UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Análisis de la robustez de los clasificadores dependiendo de la importancia de los atributos.

Trabajo Fin de Máster

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

AUTOR/A: Gálvez Aucejo, Carlos

Tutor/a: Ramírez Quintana, María José

CURSO ACADÉMICO: 2021/2022

Resumen

La profusión de aplicaciones basadas en el aprendizaje automático, en especial aquellas críticas para la seguridad como son los sistemas de conducción autónoma y los tratamientos médicos, ha despertado una preocupación natural entre los usuarios sobre su confiabilidad. En este entorno se hace necesario poder razonar sobre el comportamiento de los sistemas relacionado con la corrección, robustez, privacidad, eficiencia y equidad. En este TFM se propone analizar la robustez de los clasificadores en función de la importancia de los atributos. La idea es inyectar un porcentaje de ruido en los atributos más importantes y evaluar cómo esta perturbación afecta a las predicciones del clasificador. Para ello, se realiza un estudio experimental usando una colección de conjunto de datos y diversos clasificadores entrenados mediante las técnicas de aprendizaje más ampliamente conocidas.

Palabras clave: aprenentatge automàtic, classificadores, robustesa, soroll, evaluación, R

Resum

La profusió d'aplicacions basades en l'aprenentatge automàtic, en especial aquelles crítiques per a la seguretat com són els sistemes de conducció autònoma y els tractaments mèdics, ha despertat una preocupació natural entre els usuarios sobre la seua confiabilitat. En aquest entorn es fa necessaria poder raonar sobre el comportament dels sistemes relacionat amb la correcció, robustessa, privacitat, eficiència y equitat. En aquest TFM es proposa analitzar la robustessa dels classificadors en funció de la importància dels atributs. La idea es introduïr un porcentatge de soroll als atributs més importants y evaluar com esta perturbació afecta a les prediccions del classificador. Per a aconseguir això, es realitza un estudi experimental empleant una colecció de conjunt de dades y diversos classificadors entrenats mitjançant les tècniques d'aprenentatge més ampliament conegudes.

Paraules clau: aprendizaje automático, clasificadores, robustez, ruido, evaluación, R

Abstract

The profusion of applications based on machine learning, especially those critical for security, such as autonomous driving systems and medical treatments, has awakened a natural concern among users about its reliability. In this environment, it is necessary to be able to argue about the behaviour of the systems about correctness, robustness, privacy, efficiency and equity. In this master's dissertation, it is proposed to analyze the robustness of the classifiers depending on the attribute's importance. The idea is to inject a percentage of noise in the most important attributes and evaluate how this perturbation affects the classifier's predictions. For this, it is made an experimental study using a collection of datasets and different classifiers trained with the most known learning techniques.

Key words: machine learning, classifiers, robustness, noise, testing, R

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Históricamente, se puede hablar de que los inicios del aprendizaje automático fueron en 1943, cuando se presenta un artículo científico con un primer modelo matemático que imita el comportamiento de una neurona biológica [14]. En 1950 sucede uno de los hitos más importantes ya no solo en relación al aprendizaje automático, sino a nivel de computación: Alan Turing crea un test para determinar si un ordenador tiene inteligencia real en el que el computador trata de engañar al humano pensando que está hablando con otro humano [27]. Otro hito que marcó huella fue obra de Arthur Samuel como trabajador de IBM, que se encargó de crear el primer programa de ordenador que aprendía por sí mismo y jugaba a la damas [22]. En los años 50 se disponía de poca memoria pero esto desencadenó la invención de estrategias, que más tarde evolucionarían en algoritmos, como el algoritmo *minimax*. Se dice que fue el propio Arthur Samuel quien acuñó el término *machine learning*. Pocos años después Frank Rosenblatt inventó el perceptrón, uno de los primeros algoritmos de clasificación binaria, que causó revuelo en los medios de comunicación. Durante los años 60 también hubieron hitos importantes como la creación del algoritmo *nearest neighbour* y relacionados con el aprendizaje en juegos. En las decádas de los 70, los avances en la tecnología permitía una mayor memoria y con ello mejorar el rendimiento de los algoritmos. Además, en esta década se desarrolló la propagación hacia atrás, técnica utilizada actualmente en redes neurales profundas. A finales de los 70 y principios de los 80 hubo cierto cambio en la dirección que tomó la investigación del campo. En este periodo disminuyó la investigación sobre redes neuronales, lo que causó una división entre inteligencia artificial y aprendizaje automático. La década de los 90 fue un periodo glorioso en cuanto a número e importancia de algoritmos nuevos: algoritmos de *boosting* para reducir el sesgo (1990, Robert E. Schapire [23]) bosque aleatorio (1995, Tin Kam Ho [9]) y máquinas de vectores de soporte (Corinna Cortes, Vladimir Vapnik, 1995 [3]). En el año 1998 se lanzó la base de datos del MNIST, con un primer conjunto de datos con dígitos escritos a mano que sirvió para evaluar el reconocimiento de escritura a mano. A medida que nos acercamos a la actualidad encontramos más trabajos y avances sobre reconocimiento del habla o reconocimiento facial. En 2014, Facebook presentó DeepFace, un algoritmo que reconocía personas en fotografías con la misma precisión que un humano [24]. En la actualidad, el aprendizaje automático es responsable de muchos hitos en la tecnología. Sin embargo, hay ciertas preocupaciones en relación a ataques enfocados en la introducción de datos perturbados para redirigir el modelo hacia el comportamiento que desea el atacante.

1.1 Motivación

Actualmente, gran parte de la economía y sociedad se apoya con plena confianza en las decisiones de algoritmos de clasificación. Por ejemplo, una entidad bancaria necesita saber si es seguro conceder un crédito a cierta persona: el trabajador introduce todos los datos del cliente y del tipo de préstamo, como los ingresos mensuales, la edad o la cantidad de dinero que se solicita. Entonces, el ordenador le devuelve que es seguro conceder el crédito al cliente. Esto ocurre porque ha basado su respuesta en miles de ejemplos anteriores. Pero, ¿qué ocurre si estos datos con los que se ha entrenado el algoritmo se viesen perturbados? Puede que esta vez el ordenador al introducir los mismos datos que en el caso anterior se deniegue el crédito.

Casos como este pueden ocurrir en muchos ámbitos y sectores profesionales por lo que es crucial identificar y dar respuesta al ruido, ya que puede derivar en un decremento importante de la precisión de las predicciones. Una manera de hacerlo es utilizando algoritmos resistentes, robustos, al ruido. Cuanto más robusto sea un algoritmo menos le afectará el ruido que se haya podido introducir. El ruido puede aparecer en el dataset de manera inesperada o sistemática, muchas veces difícil de esperar. Los modelos con ruido se pueden clasificar de varias maneras:

a) **por origen**, aleatorio o adversario. Si el ruido se debe a una mala especificación del modelo o por un fallo puntual de un sensor se trata de ruido aleatorio. Sin embargo, si se trata de una obra de un individuo malicioso estamos ante ruido adversario.

b) **por objetivo**, hacia atributo o hacia la clase. El primer caso es cuando el ruido introducido va destinado al atributo más importante o a cualquier otro. El otro objetivo puede ser la etiqueta de clase del dataset.

El ruido puede deberse a una mala especificación del modelo, por accidente o por obra de un individuo malicioso que busque empeorar el rendimiento del algoritmo. Esta última situación está cogiendo cada vez más repercusión en el sector del aprendizaje automático, lo que ha dado lugar a una nueva rama de investigación llamada *adversarial learning*.

Es por ello que este trabajo busca estudiar el comportamiento de diferentes clasificadores y aportar al campo de investigación ciertas conclusiones que sirvan como base para futuras decisiones sobre qué algoritmo es más adecuado en presencia de ruido.

1.2 Objetivos

El objetivo principal del trabajo es estudiar experimentalmente el efecto de inyectar ruido a los atributos más importantes de cada conjunto de datos. A raíz de los resultados obtenidos, se extraen conclusiones sobre la robustez y el sobreajuste de los modelos entrenados. El objetivo final es proporcionar al lector una idea sobre la importancia que tiene el sobreajuste y la robustez respecto al ruido de una manera experimental.

1.2.1. Objetivos específicos

A continuación se exponen objetivos más concretos que se buscan alcanzar con el desarrollo de este trabajo final de máster:

- Conocer las características de los diferentes algoritmos de clasificación binaria.
- Limpiar y adecuar los conjuntos de datos para la generación de los modelos.

- Evaluar el impacto del ruido en las predicciones de los clasificadores cuando se inyecta en el conjunto de entrenamiento.
- Evaluar el impacto del ruido en las predicciones de los clasificadores cuando se inyecta en el conjunto de testing.
- Analizar en base a los resultados qué algoritmos sufren más de sobreajuste cuando se introduce ruido.
- Analizar en base a los resultados qué algoritmos son los más robustos al ruido.

1.3 Estructura de la memoria

La estructura de la memoria proporciona al lector un breve resumen ordenado de cada uno de los capítulos y secciones que se pueden encontrar en el trabajo.

El Capítulo 1, en el que se encuentra el lector, introduce el tema del proyecto junto con los aspectos que han motivado al autor a llevarlo a cabo. Además se presentan los objetivos que se pretenden alcanzar con el desarollo de este proyecto.

Es en el Capítulo 2 donde se exponen los conceptos más importantes sobre el proyecto y el Aprendizaje Automático. Este capítulo es necesario para que el lector puedan comprender todos los términos a los que se hace referencia a lo largo del documento. Se presenta el término de aprendizaje automático y el proceso seguido para crear un modelo preciso. Además, se presenta el concepto de importancia de los atributos y su papel en este trabajo. Finalmente, se presentan las tecnologías utilizadas para el desarrollo técnico del proyecto.

En el Capítulo 3 se muestra la configuración de los experimentos realizados. Aquí se presentan los conjuntos de datos seleccionados y la limpieza que se les aplica. Se explican los algoritmos clasificadores que se utilizan y, finalmente, se puede encontrar el diseño del experimento junto a los parámetros de este.

El Capítulo 4 está dedicado al primer experimento. En él se presentan el concepto de sobreajuste, se describe la implementación y se muestran los resultados.

El Capítulo 5 corresponde al segundo experimento. De nuevo, encontraremos desde el concepto de robustez asociado al experimento hasta los resultados.

En el Capítulo 6 se pueden encontrar las conclusiones del proyecto, donde se sintetizan los resultados más importantes de los experimentos. Además se aporta una sección de trabajo futuro.

Finalmente, existe un apartado con la bibliografía y una sección con el glosario.

CAPÍTULO 2
Conceptos relacionados

En este capítulo se exponen los conceptos más importantes que se utilizan durante todo el trabajo. El objetivo de ello es poner en contexto al lector y unificar el vocabulario empleado. Además, se ha elaborado un glosario donde se recogen más términos relacionados. Se explica el término de aprendizaje automático de manera detallada, el proceso que se sigue para conseguir un modelo matemático capaz de predecir con cierta precisión y se expone uno de los puntos en los que se centra el proyecto: hallar la importancia de los atributos del conjunto de datos. Además se comentan las tecnologías utilizadas para llevar a cabo la implementación del trabajo.

2.1 Aprendizaje automático

El aprendizaje automático o *Machine Learning* es una rama de la computación que estudia las técnicas que ayudan a aprender buscando patrones recurrentes para intentar predecir, clasificar o describir la información contenida en los datos que se le proporcionan.

Aunque en todas las tareas se aprenda sobre los datos, existen dos tipos de algoritmos o *learners* dependiendo del tiempo y la cantidad de supervisión que se necesita en el entrenamiento: *supervised learning* y *unsupervised learning*.

2.1.1. Aprendizaje supervisado

El primer grupo, aprendizaje supervisado, genera modelos predictivos, es decir, una herramienta matemática que proporcione una predicción precisa [11]. Esto se consigue especificando una variable objetivo y la variable predictora, o un conjunto de variables independientes, del *dataset*. El algoritmo en un modelo predictivo trata de encontrar relaciones entre la variable objetivo y los atributos.

Dentro de este tipo de aprendizaje existen dos grupos de problemas: de clasificación y de regresión. A continuación se explican en qué consisten ambos problemas aportan ejemplos ilustradores.

Clasificación

Los problemas de clasificación, sobre los que se trabaja en este proyecto, consisten en predecir una variable categórica. La respuesta puede ser binaria o multinomial cuando existen más de dos posible valores. En ocasiones, no solo se desea conocer la clase a la que pertenece un nuevo registro sino la probabilidad de que esto sea así. Es decir, el problema

sigue siendo de predicción numérica pero el algoritmo decide quedarse con la clase que tenga mayor probabilidad. Algunos ejemplos de problemas de clasificación son:

- ¿Es este mensaje correo no deseado? (Sí/No)
- ¿Qué clase de animal es este? (Ave/Reptil/Pez)
- ¿Es seguro conceder a este cliente un crédito? (Sí/No)

Regresión

Cuando el problema busca predecir un resultado numérico hablamos de problemas de regresión. De nuevo, en base a las variables predictoras se intenta predecir un valor dentro de un rango contínuo. Un ejemplo sería calcular el coste de una vivienda en base a las medidas y el año de construcción.

2.1.2. Aprendizaje no supervisado

El objetivo es entender mejor y describir los datos trabajando sin una variable objetivo. Esencialmente, consiste en buscar agrupaciones entre los datos a través de herramientas estadísticas. Un ejemplo de tarea de aprendizaje no supervisado es el *clustering* o agrupamiento , que consiste en segmentar las observaciones en grupos similares en base a sus atributos. Un ejemplo de agrupamiento puede ser segmentar en grupos a los clientes habituales de un supermercado en base a sus compras. Otra tarea típica es la reducción de dimensionalidad con el fin de conseguir un conjunto de variables más pequeño pero sin correlación.

Análisis exploratorio de datos

El aprendizaje no supervisado en muchas ocasiones forma parte del análisis exploratorio de datos (EDA) impulsado por John W. Tukey [26]. Este tipo de análisis es un enfoque en el que se anima al investigador a tomar una actitud activa hacia el análisis de los datos con el fin de presentar hipotésis nuevas. Además, el autor acompañaba lo anterior con nuevas e innovadoras herramientas conceptuales e instrumentales respecto a la estadística descriptiva hasta el momento.

2.2 Proceso de modelado

El presente apartado está destinado a la descripción de cada uno de los pasos requeridos para lograr un modelo de machine learning que se ajuste a las necesidades del proyecto. Algunos de los puntos clave son la distribución estratégica de los datos en entrenamiento y pruebas, creación del modelo o la elección de un método de remuestreo.

2.2.1. Separación de los datos

El proceso de separación de los datos, o *data splitting*, es crucial para conseguir un buen modelo. El principal objetivo del proceso de aprendizaje automático es encontrar un algoritmo que no solo se ajuste a los datos que se le proporcionen, sino que también pueda predecir con alta precisión un resultado futuro. Esta característica es la generalización de un algoritmo y nos ayuda a saber cómo de bien se adecúa el algoritmo a datos no vistos anteriormente.

Para conseguir esto, el primer paso es separar todos los datos en dos subconjuntos, el conjunto de entrenamiento y el de pruebas:

El conjunto de entrenamiento son los datos que se van a usar para entrenar los modelos, ajustar parámetros, comparar modelos entre otras actividades.

El conjunto de pruebas existe para, una vez escogido el modelo final, evaluar el rendimiento del modelo de manera imparcial.

Es recomendable separar los datos entre estos dos grupos manteniendo una relación entre 60 % - 40 % y 80 % - 20 %, conjunto de entrenamiento y de pruebas, respectivamente. Esto es debido a que si se emplean más del 80 % de las instancias en el entrenamiento, no quedarán suficientes muestras para realizar una buena evaluación del modelo. Además, es posible que se ajuste tanto a los datos de entrenamiento que el modelo no sea capaz de generalizar correctamente, lo que se conoce como sobreajuste. En el lado opuesto, si los datos de entrenamiento son menos de un 60 % puede que no sean suficientes para conseguir un modelo con un buen rendimiento, ocasionando subajuste.

Existen varios métodos para separar los datos, pero normalmente se opta por utilizar el muestreo aleatorio simple o el muestreo estratificado.

El muestreo aleatorio simple es la manera más sencilla de dividir los datos en subconjuntos de entrenamiento y pruebas. No se hace ningún control sobre los atributos.

El muestreo estratificado tiene el mismo objetivo pero sí que mantiene un cierto control sobre las distribuciones de los valores de la variable objetivo. Este muestreo se suele utilizar cuando la variable de respuesta está, o corre peligro de que esté, **altamente desbalanceada**.

El problema de tener datos desbalanceados es que puede afectar a las predicciones y al rendimiento del modelo, y más si se trata de tareas de clasificación donde la clase minoritaria es inferior al 5 %. Existe una medida que ayuda a relajar este problema y se trata de aumentar o disminuir el número del instancias a usar.

El *up-sampling* se emplea cuando la cantidad de datos es insuficiente y se intentan balancear incrementando el número de muestras de la clase minoritaria. La generación de nuevas instancias es usando repetición o *bootstrapping*.

El *down-sampling* , en cambio, reduce el número de registros cuyas clases son abundantes para equilibrar las frecuencias con respecto a las minoritarias.

2.2.2. Métodos de remuestreo

Tras haber separado los datos en subconjuntos de entrenamiento y pruebas, queda abierta una duda: ¿Cómo evaluamos el rendimiento de la capacidad de generalización del modelo sin utilizar el subconjunto de muestras para pruebas? Evaluar con una métrica de error sobre los datos de entrenamiento podría darnos una visión sesgada porque funcionan muy bien en los datos de este subconjunto pero no generaliza bien con los de pruebas

La opción es usar un enfoque que incluya validación, es decir, dividir el conjunto de entrenamiento en dos subconjuntos: el de entrenamiento y el de validación. Sin embargo, a menos que el *dataset* inicial sea muy grande, una sola partición de validación puede ser poco fiable. Para resolver este problema surge la validación cruzada.

Validación cruzada

La validación cruzada es una método estadístico para evaluar y comparar algoritmos de aprendizaje diviendo los datos en dos subconjuntos: entrenamiento y validación. Ambos subconjuntos se deben cruzar en varias iteraciones de manera que cada dato tenga la posibilidad de validarse. La técnica más común es validación cruzada de k pliegues, *k-fold cross-validation*, que consiste en dividir los datos de forma aleatoria en k grupos, pliegues, de aproximadamente el mismo tamaño. En la Figura 2.1 se puede observar el proceso: si el número de pliegues es k, se utilizarán $k-1$ para entrenamiento y un pliegue para pruebas; esto repetido k iteraciones y cada vez con un pliegue distinto siendo el de validación. Finalmente, se calcula la media de los errores, lo que nos proporciona una aproximación del error que se espera en datos no vistos anteriormente.

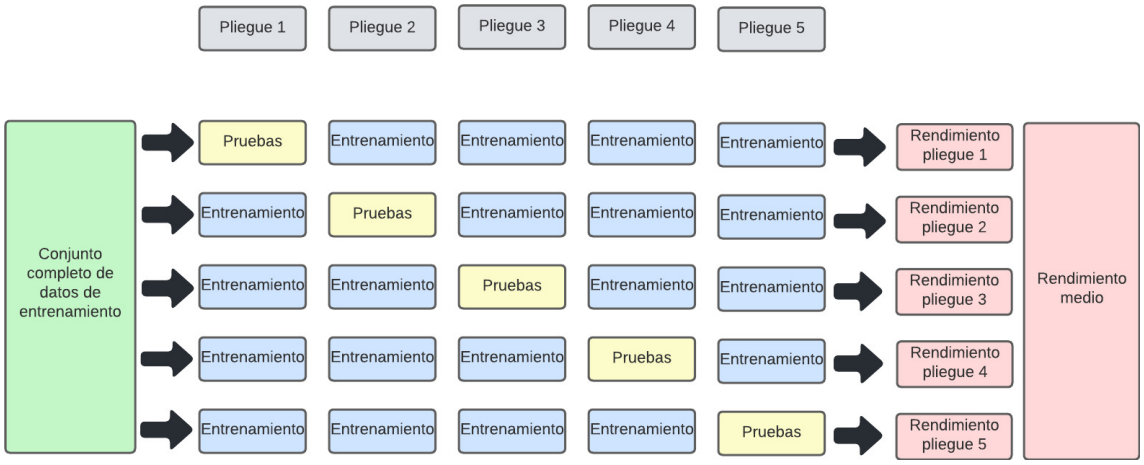

Figura 2.1: Proceso de la validación cruzada con 5 pliegues

La elección de la k es importante. Normalmente, en la validación cruzada se usa $k = 5$ o $k = 10$, ya que valores más altos pueden provocar costes computaciones demasiado elevados. De acuerdo con [15], se considera que $k = 10$ es una buena opción en muchas ocasiones y tiene un rendimiento similar a emplear *leave-one-out cross-validation*

Bootstraping

Se trata de otra técnica de remuestreo aleatorio pero con reemplazo. Es decir, tras seleccionar un dato para incluirlo en el subconjunto este sigue quedando disponible para volver a elegirlo. Una muestra de *bootstrap* es del mismo tamaño que el *dataset* original desde el que se construye.

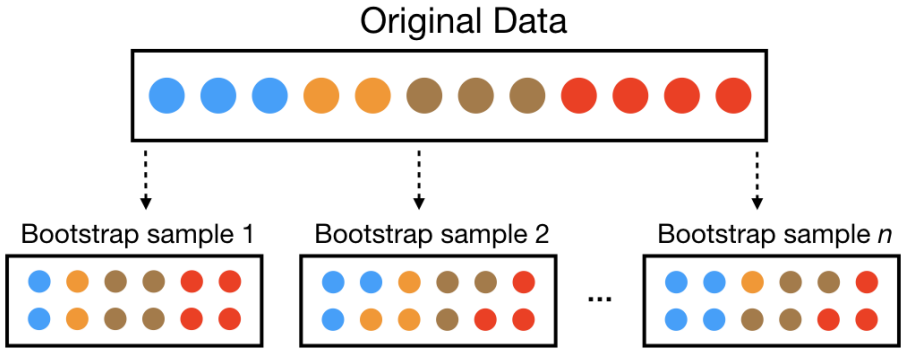

Figura 2.2: Proceso de bootstrapping. Imagen extraída de [2].

2.2.3. Errores de predicción

Los posibles errores de una predicción se pueden descomponer en el sesgo existente y la varianza. Si el modelo es demasiado sencillo, tiene pocos parámetros, puede tener un alto sesgo y poca varianza, provocando subajuste. Por otro lado, si el modelo tiene un gran número de parámetros, es muy complejo, tendrá una alta varianza y poco sesgo, dando lugar a un modelo *overfitted*. Lo ideal sería un sesgo bajo y una varianza mínima, sin embargo, hay que buscar un término medio para construir un buen modelo.

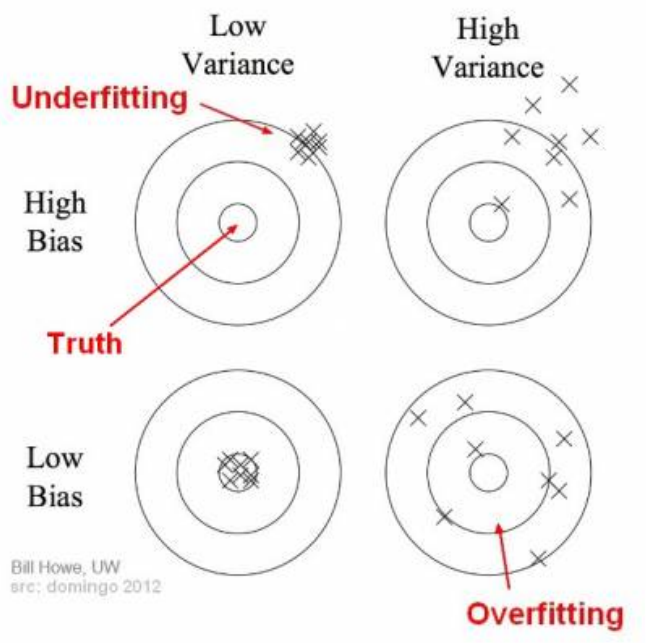

Figura 2.3: Tradeoff entre el sesgo y la varianza. Imagen extraída de [18].

Sesgo

Es la diferencia entre la respuesta esperada del modelo y el valor correcto que se intenta predecir. Mide cómo de lejos están las predicciones del modelo de los valores correctos, lo que proporciona una idea de cómo se ajusta el modelo a los datos. Además, es importante saber que modelos con un sesgo alto no se ven muy afectado por el ruido introducido en el remuestreo.

Varianza

La varianza es la variabilidad en los resultados de un modelo para una instancia del conjunto de datos. Algunos modelos son muy flexibles y adaptables a ciertos patrones, sin embargo, esto puede causar un problema de sobreajuste. Este problema puede aliviarse con métodos de remuestreo y configurando hiperparámetros, tema que no abarcaremos en este proyecto.

2.2.4. Evaluación del modelo

Actualmente, es común evaluar la precisión en las predicciones de un modelo y su rendimiento mediante funciones de pérdida, *loss functions*. Estas funciones son métricas

que comparan el valor real o conocido con el que predice el modelo. El objetivo con estas funciones, también conocidas como funciones de error, es minimizarlas, lo que significa un mejor modelo. Existen diferentes tipos de funciones de pérdida dependiendo del si el problema es de regresión o clasificación. Debido a que este trabajo está relacionado con problemas de clasificación se ha considerado adecuado centrarse en las métricas de este tipo.

Modelos de clasificación

Para este apartado es conveniente presentar primero la **matriz de confusión**: se trata de una tabla de dimensiones NxN, siendo N el número de clases, que recoge los aciertos y fallos del modelo de clasificación. Uno de los ejes de la matriz de confusión es la etiqueta que predice el modelo mientras que el otro eje indica la respuesta correcta.

En la Figura 2.4 se puede observar ver una matriz de confusión con N=2 para un modelo de clasificación binaria. La celda *Verdaderos Positivos* representa al conjunto de casos en los que el modelo predice correctamente la clase positiva. De esta manera, un verdadero negativo ocurre cuando el modelo predice correctamente la clase negativa.

Por otro lado, los falsos positivos son aquellos casos en los que el modelo predice incorrectamente la clase positiva. Es decir, que el valor real es negativo y el modelo devuelve positivo. Y por último, se da un falso negativo cuando el modelo devuelve una predicción incorrecta de la clase negativa.

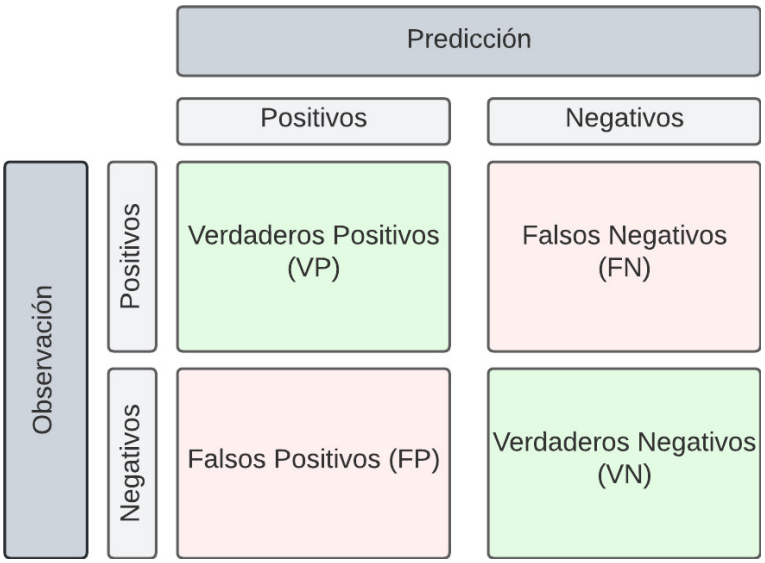

Figura 2.4: Tabla de confusión

Accuracy

La precisión, o *accuracy*, es una métrica para evaluar modelos de clasificación e indica el número de predicciones que el modelo acierta. En la Ecuación 2.1 se puede observar la definición formal. La precisión es muy utilizada en problemas de clasificación binaria o multiclase, sin embargo cuando el conjunto de datos tiene la clase desequilibrada.

$$Accuracy = \frac{Número\ de\ aciertos}{Número\ total\ de\ predicciones} = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.1}$$

Kappa

La puntuación Kappa es una métrica que compara una precisión observada con una precisión esperada. La estadística kappa sirve para evaluar un solo clasificador y para evaluar varios entre ellos. Además, tiene en cuenta valores de probabilidad aleatoria, lo que generalmente significa menos resultados engañosos que simplemente usando la precisión como métrica. Para el cálculo es necesario conocer primero la precisión observada, *observed accuracy* y la precisión esperada, *expected accuracy*.

La primera de ellas es el número de instancias que fueron clasificadas correctamente en toda la matriz de confusión. Básicamente, es la precisión tal y como se ha descrito en la Subsubsección 2.2.4.

La precisión esperada se define como la precisión que se espera que un clasificador aleatorio alcance en base a la matriz de confusión. Está relacionada con el número de instancias de cada clase y el número de instancias que el clasificador coincide con la realidad. Para calcular la precisión esperada es necesario saber que la frecuencia marginal para una clase por un determinado *rater* o evaluador es la suma de todas las instancias que ese evaluador indicó que eran de esa clase. En este caso solo puede haber dos *rater*: la observación o la predicción del modelo. Dicho esto, la precisión esperada se calcula tal y como se muestra en la Ecuación 2.2.

$$expectedAccuracy = \frac{(mFreq_{A,X} * mFreq_{A,Y})/30 \; + \; (mFreq_{B,X} * mFreq_{B,Y})/30}{30} \tag{2.2}$$

siendo $mFreq_{A,X}$ la frecuencia marginal de la clase A por el *rater* X.

Finalmente la puntuación kappa podrá calcularse siguiendo la Ecuación 2.3.

$$Kappa = \frac{observedAccuracy - expectedAccuracy}{1 - expectedAccuracy} \tag{2.3}$$

El paquete de R *caret* que se utiliza para la evaluación de los modelos devuelve un coeficiente Kappa que tiene como rango [-1, 1]. En la Figura 2.5 se pueden consultar tres escalas del grado de acuerdo entre los diferentes evaluadores en base al valor de Kappa obtenido a través de la matriz de confusión. Los valores, como se ha comentado, se pueden extender hasta -1, sin embargo las escalas se suelen centrar en los valores positivos.

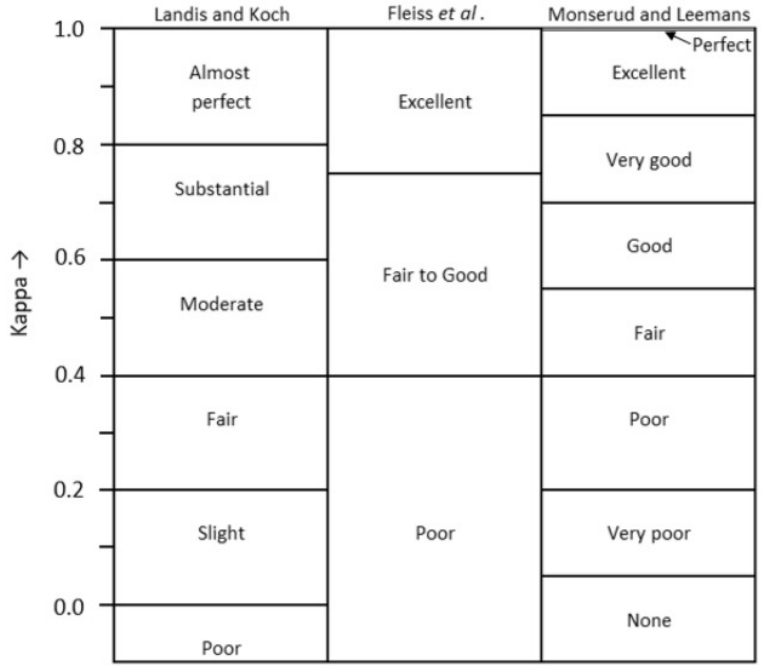

Figura 2.5: Tabla de interpretación del coeficiente Kappa según diferentes autores. Las escalas pertenecen a Landis and Koch (1977) [12], Fleiss et al. (2013) [8] y Monserud and Leemans (1992) [17].

2.3 Importancia del atributo

Uno de los puntos clave para la consecución del objetivo del trabajo es analizar la importancia que tienen cada uno de los atributos de un conjunto de datos con respecto al resto. Dado un modelo, calculado a partir de una técnica determinada, es posible extraer la importancia que tiene un atributo para ese modelo mediante permutaciones. La puntuación de la importancia se calcula haciendo permutaciones entre los valores de una columna, o atributo, del *dataset* y observando cómo afecta esto al error. Si el error aumenta significativamente cuando alteramos los valores, se entiende que ese atributo es importante para el modelo. Además este enfoque es totalmente independiente del modelo; se trata de un algoritmo genérico que sirve sea cual sea la técnica a través de la cual se ha extraído el modelo.

La importancia del atributo es una herramienta muy útil en el campo del aprendizaje automático. En primer lugar, permite entender los datos más fácilmente puesto que muestra cómo de fuerte son las relaciones entre los atributos y la variable objetivo, en nuestro caso, la clase y por lo tanto conocer cuáles son irrelevantes. Por otro lado, esta técnica permite mejorar el modelo eliminando los atributos con menos importancia y manteniendo los importantes. Así se consigue un modelo más sencillo y con mayor rendimiento.

Para conseguir la importancia de cada uno de los atributos de los datasets se ha utilizado el paquete de R *iml*. En la Subsubsección 2.4.1 se explican qué son los paquetes del lenguaje R. En primer lugar es necesario crear un objeto *Predictor* que será el contenedor que guarde tanto el modelo como los datos. El código asociado está descrito en el Listing 2.1.

```
1 predictor = Predictor$new(fit)
```

Listing 2.1: Creación del contenedor con la función *Predictor*

En general, es suficiente pasándole como argumento el modelo que se ha generado con anterioridad, en este caso *fit* objeto creado a través de la función *train* de *caret*. Sin embargo, hay ocasiones en las que no es capaz de reconocer cuáles son las variables predictoras y la variables de respuesta por lo que hay que indicárselo explícitamente, tal y como se muestra en el Listing 2.2.

```
X = df[-which(names(df)=="class")]
y = df$class
predictor = Predictor$new(fit, data = X, y = y, type = "prob")
```

Listing 2.2: Código alternativo para la creación del contenedor con la función *Predictor*

Una vez creado el contenedor *predictor*, la función *FeatureImp* realiza el cálculo de importancia de los atributos de la manera en la que se ha descrito al comienzo de esta sección. Esta función recibe como argumentos el objeto *predictor* anteriormente creado y una función de pérdida, que al tratarse de un problema de clasificación, corresponde indicar *ce*. En el Listing 2.3 se puede observar cómo se traduce a código R.

```
featImp = FeatureImp$new(predictor, loss = "ce")
```

Listing 2.3: Cálculo de la importancia de los atributos

Finalmente, del nuevo objeto creado *featImp* se consultan los resultados y obtenemos el atributo con mayor importancia. En el Listing 2.4 se puede ver cómo se ha conseguido.

```
mostImpAttr = featImp$results[which.max(featImp$results$importance),'feature']
```

Listing 2.4: Extracción del atributo más importante

2.4 Tecnologías utilizadas

2.4.1. Lenguaje R

R es un entorno de desarrollo software gratis para estadística y gráficos de código abierto [20]. La primera implementación fue a principios de 1990 por dos miembros de la Universidad de Auckland. En 1995 se estableció como proyecto de código abierto, dirigido a partir de 1997 por R Core Group, y en febrero del año 2000 se lanzó la primera versión de R. El lenguaje se ha seguido utilizando y mejorando introduciéndole nuevas funcionalidades y herramientas hasta el punto en el que actualmente existen más de 10000 librerías creadas por la comunidad que se pueden consultar en CRAN [25].

Figura 2.6: Logo del lenguaje R

Paquetes de R

Los paquetes, o librerías, de R son colecciones de funciones de R, datos y código compilado. R ya trae consigo varios paquetes, sin embargo hay muchos nuevos paquetes que permiten extender la funcionalidad del lenguaje. Existen paquetes de todo tipo: facilitar el manejo de fechas, análizar datos genómicos, *web scraping* y muchos más. A continuación, se comentan los paquetes escogidos para llevar a cabo las diferentes tareas relacionadas con el proyecto.

caret *Classification and Regression Training* [13] ofrece diversas funciones relacionadas con el entrenamiento y el dibujo de modelos de clasificación y regresión en gráficas.

tidyverse Se trata de un conjunto de paquetes compatibles entre sí gracias a su diseño de la API y a que comparten representaciones de la información [28]. *Core Tidyverse* incluye paquetes útiles para diferentes aspectos de análisis de datos como *ggplot2*, relacionado con la creación de gráficas, *dplyr*, para enriquecer la gramática de manipulación de datos, o *readr*, para facilitar la lectura de ficheros CSV.

iml *Interpretable Machine Learning* [16] es un paquete que contiene métodos de interpretabilidad para analizar el comportamiento y las predicciones de un modelo de aprendizaje automático. La función principal de este paquete en el proyecto es para obtener la importancia de los atributos.

citation Se trata de un paquete que facilita la citación de los paquetes de R obteniendo la información del repositorio CRAN y la muestra en formato Bibtex [5].

xtable Paquete que permite exportar los dataframes de R a formato de tabla de LaTeX [4].

ggpubr Se trata de un paquete ideal para la transformación de los *plots* generados por el paquete *ggplot2* (incluído en *tidyverse*) a otros preparados para publicaciones o artículos [10].

2.4.2. RStudio

RStudio es un IDE libre que trabaja con el lenguaje R y está orientado a la computación estadística y construcción de gráficas. Incluye un gran abanico de funcionalidades y funciona en la mayoría de plataformas. Cuenta con consola, editor de texto con resaltado de sintaxis, herramientas para el trazado, la depuración y la gestión del espacio de trabajo. El IDE está escrito en varios lenguajes como Java, C++ y JavaScript, mientras que su interfaz gráfica se ha desarrollado con el *framework* Qt. En 2009, J.J. Allaire fundó la organización RStudio y se lanzaron algunos productos gratis y de código abierto pero hasta febrero de 2011 no se lanzó la primera versión beta de forma pública del IDE RStudio. La versión 1.0 fue lanzada en noviembre de 2016 y actualmente, la organización, mantiene y promociona algunos de los paquetes de R más utilizados como Tidyverse, Shiny o TensorFlow.

Figura 2.7: Logo de RStudio

CAPÍTULO 3

Configuración del experimento

A continuación se comentan todos los componentes necesarios para llevar a cabo los experimentos. En la Sección 3.1 se presentan los conjuntos de datos utilizados para el experimento y a continuación se muestra la limpieza por la que han tenido que pasar, en la Sección 3.2. En la Sección 3.3 se muestran los distintos métodos que se han escogido junto a una breve descripción de cada uno de ellos. Y finalmente, en la Sección 3.4 se explican los distintos factores que componen el experimento.

3.1 Conjuntos de datos

Se han seleccionado un total de 12 *datasets* que sirven como base para los experimentos de este trabajo. Los conjuntos de datos se han extraído del portal OpenML y la mayoría, del repositorio UCI Machine Learning [6]. Las descripciones que se aportan de cada uno de los datasets también han sido extraídas de estas páginas.

En la Tabla 3.1 se puede ver una tabla resumen con información de los conjuntos de datos recién extraídos y leídos en R, es decir, sin haber realizado ninguna modificación. En esta tabla se indica, para cada conjunto de datos, su nombre, el número de filas, el número de columnas, el número de atributos de tipo *character*, el número de atributos de tipo *numeric* y el número de atributos de tipo *logical*.

Nombre	ID	Filas	Columnas			
			Total	chr	dbl	lgl
Mushroom	mushroom	8124	23	22	0	1
Breast Cancer Wisconsin	wdbc	569	32	1	31	0
Ionosphere	ionosphere	351	35	1	34	0
Scene	scene	2407	300	0	300	0
Cleveland Heart Disease	heart	303	14	2	12	0
Banknote	banknote	1372	5	0	5	0
Tic-Tac-Toe	tictac	958	10	10	0	0
Haberman	haberman	306	4	0	4	0
Bank Marketing	bankmkt	45211	17	10	7	0
Spambase	spambase	4601	58	0	58	0
Blood Transfusion	transfusion	748	5	0	5	0
Diabetes	diabetes	768	9	0	9	0

Tabla 3.1: Tabla resumen de los *datasets* escogidos previo a la limpieza

El dataset *Mushroom* describe muestras de 23 especies de setas de las familias Agaricus y Lepiota. Cada muestra se identifica como comestible, o no comestible, siendo este último grupo compuesto por venenosas y de estado desconocido o no recomendable. Extraído del repositorio UCI Machine Learning.

El dataset *Breast Cancer Wisconsin (Diagnostic)* contiene información del núcleo de la célula presente en la imágenes de pechos. La clase es el diagnóstico que se realiza y puede tener dos valores: maligno o benigno. Extraído del repositorio UCI Machine Learning.

Ionosphere recoge datos de un radar para averiguar si ciertos electrones atraviesan la ionosfera, clase *Good*, o las señales no consiguen atravesar la capa, clase *Bad*. Extraído del repositorio UCI Machine Learning.

El dataset *Scene* contiene información sobre un estudio que detecta árboles enfermos a través de imagenes satelitales. La clase puede ser un árbol enfermo, w, o cualquier otro elemento, n. Extraído de OpenML.

El dataset *Cleveland Heart Disease* tiene como atributo objetivo la presencia o ausencia de una cardiopatía. El valor 0 indica ausencia y del 1 al 4 denota presencia. Extraído del repositorio UCI Machine Learning.

Otro de los conjuntos de datos utilizados es *Banknote Authentication*. La información procede de imágenes de un tamaño de 400x400 de billetes falsos y verdaderos. El objetivo es determinar, a través de los atributos como la varianza o la falta de simetría, la legitimidad del billete. Extraído del repositorio UCI Machine Learning.

El conjunto de datos llamado *Tic-Tac-Toe Endgame* recoge todas las posibles configuraciones distintas del juego *tic-tac-toe*. El objetivo es saber si la partida finalizada es una victoria en base a la posición de las fichas. Extraído del repositorio UCI Machine Learning.

El conjunto de datos *Haberman's Survival* contiene casos de un estudio sobre la supervivencia de pacientes que han pasado por alguna cirugía de cáncer de mama. En base a la edad del paciente al momento de la operación y otros atributos, se desea saber si el paciente sobrevivió pasados 5 años desde la operación. Extraído del repositorio UCI Machine Learning.

El conjunto de datos *Bank Marketing* está relacionado con campañas de marketing a través de llamadas telefónicas de un banco portugués. Algunas de las variables de entrada son datos como la edad del cliente, su trabajo, su nivel educativo o la duración de la llamada. El objetivo es predecir si el cliente se suscribirá a un depósito a plazo fijo. Extraído del repositorio UCI Machine Learning.

Spambase es un dataset contiene una colección de emails que individuos han clasificado como correo no deseado o correo deseado. Algunas de las variables de entrada son la frecuencia de ciertas palabras, el uso de mayúsculas o la longitud de las palabras en mayúscula. Extraído del repositorio UCI Machine Learning.

El conjunto de datos llamado *Blood Transfusion Service Center* contiene datos de un centro de transfusión de sangre en Taiwán. Muestra datos como la frecuencia con la que dona una persona, la cantidad o la última vez que donó. El objetivo es saber si la persona donaría en marzo de 2007. Extraído del repositorio UCI Machine Learning.

El dataset *Diabetes* tiene como objetivo predecir si un paciente tiene diabetes o no, basándose en ciertas medidas diagnosticadas que se en incluyen en el conjunto de datos. Todos los pacientes son mujeres mayores de 21 años y algunas variables predictoras son su edad, el número de embarazos que ha tenido o el nivel de insulina. Extraído de OpenML.

3.2 Limpieza de datos

Esta sección está dedicada al proceso de limpieza por el que pasan los datasets antes de ser utilizados por los algoritmos de clasificación. Se explica qué es la limpieza de datos, por qué es necesaria y la implementación en R que se ha llevado a cabo.

3.2.1. Exploración inicial

Es crucial familiarizarse con los conjuntos de datos escogidos para poder realizar correctamente la limpieza de los datos. Pese a que algunos metadatos del dataset aparezcan en los portales de los que se han obtenido, es necesaria más información.

Inicialmente, utilizar la función `str(df)` donde *df* es el *dataframe* una vez leído con la función `read_csv(filename)`.

```
> str(df)
tibble [8,124 × 22] (S3: tbl_df/tbl/data.frame)
 $ class               : Factor w/ 2 levels "e","p": 2 1 1 2 1 1 1 1 2 1 ...
 $ capshape            : chr [1:8124] "x" "x" "b" "x" ...
 $ capsurface          : chr [1:8124] "s" "s" "s" "y" ...
 $ capcolor            : chr [1:8124] "n" "y" "w" "w" ...
 $ bruises             : num [1:8124] 1 1 1 1 0 1 1 1 1 1 ...
 $ odor                : chr [1:8124] "p" "a" "l" "p" ...
```

Figura 3.1: Salida de la función str(df).

Como se aprecia en la Figura 3.1, aporta información valiosa como el número de filas y columnas, un listado con los nombres de las diferentes columnas junto a sus tipos (chr, num, factor...) e incluso los primeros valores de cada columna. Desde este momento se puede detectar ya un aspecto muy importante: que los valores de las columnas no se correspondan con el tipo que deberían tener. Puede que la columna se titule Edad y su tipo sea *chr*, similar al tipo *string* de otros lenguajes, y sus valores sean enteros. Esto es un problema muy común en la lectura de archivos que perjudica la calidad de los datos.

3.2.2. Limpieza de datos

La limpieza de datos es el proceso de identificar, corregir o borrar datos imprecisos que pueda contener un conjunto de datos. Es esencial este paso con el fin de proveer al algoritmo un conjunto de datos de calidad para optimizar su rendimiento y, en ocasiones, para que simplemente funcione. No se puede confiar en que los datos que se han obtenido de una fuente externa estén preparados para el objetivo que le queremos dar en este proyecto. A continuación se mencionan los errores más importantes que se han identificado y resuelto para un funcionamiento correcto de los algoritmos de clasificación.

Aprovechando lo comentado en la anterior sección, se deben cambiar los tipos de las columnas en caso de ser necesario. La columna X56 del dataset *Spambase* inicialmente se detectó de tipo *chr*, sin embargo, consultando su contenido se ve necesario un cambio de tipo.

```
1 df$X56 = as.integer(df$X56)
```

Listing 3.1: Cambio de tipo de la columna X56 del objeto df

Es importante conocer cómo catalogó el autor de los datasets a los registros cuyo valor se desconoce razón. Es común encontrar interrogantes (?), la palabra *unknown* o valores imposibles como negativos (-1) o muy altos (99999...). En R existe un valor especial llamado *NA* que engloba todo tipo de valores faltantes. Contar cuantos valores faltantes tiene un dataset y saber como se van a tratar también es una tarea importante que determina la calidad de los datos

```
1 df[df=='?'] = NA
2 sum(is.na(df))
3 df = df[complete.cases(df),]
```

Listing 3.2: Tratamiento de valores faltantes

En este caso se ha identificado que el creador del dataset indicó los valores faltante con el símbolo ¿". En la primera línea de Listing 3.2 se ha sustituido en todo el *dataframe* el valor ? por NA. A continuación, se obtiene el número de valores faltantes en todo el dataframe y se decide prescindir de todas las filas que no contengan todas las columnas completas. Esta no es la única opción posible para tratar con NA pero sí la que se ha optado en este trabajo final de máster.

Otra acción que se ha tomado para continuar con la limpieza es la eliminación de datos innecesarios. Columnas que solo tienen un valor o columnas que son identificadores de filas no aportan nada al modelo. Es por ello que se ha decidido eliminar todas ellas.

```
unique(df$veiltype) # [1] p
df = dplyr::select(df, -veiltype)
```

Listing 3.3: Identificación de columnas innecesarias

En la primera línea del Listing 3.3 se consultan los valores únicos, diferentes, que tiene la columna *veiltype* del dataset Mushroom. Como solo tiene un valor, p, se elimina la columna con la segunda instrucción al no aportar ningún valor al modelo.

Finalmente, se han considerado datasets que no estén muy desbalanceados, es decir, que ambas clases tengan un número similar de instancias. Esto se hace porque no se quiere considerar un factor determinante a la hora de sacar conclusiones de los experimento. Para conocer esto se utiliza la instrucción mostrada en el Listing 3.4 y devuelve una tabla con la proporción de instancias que tiene cada clase.

```
prop.table(table(df$class))
```

Listing 3.4: Comprobar balanceo de clases

En la Tabla 3.2 se puede ver cómo han quedado los datasets una vez se ha realizado la limpieza de cada uno de ellos. Además se aporta el balanceo de clases para cada uno de los conjuntos de datos.

Nombre	Filas	Columnas				Class Balance
		Total	chr	num	factor	
Mushroom	8124	22	21	0	1	0.518:0.482
Breast Cancer Wisconsin	569	31	0	30	1	0.627:0.373
Ionosphere	351	34	0	33	1	0.359:0.641
Scene	2407	300	0	299	1	0.821:0.179
Cleveland Heart Disease	299	14	0	13	1	0.535:0.465
Banknote	1372	5	0	4	1	0.555:0.445
Tic-Tac-Toe	958	10	9	0	1	0.347:0.653
Haberman	306	4	0	3	1	0.735:0.265
Bank Marketing	7842	17	9	7	1	0.772:0.228
Spambase	4601	58	0	57	1	0.606:0.394
Blood Transfusion	748	5	0	4	1	0.762:0.238
Diabetes	768	9	0	8	1	0.651:0.349

Tabla 3.2: Tabla resumen de los datasets utilizados una vez se ha hecho la limpieza

3.3 Clasificadores empleados

En este trabajo se emplean una amplia variedad de algoritmos de clasficación binaria. En esta sección se hace una breve explicación de cada uno de ellos y de la familia a la que pertenecen.

3.3.1. Algoritmos

Se ha decidido usar un total de 7 algoritmos clasificadores con el fin de abarcar todas las familias posibles. Algunos de los clasificadores que se exponen sirven tanto para pro-

blemas de regresión como de clasificación, sin embargo, a continuación se dará una breve explicación de cada uno de ellos haciendo hincapié en su aspecto como clasificador.

Árbol de decisión

Los modelos basados en árboles son un tipo de algoritmos no parametrizables que funcionan particionando el espacio de las variables en varias regiones, sin solaparse, en base a unas reglas de particionado. Un árbol, por tanto, es una serie de reglas que permite predecir formulando preguntas de sí o no sobre las distintas variables del conjunto de datos. Uno de los métodos para generar el árbol es CART: se parte los datos de entrenamiento en subgrupos con valores de respuesta similares, estos subgrupos, de manera recursiva, generan nuevas particiones binarias a partir de preguntas de sí o no hasta que se llega a un cierto nivel de profundidad. Para la partición de los subgrupos se busca un criterio en concreto que se puede resumir en minimizar las diferencias en los nodos terminales. Entre las ventajas que ofrece este método están que requiere muy poco preprocesamiento y que la lógica que hay detrás es fácil de entender.

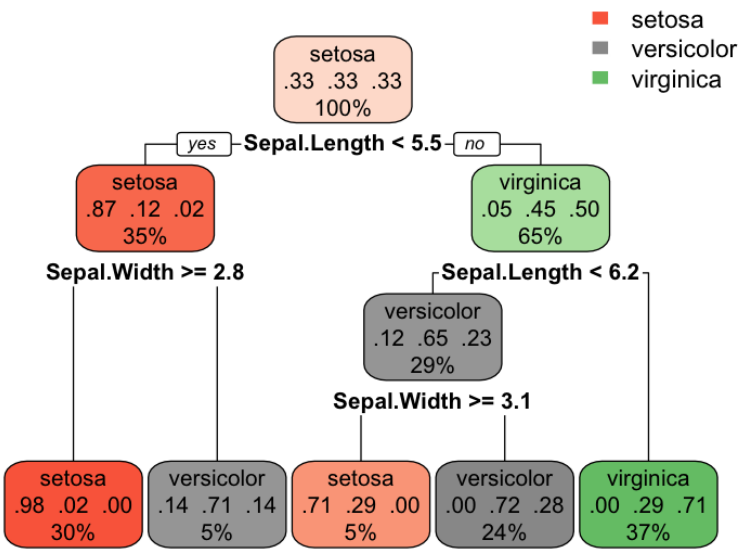

Figura 3.2: Ejemplo de representación de árbol de decisión con el dataset Iris. Imagen extraída de [2].

Bosque aleatorio

Random Forest, o bosque aleatorio, es un método que utiliza múltiples algoritmos de aprendizaje, es decir, un *ensemble*. Puede servir tanto para regresión como para clasificación y utiliza múltiples árboles de decisión aleatorizados para finalmente calcular una media de sus predicciones. Gracias a su estructura paralela, consultar la img:randomForest, tiene un gran rendimiento comparado con otros clasificadores modernos y mejora cuantos más árboles de decisión se añadan. Sin embargo, hay un límite donde la mejora de rendimiento no sale rentable debido a la gran potencia de cálculo que significa añadir más árboles.

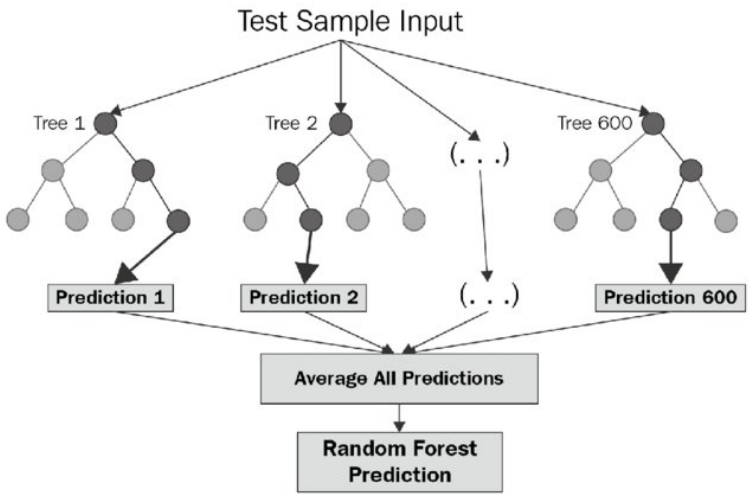

Figura 3.3: Arquitectura paralela del algoritmo Bosque Aleatorio. Imagen extraída de [21].

Máquinas de vector de soporte

Las máquinas de vector de soporte, o SVM tienen un enfoque directo a la clasificación binaria que consiste en encontrar un híperplano en el espacio de las variables que separe la dos clases de la mejor manera posible. Este clasificador surge como solución a los clasificadores de vectores de soporte cuando los datos no son separables linealmente. Es por ello que las SVM permiten aumentar la dimensionalidad a través de *kernels*, funciones complejas sobre los datos que transforman el espacio de las variables limitado en uno de mayores dimensiones.

En el experimento realizado se han utilizado **dos kernels**: el lineal, que equivale a un clasificador de vector de soporte, y el kernel radial, muy útil en conjuntos de datos con forma de espiral, tal y como aparece en la Figura 3.4.

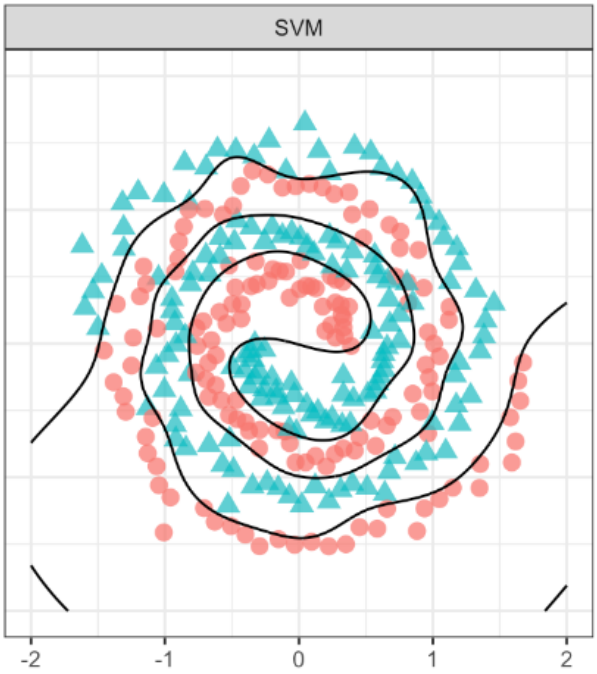

Figura 3.4: Clasificador máquina de vector de soporte con kernel radial sobre un dataset con forma de espiral. Imagen extraída de [2].

KNN

K-Nearest Neighbors (KNN), es un algoritmo que basa su predicción en la similitud con el resto de observaciones. En un problema de clasificación, el algoritmo identifica las k observaciones más parecidas o cercanas al elemento a predecir y devuelve como solución la clase más común de entre esas observaciones. Las medidas de distancia más utilizadas son la Euclídea y la Manhattan; mientras que la primera mide la línea recta entre dos puntos, la segunda, más utilizada en clasificación binaria, mide el tiempo de viaje entre dos puntos, como se puede apreciar en la Figura 3.5. Las muestras de entrenamiento son necesarias en tiempo de ejecución y por ello esto puede llegar a ser ineficiente. Otro aspecto muy importante de este algoritmo es la selección de la k , ya que un valor muy bajo tiende a provocar un sobreajuste, y valores muy altos pueden llevar a un subajuste. No existe una regla general para seleccionar la mejor , depende del tipo de datos, pero sí que se suele elegir un número impar con el fin de evitar empates.

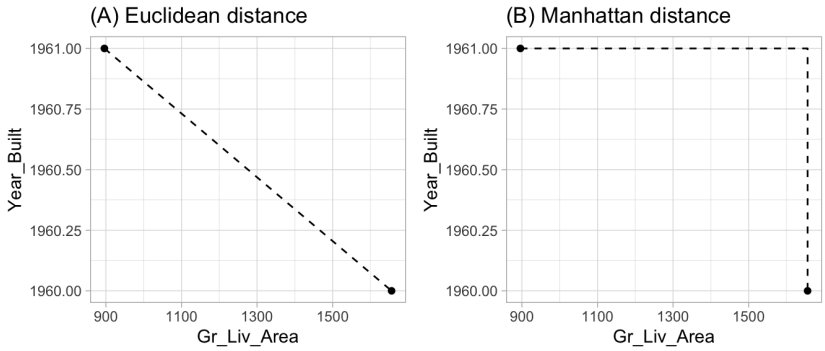

Figura 3.5: Diferencia gráfica entre las distancias Euclídea y de Manhattan. Imagen extraída de [2]

Bayesiano ingenuo

El clasificador *Naive Bayes* es un modelo probabilístico basado en el teorema de Bayes, Ecuación 3.1. Realmente, existen varios clasificadores basados en el teorema de Bayes en función del algoritmo empleado, pero todos tienen en común que las variables predictoras son independientes e iguales entre sí, de ahí el origen en el nombre de *ingenuo*.

Para conocer cómo funciona el algoritmo en primer lugar se convierte el conjunto de datos en una tabla de frecuencia. A partir de esta tabla, se genera una tabla de probabilidad como la que se observa a la derecha en la Figura 3.6.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Figura 3.6: Proceso de transformación de datos para generar un clasificador Naive Bayes.

A partir de este momento, se aplicaría la ecuación de *Naive Bayes*, Ecuación 3.1, para calcular la probabilidad a posteriori de cada clase. La clase con mayor probabilidad es la salida de la predicción.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{3.1}$$

donde A correspondería a la variable de respuesta y B es el conjunto de variables predictoras.

Este clasificador es sencillo y rápido. Da buenos resultados incluso en clasificación multiclase, siempre y cuando se mantenga la asunción de independencia, hecho que en situaciones reales no se da muy frecuentemente.

Regresión logística

Mientras que la regresión lineal sirve para aproximar la relación lineal entre variable respuesta continua y el conjunto de variables predictoras, el clasificador de regresión logística se utiliza cuado la variable de respuesta es binaria. Para evitar errores del modelo lineal en una respuesta binaria, tenemos que modelar la probabilidad de nuestra respuesta con una función que devuelva valores entre 0 y 1. La función utilizada en este caso es la función logística, que se puede ver en la Ecuación 3.2.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

3.4 Diseño de los experimentos

3.4.1. Factor experimental 1: Porcentaje de instancias alteradas

El porcentaje de instancias alteradas indica el número de filas del conjunto de datos que se modifican introduciéndoles ruido. En este caso se ha optado por alterar el 100 % de las instancias, por lo que en este aspecto, este parámetro quedará constante durante todo el experimento. En algunas situaciones reales no ocurre que el 100 % de las instancias se vean afectadas por ruido, por ejemplo un fallo puntual de lectura de un sensor. Debido al gran aumento en coste computacional que supone, queda como trabajo futuro el realizar el experimento con esta variable, estableciendo varios niveles de número instancias perturbadas, tal y como se ha anotado en la Sección 6.1.

3.4.2. Factor experimental 2: Nivel de ruido introducido

El nivel de ruido es la cantidad de ruido introducido en las instancias. Suponiendo que los conjuntos de datos escogidos están limpios, el nivel de ruido se alterará introduciendo ruido de manera artificial. Esto se puede lograr de varias maneras como permutando el valor de las clases de algunas instancias. Sin embargo, el foco de este experimento está en la importancia de los atributos por tanto el ruido se introduce alterando los valores del atributo más importante.

La mayoría de los conjuntos de datos seleccionados están balanceados, por lo que el ruido se introduce en las instancias independientemente de si se trata de la clase minoritaria o no.

En el presente experimento se usan 3 niveles de ruido: 5, 10, 15. Este valor se utiliza de dos maneras distintas en función de si el atributo más importante que se altera es nominal o numérico.

Para inyectar ruido se han seguidos las directrices establecidas en el artículo [7], que se resume en que las instancias seleccionadas para que se les inyecte el ruido verán afectado su valor dentro de un rango de posibles valores. Este ruido es generado aleatoriamente usando distribuciones de probabilidad conocidas, pero dependiendo de si el atributo es nominal o númerico cambia el procedimiento.

Para **atributos nominales** o categóricos, lo primero es obtener la frecuencia de cada posible valor del propio atributo. El resultado es un vector de frecuencias $p = (p_A, p_B, p_C)$ donde A, B, C son los posibles valores que puede tomar el atributo y p_X es la probabilidad de que aparezca el valor X en el atributo. Sobre la instancia que se pretenda inyectar el ruido, se realiza un one hot encoding del atributo en cuestión, tal y como se muestra en la Figura 3.7, y se guarda en un vector t. Para insertar un nivel de ruido ν, calculamos $\alpha = 1 - e^{-\nu}$. Con este valor se puede calcular la nueva probabilidad: $p' = \alpha \times p + (1 - \alpha) \times t$

Figura 3.7: One-hot encoding sobre el atributo *color*. Imagen extraída de [19]

Esto se traduce en el siguiente código R que podemos ver en el Listing 3.5. La primera instrucción sirve para obtener la frecuencia de cada uno de los posibles valores de la columna correspondiente al atributo más importante.

```
newAtrib = dfTrain %>%
  group_by(dfTrain[[mostImpAttr]]) %>%
  dplyr::summarise(n = n()) %>%
      mutate(Freq = n/sum(n))
p = newAtrib$Freq
alpha = 1-exp(-v)
t = mutate(dfTrain, mostImpAttrP = paste(mostImpAttr,dfTrain[[mostImpAttr]],
    sep = "_"),
  attrP_value = 1)
t = t[!duplicated(t), ]
t = tidyr::spread(t, key=mostImpAttrP, value = attrP_value, fill=0)
t = dplyr::select(t, starts_with(paste0(mostImpAttr,"_")))
for(row in 1:nrow(t)){
  p1 = alpha*p+(1-alpha)*t[row,]
     dfTrain[[mostImpAttr]][row] = sample(unique(dfTrain[[mostImpAttr]]),1,prob
         = p1, replace = FALSE)
}
```

Listing 3.5: Código para el proceso de inyección de ruido en atributos nominales

Para **atributos numéricos**, en primer lugar, calculamos la desviación estándar σ de todos los valores del atributo seleccionado. Siendo x el valor a modificar, obtenemos el nuevo valor a través de una distribución normal y usando x como media y $\sigma \times \nu$ como la desviación estándar. En la Ecuación 3.3 se puede ver la fórmula.

$$x\prime = N(x, \sigma \times \nu) \tag{3.3}$$

3.4.3. Factor experimental 3: Subconjunto alterado

En este experimento es posible introducir ruido en dos subconjuntos del *dataset*: el subconjunto de entrenamiento o el de pruebas. Es por ello que se ha dividido el trabajo en dos experimentos. En el primero se alteran con ruido los datos de training, y en el segundo, los datos de testing. Con ello se consigue tener dos tipos de experimentos aprovechando los mismos datasets, clasificadores y niveles de ruido.

CAPÍTULO 4

Experimento 1: Ruido en training

En este primer experimento se inyecta ruido artificial en el subconjunto de datos de entrenamiento, concretamente en el atributo más importante de cada dataset. El objetivo es estudiar cómo afecta la perturbación de los datos a la precisión de los clasificadores. La inyección de ruido sobre los datos de entrenamiento es una de las principales razones por las que surge el overfitting, o sobreajuste, dando lugar a predicciones erróneas.

4.1 Overfitting

El sobreajuste, o *overfitting* en inglés, es un problema a prestar atención en el aprendizaje automático supervisado. El sobreajuste aparece a raíz de la presencia de ruido, junto al limitado tamaño de los conjuntos de entrenamiento y la complejidad de los clasificadores. Cuando un *dataset* es pequeño y contiene una gran cantidad de ruido, es más probable que el modelo termine aprendiendo de esos datos perturbados y afecte a las predicciones futuras [29]. La consecuencia directa del *overfitting* es que impide generalizar perfectamente los modelos para ajustarse tanto a los datos de entrenamiento como a los datos de test. Un modelo que se sobreajusta al conjunto de entrenamiento tiene ciertas dificultades para tratar con instancias nuevas del conjunto de pruebas, que posiblemente sean diferentes a las de entrenamiento.

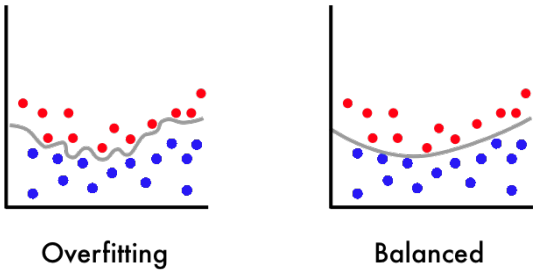

Figura 4.1: Overfitting vs modelo balanceado

4.2 Descripción e implementación

En esta sección se proporciona al lector una breve descripción del experimento junto a un diagrama representando el proceso que se ha seguido. También se va mostrando la implementación del experimento en el lenguaje R.

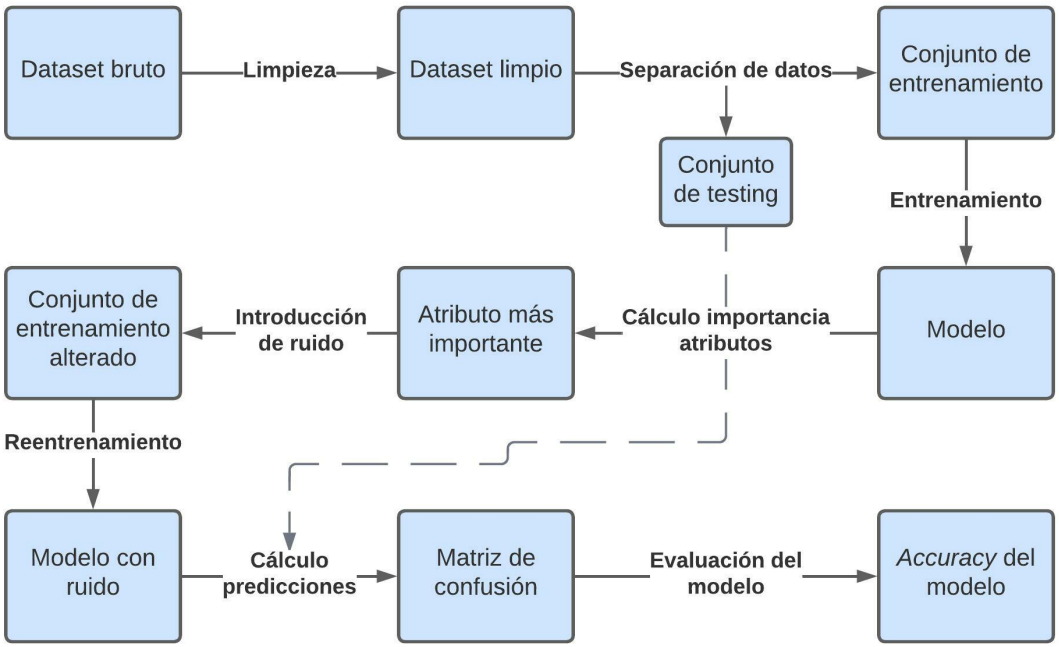

Figura 4.2: Proceso del experimento 1

Inicialmente, se parte del dataset en bruto al que se le hace una limpieza de datos, comentada en la Sección 3.2, quedándonos con el dataset listo para ser utilizado. A continuación, se separan los datos con una distribución de 70 % de muestras para entrenamiento y 30 % para testing, dando lugar a los dos subconjuntos, *dfTrain* y *dfTest*.

```
1 traIndex = createDataPartition(df$class, p=.7, list=FALSE)
2 dfTrain = df[trainIndex,]
3 dfTest = df[-trainIndex,]
```

Listing 4.1: Separación de los datos

Con los datos de entrenamiento, se calcula el primer modelo de clasificación. En el Listing 4.2 se observa el cálculo del modelo con la técnica *decisionTree*, pero el código es equivalente para cualquier método teniendo que modificar solo el argumento *method*. Es importante detenerse y apreciar el papel que tiene la función *caret::train()* ya que, gracias a su capacidad como agregador, permite aplicar todos los motores directamente cambiando uno de sus parámetros. Los meta-motores permiten mantener una consistencia en cuanto a cómo se ha especificado las entradas y se han extraído las salidas, sin embargo, pueden ser algo menos flexibles. El primer parámetro de la función *train* corresponde a la fórmula para especificar cuales son los atributos y cuál es la clase. En este caso la columna *class* es la clase y el punto (.) denota el resto de atributos del dataframe *dfTrain*.

El parámetro *trControl* hace referencia a la configuración del remuestreo utilizado en el entrenamiento. En la linea 1 se puede ver la definición del remuestreo, **validación cruzada con 5 pliegues**.

```
1 control = trainControl(method = "repeatedcv",number = 5, verboseIter = TRUE)
2 fit = caret::train(class~., data = dfTrain, method = "rpart", trControl =
    control)
```

Listing 4.2: Validación cruzada y cálculo del modelo

Con este modelo, objeto *fit* en el código, es con el que se extrae el atributo más importante. En la Sección 2.3 se encuentra detallado el procedimiento para obtener el atributo más importante de cada dataset por lo que se omitirá en este apartado.

El siguiente paso es la introducción de ruido únicamente en el atributo más importante. En la Subsección 3.4.2 se comenta de manera exhaustiva cómo se ha llevado a cabo la perturbación de los datos y cómo se ha implementado en el lenguaje R.

Ahora que están alterados los datos de entrenamiento, se realiza otro cálculo de para obtener un modelo afectado por el ruido. Se lleva a cabo de la misma manera que en el Listing 4.2. Una vez obtenido este nuevo modelo se realizan las predicciones con el conjunto de datos destinado al testing y se obtiene la matriz de confusión. La implementación correspondiente se encuentra en el Listing 4.3

```
predict_unseen = predict(fit,dfTest)
cm = caret::confusionMatrix(predict_unseen, dfTest$class)
```

Listing 4.3: Predicción y cálculo de la matriz de confusión

4.3 Resultados

En esta sección se exponen y se comentan los resultados obtenidos del experimento realizado. En este experimento se ha utilizado como métrica la precisión, *accuracy*,

En la Figura 4.3 se muestran 12 gráficas, cada una de ellas correspondiente a un conjunto de datos. En cada gráfica podemos observar en el eje Y la precisión y en el eje X los diferentes niveles de ruido en porcentaje. Cada gráfica está identificada con una letra de la A a la L que, por motivos de espacio, hace referencia a un dataset en concreto. En la Tabla 4.1 se muestra a qué dataset corresponde cada letra.

Identificador	Dataset
A	Bank Marketing
B	Banknote
C	Diabetes
D	Haberman
E	Cleveland Heart Disease
F	Ionosphere
G	Mushroom
H	Scene
I	Spambase
J	Tic Tac Toe
K	Blood Transfusion
L	Breast Cancer Wisconsin

Tabla 4.1: Índice de letra identificadora y su dataset correspondiente en Figura 4.3

En base a los resultados vistos en la Figura 4.3, se puede observar la aparición de ciertos patrones:

En las gráficas correspondientes a los datasets A, B, C, G y H se puede apreciar una disminución general importante de la precisión cuando se pasa de un conjunto de datos sin ruido a un nivel de ruido del 5 %. Sin embargo, desde este punto en adelante, el ruido no afecta tanto e incluso los clasificadores no pierden precisión a pesar de aumentar el porcentaje de ruido hasta el 15 %.

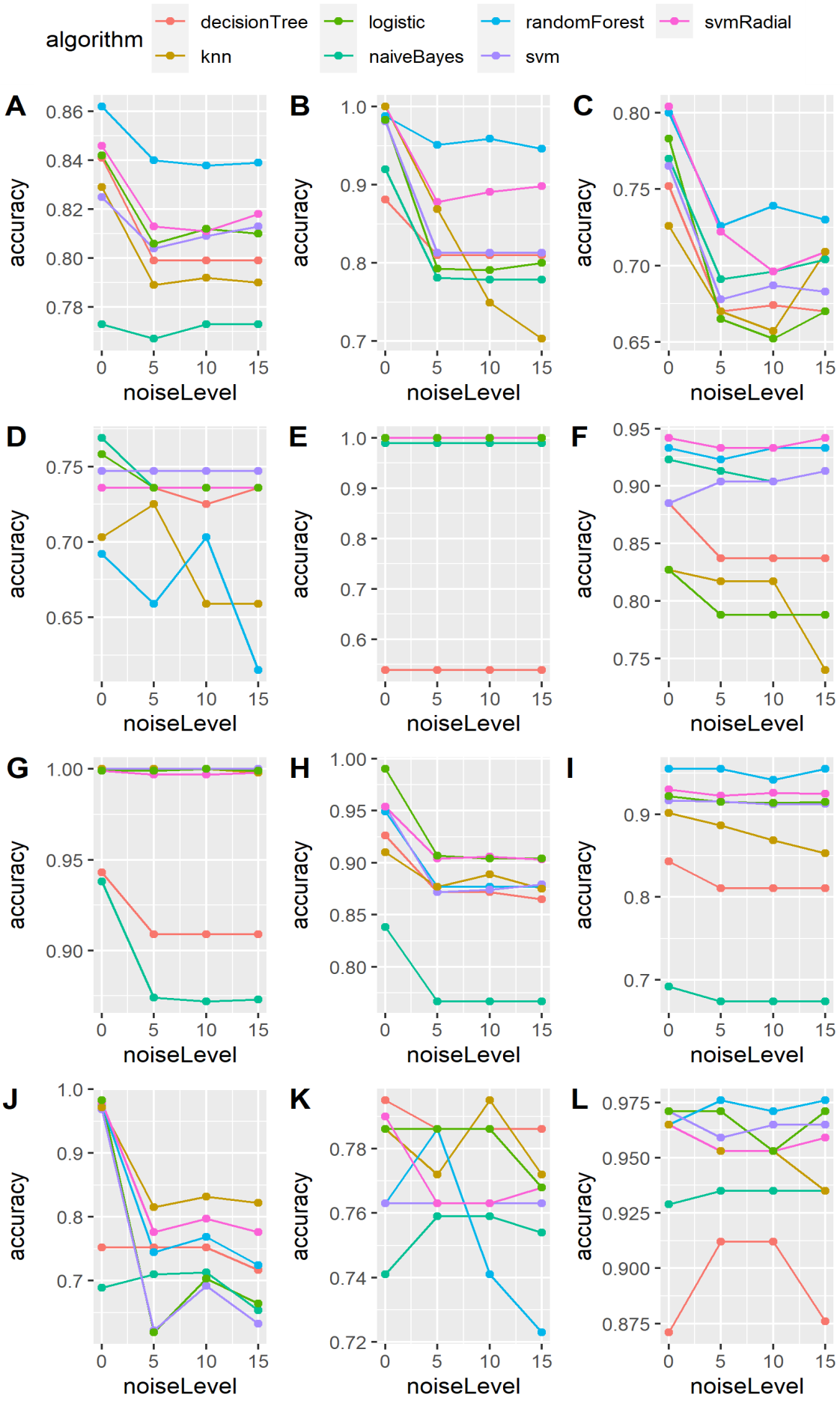

Figura 4.3: Resultados Experimento 1: Ruido en training

Existen otros datasets (K, L) donde los resultados son más caóticos por culpa de algunas técnicas que no tienen un comportamiento general muy claro. Por ejemplo, la técnica *randomForest* en el dataset con la etiqueta K obtiene una mayor precisión con la introducción de ruido del 5 % pero en los siguientes niveles la métrica se ve muy afectada negativamente. La técnica *knn* en este dataset disminuye, aumenta y finalmente disminuye su precisión.

En la Tabla 4.2 se pueden observar estos datos representados de otra manera permitiendo sacar conclusiones más fundadas. Aquí se puede ver la precisión media de las diferentes técnicas dependiendo del ruido y, en las últimas 3 columnas, las diferencia en precisión entre los niveles de ruido 0 - 5, 0 - 10 y 0 - 15, respectivamente.

	Noise Levels						
	0	5	10	15	diff0to5	diff0to10	diff0to15
decisionTree	81.64	78.61	78.55	77.96	3.03	3.09	3.68
knn	88.50	84.78	83.43	82.13	3.72	5.07	6.37
logistic	90.37	83.21	83.66	83.54	7.16	6.71	6.83
naiveBayes	83.09	79.97	79.97	79.59	3.12	3.12	3.50
randomForest	90.69	86.98	87.27	85.98	3.71	3.42	4.71
svm	89.81	83.98	84.72	84.35	5.83	5.09	5.46
svmRadial	91.21	86.65	86.74	86.93	4.56	4.47	4.28

Tabla 4.2: Tabla resumen de la precisión de los distintos clasificadores dependiendo del nivel de ruido en el conjunto de training.

Con estos nuevos datos se puede observar que inicialmente, cuando no hay ruido en los datasets, la técnica con mayor precisión es *svmRadial* (91.21 %), máquina de vector de soporte con kernel radial, seguida muy de cerca por *randomForest* y *logistic*. En los niveles intermedios de ruido (5-10 %) se distancian ligeramente, incluso sale ganadora *randomForest*. Pero, con un 15 % de ruido, vuelve a tener prácticamente un 1 % de precisión de ventaja *svmRadial* sobre *randomForest*. Por otro lado, la peor técnica en cuanto a precisión, a cualquier nivel de ruido es Árbol de decisión, con un 81.64 % sin ruido.

La técnica que más afectada se ve por el ruido es *logistic*, que pierde hasta un 7.16 % de precisión de media. Sin embargo, junto a *naiveBayes* y *svmRadial*, es una de los métodos en los que por más que se añada ruido no afecta prácticamente a la precisión de los modelos.

Pese a todo lo mencionado, los resultados varían mucho con respecto al dataset utilizado por lo que no se puede generalizar ninguna conclusión de una manera segura.

CAPÍTULO 5
Experimento 2: Ruido en testing

El segundo experimento se centra en la introducción de ruido en los datos de testing, de nuevo en el atributo más importante. Gracias a este experimento se puede estudiar cómo de resistente es el modelo desarollado a los cambios. Dependiendo de cómo afecten los datos perturbados a la precisión del modelo se podrá concluir si el modelo es robusto o no.

5.1 Robustez

La robustez, según el glosario de estándar de IEEE para terminología de ingeniería software [1], consiste en el grado en el que un sistema o componente puede funcionar correctamente en presencia de entradas inválidas o condiciones del entorno tensas. La definición que aporta Zhang J et al (2022) sobre la robustez es la diferencia entre la corrección del sistema y la corrección del sistema con perturbaciones sobre cualquiera de sus componentes. También existe un término, surgido a raíz del *adversarial learning*, llamado *adversarial robustness*. Este tipo de robustez está orientada a los datos perturbados difíciles de detectar.

Para el objetivo del proyecto, se entiende por robustez de un algoritmo "la propiedad que indica cómo de correcto es un algoritmo mientras se está testeando con un nuevo conjunto de datos". Cuanto mayor sea la robustez de un algoritmo, menor es la distancia entre el error de entrenamiento y el error de testing.

5.2 Descripción e implementación

El objetivo de este experimento es evaluar si el modelo cambia de decisión cuando el conjunto de test es perturbado. Por tanto, una vez se ha obtenido el atributo más importante del dataset, se introduce el ruido en él, pero esta vez en el subconjunto de testing. El siguiente paso es el cálculo de predicciones con este conjunto alterado y el modelo ya entrenado. Estas predicciones que ha realizado el modelo se comparan con las realizadas sin haber alterado el conjunto de testing. Otro cambio con respecto al experimento anterior es el uso de la métrica Kappa para la evaluación del modelo, que se explica en profundidad en la Subsubsección 2.2.4. Es importante indicar que este experimento se trata de una prueba de concepto debido a que no se aplican varias iteraciones o pliegues sobre el conjunto de testing para confirmar la validez de los resultados de una manera más estricta.

En el Figura 5.1 se puede ver la implementación de la parte de predicción y cálculo de la matriz de confusión.

```
predict_unseen = predict(fit,dfTest)
if(v==0){
  cm = caret::confusionMatrix(predict_unseen, predict_unseen)
}else{
  cm = caret::confusionMatrix(predict_unseen, predict(fit,dfTestNoised))
}
resultOnTest[nrow(resultOnTest) + 1,] = c(dataset,method,v, 100, cm$overall["
    Accuracy"], cm$overall["Kappa"])
```

Listing 5.1: Predicción y cálculo de la matriz de confusión

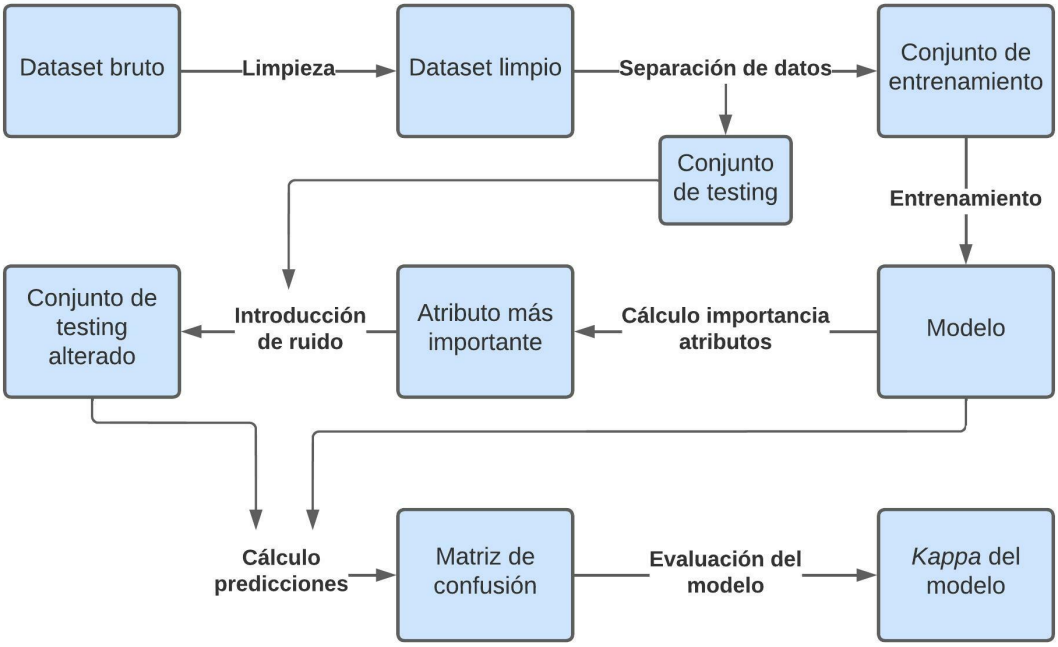

Figura 5.1: Proceso del experimento 2

5.3 Resultados

En la Figura 5.2 se muestran las gráficas obtenidas de cada uno de los datasets. En la Tabla 4.1 del capítulo anterior se encuentra a qué letra corresponde cada dataset de la Figura 5.2.

En referencia a la Figura 5.2, se puede apreciar claramente un decremento importante del coeficiente Kappa cuando se introduce ruido. Un patrón que se cumple en la mayoría de los casos es que el mayor descenso, en cuanto a valor Kappa, se produce entre el nivel de 0 % de ruido y el 5 % de nivel del ruido.

Otro dato destacable es que únicamente *randomForest*, *naiveBayes*, *svm* y *svmRadial* se mantienen por encima de 0.8 una vez se ha introducido ruido. Uno de los datasets en los que peores resultados han tenido la mayoría de las técnicas en el D, *Haberman*, que coincide con ser el segundo dataset con menos instancias (306).

La Tabla 5.1 muestra el coeficiente kappa medio por cada técnica junto con los diferentes niveles de ruido. Además, en las últimas 3 columnas, se encuentran las diferencia en kappa entre los niveles de ruido 0 - 5, 0 - 10 y 0 - 15, respectivamente.

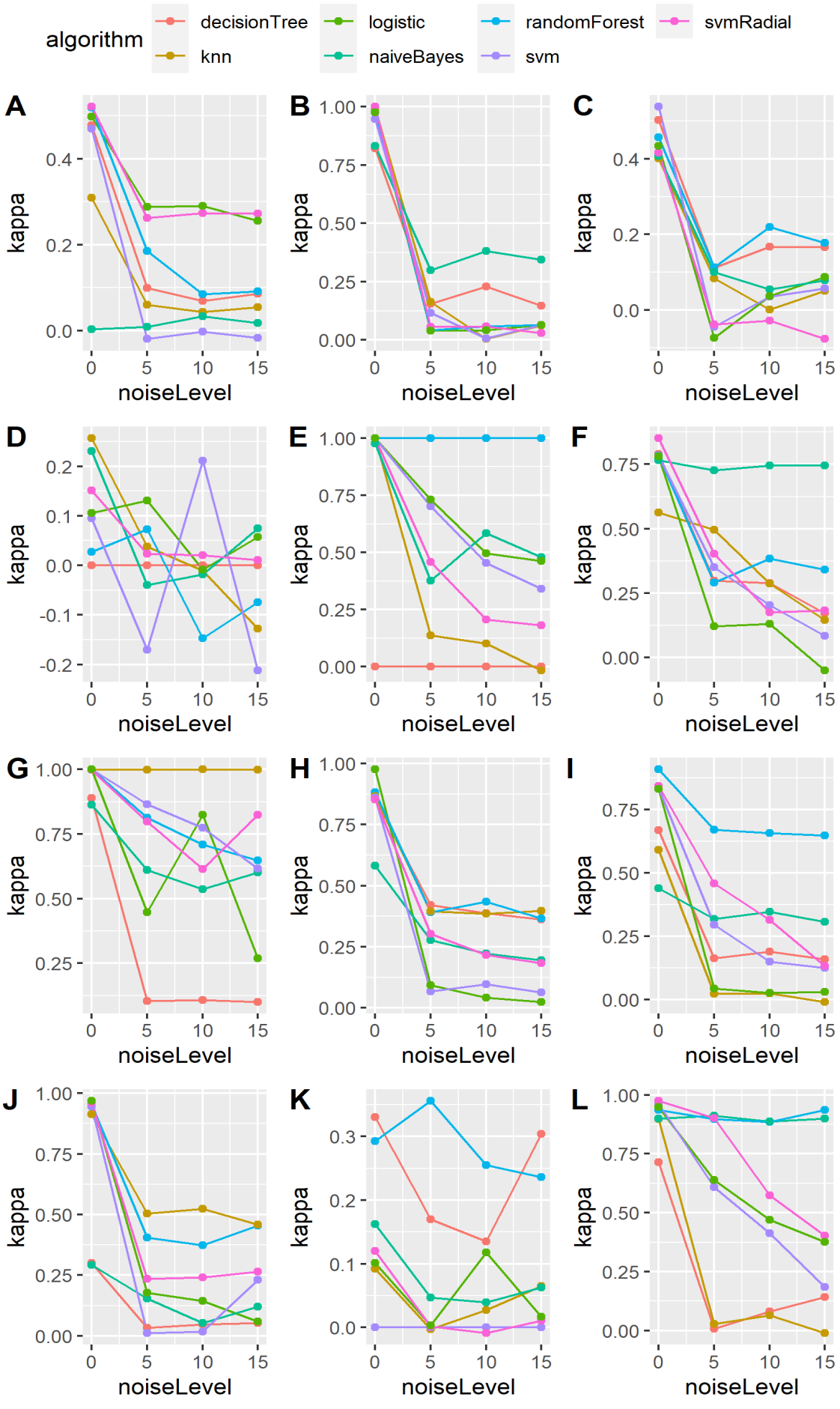

Figura 5.2: Resultados Experimento 2: Ruido en testing (Kappa)

	Noise Levels						
	0	5	10	15	diff0to5	diff0to10	diff0to15
decisionTree	0.53	0.13	0.14	0.14	0.40	0.39	0.39
knn	0.66	0.24	0.20	0.17	0.41	0.45	0.48
logistic	0.72	0.22	0.22	0.14	0.50	0.50	0.58
naiveBayes	0.54	0.32	0.32	0.33	0.22	0.22	0.21
randomForest	0.73	0.44	0.41	0.41	0.29	0.32	0.32
svm	0.70	0.23	0.20	0.13	0.47	0.51	0.57
svmRadial	0.72	0.32	0.22	0.20	0.40	0.50	0.52

Tabla 5.1: Tabla resumen del coeficiente Kappa de los distintos clasificadores dependiendo del nivel de ruido en el conjunto de testing.

En base a la Tabla 5.1, se puede apreciar que las técnicas *logistic*, *randomForest* y *svmRadial* tienen de media un alto coeficiente Kappa cuando no hay ruido en los datos.

En cuanto se introduce ruido, un 5 %, las técnicas que mejor resisten son *randomForest*, que solo decrementa 0.29 puntos desde el valor inicial 0.73, y *naiveBayes*, que solo decrementa 0.22 puntos desde el valor inicial 0.54. El resto de técnicas se ven bastante afectadas llegando a decrementar 0.50 el coeficiente Kappa como es el caso de la técnica *logistic*.

Si avanzamos hasta que se ha introducido un nivel de ruido del 10 %, se puede observar que no hay grandes cambios, ligeros decrementos salvo en la técnica *svmRadial* que decrementa en 0.10 más el coeficiente.

Cuando el ruido inyectado sobre el conjunto de testing está al 15 % la mayoría de coeficientes se mantienen excepto las técnicas *logistic* y *svm* que ven otro decremento del 0.08 y 0.06, respectivamente. Con estos datos finales se puede concluir que con ruido al 15 % los clasificadores más afectados negativamente han sido los dos mencionados anteriormente seguidos de *svmRadial*. *naiveBayes* es la técnica que menos le afecta el ruido en cualquiera de los niveles que se han probado pero *randomForest* es la técnica que mejores resultados ha dado en todos los niveles.

Finalmente, se puede concluir con que ninguna de las técnicas es robusta si el ruido afecta al atributo más importante ya que se aprecia un decremento importante del coeficiente kappa. Un factor a tener en cuenta es que se están perturbando el 100 % de las instancias pese a utilizar un nivel de ruido bajo.

CAPÍTULO 6
Conclusiones

En este trabajo se analiza el efecto de introducir diferentes niveles de ruido en el atributo más importante de distintos datasets utilizando una gran variedad de algoritmos de clasificación binaria. En el primer experimento se introduce el ruido en el subconjunto de training con el fin de estudiar cómo se comporta la precisión del modelo y poder analizar el efecto del *overfitting*. De este primer experimento se ha podido extraer que en una serie de conjuntos de datos (*Bank Marketing* y *Diabetes*, entre otros) la precisión disminuía al introducir un 5 % de ruido pero a partir de ese nivel, pese a haber más ruido, la precisión se mantiene. En otros conjuntos de datos, sin embargo, los resultados son más caóticos y se puede ver cómo la métrica *accuracy* para cada técnica tiene un comportamiento muy diferente con respecto al resto (el dataset *Blood Transfusion* es un claro ejemplo). Las técnicas a las que menos les afecta los aumentos de ruido son *Naive Bayes*, *SVM* y *Regresión Logística*. Sin embargo, de media, los métodos que mayor precisión tienen son *Random Forest* y *SVM Radial*.

En el segundo experimento se pretende estudiar la robustez de la técnica viendo si el modelo cambia de clase cuando se modifica el conjunto de test. Este experimento ha sido realizado como prueba de concepto ya que no se ha realizado ningún tipo de remuestreo o pliegues para el conjunto de testing. En él se ha concluído que ninguna técnica es robusta frente a la introducción de ruido en el atributo más importante ya que, incluso con pequeños porcentajes de ruido, se han visto muy afectados negativamente los coeficientes de la métrica Kappa.

A modo de retrospectiva, se han considerado un total de 12 datasets de clasificación binaria bastante variados en cuanto a temática, balanceo de clases, número de instancias y número de variables independientes. Estos datasets, tras ser limpiados, reúnen más de 28000 filas y más de 500 columnas que sirven como base del proyecto junto con 7 de los algoritmos más conocidos de clasificación. Además, se utiliza validación cruzada con 5 pliegues para conseguir una mayor validez de los resultados. Se ha dividido el trabajo en dos experimentos: uno sobre el subconjunto de entrenamiento y otro sobre el de testing, introduciendo ruido en 4 niveles diferentes.

6.1 Trabajo a futuro

El presente trabajo final de máster se puede ampliar de muchas maneras. A continuación se recogen algunas ideas para crear un conjunto de experimentos y tener una visión más global de cómo afecta el ruido a diferentes conjuntos de datos y algoritmos en base a los diferentes parámetros que se podrían modificar:

Problemas de regresión Realizar los experimentos con tareas de regresión sería muy interesante para conocer cómo afecta el ruido. Estos nuevos experimentos darían lugar a nuevos conjuntos de datos, nuevos algoritmos e incluso nuevas métricas de evaluación del modelo.

Conjuntos de datos multiclase Los conjuntos de datos que se han utilizado en este trabajo solo tienen dos clases, por lo que es interesante estudiar cómo se comportan los modelos frente al ruido cuando hay más de dos clases.

Desbalanceo de clases En el presente trabajo se han utilizado conjuntos de datos mayormente balanceados por lo que otra opción es ver cómo afectaría la introducción de ruido en conjuntos de datos con clases desbalanceadas.

Número de instancias perturbadas Otro parámetro que se puede modificar para generar otra serie de experimentos puede ser el porcentaje de instancias de los conjuntos de datos que se alteran. Se pueden establecer varios niveles y analizar la precisión del modelo dependiendo de si se alteran el 100 % de las instancias o solo el 50 %.

Número de variables de entrada Un factor muy relacionado con el sobreajuste, la varianza y el sesgo es el número de inputs que recibe el modelo. Dependiendo de la complejidad del modelo, si hay muchos atributos que se tienen en cuenta para el modelo o no, este tendrá mayor precisión o consistencia.

Optimización de hiperparámetros Algunos algoritmos mejoran mucho su rendimiento cuando se ajustan sus parámetros de una manera óptima. Por ejemplo, no se ha especificado en ningún momento el parámetro k para el algoritmo kNN. La idea es ver cómo afecta el ruido a los algoritmos comparando su versión optimizada con su versión base.

Bibliografía

[1] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990.

[2] Brad Boehmke and Brandon Greenwell. *Hands-on machine learning with R*. Chapman & Hall/CRC The R Series. CRC Press, London, England, November 2019.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[4] David B. Dahl, David Scott, Charles Roosen, Arni Magnusson, and Jonathan Swinton. *xtable: Export Tables to LaTeX or HTML*, 2019. R package version 1.8-4.

[5] Jan Philipp Dietrich and Waldir Leoncio. *citation: Software Citation Tools*, 2022. R package version 0.6.3.

[6] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[7] Cèsar Ferri, José Hernández-Orallo, Adolfo Martínez-Usó, and M José Ramírez-Quintana. Identifying dominant models when the noise context is known.

[8] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. *Statistical methods for rates and proportions*. john wiley & sons, 2013.

[9] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995.

[10] Alboukadel Kassambara. *ggpubr: 'ggplot2' Based Publication Ready Plots*, 2020. R package version 0.4.0.

[11] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer New York, 2013.

[12] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159, March 1977.

[13] Kuhn Max. Building predictive models in r using the caret package. *Journal of Statistical Software*, 28, 11 2008.

[14] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943.

[15] Annette M. Molinaro, Richard Simon, and Ruth M. Pfeiffer. Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, 21(15):3301–3307, 05 2005.

[16] Christoph Molnar, Bernd Bischl, and Giuseppe Casalicchio. iml: An r package for interpretable machine learning. *JOSS*, 3(26):786, 2018.

[17] Robert A Monserud and Rik Leemans. Comparing global vegetation maps with the kappa statistic. *Ecological modelling*, 62(4):275–293, 1992.

[18] Reza Moradi, Reza Berangi, and Behrouz Minaei. A survey of regularization strategies for deep models. *Artificial Intelligence Review*, 53(6):3947–3986, December 2019.

[19] George Novack. Building a one hot encoding layer with tensorflow, Jun 2020.

[20] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022.

[21] Jessica M Rudd and Herman "gene" Ray. An empirical study of downstream analysis effects of model pre-processing choices. *Open J. Stat.*, 10(05):735–809, 2020.

[22] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2):206–226, 2000.

[23] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990.

[24] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[25] R Core Team. The comprehensive r archive network.

[26] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[27] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, October 1950.

[28] Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Lin Pedersen, Evan Miller, Stephan Milton Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Paige Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo, and Hiroaki Yutani. Welcome to the tidyverse. *Journal of Open Source Software*, 4(43):1686, 2019.

[29] Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168:022022, feb 2019.

Glosario

adversarial learning Rama de investigación del aprendizaje automático que estudia los ataques hacia los algoritmos y las defensas frente a esos ataques.. 2

dataframe Un dataframe es una estructura de datos de dos dimensiones en la que se guarda información de diferentes tipos ordenada por columnas. Similar a una hoja de cálculo o una tabla SQL.. 17

hiperparámetro Parámetro cuyo valor ayuda a ajustar y optimizar el proceso de aprendizaje de un algoritmo. No todos los algoritmos tienen hiperparámetros pero la mayoría tienen uno o más.. 9

leave-one-out cross-validation Caso especial de validación cruzada donde el número de pliegues iguala al número de instancias en el data. El algoritmo se aplica entonces una vez por cada instancia, usando todas las otras como entrenamiento y utilizando esa como único valor de testeo.. 8

one hot encoding Codificación que crea una columna por cada valor distinto que pueda obtener la característica a codificar. Para cada instancia o registro se marcará con un 1 la columna con el valor de ese registro y se marcará con 0 el resto de columnas. En la figura Figura 3.7 se puede ver la codificación con un ejemplo.. 24

sobreajuste Fenómeno que ocurre cuando el modelo aprende de una manera muy detallada a partir de los datos de entrenamiento hasta el punto de afectar negativamente en el rendimiento del modelo con datos nuevos.. 3, 7, 9, 22, 27, 38

subajuste Fenómeno que ocurre cuando el modelo no se ajusta correctamente y no tiene un buen rendimiento en los datos de entrenamiento.. 7, 22

variable categórica Una variable categórica, o nominal, es una variable de tipo cualitativo o que sus valores representan categorías sin clasificación (orden) entre ellas.. 5

variable predictora Atributo, variable independiente, característica, variable de entrada o predictor son los sinónimos de **variable predictora** en el ámbito del aprendizaje automático que se usan en este documento.. 5

variable objetivo Variable dependiente, variable respuesta, variable de salida o clase son los sinónimos de **variable objetivo** en el ámbito del aprendizaje automático que se usan en este documento.. 5

Siglas

CART Classification And Regression Tree. 20

CRAN The Comprehensive R Archive Network. 13, 14

EDA Exploratory Data Analysis. 6

IDE Integrated Development Environment. 14

KNN K-Nearest Neighbors. 22

MNIST Modified National Institute of Standards and Technology. 1

SVM Support Vector Machine. 21