Técnicas de aprendizaje automático para la predicción de bacteriemias hospitalarias

Por
Gómez Rodríguez, Diego – Ramos Fuentes, Víctor

UNIVERSIDAD COMPLUTENSE
MADRID

Grado en Ingeniería del Software e Ingeniería Informática
Facultad de Informática

Directores : Óscar Garnica Alcázar y José Manuel Ruiz Giardín

Técnicas de aprendizaje automático para la predicción de bacteriemias hospitalarias

Madrid, 2019–2020

Autorización de difusión y uso

Los autores de este trabajo autorizan a la Universidad Complutense de Madrid a emplear tanto el código como la memoria elaborada, únicamente con fines didácticos y mencionando a los autores del mismo.

Diego Gómez Rodríguez

Víctor Ramos Fuentes

Agradecimientos

En primer lugar, agradecer el apoyo de Óscar y José Manuel por aportarnos la idea y los datos necesarios para realizar este proyecto, a Óscar en particular por su dedicación y apoyo a lo largo del mismo. Por otro lado, a aquellas personas que han estado apoyándonos durante la carrera y a lo largo de este proyecto, pareja, amigos y familiares, gracias por animarnos siempre a continuar.

Sobre TEFLON

Teflon(cc0 1.0(documentación) MIT(código))es una plantilla de LaTeX creada por David Pacios Izquierdo con fecha de Enero de 2018. Con atribuciones de uso CC0.

Esta plantilla fue desarrollada para facilitar la creación de documentación profesional para Trabajos de Fin de Grado o Trabajos de Fin de Máster. La versión usada es la 1.3.

V:1.3 Overleaf V2 with pdfLaTeX, margin 1in, NO-bib

Índice general

Resumen

El objetivo de este proyecto es acelerar el proceso de detección de bacteriemia así como su diagnóstico, ya que una rápida detección de la misma permite un tratamiento temprano, que frena la evolución de la bacteriemia aumentando significativamente la probabilidad de recuperación del paciente. En este proyecto buscaremos un modelo clasificador basado en algoritmos de aprendizaje automático con una tasa de acierto considerable que permita detectar qué pacientes pueden presentar bacteriemia.

Actualmente, se utilizan los hemocultivos para la detección de bacteriemia en la sangre del paciente. Empezaremos limpiando los datos aportados por el Hospital Universitario de Fuenlabrada, clasificando qué atributos resultan más interesantes para el estudio y, posteriormente, los someteremos a una normalización buscando que no haya atributos que se sobrepongan a otros debido a la naturaleza de sus magnitudes. También, aplicaremos las técnicas: *separate class mehtod*, *complete case data*, imputación de datos ausentes y descarte de atributos y/o casos de estudio, con la finalidad de gestionar los datos ausentes. Analizaremos los resultados y escogeremos la mejor técnica para el tratamiento de este tipo de datos.

Para intentar mejorar el modelo, estudiaremos las técnicas auxiliares de *principal component analisys* y *one hot encoder*.

Finalmente, aplicaremos los algoritmos de aprendizaje automático *support vector machine*, *random forest* y *k-nearest neighbors* para generar diferentes modelos de clasificación y estudiaremos cuál de ellos se ajusta mejor a los datos del estudio y obtiene una tasa de acierto más elevada.

Palabras Claves

- Bacteriemias
- Dataframe
- Técnicas de preprocesado de datos
- Sobreajuste y subajuste
- K-Fold Cross Validation

- Principal Component Analysis (PCA)
- Support Vector Machine (SVM)
- Random Forest (RF)
- K-Nearest Neighbors (KNN)
- Hemocultivo

Abstract

In this project we want to accelerate bacteremia detection process and its diagnosis. A quick detection allows for an early treatment, which will reduce the evolution rate. In this project, we will look for a classifier model based on machine learning algorithms with a remarkable accuracy rate that allows for detecting which patients will present bacteremia.

Nowadays, doctors use blood cultures to detect bacteremia in the patient's blood. We will start by cleaning up the data provided by Fuenlabrada's University Hospital, classifying which features are more interesting for this porject. Later, we will normalize the data to avoid features to overlap with others due to the nature of their magnitudes. Also, we will apply *separate class method*, *complete case data*, *missing values imputation* and *discarding instances and/or atributes*, in order to manage the missing values. We will analyse the results and choose the best technique for the treatment of this type of data.

To try to improve the model, we will study the following auxiliary techniques: *principal component analysis* and *one hot encoder*.

Finally, we will apply the following machine learning algorithms: *support vector machine*, *random forest classifier* and *k-nearest neighbor*, to generate different classifier models and check which one fits better with the studied data and obtains a higher accuracy rate.

Keywords

- Bacteremia
- Dataframe
- Pre-Processing Data
- Overfitting y underfitting
- K-Fold Cross Validation
- Principal Component Analysis (PCA)
- Support Vector Machine (SVM)
- Random Forest (RF)
- K-Nearest Neighbors (KNN)

- Accuracy

Capítulo 1

Introducción

1.1. Las bacteriemias

Se define como bacteriemia la presencia de bacterias en la sangre. En condiciones normales la sangre no presenta bacterias, de tal forma que su presencia puede asociarse a infecciones importantes que pueden tener repercusión en la vida de los pacientes que la padecen.

En la figura 1.1, se muestra una instantánea de un vaso sanguíneo de un paciente al que se le ha diagnosticado una bacteriemia, se puede apreciar como dentro del mismo aparecen glóbulos rojos, blancos, y unos cuerpos extraños de color verde que representan a las bacterias que se han introducido debido a la infección.

Figura 1.1: Bacteriemia en torrente sanguíneo
[1]

En cuanto al origen de la bacteriemia, ésta suele estar producida por una infección localizada en algún lugar concreto del organismo. Esta infección localizada, en ocasiones de

forma secundaria, favorece el paso de las bacterias a la sangre produciendo bacteriemia. **Los orígenes más frecuentes** de bacteriemia son infecciones de distintos orígenes como el origen urinario (prostatitis, pielonefritis), origen respiratorio (neumonías), origen vascular (catéteres colocados en vena para administrar medicaciones que se infectan), origen digestivo (infecciones de vía biliar como colecistitis o colangitis), piel y partes blandas (celulitis o miositis), o huesos (osteomielitis). Cuando se desconoce el origen infeccioso de la bacteriemia se habla de bacteriemia primaria o idiopática.

Determinados procedimientos médicos pueden favorecer también el paso de bacterias a la sangre, en pacientes previamente sanos, desde sitios que habitualmente y de forma fisiológica están colonizados por bacterias. Al realizar la maniobra diagnóstica o terapeútica las bacterias pueden pasar a la sangre como por ejemplo con sondajes en la vejiga urinaria o endoscopias del tubo digestivo (colonoscopias).

Así mismo, determinados hábitos como el uso de drogas por vía parenteral pueden favorecer el paso de bacterias de la piel a la sangre produciendo bacteriemias.

Incluso situaciones cotidianas como el cepillado de dientes puede producir bacteriemia transitoria al pasar las bacteriemias de la boca (que en condiciones normales tiene gran cantidad de bacterias) a la sangre si el cepillado es intenso y con sangrado. Las bacteriemias asociadas a estos procedimientos de forma habitual son controladas por el Sistema inmunológico del cuerpo humano. Cuando este control no es suficiente para controlar la bacteriemia es cuando se produce la afectación general del cuerpo humano con o sin aparición de otras infecciones a otros niveles del cuerpo a través de las bacterias que se encontraban en la sangre.

La presencia de bacterias en la sangre puede por tanto tener su origen en infecciones localizadas en cualquier parte del organismo, pero a su vez estas bacterias en la sangre pueden diseminar la infección a otros lugares del cuerpo humano como puede ser válvulas cardíacas (endocarditis), articulaciones (artritis), estructuras óseas (osteomielitis), sistema nervioso central (meningitis y abscesos cerebrales) entre otras.

Acompañando a la bacteriemia se producen una serie de síntomas en el paciente que van desde la presencia de fiebre, escalofríos, tiritona, alteraciones analíticas con elevación de glóbulos blancos, aumento de determinadas proteínas en la sangre como la proteína C reactiva, aumento de la frecuencia cardíaca (taquicardia) o aumento de la frecuencia respiratoria (taquipnea). En conjunto, todo este tipo de respuesta recibe el nombre de **"síndrome de respuesta inflamatoria sistémica"**, que si se acompaña de tensión arterial muy baja (shock) puede producir daños en otros órganos, haciendo que esos órganos dejen de funcionar adecuadamente (insuficiencia renal o insuficiencia cardíaca o insuficiencia respiratoria). En estos casos, la bacteriemia recibe el nombre de **"sepsis severa"** que puede terminar con la muerte del paciente.

En función de la duración de la bacteriemia, las bacteriemias pueden ser bacteriemias continuas (las bacterias están permanentemente en sangre porque el foco de infección se encuentra dentro del torrente sanguíneo como, por ejemplo, en todas las infecciones intravasculares como la endocarditis) o bacteriemias transitorias que son las más frecuentes en las que el foco de origen de la infección está fuera del torrente sanguíneo (por ejemplo, una neumonía) y de forma puntual desde ese foco las bacterias pasan a la

sangre.

En función del tipo de bacterias que producen las bacteriemias, éstas se dividen en bacteriemias por gram positivos, bacteriemias por gram negativos y fungemias.

En función del lugar donde se producen las bacteriemias, éstas se dividen en bacteriemias hospitalarias (las producidas en el hospital), bacteriemias comunitarias (las producidas fuera del hospital) y bacteriemias asociadas a cuidados sanitarios como las que pueden producirse en residencias de ancianos o en pacientes que acuden al hospital de forma puntual a recibir determinados tratamientos o realizarse pruebas diagnóstico-terapeúticas concretas sin quedar ingresados en el hospital. Conocer el lugar donde se ha producido la bacteriemia tiene importancia porque van asociadas a determinados tipos de bacteriemia más agresivas y de más difícil tratamiento. Así en general las bacteriemias hospitalarias suelen ser de más difícil tratamiento que las bacteriemias comunitarias.

En la tabla 1.2 [2], los autores del trabajo describen la asociación entre el tipo de microorganismo de la bacteriemia y el lugar de adquisición de la bacteriemia con las mortalidades asociadas, donde puede observarse que **la mortalidad asociada a bacteriemia oscila desde un 11 por ciento hasta un 37 por ciento** según el lugar y tipo de microorganismo asociado a bacteriemia.

Adquisición de la bacteriemia	Incidencia*	Etiología (%)				Microorganismos principales	Polimicrobiana (%)	Origen** (%)	Mortalidad (%)
		Gram+	Gram–	Hongos	Anaerobios				
Comunitaria	6-10	31	68	0	1	*E. coli* *S. pneumoniae* *S. aureus*	5-6	Urinario (46-53) Respiratorio (12-27) Desconocido (9)	11-16
Asociada a cuidados sanitarios	–	32	64	0,3	3	*E. coli* *S. aureus* *K. pneumoniae*	7-8	Urinario (17-43) Catéter vascular (12-42) Desconocido (12)	20-34
Nosocomial	6	65	25	9,5	0-2	ECN *S. aureus* Enterococos	13-53	Catéter vascular (26-52) Urinario (18-33) Desconocido (16)	27-37

*Expresada en n.º episodios por 1.000 ingresos.
**Origen de la bacteriemia por orden de frecuencia. Finalmente porcentaje de bacteriemias de origen desconocido.
ECN: estafilococos coagulasa negativa.

Figura 1.2: Asociación entre el tipo de microorganismo de la bacteriemia y el lugar de adquisición de la bacteriemia con las mortalidades asociadas [2]

El medio de detectar bacterias en la sangre es a través de los cultivos de sangre en unos frascos que contienen unos medios de crecimiento para dichas bacterias y que se llaman **hemocultivos**. Los frascos con estos medios de crecimiento son de dos tipos: frascos con medio de crecimiento para bacterias aerobias y frascos con medios de crecimiento para bacterias anaerobias.

En el paciente en el que existe sospecha de que pueda tener una bacteriemia (fiebre, escalofríos, tiritona, hipotensión, ...) se extrae sangre y se introduce en los frascos de crecimiento (ver figura 1.3), es decir en los **frascos de hemocultivos**. De forma habitual se extrae un volumen de sangre que oscila entre los 20 y 40 ml de sangre por paciente.

Figura 1.3: Frascos de hemocultivos
[3]

Una vez obtenidos los hemocultivos, estos se introducen en un sistema automatizado (ver figura 1.4) que es capaz de detectar la producción de CO_2 en los frascos donde está la sangre del paciente lo que significa que hay colonias bacterianas multiplicándose. El sistema marca el tiempo en horas que ha tardado en detectar la producción de CO_2. En este caso hablamos de hemocultivos positivos. Posteriormente el microbiólogo cogerá los frascos de **hemocultivos positivos** y con una serie de tinciones y técnicas microbiológicas acabará de identificar el microorganismo causante de la bacteriemia y los antibióticos a los que es sensible. En los hemocultivos positivos todo el proceso hasta la identificación del microorganismo puede llegar a tardar 5 días.

Figura 1.4: Sistema Automatizado
[4]

En el caso que el sistema automatizado no detecte producción de CO_2 en los frascos de hemocultivos, se informa como frascos de **hemocultivos negativos**. Los frascos se dejan hasta 5 días incubándose en el sistema detector de crecimiento hasta darlos definitivamente como negativos si en esos 5 días no ha habido crecimiento.

Los problemas que presenta esta técnica de identificación de bacteriemia vienen dados por los llamados "**hemocultivos falsos positivos**". En estos casos se produce crecimiento bacteriano en los frascos de hemocultivos pero son bacterias que habitualmente provienen de la piel y no de la sangre. Se trata pues de muestras contaminadas. La con-

taminación se da en la fase de extracción de sangre, en la mayoría de los casos por una mala limpieza de la superficie de extracción. Estos "hemocultivos falsos positivos" no son pues diagnósticos de bacteriemia pero suponen un gasto de tiempo y medios hasta su identificación microbiológica, y el posible uso inadecuado de antibióticos por la sospecha de una bacteriemia que no es real.

Debido a la alta mortalidad asociada a bacteriemia y la necesidad de iniciar precozmente un tratamiento antibiótico eficaz, es de interés tratar de predecir precozmente qué pacientes pueden sufrir este tipo de patología que presenta una alta morbilidad y mortalidad.

El juicio de los clínicos a la hora de predecir la bacteriemia es bajo. Para intentar mejorar la predictibilidad de la bacteriemia para decidir si extraer o no hemocultivos se han empleado distintas estrategias como la medición de determinadas sustancias en la sangre que son marcadoras de respuesta inflamatoria sistémica y también la creación de modelos clínicos predictivos de bacteriemia basados en datos clínicos, epidemiológicos y analíticos de los pacientes. Hasta el momento actual no se han encontrado modelos clínicos predictivos realmente útiles, ya que en general son aplicables a grupos extremos de muy alto riesgo de bacteriemia y a grupos de bajo riesgo, pero en general no son aplicables para los grupos de riesgo o probabilidad intermedia.

Y el poder predecir la presencia de bacteriemia antes de tener el resultado de los hemocultivos también es importante de cara a decidir el inicio de tratamiento antibiótico, que se ha demostrado útil en la disminución de la mortalidad cuando se realiza adecuadamente de forma precoz.

En resumen, las bacteriemias son procesos infecciosos que suponen la presencia de bacterias en la sangre y que pueden tener distintos orígenes. Se diagnostican a través del cultivo de la sangre (hemocultivos) cuyo proceso diagnóstico puede llegar a tardar hasta 5 días. Las bacteriemias tienen una elevada morbimortalidad, y se benefician de un tratamiento antibiótico adecuado precoz, no existiendo en el momento actual datos clínicos, analíticos o epidemiológicos que nos permitan predecir su presencia en el momento de la valoración inicial de un paciente.

1.2. Antecedentes

La **Inteligencia Artificial o IA** lleva varios años en auge, transformando nuestra sociedad al completo. De sobra es conocido por todos, el empleo de diversas técnicas en nuestro día a día, como puede ser búsquedas por Internet, sistemas de recomendación como Netflix o Youtube...etc.

En el futuro más próximo, la Inteligencia Artificial se expandirá a sectores como la medicina o el transporte. De hecho, incluyendo un poco de contexto cultural, recientemente Ginni Rometty, directora ejecutiva de IBM, afirmó lo siguiente: *"el 100 por ciento de los puestos de trabajo serán diferentes"*.

Desde la aparición de la Inteligencia Artificial en la década de 1960, no se notó un auge en las publicaciones en este campo hasta el 2013, donde dichos artículos se han disparado

hasta alcanzar cotas muy altas.

El aprendizaje automático, o *machine learning*, es un tipo de IA que se enfoca en técnicas que permitan a las máquinas mejorar en su toma de decisiones de cara a predecir, a partir de nuevos datos distintos a los empleados para el entrenamiento, el comportamiento de dichos datos. Dos de los tipos de algoritmos de aprendizaje automático son : *algoritmo supervisado*, que requiere de un grupo de datos de entrada que le atribuya las entradas y las salidas deseadas, se encarga de encontrar un método para determinar cómo generar dichas salidas. Para ello, el algoritmo detecta patrones en los datos, sintetiza a partir de las observaciones en las entradas y realiza predicciones. Este proceso se rehace de forma continua hasta que la técnica consigue alcanzar una precisión óptima, y *algoritmo no supervisado*, que estudia el conjunto de datos proporcionado como entrada identificando patrones con la salvedad de no contar con una respuesta por parte de los datos para corroborar la veracidad de su predicción. En consecuencia, el modelo intenta reestructurar los datos de entrada para adaptarlo a sus patrones. Este tipo de algoritmo va mejorando a medida que toma decisiones. Sus resultados se ven mejorados cuanto mayor sea el volumen de datos de entrada.

1.3. Objetivos

El objetivo principal de nuestro trabajo es implementar técnicas de aprendizaje automático sobre un conjunto de datos de pacientes dados, para intentar predecir la aparición de bacteriemias hospitalarias, sin necesidad de esperar al resultado de los hemocultivos. Los beneficios que aportaría este trabajo son, por un lado, la posibilidad de comenzar el tratamiento del paciente antes y, por otro, en el caso de que se obtuviera una precisión muy elevada de las técnicas, reducir el número de hemocultivos pues estos solo serían prescritos en los casos en los que la predicción de la técnicas no tuviera una alta fiabilidad.

Los datos han sido aportados por el Hospital Universitario de Fuenlabrada, son valores reales obtenidos en la práctica clínica y se han anonimizado convenientemente para cumplir la Ley de Protección de Datos Personales [5].

Tal y como se puede apreciar en el repositorio de Github proporcionado en la entrega de nuestro trabajo, cada técnica que se ha utilizado aparece de forma independiente en su propio *notebook*. Las técnicas de IA utilizadas son *SVM, RF, KNN y PCA*.

Cada algoritmo contiene diferentes métodos de interpretabilidad para poder entender de una forma más sencilla todos los conceptos que han llevado al algoritmo a tomar la decisión de si un paciente presenta síntomas de presencia de bacteriemias o no.

1.4. Plan de Trabajo

Para el desarrollo completo de este trabajo, se ha dinamizado el proyecto en entregas semanales donde Óscar Garnica nos iba facilitando un feedback sobre nuestro desempeño, dichas entregas se acordaban en la reunión de la semana anterior.

Los pasos a seguir durante el proyecto has sido los siguientes:

a). Durante las primeras semanas nos hemos ido introduciendo en el mundo de la IA y familiarizándonos con el entorno del lenguaje Python, iniciamos los primeros scripts en *Spyder*, pero tras una sugerencia sobre *Jupyter Notebook* y *Jupyter Lab*, decidimos emplear estos últimos por su facilidad para ejecutar paso a paso cada línea de código.

b). Tras familiarizarnos con el lenguaje y el entorno, empezamos a estudiar el comportamiento de las técnicas de forma individualizada para comprender que hacía nuestro algoritmo.

c). Después, se intentaba implementar la técnica indicada en su notebook correspondiente.

d). Al comprobar su funcionamiento, empleábamos métodos de interpretabilidad para profundizar en el conocimiento.

e). Finalmente, se volcaban todos los resultados pertinentes al apartado de conclusiones de cada técnica.

Para comunicarnos y llevar un control sobre las tareas a realizar hemos utilizado las siguientes herramientas:

- Para la comunicación, la herramienta Slack.
- Todo el trabajo que se iba realizando individualmente o en conjunto, se iba subiendo a un repositorio de Github.
- Para llevar el control de todo el proyecto, se creó un tablero en Trello.

Capítulo 2

Tratamiento de datos

2.1. Elaboración de un conjunto de datos sólido

Una de las partes más importantes durante el desarrollo e implementación de un modelo de aprendizaje automático, independientemente del campo en el que se desee aplicar, es elaborar un conjunto de datos factible, fiable y lo suficientemente amplio para facilitar en la medida de los posible el entrenamiento del modelo. Inicialmente, contamos con un conjunto de datos anonimizados suministrados por José Manuel Ruiz Giardín, tutor del trabajo y médico del Hospital Universitario de Fuenlabrada. En dichos datos, se ha realizado un exhaustivo repaso a todas y cada uno de los atributos que contenía el fichero. Tras una puesta en común con ambos directores del proyecto, se decidió eliminar ciertas atributos que se consideraron prescindibles, bien por considerarse irrelevantes para el estudio o por su inclusión en un momento posterior a los resultados del *hemocultivo* ya que funcionarían como predictores del resultado de las técnicas aplicadas. En un primer momento, se contaba con **117** atributos que, tras su análisis, fueron reducidas a **73**.

En las tablas 2.1 y 2.2 se ven reflejados los atributos (*features*) seleccionados para el estudio y los atributos descartados junto con una breve descripción de cada una de ellos.

Cuadro 2.1: Tabla de atributos utilizadas en el estudio

Atributos utilizados en el estudio	
periodo	Año del caso de estudio
mes	Mes del caso de estudio
dia	Día del caso de estudio
edad	Edad de paciente
sexo	Sexo del paciente
trombope	Trombopenia
m_digest	Manipulaciones digestivas

Atributos utilizados en el estudio	
m_genitu	Manipulaciones genitourinarias
m_respir	Manipulaciones respiratorias
m_vascul	Manipulaciones vasculares
Coagulación	Coagulación alterada
so	Análisis sistemático de orina
sedorina	Sedimento de orina
Insrenal	Insuficiencia renal
cirugia	Cirugía previa
diashosp	Días en el hospital a extracción de *hemocultivos*
cateter	Días de colocación del último catéter
cateter1	Tipo de cateter-vía
Clasifica	Clasificación final del *hemocultivo*
Inghosp1m	Ingresos hospitalarios en los últimos 30 días al actual ingreso
Prifrpos	Primer frasco del *hemocultivo* que crece
Inghosp12m	Ingresos hospitalarios de más de 48 horas en los últimos 12 meses sin incluir el actual
Especialid	Especialidad donde se produce la bacteriemia
COMORBIL	Comorbilidad: Presencia de más de una enfermedad simultáneamente
adquisic	Adquisición
Diabetes	Diabetes
durac	Días de fiebre antes de *hemocultivo*
Cardiopatia	Cardiopatia
tas	Tensión arterial sistólica
tad	Tensión arterial diastólica
fc	Frecuencia cardiaca
anhonpol	Microorganismos Anaerobios
Anaerobio	Anaerobios frente resto bacteriemias(incluye polimicrobianas)
Hongos	Hongos frente a las demás bacteriemias
Polimicr	Bacteriemia de origen polimicrobiano
microbpoli	Gérmenes de la bacteriemia polimicrobiana
medio	Medio de crecimiento del verdadero positivo
Enfresp	Enfermedad respiratoria crónica
Neoplasia	Neoplasia activa

Atributos utilizados en el estudio	
primtemp	Primera temperatura en urgencias con la que se extraen *hemocultivos* (Axilar)
frasae	Crecimiento, al menos, en medios aerobios
frasanae	Crecimiento al menos en medios anaerobios
Hepatopatia	Hepatopatía
fiebre	Fiebre en el momento de extraer los *hemocultivos* (Temperatura axilar por encima de $38^{o}C$)
Udvp	7 Adicción a drogas parenterales
hipotens	Hipotensión
frasextr	Frascos de *hemocultivos* extraídos.
Alcoholismo	Paciente alcohólico
Vasopre	Uso de agentes vasopresores en el momento de la bacteriemia
glucosa	Glucemia
Otrascomor	Otras comorbilidades
Intubacion	Necesidad de intubación en el momento de la bacteriemia
Urea	Urea en sangre en mg/dl
enfbasWeinst	Enfermedad de base por Weinstein
RCP	Reanimación cardiaca en el momento de la bacteriemia
creatin	Creatinina
esteroid	Esteroides
Alerta	Grado de consciencia del paciente en el momento de la bacteriemia
pcr	Valor PCR del paciente
drogadic	Drogadicción
origensos	origen sospechado de la bacteriemia a extracción de *hemocultivos*
leuc	Leucocitos
antibiot	Antibioticos
hgb	Hemoglobina
inmunosu	inmunosupresores
ucidiashem	Días en UCI a la extracción del hemocultivo
pmfn	%PMN (leucocito polimorfonuclear)
neutrope	Neutropenia (número anormal de neutrófilos)
sintomas	Síntomas localizadores de la fiebre
stlocal	Síndrome localizador
leucocit	Leucocitosis

Atributos utilizados en el estudio	
plaqut	Plaquetas

Cuadro 2.2: Tabla de atributos descartados del estudio

Atributos descartados	
edada	Edad por grupo
edada75	edad mayor o igual a 75 años
edada85	edad mayor o igual a 85 años
Diasdet	Tiempo de detección en días
microrga	Microrganismo
identif	Identificación
Microrgagrupo	Grupo al que pertenece el microrganismo
Stafcoag	Staphylococcus coagulasa (Tipo de Bacteriemia)
gram	Tinción de Gram
n_frasco	Número de frascos
contamin	Crecimiento de contaminantes
mediocon	Medio de crecimiento del microrganismo contaminante
antibiograma	Anbiograma del microorganismo
antiresist	Antibiograma categorizado
pcrcate	PCR categorica
hbcateg	Hb categorica
comentar	Comentarios
enf_bas1	MIRAR
diagnost	Diagnóstico de la bacteriemia
servicio	Servicio en el que se produce la bacteriemia
urgencias	Hemocultivos extraídos en urgencias
temporal	Cálculo de la temperatura oral en función de la axilar. $T^0oral = T^0axilar + 0,5$
fiebrePitt	Clasificación de temperatura oral escala Pitt
PITT	Escala Pitt en pacientes no de UCI. Valor numérico
metastas	Metástasis
metasta1	Si hay metástasis séptica, en qué lugar se produce
evolucio	Evolución
muerte	Muerte
origen	Origen de la bacteriemia
dxfinal	Diagnóstico final
origenf	Origen de la bacteriemia en el departamento final
origenva	Origen vascular
atbempir	Tratamiento antibiótico empírico
t_empiri	Tratamiento empírico adecuado o inadecuado
ttoesp	Tratamiento específico adecuado al que se cambió
t_especi	Tratamiento específico adecuado o inadecuado
diastto	Días de tratamiento hasta inicio de tratamiento adecuado
t_quirur	Indicación o no del tratamiento quirúrgico
uci	Hemocultivos extraídos en UCI

Atributos descartadas	
motuci	Motivo de ingreso en UCI
desmotuci	Motivo de ingreso en UCI escrito
consfieb	Consulta por fiebre
dest	Destino
vuelta_a	Vuelta a urgencias
tratamie	Tratamiento Antibiótico

Tras este exhaustivo repaso a toda la información facilitada por el Hospital Universitario de Fuenlabrada y justo antes de empezar a implementar nuestros modelos, debemos adecuar los datos para facilitar el entrenamiento de las técnicas.

Antes de empezar con el tratamiento de datos, realizamos un estudio sobre el número de datos perdidos (*missing*) en cada atributo para utilizarlo como referencia a la hora de escoger la técnica de preprocesado más adecuada. La figura 2.1 muestra el porcentaje de datos perdidos para cada atributo del archivo de datos filtrados, los atributos contienen entre un 20 % y un 30 % de datos perdidos.

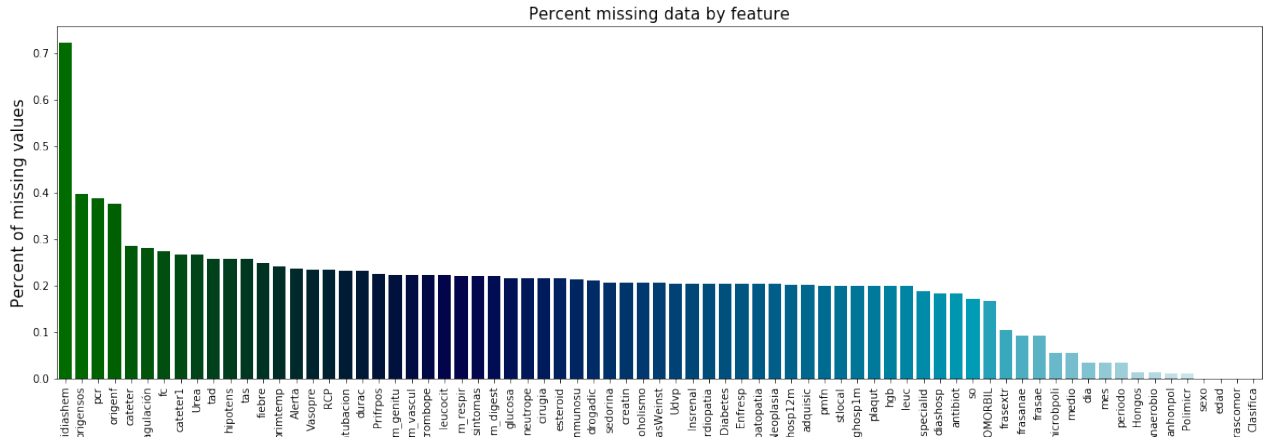

Figura 2.1: Estudio de datos perdidos de nuestro conjunto de datos de estudio

2.2. Separate Class Method

Debido a la presencia de gran cantidad de datos perdidos en nuestro *dataframe* (Véase figura 2.1) y al requerimiento por parte de algoritmos de aprendizaje automático de trabajar con datos completos, sometimos nuestros datos a *Separate class method* [6], una de las técnicas más recomendadas por su impacto final en los resultados del modelo.

Esta técnica se basa en definir una nueva categoría para representar los datos perdidos de una atributo. Esto es, cada atributo posee su propia categoría para representar sus datos perdidos. Por ejemplo, dado el atributo F con categorías c_1, c_2, c_3 y datos perdidos, una vez aplicada la técnica, las categorías pasarían a ser c_1, c_2, c_3 y c_4 siendo c_4 la categoría representante de los datos perdidos. En la figura 2.2 se puede ver un ejemplo en el que la

categoría *color* puede clasificar en 3 colores: *verde, azul* y *rojo*. Una vez aplicada la técnica la categoría clasificaría las variables en *verde, azul, rojo* y *none*. *None* representaría los valores perdidos que puedan aparecer.

caso	color
1	verde
2	rojo
3	verde
4	azul
5	
6	azul

Caso	Color
1	verde
2	rojo
3	verde
4	azul
5	none
6	azul

Figura 2.2: Ejemplo separate class method

En cambio, en el caso de los atributos de tipo numérico debemos darle un valor que esté fuera del rango de los valores que toma el atributo, de este modo estaremos creando la separación necesaria entre los datos perdidos y los valores presentes en el *dataframe*. En nuestro caso, inicialmente escogimos el valor -3000 ya que no teníamos valores negativos en nuestro *dataframe* y nos permitía crear esa separación citada anteriormente entre los valores existentes y los datos perdidos.

2.3. Complete Case Data

La técnica *Complete case data* [6] elimina los casos de estudio de nuestro *dataframe* en los que aparezcan datos perdidos, es decir, eliminaría aquellos casos de estudio que presenten atributos vacíos. Esta fase es necesaria ya que los algoritmos de aprendizaje automático que serán utilizados requieren que no haya datos perdidos. El uso de esta técnica no permitiría que, una vez obtenido un modelo entrenado, probado y con un buen nivel de exactitud, evaluar un caso nuevo con atributos vacíos. Este método viene incluido en gran cantidad de paquetes de estudio estadístico y es utilizado de manera predeterminada en gran cantidad de programas.

De cara al estudio de tratamiento de datos, se descarta esta técnica debido al reducido número de casos completos resultantes, en el caso de los datos sin filtrar solo resultaban 3 casos completos y una vez los datos fueron filtrados 476 casos completos.

2.4. Descartando casos y/o atributos

Debido al gran volumen de datos perdidos con el método *complete case data* se buscaron alternativas que permitiesen cubrir los datos perdidos pero mantener gran parte de los casos de estudio de nuestros datos. Este método consiste en estudiar el alcance de los datos perdidos en cada caso y atributo. Posteriormente elimina progresivamente los casos y los atributos que más datos perdidos presenten [7].

Basándonos en la técnica de *complete case data* y en los resultados obtenidos tras su aplicación, desarrollamos un algoritmo para mejorarlos, reduciendo el número de atributos y maximizando el número de observaciones completas. Más concretamente, el algoritmo ordena los atributos en orden decreciente respecto al número de datos perdidos, va iterando sobre el número de atributos que componen el *dataframe* de entrada y en cada iteración calcula el número de casos completos. Antes de volver a iterar, borramos la siguiente atributo que más datos perdidos presente. Al finalizar, escogemos la relación número de casos - número de atributos que más datos nos aporte. En la figura 2.3 se puede observar el punto óptimo cerca de 50 atributos, pero para obtener el punto óptimo de manera más exacta el algoritmo utiliza el producto entre atributos y pacientes completos, eligiendo de este modo la relación *Pacientes completos - número de* atributos que más datos aporte. La figura 2.4 muestra los resultados obtenidos.

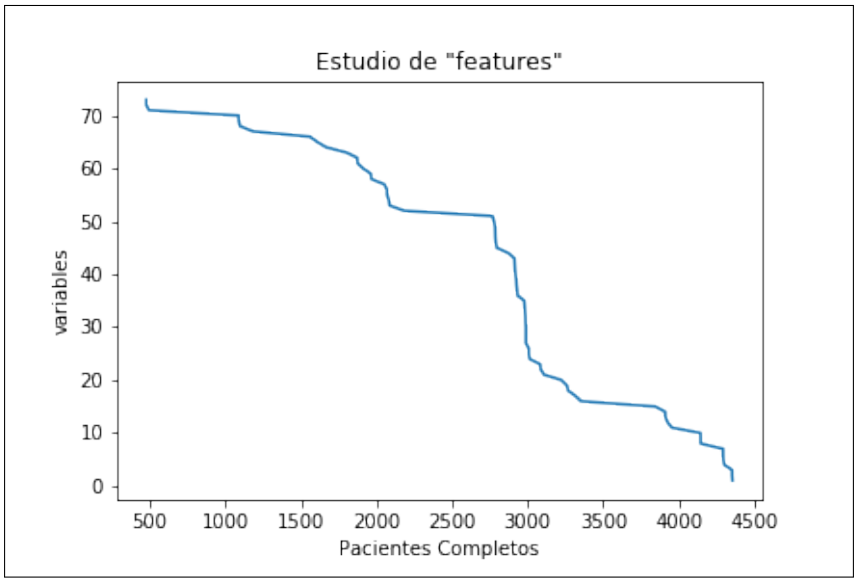

Figura 2.3: Análisis del número de casos después de descarte de casos y/o atributos

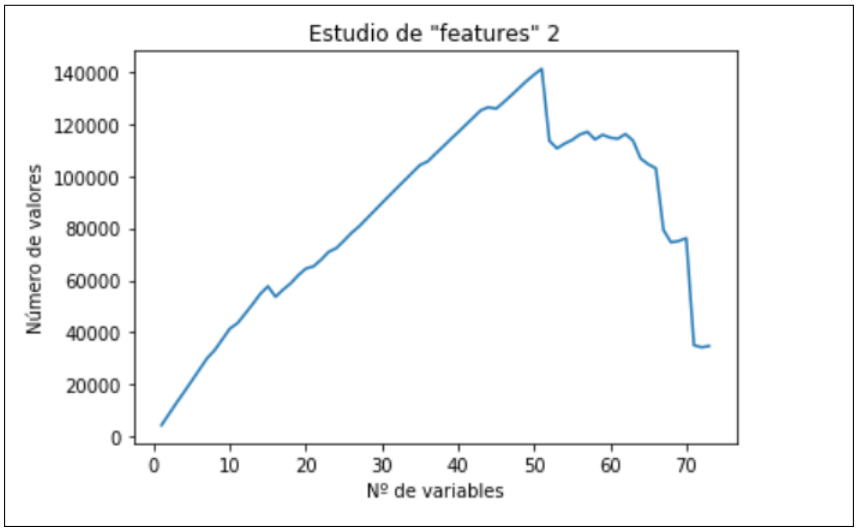

Figura 2.4: Análisis del número de datos después de descarte de casos y/o atributos

Como resultado del método encontramos un punto óptimo en **51** atributos y **2768** pacientes/casos de prueba. En la tabla 2.3 se reflejan los atributos escogidos tras el análisis.

Cuadro 2.3: Atributos mantenidos tras la aplicación de Descarte de casos y/o atributos

atributos escogidas por el algoritmo				
Clasifica	edad	sexo	Polimicr	anhonpol
Anaerobio	Hongos	periodo	mes	dia
medio	microbpoli	frasae	frasanae	frasextr
COMORBIL	so	antibiot	diashosp	Especialid
hgb	Inghosp1m	plaqut	stlocal	pmfn
leuc	Inghosp12m	Neoplasia	Hepatopatia	Enfresp
Diabetes	Cardiopatia	Insrenal	Udvp	Alcoholismo
creatin	sedorina	enfbasWeinst	drogadic	inmunosu
esteroid	cirugia	neutrope	glucosa	sintomas
m_digest	m_respir	leucocit	trombope	m_genitu
m_vascul				

Esta técnica elimina atributos que podrían tener un gran impacto en los resultados de nuestro algoritmo clasificador. Como método preventivo se recomienda evaluar la relevancia de cada atributo antes de que sea eliminada del *dataframe*.

2.5. Imputación de los datos perdidos

Otra técnica utilizada en aprendizaje automático para lidiar con el problema de los datos perdidos es la **imputación de valores**. Consiste en imputar los valores perdidos con alguna técnica complementaria (media, moda, mediana,..). Es decir, sustituir los valores perdidos por el valor calculado por el método de imputación escogido. Entre los más comunes, hay técnicas más sencillas como las ya mencionadas, la media, la mediana y la moda, y otras más complejas que se basan en algoritmos de aprendizaje automático como *nearest neighbor* entre otros.

En nuestro caso, al presentar gran cantidad de atributos de tipo categórico cuyas clasificaciones se hacen con valores enteros, la media y la mediana no se adaptan bien como método de imputación, ya que generarían un valor decimal. Esto tendría el mismo resultado que el método de *separate class method* definido previamente. Consecuentemente, realizamos pruebas con la moda para todos los atributos y por otro lado, con una combinación de la media para los atributos numéricos y la moda para los categóricos.

Para ver si está técnica nos ofrecía buenos resultados, realizamos un análisis con las tres técnicas que teníamos pensado utilizar más a delante, *random forest, support vector*

machine y *k-nearest neighbor*. Ejecutamos cada algoritmo numerosas veces registrando el mejor resultado, el peor y la media de los resultados, y después analizamos los resultados que veremos más a delante en el apartado de conclusiones.

2.6. Normalización de datos

Tras observar las diferentes escalas que presentan los datos que sometemos a estudio, más concretamente en los atributos numéricos, normalizamos los valores. De este modo los valores presentados por los diferentes atributos generan una separación basada en la misma escala. Evitando que un atributo con valores entre el 0 y 100 genere una mayor separación que un atributo con valores entre el 0 y el 1.

Para ello aplicamos una técnica bastante sencilla pero de gran utilidad. Para cada atributo buscamos su valor máximo y posteriormente dividimos todos los valores entre el máximo calculado, obteniendo así en cada atributo valores entre el 0 y el 1. En la figura 2.5 podemos ver un ejemplo. Se representada el mismo atributo, en la línea 1 normalizado, con el valor máximo y mínimo que toma el atributo. En la segunda línea de ejecución vemos el mismo atributo pero esta vez sin normalizar, con valores entre el 1 y el 1397.

```
[1]: df_normalizado['glucosa'].describe()
```

```
[1]: count    3416.000000
     mean        0.099053
     std         0.050484
     min         0.000716
     25%         0.070866
     50%         0.085183
     75%         0.108805
     max         1.000000
     Name: glucosa, dtype: float64
```

```
[2]: df_no_normalizado['glucosa'].describe()
```

```
[2]: count    3417.000000
     mean      138.615569
     std        71.882425
     min         1.000000
     25%        99.000000
     50%       119.000000
     75%       152.000000
     max      1397.000000
     Name: glucosa, dtype: float64
```

Figura 2.5: Comparativa entre el mismo atributo normalizado y no normalizado

En nuestro caso, también combinamos esta técnica con alguna de las citadas anteriormen-

te para ver si obteníamos mejores valores de exactitud en los modelos. Concretamente, combinamos la normalización con la técnica de *separate class method* descrita previamente en la sección 2.2. La técnica busca una nueva categoría para los valores perdidos en los atributos categóricos y un valor que genere suficiente separación en el modelo de los valores existentes para los atributos numéricos. En este caso, utilizamos el valor $-0,5$ dado el rango de valores sobre el que se esté trabajando.

2.7. One Hot Encoder

Tras el estudio de las diferentes técnicas de aprendizaje automático, encontramos que gran cantidad de ellos no tratan los atributos categóricos como tal, sino que las tratan como valores numéricos. Esto supone un problema ya que un algoritmo podría darle mayor importancia a un valor superior de un atributo categórico cuando todos los valores que toma son igual de importantes, lo que significaría que nuestro algoritmo de aprendizaje automático no estaría funcionando correctamente y podría tener consecuencias directas en la exactitud final del modelo.

La técnica más utilizada para evitar problemas de este tipo es ***One hot encoder***, una técnica que recorre el *dataframe* y separa cada atributo de tipo categórico en subcategorías, es decir, por cada categoría presente en un atributo, el algoritmo genera un nuevo atributo. Como consecuencia directa, esta técnica puede aumentar considerablemente el número de atributos que componen el *dataframe* dependiendo del número de atributos categóricos presentes y las subcategorías que lo compongan.

En la figura 2.6 se puede ver la transformación de los datos de la categoría "color" en tres subcategorías, en ellas se indica con el valor **1** aquellas en las que anteriormente eran del color específico y con 0 aquellas que no. Por ejemplo, en el atributo inicial el color "verde" era representado por el valor **1** dentro de la categoría "color", mientras que en la tabla de la derecha aparece la columna verde, donde el valor **1** representa el color verde, y **0** la ausencia de él.

nombre color	color
verde	1
rojo	2
azul	3
rojo	2
verde	1

verde	rojo	azul
1	0	0
0	1	0
0	0	1
0	1	0
1	0	0

Figura 2.6: Ejemplo de la transformación de los datos mediante One Hot Encoder

Capítulo 3

Sobreajuste y Subajuste

Los algoritmos de aprendizaje automático supervisado requieren de un amplio volumen de datos que serán utilizados en la fase de entrenamiento (fase en la que el algoritmo de aprendizaje busca la relación entre los casos suministrados y la salida esperada) y en la fase de prueba o validación (fase en la que se suministra un menor número de datos sobre los que no ha trabajado e intenta predecir un resultado para cada caso). Con las predicciones realizadas en esta fase y la clasificación que se conoce para cada caso, podemos calcular el nivel de exactitud del modelo. El procedimiento descrito anteriormente se denomina **aprendizaje inductivo**. La capacidad de inducción de un modelo determina el nivel de precisión que posee un algoritmo a la hora de intentar resolver un problema parecido a los suministrados como ejemplo. El objetivo de cualquier algoritmo de aprendizaje automático es generalizar bien los datos de entrenamiento en cualquier dominio del problema. La finalidad de la técnica es predecir acciones futuras sobre datos nunca antes vistos.

A veces, la exactitud que posee un algoritmo no es fiable. Esto se debe a que los datos están "mal ajustados". Los principales causantes de un modelo con una exactitud poco fiable se denominan **sobreajuste** y **subajuste**.

El subajuste o **underfitting** del modelo consiste en una elevada generalización de los datos, es decir, el número de casos empleados en la fase de entrenamiento es escaso y provoca que el modelo generalice excesivamente. Este elevado nivel de generalización causará problemas al clasificar nuevos casos de entrada y provocará valores de exactitud bajos. También tenemos el caso contrario: el sobreajuste o **overfitting**. En este caso, el volumen de datos empleado para testear es significativamente más pequeño que el recomendado, provocando que el modelo tenga un nivel de generalización muy bajo. Esto conllevaría una tasa de exactitud del modelo muy baja fuera de los casos incluidos en nuestro conjunto de datos de estudio, es decir, este modelo no sería útil con casos nuevos. Ahora bien, una vez expuestos los dos mayores problemas que se suelen plantear, ¿cómo conseguimos unos datos válidos y fiables para nuestro modelo? Se debe encontrar un punto medio (**sweet spot**). La figura 3.1 refleja gráficamente lo explicado anteriormente. En ella se representa gráficamente un modelo que presenta un subajuste donde se observa la elevada generalización, un modelo con un ajuste ideal y un modelo con un sobreajuste que presenta una curva muy ajustada a los puntos y con escasa capacidad de generalización.

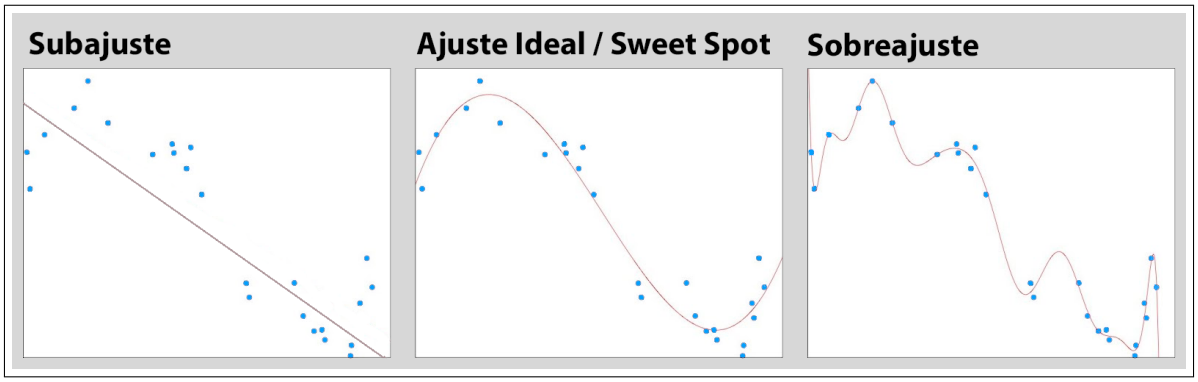

Figura 3.1: Comparación entre subajuste, ajuste ideal y sobreajuste (Imagen generada a partir de [8])

3.1. Lidiar con el sobreajuste y el subajuste

Para evitar estos problemas se recomienda realizar comprobaciones sobre los datos con los que se va a trabajar [9]:

- Garantizar gran cantidad de casos para entrenar y validar el modelo. Para ello, buscamos trabajar con la mayor parte de los datos aportados por el Hospital Universitario de Fuenlabrada mediante las técnicas reflejadas en el capítulo 2. Gracias al estudio de los atributos y las pruebas realizadas con todas las técnicas, cabe destacar Separate Class Method, técnica que permite mantener todos los casos a pesar de los datos perdidos.

- Para los casos en los que los datos vayan a ser empleados en una clasificación, se recomienda que los datos estén **equilibrados**. Es importante que los datos de entrenamiento estén balanceados y, en caso de ser una clasificación múltiple, que los casos sean representativos de todas las clases. Es decir, que haya un porcentaje de casos semejante para cada una de las clasificaciones posibles pare evitar sesgos innecesarios. Para revisar si nuestros datos estaban balanceados inicialmente, analizamos el porcentaje de casos que presentaban bacteriemia y el porcentaje de casos que no. Los resultados obtenidos son de un **48,72 %** de casos con bacteriemia frente a un **51,27 %** de no bacteriemia lo que nos indica que los datos están bastante equilibrados.

- Subdividir el conjunto de datos, una gran parte de ellos destinados al entrenamiento del modelo y la parte restante para validarlo. Esta división permite evaluar el modelo generado a través de los datos de entrenamiento con casos diferentes y también permite detectar los efectos del sobreajuste y subajuste. Esta separación se puede realizar fácilmente con la ayuda de la librería *sklearn* con el método *train_test_split*, este método requiere de un parámetro (*test_size*) que indica la proporción de datos que compondrá el conjunto destinado a validar el modelo. El valor recomendado depende del volumen de datos que conformen el *dataframe*, por norma general este valor oscila entre $[0{,}2, 0{,}3]$ aunque si el volumen de datos es muy grande, 1 millón de datos por ejemplo, podría reducirse. La figura 3.2 representa la

utilización de esté método, siendo X el *dataframe* con los casos e Y la clasificación de cada caso.

```
from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size=0.2)
```

Figura 3.2: Subdivisión de los datos para las fases de entrenamiento y validación con la ayuda de la librería *sklearn*

Dado que esta división se realiza aleatoriamente, a priori, no se puede saber si los datos están equilibrados, al igual que se desconoce si esta es la mejor combinación para generar un modelo de clasificación preciso y sin presencia de sobreajuste o subajuste. Para comprobar que los datos generaban buenos resultados, *k-fold cross validation* (explicado en la sección 3.2) permite generar múltiples modelos con los datos y calcular una media de exactitud. De modo que se puede analizar si los valores de exactitud que se obtienen se encuentran cerca de la media.

- Se debe evitar la cantidad excesiva de atributos, ya que generaría gran cantidad de dimensiones en nuestro modelo. Esto se debe a que cada atributo compone una dimensión del espacio muestral del modelo. El número de dimensiones del espacio muestral debe ser proporcional al número de casos de los que se disponga para realizar el estudio, es decir, a mayor número de atributos, mayor número de casos de estudio, o al revés, en el caso de tener pocos casos de estudio, se debe utilizar pocos atributos. Para ello, existe una herramienta muy útil sobre la que apoyarse, *principal component analysis* (presentada en la sección 3.4).

3.2. K-Fold Cross Validation

Para implementar cualquier técnica de aprendizaje automático es necesario trabajar con un conjunto de datos voluminoso y con múltiples atributos, de esta forma es más probable obtener un resultado más satisfactorio al finalizar el proceso. Los datos son divididos en dos, una parte de ellos es destinada al entrenamiento del modelo, mientras que la otra es empleada para llevar a cabo la validación del modelo generado anteriormente y evaluar la precisión de las predicciones del algoritmo clasificador empleado. Dado que los datos son separados aleatoriamente, no se puede asegurar que dicha separación sea la mejor forma de conseguir el mayor porcentaje de acierto en la predicción. Tampoco se puede asegurar que estos datos no generen sobreajuste o subajuste, por lo que se recomienda la utilización de *k-fold cross validation* para revisar que los valores de precisión del modelo obtenidos con la partición aleatoria y la media de precisión generada por esta técnica, son semejantes.

K-fold cross validation permite generar diferentes modelos a partir del mismo conjunto de datos. La técnica divide en k subconjuntos diferentes a partir del conjunto original y genera un modelo de modo que cada subconjunto de datos es utilizado tanto para la parte de entrenamiento como en la parte de validación. Más concretamente, mezcla aleatoriamente el *dataframe* y lo subdivide en k grupos iguales. Para cada grupo k, toma este

grupo k como conjunto de datos de validación y el resto de grupos formarán el conjunto de datos de entrenamiento. Entrena el modelo con los datos de entrenamiento, lo evalúa con los datos de validación, guarda el valor de exactitud obtenido y descarta el modelo. Finalmente, hace la media de los valores de exactitud de los k modelos distintos para obtener una valoración media de cómo se comportan los datos, previamente ordenados aleatoriamente, con el modelo y permite analizar los valores resultantes [10].

La figura 3.3 representa gráficamente como *k-fold cross validation* selecciona los distintos subconjuntos y forma los conjuntos de datos para las fases de entrenamiento y validación del modelo.

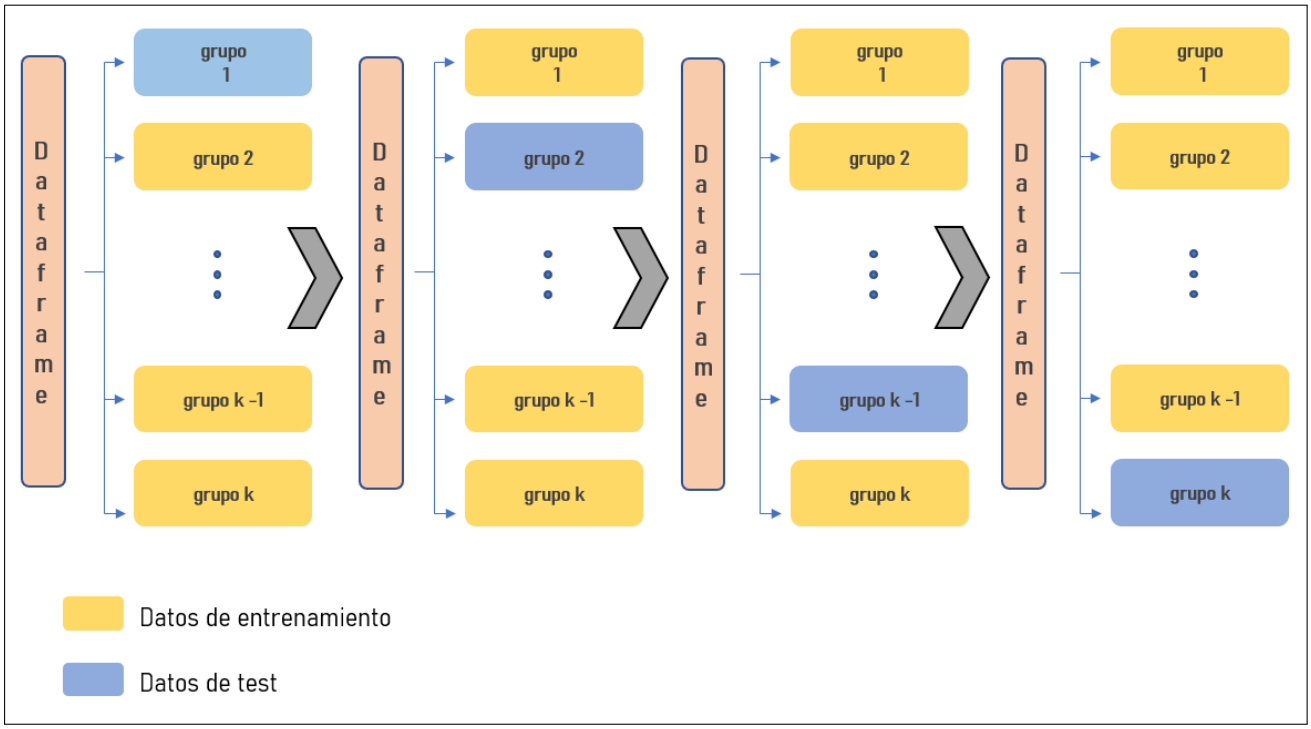

Figura 3.3: Representación gráfica de la selección de los conjuntos de datos de entrenamiento y validación en *k-fold cross validation*

3.3. Grid Search Cross Validation

GridSearchCV es una clase perteneciente a la librería *sklearn* que permite, dado un modelo (*model*) y un conjunto de parámetros de entrada (*param_grid*), entrenar un modelo y evaluar qué parámetros de entrada son los mejores para dicho modelo. Los parámetros de entrada pueden contener listas de elementos, por ejemplo, un modelo clasificador basado en *k-nearest neighbor* (capítulo 6) requiere como parámetro de entrada el número de vecinos (*n_neighbors*). En este caso, la función tendría como entrada una variable *params* con una lista que contenga los valores de *n_neighbors* que se quieran probar. Esta clase también realiza una validación cruzada utilizando *k-fold cross validation* (sección 3.2). El parámetro k de esta técnica se establece con el parámetro de entrada "cv" de la clase. En caso de no fijarse un valor, la clase realizaría un 5-*fold cross validation* ya que así está definida por defecto.

Esta clase facilita bastante el desarrollo de los algoritmos a nivel de codificación, ya que centraliza y controla la fijación de parámetros de entrada de los modelos. Gracias a ella, permite pasar los parámetros en forma de listas y ella se encarga de realizar el entrenamiento de los modelos para cada combinación posible entre los parámetros. Además, almacena el modelo que mejores resultados da y su configuración de parámetros, una lista con los resultados de precisión media del modelo para los diferentes grupos generados por *k-fold cross validation* y una lista con los resultados para cada grupo.

En general, ayuda con la gestión de bucles que serían necesarios en caso de querer codificar manualmente los parámetros del modelo, con el almacenamiento de resultados de todos los modelos entrenados y la inclusión de la técnica *k-fold cross validation* para el estudio de la posible aparición de sobreajuste y subajuste.

La figura 3.4 presenta un fragmento de código en el que son definidos los diferentes parámetros de entrada de la clase. En este caso, continuando con el ejemplo citado previamente, un modelo clasificador *k-nearest neighbor* con una lista de elementos $[1 - 20]$ para el parámetro *n_neighbors*. y 10-*fold cross validation*. Es observable la sencillez de codificar un modelo clasificador con esta clase.

```
from sklearn.model_selection import GridSearchCV
# definición del modelo clasificador
model = KNeighborsClassifier()
# definición de los parametros sobre los que se desee evaluar el modelo
params ={'n_neighbors': range(1,20)}
# Definición del grid
grid = GridSearchCV(model, params, cv=10)
# entrenamiento de los diferentes modelos
grid.fit(X_Train, Y_Train)
```

Figura 3.4: Ejemplo de codificación de un modelo con GridSearchCV

3.4. Principal Component Analysis

Principal Component Analysis (PCA) es un método estadístico que nos brinda la posibilidad de simplificar la complejidad de espacios muestrales con muchas dimensiones pero manteniendo la información. Aplicado al aprendizaje automático se define como un método estadístico que se basa en el estudio de atributos no correlacionados con el fin de reducir la dimensionalidad de nuestro conjunto de datos. Este método se apoya en el uso de los autovalores y autovectores que conforman la matriz de covarianza de nuestros datos de entrada.

PCA busca escoger un nuevo sistema de coordenadas para representar la información de nuestro conjunto base siguiendo estos pasos:

- Estandarizar los datos de entrada. Este proceso se utiliza con el fin de que no haya

atributos que se sobrepongan a otras debido a la naturaleza de sus magnitudes. En el caso de que todos los datos se representasen en la misma magnitud este proceso no sería necesario.

- Obtener los autovalores y los autovectores de la matriz de covarianza o de la matriz de correlación construidas a partir del *dataframe* original estandarizado. Al haber estandarizado los valores, las dos matrices coinciden.
- Recoger los autovalores en orden decreciente (de mayor a menor) y escoger los k autovectores que se corresponden con los k autovalores más grandes, siendo k el número de dimensiones del subespacio de características nuevo. Hay que tener presente que un número k bajo podría generar sobreajuste.
- Construcción de la matriz de proyección W a partir de los k autovectores seleccionados.
- Transformar el *dataframe* estandarizado para adaptarlo al nuevo subespacio de k dimensiones usando W.

PCA permite obtener información de los atributos tanto de manera individual como en conjunto. De forma individual busca analizar el atributo y ver su varianza, y a su vez trata de establecer relaciones entre atributos buscando la existencia de redundancia en los datos. Como resultado, obtenemos un conjunto de componentes principales generado por PCA. Cada una de estas componentes será una combinación lineal de los atributos del conjunto de datos inicial [11]. Las componentes principales generadas no están correlacionadas entre si, es decir, son independientes.

Supongamos un *dataframe* compuesto por n casos y m atributos (F_1, F_2, F_3, ... , F_m), es decir, un *dataframe* con un espacio muestral compuesto por m dimensiones. PCA nos facilita encontrar un subconjunto menor de atributos ($k < m$) semejante a nuestro espacio muestral original. Para su codificación en Python, aunque es posible implementar la técnica paso a paso apoyándose en las librerías propias del lenguaje, la librería *sklearn* incluye un método PCA que simplifica el proceso. Con este método podemos implementar PCA de dos modos, en base al porcentaje de varianza acumulada por las componentes principales o mediante el número de componentes principales k que queremos que tenga nuestro nuevo *dataframe*.

Principalmente, esta técnica se aplica con el fin de reducir la dimensionalidad de los datos y, en ocasiones, puede generar un aumento de la exactitud del modelo. En las redes neuronales se utiliza para reducir los tiempos de ejecución, los recursos que consume y la complejidad. Por el contrario, esta técnica reduce la explicabilidad de nuestro modelo [12], por un lado, por la baja explicabilidad de los resultados de la técnica y, por otro lado, la sustitución de los atributos de los datos de entrada por las componentes principales impide que no se pueda conocer la influencia real de los atributos sobre el algoritmo clasificador que se aplicará después.

En caso de obtener grandes mejoras en la exactitud del modelo clasificador generado después de aplicar PCA sobre los datos y se quiera utilizar esta técnica, es posible conocer de manera aproximada cómo se mezclan los atributos originales sobre las componentes principales usando el método **pca.components_** de la librería *sklearn.decomposition*. Este

método devuelve una lista de las componentes principales y para cada una de ellas, una lista con la influencia de cada atributo. La figura 3.5 representa gráficamente mediante un gráfico de barras las 4 primeras componentes principales de la lista devuelta por el método mencionado anteriormente [13]. A pesar de conocer la influencia de los atributos en este caso es difícil conocer la influencia final de cada atributo sobre el modelo y, como consecuencia, hay una perdida de explicabilidad del modelo.

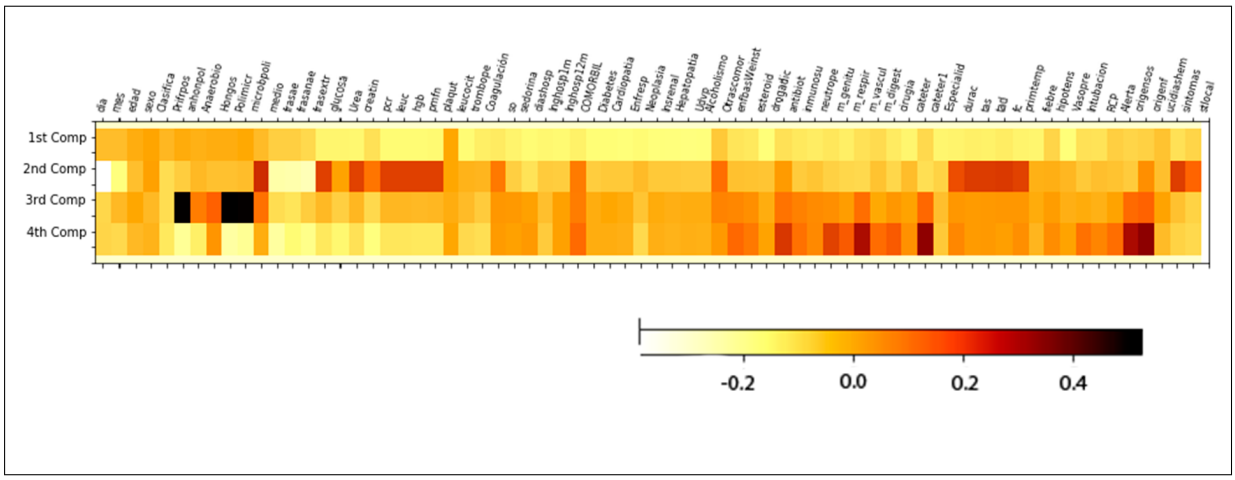

Figura 3.5: Influencia de los atributos originales sobre las componentes principales calculadas por PCA

Capítulo 4

Support Vector Machine

4.1. Introducción

Una máquina de vectores de soporte, o SVM, es un algoritmo de aprendizaje supervisado utilizado en problemas de regresión y clasificación. Esta técnica se utiliza a menudo debido a su alta tasa de acierto en comparación con otras técnicas como puede ser la regresión logística. Se emplea en diversidad de aplicaciones tales como: la detección de rostros, el reconocimiento de voz, la extracción de información en minería de datos, detección de intrusos, clasificación de correos electrónicos como spam, clasificación de genes y reconocimiento de escritura.

Una de las mayores virtudes que presenta la SVM es su capacidad de tratar con datos categóricos y cuantitativos, algo presente en nuestro dataset. Las ventajas que se obtienen al emplear esta técnica sobre un conjunto de datos son las siguientes: (1) se consiguen buenos resultados en altas dimensiones y aunque el número de dimensiones sea mayor al de muestras, el algoritmo no se ve afectado; (2) al utilizar un subconjunto de datos durante la fase de entrenamiento se emplea la memoria de forma más óptima. SVM funciona bien con un amplio espacio dimensional y con un alto margen de separación. Pero, no todo iban a ser cualidades, también nos encontramos inconvenientes: SVM no proporciona directamente estimaciones de probabilidad, sino que se calculan utilizando validación cruzada quíntuple, más conocida como 5-fold cross validation (sección 3.2).

4.2. Funcionamiento

Para entender como funciona este algoritmo, se debe tener nociones sobre tres conceptos clave:

a). Un hiperplano es un plano de dimensión $n - 1$.

b). Los vectores de soporte son aquellos datos que están más cercanos al hiperplano. Estos vectores definirán mejor el punto exacto hasta el cual puede situarse el margen del hiperplano. Son claves para el desarrollo del clasificador.

c). Por último, el margen es la separación entre las dos líneas en los puntos de clase más cercanos y se calcula como la distancia perpendicular de la línea a los vectores de soporte o puntos más próximos. Para que un margen sea considerado óptimo, debe ser mayor entre las clases y si el margen es menor, entonces no será considerado provechoso.

Este algoritmo tiene un funcionamiento relativamente simple, ya que su misión es encontrar un hiperplano en el espacio multidimensional que separe a las diferentes clases. Este hiperplano debe separarlas generando el margen más ancho posible, intentando minimizar el de error a fin de facilitar así la clasificación de un nuevo dato.

En la figura 4.1 se puede observar de forma clara como las diferentes clases de datos (representadas por estrellas rojas la clase A y triángulos verdes la clase B) y sus vectores de soporte son separadas por un hiperplano, que es la recta de color negro, y este a su vez, intenta ampliar al máximo posible el margen de separación de clases.

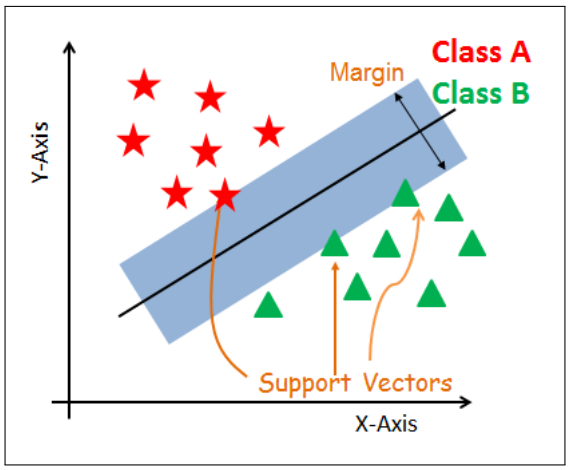

Figura 4.1: Separación de clases por hiperplano

El objetivo principal es separar el conjunto de datos de la forma más óptima. La distancia entre dos puntos cercanos se denomina margen. La finalidad es seleccionar un hiperplano con el máximo margen factible entre los vectores de apoyo en el conjunto de datos dado. Para buscar el mejor hiperplano, SVM sigue los siguientes pasos: primero, tal y como se

muestra en la figura 4.2, se generan tantos hiperplanos como sean posibles tratando de segregar los vectores de soporte. En el segundo paso, los hiperplanos de color naranja y azul, no separan las clases de forma correcta, con lo cual nos quedamos únicamente con el de color negro. El resultado se puede observar en la figura 4.1.

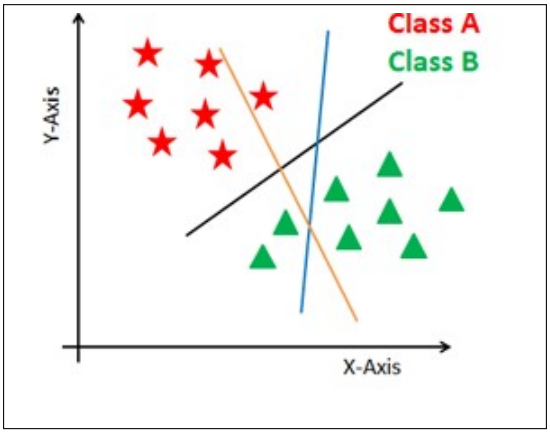

Figura 4.2: Generación de múltiples hiperplanos

En ocasiones, algunos problemas no se pueden resolver con un hiperplano lineal. En esos casos, el SVM modifica su función kernel para adaptarse a esta coyuntura y así generar un espacio dimensional distinto, en el que se puedan segregar los datos de manera lineal.

En la figura 4.3 se expone un ejemplo de una circunstancia donde no se puede aplicar un hiperplano lineal. En la imagen de la izquierda, los vectores de soporte se distribuyen de forma inclasificable. Por el contrario, en la imagen de la derecha, el kernel del SVM se ha adaptado para facilitar que se pueda trazar el hiperplano sin problema.

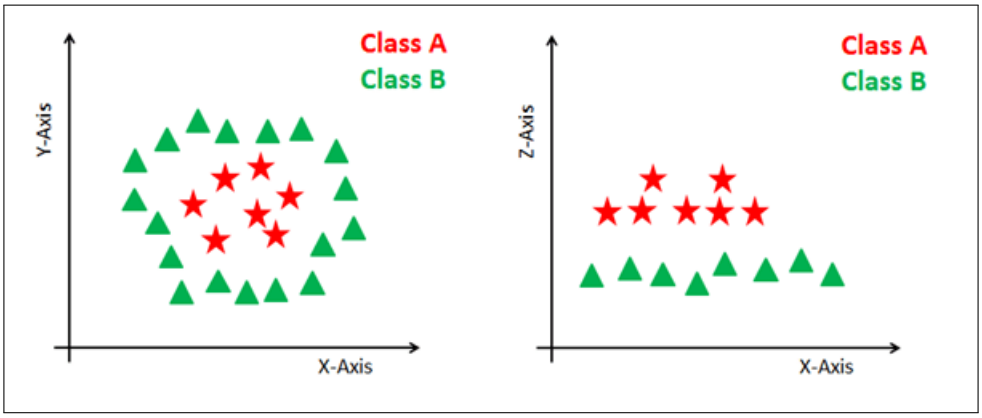

Figura 4.3: Problemas con hiperplanos
[14]

4.3. Función Kernel

SVM utiliza una función llamada núcleo o kernel que se emplea para transformar un espacio de datos de entrada para adaptarlo a la situación requerida. La figura 4.3 mostrada

y explicada anteriormente define muy bien esta función. El núcleo convierte un problema de variables no separables en otro con variables separables, ayudando a aumentar la precisión del algoritmo clasificador. Una función tiene que cumplir tres condiciones para que sea considerada un núcleo: debe ser una función continua, positiva y simétrica. Estas características son demandadas debido a que son esenciales para que dicha función sea expresada como un producto escalar en un espacio dimensional alto. El espacio dimensional producido por las funciones núcleo se define usando cada atributo de los datos en un conjunto de puntos de un espacio n-dimensional. Así, es mucho más sencillo establecer relaciones entre los datos.

Hay 3 tipos de funciones núcleo, K: la función lineal, la función polinómica y la función de base radial.

- Lineal: es el producto de un dato concreto con sus dos observaciones más cercanas. El producto entre dos vectores es la suma de la multiplicación de los valores de entrada.

$$K(x, x') = \langle x, x' \rangle = \sum_i x_i \cdot x'_i \tag{4.1}$$

- Polinomial: un núcleo polinomial es más general que un núcleo lineal, ya que este tipo de función es capaz de generar un hiperplano rectilíneo o curvo.

$$K(x, x') = (1 + \langle x, x' \rangle)^d \tag{4.2}$$

  El valor de d indica el grado del polinomio. Por defecto, toma el valor de 1, lo que señala que un problema así puede ser solucionado por un núcleo lineal o polinomial de grado 1. El grado del polinomio se debe especificar manualmente en el algoritmo.

- Radial Basis: es la función más utilizada en un clasificador de máquinas de vectores de soporte, ya que es capaz de mapear un conjunto de datos de entrada en un espacio dimensional infinito.

$$K(x, x') = \exp\left(-\gamma * \|x – x'\|^2\right) \tag{4.3}$$

  γ es un parámetro que oscila entre 0 y 1. Su valor por defecto es $\gamma = 0{,}1$, pero se puede especificar manualmente.

Dado que la función de núcleo más generalizada es Radial Basis, y es la que mejor se adecúa a nuestro espacio de entrada, será el método que utilizaremos en el estudio.

4.4. Ajuste de hiper-parámetros en un modelo SVM

En aprendizaje automático se utilizan modelos para aproximar funciones. Dichos modelos emplean ciertos parámetros cuya utilidad es aproximar la función. Un ejemplo de ello,

para quien conozca las redes neuronales, son los pesos que usan las mismas para modelar los patrones en los datos. Estos parámetros son denominados hiper-parámetros, se configuran antes del entrenamiento del modelo y no forman parte de él; los valores escogidos se mantienen constantes a lo largo del proceso de entrenamiento. El rendimiento del modelo depende en gran medida de los valores de hiper-parámetros seleccionados. El objetivo de la exploración de estos valores es buscar a través de varias configuraciones de hiper-parámetros para encontrar la configuración que tenga el mejor rendimiento. Los parámetros claves en SVM son Gamma (γ) y Regularización (C).

Gamma se encarga de proporcionar al modelo el valor exacto de ajuste que debe tener con respecto a los datos de entrenamiento, con lo cual, un valor bajo de Gamma se ajustará muy poco al conjunto de datos de entrenamiento. Mientras que un valor alto se acoplará en demasía el modelo a los datos de entrada. Un valor bajo de este hiper-parámetro hará que el algoritmo se fije únicamente en los valores cercanos al hiperplano, a la vez que, un valor alto inducirá al modelo a fijarse en todos los valores de la gráfica.

Los valores que puede tomar Gamma en el paquete de Python utilizado son "auto", "scale", o un valor dentro del intervalo entre 0 y 1. El valor "auto" es el resultado de dividir 1 entre el número de atributos.

$$\gamma = \frac{1}{\text{atributos}} \tag{4.4}$$

El valor "scale" se calcula con una fórmula parecida a "auto", pero con una diferencia en el denominador ya que este se multiplica por la varianza de los datos, X.Var().

$$\gamma = \frac{1}{\text{atributos} \cdot X.Var()} \tag{4.5}$$

La regularización (C) es un parámetro de penalización y representa el coste de una clasificación errónea de un valor. Indica al algoritmo el nivel de error soportable, es decir, cuanto puede aumentar el margen del hiperplano para favorecer los resultados de la predicción. Así, es como se controla el equilibrio entre el límite de decisión y el porcentaje aceptado de fallo. Un valor bajo de C implicará un margen pequeño de hiperplano. En contraposición, si el valor es alto, el margen será mayor.

Para escoger los mejores valores de los parámetros ya mencionados y desarrollar un modelo de SVM óptimo se ha empleado de la clase Grid Search Cross Validation, explicada en la sección 3.3. A esta clase se le ha suministrado un modelo sencillo, un array con todos los valores posibles para los parámetros de regularización y gamma, además de especificar el parámetro de cross validation que detalla el número de splits que se ejecutarán sobre los datos (sección 3.2. Este intervalo de valores se muestra en la figura siguiente:

```
1 #Modelo SVM
2 model = SVC(random_state=0)
3 #Valores de estudio de gamma y C
4 params=[
5 {'gamma' : [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9, 'auto', 'scale']},
6 {'C' : [0.001,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,3,4,5,6,7,8,9,10,
7       20,30,40,50,60,70,80,90,100,200,300,400,500,600,700,800,900]}
```

```
]
#Función GridSearchCV con las entradas
grid = GridSearchCV(model, params, cv=10)
```

Una vez ejecutado el método GridSearch, la variable grid guarda el modelo ideal de SVM, que será entrenado por los datos de entrenamiento. Finalmente, para conocer los hiper-parámetros óptimos desarrollados por el método GridSearchCV se ejecutará la función *best_estimator_.get_params()*.

```
#Entrenamiento del modelo
grid.fit(X_Train, Y_Train)
#Conocer los parámetros con los que se ha entrenado SVM
grid.best_estimator_.get_params()
```

En la figura 4.4 se muestran los valores adoptados en el algoritmo. La regularización tiene un valor de 3, el kernel elegido es rbf y gamma es "auto".

```
El valor de Gamma óptimo según GridSearchCV es : auto
El valor de C óptimo según GridSearchCV es : 3
```

Figura 4.4: Valores parámetros SVM

Las conclusiones que se muestran en la siguiente sección corresponderán a la configuración óptima mostrada en la figura superior.

4.5. Conclusiones del modelo

Tras especificar los valores de los hiper-parámetros mencionados en el modelo junto a la función núcleo estimada, la tasa de acierto en la predicción conseguida es la siguiente:

```
La tasa de acierto dada por el modelo SVM óptimo es: 0.8860832137733142
```

Figura 4.5: Tasa de acierto del modelo SVM

Para averiguar si el algoritmo transmite esta precisión a los datos de validación, se introducen los datos de validación al modelo. Los resultados se observan en la figura 4.6

```
La tasa de acierto tras el testing es: 0.8589449541284404
```

Figura 4.6: Tasa de acierto SVM al validar

Las figuras 4.5 y 4.6 muestran un porcentaje similar, lo que sugiere un algoritmo correctamente entrenado y una ausencia de sobreajuste.

Para poder conocer a fondo el grado de precisión del modelo en cuestión, existen diversas técnicas de interpretabilidad para los modelos de aprendizaje automático. Destacan las siguientes cuatro métricas:

a). La precisión (**precision**) es la relación entre las predicciones positivas correctas y el total de predicciones positivas. Esta métrica está relacionada con una baja tasa de falsos positivos (aquellos pacientes sin bacteriemia que según el modelo sí están infectados), ya que la menor tasa de falsos positivos indica que el modelo predice la bacteriemia de forma óptima.

$$\text{precision} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Positivos}} \tag{4.6}$$

b). La sensibilidad (**recall**) es el porcentaje entre las predicciones positivas correctas y todas las observaciones hechas por el modelo.

$$\text{sensibilidad} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Negativos}} \tag{4.7}$$

c). La puntuación F1 (**F1-score**) es el promedio entre la precisión y la sensibilidad, y tiene en cuenta tanto los falsos positivos como los falsos negativos.

$$\text{F1} - \text{score} = \frac{2 * \text{sensibilidad} + \text{precision}}{\text{sensibilidad} + \text{precision}} \tag{4.8}$$

d). El recuento (**support**) es el número de entradas de cada clase (1 o 3) dentro del grupo de datos destinado a validar el modelo.

Los resultados obtenidos de estas métricas con nuestro modelo se muestran en la figura 4.7 de la cual se extraen los siguientes resultados: de todos los pacientes que el modelo identifica con bacteriemia, son correctos un 82 por ciento, es decir, se equivoca en un 18 por ciento. Mientras que, para detectar pacientes no infectados, su precisión mejora hasta un 91 por ciento. Por otra parte, si se atiende a la sensibilidad, el algoritmo identifica al 91 por ciento los pacientes que según los datos reales de entrada, están infectados. Esta métrica desciende al 81 por ciento en el caso de la detección por el modelo de pacientes no infectados. La media de ambas métricas está cifrada en un 86 por ciento.

	precision	recall	f1-score	support
1	0.82	0.91	0.86	417
3	0.91	0.81	0.86	455

Figura 4.7: Resultados obtenidos SVM

4.5.1. Matriz de confusión

La matriz de confusión es una herramienta muy utilizada en el mundo de la Inteligencia Artificial puesto que permite mostrar de forma clara y concisa los resultados obtenidos en el modelo diseñado. Nuestro atributo dependiente es Clasifica, cuyo valor puede ser si hay bacteriemia o no. Por ello, nuestra matriz solo tendrá dos columnas y dos filas. Las columnas representarán el resultado de la clasificación y las filas los valores reales del atributo. De esta forma, nuestra matriz queda subdividida en cuatro cuadrantes. El cuadrante superior izquierdo muestra los valores que la técnica ha identificado como bacteriemia de forma correcta. En la parte superior derecha se observan los valores sin bacteriemia de forma errónea, ya que el resultado debería haber sido diferente. En el cuadrante inferior izquierdo, el modelo debería no haber identificado la presencia de bacteriemia, pero no ha sido así, Por último, en el último cuadrante, quedan representadas aquellas predicciones acertadas sobre la falta de presencia de bacteriemia. Cuanto mayor sea la coincidencia entre las predicciones y el valor real, la matriz adquirirá un tono verde más oscuro, obtendrá un color blanco en caso contrario.

La figura 4.8 es la matriz de confusión del algoritmo SVM empleado bajo la configuración descrita anteriormente. Se extraen los siguientes resultados: el número de aciertos en la predicción tanto de casos positivos como negativos es muy superior a los fallos que haya podido generar el algoritmo, siendo superior la tasa de acierto al predecir casos positivos de bacteriemia.

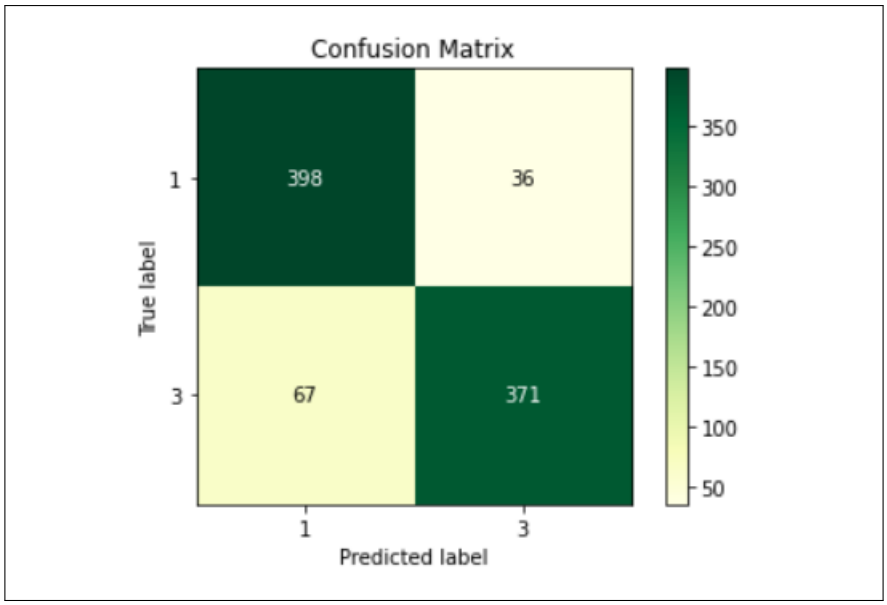

Figura 4.8: Matriz de confusión SVM

4.5.2. Peso de las atributos en el algoritmo

A lo largo de este capítulo, se ha hablado del conjunto de datos con el cual se entrena y valida el modelo, pero no conocemos la influencia de cada atributo en este proceso. Por ello, utilizando la librería *eli5* se llevará a cabo dicho estudio. Es clave conocer el

peso de los atributos, ya que permitiría con menos parámetros, evaluar si el paciente está infectado, es decir, si solo se conocieran los valores de los atributos claves, el modelo podría predecir el estado del paciente sin necesitar todas los atributos que se han utilizado en el entrenamiento y la validaci ónde la técnica.

En la figura 4.9 se observa una lista de atributos con su correspondiente peso asignado y la variación del mismo, ya que el algoritmo ajusta los pesos adaptándose a los valores proporcionados de cada paciente. En este caso, los atributos que cobran una mayor importancia son: *origensos*, el origen sospechado de la bacteriemia a extracción de hemocultivos, *sedorina*, sedimento de orina, *microbpoli*, gérmenes de la bacteriemia polimicrobiana, *frasanae*, crecimiento en frascos de anaerobios y *Prifrpos*, primer frasco donde crece el patógeno. El primer atributo mencionada destaca por encima del resto con un peso cercano a 0.4.

Weight	Feature
0.4046 ± 0.0326	origensos
0.0683 ± 0.0245	sedorina
0.0523 ± 0.0212	microbpoli
0.0408 ± 0.0098	frasanae
0.0266 ± 0.0107	Prifrpos
0.0170 ± 0.0107	medio
0.0165 ± 0.0110	fiebre
0.0156 ± 0.0079	so
0.0128 ± 0.0037	Hongos
0.0115 ± 0.0071	Alerta
0.0092 ± 0.0186	mes
0.0078 ± 0.0037	Coagulación
0.0064 ± 0.0018	leucocit
0.0060 ± 0.0080	anhonpol
0.0060 ± 0.0132	enfbasWeinst
0.0046 ± 0.0077	hipotens
0.0046 ± 0.0096	frasae
0.0041 ± 0.0045	stlocal
0.0032 ± 0.0069	m_respir
0.0028 ± 0.0034	sintomas
… 49 more …	

Figura 4.9: Influencia de los atributos en SVM

Capítulo 5

Random Forest

5.1. Introducción

El algoritmo de **random forest** (RF) es una técnica que genera múltiples árboles de decisión con su consecuente salida individual y calculará el resultado haciendo el promedio de las salidas. La figura 5.1 representa lo previamente dicho.

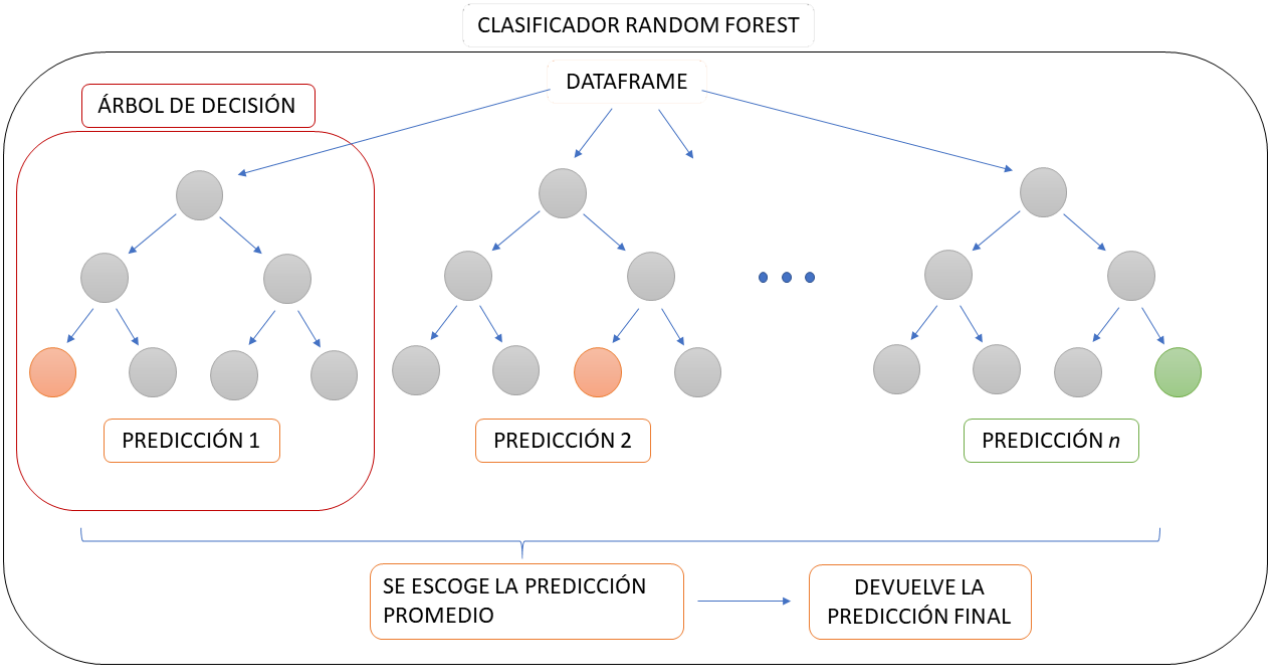

Figura 5.1: Funcionamiento sencillo de Random Forest

Es un modelo de aprendizaje supervisado que incluye distintos métodos en la su fase de entrenamiento para lidiar los problemas de sobreajuste y subajuste (Capítulo 3). Este algoritmo es utilizado para la resolución de problemas de regresión y clasificación. Posee un correcto funcionamiento aún sin ajustar sus parámetros propios y se mantiene estable al introducir nuevos datos. En contrapartida, requiere tiempos de procesamiento elevados, es difícil de interpretar y los *dataframes* pequeños no son procesados de forma óptima.

5.2. Funcionamiento

El algoritmo de **random forest** se basa en la utilización de múltiples árboles de decisión para, a partir de la combinación de los resultados de todos ellos, dar un resultado libre de sesgos y lo más preciso posible. En la fase de entrenamiento RF genera el mayor número posible de árboles de decisión con los que, posteriormente, dará una clasificación para el caso de estudio a evaluar. Para lidiar con los sesgos que puedan aparecer en el modelo, el algoritmo aplica dos técnicas: *bagging* y *feature randomness*. Gracias a ellas, se obtiene un mayor número de árboles de decisión correlacionados entre si de la menor manera posible.

El algoritmo de *random forest* requiere del método de *bagging* para poder generar una mayor cantidad de árboles de decisión. Más concretamente, este método se basa en la selección aleatoria y uniforme de k conjuntos de datos con reemplazamiento para, posteriormente, generar árboles de decisión a partir de ellos. A diferencia de otras técnicas de *boosting*, *bagging* presenta la característica de que se puede correr en paralelo permitiendo generar todos los conjuntos a la vez [15]. El método de *feature randomness* comparte el mismo fin que la técnica de *bagging*, pero se basa en la utilización de subconjuntos aleatorios de atributos en el momento de escoger la próxima variable ocupante del nodo de decisión [15]. Con la combinación de estos métodos, el algoritmo de *random forest* es capaz de romper con los sesgos que pudiesen estar presentes y basar su modelo de estudio sobre una mayor cantidad de árboles de decisión no correlacionados entre sí.

En primera instancia, el clasificador *random forest* genera multitud de árboles de decisión, cada árbol de decisión es entrenado con una muestra de los datos de entrenamiento generada a partir del método de *bagging* y en cada nodo, el algoritmo busca en un subconjunto aleatorio de atributos (*feature randomness*) para determinar la división. Para clasificar un caso de estudio en *random forest*, el caso es enviado como entrada a todos los arboles de decisión presentes en el "bosque" generado previamente y la clasificación es determinada en función de los votos recogidos. Es decir, la clasificación final se calculará mediante la media de los resultados ofrecidos por los diferentes árboles de decisión que componen el modelo [16].

5.3. Ajuste de un modelo de Random Forest

Para realizar el ajuste del modelo basado en este clasificador es necesario ajustar el parámetro **n_estimators**. Este parámetro representa el número de árboles que compondrá el

modelo y sobre los cuales se evaluará cada caso de estudio. Además, se fija el parámetro *random_state* para poder replicar los valores de exactitud del modelo con los mismos parámetros de entrada. Este parámetro ayuda a controlar la aleatoriedad del algoritmo al generar los árboles de decisión.

Para encontrar el número óptimo de estimadores, se incluye la clase *GridSearchCV* (sección 3.3). Se establece como parámetros una lista de *n_estimators* de valores comprendidos entre el 1 y el 90, también se define *random_state* en 0. La correcta elección del número de estimadores influye significativamente en la exactitud del modelo.

```
1  from sklearn.model_selection import GridSearchCV
2  # Definición del modelo clasificador RF
3  model = RandomForestClassifier(random_state=0)
4  # Definición de los parametros sobre los que queremos que itere nuestro algoritmo
5  params ={'n_estimators': range(1,90)}
6  # Definición del grid que ejecutará nuestro modelo en función de los parámetros
7  # introducidos(params) realizando un 10-fold cross validation (cv=10)
8  grid = GridSearchCV(model, params, cv=10)
```

Figura 5.2: Codificación de un modelo RF con GridSearchCV

Una vez está definido el *GridSearchCV* se entrena el modelo como muestra la figura 5.3, y una vez termina la ejecución podemos recoger los datos del mejor estimador gracias a la línea número 6 de la figura. En este caso, el mejor valor devuelto es *n_estimators* igual a 49 con un valor de exactitud medio de 0,891. En este caso, se debe tener en cuenta que *GridSearchCv* no retorna los valores máximos sino que, devuelve el modelo que mejor media ha obtenido en la ejecución de los grupos realizada por la validación cruzada.

```
1  # El gridSearch definido entrena los modelos en base a los datos de los
2  # parámetros introducidos
3  grid.fit(X_Train, Y_Train)
4
5  # Se recogen los valores óptimos para el modelo RF
6  grid.best_estimator_.get_params()
```

Figura 5.3: Codificación de un modelo RF con GridSearchCV

Una vez se conocen los valores devueltos, validamos el modelo y el valor de exactitud devuelto con el fin de detectar problemas de sobreajuste o subajuste. Para ello, se calculan las predicciones para un nuevo conjunto de datos no utilizados durante la fase de entrenamiento y la exactitud del modelo sobre este conjunto de datos. En este caso se obtienen valores de exactitud ligeramente superiores a los recogidos previamente, lo que indica que el modelo no presenta problemas de sobreajueste ni subajuste.

```
predicciones = grid.best_estimator_.predict(X_Test)
accuracy = accuracy_score(Y_Test, predicciones)
```

```
>>>  accuracy: 0.8910550458715596
```

Figura 5.4: Validación del modelo escogido por GridSearchCV

La figura 5.5 refleja gráficamente la variabilidad de la exactitud del modelo en función de el parámetro *n_estimators*, para ello se han unido los puntos que conforman tres grupos, uno naranja, representando los valores mínimos de cada grupo generado por *k-fold cross validation*, uno verde, representando la media de valores de exactitud de cada grupo y uno azul, representando los máximos. Se observa como varía la exactitud del modelo en función del valor del parámetro, en este caso, alrededor de un 5 % comparando los valores con los que componen la misma línea y alrededor de un 10 %, haciendo una comparación global. En este caso, el valor máximo encontrado se encuentra en el grupo de máximos, es por eso que no coincide con el valor devuelto comentado previamente.

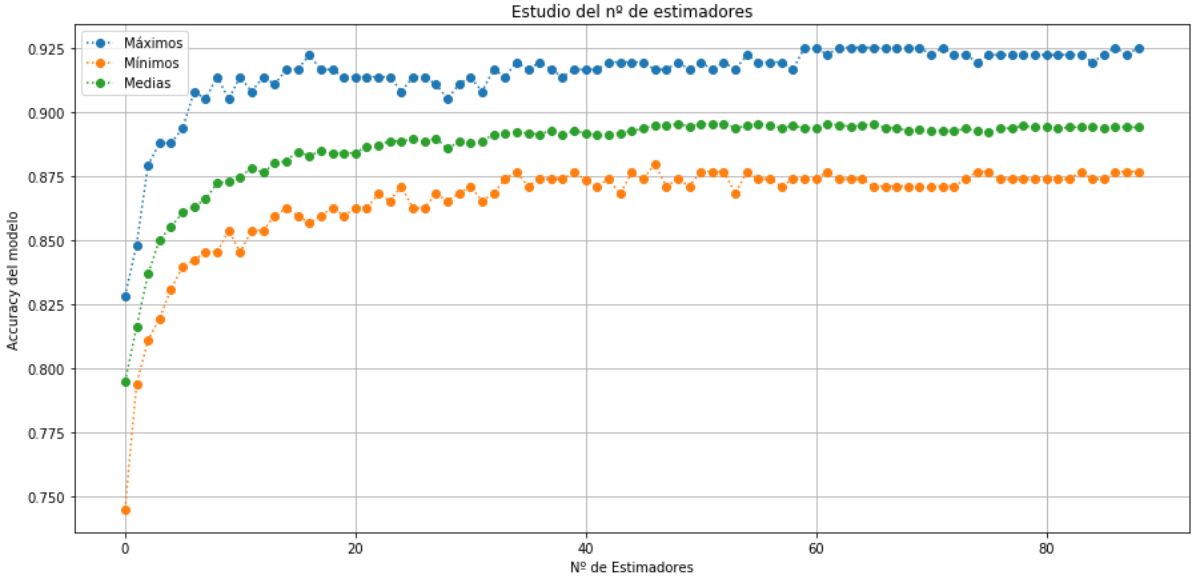

Figura 5.5: Representación de exactitudes frente al número de estimadores.

5.4. Conclusiones del modelo

Después de fijar el número de estimadores óptimo en 49, siendo éste el número de árboles de decisión que compondrán el modelo y a partir del fichero de datos que combina las técnicas de normalización (sección 2.6) y *separate class method* (sección 2.2) la tasa de acierto es de 0,89105504, es decir, un 89,1 % de exactitud aproximadamente. La figura 5.6 muestra las métricas recogidas del modelo. La precisión del modelo para la predicción de bacteriemia de un 86 % y de no bacteriemia un 91 %, la sensibilidad (*recall*) para la detección de bacteriemia es del 92 % de los casos presentes en el *dataframe*, es decir, de

todos los casos de bacteriemia de los casos de entrada, detecta el 91 % y, en el caso de no bacteriemia, el 84 %.

	precision	recall	f1-score	support
1	0.86	0.92	0.89	441
3	0.91	0.84	0.88	431

Figura 5.6: Métricas del modelo clasificador basado en RF

5.4.1. Matriz de confusión

Como se refleja en el capítulo anterior, la matriz de confusión (Figura 5.7) representa gráficamente los aciertos y fallos en las predicciones realizadas por el modelo diferenciando entre: verdaderos positivos o predicciones de bacteriemia acertadas (esquina superior izquierda), falsos positivos o predicciones de bacteriemia erróneas (esquina inferior izquierda), verdaderos negativos o predicciones de no bacteriemia acertadas(esquina inferior derecha) y falsos negativos o predicciones de no bacteriemia erróneas (esquina superior derecha). A partir de los datos reflejados en la matriz de confusión, es posible calcular la exactitud del modelo:

$$\begin{aligned} exactitud &= \frac{\text{Verdaderos Positivos } + \text{ Verdaderos Negativos}}{\text{N}^o \text{ total casos}} \\ &= \frac{382 + 395}{382 + 395 + 32 + 63} = 0{,}89105504 \end{aligned} \tag{5.1}$$

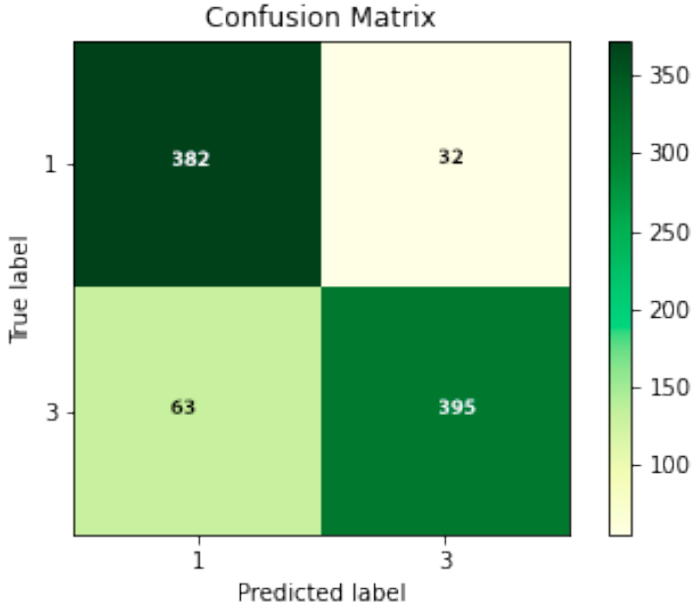

Figura 5.7: Matriz de confusión

Por otro lado, la curva de característica operativa del receptor (ROC) representa gráficamente el rendimiento de un modelo de clasificación. Esta curva enfrenta la sensibilidad o *recall* mencionada previamente (véase ecuación (4.7)) frente a la tasa de falsos positivos (FPR, ecuación (5.2))[17].

$$FPR = \frac{\text{Falsos Positivos}}{\text{Falsos Positivos} + \text{Verdaderos Negativos}} \tag{5.2}$$

Como se indica previamente, la curva ROC representa gráficamente la tasa de verdaderos positivos frente a la tasa de falsos positivos para todos los valores del umbral de clasificación, este umbral representa la proporción de votos de "bacteriemia" frente a "no bacteriemia" devueltos por cada uno de los árboles de decisión para hacer la clasificación. Es decir, en función del umbral de clasificación, se requerirá un mayor o menor número de votos de tipo "bacteriemia" para clasificar finalmente un caso de estudio como positivo. En lugar de evaluar el modelo múltiples veces y calcular los puntos de la curva, se aplica un algoritmo denominado *"area under the curve"* (AUC) o área bajo la curva. Este algoritmo calcula de manera eficiente la agregación del rendimiento total del modelo. La figura 5.8 representa gráficamente la curva ROC (azul) con un área bajo la curva (AUC) de 0.94. La recta naranja representa la curva ROC de un modelo con un 50 % de precisión. Un claro ejemplo sería la utilización de una moneda para decidir, esta técnica tendría un 50 % de probabilidades de acertar. En definitiva, cuanto más rápido llegue la curva al máximo (1) mayor será el área bajo la curva y mejor será el modelo.

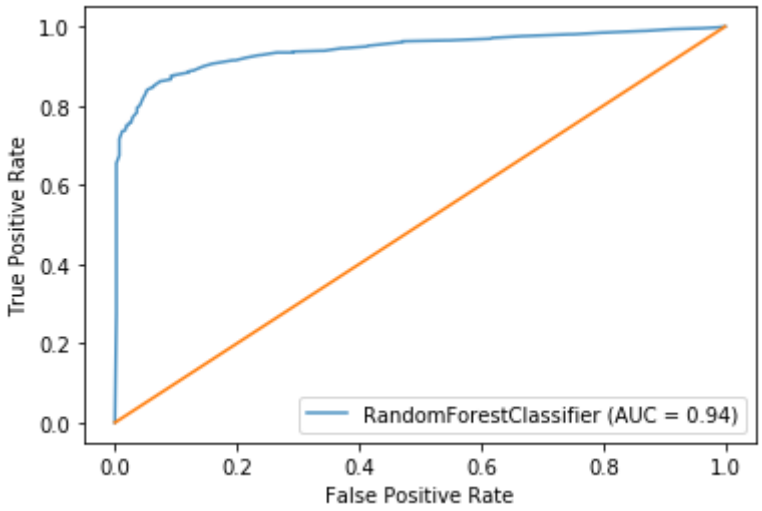

Figura 5.8: Curva ROC del modelo *random forest*

5.4.2. Peso de los atributos en el algoritmo

En esta sección se da a conocer la influencia que tienen los atributos sobre el modelo clasificador basado en RF utilizado en el proyecto. Es bastante importante ya que permite conocer que atributos son las que realmente tienen un mayor impacto sobre el modelo y, por consiguiente, son más importantes incluir en los casos que se vayan a evaluar en un

futuro, es decir, se buscará que los casos de estudio contengan, al menos, los atributos que encabezan la lista. Para obtener los datos es posible apoyarse sobre la librería *shap* que incluye un método particular para los algoritmos que utilizan árboles, *TreeExplainer*. La figura 5.9 refleja la lista de atributos más importantes del modelo. Se puede ver que destacan *origensos* (origen sospechado de la bacteriemia), *medio* (medio de crecimiento de la bacteriemia) y *frasanae* (crecimiento al menos en frasco de anaerobios).

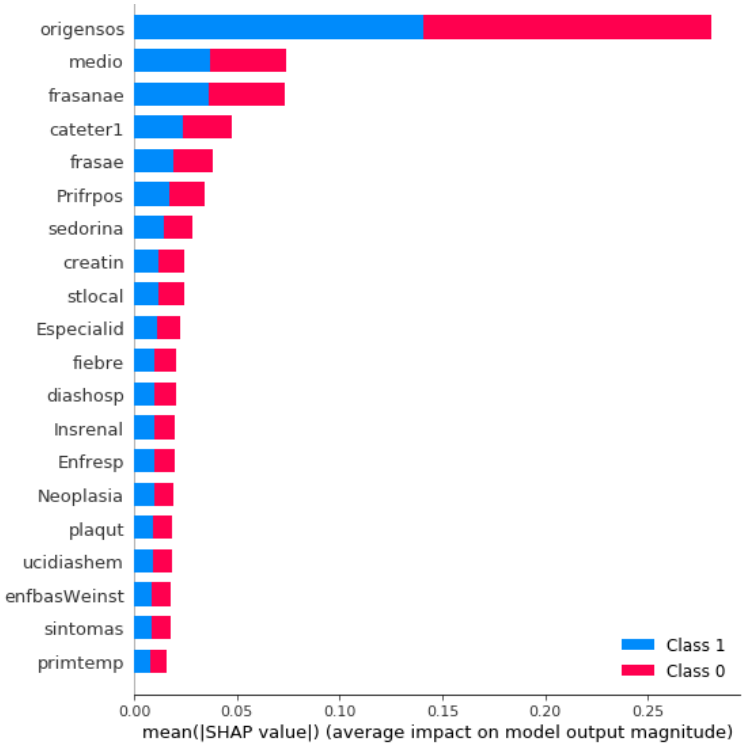

Figura 5.9: Influencia de los atributos en el modelo basado en RF

Capítulo 6

K-Nearest Neighbors

6.1. Introducción a K-Nearest Neighbor

K-NN es uno de los algoritmos más sencillos dentro del mundo del aprendizaje automático y por ello es muy recomendado para todo aquel que desee iniciarse en este ámbito. Sencillamente, es una técnica que clasifica un nuevo valor en función de sus vecinos o puntos más cercanos. Se emplea principalmente en sistemas de recomendación, detección de anomalías y búsqueda semántica. Una de las mayores desventajas que presenta el modelo es que utiliza todo el dataset para clasificar cada punto, lo cual requiere de un uso masivo de memoria y procesamiento. Por ello, K-NN suele funcionar mejor con volúmenes de datos pequeños.

6.2. Funcionamiento de K-NN

Este algoritmo clasifica cada entrada nueva en el grupo que le corresponda en función de la variable k, es decir, según los k vecinos más próximos a un grupo determinado. Para ello, calcula la distancia de la nueva entrada a cada dato ya existente en el modelo y ordena esas distancias de menor a mayor para escoger el grupo al que pertenece. El grupo escogido será aquel que represente la menor distancia, o bien, aquel grupo con mayor representación en un espectro longitudinal mayor. Por ejemplo, la figura 6.1, se aprecia la representación de un algoritmo 3-NN, donde para la evaluación de cada caso de estudio (representado con un triángulo naranja en el gráfico) se toma como referencia los tres puntos más próximos al estudio. En este caso en particular, el clasificador devolverá que dicho punto pertenece al grupo amarillo a pesar de que el punto azul sea más próximo ya que el grupo amarillo tiene mayor representación que el azul.

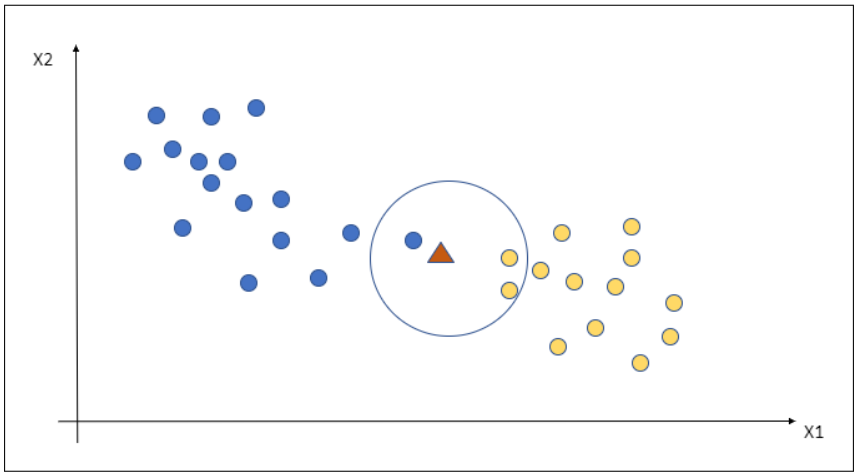

Figura 6.1: Representación gráfica de una clasificación 3-NN

Partiendo del funcionamiento de K-NN, se aprecia que no es un algoritmo que aprenda y después se base en lo aprendido, sino que memoriza las instancias en el período de entrenamiento y posteriormente conformarán su base de conocimiento en la fase de predicción. Cabe destacar como dato importante la imparidad del parámetro k para evitar empates a la hora de escoger el grupo al que debe pertenecer el nuevo dato.

6.3. Ajuste de un modelo K-NN

Ya se ha detallado en la sección anterior el parámetro clave k sobre el cual gira el comportamiento de K-NN. Por ello, se realiza un estudio de dicho parámetro para escoger el valor que más optimice el modelo, suministrando a la función GridSearchCV (detallada en la sección 3.3) un intervalo de posibles valores para k, un modelo del algoritmo a optimizar, en este caso KNN, y un valor para el cross validation de la propia función, método explicado en la sección 3.2.

```
#Modelo para método Grid
model = KNeighborsClassifier()
#Intervalo de valores de K
params ={'n_neighbors': range(1,20)}
#Función Grid
grid = GridSearchCV(model, params, cv=10)
```

Una vez obtenido el modelo ideal en la variable grid con la variable k determinada en 14, se procede a comparar la tasa de precisión del modelo devuelto por la función GridSearchCV con la tasa de precisión de la fase de validación. El modelo tiene un porcentaje de 85 por ciento y la fase de validación 87. De estos porcentajes se extrae la idea de tener unos datos equilibrados, puesto que ambos resultados son bastante similares y nos alejan del concepto de sobre-ajuste en los datos, para ver más información acerca de este concepto ver sección 3.

6.4. Conclusiones del modelo

Para poder conocer a fondo el grado de precisión o de acierto del modelo en cuestión, existen diversas métricas para medir el desempeño de la técnica. Destacan, sobre todo, las cuatro citadas en 4.5.

Los resultados obtenidos de estas métricas con el modelo se muestran en la figura 6.2, donde se aprecia un 83 por ciento de acierto en el modelo al predecir pacientes infectados y un 93 por ciento al detectar pacientes sanos. Con respecto a la sensibilidad, el algoritmo detecta al 93 por ciento aquellos pacientes infectados con respecto a los infectados en los datos de entrada. Esta tasa disminuye al tratarse de aquellos pacientes sanos, puesto que el porcentaje baja al 81 por ciento. La media de ambas métricas quedan fijadas en un 88 y 87 por ciento respectivamente.

	precision	recall	f1-score	support
1	0.83	0.93	0.88	429
3	0.93	0.81	0.87	443

Figura 6.2: Resultados obtenidos KNN

6.4.1. Matriz de confusión

La figura 6.3 es la matriz de confusión del algoritmo KNN empleado bajo la configuración de k ya descrita anteriormente. Los resultados obtenidos son los siguientes: el número de pacientes detectados con bacteriemia correctamente (cuadrante superior izquierdo) es muy superior a aquellos detectados con bacteriemia erróneamente (cuadrante superior derecho). Los pacientes a los cuales no se les ha detectado bacteriemia correctamente (cuadrante inferior derecho) también es superior a aquellos a los que no se les ha detectado erróneamente (cuadrante inferior izquierdo), pero con una tasa de acierto menor.

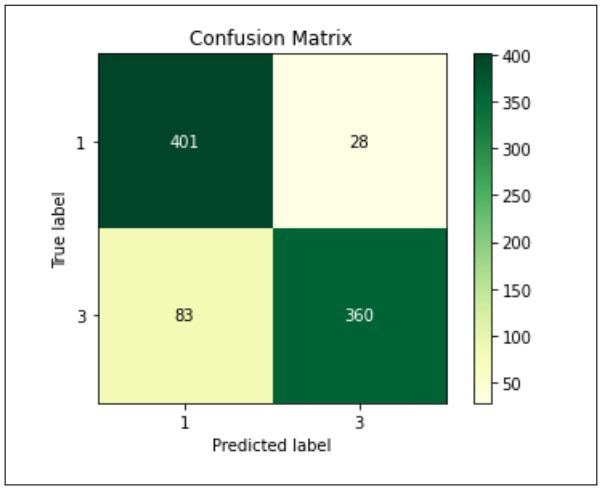

Figura 6.3: Matriz de confusión

6.4.2. Peso de las atributos en el algoritmo

La figura 6.4 muestra el listado de todas los atributos suministrados al modelo en función del peso de las mismas en el algoritmo en orden descendente. Los atributos destacados son: *origensos*, el origen sospechado de la bacteriemia a extracción de hemocultivos, *medio*, el medio de crecimiento del verdadero positivo, *mes*, el mes de recogida de la bacteriemia, *Prifrpos*, primer frasco donde crece el patógeno y *sedorina*, sedimento de orina. Estos cinco atributos tienen un peso superior a 0.02, destacando medio con casi un 0.05 y origensos, que se desmarca llegando a valores de un 0.12.

Weight	Feature
0.1266 ± 0.0160	origensos
0.0459 ± 0.0096	medio
0.0275 ± 0.0098	mes
0.0268 ± 0.0097	Prifrpos
0.0204 ± 0.0081	sedorina
0.0167 ± 0.0085	microbpoli
0.0108 ± 0.0018	fiebre
0.0099 ± 0.0079	stlocal
0.0099 ± 0.0069	anhonpol
0.0096 ± 0.0061	enfbasWeinst
0.0083 ± 0.0049	Coagulación
0.0076 ± 0.0047	sexo
0.0057 ± 0.0029	Inghosp1m
0.0053 ± 0.0023	leucocit
0.0046 ± 0.0021	frasae
0.0041 ± 0.0037	Insrenal
0.0039 ± 0.0011	Anaerobio
0.0034 ± 0.0044	trombope
0.0030 ± 0.0049	Diabetes
0.0023 ± 0.0038	hipotens
… 49 more …	

Figura 6.4: Influencia de atributos en KNN

Capítulo 7

Resultados de los modelos clasificadores generados

A continuación, se recogen los resultados de los estudios realizados sobre los distintos algoritmos de aprendizaje automático presentes en los capítulos previos. Para cada uno de ellos, se ha realizado un estudio de como trabajan las diferentes técnicas propuestas en el capítulo 2 de tratamiento de datos, es decir, técnicas de manejo de datos perdidos, normalización de valores para cada atributo y manejo de los atributos categóricos.

7.1. Introducción

Después de ver todas las técnicas, métodos y algoritmos empleados a lo largo del proyecto, se desarrolló un algoritmo para generar gráficas con los resultados de los algoritmos de aprendizaje máquina. Tras aplicar las técnicas de tratamiento de datos (descritas en el capítulo 2) sobre los datos de los pacientes se obtiene como resultado números *dataframes* diferentes almacenados en archivos CSV para su posterior utilización. El cuadro 7.1 muestra los nombres de los ficheros generados y una breve explicación de las técnicas de tratamiento de datos utilizadas para su creación.

Cuadro 7.1: Cuadro explicativo del contenido de los ficheros resultantes de las técnicas de tratamiento de datos

Nombre del Fichero	Descripción
nan_mixed_imputation	Imputación de los valores perdidos mediante imputación de valores categóricos con moda y de valores numéricos con medias
nan_mode_imputation	Imputación de los valores nulos mediante imputación de valores por moda
separate_class_method_data	Imputación de valores perdidos mediante la técnica de *separate class method*

discarding	Imputación de valores perdidos mediante la técnica de descartando atributos y/o casos de estudio
one_hot_discarding	Aplicación de *one hot encoder* sobre el fichero discarding.csv. Este fichero combina *one hot encoder* y el descarte de atributos y/o casos de estudio
one_hot_separate	Aplicación de *one hot encoder* sobre el fichero separate_class_method_data.csv. Este fichero combina *one hot encoder* y *separate class method data*
one_hot_mixed_imputation	Aplicación de *one hot encoder* sobre el fichero nan_mixed_imputation.csv. Este fichero combina *one hot encoder* y la imputación de valores nulos mediante una regla mixta (media para valores numéricos y moda para valores categóricos)
normalized_discarding	Normalización de los valores recogidos por el fichero discarding.csv. De este modo se obtienen los datos resultantes de la técnica de descarte de atributos y/o valores normalizados.
normalized_separate	Normalización de los valores recogidos por el fichero separate_class_method_data.csv. De este modo se obtienen los datos resultantes de la técnica *separate class method data* normalizados.
normalized_mixed_imputation	Normalización de los valores recogidos por el fichero nan_mixed_imputation.csv. De este modo se obtienen los datos resultantes de la imputación mixta de valores perdidos normalizados.
one_hot_normalized_separate	Datos resultantes de la combinación de *one hot encoder*, la normalización de datos y *separate class method*
one_hot_normalized_discarding	Datos resultantes de la combinación de *one hot encoder*, la normalización de datos y descartando atributos y/o casos de estudio
one_hot_normalized_imputation	Datos resultantes de la combinación de *one hot encoder*, la normalización de datos y la imputación mixta de datos perdidos

El algoritmo desarrollado ejecuta de forma continua los tres clasificadores recogiendo datos de precisión, *recall*, *f1-score* y *support*. A su vez recoge los valores mínimos, medios

y máximos de la exactitud de los distintos modelos generados por cada algoritmo clasificador, fichero de datos de entrada y parámetros característicos de cada clasificador. Se apoya en el método *GridSearchCV* (mencionado en la sección 3.3) para la ejecución de los diferentes clasificadores con los parámetros deseados y el valor de k para la generación de grupos de *k-fold cross validation* (Sección 3.2).

7.2. SVM

En esta sección se van a detallar todos los resultados obtenidos al introducir como datos de entrada en el modelo SVM (Capítulo 4) cada uno de los ficheros generados a lo largo de este proyecto cuyas técnicas de limpieza y preprocesado se han explicado en los capítulos referentes a ellos (Capítulos 2 y 3).

La figura 7.1 muestra cuatro grupos de datos enlazados por líneas discontinuas donde cada una de ellas representa un tipo de dato extraído: los puntos azules representan la tasa de acierto máxima de los modelos SVM, los naranjas los máximos en la fase de validación, los verdes las medias de las tasas de acierto en los modelos generados y los rojos los mínimos obtenidos. De la gráfica se extrae que los ficheros óptimos son *normalized_separate* y *one_hot_normalized_separate*. En efecto, el fichero utilizado para los resultados mostrados en los capítulos 4, 5 y 6, SVM, RF y KNN respectivamente, ha sido *normalized_separate*. Esta elección se debe al gran número de atributos presentes en los resultados de la técnica *one hot encoder* y el aumento de complejidad que generan. Como se refleja en la sección 3.1 hay que evitar el uso de un número elevado de atributos ya que pueden generar sobreajuste y subajuste.

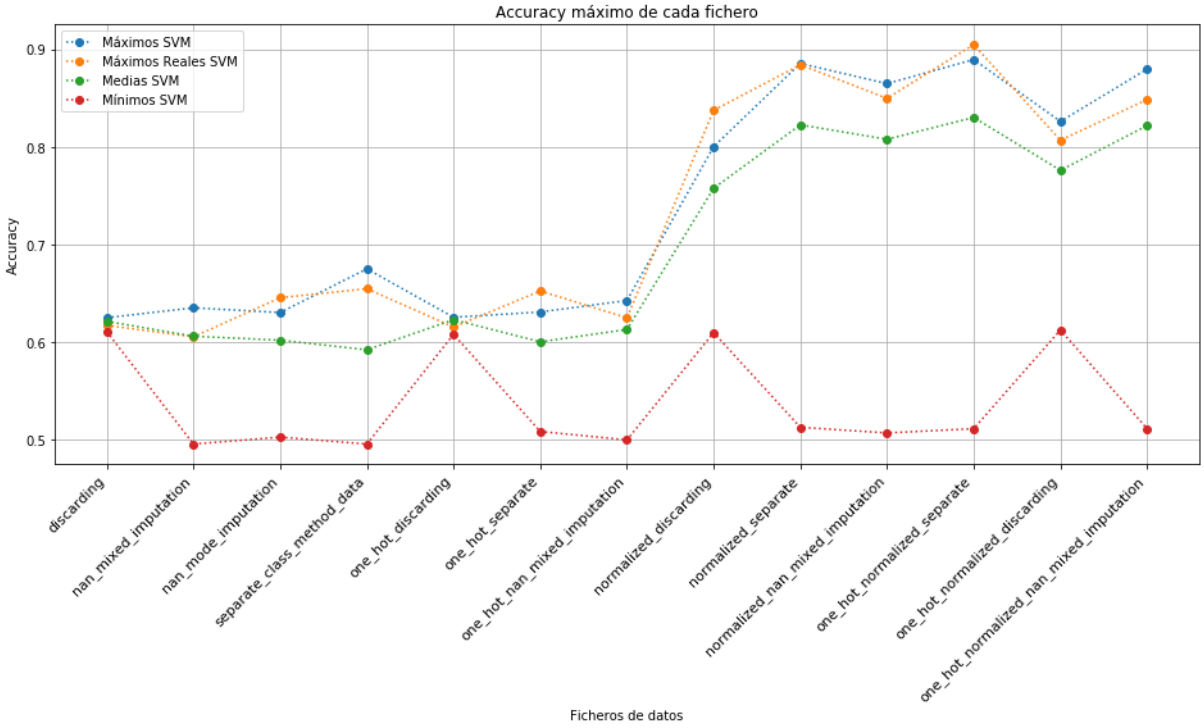

Figura 7.1: Resultados globales del clasificador Support Vector Machine

La tabla 7.2 especifica los resultados obtenidos con las métricas detalladas en la sección 4.5, tanto para pacientes infectados como para los que no lo están, de todos los ficheros generados en el proyecto. Para esta técnica existe un problema de compatibilidad entre *gridSearchCV* y el fichero de datos *discarding* y en la tabla se muestran 0 en los resultados.

Cuadro 7.2: Resultados globales de SVM

nombre	tipo	precision	recall	f1	support
nan_mixed_imputation	Bacteriemia	0.543	0.985	0.700	408
nan_mixed_imputation	No bacteriemia	0.954	0.271	0.422	464
nan_mode_imputation	Bacteriemia	0.584	0.969	0.729	430
nan_mode_imputation	No bacteriemia	0.918	0.330	0.485	442
separate_class_method_data	Bacteriemia	0.580	0.980	0.729	413
separate_class_method_data	No bacteriemia	0.954	0.361	0.524	459
discarding	Bacteriemia	0.617	1.0	0.763	342
discarding	No bacteriemia	0 .0	0.0	0.0	212
one_hot_discarding	Bacteriemia	0.615	1.0	0.762	341
one_hot_discarding	No bacteriemia	0.0	0.0	0.0	213
one_hot_separate	Bacteriemia	0.593	0.995	0.743	441
one_hot_separate	No bacteriemia	0.984	0.301	0.461	431
one_hot_mixed_imputation	Bacteriemia	0.564	0.983	0.716	421
one_hot_mixed_imputation	No bacteriemia	0.949	0.290	0.444	451
normalized_discarding	Bacteriemia	0.862	0.895	0.878	363
normalized_discarding	No bacteriemia	0.785	0.727	0.755	191
normalized_separate	Bacteriemia	0.869	0.9	0.884	430
normalized_separate	No bacteriemia	0.899	0.868	0.883	442
normalized_mixed_imputation	Bacteriemia	0.808	0.889	0.847	408
normalized_mixed_imputation	No bacteriemia	0.893	0.814	0.852	464
one_hot_normalized_separate	Bacteriemia	0.883	0.927	0.904	425
one_hot_normalized_separate	No bacteriemia	0.927	0.883	0.904	447
one_hot_normalized_discarding	Bacteriemia	0.814	0.900	0.855	352
one_hot_normalized_discarding	No bacteriemia	0.787	0.643	0.708	202
one_hot_normalized_imputation	Bacteriemia	0.818	0.884	0.85	423
one_hot_normalized_imputation	No bacteriemia	0.881	0.815	0.847	449

7.3. RF

Al igual que se ha explicado en la sección anterior, se muestran y detallan los resultados obtenidos con el modelo clasificador basado en *random forest*. La figura 7.2 muestra los diferentes grupos de resultados producto del análisis de los modelos de RF. El grupo de color azul representa los máximos de los modelos en la fase de entrenamiento, el verde representa las medias de los diferentes modelos generados, el naranja los resultados generados en la fase de validación de los modelos calificados como mejores por *GridSearchCV* (sección 3.3) y en rojo, los valores mínimos recogidos. Los mejores resultados son obtenidos de nuevo por los ficheros *normalized_separate* y *one_hot_normalized_separate*, aunque en este caso también destaca el archivo *one_hot_separate*. Tal y como se explicó en la sección anterior, los ficheros generados a partir de la técnica *one_hot_encoder* quedan descartados. El fichero escogido para este modelo es *normalized_separate* englobando las técnicas de normalización de datos (sección 2.6) y *separate class method* (sección 2.2).

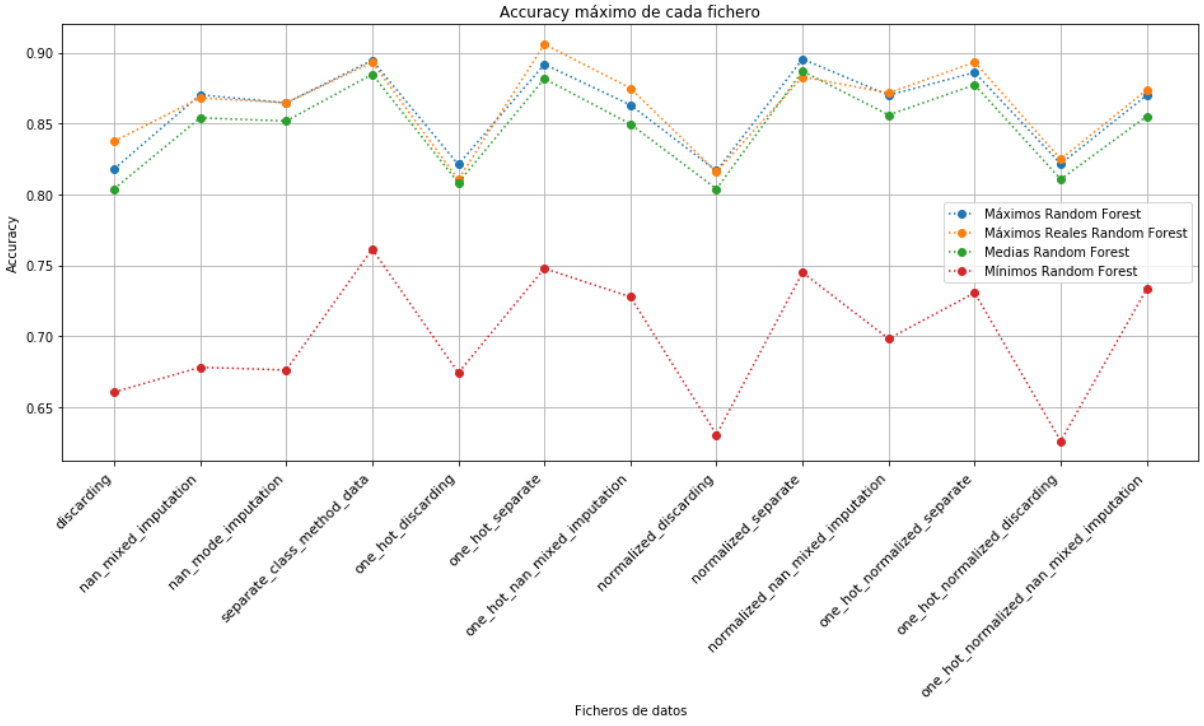

Figura 7.2: Resultados globales del clasificador Random Forest

La tabla 7.3 muestra los valores utilizados para la evaluación de los diferentes modelos, precisión, *recall, f1-score* y *support*. Si se analizan los valores de la columna de *precision* y *recall* se ratifican las conclusiones de la gráfica de la figura 7.2. Los mejores valores vuelven a presentarse para los ficheros que incluyen la técnica de *separate class method*.

Cuadro 7.3: Resultados globales de RF

nombre	tipo	precision	recall	f1	support
nan_mixed_imputation	Bacteriemia	0.832	0.909	0.869	421
nan_mixed_imputation	No bacteriemia	0.907	0.829	0.866	451
nan_mode_imputation	Bacteriemia	0.835	0.898	0.865	424
nan_mode_imputation	No bacteriemia	0.896	0.832	0.863	448
separate_class_method_data	Bacteriemia	0.858	0.930	0.892	417
separate_class_method_data	No bacteriemia	0.930	0.859	0.893	455
discarding	Bacteriemia	0.842	0.918	0.878	355
discarding	No bacteriemia	0.826	0.693	0.754	199
one_hot_discarding	Bacteriemia	0.825	0.880	0.852	344
one_hot_discarding	No bacteriemia	0.780	0.695	0.735	210
one_hot_separate	Bacteriemia	0.876	0.938	0.906	424
one_hot_separate	No bacteriemia	0.937	0.875	0.905	448
one_hot_mixed_imputation	Bacteriemia	0.861	0.891	0.876	434
one_hot_mixed_imputation	No bacteriemia	0.888	0.858	0.873	438
normalized_discarding	Bacteriemia	0.825	0.904	0.862	355
normalized_discarding	No bacteriemia	0.793	0.658	0.719	199
normalized_separate	Bacteriemia	0.858	0.926	0.891	450
normalized_separate	No bacteriemia	0.914	0.836	0.873	422
normalized_mixed_imputation	Bacteriemia	0.834	0.923	0.876	432
normalized_mixed_imputation	No bacteriemia	0.916	0.820	0.865	440
one_hot_normalized_separate	Bacteriemia	0.853	0.937	0.893	417
one_hot_normalized_separate	No bacteriemia	0.937	0.852	0.892	455
one_hot_normalized_discarding	Bacteriemia	0.813	0.918	0.862	332
one_hot_normalized_discarding	No bacteriemia	0.849	0.684	0.758	222
one_hot_normalized_imputation	Bacteriemia	0.843	0.910	0.875	426
one_hot_normalized_imputation	No bacteriemia	0.907	0.838	0.871	446

7.4. KNN

En esta sección se muestran y analizan los resultados de la selección de parámetros para el clasificador basado en *k-nearest neighbor* y el análisis de los resultados para cada uno de los ficheros de datos generados mediante las diferentes técnicas de tratamiento de datos (capítulo 2).

La figura 7.3 muestras los valores de exactitud máximos (azul), medios (verde), má-

ximos de la fase de validación(naranja) y los mínimos(rojo) para los diferentes ficheros de datos. Los ficheros que mejores valores de exactitud presentan son los que contienen datos tratados con el método de *separate class method*, *normalized_separate*, *one_hot_normalized_separate*.

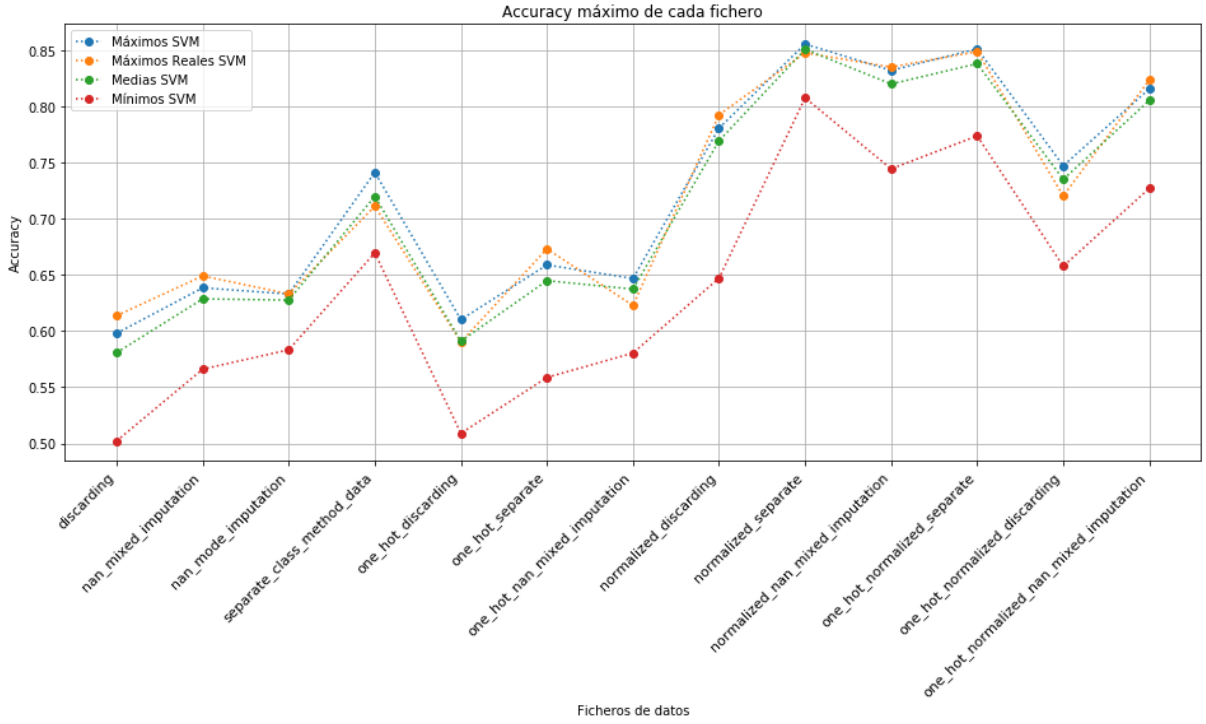

Figura 7.3: Resultados globales del clasificador Random Forest

La tabla 7.4 refleja los valores de *precision, recall, f1-score* y *support* de los mejores modelos generados a partir de los diferentes ficheros de datos, estos valores ratifican la elección de los modelos a partir de los resultados mostrados en la figura 7.3.

Cuadro 7.4: Resultados globales de K-NN

nombre	tipo	precision	recall	f1	support
nan_mixed_imputation	Bacteriemia	0.601	0.812	0.691	422
nan_mixed_imputation	No bacteriemia	0.738	0.495	0.593	450
nan_mode_imputation	Bacteriemia	0.600	0.728	0.658	423
nan_mode_imputation	No bacteriemia	0.679	0.543	0.603	449
separate_class_method_data	Bacteriemia	0.671	0.783	0.723	420
separate_class_method_data	No bacteriemia	0.761	0.643	0.697	452
discarding	Bacteriemia	0.642	0.861	0.735	346
discarding	No bacteriemia	0.466	0.201	0.281	208
one_hot_discarding	Bacteriemia	0.620	0.875	0.726	344
one_hot_discarding	No bacteriemia	0.376	0.123	0.186	210

nombre	tipo	precision	recall	f1	support
one_hot_separate	Bacteriemia	0.597	0.877	0.710	399
one_hot_separate	No bacteriemia	0.828	0.501	0.624	473
one_hot_mixed_imputation	Bacteriemia	0.577	0.804	0.672	420
one_hot_mixed_imputation	No bacteriemia	0.714	0.453	0.554	452
normalized_discarding	Bacteriemia	0.817	0.874	0.845	359
normalized_discarding	No bacteriemia	0.735	0.641	0.684	195
normalized_separate	Bacteriemia	0.794	0.927	0.855	425
normalized_separate	No bacteriemia	0.917	0.771	0.838	447
normalized_mixed_imputation	Bacteriemia	0.845	0.822	0.834	440
normalized_mixed_imputation	No bacteriemia	0.824	0.847	0.835	432
one_hot_normalized_separate	Bacteriemia	0.799	0.917	0.854	422
one_hot_normalized_separate	No bacteriemia	0.909	0.784	0.842	450
one_hot_normalized_discarding	Bacteriemia	0.729	0.878	0.797	347
one_hot_normalized_discarding	No bacteriemia	0.691	0.454	0.548	207
one_hot_normalized_imputation	Bacteriemia	0.810	0.829	0.819	422
one_hot_normalized_imputation	No bacteriemia	0.836	0.817	0.826	450

7.5. PCA

Esta sección refleja los resultados obtenidos por la técnica de PCA (descrita en la sección 3.4). Como se comenta en la sección dedicada a PCA, esta técnica presenta una baja explicabilidad y, por consiguiente, transmite esa baja explicabilidad a los algoritmos de aprendizaje máquina. El fichero de datos seleccionado para hacer las pruebas es *one_hot_normalized_separate_class_method_data*, fichero que combina las técnicas de *one hot encoder* (sección 2.7), la normalización de datos (sección 2.6) y *separate class method* (sección 2.2). En este caso, se escoge el archivo con *one hot encoder* para reducir con PCA el número de atributos que la técnica produce. La tabla 7.5 muestra los resultados obtenidos con los diferentes valores de varianza acumulada, se organiza en varías columnas que reflejan la proporción de varianza acumulada de PCA, el número de atributos que resultan tras la aplicación de la técnica para cada valor de varianza acumulada, la tasa de acierto del modelo en la fase de entrenamiento, la tasa de acierto del modelo en la fase de validación, y el porcentaje de diferencia entre ambos. Después de analizar los datos, se puede ver que hay una ligera mejora respecto a los datos originales, aumentando la tasa de acierto hasta el 90 % en algunos casos.

Cuadro 7.5: Resultados de PCA

Porcentaje	N^o de atributos	Exactitud	Diferencia	Exactitud de la validación
0.35	9	0.81	0.22 %	0.81
0.36	9	0.80	-0.61 %	0.80

Porcentaje	N^o de atributos	Exactitud	Diferencia	Exactitud de la validación
0.37	11	0.80	1.59 %	0.82
0.38	12	0.82	-0.13 %	0.82
0.39	12	0.83	-0.76 %	0.82
0.40	14	0.84	0.53 %	0.85
0.41	15	0.83	1.33 %	0.84
0.42	18	0.82	2.14 %	0.85
0.43	17	0.83	1.88 %	0.85
0.44	20	0.82	3.17 %	0.85
0.45	22	0.83	1.51 %	0.85
0.46	22	0.84	1.16 %	0.85
0.47	24	0.86	-3.57 %	0.82
0.48	26	0.86	-2.71 %	0.83
0.49	27	0.85	-0.59 %	0.84
0.50	28	0.85	0.56 %	0.86
0.51	30	0.86	-0.05 %	0.86
0.52	33	0.86	-1.85 %	0.85
0.53	35	0.84	1.10 %	0.85
0.54	39	0.85	1.33 %	0.87
0.55	43	0.86	0.56 %	0.87
0.56	41	0.86	-0.91 %	0.85
0.57	41	0.86	-0.10 %	0.86
0.58	46	0.85	0.07 %	0.85
0.59	50	0.87	1.10 %	0.88
0.60	55	0.87	-0.56 %	0.86
0.61	54	0.88	-1.31 %	0.86
0.62	56	0.87	-0.56 %	0.86
0.63	56	0.88	-0.62 %	0.87
0.64	58	0.88	-2.51 %	0.85
0.65	64	0.86	0.67 %	0.86
0.66	65	0.88	-1.74 %	0.86
0.67	75	0.90	-2.31 %	0.88
0.68	69	0.87	0.41 %	0.87
0.69	75	0.89	-2.46 %	0.87
0.70	76	0.91	-4.41 %	0.86
0.71	77	0.86	2.79 %	0.89
0.72	86	0.89	-0.22 %	0.88
0.73	88	0.88	-1.51 %	0.87
0.74	86	0.86	1.47 %	0.88
0.75	93	0.87	1.79 %	0.88
0.76	96	0.88	0.15 %	0.89
0.77	96	0.86	4.28 %	0.90
0.78	95	0.87	0.64 %	0.88
0.79	98	0.87	1.50 %	0.88
0.80	104	0.87	2.59 %	0.89
0.81	113	0.88	-0.05 %	0.88

Porcentaje	N^o de atributos	Exactitud	Diferencia	Exactitud de la validación
0.82	114	0.88	-0.19 %	0.88
0.83	111	0.87	1.67 %	0.89
0.84	117	0.89	-4.15 %	0.85
0.85	118	0.88	-1.88 %	0.86
0.86	125	0.89	-0.54 %	0.88
0.87	128	0.90	-1.63 %	0.89
0.88	125	0.88	0.30 %	0.89
0.89	138	0.86	1.45 %	0.87
0.90	140	0.87	2.51 %	0.89
0.91	134	0.88	0.96 %	0.89
0.92	145	0.88	2.05 %	0.90
0.93	152	0.87	2.33 %	0.89
0.94	152	0.89	-1.48 %	0.87
0.95	161	0.88	-0.13 %	0.88
0.96	167	0.88	1.64 %	0.89
0.97	173	0.89	-0.74 %	0.89
0.98	174	0.87	1.82 %	0.89
0.99	188	0.89	-0.11 %	0.89

La figura 7.4 muestra las tasas de acierto obtenidas en la fase de entrenamiento (azul) y la tasa de acierto obtenida en la fase de validación (naranja). De este modo, se muestra la diferencia entre ambas tasas de acierto y la importancia de utilizar un valor de varianza acumulada adecuado ya que puede mejorar considerablemente los resultados.

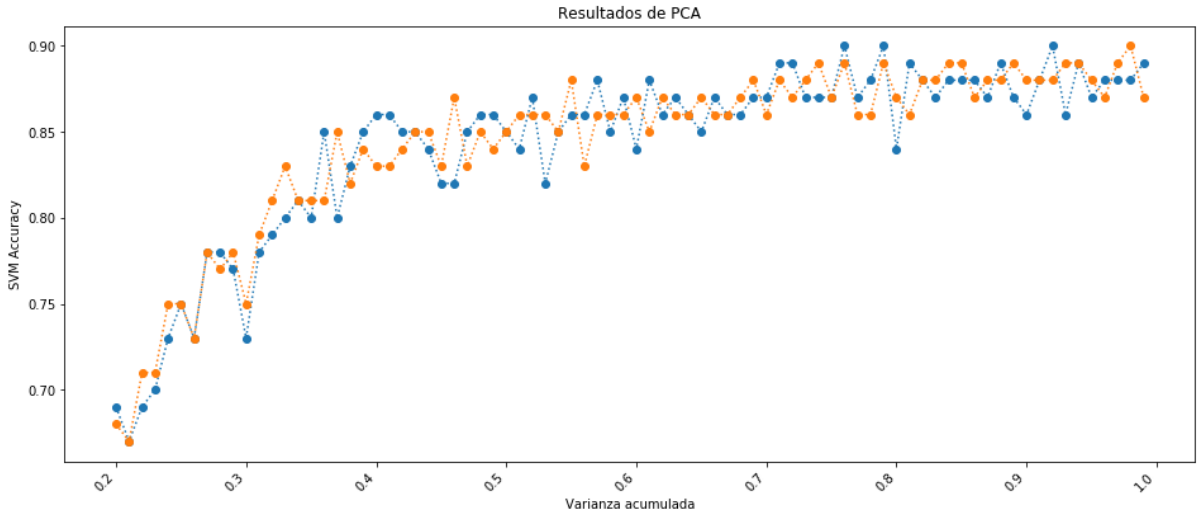

Figura 7.4: Resultados de PCA con el algoritmo de SVM

Por el contrario, esta técnica reduce considerablemente la explicabilidad del modelo. Esto se debe a la utilización de las componentes principales como nuevos atributos y, como consecuencia, la pérdida de los atributos originales. Dado que el estudio tiene como propósito final la utilización de los algoritmos de aprendizaje máquina en el campo de

la medicina, la explicabilidad del modelo tiene gran importancia. Es por esto que la utilización de esta técnica se descarta.

Capítulo 8

Nuevo estudio

8.1. Introducción: ¿Porqué un nuevo estudio?

Tras realizar el estudio completo de las técnicas de aprendizaje automático descritas en los capítulos anteriores, se descartaron los atributos: *frasae*, crecimiento al menos en frasco de aerobios, *frasanae*, crecimiento al menos en frasco de anaerobios *prifrpos*, primer frasco que crece y *medio*, medio de crecimiento del verdadero positivo. Estos atributos fueron descartados ya que representaban valores recogidos transcurridas las primeras 24 horas del hemocultivo, cuando ya empiezan a producirse resultados aunque todavía faltan de 3 a 4 días para tener resultados concluyentes. Transcurridas las primeras 24 horas el hemocultivo puede tener crecimiento bacteriano si bien todavía no se está en condiciones de saber si es debido a contaminación o puede que no haya crecimiento pero que sea debido a un proceso de crecimiento lento. Como consecuencia, los análisis reflejados en los capítulos previos reflejan la efectividad del 85-90 % para casos que requieren hemocultivos con un periodo mínimo de incubación de 1 día. En este capítulo se analizan y reflejan los resultados sin tener en cuenta los resultados no-concluyentes de las primeras 24 horas del hemocultivo.

8.2. Resultados en SVM

El ajuste de los parámetros de este algoritmo se ha realizado según refleja la sección 4.4. A causa de la modificación de los datos de entrada, ha sido necesario volver a ejecutar los algoritmos de entrenamiento y ajuste de parámetros. Como resultado, destaca el modelo con $\gamma =$'auto' y $C = 0{,}9$.

Tras optimizar los hiper-parámetros del modelo, se comparan las tasas de acierto del modelo ideal en la fase de entrenamiento y el resultante de su fase de validación para descartar un sobreajuste o subajuste en el modelo (sección 3). Los porcentajes de exactitud obtenidos son de un 85 % en la fase de entrenamiento del modelo ideal y un 86 % en la

fase de validación. Por tanto, estos valores indican la ausencia de subajuste y sobreajuste.

La figura 8.1 representa los diferentes valores del modelo para las métricas explicadas en la sección 4.5. En comparativa, la precisión en caso de pacientes infectados se mantiene igual, mientras que para los pacientes sin bacteriemia sube a un 93 por ciento. La sensibilidad (*recall*) de los pacientes enfermos sube a un 95 % y, en los no infectados, baja a un 77 %. Las medias (*f1-score*) para ambos casos suben al 88 % y descienden al 85 % respectivamente. Por último, el balanceo de datos de entrada (*support*) aumenta 31 casos de pacientes infectados.

	precision	recall	f1-score	support
1	0.82	0.95	0.88	448
3	0.93	0.77	0.85	424

Figura 8.1: Explicación detallada tasa de acierto SVM en nuevo estudio

La nueva matriz de confusión (figura 8.2) se aprecia en comparación a la matriz de la sección 4.5.1 un aumento de los verdaderos positivos, pasando de 398 a 425, los falsos positivos también aumentan, de 67 a 96. Con respecto a los negativos positivos, se desciende desde 371 hasta 328 y los falsos negativos bajan de 36 a 23.

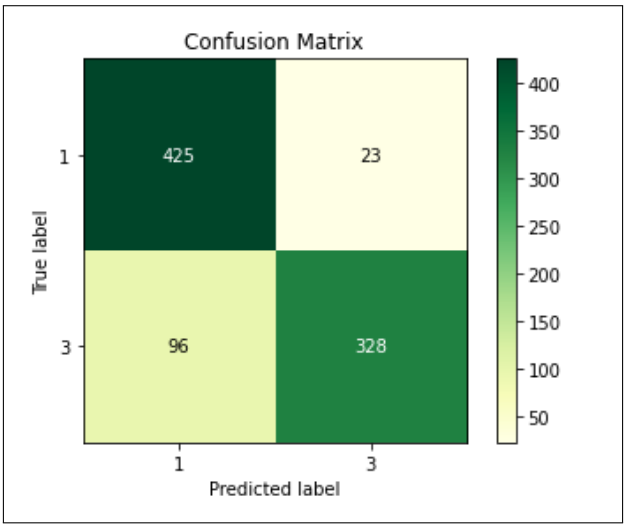

Figura 8.2: Matriz de confusión SVM en nuevo estudio

Por último, el peso otorgado en el atributo, en comparación con lo mostrado en la sección 4.5.2, la figura 8.3 vuelve a otorgar el mayor peso al atributo *origensos*, aumentándolo hasta un 0.5394. *Sedorina* sigue siendo la segunda variable con más peso aún descendiendo a un 0.0573. La tercera variable con más pesos también es *microbpoli*, descendiendo a un 0.0408. A continuación aparece *Anaerobio*, anaerobio frente al resto de bacteriemias incluyendo a las polimicrobianas, con un peso de 0.0183. Cierra el ranking de los cinco primeros el atributo *so*, sistemático orina, con un peso de 0,0092.

Weight	Feature
0.5394 ± 0.0853	origensos
0.0573 ± 0.0139	sedorina
0.0408 ± 0.0089	microbpoli
0.0183 ± 0.0092	Anaerobio
0.0092 ± 0.0151	so
0.0078 ± 0.0103	anhonpol
0.0046 ± 0.0029	cateter1
0.0046 ± 0.0136	mes
0.0046 ± 0.0077	Enfresp
0.0041 ± 0.0079	m_vascul
0.0037 ± 0.0055	enfbasWeinst
0.0032 ± 0.0094	fiebre
0.0032 ± 0.0062	hipotens
0.0028 ± 0.0034	Hongos
0.0028 ± 0.0067	sintomas
0.0023 ± 0.0029	Otrascomor
0.0014 ± 0.0037	COMORBIL
0.0014 ± 0.0062	inmunosu
0.0009 ± 0.0037	m_genitu
0.0005 ± 0.0034	Coagulación
… 45 more …	

Figura 8.3: Pesos de los atributo SVM en nuevo estudio

8.3. Resultados en RF

Para el análisis del modelo clasificador basado en *random forest* en este nuevo estudio, se ha llevado a cabo el mismo procedimiento que en el estudio anterior (capítulo 5). Se ha ajustado el número de estimadores de *random forest* según la sección 5.3, dando lugar a un valor para el parámetro $n\_estimators$ igual a 65. La tasa de acierto del modelo ideal en su fase de entrenamiento es de un 87 % mientras que en la fase de validación el porcentaje disminuye ligeramente llegando a un 86,9 %, es decir, debido a la ligera variación detectada se puede afirmar que no hay presencia de sobreajuste ni subajuste.

La figura 8.4 muestra las métricas del modelo óptimo de este nuevo estudio con RF. En la precisión de bacteriemia (1) se ve un descenso de un 6 % y un aumento del 2 % para no bacteriemia(3), en los datos de *recall* hay un ligero aumento de un 1 % para bacteriemia y de un 3 % para no bacteriemia.

	precision	recall	f1-score	support
1	0.80	0.93	0.86	404
3	0.93	0.81	0.86	468

Figura 8.4: Explicación detallada tasa de acierto RF

La matriz de confusión muestra ligero un descenso de verdaderos positivos, de 382 a 375, un aumento de falsos positivos, de 63 a 91, los verdaderos negativos bajan de 395 a 377 y los falsos negativos bajan de 32 a 29. Es decir, se ha producido un ligero descenso en la exactitud del modelo para realizar la clasificación de bacteriemia y hay un ligero aumento en la predicción de no bacteriemia.

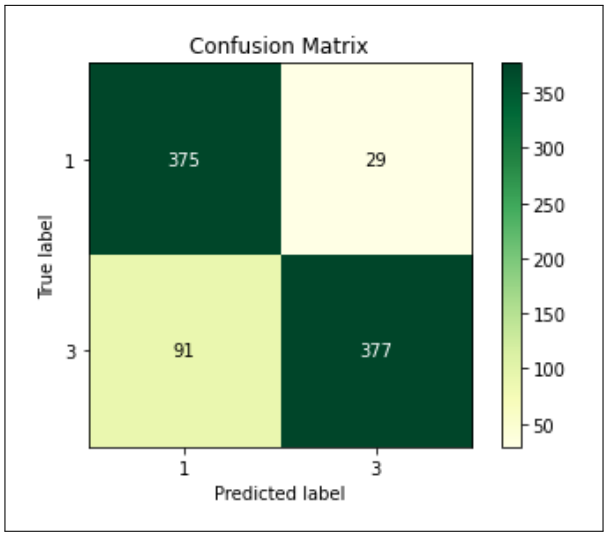

Figura 8.5: Matriz de confusión RF

La figura 8.6 muestra los pesos de los diferentes atributos con mayor importancia. Se observa que *origensos* se mantiene en los primeros puestos, mientras que al quitar los atributos dependientes del hemocultivo, *cateter1* (Tipo de cateter-vía) ocuparía la segunda posición por encima de *creatin* (creatinina) y *stlocal* (síndrome localizador).

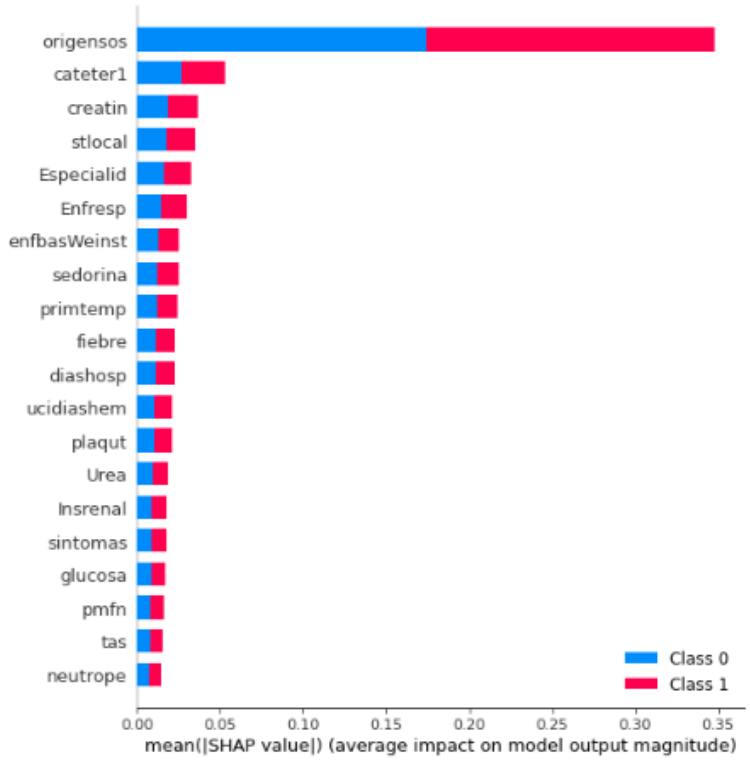

Figura 8.6: Pesos de los atributos RF

La figura 5.8 muestra la curva ROC del modelo RF, en este caso disminuyendo el valor de *AUC* a 0.93. A pesar de haber disminuido sigue siendo un valor bueno para este modelo.

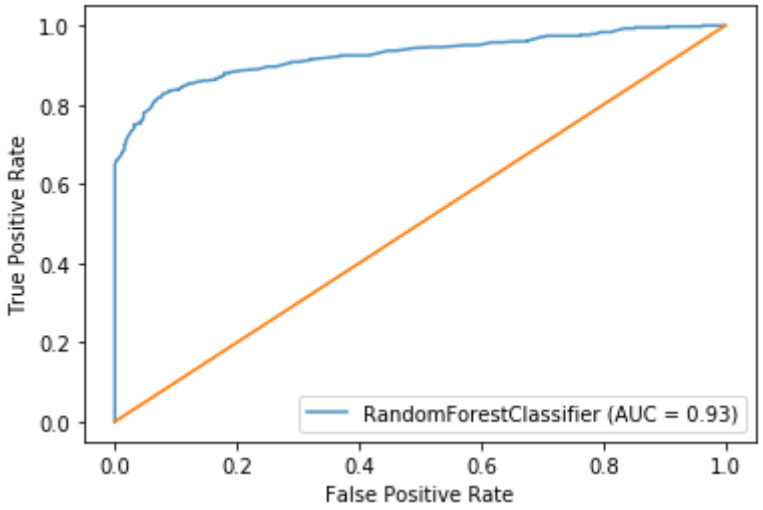

Figura 8.7: Resultados ROC en el nuevo modelo RF

8.4. Resultados en KNN

Este algoritmo ha sido de nuevo ajustado acorde a lo detallado en la sección 6.3, donde el resultado del parámetro k disminuye de 13 a 7. Con este nuevo modelo ideal generado se vuelve a comparar la tasa de acierto producida por el modelo junto a la fase de validación para descartar de igual forma un sobre-ajuste (sección 3) en los atributos de entrada restantes. Los resultados no muestran indicadores de sobre-ajuste, con la salvedad de observar un descenso de un 2 por ciento respectivo tanto en la tasa de acierto del modelo (de 85 a 83) como en la tasa de validación (de 87 a 85).

En la figura 8.8 se muestran los valores de las métricas precision, recall, f1_score y support explicadas en la sección 4.5. Con respecto a los resultados de la sección 6.4 se observa una bajada del 2 por ciento en los valores de precision en el caso de pacientes con bacteriemia y de recall o sensibilidad en los pacientes no infectados, mientras que para la precision de los pacientes sin bacteriemia y el recall o sensibilidad de aquellos infectados, el descenso es de un 1 por ciento. Con respecto al f1_score, la media desciende otro 2 por ciento en ambos casos. Por último, la métrica support indica un balanceo de datos similar, con un aumento de 8 pacientes infectados con respecto al anterior estudio.

	precision	recall	f1-score	support
1	0.81	0.92	0.86	421
3	0.92	0.79	0.85	451

Figura 8.8: Explicación detallada tasa de acierto KNN en nuevo estudio

Los resultados reflejados en la figura 8.8 se manifiestan de igual manera en la matriz de confusión (explicación en la sección 4.5.1), donde ese 2 por ciento que se perdía en la tasa de acierto aumenta los falsos positivos de 83 a 94 y los falsos negativos de 28 a 32.

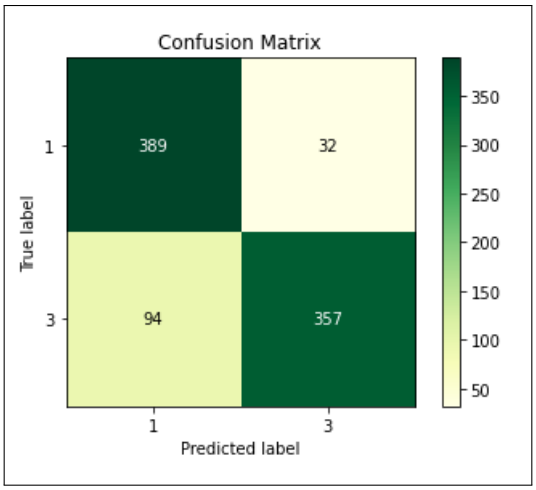

Figura 8.9: Matriz de confusión KNN en nuevo estudio

Por último, tras descartar los atributos detallados en la introducción del capítulo, el listado de los pesos asignados a cada atributo por el modelo queda modificado. El anterior listado se puede observar en la sección 6.4.2. La figura 8.10 vuelve a otorgarle el mayor peso a la variable *origensos* aumentándolo desde un 0.1266 hasta un 0.1858, a continuación aparece *microbpoli*, que ve aumentado su peso de 0.0167 hasta un 0.0209 convirtiéndose en la segunda variable con más peso. En tercer lugar aparece *sedorina*, cuyo peso desciende a 0.0103. En cuarto lugar, *trombope*, trombopenia (enfermedad donde se registran bajos niveles de plaquetas impidiendo una coagulación normal de la sangre), cuyo peso es de 0.0071. En quinto lugar, **mes** y *stlocal*, síndrome localizador, que tienen el mismo peso pero la variación le otorga más influencia a *mes*.

Weight	Feature
0.1858 ± 0.0263	origensos
0.0209 ± 0.0093	microbpoli
0.0103 ± 0.0060	sedorina
0.0071 ± 0.0120	trombope
0.0064 ± 0.0055	mes
0.0064 ± 0.0075	stlocal
0.0062 ± 0.0099	anhonpol
0.0060 ± 0.0084	fiebre
0.0053 ± 0.0063	Coagulación
0.0053 ± 0.0112	cateter1
0.0053 ± 0.0018	Anaerobio
0.0046 ± 0.0086	Inghosp1m
0.0044 ± 0.0042	sintomas
0.0034 ± 0.0021	pcr
0.0034 ± 0.0025	Cardiopatia
0.0032 ± 0.0027	Hepatopatia
0.0025 ± 0.0009	leuc
0.0025 ± 0.0009	hgb
0.0025 ± 0.0022	Especialid
0.0025 ± 0.0027	Alerta
... 45 more ...	

Figura 8.10: Peso de los atributos en KNN en nuevo estudio

Capítulo 9

Conclusiones

En este capítulo se recogen las conclusiones generales del proyecto, diferenciando entre los dos modelos que se ha buscado generar después de encontrar atributos dependientes de datos recogidos del hemocultivo transcurridas las primeras 24 horas. Diferenciamos en dos modelos de clasificación, uno que permite realizar predicciones sin dependencia del hemocultivo, y otro, que realiza predicciones con los datos recogidos transcurridas las primeras 24 horas.

Después de analizar las técnicas de tratamiento de datos perdidos, *complete case data* es descartada. Esta técnica tenía como resultado pocos casos de estudio (ya que elimina todos los datos de un paciente si alguna de las variables está vacía) y sería imposible realizar predicciones sobre casos incompletos. "Descartando casos y/o atributos" es descartada porque también obligaría a utilizar casos de estudio completos. *Separate class method* y la imputación de valores perdidos sí consideran variables vacías y contemplan todos los casos de estudio. Tras los resultados obtenidos, *separate class method* ha sido escogida para el tratamiento de valores perdidos, ya que es la técnica que mejores resultados presenta en todos los análisis. Todos los modelos entrenados con datos tratados por esta técnica presentan una tasa de acierto superior a la obtenida mediante las técnicas previas.

Después, los datos fueron normalizados para evitar la presencia de sesgos dependientes de las magnitudes propias de cada atributo, es decir, para evitar que el modelo dé mayor importancia a un atributo solo por la naturaleza de su magnitud. Asimismo, se realizaron pruebas con *one hot encoder*, que permite dividir las categorías en varios atributos, uno por cada subcategoría. A pesar de generar una ligera mejora en la precisión del modelo, incrementa considerablemente el número de atributos y por consiguiente, el número de dimensiones del modelo. Esto podría generar un subajuste en el modelo y aumentaría su complejidad. Para tratar de reducir el número de atributos se aplicó *principal component analysis* pero, debido a su baja explicabilidad y a la pérdida de los atributos originales del *dataframe*, ambas técnicas fueron descartadas.

En definitiva, se utilizará el fichero de datos que recoge los resultados obtenidos tras combinar *separate class method* y la normalización de datos.

Para el modelo independiente del hemocultivo, se ha decidido utilizar *random forest* como algoritmo de aprendizaje automático, ya que con este algoritmo se han registrado modelos con una exactitud ligeramente superior al resto de algoritmos evaluados. Además, este algoritmo tiene un nivel de explicabilidad superior a SVM (algoritmo que presenta resultados semejantes pero requiere de una mayor cantidad de recursos y presenta un elevado tiempo de ejecución). El modelo resultante de este algoritmo entrenado sobre el fichero de datos mencionado previamente presenta una exactitud aproximada del 87 %.

Por otro lado, para el modelo dependiente de los resultados recogidos transcurridas las primeras 24 horas del hemocultivo, el algoritmo de aprendizaje automático escogido es *random forest*, coincidiendo con el algoritmo escogido para el modelo independiente del hemocultivo. El modelo obtenido tras el estudio presenta una exactitud que oscila en torno al 90 %.

Estos dos modelos permiten tener predicciones de "bacteriemia" o "no bacteriemia" sobre los pacientes. El modelo independiente del hemocultivo presenta una precisión cercana al 87 %, y trascurridas las primeras 24 horas del hemocultivo, el modelo dependiente proporciona una precisión del 90 % aproximadamente.

Capítulo 10

Conclusions

This chapter includes the general coclusions of the project discerning between two models. We decided to separate this project into two models after finding blood culture dependent data: a model that allows to make predictions without dependence on blood culture, and another one that makes predictions with data collected from blood culture after the first 24 hours.

After analyzing the missing data proccesing techniques, *complete case data* is discarded. This technique resulted in a low number of instances (it removes all patient data if it contains empty attributes) and it would be impossible to make predictions about incomplete instances. *Discarding instances and/or atributes* is discarded because it would also force to use complete instances. *Separate class method* and *missing value imputation* consider empty attributes and they take into account all the instances. According to the obtained results, *separate class method* has been chosen to process the missing data, as technique presents the best results in all analysis. All models trained with data treated with this technique show a higher accuracy rate than those obtained with previous techniques.

Then, the data were normalized to avoid the presence of biases depending on the magnitudes of each attributes, that is, to prevent the model from giving greater importance to an attribute beacuse of the nature of its magnitude. In the same way, tests were also done with *one hot encoder*, that allows the categorical attributes to be divided into several attributes, one for each subcategory. Despite generating a slight improvement in model's accuracy, it increases considerably the number of attributes and consequently, the number of dimensions of the model. This could generate underfitting in the model and increase its complexity. Trying to reduce the number of attributes, *principal component analysis* was applied but, due to its low explicability and the loss of the original attributes of the dataframe (this technique makes linear combinations with all the attributes to generate main components), both techniqeus were discarted.

In short, the data file containing the data treated with normalization and *separate class method* will be used.

For the blood culture independent model, we decided to use *random forest classifier* like machine learning algorithm, because with this algorithm models with sightly higher

accuracy than the other algorithms have been registered. Moreover, this algorithm has a higher explicability level than SVM (algorithm that presents similiar results but requires a high level of resources and a large execution time). The resulting model trained with this algorithm and the data chosen above shows an accuraccy around 87 %.

On the other side, for the blood culture dependent model, which requires data collected after the first 24 hours of blood culture, the chosen machine learning algorithm is *random forest* too. The model obtained after the study presents an accuracy close to 90 %.

This two models allow for making "bacteremia" and "no bacteremia" predictions in patients. The blood culture independent model allows for predictions arround 87 % accuracy and after ther first 24 hours of the blood culture, the second model obtains predictions close to 90 % accuracy.

Capítulo 11

Desarrollo futuro

Este proyecto ha permitido obtener dos modelos clasificadores con exactitudes considerables, pero debido a la falta de tiempo, quedan técnicas por estudiar. Una propuesta interesante para continuar el proyecto sería estudiar los modelos de imputación de datos perdidos utilizando algoritmos de aprendizaje automático. Debido al gran número de atributos presentes en el modelo, esta tarea requerirá un elevado tiempo de estudio y desarrollo. Para poder simplificar el proceso, sería interesante estudiar más en profundidad los atributos y los elementos de los atributos categóricos. Si se combinase este estudio con la técnica de *one hot encoder* (técnica que genera un atributo por cada cada elemento perteneciente a una categoría, sección 2.7) podrían optimizarse los datos y ver el impacto de cada elemento perteneciente a una categoría, es decir, una vez realizada la separación de las categorías en diferentes atributos, se podría analizar individualmente cada uno de ellos y ver su importancia. En resumen, aplicar la técnica de *one hot encoder*, realizar un estudio para reducir el número de atributos y aplicar métodos de imputación basados en algoritmos de aprendizaje máquina permitiría obtener un *dataframe* optimizado y podría obtenerse un modelo de mayor exactitud.

Por otro lado, podría estudiarse un modelo clasificador basado en redes neuronales artificiales (algoritmo de aprendizaje automático), a pesar de presentar una baja explicabilidad y una elevada complejidad, este método basado en su equivalente biológico podría detectar relaciones complejas entre los atributos y generar un modelo clasificador. Sería interesante trabajar con ambas propuestas y englobar los resultados con los obtenidos en este proyecto.

Además, gracias a los resultados obtenidos y realizándose una fase de pruebas, estos modelos podrían emplearse como apoyo en los hospitales para la detección de bacteriemias. En una fase inicial, evaluando los resultados y el comportamiento con los casos reales que puedan presentarse en un hospital y si los resultados son fructíferos, que sirva de apoyo para el diagnóstico de los pacientes.

Los modelos obtenidos en el proyecto permiten realizar una clasificación de "bacteriemia" o "no bacteriemia", pero si se dispusiese de un mayor número de casos de estudio, podría buscarse un modelo que realizase una clasificación diferenciando también, en los distintos tipos de bacteriemia. De esta manera, podría acelerarse el diagnóstico final del

hemocultivo así como su tratamiento.

Capítulo 12

Contribución de cada alumno

En este capítulo reflejamos las partes desarrolladas por cada alumno, desde un principio hemos querido participar en todas las partes del trabajo y por ello, decidimos trabajar de manera concurrente sobre cada técnica empleada. Para ello, hemos realizado estudios de manera separada y posteriormente, realizamos una puesta en común de los resultados. A parte, realizamos videollamadas y reuniones presenciales para poder trabajar de forma común sobre alguna de las técnicas más complejas. De este modo, a pesar de tener que realizar un esfuerzo mayor, nos asegurábamos conocer todas las técnicas empleadas y en caso de ser necesario, poder resolver cualquier conflicto sin dependencias. Además, hemos contando con la el apoyo y orientación de Óscar a lo largo de todo el proyecto, lo cual nos ha ayudado a ser constantes y organizados.

12.1. Diego Gómez Rodríguez

A raíz de la reunión inicial con los tutores y de fijar los objetivos del proyecto, el lenguaje y las herramientas que íbamos a emplear en su desarrollo, los primeros días estuve familiarizándome con las tecnologías que íbamos a emplear. A pesar de tener conocimientos sobre el lenguaje utilizado (*python*) y alguna de las técnicas empleadas, fue necesario complementar mis conocimientos con búsquedas de información , lecturas y cursos online y presenciales sobre *python* y la librería *sklearn*. Además, a medida que ha ido avanzando el proyecto he ido complementando todos los conocimientos. Lo mismo ha ocurrido con las bacteriemias y las técnicas empleadas en la detección de las mismas, en este caso, me he apoyado en las explicaciones de José Manuel y Óscar, y a medida que ha ido avanzando el proyecto, he profundizado más sobre ello.

En la fase de tratamiento de datos, tras una primera limpieza de los atributos con los tutores en la que escogimos que atributos eran importantes mantener y cuales no, tanto mi compañero como yo trabajamos con las técnicas mencionadas, en esta ocasión yo me centré en las técnicas de tratamiento de datos mientras mi compañero avanzada en el estudio de SVM. Durante esta fase, realizamos numerosas reuniones para dar a conocer los avances que íbamos haciendo y para explicarnos las distintas técnicas que se estaban

llevando a cabo. Empecé realizando un estudio sobre la cantidad de datos perdidos que había presentes en los datos que nos habían facilitado los tutores. Tras observar una cantidad considerable de datos perdidos recopile información sobre el tratamiento de este tipo de datos. En primer lugar apliqué *complete case data*, pero dado el bajo número de casos resultantes decidí continuar con otras técnicas. Las segundas pruebas fueron con un algoritmo que ordenaba los atributos del *dataframe* en función del número de datos perdidos que contenía e iba eliminando los atributos de uno en uno. Una vez terminaba la ejecución conocíamos la relación *numero de casos de estudio − atributos* que mayor cantidad de datos aportaba. En esta ocasión los resultados fueron más satisfactorios. Esta técnica permitía mantener un gran porcentaje de las instancias pero teníamos que renunciar a algunos atributos. En un principio, decidimos realizar este estudio por intuición pero gracias a un artículo que nos facilitó Óscar descubrimos que se denominaba *descartando atributos y/o casos de estudio*. Basándonos en el documento, descubrimos otra técnica bastante interesante, *Separate class method*. Esta técnica añadía una nueva clase a los atributos categóricos que representa los datos perdidos, y en el caso de atributos numéricos un valor que generase la separación necesaria entre los datos originales y el dato perdido. Finalmente, realicé un estudio sobre la imputación de valores perdidos. En esta ocasión, esta técnica ofrecía numerosos métodos de imputación (media, moda, mediana, y numerosos algoritmos de aprendizaje máquina que realizasen una predicción sobre estos valores). Debido al estado del proyecto, las fechas de entrega y los numerosos atributos que presentaba el proyecto, nos centramos en la media y la moda. Debido a la presencia de clases categóricas, la media daría un valor decimal que posteriormente el modelo detectaría como una nueva categoría y resultaría el mismo efecto que la técnica de *separate class method*. En esta ocasión hice pruebas con imputación mediante moda y por otro lado, desarrollé un algoritmo que realiza una imputación mixta, la moda para los atributos categóricos y la media para los atributos con valores numéricos.

En esta misma fase de tratamiento de datos, buscamos una manera de trabajar mejor con los atributos categóricos. Tras una búsqueda sobre qué técnicas eran las más indicadas para tratar este tipo de atributos, decidimos aplicar *one hot encoder*, técnica que separaba los atributos categóricos en diferentes atributos, es decir, separa un atributo de tipo categórico en tantos atributos como elementos distintos pueda clasificar. Como consecuencia, esta técnica tiene un gran impacto sobre el número de atributos totales del *dataframe*, y por consiguiente, un aumento de las dimensiones del modelo (una dimensión por atributo), generando un fuerte impacto sobre el tiempo de ejecución y la complejidad y aumentando las posibilidades de obtener un modelo con subajuste.

Durante este periodo, encontramos problemas con la explicabilidad del modelo SVM. Dada la importancia de la explicabilidad de los modelos en este tipo de proyectos, estuve investigando a cerca de como aportarla a nuestros modelos. Para ello me centre en buscar técnicas que nos sirviesen para todos los modelos, en primer lugar encontré la librería *shap*, esta librería permite estudiar la explicabilidad de numerosos tipos de algoritmos de aprendizaje automático, pero en particular me centre en *TreeExplainer*, dedicado al estudio de explicabilidad de modelos entrenados sobre arboles de decisión. Como resultado, obtiene una gráfica que representa los pesos sobre el modelo de los atributos más representativos. Debido a la utilización de esta librería en numerosos proyectos basados en RF, decidimos aplicarla en nuestro algoritmo RF y buscar alternativas para SVM y K-NN. Finalmente, empleamos *eli5* en SVM y K-NN, una librería que nos aportaba los

pesos de los atributos sobre el modelo, funcionalidad semejante a la empleada en RF. Además, basándome en las indicaciones de Óscar sobre la importancia de la curva ROC y el valor de AUC en los estudios médicos, implementé esta funcionalidad en RF. Una técnica muy interesante que permitía conocer la eficacia de nuestro modelo RF para todos los umbrales de clasificación.

En paralelo y por recomendación de Óscar, realicé un estudio sobre *análisis de componentes principales* (PCA). PCA permitía reducir el número de atributos sustituyéndolos por componentes principales (cada componente es resultado de una combinación lineal de los atributos del *dataframe*). A su vez, este método podría ser la solución a los aspectos negativos de *one hot encoder.*

Una vez terminados los estudios, hicimos una puesta en común y sacamos las conclusiones basándonos en los resultados recogidos. Tras el análisis y probar el comportamiento de estas técnicas con *support vector machine*, a pesar de no haber realizado pruebas exhaustivas, decidimos descartar PCA a causa de la sustitución de los atributos originales por las componentes principales. PCA presenta un bajo nivel de explicabilidad y, una vez aplicado sobre los datos de trabajo, transmite este problema al modelo. Además, descartamos *complete case data* debido al escaso número de pacientes completos porque no nos permitía entrenar un modelo clasificador.

Una vez terminado el análisis de las técnicas de tratamiento de datos de forma individual, realizamos una combinación de las diferentes técnicas con el fin de ver como interactuaban entre ellas. Todos los algoritmos desarrollados para aplicar estos métodos de tratamiento de datos generan un archivo CSV para poder realizar los estudios de los algoritmos de aprendizaje automático de una manera más sencilla y cómoda posteriormente.

Al terminar la fase de tratamiento de datos, nos centramos en el desarrollo de los algoritmos de aprendizaje automático y el ajuste de sus hiper-parámetros. Comencé el desarrollo de K-NN, para esta técnica hay que ajustar el parámetro k, el número de "vecinos" que toma como referencia para clasificar cada caso de estudio. Para ello el algoritmo entrena un modelo para cada $k \in [1 - 20]$, devolviendo el valor de k asociado al modelo de mayor precisión.

Posteriormente, desarrollamos RF. En esta ocasión, lo realizamos de forma conjunta, tanto el desarrollo como el ajuste de los hiper-parámetros. En este caso, ajustamos el número de estimadores (número de arboles de decisión que compondrán el "bosque" sobre el que trabaja RF) e incorporamos un parámetro (*random_state*) que ayuda a controlar la aleatoriedad del algoritmo al generar los distintos árboles de decisión. Al terminar, continuamos con el desarrollo de SVM y K-NN que habíamos iniciado de manera individual. Por recomendación de Óscar, introdujimos la librería *GridSearchCV*, que automatizaba el ajuste de hiper-parámetros. También, nos permitía conocer valores de precisión tanto en la fase de entrenamiento como en la fase de validación, y aplica el método de *k-fold cross validation*. Todo en conjunto, nos permite estudiar la presencia de sobreajuste o subjuste en los modelo.

Una vez finalizada la fase de tratamiento de datos y la fase de desarrollo, decidimos trabajar juntos en el desarrollo de un algoritmo que ejecutase SVM, RF y K-NN sobre todos los ficheros de datos generados en la fase de tratamiento. De este modo, hemos

podido ver los resultados de una manera global y nos ha servido de apoyo para extraer las conclusiones del proyecto.

Tras analizar los resultados globales y estudiar los atributos que más peso tienen sobre los modelos obtenidos en el análisis general, descubrimos que podría existir dependencia entre algunos atributos y el hemocultivo realizado para detectar la bacteriemia. Más tarde, José Manuel nos confirmó que algunos atributos habían sido recogidos transcurridas las primeras 24 horas del hemocultivo. En ese momento decidimos, entre nosotros y los directores, hacer dos modelos clasificadores, uno independiente a los resultados del hemocultivo y un segundo, que tuviese en cuenta los atributos dependientes de los primeros resultados del hemocultivo.

Para incluir los resultados de ambos modelos en la memoria, eliminé los atributos de los datos originales y realicé de nuevo el tratamiento de datos. Después, me reuní con mi compañero de nuevo para realizar el análisis de los nuevos datos en conjunto.

En cuanto a la memoria, fuimos rellenando los capítulos paralelamente a la fase de tratamiento de datos y la fase de codificación, pero una vez terminadas estas fases, nos separamos los diferentes capítulos para profundizar más, y posteriormente, repasamos y completamos los capítulos del otro compañero asegurándonos así.

Como comentábamos en un principio, hemos intentado estar presentes ambos compañeros en las diferentes fases del proyecto, esto nos ha permitido conocer el proyecto al completo y poder realizar modificaciones y correcciones de manera más ágil e independiente.

12.2. Víctor Ramos Fuentes

El proyecto se inició tras una reunión con Óscar Garnica donde se fijaron los objetivos del proyecto, el plan a llevar a cabo para elaborarlo y las técnicas de comunicación que se iban a utilizar durante el curso. Óscar nos hizo una introducción general a lo que debíamos desarrollar a lo largo del año y nos suministró unos archivos csv procedente del otro director, José Manuel Ruiz. Cabe recalcar que el lenguaje elegido para implementar el proyecto fue *python*, un lenguaje que ya había estudiado en la asignatura de *Aprendizaje automático y Big Data* cursado el año anterior en la propia facultad como optativa.

Tras revisar documentación disponible en buscadores sobre tratamiento de datos y limpieza de los mismos, se celebró otra reunión en el despacho de Óscar, en la cual se contó con la presencia de José Manuel, quien nos explicó de forma general el contexto de las bacteriemias en el organismo, nos habló de las técnicas actuales para detectarlo (estudios estadísticos), y de los 117 atributos que se nos facilitaban desde el hospital recogidos en el fichero csv suministrado en la reunión anterior con Óscar. Tras fijar un número estimado de atributos que debían ser eliminados del estudio, bien por no ser útiles, o bien porque eran atributos que no aportaban información. Para poder conocer más a fondo las bacteriemias y lo que suponían dentro del mundo de la medicina, tanto mi compañero como yo buscamos numerosa información acerca de ello, redactando consecuentemente la primera sección del primer capítulo de esta memoria donde se intenta introducir al lector en un contexto sobre este fenómeno.

Una vez teníamos disponibles los datos con los correspondientes atributos, procedí a estudiar detenidamente el dataset. Para ello, haciendo uso de un notebook, detecté un problema con la codificación de los datos, se trataba de un tipo de datos *latin-1*, cuando el estándar utilizado era *UTF-8*. Con un simple cambio de la codificación quedó solucionado. Una vez obtenidos los datos de forma clara y limpia, investigué acerca de las técnicas de sub-ajuste y sobre-ajuste.

Tras tener estudiados y procesados los datos, para poder preparar la técnica de *support vector machine*, estuve leyendo documentación variada como la propia web de sklearn. De cara a conocer en profundidad los distintos tipos de algoritmos de inteligencia artificial hice el curso de Udemy sobre Machine Learning. Elaborando el notebook de SVM, hice un estudio acerca de los múltiples hiper-parámetros del modelo. De igual forma, realicé varias búsquedas por la web acerca de las distintas funciones kernel del algoritmo, escogiendo la que parecía más adecuada para el desarrollo. Una vez implementado el modelo y conseguir los primeros resultados con el dataset limpio, intenté obtener interpretabilidad del algoritmo basándome en varias librerías pero no obtuve resultado alguno.

Al acabar de desarrollar el algoritmo de SVM, constaté junto a mi compañero los resultados que se habían obtenido con respecto a las técnicas de preprocesado de datos, *separate class method data* y *complete case data*, ambos concluímos que esta última técnica generaba muy pocos casos de uso, con lo cual fue descartada. Uno de los mayores problemas a los que nos hemos enfrentado ha sido el tratamiento de datos categóricos que disminuía la tasa de acierto de nuestro SVM, para solucionarlo aplicamos la técnica *one hot encoder*, donde separamos los atributos categóricos en distintos atributos, generando un mayor número de atributos aumentando la complejidad y el tiempo de ejecución al

procesar el nuevo conjunto de datos.

Después de generar el primer modelo, aún sin interpretabilidad, me puse a desarrollar con mi compañero otro notebook con el algoritmo de *random forest*, donde primero realicé un estudio del número de estimadores del modelo, una vez obtenido un número óptimo, desarrollé el modelo con la librería *sklearn* y conseguí implementar ciertos métodos de interpretabilidad como la matriz de confusión, un estudio detallado del accuracy del modelo, el uso de la librería eli5 y de *TreeExplainer* para la importancia de los atributos. Para poder constatar que el dataset tomaba los datos de entrenamiento y validación de forma aleatoria y no contribuía a la tasa de acierto del modelo, busqué información acerca de la técnica *k-fold cross validation*, la cual implementé en un notebook aparte. Mi compañero Diego mientras tanto estuvo empezando la técnica *KNN*.

Una vez teníamos desarrollados las tres técnicas de Inteligencia Artificial sobre las cuales giraba nuestro proyecto, Óscar nos recomendó el curso de Inteligencia Artificial que impartía Google en la propia facultad. Por supuesto, tanto mi compañero Diego como yo asistimos y extrajimos muchas ideas a implementar en nuestro trabajo para mejorarlo.

Una vez obtenidos los resultados de las tres técnicas, intente retomar la interpretabilidad de SVM, lo cual conseguí basándome en la librería eli5 que ya implementé en *random forest*.

En paralelo a ello, estuve ayudando a mi compañero con la técnica PCA, *análisis de componentes principales*, la cual podría solucionarnos el problema de la técnica *one hot encoder* mencionada anteriormente.

En una reunión con Óscar donde le comentamos los resultados obtenidos por las tres técnicas, nos sugirió que el estudio de los parámetros lo realizásemos con la función *GridSearchCV*, con lo cual rehicimos los tres notebooks de las técnicas generando unos nuevos que incluían dicha función.

Al ejecutar todos los ficheros con todas las técnicas, volcamos los resultados en un capítulo general para exponer las conclusiones de este estudio, hicimos una tabla donde se recogían todos y cada uno de los porcentajes explicativos de los algoritmos. Tras analizar todos los resultados obtenidos, ambos nos dimos cuenta de que ciertos atributos, los que más peso obtenían en la interpretabilidad de los modelos, guardaban relación con los días que llevaban en los hemocultivos. Algunos destacaban el primer día de hemocultivo y otros requerían de más tiempo para concluir un resultado. Por ello, ampliamos un nuevo capítulo la memoria, donde tras eliminar del conjunto de datos dichos atributos, procesábamos de nuevos todas las técnicas y algoritmos obteniendo nuevos resultados.

Para finalizar, destacar que tanto la memoria como el desarrollo del código se ha realizado de la forma más equitativa posible, ambos hemos estado involucrados en todos y cada uno de los pasos que se han dado, hemos asistido de forma conjunta a todas las reuniones con Óscar y a los cursillos de formación tanto presenciales como online durante la pandemia sobre la referenciación de imágenes y citas bibliográficas.

Índice de figuras

Índice de cuadros

Bibliografía

[1] Imagen bacteriemia. http://isanidad.com/wp-content/uploads/2020/06/bacterias-en-sangre.jpg.

[2] José Miguel Cisneros-Herreros, Javier Cobo-Reinoso, Miquel Pujol-Rojo, Jesús Rodríguez-Baño, and Miguel Salavert-Lletí. Guía para el diagnóstico y tratamiento del paciente con bacteriemia. guías de la sociedad española de enfermedades infecciosas y microbiología clínica (SEIMC). *Enfermedades Infecciosas y Microbiología Clínica*, 25(2):111–130, February 2007.

[3] Imagen frascos hemocultivo. http://aprendiendoenpracticas.blogspot.com/2012/12/hemocultivo.html.

[4] Imagen sistema automatizado. http://azadiagnostics.com/images/fecility/8.jpg.

[5] Ley orgánica 3/2018, de 5 de diciembre, de protección de datos personales y garantía de los derechos digitales. https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf.

[6] An investigation of missing data methods for classification trees applied to binary response data. http://people.stern.nyu.edu/jsimonof/jmlr10.pdf.

[7] Gustavo E. A. P. A. Batista and Maria Carolina Monard. An analysis of four missing data treatment methods for supervised learning. *Applied Artificial Intelligence*, 17(5-6):519–533, 2003.

[8] Imagen overfitting y underfitting. https://pramodsingla.com/category/artificial-intelligence/machine-learning/.

[9] ¿qué es overfitting y cómo evitarlo? https://empresas.blogthinkbig.com/que-es-overfitting-y-como-evitarlo-html-2.

[10] A gentle introduction to k-fold cross-validation. https://machinelearningmastery.com/k-fold-cross-validation/.

[11] Rpubs - análisis de componentes principales (pca). https://rpubs.com/Cristina_Gil/PCA.

[12] Petra Perner. *Advances in Data Mining. Applications and Theoretical Aspects: 9th Industrial Conference, ICDM 2009, Leipzig, Germany, July 20 - 22, 2009. Proceedings*, volume 5633. Springer-Verlag Berlin Heidelberg, 01 2009.

[13] Understanding pca (principal component analysis) with python. https://towardsdatascience.com/dive-into-pca-principal-component-analysis-with-python-43ded13ead21.

[14] Imagen svm. https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python.

[15] G. J. Briem, J. A. Benediktsson, and J. R. Sveinsson. Multiple classifiers applied to multisource remote sensing data. *IEEE Transactions on Geoscience and Remote Sensing*, 40(10):2291–2299, 2002.

[16] P. O. Gislason, J. A. Benediktsson, and J. R. Sveinsson. Random forest classification of multisource remote sensing and geographic data. In *IGARSS 2004. 2004 IEEE International Geoscience and Remote Sensing Symposium*, volume 2, pages 1049–1052 vol.2, 2004.

[17] Curva de roc y auc. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=es-419.

PASCAL
ENERO 2018
Ult. actualización 25 de junio de 2020
LaTeX lic. LPPL & powered by TEFLON CC-ZERO