Predictor para el Síndrome de Lynch: Comparativa y análisis de algoritmos de machine learning

Marta Muñoz López
Máster universitario en Bioinformática y bioestadística UOC-UB
TFM-Estadística y Bioinformática 1

Pau Andrio Balado

01/2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Predictor para el Síndrome de Lynch: Comparativa y análisis de algoritmos de machine learning*
Nombre del autor:	*Marta Muñoz López*
Nombre del consultor/a:	*Pau Andrio Balado*
Fecha de entrega (mm/aaaa):	01/2019
Titulación:	*Máster universitario en Bioinformática y bioestadística UOC-UB*
Área del Trabajo Final:	*TFM-Estadística y Bioinformática 1*
Idioma del trabajo:	*Español*
Palabras clave	*Lynch, clasificador, comparador*

Resumen del Trabajo (máximo 250 palabras):

En el desarrollo de este proyecto se ha analizado el síndrome de Lynch, un síndrome hereditario que aumenta las posibilidades de padecer ciertos tipos de cáncer. Este síndrome se detecta debido a mutaciones en los genes: MLH1, MSH1, MSH2, MSH6 o PMS2. Se ha querido desarrollar un predictor del síndrome partiendo de ciertas características de los individuos. Para ello, el primer paso ha sido recolectar datos para poder aplicar distintos algoritmos de Machine Learning. Tras encontrar los datos más adecuados, se ha realizado un análisis de estos, luego se han aplicado distintos algoritmos de clasificación para al final compararlos todos. La librería principal utilizada en el desarrollo del producto ha sido scikit-learn de Python. Dentro de todos los algoritmos utilizados, en algunos de ellos se ha realizado un estudio por minorizado de los parámetros configurables para de esta forma obtener los resultados más óptimos. Todo el trabajo se ha visto marcado debido a los datos, ya que la muestra de estos era pequeña.

Abstract (in English, 250 words or less):

Along the development of this project, Lynch Syndrome has been analysed, an hereditarian syndrome which increases the possibilities of suffering different kinds of cancer. This syndrome is detected due to mutations in different genes like: MLH1, MSH1, MSH2, MSH6 o PMS2. A predictor has been developed coming from individual characteristics. First of all, data have been collected in order to apply to different machine learning algorithms. After finding the most suitable data, an analysis has been done. After that, different classification algorithms have been studied with the purpose of comparing all of them. The main library used in the product development has been Python scikit-learn. Some

algorithms have been searched of in detail to obtain optima results. The project has been determined because of a small sample data.

Índice

1. Introducción 1
1.1 Contexto y justificación del Trabajo 1
1.2 Objetivos del Trabajo 1
1.3 Enfoque y método seguido 1
1.4 Planificación del Trabajo 2
1.5 Breve sumario de productos obtenidos 4
1.6 Breve descripción de los otros capítulos de la memoria 5
2. Síndrome de Lynch 7
2.1. Criterios de Ámsterdam para el Síndrome de Lynch 8
3. Recolectar los datos 10
4. Análisis inicial de los datos 12
4.1. Limitaciones de los datos 15
5. Procesar los datos 17
6. Valoración de los resultados obtenidos 19
7. Algoritmo: K-Nearest-Neighbors 20
7.1. Entrenamiento del modelo 20
7.2. Evaluación del modelo 22
8. Algoritmo: Support Vector Machines 24
8.1. Entrenamiento del modelo 24
8.2. Evaluación del modelo 25
9. Algoritmo: Decision Tree 27
9.1. Entrenamiento del modelo 27
9.2. Evaluación del modelo 32
10. Algoritmo: Random Forest 36
10.1. Entrenamiento del modelo 36
10.2. Evaluación del modelo 41
11. Algoritmo: Bernoulli Naive Bayes 43
11.2. Evaluación del modelo 43
12. Algoritmo: Linear Discriminant Analysis (LDA) 45
12.1. Entrenamiento del modelo 45
12.2. Evaluación del modelo 45
13. Algoritmo: Quadratic Discriminant Analysis (QDA) 47
13.1. Entrenamiento del modelo 47
13.2. Evaluación del modelo 47
14. Algoritmo: Gaussian process classification (GPC) 49
14.1. Entrenamiento del modelo 49
14.2. Evaluación del modelo 49
15. Comparación de todos los algoritmos 51
16. Futuras líneas de investigación 52
17. Conclusiones 53
18. Glosario 54
19. Bibliografía 56
20. Anexos 57

Lista de figuras

Figura 1: Diagrama de Gantt para el proyecto 4
Figura 2: Gráfico de distribución del atributo 'Gene' 12
Figura 3: Gráfico de distribución del atributo 'Region' 13
Figura 4: Gráfico de distribución del atributo 'DNA change type' 13
Figura 5: Gráfico de distribución del atributo 'AA change type' 14
Figura 6: Gráfico de relación entre 'Category' y 'Disease' 14
Figura 7: Gráfico de correlación entre todos los atributos 15
Figura 8: Representación de la matriz de confusión 19
Figura 9: Llamada al clasificador 'knn' 21
Figura 10: Gráfica de exactitud con cross-validation para distintos parámetros de knn 21
Figura 11: Valores obtenidos de la matriz de confusión para los distintos algoritmos de knn 23
Figura 12: Tabla comparativa de exactitud para distintos algoritmos de SVM 24
Figura 13: Gráfica de exactitud con cross-validation para distintos grados de SVM polinomial 25
Figura 14: Gráfica comparativa de resultados para la clase 'Disease' 29
Figura 15: Gráfico de exactitud media con cross-validation para valores de 'min_samples_split' para 'Decision tree' 30
Figura 16: Gráfico de exactitud media con cross-validation para valores de 'min_samples_leaf' para 'Decision tree' 31
Figura 17: Gráfico de exactitud media con cross-validation para valores de 'max_depth' para 'Decision tree' 32
Figura 18: Árbol de decisión obtenido 34
Figura 19: Gráfico de exactitud media con cross-validation para valores de ‘n_estimators' para 'Random Forest’ 37
Figura 20: Gráfico de exactitud media con cross-validation para valores de ‘max_features’ para 'Random Forest’ 38
Figura 21: Gráfico de exactitud media con cross-validation para valores de ‘min_samples_leaf’ para 'Random Forest’ 39
Figura 22: Gráfico de exactitud media con cross-validation para valores de ‘min_samples_split’ para 'Random Forest’ 40
Figura 23: Importancia de cada atributo para el modelo 42
Figura 24: Entrenamiento del modelo 'BernoulliNB' 43
Figura 25: Entrenamiento del modelo 'LinearDiscriminantAnalysis' 45
Figura 26: Entrenamiento del modelo 'QuadraticDiscriminantAnalysis' 47
Figura 27: Entrenamiento del modelo 'GaussianProcessClassifier' 49
Figura 28: Tabla comparativa de exactitud de los algoritmos 51

1. Introducción

1.1 Contexto y justificación del Trabajo

La elección de este proyecto se justifica doblemente, por una parte, la intención de descubrir las posibilidades de Machine Learning en Python, ya que siempre había desarrollado modelos de Machine Learning en R y Python es una herramienta que utilizo a diario en mi ámbito de trabajo.

La elección de estudio de Síndrome de Lynch surge debido a que en mi familia se han detectado casos de síndrome de Lynch y se están realizando test genéticos para detectar los posibles casos. A partir de esta situación comencé a investigar sobre el síndrome.

1.2 Objetivos del Trabajo

I. Aplicar conocimientos y resolver problemas en el ámbito de la Bioinformática y la Bioestadística, tanto en entornos conocidos como en entornos nuevos.
II. Capacidad de búsqueda, gestión y uso de información y recursos en el ámbito de la Bioinformática y la Bioestadística.
III. Aplicar conocimientos adquiridos en el máster sobre Machine Learning.
IV. Aplicar los conocimientos de Machine Learning en el entorno de Python.
V. Determinar un conjunto de datos que sea de interés para el proyecto a desarrollar.
VI. Desarrollar modelos y algoritmos.
VII. Interpretar los resultados científicos de forma correcta.

1.3 Enfoque y método seguido

El método a seguir son los pasos estándar para realizar un experimento de Machine Learning:

1. Determinar los objetivos del estudio.
2. Obtener los datos.
3. Preparar los datos para su posterior análisis.
4. Determinar los elementos a analizar.
5. Elección del modelo a aplicar.
6. Aplicar el modelo. Primero se aplica sobre un subconjunto de datos para entrenar el modelo.
7. Evaluación de los datos obtenidos.
8. Ajustes del modelo. La evaluación del modelo puede determinar que hay ciertos parámetros que son necesarios mejorar para obtener una mejor evaluación.
9. Presentación de los datos.

1.4 Planificación del Trabajo

A continuación, se detallan las tareas en relación con los objetivos:

I. Aplicar conocimientos adquiridos en el máster sobre Machine Learning.
   a. Búsqueda bibliográfica: Partiendo del problema que se pretende resolver, en este caso un predictor para detectar el síndrome de Lynch, y partiendo de los conocimientos adquiridos a lo largo del máster se pretende determinar distintas alternativas para resolver el caso de estudio.

II. Aplicar los conocimientos de Machine Learning en el entorno de Python.
   a. Instalación de herramientas y lectura de documentación: Una vez determinado que el lenguaje a utilizar es Python, hay que instalar todos los entornos necesarios, así como las librerías que se van a utilizar.

III. Determinar un conjunto de datos que sea de interés para el proyecto a desarrollar.
   a. Búsqueda de datos a estudiar: Para el desarrollo del predictor es necesario encontrar un conjunto de datos de pacientes que sea suficiente (en cantidad), así como útil para el desarrollo del modelo (de calidad). Los campos tienen que ser apropiados para poder procesarlos de forma correcta y que aporten información útil para el algoritmo.
   b. Recolectar los datos: La búsqueda del conjunto de datos idóneo puede ser tediosa, ya que en muchos casos suelen ser privados o se necesitan autorizaciones específicas de las plataformas.
   c. Preprocesar y explorar los datos: Una vez obtenidos los datos adecuados, es necesario formatearlos de forma que se puedan utilizar y hacer un primer análisis de las características de los mismos.

IV. Desarrollar modelos y algoritmos e interpretar los resultados de forma correcta.
   a. Elección del modelo: Dependiendo de los datos procesados y el fin que queramos obtener hay que determinar cuál es el algoritmo que utilizaremos para el estudio a realizar.
   b. Entrenar el modelo: Hay que determinar que set del conjunto de datos va a utilizarse para realizar el entrenamiento. Hay que denotar una mejora incremental en el proceso de entrenamiento, si los resultados no son correctos, habrá que volver a realizar el proceso.
   c. Evaluar el modelo: Una vez entrenada la máquina, hay que verificar la precisión del modelo ya entrenado. La exactitud tiene que ser mayor o igual a 50%.
   d. Ajustes del modelo: Si durante la evaluación no obtuvimos buenas predicciones hay que volver al paso de entrenar el modelo, ajustando ciertos parámetros.

e. Presentación de los datos obtenidos: Una vez que se hayan obtenido resultados adecuados, hay que realizar un informe para presentar los resultados de forma correcta.

V. Interpretar los resultados científicos de forma correcta.

a. Redacción de la memoria del TFM: Una vez concluido todo el proceso de desarrollo hay que elaborar una memoria entregable redactando todo el proceso, los problemas encontrados y las conclusiones, justificando de forma correcta el desarrollo de la asignatura.

b. Elaboración de la presentación: Otra parte evaluable es la presentación pública, para esto hay que elaborar una presentación concisa y completa.

c. Defensa pública.

En la siguiente tabla se especifican las fechas establecidas para las tareas y PECs.

Descripción de la tarea	Tarea	Fecha de Inicio	Fecha de Fin	Días
PEC0 - Definición de los contenidos del trabajo				
Búsqueda bibliográfica	1.a	19-septiembre	1-octubre	9 días
PEC1 - Plan de trabajo				
Búsqueda de datos a estudiar	2.a	2-octubre	8-octubre	5 días
Instalación de herramientas y lectura de documentación	3.a	9-octubre	15-octubre	5 días
PEC2 - Desarrollo del trabajo - Fase 1				
Recolectar los datos	3.b	16-octubre	22-octubre	5 días
Preprocesar y explorar los datos	3.c	23-ocubre	29-octubre	5 días
Elección del modelo	4.a	30-octubre	6-noviembre	6 días
Entrenar el modelo	4.b	7-noviembre	19-noviembre	9 días
PEC3 - Desarrollo del trabajo - Fase 2				
Entrenamientos de modelos	4.b2	20-noviembre	25-noviembre	4 días
Evaluar los modelos	4.c	25-noviembre	30-noviembre	5 días
Ajustes de modelos	4.d	1-diciembre	9-diciembre	8 días

Presentación de los datos obtenidos	4.e	10-diciembre	17-diciembre	6 días
PEC4 -Cierre de la memoria				
Redacción de la memoria del TFM	5.a	18-diciembre	2-enero	12 días
PEC5a - Elaboración de la presentación				
Elaboración de la presentación	5.b	3-enero	10-enero	6 días
PEC5b – Defensa pública				
Defensa pública	5.c	Día a determinar entre: 14/01/2019 - 23/01/2019		

El siguiente diagrama de Gantt muestra la planificación de forma gráfica:

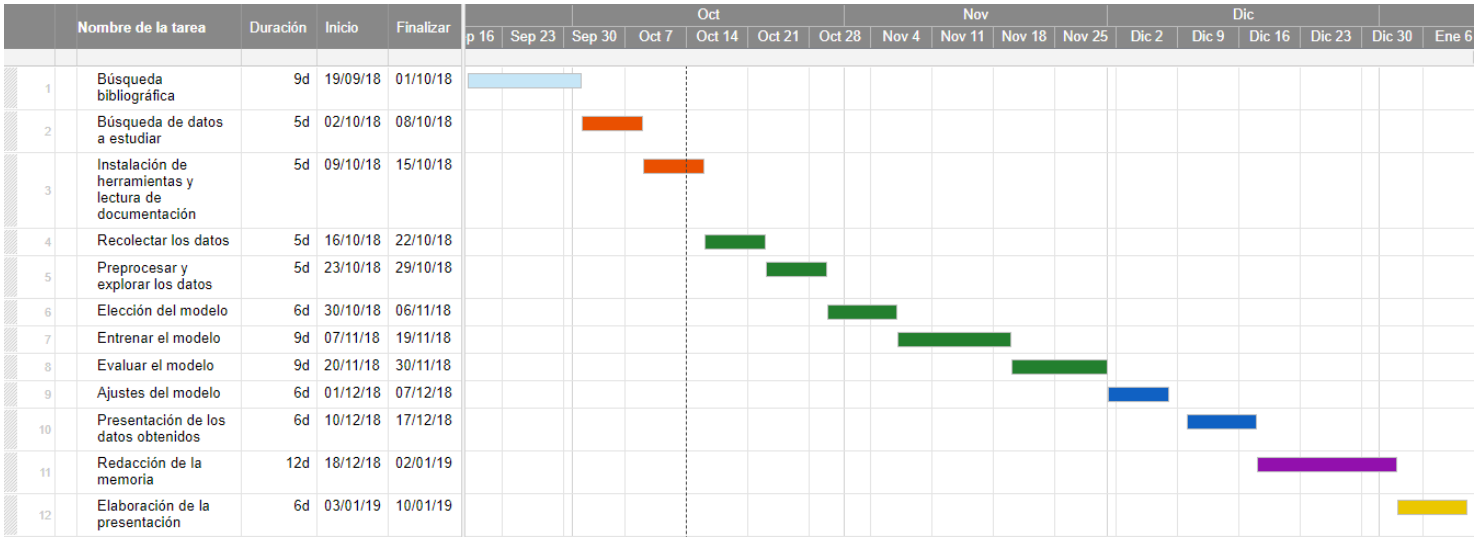

Figura 1: Diagrama de Gantt para el proyecto

1.5 Breve sumario de productos obtenidos

Tras el desarrollo de toda la asignatura los entregables que se obtenidos son:

I. PEC 0 - Definición de los contenidos del trabajo
II. PEC 1 – Plan de trabajo
III. PEC 2 – Memoria parcial – FASE 1
IV. PEC 3 – Memoria parcial – FASE 2
V. Memoria
VI. Código del desarrollo:
Todo el código se ha alojado en el repositorio de GitHub en la siguiente URL:
https://github.com/PauAndrio/TFM_Marta_Munoz
VII. Diapositivas de la presentación pública

1.6 Breve descripción de los otros capítulos de la memoria

1. Introducción
   Breve descripción de los distintos objetivos, tareas, hitos que se han querido desarrollar durante este proyecto.

2. Síndrome de Lynch
   Explicación de la temática de estudio en la que se basa todo el trabajo a desarrollar. Se explica en que consiste el síndrome y los protocolos para detectar la posibilidad de padecerlo.

3. Recolectar los datos
   Se especifica los requisitos que se buscaba del conjunto de datos a estudiar, los problemas encontrados y el porqué de algunas bases de datos descartadas.

4. Análisis inicial de los datos
   Una vez escogido la muestra de datos con la que se va a trabajar se realiza un estudio previo de todas las características de los atributos y las clases.

5. Procesar los datos
   Primero pasos y procesamientos a realizar con la muestra para poder aplicar los distintos algoritmos.

6. Valoración de los resultados obtenidos
   Definición de los distintos resultados que se van a obtener al final de evaluar cada algoritmo para realizar las comparaciones.

7. Algoritmo: K-Nearest-Neighbors
   Entrenamiento y evaluación del modelo para el algoritmo, así como estudio de los posibles parámetros configurables.

8. Algoritmo: Support Vector Machines
   Entrenamiento y evaluación para las distintas posibilidades que ofrece scikit-learn para algoritmos SVM.

9. Algoritmo: Decision Tree
   Entrenamiento y evaluación para las distintas posibilidades que ofrece scikit-learn, así como estudio de los posibles parámetros configurables.

10. Algoritmo: Random Forest
    Entrenamiento y evaluación para las distintas posibilidades que ofrece scikit-learn, así como estudio de los posibles parámetros configurables.

11. Algoritmo: Bernoulli Naive Bayes
    Entrenamiento y evaluación del modelo para el algoritmo definido en scikit-learn.

12. Algoritmo: Linear Discriminant Analysis (LDA)
    Entrenamiento y evaluación del modelo para el algoritmo definido en scikit-learn.

13. Algoritmo: Quadratic Discriminant Analysis (QDA)
    Entrenamiento y evaluación del modelo para el algoritmo definido en scikit-learn.

14. Algoritmo: Gaussian process classification (GPC)
    Entrenamiento y evaluación del modelo para el algoritmo definido en scikit-learn.

15. Comparación de todos los algoritmos
    Una vez obtenidos todos los resultados de las evaluaciones de los algoritmos se realiza una comparación de los mismos.

16. Futuras líneas de investigación
    Propuestas para seguir desarrollando el trabajo propuesto en este proyecto, ampliando el contenido con algunas variaciones o con otras líneas de investigación posible.

17. Conclusiones
    En este apartado se localizarán las conclusiones y reflexiones obtenidas al finalizar el TFM.

18. Glosario
    Definición de los términos y acrónimos más relevantes utilizados dentro de la Memoria.

19. Bibliografía
    Lista numerada de las referencias bibliográficas utilizadas dentro de la memoria y del TFM.

20. Anexos
    Listado de apartados que son demasiado largos o extensos para ser incluidos dentro de la memoria. En este capítulo se incluirán los códigos de programación.

2. Síndrome de Lynch

El síndrome de Lynch o HNPCC (en inglés) es un síndrome hereditario por el que un paciente presenta una mayor disposición a padecer determinados tipos de cáncer.

Este síndrome hereditario generalmente resulta en una mutación de la línea germinal en 1 de 4 genes MMR de ADN: MLH1, MSH1, MSH2, MSH6 o PMS2. Una mutación en la línea germinal es la mutación que se produce en las células reproductoras (óvulo o espermatozoide) y se incorpora en el ADN de las células de cada uno de los descendientes.
Los genes MMR (mismatch repair) son los genes encargados de la reparación de errores del ADN cuando se produce la copia para preparar la división celular. Las mutaciones en estos genes impiden la reparación de los errores de replicación de ADN.

También existe una posible asociación con otros tres genes: MLH4, PMS1 y EXO1, así como una eliminación de EPCAM. [1]. Las mutaciones en el gen EPCAM también conducen a una reparación deficiente del ADN, aunque el gen no está involucrado en este proceso. El gen EPCAM se encuentra junto al gen MSH2 en el cromosoma 2, ciertas mutaciones en el gen EPCAM hacen que el gen MSH2 se desactive, interrumpiendo la reparación del ADN y provocando errores acumulados en el ADN.

La mutación de genes involucrados con el síndrome de Lynch se manifiesta en 1 de cada 300 personas en el mundo, es el síndrome de predisposición cáncer de colón hereditario más común. En muchos casos, este síndrome se detecta debido a que varios individuos de una familia han padecido cáncer colorrectal o de endometrio. [2]

En el caso de los EEUU, se diagnostican unos 140000 casos de cáncer colorectal anualmente. Aproximadamente, entre un 3-5% de estos casos son causados por el síndrome de Lynch.

Actualmente, se siguen investigando otros genes que puedan estar relacionados con el síndrome de Lynch, además de los cinco ya nombrados. Puesto que en algunas familias que parece manifestarse, no se ha detectado ninguna mutación en los genes identificados con el síndrome de Lynch. [3]

Otras líneas de investigación relacionadas con el síndrome están enfocadas a las causas por las que un individuo presenta mutaciones en los genes relacionados con el síndrome, pero no han sido heredados, mutaciones adquiridas. Esta condición se detecta cuando un tumor presenta la mutación, pero no se presenta la misma mutación en sangre.

A continuación, se muestra una tabla con el porcentaje de riesgo de distintos tipos de cáncer para individuos que presentan el síndrome de Lynch frente a la población general y la edad de inicio del cáncer. Los

resultados ‘no reportados’ se indican con un guion. Las celdas etiquetadas con * indica que la combinación de todos esos tipos de cáncer presenta un riesgo de un 6%. [4]

Cáncer	Riesgo de la población general (%)	MLH1 o MSH2		MSH6		PMS2	
		Riesgo (%)	Edad media	Riesgo (%)	Edad media	Riesgo (%)	Edad media
Colón	4.5	52-82	44-61 años	10-22	54 años	15-20	61-66 años
Endometrio	2.7	25-60	48-62 años	16-26	55 años	15	49 años
Estómago	<1	6-13	56 años	≤3	63 años	*	70-78 años
Vías biliares	<1	1-4	50-57 años	-	-	*	-
Tracto urinario	<1	1-7	54-60 años	<1	65 años	*	-
Intestino delgado	<1	3-6	47-49 años	-	54 años	*	59 años
Cerebral o sistema nervioso central	<1	1-3	~50 años	-	-	*	45 años
Carcinoma sebáceo	<1	1-9	-	-	-	-	-
Páncreas	<1	1-6	-	-	-	-	-

Tabla 1: Riesgo de presentar cáncer de la población general frente pacientes con síndrome de Lynch

2.1. Criterios de Ámsterdam para el Síndrome de Lynch

Los criterios de Ámsterdam sirven para identificar a las personas candidatas a realizarse un estudio genético que demuestre la existencia de síndrome de Lynch. Los criterios se enumeran a continuación: [5]

1. Tres miembros de la familia o más deben haber presentado un cáncer de colon u otros tumores asociados (endometrio, ovario, intestino delgado, uréter).

2. Uno de los afectados debe ser pariente en primer grado de otras dos personas de la familia con algunos de los tumores antes reseñados.

3. Debe afectar como mínimo a dos generaciones.

4. Al menos una de las personas debe tener una edad inferior a 50 años cuando se realizó el diagnóstico de tumor maligno.

5. El cáncer debe estar confirmado mediante estudios de anatomía patológica.

6. Se debe descartar otras enfermedades como la poliposis adenomatosa familiar.

3. Recolectar los datos

Tras decidir la temática sobre la que se va a realizar el proyecto: un predictor para detectar la presencia del síndrome de Lynch, comenzó la búsqueda de unos datos adecuados para el desarrollo del trabajo.

El enfoque de la búsqueda se centró en base de datos de registros de pacientes que presenten una mutación genética en uno de los genes involucrados en el síndrome de Lynch: MLH1, MSH1, MSH2, MSH6, y PMS2.

La búsqueda de una base de datos adecuada fue compleja, muchas fueron descartadas por distintos motivos:

- Falta de datos: Número de registros insuficientes.
- Datos incompletos: Gran cantidad de datos nulos en cada registro.
- Pocos atributos o de poca calidad.

La base de datos ClinVar[1] contiene únicamente: localización de la mutación, gen mutado, significado clínico y estado de la revisión. Por esta razón fue descartada.

En el caso de la página de LOVD (Leiden Open Variation Database)[2] se obtienen gran cantidad de registros (filtrando por los genes involucrados en el síndrome) pero hay una cantidad notable de datos nulos.

Finalmente se escogió una base de datos de la plataforma Canadian Open Genetics Repository[3]. Está plataforma contiene una base de datos que permite filtrar por genes afectados, enfermedades, hospital y significado clínico (benigno, patogénico...).

De los cinco genes involucrados en el síndrome, se han seleccionado los datos de todos, excepto MSH1 que no tenía ningún registro. Al filtrar por los cuatro genes MMR involucrados en el síndrome, obtenemos 402 entradas y 12 atributos que se enumeran a continuación:

1. *Submitter*: Hospital o clínica en el que se han realizado los estudios de cada sujeto.
2. *Gene*: Gen en el que se presenta la mutación en el caso de estudio: MLH1, MSH2, MSH6 y PMS2.

[1] https://www.ncbi.nlm.nih.gov/clinvar/
[2] https://databases.lovd.nl/
[3] http://opengenetics.ca/

3. HGVS cDNA: Utiliza la nomenclatura HGVS, que se utiliza para describir que tipo de cambio y en qué base se ha producido la mutación.
4. *HGVS Protein*: Indica cambios en una proteína.
5. *Transcipt*: Transcripción de RefSeq.
6. *Build*: Genoma de referencia, para este caso GRCh37.
7. *Chromosome*: Cromosoma afectado, para estos datos las opciones son: 2, 3 y 7.
8. *Genomic DNA change*: Indica cambio genómico en esa mutación.
9. *DNA Change Type*: Tipo de mutación ocurrida: deleción, duplicación, inserción, sustitución e inserción-deleción (para los datos filtrados encontramos estas opciones).
10. *AA Change Type:* Tipo de mutación en aminoácido: silenciosa, sin sentido, desconocida, deleción, frameshift (marco de lectura) …
11. *Clinical Significance*: Significado clínico: patogénico, benigno, probable benigno, probable patógeno.
12. *Associated Diseases(s)*: En este caso solo existe la opción: Síndrome de Lynch.

Al descargar los datos se obtiene un archivo csv que incluye algún campo más como: región (indica el exón o intrón), referencia externa, última fecha de evaluación del registro, etc.

4. Análisis inicial de los datos

De los distintos atributos que se obtienen los más interesantes (en un principio) serían: 'Gene', 'Region', 'DNA Change Type', 'AA Change Type', 'Category' y 'Disease'.

A continuación, se muestran unas gráficas de barras que muestra para cada variable categórica la frecuencia en el conjunto de datos a estudiar. Todo el código de desarrollo de este apartado, así como la generación de gráficas se encuentra en el siguiente archivo: https://github.com/PauAndrio/TFM_Marta_Munoz/blob/master/load_excel_data_analysis.py

En el caso del atributo 'Gene' hay tres genes muy equiparados, pero el PMS2 se presenta con una frecuencia hasta 3 veces menor.

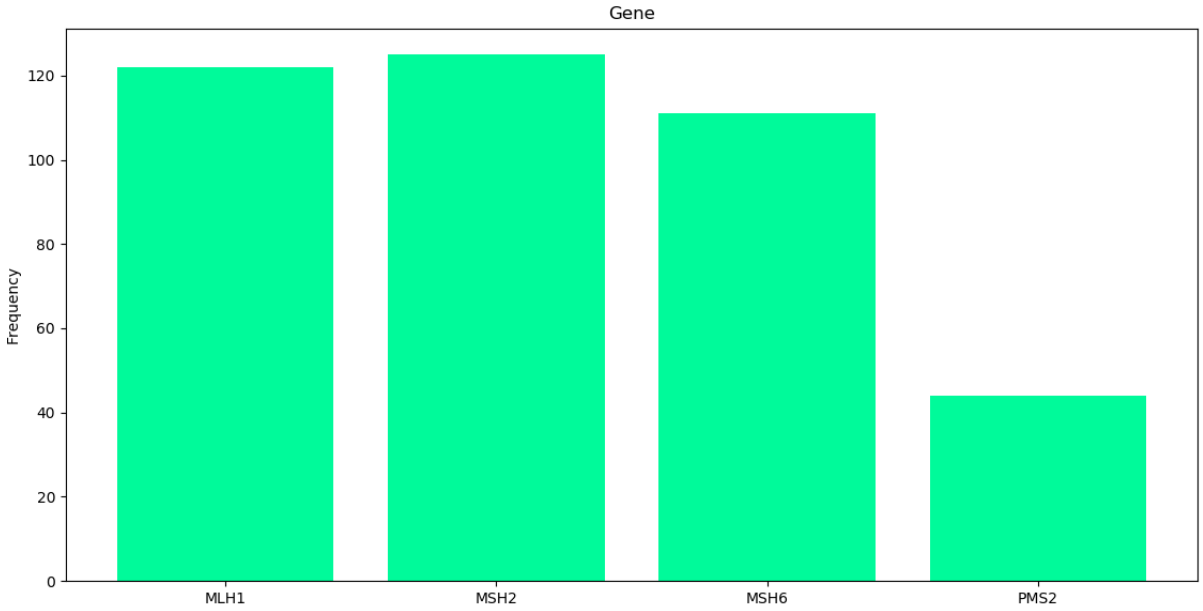

Figura 2: Gráfico de distribución del atributo 'Gene'

Para el atributo 'Region' la distribución es más equitativa, la mayoría presenta una frecuencia entre 1 y 15.

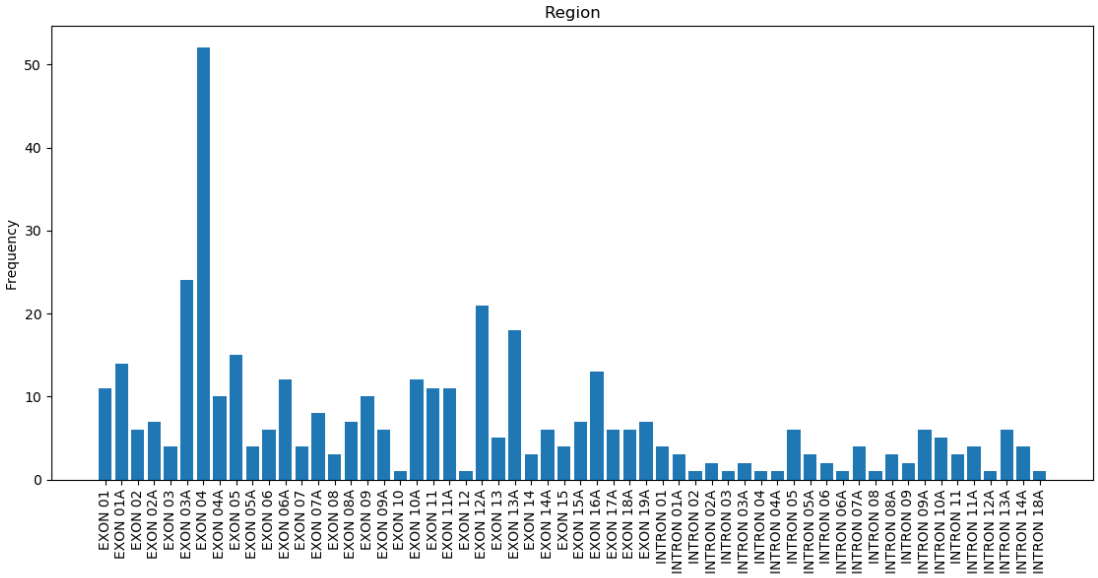

Figura 3: Gráfico de distribución del atributo 'Region'

Para los tipos de cambio en aminoácidos y ADN (*DNA Change Type* y *AA Change type*) la distribución de datos es mucho más desigual.

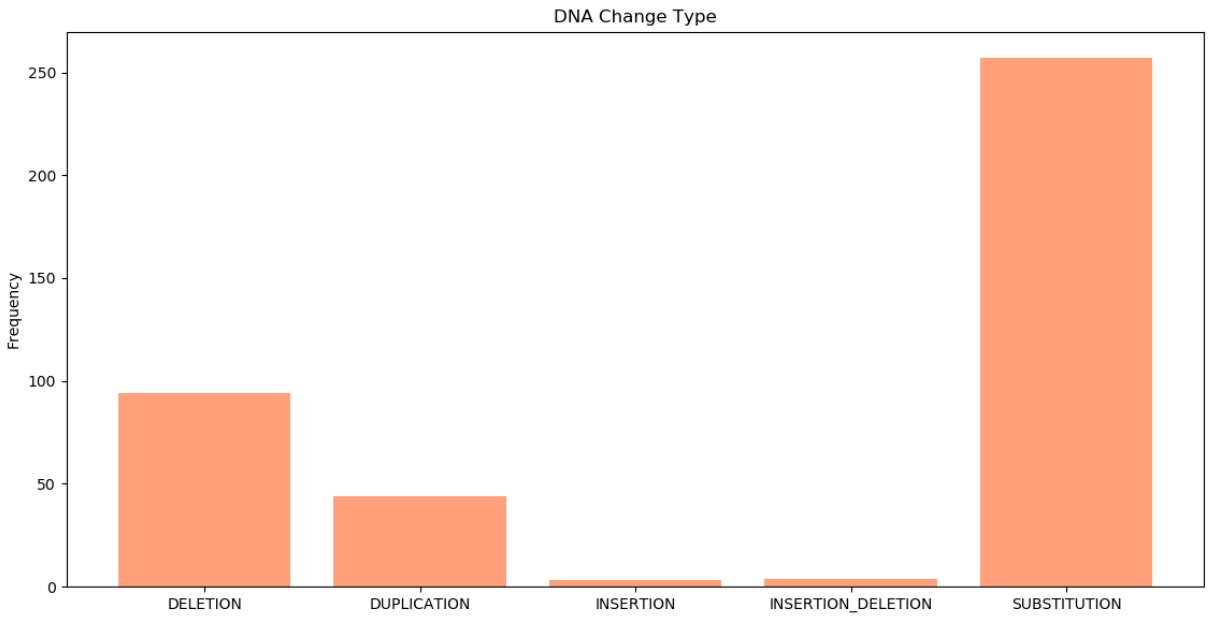

Figura 4: Gráfico de distribución del atributo 'DNA change type'

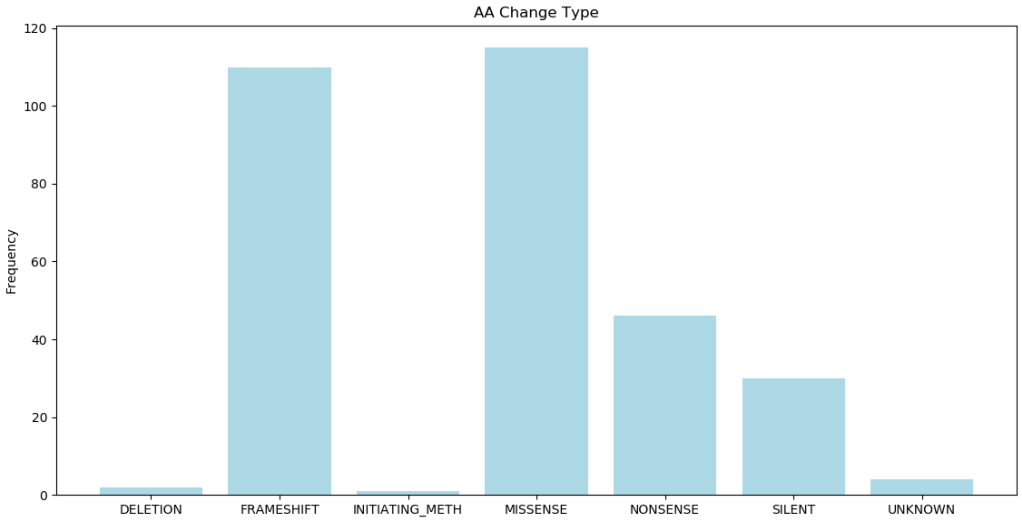

Figura 5: Gráfico de distribución del atributo 'AA change type'

Para las variables Category y Disease, la distribución es similar. La relación entre estos dos atributos tiene que estar ligada. Ya que, por ejemplo, los registros etiquetados como ‘Pathogenic’ tiene que presentar el síndrome de Lynch, como se observa en el siguiente gráfico

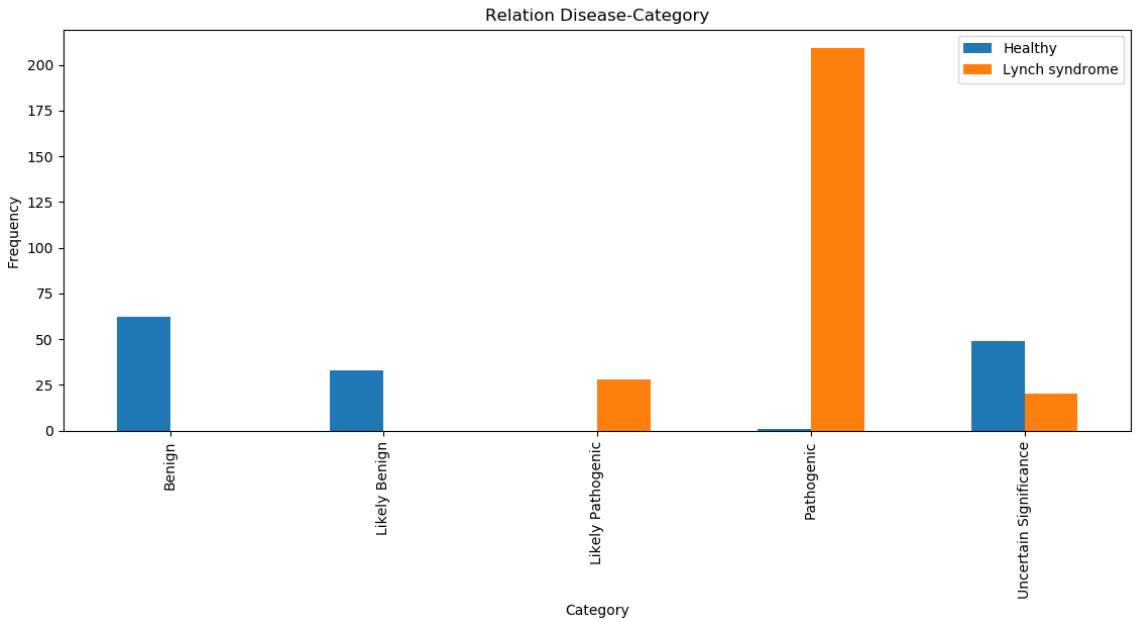

Figura 6: Gráfico de relación entre 'Category' y 'Disease'

Como última observación, es interesante comprobar la correlación entre los distintos atributos que van a formar la matriz de diseño. Para ello, se ha creado un gráfico de calor en el que se observa el grado de correlación entre todos los atributos.

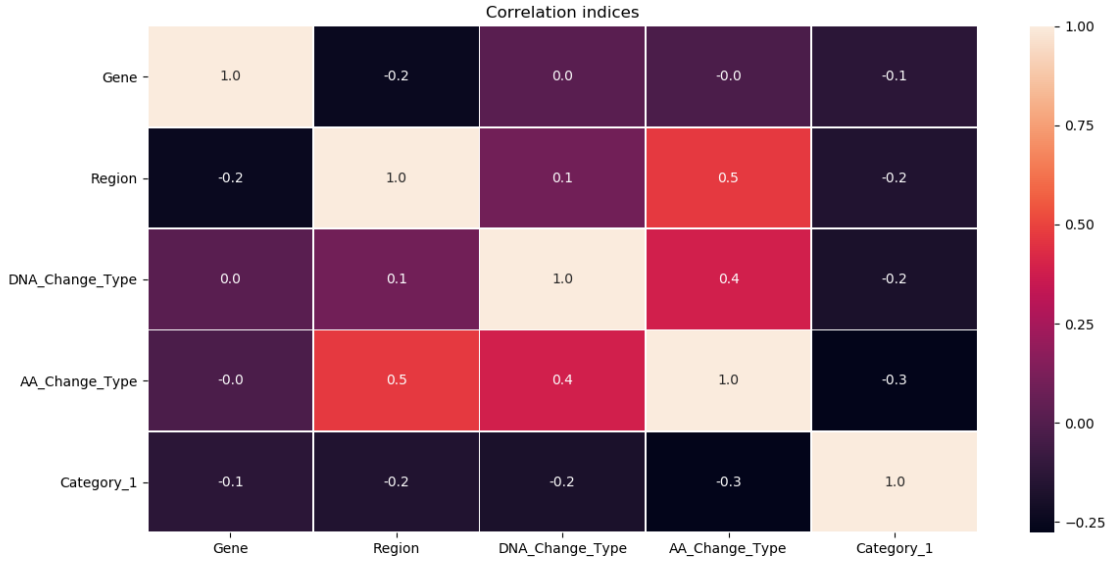

Figura 7: Gráfico de correlación entre todos los atributos

Sobre el gráfico anterior se concluye que la región en la que se produce la mutación y el tipo de cambio de aminoácido hay una correlación de un 0.5 y entre el tipo de cambio de aminoácido y tipo de cambio de ADN de un 0.4. El resto de las relaciones entre atributos no es relevante.

Para comenzar con los distintos algoritmos de Machine Learning, en un principio se consideran los atributos anteriormente desglosados como los atributos de interés. El atributo objetivo sería Disease puesto que buscamos un predictor para el síndrome de Lynch.

4.1. Limitaciones de los datos

Como se ha detectado en el apartado *3. Recolectar los datos,* las fuentes de datos para poder desarrollar este proyecto son escasas. Además, algunas de las descartadas presentaban otros problemas por los que no resultaban útiles para este estudio.

Aunque finalmente se haya elegido la base de datos Canadian Open Genetics Repository por considerar la más adecuada, este conjunto de datos presenta varios problemas a tener en cuenta.

La cantidad de muestras (304) no es suficiente para aplicar algoritmos de machine learning, además como se ha demostrado a lo largo de este apartado, hay algunos atributos cuyos valores no están equilibrados.

Para poder mitigar estos problemas e intentar evitar el sobreentrenamiento se van a aplicar dos métodos que mejoraran los resultados esperados, aunque no se resuelvan completamente: [6]

a) **Retención de datos**: Dividir el conjunto de datos entre datos de entrenamiento y datos de validación, de esta forma evitamos utilizar el mismo conjunto de datos para el entrenamiento y la evaluación de los algoritmos. Así se puede detectar cuando el algoritmo deja de generalizar y comienza a sobreajustarse con los datos de entrenamiento.

b) **Validación cruzada**: El proceso de cross-validation consiste en realizar análisis estadísticos como: la media, varianza... de los subconjuntos de datos para detectar como va evolucionando el rendimiento a través de los distintos conjuntos de datos. La validación cruzada mejora los resultados cuando la cantidad de datos es limitada, ya que calcula las estimaciones sobre el conjunto de datos completo a través de múltiples divisiones entre datos de entrenamiento y de test.

5. Procesar los datos

Partiendo de la muestra y antes de aplicar los distintos algoritmos, hay que preparar los datos para que se puedan entrenar. Los dos problemas por corregir en los datos son los siguientes:

- **Valores nulos:** Los atributos *AA Change Type* y *Disease* presentaban valores nan para ciertos registros.

  En el caso de *AA Change Type* los valores nulos se han clasificado como desconocido (‘UNKNOWN`).

  Para la clase objetivo, únicamente estaba etiquetados los registros que presenta el síndrome, con la etiqueta ‘Lynch Syndrome’, por lo que las etiquetas nulas se han etiquetado como ‘Healthy’.

  Este primer caso fue corregido en el punto anterior para poder representar las gráficas de exploración de datos de forma correcta.

- **Codificar atributos categóricos:** En el caso de este estudio, todos los atributos son categóricos, por ello hay que codificar las distintas categorías representadas con una cadena de texto por números enteros. Estos números van desde 1 hasta n, siendo n el número de categorías posibles dentro de un atributo.

  Para este proceso, se ha utilizado el LabelEncoder de sklearn.[4]

Además, ha sido necesario crear la matriz de diseño, formado por los atributos: 'Gene', 'Region', 'DNA_Change_Type', 'AA_Change_Type', y 'Category_1'. Y el vector objetivo formado por la clase Disease_1'.

La matriz de diseño está formada por tantas columnas como atributos que se utilicen como datos de entradas y filas según el número de registros. El vector objetivo es la clase que se va a predecir, es un vector de tantos registros como filas de la matriz de diseño.

La relación entre el conjunto de entrenamiento y test es de un 80 / 20 %. Para realizar la separación de los subconjuntos se ha utilizado la función: *train_test_split* con el parámetro test_size a 0.2. Esta relación del 80/20 indica que un 80% de los datos se va a utilizar para los procesos de entrenamiento del modelo y el 20% para la parte de test. La división de los dos subconjuntos se realiza de forma aleatoria.

Además, se ha aplicado la validación cruzada de K iteraciones o K-fold cross-validation. El valor de *k* elegido ha sido 5, ese valor indica el número de iteraciones que va a realizar el algoritmo con una quinta parte del conjunto total de datos. De esos subconjuntos *k-1* son utilizados para entrenamiento y 1 para test. La selección de este valor viene dada a que

[4] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

al menos cada subconjunto de datos contenga entre un 15-20 % de los datos totales.

El conjunto de datos tiene un total de 402 registros, si lo dividimos en 5 subconjuntos, cada iteración utilizaría un 19.90% del total de datos (80 entrada de datos).

Antes de entrenar cada algoritmo hay que hacer este preprocesamiento de los datos, en los algoritmos utilizados en este trabajo siempre se realizan los pasos explicados en este apartado, por lo que no se va a especificar de nuevo en cada uno.

6. Valoración de los resultados obtenidos

Para la comparación de los resultados que se vayan obteniendo a lo largo del desarrollo de este trabajo siempre se van a utilizar los mismos valores calculados a partir de la matriz de confusión. La matriz de confusión se representa con la siguiente estructura: [7]

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	Verdaderos positivos (VP)	Falsos negativos (FN)
	Síndrome de Lynch	Falsos positivos (FP)	Verdaderos negativos (VN)

Figura 8: Representación de la matriz de confusión

A partir de la matriz de confusión se calculan lo siguientes valores:

La sensibilidad obtenida (o tasa de verdaderos positivos):

$$Sensibilidad = \frac{VP}{(VP + FN)}$$

La especifidad (o tasa de positivos falsos):

$$Especifidad = \frac{VN}{(VN + FP)}$$

La exactitud (accuracy):

$$Exactitud = \frac{(VP + VN)}{(VP + VN + FP + FN)}$$

7. Algoritmo: K-Nearest-Neighbors

El método *knn* o k vecinos más cercanos es un algoritmo de aprendizaje supervisado. Clasifica cada dato según la clase a la que pertenezcan sus *k* vecinos más cercanos. Es decir, calcula la distancia de sus vecinos al dato a clasificar y los ordena según la distancia de menor a mayor, para tener en cuenta el orden de los *k* elementos primeros.

7.1. Entrenamiento del modelo

En este apartado se pretende estudiar las distintas posibilidades que el algoritmo *KNeighborsClassifier*[5] de sklearn ofrece.

Para realizar un estudio completo y con la mayor variabilidad posible se han configurado de distintas formas dos parámetros del clasificador:

- **n_neighbors**: Número de vecinos para el clasificador. Este valor es fundamental en la definición del algoritmo, con él se especifica el número de vecinos a tener en cuenta para realizar el cálculo de las distancias.
- **algorithm**: Tipo de algoritmo que utiliza el clasificador para calcular la distancia a los vecinos: brute, ball_tree y kd_tree.

  brute, es el algoritmo de fuerza bruta y consiste en calcular la distancia a todos los datos del clasificador.

  ball_tree realiza divisiones de los puntos de datos en conjunto de hiperesferas anidados.

  kd_tree realiza particiones de datos en un espacio euclídeo de *k* dimensiones.

De todos los parámetros que ofrece este clasificador, se ha considerado estos dos ya que el resto sirven, principalmente, para mejorar el rendimiento del algoritmo y la velocidad de procesamiento.

Se ha aplicado cada uno de los tres algoritmos para un rango de vecinos de 1 a 40 en los subconjuntos de entrenamiento. Este rango resulta suficiente para comprobar el comportamiento y la tendencia del clasificador.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

```
def apply_kneighbors(neighbors, X_train, X_test, y_train, y_test,
algorithm):
    knn = KNeighborsClassifier(
           n_neighbors=neighbors, algorithm=algorithm
    )
    knn.fit(X_train, y_train)
    accuracy_test = knn.score(X_test, y_test)

    pred = knn.predict(X_test)

    accuracy_cross_test = cross_val_score(knn, X, y, cv=5)

    return accuracy_cross_test.mean()
```

Figura 9: Llamada al clasificador 'knn'[6]

A continuación, se muestra una gráfica con los resultados obtenidos de las distintas iteraciones aplicando cada uno de los tres algoritmos.

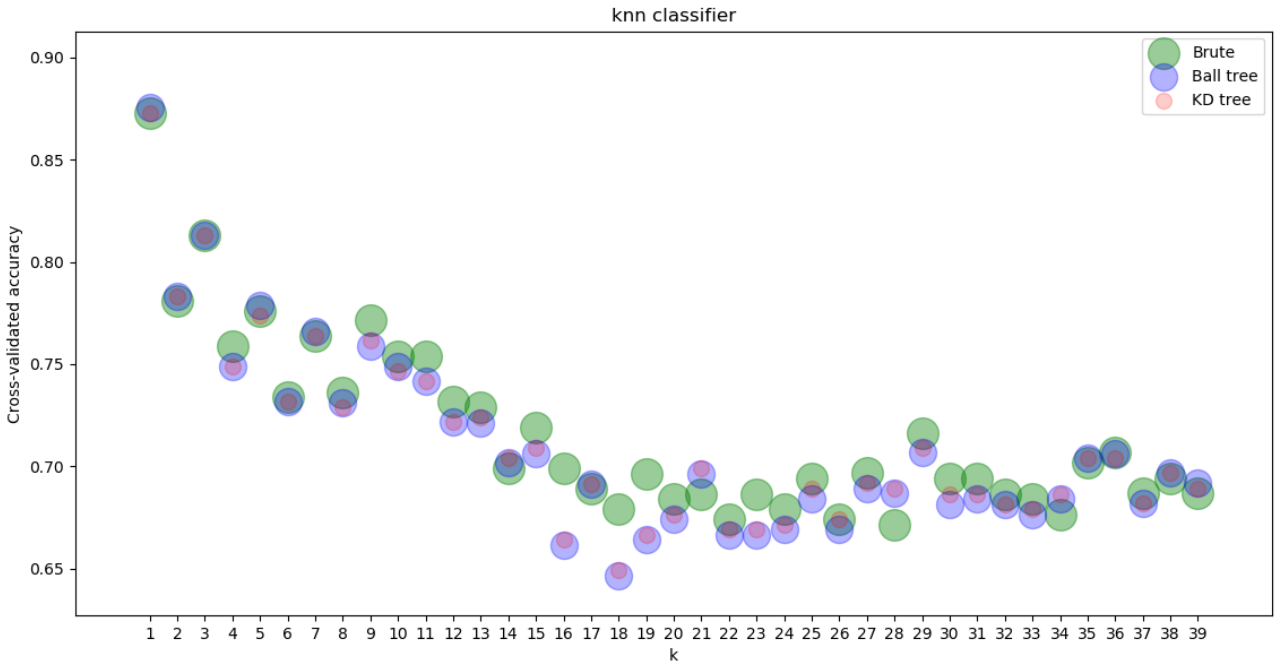

Figura 10: Gráfica de exactitud con cross-validation para distintos parámetros de knn

Como se puede observar en la gráfica (**Figura 8**) los mejores valores de exactitud los obtenemos para el valor de k=1. Como se observó en el análisis inicial esta situación puede ser debido a la correlación entre la clase y el atributo 'Category'. Este valor indica que el dato más similar (cercano) al que se está clasificando es de la misma clase.

[6] https://github.com/PauAndrio/TFM_Marta_Munoz/blob/master/kneighbors.py

7.2. Evaluación del modelo

Para comprobar cuál de los tres algoritmos es el que resultados más óptimos proporciona, se ha obtenido la matriz de confusión para cada uno de ellos con el valor *k=1*.

1. ‘brute’

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**19**	FN=11
	Síndrome de Lynch	FP=1	VN=**50**

2. ‘ball_tree’

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**22**	FN=8
	Síndrome de Lynch	FP=5	VN=**46**

3. ‘kd_tree’

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**22**	FN=8
	Síndrome de Lynch	FP=5	VN=**46**

En la siguiente tabla resumen se muestran los valores a tener en cuenta para la comparación de los resultados para cada uno de los tres algoritmos para k=1:

	brute	ball_tree	kd_tree
Exactitud	**0.85**	0.84	0.84
Sensibilidad	0.63	**0.73**	**0.73**
Especificidad	**0.98**	0.90	0.90

Figura 11: Valores obtenidos de la matriz de confusión para los distintos algoritmos de knn

8. Algoritmo: Support Vector Machines

Los algoritmos SVM son modelos de aprendizaje supervisado. Se basa en la organización de puntos (datos) en el espacio, separando las clases en espacios lo más amplios posibles mediante hiperplanos de separación definidos como el vector entre 2 puntos, cada uno de estos vectores se llama vector soporte.

Todo el código utilizado en el desarrollo de este apartado se localiza en: https://github.com/PauAndrio/TFM_Marta_Munoz/blob/master/svm.py

8.1. Entrenamiento del modelo

Para el caso del paquete *svm* de sklearn se ha comparado *LinearSVC*[7] con *SVC*[8] probando distintos tipos de kernel.

El algoritmo *LinearSVC* es simplemente un SVM con kernel lineal, los resultados obtenidos deberían ser muy similares a los de aplicar *SVC* con el parámetro kernel 'linear', pero las diferencias obtenidos se deben a la implementación interna de las funciones.

Función	kernel	Exactitud obtenida
LinearSVC	-	0.7664506172839507
SVC	linear	0.8656481481481482
SVC	rbf	0.865679012345679
SVC	poly	0.8055555555555556

Figura 12: Tabla comparativa de exactitud para distintos algoritmos de SVM

En este caso el mejor valor obtenido es el clasificador SVC con los kernel 'linear' o 'rbf'. De todas formas, resulta interesante comprobar con distintos grados para el kernel polinómico que por defecto se configura con grado 3.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[8] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

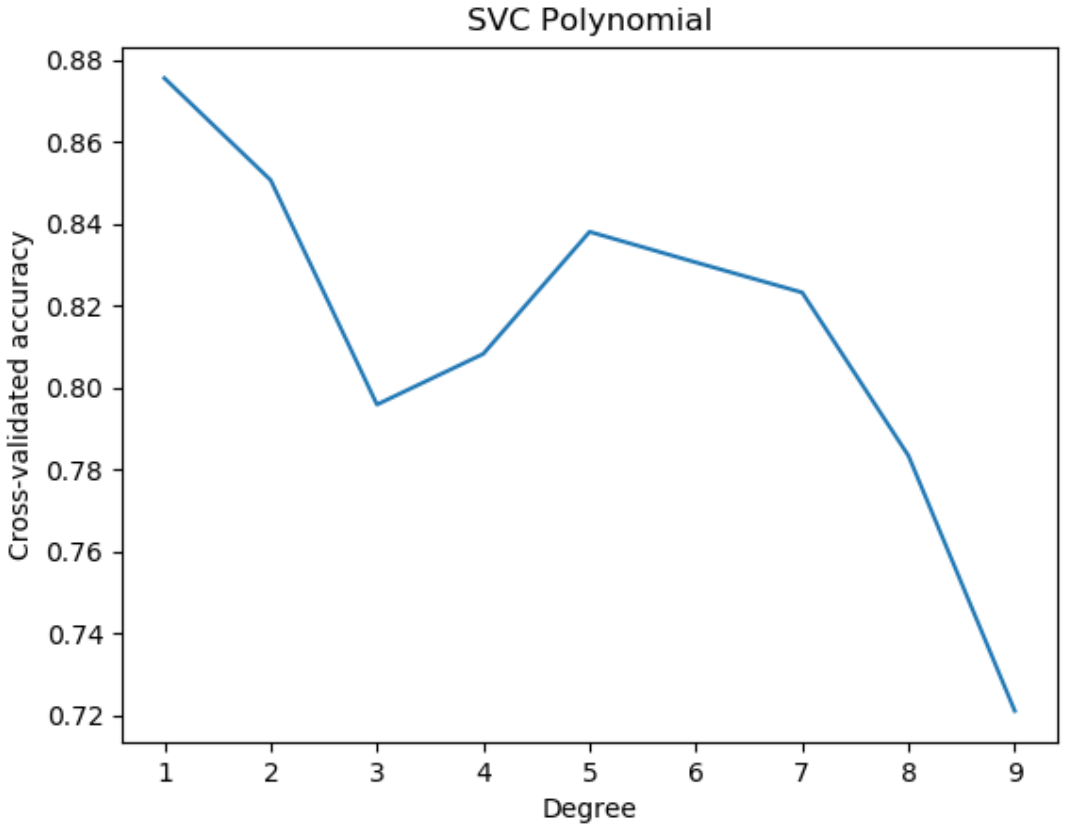

Figura 13: Gráfica de exactitud con cross-validation para distintos grados de SVM polinomial

El mejor valor de exactitud se obtiene con el polinomio de grado 1, es decir, una función lineal. Los resultados son similares a los obtenidos con svc y el kernel 'linear', como se ha podido observar en la tabla comparativa de la página anterior que ya indicaba que los mejores valores de exactitud se obtenían con el kernel 'linear' o 'rbf'.

8.2. Evaluación del modelo

La matriz de confusión que obtendríamos para el mejor caso de los estudiados en SVM (SVC con kernel 'poly' y grado 1), sería la siguiente. Como se ha comentado anteriormente, serían los mismos resultados obtenidos con kernel 'linear'.

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**25**	FN=17
	Síndrome de Lynch	FP=0	VN=**39**

Y los valores de interés a tener en cuenta en el comparador de todos los algoritmos serían:

	SVC polinomial (degree=1) o SVC linear
Exactitud	0.79
Sensibilidad	0.595
Especificidad	1.0

9. Algoritmo: Decision Tree

Los árboles de decisión son un algoritmo de aprendizaje inductivo, se basa en el descubrimiento de patrones a partir de ejemplos.

Un árbol de decisión comienza con un único nodo y luego se ramifica en resultados posibles. Cada uno de esos resultados crea nodos adicionales, que se ramifican en otras posibilidades.

Algunas características de los árboles de decisión son:

- Cada nodo que no sea una hoja representa un atributo, denominado nodo de decisión.
- Las ramas que salen de un nodo etiquetado con un atributo están etiquetadas con cada uno de los posibles valores del atributo del nodo del que parten.
- Cada hoja se corresponde con un valor de la clasificación, nodo-respuesta.

Todo el código utilizado en el desarrollo de este apartado se localiza en: https://github.com/PauAndrio/TFM_Marta_Munoz/blob/master/decision_tree.py

9.1. Entrenamiento del modelo

Para el entrenamiento con este tipo de algoritmo se ha elegido la función *DecisionTreeClassifier*[9] del paquete *tree*.

Entre la cantidad de parámetros disponibles para configurar el clasificador se han elegido los siguientes:

- **criterion**
- **class_weight**
- **min_samples_split**
- **min_samples_leaf**
- **max_depth**

Estos parámetros son los seleccionados ya que son los que influyen en los resultados de exactitud del entrenamiento. Hay otros parámetros configurables, pero en su mayoría se refieren al rendimiento y no influyen en la exactitud.

[9] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier

criterion

Con este parámetro se selecciona entre dos posibles algoritmos para medir la calidad de la división de cada rama. Se puede elegir entre 'gini' para utilizar la *Impureza de Gini* y 'entropy' para la *Ganancia de Información*.

- **Impureza de Gini**: indica para un nodo del árbol de decisión la probabilidad de no acertar con la clasificación que se realizase de forma aleatoria para un dato cualquiera.
- **Ganancia de Información**: o entropía, puede ser considerada como una medida de la incertidumbre y de la información necesaria para, poder acotar, reducir o eliminar la incertidumbre. Es decir, una medida para saber cuan de valioso es el nodo para la clasificación final.

En el caso de estudio se ha decidido aplicar los dos tipos para comparar la exactitud obtenida en ambos casos.

criterion	**Score**
entropy	0.9425617283950618
gini	0.9175617283950617

Como se puede observar los resultados obtenidos son mejores para 'entropy', además al obtener la matriz de confusión también obtenemos el mismo resultado:

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**17**	FN=2
	Síndrome de Lynch	FP=3	VN=**59**

En la matriz de confusión se obtienen 17 casos de paciente sano y 59 casos para pacientes que presenta el síndrome, ambos predichos correctamente. Obtenemos 1 individuo mal diagnosticado como enfermo y 3 casos con el síndrome que son diagnosticados como sanos.

class_weight

Este parámetro consigue equilibrar las diferencias de peso que presente la clase que se utilicen en el algoritmo. Es decir, se da un peso proporcional a las categorías de menor ocurrencia.

Al observar la clase utilizada, en este caso 'Disease', que indica si un individuo es sano para el Síndrome de Lynch o por el contrario presenta el síndrome, se obtiene una diferencia considerable:

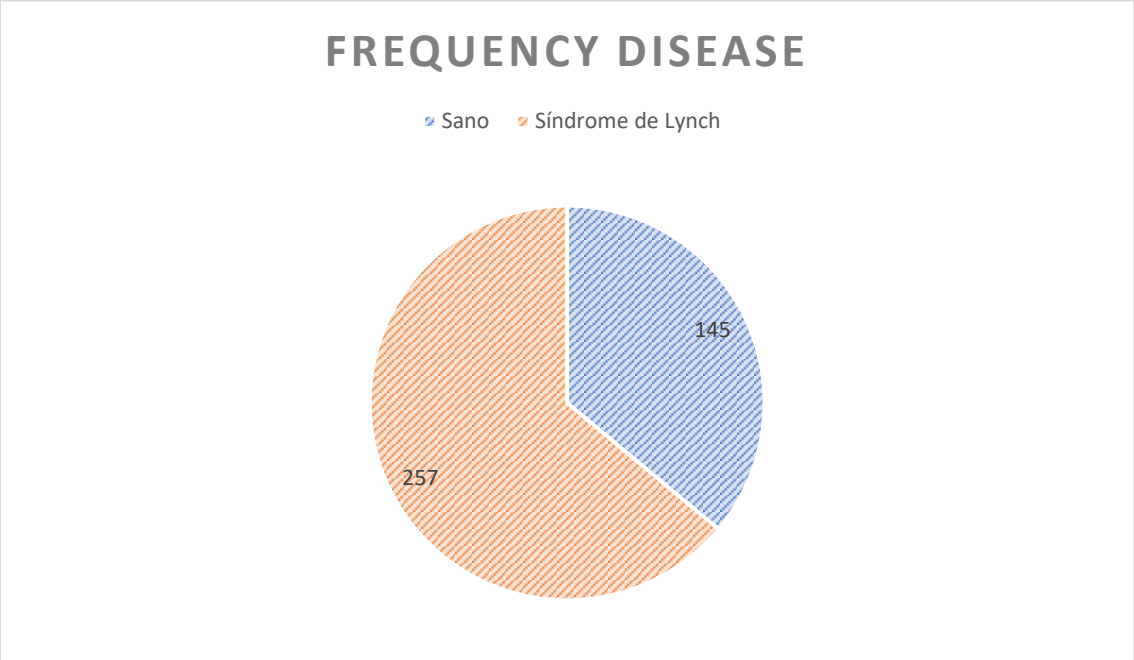

Figura 14: Gráfica comparativa de resultados para la clase 'Disease'

Para compensar el desbalance de la clase calculamos el ratio de proporción entre los pacientes que presentan el síndrome (257) y pacientes que son sanos (145), obteniendo una proporción de 1.77.

Por lo que el parámetro *class_weigth* lo configuramos con el siguiente diccionario: {0:1.77, 1:1.0}. Donde se indica que los valores 0 de la clase (sanos) van a tener un peso de 1.77, mientras que los etiquetados con el valor 1 (presentan el síndrome) su peso va a ser de 1.

Al obtener la media de los valores de accuracy obtenidos a partir del cross validation, el resultado de la media es: **0.9175617283950617**

min_samples_split

Número mínimo de muestras a considerar para realizar una división de un nodo interno.

Para probar las distintas posibilidades de este parámetro se ha configurado un rango de 5 a 25. El rango se ha definido teniendo en cuenta que se realiza validación cruzada de 5 iteracciones, por lo que un

número mayor a 20 no daría lugar a realizar apenas ramificaciones en el árbol (se parte de un subconjunto de 80 elementos), se ha elevado hasta 25 para comprobar el comportamiento del algoritmo. Un tamaño mínimo de partición de cada hoja menor que 5 no sería significativo para el conjunto de datos, debido a la variabilidad y la cantidad de atributos que se manejan.

Se ha representado en un gráfico los distintos valores de exactitud medios obtenidos con cross validation para 5 iteracciones. A continuación, se muestra el gráfico obtenido:

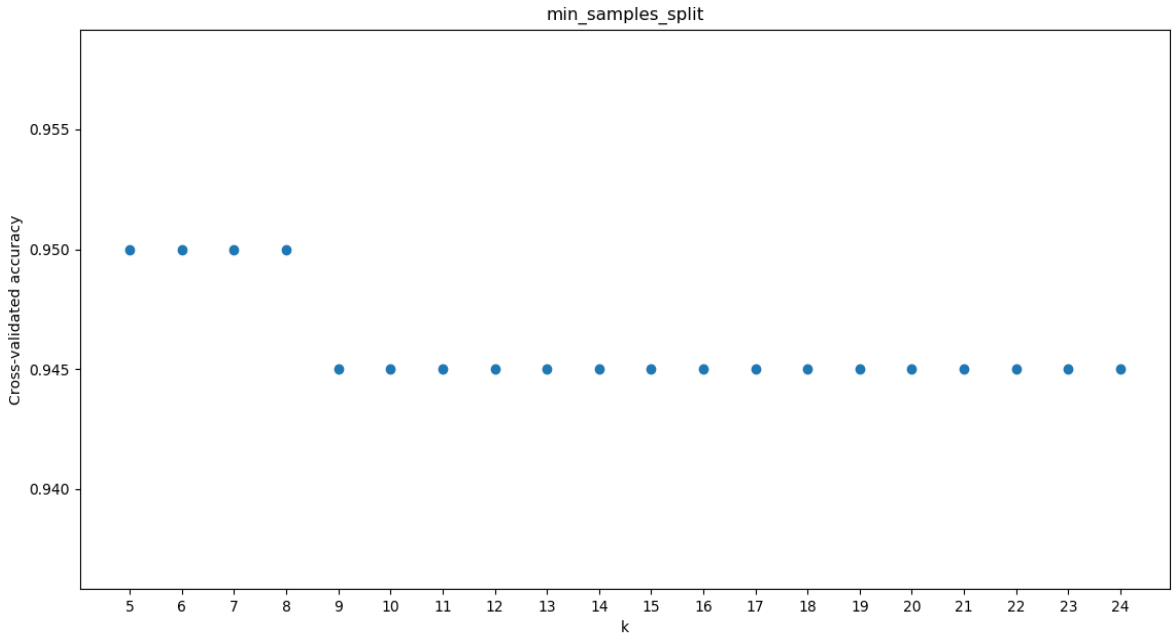

Figura 15: Gráfico de exactitud media con cross-validation para valores de 'min_samples_split' para 'Decision tree'

Como se observa los valores obtenidos del valor 5 al 9 coinciden y son mayores que los obtenidos desde el 9 en adelante.

min_samples_leaf

Número mínimo de muestras a considerar para estar en un nodo hoja (final). Dado un nodo que va a dividirse, solo dará lugar a una división si deja al menos *min_samples_leaf* muestras de entrenamiento en cada una de las ramas izquierda y derecha.

Para estudiar las posibilidades de este parámetro se procede de igual forma que en el caso de *min_samples_split*. Se han obtenido los valores de exactitud configurando el parámetro *min_samples_leaf* en el rango de 1 a 20. Este rango se ha seleccionado debido a que cada iteración de cross-validation utiliza el aproximadamente 80 registros (son 5 iteraciones

de 403 registros), por lo que un número mayor que 20 no tiene sentido puesto que supondría muy pocos nodos.

La gráfica que se muestra a continuación muestra los distintos valores de accuracy medio obtenido por cross-validation para todos los valores de rango de 1 a 20:

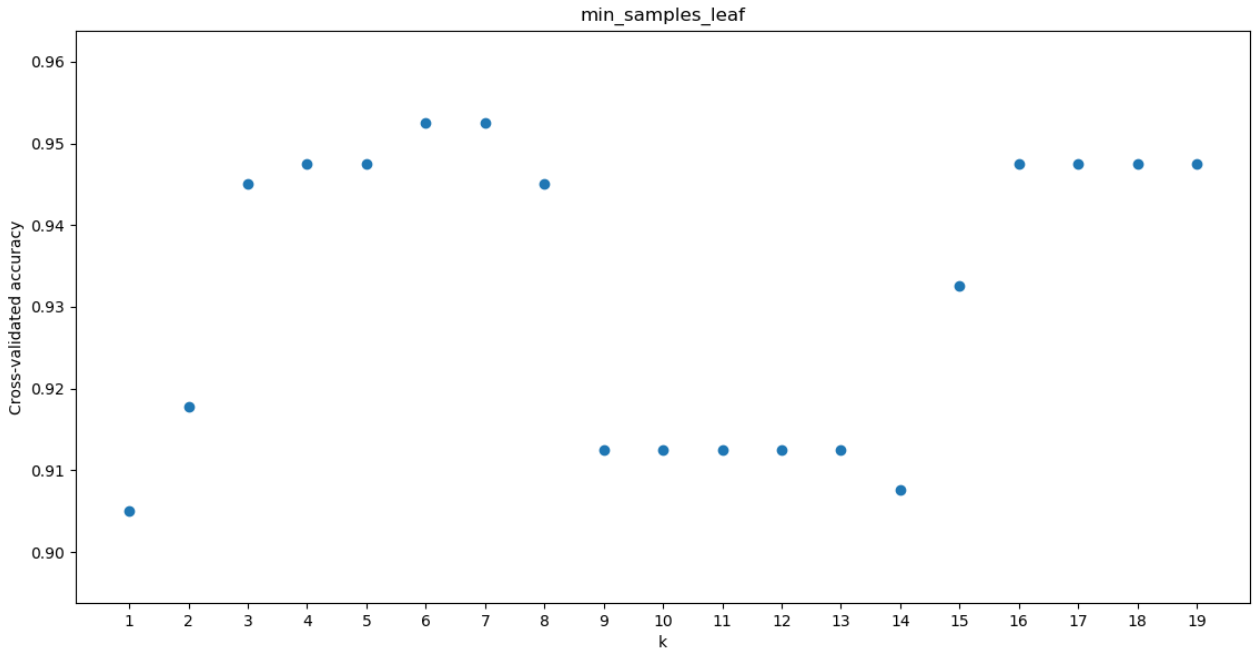

Figura 16: Gráfico de exactitud media con cross-validation para valores de 'min_samples_leaf' para 'Decision tree'

Como se observa los mayores rangos de exactitud se encuentran entre los valores de *min_samples_leaf* 6 y 7. Al ser un clasificador binario y tener la variable 'Category' altamente relacionada con la clase con hojas de 6 o 7 elementos es suficiente.

max_depth

Máximo nivel de profundidad del árbol. La profundidad de un árbol es el número de aristas desde la raíz del árbol hasta un nodo.

Se procede igual que en los dos parámetros estudiados anteriormente, se prueban distintos valores de *max_depth* y para cada uno de estos valores se obtiene su valor de exactitud. En este caso el valor *max_depth* se ha probado en el rango de 1 a 20.

El gráfico obtenido se muestra a continuación:

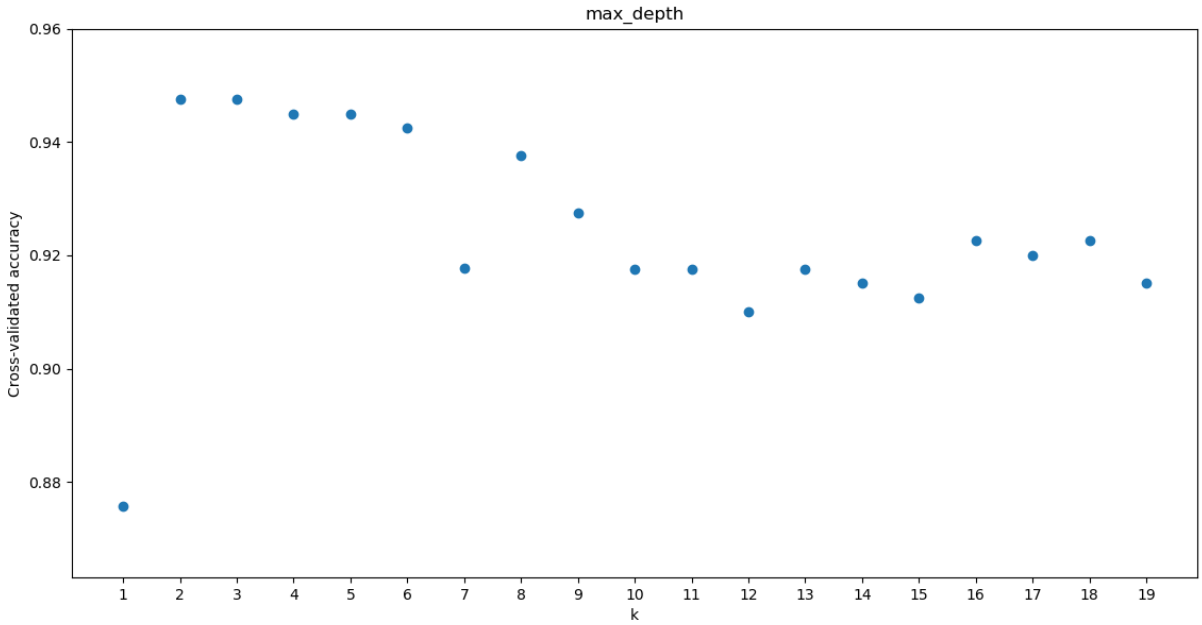

Figura 17: Gráfico de exactitud media con cross-validation para valores de 'max_depth' para 'Decision tree'

En esta ocasión el mayor score obtenido se presenta con el valor para *max_depth* de 2 y 3. De igual forma que en los anteriores casos, debido a la muestra pequeña de datos y ser un clasificador binario, en poca profundidad se resuelve el árbol en su totalidad.

9.2. Evaluación del modelo

A lo largo de las pruebas de los distintos parámetros se ha ido añadiendo a un diccionario de datos los mejores valores obtenidos para cada parámetro. A continuación, se muestra el diccionario obtenido:

Parámetro	Mejor valor obtenido
'criterion'	'entropy'
'class_weight'	{0: 1.77, 1: 1.0}
'min_samples_split'	5
'min_samples_leaf'	6
'max_depth'	2

Como se ha comentado a lo largo de la explicación de cada parámetro, en algunos casos varios posibles valores han presentado la misma exactitud, en este caso se ha elegido el primero de los valores probados.

La exactitud obtenida utilizando estos parámetros ha sido: **0.9475**.

La matriz de confusión obtenida para estos valores es:

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**19**	FP=0
	Síndrome de Lynch	FN=6	VN=**56**

Los resultados obtenidos a partir de la matriz:

	Decision Tree
Exactitud	0.93
Sensibilidad	0.90
Especificidad	1.0

Además, se ha utilizado una herramienta para representar el árbol de decisión de forma gráfica.

Se utiliza *export_graphviz*, un método que permite exportar los resultados obtenidos de una clasificación de un árbol de decisión al formato DOT de Graphviz[10]. También se utiliza el método *graph_from_dot_data* para crear el gráfico y exportarlo a un archivo PNG.

El árbol de decisión obtenido se muestra a continuación, se puede observar todos los nodos y hojas que se han ido formando para el algoritmo. De esta forma se visualiza como se ha ido formando el clasificador y todos los pasos.

[10] https://graphviz.gitlab.io/documentation/

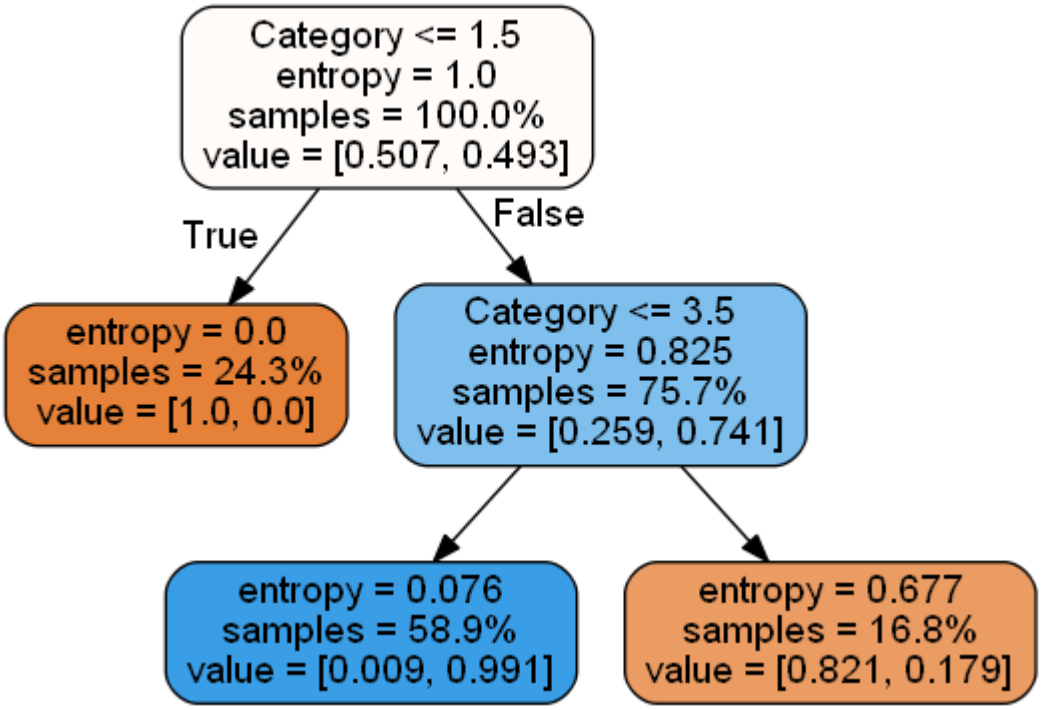

Figura 18: Árbol de decisión obtenido

En cada nodo se obtiene una serie de valores:

- **Samples**: indica la cantidad de individuos por hoja, en porcentaje.
- **Value**: el primer valor son los individuos sanos de la muestra y el segundo los individuos que presentan el síndrome (tanto por 1).
- **Entropy**: el valor de información ganada.

A continuación, se especifica los resultados obtenidos del árbol concreto:

- En la raíz se parte de todos los individuos, de los cuales un 50% es sano y un 49.3% presenta el síndrome.
- El nodo a la izquierda segrega del nodo raíz los que cumple el valor de categoría menor o igual a 1.5, representa un 24.3% del total. Además, todos los individuos de este nodo no presentan el síndrome. Este nodo es nodo hoja y por lo tanto final.
- El nodo a la derecha selecciona todos los individuos que cumple que el valor de categoría es mayor a 1.5, representa un 75.7%. De esta hoja un 25.9% no presenta el síndrome y un 74.1% si. Se puede observar que el valor ganado de este nodo es de 82.5 %.
- En el último nivel, a la izquierda tenemos una hoja que serían todos los individuos cuyo valor categoría es menor a 3.5. Representan un 58.9% del total de individuos. Un 99.1% manifiesta el síndrome de Lynch.

- En el último nivel a la derecha, tenemos una hoja que representa un 16.8% de la muestra total, en el que están los individuos cuya categoría es mayor a 3.5. Un 82.1% no manifiesta el síndrome y un 17.9% sí.

10. Algoritmo: Random Forest

Random Forest es un algoritmo predictivo que usa la técnica de Bagging (promediar modelos ruidosos, pero aproximadamente imparciales y así reducir la variación) para combinar un conjunto de diferentes árboles (Decision Tree), donde cada árbol es construido con observaciones y variables aleatorias. [8]

Todo el código utilizado en este apartado se encuentra en: https://github.com/PauAndrio/TFM_Marta_Munoz/blob/master/random_forest.py

10.1. Entrenamiento del modelo

Igual que en el algoritmo de 'Decision Tree', se han evaluado distintos valores para una serie de parámetros disponibles para el clasificador: *RandomForestClassifier*[11].

Entre la cantidad de parámetros disponibles para configurar el clasificador se han elegido los siguientes:

- **n_estimators**
- **max_features**
- **min_samples_split**
- **min_samples_leaf**

Como se observa todos los parámetros son compartidos con el algoritmo 'Decision Tree', excepto n_estimators.

n_estimators

Este parámetro se refiere al número de árboles a crear que va a contener el 'bosque'.

Como se ha procedido en anteriores pruebas de parámetro, se va a considerar un rango y obtener los valores de exactitud de cada uno de los casos para elegir un valor óptimo. En este caso el rango de prueba se ha ampliado de 1 a 200.

Los resultados obtenidos se muestran en el siguiente gráfico.

[11] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

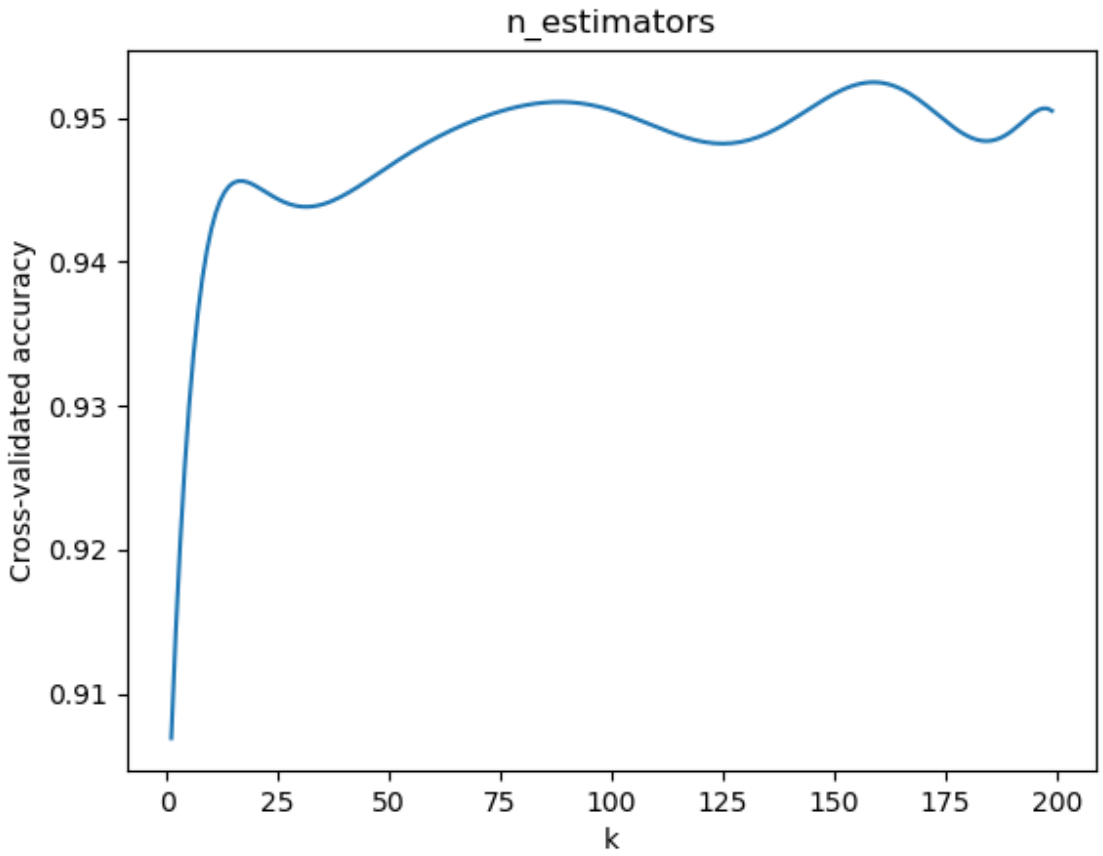

Figura 19: Gráfico de exactitud media con cross-validation para valores de 'n_estimators' para 'Random Forest'

Como se observa en este gráfico al principio la exactitud va aumentando hasta que empieza a estabilizarse. Teniendo en cuenta el tamaño de la muestra de datos tiene sentido, en un principio se presupone que cuando mayor este valor, mejor exactitud se obtiene.

max_features

Con este parámetro se configura el máximo de variables que puede contener un árbol de decisión.

En esta ocasión se ha probado con un rango que va de 0.10 a 1 (en pasos de 0.1), además se le añade otro parámetro que sería 'sqrt' y se refiere al valor de raíz cuadrada de las variables disponibles. Hay que tener en cuenta que por defecto el valor de número de árboles de un bosque (n_estimators) es 10.

Los resultados obtenidos se muestran en la siguiente gráfica:

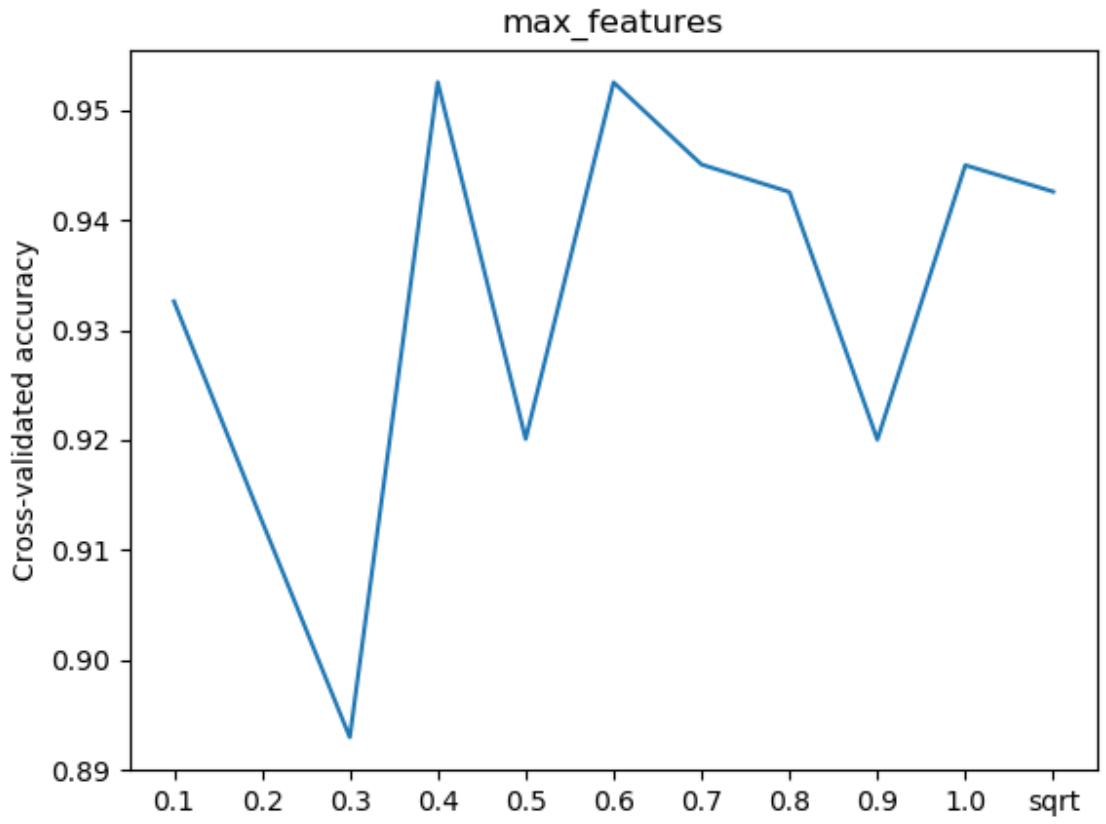

Figura 20: Gráfico de exactitud media con cross-validation para valores de 'max_features' para 'Random Forest'

El valor más alto obtenido es 0.6 que serían unas 3.6 variables por iteración (0.6*6 variables disponibles), lo que supone algo más de la mitad.

min_samples_leaf

Este parámetro ya se ha utilizado en el algoritmo de Decision Tree. Se refiere al número mínimo de muestras a considerar para estar en un nodo hoja (final). Dado un nodo que va a dividirse, solo dará lugar a una división si deja al menos *min_samples_leaf* muestras de entrenamiento en cada una de las ramas izquierda y derecha.

El rango en el que hemos probado es de 1 a 50.

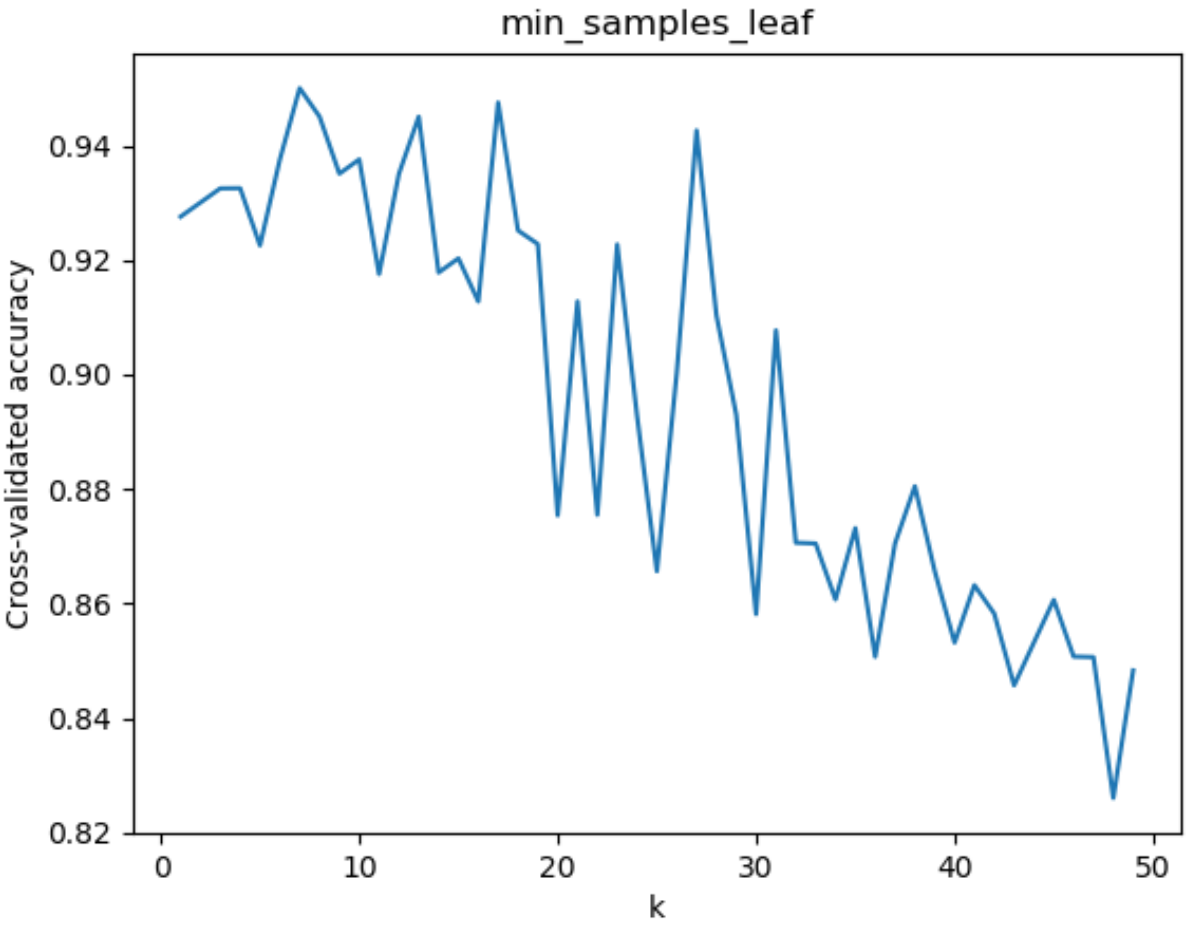

Figura 21: Gráfico de exactitud media con cross-validation para valores de 'min_samples_leaf' para 'Random Forest'

La tendencia en este gráfico es claramente descendiente, desde el valor 1, que sería el mínimo valor posible.

Hay que tener en cuenta que por defecto el valor de número de árboles de un bosque (n_estimators) es 10. Además, se hace k-fold-cross-validation con k igual a 5. Por lo que la cantidad de datos utilizados en cada árbol es muy reducida, por eso los valores van descendiendo con el aumento de número de datos por hoja.

min_samples_split

También se ha utilizado para el algoritmo 'Decision Tree', del apartado anterior. Número mínimo de muestras a considerar para realizar una división de un nodo interno.

Para probar las distintas posibilidades de este parámetro se ha configurado un rango de 2 a 50 y se ha representado en un gráfico los distintos valores

score obtenidos para cada valor del rango. A continuación, se muestra el gráfico obtenido:

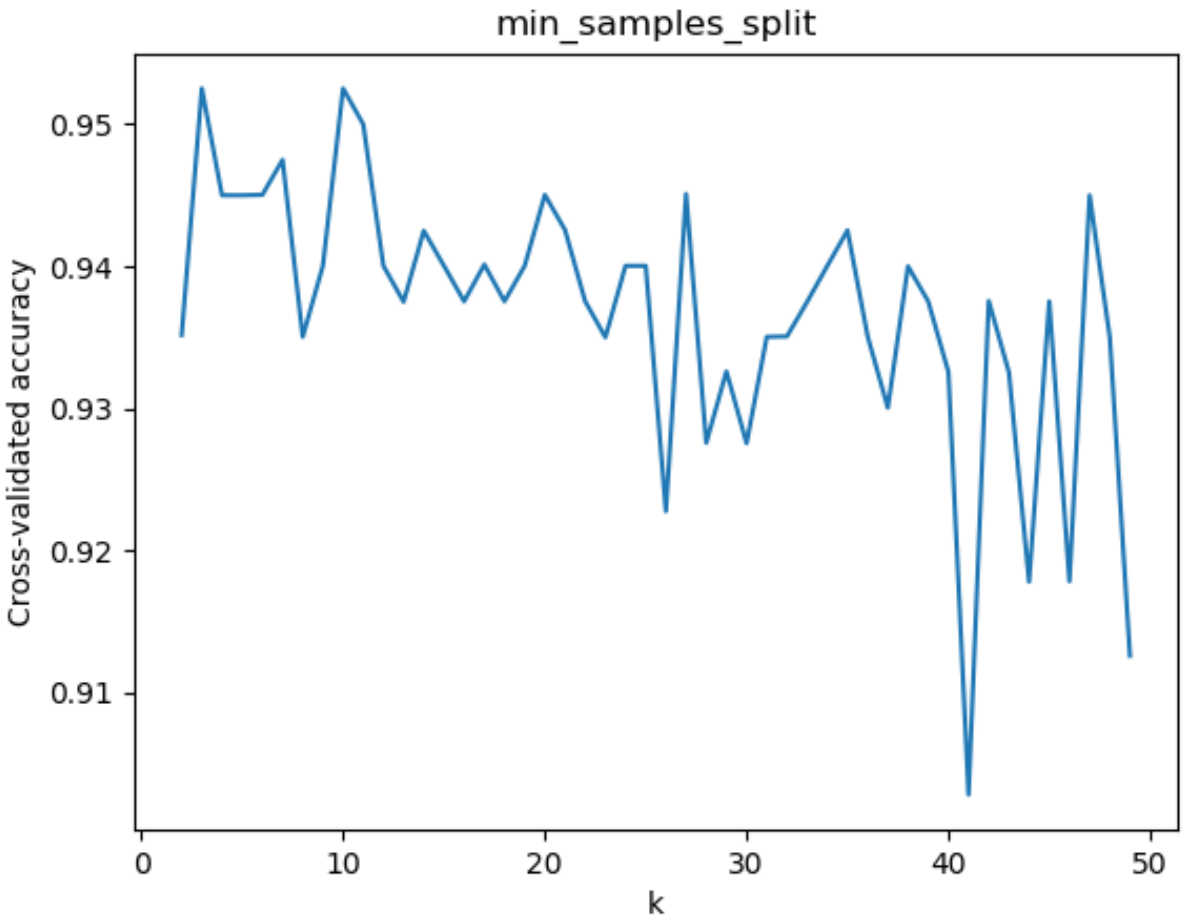

Figura 22: Gráfico de exactitud media con cross-validation para valores de 'min_samples_split' para 'Random Forest'

Como todos los parámetros configurables para el caso de estudio, una restricción importante es el tamaño de la muestra de datos, por eso, con valores pequeños el accuracy obtenido es mejor.

Ocurre lo mismo que con el parámetro anterior teniendo en cuenta que no se ha configurado el número de árboles (por defecto es 10) y con la configuración utilizada en cross-validation, es necesario que las restricciones sobre el número de datos que se manejan en cada división sea mínima.

10.2. Evaluación del modelo

De igual forma que para 'Decision Tree', se han ido almacenando los valores de cada parámetro que proporcionaban una exactitud mayor para poder ejecutarlos en una llamada conjunta.

Los valores obtenidos para cada uno se especifican en la siguiente tabla, como se ha comentado con anterioridad estos datos se han probado de forma aislada, quizás otro estudio interesante sería ir anidando los distintos parámetros y de esta forma observar la dependencia entre ellos.

Parámetro	Mejor valor obtenido
'n_estimators'	72
'max_features'	0.6
'min_samples_leaf'	1
'min_samples_split'	11

El valor de exactitud obtenido, utilizando estos parámetros, ha sido: **0.9425**.

La matriz de confusión obtenida para estos valores es:

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**38**	FP=0
	Síndrome de Lynch	FN=3	VN=**50**

Los resultados obtenidos a partir de la matriz:

	Decision Tree
Exactitud	0.967
Sensibilidad	0.93
Especificidad	1.0

Además, se ha utilizado la función *ExtraTreesClassifier*[12] para poder visualizar la importancia de las variables para el modelo.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

Para mostrar de forma visual esta clasificación se ha utilizado un gráfico de barras.

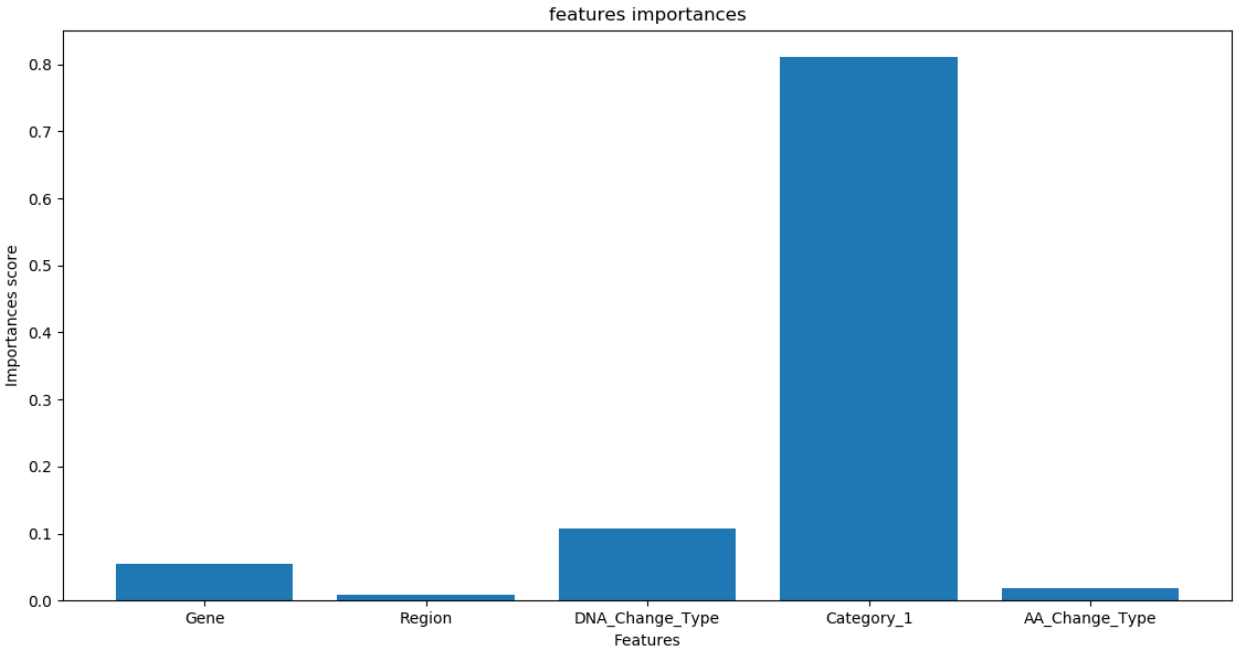

Figura 23: Importancia de cada atributo para el modelo

En esta gráfica se ve la importancia que aporta el atributo ‘Category_1’, mientras que ‘Region’ da poco valor al modelo. Sería interesante realizar todos los algoritmos con distintos conjuntos de atributos.

11. Algoritmo: Bernoulli Naive Bayes

Es un clasificador Bayesiano ingenuo, un clasificador probabilístico fundamentado en el teorema de Bayes y algunas hipótesis simplificadoras, de ahí el nombre ingenuo.[13]

El teorema de Bayes se define:

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{P(B)}$$

Expresa la probabilidad condicional de un evento aleatorio A dado B en términos de la distribución de probabilidad condicional del evento B dado A y la distribución de probabilidad marginal de sólo A.

Un clasificador de Bayes ingenuo asume que ninguna característica es dependiente de otra, todas son independientes y no están relacionadas entre sí.

11.1. Entrenamiento del modelo

Para este algoritmo se ha utilizado *BernoulliNB*[14] del paquete naive_bayes de sklearn. De los posibles parámetros configurables no se ha considerado ninguno de interés, por lo que se ha entrenado el modelo con el conjunto de datos.

```
from sklearn.naive_bayes import BernoulliNB
…

bnb = BernoulliNB()
bnb.fit(X_train, y_train)
accuracy_bnb = cross_val_score(bnb, X, y).mean()
print('Score: BernoulliNB {}'.format(accuracy_bnb))
```

Figura 24: Entrenamiento del modelo 'BernoulliNB' [16]

11.2. Evaluación del modelo

Con el conjunto de datos que se está evaluando durante todo el desarrollo del trabajo y el algoritmo BernoulliNB, configurado con todos los parámetros por defecto, se obtiene un accuracy de **0.7119878191886571**.

La matriz de confusión obtenida para estos valores es:

[13] https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html
[14] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html
[15] https://github.com/PauAndrio/TFM_Marta_Munoz/blob/master/mix_algorithms.py

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**15**	FP=10
	Síndrome de Lynch	FN=0	VN=**56**

Los resultados obtenidos a partir de la matriz:

	Bernoulli Naive Bayes
Exactitud	0.876
Sensibilidad	1.0
Especificidad	0.85

12. Algoritmo: Linear Discriminant Analysis (LDA)

El Análisis Discriminante Lineal es un método de clasificación supervisado de variables cualitativas en el que dos o más grupos son conocidos a priori y nuevas observaciones se clasifican en uno de ellos en función de sus características. Haciendo uso del teorema de Bayes, LDA estima la probabilidad de que una observación, dado un determinado valor de los predictores, pertenezca a cada una de las clases de la variable cualitativa:

$$P(Y = k|X = x)$$

Finalmente se asigna la observación a la clase k para la que la probabilidad predicha es mayor. [9]

12.1. Entrenamiento del modelo

En este caso el algoritmo utilizado es el implementado en scikit-learn: *LinearDiscriminantAnalysis*[16]. En este caso también se han utilizado las variables por defecto

```
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
…

lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
accuracy_lda = cross_val_score(lda, X, y).mean()
```

Figura 25: Entrenamiento del modelo 'LinearDiscriminantAnalysis' [18]

12.2. Evaluación del modelo

La precisión media obtenida con cross-validation es: **0.768368405812763**

La matriz de confusión obtenida para estos valores es:

[16] https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html
[17] https://github.com/PauAndrio/TFM_Marta_Munoz/blob/master/mix_algorithms.py

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**15**	FP=10
	Síndrome de Lynch	FN=3	VN=**53**

Los resultados obtenidos a partir de la matriz:

	Linear Discriminant Analysis
Exactitud	0.96
Sensibilidad	0.83
Especificidad	0.84

13. Algoritmo: Quadratic Discriminant Analysis (QDA)

El clasificador cuadrático *QDA* se asemeja en gran medida al *LDA*, con la única diferencia de que el *QDA* considera que cada clase *k* tiene su propia matriz de covarianza ∑k y, como consecuencia, la función discriminante toma forma cuadrática:

$$\log\big(P(Y=k|X=x)\big) = -\frac{1}{2}\log\left|\sum k\right| - \frac{1}{2}(x-\mu_k)^T\sum_k^{-1}(x-\mu_k) + \log(\pi^k)$$

Para poder calcular la posterior probabilidad a partir de esta ecuación discriminante es necesario estimar, para cada clase, ∑k, μk y πk a partir de la muestra. Cada nueva observación se clasifica en aquella clase para la que el valor de la *posterior probabilidad* sea mayor. *QDA* genera límites de decisión curvos por lo que puede aplicarse a situaciones en las que la separación entre grupos no es lineal. (Rodrigo, Análisis discriminante lineal (LDA) y Análisis discriminante cuadrático (QDA), 2016)

13.1. Entrenamiento del modelo

En este caso el algoritmo utilizado es el implementado en scikit-learn: *QuadraticDiscriminantAnalysis*[18].

Para este algoritmo ningún parámetro ha resultado interesante para evaluar. Por lo que se procede directamente entrenarlo. Procediendo a utilizar la función fit una vez definido el clasificador:

```
from sklearn.discriminant_analysis import
QuadraticDiscriminantAnalysis
…

qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)
accuracy_qda = cross_val_score(qda, X, y).mean()
print('Score: QDA {}'.format(accuracy_qda))
```

Figura 26: Entrenamiento del modelo 'QuadraticDiscriminantAnalysis' [20]

13.2. Evaluación del modelo

[18] https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html
[19] https://github.com/PauAndrio/TFM_Marta_Munoz/blob/master/mix_algorithms.py

El score obtenido para el conjunto de datos ha sido: **0.8131456335698283.**

La matriz de confusión obtenida para estos valores es:

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**23**	FP=2
	Síndrome de Lynch	FN=10	VN=**46**

Los resultados obtenidos a partir de la matriz:

	Quadratic Discriminant Analysis
Exactitud	0.85
Sensibilidad	0.70
Especificidad	0.96

14. Algoritmo: Gaussian process classification (GPC)

Clasificación de procesos gaussianos (GPC) basada en la aproximación de Laplace.

Se enfoca en modelar las probabilidades posteriores, definiendo ciertas variables latentes: f_i es la variable latente para el patrón i.

Si se considera un caso de dos clases (binario): f_i es una medida de pertenencia a la clase C_1, por lo tanto:

- Si f_i es positivo y grande, entonces el patrón i pertenece a la clase C_1 con una gran probabilidad.
- Si f_i es negativo y grande, entonces el patrón i pertenece a la clase C_2 con una gran probabilidad.
- Si f_i es cercano a 0, no se está seguro a la clase que pertenece.

[10]

14.1. Entrenamiento del modelo

Para este algoritmo se ha utilizado: *GaussianProcessClassifier*[20]. Se ha utilizado el algoritmo con todos los parámetros configurados por defecto.

Por lo tanto, se procede a definir el clasificador y entrenarlo.

```
from sklearn.gaussian_process import
GaussianProcessClassifier
…

gpc = GaussianProcessClassifier()
gpc.fit(X_train, y_train)
accuracy_gpc = cross_val_score(gpc, X, y).mean()
print('Score: GaussianProcessClassifier
{}'.format(accuracy_gpc))
```

Figura 27: Entrenamiento del modelo 'GaussianProcessClassifier' [22]

14.2. Evaluación del modelo

El score obtenido para el conjunto de datos ha sido: **0.7982947972474026**

[20] https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessClassifier.html
[21] https://github.com/PauAndrio/TFM_Marta_Munoz/blob/master/mix_algorithms.py

La matriz de confusión obtenida para estos valores es:

		Predicciones	
		Sano	Síndrome de Lynch
Reales	Sano	VP=**20**	FP=5
	Síndrome de Lynch	FN=5	VN=**51**

Los resultados obtenidos a partir de la matriz:

	Gaussian process classification
Exactitud	0.88
Sensibilidad	0.80
Especificidad	0.91

15. Comparación de todos los algoritmos

A pesar de la limitación debido a la cantidad insuficiente de datos, se ha procedido probando los distintos algoritmos. Los resultados obtenidos comparando entre algoritmos son los esperados, partiendo de las necesidades del problema. Es decir, el propósito del trabajo ha sido realizar un clasificador del síndrome de Lynch (binario) utilizando como entrada de datos 6 atributos categóricos.

En la siguiente tabla se puede observar la exactitud de cada algoritmo estudiado, cada uno se ha configurado de forma más óptima posible.

Algoritmo	Exactitud
KNN	0.85
SVM	0.86
Decision Tree	**0.95**
Random Forest	0.94
Bernoulli Naive Bayes	0.71
LDA	0.77
QDA	0.81
GPC	0.80

Figura 28: Tabla comparativa de exactitud de los algoritmos

Se observa que todos los valores obtenidos son muy altos. Durante el desarrollo del proyecto, el atributo ‘Category’ marcaba de forma muy relevante la clasificación, puede ser la razón de los altos resultados.

16. Futuras líneas de investigación

Como primera propuesta sería muy interesante y necesario, ampliar la muestra de datos que se ha utilizado, para ello sería necesario que la comunidad científica hiciese públicos más datos de pacientes que se hayan realizado las pruebas de diagnóstico del Síndrome de Lynch.

Por otro lado, se podría enfocar el desarrollo de este trabajo con distintas vertientes. Por un lado, podría resultar de interés descartar ciertos atributos como 'Category', que está altamente relacionado con la clase 'Disease' y comparar los resultados obtenidos con los publicados en este trabajo.

Otra modificación interesante de la matriz de diseño sería descartar los atributos que aportan poca relevancia al modelo, como por ejemplo 'Region'. Para comporar si hay diferencias significativas respecto a los resultados obtenidos.

Otra prueba interesante para comprobar los distintos valores óptimos de los parámetros sería haber ido añadiendo cada parámetro de forma anidada, es decir, una vez obtenido el valor óptimo para un parámetro el siguiente parámetro a testear que hubiese sido dependiente del mejor valor del anterior parámetro. Ya que ciertos parámetros darán un resultado u otro dependiendo de otras configuraciones del algoritmo.

También puede resultar de interés añadir otros tipos de algoritmos para añadir a la comparativa.

En otro ámbito, sería interesante estudiar si en otras especies existe algún tipo de mutación hereditaria que suponga una mayor predisposición a algunos tipos de cáncer.

17. Conclusiones

Como conclusión principal: la importancia de los datos para Machine Learning. A pesar de haber dedicado una parte importante del inicio del trabajo a la búsqueda de datos adecuadas, pronto se concluyó que aun siendo los más adecuados dentro de los estudiados, no iban a ser los más idóneos. El inconveniente principal de la muestra de datos seleccionada es el tamaño de esta, no son datos suficientes para utilizar en Machine Learning. Esta característica ha ido marcando todo el progreso del trabajo, a pesar de intentar paliar los efectos con distintos métodos.

Respecto a los objetivos, se ha logrado estudiar bastantes clasificadores de la biblioteca Scikit-learn, también se han estudiado otras bibliotecas como Pandas o NumPy. Además, se han afianzado conocimientos de Machine Learning adquiridos durante el desarrollo del máster. Se ha realizado un análisis previo de los datos para poder anticipar los inconvenientes que se iban a encarar en los estudios de los algoritmos.

La planificación a seguir se ha ido modificando a lo largo del desarrollo de producto, la principal diferencia ha sido añadir muchos algoritmos a estudiar, cuando al principio hablamos de un único predictor. Por otro lado, por la naturaleza del desarrollo de los problemas de machine learning, los pasos a seguir si se han seguido.

18. Glosario

- Algoritmo: Es una serie de pasos matemáticos u operacionales específicas para resolver un problema o realizar una tarea.
- Análisis de datos: El proceso de obtener una comprensión de los datos mediante la consideración de muestras, mediciones y visualizaciones. El análisis de datos puede ser particularmente útil cuando se recibe por primera vez un conjunto de datos, antes de crear el primer modelo. También es crucial para comprender los experimentos y problemas de depuración del sistema.
- Atributo: Variable de entrada que se usa para realizar predicciones.
- Atributo discreto: Atributo con un conjunto finito de valores.
- Clase: Categoría de salida de los datos. También llamada categorías.
- Clasificación binaria: Tipo de tarea de predicción que da como resultado una de dos clases mutuamente exclusivas.
- Conjunto de entrenamiento(training): Subconjunto del conjunto de datos que se usa para entrenar un modelo.
- Conjunto de prueba (test): Subconjunto dentro del conjunto de datos que se usa para probar un modelo después de que este pasó por la evaluación inicial a través del conjunto de validación.
- Datos categóricos: Atributos que tienen un conjunto discreto de valores posibles.
- Entrenamiento: Proceso de determinar los parámetros ideales que conforman un modelo.
- Exactitud (accuracy): Fracción de predicciones que se realizaron correctamente en un modelo de clasificación.
- Falso negativo (FN, false negative): Ejemplo en el que el modelo predijo de manera incorrecta la clase negativa. Por ejemplo, el modelo infirió que un mensaje de correo electrónico en particular no era spam (la clase negativa), pero ese mensaje de correo electrónico en realidad era spam.
- Falso positivo (FP, false positive): Ejemplo en el que el modelo predijo de manera incorrecta la clase positiva. Por ejemplo, el modelo infirió que un mensaje de correo electrónico en particular era spam (la clase positiva), pero ese mensaje de correo electrónico en realidad no era spam.
- HPNCC: Cáncer colorrectal hereditario no asociado a poliposis o síndrome de Lynch.

- Librería: un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.
- Matriz de confusión: Tabla de N×N que resume el nivel de éxito de las predicciones de un modelo de clasificación; es decir, la correlación entre la etiqueta y la clasificación del modelo.
- Modelo: Es la representación matemática de las relaciones en un conjunto de datos. O lo que es lo mismo, es una forma simplificada y matemáticamente formalizada de aproximarse a la realidad y hacer predicciones a partir de esta aproximación.
- Modelo de clasificación: Tipo de modelo de aprendizaje automático para distinguir entre dos o más clases discretas.
- NumPy: Biblioteca matemática de código abierto que proporciona operaciones entre matrices eficaces en Python. Pandas se basa en Numpy.
- Pandas: API de análisis de datos orientada hacia las columnas. Muchos marcos de trabajo de Aprendizaje Automático admiten las estructuras de datos de Pandas como entrada.
- Parámetro: Es una variable utilizada para recibir valores de entrada en una rutina, subrutina o método.
- Precisión: Porcentaje de casos clasificados correctamente.
- Predicción: Resultado de un modelo cuando se le proporciona un ejemplo de entrada.
- Python: Lenguaje de programación de orientado a objetos que puede aplicarse desde la creación de aplicaciones al desarrollo web.
- Scikit-learn: Plataforma popular de AA de código abierto
- Verdadero negativo (VN): Un ejemplo en el que el modelo predijo correctamente la clase negativa.
- Verdadero positivo (VP): Ejemplo en el que el modelo predijo correctamente la clase positiva**.**

19. Bibliografía

[1] Lynch Syndrome International (LSI). (s.f.). Obtenido de https://lynchcancers.com/

[2] Cancer.Net Editorial Board. (Agosto de 2018). Cancer.Net. Obtenido de https://www.cancer.net/cancer-types/lynch-syndrome

[3] Shi-Long Lu, M. K. (1998). HNPCC associated with germline mutation in the TGF-β type II receptor gene. *Nature*.

[4] The National Comprehensive Cancer Network® (NCCN®) . (s.f.). *NCCN GUIDELINES FOR DETECTION, PREVENTION, & RISK REDUCTION.* Obtenido de https://www.nccn.org/

[5] A, R. C. (Diciembre de 2007). Síndrome de Lynch. Prevalencia, características clínicas y tratamiento. (U. N. (Paraguay)., Ed.) *Mem. Inst. Investig. Cienc. Salud, Vol. 5*.

[6] Briega, R. E. (s.f.). *Matemáticas, análisis de datos y python*. Obtenido de https://relopezbriega.github.io/index.html

[7] Recuero, P. (Enero de 2018). *LUCA: AI Powered Decisions.* Obtenido de https://data-speaks.luca-d3.com/2018/01/ML-a-tu-alcance-matriz-confusion.html

[8] aporras. (Enero de 2015). *Random forest vs simple tree.* Obtenido de https://quantdare.com/random-forest-vs-simple-tree/

[9] Rodrigo, J. A. (Septiembre de 2016). *Análisis discriminante lineal (LDA) y Análisis discriminante cuadrático (QDA)*. Obtenido de https://rpubs.com/Joaquin_AR/233932

[10] Atiya, A. (Febrero de 2011). *Gaussian Processes for Classification.* Obtenido de http://www.ideal.ece.utexas.edu/seminar/GP-austin.pdf

[11] https://developers.google.com/machine-learning/glossary/?hl=es-419

20. Anexos

Los anexos del código se han subido al repositorio GitHub y se encuentran disponibles en el siguiente enlace:
https://github.com/PauAndrio/TFM_Marta_Munoz