Uso de algoritmos de aprendizaje automático aplicados a bases de datos genéticos

Rosario Gago Utrera
Máster universitario en Bioinformática y Bioestadística

Consultor: Pau Andrio Balado

Junio de 2017

Ficha del trabajo final

Título del trabajo:	Uso de algoritmos de aprendizaje automático aplicados a bases de datos genéticos
Nombre del autor:	Rosario Gago Utrera
Nombre del consultor:	Pau Andrio Balado
Fecha de entrega:	06/2017
Titulación:	Máster universitario en Bioinformática y Bioestadística
Área del Trabajo Final:	Programación para la Bioinformática
Idioma del trabajo:	Castellano
Palabras claves:	Machine Learning, SNP, HapMap

Resumen

La cantidad de datos biológicos disponible para su análisis se ha multiplicado exponencialmente a lo largo de la última década, y sigue aumentando a un ritmo vertiginoso. Ahora se dispone de datos de naturaleza muy variada, por ejemplo: secuencias de ADN, simulaciones de dinámica de proteínas, información sobre niveles de expresión...

A medida que crecen en número y variedad la información disponible también se va dificultando el proceso para conseguir extraer información útil de ella, se hace necesario recurrir a procedimientos automatizados que intenten ayudar en la tarea de analizar los datos.

Este trabajo se ha centrado en unos datos concretos del proyecto de HapMap, acotados a nivel de cromosoma, y trabajando con datos de SNPs comunes.

Sobre estos datos se ha realizado un estudio comparativo de diferentes técnicas de aprendizaje automático con el propósito de determinar cuáles proporcionan mejores resultados, partiendo desde la optimización de la representación de los datos computacionalmente y llegando hasta la comparación de los resultados obtenidos.

Abstract

The quantity of biological data available has multiplicated exponentially during the last decade, and keeps on increasing very rapidly. Now, data of various kinds are available, such as: DNA sequences, protein dynamic simulations, expression levels information...

As the information available grows in number and variety, the process of obtaining usefull information out of it gets also more complicated and automated procedures to analyse the data become necessary.

This work has focused on some specific data from the HapMap project, enclosed in a chromosome selected, and working with SNPs common data.

A study to determine the best way of representing them to treat them with Machine Learning techniques has been carried out. The way of optimising the computational treatment has also been searched for. Some algorithms have been trained comparing the performance obtained from each of them and the results obtained have been compared.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto y Justificación del Trabajo

1.1.1. Descripción general

El trabajo fin de máster (TFM) que se presenta en este documento se ha centrado en el estudio y aplicación de técnicas de "Machine Learning" (aprendizaje automático) a datos biológicos del proyecto HapMap con el objetivo de intentar determinar que mutaciones son más significativas a la hora de agrupar los datos y realizar predicciones sobre ellos.

Se han comparado varios algoritmos para poder clasificar por población o predecir la población de un individuo dada su colección de SNPs. Los resultados obtenidos se han recogido en un informe contenido en la presente memoria. Todo el código desarrollado se ha preparado y planteado para poder tratar con datos diferentes a los que se han utilizado inicialmente.

1.1.2. Justificación del TFM

La cantidad de datos biológicos disponible para su análisis se ha multiplicado exponencialmente a lo largo de la última década, y sigue aumentando a un ritmo vertiginoso. Ahora se dispone de datos de naturaleza muy variada, por ejemplo: secuencias de ADN, simulaciones de dinámica de proteínas, información sobre niveles de expresión...

A medida que crece en número y variedad la información disponible también se va dificultando el proceso para conseguir extraer información útil de ella, se hace necesario recurrir a procedimientos automatizados que intenten ayudar en la tarea de analizar los datos.

Este trabajo se ha centrado en unos datos concretos del proyecto de HapMap, acotados a nivel de cromosoma, y trabajando con datos de SNPs comunes.

Sobre estos datos se ha realizado un estudio para determinar la mejor forma de representarlos de cara a tratarlos con técnicas de "Machine Learning". Además se ha buscado optimizar la eficiencia del código desarrollado para tratar grandes volúmenes de datos. Se han entrenado y probado varios algoritmos comparando el rendimiento obtenido por cada uno de ellos, y se han recogido y comparado los resultados en un informe.

Además ha permitido la realización de un trabajo completo, aplicando los conocimientos adquiridos durante el máster y combinándolos con la experiencia y perfil previos a su inicio.

1.2. Objetivos del Trabajo

1.2.1. Objetivos generales

Objetivo 1. Desarrollar una serie de análisis sobre los datos de HapMap para detectar qué características de estos datos son las más significativas para conseguir agrupar los datos, clasificarlos y realizar predicciones sobre estos.

Objetivo 2. Realizar una comparativa del comportamiento de diferentes algoritmos de machine learning.

Objetivo 3. Realizar un informe donde se puedan mostrar los resultados. Permitir el entrenamiento de nuevos modelos.

1.2.2. Objetivos específicos

Objetivo 1. 1.1 Familiarizarse con el proyecto de HapMap y sus datos. Seleccionar y obtener los datos del proyecto de HapMap sobre los que se va a trabajar inicialmente.

1.2 Analizar y realizar las transformaciones necesarias sobre los datos para poder aplicar técnicas de Machine Learning.

Objetivo 2. 2.1 Aplicar varios algoritmos de Machine Learning a los datos preparados, seleccionando entre todas las pruebas realizadas las que obtengan mejor resultado.

2.2 Realizar una comparación del rendimiento de los modelos generados. Se realizarán pruebas de dos tipos: clusterización y predicción.

Objetivo 3. 3.1 Recoger los resultados obtenidos en un informe.

3.2 Probar la generalización de los análisis implementados a otros datos de estructura similar.

1.3. Enfoque y método seguido

El enfoque seguido para la realización de este TFM ha quedado definido por el camino natural que marcan los objetivos principales, ya que si se observan desde un nivel de abstracción elevado se puede comprobar que el primero hace referencia a la selección, recogida y preparación de datos, el segundo a su tratamiento computacional, y el tercero a la visualización, publicación y generalización de los resultados.

Entrando en un nivel mayor de detalle, se seleccionaron datos de SNPs de individuos de varias poblaciones asociados a un cromosoma concreto elegido al azar. El origen de los datos ha sido el repositorio FTP del proyecto HapMap.

En la primera fase del trabajo se analizaron y estudiaron los datos seleccionados, realizando las transformaciones necesarias y determinando la mejor representación posible para facilitar la ejecución de algoritmos de Machine Learning.

En la segunda fase, que arrancó a continuación de la anterior y tubo como punto de inicio los datos formateados generados por aquella, se seleccionaron varios algoritmos de Machine Learning, y se entrenaron con diversos parámetros y mediante diferentes técnicas. Se realizaron múltiples pruebas, tanto de clasificación como de predicción, y se descartaron los parámetros y métodos que dieron resultados pobres, por lo tanto se puede considerar que se ha utilizado un enfoque “prueba y error”. Para trabajar con los algoritmos se ha utilizado la librería “scikit-learn” de Python, ya que se trata de unos de los principales referentes dentro de este campo.

La última fase del trabajo se centró en la presentación de los resultados obtenidos con los mejores clasificadores/predictores.

1.4. Planificación del Trabajo

1.4.1. Tareas

Los objetivos descritos en el apartado anterior se desglosaron en una serie de tareas que se presentan a continuación. Se visualizará la foto final de las tareas realizadas, ya que algunas tuvieron que ser replanificadas o surgieron durante la ejecución del proyecto sin estar previstas:

Objetivo 1.1 **Familiarización con el proyecto de HapMap:** Búsqueda de información sobre el significado, evolución y estado de este proyecto. Aunque se ha detectado que la web principal de este proyecto se cerró hace unos meses debido a temas de seguridad, y a que el proyecto ha sido superado por "1000 genomes" se ha decidido trabajar con sus datos, al estar todavía disponibles vía FTP y seguir siendo igual de válidos.

Selección los datos: Se ha trabajado con datos de la fase II de 2008-10. Seleccionando inicialmente el cromosoma 15 de todas las poblaciones.

Obtención de los datos: Los datos se han descargado desde el servidor FTP ftp://ftp.ncbi.nlm.nih.gov/, mediante un procedimiento automático de descarga implementado.

Objetivo 1.2 **Análisis descriptivo:** Familiarización con los datos concretos con los que se va a trabajar y con sus características principales.

Diseño transformación de datos: Analisis y diseño de la mejor manera de representar la información para poder utilizarla en algoritmos de machine learning.

Implementación algoritmo transformación: Implementación de las funciones que realizan la transformación de los datos. Realizada en lenguaje Python.

Objetivo 2.1 **Familiarización con la librería "scikit-learn" de Python:** Búsqueda de información y documentación sobre el funcionamiento de la principal librería utilizada para implementar técnicas de Machine Learning en Python. Esta tarea se ha realizado previamente a la implementación para disponer del conocimiento necesario para tomar las decisiones de diseño correctas. Se ha revisado la documentación oficial de la librería y se ha practicado con algunos de los tutoriales-ejemplos oficiales.

Seleccionar los algoritmos de ML a implementar: Se han elegido varios algoritmos implementados en la librería scikit-learn para trabajar con ellos.

Implementación y pruebas de clusterización y predicción:: Utilizar en lenguaje Python los diferentes algoritmos seleccionados. Para cada uno de ellos se han realizado pruebas con diferentes parámetros. Como se ha tratado de un proceso de prueba-error y bastante interactivo, se ha decidido apoyar estas fases, para mantener mejor el orden, en los notebooks de Python.

Optimización algoritmos: Al iniciar las ejecuciones con los datos completos se detectaron tiempos de proceso desbordados, que obligaron a volver a pasos anteriores para poder optimizar el código y su funcionalidad. Esta tarea no estaba prevista inicialmente, aunque ha resultado muy enriquecedora, ya que ha permitido aumentar el conocimiento en Python al tratar temas más profundos (e incluso a conocer algunas de sus limitaciones, que obligaron a descartar algunas opciones), y también ha resultado enriquecedor porque ha ayudado a la estudiante a tener más presente la principal complejidad al tratar con datos biológicos, el gran volumen de datos.

Objetivo 3.1 **Informe comparativo de resultados:** Preparar un informe comparativo con el resultado de las pruebas realizadas.

Objetivo 3.2 **Pruebas de generalización e integración:** Analizar el comportamiento de los algorimos con los nuevos datos. introducidos.

Además de las tareas descritas, propias de la naturaleza del proyecto, hay otras tareas adicionales, que se deben realizar como soporte al TFM:

Tareas de Soporte **Plan de trabajo - PEC 1:** Desarrollar el plan de trabajo.

Desarrollo del trabajo I - PEC 2: Reflejar en el documento PEC2 la evolución del trabajo.

Desarrollo del trabajo II - PEC 3: Reflejar en el documento PEC3 la evolución del trabajo.

Preparación de la memoria: Completar el documento de la memoria final.

Preparación de la presentación: Crear la presentación/resumen para la defensa del trabajo.

Preparación de la defensa: Preparar la defensa del trabajo.

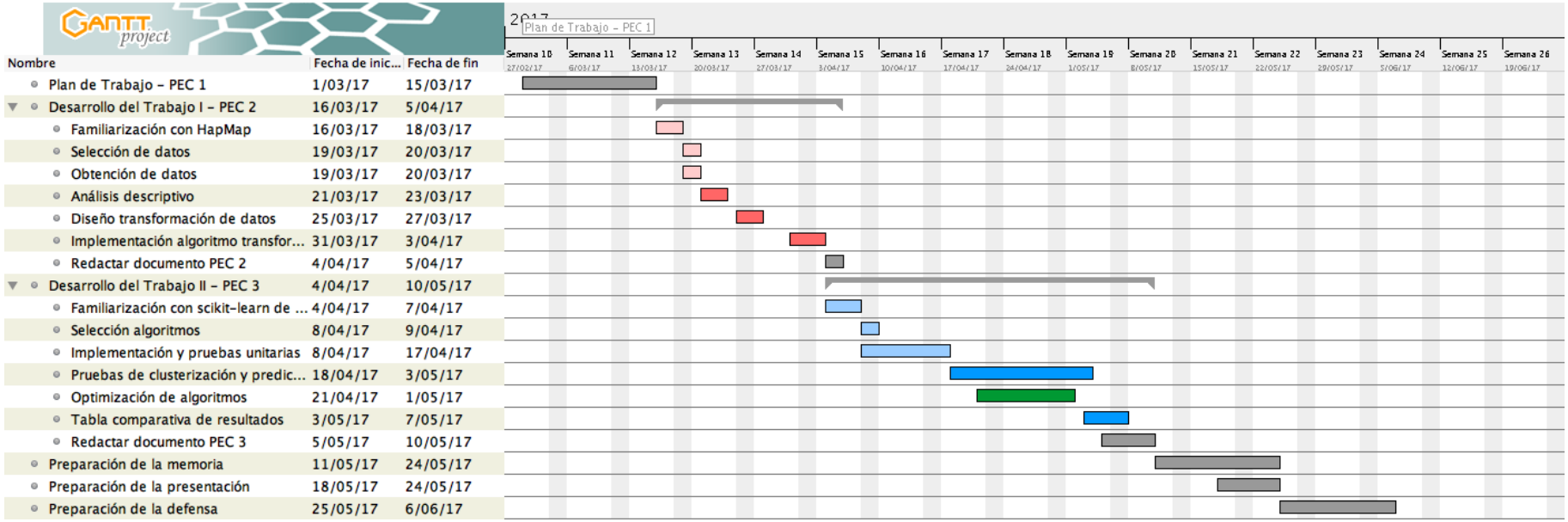
1.4.2. Calendario

A continuación se incluye el calendario de tareas de la realización del TFM.

Nombre	Fecha de inicio	Fecha de fin
Plan de Trabajo - PEC 1	01/03/17	15/03/17
Desarrollo del Trabajo I - PEC 2	16/03/17	05/04/17
Familiarización con HapMap	16/03/17	18/03/17
Selección de datos	19/03/17	20/03/17
Obtención de datos	19/03/17	20/03/17
Análisis descriptivo	21/03/17	23/03/17
Diseño transformación de datos	25/03/17	27/03/17
Implementación algoritmo transformación	31/03/17	03/04/17
Redactar documento PEC 2	04/04/17	05/04/17
Desarrollo del Trabajo II - PEC 3	04/04/17	10/05/17
Familiarización con scikit-learn de Python	04/04/17	07/04/17
Selección algoritmos	08/04/17	09/04/17
Implementación y pruebas unitarias	08/04/17	17/04/17
Pruebas de clusterización y predicción	18/04/17	03/05/17
Optimización de algoritmos	21/04/17	01/05/17
Tabla comparativa de resultados	03/05/17	07/05/17
Redactar documento PEC 3	05/05/17	10/05/17
Preparación de la memoria	11/05/17	24/05/17
Preparación de la presentación	18/05/17	24/05/17
Preparación de la defensa	25/05/17	06/06/17

Tabla 1.1: Calendario de tareas

Para visualizar de una manera más intuitiva el calendario se ha realizado un diagrama de Gantt con la temporización de las diferentes tareas. Las tareas asociadas a cada objetivo se han representado con un color diferente. Además el color gris hace referencia a las tareas de soporte del TFM.



Nombre	Fecha de inic...	Fecha de fin
Plan de Trabajo - PEC 1	1/03/17	15/03/17
Desarrollo del Trabajo I - PEC 2	16/03/17	5/04/17
Familiarización con HapMap	16/03/17	18/03/17
Selección de datos	19/03/17	20/03/17
Obtención de datos	19/03/17	20/03/17
Análisis descriptivo	21/03/17	23/03/17
Diseño transformación de datos	25/03/17	27/03/17
Implementación algoritmo transfor...	31/03/17	3/04/17
Redactar documento PEC 2	4/04/17	5/04/17
Desarrollo del Trabajo II - PEC 3	4/04/17	10/05/17
Familiarización con scikit-learn de ...	4/04/17	7/04/17
Selección algoritmos	8/04/17	9/04/17
Implementación y pruebas unitarias	8/04/17	17/04/17
Pruebas de clusterización y predic...	18/04/17	3/05/17
Optimización de algoritmos	21/04/17	1/05/17
Tabla comparativa de resultados	3/05/17	7/05/17
Redactar documento PEC 3	5/05/17	10/05/17
Preparación de la memoria	11/05/17	24/05/17
Preparación de la presentación	18/05/17	24/05/17
Preparación de la defensa	25/05/17	6/06/17

Figura 1.1: Planificación de las tareas - Diagrama de Gantt

1.4.3. Hitos

Los hitos del proyecto se han hecho coincidir con las diferentes fechas en las que hay que realizar las entregas siguiendo con el plan docente.

Estas entregas servirán además para detectar, analizar y solventar desviaciones sobre el plan previsto inicialmente.

Los hitos definidos son:

PEC 1 - Plan de trabajo 15 de Marzo

PEC 2 - Desarrollo del trabajo I 05 de Abril

PEC 3 - Desarrollo del trabajo II 10 de Mayo

Memoria y presentación del trabajo 24 de Mayo

Preparación de la defensa 06 de Junio

1.5. Breve sumario de productos obtenidos

En este TFM se han generado los siguientes entregables:

1. **Plan de trabajo**.
2. **Memoria** (este documento).
3. **Código fuente**.
4. **Presentación virtual**.

1.6. Breve descripción de los otros capítulos de la memoria

El trabajo se estructura en torno a los objetivos ya definidos en apartados anteriores, ya que estos son totalmente secuenciales (la realización de cada uno depende del resultado del objetivo anterior).

La estructura de la memoria final sigue el modelo especificado en la plantilla proporcionada por el equipo docente.

1. Introducción
   1.1 Contexto y justificación del Trabajo
   1.2 Objetivos del Trabajo
   1.3 Enfoque y método seguido
   1.4 Planificación del Trabajo
   1.5 Breve sumario de productos obtenidos
   1.6 Breve descripción de los otros capítulos de la memoria
2. El proyecto HapMap. Preparación de los datos: En este capítulo se decribe la primera fase de tratamiento de los datos, desde su selección hasta su preparación para aplicar sobre ellos algoritmos de machine learning.
3. Aplicación de algoritmos de Machine Learning: En este capítulo se describe la segunda fase, en la que se han realizado pruebas con diversos algoritmos de machine learning sobre los datos.

4. Resultados. Generalización del proceso: En este capítulo se muestran los resultados obtenidos y se realizan pruebas del proceso con otros conjuntos de datos.
5. Conclusiones
6. Anexos
7. Bibliografía

Capítulo 2

El proyecto Hapmap. Preparación de los datos

2.1. Introducción

El presente capítulo describe la primera fase del trabajo, relacionada con la obtención y preparación de los datos.

Al trabajar con cualquier tipo de datos de manera computerizada un punto clave es entenderlos, ya que si no se sabe que se tiene entre manos es posible llegar a conclusiones totalmente erróneas. Este detalle se magnifica en el entorno en el que se centra este trabajo, los datos biológicos, debido a su alta densidad y complejidad es importante dedicar un tiempo inicial a seleccionarlos y comprenderlos.

Por ese motivo el principio de este capítulo se dedicará a introducir teoricamente los datos. A continuación se describirá de una manera más práctica el trabajo que se ha realizado con ellos.

2.2. El proyecto HapMap

2.2.1. Nociones iniciales

El **Proyecto Internacional HapMap** [1, 2] fue una organización que tenía como objetivo desarrollar un mapa de haplotipos del genoma humano, para describir patrones comunes de la variación genética humana. Se utiliza para encontrar variantes genéticas que afectan a la salud, la enfermedad, las respuestas a las drogas y los factores ambientales.

En el proyecto colaboraron investigadores de centros académicos, grupos de investigación biomédica y empresas privadas de Canadá, China, Japón, Nigeria, Reino Unido y Estados Unidos. La información producida por el proyecto está disponible públicamente para investigadores de todo el mundo.

El proyecto se inició en Octubre de 2002, y los datos se publicaron en tres fases diferentes. Las dos primeras fases trabajaron con individuos de cuatro poblaciones diferentes. La última fase llegó a abarcar once grupos mundiales.

Su sitio web se cerró de forma precipitada en Junio de 2016, debido a unos problemas de seguridad informática. NCBI ya estaba planeando la retirada de la web, ya que el uso de los datos había ido disminuyendo en favor del proyecto *1000 genomes*, que se ha convertido en un nuevo estándar para la investigación de genética de poblaciones.

A pesar de ello los datos todavía se pueden descargar vía FTP desde la url `ftp://ftp.ncbi.nlm.nih.gov/hapmap/`.

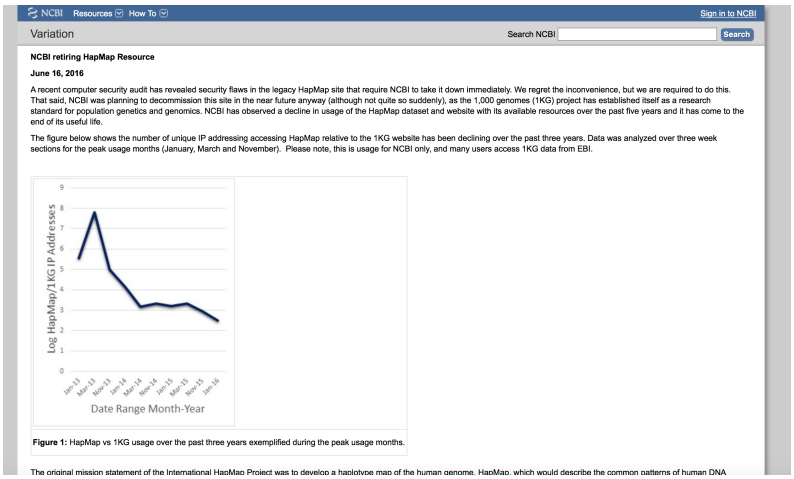


Figura 2.1: Aspecto actual del sitio web de HapMap.

2.2.2. Conceptos relacionados

Para posicionar la importancia y el alcance real del proyecto resulta útil recordar el significado de algunos términos relacionados con el mismo:

SNPs: Un polimorfismo de un solo nucleótido (SNPs) en una variación en una secuencia de ADN que afecta a una única base y que aparece en un número elevado de individuos. El proyecto HapMap se centra sólo en los SNP comunes, aquellos en los que cada alelo ocurre en al menos el 1 % de la población.

Haplotipo [3]: Consiste en un conjunto de polimorfismos de un solo nucleótido (SNPs) en un cromosoma particular que están estadísticamente asociados. Constituyen una herramienta importante en genética de poblaciones, permiten evaluar el grado de diferenciación genética entre diferentes poblaciones.

Genética de poblaciones: Es el estudio de las fuerzas que alteran la composición genética de una especie. Se ocupa de los mecanismos de cambio microevolutivo: mutación, selección natural, flujo génico y deriva génica.

2.2.3. Fases

El proyecto consta de tres fases. Cada una se orienta a la identificación de un determinado número de SNPs en el genoma. Una fase finaliza al publicar los datos conseguidos. Las fases del proyecto son:

- Fase I (2005): Se identificaron más de un millón de SNPs.
- Fase II (2007): Se identificaron dos millones de SNPs adicionales a los ya obtenidos en la fase anterior.
- Fase III (2009): Se aumentó el número de poblaciones analizadas a 11 y se identificaron 1.6 millones adicionales de SNPs.

En este trabajo se han utilizado datos relativos a la fase II del proyecto.

Población analizada Fase II

Los individuos que sirvieron de muestra para esta fase también se utilizaron en la Fase I.

El proyecto partió de poblaciones de diferente origen (africano, asiático y europeo). En total se extrajo sangre periférica de 270 personas, a partir de las cuales se obtuvo el ADN. Estas personas se clasificaron en cuatro grupos:

Nombre	Numero Individuos	Descripción
CEU	90	30 conjuntos de padres e hijo de Estados Unidos con origen europeo.
CHB	45	individuos sin relación genética de China.
JPT	45	individuos sin relación genética de Japón.
YRI	90	30 conjuntos de padres e hijo de Nigeria.

Tabla 2.1: Población analizada Fase II HapMap

2.3. Preparación de los datos

2.3.1. Selección de los datos

El proyecto HapMap, a pesar de estar descontinuado, sigue resultando muy interesante a la hora de abordar un TFM, por diversas razones:

- Son datos biológicos obtenidos de sujetos reales.
- Son accesibles libremente a la comunidad científica.
- Aunque el proyecto está cerrado los datos siguen resultando totalmente válidos.

Se ha decidido trabajar con los datos de la Fase II, y reducir el estudio a un cromosoma particular. Esta limitación se ha realizado por motivos técnicos, la cantidad de información disponible es tan extensa que es imposible abordar el proyecto en un PC doméstico sin establecer esta condición.

Se ha utilizado concretamente el **cromosoma 15**, aunque el código desarrollado se puede utilizar para trabajar con otros cromosomas de la misma fase del proyecto.

2.3.2. Obtención de los datos

Los datos se han descargado directamente (mediante un proceso semiautomático que se describirá en el apartado técnico) desde la siguiente ruta del servidor FTP de NCBI:

`ftp://ftp.ncbi.nlm.nih.gov/hapmap/genotypes/2008-10_phaseII/fwd_strand/non-redundant`

En esta ruta se puede observar que existen 5 ficheros de datos para cada cromosoma, uno por cada población descrita en la fase y otro que combina los datos de las poblaciones JPT y CHB. En este trabajo se han utilizado únicamente los cuatro ficheros que contienen información de cada población independientemente.

Los ficheros de datos son ficheros de texto comprimidos en formato gz, y su nombre sigue el siguiente patrón:

genotypes_chr<NUMBER>_<POB>_r24_nr_b36_fwd.txt.gz

Dónde:

- NUMBER: Hace referencia al número de cromosoma al que corresponden los datos.
- POB: Hace referencia a la población a la que corresponden los datos. Puede tomar cinco valores diferentes: CEU, CHB, JPT, JPT + CHB y YRI.

2.3.3. Formato de los ficheros

Los datos de cada población/cromosoma están contenidos en un fichero de texto plano comprimido en formato .gz.

Una vez descomprimidos se puede observar que el fichero está formado por una serie de campos separados por espacios.

La primera línea contiene una cabecera que identifica cada uno de los campos. El resto de líneas hacen referencia cada una a la información de un SNP concreto.

En el servidor FTP de origen de los datos se puede encontrar un fichero de documentación que explica en detalle el formato de los datos (00README.txt) y el significado de las diferentes carpetas de clasificación.

En la siguiente tabla se indica brevemente el significado de cada campo:

Posición	Nombre	Descripción
1	rs#	refSNP - Identificador del SNP en el momento de la liberación
2	SNPalleles	Alelos SNP según dbSNP
3	chrom	Cromosoma
4	pos	Posición del SNP en el cromosoma
5	strand	Cadena de la secuencia de referencia
6	genome_build	Versión de referencia del ensamblaje de la secuencia
7	center	Centro de genotipado HapMap que produjo el genotipo
8	protLSID	LSID del protocolo
9	assayLSID	LSID del ensayo
10	panelLSID	LSID del panel
11	QC_code	Código QC
12 -	NAxxxxx	Valores observados de las muestras. En la primera fila aparecen sus identificadores.

Tabla 2.2: Descripción de los campos del fichero

Se incluyen como ejemplo visual las primeras líneas de uno de los ficheros:

```
rs# SNPalleles chrom pos strand genome_build center protLSID assayLSID panelLSID QC_code NA18524 NA18526
 NA18529 NA18532 NA18537 NA18540 NA18542 NA18545 NA18547 NA18550 NA18552 NA18555 NA18558
 NA18561 NA18562 NA18563 NA18564 NA18566 NA18570 NA18571 NA18572 NA18573 NA18576 NA18577
 NA18579 NA18582 NA18592 NA18593 NA18594 NA18603 NA18605 NA18608 NA18609 NA18611 NA18612
 NA18620 NA18621 NA18622 NA18623 NA18624 NA18632 NA18633 NA18635 NA18636 NA18637
rs8028850 C/T chr15 18305964 + ncbi_b36 imsut-riken urn:lsid:imsut-riken.hapmap.org:Protocol:genotyping:1
 urn:lsid:imsut-riken.hapmap.org:Assay:8028850:22 urn:LSID:dcc.hapmap.org:Panel:Han_Chinese:1 QC+ TT TT TT
 TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT TT
 TT TT TT TT TT TT TT TT
rs12443141 A/T chr15 18309845 + ncbi_b36 perlegen urn:lsid:perlegen.hapmap.org:Protocol:Genotyping_1.0.0:2
 urn:lsid:perlegen.hapmap.org:Assay:25759.8159764:1 urn:LSID:dcc.hapmap.org:Panel:Han_Chinese:2 QC+ AT AT
 AT AT TT AT TT AT AT AT AT AA AT TT AT TT AT AT AT AT AT TT TT TT AA AT TT TT AT AA AT TT AT AT TT TT
 AT TT AA AT TT AT AT TT TT
rs10444932 A/G chr15 18314804 + ncbi_b36 perlegen urn:lsid:perlegen.hapmap.org:Protocol:Genotyping_1.0.0:2
 urn:lsid:perlegen.hapmap.org:Assay:25759.7753679:1 urn:LSID:dcc.hapmap.org:Panel:Han_Chinese:2 QC+ AA AA
 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
 AA AA AA AA AA AA AA AA AA AA
rs9328876 G/T chr15 18315211 + ncbi_b36 perlegen urn:lsid:perlegen.hapmap.org:Protocol:Genotyping_1.0.0:2
```

```
urn:lsid:perlegen.hapmap.org:Assay:25759.8148165:1 urn:LSID:dcc.hapmap.org:Panel:Han_Chinese:2 QC+ GG
GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG GG
GG GG GG GG GG GG GG GG GG GG GG GG GG GG
```

Ficheros de datos del cromosoma 15

Como se ha indicado anteriormente, se ha seleccionado inicialmente el cromosoma 15 para trabajar sobre él, por lo que se han almacenado sus datos en local.

Se incluye una pequeña tabla resumen con algunos detalles sobre sus ficheros de datos, esto ayuda a conocer mejor su volumen y significado:

Nombre	Muestras	Número de SNPs	Tamaño en bytes (comprimido)
genotypes_chr15_CEU_r24_nr.b36_fwd.txt.gz	90	106814	4179166
genotypes_chr15_CHB_r24_nr.b36_fwd.txt.gz	45	107363	3070110
genotypes_chr15_JPT_r24_nr.b36_fwd.txt.gz	45	107363	3066041
genotypes_chr15_YRI_r24_nr.b36_fwd.txt.gz	90	102431	4306272

Tabla 2.3: Ficheros de datos del cromosoma 15

2.3.4. Transformación de los datos

En los siguientes apartados se describen brevemente los pasos realizados para filtrar y transformar los datos de entrada en datos válidos sobre los que aplicar algoritmos de Machine Learning.

Selección de los datos SNPs comunes

El primer filtro que se va a aplicar es seleccionar que SNPs son comunes a todas las poblaciones que se están estudiando, es decir, aparecen en los cuatro ficheros de datos, y por lo tanto tenemos información sobre ellos para todos los individuos de la muestra.

Para el cromosoma 15 con el que se está realizando el trabajo inicialmente se han detectado un total de **99440** SNPs comunes.

El número de datos a procesar se puede comprobar que es muy elevado, se podría realizar un segundo filtro sobre estos SNPs seleccionando aquellos relacionados con alguna característica biológica concreta que sea caso de estudio, pero en este trabajo se está tratando la generalidad de los datos, para intentar obtener conclusiones a partir de ellos.

Reordenación de los datos

El objetivo del trabajo es intentar clasificar o predecir la población de un individuo a partir de sus SNPs.

La organización original de los ficheros de datos se centra en los SNPs, no en los individuos, es necesario cambiar este enfoque, de manera que al cargar los datos en memoria para trabajar con ellos se genere una matriz en la que cada fila corresponde a un individuo y cada columna a una mutación.

Codificación de la información

Además de organizar los datos en forma de matriz, la información se debe codificar de forma numérica. Esto permite homogeneizar la información y poder tratarla computacionalmente (en los ficheros los SNPs vienen representados por los alelos de cada individuo, por lo que son poco generales).

Para recodificar los datos se han asignado los siguientes valores a la posición correspondiente del individuo/SNP de la matriz:

0: Ambos alelos con el wildtype.

1: Uno de los alelos mutados.

2: Los dos alelos mutados.

Moda de su población: No está disponible la información del individuo.

2.4. Detalles técnicos. Implementación.

Hasta ahora se han descrito a grandes rasgos las acciones realizadas con los datos para prepararlos para su procesamiento, pero no se ha indicado cómo se han ejecutado. Este apartado describirá la parte técnica y las dificultades encontradas durante su realización.

Una parte relevante de este trabajo ha sido desarrollar código reutilizable, que facilite la repetición del estudio comparativo descrito en este documento con otros juegos de datos de estructura similar. Este apartado va a intentar describir cómo se ha realizado el proceso a bajo nivel para transmitir la información necesaria para reproducirlo. Esto implica que alguna información indicada es muy técnica. Si se desea realizar una lectura del informe a mayor nivel no es necesario centrarse mucho en este apartado.

2.4.1. El lenguaje Python

Python [4] es un lenguaje de programación interpretado y multiparadigma (soporta orientación a objetos, programación imperativa y programación funcional), cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Además es multiplataforma.

Es administrado por la Python Software Foundation y posee una licencia de código abierto, denominada Python Software Foundation License.

Los motivos principales por los que se ha elegido este lenguaje para realizar el trabajo son:

1. Es un lenguaje muy utilizado dentro del campo del “Machine Learning”. Posee una librería muy potente y utilizada para este tipo de tareas.

2. Se trata de un lenguaje interpretado y multiplataforma. El código desarrollado se pueden utilizar en cualquier entorno que disponga de interprete. Es interesante que los scripts obtenidos no se encuentren ligados a ninguna plataforma o sistema operativo concreto.

3. Dentro del campo de la bioinformática se trata de uno de los lenguajes de programación más utilizados.

4. También es un lenguaje muy empleado en en el procesamiento de grandes cantidades de datos (“Data Science” y “Big Data”, ya que dispone de librerias adecuadas para ello.

May 2017	May 2016	Change	Programming Language	Ratings	Change
1	1		Java	14.639%	-6.32%
2	2		C	7.002%	-6.22%
3	3		C++	4.751%	-1.95%
4	5	↑	Python	3.548%	-0.24%
5	4	↓	C#	3.457%	-1.02%
6	10	⇈	Visual Basic .NET	3.391%	+1.07%
7	7		JavaScript	3.071%	+0.73%
8	12	⇈	Assembly language	2.859%	+0.98%
9	6	↓	PHP	2.693%	-0.30%
10	9	↓	Perl	2.602%	+0.28%

Figura 2.2: Ranking languajes de programación Mayo 2017. Índice TIOBE

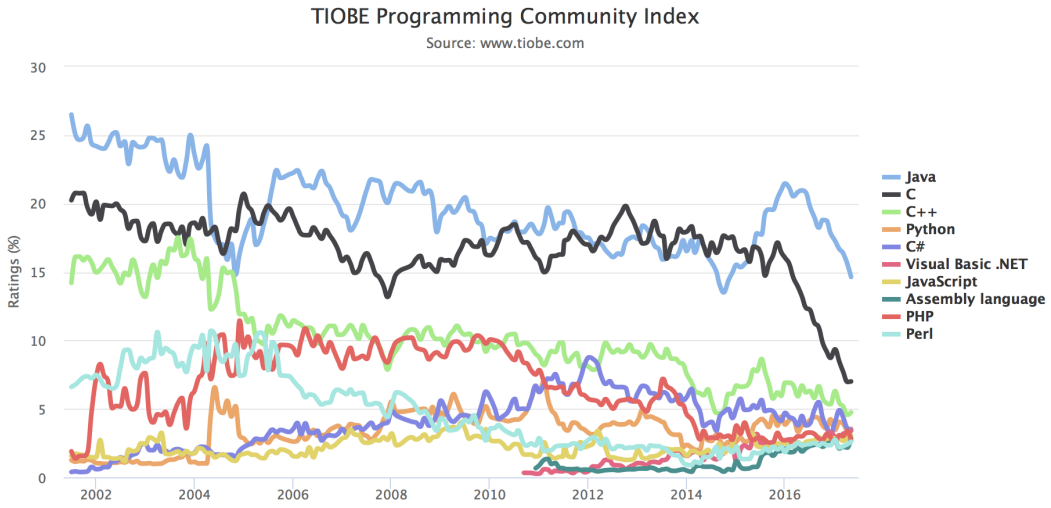


Figura 2.3: Evolución índice TIOBE 2002-2017

Evolución del lenguaje Python

Se han intentado definir varios metodos para analizar la popularidad y grado de utilización de los diferentes lenguajes de programación. Existen índices que se publican periódicamente e intentan determinar que lenguaje es el más utilizado o el más popular.

A continuación se exponen aquí dos de estos índices, en los que se observa que el lenguaje Python está entre los primeros y creciendo en popularidad, debido a su adaptabilidad a diversos campos de aplicación.

El índice TIOBE [5] es una medida de la popularidad de los lenguajes de programación. Creado y mantenido por *TIOBE Company*, se publica mensualmente desde el año 2001. Se calcula a partir de los resultados obtenidos al buscar en múltiples motores de búsqueda ciertas frases que contienen el nombre de los lenguajes a medir.

Mensualmente se genera un ranking con los 50 lenguajes más populares, y una gráfica comparativa con los 10 primeros. En la imagen 2.2 podemos visualizar los 10 lenguajes más populares según el índice TIOBE en mayo de 2017. En la imagen 2.3 podemos visualizar la evolución de la popularidad de los 10 primeros lenguajes de programación desde el 2002 hasta la actualidad.

Worldwide, May 2017 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	22.7 %	-1.3 %
2		Python	15.7 %	+3.5 %
3		PHP	9.3 %	-1.1 %
4		C#	8.3 %	-0.5 %
5		Javascript	7.9 %	+0.5 %
6		C++	6.9 %	-0.2 %
7		C	6.7 %	-0.1 %
8		Objective-C	3.8 %	-0.9 %
9		R	3.6 %	+0.4 %
10		Swift	2.8 %	-0.1 %

Figura 2.4: Ranking languajes de programación Mayo 2017. Índice PYPL

El índice PYPL (PopularitY of Programming Language) es otra medida de la popularidad de los lenguajes de programación [6], que se basa en medir la frecuencia con la que se buscan tutoriales de cada lenguaje en Google. En la imagen 2.4 podemos visualizar los 10 primeros lenguajes del ranking en mayo de 2017.

2.4.2. Organización y estructura del código

Esta primera parte del trabajo de obtención y preparación de los datos se ha planteado como un único proceso automático, que puede ser ejecutado de manera batch o desatendida. Este aspecto resulta muy interesante porque permite programar su ejecución con otros juegos de datos, para poder realizar estudios similares o de mayor alcance.

La estructura del programa se ha organizado en varios componentes con el objetivo de que sea genérico y reutilizable. Los principales elementos son:

Procesos: Dirigen la ejecución de las acciones para la obtención, transformación y preparación de datos.

Librerías: Contienen la lógica interna de los procesos implementados.

Fichero de configuración: Personaliza la ejecución de los procesos (datos a recoger, servidor FTP...)

Basándose en estos componentes, el proyecto se ha organizado en una serie de directorios, que se describen a continuación:

/: En el directorio principal se almacen los scripts que dirigen todo el proceso.

conf: Subdirectorio para almacenar ficheros de configuración

data: Subdirectorio dónde se guardan los datos de trabajo.

utils: Subdirectorio que almacena los paquetes o librerías complementarios al proceso.

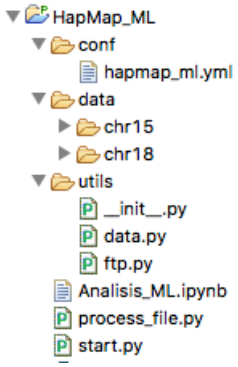


Figura 2.5: Organización y estructura del código

Fichero de configuración

Se ha creado un fichero de configuración, *hapmap_ml.yml*, escrito en formato YAML, y almacenado en el directorio de configuración. A partir de este fichero se puede modificar el comportamiento del proceso (dónde deposita los datos, de dónde los recoge, que datos trata... etc).

El YAML ('YAML Ain't Another Markup Language") [7] es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python y Perl.

Los principales parámetros que se pueden configurar en este fichero son:

Nombre	Valor actual	Descripción
data_directory	data/chr	Directorio de almacenamiento de los datos
process_subdirectory	PROCESS	Subdirectorio para almacenar los datos intermedios del proceso
ready_subdirectory	READY	Subdirectorio para almacenar los datos finales del proceso
server	ftp.ncbi.nlm.nih.gov	Servidor FTP del que se descargaran los datos
dir	hapmap/genotypes/2008-10_phaseII/fwd_strand/non-redundant	Path del servidor FTP del que se descargarán los datos
template_filename	genotypes_chr<NUM>_<POB>_	Plantilla del nombre de los ficheros a descargar del servidor FTP
poblations	CEU, CHB, JPT, YRI	Poblaciones a tratar

Tabla 2.4: Parámetros de configuración

Utilidades

La mayoría de la funcionalidad del proceso se ha implementado en dos librerías, para poder mejorar la reutilización del código.

utils/data.py

Esta librería contiene el núcleo principal del proceso. Las funciones que contiene realizan toda la parte de cálculo del proceso.

Las funciones definidas son:

Nombre	Descripción
unzip_file	Descomprime ficheros en formato gz
read_data	Lee ficheros de datos del directorio de entrada
check_data	Comprueba si existen datos intermedios preprocesados
analyze_snps	Busca los snps que se repiten en todos los ficheros de datos
process_poblation	Trata y convierte los datos de una poblacion
merge_data	Fusiona los datos parciales de cada poblacion

Tabla 2.5: Funciones librería utils/data.py

utils/ftp.py

Esta librería contiene el código encargado de conectarse con el servidor FTP y descargar los ficheros.

Tiene definida una única función:

Nombre	Descripción
getfiles	Obtiene ficheros de datos desde servidor ftp

Tabla 2.6: Funciones librería utils/ftp.py

Procesos

Para dirigir y gestionar todo el proceso automatizado se han creado dos scripts que pueden ser lanzados directamente desde la línea de comandos del sistema y recibir parámetros. Aunque solo uno de ellos gestiona el proceso principal, el otro se utiliza para realizar cálculos intermedios.

start.py

Este scripts constituye la parte principal del proceso automatizado. Puede recibir un parámetro numérico, que indica el cromosoma con el que se quiere trabajar, pero si no se especifica ninguno trabaja con el cromosoma 15.

Las principales acciones que realiza son:

- Lee el fichero de configuración YAML.
- Comprueba si se ha proporcionado un número de cromosoma para trabajar con él. En caso contrario selecciona el cromosoma 15.
- Comprueba si ya existe un directorio de datos del cromosoma a generar. Si no existe lo crea.
- Comprueba si en el directorio están descargados los ficheros fuente. En caso contrario se conecta al servidor FTP y los descarga.
- Comprueba si existen datos intermedios (procesamiento parcial de cada fichero de datos). Si no existen genera estos ficheros. Este paso es el que requiere una mayor potencia computacional. Ha sido necesario replantear el proceso para optimizar este paso.
- Comprueba si existe el directorio de resultados, en caso negativo lo crea y fusiona los resultados parciales.

El script se puede lanzar con el propósito de generar todos los pasos del proceso o sólo alguno de ellos. Para controlar este aspecto el punto clave es el directorio de datos, y la información que contiene. Si se desea rehacer los calculos se deberán borrar los resultados anteriores de dicho directorio.

process_file.py

Este scripts surgió como idea para optimizar y dividir en partes más pequeñas el proceso principal. Se encarga de procesar individualmente cada uno de los ficheros de datos para generar su matriz. Necesita recibir dos parámetros:

- Directorio de trabajo donde depositar la salida. En este directorio debe existir el fichero commons_snps.txt con los snps comunes.
- Fichero a procesar con su path completo.

2.4.3. El directorio de datos

Los datos generados por el proceso se almacenan en el directorio data. Si se desea rehacer algún cálculo o consultar resultados manualmente se puede acceder a este directorio.

Dentro del directorio hay un subdirectorio con nombre *chr<CHROMOSOME_NUMBER>*, para cada cromosoma tratado. Dentro del subdirectorio se almacenan directamente los ficheros de datos descargados del servidor FTP.

Además existen dos subdirectorios, para almacenar los datos tratados por los scripts.

El directorio PROCESS

Este directorio se utiliza para almacenar los ficheros intermedios del proceso. Si ya se han procesado los datos existirían los siguientes ficheros:

commons_snps.txt: Listado de SNPs comunes detectados.

***.INV:** Contiene los nombres de las muestras. Hay uno por cada fichero de datos.

***.POB:** Contiene la clasificación de las muestras (población a la que pertenecen). Hay uno por cada fichero de datos.

***.OK:** Contiene la matriz de relación entre individuos y SNPs. Hay uno por cada fichero de datos.

El directorio READY

Este directorio se utiliza para fusionar en ficheros únicos los datos intermedios generados. Contiene los siguientes ficheros:

SNPS.DAT: Listado de SNPs comunes detectados.

INDIV.DAT: Contiene los nombres de las muestras.

POBLA.DAT Contiene la clasificación de las muestras (población a la que pertenecen).

MATRIX.DAT: Contiene la matriz de relación entre individuos y SNPs.

Ejemplo. Contenido del directorio de datos del cromosoma 15.

Se incluye como ejemplo ilustrativo el contenido del directorio de datos del cromosoma 15:

Figura 2.6: Directorio de datos cromosoma 15

2.5. Optimización del rendimiento del proceso.

2.5.1. Introducción

Inicialmente cuando comenzó el desarrollo del TFM y de los scripts presentados, se preparó un juego de datos de pruebas muy reducido. Se trabajaba solo con tres individuos por fichero de datos (un total de 12 muestras), y solo 14 snps por fichero, de los que 10 eran comunes a todos.

Esta reducción tan drástica del volumen de datos se hizo para poder revisar manualmente que la lógica de las funciones era correcta.

Cuando se dispuso de scripts funcionales y correctos se procedió a escalar los datos al volumen total, y se descubrió que la solución implementada no era aceptable a nivel de tiempos.

Se decidió invertir esfuerzo en optimizar estos tiempos, para disponer de un proceso más funcional y reutilizable.

2.5.2. Estimación inicial de tiempos

El volumen de los datos del cromosoma 15 era el siguiente:

Nombre	Muestras	Número de SNPs
genotypes_chr15_CEU_r24_nr.b36_fwd.txt.gz	90	106814
genotypes_chr15_CHB_r24_nr.b36_fwd.txt.gz	45	107363
genotypes_chr15_JPT_r24_nr.b36_fwd.txt.gz	45	107363
genotypes_chr15_YRI_r24_nr.b36_fwd.txt.gz	90	102431

Tabla 2.7: Volumen de datos del cromosoma 15

Y el número se SNPs comunes 99440.

La primera versión del proceso se caracterizaba por:

- Generaba una gran matriz de datos de tamaño 270 x 99440.
- Los SNPs y los individuos no estaban ordenados, por lo que los tiempos de buscar su posición en la matrix no estaban optimizados.
- Los ficheros se trataban de forma secuencial uno por uno.

Con estas condiciones la velocidad estimada de procesamiento era de unas 100000 líneas de fichero de entrada cada 24 minutos. En total hay 423971 líneas a tratar, por lo tanto el tiempo de procesamiento estimado era de unos 102 minutos, algo más de una hora y media.

2.5.3. Primera etapa de optimización. Optimización de código

Se realizó inicialmente una optimización del código desarrollado, basada en eliminar acciones que se realizaban varias veces (guardando en una variable temporal la primera ejecución y reutilizándola), y en ordenar los SNPs con lo que mejoró el tiempo de búsqueda.

Esta segunda versión se caracterizaba por:

- Código optimizado para evitar ejecutar acciones múltiples veces.
- Generaba una gran matriz de datos de tamaño 270 x 99440.
- Los SNPs y los individuos están ordenados, por lo que los tiempos de buscar su posición en la matrix están optimizados.
- Los ficheros se trataban de forma secuencial uno por uno.

Con estas condiciones los tiempos se redujeron a la mitad. La nueva velocidad estimada de procesamiento era de unas 100000 líneas de fichero de entrada cada 12 minutos. En total hay 423971 líneas a tratar, por lo tanto el tiempo de procesamiento estimado era de unos 51 minutos.

2.5.4. Segunda etapa de optimización. Paralelismo

Los resultados obtenidos seguían sin ser muy eficientes. Hoy en día hasta los PCs domésticos disponen de procesadores multinúcleo y multiproceso, capaces de ejecutar varios procesos simultáneamente.

El siguiente paso natural en la optimización de tiempos era intentar paralelizar el tratamiento de los ficheros, para poder sacar mejor provecho de la potencia computacional disponible.

Threads en Python. GIL

La primera opción que se intentó abordar fue adaptar el script de Python para que utilizara threads que se pudieran procesar en paralelo.

Las pruebas realizadas tras este intento no fueron satisfactorias, no se consiguieron mejorar los tiempos.

Tras investigar las causas se descubrio que era debido al funcionamiento de la implementación de Python que se estaba utilizando. CPython es la implementación más extendida de dicho lenguaje, y es también la más rápida de las existentes en la mayoría de los benchmarks exceptuando aquellos en los que se usan múltiples hilos en sistemas que disponen más de un procesador o procesadores con más de un núcleo.

Esto es debido a que esta implementación de CPython solo permite que un único thread ejecute bytecode a la vez por lo que se pierde la potencia de los sistemas SMP, para controlar esta exclusión se utiliza el *Global Interpreter Lock* también conocido como *GIL* [8].

Por lo tanto aunque se estaban creando varios hilos con el objetivo de que se ejecutaran en paralelo en los núcleos del procesador, el intérprete de python evitaba esta acción.

Subprocesos en Python

Se decidió probar una segunda opción, desde el script principal de Python realizar llamadas al sistema, mediante la creación de subprocesos asíncrono. Cada uno de estos procesos ejecuta un nuevo intérprete de Python y el script process_file.py.

El script principal se queda a la espera (un tiempo máximo de 45 minutos), realizando comprobaciones cada minuto para ver si el resto de subprocesos han finalizado.

Con esta solución se consiguió tener los cuatro subprocesos del procesador trabajando en paralelo, aunque el rendimiento no se multiplicó por cuatro, ya que el equipo no tenía más núcleos de ejecución disponibles, por lo tanto estos debían compartirse con otros procesos del sistema. El tiempo conseguido en procesar las 423971 líneas con este sistema se redujo a unos 16 minutos. Consiguiendo el objetivo de mejorar considerablemente los tiempos desde la primera versión del proceso.

La versión final implementada destaca por las siguientes características:

- Código optimizado para evitar ejecutar acciones múltiples veces.
- Los SNPs están ordenados, por lo que los tiempos de buscar su posición en la matrix están optimizados.
- Dos scripts de python, uno controla la ejecución y otro realiza cálculos específicos de cada fichero de datos.
- Los cuatro ficheros de datos se tratan de forma paralela.
- Se generan cuatro matrices de menor tamaño, una por fichero, que se almacenan en disco y posteriormente se fusionan.

Capítulo 3

Aplicación de algoritmos de Machine Learning

3.1. Introducción

El presente capítulo describe la segunda fase del trabajo, relacionada con las pruebas de diversos algoritmos de machine learning sobre los datos preparados anteriormente.

El enfoque de esta fase ha sido muy práctico, basado en el ensayo y en el mecanismo de prueba-error. Se han utilizado herramientas interactivas para facilitar su realización y ordenación.

Este capítulo comienza con una breve introducción teórica a las técnicas de aprendizaje automático. A continuación se entra en materia, explicando y mencionando los algoritmos concretos seleccionados. Por último, al igual que se hizo en el capítulo anterior, se mencionan los aspectos técnicos e informáticos destacables de esta fase.

3.2. Machine learning

3.2.1. Nociones iniciales

El *Machine Learning* [9, 10] es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a los computadores aprender, es decir, generalizar comportamientos y conocimientos a partir de una información suministrada en forma de ejemplos.

El *Machine Learning* puede ser visto como un intento de automatizar algunas partes del método científico mediante métodos matemáticos. Algunos sistemas intentan eliminar toda necesidad de intuición o conocimiento experto de los procesos de análisis de datos, aunque sigue siendo necesario siempre un diseñador del sistema que especifique la forma de representación de los dátos y los métodos de manipulación. Otros sistemas tratan de establecer un marco de colaboración entre el experto y la computadora.

3.2.2. Problemas de apredizaje automático

En general, un problema de aprendizaje automático utiliza un conjunto de n muestras de datos para intentar predecir propiedades de los datos desconocidos.

Cada uno de los números o datos de los que se dispone para cada muestra se denominan atributos o

características.

Estos problemas se suelen clasificar en dos categorías:

Aprendizaje supervisado: Los datos en estos casos disponen de atributos adicionales que son los que se intentan predecir. Dentro de esta categoría destacan los algoritmos de *clasificación*, en los que las muestras están etiquetadas como pertenecientes a dos o más clases, y se quiere aprender a predecir la clase de datos sin etiquetar.

Aprendizaje no supervisado: Los datos de entrenamiento consisten en un conjunto de vectores de entrada sin ningún valor o etiqueta correspondiente. El objetivo en estos casos puede ser descubrir grupos de ejemplos similares dentro de los datos (clustering).

En este trabajo se han probado algoritmos de las dos clases principales indicadas.

3.3. Selección de algoritmos

3.3.1. Aprendizaje supervisado. Algoritmos de clasificación

A continuación se describen brevemente los algoritmos seleccionados para realizar las pruebas.

El algoritmo kNN

El algoritmo **k-NN** (K nearest neighbors) es un método de clasificación supervisada, basado en el entrenamiento mediante ejemplos cercanos en el espacio de los elementos.

Su funcionamiento básico es clasificar los elementos asignándoles la clase que poseen elementos considerados similares. Para ello se comprueba la clase a la que pertenecen los k objetos más cercanos (analizados durante la fase de entrenamiento del algoritmo), clasificando el elemento en la clase más repetida en estos k objetos.

A pesar de su aparente simplicidad, estos métodos resultan en la práctica bastante potentes y se obtienen buenos resultados con ellos. Por este motivo se ha seleccionado para la realización de pruebas.

Fortalezas y debilidades

En la siguiente tabla se resumen las fortalezas y debilidades de este algoritmo, que a pesar de ser uno de los más simples, sigue siendo ampliamente utilizado.

Fortalezas	Debilidades
- Simple y efectivo	- No produce un modelo, limitando la habilidad para entender como se relacionan las características con las clases
- No hace suposiciones sobre la distribución de los datos	- Requiere seleccionar un valor para k apropiado
- Fase de entrenamiento rápida	- Fase de clasificación lenta
	- Características nominales y datos perdidos requieren procesamiento adicional

Regresión logística

Aunque su nombre puede llevar a confusión, la **regresión logística** es un modelo lineal que se utiliza para clasificación, no para regresión. También se le conoce como clasificador de máxima entropía (MaxEnt).

Es un tipo de análisis utilizado para predecir el resultado de una variable categórica (una variable que puede adoptar un número limitado de categorías) en función de las variables independientes o predictoras. Es útil para modelar la probabilidad de un evento ocurriendo como función de otros factores.

Support Vector Machines (SVM)

SVM son un conjunto de algoritmos de aprendizaje supervisado relacionados con problemas de regresión y clasificación.

Una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita). Una buena separación entre las clases permitirá un clasificación correcta.

Fortalezas y debilidades

En la siguiente tabla se resumen las fortalezas y debilidades de este algoritmo.

Fortalezas	Debilidades
- Se puede utilizar para problemas de clasificación o de predicción numérica	- Encontrar el mejor modelo requiere realizar varias pruebas de diversas combinaciones kernel/parámetros
- No está muy influenciado por los datos “ruidosos” y no es propenso al overfitting	- Puede ser lento de entrenar, sobre todo si los datos de entrada tienen muchos ejemplos o parámetros
- Mas fácil de utilizar que una red neuronal	- Resultados difíciles (o imposibles) de interpretar (caja negra)
- Está ganando popularidad gracias a su alta precisión y a sus buenos resultados en concursos de “data mining”	

Árboles de decisión

Los **árboles de decisión** son clasificadores potentes, que utilizan un estructura de árbol para modelar la relación entre las características del modelo y los potenciales resultados.

Un punto interesante de estos modelos es que la estructura generada no se emplea únicamente para uso interno del algoritmo, algunas implementaciones permiten obtener el modelo construído (árbol) con la relación entre las características.

Fortalezas y debilidades

En la siguiente tabla se resumen las fortalezas y debilidades de este algoritmo.

Fortalezas	Debilidades
- Simple de entender e interpretar. Se pueden visualizar lo árboles construidos. - No es necesario preparar mucho los datos.	- Se pueden generar árboles muy complejos, que no generalizan bien los datos. - Pequeñas variaciones en los datos producen grandes cambios en la estructura generada.

Fortalezas	Debilidades
- Puede manejar datos numéricos y categóricos.	- El problema de encontrar el árbol óptimo está clasificado como NP-completo, por lo tanto se usan métodos heurísticos para resolverlo.
- El coste de utilizar el árbol para realizar predicciones es logarítmico.	

Random Forest

Los **Random Forest** se basan en la construcción de una serie de árboles predictores, con un subconjunto de los datos, para después promediar los resultados obtenidos.

El algoritmo funciona bien con muchas clases de problemas, puede manejar datos con ruido, y seleccionar únicamente las características más importantes. Son adecuados para datos con un gran número de características.

Su principal debilidad es que el modelo que producen no es fácilmente interpretable.

3.3.2. Aprendizaje supervisado. Algoritmo de clustering

En este caso se va a probar un único algoritmo, pero con diversos parámetros.

K-means

K-means [11] es un método de clustering, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano. Ei.

El problema es computacionalmente difícil (NP-completo), por lo tanto para poder abordarlo se deben recurrir a heurísiticas que permitan converger más rápidamente a un óptimo local.

3.4. Detalles técnicos. Implementación

Esta segunda parte del trabajo se ha basado también en el lenguaje Python, pero desde otra orientación muy diferente, ya que se ha tratado de un proceso de prueba-error, por lo tanto se necesitaba una herramienta que permitiera interactividad.

3.4.1. Notebook de iPython

Como herramienta básica para esta fase del trabajo se ha empleado el *Notebook* de iPython [12], que se caracteriza por basarse en una interfaz web y ser interactivo.

IPython es un shell interactivo que añade funcionalidades extra al modo interactivo incluido con Python. Es un componente del paquete SciPy.

El Notebook de IPython es un documento JSON que contiene una lista ordenada de entradas/salidas las cuales pueden tener código, o datos de diferentes formatos. Una característica importante es que pueden ser convertidos a otros formatos de archivos como HTML, LaTeX, PDF, Python, etc.

En el trabajo se ha utilizado el notebook convinando anotaciones en Markdown y código en Python, con el objetivo de ser directamente exportable y constituir un informe de los resultados.

3.4.2. Scikit-learn

Scikit-learn [13, 14]es una librería de software libre para el lenguaje de programación Python. Incluye algoritmos de aprendizaje automático de diversos tipos: clasificación , regresión y agrupación. Además está preparada para interactuar con las bibliotecas numéricas y científicas Python NumPy y SciPy.

La versión estable en la actualidad es la 0.18, liberada en septiembre de 2016.

Esta librería ha demostrado ser muy potente y completa, constituyendo un pilar importante en la aplicación de técnicas de machine learning en Python.

Además de ser una librería extensa también es destacable la documentación de la que dispone, con muchos ejemplos y tutoriales. Se puede consultar en la propia web oficial, `http://scikit-learn.org/stable/documentation.html`.

Debido a todas las características mencionadas, se ha decidido utilizar esta librería para la realización de esta fase del proyecto.

Procedimiento aprendizaje supervisado

Para realizar las pruebas de aprendizaje supervisado se ha utilizado un procedimiento automático, que ha probado los cinco algoritmos seleccionados y ha realizado una predicción con los dos que han proporcionado mejor resultado.

Este procedimiento se ha basado en:

- Los conjuntos de entrenamiento y de validación se han generado aleatoriamente, de manera que el juego de datos de validación contiene el 20 % de las muestras del conjunto total. Los mismos datos se han utilizado para probar con todos los algoritmos.
- Para realizar las pruebas se ha utilizado validación cruzada de 10 iteraciones. La validación cruzada o cross-validation es una técnica que se aplica para evaluar los resultados de un análisis estadístico y garantizar que son independientes de la partición entre datos de entrenamiento y prueba. Consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones, con el objetivo de estimar la precisión del modelo al llevarlo a la práctica.
- Como estimador de la precisión de los modelos generados se ha utilizado la métrica *accuracy* o *exactitud*, que muestra mejores resultados cuanto más se acerca a 1.
- Se realizaron algunas pruebas de ejecución de estos algoritmos tras realizar algunas pruebas con selección de features, pero se descartaron los resultados por ser peores o similares a los obtenidos al utilizar todo el conjunto de datos.

Procedimiento aprendizaje no supervisado

Las pruebas de aprendizaje no supervisado se han centrado en el algortimo K-means.

Se han realizado tres ejecuciones del algoritmo con diferentes parámetros, de cada una de las ejecuciones se ha calculado una serie de métricas:

k-means++: Selecciona los centros iniciales usando heurísticas para acelerar la convergencia.

ramdom: Elige observaciones aleatorias como centroídes iniciales.

Basado en PCA: Se ha reducido la dimensión de las observaciones utilizando PCA.

Capítulo 4

Resultados. Generalización del proceso

4.1. Introducción

En este capítulo se describe los resultados obtenidos y como se puede extender el proceso aplicado a otros conjuntos de datos.

4.2. Resultados Cromosoma 15

El cromosoma 15 fue el elegido para realizar el estudio inicial y la preparación de los pipeline's y procesos de análisis. En este punto se muestran los resultados obtenidos.

4.2.1. Algoritmos de clasificación

Se han probado los cinco algoritmos seleccionados utilizando validación cruzada de 10 iteraciones. Se ha empleado la "accuracy" o "exactitud" como métrica del funcionamiento del algoritmo.

En la siguiente tabla se muestran los resultados obtenidos para cada uno de los algoritmos. Para poder compararlos se ha utilizado la media y la desviación estándar de las iteraciones ejecutadas:

Algoritmo	Media	Desviación
KNN	0.962987	0.039787
LR	0.985714	0.030491
SVM	0.787662	0.070301
TREE	0.837879	0.056489
FOREST	0.865801	0.062741

Tabla 4.1: Resumen clasificación cromosoma 15

Se muestran también los resultados en la siguiente gráfica:

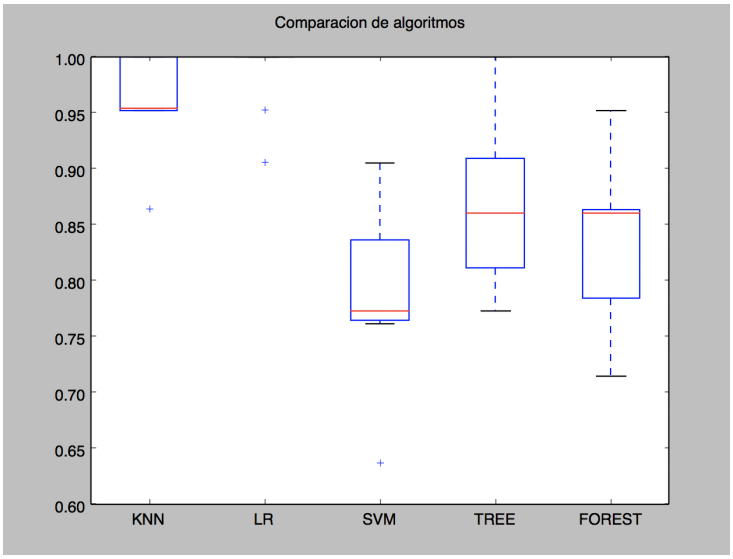


Figura 4.1: Resultados clasificación cromosoma 15

Los mejores resultados, con unas valores muy buenos, los han proporcionado los algoritmos de Regresión logística y KNN. Los peores resultados se han obtenido con el algoritmo SVM.

Las cifras que se observan para los árboles de decisión y los bosques aleatorios son las conseguidas en la ejecución reflejada en el documento. Debido al funcionamiento de estos algoritmos los valores exactos varían en cada prueba, pero se encuentran en un rango muy similar, dando unos resultados intermedios a nivel de calidad.

4.2.2. Algoritmos de agrupación

Como algoritmo de agrupación se ha utilizado k-means, pero con diferentes parámetros ya descritos en el capítulo anterior. En la siguiente tabla se muestran los resultados obtenidos:

Opciones	Tiempo	Inertia	Homo	Compl	V-meas	ARI	AMI	silhouette
k-means++	13.59s	5594390	0.826	0.833	0.829	0.730	0.824	0.085
random	12.01s	5594885	0.826	0.835	0.831	0.733	0.824	0.086
PCA-based	1.60s	5603480	0.827	0.828	0.827	0.861	0.825	0.089

Tabla 4.2: Resumen agrupación cromosoma 15

Los resultados de las tres pruebas han manifestado unos parámetros muy similares, excepto en los tiempos de ejecución, dónde ha destacado el algoritmo basado en realizar una reducción de la dimensionalidad mediante PCA.

4.2.3. Resumen

En el cromosoma 15 de detectaron un total de 99440 SNPs comunes, que se han utilizado para intentar clasificar y clusterizar los datos.

Inicialmente parecía preocupante el comportamiento que iban a tener los algoritmos utilizados con un número tan elevado de features, sobretodo a nivel de tiempos de computación, pero los resultados obtenidos han sido muy buenos.

En los algoritmos supervisados de clasificación, las mejores puntuaciones se han obtenido con los algoritmos de regresión logística y KNN. Con este último destaca la buena puntuación obtenida a pesar de tratarse de un algoritmo bastante sencillo, queda claro que a pesar de su idea simple su potencia y aplicabilidad son elevadas.

En las pruebas de aprendizaje no supervisado también se han obtenido buenos resultados en tiempos razonables. Destaca bastante la mejora obtenida en tiempos al aplicar la reducción de features mediante PCA, sin afectar al resto de métricas.

4.3. Generalización del proceso

4.3.1. Trabajando con el cromosoma 18

Uno de los objetivos de este TFM era intentar crear un proceso genérico que se pudiera aplicar también a otros datos de naturaleza y formato similar a los originales.

Con esta idea se creó todo el código reutilizable y con la opción de customizar el cromosoma a utilizar.

Se han realizado pruebas del proceso utilizando el cromosoma 18.

Para repetir el proceso con este cromosoma los pasos realizados fueron básicamente:

1. Lanzar el proceso batch de recogida y preparación de datos, a través del script *start.py* proporcionándole como parámetro el número 18.

2. Iniciar el notebook de iPython del estudio de machine learning, y modificar al principio en una variable el número del cromosoma a utilizar.

3. Exportar el informe de resultados y analizar la información obtenida.

4.3.2. Algoritmos de clasificación

En la siguiente tabla se muestran los resultados obtenidos para cada uno de los algoritmos. Para poder compararlos se ha utilizado la media y la desviación estándar de las iteraciones ejecutadas:

Algoritmo	Media	Desviación
KNN	0.911688	0.048735
LR	0.981818	0.022268
SVM	0.787662	0.070301
TREE	0.879221	0.083426
FOREST	0.856710	0.055740

Tabla 4.3: Resumen clasificación cromosoma 18

Se muestran también los resultados en la siguiente gráfica:

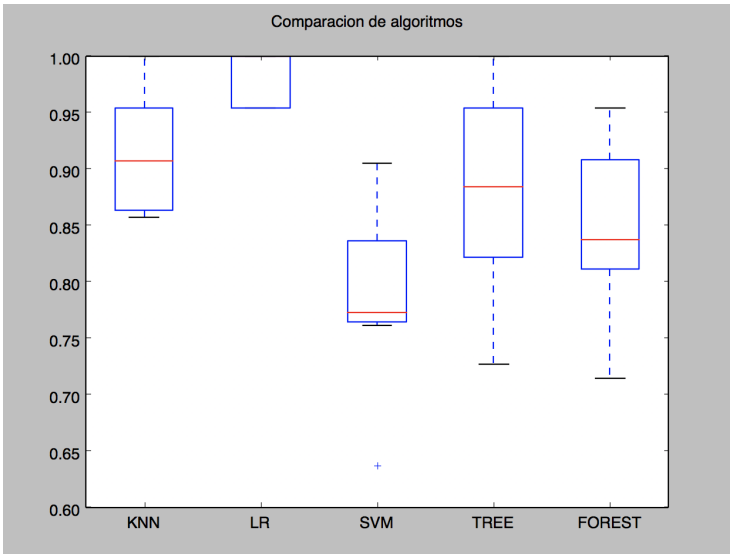


Figura 4.2: Resultados clasificación cromosoma 18

Los mejores resultados coinciden con los obtenidos con el cromosoma anterior, los algoritmos de Regresión logística y KNN. Lo mismo ocurre con los peores resultados y los valores intermedios.

Se puede observar que el rendimiento de todos los algoritmos ha sido muy similar con ambos conjuntos de datos.

4.3.3. Algoritmos de agrupación

Como algoritmo de agrupación se ha utilizado k-means, pero con diferentes parámetros. En la siguiente tabla se muestran los resultados obtenidos:

Opciones	Tiempo	Inertia	Homo	Compl	V-meas	ARI	AMI	silhouette
k-means++	15.88s	5944178	0.826	0.842	0.834	0.743	0.824	0.066
random	14.47s	5947275	0.826	0.833	0.829	0.730	0.824	0.066
PCA-based	1.87s	5960472	0.834	0.835	0.834	0.869	0.832	0.069

Tabla 4.4: Resumen agrupación cromosoma 18

Los resultados de las tres pruebas han manifestado unos parámetros muy similares, excepto en los tiempos de ejecución, dónde ha destacado el algoritmo basado en realizar una reducción de la dimensionalidad mediante PCA.

4.3.4. Resumen

En el cromosoma 18 de detectaron un total de 110859 SNPs comunes, que se han utilizado para intentar clasificar y clusterizar los datos.

En los algoritmos supervisados de clasificación, al igual que ocurrió con el cromosoma 15, las mejores puntuaciones se han obtenido con los algoritmos de regresión logística y KNN.

En las pruebas de aprendizaje no supervisado también se han obtenido buenos resultados en tiempos razonables.

Resulta destacable el hecho de que los resultados han sido bastante similares en ambos cromosomas (a nivel de

qué modelo funciona mejor), es posible que se haya encontrado un buen método de clasificación para estos datos, que podría servir de punto de partida para la construcción de una aplicación de predicción.

Capítulo 5

Conclusiones

Es este capítulo se van a exponer, desde un punto de vista personal y subjetivo, las lecciones aprendidas durante la realizacion de este TFM. Además se propondrán posibles evoluciones o mejoras que se podrían aplicar sobre los desarrollos realizados.

5.1. Objetivos, planificación y metodología

Considero que los objetivos generales de este proyecto se han alcanzado de manera satisfactoria. Se ha conseguido trabajar y obtener modelos buenos de "Machine Learning" para trabajar con datos de HapMap, y se ha generado un proceso reutilizable.

La planificación ha necesitado ser adaptada a lo largo del desarrollo del proyecto, ya que en la primera fase se encontró una dificultad mayor de lo esperada, los tiempos de proceso.

Se decidió adaptar el alcance del proyecto, centrando más atención de la prevista inicialmente en solucionar estos problemas de rendimiento, en lugar de invertir dicho tiempo en mejorar la visualización de los resultados.

La metodología planificada inicialmente se ha podido seguir sin problemas, por lo que ha demostrado ser adecuada.

5.2. Lecciones aprendidas

A nivel personal este trabajo ha servido para asentar y ampliar conocimientos de "Machine Learning" y del lenguaje de programación Python, así como de experiencia dentro del campo de la bioinformática, al trabajar con datos biológicos reales. Un aspecto clave que ha permitido asimilar es el gran volumen de datos de los que se dispone, y la gran potencia computacional necesaria para poder tratarlos, incluso en un caso como el que se ha trabajado, en el que se partía de acotar los datos a un único cromosoma para poder asumirlo su ejecución con un portatil doméstico.

El conocimiento adquirido mediante la realización de este TFM ha superado mis expectativas. Su realización me ha permitido:

- Agrupar y poner en práctica una buena parte de los conocimientos adquiridos durante la realización de mi Máster Universitario en *Bioinformática y Bioestadística*. Los ladrillos que han ido construyendo las asignaturas se han asentado y encontrado su relación gracias a este trabajo.
- Trabajar desde el inicio con datos biológicos reales.

- Entender la dificultad que conlleva tratar con este tipo de información debido a su gran volumen.
- Incrementar mis conocimientos sobre las técnicas de Machine Learning.
- Mejorar mi soltura trabajando con Python, y sus tecnologías asociadas.
- Ampliar y reforzar mis conocimientos de LaTeX, ya que ha sido el lenguaje empleado para escribir esta memoria.

5.3. Posibles líneas de trabajo futuras

El conocimiento y código obtenido en este trabajo constituye un primer paso inicial de acercamiento y entendimiento de los datos de HapMap.

A partir de las pruebas realizadas, las conclusiones obtenidas y de parte del código ya desarrollado, se podría crear una aplicación para realizar clasificación de individuos.

También se podría mejorar la visualización de los resultados, aspecto que ha tenido que reducirse en este trabajo por centrar la atención y el tiempo en aspectos de rendimiento.

Otra posible ampliación de este trabajo sería realizar un estudio comparativo similar extendido a todos los cromosomas. En las pruebas realizadas sobre dos cromosomas diferentes algunas técnicas han destacado por sus buenos resultados. Un estudio en esta línea serviría para analizar si este comportamiento se puede generalizar.

Apéndice A

El entorno tecnológico

A.1. Descripción del hardware

Los procesos definidos y realizados en este TFM han evidenciado la gran potencia computacional necesaria para tratar con datos biológicos.

Para la realización de este trabajo se ha utilizado un ordenador portatil doméstico, teniendo en cuenta las limitaciones que ello supone. Se especifican aquí sus principales características:

Modelo: Macbook Pro (13-inch, Late 2016, Four Thunderbolt 3 Ports)

Sistema Operativo: macOS Sierra Versión 10.12.5

Procesador: Intel Core i5 Skylake (6267U) 2.9 GHz dual-core

Núcleos del procesador: 2

Número de subprocesos: 4

Memoria RAM: 16 GB 2133 MHz LPDDR3

Disco duro: 256 GB SSD

A.2. Herramientas software

A continuación se van a describir brevemente las herramientas software utilizadas para la realización de este TFM.

Eclipse: Entorno de desarrollo integrado libre y multiplataforma. Se ha utilizado como herramienta para la programación de los scripts, incluyendo las fases de escritura de código, compilación y ejecución.

PyDev: Plugin para eclipse que permite trabajar con el lenguaje Python.

Dropbox: Servicio de alojamiento de archivos multiplataforma en la nube, operado por la compañía Dropbox. Utilizado como repositorio de datos del proyecto.

TeXWorks: Entorno de edición y trabajo con LaTeX libre y multiplataforma, distribuido bajo la licencia GPL. Utilizado para escribir la documentación y la memoria.

GIMP: Programa de edición de imágenes digitales libre y gratuito. Forma parte del proyecto GNU. Es el programa de manipulación de gráficos disponible en más sistemas operativos (Unix, GNU/Linux, FreeBSD, Solaris, Microsoft Windows y Mac OS X, entre otros). Empleado para editar y realizar transformaciones de formato en las imagenes utilizadas para la documentación de los productos desarrollados y para la memoria.

Apéndice B

Código fuente - Preparación y transformación de datos

En este anexo se incluyen los listados del código fuente generado durante la primera fase de este TFM.

B.1. Script principal - start.py

```
import utils.ftp
import utils.data
import yaml
import os
import numpy as np
import subprocess
import shlex
import time
import sys

# Se lee el fichero de configuracion con las opciones.
with open("conf/hapmap_ml.yml", 'r') as stream:
    app_config = yaml.load(stream)

# Se selecciona un cromosoma por defecto.
chromosome_number = 15

# Se puede especificar un cromosoma diferente como parametro.
if len(sys.argv) == 2:
        chromosome_number = sys.argv[1]
else:
        print "Este programa necesita dos parametros";
        sys.exit(5)


dataDir = app_config['data_directory'] + str(chromosome_number) + "/"
processDir = dataDir + app_config['process_subdirectory'] + "/"
readyDir = dataDir + app_config['ready_subdirectory'] + "/"

# Se comprueba si existe el directorio de datos, en caso contrario se crea.
if (os.path.exists(dataDir)):
```

```
    print "Existe directorio dataDir."
else:
    print "No existe directorio dataDir. Se crea."
    # Se crea el directorio de datos
    os.mkdir(dataDir)

# Se comprueba si hay archivos de datos para procesar, en caso contrario se descargan.

if len(os.listdir(dataDir)) == 0:
    print "No hay archivos, se descargan desde el servidor FTP."
    utils.ftp.getfiles(app_config, chromosome_number, dataDir)
else:
    print os.listdir(dataDir)

if not utils.data.check_data (dataDir, processDir):
    data = utils.data.read_data (dataDir)

    commons_snps = utils.data.analyze_snps (data)
    commons_snps.sort()
    print len(commons_snps)

    txt_snps = '\n'.join(commons_snps)
    archivo = open(processDir + "commons_snps.txt", "w")
    archivo.write(txt_snps)
    archivo.close()

    for pob in data:
        cmds = "python process_file.py "+ processDir + " " + dataDir + pob[0]
        args = shlex.split(cmds)
        subprocess.Popen(args)

    minutos = 0
    while not utils.data.check_data (dataDir, processDir) and minutos < 45:
        minutos = minutos + 1
        time.sleep(60)
        print "Procesando datos..."


if not os.path.exists(readyDir):
    "Fusionando datos..."
    os.mkdir(readyDir)
    utils.data.merge_data (processDir, readyDir)

print "Proceso finalizado."
```

B.2. Script de proceso de ficheros - process_file.py

```
import sys
import utils.data
import numpy as np

################################################################################
#
# Script que procesa cada uno de los ficheros de datos para generar su matrix.
# Necesita recibir dos argumentos:
```

```
#       - Directorio de trabajo donde depositar la salida. En este directorio
#         debe existir un fichero commons_snps.txt con los snps a recoger en
#         la matrix.
#       - Fichero a procesar con su path completo.
#
#===  FUNCTION  ================================================================

def process_data(pro_dir, filename):
    with open(process_dir + "commons_snps.txt", "r") as ins:
        data_snps = ins.read()
        snps = data_snps.split("\n")

    file_out_template = process_dir + filename.split("/")[-1]
    (data, names, pobs) = utils.data.process_poblation (filename, snps)


    txt_names = '\n'.join(names)
    archivo = open(file_out_template + ".INV", "w")
    archivo.write(txt_names)
    archivo.close()

    pob_names = '\n'.join(pobs)
    archivo = open(file_out_template + ".POB", "w")
    archivo.write(pob_names)
    archivo.close()

    np.savetxt(file_out_template + ".OK", data, fmt='%d')


if len(sys.argv) == 3:
    process_dir = sys.argv[1]
    file = sys.argv[2]
    print process_dir + "␣" + file
    process_data (process_dir, file)

else:
    print "Este␣programa␣necesita␣dos␣parametros";
    sys.exit(5)
```

B.3. Librería utils - ftp

```
from ftplib import FTP

#===  FUNCTION  ================================================================
#         NAME:  getfiles
#      PURPOSE:  Obtiene ficheros de datos desde servidor ftp
#   PARAMETERS:  conf
#                                   cromosoma
#                datadir
#      RETURNS:  none
#  DESCRIPTION:  Conecta al servidor FTP y obtiene los ficheros de un cromosoma
#                concreto
#       THROWS:  no exceptions
#     COMMENTS:  none
#     SEE ALSO:  n/a
```

```
#=============================================================================
def getfiles (conf, cromosoma, datadir):
        # Realizamos una conexion FTP

        ftp = FTP(conf['server'])
        ftp.login()             # user anonymous

        ftp.cwd(conf['dir'])
        contador = 1

        for pob in conf['poblations']:
                files = ftp.nlst(conf['template_filename'].replace("<NUM>",str(cromosoma))
                for file in files:
                        print str(contador) + "␣" + file
                contador = contador + 1
                dest_file = datadir+file
                ftp.retrbinary('RETR␣'+file, open(dest_file,'wb').write)
        ftp.quit()
```

B.4. Librería utils - data

```
import gzip
import os
import numpy as np
import shutil


#===  FUNCTION  ================================================================
#          NAME:  unzip_file
#       PURPOSE:  Descomprime ficheros en formato gz
#    PARAMETERS:  file
#       RETURNS:  Contenido descomprimido del fichero
#   DESCRIPTION:  Descomprime un fichero en formato gz cuyo nombre se ha
#                 recibido como parametro, y devuelve su contenido
#        THROWS:  no exceptions
#      COMMENTS:  none
#      SEE ALSO:  n/a
#=============================================================================

def unzip_file (file):
    fr = gzip.open(file, 'rb')
    data = (fr.read().decode("utf-8"))
    fr.close()
    return data


#===  FUNCTION  ================================================================
#          NAME:  read_data
#       PURPOSE:  Lee ficheros de datos del directorio de entrada
#    PARAMETERS:  dir
#       RETURNS:  Array con ficheros + contenido descomprimido
#   DESCRIPTION:  Lee todos los ficheros de datos del directorio de entrada y
#                 los descomprime
#        THROWS:  no exceptions
#      COMMENTS:  none
```

```
#      SEE ALSO:  n/a
#===============================================================================
def read_data (dir):
    files_list = os.listdir(dir)

    data = []

    for file in files_list:
        if os.path.splitext(file)[1] == ".gz":
            data.append([file, unzip_file(dir + file)])

    return data


#===  FUNCTION  ================================================================
#         NAME:  check_data
#      PURPOSE:  Comprueba si existen datos intermedios preprocesados
#   PARAMETERS:  datadir, processdir
#      RETURNS:  true or false
#  DESCRIPTION:  Comprueba si existen datos intermedios preprocesador en el
#                directorio correspondiente, y si estan completos. Si faltan
#                datos limpia el directorio y devuelve false.
#       THROWS:  no exceptions
#     COMMENTS:  none
#      SEE ALSO:  n/a
#===============================================================================
def check_data (datadir, processdir):
    if not os.path.exists(processdir):
        os.mkdir(processdir)
        return False

    files_list = os.listdir(datadir)

    allFiles = True
    for file in files_list:
        name, ext = os.path.splitext(file)
        if ext == ".gz":
            if not os.path.isfile(processdir + file + ".OK"):
                allFiles = False

    return allFiles


#===  FUNCTION  ================================================================
#         NAME:  analyze_snps
#      PURPOSE:  Calcula los snps comunes
#   PARAMETERS:  data
#      RETURNS:  Array con el nombre de los snps encontrados
#  DESCRIPTION:  Busca los snps que se repiten en todos los ficheros de datos
#       THROWS:  no exceptions
#     COMMENTS:  none
#      SEE ALSO:  n/a
#===============================================================================
def analyze_snps (lista_datos):
    commons = set()
    for pob in lista_datos:
```

```
        snps = []
        lines = pob[1].split(os.linesep)
        del(lines[-1])
        contador = 1
        for line in lines:
            if (contador != 1):
                snps.append(line.split()[0])
            contador = contador + 1
        if len(commons) == 0:
            commons = set (snps)
        else:
            tmp = set(snps)
            commons = commons & tmp
    return list(commons)


#===  FUNCTION  ================================================================
#         NAME:  process_poblation
#      PURPOSE:  Trata y convierte los datos de una poblacion
#   PARAMETERS:  filename, snps
#      RETURNS:  Matriz de datos, individuos, clasificacion
#  DESCRIPTION:  Construye la matriz de datos a partir de los SNPs comunes
#                y el fichero de datos de la poblacion
#       THROWS:  no exceptions
#     COMMENTS:  none
#     SEE ALSO:  n/a
#===============================================================================
def process_poblation (filename, snps):
    pob_name = filename.split("/")[-1].split("_")[2]

    lines = unzip_file(filename).split(os.linesep)
    del(lines[-1])
    names = lines[0].split()[11:]

    matrix_pob = np.full ((len(names),len(snps)),-1)
    contador = 1

    for line in lines[1:]:
        # Se construye el diccionario para traducir el SNP
        snp, snpalleles = line.split()[0:2]
        al1 = snpalleles[0]
        al2 = snpalleles[2]
        values = {}
        values[al1+al1] = 0
        values[al1+al2] = 1
        values[al2+al1] = 1
        values[al2+al2] = 2
        values[u'NN'] = -2
        try:
            position = snps.index(snp)
            trad_tmp = []
            for i in range (0, len(names)):
                trad_tmp.append(values[line.split()[11+i]])
                matrix_pob[i][position] = trad_tmp[i]

            if trad_tmp.count(-2) > 0:
```

```
                # Calculo de la moda
                moda = -2
                max = -1
                for j in range (0,3):
                    #print "Moda analizo valor: " + str (j)
                    if trad_tmp.count(j) > max:
                        max = trad_tmp.count(j)
                        moda = j

                for k in range (0, len(trad_tmp)):
                    if trad_tmp[k] == -2:
                        matrix_pob[k][position] = moda

        except Exception, ex:
            pass

        contador = contador + 1

    list_pob = []
    for i in range(len(names)):
        list_pob.append(pob_name)


    return (matrix_pob, names, list_pob)


#=== FUNCTION ================================================================
#         NAME: merge_data
#      PURPOSE: Fusiona los datos parciales de cada poblacion
#   PARAMETERS: datadir, mergedir
#      RETURNS: none
#  DESCRIPTION: Recoge los ficheros de cada poblacion del fichero de datos
#               y los fusiona y almacena en el fichero de salida
#       THROWS: no exceptions
#     COMMENTS: none
#     SEE ALSO: n/a
#==============================================================================
def merge_data (datadir, mergedir):

    files_list = os.listdir(datadir)

    shutil.copy (datadir + "commons_snps.txt", mergedir + "SNPS.DAT")

    individuos = []
    poblaciones = []
    matrix = []

    allFiles = True
    for file in files_list:
        name, ext = os.path.splitext(file)
        if ext == ".OK":
            print "Analizando: " + file
            with open(datadir + name + ".INV", "r") as ins1:
                data_inv = ins1.read()
                tmp_inv = data_inv.split("\n")
                individuos = individuos + tmp_inv
```

```
            with open(datadir + name + ".POB", "r") as ins2:
                data_pob = ins2.read()
                tmp_pob = data_pob.split("\n")
                poblaciones = poblaciones + tmp_pob

            tmp_mat = np.loadtxt(datadir + file)
            if len(matrix) == 0:
                matrix = tmp_mat
            else:
                matrix = np.concatenate((matrix, tmp_mat), axis=0)

    txt_names = '\n'.join(individuos)
    archivo = open(mergedir + "INDIV.DAT", "w")
    archivo.write(txt_names)
    archivo.close()

    pob_names = '\n'.join(poblaciones)
    archivo = open(mergedir + "POBLA.DAT", "w")
    archivo.write(pob_names)
    archivo.close()

    np.savetxt(mergedir + "MATRIX.DAT", matrix, fmt='%d')
```

B.5. Fichero de configuración YAML - hapmap_ml.yml

```
# Fichero YAML de configuracion de los parametros para los scripts


# Directorio de datos
data_directory: data/chr

# Subdirectorio de procesamiento
process_subdirectory: PROCESS
ready_subdirectory: READY

#################################
#              FTP              #
#################################
# Servidor Origen de los ficheros
server: ftp.ncbi.nlm.nih.gov

# Directorio de datos
dir: hapmap/genotypes/2008-10_phaseII/fwd_strand/non-redundant

# Plantilla ficheros a recoger
template_filename: genotypes_chr<NUM>_<POB>_

# Listado de poblaciones:
poblations:
 - CEU
 - CHB
 - JPT
 - YRI
```

Apéndice C

Informes completos iPython

En este anexo se incluyen los informes completos generados a partir de iPython, obtenidos mediante la opción de exportar a formato PDF el Notebook utilizado para la realización de las pruebas de algoritmos de "Machine Learning".

C.1. Cromosoma 15

A continuación se incluye el informe completo de las pruebas realizadas con el cromosoma 15.

Analisis_ML CHR15

1 ANÁLISIS MACHINE LEARNING

1.1 Introducción

Este documento va a servir para realizar pruebas de ejecucion de diversos algoritmos de Machine Learning sobre el proyecto de HapMap.

Para realizar el informe se utilizan datos preprocesados previamente.

1.2 Preparación y lectura de datos

Se importan las librerías que se van a utilizar para realizar el informe.

```
In [1]: import os
        import numpy as np
        import matplotlib.pyplot as plt
        from time import time

        from sklearn import model_selection
        from sklearn.feature_selection import VarianceThreshold
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import chi2
        from sklearn import metrics
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import adjusted_rand_score
        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.svm import SVC
        from sklearn.cluster import KMeans
        from sklearn.cluster import MeanShift
        from sklearn.cluster import DBSCAN
        from sklearn.decomposition import PCA
```

El primer paso es indicar el número del cromosoma sobre el que se quiere realizar el estudio. Por defecto se trabajará con el cromosoma 15.

```
In [2]: # Cromosoma a utilizar para el informe, sus datos ya se
        # han depositado y preparado en la ruta indicada.
        cromosoma = 15
```

Se configura el directorio por defecto del que se recogeran los datos y se listan los ficheros para comprobar que están todos. Los ficheros esperados son:

- INDIV.DAT: Contiene los nombres identificativos de cada individuo.
- POBLA.DAT: Contiene el código de población correspondiente a cada individuo.
- SNPS.DAT: Contiene los nombres de los SNPS comunes a todas las poblaciones.
- MATRIX.DAT: Contiene los de SNPS x Individuos. Puede tomar valores de 0, 1 y 2.

```
In [3]: data_dir = "data/chr" + str(cromosoma) + "/READY/"
        print data_dir

        files_list = os.listdir(data_dir)
        print files_list
```

```
data/chr15/READY/
['INDIV.DAT', 'MATRIX.DAT', 'POBLA.DAT', 'prueba.DAT', 'SNPS.DAT']
```

1.2.1 Lectura de datos

Se procede a cargar los datos de los ficheros en memoria

```
In [4]: # Se inicializan las variables

        snps = []
        individuos = []
        poblaciones = []
        matrix = []

        # Se carga primero el listado de snps:
        with open(data_dir + "SNPS.DAT", "r") as ins:
            data_snps = ins.read()
            snps = data_snps.split("\n")

        print "Se han cargado un total de " + str(len(snps)) + " SNPS.\n"


        # Se carga a continuacion los nombres de los individuos:
        with open(data_dir + "INDIV.DAT", "r") as ins:
            data_indv = ins.read()
            individuos = data_indv.split("\n")

        print "Se han cargado un total de " + str(len(individuos)) + " individuos.\n"

        # Se carga a continuacion la clasificacion de los individuos en poblaciones:
```

```
        with open(data_dir + "POBLA.DAT", "r") as ins:
            data_pob = ins.read()
            poblaciones = data_pob.split("\n")

        print "Se han cargado un total de " + str(len(poblaciones)) + " clasificaciones.\n"
        # Vamos a comprobar cuantos individuos hay clasificados en cada población:

        lista_pob = list(set(poblaciones))
        lista_pob.sort()
        for cod_pob in lista_pob:
            print "Hay " + str(poblaciones.count(cod_pob)) + " individuos de la población " + co


        # Por último se carga la matrix de datos:
        matrix = np.loadtxt(data_dir + "MATRIX.DAT")

        print "\nLas dimensiones de la matriz cargada en memoria son " + str(matrix.shape) + ".\
```

```
Se han cargado un total de 99440 SNPS.

Se han cargado un total de 270 individuos.

Se han cargado un total de 270 clasificaciones.

Hay 90 individuos de la población CEU.
Hay 45 individuos de la población CHB.
Hay 45 individuos de la población JPT.
Hay 90 individuos de la población YRI.

Las dimensiones de la matriz cargada en memoria son (270, 99440).
```

Se han cargado todos los datos y se ha comprobado visualmente que los números de individuos, SNPS y su clasificación cuadran.

1.2.2 Modelos de Clasificación

Preparación de los conjuntos de training y test El primer paso para probar modelos de clasificación es preparar dos conjuntos de datos. Uno será el conjunto de train (que se utilizará para entrenar el algoritmo), y el otro el conjunto de test, para probar la calidad de la clasificación conseguida.

Se generan esos conjuntos utilizando la función *model_selection.train_test_split* de la librería sklearn. Se van a generar los conjuntos de forma aleatoria, reservando un 20% de los individuos para realizar la validación de los algoritmos.

```
In [5]: # Se configura el tamaño del conjunto de validación como un 20% del set de datos
        validation_size = 0.20
        seed = 2356
```

```
        data_train, data_validation, class_train, class_validation = model_selection.train_test_

        # Se comprueban los tamaños de las muestras generadas
        print data_train.shape
        print data_validation.shape
        print len(class_train)
        print len(class_validation)

        # Se comprueba que efectivamente los valores seleccionados son aleatorios,
        # mostrando la clasificación de los individuos de validación.
        print class_validation
```

```
(216, 99440)
(54, 99440)
216
54
['CHB', 'CEU', 'YRI', 'YRI', 'YRI', 'JPT', 'YRI', 'CEU', 'JPT', 'CEU', 'YRI', 'CEU', 'YRI', 'CHB
```

Comparativa de rendimiento de algoritmos Para comparar el rendimiento de los algoritmos se utilizará la métrica *accuracy*.

```
In [6]: # Para determinar la accuracy de los modelos se utilizará 10-fold cross validation

        scoring = 'accuracy'
```

Los modelos que se van a probar son:

- K nearest neighbors
- Regresión logística
- Support Vector Machines
- Árboles de decisión
- Random Forest

```
In [7]: # Se realiza un bucle para probar modelos

        modelos = []
        modelos.append(('KNN', KNeighborsClassifier()))
        modelos.append(('LR', LogisticRegression()))
        modelos.append(('SVM', SVC()))
        modelos.append(('TREE', DecisionTreeClassifier()))
        modelos.append(('FOREST', RandomForestClassifier()))

        # Se evaluan uno por uno
        resultados = []
        medias = []
        nombres = []
```

```
        for nombre, modelo in modelos:
            kfold = model_selection.KFold(n_splits=10, random_state=seed)
            cv_results = model_selection.cross_val_score(modelo, data_train,
                                                         class_train, cv=kfold,
                                                         scoring=scoring)
            resultados.append(cv_results)
            nombres.append(nombre)
            medias.append (cv_results.mean())
            msg = "%s: %f (%f)" % (nombre, cv_results.mean(), cv_results.std())
            print(msg)
```

```
KNN: 0.962987 (0.039787)
LR: 0.985714 (0.030491)
SVM: 0.787662 (0.070301)
TREE: 0.837879 (0.056489)
FOREST: 0.865801 (0.062741)
```

Se seleccionan los dos mejores modelos para realizar una predicción con ellos:

```
In [8]: max = 0
        max2 = 0
        indice = -1
        indice2 = -1

        for i in range(len(medias)):
            if medias[i] > max:
                max2 = max
                indice2 = indice
                max = medias[i]
                indice = i
            elif medias[i] > max2:
                max2 = medias[i]
                indice2 = i


        # Mejor modelo
        (nombre1, model1) = modelos[indice]
        print "El modelo con mejor score es: " + nombre1
        print "Se realiza la predicción de validación:"
        model1.fit(data_train, class_train)
        predicciones = model1.predict(data_validation)
        print(confusion_matrix(class_validation, predicciones))
        print(classification_report(class_validation, predicciones))
```

```
        # Segundo modelo
        (nombre2, model2) = modelos[indice2]
        print "El modelo con segundo mejor score es: " + nombre2
        print "Se realiza la predicción de validación:"
        model2.fit(data_train, class_train)
        predicciones = model2.predict(data_validation)
        print(confusion_matrix(class_validation, predicciones))
        print(classification_report(class_validation, predicciones))
```

```
El modelo con mejor score es: LR
Se realiza la predicción de validación:
[[19  0  0  0]
 [ 0  8  0  0]
 [ 0  0  6  0]
 [ 0  0  0 21]]
             precision    recall  f1-score   support

        CEU       1.00      1.00      1.00        19
        CHB       1.00      1.00      1.00         8
        JPT       1.00      1.00      1.00         6
        YRI       1.00      1.00      1.00        21

avg / total       1.00      1.00      1.00        54

El modelo con segundo mejor score es: KNN
Se realiza la predicción de validación:
[[19  0  0  0]
 [ 0  8  0  0]
 [ 0  0  6  0]
 [ 0  0  0 21]]
             precision    recall  f1-score   support

        CEU       1.00      1.00      1.00        19
        CHB       1.00      1.00      1.00         8
        JPT       1.00      1.00      1.00         6
        YRI       1.00      1.00      1.00        21

avg / total       1.00      1.00      1.00        54
```

1.2.3 Modelos de Clustering

Para probar modelos de clustering no es necesario preparar conjuntos de training y test, por lo tanto se trabajará con los datos completos.

K-Means Se van a realizar pruebas de partición empleando el algoritmos K-Means con diversos parámetros.

```
In [9]: n_digits = 4
        sample_size = 270

        print('init         time      inertia    homo   compl  v-meas     ARI    AMI  silhouette'
        def bench_k_means(estimator, name, data, labels):
            t0 = time()
            estimator.fit(data)
            print('% 9s   %.2fs    %i   %.3f   %.3f   %.3f   %.3f   %.3f    %.3f'
                  % (name, (time() - t0), estimator.inertia_,
                     metrics.homogeneity_score(labels, estimator.labels_),
                     metrics.completeness_score(labels, estimator.labels_),
                     metrics.v_measure_score(labels, estimator.labels_),
                     metrics.adjusted_rand_score(labels, estimator.labels_),
                     metrics.adjusted_mutual_info_score(labels,  estimator.labels_),
                     metrics.silhouette_score(data, estimator.labels_, metric='euclidean', sampl

        bench_k_means(KMeans(init='k-means++', n_clusters=n_digits, n_init=10), name="k-means++"
                      labels = poblaciones)
        bench_k_means(KMeans(init='random', n_clusters=n_digits, n_init=10), name="random", data
                      labels = poblaciones)

        pca = PCA(n_components=n_digits).fit(matrix)
        bench_k_means(KMeans(init=pca.components_, n_clusters=n_digits, n_init=1), name="PCA-bas
                       labels = poblaciones)
```

```
init         time      inertia    homo   compl  v-meas     ARI    AMI  silhouette
k-means++   13.59s    5594390   0.826   0.833   0.829   0.730   0.824    0.085
   random   12.01s    5594885   0.826   0.835   0.831   0.733   0.824    0.086
PCA-based    1.60s    5603480   0.827   0.828   0.827   0.861   0.825    0.089
```

```
In [ ]:
```

C.2. Cromosoma 18

A continuación se incluye el informe completo de las pruebas realizadas con el cromosoma 18.

Analisis_ML CHR18

1 ANÁLISIS MACHINE LEARNING

1.1 Introducción

Este documento va a servir para realizar pruebas de ejecucion de diversos algoritmos de Machine Learning sobre el proyecto de HapMap.

Para realizar el informe se utilizan datos preprocesados previamente.

1.2 Preparación y lectura de datos

Se importan las librerías que se van a utilizar para realizar el informe.

```
In [1]: import os
        import numpy as np
        import matplotlib.pyplot as plt
        from time import time

        from sklearn import model_selection
        from sklearn.feature_selection import VarianceThreshold
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import chi2
        from sklearn import metrics
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import adjusted_rand_score
        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.svm import SVC
        from sklearn.cluster import KMeans
        from sklearn.cluster import MeanShift
        from sklearn.cluster import DBSCAN
        from sklearn.decomposition import PCA
```

El primer paso es indicar el número del cromosoma sobre el que se quiere realizar el estudio. Por defecto se trabajará con el cromosoma 15.

```
In [2]: # Cromosoma a utilizar para el informe, sus datos ya se
        # han depositado y preparado en la ruta indicada.
        cromosoma = 18
```

Se configura el directorio por defecto del que se recogeran los datos y se listan los ficheros para comprobar que están todos. Los ficheros esperados son:

- INDIV.DAT: Contiene los nombres identificativos de cada individuo.
- POBLA.DAT: Contiene el código de población correspondiente a cada individuo.
- SNPS.DAT: Contiene los nombres de los SNPS comunes a todas las poblaciones.
- MATRIX.DAT: Contiene los de SNPS x Individuos. Puede tomar valores de 0, 1 y 2.

```
In [3]: data_dir = "data/chr" + str(cromosoma) + "/READY/"
        print data_dir

        files_list = os.listdir(data_dir)
        print files_list
```

```
data/chr18/READY/
['INDIV.DAT', 'MATRIX.DAT', 'POBLA.DAT', 'SNPS.DAT']
```

1.2.1 Lectura de datos

Se procede a cargar los datos de los ficheros en memoria

```
In [4]: # Se inicializan las variables

        snps = []
        individuos = []
        poblaciones = []
        matrix = []

        # Se carga primero el listado de snps:
        with open(data_dir + "SNPS.DAT", "r") as ins:
            data_snps = ins.read()
            snps = data_snps.split("\n")

        print "Se han cargado un total de " + str(len(snps)) + " SNPS.\n"


        # Se carga a continuacion los nombres de los individuos:
        with open(data_dir + "INDIV.DAT", "r") as ins:
            data_indv = ins.read()
            individuos = data_indv.split("\n")

        print "Se han cargado un total de " + str(len(individuos)) + " individuos.\n"

        # Se carga a continuacion la clasificacion de los individuos en poblaciones:
```

```
        with open(data_dir + "POBLA.DAT", "r") as ins:
            data_pob = ins.read()
            poblaciones = data_pob.split("\n")

        print "Se han cargado un total de " + str(len(poblaciones)) + " clasificaciones.\n"
        # Vamos a comprobar cuantos individuos hay clasificados en cada población:

        lista_pob = list(set(poblaciones))
        lista_pob.sort()
        for cod_pob in lista_pob:
            print "Hay " + str(poblaciones.count(cod_pob)) + " individuos de la población " + co


        # Por último se carga la matrix de datos:
        matrix = np.loadtxt(data_dir + "MATRIX.DAT")

        print "\nLas dimensiones de la matriz cargada en memoria son " + str(matrix.shape) + ".\
```

```
Se han cargado un total de 110859 SNPS.

Se han cargado un total de 270 individuos.

Se han cargado un total de 270 clasificaciones.

Hay 90 individuos de la población CEU.
Hay 45 individuos de la población CHB.
Hay 45 individuos de la población JPT.
Hay 90 individuos de la población YRI.

Las dimensiones de la matriz cargada en memoria son (270, 110859).
```

Se han cargado todos los datos y se ha comprobado visualmente que los números de individuos, SNPS y su clasificación cuadran.

1.2.2 Modelos de Clasificación

Preparación de los conjuntos de training y test El primer paso para probar modelos de clasificación es preparar dos conjuntos de datos. Uno será el conjunto de train (que se utilizará para entrenar el algoritmo), y el otro el conjunto de test, para probar la calidad de la clasificación conseguida.

Se generan esos conjuntos utilizando la función *model_selection.train_test_split* de la librería sklearn. Se van a generar los conjuntos de forma aleatoria, reservando un 20% de los individuos para realizar la validación de los algoritmos.

```
In [5]: # Se configura el tamaño del conjunto de validación como un 20% del set de datos
        validation_size = 0.20
        seed = 2356
```

```
        data_train, data_validation, class_train, class_validation = model_selection.train_test_

        # Se comprueban los tamaños de las muestras generadas
        print data_train.shape
        print data_validation.shape
        print len(class_train)
        print len(class_validation)

        # Se comprueba que efectivamente los valores seleccionados son aleatorios,
        # mostrando la clasificación de los individuos de validación.
        print class_validation
```

```
(216, 110859)
(54, 110859)
216
54
['CHB', 'CEU', 'YRI', 'YRI', 'YRI', 'JPT', 'YRI', 'CEU', 'JPT', 'CEU', 'YRI', 'CEU', 'YRI', 'CHB
```

Comparativa de rendimiento de algoritmos Para comparar el rendimiento de los algoritmos se utilizará la métrica *accuracy*.

```
In [6]: # Para determinar la accuracy de los modelos se utilizará 10-fold cross validation

        scoring = 'accuracy'
```

Los modelos que se van a probar son:

- K nearest neighbors
- Regresión logística
- Support Vector Machines
- Árboles de decisión
- Random Forest

```
In [7]: # Se realiza un bucle para probar modelos

        modelos = []
        modelos.append(('KNN', KNeighborsClassifier()))
        modelos.append(('LR', LogisticRegression()))
        modelos.append(('SVM', SVC()))
        modelos.append(('TREE', DecisionTreeClassifier()))
        modelos.append(('FOREST', RandomForestClassifier()))

        # Se evaluan uno por uno
        resultados = []
        medias = []
        nombres = []
```

```
        for nombre, modelo in modelos:
            kfold = model_selection.KFold(n_splits=10, random_state=seed)
            cv_results = model_selection.cross_val_score(modelo, data_train,
                                                         class_train, cv=kfold,
                                                         scoring=scoring)
            resultados.append(cv_results)
            nombres.append(nombre)
            medias.append (cv_results.mean())
            msg = "%s: %f (%f)" % (nombre, cv_results.mean(), cv_results.std())
            print(msg)
```

```
KNN: 0.911688 (0.048735)
LR: 0.981818 (0.022268)
SVM: 0.787662 (0.070301)
TREE: 0.879221 (0.083426)
FOREST: 0.856710 (0.055740)
```

Se seleccionan los dos mejores modelos para realizar una predicción con ellos:

```
In [8]: max = 0
        max2 = 0
        indice = -1
        indice2 = -1

        for i in range(len(medias)):
            if medias[i] > max:
                max2 = max
                indice2 = indice
                max = medias[i]
                indice = i
            elif medias[i] > max2:
                max2 = medias[i]
                indice2 = i


        # Mejor modelo
        (nombre1, model1) = modelos[indice]
        print "El modelo con mejor score es: " + nombre1
        print "Se realiza la predicción de validación:"
        model1.fit(data_train, class_train)
        predicciones = model1.predict(data_validation)
        print(confusion_matrix(class_validation, predicciones))
        print(classification_report(class_validation, predicciones))
```

```
        # Segundo modelo
        (nombre2, model2) = modelos[indice2]
        print "El modelo con segundo mejor score es: " + nombre2
        print "Se realiza la predicción de validación:"
        model2.fit(data_train, class_train)
        predicciones = model2.predict(data_validation)
        print(confusion_matrix(class_validation, predicciones))
        print(classification_report(class_validation, predicciones))

El modelo con mejor score es: LR
Se realiza la predicción de validación:
[[19  0  0  0]
 [ 0  8  0  0]
 [ 0  0  6  0]
 [ 0  0  0 21]]
             precision    recall  f1-score   support

        CEU       1.00      1.00      1.00        19
        CHB       1.00      1.00      1.00         8
        JPT       1.00      1.00      1.00         6
        YRI       1.00      1.00      1.00        21

avg / total       1.00      1.00      1.00        54

El modelo con segundo mejor score es: KNN
Se realiza la predicción de validación:
[[19  0  0  0]
 [ 0  7  1  0]
 [ 0  0  6  0]
 [ 0  0  0 21]]
             precision    recall  f1-score   support

        CEU       1.00      1.00      1.00        19
        CHB       1.00      0.88      0.93         8
        JPT       0.86      1.00      0.92         6
        YRI       1.00      1.00      1.00        21

avg / total       0.98      0.98      0.98        54
```

1.2.3 Modelos de Clustering

Para probar modelos de clustering no es necesario preparar conjuntos de training y test, por lo tanto se trabajará con los datos completos.

K-Means Se van a realizar pruebas de partición empleando el algoritmos K-Means con diversos parámetros.

```
In [9]: n_digits = 4
        sample_size = 270

        print('init        time      inertia    homo   compl  v-meas     ARI    AMI  silhouette'
        def bench_k_means(estimator, name, data, labels):
            t0 = time()
            estimator.fit(data)
            print('% 9s   %.2fs    %i   %.3f   %.3f   %.3f   %.3f   %.3f    %.3f'
                  % (name, (time() - t0), estimator.inertia_,
                     metrics.homogeneity_score(labels, estimator.labels_),
                     metrics.completeness_score(labels, estimator.labels_),
                     metrics.v_measure_score(labels, estimator.labels_),
                     metrics.adjusted_rand_score(labels, estimator.labels_),
                     metrics.adjusted_mutual_info_score(labels,  estimator.labels_),
                     metrics.silhouette_score(data, estimator.labels_, metric='euclidean', sampl

        bench_k_means(KMeans(init='k-means++', n_clusters=n_digits, n_init=10), name="k-means++"
                      labels = poblaciones)
        bench_k_means(KMeans(init='random', n_clusters=n_digits, n_init=10), name="random", data
                      labels = poblaciones)

        pca = PCA(n_components=n_digits).fit(matrix)
        bench_k_means(KMeans(init=pca.components_, n_clusters=n_digits, n_init=1), name="PCA-bas
                       labels = poblaciones)
```

```
init        time      inertia    homo   compl  v-meas     ARI    AMI  silhouette
k-means++   15.88s    5944178   0.826   0.842   0.834   0.743   0.824    0.066
   random   14.47s    5947275   0.826   0.833   0.829   0.730   0.824    0.066
PCA-based   1.87s    5960472   0.834   0.835   0.834   0.869   0.832    0.069
```

```
In [ ]:
```

Bibliografía

[1] Varios, "Wikipedia - international hapmap project." https://en.wikipedia.org/wiki/International_HapMap_Project.

[2] Varios, "Nhgri - about the international hapmap project." https://www.genome.gov/11511175/.

[3] Varios, "Wikipedia - haplotipo." https://es.wikipedia.org/wiki/Haplotipo.

[4] Varios, "Wikipedia - python." https://es.wikipedia.org/wiki/Python.

[5] Varios, "Wikipedia - tiobe index." https://en.wikipedia.org/wiki/TIOBE_index.

[6] Varios, "Wikipedia - measuring programming language popularity." https://en.wikipedia.org/wiki/Measuring_programming_language_popularity.

[7] Varios, "Wikipedia - yaml." https://es.wikipedia.org/wiki/YAML.

[8] Varios, "Multiprocesamiento en python: Global interpreter lock (gil)." https://www.genbetadev.com/python/multiprocesamiento-en-python-global-interpreter-lock-gil.

[9] B. Lantz, *Machine Learning with R Second Edition*. Packt Publishing Ltd., July 2015.

[10] Varios, "Wikipedia - aprendizaje automático." https://es.wikipedia.org/wiki/Aprendizaje_automático.

[11] Varios, "Wikipedia - k-means clustering." https://en.wikipedia.org/wiki/K-means_clustering.

[12] Varios, "Wikipedia - ipython." https://es.wikipedia.org/wiki/IPython.

[13] Varios, "Wikipedia - scikit-learn." https://en.wikipedia.org/wiki/Scikit-learn.

[14] Varios, "Documentation of scikit-learn 0.18." http://scikit-learn.org/stable/documentation.html.