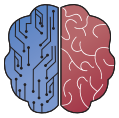
UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería de la Salud



INGENIERÍA
DE LA SALUD

TFG del Grado en Ingeniería de la Salud

Análisis bioinformático para la detección de biomarcadores del cáncer empleando Biopython

Documentación Técnica

Presentado por Gabriel Collado Santamaría
en Universidad de Burgos

8 de julio de 2024

Tutores: Rubén Ruiz González – Antonia Maiara
Marques do Nascimento

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este anexo se realizará la explicación de la planificación temporal y económica relacionada con el proyecto. Adicionalmente se comentarán las principales leyes y normativas vinculadas con este.

A.2. Planificación temporal

Para la organización temporal del proyecto no se utilizó ninguna metodología específica en su desarrollo sino un enfoque incremental e iterativo. Inicialmente, cada dos semanas tenía lugar una reunión con el objetivo de comentar los avances obtenidos hasta el momento y marcar nuevas propuestas a alcanzar para futuras ocasiones; en los últimos meses de desarrollo, las reuniones fueron realizadas cada semana.

La planificación temporal se realizó de la siguiente manera:

- **23/01/2024 - 06/02/2024**: Se comenzó realizando cuatro tareas principales:
  - Búsqueda bibliográfica de proyectos relacionados con el tema a abarcar
  - Búsqueda de bases de datos biológicas en abierto.
  - Primera toma de contacto con el paquete de *Biopython*.
  - Se comenzó con la creación del repositorio de *GitHub* correspondiente.

- **06/02/2024-20/02/2024**
  - Selección de algunos de los artículos encontrados, con la intención de que fueran tomados como referencia y pendientes de una lectura más exhaustiva. Alguno de los artículos sería: [Chen et al., 2022].
  - Prolongación en la búsqueda de alguna base de datos biológica en abierto complementaria a *NCBI*, como fue el caso de *TCGA*.
  - Continuación con la experimentación del paquete *Biopython*, creación del *notebook Pruebas iniciales TFG*.
  - Estudio de herramientas adicionales y compatibles con *Biopython*, como es *TCGAIntegrator*.
- **20/02/2024-05/03/2024**
  - Familiarización e intento de la descarga de datos controles y patológicos desde *NCBI* y *TCGA*.
  - Documentación y formación de las herramientas disponibles en el campo de la bioinformática.
  - Planificación del desarrollo del proyecto, con las estrategias a seguir y técnicas que se aplicarán.
- **05/03/2024-19/03/2024**
  - Reunión para la explicación de la base de datos de *TCGA*.
  - Comienzo y recopilación de documentación para la realización de la memoria.
  - Estudio de posible aplicación complementaria del lenguaje de `R`.
- **19/03/2024-02/04/2024**
  - Descarga completa de datos patológicos en archivos de formato `fasta` procedentes de *NCBI*.
  - Creación del proyecto en *Overleaf* mediante la plantilla correspondiente. Adición de memoria realizada hasta la fecha, junto con bibliografía e imágenes.
  - Primera toma de contacto con *Overleaf* y el lenguaje de LaTeX.
  - Continuación del apartado de introducción del proyecto y la memoria.
  - Comienzo de script de`R`.

- **02/04/2024-16/04/2024**
  - Comienzo de tratamiento de datos. Creación de un primer *notebook*, *Alineamiento de secuencias*.
  - Desarrollo de script de `R`.
  - Continuación con de la memoria.
- **16/04/2024-30/04/2024**
  - Reanudación del *notebook* de lenguaje `Python` *Alineamiento de secuencias*.
  - Descarga y configuración de algoritmos `Clustal` y `Muscle` para alineamiento múltiple de datos
  - Continuación y finalización del desarrollo de script de R.
- **30/04/2024-14/05/2024**
  - Creación y desarrollo de script de `R` en *Google Colab* debido a imprevistos en la actualización de paquetes.
  - Búsqueda de nuevas secuencias control.
  - Continuación con la memoria.
  - Obtención de alineamientos múltiples de secuencias patológicas.
- **14/05/2024-28/05/2024**
  - Descarga de secuencias control y tratamiento sobre el *notebook* desarrollado.
  - Obtención de alineamientos múltiples de controles.
  - Obtención de secuencias consenso patológicas y controles mediante *Biopython*.
- **28/05/2024-11/06/2024**
  - Comienzo de la etapa analítica sobre el *notebook* implementado, renombrado *Reconocimiento de biomarcadores*.
  - Estudio de la herramienta de `MSABrowser` y `NCBI MSAViewer` para visualización.
  - Continuación con la memoria.
- **11/06/2024-17/06/2024**

Dispositivo	Coste en €	Coste amortizado en €
Ordenador portátil HP	785 €	63,60 €

Tabla A.1: Costes hardware del proyecto realizado

  - Comienzo de documentación relativa a anexos en *Overleaf* y continuación de la memoria.
  - Obtención y procesado de tablas de variantes procedentes de *gnomAD*.
  - Finalización de la etapa de análisis.

- **17/06/2024-03/07/2024**
  - Continuación y finalización de documentación de anexos y memoria.
  - Finalización del *notebook Reconocimiento de biomarcadores* implementado.

Planificación económica

Para la planificación económica del proyecto, se han calculado los costes hardware, software y personales invertidos en el desarrollo del trabajo:

- **Costes Hardware:** No se ha tenido que adquirir ningún nuevo dispositivo hardware, y el único empleado ha sido el ordenador portátil del estudiante (*HP* 15s-fq2039ns con 8 GB de RAM y un procesador *Intel*© Core™ i7 de octava generación y 1.8GHz) [HP, 2024]. El portátil mencionado, se encuentra en el mercado a unos 785 euros. Cabe destacar que el equipo tiene una antiguedad de 6 años y 6 meses, de forma que su coste amortizado a lo largo de estos 6 meses de desarrollo sería:

$$\begin{aligned} \frac{785\ €}{6{,}5\ \text{años} \times 12\ \text{meses/año}} &= 10{,}06\ €/\text{mes} \\ 10{,}60\ €/\text{mes} \times 6\ \text{meses} &= 63{,}60\ € \end{aligned} \tag{A.1}$$

- **Costes Software:** En la elaboración del proyecto no ha sido necesaria la compra o suscripción de ninguna aplicación o licencia. Pero en su día con la compra del dispositivo, también tuvo lugar la adquisición

Software	**Coste en €**
Windows 11	11,15€
Office 365	42€
Coste total	53,15€

Tabla A.2: Costes software del proyecto realizado

Concepto	**Coste en €**
Salario mensual neto	427,06€
Seguridad Social (28,3 %)	228,95€
Retención IRPF (19 %)	153,71€
Salario mensual bruto	809,72€
Coste total	**4858,3€**

Tabla A.3: Costes personales del proyecto realizado

del sistema operativo, *Windows 10* en el momento, y en la actualidad, *Windows 11*, unos 145€ [Microsoft, 2024].

$$\frac{145\ €}{6{,}5\ \text{años} \times 12\ \text{meses/año}} = 1{,}86\ €/\text{mes}$$
$$1{,}86\ €/\text{mes} \times 6\ \text{meses} = 11{,}15\ € \tag{A.2}$$

Adicionalmente, y como herramienta de apoyo, se ha empleado el almacenamiento en la nube de *Onedrive* y las diferentes aplicaciones del paquete *Office 365*, ya sea para la escritura de la documentación, o herramienta de comunicación con los tutores. Siendo estudiante, el coste de la licencia es inexistente, ya que la *UBU* proporciona este servicio. En caso de no presentar esta ventaja, el uso del software por su parte sería de 7€/mes, unos 42€ en total [Microsoft, 2024].

- **Costes personales:** A lo largo de estos 6 meses, se habrán invertido en el desarrollo del proyecto aproximadamente unas 350 horas, que repartidas en 6 meses quedarían como unas 58,3 horas. Dentro del territorio español, un ingeniero biomédico puede cobrar un salario mensual bruto de unos 2500€ [Indeed, 2022].

  Si la media de horas de trabajo por mes son de unas 180 horas, entonces se cobraría a 13,89€/hora. De forma que al mes se cobrarían unos 809,72 € brutos, 4858,30€ por los 6 meses. Repartido con impuestos quedaría de la siguiente manera:

- **Costes totales**: Los costes totales serían los siguientes:

Concepto	Coste en €
Coste Hardware	63,60€
Coste Software	53,15€
Costes personales	4858,30€
Coste total	**4975,05€**

Tabla A.4: Costes totales del proyecto realizado

Viabilidad legal

En este apartado se tratarán los aspectos legales relacionados con el estudio desarrollado. Para la viabilidad legal del estudio no fue necesario la realización de ningún formulario.

Legislación vigente

Al tratarse de un estudio bioinformático, y en caso de existir alguna interacción con la legislación, este estaría relacionado con la normativa española en materia de acceso y utilización de recursos genéticos. Esta viene definida a nivel de territorio español por la *ley 42/2007 de Patrimonio Natural y de la Biodiversidad*, tras su modificación por la *Ley 33/2015*; y el *Real Decreto 124/2017, de 24 de febrero*, la cual regula el acceso a taxones silvestres y su utilización. Estas normativas garantizan el cumplimiento del *Protocolo Nagoya* y el *Reglamento (UE) Nº 511/2014*, por parte de la nación española [Ministerio para la Transición Ecológica y el Reto Demográfico, 2024]. Según estas se requeriría de la autorización para acceso otorgada por la autoridad competente, en esta ocasión, por el *NCBI*. Pero el *NCBI* a tratarse de una base de datos en abierto, no se requiere de ningún consentimiento expreso por su parte para la utilización de su información contenida para fines no lucrativos. Aclarar, a su vez, que no ha sido necesario la realización de ningún formulario.

Licencias

Todas las herramientas empleadas en el proyecto son de de código abierto. Entre ellas destacamos los *Jupyter notebooks* implementados, donde su licencia, junto a las de los paquetes correspondientes utilizados, son según la Tabla A.5:

Librería	**Licencia**
statistics	GNU General Public License v3.0
matplotlib	MDT License
numpy	BSD-3-Clause License
pandas	BSD-3-Clause License
Biopython	BSD-3-Clause License
subprocess	PSF License
random	PSF License
warnings	PSF License
panel	BSD-3-Clause License
logomaker	MIT License

Tabla A.5: Librerías utilizadas en el proyecto, junto con sus licencias.

Apéndice B

Documentación de usuario

B.1. Requisitos software y hardware para ejecutar el proyecto.

Para la realización de este proyecto se requirió de la instalación de ciertos *softwares*. Algunos de ellos ya instalados previamente por el estudiante para anteriores ocasiones. y otros que no, necesitando incluso una configuración y ciertas medidas que fueron tomadas para su correcto funcionamientos.

Anaconda navigator [Anaconda, 2024] es uno de los principales requisitos y necesario para la utilización de *Jupyter notebook* [Jupyter Team, 2015]. En este, se ejecutó el código mediante un núcleo o kernel central, que presenta por defecto `Python` [Python Software Foundation, 2024] como lenguaje de programación. Existe la posibilidad de trabajar con otros lenguajes de programación diferentes, pero la librería y paquetes a utilizar pertenecen al primer lenguaje nombrado.

Ciertos *softwares*, correspondientes a algoritmos de alineamiento múltiple de secuencias tuvieron que ser instalados. Cada uno de los cuales se basa en una metodología distinta, de forma progresiva como es el caso de `ClustalW` [Dineen, 2016], y de forma iterativa para el caso de `Muscle` [Edgar, 2021].

Se recomienda a los miembros del tribunal o futuros estudiantes que quieran reutilizar o comprobar los resultados conseguidos, emplear los mismos medios y *softwares* comentados en los próximos apartados, a pesar de la existencia de otros alternativos, para una correcta y asegurada ejecución. Destacar a su vez que el proyecto desarrollado puede realizarse en otros dispositivos con unos sistemas operativos distintos.

B.2. Instalación / Puesta en marcha

Como se mencionó en el anterior apartado *Anaconda navigator*, es la base sobre la cual se utilizó *jupyter notebook* y se elaboró el código del proyecto, por ello sería la primera instalación que se tendría que realizar.

Se comienza visitando la página web oficial de *Anaconda.org*, donde una vez ya se encuentre en ella, se deberá seccionar la opción de *Free Download*. Esto reconducirá a una nueva página, dónde habrá que introducir un correo electrónico para poder realizar la descarga y hacer uso de la estación de trabajo. Al introducir el correo, se mandará un enlace para la descarga a este, con el que se podrá acceder a una página, donde finalmente permita la elección del sistema operativo para la instalación.

Una vez descargado y ejecutado el archivo correspondiente, se define la ruta de instalación en el dispositivo y se seleccionan las distintas opciones deseadas.

Terminada ya la instalación, se podría iniciar la App y ejecutar como administrador para que la futura descarga de librerías no ocasionara problemas. También se podría hacer la creación de entornos virtuales que se consideren oportunas, para una mayor optimización de los errores que pudieran haber.Adicionalmente se podría instalar la aplicación de *Powershell Prompt*, para una mayor comodidad del programador, al permitir el uso de la consola desde el navegador instalado (Figura B.1).

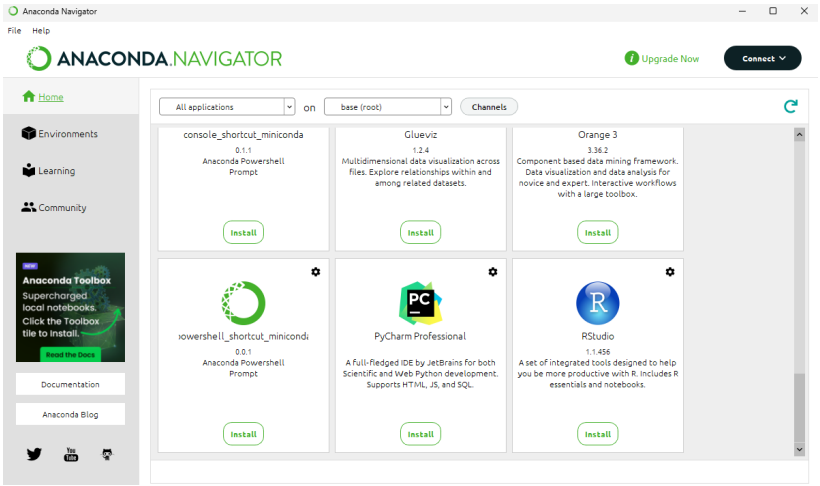


Figura B.1: Estación de inicio de *Anaconda*. Fuente: elaboración propia.

La instalación de *Jupyter notebook* con *Anaconda navigator* previamente instalado es muy sencilla. Como se puede ver en la Figura B.2 , *Jupyter notebook* aparece como una de las aplicaciones por defecto de la estación de trabajo, de forma, que únicamente se requeriría seleccionar su descarga, mediante la opción visible que presenta *Anaconda*.

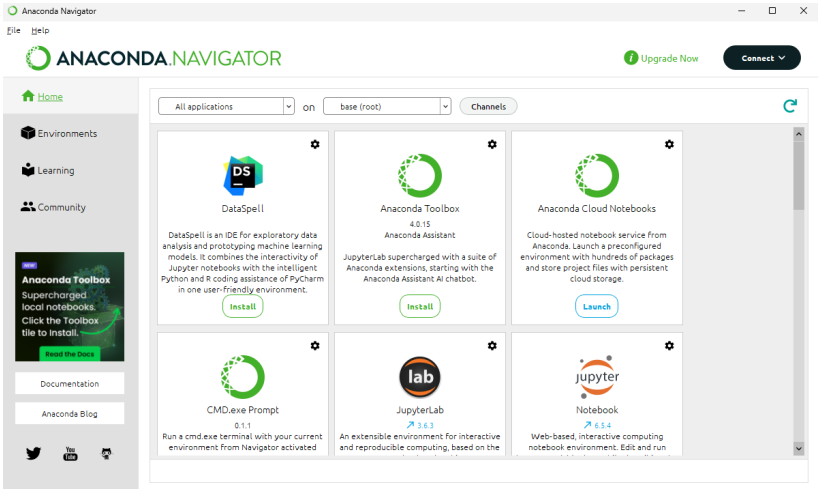


Figura B.2: Estación de inicio de *Anaconda* con la aplicación de *jupyter notebook* instalada. Fuente: elaboración propia.

Dentro de *Jupyter notebook* y mediante la creación de un *notebook* dedicado, se podría instalar el paquete central del proyecto, *Biopython* [Chang et al., 2024]. Haciendo uso de la funcionalidad *pip install* y el nombre *biopython*, se instala el paquete nombrado (Figura B.3). Con el comando *pip show* se mostraría la información más relevante del paquete y se utilizaría como vía para la comprobación de la instalación correcta de este (Figura B.4).

Adicionalmente al paquete de *Biopython*, se requiere de la instalación/carga de unos cuantos más. Necesarios todos ellos para la correcta ejecución del *notebook* implementado. En la Figura B.5 se presentan los imports necesarios para la ejecución del *notebook*.

Una vez se tenga el entorno de programación junto con los paquetes de interés instalados, se podrían instalar los *softwares* relativos a los algoritmos de alineamientos.

Comenzando por `ClustalW` o `Clustal Omega`, y al igual que se realizó con *Anaconda*, se debe visitar la página de *clustal.org* (Figura B.6),

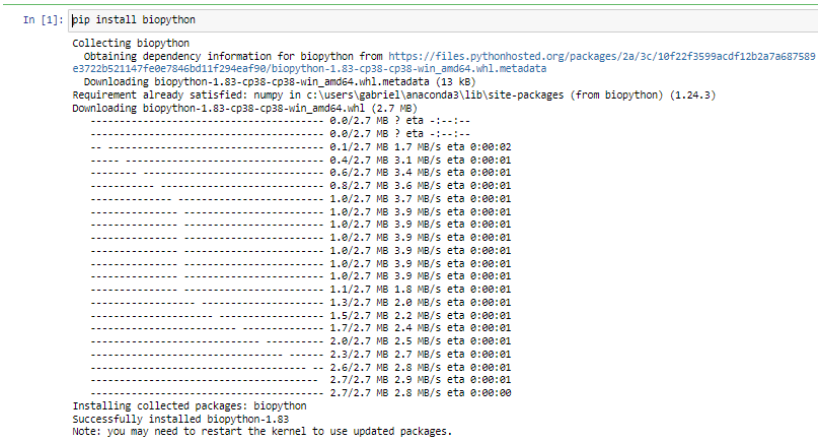


Figura B.3: Instalación del paquete *Biopython* en *jupyter notebook*. Fuente: elaboración propia.

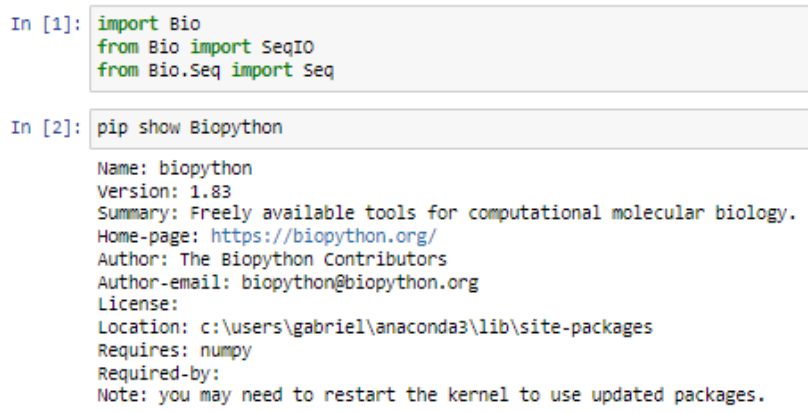


Figura B.4: Comprobación de la instalación del paquete *Biopython* en *Jupyter notebook*. Fuente: elaboración propia.

y ya en ella seleccionar el algoritmo de última versión, `Clustal Omega` [Dineen, 2016]. Nuevamente, en la página emergente, se lleva a cabo una selección del enlace a utilizar según el sistema operativo utilizado. Una vez sea descargado, se extraerá mediante un descompresor, dando lugar a numerosos archivos (Figura B.7).Entre los archivos, encontraremos un documento `txt` que junto con la consola del dispositivo, nos permitirá terminar la instalación configurando nuestro software para su uso.

El seguimiento del `txt`, depende del sistema operativo, y ha demostrado ser ambiguo al contener unas instrucciones como específicas y nombres

```
import Bio
from Bio import SeqIO
from Bio.Seq import Seq
from Bio.SeqUtils import gc_fraction
from Bio.SeqRecord import SeqRecord
from Bio.Align import MultipleSeqAlignment
from Bio import AlignIO
from Bio import Align
from Bio.Align import AlignInfo
import statistics as stats
import subprocess
import warnings
import random
import pandas as pd
import numpy as np
import panel as pnplt
import panel.widgets as pnw
from bokeh.plotting import figure
from bokeh.models import ColumnDataSource, Plot, Grid, Range1d
from bokeh.models.glyphs import Text, Rect
from bokeh.layouts import gridplot
from collections import Counter
from matplotlib import colors
import logomaker
import matplotlib.pyplot as plt
```

Figura B.5: Bloque con las importaciones necesarias para el funcionamiento del *notebook*. Fuente: elaboración propia.

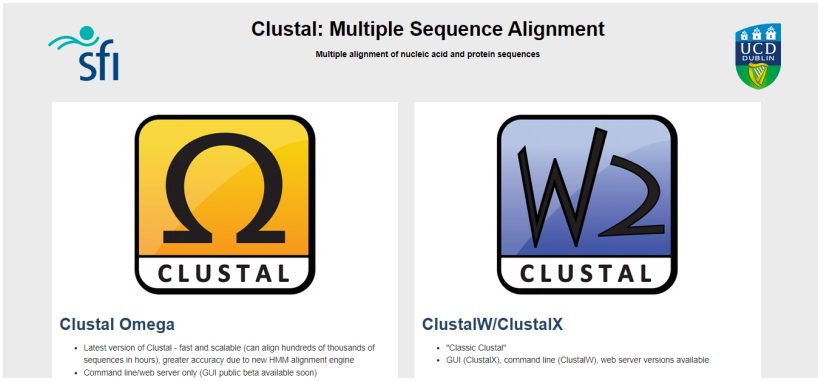
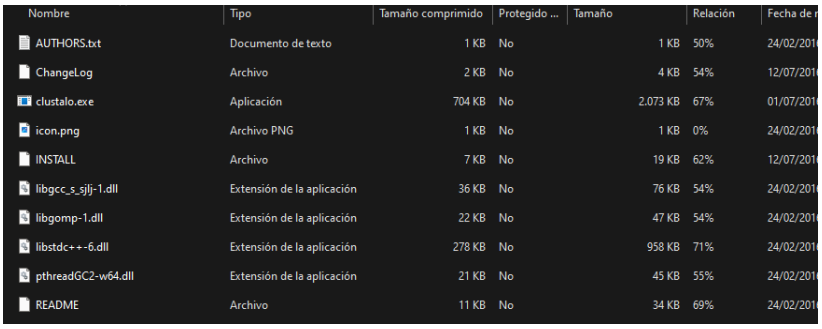


Figura B.6: Página inicial del *software* de `Clustal`. Fuente: [Dineen, 2016].

de archivos en ellas inexistentes en la carpeta descomprimida, lo cual ha dificultado su instalación. Por tanto, se recomienda seguir las instrucciones en la medida de lo posible, crear una carpeta con los archivos contenidos del archivo previamente comprimido, en el mismo directorio que el *notebook* de `Python` en el que se pretenda ejecutar los alineamientos múltiples de



Nombre	Tipo	Tamaño comprimido	Protegido ...	Tamaño	Relación	Fecha de r
AUTHORS.txt	Documento de texto	1 KB	No	1 KB	50%	24/02/201
ChangeLog	Archivo	2 KB	No	4 KB	54%	12/07/201
clustalo.exe	Aplicación	704 KB	No	2.073 KB	67%	01/07/201
icon.png	Archivo PNG	1 KB	No	1 KB	0%	24/02/201
INSTALL	Archivo	7 KB	No	19 KB	62%	12/07/201
libgcc_s_sjlj-1.dll	Extensión de la aplicación	36 KB	No	76 KB	54%	24/02/201
libgomp-1.dll	Extensión de la aplicación	22 KB	No	47 KB	54%	24/02/201
libstdc++-6.dll	Extensión de la aplicación	278 KB	No	958 KB	71%	24/02/201
pthreadGC2-w64.dll	Extensión de la aplicación	21 KB	No	45 KB	55%	24/02/201
README	Archivo	11 KB	No	34 KB	69%	24/02/201

Figura B.7: Archivos generados de la descompresión del *software* de `Clustal`. Fuente: elaboración propia.

secuencia. También sería necesario redirigir los archivos de extensión `.batch` y actualizar las variables de entorno del sistema (`Path`), a través de la adición de la ruta donde se encuentra la carpeta de `Clustal` creada con anterioridad.

Tras la completa configuración del algoritmo de `Clustal`, y con la ejecución de un bloque de código con los comandos necesarios, el algoritmo de `Clustal` ya podría utilizarse en el *notebook* de `Python` deseado.

Para la instalación de `Muscle` se planteará una estrategia similar a la empleada para `Clustal`. Se iniciaría accediendo a la página web de *drive5* (Figura B.8), *drive.com* [Edgar, 2021]. Una vez en ella se seleccionaría la opción de `Muscle`, la cual daría lugar a una nueva página con información relativa, además de la opción para descargarlo. Al seleccionarla, se actualizará la página, redirigiéndose a un repositorio de *Github* con las diferentes versiones disponibles a descargar.

Al finalizar el proceso de descarga, debe ser descomprimido, dando lugar a cierto número de archivos. Al contrario que `Clustal`, entre los archivos de `Muscle` no habrá ninguno sin extensión de nombre `INSTALL`, sino que habrá un archivo de nombre `README.md`, que contendrá un enlace al manual de `Muscle` situado en la misma página web recientemente visitada (Figura B.9). Siguiendo su configuración e instalación, se debería crear una carpeta en el mismo directorio que el *notebook* utilizado, al cual será redirigido su archivo `exe`. Finalmente, al igual que con `Clustal`, se editarían las variables de entorno del sistema (`Path`), incluyendo la ruta de la carpeta de `Muscle` creada.

Figura B.8: Página inicial del *software* de `Muscle`. Fuente: [Edgar, 2021]

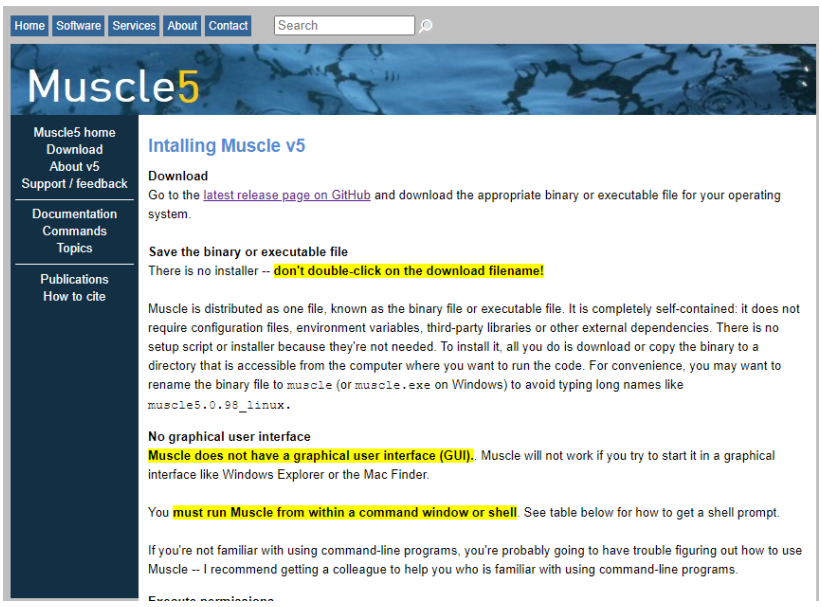


Figura B.9: Página con el manual de instalación del *software* de `Muscle`. Fuente: [Edgar, 2021]

B.3. Manuales y/o Demostraciones prácticas

El uso de *Jupyter notebook* está más que justificado, debido a que gran parte del proyecto se encuentra desarrollado en un *notebook* con un kernel propio de `Python 3`, llamado Reconocimiento de biomarcadores con

Biopython. En él, tiene lugar el desarrollo de un conjunto de bloques de código y narrativos, con la finalidad de justificar y proponer una solución al problema planteado. Además, el uso de estos llamados *notebooks*, no solo se limitaron a uno, sino que tuvo lugar la creación de alguno adicional para la práctica y afianzamiento de las posibilidades presentadas en el paquete estudiado.

A lo largo de los *notebooks* elaborados, se empleó el paquete de *Biopython*. Aquel el cual facilitó y permitió el análisis de las secuencias controles y patológicas, permitiéndonos definir a través de diversas funciones el diagnóstico o estado clínico de las posibles mutaciones que pudieran presentar.

Cabe destacar, que la demostración práctica de ciertos paquetes que previamente han debido de ser instalados, representados en la Figura B.5, se encuentran explicados en el apartado de Tratamiento en el anexo de Datos.

Primeramente, antes del comienzo de los análisis y obtención de los resultados del proyecto, se realizó la creación de las siguientes funciones `realizar_alineamiento` y `obtener_alineamientos`. Donde, la primera de ellas, hace uso del paquete `subprocess` [Python Software Foundation, 2023], encargado de la ejecución de nuevos programas desde el *notebook* de `Python` implementado, y su función `run()`, para la realización de los diferentes alineamiento múltiples. Escogiendo el algoritmo a ejecutar, la ruta de origen del archivo con las secuencias y la ruta de destino del archivo resultado, se genera un archivo de extensión `fasta` con los correspondientes alineamientos. Para la segunda función, se hace uso del paquete `AlignIO` y su función `read()`, para la lectura y retorno de todos los alineamientos presentes en el archivo de una sola vez. Los ejemplos de las funciones mencionadas se muestran en la Figura B.10 y Figura B.11.

```
realizar_alineamiento("clustalo","Datos\Secuencias patológicas filtradas\BRCA1 exon 5.fasta","Resultados\Alineamiento secuencias
realizar_alineamiento("clustalo","Datos\Secuencias patológicas filtradas\BRCA1 exon 19.fasta","Resultados\Alineamiento secuencia

realizar_alineamiento("clustalo","Datos\Secuencias patológicas filtradas\BRCA2 exon 2.fasta","Resultados\Alineamiento secuencias
realizar_alineamiento("clustalo","Datos\Secuencias patológicas filtradas\BRCA2 exon 11.fasta","Resultados\Alineamiento secuencia

realizar_alineamiento("clustalo","Datos\Secuencias patológicas filtradas\TP53 exon 5.fasta","Resultados\Alineamiento secuencias
realizar_alineamiento("clustalo","Datos\Secuencias patológicas filtradas\TP53 exon 8.fasta","Resultados\Alineamiento secuencias

realizar_alineamiento("clustalo","Datos\Secuencias patológicas filtradas\PIK3CA exon 9.fasta","Resultados\Alineamiento secuencia
realizar_alineamiento("clustalo","Datos\Secuencias patológicas filtradas\PIK3CA exon 20.fasta","Resultados\Alineamiento secuenci
```

Figura B.10: Código correspondiente a la realización de los diferentes alineamientos mediante el algoritmo de `Clustal`. Fuente: elaboración propia.

Por tanto, tras la aplicación de ambas funciones mediante varias llamadas, se obtiene tanto para controles (Figura B.12), como patológicos (Figura B.13), su *MSA* (Multiple Sequences Alignment) de `Muscle` y `Clustal`.

```
p_c_algn_brca1_exon5=obtener_alineamientos("Resultados\Alineamiento secuencias patológicas\Resultados alineamiento clustal\clust
p_c_algn_brca1_exon19=obtener_alineamientos("Resultados\Alineamiento secuencias patológicas\Resultados alineamiento clustal\clus

p_c_algn_brca2_exon2=obtener_alineamientos("Resultados\Alineamiento secuencias patológicas\Resultados alineamiento clustal\/clus
p_c_algn_brca2_exon11=obtener_alineamientos("Resultados\Alineamiento secuencias patológicas\Resultados alineamiento clustal\clus

p_c_algn_tp53_exon5=obtener_alineamientos("Resultados\Alineamiento secuencias patológicas\Resultados alineamiento clustal\clusta
p_c_algn_tp53_exon8=obtener_alineamientos("Resultados\Alineamiento secuencias patológicas\Resultados alineamiento clustal\clusta

p_c_algn_pik3ca_exon9=obtener_alineamientos("Resultados\Alineamiento secuencias patológicas\Resultados alineamiento clustal\clus
```

Figura B.11: Almacenamiento de los alineamiento `Clustal` realizados en variables globales. Fuente: elaboración propia.

```
>OR780020.1 Homo sapiens isolate n18 BRCA1 (BRCA1) gene, exon 5 and partial cds
--------------TCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTT
AAAAGGTTGATAATCACTTGCTGAGTGTGTTTCTCAAACAATTTAATTTCAGGAGCCTAC
AAGAAAGTACGAGATTTAGTCAACTTGTTGAAGAGCTATTGAAAATCATTTGTGCTTTTC
AGCTTGACACAGGTTTGGAGTGTAAGTGTTGAATATCCCAAGAATGACACTCAAGTGCTG
TCCATGAAAACTCAGGAAGTTTGCACAATTACTTTCTATGACGTGGTGATAAGACCTTTT
AGTCTAGGTTAATTTTAGTTCTGTATCTGTAATCTATTTTTAAAAAATTACTCCCACTGG
TCTCACACC---
>OR780019.1 Homo sapiens isolate b2 BRCA1 (BRCA1) gene, exon 5 and partial cds
CTACTGTTGCTGCATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTT
AAAAGGTTGATAATCACTTGCTGAGTGTGTTTCTCAAACAATTTAATTTCAGGAGCCTAC
AAGAAAGTACGAGATTTAGTCAACTTGTTGAAGAGCTATTGAAAATCATTTGTGCTTTTC
AGCTTGACACAGGTTTGGAGTGTAAGTGTTGAATATCCCAAGAATGACACTCAAGTGCTG
TCCATGAAAACTCAGGAAGTTTGCACAATTACTTTCTATGACGTGGTGATAAGACCTTTT
AGTCTAGGTTAATTTTAGTTCTGTATCTGTAATCTATTTTTAAAAAATTACTCCCACTGG
TCTCACACC---
>OR780018.1 Homo sapiens isolate n8 BRCA1 (BRCA1) gene, exon 5 and partial cds
-------------ATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTT
AAAAGGTTGATAATCACTTGCTRAGTGTGTTTCTCAAACAATTTAATTTCAGGAGCCTAC
AAGAAAGTACGAGATTTAGTCAACTTGTTGAAGAGCTATTGAAAATCATTTGTGCTTTTC
AGCTTGACACAGGTTTGGAGTGTAAGTGTTGAATATCCCAAGAATGACACTCAAGTGCTG
```

Figura B.12: Alineamientos de las secuencias control del exón 5 de *BRCA1*. Fuente: elaboración propia.

```
>MG932487.1 Homo sapiens clone I103_B25_EXON5-F_E07 breast cancer type 1 susceptibility protein
(BRCA1) gene, exon 5 and partial cds
---------ATTCTTTCTACAAAAGGAAGTAA-ATTAAATTGTTCTTTCTTTCTTTATAA
TTTATAGATTTTGCATGCTGAAACTTCTCAACCAGAAGAAAGGGCCTTCACAGTGTCCTT
TATGTAAGAATGATATAACCAAAAGGTATATAATTTGGTAATGATGCTAGGTTGGAAGCA
ACCACAGTAGGAAAAAGTAGAAATTATTTAATAACATAGCGTTCCTATAAAACCATTCAT
CAGAAAAATTTATAAAAGAGTTTTTAGCACACAGTAAATTATTTCCAAAGTTATTTTCCT
GAAAGTTTTATGGGACATCTGCCTTATACAGGTATTAGAAACTTACTGCCTTTCTCTAAT
GC------
>MG932486.1 Homo sapiens clone I103_P14_EXON5-F_G02 breast cancer type 1 susceptibility protein
(BRCA1) gene, exon 5 and partial cds
--------------TTCTACAAAAGGAAGTAA-ATTAAATTGTTCTTTCTTTCTTTATAA
TTTATAGATTTTGCATGCTGAAACTTCTCAACCAGAAGAAAGGGCCTTCACAGTGTCCTT
TATGTAAGAATGATATAACCAAAAGGTATATAATTTGGTAATGATGCTAGGTTGGAAGCA
ACCACAGTAGGAAAAAGTAGAAATTATTTAATAACATAGCGTTCCTATAAAACCATTCAT
CAGAAAAATTTATAAAAGAGTTTTTAGCACACAGTAAATTATTTCCAAAGTTATTTTCCT
GAAAGTTTTATGGGACATCTGCCTTATACAGGTATTAGAAACTTACTGCCTTTCTCTAAT
GCAAT---
>MG932485.1 Homo sapiens clone I103_P13_EXON5-F_A06 breast cancer type 1 susceptibility protein
(BRCA1) gene, exon 5 and partial cds
---------ATTCTTTCTACAAAAGGAAGTAA-ATTAAATTGTTCTTTCTTTCTTTATAA
```

Figura B.13: Alineamientos de las secuencias patológicas. Fuente: elaboración propia.

Para la obtención de secuencias consenso, que posteriormente serían comparadas, también se hace uso de diferentes objetos y funciones que pre-

senta el paquete *Biopython*. Las funciones de `obtener_consenso_archivo`, `obtener_consenso` y `guardar_consenso`, fueron las encargadas, de que a partir de un *MSA* realizado, se obtuviera y guardara en el dispositivo su secuencia consenso representativa (Figura B.14). Se desarrolló complementariamente, `crear_MultipleSeqAlignment`, una alternativa para la creación de un objeto *MSA* a partir de un archivo dado, que se daría uso con la llamada de una de las tres anteriores.

```
>P_C_BRCA1_5 consensus patologic clustal BRCA1 exon5
CTTTTGAGTATTCTTTCTACAAAAGGAAGTAAXATTAAATTGTTCTCTTCTTTCTTTATA
ATTTATAGATTTTGCATGCTGAAACTTCTCAACCAGAAGAAAGGGCCTTCACAGTGTCCT
TTATGTAAGAATGATATAACCAAAAGGTATATAATTTGGTAATCTTGTTGAAGAGCTATT
GAAAATCATTTGTGATGCTAGGTTGGAAGCAACCACAGTAGGAAAAAGTAGAAATTATTA
ATAATAACATAGCGTTCCTATAAAACCATTCATCAGAAAAATTTATAAAAGAGTTTTTAG
CACACAGTAAATTATTTCCAAAGTTATTTTCCTGAAAGTTTTATGGGACATCTGCCTTAT
ACAGGTATTAGAAACTTACTGCCTTTGCTCTAATGCAAXAAA
```

Figura B.14: Secuencia consenso patológica ejemplo del exón 5 del gen *BRCA1* mediante `Clustal`. Fuente: elaboración propia.

Las secuencias consenso determinadas, como la mostrada en la Figura B.14, serían almacenadas en archivos `fasta`, con un identificador y descripción asociados.

La realización y guardado de los alineamientos y secuencias consenso, permitiría finalmente iniciar la etapa de análisis. Se analizaría en primer lugar, las secuencias de forma independiente.

Las funciones `num_nucleotidos`, `crear_barras_nucleotidos`, `num_aa` y `crear_barras_aa` se usan para implementar una función conjunto llamada `analisis_secuencia`. A partir de la cual se conseguiría un estudio básico de una secuencia de nucleótidos pasada. El estudio de contenido en GC como parámetro presenta gran importancia en bioinformática, debido a que puede representar la estabilidad del ADN y ARN, la clasificación de algoritmos y comparación de genomas. También el contenido dentro de la secuencia, en especial para el caso que acontece, pudiendo servir como una medida de la heterogeneidad contenida en el alineamiento múltiple. Y por último, y si existe la posibilidad, el contenido de aminoácidos al cual daría lugar, reflejo de posibles cambios significativos a otros niveles genómicos.

Los análisis independientes de secuencias consenso patológicas, han demostrado que existen ciertas diferencias entre las secuencias consenso obtenidas mediante distintos algoritmos en algunas ocasiones. En los casos

de *BRCA1/5* , *BRCA2/2*, *PIK3CA/9*, *TP53/5* y *TP53/8*, se han obtenido resultados de análisis similares, cierta similitud de longitudes para ambas secuencias, un contenido similar y parecido en porcentaje de bases desconocidas. Mientras que para, *BRCA1/19* y *BRCA2/11* las diferencias se hacen visibles.Las longitudes de las secuencias presentan mayor diferencia de nucleótidos, el contenido es variable y sobre todo la cantidad de bases ambiguas/desconocidas contenidas en la secuencia es altamente heterogénea, respecto al resto. Las bases ambiguas en las secuencias consenso indican la variabilidad entre los alineamientos, lo cual afecta los resultados obtenidos. Esta variabilidad ha demostrado generar diferencias sustanciales en los resultados. Por lo tanto, frente a secuencias notablemente heterogéneas, la eficacia y el comportamiento de los algoritmos empleados se ven comprometidos. Esto subraya la importancia de seleccionar el algoritmo adecuado en función de la similitud entre las secuencias bajo estudio. Como se ha mencionado anteriormente, para *BRCA1/19* en la Figura B.15 y Figura B.16.

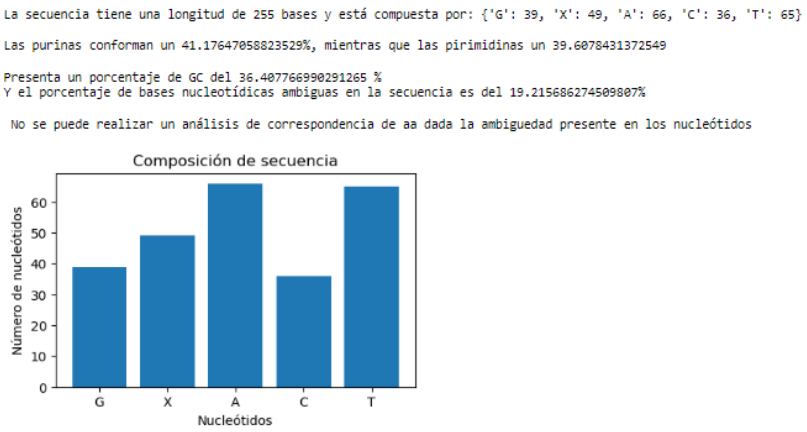


Figura B.15: Análisis de la secuencia consenso patológica del exón 19 del gen *BRCA1* mediante `Clustal`. Fuente: elaboración propia.

La variabilidad mencionada y justificada anteriormente entre los diferentes algoritmos, para el caso de las secuencias controles, es prácticamente inexistente. Para cada secuencia consenso control estudiada, tanto si se realiza un alineamiento con el algoritmo de `Muscle`, como con el de `Clustal`, se consiguen los mismos resultados. Longitudes de secuencia, contenidos de GC y cantidad de bases ambiguas similares. La variabilidad vista en el análisis de secuencias consenso patológicas no se encuentra presente de forma general, para estas secuencias consenso controles, lo cual justifica estos resultados. *BRCA2/11* es el único caso cuyos análisis dan a entender

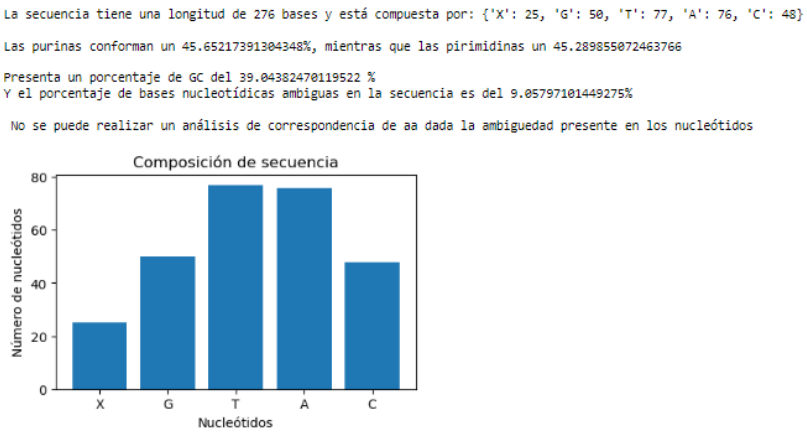


Figura B.16: Análisis de la secuencia consenso patológica del exón 19 del gen *BRCA1* mediante `Muscle`. Fuente: elaboración propia.

que las secuencias controles a partir de las cuales se genera la secuencia consenso, por pocas que sean, son heterogéneas (Figura B.17 y Figura B.18).

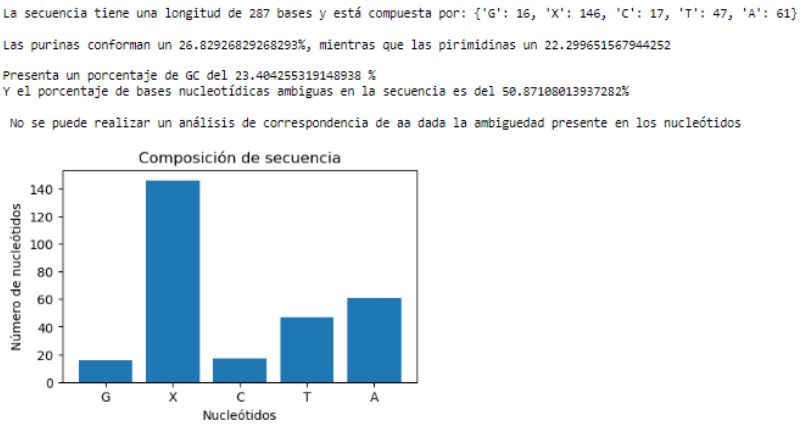


Figura B.17: Análisis de la secuencia consenso control del exón 11 del gen *BRCA2* mediante `Clustal`. Fuente: elaboración propia.

También se podría afirmar, aunque con menos confianza, que los algoritmos con un bajo número de secuencias a alinear, no presentan prácticamente diferencias en sus resultados. Un ejemplo de ello es el caso del ya mencionado *BRCA2/11*. Sus resultados, al compararlos con los obtenidos previamente en las secuencias consenso patológicas, serían similares si no fuera por la reducción de bases ambiguas observada al pasar del caso de `Clustal` a

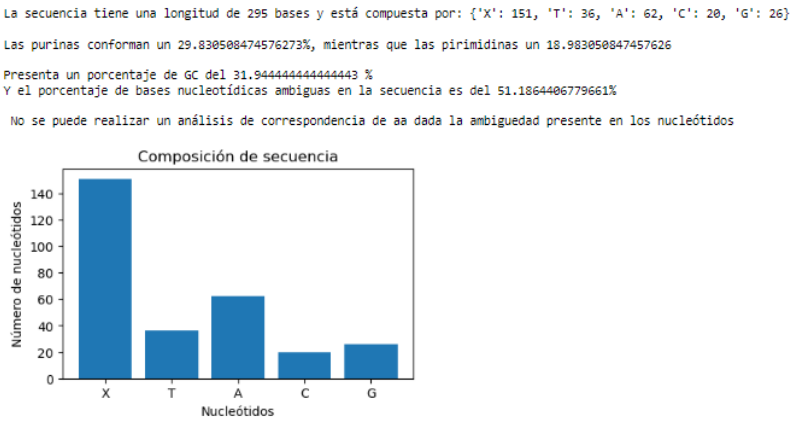


Figura B.18: Análisis de la secuencia consenso control del exón 11 del gen *BRCA2* mediante `Muscle`. Fuente: elaboración propia.

`Muscle`. En los casos control,para el mismo gen y exón, esta reducción no tiene lugar.

Al comparar el análisis básico de secuencias consenso entre patológica y controles, encontramos:

- *BRCA1/5*: Una longitud de secuencias consenso variable (unas 50 bases aproximadamente). Un ligero mayor contenido de GC en la secuencia control respecto a la patológica, lo cual afirma una lógica mayor estabilidad por parte de la secuencia conjunto sana. Además de una ligera disminución también en las controles del número de bases ambiguas presentes, es decir, representando unas bases nitrogenadas más conservadas (Figura B.20 y Figura B.19).
- *BRCA2/11*: Para esta ocasión la diferencia entre longitudes es extrema, llegando a haber una diferencia de más de 600 bases. Sin embargo el contenido de GC para ambas es muy semejante, siendo la diferencia entre porcentajes de algunas décimas. Por otro lado, las bases nucleotídicas ambiguas para la secuencia control de es aproximadamente la mitad del contenido, y del 28.5 % para la patológica; una situación poco lógica en lo que lo esperado habría sido justamente lo contrario (Figura B.18 y Figura B.21).
- *TP53/5*: La longitud en ambos casos no es la misma, pero son ciertamente similares. El contenido o porcentaje de GC en la secuencia es mayor para el caso control. Ninguno de ellos presenta base ambigua

alguna. Por último, la cantidad y tipos de aminoácidos, son muy distintos, debido en parte a la diferencia del número de bases existente (Figura B.22 y Figura B.23).

- *TP53/8*: Para el exón 8 de este mismo gen, encontramos longitudes muy diferentes, logrando incluso la secuencia patológica doblar la longitud de la secuencia control. El contenido de GC es ampliamente superior en la secuencia control. Para ambas, el contenido de bases ambiguas es nulo, y el gráfico de aa, al igual que en el anterior exón, es distinto en número y tipos (Figura B.24 y Figura B.25).

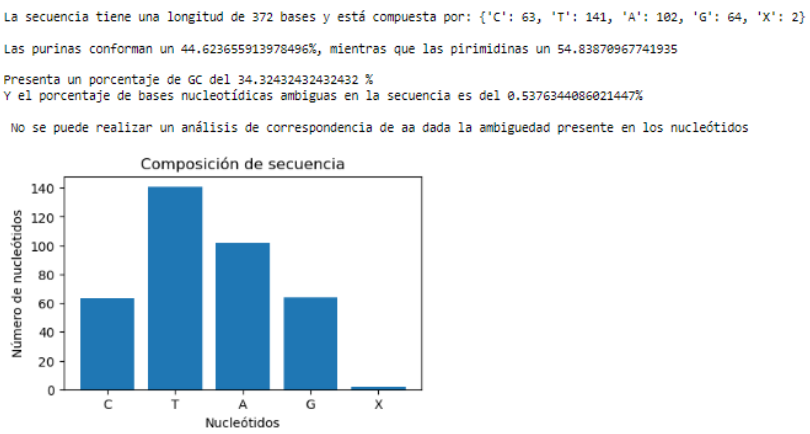


Figura B.19: Análisis de la secuencia consenso control del exón 5 del gen *BRCA1* mediante `Muscle`. Fuente: elaboración propia.

Posteriormente tiene lugar la comparativa de secuencias consenso mediante confrontación, es decir, alineamiento pareado. Seis funciones en total fueron creadas para el análisis de estas comparaciones. Dos de ellas, `encontrar_mejor_alineamiento` y `alineamiento`, destinadas a la realización del alineamiento mediante el manejo de las estructuras de los objetos y funciones del paquete `Align`; tanto desde una perspectiva global, como local. Cabe destacar, que para ambos alineamientos se les fueron definidos valores por defecto para una situación de coincidencia, diferencia, hueco de apertura y hueco de extensión. Tres más fueron empleadas para la representación de una comparativa visual de las secuencias, a través de un gráfico de puntos. Por último, existe una función llamada `comparacion_secuencias_consenso`, que agrupa y llama a las demás funciones, obteniendo así un resumen analítico visualmente atractivo.

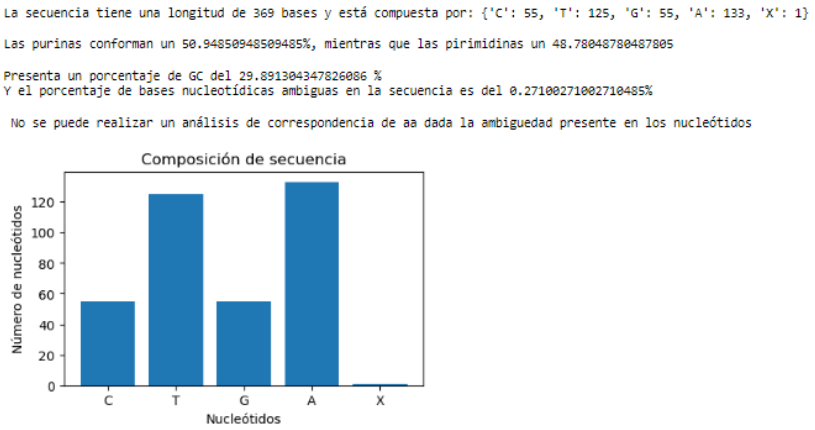


Figura B.20: Análisis de la secuencia consenso patológica del exón 5 del gen *BRCA1* mediante `Muscle`. Fuente: elaboración propia.

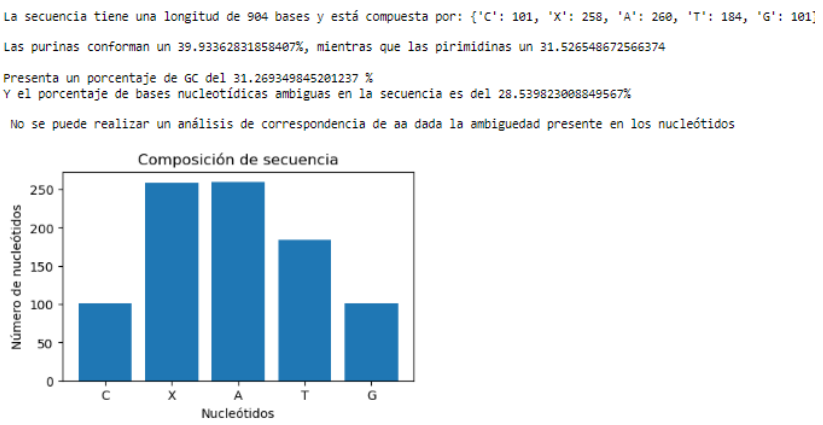


Figura B.21: Análisis de la secuencia consenso patológica del exón 11 del gen *BRCA2* mediante `Muscle`. Fuente: elaboración propia.

En teoría, teniendo en cuenta que las secuencias nucleotídicas a comparar representan un mismo exón de un mismo gen, el alineamiento de las secuencias comparadas debería de mostrar más similitudes que diferencias y *gaps*. Esto se debe a que la mayoría de las mutaciones que tienen lugar sobre las secuencias, sean malignas o benignas, son *SNV*(*Simple Nucleotide Variant*) y que la comparación realizada en esta situación, es sobre secuencias consenso. Se esperaba encontrar algo parecido a lo mostrado en la Figura B.26, pero con la presencia de ciertos *mismatches*, que representarían las posibles mutaciones más comunes que habrían tenido lugar dentro de la secuencia patológica.

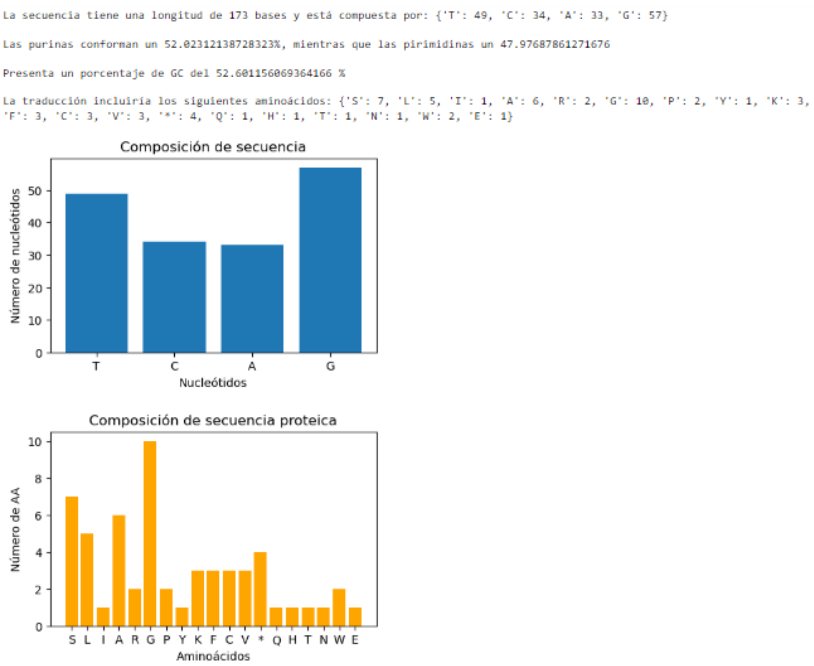


Figura B.22: Análisis de la secuencia consenso patológica del exón 5 del gen *TP53* mediante `Muscle`. Fuente: elaboración propia.

Pero, contrario a lo que en un principio cabría esperar, de forma general, los alineamientos globales de las secuencias presentan menores similitudes que los alineamientos locales (Figura B.27). Dichas puntuaciones de alineamientos son más propios de secuencias divergentes, que de secuencias similares; debido a que en un alineamiento local, se comparan fragmentos de las secuencias en vez de la secuencia completa. Por tanto, estos resultados revelan que las secuencias consenso patológicas y de control correspondientes para un mismo exón y gen, pero diferente diagnóstico, comparadas en el *notebook*, son más diferentes que similares.

Lo mismo sucede en la representación del gráfico *Dot plot*, donde se estimaba encontrar un gráfico que contuviera una diagonal en sentido descendente en su mayoría continua. Esta, representaría las coincidencias encontradas entre ambas secuencias para las distintas posiciones de las bases contenidas (Figura B.28).

Por otro lado, dependiendo del algoritmo empleado para calcular el alineamiento y sus respectivas secuencias consenso, se obtendrán unos mejores o peores resultados. En la mayoría de ocasiones y al igual que sucedió en los análisis de secuencias individuales, el algoritmo de `Muscle` presenta unos

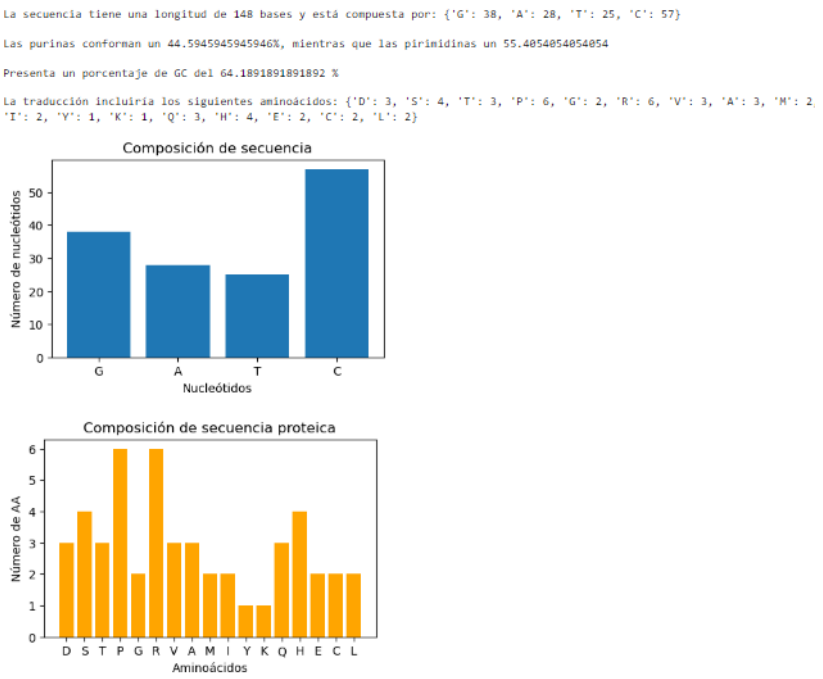


Figura B.23: Análisis de la secuencia consenso control del exón 5 del gen *TP53* mediante `Muscle`. Fuente: elaboración propia.

mejores resultados que el algoritmo de `Clustal`. Los ejemplos gráficos se incluyen en la Figura B.29, Figura B.30, Figura B.31 y Figura B.32.

Debido a las diferencias dadas, y a pesar de la gran utilidad que representa el alineamiento de secuencias, los resultados no son lo suficientemente buenos ni representativos de lo que se pretendía comprobar. El alineamiento mediante `Muscle` del gen *BRCA1* para el exón 5, es el resultado más parecido al tomado como idea o referencia. Mientras que otros, como los realizados mediante `Clustal` para el exón 5 del gen *TP53* o el exón 11 del gen *BRCA2*, presentan la mayoría de sus coincidencias entre bases ambiguas, resultando en los peores resultados.

Los alineamientos obtenidos, al presentar grandes longitudes, podrían llegar a dificultar la detección de biomarcadores de una forma visual, posibilitando la pérdida de alguno de ellos. Por tanto, de forma alternativa se implementaron un conjunto de funciones, cuya función era automatizar de una forma relativamente eficiente, la detección de tres tipos de mutaciones comúnes a nivel de secuencia:

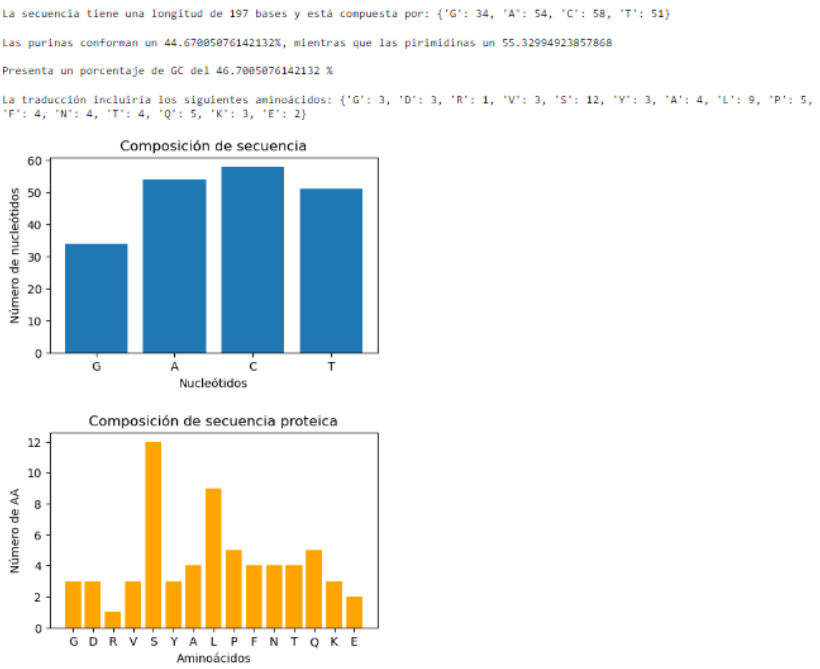


Figura B.24: Análisis de la secuencia consenso patológica del exón 8 del gen *TP53* mediante `Muscle`. Fuente: elaboración propia.

- Inserciones (incluido duplicaciones): una introducción atípica de material genético adicional, que puede provocar graves consecuencias en la secuencia. Esta mutación puede ser detectada en la comparativa como una amplia y generalizada sucesión de discrepancias o *mismatches* a partir de cierto par de bases comparados, sumado a una alta similitud o bajo número de diferencias en las bases correspondientes al sentido opuesto.

- Deleciones: se trata de la eliminación de material genético, lo cual puede provocar una situación muy similar a la de una inserción, causando importantes consecuencias. Dentro del alineamiento puede darse un posible reconocimiento de estas, por la aparición de un *gap* o hueco en la secuencia patológica o *query*, y que se encuentre relleno para esa misma posición en la secuencia *target* o control.

- Sustituciones: una de las mutaciones o cambios más comunes que puede sufrir la secuencia de nucleótidos. Consiste en el cambio de una base nitrogenada por otra, pudiendo producir cambios de alta o baja importancia, benignos o malignos, dada la degeneración del código genético presente en los tripletes. Suelen darse para una única

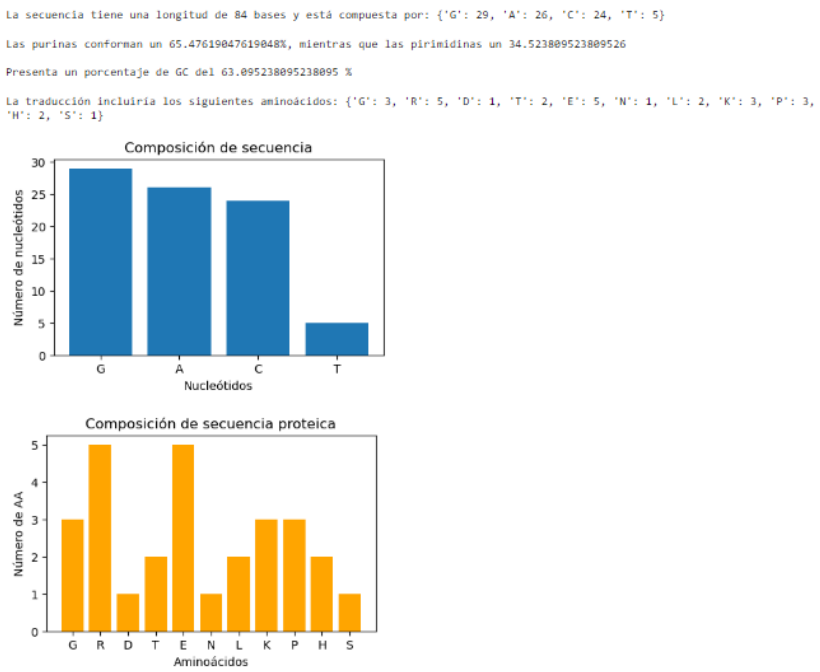


Figura B.25: Análisis de la secuencia consenso control del exón 8 del gen *TP53* mediante `Muscle`. Fuente: elaboración propia.

base (*SNV*) y representarse mediante *mismatches* en el alineamiento, es decir, una diferencia o no coincidencia entre la secuencia *query* o patológica y la *target* o control.

A partir de estas bases teóricas propuestas, se desarrollaron las funciones ya comentadas (`detectar_sustituciones`, `detectar_indels`, `hallar_matriz_puntuaciones`, `calcular_matrices_pareado`, `obtener_cadena_expresiones`, y por último, una función conjunto que se encarga de llamar a todas ellas, `analisis_pareado_matrices`. La detección de mutaciones se consigue a partir de los alineamientos generados y sus correspondientes matrices de frecuencia/sustituciones y puntuación. La primera de ellas mostrará las diferentes coincidencias contenidas en la confrontación; mientras que la segunda, con un filtro estadístico realizado por *odds-ratio* y ciertos cálculos más, reflejará de forma numérica la comparativa entre incidencia dada y esperada. Tras la llamada a la función `analisis_pareado_matrices`, se obtiene el alineamiento correspondiente, ambas matrices mencionadas, ciertas estadísticas relativas al alineamiento, dos listas con los posibles *indels* presentados y un *Dataframe* con las diferentes sustituciones que podrían haber tenido lugar.

```
----------------------------------ALINEAMIENTO GLOBAL BRCA1 EXON 5-------------------------------
target           0 CTTTTGAGTATTCTTTCTACAAAAGGAAGTAAAATTAAATTGTTCTTTCTTTCTTTATAA
                   0 ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
query            0 CTTTTGAGTATTCTTTCTACAAAAGGAAGTAAAATTAAATTGTTCTTTCTTTCTTTATAA

target          60 TTTATAGATTTTGCATGCTGAAACTTCTCAACCAGAAGAAAGGGCCTTCACAGTGTCCTT
                   60 ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
query           60 TTTATAGATTTTGCATGCTGAAACTTCTCAACCAGAAGAAAGGGCCTTCACAGTGTCCTT

target         120 TATGTAAGAATGATATAACCAAAAGGTATATAATTTGGTAATGATGCTAGGTTGGAAGCA
                  120 ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
query          120 TATGTAAGAATGATATAACCAAAAGGTATATAATTTGGTAATGATGCTAGGTTGGAAGCA

target         180 ACCACAGTAGGAAAAAGTAGAAATTATTTAATAACATAGCGTTCCTATAAAACCATTCAT
                  180 ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
query          180 ACCACAGTAGGAAAAAGTAGAAATTATTTAATAACATAGCGTTCCTATAAAACCATTCAT

target         240 CAGAAAAATTTATAAAAGAGTTTTTAGCACACAGTAAATTATTTCCAAAGTTATTTTCCT
                  240 ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
query          240 CAGAAAAATTTATAAAAGAGTTTTTAGCACACAGTAAATTATTTCCAAAGTTATTTTCCT

target         300 GAAAGTTTTATGGGACATCTGCCTTATACAGGTATTAGAAACTTACTGCCTTTCTCTAAT
                  300 ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
query          300 GAAAGTTTTATGGGACATCTGCCTTATACAGGTATTAGAAACTTACTGCCTTTCTCTAAT

target         360 GCAAXAAA 368
                  360 |||||||| 368
query          360 GCAAXAAA 368


El alineamiento global tiene 0 huecos, 368 identidades, 0 mismatches

Presenta una puntuación de 368.0
```

Figura B.26: Ejemplo de una alineamiento perfecto entre secuencias iguales. Fuente: elaboración propia.

```
target           0 -------------------------------------------GATTCCACACCCCCGCCC
                360 -------------------------------------------.|.....|...|.....
query          360 XAXGTGAXXAAATTTTAXXXCXTXAXXAXXAXATGXCTXXXTTAAAGAXXAXTTCTXAXX

target          18 GGCACCCGCGTCCGCGCCATGGCCATCTACAAGCAGTCACAGCACATGACGGAGGTTGTG
                420 ..........|.|....|.|.................|.|.||.....||....|.|..||.
query          420 XATXXXXXGXTXCTGATCTTXAXGTXAXXATGAATGCCCCAXXXATTGGTCAATGXXGTX

target          78 AGGCGCTGCCCCCACCATGAGCGCTGCTCAGATAGCGATGGTCTGGCCCCTCCTCAGCAT
                480 ....|..||...||...|||..|.........|.|.....|...|.....|.........
query          480 XAXXGXAGCAXXCAATTTGAAGGTACAGXTXAAAXTXXXXGXXXGTTTXXTXGXXCAXCX

target         138 CTTATCCGAG----------------------------------------------------
                540 .||.....|.----------------------------------------------------
query          540 XTTGXAAAATXTGTXXAGXTXTXXXXXXCTXXAXXAXAXAXXXXAAXAAAAAAGTGXTTX

target         148 ----------------------------------------------------------------
                600 ----------------------------------------------------------------
query          600 XXXXXXATTXAAXAXATGAAAATGAXXXXXXXTTTAXXXXXAXXGTXXTAGXGTAAXXXA

target         148 ---------------------------------------------------- 148
                660 ---------------------------------------------------- 710
query          660 CCTTCAXXXXAGXTXXXAXTXXXXXAAAAXACXXXTCTAAGTACXXXAGA 710


El alineamiento global tiene 562 huecos, 34 identidades, 114 mismatches

Presenta una puntuación de 34.0
```

Figura B.27: Ejemplo de una alineamiento global obtenido de la confrontación de secuencias consenso. Fuente: elaboración propia.

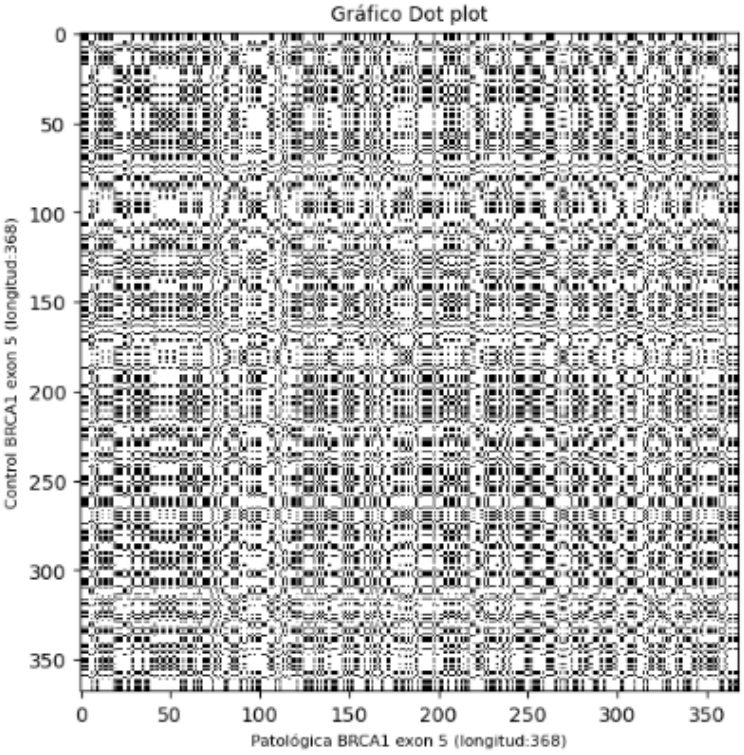


Figura B.28: Ejemplo de un gráfico *Dot plot* correspondiente a un alineamiento con una coincidencia de bases del 100 %. Fuente: elaboración propia.

Ninguno de los análisis pareados obtenidos parece presentan la posibilidad de contener mutaciones de tipo deleción en su secuencia patológica confrontada. Al contrario sucede con las inserciones, las cuales no pueden ser más numerosas, al presentar grandes diferencias de longitud entre secuencia control y patológica para la mayoría de los casos. Las sustituciones se tratan de algo más específico, variando aquella más frecuente dependiendo del exón estudiado. Un ejemplo sería el mostrado en la Figura B.33.

Adicionalmente, se realizó la creación de otras tres funciones destinadas a la comparativa de las posibles mutaciones halladas, respecto aquellas reconocidas en la base de datos de *gnomAD*. Para que estas actuaran, como un posible filtro y mecanismo de validación. Esta comparación automatizada, únicamente podrían aplicarse para aquellas mutaciones de los exones contenidas en *gnomAD*, cuyo sentido coincidiera con el presentado por las secuencias obtenidas, es decir 5' $\longrightarrow$ 3'. Teniendo que recurrir para el resto de los casos a la literatura.

```
---------------------------------ALINEAMIENTO GLOBAL BRCA1 EXON 5-----------------
target           0 CTACTGTTGCTGCATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTT
                   0 --..|.|...|.....||.|....|......||.|..|...||.||.|||...|.|...|
query            0 --CTTTTGAGTATTCTTTCTACAAAAGGAAGTAAAATTAAATTGTTCTTTCTTTCTTTAT

target          60 AAAAGGTTGATAATCACTTGCTXAGTGTGTTTCTCAAACAATTTAATTTCAGGAGCCTAC
                   60 ||....|.|||..|...|.....|...|.|....||.|..|.......|....||..|.|
query            58 AATTTATAGATTTTGCATGCTGAAACTTCTCAACCAGAAGAAAGGGCCTTCACAGTGTCC

target          120 AAGAAAGTACGAGATTTAGTCAACTTGTTGAAGAGCTATTGAAAATCATTTGTGCTTTTC
                  120 ...|....|..|...|....|.|...|.|..|.|..|....||......|.|........
query           118 TTTATGTAAGAATGATATAACCAAAAGGTATATAATTTGGTAATGATGCTAGGTTGGAAG

target          180 AGCTTGACACAGGTTTGGAGTGTAAGTGTTGAATATCCCAAGAATGACACTCAAGTGCTG
                  180 ..........|||........|.||.|.||.||||.|..|....|...|...||....|.
query           178 CAACCACAGTAGGAAAAAGTAGAAATTATTTAATAACATAGCGTTCCTATAAAACCATTC

target          240 TCCATGAAAACTCAGGAAGTTTGCACAATTACTTTCTATGACGTGGTGATAAGACCTTTT
                  240 ..||..||||.|.|..||.........|..||....||.....|....|.|.....|||.
query           238 ATCAGAAAAATTTATAAAAGAGTTTTTAGCACACAGTAAATTATTTCCAAAGTTATTTTC

target          300 AGTCTAGGTTAATTTTAGTTCTGTATCTGTAATCTATTTTTAAAAAATTACTCCCACTGG
                  300 .....||.||.||...|..||||..|.....|..||||...||...|.|.|.........
query           298 CTGAAAGTTTTATGGGACATCTGCCTTATACAGGTATTAGAAACTTACTGCCTTTCTCTA

target          360 TCTCACACCXCA 372
                  360 ...||.....-- 372
query           358 ATGCAAXAAA-- 368

El alineamiento global tiene 4 huecos, 121 identidades, 247 mismatches

Presenta una puntuación de 121.0
```

Figura B.29: Ejemplo de un alineamiento global del exón 5 de *BRCA1* mediante `Clustal`. Fuente: elaboración propia.

El exón 11 de *BRCA2* (Figura B.34) fue la única secuencia cuyas mutaciones se encontraban recogidas en la base de datos, en el mismo sentido del que se tenía en las descargadas; además de presentar unas secuencias consenso control y patológica. En su comparativa, se demostró que la mutación más frecuente en la mayoría de las ocasiones se mostraba como una variante maligna del exón, la cual provocaba una mutación sin sentido que daría lugar a la traducción de otro aa (aminoácido) diferente. También se reconocieron el resto de las sustituciones de naturaleza maligna, y se determinó la inexistencia de sustituciones benignas.

Un aspecto que también fue interesante de explorar mediante el alineamiento de secuencias, fue la comparativa de secuencias patológicas cuales quiera de un exón respecto de su consenso control. Un estudio enfocado en la detección de mutaciones puntuales y singulares, al no compararse secuencias consenso entre sí como en el caso anterior.

El número de comparaciones globales de secuencias individuales respecto a una consenso, es excesivamente alto y de un alto costo computacional, para el cual el sistema de cómputo del que se dispone no está prepara-

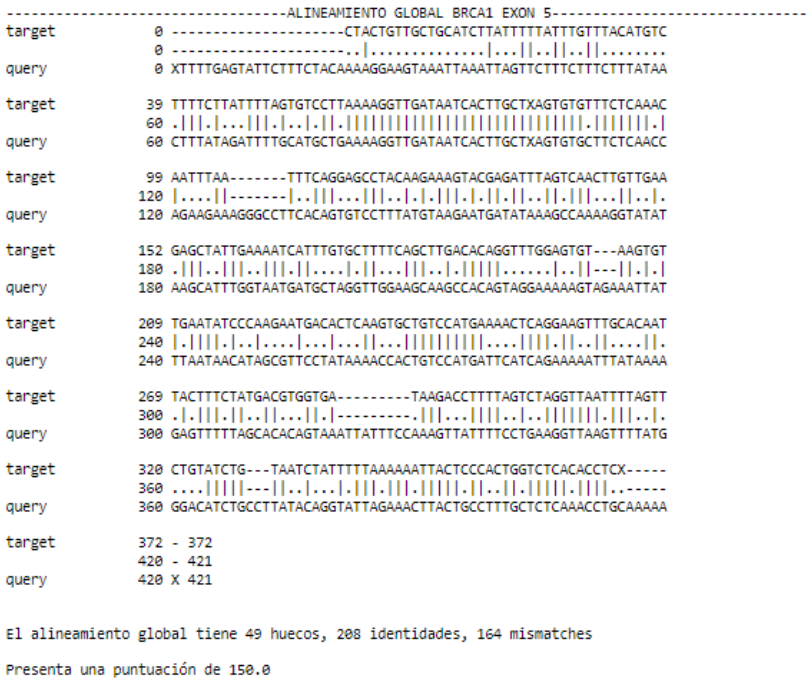


Figura B.30: Ejemplo de un alineamiento global del exón 5 de *BRCA1* mediante `Muscle`. Fuente: elaboración propia.

do. De forma que, para tal propósito, se desarrolló una función llamada `seleccion_dos_secuencias_aleatorias`. Esta función selecciona y devuelve dos secuencias aleatorias de un archivo pasado, las cuales posteriormente serán confrontadas con las funciones mencionadas.

En el alineamiento en el que se confrontaban secuencias patológicas aleatorias con la secuencia consenso control para dicho exón, tenía por objetivo la identificación de sustituciones singulares. Para ciertos exones, como son los correspondientes a *BRCA1*, exón 5 y exón 8 de *TP53*, en la comparación de sus diferentes secuencias con la secuencia consenso, se obtienen las mismas mutaciones habituales. Y de manera opuesta, para la primera secuencia del exón 11 de *BRCA2*, se obtiene una diferente mutación de mayor frecuencia (T>A), con respecto a la segunda y la consenso (A>T). Aunque la similitud de nucleótidos presente da a creer de la existencia de algún tipo de relación entre ambas. Esta singularidad mostrada por las secuencias de *BRCA2*, estaría relacionada por la alta variabilidad mencionada en reiteradas ocasiones entre sus secuencias.

```
---------------------------------ALINEAMIENTO LOCAL BRCA1 EXON 5-------------------------------
target          7 TGCTGCATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTTAAAAGGT
                0 |..||..|.||.|||.||.......|..|....||...|..||...|.|...|..|...|
query           1 TTTTGAGTATTCTTTCTACAAAAGGAAGTAAAATTAAATTGTTCTTTCTTTCTTTATAAT

target         67 TGATAATCACTTGCTXAGTGTGTTTCTCAAACA-ATTTAATTTCAGGAGCCTACAAGAAA
               60 |.|||.....|...|........|||||||.||-|...||...|.....|..........
query          61 TTATAGATTTTGCATGCTGAAACTTCTCAACCAGAAGAAAGGGCCTTCACAGTGTCCTTT

target        126 GTACGAGATTTAGTCAACTTGTTGAAGAGCTATTGAAAATCATTTGTGCTTTTCAGCTTG
              120 .|...|||.|.|...|||.....|.......|||....|....|..|...||..|.....
query         121 ATGTAAGAATGATATAACCAAAAGGTATATAATTTGGTAATGATGCTAGGTTGGAAGCAA

target        186 ACACAGGTTTGGAGTGTAAGTGTTGAATATCCCAAGAATGACACTCAAGTGCTGTCCATG
              180 .|||||.....|..|||....||...||.....|.|..|...||..|...|..|..|||.
query         181 CCACAGTAGGAAAAAGTAGAAATTATTTAATAACATAGCGTTCCTATAAAACCATTCATC

target        246 AAAACTCAGGAAGTTTGCACAATTACTTTCTATGACGTGGTGATAAGACCTTTTAGTCTA
              240 |.||.....|.....|.....|||.....|..|..|..|....|.|..|.||...||.
query         241 AGAAAAATTTATAAAAGAGTTTTTAGCACACAGTAAATTATTTCCAAAGTTATTTTCCTG

target        306 GGTTAATTTTAGTTCTGTATCTGTAATCTATTTTTAAAAAATTACTCCCACTGGTCTCAC
              300 ......||.|.|..|.....|..||..|...|.|||.|||.|||||.||..|.......
query         301 AAAGTTTTATGGGACATCTGCCTTATACAGGTATTAGAAACTTACTGCCTTTCTCTAATG

target        366 ACCXCA 372
              360 ...|.| 366
query         361 CAAXAA 367


El alineamiento local tiene 1 huecos, 131 identidades, 234 mismatches

Presenta una puntuación de 126.0
```

Figura B.31: Ejemplo de un alineamiento local del exón 5 de *BRCA1* mediante `Clustal`. Fuente: elaboración propia.

Genes	Secuencia aleatoria - Consenso		Entre consensos
	Primera secuencia - Consenso	Segunda secuencia - Consenso	
BRCA1	T>A	T>A	T>A
BRCA2	T>A	A>T	A>T
TP53/5	C>T	C>T	C>T
TP53/8	G>A	G>A	G>A

Tabla B.1: Mutaciones más frecuentes por cada gen, entre secuencias consenso patológica - control, y secuencias patológicas aleatorias - secuencia consenso control

La equiparación entre las mutaciones de mayor frecuencia, entre las secuencias consenso, y las secuencias patológicas aleatorias con la consenso control, se muestran en la Tabla B.1:

Como se puede apreciar, para la gran mayoría de los exones se mantiene una mutación habitual común, mientras que para el exón de valor de *BRCA2*, todas las mutaciones habituales en las confrontaciones son distintas. Esta comprobación afianza la afirmación de variabilidad entre sus secuencias ya comentada.

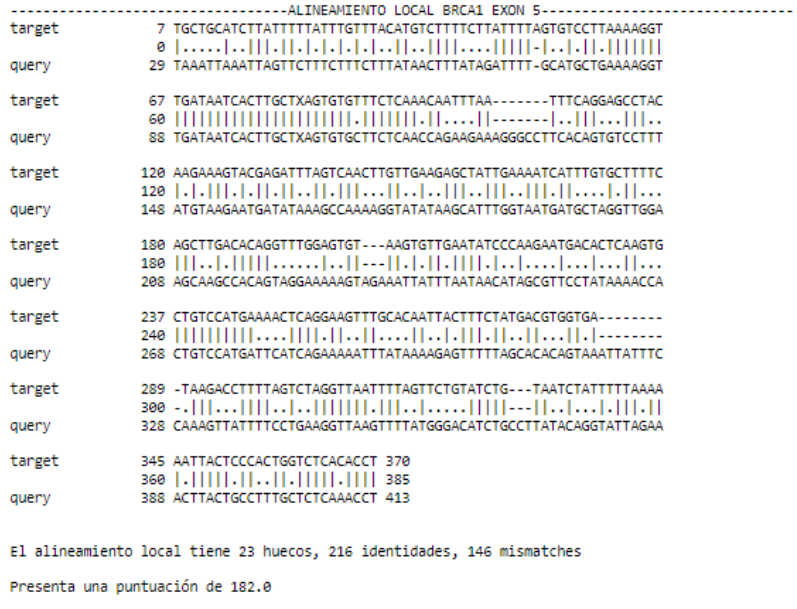


Figura B.32: Ejemplo de un alineamiento local del exón 5 de *BRCA1* mediante `Muscle`. Fuente: elaboración propia.

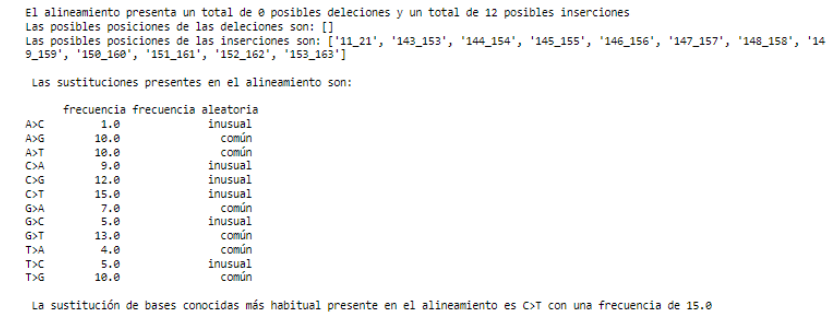


Figura B.33: Análisis de matrices de un alineamiento pareado correspondiente al exón 5 del gen *TP53*. Fuente: elaboración propia.

Al igual que se desarrolló con el alineamiento de secuencias pareado, se estudiarán las mutaciones por sustitución comunes en los alineamientos múltiples de patológicos y controles (este último en caso de que lo dispongan). Asimismo, se incluirán los análisis de los alineamientos múltiples de ambos algoritmos, con el propósito de encontrar diferencias entre sus resultados.

Los resultados tabulados de B.2 y B.3 exponen las mutaciones o cambios de nucleótidos más habituales para cada uno de los exones estudiando,

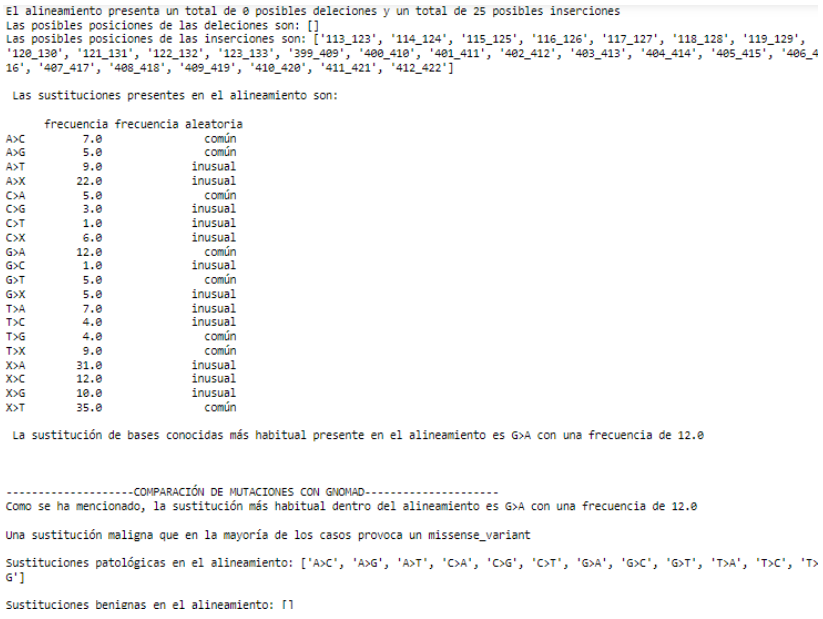


Figura B.34: Análisis de matrices de un alineamiento pareado correspondiente al exón 11 del gen *BRCA2*. Fuente: elaboración propia.

	Alineamiento Clustal	
Gen/Exón	Controles	Patológicos
BRCA1/5	R>G/G>R	A>T/T>A
BRCA1/19	No hay controles	A>G/G>A
BRCA2/2	No hay controles	A>G/G>A
BRCA2/11	A>T/T>A	A>T/T>A
PIK3CA/9	No hay controles	A>G/G>A
TP53/5	A>C/C>A	A>G/G>A
TP53/8	No hay cambios en secuencias controles	G>T/T>G

Tabla B.2: Sustituciones de mayor frecuencia en los alineamientos de múltiples secuencias de controles y patológicos mediante el algoritmo de `Clustal`

tanto en casos enfermos, como en sanos. Existe diferencia entre aquellos cambios presentados en controles y patológicos, salvo para el exón 11 del gen *BRCA2*. Resaltar a su vez, que algunas de las mutaciones se mantienen de la comparación anteriormente vista con secuencias controles, como es el caso de *BRCA1*, *BRCA2* y el exón 8 de *TP53* para `Muscle`. La conservación de estos cambios, afianzan algunas de las posibles sustituciones patológicas detectadas. Por otro lado, la amplia diversidad del resto, da a entender

	Alineamiento Muscle	
Gen/Exón	Controles	Patológicos
BRCA1/5	C>A/A>C	A>T/T>A
BRCA1/19	No hay controles	A>G/G>A
BRCA2/2	No hay controles	A>G/G>A
BRCA2/11	A>T/T>A	A>T/T>A
PIK3CA/9	No hay controles	A>G/G>A
TP53/5	A>C/C>A	A>G/G>A
TP53/8	No hay cambios en secuencias controles	G>A/A>G

Tabla B.3: Sustituciones de mayor frecuencia en los alineamientos de múltiples secuencias de controles y patológicos mediante el algoritmo de `Muscle`

la gran posibilidad de mutaciones que pueden llegar a tener lugar, incluso en aquellas controles o de tipo salvaje, y la aceptación de que los cambios tendrán lugar de una forma inevitable.

Tal y como se acaba de mencionar, también existen diferencias entre las mutaciones, dependiendo del algoritmo empleado. En concreto, estas se presentan en el exón 8 del gen *TP53* y 5 exón del gen *BRCA1*. Con el empleo del algoritmo de `Muscle`(B.36), como se ha podido comprobar a lo largo de la etapa analítica, se obtiene un mejor resultado que con el algoritmo de `Clustal`(B.35), debido a su propia estrategia de funcionamiento.

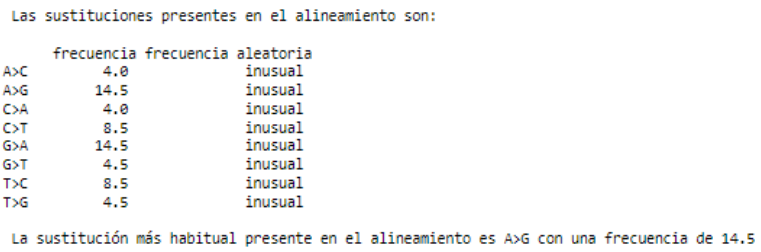
```
Las sustituciones presentes en el alineamiento son:

     frecuencia frecuencia aleatoria
A>C         4.0              inusual
A>G         4.0              inusual
C>A         4.0              inusual
C>T         8.5              inusual
G>A         4.0              inusual
G>T        11.5              inusual
T>C         8.5              inusual
T>G        11.5              inusual

La sustitución más habitual presente en el alineamiento es G>T con una frecuencia de 11.5
```

Figura B.35: Análisis de matrices de un alineamiento múltiple de `Clustal` correspondiente al exón 8 del gen *TP53*. Fuente: elaboración propia.

La comparación de alineamientos múltiples, de los genes con tablas de variantes procedentes de *gnomAD*, presentan unas comparaciones más variadas que en el caso de la confrontación de secuencias consenso, al reconocerse tanto variables benignas como malignas. Dando lugar por ello, a unas conclusiones que parecieran ser más realistas, como se puede observar en B.37.

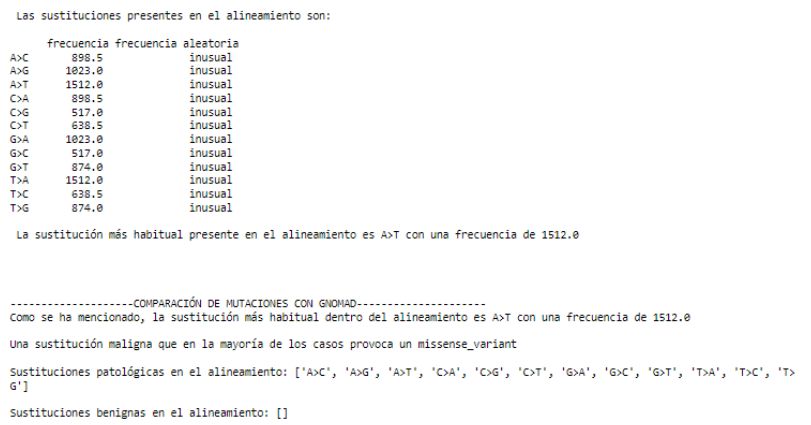


```
Las sustituciones presentes en el alineamiento son:

     frecuencia frecuencia aleatoria
A>C         4.0              inusual
A>G        14.5              inusual
C>A         4.0              inusual
C>T         8.5              inusual
G>A        14.5              inusual
G>T         4.5              inusual
T>C         8.5              inusual
T>G         4.5              inusual

La sustitución más habitual presente en el alineamiento es A>G con una frecuencia de 14.5
```

Figura B.36: Análisis de matrices de un alineamiento múltiple de `Muscle` correspondiente al exón 8 del gen *TP53*. Fuente: elaboración propia.



```
Las sustituciones presentes en el alineamiento son:

     frecuencia frecuencia aleatoria
A>C       898.5              inusual
A>G      1023.0              inusual
A>T      1512.0              inusual
C>A       898.5              inusual
C>G       517.0              inusual
C>T       638.5              inusual
G>A      1023.0              inusual
G>C       517.0              inusual
G>T       874.0              inusual
T>A      1512.0              inusual
T>C       638.5              inusual
T>G       874.0              inusual

La sustitución más habitual presente en el alineamiento es A>T con una frecuencia de 1512.0


--------------------COMPARACIÓN DE MUTACIONES CON GNOMAD----------------------
Como se ha mencionado, la sustitución más habitual dentro del alineamiento es A>T con una frecuencia de 1512.0

Una sustitución maligna que en la mayoría de los casos provoca un missense_variant

Sustituciones patológicas en el alineamiento: ['A>C', 'A>G', 'A>T', 'C>A', 'C>G', 'C>T', 'G>A', 'G>C', 'G>T', 'T>A', 'T>C', 'T>
G']

Sustituciones benignas en el alineamiento: []
```

Figura B.37: Análisis de matrices de un alineamiento múltiple de `Muscle` correspondiente al exón 11 del gen *BRCA2*. Fuente: elaboración propia.

La reutilización del código y funciones, `visualizacion_MSA` y `get_colors`, realizadas por Shtrauss en Kaggle [Shtrauss, 2023], permitieron una representación interactiva de los alineamientos múltiples realizados por `Clustal` y `Muscle` para los diferentes genes. Basada en la creación de un array 2D con cuadrícula, es rellenada con los diferentes caracteres que conforman las distintas secuencias, a los cuales se les a asociado un color predeterminado distintivo. Esta visualización colorida, permite de manera clara y fácil la determinación de las mutaciones y secuencia a la cual estas corresponden. Se trata de una herramienta de gran utilidad, pudiéndose observarse en ella, detalles, que hasta esas alturas de los análisis no se conocían con firmeza (Figura B.38 y Figura B.39). De igual forma que el visualizador interactivo de *MSA*, existen otras alternativas visuales como el conocido gráfico de *sequence logo*. *Sequence logo*, es una herramienta visual ampliamente utilizada en el mundo de la bioinformática, dada su

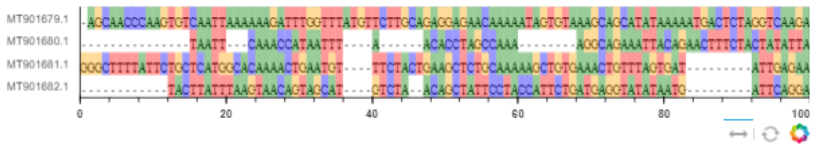


Figura B.38: Visualizador interactivo de *MSA* correspondiente a las secuencias control del exón 11 de *BRCA2*. Fuente: elaboración propia.

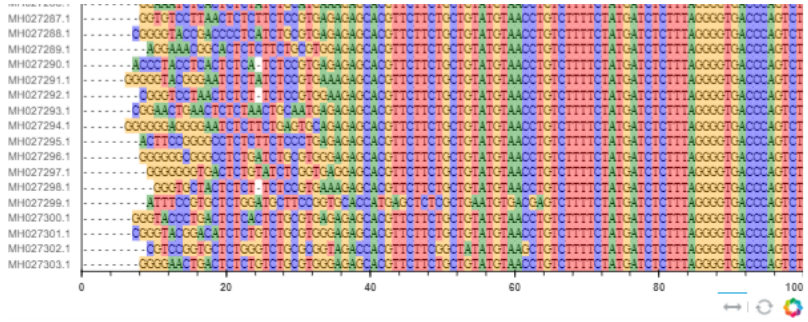


Figura B.39: Visualizador interactivo de *MSA* correspondiente a las secuencias patológicas del exón 19 de *BRCA1*. Fuente: elaboración propia.

gran utilidad. Similar al visualizador recientemente comentado, representa la información de las secuencias de forma vertical en lugar de horizontal, permitiendo la visualización de las diferentes bases y frecuencias en la que pueden encontrarse en una determinada posición. Claros ejemplos de ello son la Figura B.40 y Figura B.41.

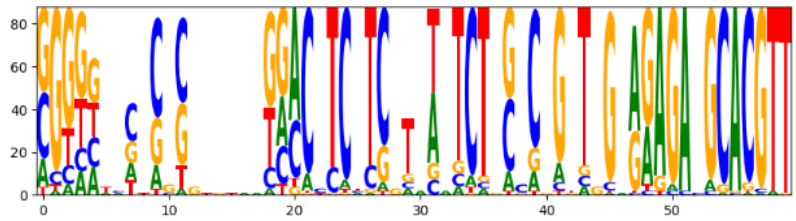


Figura B.40: Gráfico *sequence logo* representativo del *MSA* correspondiente a las secuencias patológicas del exón 19 de *BRCA1*. Fuente: elaboración propia.

El resto de resultados se encuentran en el *notebook* entregado llamado *Reconocimiento de biomarcadores con Biopython*.

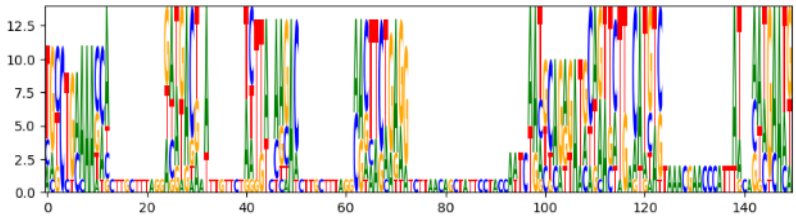


Figura B.41: Gráfico *sequence logo* representativo del *MSA* correspondiente a las secuencias patológicas del exón 11 de *BRCA2*. Fuente: elaboración propia. Fuente: elaboración propia.

Apéndice C

Manual programador

C.1. Estructura de directorios

Los directorios que contienen los datos y los *notebooks* necesarios para evaluar el trabajo presentado se encuentran en el repositorio de *GitHub* al que el tribunal de este Trabajo de Fin de Grado tiene acceso: *Biopython-reconocimiento-biomarcadores*. En dicho repositorio se pueden encontrar los siguientes directorios:

- **Algoritmos de MSA**: directorio en el cual se recogen los directorios de ambos algoritmos de alineamiento múltiple.
  - **ClustalOmega**: Uno de ellos con todos los archivos que conforman el *software* del algoritmo al completo.
  - **Muscle**: Otro únicamente contiene el archivo ejecutable.
- **clustal**: en él se encuentra el software completo de `Clustal`, uno de los algoritmos de alineamiento múltiple empleados en el *notebook* de `Python`, *Reconocimiento de biomarcadores con Biopython*.
- **muscle**: en él se encuentra el archivo ejecutable de `Muscle`, otro de los algoritmos de alineamiento múltiple empleados en el *notebook* de `Python`, *Reconocimiento de biomarcadores con Biopython*.
- **Datos**: directorio encargado de almacenar los datos brutos y procesados utilizados para el proyecto.
  - **Datos pacientes enfermos**: carpeta que incluye las secuencias patológicas en bruto repartidas en cuatro archivos de formato

`fasta`. Cada uno de ellos, correspondiente a cada uno de los genes: *BRCA1,BRAC2,PIK3CA* y *TP53*.

- **Datos pacientes sanos**: carpeta que incluye las secuencias control en bruto repartidas en cinco archivos de formato `fasta`. Cada uno de ellos, correspondiente a cada uno de los exones de los genes a estudio: exón 5 para *BRCA1*, exones 2 y 11 para *BRAC2*, y, por último, exones 5 y 8 para *TP53*.
- **Secuencias control filtradas**: carpeta que incluye las secuencias secuencias control ya procesadas, listas para el análisis conveniente. El conjunto de estas, está formado por cinco archivos de formato `fasta` de menor tamaño y mismo nombre que los originales.
- **Secuencias patológicas filtradas**: directorio que contiene las secuencias patológicas ya procesadas en formato `fasta`. Hasta ocho archivos se llegan a incluir en este directorio, uno por cada uno de los exones de mayor frecuencia a estudiar, dos por cada gen: 5 y 19 de *BRCA1*, 2 y 11 de *BRCA2*, 9 y 20 de *PIK3CA*; y 5 y 8 de *TP53*.
- **Tablas variantes**: este directorio comprende dos archivos de extensión `csv`, Tabla variantes *BRCA2* y *PIK3CA*. Cada uno de ellos engloba las variantes existentes documentadas en *gnomAD* para un gen en concreto.

- **Documentación**: directorio donde se encuentran los artículos considerados para la elaboración del proyecto. Hayan sido utilizados como fuente de inspiración o base teórica.
- **Imágenes**: comprende las imágenes utilizadas en el desarrollo de la memoria y los anexos.
- **Resultados**: directorio que integra los alineamientos y secuencias consenso obtenidas de la ejecución del *notebook Reconocimiento de biomarcadores con Biopython.ipynb*.

  - **Alineamiento secuencias control**: fichero que incluye los alineamientos mediante `Clustal` y `Muscle` de las secuencias controles filtradas.
    - **Resultados alineamiento clustal**: abarca cuatro ficheros de extensión `.afa`, que contienen los alineamientos `Clustal` correspondientes a los exones pertenecientes a las secuencias controles filtradas.

        - **Resultados alineamiento muscle**: abarca cuatro ficheros de extensión `.afa`, que contienen los alineamientos `Muscle` correspondientes a los exones pertenecientes a las secuencias controles filtradas.
    - **Alineamiento secuencias patológicas**: carpeta con los alineamientos de las secuencias patológicas procesadas de `Clustal` y `Muscle`.
        - **Resultados alineamiento clustal**: recoge ocho ficheros de extensión `.afa`, cada uno de ellos, por cada exón de los cuatro genes a analizar. Contienen los alineamientos `Clustal` de las secuencias patológicas seleccionadas.
        - **Resultados alineamiento muscle**: recoge ocho ficheros de extensión `.afa`, cada uno de ellos, por cada exón de los cuatro genes a analizar. Contienen los alineamientos `Muscle` de las secuencias patológicas seleccionadas.
    - **Secuencias consenso**: engloba todas las secuencias consenso de `Clustal` y `Muscle`, para patológicos y controles, conseguidas de la ejecución del *notebook*.
        - **Secuencias consenso controles**: directorio con las secuencias consenso provenientes de `Clustal` y `Muscle` relativas a las secuencias controles.
            - **Clustal**: contiene cuatro archivos `fasta` correspondientes a las secuencias consenso control de `Clustal`.
            - **Muscle**:contiene cuatro archivos `fasta` correspondientes a las secuencias consenso control de `Muscle`.
        - **Secuencias consenso patológicas**: carpeta dónde se incluyen las secuencias consenso provenientes de `Clustal` y `Muscle` relativas a las secuencias patológicas.
            - **Clustal**: fichero poseedor de las siete secuencias consenso procedentes de los alineamientos `Clustal` de cada uno de los exones patológicos.
            - **Muscle**:fichero poseedor de las siete secuencias consenso procedentes de los alineamientos `Muscle` de cada uno de los exones patológicos.

- **Reconocimiento de biomarcadores con Biopython.ipynb**: *notebook* de `Python` que recoge el desarrollo del proyecto en su totalidad. Desde los filtrados de secuencias, hasta la ejecución de los alineamientos y análisis de resultados.

C.2. Compilación, instalación y ejecución del proyecto

Al emplear un *notebook* de `Python`, el código implementado, se encuentra repartido en numerosos bloques de código. Cada uno de estos, de forma habitual, requerirá la ejecución de celdas anteriores al basarse en una programación secuencial, donde el orden presenta una gran importancia.

En primer lugar se deben de haber instalado correctamente los requerimientos comentados en el apartado de *Requisitos software y hardware para ejecutar el proyecto.* del anexo *C: Manual del usuario.* Una vez se haya comprobado la correcta instalación se podría pasar a la ejecución de las diferentes celdas de código.

Se iniciaría con la ejecución de la primera celda, la cual contiene las importaciones de paquetes complementarios y necesarios para una correcta ejecución. A continuación, se cargarían las dos primeras funciones del *notebook*, `obtener_tipo_extension` (Listing C.1) y `obtener_registros` (Listing C.2), destinadas a la lectura de archivos `fastq`, `genbank` y `fasta`.

Listing C.1: Código implementado para la función `obtener_tipo_extension` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def obtener_tipo_extension(extension):
2      """
3      Obtiene el tipo de formato de archivo asociado a una extensión
           ↪ de archivo dada mediante el empleo de un diccionario con
           ↪  pares clave- valor.
4      -------------
5      Parámetros:
6      - extension: La extensión de archivo para la cual se desea
           ↪ obtener el tipo de formato.
7                       las extensiones válidas son 'fasta', 'fna',
                            ↪ 'ffn', 'faa', 'frn',
8                       'fastq', 'fq', 'gb', 'gbk'.
9      ------------
10     Devuelve:
11     - diccionario_extensiones[extension]: El tipo de formato de
           ↪ archivo asociado a la extensión dada. Puede ser:
12             - 'fasta' para extensiones: 'fasta', 'fna', 'ffn', '
                   ↪ faa', 'frn'
13             - 'fastq' para extensiones: 'fastq', 'fq'
14             - 'genbank' para extensiones: 'gb', 'gbk'
```

```
15      """
16      diccionario_extensiones = {'fasta':'fasta','fna':'fasta','ffn':
            ↪ 'fasta','faa':'fasta','frn':'fasta',
17                  'fastq':'fastq','fq':'fastq',
18                  'gb':'genbank','gbk':'genbank'}

20      #duvuelve la extensión correspondiente
21      return diccionario_extensiones[extension]
```

Listing C.2: Código implementado para la función `obtener_registros` del *notebook Reconocimiento de biomarcadores con Biopython*

```
2   def obtener_registros(ruta_archivo):
3       """
4       Obtiene los registros de un archivo pasado con una extensión
            ↪ específica mediante el empleo de la función parse del
            ↪ paquete
5       SeqIO.
6       -------------
7       Parámetros:
8       - ruta_archivo: ruta del archivo a leer
9       ------------
10      Devuelve:
11      - lista_secuencias: lista que contiene todos los registros
            ↪ contenidos en el archivo.
12      """
13      #separamos el string pasado para facilitar el proceso
14      nombre = ruta_archivo.split('.')

16      #conseguimos la extensión del archivo
17      extension = nombre[-1]

19      #obtenemos una extensión común a la pasada en el argumento
20      tipo_ext = obtener_tipo_extension(extension)

22      #obtenemos todos los registros mediante el iterador parse
23      registros=SeqIO.parse(ruta_archivo,tipo_ext)

25      #creamos la lista donde irán almacenados nuestros registros
26      lista_registros=[]

28      #se recorre el iterador
```

```
29      for registro in registros:

31          #se agrega el registro a la lista
32          lista_registros.append(registro)

34      #se devuelve la lista con los registros del archivo
35      return lista_registros
```

Con esto, daría comienzo la etapa de filtrado, donde tanto para las secuencias controles, como las patológicas, se utilizarán en su mayoría las mismas funciones implementadas. A excepción de algunos bloques de código independientes. Para continuar con el proceso de filtrado se ejecutarían las funciones de `exones` (Listing C.3), `crear_dic` (Listing C.4) y `max_secs` (Listing C.5), destacando la primera, dado que se encarga de reestructurar los datos en una lista de listas de pares.

Listing C.3: Código implementado para la función `exones` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def exones(num_exones_gen):
2      """
3      Obtiene los exones/regiones codificantes de la lista de
           ↪ registros obtenida anteriormente en forma de una lista
           ↪ de pares de listas, que contienen el número y secuencia
           ↪ del exon correspondiente.
4      -------------
5      Parámetros:
6      - num_exones_gen: lista con los resgitros correspondientes a
           ↪ regiones codificantes de un gen de interés.
7      ------------
8      Devuelve:
9      - num_exones: lista de pares de lista, con las secuencias
           ↪ asociadas a un número de exón concreto.
10     """
11     #creación de la lista que contendrá los pares de listas
12     num_exones=[]

14     #bucle que recorre la lista de registros codificantes pasado
15     for exon in num_exones_gen:

17         #obtención de la posición del string que contiene el número
               ↪  para su posterior uso
```

```
        indice_num_exon = exon.description.split(" ").index("exon")
            ↪ +1

        #se añade la lista creada con el exon y su número
            ↪ correspondiente, además de la secuencia asociada
        num_exones.append(["exon "+exon.description.split(" ")[
            ↪ indice_num_exon],exon.seq])

    #se devuelve la lista de pares de listas
    return num_exones
```

Listing C.4: Código implementado para la función `crear_dic` del *notebook* *Reconocimiento de biomarcadores con Biopython*

```
def crear_dic(exones_gen):
    """
    Crea un diccionario mediante la lista de listas de pares de los
        ↪  exones pasado,en el que la clave es número del exón, y
        ↪ el valor, las secuencias asociadas.
    -------------
    Parámetros:
    - exones_gen: lista de listas de pares con los números de exó
        ↪ nes y las secuencias asociadas a estos.
    ------------
    Devuelve:
    - dict_exones: diccionario de pares exón y su lista de
        ↪ secuencias correspondientes.
    """
    #creación del diccionario a devolver
    dict_exones={}

    #se recorre la lista pasada
    for exon in exones_gen:

        #condicional que comprueba si el exon ya ha sido añadido
            ↪ como clave en el diccionario
        if exon[0] not in dict_exones.keys():

            #creación de la lista que servirá como valor en el par
            dict_exones[exon[0]]=[]

        #se añade la secuencia correspondiente al exón a la lista
            ↪ presente como valor
```

```
24          dict_exones[exon[0]].append(exon[1])

26      #se devuleve el diccionario con los exones y su lista de
            ↪ secuencias asociada
27      return dict_exones
```

Listing C.5: Código implementado para la función `max_secs` del *notebook* *Reconocimiento de biomarcadores con Biopython*

```
1   def max_secs(diccionario,corte):
2       """
3       Devuelve una lista de aquellos exones con un mayor valor. El nú
            ↪ mero de exones que devuelva estará determinado por el nú
            ↪ mero
4       entero que se pase como argumento.
5       -------------
6       Parámetros:
7       - diccionario: diccionario que contiene como claves los exones
            ↪ pertenecientes a un gen determinado, y como valores,
            ↪ listas
8       con todas las secuencias asociadas a dicho exón.

10      - corte: número entero que determina el número de exones a
            ↪ devolver.
11      ------------
12      Devuelve:
13      - max_exones: lista con los exones que presenten un mayor valor
            ↪ .
14      """
15      #creación de las listas correspondientes a los exones de mayor
            ↪ valor y número de secuencias
16      num_secs=[]
17      max_exones=[]

19      #bucle que recorre los valores del diccionario pasado, es decir
            ↪ , las listas
20      for exon in diccionario.values():

22          #se añaden la longitud de las listas a la lista
                ↪ anteriormente creada
23          num_secs.append(len(exon))
```

```
25      #se ordenan los números de las secuencias que contenían las
            ↪ listas
26      num_secs.sort()

28      #se obtienen los números más grandes de la lista, el número de
            ↪ ellos que devuelva viene dado por el integer pasado
29      #como argumento
30      num_secs_top= num_secs[::-1][:corte]

32      #se filtran aquellas listas que contienen un mayor número de
            ↪ secuencias
33      secuencias_filtradas=list(filter(lambda secuencias: len(
            ↪ secuencias) in num_secs_top ,diccionario.values()))

35      #se recorre la lista de secuencias que han sido filtradas
36      for secuencias in secuencias_filtradas:

38          #se agregan a la lista exones los exónes con mayor número
                ↪ de secuencias
39          max_exones.append(list(diccionario.keys())[list(diccionario.
                ↪ values()).index(secuencias)])

41      #se devuelven los exones con mayor número de secuencias
42      return max_exones
```

Para finalizar el proceso de filtrado, se deberían de ejecutar dos últimas funciones, `calcular_mediana` (Listing C.6) y `comprobar_longitudes` (Listing C.7), para filtrar todas aquellas secuencias.

Listing C.6: Código implementado para la función `calcular_mediana` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def calcular_mediana(dict_secuencias):
2      """
3      Devuelve una lista de medianas mediante la función median del
           ↪ paquete stats, con las dos medianas correspondientes a
           ↪ las
4      secuencias de los dos exones de mayor frecuencia.
5      -------------
6      Parámetros:
7      - dict_secuencias: diccionario que contiene como claves los dos
           ↪  exones de mayor frecuencia pertenecientes a un gen
8      determinado, y como valores, listas con todas las secuencias
           ↪ asociadas a dicho exón.
```

```
9       ------------
10      Devuelve:
11      - medianas: lista con las medianas de los dos exones de mayor
            ↪ frecuencia.
12      """
13      #se crea la lista de medianas
14      medianas=[]

16      #se recorre en un primer bucle las claves del diccionario de
            ↪ secuencias
17      for j in list(dict_secuencias):

19          #se crea una lista que contenga las longitudes de todas las
                ↪  secuencia de un exón
20          long=[]

22          #se recorre con un segundo bucle la lista de objetos Seq o
                ↪ secuencias del exon
23          for i in dict_secuencias[j]:

25              #se añade la longitud de la secuencia iterada a la
                    ↪ lista longitud
26              long.append(len(str(i)))

28          #se calcula la mediana con la función median
29          mediana=stats.median(long)

31          #se agrega la mediana calculada a la lista de medianas
32          medianas.append(mediana)

34      #se devuelve la lista con las dos medianas
35      return medianas
```

Listing C.7: Código implementado para la función `comprobar_longitudes` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1   def comprobar_longitudes(dict_secuencias,porcentaje_umbral):
2       """
3       Devuelve un diccionario reducido respecto el original, con
            ↪ aquellas secuencias que cumplan el umbral de longitud
            ↪ calculado
4       mediante la mediana correspondiente de la función calcular
            ↪ mediana, y un porcentaje umbral específico.
```

```
5       -------------
6       Parámetros:
7       - dict_secuencias: diccionario que contiene como claves los dos
            ↪  exones con mayores frecuenciapertenecientes a un gen
8       determinado, y como valores, listas con todas las secuencias
            ↪ asociadas a dicho exón.

10      - porcentaje_umbral: número entero que servirá de porcentaje
            ↪ para calcular un umbra adaptado a las longitudes de cada
            ↪  lista
11      de secuencias
12      ------------
13      Devuelve:
14      - max_exones: lista con los exones que presenten un mayor valor
            ↪ .
15      """
16      #se crea el diccionario que contendrá las secuencias filtradas
17      diccionario_filtrado={}

19      #se calculan las medianas mediante la función calcular mediana
20      medianas=calcular_mediana(dict_secuencias)

22      #se recorren las claves del diccionario pasado
23      for j in list(dict_secuencias):

25          #se crea la lista de las secuencias filtradas
26          secs_filtradas=[]

28          #se crea un segundo bucle para recorrer la lista
                ↪ correspondiente al valor de cada exón
29          for i in dict_secuencias[j]:

31              #se comprueba que la longitud de la secuencia iterada
                    ↪ se encuntre entre un valor máximo y mínimo de la
                    ↪ mediana
32              if medianas[list(dict_secuencias).index(j)] - medianas[
                    ↪ list(dict_secuencias).index(j)] * (
                    ↪ porcentaje_umbral/100) <= len(i) <= medianas[list
                    ↪ (dict_secuencias).index(j)] + medianas[list(
                    ↪ dict_secuencias).index(j)] * (porcentaje_umbral
                    ↪ /100):
```

```
34                #aquella secuencia que lo cumpla será agregada a la
                      ↪ lista de secuencias con logitudes filtradas
35                secs_filtradas.append(i)

37            #tras finalizar el bucle se asocia la lista de secuencias
                  ↪ filtradas como valor del exón analizado
38            diccionario_filtrado[j]=secs_filtradas

40        #se devuelve un nuevo diccionario con dos pares clave-valor con
              ↪  unos valores reducidos
41        return diccionario_filtrado
```

Completado el proceso de tratamiento, para guardar y trabajar con las secuencias de una manera más sencilla, se desarrolla la función `almacenar_secs` (Listing C.8).

Listing C.8: Código implementado para la función `almacenar_secs` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def almacenar_secs(diccionario_filtrado,secuencias_originales,gen,
       ↪ origen):
2      """
3      Almacena en el dispositivo las secuencias filtradas en un
           ↪ archivo fasta, mediante la función write del paquete
           ↪ SeqIO.
4      -------------
5      Parámetros:
6      - diccionario_filtrado: diccionario de un gen determinado
           ↪ filtrado con los exones de mayor frecuencia y las
           ↪ secuencias de
7      longitud similar.

9      - secuencias_originales: lista con los registros de las
           ↪ secuencias originales del gen determinado.
10     - gen: gen específico del diccionario y lista pasados, será
           ↪ aquel del que se almacene información.
11     - origen: naturaleza de los registros con las secuencias a
           ↪ almacenar, pueden ser patológicas o de control.
12     ------------
13     Devuelve:
14     En esta función no se devuelve ninguna salida.
15     """
```

```
17      #se crea a lista de claves pertenecientes al diccionario
            ↪ filtrado
18      lista_keys=list(diccionario_filtrado.keys())

20      #se crea una futura lista de ids con la que eliminaremos con
            ↪ ella posibles repeticiones de secuencias en nuestro
21      #archivo fasta
22      lista_ids=[]

24      #contador correpondiente a la llave a iterar
25      num_llave=0

27      #se crea un primer bucle que se realizará mientras el contador
            ↪ recientemente creado no supere la longitud de la lista
            ↪ de
28      #claves
29      while num_llave < len(lista_keys):

31          #se crea la lista secuencias donde se irán añadiendo
32          lista_sec=[]

34          #se crea un segundo bucle que recorre la lista de registros
                ↪  originales
35          for secuencia in secuencias_originales:

37              #se crea a su vez un tercer bucle que recorre las
                    ↪ secuencias filtradas para cada llave
38              for sec_filtrada in diccionario_filtrado[lista_keys[
                    ↪ num_llave]]:

40                  #se comprueba si la secuencia iterada es igual a la
                        ↪ secuencia filtrada iterada, además de si el
                        ↪ id
41                  #correspondiente se encuentra en la lista
                        ↪ anteriormente creada
42                  if str(secuencia.seq)==str(sec_filtrada) and
                        ↪ secuencia.id not in lista_ids:

44                      #se añade la secuencia a la lista de secuencias
45                      lista_sec.append(secuencia)

47                      #se añade el id a la lista de ids
48                      lista_ids.append(secuencia.id)
```

```
50            #se comprueba el origen de las secuencias pasadas
51            if origen == "sana":

53                #en caso de ser de origen sano se almacenan en el
                      ↪ directotio de Secuencias control filtradas
54                SeqIO.write(lista_sec,f"Datos\Secuencias control
                      ↪ filtradas\{gen.upper()} {lista_keys[num_llave
                      ↪ ]}.fasta","fasta")

56            #en caso de ser de origen patológico
57            if origen == "enferma":

59                #se almacena en Secuencias patológicas filtradas
60                SeqIO.write(lista_sec,f"Datos\Secuencias patológicas
                      ↪  filtradas\{gen.upper()} {lista_keys[
                      ↪ num_llave]}.fasta","fasta")

62        #se pasa a la siguiente clave o exón
63        num_llave=num_llave+1
```

Para poder realizar y ejecutar los alineamientos propuestos, deben de cargarse dos funciones fundamentales: `realizar_alineamiento` y `obtener_alineamientos`.

Listing C.9: Código implementado para la función `realiza_alineamiento` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def realizar_alineamiento(algoritmo,ruta_secuencias,
       ↪ ruta_alineamiento):
2      """
3      Realiza la confrontación de secuencias pasadas a partir de un
           ↪ algoritmo de alineamiento determinado
4      -------------
5      Parámetros:
6      - algoritmo: tipo de algoritmo a utilizar en el alineamiento.
7      - ruta_secuencias: ruta de las secuencias a alinear en el
           ↪ dispositivo.
8      - ruta_alineamiento: ruta donde se almacenará el alineamiento
           ↪ en nuestro dispositivo.
9      ------------
10     Devuelve:
11     En esta función no se devuelve objeto alguno.
12     """
13     #si el algoritmo a utilizar es clustalo
```

```
14      if algoritmo == "clustalo":

16          #ejecuta el algoritmo con lo parámetros necesarios para
                ↪ clustal
17          subprocess.run([algoritmo, "-i",ruta_secuencias, "-o",
                ↪ ruta_alineamiento,"-v","--force","--outfmt=fasta"],
                ↪ capture_output=True)

19      #si el algoritmo a utilizar es muscle
20      elif algoritmo== "muscle.exe":

22          #ejecuta el algoritmo con lo parámetros necesarios para
                ↪ muscle
23          subprocess.run([algoritmo,"-align", ruta_secuencias,"-
                ↪ output", ruta_alineamiento],capture_output=True)
```

Listing C.10: Código implementado para la función `obtener_alineamientos` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def obtener_alineamientos(ruta_alineamiento,formato):
2      """
3      Retorna el alineamiento de una ruta pasada, con un formato
           ↪ específico
4      -------------
5      Parámetros:
6      - ruta_alineamiento: ruta donde se encontrará el alineamiento
           ↪ en nuestro dispositivo.
7      - formato: formato del archivo que contiene los alineamientos.
8      ------------
9      Devuelve:
10     Alineamientos consultados.
11     """
12     #se realiza la lectura de los alineamientos mediante la función
           ↪  read del paquete AlignIO
13     alineamientos=AlignIO.read(ruta_alineamiento,formato)

15     #devuelve los alineamientos consultados
16     return alineamientos
```

Otro aspecto importante para poder comenzar los análisis, es la obtención de las secuencias consenso representativas mediante las funciones principales `obtener_consenso` y `guardar_consenso`. Aunque previamente a su llamada

debe declararse la función complementaria `crear_MultipleSeqAlignment`. De forma que primero se carga la función descrita en el Listing C.11.

Listing C.11: Código implementado para la función `crear_MultipleSeqAlignment` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def crear_MultipleSeqAlignment(archivo,formato):
    """
    Crea un objeto MultipleSeqAlignment a partir de un archivo de alineamiento múltiple de secuencias.
    -------------
    Parámetros:
    - archivo: archivo msa a convertir.
    - formato: formato del archivo que contiene los alineamientos.
    ------------
    Devuelve:
    Objeto MultipleSeqAlignment creado a partir del archivo pasado.
    """

    #lectura y almacenamiento en lista del archivo
    alineamientos=list(Align.read(archivo,formato))

    #se crea la lista vacía de alineamientos
    lista_alineamientos=[]

    #se genera un contador que actuará de id provisional para los diferentes alineamientos contenidos
    cont=1

    #se recorren los alineamientos
    for alineamiento in alineamientos:

        #para cada alineamiento se crea un objeto SeqRecord a partir de su secuencia e id establecido
        objeto_SeqRecord=SeqRecord(Seq(alineamiento),id="alineamiento "+ str(cont) )

        #se añade el objeto SeqRecord creado a la lista de alineamientps
        lista_alineamientos.append(objeto_SeqRecord)

        #se aumenta en una unidad el contador
        cont=cont+1
```

```
34      #se crea mediante la función MultipleSeqAlignment del paquete
            ↪ Bio.Align un objeto MultipleSeqAlignment
35      multiples_alineamientos=MultipleSeqAlignment(
            ↪ lista_alineamientos)

37      #se devuleve el objeto MultipleSeqAlignment recientemente
            ↪ creado
38      return multiples_alineamientos
```

Y con ella, las funciones descritas en el Listing C.12 y Listing C.13.

Listing C.12: Código implementado para la función `obtener_consenso` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1   def obtener_consenso(msa):
2       """
3       Se genera la secuencia consenso del msa dado, mediante las
            ↪ funciones SummaryInfo y dumb_consensus de un objeto
4       crear_MultipleSeqAlignment.
5       -------------
6       Parámetros:
7       - msa: objeto MultipleSeqAlignment con los alineamientos
            ↪ contenidos
8       ------------
9       Devuelve:
10      Secuencia consenso correspondiente al alineamiento multiple
            ↪ pasado.
11      """

13      #se obtiene el resumen del msa mediante la función SummaryInfo
14      resumen_alineamiento_msa=AlignInfo.SummaryInfo(msa)

16      #se obtiene la secuencia consenso mediante la función
            ↪ dumb_consensus
17      sec_consenso=resumen_alineamiento_msa.dumb_consensus()

19      #se devuelve la secuencia consenso del msa pasado
20      return sec_consenso
```

Listing C.13: Código implementado para la función `guardar_consenso` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1   def guardar_consenso(sec_consenso,info,ruta_archivo,formato):
```

```
    """
    Se almacena la secuencia consenso de interés en la ruta y formato pasados
    -------------
    Parámetros:
    - sec_consenso: secuencia consenso a almacenar
    - info: información adicional de la secuencia consenso a almacenar
    - ruta_archivo: ruta del dispositivo donde guardar la secuencia consenso
    - formato: formato del archivo a guardar
    ------------
    Devuelve:
    En esta función no se devuelve objeto alguno.
    """
    #fragmentación por el carácter "-" de la información pasada
    info_separado=info.split('-')

    #id de la secuencia consenso a almacenar
    num_id=info_separado[0]

    #descripción de la secuencia a almacenar
    descripcion=info_separado[1]

    #creación del objeto SeqRecord con la secuencia, el id y descripción pasados
    objeto_seqRecord=SeqRecord(sec_consenso,id=num_id,description=descripcion)

    #almacenamiento del objeto recién creado con la función write del paquete SeqIO
    SeqIO.write(objeto_seqRecord,ruta_archivo,formato)
```

De forma paralela, se realiza un tratamiento adicional de las tablas `csv`, que contienen las variantes existentes de ciertos genes. En este y futuros procesos, el cargado y declaración de las funciones: Listing C.14, Listing C.15, Listing C.16 y, finalmente, Listing C.17; son de gran importancia.

Listing C.14: Código implementado para la función `filtrar_ids` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def filtrar_ids(lista_ids,tabla):
    """
```

```
    Función que filtra una lista pasada de ids, en función de si se encuentran presentes en una tabla de variantes pasadas.
    -------------
    - lista_ids: lista que contiene los ids correspondientes a las variantes en la región de interés.
    - tabla: tabla de variantes pasada.
    ------------
    Devuelve:
    Un diccionario de las variantes correspondiantes a una región determinada.
    """

    #se crea un diccionario a partir de la tabla de variantes para facilitar su manipulación
    diccionario_gen=tabla.set_index("gnomAD ID").to_dict(orient="index")

    #se crea un diccionario donde se incluirán las variantes ya filtradas
    dict_gen_filtrado={}

    # se recorren los valores de la tabla
    for pos in range(len(diccionario_gen.values())):

        #en caso de que el id se encuentre presente en la lista pasada
        if list(diccionario_gen)[pos] in lista_ids:

            #se agrega al diccionario anteriormente creado con su id y valores asociados
            dict_gen_filtrado[list(diccionario_gen)[pos]]= diccionario_gen[list(diccionario_gen)[pos]].values()

    #se imprimen las longitudes de ambos diccionarios para visualizar por pantalla el filtrado conseguido
    print(len(diccionario_gen))
    print(len(dict_gen_filtrado))

    #se devuelve el diccionario filtrado con las variantes de la región de interés
    return dict_gen_filtrado
```

Listing C.15: Código implementado para la función `comprobar_naturaleza` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def comprobar_naturaleza(naturaleza):
    """
    Función que comprueba la naturaleza de una variante.
    -------------
    - naturaleza: naturaleza de la variante a comprobar.
    ------------
    Devuelve:
    String con la clasificación de la naturaleza de la variante a
        ↪ comprobar.
    """

    #en caso de ser de naturaleza desconocida
    if naturaleza=="Uncertain significance" or naturaleza=='not
        ↪ provided':

        #devuelvo el string correspondiente
        return 'incierto'

    #si es de naturaleza benigna
    elif naturaleza=='Likely benign' or naturaleza=='Benign':

        #devuelvo el string correspondiente
        return 'benigno'

    #si es de naturaleza maligna
    elif naturaleza=='Pathogenic' or naturaleza=='Pathogenic/Likely
        ↪  pathogenic' or naturaleza=='Likely pathogenic':

        #devuleve el string correspondiente
        return 'maligno'
```

Listing C.16: Código implementado para la función `comprobar_mutacion` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def comprobar_mutacion(mutacion):
    """
    Función que comprueba la mutación de la variante a examinar.
    -------------
    - mutacion: tipo de mutación de la variante a examinar.
    ------------
    Devuelve:
```

```
8       Un string con el tipo de mutación que presenta la variante y
            ↪ dependiendo del caso una lista de strings o una
9       string con la región donde esta tiene lugar.
10      """

12      #en caso de que la mutación sea una delecion
13      if 'del' in mutacion:

15          #se hacen los replazos necesarios
16          region_del=mutacion.replace("c.","").replace("del","")

18          #se devuelve el tipo de mutación y la region
19          return 'del',region_del

21      #si la mutación es una inserción
22      elif 'ins' in mutacion:

24          #se hacen los replazos necesarios
25          mutacion_separada=mutacion.replace("c.","").split("ins")

27          #se obtiene con la region y la inserción producida
28          region_ins = mutacion_separada[0]
29          n_ins = mutacion_separada[1]

31          #se devuelve el tipo de mutación y la region
32          return 'ins',[region_ins,n_ins]

34      #si la mutación es una duplicación
35      elif 'dup' in mutacion:

37          #se obtiene la región duplicada
38          region_dup= int(mutacion.replace("c.","").replace("dup","")
                ↪ )

40          #se devuelve el tipo de mutación y la region
41          return 'dup', region_dup

43      #si la mutacion es una sustitución
44      else:
45          #se hacen los resplazos correspondientes para obtener la
                ↪ sustitución y la región donde tiene lugar esta
46          sustitucion=mutacion.replace("c.","").replace("0","").
                ↪ replace("1","").replace("2","").replace("3","").
```

```
            ↪ replace("4","").replace("5","").replace("6","").
            ↪ replace("7","").replace("8","").replace("9","").
            ↪ replace("+","").replace("-","")
47          region_sust=mutacion.replace("A","").replace("C","").
            ↪ replace("T","").replace("G","").replace("c.","").
            ↪ replace(">","").replace("+","").replace("-","")

49          #se devuelve el tipo de mutación y la regio
50          return sustitucion,[region_sust]
```

Listing C.17: Código implementado para la función `crear_dic_variaciones` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1   def crear_dic_variaciones(dic_gen):
2       """
3       Función que crear un diccionario de variantes según un
            ↪ diccionario con los ids filtrados de la región de interé
            ↪ s.
4       -------------
5       - dic_gen: diccionario con las variantes de interés.
6       ------------
7       Devuelve:
8       Un diccionario final que reúne aquellas mutaciones que tienen
            ↪ lugar en la región de interés con información adicional.
9       asociada en forma de valor
10      """

12      #se crea el diccionario que será devuelto al final de la funció
            ↪ n
13      dic_final={}

15      #se recorren los ids del diccionario con las variantes
            ↪ filtradas
16      for reg in list(dic_gen):

18          #se guardan en una variable local los valores de la clave
                ↪ iterada
19          valores_clave=list(dic_gen[reg])

21          #se almacena el tipo de mutación que presenta
22          mutacion = valores_clave[2]
```

```
24          #se almacena la naturaleza que presenta
25          naturaleza = valores_clave[4]

27          #se almacena la consecuencia que tiene lugar dada la mutaci
                ↪ ón
28          consecuencia= valores_clave[3]

30          #se obtiene el tipo y región de mutación al llamar a la
                ↪ función comprobar _mutación
31          tipo_mutacion,region_mutacion=comprobar_mutacion(mutacion)

33          #se obtiene la naturaleza de la mutación al llamar a la
                ↪ función comprobar_naturaleza
34          tipo_naturaleza= comprobar_naturaleza(naturaleza)

36          #en caso de que el tipo de mutación ya se encuentre en las
                ↪ claves del diccionario a devolver
37          if tipo_mutacion in dic_final.keys():

39              #si la naturaleza es de tipo incierto
40              if tipo_naturaleza == 'incierto':

42                  #se aumenta en una unidad la frecuencia del tipo de
                        ↪ mutación
43                  dic_final[tipo_mutacion]['frecuencia total']=
                        ↪ dic_final[tipo_mutacion]['frecuencia total'
                        ↪ ]+1

45                  #se aumenta la frecuencia de naturaleza incierta en
                        ↪ una unidad
46                  dic_final[tipo_mutacion]['frecuencia incierta']=
                        ↪ dic_final[tipo_mutacion]['frecuencia incierta
                        ↪ ']+1

48              #en caso de que la naturaleza sea benigna
49              elif tipo_naturaleza == 'benigno':

51                  #se aumenta en una unidad la frecuencia del tipo de
                        ↪ mutación
52                  dic_final[tipo_mutacion]['frecuencia total']=
                        ↪ dic_final[tipo_mutacion]['frecuencia total'
                        ↪ ]+1
```

```
54              #se aumenta la frecuencia de naturaleza benigna en
                    ↪ una unidad
55              dic_final[tipo_mutacion]['frecuencia benigna']=
                    ↪ dic_final[tipo_mutacion]['frecuencia benigna'
                    ↪ ]+1

57          #para el caso de naturaleza maligna
58          else:

60              #se aumenta en una unidad la frecuencia del tipo de
                    ↪ mutación
61              dic_final[tipo_mutacion]['frecuencia total']=
                    ↪ dic_final[tipo_mutacion]['frecuencia total'
                    ↪ ]+1

63               #se aumenta la frecuencia de maligna incierta en
                     ↪ una unidad
64              dic_final[tipo_mutacion]['frecuencia maligna']=
                    ↪ dic_final[tipo_mutacion]['frecuencia maligna'
                    ↪ ]+1

66          #se agregan las regiones donde estas tienen lugar y el
                ↪ efecto de esta
67          dic_final[tipo_mutacion]['Region mutacion'].append(
                ↪ region_mutacion)
68          dic_final[tipo_mutacion]['Efecto en mutacion'].append(
                ↪ consecuencia)

70      #en caso de que no se encontrara la mutación en las claves
            ↪ del diccionario a devolver
71      else:

73          #en caso de naturaleza incierta
74          if tipo_naturaleza == 'incierto':

76              #se agrega el primer elemento
77              dic_final[tipo_mutacion]={'frecuencia total': 1,'
                    ↪ frecuencia maligna':0,'frecuencia benigna':0,
                    ↪ 'frecuencia incierta':1,'Region mutacion':[
                    ↪ region_mutacion],'Efecto en mutacion':[
                    ↪ consecuencia]}

79          #en caso de una naturaleza benigna
```

```
80            elif tipo_naturaleza == 'benigno':

82                #se agrega el primer elemento
83                dic_final[tipo_mutacion]={'frecuencia total': 1,'
                      ↪ frecuencia maligna':0,'frecuencia benigna':1,
                      ↪ 'frecuencia incierta':0,'Region mutacion':[
                      ↪ region_mutacion],'Efecto en mutacion':[
                      ↪ consecuencia]}

85            #maligno
86            else:

88                #se agrega el primer elemento
89                dic_final[tipo_mutacion]={'frecuencia total': 1,'
                      ↪ frecuencia maligna':1,'frecuencia benigna':0,
                      ↪ 'frecuencia incierta':0,'Region mutacion':[
                      ↪ region_mutacion],'Efecto en mutacion':[
                      ↪ consecuencia]}

91    #se devuelve el diccionario con las mutaciones correspondientes
92    return dic_final
```

Para la primera fase del análisis, se pretende realizar un análisis básico de las diferentes secuencias de forma individual. Para conseguirlo, se realizó una aplicación del la metodología divide y vencerás mediante la creación de diferentes funciones simples (Listing C.18, Listing C.19, Listing C.20 y Listing C.21), para posteriormente unirse en una función compleja, Listing C.22.

Listing C.18: Código implementado para la función `num_nucleotidos` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def num_nucleotidos(secuencia):
2      """
3      Función que calcula los nucleótidos y cantidad de cada uno de
           ↪ ellos presente en la secuencia.
4      -------------
5      - secuencia: secuencia de la cual se calculará su contenido.
6      ------------
7      Devuelve:
8      Un diccionario con los diferentes nucleótidos y su frecuencia
           ↪ que conforman la secuencia.
9      """
```

```
10      #se crea un diccionario donde se agregarán los nucleótidos
11      dic_nucleotidos={}

13      #se recorre la secuencia de nucleótido en nucleótido
14      for nucleotido in secuencia:

16          #se comprueba si el nucleótido iterado se encuentra en el
                ↪ diccionario a rellenar
17          if nucleotido in dic_nucleotidos.keys():

19              #en caso de que si se aumenta la frecuencia de este en
                    ↪ una unidad
20              dic_nucleotidos[nucleotido]=dic_nucleotidos[nucleotido
                    ↪ ]+1

22          #en caso contrario
23          else:

25              #se inicia el contador del nucleótido como clave
26              dic_nucleotidos[nucleotido]=1

28      #se devuelve el diccionario con la composición de la secuencia
            ↪ correspondiente
29      return dic_nucleotidos)
```

Listing C.19: Código implementado para la función `crear_barras_nucleotidos` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1   def crear_barras_nucleotidos(secuencia):
2       """
3       Función que representa la composición de nucleótidos de la
            ↪ secuencia pasada de una forma gráfica.
4       -------------
5       - secuencia: secuencia de la cual se mostrará su contenido.
6       ------------
7       Devuelve:
8       En esta función no se devuelve objeto alguno.
9       """
10      #se comienza estableciendo el tamaño de la figura
11      plt.figure(figsize=(5,3))
```

```
    #se crea el gráfico de barras a representar mediante las claves
        ↪  y valores del diccionario de nucleótidos obtenido
    #en la función num_nucleotidos
    plt.bar(num_nucleotidos(secuencia).keys(),num_nucleotidos(
        ↪ secuencia).values())

    #se agrega una etiqueta para el eje y
    plt.ylabel("Número de nucleótidos")

    #se agrega una etiqueta para el eje x
    plt.xlabel("Nucleótidos")

    #se agrega un título para el gráfico
    plt.title("Composición de secuencia")
```

Listing C.20: Código implementado para la función `num_aa` del *notebook* *Reconocimiento de biomarcadores con Biopython*

```
def num_aa(secuencia):
    """
    Función que calcula el contenido de aminoácidos en la secuencia
        ↪  pasada.
    -------------
    - secuencia: secuencia de la cual se mostrará su contenido de
        ↪ animo ácidos.
    ------------
    Devuelve:
    Un diccionario con todos los aminoácidos y sus frecuencias
        ↪ correspondientes que componen la secuencia.
    """

    #se crea un diccionario donde se agregarán los aa
    dic_aa={}

    #mediante la función translate del paquete Biopthon se traduce
        ↪ la secuencia a proteína
    secuencia_prot=secuencia.translate()

    #se recorren los aminoácidos que componen la proteína
    for aa in secuencia_prot:

        #se comprueba si el aa iterado se encuentra en el
            ↪ diccionario a rellenar
```

```
21          if aa in dic_aa.keys():

23              #en caso de que si se aumenta la frecuencia de este en
                    ↪ una unidad
24              dic_aa[aa]=dic_aa[aa]+1

26          #en caso contrario
27          else:

29              #se inicia el contador del aa como clave
30              dic_aa[aa]=1

32      #se devuelve el diccionario con la composición de la secuencia
            ↪ correspondiente
33      return dic_aa
```

Listing C.21: Código implementado para la función `crear_barras_aa` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def crear_barras_aa(secuencia):
2      """
3      Función que representa la composición de aa de la secuencia
           ↪ pasada de una forma gráfica.
4      -------------
5      - secuencia: secuencia de la cual se mostrará su contenido.
6      ------------
7      Devuelve:
8      En esta función no se devuelve objeto alguno.
9      """
10     #se comienza estableciendo el tamaño de la figura
11     plt.figure(figsize=(5,3))

13     #se crea el gráfico de barras a representar mediante las claves
           ↪  y valores del diccionario de aa obtenido
14     #en la función num_aa
15     plt.bar(num_aa(secuencia).keys(),num_aa(secuencia).values(),
           ↪ color='orange')

17     #se agrega una etiqueta para el eje y
18     plt.ylabel("Número de AA")

20     #se agrega una etiqueta para el eje x
21     plt.xlabel("Aminoácidos")
```

```
23      #se agrega un título para el gráfico
24      plt.title("Composición de secuencia proteica")
```

Listing C.22: Código implementado para la función `analisis_secuencia` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1   def analisis_secuencia(secuencia):
2       """
3       Función que realiza el análisis básico de una secuencia de
            ↪ nucleótidos.
4       -------------
5       - secuencia: secuencia a analizar.
6       ------------
7       Devuelve:
8       En esta función no se devuelve objeto alguno.
9       """

11      #se calcula el contenido en nucleótidos de la secuencia
12      contenido=num_nucleotidos(secuencia)

14      #A y G purinas, y T y C pirimidinas

16      #se calcula el porcentaje de purinas
17      porcentaje_purinas=((contenido["A"]+contenido["G"])/len(
            ↪ secuencia))*100

19      #se calcula el porcentaje de pirimidinas
20      porcentaje_pirimidinas=((contenido["C"]+contenido["T"])/len(
            ↪ secuencia))*100

22      #se imprime la longitud y contenido por pantalla
23      print(f"La secuencia tiene una longitud de {len(secuencia)}
            ↪ bases y está compuesta por: {contenido}")

25      #se crea el gráfico de barras correspondiente a los nucleótidos
            ↪  que conforman la secuencia
26      crear_barras_nucleotidos(secuencia)

28      #se imprime por pantalla el porcentaje de purinas y el
            ↪ porcentaje de pirimidinas
```

```
29      print(f"\nLas purinas conforman un {porcentaje_purinas}%,
            ↪ mientras que las pirimidinas un {porcentaje_pirimidinas}
            ↪ ")

31      #se calcula el porcentaje de bases desconocidas
32      bases_desconocidas=100-((contenido["A"]+contenido["G"]+
            ↪ contenido["C"]+contenido["T"])/len(secuencia))*100

34      #se muestra el contenido en gc de la secuencia
35      print(f"\nPresenta un porcentaje de GC del {gc_fraction(
            ↪ secuencia)*100} %")

37      #en caso de que la secuencia presentara bases no conocidas
38      if bases_desconocidas!=0:

40          #se imprimen por pantalla
41          print(f"Y el porcentaje de bases nucleotídicas ambiguas en
                ↪ la secuencia es del {bases_desconocidas}%")
42          print("\n No se puede realizar un análisis de
                ↪ correspondencia de aa dada la ambiguedad presente en
                ↪  los nucleótidos")

44      #en caso de que no haya se podría realizar la traducción de la
            ↪ secuencia en aa
45      else:

47          #se calcula el contenido de aa en la secuencia
48          aa=num_aa(secuencia)

50          #se muestran la frecuencia de estos por pantalla
51          print(f"\nLa traducción incluiría los siguientes aminoá
                ↪ cidos: {aa}")

53          #se crea el gráfico de barras correspondiente a los aa que
                ↪ conforman la secuencia
54          crear_barras_aa(secuencia)
```

El alineamiento pareado de secuencias se realizará a partir de una función principal como es `alineamiento` (Listing C.24), y una complementaria la cual asegura la elección del mejor de los calculados, `encontrar_mejor_alineamiento` (Listing C.23).

Listing C.23: Código implementado para la función `encontrar_mejor_alineamiento` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def encontrar_mejor_alineamiento(alineamientos):
    """
    Función que comprueba el mejora alineamiento y lo devuelve.
    -------------
    - alineamientos: alineamientos a comprobar.
    ------------
    Devuelve:
    El alineamiento con el mejor score.
    """
    #se establece una puntuación como mejor score que se irá
        ↪ actualizando
    mejor_puntuacion=alineamientos[0].score

    #se establece un alineamiento como el mejor que se irá
        ↪ actualizando
    mejor_alineamiento=alineamientos[0]

    #se recorren los alineamientos
    for alineamiento in alineamientos:

        #se compara que la puntuación del alineamiento iterado
            ↪ supere la mejor puntuación guardada
        if alineamiento.score > mejor_puntuacion:

            #se asigna un nuevo score como mejor puntuación
            mejor_puntuacion = alineamiento.score


            #se asigna un nuevo alineamiento como mejor
                ↪ alineamiento
            mejor_alineamiento = alineamiento

    #se devuelve el mejor alineamiento
    return mejor_alineamiento
```

Listing C.24: Código implementado para la función `alineamiento` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def alineamiento(secuencia_target,secuencia_query,modo,match_score
    ↪ =1,mismatch_score=0):
```

```
    """
    Función que realiza un alineamiento global o local de las dos secuencias pasadas, con unas puntuaciones de coincidencias y
    no coincidencias pudiendo ser definidas.
    -------------
    - secuencia_target: primera secuencia a alinear, generalmente la secuencia control.
    - secuencia_query: segunda secuencia a alinear, generalmente la  secuencia patológica.
    - modo: tipo de alineamiento a realizar.
    - match_score: posible argumento pasado que define la puntuació n de una coincidencia en el alineamiento.
    - mismatch_score: posible argumento pasado que define la puntuación de una no coincidencia en el alineamiento.
    ------------
    Devuelve:
    El alineamiento local o global realizado sobre las dos secuencias pasadas.
    """
    #en caso de realizar un alineamiento global
    if modo=="global":

        #se establece el alineador mediante la función PairwiseAligner del paquete Align
        alineador_global=Align.PairwiseAligner(mode="global",match_score=match_score,mismatch_score=mismatch_score,target_internal_open_gap_score=-10,target_internal_extend_gap_score=-1,query_internal_open_gap_score=-10,query_internal_extend_gap_score=-1)

        #con el alineador ya creado se llama a la función align para el alineamiento de las dos secuencias
        alineamientos_globales=alineador_global.align(secuencia_target,secuencia_query)

        #obtención del mejor alineamiento con la función anteriormente creada
        alineamiento_global=encontrar_mejor_alineamiento(alineamientos_globales)

        #se devuelve el alineamiento global realizado
```

```
28          return alineamiento_global

30      #en caso de realizar un alineamiento local
31      elif modo=="local":

33          #se establece el alineador mediante la función
                ↪ PairwiseAligner del paquete Align
34          alineador_local=Align.PairwiseAligner(mode="local",
                ↪ match_score=match_score, mismatch_score=
                ↪ mismatch_score,target_internal_open_gap_score=-5,
                ↪ target_internal_extend_gap_score=-0.5,
                ↪ query_internal_open_gap_score=-5,
                ↪ query_internal_extend_gap_score=-0.5)

36          #con el alineador ya creado se llama a la función align
                ↪ para el alineamiento de las dos secuencias
37          alineamientos_locales=alineador_local.align(
                ↪ secuencia_target,secuencia_query)

39          #obtención del mejor alineamiento con la función
                ↪ anteriormente creada
40          alineamiento_local=encontrar_mejor_alineamiento(
                ↪ alineamientos_locales)

42          #se devuelve el alineamiento local realizado
43          return alineamiento_local

45      #en caso de que no se introduzca un modo de alineamiento válido
46      else:
47          print("Introduzca un modo de alineamiento válido(global o
                ↪ local)")
```

Adicionalmente se crean tres funciones, para una presentación más visual de una confrontación entre las secuencias consenso (Listing C.25, Listing C.26 y Listing C.27).

Listing C.25: Código implementado para la función `crear_matriz` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def crear_matriz(nfilas,ncolumnas):
2      """
3      Crea una matriz de ceros para la generación de un gráfico dot
           ↪ plot válido.
4      -------------
```

```
    - nfilas: número de filas de la matriz a crear.
    - ncolumnas: número de columnas de la matriz a crear.
    ------------
    Devuelve:
    Una matriz bidimensional de ceros.
    """
    #se devuelve una matriz de ceros de nfilas x ncolumnas
    return np.zeros((nfilas,ncolumnas))
```

Listing C.26: Código implementado para la función `crear_dotplot` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def crear_dotplot(secuencia1,secuencia2):
    """
    Crea una matriz que representará las coincidencias y no
        ↪ coincidencias presenten en la comparación de dos
        ↪ secuencias pasadas.
    -------------
    - secuencia1: primera secuencia a confrontar.
    - secuencia2: segunda secuencia a confrontar.
    ------------
    Devuelve:
    Una matriz de coincidencias de las secuencias pasadas.
    """

    #se crea la matriz de ceros correspondiente
    matriz_dotplot=crear_matriz(len(secuencia1),len(secuencia2))

    #se recorre la primera dimensión de la matriz (filas)
    for i in range(len(secuencia1)):

        #se recorre la segunda dimensión de la matriz (columnas)
        for j in range(len(secuencia2)):

            #se comparan los nucleótidos de ambas secuencias para
                ↪ las posiciones iteradas
            if secuencia1[i]==secuencia2[j]:

                #en caso de coincidencia se actualiza el valor de la
                    ↪  matriz de ceros a 1
                matriz_dotplot[i,j]=1

    #se devuelve la matriz de coincidencias
```

```
28      return matriz_dotplot
```

Listing C.27: Código implementado para la función `mostrar_dotplot` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def mostrar_dotplot(secuencia1,secuencia2,id1="Secuencia 1
       ↪ desconocida",id2="Secuencia 2 desconocida"):
2      """
3      Función que muestra el gráfico dot plot de las dos secuencias
           ↪ pasadas.
4      -------------
5      - secuencia1: primera secuencia a confrontar
6      - secuencia2: segunda secuencia a confrontar.
7      ------------
8      Devuelve:
9      No se devuelve objeto alguno.
10     """

12     #se asigna el tamaño a la figura
13     plt.figure(figsize=(8,6))

15     #se asocia unas etiquetas correspondientes al grafico para el
           ↪ eje y
16     plt.ylabel(id1 + " (longitud:%i)" % len(secuencia1),size= 8)

18     #se asocia unas etiquetas correspondientes al grafico para el
           ↪ eje x
19     plt.xlabel(id2 + " (longitud:%i)" % len(secuencia2),size = 8)

21     #se asigna un título al gráfico creado
22     plt.title("Gráfico Dot plot",size=10)

24     #se crea la matriz de coincidencia que representará el el grá
           ↪ fico dot plot
25     datos=crear_dotplot(secuencia1,secuencia2)

27     #se imprime por pantalla el gráfico creado
28     plt.imshow(datos,interpolation='none',cmap="Greys")
29     plt.show()
```

Todas ellas, tal y como se ha realizado en otras ocasiones, conforman una función conjunta de análisis, encargada de realizar la llamada a cada una.C.28

Listing C.28: Código implementado para la función `comparacion_secuencias_consenso` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def comparacion_secuencias_consenso(sec_consenso_control,
    ↪ sec_consenso_patologica,nombre_gen,match_score=1,
    ↪ mismatch_score=0):
    """
    Función que raliza la comparación entre secuencias consenso
        ↪ patológicas y controles.
    -------------
    - sec_consenso_control: secuencia control a alinear.
    - sec_consenso_patologica: secuencia patológica a alinear.
    - nombre_gen: nombre del gen que se va a comparar.
    - match_score: posible argumento pasado que define la puntuació
        ↪ n de una coincidencia en el alineamiento.
    - mismatch_score: posible argumento pasado que define la
        ↪ puntuación de una no coincidencia en el alineamiento.
    ------------
    Devuelve:
    No se devuelve objeto alguno.
    """
    #se fragmenta el nombre del gen en el gen y el exon a comparar
    gen=nombre_gen.split(" ")[0].upper()
    exon=nombre_gen.split(" ")[1]

    #se realiza el alineamiento global entre la secuencia control y
        ↪  la secuencia patológica
    alineamiento_global=alineamiento(sec_consenso_control,
        ↪ sec_consenso_patologica,modo="global",match_score=
        ↪ match_score,mismatch_score=mismatch_score)


    #se realiza el alineamiento local entre la secuencia control y
        ↪ la secuencia patológica
    alineamiento_local=alineamiento(sec_consenso_control,
        ↪ sec_consenso_patologica,modo="local",match_score=
        ↪ match_score,mismatch_score=mismatch_score)

    #se imprime por pantalla toda la comparación realizada
    print(f"---------------------------------ALINEAMIENTO GLOBAL {
        ↪ gen} EXON {exon}-------------------------------")
    print(alineamiento_global)
    print()
```

```
29      print(f"El alineamiento global tiene {alineamiento_global.
            ↪ counts().gaps} huecos, {alineamiento_global.counts().
            ↪ identities} identidades, {alineamiento_global.counts().
            ↪ mismatches} mismatches")
30      print()
31      print(f"Presenta una puntuación de {alineamiento_global.score}"
            ↪ )
32      print("\n\n")
33      print(f"----------------------------------ALINEAMIENTO LOCAL {
            ↪ gen} EXON {exon}-------------------------------")
34      print(alineamiento_local)
35      print()
36      print(f"El alineamiento local tiene {alineamiento_local.counts
            ↪ ().gaps} huecos, {alineamiento_local.counts().identities
            ↪ } identidades, {alineamiento_local.counts().mismatches}
            ↪ mismatches")
37      print()
38      print(f"Presenta una puntuación de {alineamiento_local.score}")
39      print("\n\n\n")
40      #se muestra el gráfico dot plot que compara ambas secuencias
41      mostrar_dotplot(sec_consenso_control,sec_consenso_patologica,f"
            ↪ Control {gen} exon {exon}",f"Patológica {gen} exon {exon
            ↪ }")
42      print()
43      print(f"La diferencia de puntuaciones entre los distintos
            ↪ algoritmos es de {abs(alineamiento_global.score-
            ↪ alineamiento_local.score)}")
```

El análisis mediante matrices de sustituciones para los alineamientos son una parte troncal del proyecto, y por ello su correcta ejecución es crucial. En el desarrollo de dicho análisis intervienen numerosas y complejas funciones. Por tanto la ejecución la podemos dividir en diferentes conjuntos. Primeramente el cálculo de ambas matrices (Listing C.29 y Listing C.30).

Listing C.29: Código implementado para la función `hallar_matriz_puntuaciones` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1   def hallar_matriz_puntuaciones(alineamiento):
2       """
3       Calcula la matriz de puntuaciones correspondiente a un
            ↪ alineamiento.
4       -------------
```

```
    - alineamiento: alineamiento sobre se obtendrá la matriz de puntuaciones.
    ------------
    Devuelve:
    Una matriz bidimensional con las puntuaciones correspondientes a las diferentes combinaciones de los carácteres que
    conforman las diferentes secuencias.
    """
    #se hallan las frecuencias observadas mediante la matriz de sustituciones del alineamiento pasado
    frecuencias_observadas= alineamiento.substitutions/alineamiento.substitutions.sum()

    #se actualizan las frecuencias observadas y se hace simetrica a la matriz:
    #tendríamos con ello la probabilidad de cada sustitución
    frecuencias_observadas= (frecuencias_observadas+frecuencias_observadas.transpose())/2.0

    #se calcula la probabilidad de encontrar un nucleótido en dicha secuencia
    probabilidad_nucleotidos=frecuencias_observadas.sum(0)

    #se realiza el producto cartesiano de la matriz de probabilidades
    frecuencias_esperadas= probabilidad_nucleotidos[:,None]*probabilidad_nucleotidos[None,:]

    #se evaluan las frecuencias con los valores teóricos que deberían presentar por azar mediante el cáculo de odds ratio
    oddsratios = frecuencias_observadas / frecuencias_esperadas

    #se calcula la matriz de puntuación
    matriz_puntuacion = np.log2(oddsratios)

    #se devuleve la matriz de puntución recientemente calculada
    return matriz_puntuacion
```

Listing C.30: Código implementado para la función `calcular_matrices_pareado` del *notebook* *Reconocimiento de biomarcadores con Biopython*

```
def calcular_matrices_pareado(alineamiento_pareado):
```

```
    """
    La función calcula las matrices correspondientes al
        ↪ alineamiento pareado.
    -------------
    - alineamiento_pareado: alineamiento pareado sobre el que se
        ↪ obtendrán las matrices.
    ------------
    Devuelve:
    Dos matrices correspondientes a la matriz de frecuencia y
        ↪ sustitución calculadas.
    """

    #se calcula la matriz de puntuaciones con la funcion recién
        ↪ creada con la finalidad de expresar como han sido las
    #frecuencias encontradas en el alineamiento
    matriz_puntuacion = hallar_matriz_puntuaciones(
        ↪ alineamiento_pareado)

    #se imprime por pantalla ambas matrices
    print("Matriz de sustitución del alineamiento pareado: \n")
    print(alineamiento_pareado.substitutions)
    print()
    print()
    print("Matriz de puntuación del alineamiento pareado: \n")
    print(matriz_puntuacion)

    #se devuelven ambas matrices
    return alineamiento_pareado.substitutions,matriz_puntuacion
```

Tras ello, el segundo conjunto y el considerado más importante, centrado en la detección de los biomarcadores presentes en las secuencias patológicas. Un conjunto compuesto por gran variedad de funciones, incluyendo: Listing C.31, Listing C.32, Listing C.33, Listing C.34 y Listing C.35.

Listing C.31: Código implementado para la función `obtener_cadena_expresiones` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def obtener_cadena_expresiones(alineamiento):
    """
    Función que devuelve las expresiones correspondientes a los
        ↪ matches, mismatches y gaps de un alineamiento.
    -------------
```

```
5       - alineamiento: alineamiento a partir del cual se obtendrá las
            ↪ expresiones resultantes de un alineamiento dado.
6       ------------
7       Devuelve:
8       Un string con las expresiones resultantes de un alineamiento
            ↪ dado.
9       """

11      #se crea el string de expresiones que se devolverá a finalizar
            ↪ la función
12      cadena_expresiones=""

14      #secuencia control teórica
15      secuencia_target=alineamiento[0]

17      #secuencia patologica teórica
18      secuencia_query=alineamiento[1]

20      #se recorre la secuencia control
21      for n in range(len(secuencia_target)):

23          #si la secuencia control coincide en valor con la secuencia
                ↪  patológica en una misma posición
24          if secuencia_target[n]==secuencia_query[n]:

26              #se añade al string inicial una expresión equivalente a
                    ↪  un match
27              cadena_expresiones=cadena_expresiones+"|"

29          #si tiene lugar un gap
30          elif (secuencia_target[n]!= '-' and secuencia_query[n]=='-'
                ↪ ) or (secuencia_target[n]== '-' and secuencia_query[
                ↪ n]!='-') or (secuencia_target[n]== '-' and
                ↪ secuencia_query[n]!='-'):

32              #se añade al string inicial una expresión equivalente a
                    ↪  un gap
33              cadena_expresiones=cadena_expresiones+"-"

35          #en caso alternativo
36          else:
```

```
38            #se añade al string inicial una expresión equivalente a
                  ↪ un mismatch
39            cadena_expresiones=cadena_expresiones+"."

41    #se devuelve el string con las expresiones resultantes del
          ↪ alineamiento
42    return cadena_expresiones
```

Listing C.32: Código implementado para la función `detectar_sustituciones` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def detectar_sustituciones(alineamiento):
2      """
3      Detecta las sustituciones presentes en un alineamiento,
           ↪ devolviendo una diccionario con estas y las frecuencias
           ↪ que
4      presentan.
5      -------------
6      - alineamiento: alineamiento el cual se analizarán las posibles
           ↪  sustituciones que presente.
7      ------------
8      Devuelve:
9      Un diccionario en el que las claves serán las sustituciones y
           ↪ los valores asociados diccionarios con las
           ↪ correspondientes
10     frecuencias.
11     """

13     #se crea el diccionario de sustituciones a completar
14     dic_sustituciones={}

16     #a partir de la matriz de frecuencias/sustitucion generado con
           ↪ el alineamiento
17     #se obtienen las matrices de frecuencias/sustitucion y
           ↪ puntuaciones específicas
18     matriz_sustitucion=alineamiento.substitutions
19     matriz_puntuacion=hallar_matriz_puntuaciones(alineamiento)

21     #guardamos el alfabeto presente en la matriz de frecuencias/
           ↪ sustitucion
22     alfabeto=matriz_sustitucion.alphabet
```

```
24      #se recorre la matriz de frecuencias/sustitucion en una primera
            ↪  dimension (filas)
25      for i in range(0,len(matriz_sustitucion)):

27          #se recorre la matriz de frecuencias/sustitucion en una
                ↪ segunda dimension (columnas)
28          for j in range(0,len(matriz_sustitucion)):

30              #si el número de fila no coincide con el número de
                    ↪ columna
31              if i!=j:

33                  #si la frecuencia en positiva y no es nula
34                  if matriz_puntuacion[i][j]>0 and matriz_sustitucion[
                        ↪ i][j]!=0:

36                      #se almacena la sustitución como clave
37                      clave=alfabeto[i]+">"+alfabeto[j]

39                      #se asocia como valor un diccionario con la
                            ↪ frecuencia de dicha coincidencia y una
                            ↪ frecuencia aleatoria
40                      #común o mayor de lo esperada en el alineamiento
41                      dic_sustituciones[clave]= {"frecuencia":
                            ↪ matriz_sustitucion[i][j],"frecuencia
                            ↪ aleatoria":"común"}

43                  #si la frecuencia en negativa pero no nula
44                  elif matriz_puntuacion[i][j]<0 and
                        ↪ matriz_sustitucion[i][j]!=0:

46                      #se almacena la sustitución como clave
47                      clave=alfabeto[i]+">"+alfabeto[j]

49                      #se asocia como valor un diccionario con la
                            ↪ frecuencia de dicha coincidencia y una
                            ↪ frecuencia aleatoria
50                      #inusual o menor de lo esperada en el
                            ↪ alineamiento
51                      dic_sustituciones[clave]= {"frecuencia":
                            ↪ matriz_sustitucion[i][j],"frecuencia
                            ↪ aleatoria":"inusual"}
```

```
53                #en caso alternativo
54                else:
55                    #si el valor en la amtriz de frecuencias/
                          ↪ sustitución no es nulo
56                    if matriz_sustitucion[i][j]!=0:

58                        #se almacena la sustitución como clave
59                        clave=alfabeto[i]+">"+alfabeto[j]

61                        #se asocia como valor un diccionario con la
                              ↪ frecuencia de dicha coincidencia y una
                              ↪  frecuencia aleatoria
62                        #neutra o esperada en el alineamiento
63                        dic_sustituciones[clave]= {"frecuencia":
                              ↪ matriz_sustitucion[i][j],"frecuencia
                              ↪ aleatoria":"neutra"}

65    #se devuelve un diccionario con las diferentes posibles
          ↪ sustituciones encontradas
66    return dic_sustituciones
```

Listing C.33: Código implementado para la función `detectar_indels` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def detectar_indels(alineamiento):
2      """
3      Detecta las posibles inserciones y deleciones presentes en un
           ↪ alineamiento.
4      --------------
5      - alineamiento: alineamiento el cual se analizarán los posibles
           ↪  indels que presente.
6      ------------
7      Devuelve:
8      Un diccionario en el que las claves serán las mutaciones de
           ↪ inserción y deleción y los valores asociados
           ↪ diccionarios
9      con las correspondientes frecuencias posiciones y agregaciones
           ↪ y eliminaciones que presentan.
10     """
11     #a partir del propio alineamiento
12     #se crea el diccionario de inserciones y deleciones
```

```
13      dic_indels={'del':{'frecuencia':0,'deleciones':[],'posiciones
            ↪ en secuencia':[]},'ins':{'frecuencia':0,'inserciones':[],
            ↪ 'posiciones en secuencia':[]}}

15      #bandera para limitar deteccion de deleciones
16      bandera_del=0

18      #secuencia control
19      secuencia_target=alineamiento[0]

21      #secuencia patologica
22      secuencia_query=alineamiento[1]

24      #detectar del
25      #se recorre la secuencia control
26      for n in range(len(secuencia_target)):

28          #si la secuencia patológica presenta un gap
29          if secuencia_query[n]!='-':

31              #se añade una unidad a la bandera creada
32              bandera_del=bandera_del+1

34          #si la bandera es dintinto de cero
35          if bandera_del !=0:

37              #si la secuencia control no presenta un hueco y la
                    ↪ secuencia patológica si, podría ser un indicio de
                    ↪  un gap
38              if secuencia_target[n]!= '-' and secuencia_query[n]=='-
                    ↪ ':

40                  #se agrega una nueva delecion al diccionario de
                        ↪ valores de la clave del
41                  #se aumenta su frecuencia en uno
42                  dic_indels['del']['frecuencia']= dic_indels['del']['
                        ↪ frecuencia']+1

44                  #se añade la deleción del nucleótido a la lista de
                        ↪ deleciones
45                  dic_indels['del']['deleciones'].append('del'+ str(
                        ↪ secuencia_target[n]))
```

```
47                #se añaden a si vez las posiciones de las deleciones
                      ↪  que han tenido lugar
48                dic_indels['del']['posiciones en secuencia'].append(
                      ↪ n+1)


51    #detectar ins (or (secuencia_target[n]!= '-' and
          ↪ secuencia_query=='-'))

53    #como en el caso anterior se crea una bandera la limitar la
          ↪ deteccion de inserciones
54    bandera_ins=0

56    #para este caso se obtiene la cadena de expresiones del
          ↪ alineamiento a analizar
57    cadena_expresiones_alineamiento=obtener_cadena_expresiones(
          ↪ alineamiento)

59    #se recorre desde ciertas posiciones para evitar posibles
          ↪ errores de indexación
60    for n in range(10,len(secuencia_target)-10):

62        #se cuentan los mismatches presentes en las 10 posiciones
              ↪ anteriores
63        mis_10_anteriores=cadena_expresiones_alineamiento[n-10:n].
              ↪ count('.')

65        #se cuentan los mismatches presentes en las 10 posiciones
              ↪ posteriores
66        mis_10_posteriores=cadena_expresiones_alineamiento[n:n+10].
              ↪ count('.')

68        #si la secuencia patológica no presenta hueco para la
              ↪ posición iterada
69        if secuencia_query[n]!='-':

71            #se aumenta en una unidad la bandera limitante de
                  ↪ inserciones
72            bandera_ins=bandera_ins+1

74        #si la bandera es distinta de cero
75        if bandera_ins !=0:
```

```
77              #si el número de mismatches de las 10 posiciones
                    ↪ anteriores supera notablemente el número de ellas
                    ↪  en las 10
78              #posteriores, se detecta una posible situación de
                    ↪ mismatch.
79              if mis_10_anteriores>mis_10_posteriores*3:

81                  #se agrega una nueva inserción al diccionario de
                        ↪ valores de la clave ins
82                  #se aumenta su frecuencia en uno
83                  dic_indels['ins']['frecuencia']=dic_indels['ins']['
                        ↪ frecuencia']+1

85                  #se añade la inserción del nucleótido a la lista de
                        ↪ inserciones
86                  dic_indels['ins']['inserciones'].append('ins'+ str(
                        ↪ secuencia_target[n]))

88                  #se añaden a si vez las posiciones de las
                        ↪ inserciones que han tenido lugar
89                  dic_indels['ins']['posiciones en secuencia'].append(
                        ↪ str((n+1)-10)+"_"+str(n+1))

91              #si el número de mismatches de las 10 posiciones
                    ↪ anteriores es notablemente inferior al número de
                    ↪ ellas en las 10
92              #posteriores, se detecta una posible situación de
                    ↪ mismatch.
93              if mis_10_anteriores*3<mis_10_posteriores:

95                  #se agrega una nueva inserción al diccionario de
                        ↪ valores de la clave ins
96                  #se aumenta su frecuencia en uno
97                  dic_indels['ins']['frecuencia']=dic_indels['ins']['
                        ↪ frecuencia']+1

99                  #se añade la inserción del nucleótido a la lista de
                        ↪ inserciones
100                 dic_indels['ins']['inserciones'].append('ins'+ str(
                        ↪ secuencia_target[n]))

102                  #se añaden a si vez las posiciones de las
                         ↪ inserciones que han tenido lugar
```

```
103                 dic_indels['ins']['posiciones en secuencia'].append(
                        ↪ str(n+1)+"_"+str((n+1)+10))

105     #se imprime por pantalla la informacion general
106     print(f"El alineamiento presenta un total de {dic_indels['del
            ↪ ']['frecuencia']} posibles deleciones y un total de {
            ↪ dic_indels['ins']['frecuencia']} posibles inserciones")
107     print(f"Las posibles posiciones de las deleciones son: {
            ↪ dic_indels['del']['posiciones en secuencia']}")
108     print(f"Las posibles posiciones de las inserciones son: {
            ↪ dic_indels['ins']['posiciones en secuencia']}")

110     #se devuelve el diccionario con los indels detectados
111     return dic_indels
```

Listing C.34: Código implementado para la función `crear_dic_sust_frec` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def crear_dic_sust_frec(dic_sustituciones):
2      """
3      Crea un diccionario de sustituciones:frecuencia a partir de
           ↪ otro más complejo.
4      -------------
5      - dic_sustituciones: diccionario con las posibles sustituciones
           ↪  detectadas.
6      ------------
7      Devuelve:
8      Un diccionario en el que las claves serán las mutaciones de
           ↪ sustitución y los valores asociados a cada una de ellas.
9      """
10     #se crea el diccionario a completar con las sustituciones y
           ↪ frecuencias
11     dic_sust_frec={}

13     #se recorren las claves del diccionario de sustituciones
14     for clave in list(dic_sustituciones):

16         #se añade la sustitución como clave y la frecuencia
               ↪ correspondiente como valor al diccionario a devolver
17         dic_sust_frec[clave]= dic_sustituciones[clave]['frecuencia'
               ↪ ]

19     #se devuelve el diccionario
```

```
20      return dic_sust_frec
```

Listing C.35: Código implementado para la función `hallar_mayor_frec` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def hallar_mayor_frec(dic_sustituciones):
2      """
3      Función que a partir de un diccionario de sustituciones dado,
           ↪ detecta aquella de mayor frecuencia.
4      -------------
5      - dic_sustituciones: diccionario con las posibles sustituciones
           ↪  detectadas.
6      ------------
7      Devuelve:
8      Una lista con una clave correspondiente con la sustitución más
           ↪ frecuente y la frecuencia correspondiente a esta.
9      """
10     #NO TIENE EN CUENTA LOS NUCLEOTIDOS AMBIGUOS

12     #se crea un contador
13     contador=0

15     #se crea un diccionario más sencillo y unidimensional con la
           ↪ función crear_dic_sust_frec
16     dic_sust_frec=crear_dic_sust_frec(dic_sustituciones)

18     #asociamos las frecuencias a una lista
19     lista_valores=list(dic_sust_frec.values())

21     #ordenamos la lista de frecuencias de mayor a menor
22     lista_valores.sort(reverse=True)

24     #se recorre la lista de frecuencias
25     for valor in lista_valores:

27         #se recorren las claves del diccionario simplificado
28         for clave in list(dic_sust_frec):

30             #si la frecuencia de la clave iterada coincide con el
                   ↪ valor asignado como mayor y la llave no es
                   ↪ ambigua
31             if dic_sust_frec[clave]==valor and 'X' not in clave:
```

```
33                #se devuelve la lista con la clave conocida y la
                      ↪ mayor frecuencia
34                return [clave,dic_sust_frec[clave]]
```

Asimismo, se implementó una función llamada `analizar_sustituciones` (Listing C.36) para aquellos genes que presentaran su tabla de variantes procedente de *gnomAD* disponible. Al ejecutarla en conjunto con la función análisis (Listing C.37) conformada por el resto de las ya mencionadas, generaría una comparativa que serviría como discusión para dicho caso.

Listing C.36: Código implementado para la función `analizar_sustituciones` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def analizar_sustituciones(dic_variantes,dic_sustituciones):
2      """
3      Función que analiza e indica las sustituciones que mayor
           ↪ probabilidad tienen de darse en un contexto más realista
           ↪  según el
4      diccionario de la tabla de variantes registradas.
5      -------------
6      - dic_variantes: diccionario con las variantes de la región de
           ↪ interés.
7      - dic_sustituciones: diccionario con las posibles sustituciones
           ↪  detectadas.
8      ------------
9      Devuelve:
10     No se devuelve objeto alguno.
11     """

13     #se crean las listas de sustituciones patológicas y benignas
           ↪ inicialmente vacías
14     sust_patologicas=[]
15     sust_benignas=[]

17     #se recorren las claves del diccionario de posibles
           ↪ sustituciones
18     for sustitucion in list(dic_sustituciones):

20         #en caso de no ser la clave iterada una sustitución ambigua
21         if 'X' not in sustitucion:
```

```
23            #si la frecuencia de la sustitución es mayor en casos
                  ↪ maligno que en benignos
24            if dic_variantes[sustitucion]['frecuencia maligna']>
                  ↪ dic_variantes[sustitucion]['frecuencia benigna']:

26                #la sustitución se agrega a la lista de
                      ↪ sustituciones patológicas
27                sust_patologicas.append(sustitucion)

29            #en caso contrario
30            else:

32                #se añade la sustitución a la lista de sustituciones
                      ↪  benignas
33                sust_benignas.append(sustitucion)

35    #se obtiene la mutación más frecuente
36    mutacion_frecuente=hallar_mayor_frec(dic_sustituciones)[0]

38    #se obtiene la frecuencia de la mutación más frecuente
39    frecuencia_mutacion_frecuente=hallar_mayor_frec(
          ↪ dic_sustituciones)[1]

41    #se imprime por pantalla
42    print('--------------------COMPARACION DE MUTACIONES CON GNOMAD
          ↪ ----------------------')

44    print(f'Como se ha mencionado, la sustitución más habitual
          ↪ dentro del alineamiento es {mutacion_frecuente} con una
          ↪ frecuencia de {frecuencia_mutacion_frecuente}\n')

46    #si la mutación mas frecuente se encuentra en la lista de
          ↪ mutaciones patológicas
47    if mutacion_frecuente in sust_patologicas:
48        print(f"Una sustitución maligna que en la mayoría de los
              ↪ casos provoca un {Counter(dic_variantes[
              ↪ mutacion_frecuente]['Efecto en mutacion']).
              ↪ most_common(1)[0][0]}\n")

50    #en caso contrario
51    else:
52        print(f"Una sustitución benigna que en la mayoría de los
              ↪ casos provoca un {Counter(dic_variantes[
```

```
            ↪ mutacion_frecuente]['Efecto en mutacion']).
            ↪ most_common(1)[0][0]}\n")

54      #se imprime por pantalla las sustituciones correspondientes a
            ↪ patológicas y benignas
55      print(f'Sustituciones patológicas en el alineamiento: {
            ↪ sust_patologicas}\n')
56      print(f'Sustituciones benignas en el alineamiento: {
            ↪ sust_benignas}\n')
```

La función de análisis de matrices estándar para los alineamientos realizados, sería la descrita en el Listing C.37.

Listing C.37: Código implementado para la función `analisis_pareado_matrices` del *notebook* *Reconocimiento de biomarcadores con Biopython*

```
1  def analisis_pareado_matrices(alineamiento_pareado):
2      """
3      Función que analiza mediante el alineamiento de dos secuencias
           ↪ y el calculo de sus matrices, las posibles mutaciones
4      que pueden tener lugar sobre la secuencia patológica.
5      -------------
6      - alineamiento_pareado: alineamiento pareado de secuencias.
7      ------------
8      Devuelve:
9      No se devuelve objeto alguno.
10     """
11     #se imprime por pantalla el análisis correspondiente
12     print(f"\n ------------------------------ALINEAMIENTO GLOBAL
           ↪ ------------------------------\n")

14     #se imprime por pantalla el número de gaps, matches y
           ↪ mismatches que se tienen en el alineamiento
15     print(f"El alineamiento global tiene {alineamiento_pareado.
           ↪ counts().gaps} huecos, {alineamiento_pareado.counts().
           ↪ identities} identidades, {alineamiento_pareado.counts().
           ↪ mismatches} mismatches")
16     print()

18     #se muestra el alineamiento calculado
19     print(alineamiento_pareado)
```

```
21      #se obtienen las matrices de sustitución y puntuación del
            ↪ alineamineto pareado
22      matriz_sustituciones,matriz_puntuaciones=
            ↪ calcular_matrices_pareado(alineamiento_pareado)
23      print("\n")

25      #se calculan los indels
26      indels=detectar_indels(alineamiento_pareado)

28      #se calculan las sustituciones
29      sustituciones=detectar_sustituciones(alineamiento_pareado)

31      #se obtiene la mutación más frecuente
32      mutacion_frecuente=hallar_mayor_frec(sustituciones)[0]
33      #print(sustituciones)

35      #la frecuencia de la mutación más frecuente también se obtiene
36      frecuencia_mutacion_frecuente=hallar_mayor_frec(sustituciones)
            ↪ [1]

38      #se crea un DataFrame mediante el diccionario de sustituciones
39      sustitucionesDF=pd.DataFrame.from_dict(sustituciones,orient='
            ↪ index')


42      #se imprime por pantalla el resto de información de valor
43      print(f"\n Las sustituciones presentes en el alineamiento son:\
            ↪ n\n {sustitucionesDF}")
44      print(f'\n La sustitución de bases conocidas más habitual
            ↪ presente en el alineamiento es {mutacion_frecuente} con
            ↪ una frecuencia de {frecuencia_mutacion_frecuente}')
```

Para completar la etapa de análisis en la comparación pareada con secuencias patológicas independientes y secuencias consenso control, se declararía y ejecutaría una última función, descrita en Listing C.38. También el resto de las funciones mencionadas, Listing C.31, Listing C.32, Listing C.33, Listing C.34, Listing C.35 y Listing C.37, serían necesarias de ejecutar previamente.

Listing C.38: Código implementado para la función `seleccion_dos_secuencias_aleatorias` del *notebook* *Reconocimiento de biomarcadores con Biopython*

```
def seleccion_dos_secuencias_aleatorias(archivo,formato):
    """
    Devuelve dos secuencias aleatorias de un archivo pasado.
    -------------
    - archivo: archivo que contiene un gran número de secuencias.
    - formato: formato del archivo a leer.
    ------------
    Devuelve:
    Dos secuencias del archivo pasado
    """

    #se leen las secuencias presentes en el archivo y se almacenan en una lista para su siguiente manipulación
    secuencias=list(SeqIO.parse(archivo,formato))

    #se obtienen dos secuencias aleatorias mediante la función randint del paquete random
    secuencia1=secuencias[random.randint(0,len(secuencias)-1)].seq
    secuencia2=secuencias[random.randint(0,len(secuencias)-1)].seq

    #se devuelven las dos secuencias obtenidas
    return secuencia1,secuencia2
```

Para el análisis de alineamientos múltiples, se deben ejecutar hasta tres funciones, siendo una de ellas la encargada de llamar a las otras dos. Dependiendo de si se presentan o no secuencias controles para los diferentes exones, se emplearán las siguientes funciones (Listing C.39, Listing C.40 y Listing C.32).

Listing C.39: Código implementado para la función `calcular_matrices_multiple` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def calcular_matrices_multiple(archivo,formato):
    """
    Calcula las matrices de sustitucion y puntuación de un alineamiento multiple.
    -------------
    - archivo: archivo que contiene el alineamiento múltiple de secuencias.
    - formato: formato del archivo a leer.
    ------------
    Devuelve:
```

```
9       Dos matrices del msa leído, una la matriz de sustituciones, y
            ↪ la otra una matriz de puntuaciones.
10      """

12      #se crea un objeto crear_MultipleSeqAlignment
13      msa=crear_MultipleSeqAlignment(archivo,formato)

15      #se calcula a partir de este su matriz de puntuaciones
16      matriz_puntuacion = hallar_matriz_puntuaciones(msa)

18      #se imprimen por pantalla
19      print("Matriz de sustitución con la frecuencia de los pares de
            ↪ bases en el alineamiento múltiple: \n")
20      print(msa.substitutions)
21      print()
22      print()
23      print("Matriz de puntuación con los cambios más y menos
            ↪ esperados para cada par de bases en el alineamiento
            ↪ pareado es el siguiente: \n")
24      print(matriz_puntuacion)

26      #se devuleven ambas matrices del msa
27      return msa.substitutions, matriz_puntuacion
```

Listing C.40: Código implementado para la función `analisis_multiple_matrices` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1   def analisis_multiple_matrices(archivo,formato):
2       """
3       Realiza un análisis de un msa contenido en un archivo pasado
4       -------------
5       - archivo: archivo que contiene el alineamiento múltiple de
            ↪ secuencias.
6       - formato: formato del archivo a leer.
7       ------------
8       Devuelve:
9       No se devuelve objeto alguno.
10      """

12      #se imprime por pantalla el análisis correspondiente
13      print("\n ------------------------------ALINEAMIENTO MULTIPLE
            ↪ ------------------------------\n")
```

```
15      #se calculan ambas matrices
16      matriz_sustituciones,matriz_puntuaciones=
            ↪ calcular_matrices_multiple(archivo,formato)
17      print("\n")

19      #se detectan las sustituciones presenten en el archivo msa
20      sustituciones=detectar_sustituciones(crear_MultipleSeqAlignment
            ↪ (archivo,formato))

22      #en caso de que el alineamiento múltiple presente sustituciones
23      if len(sustituciones) != 0 :
24          #se obtiene la mutación frecuente
25          mutacion_frecuente=hallar_mayor_frec(sustituciones)[0]

27          #su correspondiente frecuencia también
28          frecuencia_mutacion_frecuente=hallar_mayor_frec(
                ↪ sustituciones)[1]

30          #se crea un DataFrame mediante el diccionario de
                ↪ sustituciones
31          sustitucionesDF=pd.DataFrame.from_dict(sustituciones,orient
                ↪ ='index')
32          print(f"\n Las sustituciones presentes en el alineamiento
                ↪ son:\n\n {sustitucionesDF}")
33          print(f'\n La sustitución más habitual presente en el
                ↪ alineamiento es {mutacion_frecuente} con una
                ↪ frecuencia de {frecuencia_mutacion_frecuente}\n\n')
```

Listing C.41: Código implementado para la función `comparacion_msa` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1   def comparacion_msa(archivo_control,archivo_patologica,formato):
2       """
3       Compara mediante llamadas a otras funciones dos archivos msa.
4       -------------
5       - archivo_control: archivo que contiene el alineamiento mú
            ↪ ltiple de secuencias controles.
6       - archivo_patologica: archivo que contiene el alineamiento mú
            ↪ ltiple de secuencias patológicas.
7       - formato: formato de los archivos a leer.
8       ------------
9       Devuelve:
```

```
    No se devuelve objeto alguno.
    """
    #se imprime por pantalla el análisis del msa de controles y enfermos
    print("\n ----------------------CONTROL----------------------\n")
    analisis_multiple_matrices(archivo_control,formato)
    print("\n ----------------------PATOLOGICO----------------------\n")
    analisis_multiple_matrices(archivo_patologica,formato)
```

Por último, para la visualización de los alineamientos múltiples de `Clustal` y `Muscle` sobre las secuencias patológicas y controles, se deberán ejecutar las funciones correspondientes [Shtrauss, 2023], codificadas en el Listing C.42 y Listing C.43. En caso de optar por una descripción visual de los alineamientos en vertical, típica de *sequence logo*, se tendrá que ejecutar (Listing C.44).

Listing C.42: Código implementado para la función `visualizacion_msa` del *notebook Reconocimiento de biomarcadores con Biopython*

```
def visualizacion_msa(alineamientos, tamano_fuente="9pt", anchura_grafico=800):
    """
    Función que imprime por pantalla una visualización interactiva de los alineamientos pasado por argumento.
    -------------
    - alineamientos: alineamientos múltiples a visualizar grá ficamente.
    - tamano_fuente: tamaño de los nucleótidos de las secuencias a visualizar.
    - anchura_grafico: anchura del gráfico a mostrar.
    ------------
    Devuelve:
    Gráfico interactivo del msa obtenido mediante la función gridplot de la biblioteca de Bokeh.
    """

    #obtención de secuencias de los alinemamientos
    secuencias = [registro.seq for registro in (alineamientos)]
    #obtención de los ids de los alineamientos
    ids = [registro.id for registro in alineamientos]
    #obtención de los nucleótidos de cada secuencia
    text = [n for secuencia in list(secuencias) for n in secuencia]
```

```
19      #obtención de colores en función de los carácteres presentes en
            ↪ las secuencias
20      colores = get_colors(secuencias)
21      #longitud de la primera secuencia
22      N = len(secuencias[0])
23      #número de secuencias que conforman el alineamiento
24      S = len(secuencias)
25      width = .4

27      #Dimensiones del alineamiento proporcionado
28      x = np.arange(0.5,N+0.5)
29      y = np.arange(0,S,1)
30      #arrays 2D con las cuadrículas de las coordenadas
31      xx, yy = np.meshgrid(x, y)
32      #arrays con las coordenadas
33      gx = xx.ravel()
34      gy = yy.flatten()
35      #coordenadas ajustadas para los rectángulos
36      recty = gy+.5
37      h= 1/S
38      #datos proporcionados a Bokeh, que contiene coordenadas,texto y
            ↪ colores
39      source = ColumnDataSource(dict(x=gx, y=gy, recty=recty, text=
            ↪ text, colors=colores))
40      #altura del gráfico
41      altura_grafico = len(secuencias)*15+50
42      x_range = Range1d(0,N+1, bounds='auto')
43      if N>100:
44          viewlen=100
45      else:
46          viewlen=N
47      #rango de visualizaciñon inicial
48      rango_visualizacion = (0,viewlen)
49      tools="xpan, xwheel_zoom, reset, save"

51      #objeto interactivo personalizado de Bokeh
52      p = figure(title=None, plot_width=anchura_grafico, plot_height=
            ↪ altura_grafico,
53                 x_range=rango_visualizacion, y_range=ids, tools="
                       ↪ xpan,reset",
54                 min_border=0, toolbar_location='below')#, lod_factor
                       ↪ =1)
```

```
    #para mostrar los carácteres en forma de texto de las secuencias
    glyph = Text(x="x", y="y", text="text", text_align='center', text_color="black",
                 text_font="monospace",text_font_size=tamano_fuente)
    #para mostrar los carácteres de la secuencia con colores
    rects = Rect(x="x", y="recty", width=1, height=1, fill_color="colors",
                 line_color=None, fill_alpha=0.4)
    p.add_glyph(source, glyph)
    p.add_glyph(source, rects)

    #se ocultan las líneas de las cuadrículas
    p.grid.visible = False
    #estilos de las etiquetas para el eje x
    p.xaxis.major_label_text_font_style = "bold"
    #se ocultan a su vez marcas mayores y menores del eje y
    p.yaxis.minor_tick_line_width = 0
    p.yaxis.major_tick_line_width = 0

    #grafico ya configurado a mostrar
    p = gridplot([[p]], toolbar_location='below')
    return p
```

Listing C.43: Código implementado para la función `get_colors` del *notebook* *Reconocimiento de biomarcadores con Biopython*

```
def get_colors(secuencias):
    """
    Función que genera una lista de colores cada uno de los cuales está asociado a un caracter de las secuencias pasadas.
    -------------
    - secuencias: secuencias pasadas a alfabecitar por colores.
    ------------
    Devuelve:
    lista con los colores asociados a los nucleótidos.
    """
    #carácteres en formato texto de las secuencias
    text = [n for s in list(secuencias) for n in s]
    #diccionario de correspondientes a los diferentes carácteres
    clrs = {'A':'green','T':'red','G':'orange','C':'blue','-':'white','R':'purple'}
    #listado con los colores de los nucleótidos mapeados
```

```
15      colors = [clrs[n] for n in text]
16      return colors
```

Listing C.44: Código implementado para la función `grafico_sequencelogo` del *notebook Reconocimiento de biomarcadores con Biopython*

```
1  def grafico_sequencelogo(ruta_archivo,formato,limite_inicial,
       ↪ limite_final):
2      """
3      Función que imprime por pantalla un gráfico sequence logo.
4      -------------
5      - ruta_archivo: ruta del archivo que contiene el msa a
           ↪ representar
6      - formato: formato del archivo poseedor del msa.
7      - limite_inicial: número de posición de la secuencia a
           ↪ representar que actúa como límite inicial.
8      - limite_inicial: número de posición de la secuencia a
           ↪ representar que actúa como límite final.
9      ------------
10     Devuelve:
11     Gráfico interactivo del msa obtenido mediante la función
           ↪ gridplot de la biblioteca de Bokeh.
12     """

14     #lectura del archivo msa
15     secuencias_alineadas=list(SeqIO.parse(ruta_archivo,formato))

17     #se recorre la lista de secuencias alineadas, agregándolas ya
           ↪ limitadas a una nueva lista
18     secuencias= [str(registro.seq)[limite_inicial:limite_final] for
           ↪  registro in secuencias_alineadas]

20     #se convierte la lista de secuencias limitadas en una matriz de
           ↪  conteos
21     DF_conteos=logomaker.alignment_to_matrix(secuencias, to_type="
           ↪ counts")

23     #se crea un objeto Logo pendiente de representar mediante dicha
           ↪  matriz
24     logo = logomaker.Logo(DF_conteos)

26     #se imprime por pantalla el gráfico elaborado
27     plt.show()
```

Para una descripción y explicación más detalladas de cada uno de los procesos incluidos en el en *notebook Reconocimiento de biomarcadores con Biopython*, acudir a los anexos *B: Demostraciones prácticas* y *D: Tratamiento.*

C.3. Instrucciones para la modificación o mejora del proyecto.

La mejora dentro de un proyecto siempre puede ser posible, y cualquier modificación realizada con la intención de mejorarla, aumentará el valor del proyecto. En lo que a un contexto programático se refiere, las mejoras que pueden realizarse son múltiples, ya sea un cambio destinado a la optimización, eficiencia o claridad del código inicialmente implementado.

Para este proyecto existen ciertas mejores que podrían realizarse a futuro:

- La metodología de divide y vencerás aplicada en el código desarrollado podría ser mayor, con el propósito de añadir una claridad, comprensión y brevedad adicionales.
- Las estructuras del propio lenguaje utilizadas podrían ser distintas. Y cuya elección podría ser influenciada, en función del objetivo que se tenga, aumentar la manejabilidad de los datos, lograr una mayor optimización del tiempo de ejecución, mayor comodidad personal,etc. Destacando los dos primeros puntos, debido a que en un trabajo a gran escala, son unos de los aspectos que más importancia reciben dado su impacto sobre el proyecto.
- Las funciones y código elaborados podrían presentar una mejor implementación, mediante el uso de otras herramientas o paquetes alternativos que realicen las mismas acciones, de una forma más eficiente. Incluso, en algunas circunstancias realizarse con un código más sencillo que el propuesto.
- Asimismo, plantear un mismo proyecto con metodologías y estrategias distintas, como un procesamiento de datos basado en minería de datos y una clasificación automatizada de los biomarcadores encontrados mediante algún sistema de aprendizaje automático presente en `Python`.
- El uso de herramientas bioinformáticas compatibles con `Python` o externas. `MSA browser`, `DNABert`, `DeepVariant`, existen infinidad de posibilidades.

- La utilización de algoritmos alternativos, como es el caso de `T-Coffee` [Cedric Notredame and Comparative Bioinformatics Group, 2024], podrían ser interesantes de evaluar.
- Incluso la implementación de una App web o API, que a partir de los avances logrados hasta el momento y a través de una plataforma de desarrollo de web o *framework*, se logre identificar el sentido clínico de unas muestras pasadas.

Se recomienda consultar el apartado *7:líneas futuras*, contenido en la memoria del proyecto, se encuentran detalladas un mayor número de mejoras adicionales, desde el punto de vista biológico e informático.

Apéndice D

Descripción de adquisición y tratamiento de datos

Gran parte de los datos fueron recogidos del Centro Nacional para la Información Biotecnológica de Estados Unidos, mejor conocido como *NCBI*, y principalmente de la base de datos de *Nucleotide*, mediante la funcionalidad de *clipboard* que presenta la biblioteca.

Los cuales pueden dividirse en dos conjuntos de datos principales: uno asociado a pacientes enfermos y los otros a pacientes sanos que actúan como controles en este proyecto. Cuatro archivos de formato `fasta` por parte de los pacientes enfermos, y cinco archivos más del mismo formato por los pacientes sanos. Cada uno de los archivos, correspondientes a los genes y regiones codificantes a estudio causantes de cáncer de mama: *BRCA1, BRCA2, TP53* y *PIK3CA*.

Cada archivo `fasta` de cada gen, contiene un número variable de secuencias de ADN, todas ellas en sentido 5⟶3'(cadenas codificantes) de diferentes tamaños y contenidos, pero con una estructura común.

El resto, fue extraído de la base de datos de *gnomAD*. Consisten en un par de archivos de formato `csv`, que representan la información de diferentes variantes genéticas documentadas de forma tabular.

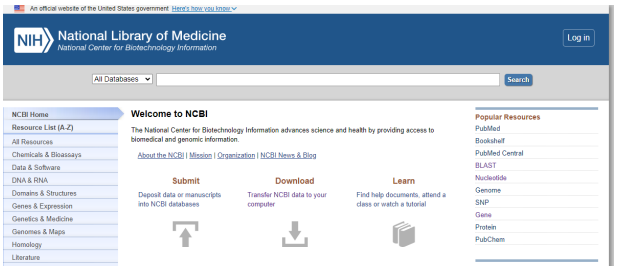


Figura D.1: Portal de inicio de *NCBI*. Fuente: [NCBI, 2024].

```
>OM524595.1 Homo sapiens isolate Sh520B1 breast cancer type 1 susceptibility protein (BRCA1) gene,
exon 20 and partial cds
AAGATCTTCTGATCCAGTAGTGTTCTGGACATTGGACTGCTTGTCCCTGGGAAGTAGCAGCAGAAATCAT
CAGGTGGTGAACAGAAGAAAAAGAAAAGCTCTTCCTTTTTGAAAGTCTGTTTTTTGAATAAAAGCCAATA
TTCTTTTATAACTAGATTTTCCTTCTCTCCATTCCCCTGTCCCTCTCTCTTCCTCTCTTCTTCCAGATCT
TCAGGGGGCTAGAAATCTGTTGCTATGGGCCCTTCACCAACATGCCCACAGGTAAGAGCCTGGGAGAACC
CCAGAGTTCCAGCACCAGCCTTTGTCTTACATAGTGGAGTATTATAAGCAAGATCCCACGATGGGGGTTC
CTCAGATTGCTGAAATGTTCTAGAGGCTATTCTATTTCTCTACCACTCTCCAAACAAAACAGCACCTAAA
TGTTATCCTATGGCAAAAAAAAACTATACCTTGTCCCCCTTCTCAAGAGCATGAAGGTGGTTAATAGTTA
GGATTCAGTATGTTATGTGTTCAGTGGC

>OM524594.1 Homo sapiens isolate Sh420B1 breast cancer type 1 susceptibility protein (BRCA1) gene,
exon 20 and partial cds
AAGATCTTCTGAATCCATGTAGTGTTCTGGACATTGGACTGCTTGTCCCTGGGAAGTAGCAGCAGAAATC
ATCAGGTGGTGAACAGAAGAAAAAGAAAAGCTCTTCCTTTTTGAAAGTCTGTTTTTTGAATAAAAGCCAA
TATTCTTTTATAACTAGATTTTCCTTCTCTCCATTCCCCTGTCCCTCTCTCTTCCTCTCTTCTTCCAGAT
CTTCAGGGGGCTAGAAATCTGTTGCTATGGGCCCTTCACCAACATGCCCACAGGTAAGAGCCTGGGAGAA
CCCCAGAGTTCCAGCACCAGCCTTTGTCTTACATAGTGGAGTATTATAAGCAAGATCCCACGATGGGGGT
TCCTCAGATTGCTGAAATGTTCTAGAGGCTATTCTATTTCTCTACCACTCTCCAAACAAAACAGCACCTA
AATGTTATCCTATGGCAAAAAAAAACTATACCTTGTCCCCCTTCTCAAGAGCATGAAGGTGGTTAATAGT
TAGGATTCAGTATGTTATGTGTTCAGTGGC
```

Figura D.2: Ejemplo secuencias fasta patológicas *BRCA1*. Fuente: elaboración propia.

D.1. Descripción formal de los datos

Descripción de los ficheros

Los tipos de ficheros de datos incluidos en el proyecto se pueden resumir en unos ficheros de formato `fasta` utilizados para el almacenamiento de las diferentes secuencias, y un par de ficheros de formato `csv`, con las variantes pertenecientes a los genes *BRCA2* y *PIK3CA*.

Los nuevos ficheros `fasta` originales se encuentran estructurados en dos partes principales:

- **Encabezado**:
  - **Identificador** de la secuencia: número identificador único de la secuencia biológica de la base de datos (*NCBI*).
  - **Organismo**: organismo al que pertenece la secuencia contenida.

    - **Descripción**: normalmente la descripción del gen asociado a la secuencia.

- **Secuencia**: secuencia de nucleótidos (ADN) con la información genética del paciente.

El tamaño de los archivos es equivalente al número de registros que contiene el archivo. Como para cada gen de interés se presentan un heterogéneo número de secuencias nucleotídicas, el tamaño también lo será, pudiendo variar entre los 396.000 kilobytes para los archivos más grandes, y los 2 kilobytes para los más pequeños, que generalmente serán los controles.

Y en el caso de los ficheros con extensión `csv`, se encuentra una estructura relacional, de tipo tabla. En la que cada variante representará una tupla conformada por diversos atributos.

Por otro lado, el peso de estos también depende mucho del gen del cual contenga las variantes, siendo de 2214 kilobytes para las pertenecientes a *BRCA2* y 335 kilobytes para las de *PIK3CA*.

Descripción de las tablas

Las tablas presentes en el proyecto provienen de los archivos con extensión `csv`, los cuales fueron obtenidos de la base de datos *gnomAD* [gnomAD Steering Committee, 2024]. Los archivos desde una perspectiva relacional exponen diferentes cardinalidades al contener distinto número de tuplas, aunque los mismos grados, dado a que el estudio de cada tupla/variante se realiza sobre los mismos atributos.Para ambas relaciones, la clave primaria utilizada se corresponde con el primer atributo o nombre *gnomAD* ID.

Dentro de los atributos que contienen cada una de las relaciones, aquellos a destacar serían:

- **VEP Annotation**: Anotación del efecto predictor de la variante, ya sea sinónima, se traduzca en un cambio de marco o carezca de sentido.
- **Clinical Significance**: se trata de aquel atributo que determina el diagnostico clínico de la variante.
- **Allele Count**: es el recuento de variantes en genotipos de alta calidad.
- **Allele Number**: número de genotipos que se consideran de alta calidad.

- **Allele Frequency**: la frecuencia de la variante en genotipos de alta calidad.(se obtiene de la división entre *Allele Count* y *Allele Number*).
- **Number of Homozygotes**: número de individuos homocigotos para la variante.
- **Number of Hemizygotes**:número de individuos heterocigotos para la variante.
- **HGVSc of Consequence**: secuencia codificante de la variante para las consecuencias más graves en la transcripción.
- **HGVSp of Consequence**: secuencia de proteínas de la variante para las consecuencias más graves en la transcripción.

D.2. Descripción clínica de los datos.

Obtención de los datos

Secuencias

En fases tempranas del proyecto se elaboró en un primer momento un análisis de aquellos carcinomas más comunes en la población a nivel global. Donde se pudo comprobar que los carcinomas de mama, próstata, pulmón, colon y recto; encabezaban los nuevos diagnósticos de carcinoma para el año 2024. Siendo la suma de ellos la causa del 48 % de los cánceres detectados en hombres y el 51 % de los cánceres detectados en mujeres para el mismo año (Figura D.3).

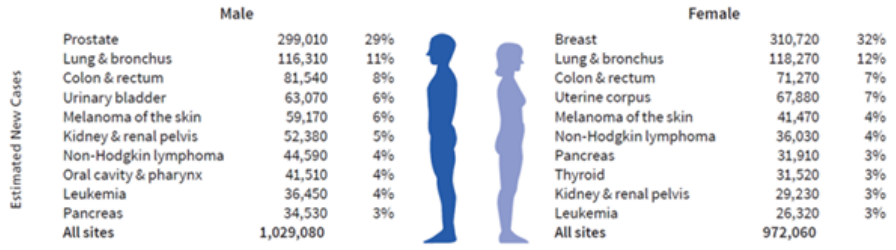


Figura D.3: Número estimado de nuevos cánceres en 2024 para hombres y mujeres. Fuente: [Siegel et al., 2024].

Con ello, y a partir de diferentes búsquedas generales sobre la biblioteca de *NCBI* [NCBI, 2024], se fueron anotando las diferentes cantidades de

Tipo de cáncer	**Secuencia de ADN**	**Genes**	**ARN**	**Proteínas**	**Número de secuencias disponibles**
Próstata	95588	92756	273835	16031	371112
Mama	134004	4009	308091	6880	444574
Colon	38114	23456	1748	123	40544
Pulmón	22501	228489	292671	24598	319404

Tabla D.1: Tabla con los diferentes números de resultados para cada carcinoma y campo

datos que se disponían de unos y otros carcinomas. Los resultados obtenidos son los presentados en la Tabla D.1.

El cáncer de mama sería finalmente el escogido, dada la inmensa cantidad de información de la cual se podría dar uso en múltiples proyectos bioinformáticos con diferentes intereses, y al propio interés personal. en busca de un mayor conocimiento y entendimiento del cáncer de mama, dada su importancia clínica y alta frecuencia en mujeres.

Por otro lado, la base de datos de *TCGA*, junto con la literatura existente en la nube consultada, aportaron al desarrollo del proyecto aquellos genes que mayor tasa de mutación y relación presentaban con el cáncer en cuestión. Estableciendo con ello los genes *TP53* y *PIK3CA* por parte de *TCGA* (Figura D.4) como los dos genes de mayor importancia; y *BRCA1* y *BRCA2* como los genes por excelencia causantes de cáncer de mama según la literatura clínica.

Dado que el dato biológico de principal interés eran las secuencias nucleotídicas de las regiones codificantes (o exones), se volvió a recurrir a la biblioteca del *NCBI*. Debido a la gran ventaja que supone su uso, respecto a otras bases de datos en cuanto a la fácil accesibilidad que muestra para la obtención de todo tipo de datos; y en especial secuencias de nucleótidos, un dato demandado y difícil de encontrar en un estado bruto.

Como ya se comentó inicialmente los datos fueron recogidos del Centro Nacional para la Información Biotecnológica de Estados Unidos, mejor conocido como *NCBI*, y principalmente de la base de datos de *Nucleotide*.

Dentro de la plataforma de *NCBI*, y mediante la utilización de diferentes términos de interés unidos por conectores lógicos en la búsqueda de los genes, se obtienen un número de resultados variable para cada una de las diferentes bases de datos que integran el *NCBI*. Entre ellas se escogió la base de

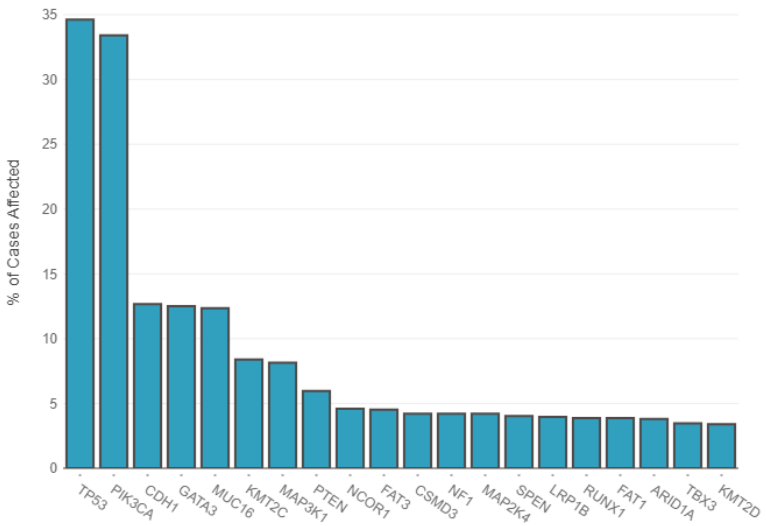


Figura D.4: Gráfica de *TCGA* que presenta los genes ordenados por frecuencia de mutación. Fuente: [NCI and National Human Genome Research Institute, 2006].

Nucleotide que contiene las secuencias de ADN y ARN e información relativa de diversos genes procedentes del conjunto de bases de datos de *Genbank, EMBL* y *DDBJ*; siendo aquella que más se ajustaba a las necesidades a cubrir por las metas planteadas.

Al acceder a esta, se encuentran numerosos *items* que contienen diferente información genómica relacionada con la búsqueda de términos realizada. A la vez que diferentes filtros por defecto que suelen ser suelen los más habituales de utilizar, y que en la obtención de datos del proyecto se llegaron a usar. Entre ellos, destacar el tipo de organismo del que serían los datos a mostrar y el tipo de moléculas a tratar. En el caso de este proyecto, los valores escogidos fueron *Homo sapiens* y *DNA/RNA genómico*, dado que se pretendía encontrar una aplicación de interés clínico directa sobre nuestra especie y sobre las secuencias nucleotídicas de los genes en cuestión (Figura D.5).

Con la incorporación de algún término y palabra clave adicional, dentro de la modalidad de búsqueda avanzada que influye la página web; se llevó a cabo una descarga masiva de todos los *items* del resultado de búsqueda logrado. Aunque esta fue la única manera viable que se encontró para optimizar el tiempo del proceso de obtención de los datos con aquellos que

Figura D.5: Ejemplo de búsqueda en *NCBI*. Fuente: [NCBI, 2024].

efectivamente eran de interés, supuso una dificultad añadida al obtener información genómica con una forma y cantidad muy variables dependiendo de casos, y, provocando con ello la reducción de la fiabilidad y lógica de los resultados que se obtuvieron (Figura D.6).

Se presentaban una serie de opciones al crear el archivo a descargar, que dependiendo de los intereses del proyecto, se escogieron unos y se descartaron otros.El registro completo sería una ventaja posterior en el tratamiento de secuencias desde el *notebook* de `Python` [Python Software Foundation, 2024], además de la elección de un formato conocido como es el formato `fasta` y la agregación de identificadores a los diferentes registros, para tener una referencia constante hacia cada una de las secuencia a estudiar y posibilitar la opción de obtener información adicional de valor en la biblioteca de *NCBI* en cualquier momento.

Variantes genéticas

Como medida comparativa de los resultados obtenidos mediante el empleo de diferentes técnicas utilizadas y presentadas en el apéndice de resultados, se realizó la descarga de unos archivos `csv` provenientes de la base de datos de *gnomAD*.

GnomAD se trata de una base de datos desarrollada por una coalición internacional de investigadores con el objetivo de juntar y relacionar datos de la secuenciación del genoma procedentes de múltiples fuentes. Dentro de los contenidos que presenta, uno de los más abundantes e interesantes son las variantes existentes que almacena de un gran número de genes, entre

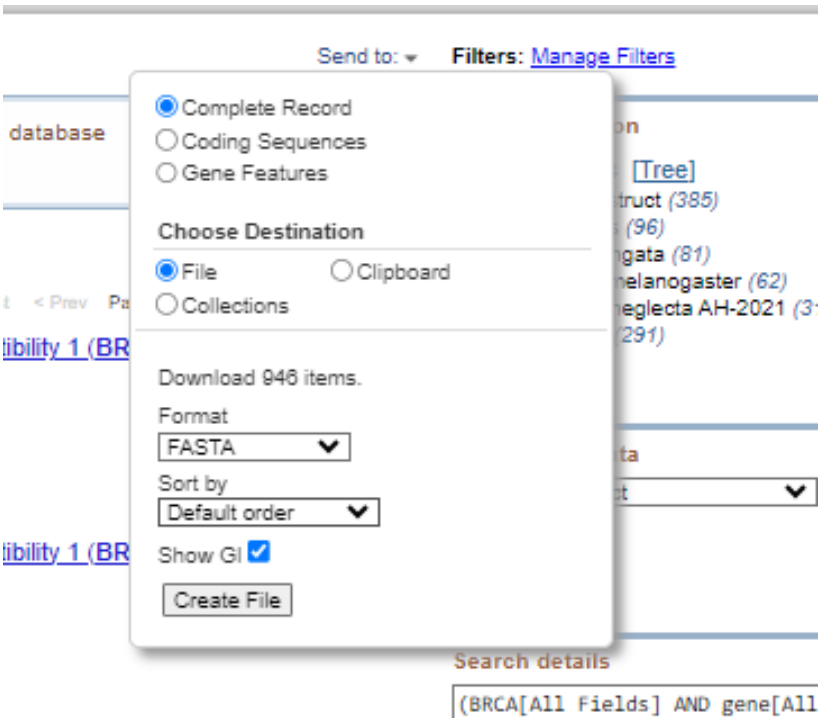


Figura D.6: Ejemplo de descarga de secuencias en *NCBI*. Fuente: [NCBI, 2024].

ellos, los cuatro genes de interés para este proyecto. Dicha razón, la convierte en una fuente de datos de gran beneficio para las metas establecidas, pueden realizar la comparativa ya mencionada con los resultados obtenidos (Figura D.7).

Sin embargo, las variantes almacenadas se presentar el sentido de la hebra codificante (5'→ 3') o de la hebra molde (3'→ 5'), y al realizarse la búsqueda de los genes *BRCA1* y *TP53*, se pudo comprobar que las variantes correspondientes formaban parte de la hebra molde, la hebra contraria de la cual se disponían los datos.

Este inconveniente produjo la necesidad de realizar una comparativa de las variantes presentes en los `csvs` con los genes de *BRCA2* y *PIK3CA*, es decir, aquellas que si se almacenaban en el sentido de la hebra codificante o 5'→ 3'; y recurrir a la literatura para analizar las diferentes mutaciones en *BRCA1* y *TP53*.

La descarga de los datos fue accesible y rápida, con una opción para ello mostrada en cada búsqueda realizada. En esta se permitía la obtención

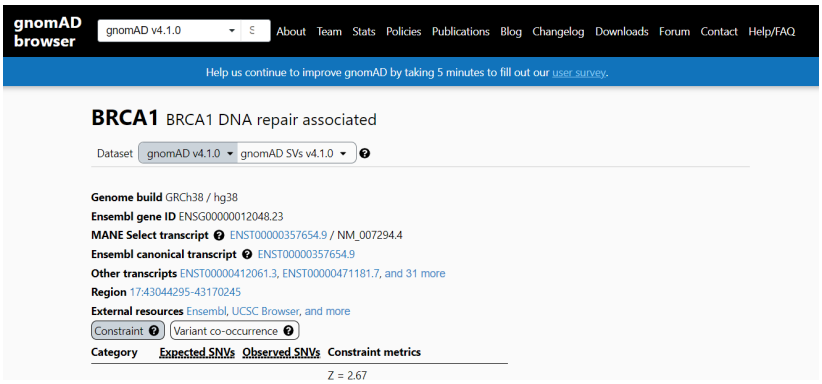


Figura D.7: Búsqueda de variantes de *BRCA1* en *gnomAD*. Fuente: [gnomAD Steering Committee, 2024].

de información relacional o tabulada en la que se pudo incluir cualquier atributo de valor, tal y como se puede observar en la Figura D.8.

Figura D.8: Tabla de variantes de *BRCA2* procedente de *gnomAD*. Fuente: elaboración propia.

Tratamiento de datos

Tratamiento para archivos `fasta`

Datos Patológicos Para poder realizar el análisis sobre los datos biológicos, primeramente deberían de ser leídos y almacenados en unas variables globales del *notebook* desarrollado. Para lograr tal objetivo se crearon las funciones `obtener_tipo_extensión` y `obtener_registros`. Con la función `parse()` del paquete `SeqIO`, proveniente del paquete *Biopython*, se define

la función `obtener_registro`, y se logra la lectura de registros o objetos `SeqRecord` presentes en los archivos `fasta` que contienen las secuencias a preprocesar.Un ejemplo de su ejecución se muestra en la Figura D.9.

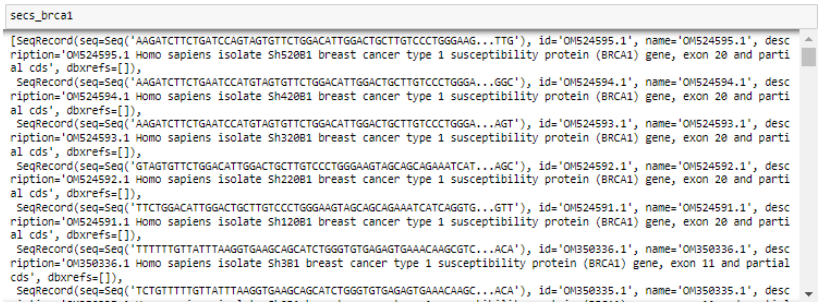


Figura D.9: Lista de objetos SeqRecord procedentes del archivo fasta de *BRCA1*. Fuente: elaboración propia.

Con su aplicación se obtuvieron un listado con todos los registros incluidos en cada archivo de cada gen. Ya con los datos pudiendo ser manipulables, se pretendió calcular el número de regiones codificantes que contenía cada lista, es decir, el número de exones; aquello que será analizado para la detección de biomarcadores. Esta vez, en lugar de recurrir al desarrollo de una función se optó por la ejecución del filtrado sobre una misma línea de código, tal y como se muestra en la Figura D.10.

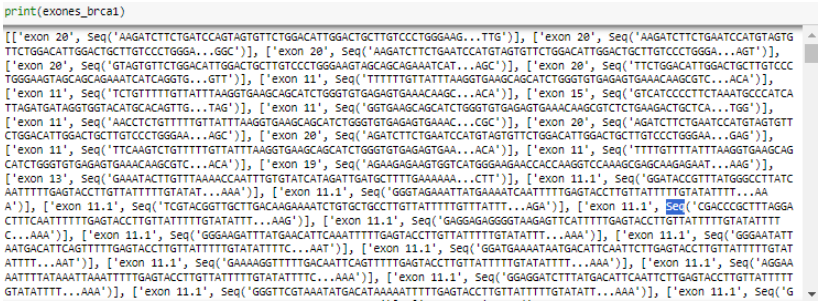


Figura D.10: Exones y secuencia asociada pertenecientes a *BRCA1*. Fuente: elaboración propia.

Al contar el número de exones, se obtuvieron números muy variables entre los distintos genes, haciendo ya visible que la cantidad de información que almacenan los diferentes genes no es homogénea, y presentan una mayor prioridad aquellos que presentan un interés clínico adicional. Al mismo

tiempo, y sobre la misma línea de código mencionada, se consiguieron unas listas que contenían las ya nombradas regiones codificantes en los correspondientes registros, las cuales fueron nuevamente almacenadas en una lista.

El desarrollo de una nueva función, `exones`, da lugar a que por cada gen de interés, se genere una lista de listas de pares, compuestas cada una de ellas por el número de exon, y su secuencia asociada. Con las estructuras obtenidas, se generan unas nuevas más optimizadas para conseguir el objetivo que se pretendía obtener, pasándose de una lista de listas, a un diccionario cuyo par estará conformado por una clave que será el exón, y un lista con todas las secuencias que tuvieran asociado dicho exón. De esta forma, se agrupan las secuencias y se garantiza una mayor facilidad de filtrado (Figura D.11).

```
{'exon 20': [Seq('AAGATCTTCTGATCCAGTAGTGTTCTGGACATTGGACTGCTTGTCCCTGGGAAG...TTG'),
  Seq('AAGATCTTCTGAATCCATGTAGTGTTCTGGACATTGGACTGCTTGTCCCTGGGA...GGC'),
  Seq('AAGATCTTCTGAATCCATGTAGTGTTCTGGACATTGGACTGCTTGTCCCTGGGA...AGT'),
  Seq('GTAGTGTTCTGGACATTGGACTGCTTGTCCCTGGGAAGTAGCAGCAGAAATCAT...AGC'),
  Seq('TTCTGGACATTGGACTGCTTGTCCCTGGGAAGTAGCAGCAGAAATCATCAGGTG...GTT'),
  Seq('AGATCTTCTGAATCCATGTAGTGTTCTGGACATTGGACTGCTTGTCCCTGGGAA...AGC'),
  Seq('AGATCTTCTGAATCCATGTAGTGTTCTGGACATTGGACTGCTTGTCCCTGGGAA...GAG'),
  Seq('TATATGACGTGTCTGCTCCACTTCCATTGAAGGAAGCTTCTCTTTCTCTTATCC...CAC'),
  Seq('TTCTTTCAGCATGATTTTGAAGTCCGAGGAGATGTGGGCAATGGAAGAAACCAC...CCA'),
  Seq('ATATATGACGTGTCTGCTCCACTTCCATTGAAGGAAGCTTCTCTTTCTCTTATC...CCC'),
  Seq('ATGATTTTGAAGTCCGAGGAGATGTGGGCAATGGAAGAAACCACCACGGTGCAA...CCA'),
  Seq('ATATGACGTGTCTGCTCCACTTCCATTGAAGGAAGCTTCTCTTTCTCTTATCCT...ACT')],
 'exon 11': [Seq('TTTTTTGTTATTTAAGGTGAAGCAGCATCTGGGTGTGAGAGTGAAACAAGCGTC...ACA'),
  Seq('TCTGTTTTTGTTATTTAAGGTGAAGCAGCATCTGGGTGTGAGAGTGAAACAAGC...ACA'),
  Seq('GGTGAAGCAGCATCTGGGTGTGAGAGTGAAACAAGCGTCTCTGAAGACTGCTCA...TGG'),
  Seq('AACCTCTGTTTTTGTTATTTAAGGTGAAGCAGCATCTGGGTGTGAGAGTGAAAC...CGC'),
  Seq('TTCAAGTCTGTTTTTGTTATTTAAGGTGAAGCAGCATCTGGGTGTGAGAGTGAA...ACA'),
  Seq('TTTTGTTTTATTTAAGGTGAAGCAGCATCTGGGTGTGAGAGTGAAACAAGCGTC...ACA'),
```

Figura D.11: Exones y secuencias asociadas pertenecientes a un gen en forma de diccionario. Fuente: elaboración propia.

Sobre estos diccionarios, son de interés aquellos exones que presenten unas listas más largas como valores, ya que será un parámetro para determinar el número de secuencias presentes para analizar. De forma que cuanto más largas sean estas listas, mayores datos, variabilidad y representación se obtendrá. Con tal de conseguir dicha meta, se realizó una nueva función que determinara, a partir del diccionario, y un número de corte utilizado como umbral, aquellos exones que presentaran un mayor número de secuencias. El nombre que recibió dicha función fue `max_secs`, y mostraba por pantalla lo contenido (Figura D.12).

Tras la llamada de la función se consiguieron los exones con mayor frecuencia (Figura D.2), y se crearon a partir de ellos unos diccionarios que únicamente tuviera en cuenta a estos, y sus secuencias asociadas.

Figura D.12: Diccionario con los exones con mayor número de secuencias. Fuente: elaboración propia.

BRCA1	Exón 5	Exón 19
BRCA2	Exón 2	Exón 11
TP53	Exón 5	Exón 8
PIK3CA	Exón 9	Exón 20

Tabla D.2: Tabla con los exones que presentan una mayor frecuencia en nuestros datos

Finalizando ya la etapa de filtrado y antes de realizar ningún análisis sobre las secuencias de los exones filtrados, se debían comprobar las longitudes de cada secuencia correspondiente a cada exón, de forma que estas no presentaran grandes variaciones de longitud entre sí y fueran lo más similares posibles; ya que, de ser así, nuestro análisis no sería considerado con la suficiente calidad como para ser válido. Entonces, aquellas secuencias consideradas como valores *outliers* o atípicos, además de aquellas lo suficientemente largas o cortas como para que peligrara la comparación a realizar, no tendrían cabida en nuestras secuencias filtradas.

La función `calcular_mediana` junto con `comprobar_longitudes` se encargarían de evaluar las longitudes de cada una de las secuencias de los exones, en función de una mediana establecida a partir de las diferentes longitudes que exponen las secuencias. La elección de la mediana sobre la media, está justificada en los altos valores *outliers* que pueden encontrarse entre las longitudes de las secuencias u objetos `Seq` asociados a un exón. De haber escogido el cálculo de la media, se habría obtenido un filtrado incorrecto, ya que el cálculo de esta métrica matemática, se basa en un

cálculo conjunto de todos los valores que conforman la lista de longitudes de las secuencias; cálculo el cual es extremadamente sensible a valores extremos, ya que estos no son representativos del conjunto (Figura D.13).

```
print("Número de secuencias para el primer exon, antes y después del filtrado")
print(len(secuencias_brca1[list(secuencias_brca1.keys())[0]]))
print(len(secuencias_brca1_filtradas[list(secuencias_brca1_filtradas.keys())[0]]))
print()

print("Número de secuencias para el segundo exon, antes y después del filtrado")
print(len(secuencias_brca1[list(secuencias_brca1.keys())[1]]))
print(len(secuencias_brca1_filtradas[list(secuencias_brca1_filtradas.keys())[1]]))
```

```
Número de secuencias para el primer exon, antes y después del filtrado
97
88

Número de secuencias para el segundo exon, antes y después del filtrado
95
92
```

Figura D.13: Diccionario con los exones con mayor número de secuencias. Fuente: elaboración propia.

En último lugar, las secuencias filtradas debieron ser almacenadas en nuevos archivos `fasta` para mejorar su posterior manipulación y correcta ejecución de las funciones de análisis a aplicar. Para ello, se hizo una función que actuara de filtro mediante la comparación del archivo original con las distintas secuencias de los diccionarios finales, y su guardado en el dispositivo mediante la función `write()` del paquete `SeqIO`.

Ya con ello el tratamiento de los datos de formato `fasta` de origen patológico habría finalizado.

Datos Control La obtención de estos datos fue ligeramente distinta respecto a los datos patológicos, ya que estos fueron tomados posteriormente y dependían de ellos. Una vez conocidos aquellos exones de mayor frecuencia o más información se tenía en el archivo obtenido de *NCBI*; la obtención de los controles estuvo determinada por esas regiones codificantes ya conocidas; incluyendo en los términos de búsqueda específicamente el número de los dos exones por gen. De esta manera, el tratamiento sería menos complejo y más breve que respecto al caso de las secuencias patológicas, pero presentando todavía ciertos procesamientos comunes mediante las mismas funciones empleadas y comentadas en la subsección anterior.

Tal y como se realizó con las secuencias patológicas se comenzó leyendo y guardando las secuencias en variables locales con la función `obtener_registros` (Figura D.14). También se comprobó la cantidad de

registros que se disponían por archivo, observándose menor cantidad que en el caso patológico y con una gran variabilidad entre ellas.

```
[SeqRecord(seq=Seq('TCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTTAAAAGGTT...ACC'), id='OR780020.1', name='OR780020.1', descrip
tion='OR780020.1 Homo sapiens isolate n18 BRCA1 (BRCA1) gene, exon 5 and partial cds', dbxrefs=[]),
 SeqRecord(seq=Seq('CTACTGTTGCTGCATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGT...ACC'), id='OR780019.1', name='OR780019.1', descrip
tion='OR780019.1 Homo sapiens isolate b2 BRCA1 (BRCA1) gene, exon 5 and partial cds', dbxrefs=[]),
 SeqRecord(seq=Seq('ATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTTAAAAGGT...TCA'), id='OR780018.1', name='OR780018.1', descrip
tion='OR780018.1 Homo sapiens isolate n8 BRCA1 (BRCA1) gene, exon 5 and partial cds', dbxrefs=[]),
 SeqRecord(seq=Seq('ATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTTAAAAGGT...TCA'), id='OR780017.1', name='OR780017.1', descrip
tion='OR780017.1 Homo sapiens isolate b26 BRCA1 (BRCA1) gene, exon 5 and partial cds', dbxrefs=[]),
 SeqRecord(seq=Seq('ATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTTAAAAGGT...CCA'), id='OR780016.1', name='OR780016.1', descrip
tion='OR780016.1 Homo sapiens isolate b14 BRCA1 (BRCA1) gene, exon 5 and partial cds', dbxrefs=[])]
```

Figura D.14: Lista con los registros pertenecientes a un archivo `fasta` de secuencias controles. Fuente: elaboración propia.

Nuevamente, las regiones codificantes eran aquellas que mayor interés presentaban en este proyecto; de forma que también se elaboró el mismo código de línea utilizado anteriormente para la obtención de una lista con los registros correspondientes (Figura D.15).

```
exon_5_control_brca1=list(filter(lambda secuencia: "exon" in secuencia.description.split(" "),secs_control_brca1_5))


exon_2_control_brca2=list(filter(lambda secuencia: "exon" in secuencia.description.split(" "),secs_control_brca2_2))

exon_11_control_brca2=list(filter(lambda secuencia: "exon" in secuencia.description.split(" "),secs_control_brca2_11))


#suma de diccionarios de los diferentes exones de brca2
exones_control_brca2=exon_2_control_brca2 + exon_11_control_brca2

exon_5_control_tp53=list(filter(lambda secuencia: "exon" in secuencia.description.split(" "),secs_control_tp53_5))

exon_8_control_tp53=list(filter(lambda secuencia: "exon" in secuencia.description.split(" "),secs_control_tp53_8))

#suma de diccionarios de los diferentes exones de tp53
exones_control_tp53=exon_5_control_tp53 + exon_8_control_tp53
```

Figura D.15: Código correspondiente al filtrado de exones para secuencias controles. Fuente: elaboración propia.

Con las listas creadas, se modifica su estructura a un diccionario de pares exón y lista de secuencias asociadas, para tener un mejor manejo y simplicidad. Se debió llamar para conseguir tal propósito a la función `crear_dic` (Figura D.16).

Se tuvo que comprobar a su vez y en último lugar, las longitudes de las secuencias control. Con dicho proceso ya realizado, se almacenarían en unos nuevos archivos junto a las secuencias patológicas ya filtradas.

Como se puede apreciar en la Figura D.17, la eliminación de secuencias y filtrado fue variable, pudiendo tener o no efecto en el número de secuencias originales. Tras este último filtrado fueron almacenadas en el dispositivo, de la misma forma que las secuencias patológicas, con la función `almacenar_sec`.

En este filtrado en comparación con el anterior, se encontró con una gran desventaja que posteriormente condicionó el número de análisis y resultados

```
dic_exon_5_brca1=crear_dic(exones(exon_5_control_brca1))
dic_exones_brca2=crear_dic(exones(exones_control_brca2))
dic_exones_tp53=crear_dic(exones(exones_control_tp53))
dic_exon_5_brca1
```

```
{'exon 5': [Seq('TCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTTAAAAGGTT...ACC'),
  Seq('CTACTGTTGCTGCATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGT...ACC'),
  Seq('ATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTTAAAAGGT...TCA'),
  Seq('ATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTTAAAAGGT...TCA'),
  Seq('ATCTTATTTTTATTTGTTTACATGTCTTTTCTTATTTTAGTGTCCTTAAAAGGT...CCA')]}
```

Figura D.16: Diccionario correspondiente al exón 5 de `BRCA1` para secuencias control. Fuente: elaboración propia.

```
print("Número de secuencias para el primer exon, antes y después del filtrado")
print(len(dic_exon_5_brca1[list(dic_exon_5_brca1.keys())[0]]))
print(len(secuencias_control_brca1_filtradas[list(dic_exon_5_brca1.keys())[0]]))
print()

print("Número de secuencias para el segundo exon, antes y después del filtrado")
print(len(dic_exones_tp53[list(dic_exones_tp53.keys())[1]]))
print(len(secuencias_control_tp53_filtradas[list(dic_exones_tp53.keys())[1]]))
```

```
Número de secuencias para el primer exon, antes y después del filtrado
5
5

Número de secuencias para el segundo exon, antes y después del filtrado
61
14
```

Figura D.17: Código correspondiente al filtrado por longitudes en las secuencias del grupo control. Fuente: elaboración propia.

que se pudo realizar. Es decir, la pérdida de genes y exones de interés dada su escasez de datos; como es el caso, del exón 19 de *BRCA1* y del gen *PIK3CA*, para el cual no se presentaron datos controles algunos de calidad para el estudio, en concreto para ninguno de sus dos exones. De forma que la comparación entre controles y patológicos, para algún caso específico ya se encontró condicionada.

Tratamiento para archivos `csv`

El resto de datos que no fueron obtenidos a partir del *NCBI*, se consiguieron a través de la base de datos de *gnomAD*, con un par de archivos `csv`. Es datos como ya se mencionó en su momento, estaban organizados de forma tabular conteniendo todo tipo de información acerca de las diferentes variantes de los genes oncológicos tratados. Pero solo serían utilizado para un par de ellos, debido al sentido que se presentaban en las secuencias descargadas.

```
nueva_tabla_brca2=tabla_brca2.filter(items=['gnomAD ID','Position','Protein Consequence','Transcript Consequence','VEP Annotatio
```

```
nueva_tabla_brca2
```

	gnomAD ID	Position	Protein Consequence	Transcript Consequence	VEP Annotation	ClinVar Clinical Significance	ClinVar Variation ID	Allele Count	Allele Number	Allele Frequency
0	13-32398767-C-G	32398767	p.Ile3418Met	c.10254C>G	missense_variant	Conflicting interpretations of pathogenicity	630661.0	1	1613610	6.197284e-07
1	13-32398764-T-C	32398764	p.Tyr3417Tyr	c.10251T>C	synonymous_variant	Likely benign	51058.0	4	1613616	2.478905e-06
2	13-32398763-A-G	32398763	p.Tyr3417Cys	c.10250A>G	missense_variant	Conflicting interpretations of pathogenicity	182306.0	3	1613842	1.859918e-06
3	13-32398763-ATATC-A	32398763	p.Ile3418LysfsTer8	c.10253_10256del	frameshift_variant	Likely benign	91745.0	7	1613724	4.337793e-06
4	13-32398762-T-C	32398762	p.Tyr3417His	c.10249T>C	missense_variant	Conflicting interpretations of pathogenicity	409517.0	5	1613786	3.098304e-06
...	...	...	...	...	...	...	...	...	...	...
5992	13-32316466-T-C	32316466	p.Pro2Pro	c.6T>C	synonymous_variant	Likely benign	415659.0	2	1613884	1.239246e-06

Figura D.18: Selección de columnas de interés del *Dataframe* creado. Fuente: elaboración propia.

La descarga de las tablas dio lugar a una serie de procesos de filtrado para su uso y facilidad de manipulación. Dicho proceso comenzó con su cargado en el *notebook* de lenguaje `Python` existente, mediante la función `read_csv` del paquete de `pandas` importado como `pd`.

Posteriormente a ello se generó un `DataFrame` con las columnas o atributos de mayor interés (*gnomAD ID, Position,Protein Consequence, Transcript Consequence, VEP Annotation, ClinVar Clinical Significance, ClinVar Variation ID, Allele Count, Allele Number, Allele Frequency*) con la función `filter` ya nombrada, y se almacenó en una variable global, aquella que en el modelo relacional es considerada como la clave primaria, el id de *gnomAD* inequívoco de las variantes (Figura D.18).

El guardado de la columna fue realizado con el propósito de realizar un segundo filtrado sobre la columna de ids, basándose en esta ocasión en las regiones genómicas de interés contenidas en estos. Regiones, obtenidas a partir del navegador genómico de *UCSC* [UC Santa Cruz, 2024], que se correspondían de manera más o menos precisa a los exones de los cuales ya se poseían sus diferentes variantes en el dispositivo, gracias al filtrado de los archivos `fasta` (Figura D.19).

El filtrado dio lugar a una lista que, en la siguiente función, `filtrar_ids`, tuvo que ser comparada con los ids de la tabla, creando con ello un diccionario filtrado de ids con sus correspondiente información en forma de valores. Además de llegarse a mostrar por pantalla, el significativo cambio del número de variantes contenida antes y después de la llamada a la función.

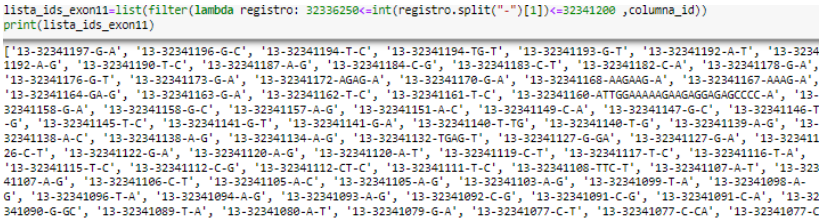


Figura D.19: Lista con los ids de interés dada una región genómica dada. Fuente: elaboración propia.

El desarrollo de dos funciones complementarias, `comprobar_naturaleza` y `comprobar_mutacion`, junto con otra función principal tras el filtrado, `crear_dic_variaciones`; propiciaron que con el diccionario de ids filtrados y con su correspondiente información pasada, se creara un nuevo y último diccionario, que resumiera las diferentes mutaciones presentadas y que caracterizaban a las diferentes variantes entre sí. De esta forma se conseguía una estructura manipulable con gran información de valor, y que sirviera como guía en la determinación de biomarcadores para un posible diagnóstico.

Este proceso de tratamiento se llevó a cabo en dos ocasiones, para las dos tablas de formato `csv` provenientes de *gnomAD*, que contenían las variantes de los exones 11 y 9, de los genes *BRCA2* y *PIK3CA* respectivamente.

Dificultades encontradas

La gran dificultad encontrada en el desarrollo del proyecto fue la gran heterogeneidad presente en los datos obtenidos.

Unos datos de estudio con un alto grado de variabilidad en forma y en número, tanto para todos los genes de interés, como para los pacientes sanos y enfermos. Los cuales dificultarán y reducirán en gran medida la obtención exitosa de resultados y análisis del proyecto, como se observará en futuros apartados. Razones las cuales lo convierten en el punto débil de este proyecto, menguando en gran medida su veracidad y eficacia.

Por otro lado, aunque en menor magnitud, tendríamos el sentido que caracteriza a cada una de las secuencias encontradas en las bases de datos, pudiendo darse dos posibilidades 5'→ 3' o 3'→ 5'. Aunque de forma general se utilice por convenio el sentido 5'→ 3', no en todas las ocasiones es así; y en ocasiones se puede presentar como un inconveniente, como se mostró en la posibilidad de utilizar tablas de variantes mediante *gnomAD*, pero únicamente para aquellas que se encontraran documentadas en sentido 5'→ 3'

. Teniendo, como consecuencia, que recurrir a otras fuentes en determinados casos, para la comparativa de biomarcadores encontrados en las secuencias.

Y es que el proceso de recogida de datos biológicos, aunque cada día vaya siendo más accesible y fácil de tratar, en la actualidad todavía se trata de algo complejo y privatizado en muchos casos, haciéndonos conscientes de que el camino a recorrer no es especialmente corto.

Descripción y Significado de los datos

Las secuencias de nucleótidos consisten en un conjunto ordenado de letras que pueden transportar información, utilizando un alfabeto compuesto por cuatro caracteres [NIH, 2024]. Estas son almacenadas comúnmente en archivos formato `fasta` con un ID y descripción asociados, en sentido 5'→ 3'.

Para el desarrollo de este proyecto se incluyen unos archivos del formato mencionado correspondientes a casos clínicos patológicos, en los que se ha diagnosticado un carcinoma de mama, o que son susceptibles de padecerlo dada su relación con la patología oncológica. En ellos será habitual encontrar diferencias y parecidos, entre sí, y con el resto de los datos, secuencias pertenecientes a individuos sanos.

Son la unidad principal de información de un organismo, utilizadas para prever cómo será la estructura secundaria, terciaria y cuaternaria de las proteínas, unas de las moléculas más importantes del organismo, ya que son energía, necesarias para el correcto funcionamiento de toda célula e involucradas en la formación y reparación del tejido. En conjunto dan lugar a los genes, indicadores de cómo funciona el organismo y reconocidos como las instrucciones de todo ser vivo [Jorde et al., 2020].

Cambios en ellas, denominadas mutaciones, pueden desde provocar el carácter afuncional de una proteína hasta ser benigna para el gen o proteína que codifica, no representando ningún cambio significativo a tener en cuenta. Cuando una secuencia sufre una mutación da lugar a lo que se conoce como variante, una versión distinta de la misma, pero con algún cambio ya sea benigno o maligno como se acaba de mencionar. Muchas de estas variantes tanto benignas como malignas son conocidas, y pueden ser comparadas con las secuencias para detectar un posible riesgo o existencia de carcinoma en el paciente [UCM, 2024].

Las variantes pueden ser recogidas en tablas `csv`, como la conseguida a través de *gnomAD*. Estas tablas presentan para cada variante, diferentes atributos, como la posición y número del cromosoma donde tienen lugar,

consecuencias a nivel de nucleótidos y aminoácidos, su significado clínico asociado y la frecuencia con la que suelen darse, entre muchos otros.

Apéndice E

Anexo de sostenibilización curricular

E.1. Introducción

En 2015 la ONU aprobó la Agenda 2030, que contiene 17 objetivos para el desarrollo sostenible. Estos pretenden ofrecer una posibilidad de que países y sociedades emprendan nuevos caminos que mejoren la vida de las personas de forma justa y equitativa.

La sostenibilidad dentro del campo de la investigación científica es esencial para asentar avances científicos y tecnológicos que beneficien tanto a presentes como en futuras generaciones. El cumplimiento de los ODS (Objetivos de desarrollo sostenible) dentro de este proyecto, garantiza este principio por ello se presenta como una meta prioritaria.

La relación del proyecto bioinformático desarrollado, con varios de los objetivos es significativa. Al pretender mejorar la detección temprana del cáncer de mama, se encuentra fuertemente vinculado con el tercer ODS planteado, Salud y Bienestar. Por otro lado, el uso de avanzadas tecnologías y datos abiertos fomenta la innovación recogida en el noveno ODS, junto con la cooperación internacional del décimo séptimo ODS, logra una reducción de desigualdades explicada en el décimo ODS. Finalmente, aunque en menor medida, la práctica bioinformática y biológica empleada contribuye a la educación de calidad, reflejada en el cuarto ODS [Gamez, 2024].

E.2. Salud y Bienestar. ODS 3

La identificación de biomarcadores permite una identificación temprana y tratamiento eficaz del cáncer, lo que puede ayudar a reducir considerablemente la tasa de mortalidad y mejora de la calidad de vida los pacientes, para este caso, de cáncer de mama. El reconocimiento de mutaciones sobre las diferentes secuencias patológicas y controles, contribuyen a un mayor conocimiento de las posibles variantes malignas y benignas, que se pueden presentar para los genes de *BRCA1*, *BRCA2*, *PIK3CA* y *TP53*.

A su vez, el desarrollo de proyectos como el planteado, actúan como inversión dentro de los sistemas sanitarios, fortaleciendo la capacidad de enfrentar posibles riesgos y reduciendo los gastos de salud en la población.

El desarrollo de tratamientos personalizados y terapias dirigidas, basados en el análisis y conclusiones obtenidas, son de vital importancia para un manejo eficaz del cáncer, lo que colabora a disminuir el número de secuelas [ONU, 2023].

E.3. Industria, Innovación e Infraestructura. ODS 9

El uso de herramientas innovadoras dentro de un campo en crecimiento como es el bioinformático, representa un avance en la infraestructura de investigación biomédica y tecnológica. La aplicación de *Biopython* como técnica de vanguardia, fomenta y explora nuevas alternativas con una realista y prometedora aplicación en las patologías que ponen en riesgo la salud del individuo. Técnicas que pueden ofrecen un valor añadido y mejora a las terapias tradicionales, al brindar enfoques complementarios para un mismo problema [Moran, 2024c].

E.4. Alianzas para lograr objetivos. ODS 17

La unión hace a la fuerza, y la compartición de datos y herramientas de forma abierta y accesible facilita avances e innovaciones en el ámbito científico. Una colaboración internacional y solidaria es fundamental para garantizar la salud a nivel global. Tanto `NCBI` como el propio paquete de *Biopython* y, por ende, el proyecto planteado, ayudan a aumentar el intercambio de conocimientos, creando y evolucionando mecanismos de diagnóstico y tratamiento oncológicos [Moran, 2024a].

E.5. Reducción de desigualdades. ODS 10

La accesibilidad en el proyecto, además de en el resto de los adelantos científicos y sanitarios, promueven la reducción de desigualdades e igualdad de oportunidades para todos los individuos. Esta cualidad es imprescindible en cualquier sociedad moderna, y cobrar especial relevancia en aquellas en desarrollo, donde es fundamental alcanzar la reducción de diferencias y lograr con ello el pleno derecho de cualquier individuo [Moran, 2024d].

E.6. Educación de Calidad. ODS 4

El desarrollo del proyecto puede servir como fuente de inspiración y ayuda para los diferentes investigadores y estudiantes que presenten interés en la temática tratada. Al ser uno de los primeros trabajos realizados sobre este tema, sirve como fuerte componente didáctico, sentado las bases de futuras investigaciones y desarrollos [Moran, 2024b].

Bibliografía

[Anaconda, 2024] Anaconda (2024). Anaconda Navigator 2014; Anaconda documentation — docs.anaconda.com. https://docs.anaconda.com/navigator/. [Accessed 08-07-2024].

[Cedric Notredame and Comparative Bioinformatics Group, 2024] Cedric Notredame and Comparative Bioinformatics Group (2024). T-COFFEE Multiple Sequence Alignment Server — tcoffee.crg.eu. https://tcoffee.crg.eu/. [Accessed 06-07-2024].

[Chang et al., 2024] Chang, J., Chapman, B., Friedberg, I., Hamelryck, T., de Hoon, M., Cock, P., Antao, T., Talevich, E., and Wilczyński, B. (2024). Biopython Tutorial and Cookbook — biopython.org. https://biopython.org/DIST/docs/tutorial/Tutorial. [Accessed 06-07-2024].

[Chen et al., 2022] Chen, Q., Wang, Y., Liu, Y., and Xi, B. (2022). Esrrg, atp4a, and atp4b as diagnostic biomarkers for gastric cancer: A bioinformatic analysis based on machine learning. *Frontiers in Physiology*, 13:905523.

[Dineen, 2016] Dineen, D. (2016). Clustal Omega - fast, accurate, scalable multiple sequence alignment for proteins — clustal.org. http://www.clustal.org/omega/. [Accessed 06-07-2024].

[Edgar, 2021] Edgar, R. C. (2021). Muscle v5 enables improved estimates of phylogenetic tree confidence by ensemble bootstrapping. *bioRxiv*.

[Gamez, 2024] Gamez, M. J. (2024). Portada - Desarrollo sostenible.

[gnomAD Steering Committee, 2024] gnomAD Steering Committee (2024). gnomAD — gnomad.broadinstitute.org. https://gnomad.broadinstitute.org/. [Accessed 06-07-2024].

[HP, 2024] HP (2024). Amazon.es — amazon.es. https://www.amazon.es/HP-15s-fq2039ns-Ordenador-port%C3%A1til-i7-1165G7/dp/B08PFKK64K. [Accessed 06-07-2024].

[Indeed, 2022] Indeed (2022). ¿Cuánto gana un ingeniero biomédico en España? https://es.indeed.com/orientacion-laboral/remuneracion-salarios/cuanto-gana-un-ingeniero-biomedico-espana. [Accessed 07-07-2024].

[Jorde et al., 2020] Jorde, L. B., Carey, J. C., and Bamshad, M. J. (2020). Genética Médica — books.google.es. https://books.google.es/books?hl=es&lr=&id=nh__DwAAQBAJ&oi=fnd&pg=PP1&dq=genetica&ots=v0WNO6eqFk&sig=xJDosDZD4nI7-GLTRCT5qzun8c8#v=onepage&q&f=false. [Accessed 06-07-2024].

[Jupyter Team, 2015] Jupyter Team (2015). Project Jupyter Documentation 2014; Jupyter Documentation 4.1.1 alpha documentation — docs.jupyter.org. https://docs.jupyter.org/en/latest/. [Accessed 06-07-2024].

[Microsoft, 2024] Microsoft (2024). Microsoft 365. https://www.microsoft.com/es-es/microsoft-365/buy/microsoft-365. [Accessed 07-07-2024].

[Microsoft, 2024] Microsoft (2024). Windows 11 Home (USB - Spanish). https://www.microsoft.com/es-es/d/windows-11-home/dg7gmgf0krt0. [Accessed 07-07-2024].

[Ministerio para la Transición Ecológica y el Reto Demográfico, 2024] Ministerio para la Transición Ecológica y el Reto Demográfico (2024). Normativa española en materia de acceso y utilización de recursos genéticos — miteco.gob.es. https://www.miteco.gob.es/es/biodiversidad/temas/recursos-geneticos/normariva_espanola_rg.html. [Accessed 06-07-2024].

[Moran, 2024a] Moran, M. (2024a). Alianzas - desarrollo sostenible. https://www.un.org/sustainabledevelopment/es/globalpartnerships/.

[Moran, 2024b] Moran, M. (2024b). Educación - desarrollo sostenible. https://www.un.org/sustainabledevelopment/es/education/.

[Moran, 2024c] Moran, M. (2024c). Infraestructura - Desarrollo sostenible. https://www.un.org/sustainabledevelopment/es/infrastructure/.

[Moran, 2024d] Moran, M. (2024d). Reducir las desigualdades entre países y dentro de ellos - Desarrollo Sostenible. https://www.un.org/sustainabledevelopment/es/inequality/.

[NCBI, 2024] NCBI (2024). National Center for Biotechnology Information — ncbi.nlm.nih.gov. https://www.ncbi.nlm.nih.gov/. [Accessed 06-07-2024].

[NCI and National Human Genome Research Institute, 2006] NCI and National Human Genome Research Institute (2006). The Cancer Genome Atlas Program (TCGA) — cancer.gov. https://www.cancer.gov/ccg/research/genome-sequencing/tcga. [Accessed 06-07-2024].

[NIH, 2024] NIH (2024). Diccionario de genética del NCI — cancer.gov. https://www.cancer.gov/espanol/publicaciones/diccionarios/diccionario-genetica/def/adn. [Accessed 06-07-2024].

[ONU, 2023] ONU (2023). Salud - Desarrollo Sostenible — un.org. https://www.un.org/sustainabledevelopment/es/health/. [Accessed 06-07-2024].

[Python Software Foundation, 2023] Python Software Foundation (2023). subprocess 2014; Gestión de subprocesos x2014; documentación de Python - 3.10.13 — docs.python.org. https://docs.python.org/es/3.10/library/subprocess.html. [Accessed 08-07-2024].

[Python Software Foundation, 2024] Python Software Foundation (2024). The Python Tutorial — docs.python.org. https://docs.python.org/3/tutorial/index.html. [Accessed 06-07-2024].

[Shtrauss, 2023] Shtrauss, A. (2023). Biopython | Bioinformatics Basics — kaggle.com. https://www.kaggle.com/code/shtrausslearning/biopython-bioinformatics-basics#6-|-MULTIPLE-SEQUENCE-ALIGNMENT. [Accessed 06-07-2024].

[Siegel et al., 2024] Siegel, R. L., Giaquinto, A. N., and Jemal, A. (2024). Cancer statistics, 2024 - PubMed — pubmed.ncbi.nlm.nih.gov. https://pubmed.ncbi.nlm.nih.gov/38230766/. [Accessed 06-07-2024].

[UC Santa Cruz, 2024] UC Santa Cruz (2024). UCSC Genome Browser Home — genome.ucsc.edu. https://genome.ucsc.edu. [Accessed 06-07-2024].

[UCM, 2024] UCM (2024). LA MUTACIÓN. https://www.ucm.es/data/cont/media/www/pag-56185/11-La%20mutaci%C3%B3n.pdf. [Accessed 06-07-2024].