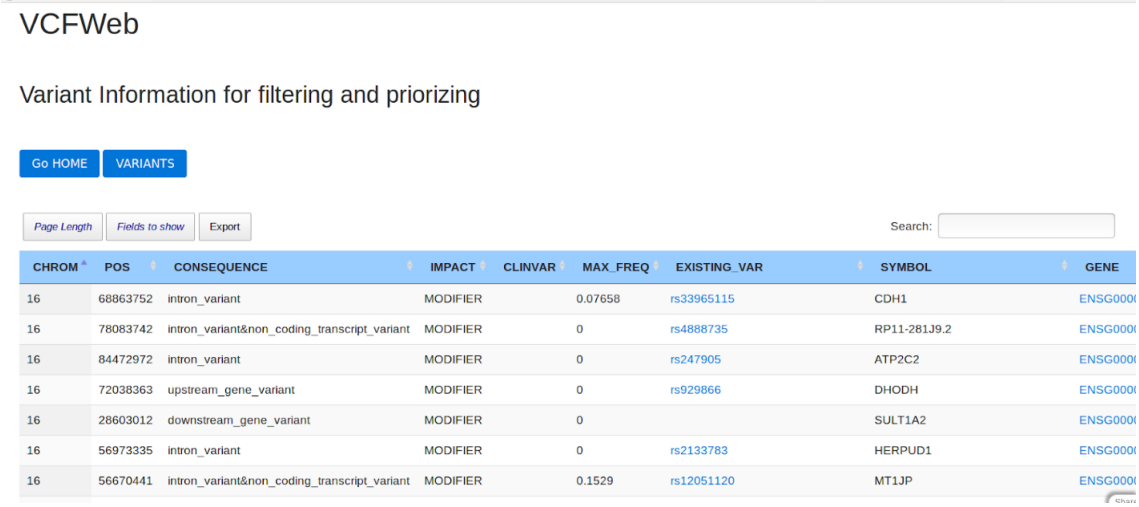


Figura 8. Ventana de filters

2.8. Filtrado, priorización y obtención de variantes cantidatas

El filtrado de variantes se mostrará con el conjunto de datos schinzel_giedion.vcf

Desde la ventana “filters” se filtrarán en primer lugar aquellas variantes con consecuencia e impacto “HIGH” y “MODERATE”.
Ambos campos (“consequence” e “impact”) hacen referencia a las mutaciones sinónimas (si el aminoácido no cambia) o no sinónimas (si el aminoácido cambia). Al filtrar por “HIGH” y “MODERATE” nos estamos quedando con aquellas mutaciones que más impacto provocan en el aminoácido. Con este primer filtro conseguimos pasar de 6831 variantes a 607.

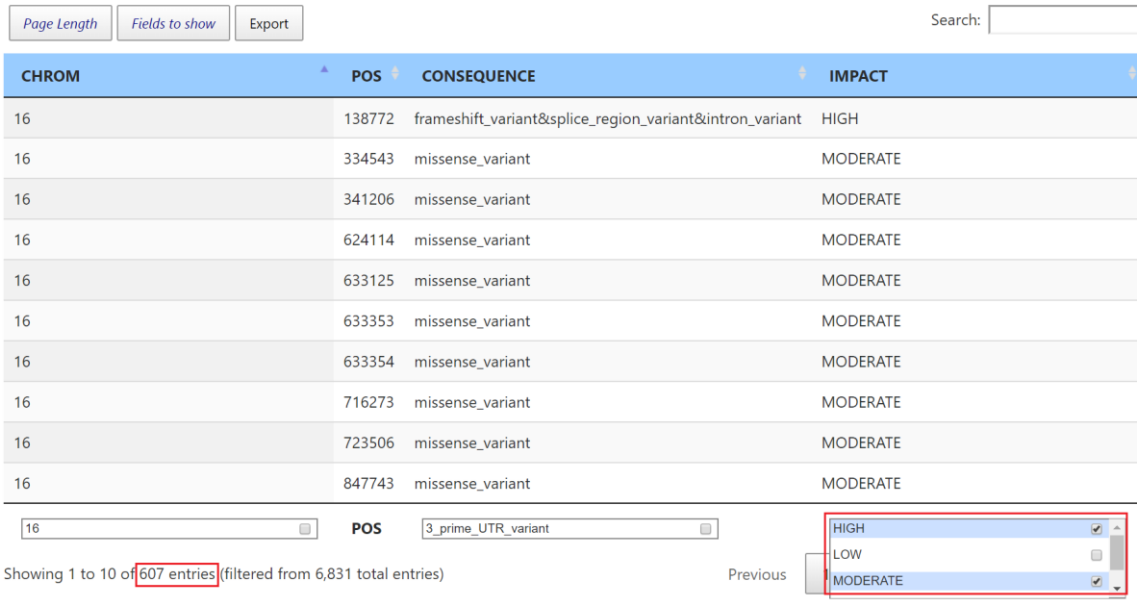


Figura 9. Resultado del primer filtro sobre las variantes.

El segundo filtro que se aplicaría en este caso es el de la frecuencia. Se establece el Umbral en 0 ya que, como se ha dicho antes, sabemos que se trata de una enfermedad que no está presente en la población, ya que los individuos que la padecen no llegan a la edad adulta.
Se reduce el número de variantes a 31.

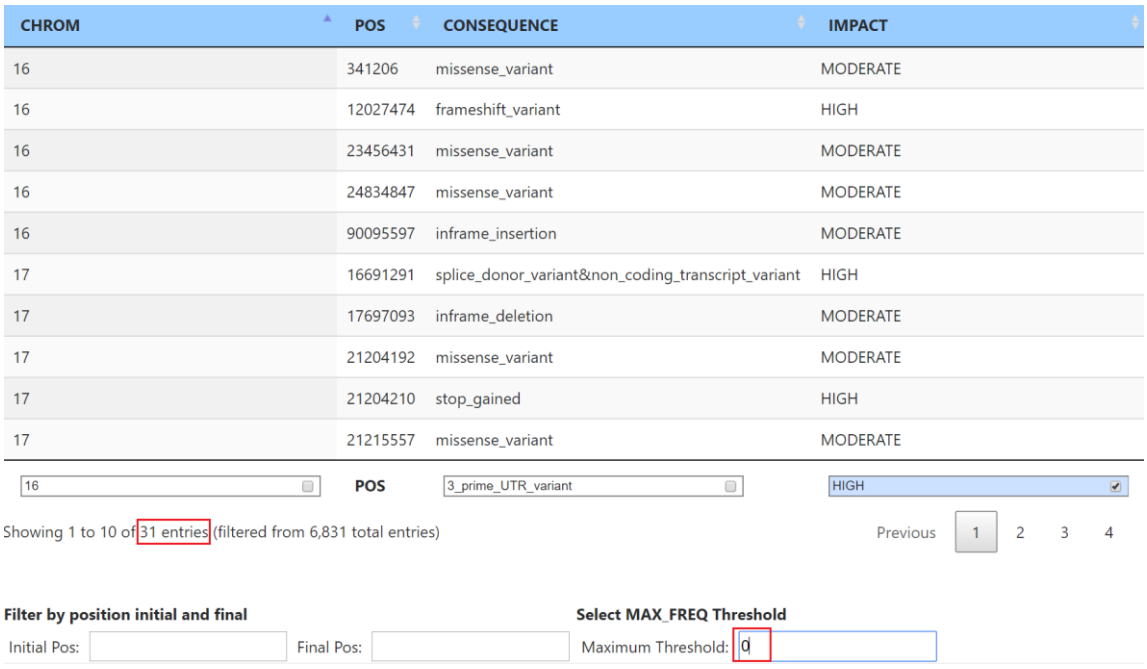


Figura 10. Resultado del segundo filtro sobre las variantes.

El tercer filtro a aplicar es SIFT=deleterious. Con esto nos quedamos con 6 variantes.

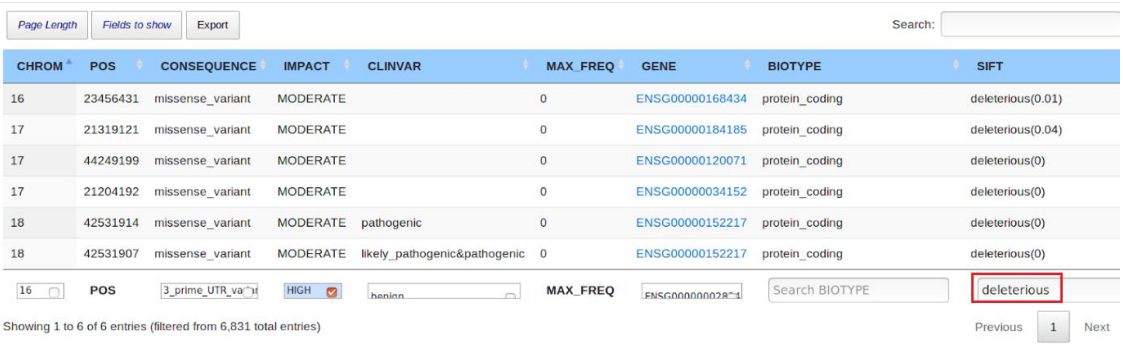


Figura 11. Resultado del tercer filtro sobre las variantes.

El cuarto filtro es PolyPhen=probably_damaging. Seguimos teniendo 6 variantes candidatas.

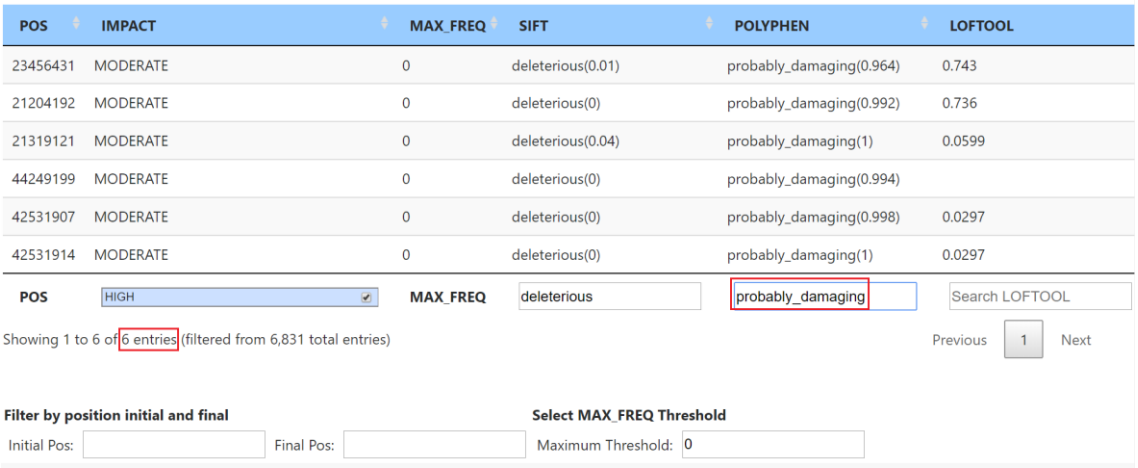


Figura 12. Resultado del cuarto filtro sobre las variantes.

Si nos fijamos en el identificador del gen y vamos pulsando los enlaces que nos llevan a Ensembl, vemos por el fenotipo que el gen SETBP1 está asociado con la enfermedad que estábamos buscando.

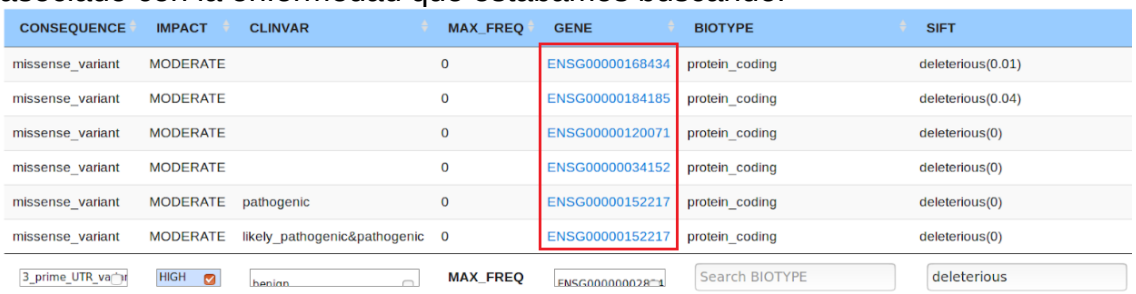


Figura 13. Consulta de la información de los genes en Ensembl.

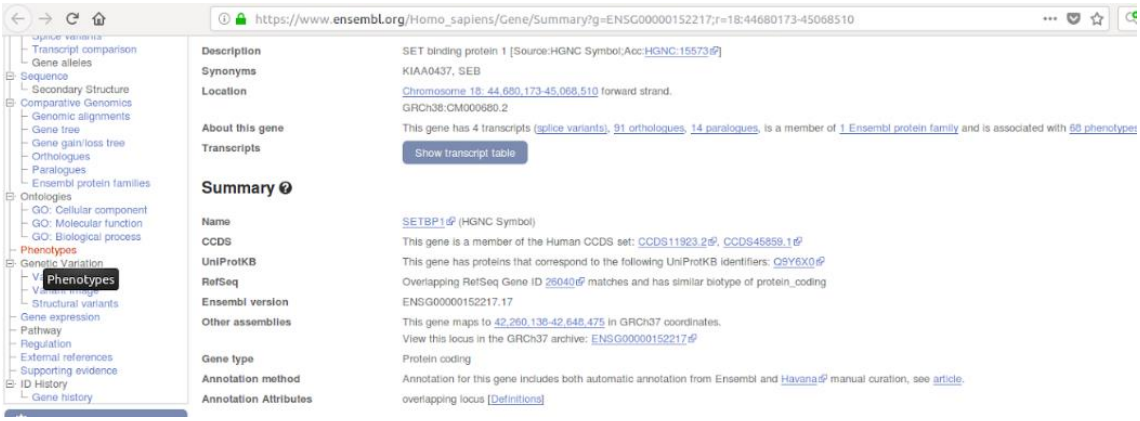


Figura 14. Consulta del fenotipo del gen SETBP1 en Ensembl.

Al filtrar por este gen en nuestra aplicación nos quedan 2 variantes muy próximas (a nivel de coordenadas). Como se sabe que las variantes que provocan este síndrome son variantes “de novo”, es muy posible que se trate de un artefacto técnico en la llamada a variantes. Pero no podemos indagar en ello porque no disponemos de los ficheros BAM.

Figura 15. Resultado del fenotipo en Ensembl

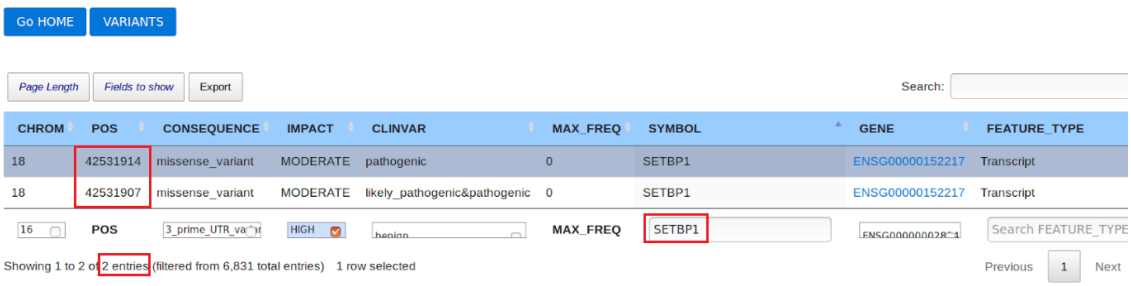


Figura 16. Resultado del tercer filtro sobre las variantes.

Una forma más directa de buscar el fenotipo en Ensembl es desde el enlace del campo “Existing_variation”, como se aprecia en la Figura 17.

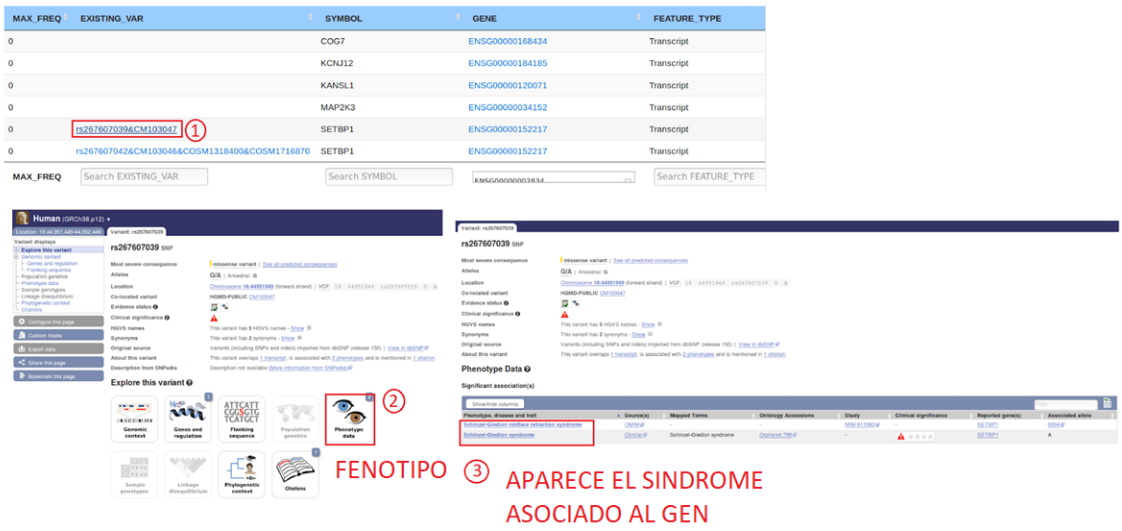


Figura 17. Fenotipo en Ensembl pulsando el enlace de *Existing_variation* .

2.9. Exportación de variantes candidatas

Una vez tenemos las variantes candidatas las podemos exportar en diferentes formatos (Excel, CSV, PDF, o copiar al portapapeles o imprimir), tanto desde la ventana de *variants* como desde la de *filters*:

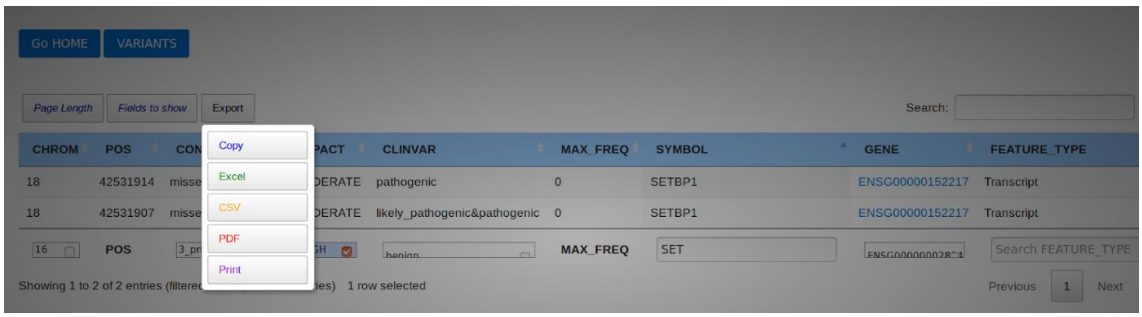


Figura 18. Posibles opciones para exportar los datos en la aplicación desde la ventana filters.

Exportar a Excel:

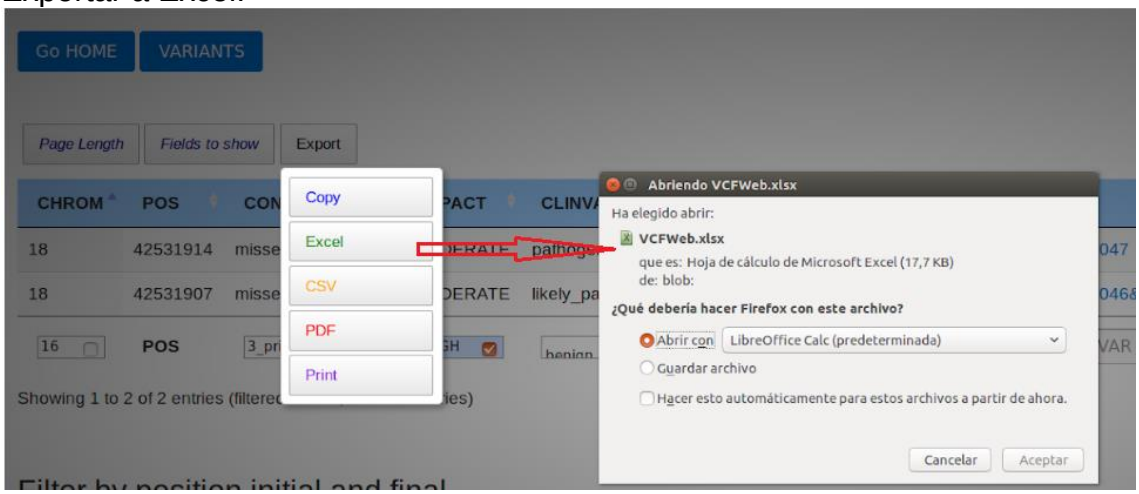


Figura 18. Exportación a Excel desde la ventana filters.

Exportar a CSV:

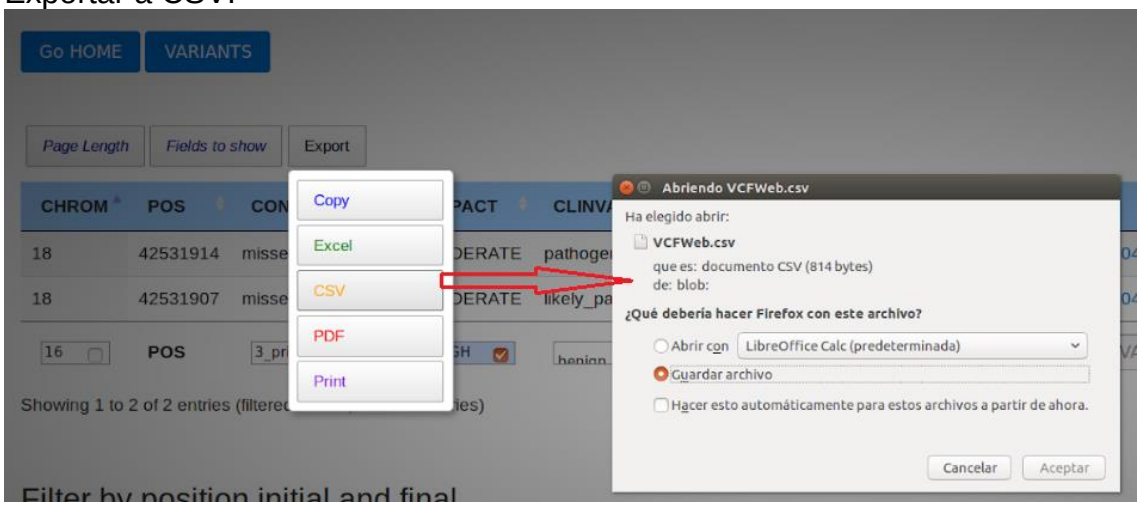


Figura 20. Exportación a CSV desde la ventana filters.

Exportar a PDF:

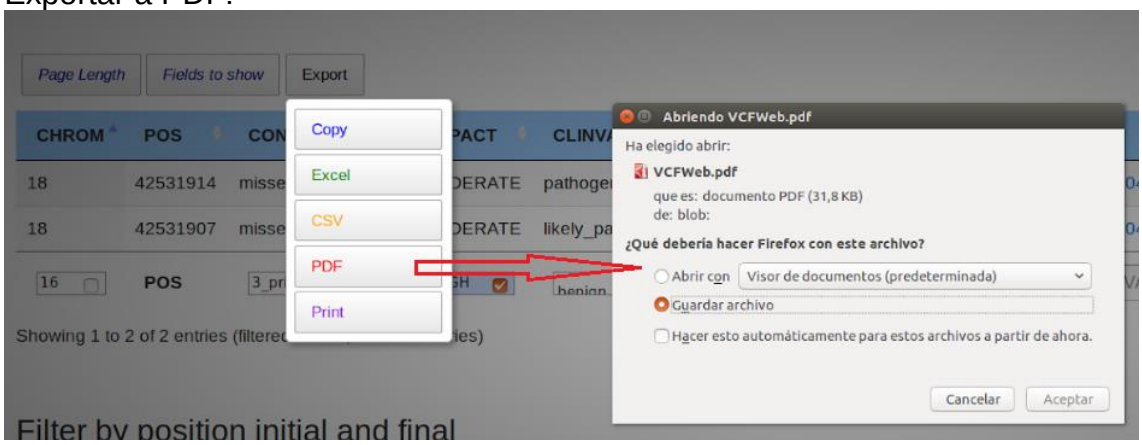


Figura 21. Exportación a PDF desde la ventana filters.

NOTA: El formato PDF está pensado para un volumen pequeño de datos, ya que en la práctica es un formato poco usable con el que trabajar, por ejemplo, si se le quiere aplicar filtros a los datos sin utilizar la aplicación web.
Es un informe meramente informativo, pensado principalmente para mostrar las variantes candidatas, una vez se apliquen todos los filtros al conjunto de variantes total, y este número no debería ser elevado. En el mismo caso estaría la opción de imprimir.

3. Conclusiones

Al comienzo de este proyecto, mis conocimientos del concepto de variante y por tanto de todos sus posibles atributos eran nulos. Con mucha ayuda por parte de los tutores externos del CNIC conseguí entender todo el proceso desde las lecturas de las secuencias de ADN en crudo hasta la obtención de un fichero VCF anotado, que yo tenía que usar como fichero de entrada a un parseador que iba a desarrollar en Python. Este lenguaje de programación lo había visto ya en varias asignaturas del màster, por lo que no me resultó complicado programar con él.

El diseño y la creación de la base de datos también me resultó bastante sencillo una vez tuve claros los conceptos biológicos.

Para la parte web me costó más empezar a desarrollar, ya que aunque soy informática, no es mi campo y estaba muy desactualizada en cuanto a tecnologías y desarrollo de aplicaciones web.

En lo referente al cumplimiento de objetivos, considero que se han logrado en su totalidad, aunque se han tenido que descartar algunas técnicas de filtrado, como el manejo de Pedigree y los paneles y UCL-BHF, porque iban a complicar demasiado el desarrollo y no se iban a poder mostrar los resultados, por tratarse de datos privados.

La planificación no se ha seguido de forma estricta ya que, como se indicó anteriormente, en ocasiones se cambió el orden a las tareas o hubo que dedicar más horas a algunes partes como el desarrollo web, para no ver perjudicada la planificación inicial.

Como líneas de trabajo futuro se añadirán más estrategias de filtrado y priorización de variantes y se combinarán con las ya existentes, como el manejo de Pedigree.

Mi valoración general del proyecto es positiva, ya que considero que he aprendido bastante sobre dos campos con los que no estaba prácticamente o nada familiarizada y he consolidado conceptos y lenguajes de programación que estudié en las assignatures del màster.

4. Glosario

ADN: ácido desoxirribonucleico

BAM: Binary Alignment Map. Es un archivo binario del archivo SAM, que es un archivo de texto que contiene los datos de alineamientos de secuencias.

BCFtools: conjunto de utilidades para la lectura / escritura de archivos BCF2 / VCF / gVCF y llamadas / filtrado / resumen de SNP y variantes de secuencia indel corta.

BWA: Burrows-Wheeler Aligner. Es un paquete de software para mapear secuencias poco divergentes contra un genoma de referencia grande, como el genoma humano.

DataTables: es un complemento para la librería jQuery Javascript. Es una herramienta altamente flexible, construida sobre los fundamentos de la mejora progresiva, que agrega todas estas funciones avanzadas a cualquier tabla HTML.

Ensembl: proyecto de investigación bioinformática que trata de desarrollar un sistema de software que produzca y mantenga anotaciones automáticas en los genomas eucariotas seleccionados.

FASTQ: El formato FASTQ es un formato basado en texto para almacenar tanto una secuencia biológica (generalmente secuencia de nucleótidos) como sus puntuaciones de calidad correspondientes. Tanto la letra de secuencia como la puntuación de calidad están codificadas con un único carácter ASCII para abreviar.

Fenotipo: expresión del genotipo en función de un determinado ambiente.

GATK: Genome Analysis Toolkit. Framework para analizar datos de secuenciación de ADN de próxima generación.

Genotipo: información genética que posee un organismo en particular, en forma de ADN.

GitHub: es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git.

Illumina: Empresa estadounidense líder en secuenciación de nueva generación (NGS) y arrays. Desarrolla, fabrica y comercializa sistemas integrados para el análisis de variación genética y función biológica.

Indel: contracción de "**in**serción o **del**eción", en referencia a los dos tipos de mutaciones genéticas que se consideran a menudo juntas a causa de su efecto similar y la incapacidad de distinguir entre ellas en una comparación de dos secuencias.

JSON: acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos.

NGS: Next Generation Sequencing o secuenciación de nueva generación.

PCR: reacción en cadena de la polimerasa (Polymerase Chain Reaction)

Pipeline: conjunto de procesos ejecutados en serie de forma que la salida de uno es la entrada del siguiente.

QC: control de calidad (Quality Control)

Samtools: es un conjunto de programas para interactuar con datos de secuenciación de alto rendimiento. Consta de tres repositorios separados: Samtools, BCFtools y HTSlib.

Síndrome de Schinzel-Giedion: enfermedad causada por mutaciones germinales de novo que se agrupan en un punto de acceso en el exón 4 del gen SETBP1.

SNP: es una variación en la secuencia de ADN que afecta a una sola base (adenina (A), timina (T), citosina (C) o guanina (G) de una secuencia del genoma.

Variante genética: variación en el material genético de una población o especie, incluyendo los genomas. En este proyecto hace referencia a la variación entre la secuencia de ADN de un individuo respecto a la secuencia del genoma de referencia del ser humano.

VCF: Variant Calling Format. Es un fichero de texto que se usa en Bioinformática para almacenar información sobre la llamada a variantes de la secuencia de genes.

Web2py: framework de desarrollo web de código abierto.

WES: secuenciación del exoma completo (Whole-Exome Sequencing)

WGS: secuenciación del genoma completo (Whole-Genome Sequencing)

5. Bibliografía

[1] https://www.ebi.ac.uk/training/online/course/ebi-next-generation-sequencing-practical-course/what-you-will-learn/what-next-generation-dna- , mayo 2018

[2] https://en.wikipedia.org/wiki/Variant_Call_Format, febrero 2018

[3] http://www.internationalgenome.org/data-portal/sample/NA12878, mayo 2018

[4] https://github.com/raonyguimaraes/mendelmd, mayo 2018

[5] http://www.web2py.com/book, marzo 2018

[6] https://en.wikipedia.org/wiki/Exome_sequencing, mayo 2018

[7] https://datatables.net/, marzo 2018

[8] https://www.python.org/, marzo 2018

[9] https://www.postgresql.org/, marzo 2018

[10] https://software.broadinstitute.org/gatk/documentation/article.php?id=7870, febrero 2018

[11] Silvia Salatino and Varun Ramraj, BrowseVCF: a web-based application and workflow to quickly prioritize disease-causative variants in VCF files, Briefings in Bioinformatics, 18(5), 2017, 774–779

[12] Steven N. Hart, Patrick Duffy, Daniel J. Quest, Asif Hossain, Mike A. Meiners and Jean-Pierre Kocher, VCF-Miner: GUI-based application for mining variants and annotations stored in VCF files, Briefings in Bioinformatics, 17(2), 2016, 346–351

[13] http://www-labgtp.na.icar.cnr.it/VAR2GO/, febrero 2018

[14] Danecek P, Auton A, Abecasis G, et al. The variant call format andVCFtools.Bioinformatics2011;27:2156–8.

[15] Paila U, Chapman BA, Kirchner R, et al. GEMINI: integrative exploration of genetic variation and genome annotations. PLoSComputBiol2013;9:e1003153.

[16] https://research.nhgri.nih.gov/software/VarSifter/index.shtml, febrero 2018

[17] http://jimb.stanford.edu/giab/, mayo 2018

[18] https://www.nist.gov/, mayo 2018

[19] https://arc.liv.ac.uk/trac/SGE, marzo 2018

[20] https://software.broadinstitute.org/gatk/best-practices/, febrero 2018

[21] http://support.illumina.com/content/dam/illumina-support/help/BaseSpaceHelp_v2/Content/Vault/Informatics/Sequencing_Analysis/BS/swSEQ_mBS_FASTQFiles.htm, mayo 2018

[22] https://support.illumina.com/help/BS_App_MDProcessor_Online_1000000007932/Content/Source/Informatics/BAM-Format.htm, mayo 2018

[23] http://www.usadellab.org/cms/?page=trimmomatic, marzo 2018

[24] https://github.com/timflutre/trimmomatic/blob/master/adapters/TruSeq3-PE.fa, marzo 2018

[25] http://bio-bwa.sourceforge.net/, marzo 2018

[26] https://en.wikipedia.org/wiki/Polymerase_chain_reaction, marzo 2018

[27] https://broadinstitute.github.io/picard/command-line-overview.html#MarkDuplicates, marzo 2018

[28] https://gatkforums.broadinstitute.org/gatk/discussion/44/base-quality-score-recalibration-bqsr, marzo 2018

[29] https://software.broadinstitute.org/gatk/documentation/tooldocs/3.8-0/org_broadinstitute_gatk_tools_walkers_bqsr_BaseRecalibrator.php, marzo 2018

[30] https://software.broadinstitute.org/gatk/documentation/tooldocs/3.8-0/org_broadinstitute_gatk_tools_walkers_haplotypecaller_HaplotypeCaller.php, marzo 2018

[31] https://software.broadinstitute.org/gatk/documentation/article.php?id=3225, marzo 2018

[32] https://www.ensembl.org/info/docs/tools/vep/index.html, marzo 2018

[33] https://www.ensembl.org/info/genome/variation/predicted_data.html#consequences, marzo 2018

[34] https://samtools.github.io/hts-specs/VCFv4.2.pdf, marzo 2018

[35] http://snpeff.sourceforge.net/VCFannotationformat_v1.0.pdf, mayo 2018

[36] Ng PC(1), Henikoff S., SIFT: Predicting amino acid changes that affect protein function, 31(13):3812-4, Nucleic Acids Res. 2003

[37] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4480630/, mayo 2018

[38] https://academic.oup.com/bioinformatics/article/33/4/471/2525582, mayo 2018

[39] https://onlinelibrary.wiley.com/doi/abs/10.1002/humu.22798, mayo 2018

[40] http://gnomad.broadinstitute.org/about, mayo 2018

[41] https://pyvcf.readthedocs.io/en/latest/, febrero 2018

[42] http://jimb.stanford.edu/giab/, mayo 2018

[43] https://www.json.org/, mayo 2018

[44] https://datatables.net/, abril 2018

6. Anexos

Se ha decidido no incluir todo el código desarrollado o utilizado en el proyecto como anexo a esta memoria, y en su lugar subir todos los scripts desarrollados, junto con los del pipeline del proceso bioinformático al repositorio de GitHub, para una mayor claridad.