Conocimiento en 1000Genome y GWAS

Ramon Nou Castell
Máster universitario en Bioinformática y Bioestadística
Programación para la Bioinformática

Consultor: Pau Andrio Balado
PRA: Maria Jesús Marco Galindo
Junio de 2018

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Conocimiento en 1000Genome y GEWAS*
Nombre del autor:	*Ramon Nou Castell*
Nombre del consultor:	*Pau Andrio Balado*
Nombre del PRA:	*Maria Jesús Marco Galindo*
Fecha de entrega (mm/aaaa):	06/2018
Titulación:	*Máster universitario en Bioinformática y Bioestadística*
Área del Trabajo Final:	*Programación para la Bioinformática*
Idioma del trabajo:	*Castellano*
Palabras clave	*Machine Learning, SNP, 1000Genome, Rendimiento*

Resumen del Trabajo

Gran parte del conocimiento biológico se encuentra dividido en varias bases de datos. Gracias a los avances en la potencia de cálculo todos estos datos se pueden analizar utilizando técnicas basadas en minería de datos, estadística y machine learning. En este trabajo nos hemos centrado en dos grandes bases de datos que se pueden utilizar para encontrar relaciones entre poblaciones y distintos fenotipos utilizando SNPs (Single Nucleotide Polymorphism). En este caso, se utilizará información de la base de datos de 1000Genome, que contiene el genoma completo de más de 1000 humanos de distintas poblaciones y los datos de la base de datos GWAS que contiene los SNPs y su relación con distintos rasgos (asma, cáncer…)

Mostraremos distintas formas para extraer información, incluyendo machine learning y posteriormente aplicaremos distintos métodos para mejorar su rendimiento tanto en el plano de la computación (añadiendo paralelismo) como mejorando la entrada/salida (mejorando la distribución y la utilización de los datos).

Finalmente analizaremos la parte de aprendizaje y extracción de conocimiento comparando distintos algoritmos y métodos, realizando un análisis más detallado de los datos.

Abstract (in English, 250 words or less):

The biological knowledge, or at least a big part of it, is divided in different databases. Thanks to the advances in the computation power, we can analyse all this data using data mining, statistical methods and machine learning techniques. In this work, we will focus in two important databases that can be

used to find relations between populations and fenotypes using SNPs (Single Nucleotide Polymorphism) as features. For this work, we will use information from 1000Genome, a database containing the sequentiation of more than 1000 humans' genome and from GWAS, another database that contains the relation between SNPs and traits (i.e., asthma or cancer).

Different ways of extracting information will be presented, including machine learning. After that, a performance analysis and optimization techniques will be applied both to computation speed (parallelism) and I/O (data distribution).

Finally, a comparative analysis of machine learning algorithms will be presented.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Gran parte del conocimiento biológico se encuentra dividido en varias bases de datos.

Gracias a los avances en la potencia de cálculo - y almacenamiento - todos estos datos se pueden analizar utilizando técnicas basadas en minería de datos, estadística y machine learning.

El objetivo es encontrar puntos de unión que puedan arrojar luz a la prevención y detección de enfermedades, además de, por ejemplo, la creación de nuevos fármacos.

En este trabajo nos hemos centrado en dos grandes bases de datos de acceso público y gratuito relacionales, para encontrar relaciones entre población (y superpoblación) y distintos fenotipos (utilizando los SNPs, Single Nucleotide Polymorphism, como enlace entre ambas bases de datos). Se cogerá información de la base de datos 1000Genome (1) (2), que contiene la secuencialización del genoma completo de 1000 humanos de distintas poblaciones, y de la base de datos (3) (GWAS, genome-wide association studies) que une fenotipos (por ejemplo, asma y cáncer) con SNPs.

Con la información, pretendemos filtrar y extraer conocimiento utilizando algunas técnicas de Machine Learning. Aunque inicialmente, y así se pretendía en la definición de objetivos inicial, el trabajo quería centrarse en los aspectos más informáticos del flujo de trabajo o “workflow”: Analizar cómo se acceden a los datos y proponer mejoras en el acceso y el rendimiento. Tras ver que algunas mejoras que se querían proponer ya estaban implementadas en alguna de las librerías utilizadas hemos aprovechado el hueco en la planificación para añadir un análisis más exhaustivo de los datos y de los métodos de machine learning.

Para la elección de este TFM se han seguido los siguientes criterios:

a) **Conocimientos previos del autor**: Poder aplicar los conocimientos en áreas relacionadas con la informática (entrada/salida y mejora de rendimiento) ha sido un punto importante. Se pretende utilizar los conocimientos extraídos del TFM para conocer mejor los "workflows" de bioinformática.
b) **Interés biológico**: Se han seleccionado dos bases de datos, recientes y utilizadas actualmente, para conocer qué datos se pueden obtener de ellas, explorar y encontrar puntos de unión entre ellas que pueden ayudar a descubrir relaciones que a priori pueden estar escondidas.
c) **Interés informático**: Analizar y optimizar los distintos procesos es un paso importante en cualquier "workflow" para mejorar el uso de los recursos informáticos.

Obtener más rendimiento, en general, sirve para reducir los costes computacionales para obtener el mismo trabajo o análogamente obtener más trabajo con los mismos costes computacionales. Por otro lado, obtener cargas de trabajo de distintas áreas, por ejemplo, la bioinformática, es útil para buscar nuevos métodos de paralelización o conocer como guardar los datos. Optimizando una aplicación modelo se puede utilizar para mejorar otros procesos o workflows. Estudiar y conocer el proceso de entrada-salida fuera de los modelos tradicionales HPC (High Performance Computing) es realmente interesante en el área del autor del trabajo.

1.2 Objetivos del Trabajo

Generales

- Objetivo 1.
  - Obtener y entender los datos de los distintos experimentos.
- Objetivo 2.
  - Crear un workflow de filtrado, análisis y obtención de resultados con esos datos.
- Objetivo 3.
  - Optimizar el workflow en el plano de rendimiento tanto computacional como de almacenamiento.

Específicos

- Objetivo 1.
  a. Entender las distintas bases de datos y realizar una inspección preliminar de la información que contienen.
- Objetivo 2.
  a. Crear distintos procesos de filtrado sobre los parámetros de entrada que se obtengan en el Objetivo 1.
  b. Analizar los datos utilizando técnicas de machine learning que nos permitan extraer conclusiones y obtener respuestas a preguntas como:

     i. Existe poblacionalmente relación entre el fenotipo ’asma’ y ’cáncer de pulmón’?. Es decir, existe una proporción significativa de individuos con los dos SNPs correspondientes.
     ii. Existe relación entre unos determinados fenotipos y la población. Es decir, hay una proporción significativa o una distribución no equitativa entre el fenotipo y el origen del individuo.
     iii. Dados una serie de fenotipos seleccionados, y utilizando los datos de 1000 Genome encontrar otros fenotipos que suelen aparecer juntos. Para realizarlo se filtrarán los datos de 1000 Genome con los fenotipos seleccionados, y se relacionarán el resto de fenotipos activos. Se mostrarán los fenotipos relacionados por probabilidad de aparición.

iv. Crear modelos de ML para predecir la población a la que pertenece un individuo a partir de su genotipo.

c. Analizar en profundidad los métodos de ML utilizados y los datos, tarea no planificada inicialmente.

- Objetivo 3.
  a. Analizar el rendimiento y proponer optimizaciones.
  b. Analizar la entrada y salida y proponer alternativas para mejorar el uso de datos.
  c. Obtener medidas y resultados de rendimiento.

1.3 Enfoque y método seguido

A continuación, se enumeran algunos posibles enfoques para realizar el trabajo.

- **Secuencial**: Se llevan a cabo del primero al último cada objetivo, una vez se ha terminado uno se realiza una evaluación del mismo y se pasa al siguiente, en ningún caso se realizarán dos objetivos al mismo tiempo ni se revisitarán de nuevo. Este enfoque permite delimitar bien los distintos objetivos, pero puede causar un desbalanceo en el caso de que un objetivo requiera más tiempo del necesario. Sin embargo, si los objetivos están bien balanceados nos permite acotar y conocer en qué punto estamos del TFM.
- **Iterativo**: Los objetivos se revisitan de forma cíclica. Este enfoque nos permite obtener un prototipo global y funcional del proyecto, mientras pasamos por las distintas fases mejorando y completando el código.
- **Mixto**: Tenemos algunas partes que se realizan de forma secuencial y no se revisitan (por ejemplo, el Objetivo 1) y otras que se visitan iterativamente (Objetivo 2 y 3) para mejorar el código. Es un enfoque válido cuando realizamos tareas de análisis sobre el software creado y se deben realizar cambios y analizar de nuevo. Tiene como punto a favor que, en cualquier momento, y según el tiempo disponible, se puede detener el proceso iterativo y obtener conclusiones.

En este TFM se ha utilizado el enfoque mixto, pero limitado a los objetivos 2 y 3, concretamente a la paralelización o modificación de los distintos workflows para mejorar el uso de los datos. El tiempo disponible para la realización del TFM y la posibilidad de aparición de distintos riesgos han limitado el número de iteraciones.

Así en la primera fase, “análisis y recolección de los datos”, hemos definido qué queremos encontrar y cómo queremos utilizar los datos.

En la segunda fase, “creación del workflow”, una vez seleccionados las entradas y las salidas hemos creado distintos procesos o tareas para recuperar los datos necesarios (filtrado) y prepararlos para los algoritmos empleados. En este punto del proceso habremos creado un workflow, a priori ineficiente, pero funcional y nos permite obtener conocimiento a

partir de los datos originales. Este workflow será optimizado en la fase final.

La fase final, “análisis y optimización del workflow” se analizarán los distintos puntos de paralelización del código y se aplicará una o más técnicas de paralelización y optimización.

Finalmente, se incluye una revisita al Objetivo 2, dado que la planificación del Objetivo 3 se ha visto reducida gracias a que parte de las mejoras que se habían planificado estaban aplicadas en las librerías escogidas.

1.4 Planificación del Trabajo

Mostraremos la planificación inicial del proyecto que no incluye la alteración realizada en el Objetivo 2 – 3 durante la entrega de la última PEC.

a) **PEC 1 - Plan de Trabajo** [06/03/18 - 19/03/18]

b) **PEC 2 - Desarrollo del trabajo - Fase 1** [20/03/18 - 23/04/18]

   a. **Análisis de datos** - Se analizan las distintas bases de datos para ver que campos nos ofrecen.

   b. **Descarga de datos** - Creación de procesos para descargar y guardar los datos, no se realizarán transformaciones en los datos.

   c. **Descripción de los datos** - Se analiza que campos son los útiles para nuestro propósito, se empieza a diseñar cómo se utilizarán.

   d. **Definición del análisis** - Una vez obtenidos los datos, se refinarán los distintos análisis que podemos realizar y que tengan sentido. Aquí definiremos el workflow del código que implementaremos.

   e. **Implementación de filtros** – Implementación de los distintos filtros que aplicaremos sobre los datos para extraer los datos que se utilizarán en la siguiente fase.

   f. **Implementación de procesos de análisis** - Se implementan los distintos procesos de análisis

   g. Redacción del documento

c) **PEC 3 - Desarrollo del trabajo - Fase 2** [24/04/18 - 21/05/18]

a. **Análisis de rendimiento** - Analizaremos el rendimiento del workflow en el plano de computación y en el de entrada/salida.

b. **Paralelización** - Se proponen distintos métodos para paralelizar el código.

c. **Análisis y optimización almacenamiento** - Se proponen distintos métodos para mejorar la entrada/salida.

d. **Gráficas comparativas e iteración** - Se hace un estudio comparativo del rendimiento original y del optimizado.

e. Redacción del documento

d) **PEC 4 - Cierre de la memoria** [22/05/18 - 05/06/18]

e) **PEC 5a - Elaboración de la presentación** [06/06/18 - 13/06/18]

f) **PEC 5b - Defensa Pública** [14/06/18 - 25/06/18]

Se muestra el diagrama de Gantt (Figura 1) con la planificación de las tareas, empezando el 06/03/18 y finalizando el 05/06/18 fecha final de entrega de la memoria (Aproximadamente 3 meses).

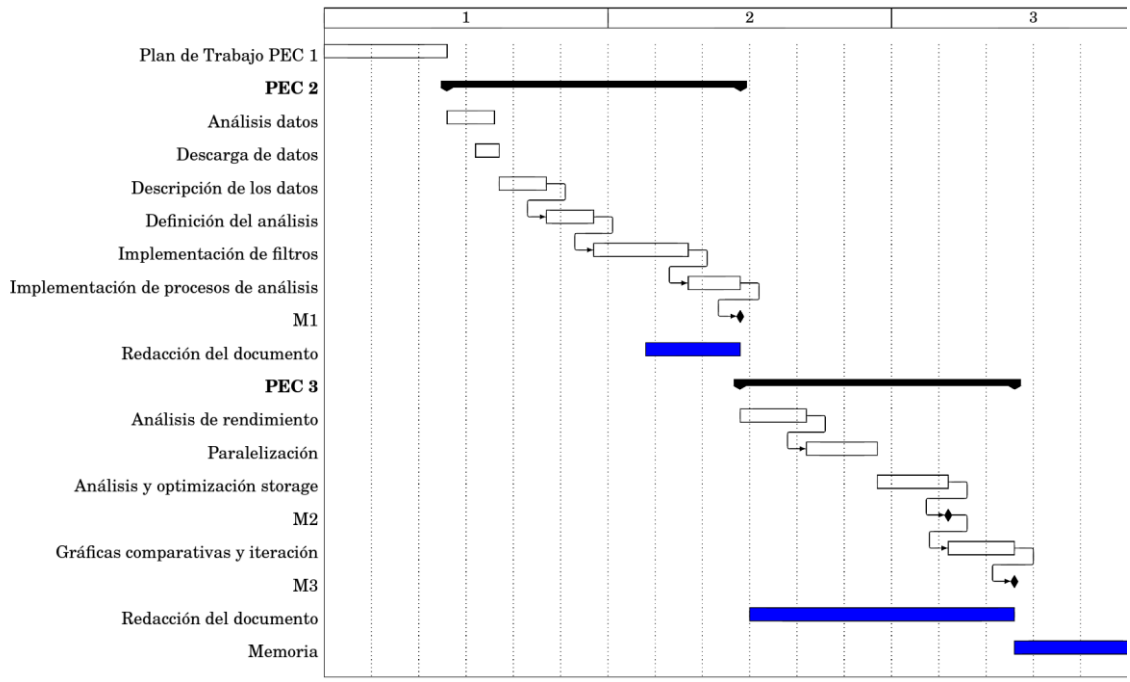

Figura 1 - Gantt Original

Los hitos definidos inicialmente son los siguientes:
M1 – Obtención del workflow sin optimizar
M2 – Obtención del workflow optimizado
M3 – Finalización del análisis de rendimiento

Al reducirse el trabajo en c) dadas las optimizaciones que ya nos ofrecía la librería scikit y matplotlib, hemos creado una nueva tarea correspondiente al objetivo 2.c, analizar con más detalle los algoritmos de ML. Así, la planificación final es la que se observa en la Figura 2.

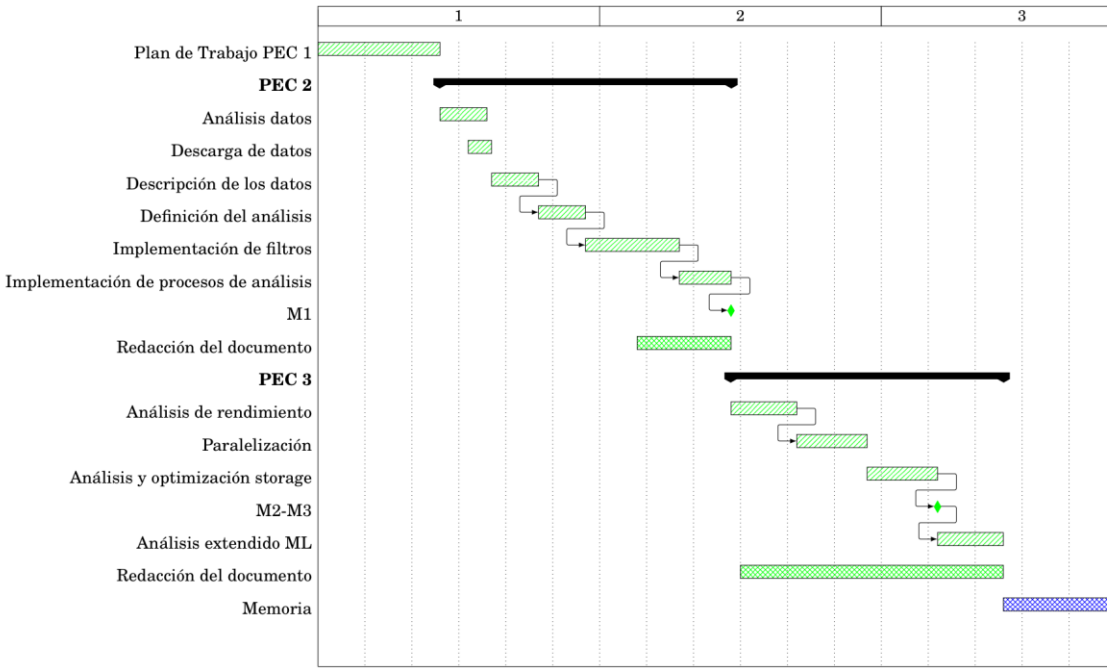

Figura 2 - Gantt Final

1.5 Breve sumario de productos obtenidos

Los productos obtenidos son la memoria, y un conjunto de código fuente en *Python* para poder realizar los distintos procesos de filtraje y análisis de los datos. Se incluyen, además, las versiones optimizadas.

El código fuente se puede encontrar en **https://github.com/mavy/BIO**

1.6 Breve descripción de los otros capítulos de la memoria

En la sección 2, describimos las dos bases de datos. En la sección 3 explicamos las decisiones técnicas que hemos tomado, por ejemplo, la utilización de Python. Entrando en la sección 4, filtrado, explicaremos los dos procesos de filtrado que se utilizan para leer las bases de datos para pasar en la sección 5 a explicar cómo se ha extraído conocimiento, junto con el análisis de ML que hemos realizado con la extensión del objetivo 2.

En la sección 6, y siguiendo la primera planificación, miramos el rendimiento de dos de los procesos de extracción de conocimiento. Finalmente, en la sección 7, acabamos con las conclusiones.

2. 1000Genome y GWAS. Descripción

Como introducción, definiremos los dos conceptos usados en este trabajo:

SNPs (Single Nucleotide Polimorphism) son variaciones de la secuencia del ADN que afecta a una sola base que se da en al menos un 1% de una población (si no llegamos, se considera una mutación puntual).

Fenotipo, es la manifestación externa de un genotipo. Por ejemplo, un cambio genético puede ser el causante de que una persona tenga los ojos azules. El fenotipo en este caso es "tener los ojos azules". No todos los cambios en el genotipo se manifiestan en un cambio en el fenotipo, aquí entrarían los genes.

A continuación, describiremos de dónde obtenemos los datos y cómo los utilizamos.

2.1. 1000Genome

Citando (4) *"El proyecto "1000 Genomes Project" se llevó a cabo entre 2008 y 2015, creando el catálogo público más largo de datos humanos genotípicos. Este proyecto se mantiene vivo gracias a IGSR (International Genome Sample Resource) permitiendo el acceso a los datos, así como continuar la incorporación de nuevos datos."*

El objetivo del proyecto **1000Genomes** era encontrar las variaciones genéticas más frecuentes (más de un 1%) en las poblaciones estudiadas. Los datos se combinaron entre diferentes muestras para detectar las variantes dentro de una región del genoma. Lo podemos encontrar descrito en la revista Nature (5).

Se planeo una fase piloto, y tres fases en el proyecto principal. La fase 1 y 3 se centraron en producir datos, mientras que la fase 2 se concentró en el desarrollo técnico. El fichero final cubre **2504** individuos de **26** poblaciones.

Todos los datos son anónimos y no incluyen datos médicos o datos de fenotipo más allá de la etnia y el género del individuo.

Posteriormente IGSR, hizo algunas mejoras en los datos: remapeó el genoma al GRCh38 (última actualización del genoma humano, de 2014)

Datos

Los datos utilizados se pueden encontrar en el FTP[1].
Los datos, ficheros con formato VCF, están separados por cromosoma (1-22, X y Y) correspondientes a la fase 3 de análisis realizado en la plataforma *Illumina*. Como únicos datos identificadores (fenotipos) del genoma tenemos la población, su superpoblación y el género. Dentro de los datos, utilizaremos sobre todo el concepto de SNPs, los polimorfismos de un solo nucleótido, estos polimorfismos

[1] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502

se marcan en el fichero si ocurren en al menos en un 1% de la población, tal como hemos indicado en la introducción.

Obtención

Para la obtención de los datos hemos realizado un pequeño código en Python (*download.py*), utilizando la librería de análisis de html BeautifulSoup (6), que descarga todos los links de una página web en un directorio. El código está claramente limitado por la conexión, con lo que no se requiere hacer ninguna optimización adicional. La descarga está formada por dos ficheros por cromosoma humano, el primero es un fichero vcf comprimido con gzip, y el segundo el índice de este fichero comprimido para poder utilizar los ficheros comprimidos directamente. En total, 19 GB de datos comprimidos aproximadamente.

Formato

Los ficheros descargados, y comprimidos, contienen el formato habitual de los datos biológicos (Tabla 1): Una lista de cabeceras con la descripción y cada línea posterior se refiere a un SNP distinto.

Tabla 1 - Descripción de los campos del fichero VCF

Nombre	Descripción	Ejemplo
CHROM	Identificador del cromosoma	1-22 / X-Y
POS	Posición en el cromosoma	2655180
ID	Identificador del SNP	Rs11575897
REF	Base de referencia	G
ALT	Base modificada	A
QUAL	Calidad	100
FILTER	Información del proceso de filtrado	PASS
INFO	Distintos campos de información	
FORMAT		GT
Individuo	Los 2504 individuos separados, con la información de cada alelo (0 indica que no se manifiesta el SNP)	0|1 o 1|0 o 1|1 o 0|0 o 1 o 0

Completado de datos

Con el fichero VCF no se observa que se pueda inferir o sacar algún dato interesante, sin embargo, unidos con los datos del fichero "*integrated call samples v3.20130502.ALL.panel*", que describen cada uno de los individuos (género, población y superpoblación).
Este fichero nos da información de cómo están distribuidos los datos entre poblaciones, superpoblaciones y género.

Las poblaciones (Tabla 2) están distribuidas en 5 superpoblaciones: AFR (Africana), AMR (América), EAS (Este Asiático), EUR (Europea), SAS (Sur Asiático).

En la Figura 3 podemos ver como se distribuyen los datos.

Tabla 2 - Poblaciones y superpoblaciones estudiadas en 1000Genomes.

Población	Descripción	Superpoblación
CHB	Han Chinese in Beijing, China	EAS
JPT	Japanese in Tokyo, Japan	EAS
CHS	Southern Han Chinese	EAS
CDX	Chinese Dan in Xishuangbanna, China	EAS
KHV	Kinh in Ho Chi Minh City, Vietnam	EAS
CEU	Utah(CEPH) with N and W.European Ancestry	EUR
TSI	Toscani in Italia	EUR
FIN	Finnish in Finland	EUR
GBR	British in England and Scotland	EUR
IBS	Iberian Population in Spain	EUR
YRI	Yoruba in Ibadan, Nigeria	AFR
LWK	Luhya in Webuye, Kenya	AFR
GWD	Gambian in Western Divisions in the Gambia	AFR
MSL	Mende in Sierra Leone	AFR
ESN	Esan in Nigeria	AFR
ASW	Americans of African Ancestry in SW USA	AFR
ACB	African Caribbeans in Barbados	AFR
MXL	Mexican Ancestry from Los Angeles USA	AMR
PUR	Puerto Ricans from Puerto Rico	AMR
CLM	Colombians from Medellin, Colombia	AMR
PEL	Peruvians from Lima, Peru	AMR
GIH	Gujarati Indian from Houston, Texas	SAS
PJL	Punjabi from Lahore, Pakistan	SAS
BEB	Bengali from Bangladesh	SAS
STU	Sri Lankan Tamil from the UK	SAS
ITU	Indian Telugu from the UK	SAS

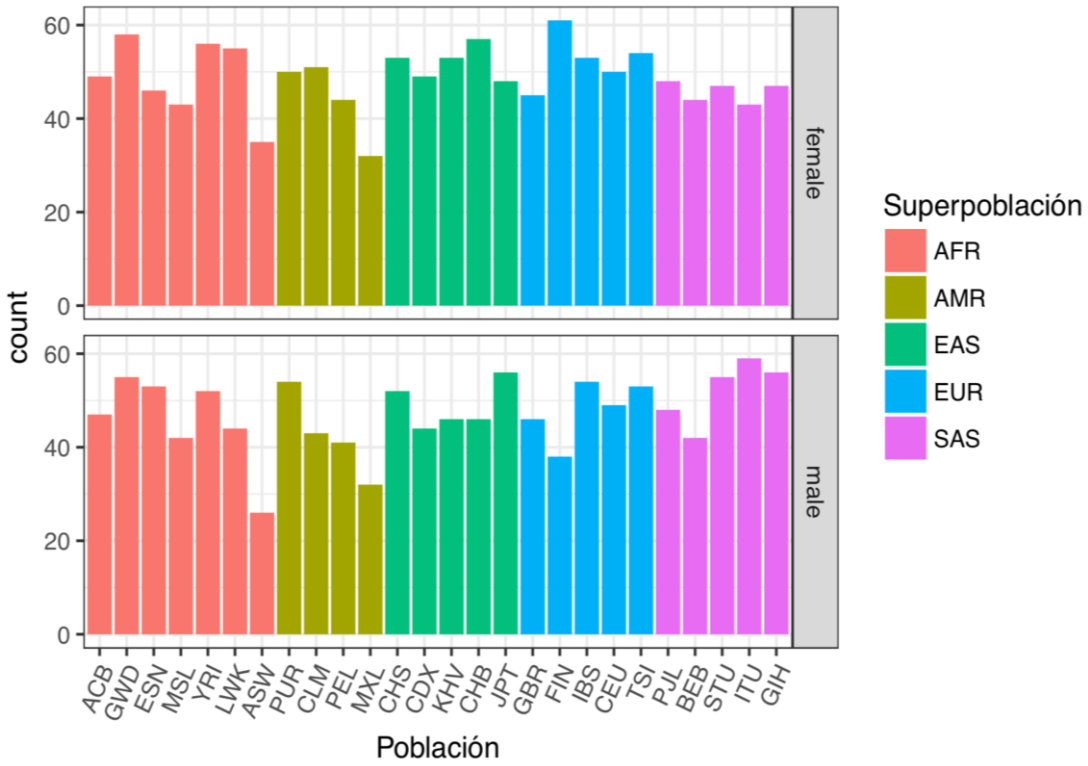

Figura 3 - Distribución de los individuos en poblaciones y superpoblaciones.

.

2.2. GWAS SNPs database

Citando la Wikipedia (7): "*En genética, un estudio de asociación del genoma completo (en inglés, GWAS (Genome-wide association study) o WGAS (Whole genome association study) es un análisis de una variación genética a lo largo de todo el genoma humano con el objetivo de identificar su asociación a un rasgo observable. Los GWAS suelen centrarse en asociaciones entre los polimorfismos de un solo nucleótido (SNPs) y rasgos como las principales enfermedades.*"

Datos

Los datos del fenotipo - SNP se han descargado del servidor GWAS [2]. Concretamente la versión 1.0.1 con anotaciones.

En el fichero descargado podemos encontrar anotados los distintos SNPs que se han identificado junto con su rasgo observable o fenotipo en distintos artículos.

[2] https://www.ebi.ac.uk/gwas/docs/file-downloads

Obtención

Los datos son un simple fichero tabulado, por lo que para su descarga se ha utilizado una simple llamada a *'wget'*[3] para obtener el fichero, que sólo tiene 42 Mbytes y 66342 entradas identificando SNPs.

Formato

Cada entrada tiene los siguientes campos descritos en la Tabla 3, aunque son de nuestro interés el campo **SNPS** (rsxxxxxx) y el **MAPPED_TRAIT** que nos indicará el fenotipo detectado. Además, usaremos el número de cromosoma (**CHR_ID**) para identificar el fichero correcto y el **CHR_POS** para la posición. Finalmente utilizando el **STRONGEST SNP-RISK ALLELE** podemos encontrar cual es el cambio en el alelo que nos da el fenotipo.

Tabla 3 - Descripción de los campos del fichero GWAS

Nombre	Ejemplo
DATE ADDED TO CATALOG	2009-09-28
PUBMEDID	18403759
FIRST AUTHOR	Ober C
DATE	2008-04-09
JOURNAL	N Engl J Med
LINK	www.ncbi.nlm.nih.gov/pubmed....
Study	Effect of variation in CHI3L1 on serum YKL-40 level, risk of asthma, and lung function
DISEASE/TRAI	YKL-40 levels
Initial Sample Size	632 Hutterite individuals
Replication Sample Size	443 European ancestry cases, 492 European ancestry controls, 206 European ancestry individuals
REGION	1q32.1
CHR_ID	1
CHR_POS	203186754
Reported Gene(s)	CHI3L1
Mapped Gene_ID	CHI3L1
UPSTREAM_GENE_ID	1116
DOWNSTREAM_GENE_ID	X
SNP_GENE_ID	X
UPSTREAM_GENE_DISTANCE	
STRONGEST SNP-RISK-ALLELE	rs4950928-G
SNPS	rs4950928
Merged	0
SNP_ID_CURRENT	4950928
CONTEXT	Upstream_gene_variant
INTERGENIC	0

[3] https://www.gnu.org/software/wget/

RISK ALLELE FREQUENCY	X
P-VALUE	X
P-VALUE_MLOG	0
P-VALUE(TEXT)	0.29
OR or BETA	1E-13
95% CI (TEXT)	13.0
PLATFORM [SNPS PASSING QC]	Affymetrix [290325]
CNV	N
MAPPED_TRAIT	YKL40 measurement
MAPPED_TRAIT_URI	http://www.ebi.ac.uk/feo/EFO...
STUDY ACCESSION	GCST000177

3. Decisiones técnicas

La implementación de los distintos códigos en este TFM se ha realizado en Python, uno de los lenguajes más utilizados en Bioinformática y del que disponemos muchas librerías. Una de las alternativas, R, nos permite realizar todo lo que hemos hecho en este TFM, pero se ha escogido Python por ser el lenguaje base de la asignatura a la que pertenece el TFM y para mejorar el conocimiento de algunas de las librerías que se utilizan, como por ejemplo scikit-learn o matplotlib.

Se ha utilizado Python 2, en vez de Python 3 (su alternativa más moderna), por tener más software y librerías compatibles.

Para las pruebas y ejecución se ha utilizado un portátil *DELL Latitude e7440* con 8GBytes de memoria, un procesador Intel I7 con 4 unidades de procesamiento, y una unidad SSD de almacenamiento. Además, se ha utilizado un *NAS* (Network Attached Storage) formado por 4 unidades HDD en RAID para probar las mejoras realizadas en la gestión de los datos.

El entorno de programación utilizado ha sido *pyCharm* (8), que ofrece un entorno integrado, muy cómodo, para Python.

Las partes comunes del código se han distribuido en un par de módulos comunes, intentando la reutilización de código. Se ha conseguido, sobre todo en la parte de abrir ficheros comunes.

4. Filtrado

En este apartado describiremos como leemos cada uno de los datos incorporados de ambas bases de datos y haremos hincapié en los distintos problemas que nos hemos encontrado.

Dados los análisis que tenemos que realizar procederemos a diseñar una serie de tareas (*workflow*) lo más común posible para poder aprovechar los distintos componentes. Los datos de origen, aunque ganaríamos rendimiento, no se van a modificar y siempre se partirá de los datos originales incluyendo la transformación en el workflow.

4.1. Fase 1 – Lectura y filtrado de datos GWAS

Tenemos que leer el fichero GWAS y extraer la relación de fenotipos e identificadores de SNPs (así como el cromosoma). Esta fase es claramente una fase que podríamos optimizar guardando ya la relación o transformando el fichero de entrada. Sin embargo, el fichero es lo suficientemente pequeño como para permitirnos utilizar el original, permitiéndonos incorporar nuevas versiones de forma sencilla. Tras este filtrado, generaremos una salida que podamos utilizar en las siguientes fases.

El proceso de filtrado se llama *read_GWAS.py*, y contiene dos funciones principales:

1. Generar una lista de las palabras clave más relevantes en el fichero (por ejemplo, asma, bronquitis o cáncer)
2. Generar la salida con los SNPs seleccionados con una o varias palabras clave, la salida es una lista de SNPs, el texto del fenotipo, el SNP junto la mutación causante y finalmente el cromosoma y su posición.

Problemas y particularidades

El formato del fichero, aunque sigue una pauta por campos bien definida, no es uniforme. Encontramos por ejemplo SNPs que se identifican con el cromosoma y la posición y posteriormente no incluyen los campos de CHR_ID o CHR_POS. Esto nos genera una larga lista de casos de uso especiales que hemos de tratar, ya sea en este proceso o en el siguiente. En nuestro caso, hemos decidido (sin ninguna razón en particular) procesarlo en la siguiente tarea del workflow.

Finalmente, los "mapped traits", tienen también un formato heterogéneo y muchos SNPs están repetidos (ya que pertenecen a distintos estudios). Aquí sí que hemos decidido fusionar los “mapped traits” y crear algunas funciones para poder buscarlos.

Funcionamiento

```
>./read_GWAS.py gwas_catalog_v1.0.1.tsv MostraKeywords
’Anorexia’,’Antibody’,’Anticoagulant’,’Arrhythmia’,’Arteritis’,’Asperger’,’Au
tism’
>./read_GWAS.py gwas_catalog_v1.0.1.tsv Selecciona "Breast cancer"
```

```
rs36194942 breast carcinoma estrogen-receptor negative breast cancer
rs36194942-A 18 27821241
```

4.2. Fase 2 – Lectura y filtrado 1000 Genomes.

Realizaremos un filtrado de cada uno de los individuos que tienen el SNP asociado al fenotipo o fenotipos seleccionados. Aceleraremos el filtrado utilizando el cromosoma encontrado en el fichero GWAS, así que la entrada será {Fenotipo, SNPs, Cromosoma} y la salida será la lista de individuos que tienen el SNP activado.

Problemas y particularidades

Una de las particularidades de este conjunto de ficheros es que están comprimidos. En un primer momento y antes de analizarlos, pensábamos que se necesitarían descomprimir los datos para acceder a ellos. Sin embargo, se utiliza el formato *tabix* (9), que genera un índice del contenido cuando se comprime con gzip. Esto nos permite reducir el espacio utilizado y permite utilizar los ficheros tal como se obtienen de la página web sin a priori grandes penalizaciones. Por contra no nos permite realizar escaneos del fichero sin descomprimirlos antes. Para poder utilizar dicho formato, hemos incluido *pytabix* (10) en el proyecto.

En el momento de unir las dos bases de datos, debemos hacerlo por el cromosoma (que nos indica el fichero a abrir) y la posición que podemos utilizar con el formato *tabix* para descomprimir correctamente la fila correspondiente al SNP seleccionado. Sin embargo, aquí nos encontramos con un problema que parecía difícil de resolver, las posiciones de los cromosomas utilizan referencias distintas en GWAS y en 1000genome. Las coordenadas obtenidas en la base de datos de GWAS utilizan una referencia distinta (*hg38*) y se ha de convertir en (*hg19*) para poder encontrar el SNP en 1000genome.

Tras investigar el problema se encontró un servicio web *LiftOver* (11) que permite transformar entre las distintas referencias, y utilizando como palabra clave *liftover* encontramos una librería Python *pyliftover* (12) que se encarga de ello utilizando un diccionario. Se reduce así el posible requerimiento de utilizar un servicio web y externo por cada SNP, que introduce una latencia grande en el proceso.

Además, como ya hemos indicado en el proceso anterior, debemos hacernos cargo de algunos SNPs que no tienen los campos completos y/o no siguen un formato uniforme. Para ello, hemos tenido que utilizar algunos bloques *try-except* para capturar excepciones en la entrada de los datos.

Todo ello está implementado en el fichero *filter_1000Genome.py* (así como su versión paralela, que comentaremos en una sección posterior). Este proceso nos filtra los distintos ficheros de 1000Genome utilizando como entrada los SNPs filtrados obtenidos con *read_GWAS*. La salida es la lista de individuos, el identificador del SNP junto si tienen el alelo activado o no, la población, la superpoblación y el género. Esta salida se utilizará en la mayoría de análisis posteriores.

Funcionamiento

Por ejemplo:

```
>./filter_1000Genome.py fenotypeData /home/rnou/files/ salida.txt
NA19454 rs4245739 1 LWK AFR male
NA19454 rs11076805 1 LWK AFR male
NA19454 rs115635831 0 LWK AFR male
HG00739 rs11155804 1 PUR AMR male
HG00739 rs4245739 1 PUR AMR male
HG00739 rs11076805 0 PUR AMR male
HG00739 rs115635831 0 PUR AMR male
NA19108 rs11155804 1 YRI AFR female
NA19108 rs4245739 1 YRI AFR female
NA19108 rs11076805 0 YRI AFR female
NA19108 rs115635831 0 YRI AFR female
```

Con estos dos procesos ya podemos definir cómo podemos utilizar los datos para extraer conocimiento.

5. Extracción de conocimiento

En la fase de definición del plan de trabajo, intentamos definir una serie de procesos (utilizando estadística tradicional y machine learning) para alcanzar dos objetivos: 1) fueran biológicamente interesantes, y 2) nos ofrecieran alguna complejidad algorítmica que nos permitiera una optimización tanto computacional como de almacenamiento.

5.1. Individuos con los SNPs manifestados.
Es un proceso muy sencillo que recorre directamente la lista obtenida en el proceso anterior para mostrarnos la lista de individuos que tienen todos los SNPs activados.

Se incluye en el fichero de código *analisis_1.py*, y tiene como parámetro el fichero de salida de *filter_1000genome.py*.

Es un proceso muy rápido, ya que todo el trabajo se realiza al leer los ficheros del genoma.

Biológicamente, nos permitiría diferenciar los individuos que tienen en el genoma los SNPs activados y observar si existe una correlación elevada entre ellos.

Funcionamiento

```
> análisis_1.py < fichero salida filter_1000Genome.py>
9 2495 [’HG00253’, ’HG00371’, ’HG00157’, ’HG02724’, ’NA20872’, ’NA20351’, ’NA11994’, ’HG02657’, ’HG03851’]
```

En este caso de ejemplo, sólo 9 individuos (de 2504) tienen los treats activos simultáneamente.

Si realizamos una búsqueda por *breast cancer*, nos saldrán muchos SNPs, pero si seleccionamos unos cuantos de ellos:

```
rs11155804   breast carcinoma … rs11155804-?       6     151625017
rs115392158  breast cancer, …   rs115392158-G      6     31347004
rs2300206    breast cancer, …   rs2300206-G        20    34002002
rs3757318    triple-negative breast …  rs3757318-A  6     151592978
```

Obtenemos una lista de 354 individuos que tienen esos 4 SNPs activos.

5.2. Distribución de un determinado SNP en las distintas poblaciones.
Utilizando los datos obtenidos en los procesos de filtrado, obtenemos un *pdf* con una serie de gráficas (generadas con matplotlib) que nos muestran por cada SNP, la proporción de cada población (respecto el total de 1000Genome) que tienen el fenotipo activado.

Problemas y particularidades

Se ha tenido que investigar la forma de crear gráficos en varias páginas con matplotlib, evitando crear n ficheros. Posteriormente en el análisis de rendimiento

se ha visto que la librería matplotlib ya genera gráficos de forma paralela, utilizando todas las CPUs disponibles, añadir más paralelismo hubiera generado un problema de memoria (Las gráficas se debían mantener en memoria). Tras observar esto, se ha modificado ligeramente la generación de las gráficas para reducir el consumo de memoria.

Funcionamento

```
> analisis_2.py fenotypeData /home/rnou/files/ outputFile.txt salida.pdf
```

Si buscamos algún fenotipo, por ejemplo "alcohol" para encontrar el siguiente SNP identificado en el siguiente estudio: "*A meta-analysis of two genome-wide association studies to identify novel loci for maximum number of alcoholic drinks*", obtenemos la Figura 4, que muestra que los individuos de la población japonesa tienen una mayor proporción del SNP manifestado, mostrando una menor capacidad de metabolización del etanol.

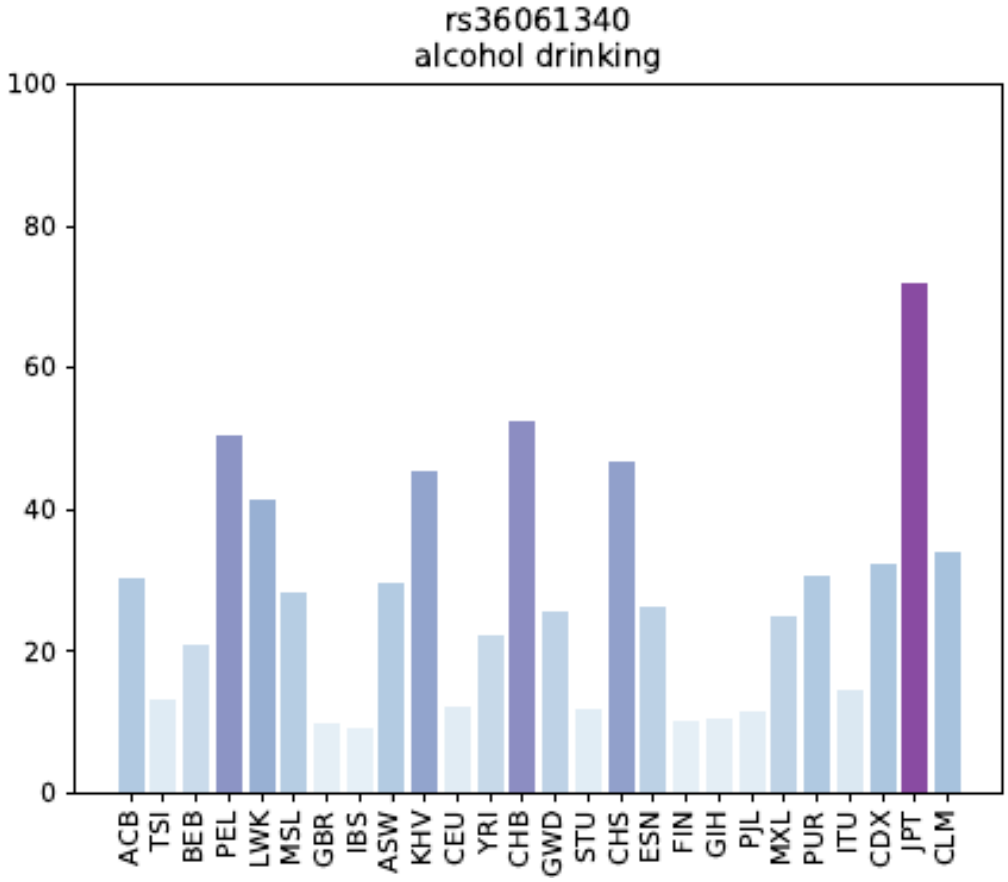

Figura 4 - Distribución del SNP rs36061340

Matplotlib

Matplotlib es una librería que se ha utilizado en la asignatura para obtener gráficas, es muy potente y nos permite generar pdf, por ejemplo, de forma sencilla (Figura 5).

```
pp = PdfPages(filename)
```

```
    for rs in prob.keys():
        x = prob[rs]
        colors = plt.cm.BuPu([float(val) / 100.0 for val in x.values()])
        p = plt.figure()
        plt.bar(x.keys(), x.values(), color=colors)
        plt.ylim((0, 100))
        plt.xticks(rotation=90)
        plt.title(rs + "\n" + encontrar_descripcion(fenotypeList, rs))
        i += 1
        pp.savefig(p)
        plt.close()
pp.close()
```

Figura 5 - Ejemplo de código python para generar un pdf con gráficas (analysis_2.py)

5.3. Buscar fenotipos relacionados a unos datos.

Utilizando los datos obtenidos en los distintos procesos de filtrado, tenemos una lista de individuos que tienen todos los SNPs activados. Con el proceso creado buscamos los otros SNPs activados (de la base de datos de GWAS). Finalmente mostramos la descripción de todos los SNPs activados por todos los individuos.

Este proceso es costoso ya que requiere revisitar la base de datos de 1000Genome, por ello podemos aplicar la misma optimización y paralelización que el *filter_1000Genome.py*.

Particularidades y problemas:

Igualmente, que en otros casos tenemos que tratar las codificaciones no estándar de los SNPs por ejemplo, con el siguiente código (Figura 6):

```
if ':' in fenotype['SNPS']:
   try:
       chromosome, position = fenotype['SNPS'].split(':')
       chromosome = chromosome.lower().split('chr')[1]
       position = position.split('_')[0]  # Casos especiales
       position = position.split('-')[0]
   except:
       continue
```

Figura 6 - Tratamiento especial de la información de GWAS. analysis_2.py

Esto nos permite extraer el cromosoma y la posición del identificador del SNP.

Funcionamiento

```
>analisis_3.py fenotypeData /home/rnou/files/ outputFile.txt
/home/rnou/files/gwas_catalog.tsv
forced expiratory volume, response to bronchodilator
pack-years measurement, systolic blood pressure
response to allopurinol, gout, uric acid measurement…
```

5.4. ML, identificación de población mediante SNPs

En el proceso de aprendizaje utilizamos como features todos los SNPs (activos o no) de una determinada población. Nos centramos en los cromosomas 1-22, preparamos los datos de entrada (una matriz de individuos x SNPs) y la guardamos en formato *pickle* (13), para acelerar las distintas pruebas posteriores.

```
if FAST:
    results = pickle.load(open("analisis3_results.p", "rb"))
else:
    results = contar_otros_snps(fenotypeList, all_fenotypes, simultani[2],
listInd, listIndY)
    pickle.dump(results, open("analisis3_results.p", "wb"))
```

Figura 7 - Ejemplo de carga y salvado con Pickle – análisis_3.py

Poder guardar estructuras internas en formato *pickle* a disco (Figura 7), nos permite repetir cálculos saltándonos algunos procesos costosos. En este caso, la creación de la matriz requiere recorrer las dos bases de datos.

Agrupamos los datos como queremos y utilizamos distintos métodos de ML (los analizaremos en el siguiente punto). Por ejemplo, utilizando scikit-learn (14), tenemos una red neuronal (Neural Network) (Figura 8) validada con *k-fold* (el método hace distintas combinaciones de particiones de los datos, entre entrenamiento y validación) que nos ofrece un método para conocer la población de un individuo utilizando sus SNPs con una precisión superior al 70%. Con las superpoblaciones, la precisión es superior al 99%.

Problemas y particularidades

Observando el rendimiento del proceso, hemos observado que scikit ya realiza una paralelización con lo cual hemos utilizado los recursos para analizar más detenidamente el proceso de ML y ofrecer un análisis de los distintos métodos disponibles y su capacidad de predicción, lo describiremos en la Sección 5.5. Análisis extendido del ML.

```
k_fold = KFold(n_splits=3, shuffle=True)
clf = MLPClassifier(solver="adam", alpha=1e-5, max_iter=40,
hidden_layer_sizes=(300, 200, 100), random_state=1, verbose=True)
results = cross_val_score(clf, JoinX, JoinY, cv=k_fold, n_jobs=-1, verbose=1)
```

Figura 8 - Código python con scikit-learn para realizar un k-fold con una red neuronal de 3 capas de 300, 200 y 100 neuronas (analisis_4.py)

5.5. Análisis extendido del ML.

El proceso creado para extraer conocimiento de los datos que tenemos, para poder averiguar con una determinada precisión de que población proviene un determinado individuo dados sus SNPs, merece un estudio más detallado.

En un primer lugar analizaremos los datos para ver si hay algún SNPs o conjunto de ellos que permite identificar de forma inequívoca a una población. Podemos utilizar dos métodos para hacerlo, uno muy visual es utilizar un árbol de decisiones para ver que “features” o SNPs son los más relevantes, pero también podemos utilizar la herramienta de selección de features incluida en scikit-learn llamada **SelectKBest**, definiendo como parámetro el número de features (en este caso SNPs) que queremos utilizar.

Miraremos algunos de los SNPs seleccionados para ver si tienen sentido (podemos utilizar el proceso que nos muestra la distribución de los SNPs en todas las poblaciones, analisis_2) y si son más relevantes en una u otra población.

Si tuviéramos problemas de rendimiento, podríamos decidir si utilizar el subconjunto reducido para las siguientes fases.

Tras esto, procederemos a analizar distintos métodos de ML para escoger el que nos ofrece más capacidad de identificación.

5.1. Árbol de decisiones

Un árbol de decisiones va dividiendo el espacio de forma binaria según el valor de las features. En nuestro caso, si el SNP está activo o no. En primer lugar, podemos construir un árbol de decisiones e inspeccionarlo para detectar features más relevantes que otras.

Uno de los parámetros para mejorar la clasificación con un árbol es la profundidad. Scikit-learn nos permite encontrar la mejor profundidad de forma sencilla, uniéndolo además con un *k-fold* para probar distintas divisiones de los datos para entrenamiento-testeo (Ejemplo en Figura 9 y Figura 10).

```
clf = tree.DecisionTreeClassifier()

parameters = {'max_depth':range(3,20)}
clf = GridSearchCV(tree.DecisionTreeClassifier(), parameters)
clf.fit(X=JoinX, y=JoinY)
tree_model = clf.best_estimator_
print (clf.best_score_, clf.best_params_)
```

Figura 9 - Código para probar distintos parámetros en un árbol de decisión. Analisis4_df.py

```
Fitting 3 folds for each of 17 candidates, totalling 51 fits
[CV] max_depth=3, score=0.195961995249, total=   7.5s
[CV] max_depth=3, score=0.194244604317, total=   7.4s
[CV] max_depth=3, score=0.195652173913, total=   9.0s
[CV] max_depth=4, score=0.275534441805, total=   9.2s
[CV] max_depth=4, score=0.270983213429, total=   9.0s
[CV] max_depth=4, score=0.271739130435, total=   8.8s
```

Figura 10 - Resultado de GridSearchCV – analisis4_df.py

Finalmente podemos imprimir el árbol (Figura 11) o generar un pdf con:

```
dot_data = tree.export_graphviz(tree_model, class_names=JoinY,out_file=None)
graph = graphviz.Source(dot_data)
graph.render("tree")
```

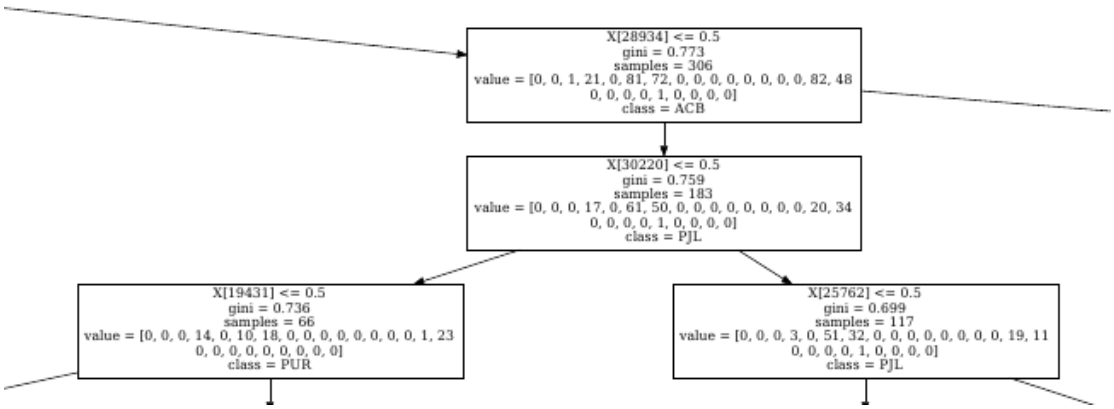

Figura 11 - Trozo del árbol de decisiones obtenido

En este caso el mejor árbol tiene una profundidad de 7, con una precisión muy baja de clasificación. Llegamos a sólo un **31%** de precisión.

Sin embargo, podemos observar algunos SNPs que dividen el espacio de poblaciones (26 en total) fácilmente:

2504 individuos: si tienen el rs2814778

= desactivado: 1793 individuos, eliminando las siguientes poblaciones (9,13,18,19,26)

= activado: 711 (de los que eliminamos 13 poblaciones, 3,4,5,6,7,10,11,12,15,16,17,22,24)

El SNP es rs2814778, que citando SNPedia[4]: "**rs2814778** is within the DARC gene, which encodes the Duffy blood group antigen [PMID 7663520]. This SNP shows an almost perfectly fixed difference in frequency between Europeans and those with African ancestry. (One exception appears to be a certain population of Czech gypsies, and certain non-Ashkenazi Jewish populations.) Additionally the Namibian San samples of the CEPH-HGDP are, uncharacteristically for Africans, all AA homozygotes for this SNP."

Además indica que "The **rs2814778** (G) allele is associated with African populations, while **rs2814778** (A) is associated with European populations and southwestern Native American populations."

Así que tenemos un SNP que diferencia poblaciones muy bien, si utilizamos este SNP en el analisis_2 (Figura 4) podemos ver la distribución poblacional y cómo se manifiesta el efecto clasificador.

[4] https://www.snpedia.com/index.php/Rs2814778

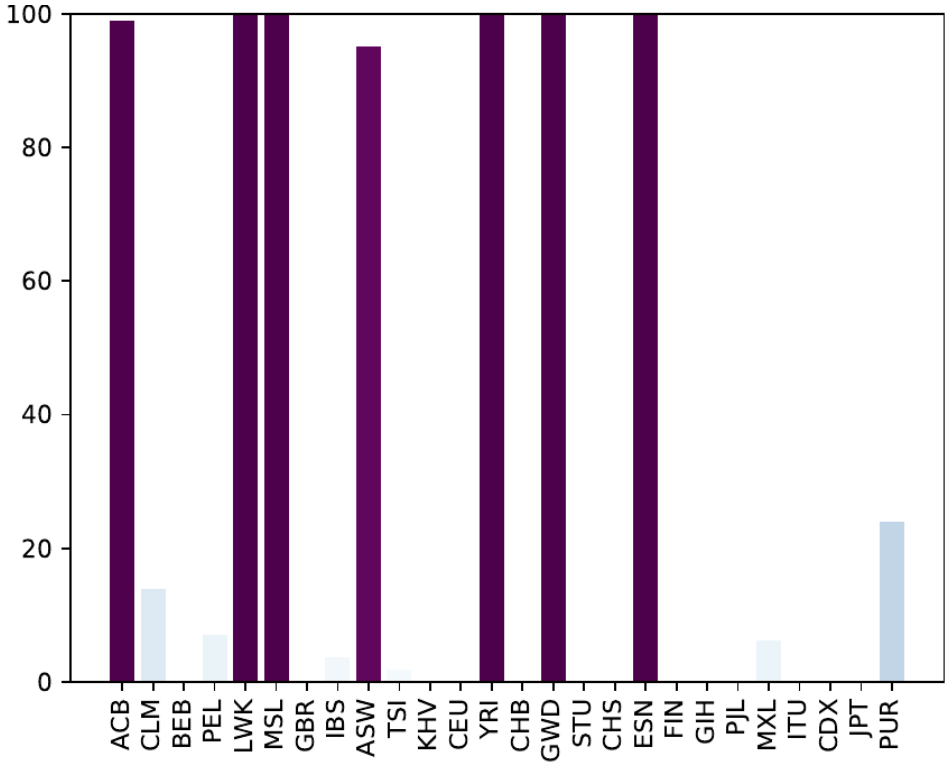

Figura 12 - Distribución del SNP rs2814778

Con el árbol de decisión hemos podido observar que podemos extraer unos cuantos SNPs que pueden llegar a identificar algunas poblaciones, sin embargo, el porcentaje de acierto es bajo.

Aunque el método puede llegar a ser potente en algunos escenarios, existe una variante llamada *random forest* que puede mejorar la predicción, utilizando varios árboles de forma simultánea y escogiendo el mejor en cada ocasión.
Pero en nuestro caso, y con los parámetros probados (Figura 13), la precisión no ha mejorado.

```
clf = RandomForestClassifier()
param_grid = {
                'n_estimators': [20, 120, 720],
                'max_depth': [5]
            }

grid_clf = GridSearchCV(clf, param_grid,verbose=6,n_jobs=-1)
grid_clf.fit(X=JoinX, y=JoinY)
tree_model = grid_clf.best_estimator_
print (grid_clf.best_score_, grid_clf.best_params_)
```

Figura 13 - Código para probar distintos parámetros de un random forest - Analisis4_rf.py

Se han probado 20, 120 y 720 árboles simultáneos de una altura de 5 con los siguientes resultados:
20: 31.1 %
120: 42.6 %
720: 47.1 %

5.2. Reducción de features con SelectKBest

Por defecto se seleccionan las *k* features que tienen un mayor f-value para la ANOVA entre la feature y la etiqueta.

```
selection = SelectKBest(k=2000)
selection.fit(JoinX, JoinY)
X = selection.transform(JoinX)
```

Hemos realizado una selección de los 50 más relevantes y hemos obtenido una lista:

rs1834640, rs6625163, rs9901616, rs2497938, rs4149433, rs1267499, rs4885150, rs4690909, rs12068879, rs11867840, rs2814778, rs11038167, rs6978230, rs16891982, rs3827760, rs11868441, rs2525776, rs8068952...

Aunque la predicción de la clasificación (si utilizamos la red neuronal mostrada en la sección anterior) es de solo un **29%**, los SNPs encontrados son los más relevantes. Si buscamos alguno de ellos obtenemos las siguientes descripciones en los artículos que los han encontrado o utilizado: "*Latitudinal Clines of the Human Vitamin D Receptor and Skin Color Genes*.", "*Male Pattern Baldness*", o la definición de la *forma del lóbulo de la oreja*.

Tras esto, probamos algunos valores diferentes de "*k*" utilizando la red neuronal presentada en capítulos anteriores, para ver cuántos SNP podemos eliminar mientras mantenemos la calidad de la predicción.

```
selection = SelectKBest(k=2000)
selection.fit(JoinX, JoinY)
X = selection.transform(JoinX)
clf = MLPClassifier(solver="adam", alpha=1e-5, max_iter=200,
hidden_layer_sizes=(400, 300, 200), random_state=1, verbose=True)

results = cross_val_score(clf, X, JoinY, cv=3, n_jobs=-1, verbose=1)
```

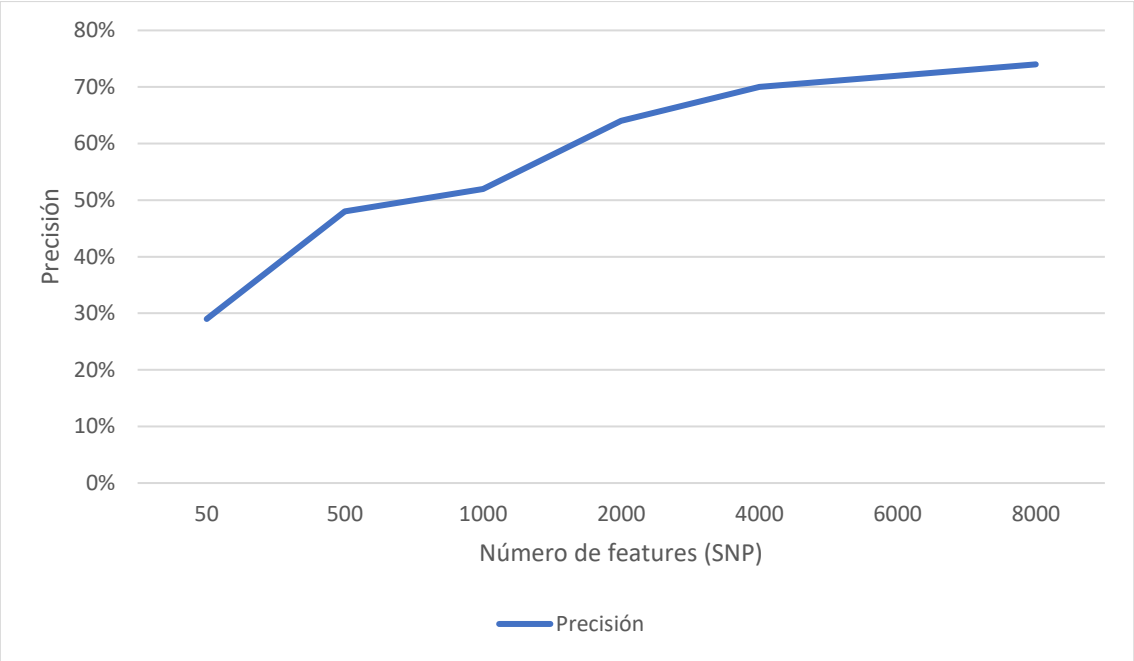

Figura 14 - Precisión de la Red Neuronal según el número de SNPs

Tal como podemos ver en la Figura 14, con 4000 SNPs obtenemos una precisión superior al **70%**.

5.3. SVM (Support Vector Machines)

Las SVM, al contrario de los árboles de decisión, debemos verlos como cajas negras. No podemos saber (de forma sencilla o lógica) como toman las decisiones. En los árboles, dados una entrada es fácil ver los nodos por los que pasa una determinada precisión.

Si intentamos predecir con SVMs tenemos algunos parámetros básicos que hemos de definir, el *kernel* y la penalización por error. Podemos hacerlo con *GridSearchCV (*Figura 15*)*.

```
parameters = {'kernel':('linear', 'rbf','poly', 'sigmoid'), 'C':[1, 10]}
clf = svm.SVC()
grid_clf = GridSearchCV(clf, parameters,verbose=6,n_jobs=-1)
```

Figura 15 - Búsqueda de parámetros para un SVM – analisis4_svm.py

Obtenemos aquí que el mejor clasificador es un SVM con un kernel *linear*, obteniendo un acierto del **82.1%**. Este valor es mayor que la red neuronal encontrada por prueba-error.

En el siguiente paso, utilizaremos *GridSearchCV* para encontrar alguna red neuronal que obtenga mejor resultado.

5.4. Red Neuronal

Las redes neuronales también son cajas negras como los SVM, además requieren de un tiempo de entrenamiento elevado (aunque luego son muy rápidas al predecir).

Como hemos de probar muchas combinaciones, limitaremos las iteraciones a 40, para reducir el tiempo de ejecución. Posteriormente con el candidato encontrado, las subiremos para encontrar la precisión final. Esto no necesariamente ha de ofrecernos la mejor alternativa, ya que algún modelo puede converger antes que otro, pero el elevado coste del entrenamiento de tantos modelos nos reduce las posibilidades de una búsqueda más exhaustiva.

Para ver la magnitud de las pruebas, la búsqueda con unos pocos parámetros nos ha llevado **2 días de ejecución** aproximadamente. Tras ello hemos encontrado unos parámetros muy similares a los que se encontraron por prueba-error anteriormente. En este caso tres layers de 300 neuronas con una precisión del **78.6%.**

5.5. Resumen de la predicción

Si colocamos juntas las precisiones obtenidos vemos en la Figura 16 que el mejor método ha resultado ser la SVM con un kernel *linear*. Esto no quiere decir que otro método no pueda ser mejor, pero sí que con el tiempo que disponemos es lo mejor que hemos encontrado. Cómo hemos visto, el número de parámetros por método es muy elevado.

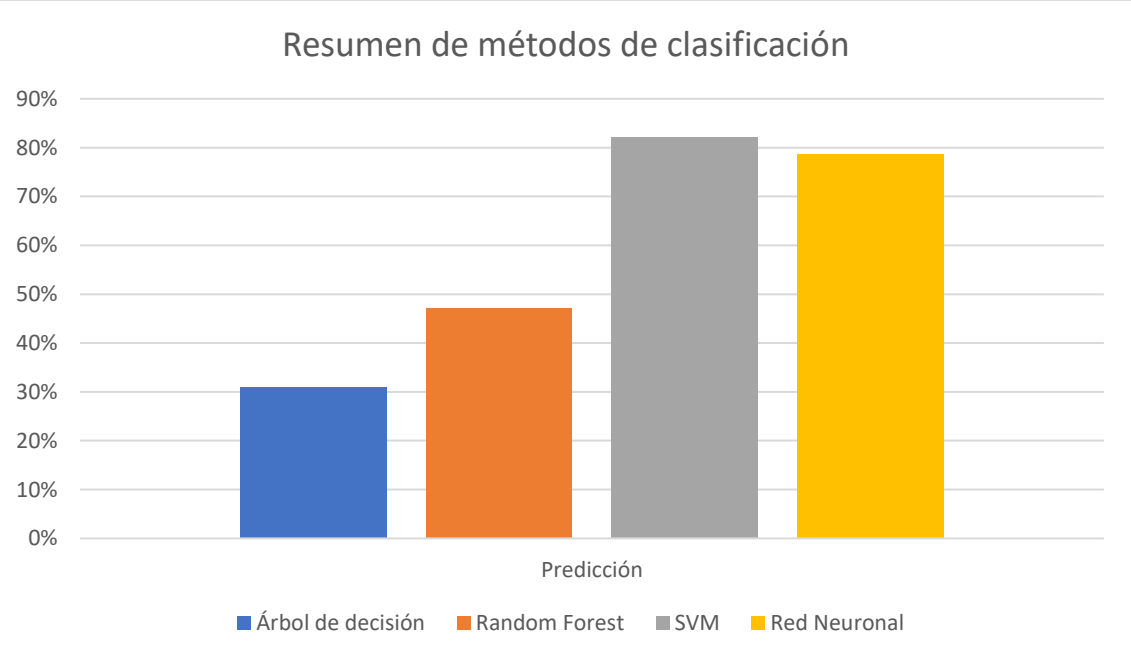

Figura 16 - Precisión con distintos métodos de ML

6. Mejora de rendimiento

Tal como hemos comentado en los primeros capítulos, la mejora de rendimiento de los códigos que se ejecutan es una parte importante. Reducir el consumo de recursos para hacer lo mismo en menos tiempo es importante para reducir costes. Por otro lado, la mejora de códigos sencillos que sólo se ejecutan una vez o que tienen un tiempo de ejecución muy pequeños ha de estar muy justificada ya que a veces puede no ser sencilla y el impacto final que tenemos es muy pequeño (Ley de Gustafson (15)).

En el caso que nos ocupa, los candidatos para mejorar el rendimiento son los siguientes:

filter_1000Genome.py y *análisis_3.py* que realizan una búsqueda en la base de datos de 1000Genome. Así como *análisis_4.py*, que es el proceso de machine learning.

En las siguientes secciones analizaremos el rendimiento de cada proceso y propondremos mejoras tanto en el plano computacional como en el del almacenamiento.

6.1. filter_1000Genome.py

El código original contiene una llamada a *filter_genome*, en la que se le pasa la lista de fenotipos seleccionados. Una posible paralelización es dividir esta llamada y juntar la salida.

```
results = filter_genome(fenotypeList, dataDir, mapping)
```

Python tiene librerías como *multiprocessing* (16) que facilitan la tarea de paralelización, pero sin embargo hemos tenido que encontrar modos para compartir los datos que se crean en cada uno de los procesos paralelos. En principio, se iban a utilizar las estructuras de *multiprocessing.Manager*, pero no han funcionado correctamente con las estructuras que ya estaban definidas. Como solución se ha utilizado una cola para guardar los resultados parciales y se han fusionado para crear el final. De este modo, no hemos tenido que cambiar demasiado el código inicial. Otras alternativas podían ser MPI4py, pero teníamos que cambiar demasiado el código y no teníamos posibilidad de probar el código en más de un nodo. Por ello utilizaremos la librería básica *multiprocessing*. Sin embargo, no nos elimina el uso del *Global Interpreter Lock* (17), que nos impide explotar al máximo dicho paralelismo aunque no sabemos el grado en el cual nos está afectando.

Para ello creamos una nueva función, **filterP**, que sólo procese los elementos que le tocan, según el número total de procesos y el su identificador.

A esta función se le ha de pasar una cola, creada con *multiprocessing*, para poder recuperar los datos de salida. Estos datos se fusionarán

(**merge_two_dicts**) en el diccionario final, para dar el mismo resultado final que el código original (Figura 17).

```
if __name__ == '__main__':
    jobs = []
    out_queue = multiprocessing.Queue()
    for i in range(procesos):
        p = multiprocessing.Process(target=filterP, args=(out_queue,
fenotypeList, dataDir, mapping, int(number), i, procesos, listInd, listIndY))
        jobs.append(p)
        p.start()

    results = dict()
    for i in range(procesos):
        a = out_queue.get()
        results = merge_two_dicts(results,a)
```

Figura 17 - Llamada paralela - filter_1000Genome_par.py

Cómo los tiempos de ejecución son rápidos, creamos una matriz de ejecuciones de 1, 2, 3 y 4 CPUs, de 1 a 16384 SNPs (en potencias de 2), generando 10 ejecuciones para tener medidas estables.

Obtenemos la Figura 19, en la que observamos como en el caso de 1 CPU, cuando llegamos a 16384 SNPs el crecimiento ya no es lineal. Esto es causado por un uso excesivo de la memoria, que provoca **swap** y **trashing**. Aunque ocurre igual con 2, 3 y 4 procesadores, el resultado no es tan notable ya que el tiempo de ejecución en el punto crítico es menor. Podemos observar como el speed up, no es de 3x o 4x por el acceso a los datos. Sin embargo, en el caso de 16384 SNPs tenemos un speed up de 2x para 2 CPUs por el efecto mencionado anteriormente.

Observando el código vemos una posible mejora en el acceso a los datos de la base de datos de 1000Genome: podemos acceder de forma ordenada y secuencial. De esta manera intentaremos tratar siempre el mismo cromosoma y evitaremos accesos de forma aleatoria.

Sin embargo, aunque sí que se observa una ligera mejora (20-50 segundos menos) no es muy grande (Figura 18) dado que el acceso aleatorio, tampoco perjudica mucho a un dispositivo no mecánico como un SSD.

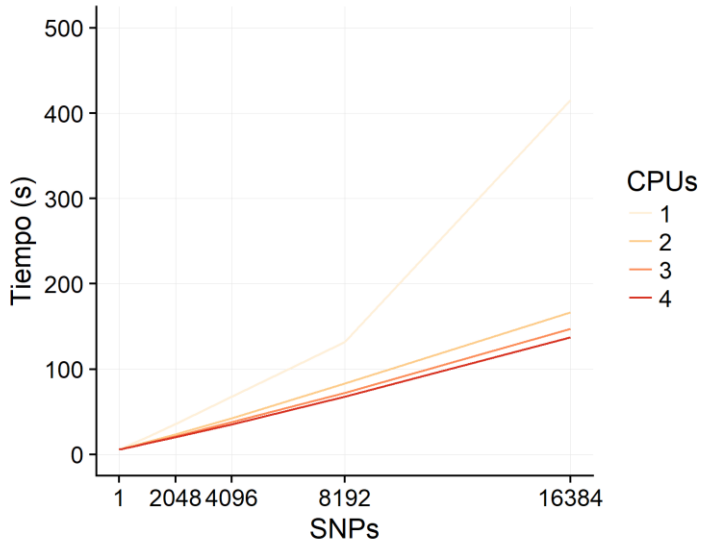

Figura 19 – filter_1000genome.py, en SSD, paralelizado, sin optimización de E/S

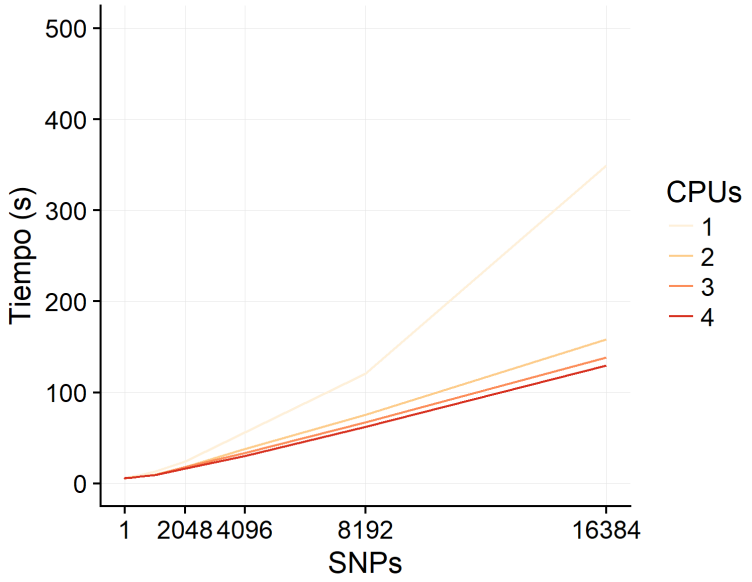

Figura 18 - filter1000_genome.py, en SSD, paralelizado, con optimización de E/S

Para probar estas mejoras, y ver que estamos en lo cierto, movemos los datos a un dispositivo con 4 HDD en RAID. Aunque el acceso es rápido, sigue siendo un conjunto de dispositivos mecánicos afectados por el tiempo de búsqueda de datos en accesos no aleatorios (*Tiempo de Acceso = Tiempo de Búsqueda + Tiempo de Lectura*).

Si repetimos el test (esta vez sólo realizamos una repetición ya que el coste es mucho mayor) sin la optimización de entrada / salida obtenemos la Figura 20.

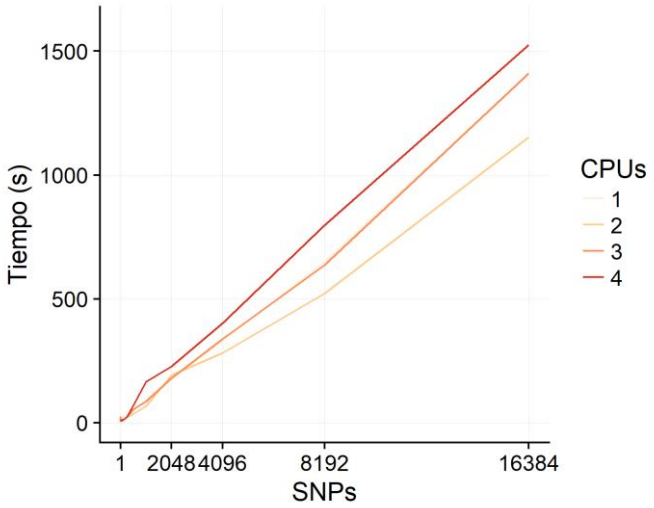

Figura 20 - filter_1000Genome sin optimización de entrada / salida (HDD)

Podemos observar como el tiempo de ejecución es igual o mayor con 3 y 4 CPUs respecto a 1 CPU (1450 segundos). Con 2 CPUs reducimos hasta los 1100 segundos. Esto es debido a que el cuello de botella es la entrada/salida y lo que conseguimos añadiendo más procesos es empeorarlo. Ahora bien, la mejora en la entrada / salida realizada nos ofrece el resultado observable en la Figura 21, en la que vemos que el tiempo de ejecución de 1 CPU se reduce hasta los 1000 segundos, mientras que la ejecución con 2 CPUs llega a 750 segundos. Tal como se observa, utilizar más procesos no tiene un efecto beneficioso en dispositivos mecánicos, ya que generamos, otra vez, patrones aleatorios que ponen de manifiesto otra vez la dificultad de recuperar datos no secuenciales.

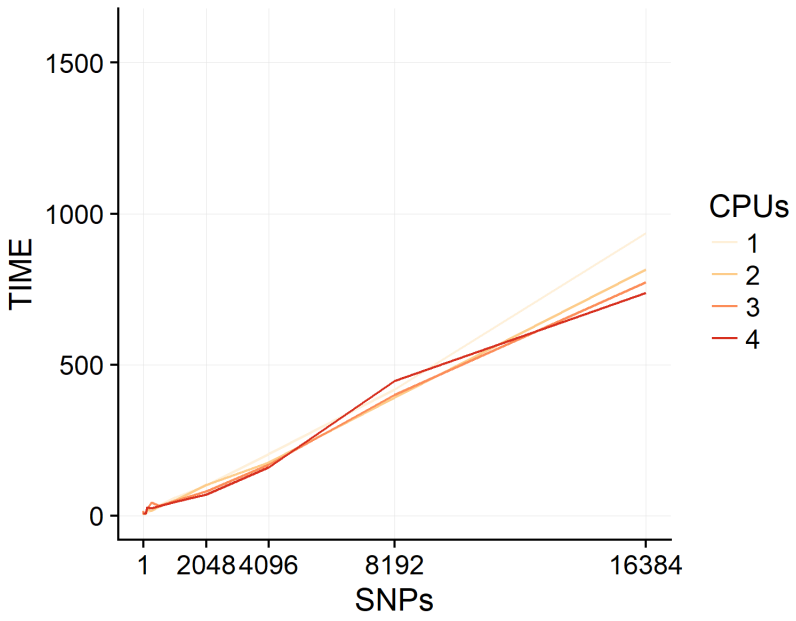

Figura 21 - filter_1000Genome paralelo con optimización de la entrada/salida (HDD)

6.2. analisis_3.py

Para el tercer proceso de análisis el resultado es similar, ya que el proceso a optimizar es muy parecido.

En este caso tenemos también gráficas en SSD sin optimizar (Figura 22) y optimizadas (Figura 23). Podemos observar que al ser el uso de memoria menor que el anterior proceso, podemos llegar a los 32768 SNPs sin llegar al punto de **trashing**.

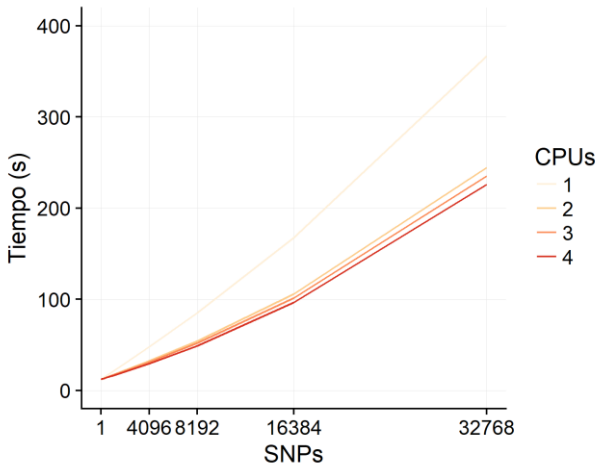

Figura 22 - análisis_3.py paralelizado sin optimización de entrada/salida (SSD)

Tal como ocurría con filter_1000Genome, tenemos un speed up elevado de 1 a 2 procesadores, pero no vemos una mejora sustancial de 2 a 4. La optimización de entrada/salida tampoco es muy notable, incluso menos que en el caso anterior.

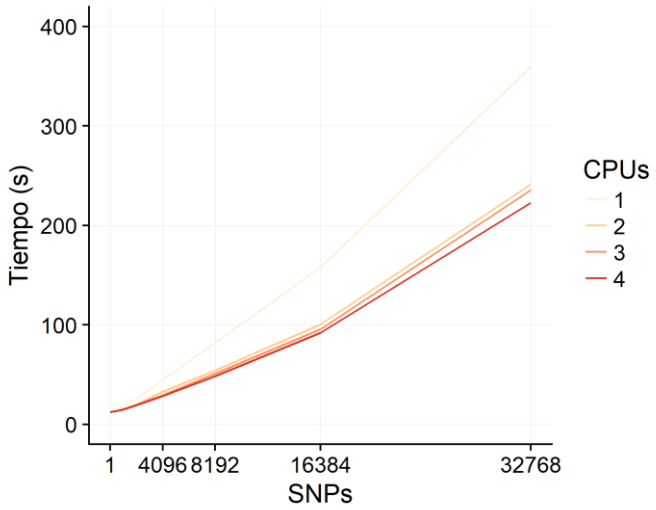

Figura 23 - analisis_3.py paralelización con optimización de entrada/salida (SSD)

Si realizamos la prueba con el dispositivo RAID con HDD, tenemos las dos gráficas: la Figura 24, sin la optimización de entrada/salida, en la que observamos cómo el coste de la E/S es mayor que la mejora de la paralelización (pasamos de 2500 a 2300 segundos de ejecución), pero sin embargo en la Figura 25 observamos como la mejora de la E/S se traduce en una reducción de más del 50% del tiempo original. Además, una ejecución más ordenada nos muestra valores de reducción de tiempo de ejecución acordes al número de CPUs utilizados (de 1500 a 1000 segundos) aunque la I/O sigue siendo teniendo un peso muy elevado en el tiempo de ejecución.

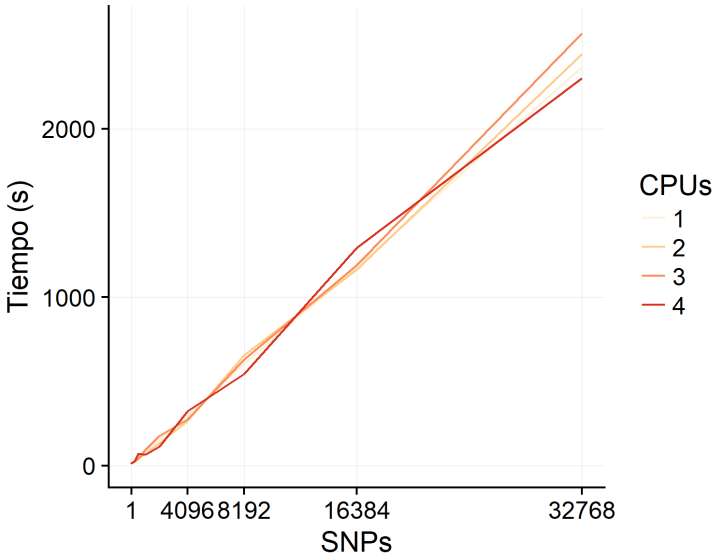

Figura 24 - analisis_3.py paralelo, sin optimizar entrada/salida sobre HDD

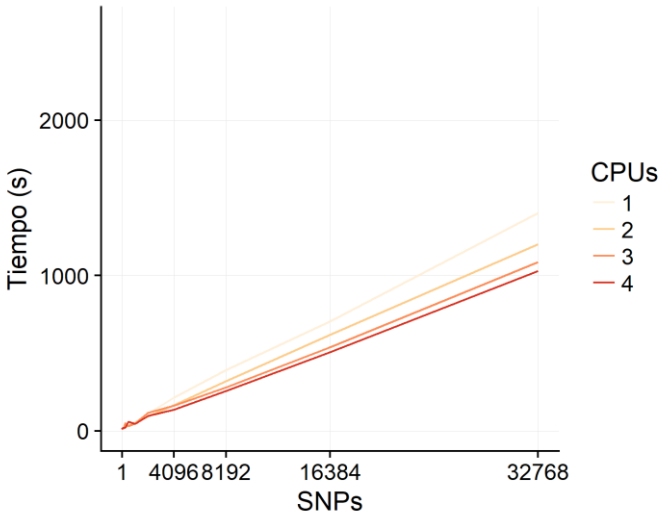

Figura 25 - analisis_3.py paralelo y optimizando la E/S (HDD)

7. Conclusiones

En este trabajo se han podido unir dos bases de datos importantes en Bioinformática. Tras crear distintos procesos hemos podido ver que los datos, aún con una tarea de homogeneización importante tienen algunas partes no homogéneas que dificultan la unión. Esto hace que el proceso de los datos requiera muchas veces un proceso de prueba y error para incorporar código que permita leer los campos no uniformes.
Tras esto hemos utilizado los distintos procesos para ver que no todo es paralelizar el código, sino que la mejora en los procesos de lectura de los datos influye de forma positiva en la optimización de los tiempos de ejecución.

Un punto positivo del trabajo, y que nos ha permitido utilizar los recursos en otras partes, es que algunas de las librerías utilizadas en Python (*scikit-learn*, por ejemplo) ya son paralelas.

En la parte más Bioinformática, hemos analizado el proceso de obtención de conocimiento e identificación de la población de los individuos utilizando sus SNPs para descubrir que algunos SNPs son más relevantes que otros y que podemos utilizar un subconjunto muy pequeño para averiguar lo que buscamos.

Tras la finalización del trabajo y según la planificación inicial, hemos cumplido todos los objetivos. Sin embargo, nos hubiera gustado probar otro tipo de paralelizaciones del código (a nivel de clúster, con *mpi4py* (18)) pero el tiempo requerido y la imposibilidad de testearlo de forma correcta nos ha llevado a realizar la paralelización dentro de un mismo nodo.

La planificación se ha seguido correctamente, y se han tenido en cuenta los periodos con más carga de trabajo personal para mover o disminuir el trabajo que se había de hacer, además, la obtención de resultados con menor coste ha permitido introducir un nuevo su objetivo que ha completado el trabajo realizado en la vertiente más biológica y de machine learning.

Como posibles trabajos futuros, podemos dibujar algunas líneas: - Explorar la paralelización con otros mecanismos (*mpi4py*, *pyCOMPSs* (19)) para ejecutar en un clúster. También podemos buscar el subconjunto mínimo de datos necesarios para encontrar la población de un individuo analizando la lista de SNPs que se obtienen con el árbol de decisión (o con otros métodos, por ejemplo, PCA).

8. Bibliografía

1. *1000 Genomes project.* **Siva, Nayanah.** 2008, NAture Publishing Group.
2. *An integrated map of genetic variation from 1,092 human genomes.* **Consortium, 1000 Genomes Project.** 7422, 2012, Nature, Vol. 491, p. 56.
3. **Burdett, T, et al.** The NHGRI-EBI Catalog of published genome-wide association studies. [En línia] 2016. www. ebiacuk/gwas.
4. **IGSR.** International Genome Sample Resource. [En línia] 2018. http://www.internationalgenome.org/about.
5. *The end of the start for population sequencing.* **Soranzo, Ewan Birney and Nicole.** 52, s.l. : Nature Publishing Group, 2015, Nature, Vol. 526.
6. **Varios.** BeautifulSoup. [En línia] 2018. https://www.crummy.com/software/BeautifulSoup/.
7. **Wikipedia.** GWAS. [En línia] 2018. www.wikipedia.com/GWAS.
8. **JetBrains.** pyCharm. [En línia] 2018. https://www.jetbrains.com/pycharm/.
9. **Li, Heng.** Tabix indexing. [En línia] 2018. http://www.htslib.org/doc/tabix.html.
10. **Slowikowski, Kamil.** pytabix software. [En línia] 2018. https://github.com/slowkow/pytabix.
11. **LiftOver. [En línia] 2018. https://genome.sph.umich.edu/wiki/LiftOver.**
12. Tretyakov, Konstantin. pyliftover. [En línia] 2018. https://pypi.python.org/pypi/pyliftover.
13. Varios. Python Pickle. [En línia] 2018. https://docs.python.org/2/library/pickle.html.
14. —. scikit-learn. [En línia] 2018. http://scikit-learn.org/stable/.
15. *Reevaluating Amdahl's Law.* Gustafson, John L. 1988, Communications of the ACM, Vol. 31.
16. Python. Multiprocessing library. [En línia] 2018. https://docs.python.org/2/library/multiprocessing.html.
17. —. Global Interpreter Lock (GIL). [En línia] 2018. https://wiki.python.org/moin/GlobalInterpreterLock.
18. mpi4py. mpi4py - python. [En línia] http://mpi4py.scipy.org/docs/.
19. *PyCOMPSs: Parallel computational workflows in Pytho.* Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M. Badia, Jordi Torres, Toni Cortes, Jesús Labarta. 1, IJHPCA , Vol. 31, p. 66-82.

9. Glosario

Fenotipo – Manifestación externa de un genotipo.
GLI - Global Interpreter Lock, Bloqueo del interprete global de Python. Muchas versiones de Python tienen un bloqueo global cuando se utilizan objetos Python que reducen las posibilidades de ejecutar el código en paralelo.
GWAS – Genome – wide association study . Análisis de una variación genética a lo largo de todo el genoma humano.
Gzip – Formato de compresión.
HDD – Disco duro magnético.
HPC – Computación de Altas prestaciones.
Machine learning (ML) – Métodos de aprendizaje automático.
Matplotlib – Librería estadística de Python, especializada en generar gráficos.
Mpi4py – Librería para ejecutar código Python como si fuera MPI. En paralelo en varios nodos.
NAS – Network Attached Storage – Almacenamiento de datos no local.
Paralelización – Posibilidad de ejecutar una tarea en paralelo para reducir el tiempo de ejecución.
PyCOMPSs – Método de paralelización de código Python automático en un clúster.
RAID – Unión de varios discos para formar uno, permite añadir paridad y/o dividir los datos en varios discos para aumentar el rendimiento.
Scikit – Librería científica de Python con métodos de machine learning.
SNPs – Single Nucleotide Polymorphism.
Speed up – Mejora del rendimiento en comparación con una línea escogida.
SSD – Disco duro sólido.
Swap – Cuando no se tiene suficiente memoria, se llevan datos al disco duro al espacio de swap. Se reduce el rendimiento.
Trashing – Si los datos que hay en el swap se utilizan muy a menudo, se produce un efecto que reduce mucho el rendimiento hasta dejar el sistema no usable.
Workflow – Flujo de trabajo, lista secuencial o paralela de tareas para conseguir un objetivo.