



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

GeneFlow

Diseño de una herramienta para el análisis de
datos de expresión genética

Autor

Alba Casillas Rodríguez

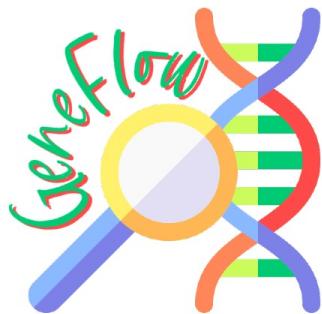
Directores

Fernando Berzal Galiano
Carlos Cano Gutiérrez



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, Septiembre de 2022



GeneFlow

Diseño de una herramienta para el análisis de datos de expresión genética

Autor

Alba Casillas Rodríguez

Directores

Fernando Berzal Galiano
Carlos Cano Gutiérrez

GeneFlow: Diseño de una herramienta para el análisis de datos de expresión genética

Alba Casillas Rodríguez

Palabras clave: Expresión de genes, biblioteca software, genómica, transcriptómica, RNA-Seq, reproducible, workflows, reflexión

Resumen

La investigación de la genética ha desatado una auténtica revolución del conocimiento. Actualmente, se está convirtiendo cada vez más en un campo dominado por el crecimiento del tamaño de los datos y por la necesidad de la comunidad científica de analizar grandes conjuntos de datos de manera eficiente. Para que los avances en la genómica sean posibles, se necesitan expertos con formación multidisciplinar en biología e informática.

El desarrollo de software en Bioinformática requiere la integración de distintas herramientas y paquetes software en flujos de trabajo. Estos flujos de trabajo generalmente se diseñan ad-hoc para cada tarea de análisis específica, requiriendo llamadas a paquetes de software desarrollados en distintos lenguajes de programación y para diferentes plataformas informáticas.

GeneFlow es una biblioteca software escalable y flexible que engloba métodos de obtención de datos, técnicas de preprocesamiento, análisis y visualización de datos, y algoritmos de Aprendizaje Automático que se pueden aplicar sobre datos genómicos. El paquete software representa en un formato semi-estructurado el proceso de trabajo, y recurre a la reflexión para facilitar la realización de experimentos complejos y garantizar su reproducibilidad. GeneFlow está diseñado para permitir que los usuarios finales puedan trabajar con varios conjuntos de datos simultáneamente sin tener que preocuparse por los detalles de implementación de bajo nivel.

GeneFlow también brinda al usuario la capacidad de reproducir un flujo de trabajo completo. El usuario puede reconstruir los objetos en memoria para que los experimentos puedan reproducirse y replicarse según sea necesario, o aplicarlos a diferentes conjuntos de datos.

El código desarrollado para este proyecto se encuentra en: Alba Casillas GitHub.

GeneFlow: Design of a tool for the analysis of gene expression data.

Alba Casillas Rodríguez

Keywords: Gene expression, software library, genomics, transcriptomics, RNA-Seq, reproducible workflows, reflection

Abstract

Studies in genetics have unleashed a complete revolution in knowledge. This field is becoming increasingly dominated by the growth of data volume and by the need for the scientific community to analyze huge datasets in an effective way. For advancements in genomics to be possible, experts with a multidisciplinary background in biology and computing are needed.

Software development in Bioinformatics requires the use of different tools and software packages in workflows. These workflows are typically designed ad-hoc for each specific analysis task, requiring calls to software packages developed in different programming languages and for different computing platforms.

GeneFlow is a scalable and flexible software library that encompasses data acquisition methods, preprocessing techniques, data analytics, data visualization, and Machine Learning algorithms that can be applied to genomic data. Our software package represents the process workflow in a semi-structured format and resorts to reflection in order to facilitate the realization of complex experiments and ensure their reproducibility. GeneFlow is designed to enable end users so that they can work with several datasets simultaneously without having to worry about low-level implementation details.

GeneFlow also provides the user with the ability of replaying a complete workflow. The user can reconstruct the objects in memory so that experiments can be reproduced and replicated as needed or applied to different datasets.

Available code is in: Alba Casillas GitHub.

Yo, **Alba Casillas Rodríguez**, alumna de la titulación **Grado en Ingeniería Informática** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**, con DNI 76738108B, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Alba Casillas Rodríguez

Granada a Septiembre de 2022.

D. **Fernando Berzal Galiano**, Profesor del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **Carlos Cano Gutiérrez**, Profesor del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *GeneFlow: Diseño de una herramienta para el análisis de datos de expresión genética*, ha sido realizado bajo su supervisión por **Alba Casillas Rodríguez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a Septiembre de 2022 .

Los directores:

Fernando Berzal Galiano Carlos Cano Gutiérrez

Agradecimientos

En primer lugar, quiero agradecerle a Fernando y a Carlos, mis profesores, toda la sabiduría, tiempo, y esfuerzo dedicado durante estos meses. Sin sus consejos, este trabajo no habría sido posible.

A mis padres, Alicia y Luis, y a mis abuelos; pues han sido mi pilar fundamental durante todos estos años, aquellos quienes nunca han perdido la fe en mí.

A mi pareja, Tomás, por todo su cariño y paciencia infinita. Por haber estado ahí cuando lo he necesitado, celebrando triunfos y consolándome en los malos momentos.

Y a mi mejor amigo, Josu, por haberme ofrecido siempre un hombro sobre el que apoyarme. Por hacer de esta carrera una etapa más amena y divertida, sobre todo este último año en el que, aunque la distancia nos separa físicamente, siempre le he sentido muy cerca.

Índice general

1. Introducción	12
1.1. Motivación	14
1.2. Estructura de la memoria	15
2. Planificación	16
2.1. Fases	16
2.2. Presupuesto	18
2.2.1. Recursos humanos	18
2.2.2. Recursos hardware	19
2.2.3. Recursos software	19
2.2.4. Costes indirectos y materiales	21
2.2.5. Coste total del proyecto	21
3. Estudio del Problema	22
3.1. Problemática	22
3.2. Contexto biológico	25
3.2.1. La información genética	25
3.2.2. Las ciencias ómicas	27
3.2.3. RNA-Seq	29
3.2.4. El cáncer	31
3.3. Herramientas existentes	33
4. Diseño	35
4.1. Diseño de clases	35
4.1.1. Diagrama de clases - Descarga y lectura de datos . . .	35
4.1.2. Diagrama de clases - Preprocesamiento de datos . . .	36
4.1.3. Diagrama de clases - Aplicación de algoritmos de Ma- chine Learning	38
4.2. Organización y contenido de los módulos	39
4.2.1. GeneFlow	39
4.2.2. Utils	40
4.2.3. Objects	40
4.2.4. Visualize	41

4.2.5. ETL	46
4.2.6. Processing	46
4.2.7. DataObject	49
4.2.8. Model	51
5. Implementación	58
5.1. Herramientas utilizadas	58
5.1.1. IDE - Spyder	58
5.1.2. Google Colaboratory	59
5.1.3. Lenguajes utilizados	59
5.1.4. Bibliotecas utilizadas	60
5.2. Fases del desarrollo	61
5.2.1. Descarga y lectura de datos	61
5.2.2. Preprocesamiento y creación del DataObject	71
5.2.3. Workflow: Creación del flujo de trabajo	74
5.3. Pruebas	75
5.4. Documentación	75
6. Casos Prácticos	77
6.1. Caso nº1: Descarga de datos de GDC.	77
6.2. Caso nº2: Creación de un DataObject.	78
6.3. Caso nº3: Visualización de datos.	79
6.4. Caso nº4: Preprocesamiento de datos.	82
6.5. Caso nº5: Ajuste de hyper-parámetros de un modelo	84
6.6. Caso nº6: Replicando el flujo de datos (Workflow y Reflexión)	86
7. Conclusiones	89
7.1. Lecciones aprendidas	89
7.2. Vías futuras	90
8. Bibliografía	91

Índice de figuras

1.1.	Proyección del almacenamiento anual para el año 2025 [3].	13
2.1.	Diagrama de Gantt estimado.	18
3.1.	Estructura del ADN [8].	25
3.2.	Genotipo humano; en este caso, de una mujer [10].	26
3.3.	Las cinco ciencias ómicas [12].	27
3.4.	Flujo de trabajo para datos de RNA-Seq usando un genoma de referencia. (Fuente: Elaboración propia).	30
3.5.	Visión general de la adquisición, análisis y uso de Machine Learning en la investigación genómica del cáncer [17].	31
4.1.	Diagrama UML que muestra la relación entre las clases que interactúan con la descarga, lectura y carga de datos.	36
4.2.	Diagrama UML que muestra las distintas clases presentes en el preprocesamiento y análisis de datos.	37
4.3.	Diagrama UML que muestra las clases esenciales para el modelado de datos y el uso de algoritmos de Aprendizaje Automático.	38
4.4.	Estructura de carpetas y ficheros.	39
4.5.	Estructura de un fichero proveniente del portal GDC [18].	41
4.6.	Gráfica que muestra la correlación entre varios conjuntos de variables duplicadas.	44
4.7.	Diagrama KDE.	45
4.8.	Diagrama de cajas.	45
4.9.	Curva ROC.	45
4.10.	Curva Precision-Recall.	45
4.11.	Cálculo del valor CPM o RPM.	48
4.12.	Cálculo del valor RPKM.	48
4.13.	Cálculo del valor TPM.	48
4.14.	Paso inicial para el análisis de datos RNA-Seq (Fuente: Elaboración propia).	49
4.15.	Estructura de un objeto DataObject (Fuente: Elaboración propia).	50

4.16. Cálculo de 'Standard Scaler' o 'Z-score'	52
4.17. Cálculo de precision.	53
4.18. Cálculo de recall (sensibilidad).	53
4.19. Cálculo de accuracy.	54
4.20. Cálculo de F1.	54
4.21. Función logit.	55
4.22. Cálculo de la probabilidad $P(Y=x)$, siendo Y la salida de interés y x una muestra específica de X.	55
4.23. Fórmula del Kernel RBF.	56
 5.1. Archivos clínicos de TCGA-ACC.	67
5.2. Estructura de un archivo de RNA-Seq de la plataforma GDC.	69
5.3. Estructura de un archivo de RNA-Seq de la plataforma Legacy GDC.	70
5.4. Página principal de la documentación. Realizada con Read the Docs.	76
5.5. Documentación de los métodos del módulo geneflow. Realizada con Read the Docs.	76
 6.1. Resultado de la descarga y lectura de datos transcriptómicos de GDC.	78
6.2. Visualización de un objeto de tipo DataObject con matriz de conteos y metadatos clínicos.	79
6.3. Resultado: Visualización de un clustermap.	80
6.4. Resultado: Visualización de un MDS según Tipo Celular.	81
6.5. Resultado: Visualización de un gráfico de Volcán.	82
6.6. Resultado: Visualización de la correlación entre variables	83
6.7. Resultado tras eliminar variables con más de un 60 % de correlación.	84
6.8. Resultado: Métricas obtenidas tras un ajuste de hyper-parámetros de Random Forest.	85
6.9. Resultado: Curva Precision-Recall para distintos umbrales.	85
6.10. Objeto DataObject y matriz de conteo iniciales.	86
6.11. DataObject obtenido después de eliminar genes poco expresados.	87
6.12. DataObject obtenido después de aplicar reflexión para reproducir un flujo de trabajo.	88

Índice de tablas

2.1.	Costes asociados a recursos humanos.	18
2.2.	Características del ordenador utilizado.	19
2.3.	Licencias de los recursos empleados.	20
2.4.	Costes asociados a recursos software.	21
2.5.	Presupuesto total del desarrollo del proyecto.	21
3.1.	Comparación entre R y Python.	23
3.2.	Comparación entre R y Python (continuación).	24

Listado de código

5.1.	Creación de un operador de la lista de filtros de la query.	64
5.2.	Creación de un operador de la lista de filtros de la query.	64
5.3.	Ejemplo de descarga de múltiples ficheros mediante la API de GDC.	65
5.4.	Descarga de un fichero por bloques.	65
5.5.	Descarga de datos mediante el uso de hebras.	66
5.6.	Expresión regular para la búsqueda de un archivo clínico desde la API de GDC.	67
5.7.	Método <i>instantiate</i> de la clase Task para instanciar una clase de forma dinámica (uso de reflexión).	71
5.8.	Métodos <i>to_dict()</i> y <i>from_dict()</i> de la clase Task para serializar y permitir la instanciación dinámica del resto de subclases de la jerarquía.	72
5.9.	Métodos <i>get_metadata()</i> de la clase Replace para obtener los parámetros (y tipos) de entrada y salida.	72
5.10.	Método que comprueba si los argumentos de una función coincide con el tipo que hay almacenado en sus metadatos.	73
5.11.	Adición de una Tarea de Proyección de datos al Workflow del objeto DataObject.	74
6.1.	Descarga de datos del cáncer de Melanoma en GDC.	77
6.2.	Lectura de los datos descargados.	78
6.3.	Descarga y lectura de datos clínicos.	79
6.4.	Creación de un objeto DataObject.	79
6.5.	Selección de genes por su varianza y cálculo del clustermap. .	80
6.6.	Creación de la figura de MDS plot.	81
6.7.	Creación del gráfico de Volcán.	82
6.8.	Cálculo de la correlación entre variables.	83
6.9.	Eliminación de variables correladas.	83
6.10.	Creación de la figura de Volcán.	84
6.11.	Creación de la figura de Volcano plot.	85
6.12.	Cálculo de normalización CPM (Counts per million).	86
6.13.	Borrado de genes poco expresados en las muestras.	87
6.14.	Réplica de un flujo de trabajo reconstruyendo los objetos a partir de diccionarios.	88

Capítulo 1

Introducción

“Antes pensábamos que nuestro futuro estaba en las estrellas. Ahora sabemos que está en nuestros genes.”, James Dewey Watson [1]. No fue hasta 1953, con el descubrimiento de la doble hélice de ADN, lo que hizo cambiar por completo la concepción que teníamos de la vida, dado que ya no se podía decir que tenía un origen divino o cósmico, sino que pasaba a tener un origen genético.

El descubrimiento de la estructura y función del ADN modificó el enfoque experimental de la Biología. Todos los organismos están cifrados en un lenguaje de cuatro letras: *A* (*Adenina*), *C* (*Citosina*), *G* (*Guanina*) y *T* (*Timina*). A partir de entonces, la biología se centró en el estudio del ADN, sus propiedades y su estructura.

La investigación de la genética, que ha avanzado más en dos lustros que en el último siglo, ha desatado una auténtica revolución de conocimiento. El ADN, si lo sabemos interpretar, es nuestro manual de instrucciones. En cada célula se encuentra toda nuestra información genética, codificada mediante una secuencia específica, componiendo una larga ristra, cuyos segmentos se conocen como *genes*. El conocimiento y la manipulación del genoma nos permite detectar cambios (mutaciones) en los genes que pueden ocasionar enfermedades y afecciones, hallarles solución, diseñar mejores alimentos, o incluso programar bacterias para fabricar compuestos específicos.

Sin embargo, los sistemas biológicos tienen numerosos detalles y factores que entran en juego. Para tan siquiera comenzar a comprender el genoma humano, se necesita la ayuda de la tecnología, ya que resulta imposible que un humano pueda leer más de 6 millones de pares de bases y encuentre cada mutación. Además, se necesita tecnología adicional para procesar las

enormes cantidades de datos que se producirían al buscar en un genoma, mediante la secuenciación, el mapeo y el análisis de su información. Esto ha llevado a la industria de la genómica al ámbito del **Big Data**.

Para tener una idea general sobre la cantidad de datos, se debe considerar que el genoma humano tiene alrededor de 20.000 y 25.000 genes, lo que lleva a un total de 6 millones de pares de bases por genoma. Esto equivale a 100 Gigabytes de datos. Se estima que un solo cuerpo humano contiene más de 140 millones de PetaBytes de información [2]. No cabe duda alguna de que el estudio de los datos genómicos avanza simultáneamente con la tecnología, la cual se necesita que sea rápida y que funcione de manera eficiente con estos gigantescos volúmenes de datos. Estimaciones prevén que la cantidad de datos genómicos a partir de 2025 generarán hasta 40.000 PB de datos al año.

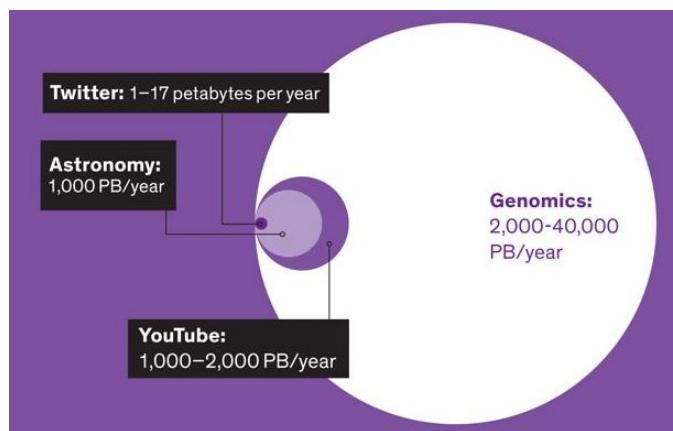


Figura 1.1: Proyección del almacenamiento anual para el año 2025 [3].

No obstante, una tecnología avanzada por sí misma no es suficiente. Para que estos avances sean posibles, se necesitan expertos con un perfil multidisciplinar, una combinación entre la biología y la informática que permita la aplicación de herramientas tecnológicas para responder a problemas biológicos y plantear soluciones.

Desafortunadamente, investigadores con una amplia formación biológica y con capacidad de realizar un análisis elaborado de los datos, experimentan normalmente dificultades en el desarrollo de programas debido a una falta de habilidades en programación. De la misma manera, en la actualidad existe una brecha en el ámbito de la bioinformática, donde una gran parte de los análisis se realizan en **R**, lenguaje de programación dedicado al análisis estadístico y gráfico; pero donde a su vez **Python** ha ganado gran terreno

gracias a la gran cantidad de recursos que proporciona para realizar estudios que involucren técnicas de **Machine** y **Deep Learning**. Esta falta de homogeneidad entre investigadores y desarrolladores, hacen que el desarrollo de software en bioinformática requiera de la integración de distintas herramientas y paquetes software en flujos de trabajo o pipelines de análisis. Estos pipelines habitualmente se desarrollan ad-hoc para cada tarea concreta de análisis y requieren llamadas a programas escritos en distintos lenguajes de programación.

Esta falta de un estándar en el análisis de datos en genómica resalta la importancia de implementar una herramienta que integre funcionalidades ofrecidas por ambos lenguajes. Por tanto, en la Sección 1.1 se justificará la necesidad de la realización de este trabajo.

1.1. Motivación

El objetivo de este proyecto es integrar varias de las funcionalidades que tanto R como Python ofrecen en diferentes APIs en un único **paquete** de análisis que facilite su uso a los analistas y bioinformáticos.

GeneFlow reúne métodos de preprocesamiento, análisis y algoritmos de Aprendizaje Automático que se pueden aplicar sobre datos genómicos. El paquete representa en formato semiestructurado el proceso de trabajo e implementa la capacidad de *reflexión*. Esto facilita al usuario la creación de experimentos complejos utilizando simultáneamente varios conjuntos de datos sin la necesidad de conocer detalles de programación a bajo nivel. Basándose en esta implementación, además, se permite la realización de experimentos replicables en el tiempo y con diferentes conjuntos de datos.

Para acotar el problema al periodo de tiempo establecido para la realización de este trabajo, el proyecto se centra en el uso de datos provenientes de estudios genómicos del **cáncer**, en concreto, datos de **RNA-Seq** (*Secuenciación de ARN*). Sin embargo, su implementación agnóstica y escalable posibilita la adición de módulos que amplíen el ámbito de trabajo fácilmente, sin requerir grandes cambios ni afectar a otros módulos del paquete.

El portal de datos de **GDC**, *Genomic Data Commons*, permite a los investigadores del cáncer y bioinformáticos buscar y descargar datos sobre el cáncer para su análisis. Se ha elegido esta plataforma como principal fuente de datos del proyecto, implementándose las funcionalidades de obtención de los datos pertinentes.

1.2. Estructura de la memoria

Para facilitar el seguimiento del informe al lector, se citan los capítulos de este:

- **Introducción:** Este capítulo introduce brevemente el contexto y la motivación del proyecto, con el objetivo de dar una vista general del propósito del trabajo realizado.
- **Planificación:** Muestra la organización, reparto de tareas y los tiempos estimados del desarrollo del proyecto. Además, se realiza un despliegue del presupuesto asociado al trabajo realizado.
- **Estudio del problema:** En este capítulo se especifica la problemática actual por la cual nace el proyecto. Además, se realiza una introducción al contexto biológico sobre el que se desarrolla principalmente el trabajo; y se hace referencia a proyectos similares ya existentes.
- **Diseño:** Se proporcionan distintos diagramas que reflejan una idea clara del diseño del sistema. A su vez, se explican los componentes de la solución escogida, analizando la arquitectura resultante y mencionando los diferentes recursos que ofrece el paquete.
- **Implementación:** Se explica el código desarrollado, proporcionando detalles técnicos de la implementación de las distintas funcionalidades, y haciendo mención a las características que diferencian a este paquete de otros trabajos.
- **Casos Prácticos:** Este apartado proporciona diversos casos prácticos donde se hace uso de distintas funcionalidades, proporcionando una idea clara de posibles flujos de trabajos que un usuario podrá desarrollar con la biblioteca.
- **Conclusiones:** Para el apartado final, se analizarán las conclusiones obtenidas de este trabajo, además de mencionar posibles vías futuras por las que encaminar el proyecto.

Capítulo 2

Planificación

2.1. Fases

A continuación, se muestra la planificación seguida durante la realización del proyecto, así como el período de tiempo empleado en cada una de las fases.

- **Fase de Investigación (marzo 2022 - abril 2022):** A finales de 2021, se realizaron reuniones previas con los profesores participantes en el proyecto, así como el registro del proyecto en la plataforma *SWAD*.

Se me proporcionaron diversas fuentes y cursos hacia los que enfocar el aprendizaje requerido para el desarrollo del trabajo. Sin embargo, esta fase se llevó a cabo entre los meses de marzo y abril de 2022, tras la finalización del primer cuatrimestre. Se propusieron las siguientes tareas:

- **Estudio de los fundamentos básicos relacionados con la Bioinformática.** Entre los cuales se llevó a cabo un aprendizaje sobre conceptos básicos biológicos y una introducción a la **transcriptómica (análisis de expresión de genes)**, así como la consulta de trabajos relacionados con este ámbito.
- **Estudio de las bases estadísticas en R.** Debido a que normalmente los análisis biológicos se realizan haciendo uso del lenguaje R, se utilizó para realizar un estudio de bases estadísticas útiles para el análisis en bioinformática, como distribuciones, p-values, intervalos de confianza, análisis exploratorio de datos, etc. Además, se realizó un proceso de investigación sobre paquetes en

dicho lenguaje enfocados a los análisis de transcriptómica, como **TCGABiolinks** o **MLSeq**.

- **Investigación de proyectos de análisis de datos ómicos e Inteligencia Artificial.** Se enfocó este proceso de investigación a la aplicación de técnicas, en R o en Python, de análisis exploratorio; así como de algoritmos de **Machine Learning** y **Deep Learning** sobre datos ómicos en relación con el cáncer, debido a la amplia cantidad de datos que se encuentran a disposición de los usuarios sobre esta enfermedad.

Durante esta fase, se realizaron diversas reuniones con ambos profesores en las que se discutió cuál debía ser el mejor enfoque para el proyecto.

- **Fase de Diseño, Desarrollo e Implementación (mayo 2022 - agosto 2022):** Esta fase ha sido la más extensa de todas. Durante este período se fueron diseñando e implementando los distintos módulos del sistema. Estos pueden definirse en tres ámbitos distintos:

- **Obtención de datos:** Durante este período de tiempo, se realizó una investigación en profundidad sobre el funcionamiento del **portal de GDC** y su **API**, a medida que se programaron diversas funcionalidades que realizasen *consultas* a la API para la obtención de datos. Funciones de descarga y lectura tanto de datos de RNA-Seq como clínicos fueron implementadas.
- **Análisis de datos:** El objetivo de esta fase resulta en la rea- lización de numerosas funcionalidades que permitan al usuario aplicar diversos análisis de manera sencilla sobre datos de RNA- Seq.
- **Machine Learning:** Esta fase se enfoca en proporcionar una es- tructura fácil de utilizar que permita a un usuario hacer uso de algoritmos de Machine Learning, dándole la opción de poder ajustar los parámetros del algoritmo y sacar medidas de evaluación fácilmente.

Durante esta fase también se implementaron distintos casos prácticos que muestran diferentes flujos de trabajo que se pueden realizar mediante el uso de la biblioteca. A su vez, también se implementaron sucesivamente baterías de pruebas sobre las funcionalidades progra- madas, para asegurar el buen funcionamiento y la rápida localización de errores.

- **Fase de Documentación:** (agosto 2022): durante la última fase del proyecto, una vez que la estructura del sistema estaba completamente definida, se recopiló toda la información obtenida durante las etapas anteriores del proyecto para ordenarlas correctamente, elaborando el informe final.

En el **diagrama de Gantt** de la Figura 2.1 se puede visualizar el orden y duración de cada una de las fases especificadas anteriormente.

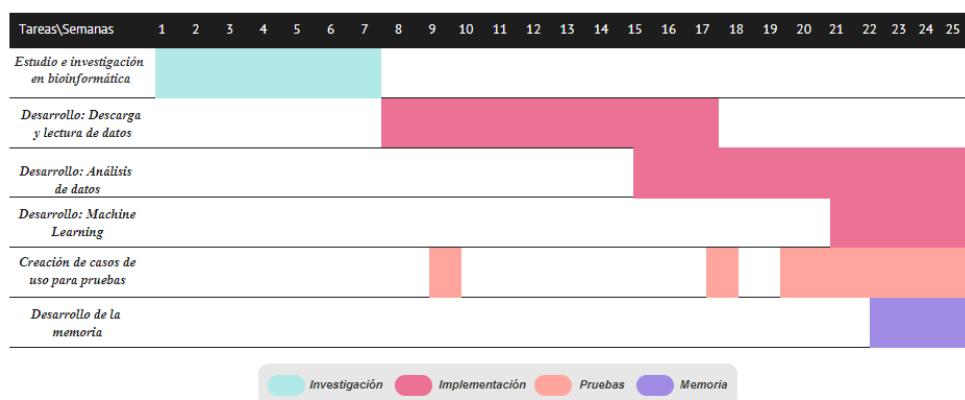


Figura 2.1: Diagrama de Gantt estimado.

2.2. Presupuesto

En este apartado se va a listar el coste de todos los recursos inventariables de hardware y software, así como el invertido en recursos humanos.

2.2.1. Recursos humanos

Para el cálculo de los costes de personal se ha considerado un único empleado, Alba Casillas Rodríguez, Ingeniera de Software. Su coste asociado se muestra en la Tabla 2.1.

Coste asociado (€/h)	Meses trabajados	Días/mes	Horas diarias	Horas (Total)	Coste (€)
30	5	22	6	880	26.400

Cuadro 2.1: Costes asociados a recursos humanos.

2.2.2. Recursos hardware

Durante la realización del proyecto se ha utilizado una *MSI Mag Z390 Tomahawk* montada con un *Intel Core i7-9700K*.

Las características del ordenador se recopilan en el Cuadro 2.2.

Componente	Características
Procesador	Intel Core i7-9700K 3.6GHz (hasta 4.90GHz, 12M de caché, 8 núcleos)
Memoria RAM	Kingston HyperX Fury RGB 16GB DDR4 3200Mhz PC-25600 (2x8GB) CL16
Disco duro principal	500 GB SSD NVMe M.2 3500 MB/s (lectura), 3,300 MB/s (escritura)
Disco duro secundario	1 TB SATA3 7200 rpm
Controlador gráfico	Gigabyte GeForce RTX 2060 Super Windforce OC 8GB GDDR6
Sistema Operativo	Windows 10 Pro
Dimensiones (L X An x Al)	395 mm x 210 mm x 450 mm

Cuadro 2.2: Características del ordenador utilizado.

Los ordenadores suelen tener un periodo de amortización estipulado de unos 4 años. El equipo costó 1986,59 € euros, por lo tanto, se amortizará 496,65 € al año. Teniendo que la duración del proyecto es de 5 meses, el coste final de los recursos materiales asciende a 207 €.

2.2.3. Recursos software

Como recursos software utilizados para el proyecto, se tiene en cuenta la licencia del Sistema Operativo utilizado: *Windows 10 Pro*.

Un apartado a tener en cuenta a la hora de estimar costes de un proyecto, son las licencias asociadas a los recursos empleados. Durante la realización del proyecto, se ha optado por hacer uso de software open-source, con el objetivo de minimizar el coste del proyecto. Es por ello, que las licencias empleadas no han tenido coste alguno.

Sin embargo, se ha elaborado una tabla de recursos software, así como de sus licencias asociadas:

Recurso	Licencia
Python 3.6	Python Software Foundation License (PSFL), compatible con GNU General Public License (GPL).
(IDE) Spyder 5.0	MIT License.
R + RStudio 4.1.2	The “Creative Commons Attribution-ShareAlike International License” version 4.0.
Colab	The 3-Clause BSD License.

Cuadro 2.3: Licencias de los recursos empleados.

Además, para el desarrollo de este proyecto, se realizaron cursos previos formativos para entender conceptos clave sobre el ámbito a tratar. Los cursos realizados fueron:

■ **Programa especializado: Bioinformática**

- **Institución:** Universidad de California en San Diego (UCSan-Diego)
- **Instructor:** *Pavel Pevzner*, Profesor del Departamento de Ciencias de la Computación de UCSanDiego.
- **Instructor:** *Philip Compeau*, Profesor del Departamento de Ciencias de la Computación de UCSanDiego.
- **Coste:** 0 €.

■ **Statistics and R**

- **Institución:** Harvard
- **Instructor:** *Rafael Irizarry*, Profesor de Bioestadística en la Universidad de Hardvard.
- **Instructor:** *Michael Love*, Profesor asistente, del departamento de Bioestadística y Genética.
- **Coste:** 0 €.

El Cuadro 2.4 muestra un resumen de los gastos asociados con relación a los recursos software.

Recursos	Coste asociado (€)
Sistema operativo Windows 10 Pro	200
Licencias	0
Cursos formativos	0

Cuadro 2.4: Costes asociados a recursos software.

2.2.4. Costes indirectos y materiales

A la hora de calcular los costes asociados a un proyecto, es importante tener en cuenta costes indirectos y de material, como la luz, conexión a Internet, material de papelería o transporte. Este coste abarca aproximadamente el 10 % del gasto en personal, estimándolo a 2.640 €.

2.2.5. Coste total del proyecto

Tras analizar los recursos asociados, es posible estimar el coste total asociado al proyecto. El coste total estimado es de 35.630,87€.

Recursos	Coste asociado (€)
Software	200
Hardware	207
Indirectos	2.640
Humanos	26.400
Total	29.447
IVA (21 %)	6.183,87
Importe facturado por el proyecto	35.630,87

Cuadro 2.5: Presupuesto total del desarrollo del proyecto.

Capítulo 3

Estudio del Problema

Para que el lector pueda entender el trabajo desarrollado, en este capítulo se presenta el contexto teórico sobre las distintas temáticas de este proyecto. Inicialmente, se analiza la problemática planteada. Después, se introducen diversos conceptos biológicos que facilitan el entendimiento del trabajo y, por último, se nombran trabajos relacionados que pueden resultar de interés.

3.1. Problemática

Actualmente existen notorias diferencias entre aquellas personas que tienen una mayor formación en Informática y aquellas con un mayor conocimiento biológico. Bioinformáticos y estadísticos que estudian esta área, suelen decantarse por **R**, lenguaje de programación dedicado al análisis estadístico y gráfico. R contiene una amplia gama de pruebas estadísticas, incluyendo pruebas paramétricas y no paramétricas para el testeo de hipótesis, además de numerosos paquetes creados por investigadores expertos. Por otro lado, aquellos quienes tienen un perfil más técnico no suelen tener una amplia experiencia utilizando este lenguaje.

Por otro lado, **Python**, atendiendo al **índice TIOBE** [4], es el lenguaje más utilizado actualmente por los desarrolladores; el 57 % de los científicos de datos y desarrolladores de aprendizaje automático lo utilizan, y el 33 % lo priorizan para el desarrollo. Además, dado que es un lenguaje fácil de usar, con soporte comunitario y corporativo, y con numerosas bibliotecas que lo vuelven bastante eficiente, es el principal lenguaje de programación por excelencia de algoritmos de **Machine Learning** y **Deep Learning**. Su integración con bibliotecas eficientes y escalables obliga a quienes realizan análisis estadísticos en R, a tener que continuar sus experimentos en Python para poder integrarlos con algoritmos de Aprendizaje Automático.

A continuación, los Cuadros 3.1 y 3.2 muestran las diferencias más notorias entre ambos lenguajes.

Característica	Python	R
General	Lenguaje de propósito general para análisis de datos y computación científica.	Entorno de programación funcional para computación estadística y gráfica.
IDE	Ipython, Pycharm, Jupyter Notebook, Spyder	RStudio, R, GUI, RKWARD
Paquetes y librerías	Numpy, Pandas, Matplotlib, Scipy, Scikit-learn, Tensorflow	caret, stringr, ggplot2, knitr, markdown, shiny, forecast, haven, tidyverse
Almacenamiento	Puede tratar con grandes cantidades de datos fácilmente con las bibliotecas adecuadas.	R ejecuta todo en memoria, por lo que su capacidad está limitada por la RAM. No es capaz de manejar cantidades de datos masivas.
Recopilación de datos	Soporta archivos CSV, SQL, JSON y webscrapping con BeautifulSoup	Puede importar archivos CSV incorporando la biblioteca readr. La biblioteca RCurl proporciona una manera simple de realizar peticiones API.
Análisis de datos	Organización de datos con los DataFrames de Pandas. Adopta un enfoque más simplificado para los proyectos de ciencia de datos.	Complejas herramientas de visualización de datos hace el análisis exploratorio de datos (EDA) más complicado.
Visualización de datos	Cuenta con herramientas muy utilizadas como Matplotlib y Seaborn.	Complejas herramientas de visualizaciones de datos.

Cuadro 3.1: Comparación entre R y Python.

Característica	Python	R
API Endpoints	Python ofrece la capacidad de envolver el código como un 'endpoint' de una API para permitir a los usuarios hacer uso de las funciones sin tener que profundizar en el código.	La única manera en R de implementar un esquema de API es usando Plumber API.
Machine Learning	Bibliotecas robustas para la ciencia de datos.	No contiene una gran variedad de proyectos relacionados con este ámbito. Sin embargo, se destaca el paquete MLSeq.
Curva de aprendizaje	Código simple y legible que facilita el aprendizaje.	Su sintaxis funcional resulta complicada para los principiantes.

Cuadro 3.2: Comparación entre R y Python (continuación).

El hecho de tener dos lenguajes de programación con mucha influencia en el estudio de datos genómicos, donde parece que las ventajas de uno son las desventajas del otro; así como una inmensa variedad de paquetes para cada una de las etapas del análisis, dificulta la realización de experimentos a investigadores y bioinformáticos con una base más pobre en programación. Además, obliga al usuario a realizar pasos extra como, por ejemplo, tener que adecuar la estructura de los datos a los argumentos de entrada de una biblioteca en concreto; o a realizar un complejo proceso de instalación de paquetes que permitan mezclar ambos lenguajes en un único trabajo.

Otro gran inconveniente a la hora de realizar proyectos de análisis de datos reside en la necesidad de expandir el proceso a conjuntos de datos diferentes; o a un mismo conjunto actualizado con el paso del tiempo. Esto conlleva un proceso laborioso y una copia o informe del trabajo seguido para poder replicarlo.

Ambos problemas se solventan con el paquete desarrollado para este proyecto, pues GeneFlow homogeniza diferentes APIs para la descarga, lectura, preprocessamiento, análisis y visualización de datos genómicos en un único paquete, y representa de forma semi-estructurada el flujo de trabajo com-

pleto de una manera sencilla de implementar para el usuario que permite, además, reconstruir sus objetos en memoria para replicar los distintos pasos realizados.

3.2. Contexto biológico

3.2.1. La información genética

Las **células** representan el componente básico del cuerpo, existiendo una gran variedad de tipos que realizan funciones diferentes. Estas células forman todos los órganos y tejidos del cuerpo. Prácticamente todas las células del organismo de una persona tienen el mismo **ácido desoxirribonucleico (ADN)**.

El ADN contiene el código para crear y mantener todo organismo, siendo el material hereditario de los seres humanos (y de casi todo el resto de organismos). El código se lee según la secuencia de cuatro bases químicas: la **Adenina (A)**, la **Citosina (C)**, la **Guanina (G)** y la **Timina (T)**. El ADN humano consta de aproximadamente tres mil millones de bases y más del 99 por ciento de esas bases son iguales en todas las personas.

Las bases de ADN se agrupan en pares, A con T y C con G, para formar unidades llamadas **pares de bases**. Cada base está unida a una molécula de azúcar y a una molécula de fosfato. En su conjunto, la base, el azúcar y el fosfato, se denomina **nucleótido**. Los nucleótidos se disponen en dos largas cadenas que forman un espiral denominada **doble hélice**. La estructura de la doble hélice es como una escalera, con las pares de bases que atraviesan el medio como travesaños y las moléculas de azúcar y fosfato en los laterales.

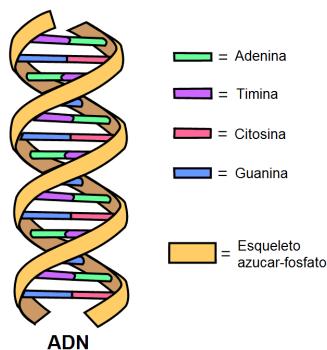


Figura 3.1: Estructura del ADN [8].

Los **genes** son secciones pequeñas de la larga cadena de ADN. Son las unidades básicas funcionales y físicas de la herencia genética. En los seres humanos, el tamaño de los genes varía desde unos pocos cientos a dos millones de bases de ADN. El **Human Genome Project (Proyecto del Genoma Humano)**[6] calcula que los seres humanos tienen entre 20,000 y 25,000 genes. La mayoría de los genes son iguales en todas las personas, pero una pequeña porción de ellos (menos del 1 por ciento del total) varía un poco de una persona a otra. Los **alelos** son formas del mismo gen con alguna pequeña variación en su secuencia de bases de ADN. Estas pequeñas diferencias determinan los rasgos únicos de cada persona.



Figura 3.2: Genotipo humano; en este caso, de una mujer [10].

A partir de los genes, se lleva a cabo la **transcripción**, mecanismo por el cual el ADN se convierte en **ARNm (ARN mensajero)**. Finalmente, la **traducción** permitirá que el ARNm se convierta en una molécula llamada **proteína**. En términos generales, el **ácido ribonucleico (ARN)**, el cual está compuesto por una cadena simple, permite que la información genética del ADN sea comprendida por las células.

Los genes funcionan como instrucciones para la formación proteínas. A veces, la modificación de un gen, conocida como **mutación**, evita que una o más de estas proteínas funcionen correctamente. Esto puede provocar que las células o los órganos modifiquen o pierdan su funcionamiento, lo que puede desencadenar una enfermedad.

3.2.2. Las ciencias ómicas

Durante la década de los 80, el término **ómica** se acuñó para referirse al estudio de un conjunto de moléculas. En las últimas décadas, el avance tecnológico ha permitido el estudio a gran escala de genes, proteínas y metabolitos, permitiendo la creación de distintas ciencias que han ayudado a un mejor entendimiento de la causa de ciertas enfermedades. Las cinco ciencias ómicas más importantes son [11]:

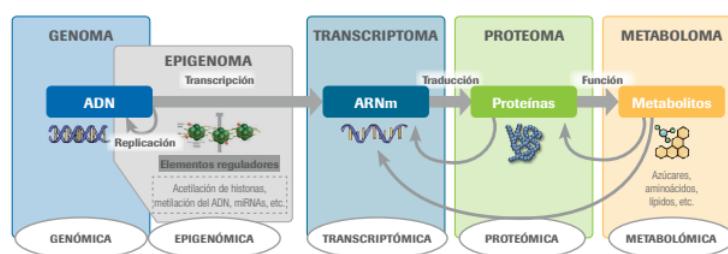


Figura 3.3: Las cinco ciencias ómicas [12].

Genómica

Siendo la primera ómica en crearse, esta ciencia se encarga del estudio del genoma o ADN. Anteriormente, la tecnología permitía estudiar unos pocos genes; sin embargo, con el avance de la tecnología, al principio del siglo XXI se reportó la secuencia del genoma humano [9], la cual reveló que solo un 1 % del genoma es diferente entre humanos (*variantes de un solo nucleótido, SNV*). Estas variantes son frecuentes y confieren susceptibilidad o protección para desarrollar enfermedades.

La genómica avanza con el desarrollo de métodos que les permite obtener secuenciaciones masivas del genoma para poder aislar fragmentos de genes desencadenantes de una enfermedad. Además, el descubrimiento de las variantes de un solo nucleótido originó que se desarrollaran técnicas de genotipificación y estudios de asociación del genoma completo. Estos estudios permiten comparar millones de variantes entre cohortes de casos y controles, resultando en la asociación de algunas variantes con la enfermedad de interés.

Epigenómica

En las últimas décadas se ha demostrado que el ADN puede plegarse formando estructuras tridimensionales que pueden regular regiones muy lejanas. La secuencia de nucleótidos, entonces, no es lo único que regula la

expresión genética, sino el enrollamiento del ADN y su posicionamiento durante la formación de estructuras complejas que construyen los cromosomas.

La epigenética se refiere al conjunto de procesos por medio de los cuales se regula la transcripción de los genes sin afectar la secuencia del ADN, mediante la adición de grupos acetilo (acetilación) de las histonas (proteínas sobre las cuales se enrolls el ADN durante la formación de los cromosomas).

Transcriptómica

El ADN se transcribe a ARNm (también conocido como transcripto). La transcriptómica es la ómica que se encarga de estudiar la expresión de los transcriptos que provienen de diferentes genes. El ARNm es específico de cada célula y de las condiciones fisiopatológicas en un determinado momento, por lo que se hace en un tejido y tiempo específico, dinámicamente.

Anteriormente, se creía que gran parte del ADN que no se transcribía a ARNm, no tenía ninguna función. Sin embargo, además del ARNm existen otros transcriptos no codificantes como los *miRNA* (micro ARN, secuencias de 21-25 nucleótidos) y los *lnc RNA* (ARN largos no codificantes, de secuencias de más de 200 nucleótidos). Actualmente, se conoce que estos transcriptos no codificantes tienen como función regular la expresión de los ARNm codificantes.

Las tecnologías utilizadas para analizar el ARNm son el uso de *micro-arrays* y la *secuenciación del ARN (RNA-Seq)*, permitiendo que la transcriptómica se aplique para el análisis de expresión de genes implicados en diferentes enfermedades como, por ejemplo, el cáncer.

Proteómica

El ARNm es traducido a proteínas (formadas por aminoácidos), las cuales se encargan de realizar la función correspondiente del gen. Una vez que son traducidas, pueden sufrir modificaciones post-traduccionales, tales como cortes, fosforilación, glucosilación, o sumolización. La proteómica estudia a las proteínas, así como las modificaciones post-trasncipciones que las regulan.

La metodología utilizada consiste en separar las proteínas por técnicas cromatográficas (líquidos, gases, etc.) o electroforéticas, digerir las proteínas, detectar sus fragmentos peptídicos y su identificación. Este estudio permi-

te la identificación de proteínas en una muestra biológica, la identificación de un perfil proteico comparando muestras entre casos y controles, y la detección de interacciones entre diversas proteínas o modificaciones post-transcriptionales. De esta forma, se logra encontrar biomarcadores que sirvan para diagnosticar enfermedades, con el fin de dar un tratamiento más adecuado en el futuro.

Metabolómica

Los metabolitos son aquellas moléculas que participan como sustratos, intermediarios o productos en las relaciones químicas del metabolismo. La metabolómica determina los cambios globales en la concentración de los metabolitos presentes en un fluido, tejido u organismo en respuesta a una variación genética, o a un estímulo fisiológico o patológico. Su metodología puede encontrar metabolitos específicos relacionados con el desarrollo de una enfermedad, o con la respuesta a un tratamiento nutricional o farmacológico.

3.2.3. RNA-Seq

Para desarrollar el proyecto en un período de tiempo adecuado, el problema se acota a tratar con datos de RNA-Seq [13].

La secuenciación del ARN (RNA-Seq) consiste en secuenciar todos los transcritos presentes dada unas condiciones específicas, encontrando como consecuencia nuevos transcritos que no se conocían anteriormente y nuevos genes involucrados en una condición clínica.

El RNA-Seq está reemplazando gradualmente a los *microarrays* (los cuales consisten en hibridar el ARNm de un determinado tejido a secuencias de genes previamente conocidas) como método preferido para el análisis del transcriptoma, ya que tiene la ventaja de poder perfilar un transcriptoma completo sin depender de ninguna secuencia genómica conocida, logrando un análisis de expresión de transcripción digital con un rango dinámico potencialmente ilimitado, revelando variaciones de secuencia (variantes de un solo nucleótido) y proporcionando detecciones de expresión génica específicas de alelos o isoformas [14].

La caracterización de la expresión génica en las células a través de la medición de los niveles de ARN con RNA-Seq se emplea con frecuencia para determinar cómo la maquinaria transcripcional de la célula se ve afectada por señales externas (p. ej., tratamiento farmacológico) o cómo las células difieren entre un estado saludable y uno enfermo.

Los niveles de expresión de ARN a menudo se correlacionan con los roles funcionales de sus genes afines. Solo del 1% al 3% de los ARN son codificantes de proteínas, mientras que más del 70% son no codificantes. Sus funciones reguladoras u otras funciones potenciales solo pueden deducirse analizando la presencia y abundancia de sus expresiones de ARN; es por ello que RNA-Seq es comúnmente aplicado para identificar la secuencia, estructura y abundancia de moléculas de ARN en una muestra específica [15].

Debido a que las aplicaciones comunes y novedosas de RNA-Seq crecen a diario y existen incluso más facetas en el análisis de datos de RNA-Seq que en la generación de datos en sí, son numerosos y variados los pasos a seguir en este ámbito. La Figura 3.4 proporciona un flujo de trabajo para datos de RNA-Seq, suponiendo que se tiene un genoma de referencia.

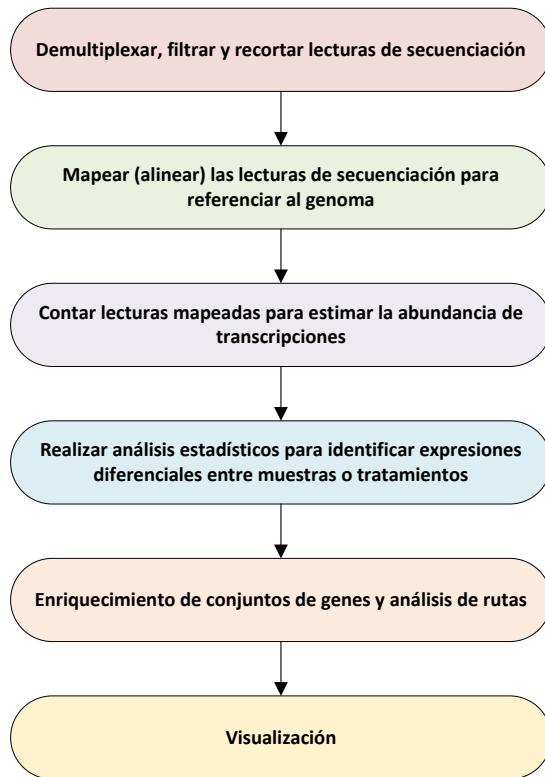


Figura 3.4: Flujo de trabajo para datos de RNA-Seq usando un genoma de referencia. (Fuente: Elaboración propia).

3.2.4. El cáncer

El **cáncer** es una enfermedad genética y una de las principales causas de muerte humana por año. Comprende un sistema biológico enrevesado que requiere un análisis meticuloso y exhaustivo, donde las mutaciones y otras anomalías genómicas y epigenómicas juegan un papel tanto en su inicio como en su progresión. Dada la complejidad del cáncer, en general se cree que hasta la fecha solo se ha identificado una pequeña fracción de las alteraciones que pueden ser útiles como marcadores característicos de tipos de tumores específicos y/o posibles dianas moleculares. Por lo tanto, para tener éxito, los análisis genómicos integrales del cáncer deben superar una amplia gama de desafíos derivados de la complejidad biológica y la heterogeneidad de los tumores y subtipos humanos.

En los últimos años, usando RNA-Seq, la investigación del cáncer está floreciendo día a día, debido a su capacidad de detectar mutaciones tempranas y de alto riesgo molecular, por lo que permite descubrir nuevos biomarcadores de cáncer y posibles objetivos terapéuticos, monitorear la enfermedad, y guiar la terapia dirigida durante los tratamientos iniciales. Además, estos avances en el estudio transcriptómico de la enfermedad, abren un gran abanico de posibilidades para usar técnicas de Machine y Deep Learning para su identificación, clasificación, prevención, tratamiento y seguimiento. [17]

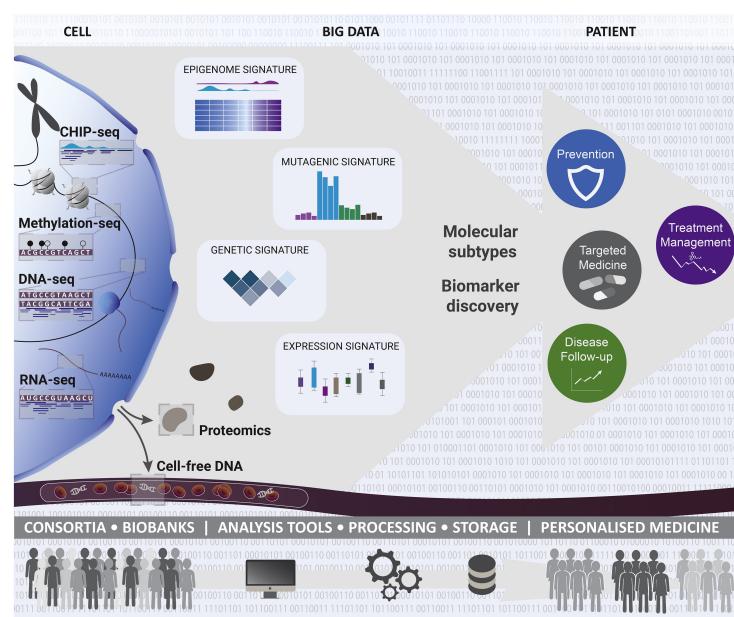


Figura 3.5: Visión general de la adquisición, análisis y uso de Machine Learning en la investigación genómica del cáncer [17].

Por los motivos mencionados anteriormente, así como la inmensa cantidad de datos en Internet a disposición del público, se ha elegido esta enfermedad para acotar este trabajo a un periodo de tiempo razonable. Por ello, la fuente de datos para el proyecto es **Genomic Data Commons (GDC) Data Portal** y, más concretamente, se trabajarán con los datos de **TCGA** (**The Cancer Genome Atlas**).

Genomic Data Commons (GDC) Data Portal

El **Genomic Data Commons (GDC)** del **Instituto Nacional del Cáncer (National Institute of Cancer, NCI)** [18], contiene más de 2,9 PetaBytes (PB) de datos genómicos y clínicos de más de 60 proyectos de investigación genómica del cáncer financiados por el NCI y otros contribuidores.

Más que un simple repositorio de datos, GDC armoniza los datos clínicos y genómicos enviados, brindando a los usuarios un rico recurso para comparar datos y resultados de diferentes proyectos, ateniendo a más de 50.000 usuarios cada mes. Su objetivo principal es hacer que los datos estén ampliamente disponibles para la comunidad y reducir las barreras para analizar conjuntos de datos genómicos complejos. GDC está diseñado para admitir no solo la exploración y la descarga de datos a través de un portal web, sino también para acceder a los datos a parir de una interfaz de programación de aplicaciones (API) por parte de cualquier persona que desee crear una aplicación o sistema utilizando datos del portal.

Su regla de oro es que las correlaciones entre las variantes genómicas en el cáncer con los parámetros clínicos a menudo revelan aspectos importantes de la patogénesis del cáncer y pueden usarse para guiar el desarrollo terapéutico. Por esta razón, GDC incluye un elaborado modelo de datos para una rica anotación de atributos clínicos. Actualmente, GDC pone a disposición de los investigadores datos clínicos y genómicos relacionados con el cáncer de decenas de miles de casos para investigar las asociaciones entre los marcadores genómicos y los pronósticos, o los resultados del tratamiento [19].

A medida que se agreguen proyectos adicionales a la plataforma, el poder estadístico de los datos armonizados crecerá, permitiendo la identificación de genes y tendencias clave que antes no habrían sido posibles en proyectos individuales y esfuerzos más pequeños. GDC es parte de un ecosistema de datos más grande desarrollado por el NCI, que incluye espacios de tra-

bajo computacionales basados en la nube, un repositorio común de datos proteómicos, y uno con datos de imágenes así como otros recursos.

The Cancer Genome Atlas (TCGA)

The Cancer Genome Atlas (TCGA) [20] ha generado, analizado y puesto a disposición datos de variación en número de copias, metilación, expresión y secuencia genómica de más de 11.000 personas que representan más de 30 tipos diferentes de cáncer, siendo uno de los programas de genómica del cáncer más ambiciosos y exitosos hasta la fecha.

TCGA es un esfuerzo de colaboración a gran escala dirigido por el **Instituto Nacional del Cáncer (NCI)** y el **Instituto Nacional de Investigación del Genoma Humano (NHGRI)** para mapear los cambios genómicos y epigenómicos que ocurren en 32 tipos de cáncer humano, incluyendo nueve tumores raros. Su objetivo es apoyar nuevos descubrimientos y acelerar el ritmo de la investigación dirigida a mejorar el diagnóstico, tratamiento y prevención del cáncer. La información generada se administra de forma centralizada y se ingresa en las bases de datos a medida que está disponible, haciendo que los datos sean rápidamente accesibles para toda la comunidad de investigación. En enero de 2015, TCGA ya había generado alrededor de 1,7 PetaBytes (PB) de datos para unos 11.500 casos de tumores y muestras de tejido normal correspondientes. Actualmente, se pueden encontrar 781.76 TeraBytes (TB) disponibles en el portal de GDC.

3.3. Herramientas existentes

R, lenguaje para análisis estadístico por excelencia, tiene numerosos paquetes que resultan útiles para el análisis de datos genómicos; sin embargo, puesto que este trabajo ha sido desarrollado en Python, simplemente se hará referencia al proyecto de R más exitoso en este ámbito.

Bioconductor

Es un proyecto de código abierto y libre desarrollo para el análisis y comprensión de la Genómica. La mayoría de sus componentes se distribuyen como paquetes R, los cuales inicialmente se centraron mayoritariamente en el análisis de microarrays. A medida que el proyecto maduró, el alcance funcional de sus paquetes software se amplió para incluir el análisis de todo tipo de datos genómicos, como SAGE, y secuenciado de datos SNV.

Algunos de sus paquetes más relacionados con el ámbito de este proyecto son:

- **TCGAbiolinks** [22]: Paquete desarrollado para abordar los desafíos con la extracción y análisis de datos genómicos del cáncer almacenados en GDC.
- **TCGAWorkflow** [23]: Analiza datos de genómica y epigenómica del cáncer utilizando distintos paquetes de Bioconductor. Este paquete es creado a partir de la necesidad de crear una herramienta que integre diferentes recursos de otras bibliotecas en una única herramienta. Describe como descargar, procesar y preparar datos de TCGA, y aprovecha varios paquetes clave de Bioconductor para extraer datos genómicos y epigenómicos biológicamente significativos.
- **MLSeq** [24]: Este paquete no solo reúne algoritmos de Aprendizaje Automático de clasificación de uso frecuente, como SVM, bagSVM, Random Forest y CART sino también enfoques novedosos que ponen a disposición del usuario para la clasificación de datos de secuenciación de ARN.

En cuanto a trabajos relacionados desarrollados en Python, cabe destacar:

RNAlyis

RNAlysis [25] es un paquete de análisis modular basado en Python para datos de secuenciación de ARN. Es usado para normalizar, filtrar y visualizar datos; permitiendo el trabajo con matrices de expresión génica y tablas de expresión diferencial, integrándose en particular con HTSeq-count de Python y DESeq2 de R.

Scanpy

Scanpy [26] es un conjunto de herramientas escalable para analizar datos de expresión génica de una sola célula (*scRNA* creado conjuntamente con AnnData). Incluye preprocessamiento, visualización, agrupación, inferencia de trayectoria y pruebas de expresión diferencial.

Capítulo 4

Diseño

4.1. Diseño de clases

A continuación, se muestran los diagramas UML asociados al diseño de este proyecto.

UML son las siglas de *Unified Modeling Language* o *Lenguaje Unificado de Modelado*. Se trata de un estándar que se ha adoptado a nivel internacional por numerosos organismos y empresas para crear esquemas, diagramas y documentación relativa a los desarrollos de software.

Para este trabajo, se presentan varios **diagramas de clases** UML para visualizar la estructura de la biblioteca al modelar sus clases, atributos, operaciones y relaciones principales.

4.1.1. Diagrama de clases - Descarga y lectura de datos

Como se puede observar en la Figura 4.1, *ETL* es la clase que representa la extracción, transformación y carga de la información proveniente de la fuente de datos. *ProcessGDC* es una implementación de dicha clase base, donde se encapsulan todos los aspectos específicos para la integración de datos del portal de GDC.

La descarga de un proyecto de GDC se realiza por medio de un objeto *GDCQuery*, agrupando todos aquellos parámetros por los que se puede filtrar la obtención de un proyecto. Además, los objetos *DataProject* y *FileProject* organizan y estructuran la información de los proyectos a descargar, controlando mejor las funcionalidades de obtención de datos.

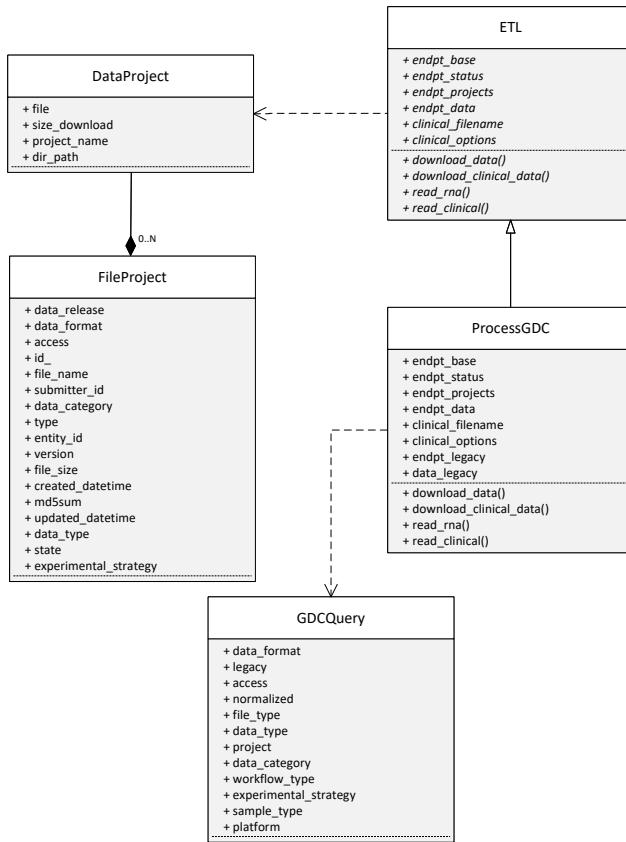


Figura 4.1: Diagrama UML que muestra la relación entre las clases que interactúan con la descarga, lectura y carga de datos.

4.1.2. Diagrama de clases - Preprocesamiento de datos

Task (o Tarea) es la clase base de la que todas las operaciones de análisis y preprocesamiento de datos heredan. Cada subclase tiene en su método *apply()* aquellas acciones a aplicar sobre los datos.

Para facilitar el uso de la reflexión en Python, desde la clase base se instancia dinámicamente el resto de clases a partir de su nombre y un diccionario con los parámetros, e inicializándose con *set_parameters()* y *get_parameters()*.

Para que el diagrama no derive en mucha confusión al incluir un gran número de clases, solo se han añadido algunas de las subclases de primer y segundo nivel. La jerarquía de subclases engloban métodos que comparten funcionalidad; por ejemplo, *QuantAnalysis*, la cual abarca distintos tipos de análisis cuantitativos sobre matrices, es la clase padre de *Normalization* y

GeneAnalysis, las cuales a su vez tienen clases que heredan de ellas para realizar distintos tipos de normalizaciones, o de operaciones que realicen un análisis estadístico sobre unos genes.

Se destacan en el diagrama las clases *Workflow*, *DataTask* y *DataObjectTask*; puesto que son las tres agrupaciones más distintivas en el preprocesamiento. *Workflow* es una composición recursiva de Tareas que permite replicar un flujo de trabajo; mientras que *DataTask* engloba tareas de preprocesamiento sobre distintos conjuntos de datos, y *DataObjectTask* permite extender este preprocesamiento a un objeto **DataObject**: una compleja composición de varios conjuntos de datos.

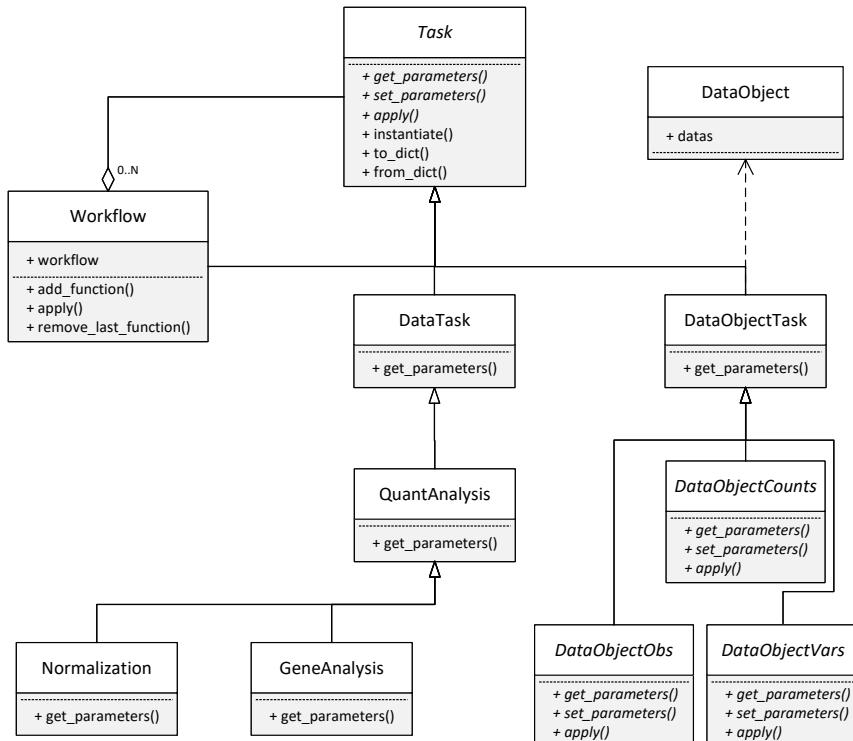


Figura 4.2: Diagrama UML que muestra las distintas clases presentes en el preprocesamiento y análisis de datos.

4.1.3. Diagrama de clases - Aplicación de algoritmos de Machine Learning

El diagrama está compuesto por *ModelSelection*, donde se implementan diversos métodos esenciales para preparar un conjunto de datos antes de ser usado con un modelo de Aprendizaje Automático; y métodos que informan del desempeño de dicho modelo. Por otro lado, *Model* representa el propio modelo de aprendizaje.

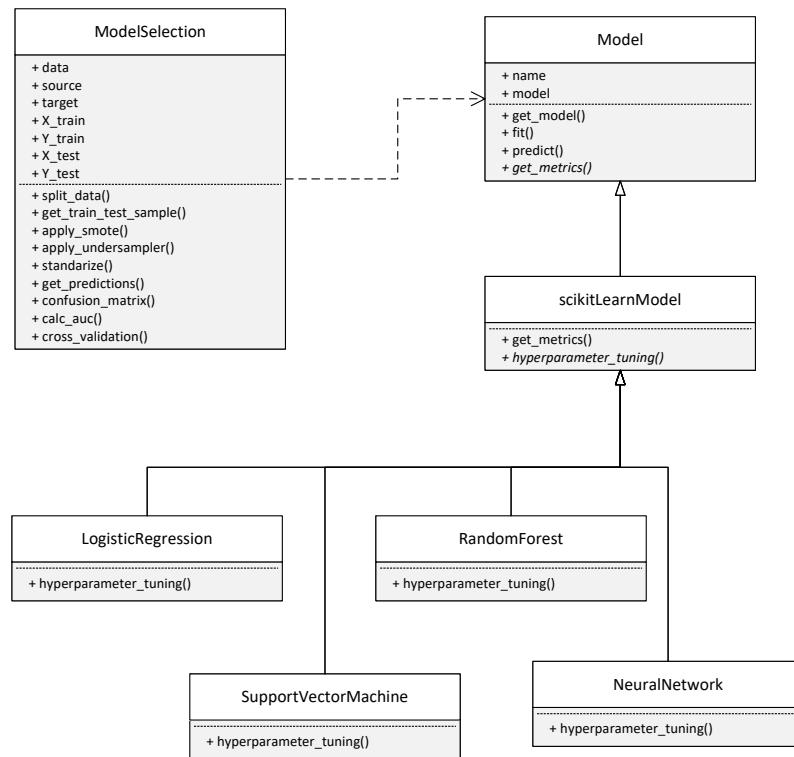


Figura 4.3: Diagrama UML que muestra las clases esenciales para el modelado de datos y el uso de algoritmos de Aprendizaje Automático.

4.2. Organización y contenido de los módulos

Las distintas funcionalidades de la biblioteca se han agrupado en **ocho** módulos, organizando los diferentes métodos según su aplicación.

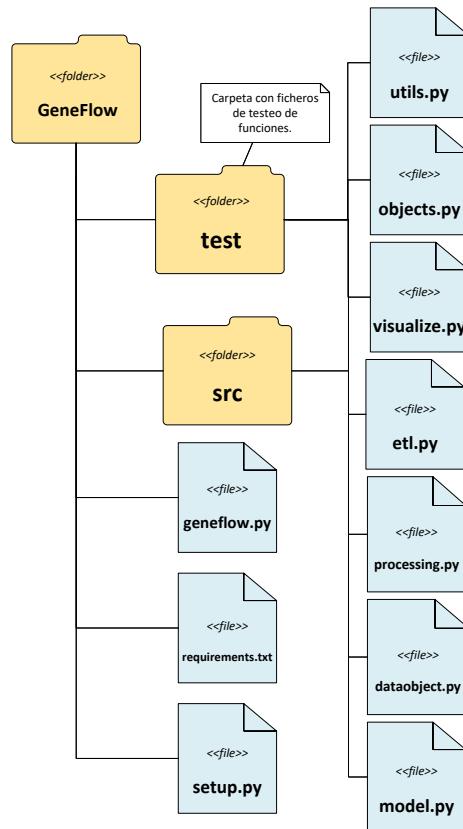


Figura 4.4: Estructura de carpetas y ficheros.

4.2.1. GeneFlow

El fichero **geneflow** engloba todos los módulos desarrollados. De esta forma, permite al usuario hacer uso de su funcionalidad **simplificando la interacción con la biblioteca**.

Con ello, se consigue el diseño de un sistema **modular y fácil de actualizar** que a su vez permite a los usuarios poder realizar **experimentos reproducibles** sin la necesidad de tener en cuenta aspectos técnicos que dificulten el desarrollo de programas de análisis de datos. Mediante el uso

de este módulo, un usuario no necesita tener conocimientos avanzados en detalles de programación para poder utilizar las distintas funcionalidades ofertadas.

4.2.2. Utils

El módulo **utils** almacena funciones de utilidad estándar para resolver problemas comunes, tanto para el usuario final de la biblioteca, como para el propio desarrollo de los demás ficheros.

Entre ellas, se encuentran funciones de creación y manejo de *querys* que permiten interactuar con la fuente de datos deseada para la búsqueda y obtención de información; funciones de chequeo de argumentos, lectura y borrado de archivos. El uso de estas funcionalidades está enfocado a la descarga y lectura de información de las fuentes de datos correctamente.

Funciones dedicadas al parseo de datos entre objetos *DataObject* (explicado en la sección 4.1.7), variables de tipo *dictionary* y estructuras *json* también se añadieron como utilidades.

4.2.3. Objects

Se tomó la filosofía de crear la biblioteca basada mayoritariamente en un **diseño orientado a objetos**. Dado que los objetos representan entidades auto-contenidas y son componentes reutilizables, el módulo *objects* define diferentes **estructuras de datos** mediante objetos con los que interactuar y trabajar en los distintos módulos del sistema.

Las distintas estructuras de datos almacenadas son:

- **GDCQuery**: Estructura de datos que representa aquella información que se puede encontrar en el **portal de GDC**. Los atributos de este objeto permiten la *búsqueda* y *descarga* de los datos de un proyecto mediante la **API** de GDC.
- **FileProject**: Estructura de datos representativa de la información y características del fichero de un proyecto a descargar. Entre sus atributos, se encuentra su ID, formato, extensión, tamaño, o fechas de creación/modificación.

Cómo la información relativa a un fichero depende de la fuente de datos de la que provenga, únicamente es un atributo obligatorio el ID

primario del fichero, de manera que el resto de atributos puedan ser completados o no dependiendo de la información disponible.

Para el caso de la plataforma *GDC*, cada uno de los ficheros a descargar tiene tres identificadores diferentes:

- *UUID*: Identificador único del fichero.
- *submitter_id*: Clave única que identifica el *caso* al que pertenece.
- *entity_id*: También definido como *barcode*, es el identificador principal de los datos de muestras biológicas dentro del proyecto TCGA, que proporciona información del tipo de muestra, tejido o participante en la muestra.

Data Category	Copy Number Variation
Data Type	Copy Number Segment
Experimental Strategy	Gendotyping Array
Platform	Affymetrix SNP 6.0

Entity ID	Entity Type	Sample Type	Case UUID	Annotations
TCGA-CR-A53-01A-11D-A29H-01	aliquot	Primary Tumor	0ac238cc-d348-41d8-8f49-bc0332ed9be4	0

Analysis ID	Workflow Type	Workflow Completion Date	Source Files
0ff1d07-613b-4158-838f-529c72e284dd	DNACopy	2018-09-05	0

Figura 4.5: Estructura de un fichero proveniente del portal GDC [18].

- **DataProject**: Estructura de datos que representa la información relativa a un proyecto. Se compone de una lista de objetos **FileProject** y almacena el tamaño total del proyecto así como el directorio en el que se encuentra (o va a ser) descargado.
- **Workflow**: Un Workflow es una lista de objetos **Log**. Mediante el Workflow, se permite crear un **flujo de trabajo reproducible** en cualquier conjunto de datos en el futuro.

4.2.4. Visualize

El módulo **visualize** engloba una diversidad de funciones que hacen posible la visualización de datos. Dichas funciones se han desarrollado haciendo uso de la biblioteca de python **plotly**. Se ha decidido implementar aquellas

funciones que tienen importancia en el contexto biológico, facilitando la representación de agrupaciones entre muestras y genes; sin embargo, todas y cada una de las funciones son válidas para trabajar con conjuntos de datos de otros ámbitos.

Teniendo en cuenta distintos usos y situaciones en los que se aplica el cálculo de una figura, se le permite al usuario:

- **Visualizar la imagen:** permite la renderización de imágenes en el contexto de un *script* o *notebook*. Por defecto, se ha decidido establecer una renderización a *svg*.
- **Guardar la imagen:** permite guardar la figura como una imagen en disco, con un nombre y extensión determinados. Las extensiones disponibles son: *png*, *jpeg*, *webp*, *svg* y *pdf*.
- **Visualizar la imagen en un navegador web:** muestra una imagen guardada en el *navegador web* por defecto del usuario, siendo permitidos: *Google Chrome*, *Mozilla Firefox*, *Opera* y *Safari*. Si no es capaz de encontrar un navegador disponible, la imagen se mostrará en la aplicación por defecto de visualización de imágenes del usuario.

Las funciones de visualización disponibles pueden agruparse en dos aplicaciones distintas:

Visualizaciones de preprocesamiento de datos:

- **Diagrama de barras:** El diagrama de barras es una forma de representar, en dos dimensiones, la frecuencia (absoluta o relativa) de una variable cuantitativa o cualitativa, pero siempre discreta y distribuida en filas. Los datos se reflejan mediante barras rectangulares de longitud proporcional a los valores representados.
- **Diagrama de cajas:** Es una representación de una variable cuantitativa o cualitativa a través de los *cuartiles*. Mediante este gráfico, se facilita la identificación de valores extremos (*outliers*), debido a que son representados mediante circunferencias fuera del máximo y mínimo. Para este método, también se posibilita la visualización del valor de la *mediana* de los valores.
- **Gráfico de estimación de densidad kernel (KDE, Kernel Density Estimation):** Este método permite la visualización de la distribución de las observaciones de un conjunto de datos. La estimación de la densidad del Kernel extrae los datos a una función de densidad de probabilidad de población estimada.

- **Heatmap:** Un mapa de calor, o heatmap, es una representación bidimensional de datos en la que los valores se representan mediante colores. Los mapas de calor son una visualización muy útil en bioinformática para visualizar relaciones entre *muestras* y *genes*, mostrando la correlación (medida de dependencia) entre las variables. Los valores de correlación oscilan entre -1 y +1.
- **Clustermap:** El clustermap utiliza la *agrupación jerárquica* para ordenar los datos por similitud. Esto reorganiza los datos de las filas y columnas, y muestra contenido similar uno al lado del otro para una comprensión más profunda de los datos.

Para llevar a cabo el clustermap, se calculan previamente los dendogramas de cada eje. Cada *dendograma* realiza un agrupamiento jerárquico de los datos y representado en forma de árbol. Los valores en el eje de profundidad del árbol corresponden a distancias entre grupos.

A continuación, se calcularía una matriz de similitud en la que se aplica la técnica *Z-score* sobre los datos, tratándose de una normalización aplicada por filas. Tras este cálculo, los árboles generados se unen al mapa de calor creado para obtener el clustermap resultante.

- **Gráfico de variables duplicadas:** Dado un conjunto de datos, crea un diagrama de puntos para mostrar la *correlación* entre variables duplicadas dos a dos. Como el número de variables duplicadas no se conoce con antelación, una cuadrícula es calculada dinámicamente para organizar cada una de las figuras de la manera más simétrica y cuadrada posible.
- **Gráfico de Escalamiento Multidimensional (MDS, Multidimensional Scaling):** Es un enfoque para representar gráficamente las relaciones entre objetos en un espacio multidimensional. La reducción de dimensiones a través de MDS se logra tomando el conjunto original de muestras y calculando una *medida de disimilitud (distancia)* para cada comparación de muestras por pares. Luego, las muestras generalmente se representan gráficamente en dos dimensiones, de modo que la distancia entre los puntos en el gráfico se aproxime lo máximo posible a su diferencia multivariante.
- **Gráfico de Volcán (Volcano plot):** Diagrama de dispersión utilizado para identificar rápidamente cambios en grandes conjuntos de datos compuestos por datos repetidos.

Este tipo de gráfico se usa comúnmente para mostrar los resultados de RNA-Seq u otros experimentos ómicos. Un diagrama de volcán es un

tipo de diagrama de dispersión que muestra la *significación estadística (p-value)* frente a la *magnitud del cambio (fold-change)*. Permite la identificación visual rápida de genes con grandes cambios de pliegue que también son estadísticamente significativos. Estos pueden ser los genes más significativos biológicamente. En un diagrama de volcán, los genes más regulados hacia arriba están hacia la derecha, los genes más regulados hacia abajo están hacia la izquierda y los genes estadísticamente más significativos están hacia arriba.

A continuación, se muestran a modo de ejemplo algunas visualizaciones de elaboración propia, realizadas haciendo uso de los métodos descritos previamente:

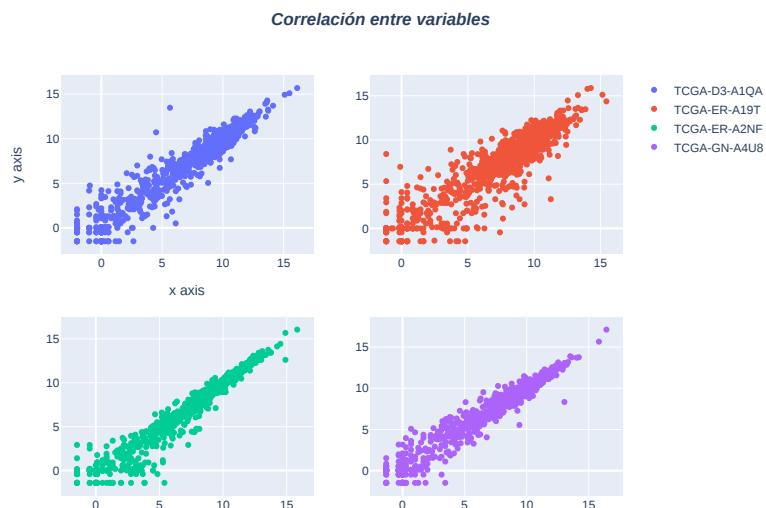


Figura 4.6: Gráfica que muestra la correlación entre varios conjuntos de variables duplicadas.

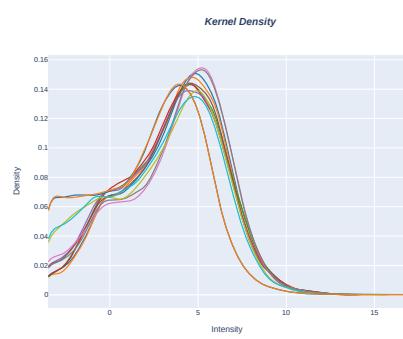


Figura 4.7: Diagrama KDE.

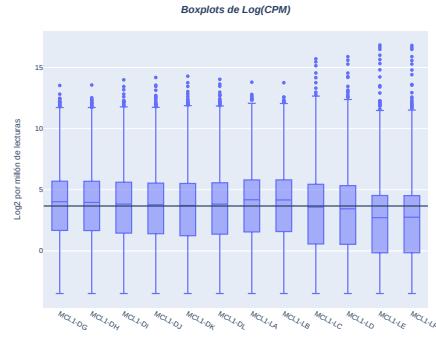


Figura 4.8: Diagrama de cajas.

Visualizaciones para mostrar el desempeño de algoritmos de Machine Learning:

- **Gráfico de matriz de confusión:** Visualización de la matriz de confusión como un *heatmap* que permite evaluar gráficamente la precisión de una clasificación.
- **Curva ROC (Receiver Operative Characteristic):** Se muestra con un gráfico de líneas la curva ROC.
- **Curva PR (Precision-Recall):** Se muestra con un gráfico de líneas los valores de la curva PR.
- **Curva PR (Precision-Recall) con umbrales:** Gráfico de la curva PR (Precision-Recall) teniendo en cuenta los umbrales. Tras calcular la pareja de valores *precision-recall* para diferentes umbrales de probabilidad, dichos umbrales pasan a ser el eje x del gráfico, mientras que los valores precision y recall para cada umbral son visualizados como una traza.

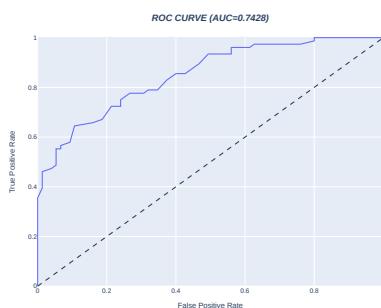


Figura 4.9: Curva ROC.

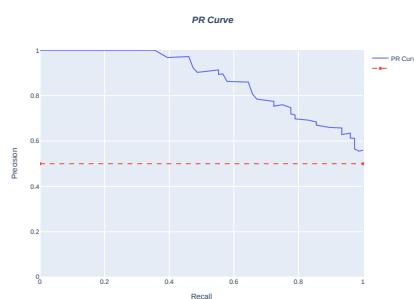


Figura 4.10: Curva Precision-Recall.

4.2.5. ETL

El módulo **etl**, como su nombre indica, se encarga de **Extraer**, **Transformar** y **Cargar** los datos. Este tipo de integración de datos se utiliza para mezclar datos de múltiples fuentes.

Para el trabajo actual, se ha enfocado este módulo en la extracción de información mediante el uso de la API del portal de **GDC: Genomic Data Portal**. Sin embargo, la estructura de clases permite una sencilla adición de cualquier otra fuente de datos a la biblioteca. Toda la extracción, descarga y lectura de datos correspondiente al portal de GDC se encuentra en la clase *ProcessGDC*, manteniendo todos los métodos comunes para llevar a cabo este proceso en una clase padre, *ETL*. De esta forma, cualquier fuente de datos nueva puede ser incluida en la biblioteca simplemente añadiendo una clase hija de *ETL*, con los métodos propios que se adecúen a las peculiaridades de cada una.

La carga de datos permite la creación de *DataFrames* **independientes** de la fuente de datos de donde fueron obtenidos, permitiendo al usuario poder realizar análisis sobre datos de expresión genómica de la misma manera que como se haría con cualquier otro conjunto de datos.

4.2.6. Processing

En GeneFlow, cualquier acción de preprocesamiento que se pueda ejecutar con la biblioteca es, por definición, una **Tarea**.

Por tanto, *Task (Tarea)* es considerada una clase abstracta de la que deriva una jerarquía de clases, cada una de ellas dedicada a una tarea de preprocesamiento de datos y organizadas por funcionalidad mediante subclases intermedias. La clase base hace uso de la reflexión para instanciar dinámicamente invocaciones a sus subclases a partir de su tipo de clase y un diccionario con los parámetros.

En programación, la **reflexión** es la capacidad de encontrar información y de permitir la modificación de objetos en *tiempo de ejecución*. Esto ayuda a los programadores a crear bibliotecas de software genéricas para mostrar y procesar diferentes formatos de datos, así como realizar la serialización o deserialización de datos para la creación de flujos complejos de manera dinámica.

Cada una de las subclases se componen de los métodos `get_parameters()` y `set_parameters()`, donde se van añadiendo los parámetros de la tarea a realizar cuando haga falta, y de un método `apply()` que define el funcionamiento de cada subclase.

Además, cada una de las tareas implementan el método `get_metadata()`, el cual indica al usuario qué parámetros espera como entrada el constructor de la clase y la función a aplicar, y qué se genera como salida; así como **qué tipo de datos son**. Además de servir como utilidad al usuario, este método permite realizar **comprobaciones de tipos** antes de que la ejecución de una tarea falle por haberse instanciado incorrectamente. Gracias a esta implementación resultaría más fácil implementar sistemas complejos de tipo Simulink o KNIME.

Con el módulo **processing**, el usuario tiene a su disposición más de 50 Tareas que le permiten realizar un amplio abanico de análisis computacionales sobre estructuras de datos básicas como DataFrames, listas o diccionarios; contando entre ellas métodos dedicados al **análisis cuantitativo matricial** y específicos al estudio de **datos de RNA-Seq**.

Entre ellos, se destaca el cálculo de correlaciones y traspuestas de datos, y métricas estadísticas como el **p-value** o el **t-statistic**, intersecciones, reemplazo de valores y eliminación de datos; así como diferentes métodos de filtrados de datos y selección y proyección de DataFrames que faciliten la reducción de dimensionalidad de conjuntos de datos que, en el caso de un análisis de datos genómicos, puede tratar con hasta más de un millón de células.

Otro aspecto muy importante a la hora de trabajar con datos ómicos reside en la *normalización* [27], donde los datos sin procesar se ajustan para tener en cuenta los factores que impiden la comparación directa de las medidas de expresión. Los métodos de normalización implementados en este paquete abarcan:

- **Counts per million (CPM)**: También llamado *RPM (Reads per million)* transforma los conteos a valores CPM, que se calculan dividiendo los conteos de cada gen entre los conteos totales de la muestra y multiplicando por 10^6 .

$$\text{CPM} = \frac{\text{Number of reads mapped to gene} \cdot 10^6}{\text{Total number of mapped reads}}$$

Figura 4.11: Cálculo del valor CPM o RPM.

- **Reads per kilo base of transcript per million (RPKM):** El RPKM es una unidad de expresión génica que mide los niveles de expresión de genes o transcripciones. Se trata de una unidad de expresión normalizada de longitud de genes que se utiliza para reconocer qué genes están diferencialmente expresados al compararlos entre diferentes condiciones experimentales. Generalmente, cuanto mayor sea el RPKM de un gen, mayor será su expresión.

$$\text{RPKM} = \frac{\text{Number of reads mapped to gene} \cdot 10^3 \cdot 10^6}{\text{Total number of mapped reads} \cdot \text{Gene length in bp}}$$

Figura 4.12: Cálculo del valor RPKM.

Mientras que el 10^3 de la fórmula se normaliza para la longitud del gen, 10^6 es para el factor de profundidad de secuenciación.

- **Transcripts per million (TPM):** Por definición, TPM y RPKM son proporcionales. Sin embargo, TPM cumple el criterio de promedio invariable. Para una muestra de ARN determinada, si tuviera que secuenciar un millón de transcritos de longitud completa, un valor de TPM representa el número de transcritos que habría visto para un gen determinado. El TPM es un método de normalización muy utilizado para algoritmos computacionales para la cuantificación de transcripciones como RSEM.

$$\text{TPM} = A \cdot \frac{1}{\sum(A)} \cdot 10^6$$

$$\text{where } A = \frac{\text{Total reads mapped to gene} \cdot 10^3}{\text{Gene length in bp}}$$

Figura 4.13: Cálculo del valor TPM.

Workflow es una subclase de Task que a su vez encapsula un conjunto de Tareas permitiéndole formar un **flujo de trabajo**. Como todas las subclases poseen una estructura común, se tratan todas de la misma manera, no solamente se da lugar a poder crear **experimentos reproducibles** sobre otros conjuntos de datos y a lo largo del tiempo sin la necesidad de volver a indicar todos los pasos realizados con anterioridad, sino que también permite la construcción de Workflows más complejos a partir de otros más simples.

4.2.7. DataObject

Aunque se pueda aplicar a numerosos conjuntos de datos, puesto que su estructura no depende del ámbito de la biblioteca, con el módulo **DataObject**, el usuario tiene la posibilidad de crear un objeto compuesto de diferentes estructuras de datos capaces de englobar todos aquellos conjuntos de datos presentes, por ejemplo, en un *análisis de datos de expresión genómica*.

Típicamente, el análisis de **datos RNA-Seq**, comienza con el procesamiento de lecturas (reads), alineadas contra un genoma, que cuantifican el número de secuencias de ARN asociadas a cada posición del genoma. Esta información se dispone en una matriz numérica (**matriz de conteos** o **count matrix**).

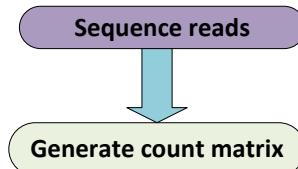


Figura 4.14: Paso inicial para el análisis de datos RNA-Seq (Fuente: Elaboración propia).

El portal de **GDC** permite la obtención de datos RNA-Seq partiendo directamente desde la matriz de conteos. Cuando los datos de un proyecto específico son descargados, se obtiene en cada fichero del proyecto los conteos asociados a los genes para una muestra específica. Una vez unidas todas las muestras en un único conjunto de datos, se obtiene la matriz de conteo. Dicha matriz permite realizar análisis estadísticos y computacionales.

Para facilitar el análisis de experimentos y estudios con múltiples muestras, se define la clase **DataObject**. La matriz de conteos se almacena en un *DataFrame*, el cual tiene *genes* a lo largo de las filas y *muestras* a lo largo de las columnas.

El objeto DataObject también almacena metadatos correspondientes a sus filas y columnas los cuales suelen incluir covariables experimentales, así como información técnica. Por ejemplo, para el análisis de datos de expresión genética, los metadatos asociados a las variables de tipo 'gen' incluyen información como el nombre del gen, su tipo (si es una proteína codificada, un pseudo-gene unitario transrito, un ARN no codificante, etc.), identificadores de exón, o referencias a bases de datos externas; mientras que los metadatos relacionados con las observaciones proporcionan información clínica de cómo fueron obtenidas, en general, englobando distinta información del paciente (región, género, edad de detección del cáncer, tipo de tumor...), del tratamiento que recibe (si está sometido o no a quimioterapia, medicamentos, periodo de tiempo entre tratamientos y sesiones...), de hábitos generales (paciente fumador, historial de alcohol...), etc.

Un objeto DataObject está compuesto por cinco áreas distintas que almacenan información:

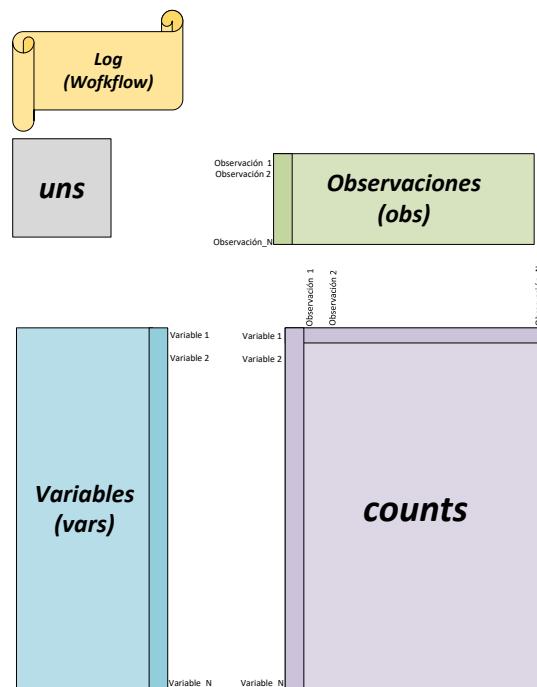


Figura 4.15: Estructura de un objeto DataObject (Fuente: Elaboración propia).

- **Counts:** almacena la matriz de conteo.
- **Variables (Vars):** almacena metadatos de las variables (genes).
- **Observaciones (Obs):** almacena metadatos de las observaciones (muestras).
- **Log:** almacena el flujo de trabajo resultante de realizar operaciones con el DataObject.
- **Uns:** estructura adicional almacenada como un *diccionario* preparada para almacenar información extra que no puede ser organizada en el resto del objeto.

El objeto *DataObject* será actualizado después de cualquier acción de preprocesamiento sobre los datos. Considerando que las filas de *Counts* se corresponden a las filas de *Vars*, y que las columnas de *Counts* son además las filas de *Obs*, cualquier cambio en la estructura de uno de estos tres conjuntos de datos, se verá reflejado a su vez en el resto de conjuntos afectados. De esta manera, se le proporciona al usuario la ventaja de poder realizar análisis más complejos, involucrando varias de las estructuras al mismo tiempo, sin tener que preocuparse de perder o desencajar la información entre los distintos conjuntos de datos, o de tener que actualizar cada uno de ellos manualmente.

4.2.8. Model

Una vez terminado el preprocesamiento de datos, **model** está destinado a aquellos procedimientos partícipes en el ámbito del *Machine Learning*.

El módulo cuenta con una clase *ModelSelection*, la cual almacena aquellos datos que trabajarán directamente con algoritmos de Machine Learning. Además, proporciona al usuario una variedad de funcionalidades para aplicar a dichos datos antes de ser entrenados por los algoritmos, contando con:

- **Partición de datos:** Ante unos datos de entrada, partitiona los datos en los conjuntos *source* y *target*, y permite la obtención de las particiones de los datos de *entrenamiento* y *test* (*xtrain*, *xtest*, *ytrain*, *ytest*).
- **Normalización:** Se calcula una normalización de datos estandarizando sus características, eliminando la media y escalando a varianza unidad. La puntuación estándar de una muestra 'x' se calcula como:

$$z = \frac{(x - u)}{s}$$

Figura 4.16: Cálculo de 'Standard Scaler' o 'Z-score'.

donde u es la media de las muestras de entrenamiento, y s es la media estándar de las muestras de entrenamiento.

- **Tratamiento de datos no balanceados:** Un conjunto de datos está desbalanceado si sus clases no están representadas aproximadamente por igual. Esto es un problema muy presente en el análisis de datos, el cual puede dar lugar a predicciones erróneas. Por tanto, se implementan técnicas de *remuestreo* comúnmente utilizadas en conjuntos de datos que muestran un fuerte desequilibrio entre clases:

- **SMOTE (Synthetic Minority Oversampling Technique):** Técnica de aumento de datos para la clase minoritaria. Procede seleccionando ejemplos que están cerca en el espacio de funciones, dibujando una línea entre los ejemplos en el espacio de funciones y dibujando una nueva muestra en un punto a lo largo de esa línea.

Específicamente, primero se elige un ejemplo aleatorio de la clase minoritaria. Luego, se encuentran k de los vecinos más cercano, seleccionando uno al azar, y se crea un ejemplo sintético en un punto seleccionado al azar entre los dos ejemplos en el espacio de características.

- **RandomUnderSampler:** Implica la selección aleatoria de ejemplos de la clase mayoritaria para eliminarlos del conjunto de datos de entrenamiento. Este proceso se puede repetir hasta que se logre la distribución de clases deseada, como un número igual de ejemplos para cada clase.
- **Combinación de ambas técnicas [5]:** La combinación de sobremuestrear la clase minoritaria y submuestrear la mayoritaria puede lograr un mejor desempeño de los algoritmos (es un espacio ROC) que aplicar solamente una única técnica al conjunto de datos.

Por ello, se hace uso de un *Pipeline* para construir una secuencia que combine ambas técnicas. Primero, se aplica la técnica de sobremuestreo de datos; luego, aplica submuestreo a la salida de la transformación de sobremuestreo antes de obtener el resultado final.

- **Matriz de confusión:** Se calcula la matriz de confusión para visualizar el desempeño de un algoritmo empleado en *aprendizaje supervisado*. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias de la clase real.
- **Cálculo de curvas:** Las curvas **ROC**, (**R**eceiver **O**perating **C**harteristic) y **PR** (**P**recision-recal) son herramientas utilizadas en la evaluación del rendimiento de clasificadores binarios, indicándonos de manera visual la relación entre la precisión y la sensibilidad del modelo.

La **precisión** es el ratio o porcentaje de clasificaciones correctas de nuestro clasificador, es decir, de todo lo que clasifica como positivo, correcta o incorrectamente (TP (*True Positives*) y FP (*False Positives*)), cuál es el porcentaje de clasificaciones correctas.

$$\text{precision} = \frac{TP}{TP + FP}$$

Figura 4.17: Cálculo de precision.

El **recall** o **sensibilidad** de un modelo es el ratio de positivos detectados en el conjunto de datos por el clasificador; es decir, de todos los positivos reales del conjunto de datos, detectados o no (TP (*True Positives*) y FN (*False Negatives*)), cuál es el porcentaje de positivos detectados.

$$\text{recall} = \frac{TP}{TP + FN}$$

Figura 4.18: Cálculo de recall (sensibilidad).

Por tanto, la **curva ROC** relaciona el recall con el ratio de falsos positivos (FP, *False Positives*), es decir, relaciona la sensibilidad de nuestro modelo con los 'fallos optimistas' (clasificar los negativos como positivos). Por otro lado, la **curva PR** es la gráfica resultante entre la precisión y el recall; de manera que permite visualizar a partir de qué recall se obtiene una degradación de la precisión y viceversa.

Se calcula el *área bajo la curva (AUC)* **ROC**, evaluando qué tan bien predice el modelo y resumiendo esta información en una sola métrica;

y el *área bajo la curva (AUC) PR*, calculado como el promedio de las puntuaciones de precisión calculadas para cada umbral de sensibilidad.

- **Validación cruzada:** También conocida como **cross-validation** es una técnica dedicada a validar el rendimiento de un modelo utilizando diferentes fragmentos del conjunto de datos como conjunto de validación; garantizando de esta manera que los datos son independientes de la partición entre datos de entrenamiento y prueba.

Se implementa el método **K-Fold Cross-Validation**, el cual consiste en dividir los datos de forma aleatoria en k grupos de aproximadamente el mismo tamaño, k-1 grupos se emplean para entrenar el modelo y uno de los grupos se emplea como validación. Este proceso se repite k veces utilizando un grupo distinto como validación en cada iteración. El proceso genera k estimaciones del error cuyo promedio se emplea como estimación final.

Las métricas utilizadas para evaluar el rendimiento del modelo de validación cruzada en el conjunto de prueba son: **accuracy**, **precision**, **recall** y **f1-score**.

Siendo **precision** y **recall** definidas en las Figuras 4.17 y 4.18, respectivamente; el valor de **accuracy** mide el porcentaje de casos que el modelo ha acertado.

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Figura 4.19: Cálculo de accuracy.

Mientras que, el valor **f1-score**, combina las medidas de precision y recall en un solo valor.

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Figura 4.20: Cálculo de F1.

A su vez, una clase padre *Model* define el modelo de Machine Learning a utilizar. La aplicación de esta clase permite crear un sistema *scalable* mediante la implementación de modelos proporcionados por distintas bibliotecas a disposición de los desarrolladores en este ámbito. Para el presente

trabajo, se enfatizó el uso de **Scikit-learn**, por lo que se tiene la clase intermedia *ScikitLearnModel*, de la cual nace una clase por cada modelo de Machine Learning propuesto.

Cada modelo utiliza como parámetros de entrada los *hyper-parámetros* correspondientes a cada uno de ellos, pues se instancia utilizando los ya existentes de *scikit-learn*. Sin embargo, se le ofrece la posibilidad al usuario de poder realizar un *ajuste de hyper-parámetros* de un modelo dado mediante una lista de valores por cada argumento con el que se quiera probar. Los modelos disponibles son:

Logistic Regression (LR)

Este tipo de modelo estadístico (también conocido como *logit*) es usado a menudo para la clasificación y el análisis predictivo. La Regresión Logística [29] [28] estima la probabilidad de que ocurra un evento en función de un conjunto de datos determinado de variables independientes. Dado que el resultado es una probabilidad, la variable dependiente está limitada entre 0 y 1.

En esencia, el modelo logístico predice el logit de Y a partir de X. El logit es el logaritmo natural (*ln*) de los *odds* (media de probabilidad relativa que tiene un evento de ocurrir frente a que no ocurra) de Y; y dichas probabilidades son ratios de probabilidad (π) de que Y ocurra o que Y no ocurra ($1 - \pi$). Un modelo simple de regresión logística es de la forma:

$$\text{logit}(Y) = \ln(\text{odds}) = \ln\left(\frac{\pi}{1 - \pi}\right) = \alpha + \beta X$$

Figura 4.21: Función logit.

Tomando el antilogaritmo de la ecuación anterior (Figura 4.13), se deriva una ecuación para predecir de que ocurra el resultado de interés de la siguiente manera.

$$\pi = (Y|x) = \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}}$$

Figura 4.22: Cálculo de la probabilidad $P(Y|x)$, siendo Y la salida de interés y x una muestra específica de X.

Random Forest Classifier (RF)

Un modelo Random Forest [30] [31] es una combinación de predictores de árboles de modo que cada árbol depende de los valores de un vector aleatorio muestreado de forma independiente (*bootstrapping*) y con la misma distribución para todos los árboles del bosque.

La predicción del conjunto se da como la predicción promediada de los clasificadores (árboles) individuales, ya que los árboles de decisión individuales suelen exhibir una gran variación y tienden a sobreajustarse. La introducción de un vector aleatorio en cada árbol también producen errores de predicción que pueden desaparecer al calcular este promedio.

Support Vector Machine (SVM)

La familia de Support Vector Machine [32] [33] son algoritmos de Aprendizaje Automático que funcionan encontrando un *hiperplano* en un espacio N-dimensional (donde N es el número de características) que clasifique claramente los puntos de datos. Para separar las dos clases de puntos de datos, se pueden elegir muchos hiperplanos posibles. Nuestro objetivo es encontrar un plano que tenga el margen máximo, es decir, la distancia máxima entre puntos de datos de ambas clases. Maximizar la distancia del margen proporciona cierto refuerzo para que los puntos de datos futuros se puedan clasificar con más confianza.

Existen ocasiones en las que no es posible encontrar un hiperplano. Esto ocurre cuando se trabaja con grandes dimensiones de datos. Para resolver este problema, Support Vector Machine se apoya en el uso de una función matemática conocida como *Kernel*, la cual se encarga de tomar datos como entrada y transformarlos en la forma requerida. El Kernel más utilizado por este algoritmo es *RBF* (*Radial Basis Function*), que, para dos muestras $x \in R^k$ y x' , representadas como vectores de características en un espacio de entrada, es definido como:

$$K(x, x') = e\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

Figura 4.23: Fórmula del Kernel RBF.

siendo $\|x - x'\|^2$ la distancia Euclídea al cuadrado entre los dos vectores de características y σ un parámetro libre.

Neural Network (NN)

Una Red Neuronal Multicapa [34] [35], es una clase de red neuronal artificial (ANN) totalmente conectada. Consta, como mínimo, de tres capas de nodos: una capa de entrada, una capa oculta, y una capa de salida. A excepción de los nodos de entrada, cada uno de estos es una neurona que utiliza una *función de activación no lineal*.

La función de activación de cada bloque convolucional realiza una operación de no linealidad donde se eliminan aquellos valores considerados no interesantes, siendo esta la forma de aprendizaje de la red neuronal.

Las redes neuronales multicapa utilizan una técnica de aprendizaje supervisado llamada *back-propagation* o *propagación hacia atrás* para el entrenamiento. La propagación hacia atrás es un proceso local. Cada capa obtiene unas entradas y puede calcular el valor de salida y el gradiente local de dicha entrada con respecto las entradas. Sin embargo, una vez que se finaliza el proceso hacia delante, durante el back-propagation, la capa aprenderá eventualmente sobre el gradiente de su valor de salida. La regla de la cadena dice que la capa debe multiplicar dicho gradiente de salida y multiplicarlo en cada gradiente que normalmente calcula para sus entradas.

Capítulo 5

Implementación

5.1. Herramientas utilizadas

En esta sección se detallan las tecnologías, lenguajes, bibliotecas, etc. utilizadas y la aplicación que han tenido dentro del proyecto.

5.1.1. IDE - Spyder

Spyder es un entorno de desarrollo integrado (IDE) multiplataforma, gratuito y de código abierto escrito en Python, para *Python*. Este mismo ha sido diseñado por y para científicos, ingenieros y analistas de datos.

Inicialmente creado y desarrollado por *Pierre Raybaut*, en 2009, Spyder ha sido mantenido y mejorado continuamente desde 2012 por un equipo de desarrolladores científicos de Python y la comunidad.

Spyder se puede usar con complementos propios y de terceros, contando con una combinación única de funcionalidades de edición, análisis y depuración de una herramienta de desarrollo integral; con la exploración de datos, la ejecución interactiva y avanzadas capacidades de visualización de gráficas.

Su disponibilidad es multiplataforma a través de *Anaconda*, en *Windows*; en *macOS*, a través de *Macports*; y en las principales distribuciones de *Linux*, como *Arch Linux*, *Debian*, *Fedora*, *Gentoo Linux*, *openSUSE* y *Ubuntu*.

5.1.2. Google Colaboratory

Google Colab es una herramienta para escribir y ejecutar código *Python* en la nube de *Google*, tratándose de una versión alojada en la nube de *Cuaderno Jupyter*. También hace posible la adición de texto enriquecido, links e imágenes.

Para usar Colab, no se necesita instalar, ejecutar ni actualizar el hardware del ordenador utilizado para cumplir con los requisitos intensivos de carga de trabajo de CPU/GPU de Python. Además, brinda acceso gratuito a la infraestructura informática como almacenamiento, memoria, capacidad de procesamiento, unidades de procesamiento de gráficos (GPU), y unidades de procesamiento de tensor (TPU).

Aunque el código desarrollado para el proyecto se realizó en el IDE Spyder, se hizo uso de esta herramienta como opción para la creación de flujos de trabajo haciendo uso de las distintas funcionalidades implementadas.

5.1.3. Lenguajes utilizados

Python

Python es un lenguaje de programación de alto nivel ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el Machine Learning (ML). A diferencia de otros lenguajes como Java o .NET, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en Python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado intérprete, por lo que no es necesario 'traducirlo' a lenguaje máquina.

Los desarrolladores utilizan Python porque es eficiente y fácil de leer y aprender, además de que se puede ejecutar en muchas plataformas diferentes. El software Python se puede descargar gratis, se integra bien a todos los tipos de sistemas y aumenta la velocidad del desarrollo.

R

R es un lenguaje de programación interpretado, es decir, ejecuta las instrucciones directamente, sin una previa compilación del programa a instrucciones en lenguaje máquina. Es comúnmente utilizado para la computación estadística y gráfica, ya que dispone de una amplia variedad de técnicas

estadísticas (modelos lineales y no lineales, pruebas estadísticas clásicas, análisis de series de tiempo, clasificación, agrupamiento, etc.) y gráficas.

Aunque no es un lenguaje que se haya utilizado directamente para realizar el proyecto, se ha utilizado para poder replicar análisis u operaciones a la misma vez que se desarrollaban en Python para, de esta forma, comprobar que se obtenían los mismos resultados que al utilizar los métodos que ofrece R. Además, por la eficacia de usar R en ciertas ocasiones, se ha llamado directamente a implementaciones de este lenguaje, por lo que ha sido necesario interpretar sus resultados en formato Python.

5.1.4. Bibliotecas utilizadas

En este apartado se comentará brevemente acerca de las bibliotecas importadas para el desarrollo de las funcionalidades del sistema.

- **requests** [36]: Biblioteca HTTP simple utilizada para implementar las peticiones de descarga de datos a la API de la fuente de datos.
- **pandas** [37]: Paquete de Python que proporciona estructuras de datos rápidas y flexibles diseñadas para que trabajar con datos 'relacionales' o 'etiquetados' sea fácil e intuitivo. Su objetivo es ser el bloque de construcción fundamental de alto nivel para realizar análisis prácticos de datos del mundo real en Python.
- **threading** [38]: Módulo para construir interfaces de alto-nivel haciendo uso de hebras. Este paquete es utilizado para aumentar la eficiencia de la descarga y lectura de datos de proyectos.
- **plotly** [39]: Librería disponible en varios lenguajes de programación como Python, Java, Julia, Matlab, etc. que permite la creación de gráficos complejos y variados.
- **scikit-learn** [40]: Paquete que proporciona herramientas sencillas y eficientes para el análisis de datos predictivo. Es un paquete de código abierto y reusable en numerosos conceptos.
scikit-learn implementa variados algoritmos de clasificación, regresión, clustering y diversas herramientas de preprocesamiento, comparación, validación y reducción de la dimensionalidad de los datos.
- **imblearn** [41]: Biblioteca de código abierto basada en scikit-learn que proporciona herramientas cuando se trata de la clasificación con clases desequilibradas.

- **rpy2** [42]: Interfaz de Python para el lenguaje R. Este paquete ejecuta R incrustado, proporcionando acceso a él desde Python utilizando la C-API de R a través de una interfaz de alto nivel que hace que las funciones de R sean objetos como las funciones de Python y proporciona una conversión perfecta a estructuras de datos numpy y pandas.

5.2. Fases del desarrollo

En este apartado se detallarán las fases más importantes del desarrollo de la biblioteca, analizando aspectos técnicos, desafíos encontrados, y soluciones propuestas.

5.2.1. Descarga y lectura de datos

Para poder realizar análisis de expresión genómica, primero se debe partir de unos datos. La plataforma **GDC (GDC Data Portal)** permite a investigadores y bioinformáticos buscar y descargar datos sobre el cáncer para su análisis.

GeneFlow ofrece funcionalidades tanto para la descarga como lectura provenientes de esta plataforma, sin necesidad de hacer una búsqueda y descarga manual de los datos, ni de realizar costosas llamadas a la *API* de GDC.

Descarga de proyectos

Para descargar datos relacionados a un proyecto de GDC, se hizo uso de un objeto **GDCQuery**, el cual almacena los distintos parámetros con los que se puede acceder a la información de un tipo de cáncer en específico. Dichos atributos son:

- **project**: Hace referencia al nombre del proyecto. Este puede ser identificado mediante el nombre del cáncer que se encuentra registrado en la plataforma o su identificador. Por ejemplo, *Adrenocortical Carcinoma* es el nombre del cáncer cuyo identificador es *TCGA-ACC*.

Además, teniendo en cuenta de la complejidad de los nombres que pueden llevar a errores de escritura, este parámetro no distingue entre mayúsculas y minúsculas.

- **experimental strategy**: Define la estrategia experimental utilizada para la caracterización molecular del cáncer. Entre ellas, se puede encontrar *Genotyping Array*, *RNA-Seq*, *Methylation Array*, *WXS*,

scRNA-Seq, etc. Para el desarrollo del proyecto este parámetro, junto al nombre del proyecto, es obligatorio.

El objetivo de este proyecto es el diseño e implementación de una biblioteca completamente escalable y flexible, que pueda incorporar distintos módulos software en una única API integrada. Sin embargo, para acotar el desarrollo del proyecto en un periodo de tiempo razonable, se ha decidido enfocar la implementación en una primera fase al análisis de datos de tipo **RNA-Seq**. De este modo, las funciones de lectura y análisis de datos se han centrado en este tipo de datos, aunque las funciones de descarga permiten obtener datos de otros tipos experimentales.

- **data category:** Define la categoría de archivos a alto nivel. Algunas de las opciones que encontramos en la plataforma son: *sequencing reads, copy number variation, transcriptome profiling, clinical*, etc.
- **data type:** Hace referencia al tipo de archivo de datos. Este parámetro es más granular que data category, ofreciendo una gran variedad de opciones, entre ellas: *copy number segment, gene expression quantification, miRNA expression quantification, single cell analysis*, etc.
- **workflow type:** Especifica el flujo de trabajo bioinformático utilizado para generar o armonizar el archivo de datos.

La **armonización** de los datos se define como una propiedad o característica física, química, biológica o microbiológica que debe estar dentro de un límite, rango o distribución apropiados para garantizar la calidad de los datos.

Algunas de las opciones ofertadas por GDC son: *DNAcopy, GCGSC miRNA Profiling, STAR -counts, CellRanfer - 10x Filtered Counts*, etc. Estableciendo este parámetro a *STAR -counts*, se obtienen los conteos de las variables.

- **platform:** Este parámetro indica la plataforma tecnológica sobre la que se produjeron los datos experimentales, entre las cuales se encuentran: *illumina, affymetrix SNP 6.0, rppa, illumina human methylation 450*, etc.
- **data format:** Indica el formato del fichero, es decir, *tsv, bam, txt...*
- **sample type:** Describe el origen de una muestra biológica utilizada para una prueba de laboratorio, como *Primary Tumor, Solid Tissue Normal*, etc.
- **legacy:** Hace referencia a la base de datos que se va a utilizar. Siendo este atributo un valor booleano, *True* significa que se utilizará la **GDC**

Legacy Archive. Esta es una copia sin modificar de GDC, donde los datos ya no se mantienen ni armonizan activamente.

- **normalized:** Este booleano solo se tiene en cuenta si el parámetro legacy es *True*, de manera que indica el *tipo de archivo*: con datos normalizados o sin normalizar.

Por defecto, se ha establecido el tipo de acceso a *open*, ya que no se puede descargar aquellos archivos que tengan un tipo de acceso *controlled*.

Para descargar los datos de un proyecto, se debe indicar una localización existente para almacenarlos en disco y los archivos se descargarán únicamente si el proyecto ya no se encuentra descargado en dicha ruta.

Tras comprobar que el proyecto existe, se realiza una **petición a la API de GDC**. Esta API es la interfaz REST externa para GDC, y es la que controla el portal web del programa para hacer la información más accesible a usuarios. La API permite buscar, ver, enviar y descargar subconjuntos de archivos de datos, metadatos y anotaciones en función de parámetros específicos.

La query que realiza la descarga de datos estará filtrada por los parámetros indicados por el usuario previamente. Un aspecto a tener en cuenta, es que la selección de un parámetro **puede inhabilitar la existencia de muestras de otros parámetros**, es decir, es posible que aunque de primeras se encuentren archivos con un *data type* de *miRNA expression quantification* para un cáncer determinado, no quiere decir que si se quiere filtrar por un *Data Category* de *Copy number variation*, se vaya a encontrar *data types* de esa categoría.

Por ello, se comprueba la existencia de cada uno de los parámetros seleccionados por el usuario, comenzando desde aquellos más abarcadores hasta los menos específicos; por lo que el filtro de la *request* va actualizándose en función de los parámetros definidos.

La adición de un parámetro al filtro se realiza mediante la creación de un operador *op-in*, donde *key* es el parámetro a añadir y *value*, su valor. Los filtros se actualizan añadiendo sucesivamente cada operador.

```

1     opin = {"op" : "in",
2             "content" : [
3                 {"field" : "key",
4                  "value" : "value"
5                 }
6             }
7
8     filters ["content"].append(opin)

```

Code Listing 5.1: Creación de un operador de la lista de filtros de la query.

Una vez comprobado que todos los valores introducidos por el usuario son correctos y existen datos que cumplen todas las características, se crean los parámetros de la query, añadiendo los filtros en formato *json*, facetas para ajustar los resultados de búsqueda (en nuestro caso, para obtener el tamaño de la query, una especificación de los campos del objeto devuelto, y el tamaño de la petición).

Tras esto, se procede a hacer la *peticIÓN* a la API. El contenido de la respuesta, es decir, la información de cada uno de los ficheros que cumplen todas las restricciones, son almacenados en formato diccionario y añadidos a la clase *FileProject* para ser posteriormente descargados.

```

1     params = {
2         "filters": ut.json.dumps(filters),
3         "facets": "file_size", # to get file size of the query
4         "fields": lfields,
5         "size": "900000",
6         "pretty": "true"
7     }
8
9     endpt = "https://api.gdc.cancer.gov/files"
10
11    response = ut.requests.get(endpt, params=params, timeout =
12                                600)
13    response = ProcessGDC.get_query(query_object)
14    content = response.content
15
16    data = ut.json.loads(content)

```

Code Listing 5.2: Creación de un operador de la lista de filtros de la query.

Atendiendo a la documentación de GDC, múltiples ficheros pueden ser descargados al especificar una lista de sus UUIDs:

```

1 data_endpt = "https://api.gdc.cancer.gov/data"
2
3 ids = [
4     "b658d635-258a-4f6f-8377-767a43771fe4",
5     "3968213d-b293-4b3d-8033-5b5a0ca07b6c"
6 ]
7
8 params = {"ids": ids}
9
10 response = requests.post(data_endpt,
11                           data=json.dumps(params),
12                           headers={
13                               "Content-Type": "application/json"
14                           })
15
16 response_head_cd = response.headers["Content-Disposition"]
17 file_name = re.findall("filename=(.+)", response_head_cd)[0]
18
19 with open(file_name, "wb") as output_file:
20     output_file.write(response.content)

```

Code Listing 5.3: Ejemplo de descarga de múltiples ficheros mediante la API de GDC.

Dependiendo del cáncer y características seleccionadas, para algunos proyectos se necesitarán descargar varios megas (MBs) de archivos; para incrementar la velocidad de descarga, se tomó la decisión de utilizar **hebras**, haciendo uso de la biblioteca **threading**, de python.

Esta decisión obligó a modificar la estructura de la descarga de ficheros, puesto que el contenido ya no podía simplemente escribirse en el fichero de salida, sino que debía ser escrito en bloques de, en este caso, 1024 Bytes.

```

1 def download_file(respFile, path_file):
2
3     with open(path_file, "wb") as output_file:
4
5         for block in respFile.iter_content(1024):
6
7             output_file.write(block)

```

Code Listing 5.4: Descarga de un fichero por bloques.

```
1 from threading import Thread
2
3 def thread_download_file(respFile, path_file):
4     Thread(target=download_file,
5            args=(respFile, path_file)).start()
6
7 endpt_data = "https://api.gdc.cancer.gov/data"
8
9 for f in fproj.get_files():
10    path_fl = os.path + "/" + f.get_id()
11
12    resp = ut.requests.post(ETL.get_endpt_data(),
13                            data=ut.json.dumps({"ids" : f.get_id()}),
14                            headers={"Content-Type" : "application/json"},
15                            stream=True)
16
17    thread_download_file(resp, path_fl)
```

Code Listing 5.5: Descarga de datos mediante el uso de hebras.

Al final del proceso, el usuario podrá encontrar, en la ruta que indicó como destino, una carpeta con el nombre del proyecto, donde se almacenan cada uno de los ficheros obtenidos desde la API. Además, asociado a cada proyecto, se genera un archivo *manifest.txt*, el cual contiene una lista de todos los identificadores de los ficheros descargados y su *entity_id* asociado; ya que este valor es de gran importancia a la hora de cargar la matriz de conteos y alinear las muestras con su metadata.

Descarga de datos clínicos

La descarga de los datos clínicos ocurre de manera similar al proceso descrito anteriormente. Para que se produzca la búsqueda y descarga de información, el usuario debe indicar tanto el nombre del proyecto, como el tipo de dato clínico a especificar.

En GDC, se encuentran los siguientes tipos de datos clínicos disponibles: **drug**, **patient**, **nte** (evolución de nuevos tumores), **radiation**, **follow_up** (información de seguimiento), y **omf** (diferentes tipos de información biomédica en relación a estados y tratamientos del paciente).

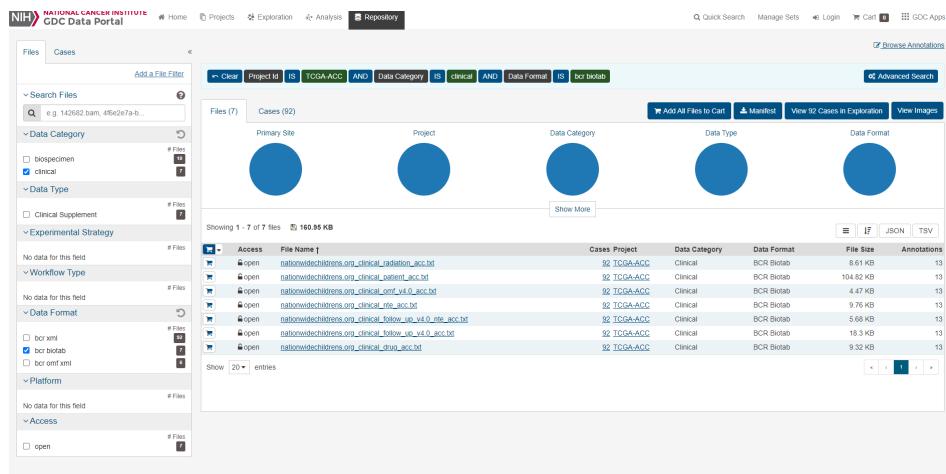


Figura 5.1: Archivos clínicos de TCGA-ACC.

Una vez comprobado que la opción clínica elegida por el usuario es correcta, se realiza una *request* a la API para obtener la información. Para que se encuentre el archivo, este debe de ser buscado con la estructura correcta de la plataforma; además, se debe de tener en cuenta que puede haber versiones actualizadas de un fichero (indicadas por un '*v4.0*', por ejemplo). Por ello, se crea una **expresión regular** que se encargue de montar la estructura correcta del fichero antes de buscarlo:

```

1 def regex_file_name(proj_id, clinical_option, file_name):
2
3     clinical_filename = "nationwidechildrens.org_clinical_"
4
5     middle_nm = clinical_filename + clinical_option
6
7     pattern = middle_nm + "(_v[0-9]{1,3}.[0-9]{1})?" + "_" +
8         proj_id + ".(txt|csv|tsv)$"
9
10    return re.match(pattern, file_name.lower()) # True or False
11
12 low_proj = project_id.split("-")[1].lower()
13
14 file_search = list(filter(
15     lambda fl: ProcessGDC.regex_file_name(low_proj, clinic_info,
16         fl['file_name']) == True, datos))

```

Code Listing 5.6: Expresión regular para la búsqueda de un archivo clínico desde la API de GDC.

Como se puede observar en el código anterior, todos los ficheros clínicos de la plataforma comienzan por *nationwidechildrens.org_clinical_*, seguido de la opción clínica a descargar; opcionalmente, se encuentra la última versión

publicada del archivo, y, por último, se añade las siglas del cáncer (es decir, para '*TCA-ACC*', el identificador es '*acc*'. Se buscará por archivos de extensión txt, csv o tsv.

Una vez que se tiene conocimiento de que el archivo clínico está disponible en la plataforma, se descarga realizando una *POST request* a la API. El archivo descargado se almacena en la carpeta del proyecto si no existe un fichero con su mismo nombre, o creando primero una carpeta con el nombre del proyecto, en caso de no existir previamente.

Descarga del GENCODE

GENCODE es un proyecto científico de investigación de genomas que forma parte de la plataforma ENCODE. Su objetivo es identificar y clasificar todas las características genéticas en los genomas humano y de ratón con gran precisión sobre la base de pruebas biológicas, y publicar estas anotaciones en beneficio de la investigación biomédica y la interpretación del genoma.

Desde la página de GDC, se puede acceder a un enlace de descarga de la última versión del GENCODE. Con este enlace, es posible construir una query que permita la descarga del mismo desde el módulo ETL. Este archivo es útil ya que proporciona información sobre cada uno de los genes; sin embargo, tratar directamente con este fichero resulta complejo. Es por eso que se hace uso de la herramienta **GenomicFeatures**, dentro del paquete de R **BiocManager** para transformar el archivo (de extensión *.gtf*) en una tabla procesable para Python.

La descarga de este fichero permite la obtención de metadatos de los genes como, por ejemplo, la longitud del gen, información necesaria para el cálculo de normalizaciones.

Lectura de datos

Para leer datos de *RNA-Seq* provenientes de la plataforma de *GDC*, es de gran importancia tener en cuenta que la estructura de los archivos de *GDC* y de *Legacy GDC* son diferentes. Sin embargo, la lectura de datos se plantea de manera que el usuario final de esta funcionalidad desconozca este tipo de detalles. Un usuario final únicamente deberá llamar a una función general llamada *read_rna*, la cual internamente, atendiendo a la estructura del archivo, identifica si proviene del archivo Legacy o no.

gene-model: GENCODE V36
gene_id gene_name gene_type unstranded stranded_first stranded_second tpm_unstranded fpkm_unstranded fpkm_uq_unstranded
N_unmapped 1118407 1118407 1118407
N_multimapping 4669549 4669549 4669549
N_nofeature 1977087 24982757 25216539
N_ambiguous 6682268 1636287 1612882
ENSG000000000003.15 TSPN46 protein_coding 1463 738 725 28.8014 7.8743 8.7594
ENSG000000000005.6 TNMD protein_coding 3 2 1 0.1311 0.0496 0.0552
ENSG000000000419.13 DPM1 protein_coding 1685 794 811 85.7688 32.4644 36.1136
ENSG00000000457.14 SCYL3 protein_coding 389 485 358 3.6458 1.3798 1.5349
ENSG00000000460.17 Clorf112 protein_coding 289 292 315 2.2578 0.8547 0.9508
ENSG00000000938.13 FGFR protein_coding 177 86 91 3.3754 1.2777 1.4214
ENSG00000000971.16 CFH protein_coding 318 168 142 2.5954 0.9484 1.0558
ENSG00000001036.14 FUA2 protein_coding 3185 2189 2289 72.7903 27.5545 30.6518
ENSG00000001084.13 GCLC protein_coding 2454 1382 1399 18.3649 6.9520 7.7334
ENSG00000001167.14 NFYA protein_coding 1143 631 623 19.3432 7.3223 8.1454
ENSG00000001460.18 STPG1 protein_coding 316 151 179 2.3949 0.9065 1.0883
ENSG00000001461.17 NIPAL3 protein_coding 994 528 479 6.8228 2.5828 2.8731
ENSG00000001497.18 LAS1L protein_coding 3273 1644 1633 16.8132 6.3646 7.8800
ENSG00000001561.7 ENPP4 protein_coding 988 471 509 13.6099 5.1520 5.7311
ENSG00000001617.12 SEMA3F protein_coding 1338 665 665 17.7740 6.7283 7.4846
ENSG00000001626.16 CFTF protein_coding 1 0 1 0.0065 0.0025 0.0027
ENSG00000001629.18 ANKIB1 protein_coding 2236 1135 1187 19.5167 7.3880 8.2184
ENSG00000001630.17 CYP51A1 protein_coding 338 188 162 6.0309 2.2830 2.5396
ENSG00000001631.16 KRIT1 protein_coding 402 205 203 4.1794 1.5820 1.7595
ENSG00000002016.18 RAD52 protein_coding 339 184 156 4.8553 1.8388 2.0446
ENSG00000002079.14 MYH16 transcribed_unitary_pseudogene 8 6 7 0.0818 0.0310 0.0344
ENSG00000002334.12 BAD protein_coding 456 1166 1165 17.2186 6.5188 7.2507
ENSG00000002549.12 LAP3 protein_coding 3354 1671 1683 57.1562 21.6348 24.0658
ENSG00000002586.28 PAR_Y CD99 protein_coding 18307 8094 10215 243.0415 92.0024 102.3441
ENSG00000002587.10 HS5ST1 protein_coding 155 72 83 1.3445 0.5090 0.5662
ENSG00000002726.21 AOC1 protein_coding 9 4 7 0.1533 0.0580 0.0646
ENSG00000002745.13 WNT16 protein_coding 5 3 2 0.0988 0.0374 0.0416
ENSG00000002746.15 HEOM1 protein_coding 19 11 9 0.0879 0.0333 0.0376
ENSG00000002822.15 MADD11 protein_coding 9 3 6 0.0883 0.0304 0.0338
ENSG00000002834.18 LASP1 protein_coding 4393 2988 2965 48.1251 15.1892 16.8966
ENSG00000002919.15 SNX11 protein_coding 1080 525 475 15.6047 5.9071 6.5711
ENSG00000002933.9 TMEM176A protein_coding 4884 2483 2419 89.1085 33.7317 37.5233
ENSG00000003856.8 MGP protein_coding 3895 2712 2724 70.0711 26.5252 29.5867
ENSG00000003996.14 KHL113 protein_coding 86 44 42 0.7943 0.3087 0.3345
ENSG00000003137.8 CYP26B1 protein_coding 118 56 62 1.5175 0.5744 0.6390
ENSG00000003147.19 JCA1 protein_coding 74 47 43 0.9774 0.3708 0.4116
ENSG00000003249.15 DBNO01 protein_coding 247 137 111 3.5782 1.3515 1.5834
ENSG00000003393.15 ALS2 protein_coding 724 341 383 4.4734 1.6934 1.8838
ENSG00000003400.15 CASP10 protein_coding 335 169 166 2.8500 1.0788 1.2001
ENSG00000003402.28 CFLAR protein_coding 2875 1637 1638 8.2486 3.1225 3.4735

Figura 5.2: Estructura de un archivo de RNA-Seq de la plataforma GDC.

Para leer un archivo RNA-Seq proveniente del portal de **GDC** (Figura 5.2) correctamente, se debe ignorar el comentario de la primera línea. La segunda línea es la cabecera del archivo, y entre las líneas 3 y 6 se puede encontrar información general no necesaria para los análisis deseados, la cual será también ignorada. A partir de la séptima línea, se mantendrá la primera (*gene_id*) y la cuarta (*unstranded*) columna.

La información **unstranded** es utilizada para análisis básicos de RNA-Seq. Cuando se desea un análisis más riguroso sobre los datos, la columna *stranded* es más adecuada. Esto se debe a que, inicialmente, solamente se tenía en cuenta qué gen era transcrita y su cuantificación (los conteos); sin embargo, más adelante, ciertos estudios demostraron que a parte del gen, es de importancia conocer de qué hebra se transcribía, obteniendo resultados más precisos. El resto de columnas del archivo contiene la información de los conteos *unstranded* con diversas normalizaciones aplicadas.

```

gene_id normalized_count
?|106130426 0.0000
?|106133144 2.3467
?|106134869 4.8748
?|10357 153.4179
?|10431 558.2527
?|136542 0.0000
?|155068 447.2348
?|26823 1.8652
?|288660 0.0000
?|317712 0.0000
?|346682 0.0000
?|388795 0.0000
?|390284 7.6720
?|391343 0.0000
?|391714 0.0000
?|404770 0.0000
?|441362 0.0000
?|442388 0.0000
?|553137 0.0000
?|57714 274.8399
?|645851 13.5389
?|652919 36.9468
?|653553 426.4744
?|728045 0.0000
?|728603 0.0000
?|728788 2.7078
?|729884 0.4513
?|8225 1076.7913
?|90288 4.5130
A1BG|1 143.2954
A1CF|29974 0.0000
A2BP1|54715 4.0617
A2LD1|87769 31.7532
A2ML1|144568 0.0000
A2M|2 12598.9938
A4GALT|53947 56.4126
A4GNT|51146 0.4513
AAA1|484744 0.0000
AAAS|8886 1183.2971
AACSL|729522 0.4513
AACS|65985 347.9489
AADACL2|344752 0.0000
AADACL3|126767 0.0000
AADACL4|343066 0.0000
AADAC|13 0.0000
AADAT|51166 138.5478

```

Figura 5.3: Estructura de un archivo de RNA-Seq de la plataforma Legacy GDC.

En cuanto un archivo de RNA-Seq proveniente de **Legacy GDC** la estructura es notablemente diferente. En este caso, un archivo solamente consta de dos columnas: el identificador del gen, y el valor de conteo. Para este caso, todas aquellas filas que empiecen con el carácter ‘?’ deben ser ignoradas. Para el resto de filas, correspondientes a cada uno de los genes, solo se mantendrá la parte del nombre a la izquierda del carácter ‘—’, la cual se corresponde al id del gen.

Para la lectura de los datos, inicialmente es necesario leer el archivo *manifest.txt*, el cual contiene una lista de todos los ficheros y el valor de la variable *entity_id* asociado. Este id pasará a ser el nombre de las muestras de la matriz de conteo. Un primer fichero es leído, de manera que el *Dataframe* con la matriz resultante obtiene la estructura: genes como filas y muestras como columnas. Tras esto, el contenido de todos los ficheros del directorio del proyecto son leídos uno a uno (exceptuando el archivo *manifest.txt* y cualquier archivo clínico disponible) y se **concatenan** consecutivamente con el leído a priori. De manera similar a la descarga de datos, también se acude al uso de **hebras** para agilizar el proceso.

La lectura de los ficheros clínicos obtenidos de GDC es más sencilla, simplemente se debe de tener en consideración que las dos primeras líneas del archivo se corresponden a una misma cabecera, con diferentes anotaciones. Por simplicidad de los nombres, se decide quedarse con la cabecera de la segunda línea. La tercera fila de este fichero es información innecesaria, por lo cual el archivo comienza a cargarse a partir de la cuarta fila. Obligatoriamente la columna de nombre *bcr_patient_barcode* será el índice del *DataFrame* resultante, ya que este se empareja con el valor *entity_id* correspondiente a las muestras de la matriz de conteos.

5.2.2. Preprocesamiento y creación del DataObject

En el procesamiento y análisis, toda acción que se puede ejecutar sobre los datos es una **Tarea**. La clase base contiene tres métodos abstractos: *get_parameters()*, *set_parameters()* y *apply()*, por lo que cada subclase implementará estos métodos según los parámetros y acciones a realizar sobre los datos. A parte, implementa tres métodos que participan activamente en la técnica de **reflexión**: *instantiate*, *to_dict* y *from_dict*. El primero de los tres métodos se encarga de instanciar las clases en tiempo de ejecución:

```

1  @staticmethod
2      def instantiate(class_str):
3          try:
4              module_path, class_name = class_str.rsplit('.', 1)
5              module = import_module(module_path)
6              klass = getattr(module, class_name)
7          except (ImportError, AttributeError):
8              raise ImportError(class_str)
9          # Dynamic instantiation
10         instance = klass()
11     return instance

```

Code Listing 5.7: Método *instantiate* de la clase Task para instanciar una clase de forma dinámica (uso de reflexión).

Este método se aprovecha de la jerarquía de clases implementada para recoger, sin la necesidad de ser indicado explícitamente, a qué clase pertenece con tan solo el nombre de su ruta (*class_str*). El método *getattr()* devuelve el valor del atributo de un objeto y es el que permite instanciar la clase necesaria en cada momento.

```

1 @staticmethod
2 def to_dict(self):
3     json = {}
4     json["type"] = self.__class__._module_ + "." + self.
5     __class__._name_
6     json.update( self.get_parameters() )
7     return json
8
9 @staticmethod
10 def from_dict(dictionary):
11     # Dynamic instantiation & parameterization
12     instance = Task.instantiate(dictionary["type"])
13     instance.set_parameters(dictionary)
14     return instance

```

Code Listing 5.8: Métodos *to_dict()* y *from_dict()* de la clase Task para serializar y permitir la instanciación dinámica del resto de subclases de la jerarquía.

Los métodos *to_dict()* y *from_dict()* se apoyan en los métodos *get_parameters()* y *set_parameters()* de las subclases para permitir su instanciación y parametrización dinámica. El primer método serializa en formato diccionario el módulo y el nombre de la clase perteneciente, así como los parámetros de entrada que necesita; mientras que el segundo método, se encarga de recoger estos datos durante la instanciación y de llenar cada subclase con la información que necesite.

Con respecto a las clases hijas de Task, a parte de implementar las acciones a aplicar sobre los datos que recibe de entrada, también poseen un método *get_metadata()*, el cual indica los distintos argumentos de entrada que espera, junto al tipo de dato; así como el tipo de valor que se genera como salida. Por ejemplo, el método para una Tarea que reemplace unos valores por otros, ambos indicados como parámetros, se formaría de la siguiente manera:

```

1 def get_metadata(self):
2     return {
3         "Input Parameters" : {
4             "to_replace" : {
5                 "type" : (str, float, int, bool, list, np.ndarray),
6                 "description" : "Value to be replaced."
7             },
8             "replaced_by" : {
9                 "type" : (str, float, int, bool),
10                "description" : "New value for replacing."
11            }
12        },
13    }

```

```

14     "Apply Input" : {
15         "data" : {
16             "type" : pd.DataFrame,
17             "description" : "DataFrame to replace."
18         }
19     },
20     "Output" : {
21         "return" : {
22             "type" : pd.DataFrame,
23             "description" : "DataFrame with replaced values."
24         }
25     }
26 }
```

Code Listing 5.9: Métodos *get_metadata()* de la clase Replace para obtener los parámetros (y tipos) de entrada y salida.

En cada constructor de una Tarea, así como en su función a aplicar, se realiza una **comprobación de tipos**, para evitar que la clase sea instanciada incorrectamente.

```

1 def check_arguments(arguments, metadata):
2     for inp in metadata.items():
3         param = inp[0]
4         typ = inp[1]["type"]
5
6         # Check the parameter in the arguments
7         arg_value = arguments[param]
8
9         if not isinstance(arg_value, typ):
10             print(f"{arg_value} type not match with {typ}")
11             sys.exit(0)
```

Code Listing 5.10: Método que comprueba si los argumentos de una función coincide con el tipo que hay almacenado en sus metadatos.

La función *check_arguments(arguments, metadata)* es la que se encarga de comprobar si el tipo de los datos introducidos por el usuario son correctos. Por ello, recibe los argumentos de entrada (usando el método *locals()*, de Python) y los metadatos. De manera que se comprueba mediante *isinstance* si ambos tipos concuerdan.

En cuanto a **DataObject**, hace uso de las funcionalidades de processing aplicadas a los distintos conjuntos de datos que forman su estructura. Es primordial para el correcto funcionamiento de la estructura que, cada vez que se realiza una acción que modifica la estructura de uno de los conjuntos de datos, se actualicen en consecuencia el resto de conjuntos que han podido ser afectados. Es decir, si por ejemplo se ha eliminado una muestra de la matriz

de conteo, la información clínica almacenada de esa misma muestra también debe ser eliminada, de forma que ambas estructuras queden actualizadas con el mismo número de muestras. De este proceso se encargan los métodos '*set*' de DataObject, los cuales son invocados tras aplicar modificaciones sobre el objeto.

5.2.3. Workflow: Creación del flujo de trabajo

Workflow es una Tarea que almacena una secuencia Tareas. Cada una de estas Tareas se podrá aplicar más adelante de forma secuencial sobre una entrada dada.

Su estructura hace que un Workflow se ejecute de la misma manera que cualquier Tarea, lo que permite crear flujos de trabajos que formen parte de otros flujos más complejos.

Un usuario puede crear un objeto de este tipo y aplicar métodos de adición y borrado de pasos. Sin embargo, si el usuario trabaja haciendo uso del objeto DataObject, la creación y adición de Tareas al flujo se realizará de manera implícita; de forma que al finalizar el análisis de los datos, dispondrá de la secuencia de pasos realizados ya generada. Durante la aplicación de cada Tarea de las que DataObject hace uso, cada una de ellas se añade al Workflow de la siguiente manera:

```
1 dobj.get_log().add_function(self, name = "Counts - Projection",
    descp = "Extract a sub-DataFrame using a list of columns over
    the Counts DataFrame.")
```

Code Listing 5.11: Adición de una Tarea de Proyección de datos al Workflow del objeto DataObject.

5.3. Pruebas

A lo largo del desarrollo de los distintos módulos del proyecto, se implementaron baterías de **pruebas unitarias** o '**Unittests**'. El objetivo de esta técnica es probar un pequeño bloque de código, normalmente una función o método, de forma independiente.

Esta metodología de trabajo ha resultado ser de gran ayuda a la hora de desarrollar el proyecto, pues este alberga una gran cantidad de funciones y procesos complejos, por lo que realizar pruebas independientes permitió comprobar qué partes funcionaban correctamente y agilizó la localización de errores en el código.

5.4. Documentación

La generación de la documentación se ha realizado usando **Read the Docs** [43] y **Sphinx** [44]. Para ello, cada una de las clases y funciones implementadas han sido comentadas siguiendo los estándares del lenguaje Python.

Read the Docs se considera una de las mejores plataformas de documentación mediante el uso de Sphinx. **Sphinx**, por otra parte, es un software generador de documentación que convierte ficheros *reStructuredText* en sitios web HTML y otros formatos, incluyendo PDF y ePub.

Se ha hecho uso de la herramienta interactiva *sphinx-quickstart*, la cual permite generar una página HTML con la documentación completa del proyecto a través de la creación de archivos *rst* (*reStructuredText*) y la ejecución de un archivo *makefile*. Una vez generada la página, esta se puede personalizar mediante la inyección de estilos *css*.

Actualmente la documentación web está compartida de forma local, pero se ha realizado pensando en el caso de que GeneFlow se distribuya, de manera que la documentación se publicaría en un servidor web junto a la descarga e instalación del mismo.

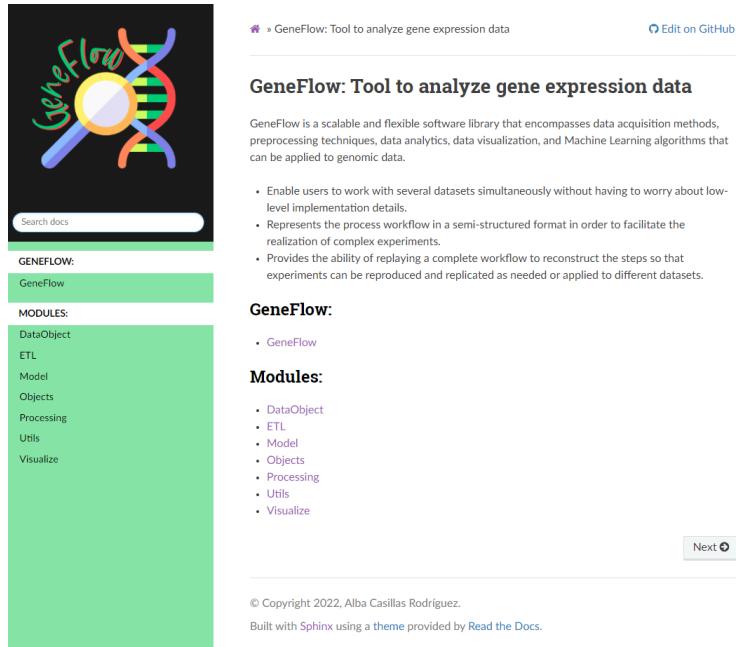


Figura 5.4: Página principal de la documentación. Realizada con Read the Docs.

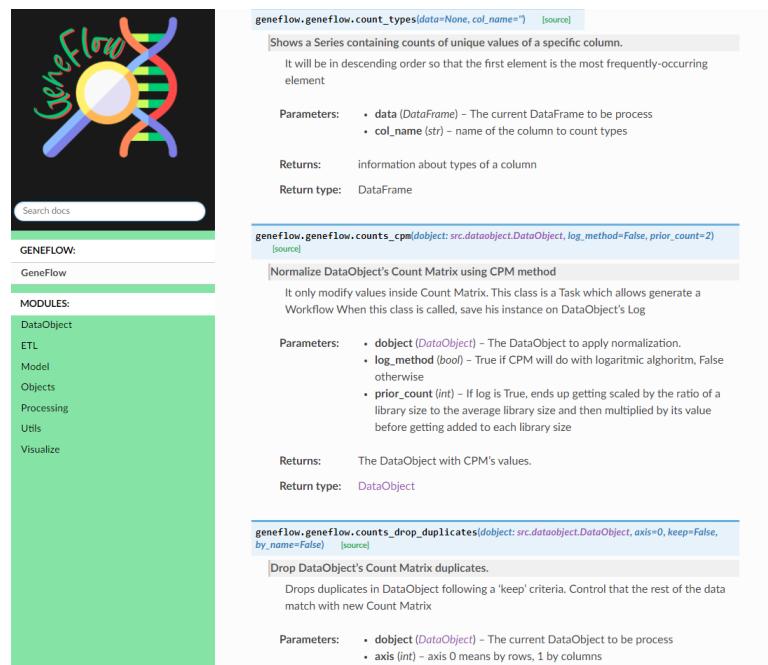


Figura 5.5: Documentación de los métodos del módulo geneflow. Realizada con Read the Docs.

Capítulo 6

Casos Prácticos

En este capítulo se expondrán una serie de casos prácticos para ilustrar cómo un usuario puede hacer uso de GeneFlow.

6.1. Caso nº1: Descarga de datos de GDC.

Un biólogo desea realizar un análisis de supervivencia en base a datos transcriptómicos del cáncer Melanoma. Para ello, quiere hacer uso de Python, pues es un lenguaje sencillo de usar. Además, sabe que en la plataforma de Genomic Data Commons está disponible una gran cantidad de datos de este tipo, pero la descarga de datos en esta plataforma exige usar una herramienta que funciona a través de la línea de comandos, y desconoce cómo indicar qué datos quiere descargar exactamente, o cómo leerlos en Python.

GeneFlow resuelve este tipo de situaciones. Ofrece funcionalidades de descarga y lectura de datos provenientes de la plataforma de GDC. A continuación, se muestra un ejemplo para la descarga de datos transcriptómicos del Melanoma (SKCM, Skin Cutaneous Melanoma).

```
1 # Query que para filtrar la descarga de datos
2 gdc_query = genf.create_gdc_query(project = "TCGA-SKCM", legacy=
    True, data_category = "Gene expression", data_type = "Gene
    Expression Quantification", experimental_strategy= "RNA-Seq",
    normalized = True)
3
4 # Descarga de datos
5 genf.gdc_download_data(gdc_query)
```

Code Listing 6.1: Descarga de datos del cáncer de Melanoma en GDC.

Una vez descargados los datos, se realiza la lectura de los mismos. El resultado se almacena en un DataFrame.

```
1 count_data = genf.gdc_read_rna("TCGA-SKCM")
2
3 count_data.head()
```

Code Listing 6.2: Lectura de los datos descargados.

Resultado:

Searching in GDC database Accessing GDC. This might take a while... Project TCGA-SKCM Checking if the parameters are correct... Downloading data for project TCGA-SKCM. Reading manifest... Manifest read. Reading the data... Joining the data... It might take a moment. Data successfully read.														
gene_id	TCGA-ER-A19G-01A-11R-A18T-07	TCGA-ER-A5EP-01A-12R-A27O-07	TCGA-GN-A19J-06A-12R-A18S-07	TCGA-GN-A26G-06A-11R-A32P-07	TCGA-EB-A3Y6-01A-11R-A329-07	TCGA-EE-A2M6-06A-12R-A18S-07	TCGA-FS-A1Z4-06A-11R-A18T-07	TCGA-EE-A17X-06A-11R-A18S-07	TCGA-D3-A5G0-06A-12R-A27O-07	TCGA-HR-A20G-06A-12R-A18U-07	TCGA-HE-A8K1-06A-21R-A37K-07	TCGA-EB-A57M-01A-51R-A311-07		
A1BG	149.7886	388.7839	477.2225	136.6594	200.4303	390.1752	78.2210	111.3032	179.4666	240.5797	...	272.1149	332.6885	304.6993
A1CF	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	...	0.0000	0.4261	0.0000
A2BP1	0.0000	0.8773	0.0000	0.0000	0.0000	0.0000	0.2908	25.5619	0.0000	0.0000	...	0.3440	0.0000	0.5053
A2LD1	60.2193	34.0476	78.6097	109.4924	108.9189	95.1715	68.7909	60.4007	61.7453	27.8925	...	106.1736	81.7980	136.5639
A2ML1	3728.0884	0.0000	0.6179	0.0000	0.0000	10.8128	0.6724	0.0000	0.3628	0.0000	...	0.0000	0.4261	1824.6589

5 rows × 473 columns

Figura 6.1: Resultado de la descarga y lectura de datos transcriptómicos de GDC.

6.2. Caso nº2: Creación de un DataObject.

El biólogo se ha percatado de que, para poder realizar el análisis de supervivencia, no solo necesita la matriz de conteo que ha descargado, sino también información clínica de las muestras para combinarlas en el análisis. Sin embargo, desconoce cómo va a comprobar que ambos conjuntos de datos coincidan en número de muestras, ni cómo trabajar conjuntamente con ellos sin perder información.

La creación de un objeto DataObject resulta muy útil en este tipo de situaciones, puesto que combina diferentes conjuntos de datos de manera que todos ellos comparten la misma información. Durante la creación del DataObject, se conservan aquellas variables encontradas en todos los conjuntos implicados. Sucede de igual manera a la hora de trabajar y modificar este objeto.

```

1 nm_clinical_type = "patient"
2
3 genf.gdc_download_clinical_data("TCGA-SKCM", nm_clinical_type,
4 legacy = True)
5 clinical_data = genf.gdc_read_clinical("TCGA-SKCM/
nationwidechildrens.org_clinical_patient_skcm.txt")

```

Code Listing 6.3: Descarga y lectura de datos clínicos.

```

1 # Creacion del DataObject
2 data = genf.create_data_object(count_data, obs_ = clinical_data)
3
4 data.summary_object()

```

Code Listing 6.4: Creación de un objeto DataObject.

```

Searching in GDC database
Clinical Information: patient already exists in the current directory.

Creating DataObject...

#####
Data Object
Dimensions: (20502, 473)
Row Names (20502): A1BG ... tAKR
Column Names (473): TCGA-3N-A9WB ... TCGA-Z2-AA3V

Var
There is no data for Var

Obs
Dimensions: (469, 61)
Row Names (469): TCGA-3N-A9WB ... TCGA-Z2-AA3V
Column Names (61): bcr_patient_uuid ... tumor_tissue_site_other
#####

```

Figura 6.2: Visualización de un objeto de tipo DataObject con matriz de conteos y metadatos clínicos.

6.3. Caso nº3: Visualización de datos.

Una investigadora está analizando los transcriptomas de células basales y luminales en las glándulas mamarias de ratonas embarazadas, con descendencia lactante y sin descendencia. Sus datos están normalizados, y quiere ordenar y mantener únicamente los 500 genes con mayor varianza para poder sacar una gráfica que le permite visualizar el grado de correlación entre muestras y genes.

La investigadora puede hacer uso de la biblioteca para seleccionar los 500 genes con mayor varianza. Los genes seleccionados serán el parámetro

de entrada para calcular un *clustermap* que le permita tener una visualización general sobre las relaciones entre sus variables y observaciones (genes y muestras, respectivamente). El cálculo del clustermap solo calcula la figura correspondiente, para que, por libre elección de la investigadora, sea ella quién decida si mostrarla o no.

```
1 # Ordena por varianza y selecciona los 500 primeros elementos (genes).
2 select_var = genf.top_variables(data_object.get_counts(), 500)
3
4 clsm = genf.clustermap(select_var)
5
6 genf.show_figure(clsm, legend = False, title = "Clustermap")
```

Code Listing 6.5: Selección de genes por su varianza y cálculo del clustermap.

A continuación, se muestra la visualización obtenida.

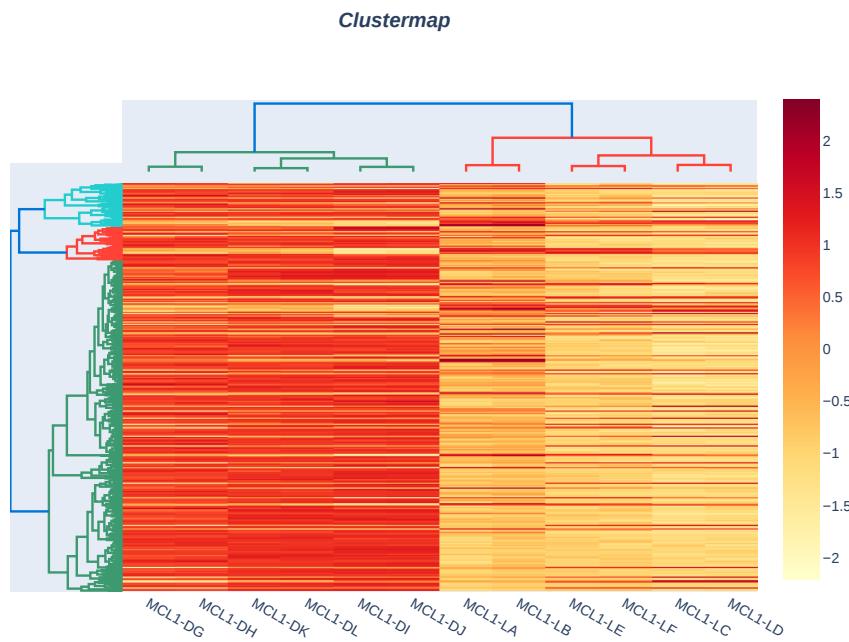


Figura 6.3: Resultado: Visualización de un clustermap.

La misma investigadora quiere obtener visualizaciones que muestren qué genes son estadísticamente más significativos y cómo se relacionan las muestras teniendo en cuenta el tipo de célula de las ratonas (basal o luminal).

Para ello, puede relacionar la matriz de conteo de los transcriptomas con la información clínica del tipo de célula para calcular las gráficas MDS y Volcano:

```

1 cell_types = genf.data_projection_name(data_object.get_obs() ,
2                                         "CellType")
3
4 mds_type = genf.mds_plot(data_object.get_counts() ,
5                           clinical_info = cell_types)
6
7 genf.show_figure(mds_type , xlabel="Leading logFC dim 1" ,
8                   ylabel="Leading logFC dim 2" ,
9                   title = "MDSplot segun Tipo Celular")

```

Code Listing 6.6: Creación de la figura de MDS plot.

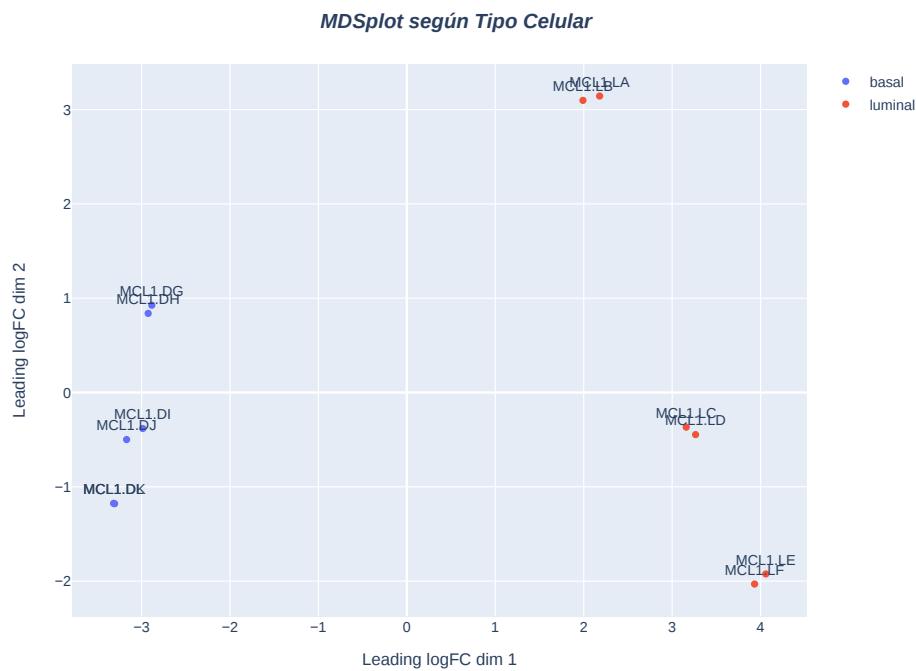


Figura 6.4: Resultado: Visualización de un MDS según Tipo Celular.

```

1 volc = genf.volcano_plot(proc_df, "CellType", "basal",
2                         "luminal", df_samples)
3
4 genf.show_figure(volc, xlabel="Log2FC", ylabel="-Log10 p-value",
5                  title = "{} vs {}".format("basal", "luminal"))

```

Code Listing 6.7: Creación del gráfico de Volcán.

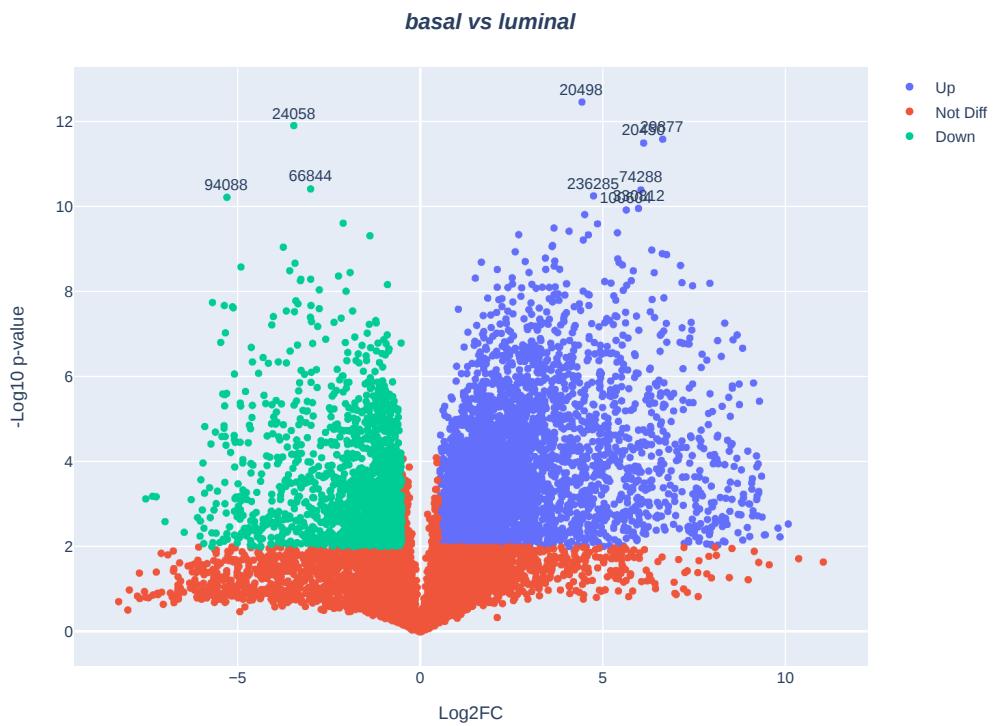


Figura 6.5: Resultado: Visualización de un gráfico de Volcán.

6.4. Caso nº4: Preprocesamiento de datos.

Durante un análisis de datos genómicos de cáncer de mama, se quiere calcular la correlación entre las variables para decidir si eliminar aquellas variables muy correladas, con el objetivo de reducir la dimensionalidad del conjunto de datos.

GeneFlow ofrece distintos cálculos útiles en una fase de preprocesamiento de datos como, por ejemplo, el cálculo de la correlación entre variables o

la eliminación de variables que superen un cierto umbral de correlación. Primero, se visualiza si las variables del conjunto de datos sufren una alta correlación entre ellas:

```

1 corr = genf.var_correlation(data_object.get_counts())
2
3 fig = genf.heatmap(x = corr.columns, y = corr.index,
4                     z = genf.ut.np.array(corr))
5
6 genf.show_figure(fig, xlabel = "x axis", ylabel = "y axis",
7                   legend = False, title = "Correlation between variables")

```

Code Listing 6.8: Cálculo de la correlación entre variables.

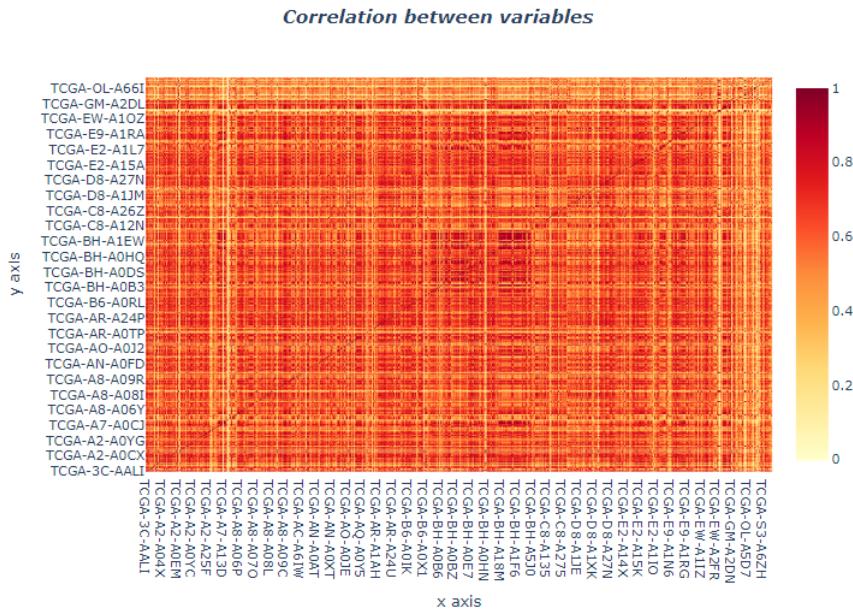


Figura 6.6: Resultado: Visualización de la correlación entre variables

Observando la gráfica anterior, el usuario observa que existen numerosas variables con un alto grado de correlación, por lo que decide eliminar aquellas que superan un umbral de 0.6 (60 % de correlación):

```

1 genes = genf.remove_correlation(data_object.get_counts(),
2                                   thresh = 0.6)

```

Code Listing 6.9: Eliminación de variables correladas.

Tras este borrado, el usuario eliminó aproximadamente la mitad de las variables.

```
DIMENSIONES ANTES DE ELIMINAR VARIABLES CORRELADAS:  
(1006, 878)  
DIMENSIONES TRAS ELIMINAR VARIABLES CORRELADAS  
(506, 878)
```

Figura 6.7: Resultado tras eliminar variables con más de un 60 % de correlación.

6.5. Caso nº5: Ajuste de hyper-parámetros de un modelo

Tras haber realizado un preprocesamiento de datos, un usuario quiere entrenar un modelo Random Forest. Sin embargo, quiere probar con distintos números de árboles (parámetro n_estimators) y obtener la mejor predicción posible.

Para ello, GeneFlow ofrece la posibilidad de crear modelos de Machine Learning y ajustar los distintos hyper-parámetros del modelo, únicamente pasándole una lista con los valores a probar por cada parámetro deseado. Este ajuste entrenará el modelo con la mejor métrica obtenida.

```
1 # Creacion del un modelo Random Forest.  
2 random_forest = genf.model_random_forest('RF',  
3                                         criterion = 'gini')  
4  
5 random_forest.hyperparameter_tuning(data.get_X_train(),  
6                                         data.get_Y_train(),  
7                                         n_estimators = [10,50,100,500])  
8  
9 random_forest.fit(data.get_X_train(), data.get_Y_train())  
10  
11 predictions_proba = data.predict_proba(random_forest)  
12  
13 predictions = data.get_predictions(predictions_proba, threshold  
14                                         = 0.5)  
15 metrics = random_forest.get_metrics(data.get_Y_test(),  
predictions, predictions_proba)
```

Code Listing 6.10: Creación de la figura de Volcán.

Como resultado, este fue el desempeño del modelo entrenado:

model_name	precision	recall	specificity	f1	accuracy
RF	0.862745	0.578947	0.906667	0.692913	0.741722

Figura 6.8: Resultado: Métricas obtenidas tras un ajuste de hyper-parámetros de Random Forest.

Con las predicciones obtenidas, el usuario también pudo visualizar gráficamente el desempeño del modelo mostrando la curva Precision-Recall para varios umbrales:

```
1 prc_vs_threshold = genf.plot_prec_recall_vs_thresh(data .  
get_Y_test() , predictions_proba)  
2  
3 genf.show_figure(prc_vs_threshold , xlabel = "Threshold" , title =  
'Precision - Recall vs Thresholds Curve')
```

Code Listing 6.11: Creación de la figura de Volcano plot.

A continuación, se muestra el desempeño del modelo entrenado:

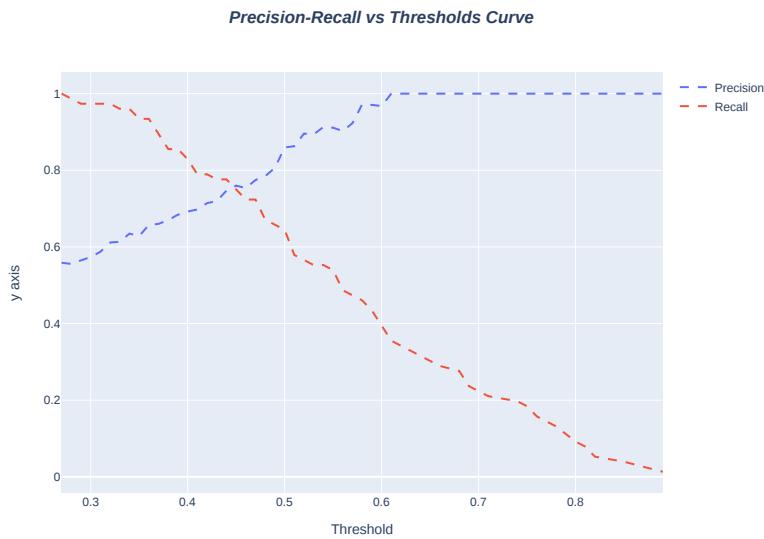


Figura 6.9: Resultado: Curva Precision-Recall para distintos umbrales.

6.6. Caso nº6: Replicando el flujo de datos (Workflow y Reflexión)

En un equipo de investigación, uno de los bioinformáticos realiza un pequeño experimento con transcriptomas de células. Decide aplicar un filtrado para eliminar genes no expresados, es decir, aquellos que tienen un bajo número de copias en las muestras. Para ello, normaliza los datos con la técnica CPM (Counts per million) y mantiene los valores de la matriz de CPMs superiores a un umbral de 0.5. Tras realizar estas operaciones, decide almacenar el flujo de trabajo en un fichero JSON.

Inicialmente, el usuario crea un objeto DataObject con sus datos, los cuales tienen la siguiente estructura:

#												
Data Object												
Dimensions: (27179, 12)												
Row Names (27179): 497097 ... 100504472												
Column Names (12): MCL1.DG ... MCL1.LF												
Var												
There is no data for Var												
Obs												
Dimensions: (12, 2)												
Row Names (12): MCL1.DG ... MCL1.LF												
Column Names (2): CellType ... Status												
#												
COUNT CPM												
MCL1.DG MCL1.DH MCL1.DI MCL1.DJ MCL1.DK MCL1.DL MCL1.LA MCL1.LB MCL1.LC MCL1.LD MCL1.LE MCL1.LF												
Gene_ID												
497097	18.856844	13.775439	2.697010	10.456480	16.442685	14.338969	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
100503874	0.043052	0.000000	0.041492	0.044120	0.000000	0.199846	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
100038431	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
19888	0.043052	0.045918	0.000000	0.000000	0.000000	0.000000	0.490386	0.138197	0.449608	0.090958	0.000000	0.000000
20671	4.563528	8.357099	3.402382	4.632618	1.997275	4.096848	0.784617	1.151641	0.809294	0.363831	0.12134	0.40556

Figura 6.10: Objeto DataObject y matriz de conteo iniciales.

Una vez cargada la estructura de datos, aplica el cálculo del CPM y el filtrado de genes poco expresados en las muestras.

```
1 data_object = genf.counts_cpm(data_object, log_method = True)
```

Code Listing 6.12: Cálculo de normalización CPM (Counts per million).

```

1 threshs = genf.remove_low_reads(data_object.get_counts(), 0.5)
2
3 subdf = genf.data_selection_filter(data_object.get_counts(),
4 filter_= threshs)
5 proc_df = genf.drop_values(subdf)
6
7 genf.set_counts(data_object, proc_df)

```

Code Listing 6.13: Borrado de genes poco expresados en las muestras.

A continuación, se muestra el resultado obtenido:

Result after removing poorly expressed genes													
# #####													
Data Object													
Dimensions: (15804, 12)													
Row Names (15804): 497097 ... 170942													
Column Names (12): MCL1.DG ... MCL1.LF													
Var													
There is no data for Var													
Obs													
Dimensions: (12, 2)													
Row Names (12): MCL1.DG ... MCL1.LF													
Column Names (2): CellType ... Status													
# #####													
MCL1.DG MCL1.DH MCL1.DI MCL1.DJ MCL1.DK MCL1.DL MCL1.LA MCL1.LB MCL1.LC MCL1.LD MCL1.LE MCL1.LF													
Gene_ID													
497097	18.856844	13.775439	2.697010	10.456480	16.442685	14.338969	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
20671	4.563528	8.357099	3.402382	4.632618	1.997275	4.096848	0.784617	1.151641	0.809294	0.363831	0.121340	0.405560	
27395	13.303116	10.744842	13.982959	13.236051	13.469996	13.489622	27.461598	21.374459	21.985821	14.917065	12.417172	13.870136	
18777	28.070005	23.647836	39.334851	41.252358	43.103987	39.519598	40.505856	39.708585	30.033801	29.379341	22.003066	23.563008	
21399	69.055657	68.647602	71.408522	58.106263	53.833535	53.259028	65.417448	57.950580	48.018113	42.113421	20.546981	20.277976	

Figura 6.11: DataObject obtenido después de eliminar genes poco expresados.

Al trabajar haciendo uso de un objeto DataObject, se ha generado implícitamente un *Workflow* con los pasos realizados en el flujo de trabajo, por lo que el usuario guarda en una carpeta personal un archivo JSON que contiene las acciones realizadas sobre los datos en un formato diccionario.

Semanas más tarde, uno de los compañeros del equipo de investigación desea reproducir el análisis realizado previamente para poder ampliarlo. En vez de realizar manualmente los mismos pasos que el otro Bioinformático, decide hacer uso del archivo que guardó su compañero, el cual solamente tiene almacenada la lista de tareas que fueron ejecutadas y los parámetros de entrada que recibieron.

Gracias a la capacidad de reflexión, el investigador puede reproducir las acciones realizadas por su compañero únicamente cargando el archivo JSON. De esta forma, se instanciarán dinámicamente las distintas tareas para ser aplicadas sobre los datos de entrada, sin necesidad de volver a realizar el experimento desde cero.

```

1 print("Using reflection...")
2
3 json_workflow = genf.workflow_from_json(
4     path = "/content/drive/MyDrive/TFG/GeneFlow/data/Reflection/
      workflow.json")
5
6 # Los análisis se realizan sobre data_object_copy , la cual es
    una copia de los datos iniciales
7 new_data_object = genf.replicate_workflow(json_workflow ,
8     data_object_copy)

```

Code Listing 6.14: Réplica de un flujo de trabajo reconstruyendo los objetos a partir de diccionarios.

```

Using reflection (reconstructing objects from dictionaries)...
#####
Data Object
Dimensions: (15804, 12)
Row Names (15804): 497097 ... 170942
Column Names (12): MCL1.DG ... MCL1.LF

Var
There is no data for Var

Obs
Dimensions: (12, 2)
Row Names (12): MCL1.DG ... MCL1.LF
Column Names (2): CellType ... Status
#####

          MCL1.DG  MCL1.DH  MCL1.DI  MCL1.DJ  MCL1.DK  MCL1.DL  MCL1.LA  MCL1.LB  MCL1.LC  MCL1.LD  MCL1.LE  MCL1.LF
Gene_ID
497097  18.856844 13.775439  2.697010 10.456480 16.442685 14.338969  0.000000  0.000000  0.000000  0.000000  0.000000
20671   4.563528  8.357099  3.402382 4.632618  1.997275  4.096848  0.784617  1.151641  0.809294  0.363831  0.121340  0.405560
27395   13.303116 10.744842 13.982959 13.236051 13.469996 13.489622 27.461598 21.374459 21.985821 14.917065 12.417172 13.870136
18777   28.070005 23.647836 39.334851 41.252358 43.103987 39.519598 40.505856 39.708585 30.033801 29.379341 22.003066 23.563008
21399   69.055657 68.647602 71.408522 58.106263 53.833535 53.259028 65.417448 57.950580 48.018113 42.113421 20.546981 20.277976

```

Figura 6.12: DataObject obtenido después de aplicar reflexión para reproducir un flujo de trabajo.

Capítulo 7

Conclusiones

7.1. Lecciones aprendidas

El objetivo de este proyecto ha sido crear un paquete software escalable y flexible que permita realizar distintos tipos de análisis sobre diferentes conjuntos de datos genómicos. Finalmente, la funcionalidad del proyecto ha sido acotada para ser implementada en un período de tiempo factible. Aunque ya había trabajado con Python anteriormente, la mayoría de los proyectos realizados en este lenguaje forzaban a simplemente desarrollar un programa sin estructura donde todo se ejecutaba de manera secuencial. Sin embargo, para este trabajo, ha sido necesario implementar un diseño orientado a objetos. Esta labor me ha supuesto un gran reto, permitiéndome así mejorar mis habilidades en programación.

A su vez, he podido desarrollar una visión más amplia a la hora de diseñar la estructura de un sistema. He aprendido las ventajas que conlleva realizar una jerarquía de clases, donde cada una de ellas representa una acción sobre los datos; así como las de utilizar técnicas de las que nunca antes había escuchado hablar, como la reflexión.

Se propuso crear este paquete de una manera sencilla y entendible para los expertos de una industria amplia y en constante avance e investigación, englobando en ella técnicas para los distintos pasos requeridos en un análisis de cualquier tipo de datos genómicos. Esta idea me ha hecho ver que, para desarrollar correctamente una herramienta, primero se debe conocer bien el ámbito al que se va a destinar, pues de esta forma no se obtendrán conclusiones incorrectas o descabelladas. Este quizás es el paso más difícil de todos: obligarse a salir fuera de la zona de confort para poder ofrecer un producto o servicio de calidad a otros sectores.

7.2. Vías futuras

En esta sección se abordan los posibles trabajos futuros relacionados con la implementación de técnicas para el análisis de datos genómicos:

- Se ha desarrollado una biblioteca software agnóstica y fácilmente escalable. Por ello, este trabajo podría ser ampliado mediante:
 - La posibilidad de descargar información de otras fuentes de datos distintas a GDC.
 - La adición de técnicas de análisis de datos enfocadas en otro tipo de variables genómicas y no únicamente a datos de RNA-Seq.
 - La implementación de nuevas técnicas de Machine Learning y Deep Learning que permitan realizar estudios más profundos y elaborados sobre los datos.
- Uso del paquete para la implementación de un entorno de programación visual de más alto nivel de abstracción. Esta herramienta se basaría en un entorno de diagramas de bloques que permita a un usuario la ejecución de experimentos complejos sin atender a detalles de programación.

Capítulo 8

Bibliografía

Bibliografía

- [1] James Dewey Watson. American geneticist and biophysicist (1928-)
<https://www.britannica.com/biography/James-Dewey-Watson>
- [2] R. Bhardwaj, A. Sethi and R. Nambiar. 2014. Big data in genomics: An overview, *IEEE International Conference on Big Data (Big Data), 2014*. doi: 10.1109/BigData.2014.7004392.
- [3] Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, Iyer R, Schatz MC, Sinha S, Robinson GE. 2015 Jul 7. Big Data: Astronomical or Genomical? *Plos Biology*, 2015. doi: 10.1371/journal.pbio.1002195.
- [4] Índice TIOBE for August 2022.
<https://www.tiobe.com/tiobe-index/>
- [5] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, SMOTE: Synthetic Minority Over-sampling Techniqu, 2002, doi: 10.1613/jair.953
- [6] The Human Genome Project
<https://www.genome.gov/human-genome-project>
- [7] Genetic Alliance; The New York-Mid-Atlantic Consortium for Genetic and Newborn Screening Services. 2009. Cómo entender la genética: Una guía para pacientes y profesionales médicos en la región de Nueva York y el Atlántico Medio. Washington (DC): Genetic Alliance; 2009 Jul 8. Available. *National Library of Medicine*, 2009.
<https://www.ncbi.nlm.nih.gov/books/NBK132207/>
- [8] Fotografía: Saluteca. ¿Conoces el ADN?
<https://www.saluteca.com/conoces-el-adn/>

- [9] Venter JC, Adams MD, Myers EW, Li PW, Mural RJ, Sutton GG, et al., 2001 Feb 16. The sequence of the human genome. *Science*, 2001. doi: 10.1126/science.1058040.
- [10] Fotografía: Universidad Compulutense de Madrid. EL CROMOSOMA EUCARIOTICO.
<https://www.saluteca.com/conoces-el-adn/>
- [11] Frigolet, Maria & Gutiérrez-Aguilar, Ruth. 2017 Sept. Ciencias “ómica”, ¿cómo ayudan a las ciencias de la salud?. *ResearchGate*, 2017. doi: 10.22201/codeic.16076079e.2017.v18n7.a3
- [12] Fotografía: Coursea. Programa especializado: Bioinformática. Pavel Pevzner.
<https://www.coursera.org/specializations/bioinformatics>
- [13] Kukurba KR, Montgomery SB. 2015 Apr 13. RNA Sequencing and Analysis. *CSH Protocols*, 2015. doi: 10.1101/pdb.top084970.
- [14] Lowe R, Shirley N, Bleackley M, Dolan S, Shafee T. 2017 May 18.. Transcriptomics technologies. *Plos Computational Biology*, 2017. doi: 10.1371/journal.pcbi.1005457.
- [15] Shui Qing Ye, Big Data Analysis for Bioinformatics and Biomedical Discoveries. *Routledge*. 2021 March 31.
- [16] Ergin S, Kherad N, Alagoz M. 2022 Jan. RNA sequencing and its applications in cancer and rare diseases. *Springer*, 2022 doi: 10.1007/s11033-021-06963-0.
- [17] Ana-Teresa M, Stephen J.S., Ana J. F., Suet-Feuhng C. 2017. Big data in cancer genomics. *ScienceDirect*, 2022. doi: org/10.1016/j.coisb.2017.07.007.
- [18] Genomic Data Commons Data Portal, from National Cancer Institute (NIH).
<https://portal.gdc.cancer.gov/>
- [19] Heath, A.P., Ferretti, V., Agrawal, S. et al. 2021. The NCI Genomic Data Commons. *Nature Genetics*, 2021.doi: 10.1038/s41588-021-00791-5
- [20] Cancer Genome Atlas Research Network, Weinstein JN, Collisson EA, Mills GB, Shaw KR, Ozenberger BA, Ellrott K, Shmulevich I, Sander C, Stuart JM. 2013 Oct. The Cancer Genome Atlas Pan-Cancer analysis project. *Nature Genetics*, 2013. doi: 10.1038/ng.2764.
- [21] Ewy M., Sean D. Statistical Genomics. Methods and Protocols. *Springer*. 2016. doi: 10.1007/978-1-4939-3578-9

- [22] Colaprico A, Silva TC, Olsen C, Garofano L, Cava C, Carolini D, Sabetot T, Malta TM, Pagnotta SM, Castiglioni I, Ceccarelli M, Bontempi G and Noushmehr H. 2015. TCGAbiolinks: An R/Bioconductor package for integrative analysis of TCGA data. *Nucleic Acids Research*, 2015. doi: 10.1093/nar/gkv1507.
- [23] Silva TC, Colaprico A, Olsen C, Angelo FD, Bontempi G, Ceccarelli M, Noushmehr H. 2022. TCGAWorkflow: TCGA Workflow Analyze cancer genomics and epigenomics data using Bioconductor packages. *F1000Research*, 2022 doi: 10.18129/B9.bioc.TCGAWorkflow
- [24] Goksuluk D, Zararsiz G, Korkmaz S, Eldem V, Zararsiz GE, Ozacetin E, Ozturk A, Karaagaoglu AE. 2019. MLSeq: Machine learning interface for RNA-sequencing data. *ScienceDirect*, 2019. doi: 10.1016/j.cmpb.2019.04.007.
- [25] Teichman, G. (2021) RNAnalysis: RNA Sequencing analysis pipeline (Python package version 2.1.1).
- [26] Isaac V., Gökcen E., Sergei R., Fidel R, Giovanni P., et al. (2018) RNAnalysis: Scanpy – Single-Cell Analysis in Python (Python package version1.9.1).
- [27] Evans C, Hardin J, Stoebel DM. 2018 Sep 28. Selecting between-sample RNA-Seq normalization methods from the perspective of their assumptions. *Oxford Academic*, 2018. doi: 10.1093/bib/bbx008.
- [28] Logistic Regression from scikit-learn.
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
- [29] Peng, Joanne & Lee, Kuk & Ingersoll, Gary. 2002 Sept. An Introduction to Logistic Regression Analysis and Reporting. *JSTOR*, 2002. doi: 10.1080/00220670209598786.
- [30] Random Forest Classifier from scikit-learn.
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- [31] Breiman, L. 2001. Random Forests. *Springer*, 2001. doi: 10.1023/A:1010933404324
- [32] Support Vector Machine from scikit-learn.
<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>
- [33] Evgeniou, Theodoros & Pontil, Massimiliano. (2001). Support Vector Machines: Theory and Applications. 2001 Jan. doi: 10.1007/3-540-44673-7_12.

- [34] Multi-layer Perceptron classifier from scikit-learn
[https://scikit-learn.org/stable/modules/generated/sklearn.
neural_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)
- [35] Popescu, Marius-Constantin & Balas, Valentina & Perescu-Popescu, Liliana & Mastorakis, Nikos.2009 July. Multilayer perceptron and neural networks. WSEAS Transactions on Circuits and Systems. *ResearchGate*, 2009.
- [36] Requests - HTTP for Humans
<https://requests.readthedocs.io/en/latest/>
- [37] pandas - powerful Python data analysis toolkit
<https://pypi.org/project/pandas/>
- [38] threading - Thread-based parallelism
<https://docs.python.org/3/library/threading.html>
- [39] plotly - Low-Code Data Apps
<https://plotly.com/>
- [40] scikit-learn - Machine Learning in Python
<https://scikit-learn.org/stable/>
- [41] imblearn - Imbalanced learn
<https://imbalanced-learn.org/stable/install.html>
- [42] rpy2 - R in Python
<https://rpy2.github.io/>
- [43] Read the Docs - Documentation Simplified
<https://docs.readthedocs.io/en/stable/>
- [44] Sphinx - Python Documentation Generator
<https://www.sphinx-doc.org/en/master/>

