UNIVERSIDAD
DE GRANADA

Aprendizaje PAC.

Clasificación no balanceada: particularización a *small disjuncts*.

Proyecto final de Carrera

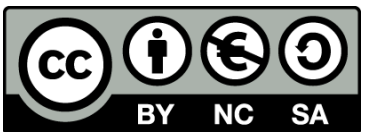
Ingeniería informática y Matemáticas

Autor

Ignacio Cordón Castillo

Tutores

Salvador García López
Francisco Herrera Triguero



Granada, septiembre de 2017

RESUMEN

El siguiente trabajo contiene una formalización matemática del aprendizaje automático, centrándonos en clasificación binaria, conocida como aprendizaje PAC. Para su formalización ofreceremos una introducción a σ-álgebras y σ-álgebras producto y demostraremos desigualdades clave para construir nuestra teoría PAC. Nuestro resultado más importante será el teorema fundamental del aprendizaje PAC, que relaciona estadística, combinatoria y aprendizaje automático. También describiremos otro paradigma de aprendizaje, más laxo que la cognoscibilidad PAC: el aprendizaje no-uniforme.

El desarrollo informático presenta el problema de la clasificación no balanceada. Describiremos distintas aproximaciones a la resolución del problema, centrándonos en una, el *oversampling*. Dentro del *oversampling* reseñaremos las ideas subyacentes en una serie de algoritmos recientes, aún no disponibles en `R`, y los implementaremos en dicho lenguaje, apoyándonos en `C++` para acelerar algunos de ellos. Por último, se hará una pequeña experimentación con *small disjuncts*, un concepto estrechamente ligado al debalanceo de clases.

PALABRAS CLAVE Aprendizaje automático, aprendizaje PAC, clasificación no balanceada, *oversampling*, *small disjuncts*

ABSTRACT

Machine learning mathematical foundations and algorithms have been one of the most studied areas in both mathematics and computer science in the past few years, due to the constant improvement in computer features. Lots of fundamental questions arise: what does it mean to extract knowledge out of data?, when can we learn from the data?, how much data do we need?

One of the most researched topics in machine learning is classification problems: given a sequence of elements belonging to a domain, which have been labeled, how could we learn from them in a way that if new samples arrive we would be able to label them according to the knowledge acquired?. We will specially focus on binary classification problems, those which only have two possible labels. We will also study, in a more empirical framework, the particular case in which one class has more examples than the other one: imbalance classification.

The aim of this work is both to introduce a mathematical theory for machine learning, which has in the PAC learning an excellent formalization, and to study and code some of the most recent algorithms that have been published in scientific journals on the topic of imbalance classification.

MATHEMATICAL INTRODUCTION

We introduce concepts such as product ring and semiring of sets, and their relationship with σ-algebras. We give the notion of measure, without distinction between σ-algebras and other sets, whereas in classic literature the latter ones are called premeasures. We also introduce the outer measures, and we advance towards the construction of a σ-algebra based on a premeasure over some space, and the extension of that outer measure to a measure. The result which will give us such extension is Carathéodory's theorem.

This introduction's purpose is to provide us with basic tools to develop the following theory and to answer the question: given a set of probabilistic spaces, does it exist a product σ-algebra space in which the product of the sets has the product of the probabilities as measure?.

This first part includes demonstrations for Markov and Hoeffding's inequalities, the former one based on the Hoeffding lemma. Those will be key inequalities in our subsequent progress.

MACHINE LEARNING INTRODUCTION AND PAC FRAMEWORK

We will provide motivations for the machine learning theory, as well as basic definitions such as domain set, label set, true labeling, instance generation, training set, hypothesis, hypothesis' error or learning algorithm. On the one hand, the error of an hypothesis respect to the true labeling function will be defined as the expectation that the two are different. On the other hand, the error over a training set will be established as the mean of the number of instances where it fails.

We will also describe a relation between hypothesis and the sets that those hypothesis classify as one of the classes. Thus, we will be able to work both with hypothesis or sets. The essential algorithms we will be using will be empirical risk minimizers, that is, all those which intend to minimize the error over a given training set.

All those concepts will sum up to the very first notion of PAC learning: we will say that a hypothesis collection is learnable if we can allways guarantee an small error from one of them with respect to the true labeling function, under certain restriction over probabilistic confidence. In particular, we will prove that every finite collection of hypothesis is learnable, and that there exist infinite learnable collections, such as rectangle hypothesis.

The PAC notions will be weakened as much as possible until we arrive to a more flexible definition of learning: APAC learning, which is rather similar to PAC learning, with the main difference that we can define our own error function, and we allow the instances of both classes to suffer from overlaping (an instance could belong to both classes). As a result, we will show that the concept of APAC implies the PAC concept under certain mild assumptions.

FUNDAMENTAL THEOREM AND FURTHER WORK

We will define Glivenko-Cantelli classes (uniform classes) of hypothesis, and show that being Glivenko-Cantelli is an stronger assumption than being APAC. As a result of the work developed under this chapter, we will prove again that finite classes are learnable, but under a random error function and with worse asymptotic complexity.

In addition to this, we will show, by means of the No Free Lunch Theorem, that if we only see half or less of the instances of a domain, that domain is not PAC learnable. Moreover, we will relate this theorem with a rather known fact: the bias-complexity tradeoff: the more complex the class of functions we are trying to learn, the lower the error and the harder the learning process.

Vapnik-Chervonenkis classes will also be introduced, along with the concept of shattering. This is a pure combinatorics definition. The Sauer-Shelah lemma will give us a relationship between the Vapnik-Chervonenkis dimension and the number of possible hypothesis in terms of their image of a training set of fixed size.

At this stage, we will have all the pieces to prove the fundamental theorem of PAC learning, which connects statistical concepts, such as Glivenko-Cantelli classes, combinatorial ones (Vapnik-Chervonenkis dimension) and PAC and APAC concepts.

Finally, at the end of the mathematics part, we will suggest another learning paradigm, called nonuniform learnability and we will prove is strictly weaker than APAC. A suitable procedure to assign a codification to certain hypothesis classes and how to learn using that codification, as well as a comparison of the studied paradigms, will also be explained.

INTRODUCTION TO THE IMBALANCE PROBLEM

In order to study some machine learning algorithms, we will weaken our concept of learning until we reach a framework in which we can easily explain one of data science most studied problems: imbalanced classification. We will describe the suitable approaches to solve that problem: undersampling and oversampling with and without filtering of instances, and cost-sensitive framework. Since most of current algorithms tend to understimate the importance of the minority examples, we should provide methods to measure the quality of an algorithm based not only on the total accuracy over the training set, but in the level of success over the minority class.

Our main focus will be on oversampling, the creation of synthetic samples belonging to the minority class. We will provide a description of a classic method: SMOTE. Although it was first described 15 years ago, it is still a reference in the treatment of imbalanced datasets, and a lot of other algorithms have arised by means of modification of the former one, such as MWMOTE.

DESCRIPTION OF THE CODED ALGORITHMS

We will analyze a series of algorithms that are not currently present in any package of R programming language. All of them provide an oversampling scheme, except for NEATER. A brief background theory for those algorithms will be provided. We will start with MWMOTE, which tries to fix some of the issues of SMOTE algorithm, such as creation of new instances using noisy ones, or creation of instances between two different minority class clusters.

RACOG will also be addresed, as an algorithm that approximates discrete distributions using marginal pairwise ones and information theory. Once the distribution has been approximated, it will extract samples following a Monte Carlo scheme called Gibbs Sampling. wRACOG will be a modification of RACOG that aims to improve classification under a certain wrapper method provided as parameter.

The next algorithm we will code is RWO. RWO finds its foundation on a statistics theorem that we will prove, and which guarantees us that under asymptotic and generation scheme assumptions, our newly generated instances will preserve the original mean and variance of the training data set.

PDFOS will be another of the selected algorithms. It is based on multivariate Gaussian kernel density estimations. We will provide an introduction to the kernel density estimators pointing out their relationship with histograms and we will give a measure of the error of the estimation, the Mean Integrated Squared Error. PDFOS will try to adapt the bandwith parameter for a sum of multivariate Gaussian functions, that is, a multiplicative constant for the unbiased covariance of the minority training samples. A gradient descendent method will be used to optimize such parameter.

The last algorithm we are going to study is NEATER, a filtering algorithm to clean up noisy instances created as a result of oversampling. NEATER is based on game theory and Nash profile equilibriums, which guarantee that if an instance could play two possible roles, it will tend to stabilize as one of them, given a series of payoffs based on their neighbours' role.

DEVELOPED SOFTWARE

An implementation for those algorithms will be made using the R programming language combined with C++ chunks. R language provides a tradeoff between ease of use for the users and speed of code. C++

will be used to speed up our algorithms when possible. Thus, we will develop a package, called `imbalance`, to fill the lack of implementation of the described algorithms in R.

It is important to note not only the code, but the methodology used to develop the code: under a GPLv2 or later license, allocated in a public Github repo with continuous integration to provide unit testing and periodic updates of its online web documentation page.

EXPERIMENTS ON SMALL DISJUNCTS

Finally, we will conduct a simple experiment on small disjuncts problem, which originate as a direct result of class imbalance problem. The experiment will somehow show that sometimes, when we overcome the imbalance of the datasets, small disjuncts tend to vanish.

We propose a simple measure of the small disjuncts which a dataset has: given a tree classifier, we will consider a small disjunct all those leaves which label less than a given threshold number of examples. We will also measure the mean size of the leaves in terms of examples classified, that is, the mean coverage.

KEYWORDS Machine learning, PAC learning, imbalanced classification, oversampling, small disjuncts

Yo, **Ignacio Cordón Castillo**, alumno de la titulación Doble Grado en Ingeniería Informática y Matemáticas de la **Facultad de Ciencias** y de la **Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones** de la **Universidad de Granada**, con DNI XXXXXXXXX, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca de ambos centros para que pueda ser consultada.

Granada, septiembre de 2017

Ignacio Cordón Castillo

D. **Salvador García López** y D. **Francisco Herrera Triguero**, profesores del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado **Aprendizaje PAC. Clasificación no balanceada: particularización a *small disjuncts***, ha sido realizado bajo su supervisión por **Ignacio Cordón Castillo** y se autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, se expide el presente informe, en Granada, a septiembre de 2017.

Salvador García López Francisco Herrera Triguero

AGRADECIMIENTOS

Me gustaría agradecer a mis tutores, en especial a Salvador García, el apoyo y la ayuda recibida durante todos estos meses, y el hecho de haberme proporcionado total libertad en la elección de muchas de las temáticas que en este trabajo se plasman.

También quiero agradecer a mis compañeros de clase, particularmente a los de LibreIM, por el excepcional ambiente durante todos estos años de carrera, así como por su compañerismo y su predisposición a enseñanzar a los demás.

Por último, gracias a mi familia, porque sin su apoyo no habría llegado hasta aquí.

ÍNDICE GENERAL

1 INTRODUCCIÓN

El aprendizaje automático o ciencia de datos ha sido un área que en los últimos años, con el desarrollo de máquinas cada vez más potentes, y ante la enorme acumulación de datos en prácticamente todos los ámbitos de la sociedad (redes sociales, venta de productos, información de pacientes, ...) ha sufrido un importante auge.

1.1 MOTIVACIÓN

El objetivo del aprendizaje automático es convertir datos en conocimiento a través de un razonamiento inductivo, de manera que proporcionándole información a una máquina seamos capaces de extraer un conocimiento (una generalización de los datos que nos permita inferir información). Surge la pregunta de por qué es necesario el aprendizaje automático o *machine learning*, si la estadística también se encarga de obtener conocimiento a partir de datos.

1.1.1 *¿Por qué necesitamos machine learning?*

i Para resolver **tareas que requieren automatización**. Entran dentro de esta categoría tanto aquellas tareas para las que no existe una axiomatización o un conocimiento exacto, como pueden ser el reconocimiento de dígitos o de voz, como aquellas tareas que requieren del análisis de un gran número de datos, y quedan fuera de la capacidad humana para realizar un análisis estadístico manual. En el primer caso necesitamos apoyarnos en conocimiento auxiliar (por ejemplo, un conjunto de dígitos o de muestras de voz preetiquetadas con los que poder comparar muestras sin etiquetar/clasificar); en el segundo, se hace necsario el uso de una máquina para poder extraer conocimiento de todos los datos.

ii **Tareas que requieren adaptatividad**. Si cambian los datos de entrada, necesitamos que los algoritmos se readapten a ellos, y no tengamos un conocimiento rígido, sino que pueda variar/mejorar en función de la entrada.

1.1.2 *Áreas relacionadas con el aprendizaje*

Entre las áreas relacionadas con el aprendizaje automático, cabe mencionar:

i **Inteligencia Artificial**. Aprender de los datos, cuando tenemos gran cantidad de ellos, es una tarea que claramente encaja con el uso de los ordenadores para ejecutar algoritmos que a un humano le sería imposible ejecutar a mano, o para extraer relaciones sutiles entre los datos que al mismo se le escaparían con facilidad.

ii **Inferencia**. Entre las diferencias que podemos mencionar con la estadística convencional, destaca la necesidad de programar las tareas, dado el volumen de datos con el que normalmente se trabaja, mientras que en muchos análisis estadísticos basta lápiz y papel. También destaca la **independencia respecto a distribución** con la que se trabaja (no se asume una distribución determinada sobre los datos). La principal diferencia del aprendizaje automático respecto a la inferencia es que la inferencia se encarga de comprobar la validez de las hipótesis que propone el estadista, mientras que el algoritmo de *machine learning* genera hipótesis para unos datos determinados, con unas ciertas condiciones de aproximación y error.

iii **Teoría de la medida**. Usaremos nociones de probabilidad y teoría de la medida para construir nuestra noción de aprendizaje: con cierta probabilidad, el error cometido deberá ser pequeño. Asumiremos conocimiento previo de variables aleatorias, y de resultados fundamentales del grado en matemáticas como el teorema de convergencia monótona.

iv **Algorítmica y optimización de algoritmos**. Debemos analizar el tiempo asintótico de los algoritmos que deben ser ejecutados sobre una máquina. Se busca que sean lo más eficientes posibles. Se ha intentado que la implementaciones realizadas tengan el menor orden de eficiencia posible, y dada la lentitud de `R` en los bucles, se han extraído algunas partes clave de dichos algoritmos para ejecutarlas sobre `C++`.

Nuestro estudio de informática estará centrado en el desbalanceo de clases (dos clases, una clase de los datos de entrenamiento más abundante que la otra), por lo que construiremos los cimientos de la teoría PAC sobre el paradigma binario. El máximo exponente de esta teoría fue Leslie Valiant, en los años 80, lo que le grangeó el más prestigioso premio a nivel de informática teórica existente hoy día: el Premio Turing, equivalente a la medalla Fields en matemáticas. Antes, en los

años 60, Vladimir Vapnik y Alexey Chervonenkis dieron una definición de aprendizaje basado en combinatoria y en los años 30, Valery Ivanovich Glivenko y Francesco Paolo Cantelli habían proporcionado una noción de aprendizaje estadístico uniforme. Valiant relacionó todos esos paradigmas entre sí.

Se ha intentado adaptar la teoría PAC de teoría computacional lo máximo posible a un *framework* matemático, proporcionando una introducción a los problemas de medibilidad de los conjuntos cuando ha sido posible. El presente trabajo constituye una revisión del trabajo existente, así como un intento de mejora, adaptación y simplificación de las demostraciones y definiciones.

Se conduce el trabajo desde la dura formalización de aprendizaje PAC, pasando por la contextualización del problema de desbalanceo de clases, hasta llegar al estudio de *small disjuncts*, un problema puramente empírico. El desarrollo informático se centra en proveer de algoritmos recientes en el campo del *oversampling*, técnica usada para paliar el desbalanceo de clases, cuya implementación no había sido liberada por sus autores.

Fundamentalmente nuestra memoria está dividida en:

- **Parte de matemáticas**:
  - Nociones básicas de probabilidad, así como desigualdades fundamentales.
  - Introdución de los primeros conceptos de aprendizaje PAC, y relajación posterior que se concreta en la teoría APAC.
  - Noción de aprendizaje uniforme de Glivenko-Cantelli.
  - Teorema de No Free Lunch.
  - Dimensión Vapnik-Chervonenkis y construcción del teorema fundamental del aprendizaje PAC.
  - Introducción al aprendizaje no-uniforme y al aprendizaje descriptivo.
- **Parte de informática**:
  - Presentación del problema de desbalanceado y de las técnicas existentes.
  - Descripción de los algoritmos implementados.
  - Descripción de las tecnologías empleadas y el paquete desarrollado.

  - Experimentación con *small disjuncts*.

- **Conclusiones**: donde se presentan las conclusiones alcanzadas con la realización del proyecto y las vías de trabajo futuro.

- **Apéndice**: que provee de un manual para el manejo del paquete `imbalance`, del lenguaje de estadística computacional `R`.

1.2 BIBLIOGRAFÍA EMPLEADA

El material principal en el que se ha basado la sección de matemáticas ha sido [Shalev-Shwartz and Ben-David, 2014], que proporciona una detallada formalización de los fundamentos matemáticos del Aprendizaje Automático. Puntualizamos además una serie de referencias extras usadas en la elaboración de esta parte:

- La primera parte del capítulo de introducción a teoría de la probabilidad ha sido confeccionado por el autor de este trabajo a partir de [Probability.net], que ofrece una demostración guiada a través de ejercicios del teorema de extensión de Carathéodory. También se ha usado como apoyo para dicho capítulo el libro [Loève, 1977].

- Para la parte final del primer capítulo de introducción a probabilidad se ha usado [Shalev-Shwartz and Ben-David, 2014], pero mayoritariamente referencias de Wikipedia [Wikipedia, 2017a,b,c].

- Para el teorema 7.3.3 sobre el teorema fundamental PAC se ha usado principalmente el material de clase proporcionado por [Fetaya, 2016], en su docencia en *Weizmann Institue of Science*.

Para la sección de informática se ha empleado:

- En el capítulo introductorio al desbalanceo, [He and Garcia, 2009]

- Cada uno de los algoritmos implementados corresponde al material descrito en un paper de revista científica: MWMOTE [Barua et al., 2014]; RACOG y wRACOG [Das et al., 2015]; RWO [Zhang and Li, 2014]; PDFOS [Gao et al., 2014] y NEATER [Almogahed and Kakadiaris, 2014]. Para la introducción a PDFOS se ha usado además material adaptado desde el libro [Silverman, 1986].

Para el desarrollo del software se ha seguido fundamentalmente [Wickham, 2015], que proporciona una guía detallada de cómo construir

correctamente un paquete del lenguaje R. También se han consultado capítulos aislados de [Gillespie and Lovelace, 2017], que ofrece información muy útil sobre cómo hacer software eficiente en R.

1.3 CONOCIMIENTO LIBRE

El contenido de este trabajo, incluyendo las imágenes no específicamente marcadas como extraídas de fuentes externas, ha sido publicado bajo una licencia Creative Commons BY-NC-SA en un repositorio de Github [1]. El código del fichero en LaTeXqueda liberado bajo una licencia GPLv3.

El software, disponible en otro repositorio de Github [2], se libera bajo una licencia GPLv2 o posterior.

1 https://github.com/ncordon/tfg
2 https://github.com/ncordon/imbalance

2

OBJETIVOS

Los objetivos que se marcaron en la propuesta inicial fueron:

i Recopilación, estudio bibliográfico y taxonomía de las técnicas existentes para el tratamiento de los *small disjuncts* en clasificación no balanceada.

ii Estudio y formalización matemática del problema utilizando una reprensetación basada en grafos.

iii Implementación, adaptación y evaluación de los modelos desarrollados.

Inicialmente el problema estaba enteramente enfocado al estudio de *small disjuncts*, pero dada la baja formalización que admite el problema, y su alta dependencia del concepto de árboles y reglas, se optó por tomar una aproximación al problema más orientada al desbalanceo de clases. Para ello se decidió profundizar en los cimientos teóricos del *machine learning*, donde ésta tarea constituiría la sección de matemáticas y el estudio del problema del desbalanceo se destinaría a dotar de contenido la sección de informática.

El primer punto de los objetivos podría considerarse sobradamente cubierto por los temas 9 y 10, donde no sólo se ha revisado literatura perteneciente a la temática, sino que también se ha profundizado en ella, siendo necesario entender los algoritmos para poder implementarlos.

El segundo punto es probablemente el punto al que menos nos hemos adaptado. No ha sido posible una formalización del problema (no se puede formalizar en términos de grafos un concepto de clasificadores de árboles), aunque sí se ha introducido el concepto en el tema 9 y se ha llevado a cabo una sencilla experimentación en el tema 12.

El tercer punto también se puede considerar alcanzado a través de la programación del paquete `imbalance` para `R`.

Sumados a estos objetivos, podríamos añadir en el ámbito de matemáticas dos nuevos que han surgido con la elaboración de dicha parte:

i Comprensión de la teoría de aprendizaje automático a través de una formalización matemática.

ii Estudio, síntesis y mejora de conceptos y definiciones existentes.

En particular, en lo referente al segundo punto, se ha intentado relacionar la teoría existente, claramente tratada como teoría computacional teórica, con la teoría de la medida. Para ilustrar este hecho, el tema 3 responde a la simple pregunta: dados $A_i \in \Sigma_i, i = 1, \ldots, n$ con (Σ_i, P_i) distribuciones, ¿puedo construir un espacio probabilístico sobre la menor σ-álgebra que contiene a $\Sigma_1 \times \ldots \times \Sigma_n$ y una función de probabilidad P sobre ella, donde $P(A_1 \times \ldots \times A_n)$ sea igual a $\prod_{i=1}^{n} P_i(A_i)$?.

Muchas demostraciones de las aquí presentadas 4.5.1, 4.2.7, 4.2.2, 7.2.2, 8.2.2, la parte de probabilidad del tema 3, ... son trabajo del autor del presente trabajo. Algunas estaban propuestas como ejercicios en las correspondientes referencias, otras han ido surgiendo de manera natural a medida que se han ido trabajando los conceptos. Asimismo, se ha ahondado lo máximo posible la definición de aprendizaje PAC 4.3.1, incluyendo el concepto de factibilidad a través de equivalencia de clases en el sentido de medida $\lfloor H \rfloor_{\mathcal{D}}$, concepto que en su referencia original era bastante más difuso, o pidiéndole a las distribuciones que contengan los conjuntos $[f \neq h]$ para que los errores sean medibles.

De las asignaturas del plan de estudios más relacionadas con el presente trabajo, cabría citar: Fundamentos y Metodología de la Programación, Algorítmica, Análisis ii (teoría de la medida), Probabilidad, Inferencia Estadística, Inteligencia de Negocio (nociones previas de ciencia de datos) o Metaheurísticas (en cosas puntuales como por ejemplo el algoritmo 12 de gradiente descendente).

Parte I

MATEMÁTICAS

3

TEORÍA DE LA PROBABILIDAD

En este capítulo introducimos el espacio de probabilidad y el espacio de probabilidad producto. El teorema de extensión de Caratheodory nos asegurará que siempre es posible definir dicho espacio de probabilidad producto. También ofrecemos resultados fundamentales sobre medidas de concentración probabilísticas, que usaremos más adelante.

3.1 σ ÁLGEBRAS, ANILLOS Y SEMIANILLOS

Sea en lo que sigue X un conjunto. Dados $A_i \subseteq X$, para $i = 1, \dots n$, notamos $\sum_{i=1}^{n} A_i$ a su unión disjunta, esto es $\bigcup_{i=1}^{n} A_i$ con $A_i \cap A_j = \emptyset$ cuando $i \neq j$.

Definición 3.1.1 (Conjunto potencia). *Definimos el conjunto potencia de X como $\mathcal{P}(X) := \{A : A \subseteq X\}$.*

Como notación para $\mathcal{P}(X)$ usaremos frecuentemente 2^X.

Definición 3.1.2 (σ-álgebra). *$\Sigma \subseteq 2^X$ es σ-álgebra de conjuntos sobre X si se verifica:*

i $X \in \Sigma$

ii *Es cerrada para complementarios: sea $A \in \Sigma$, entonces $A^c = X \setminus A \in \Sigma$*

iii *Σ es cerrada para uniones numerables: sean $\{A_n\}_{n \in \mathbb{N}} \subseteq \Sigma$, entonces:*

$$\bigcup_{n \geq 1} A_n \in \Sigma$$

Proposición 3.1.3. *Sea Σ σ-álgebra sobre X. Entonces:*

i $\emptyset \in \Sigma$

ii *Σ es cerrada para intersecciones: dados $A, B \in \Sigma$, entonces $A \cap B \in \Sigma$*

iii *Σ es cerrada para diferencias: dados $A, B \in \Sigma$, entonces $A \setminus B \in \Sigma$*

Demostración. Escribimos $\emptyset = X^c$, $A \cap B = (A^c \cup B^c)^c$ y $A \setminus B = A \cap B^c$ y usando las hipótesis de σ-álgebra se deduce fácilmente la proposición. □

Definición 3.1.4. *Sea $\mathcal{A} \subseteq 2^X$. Llamamos σ-álgebra generada por $\mathcal{A}$ y lo notamos $\Sigma(\mathcal{A})$, a la menor σ-álgebra sobre X que contiene a $\mathcal{A}$.*

$$\Sigma(\mathcal{A}) = \bigcap_{\substack{\Sigma,\, \sigma\text{-álgebra en } X \\ \mathcal{A} \subseteq \Sigma}} \Sigma$$

Definición 3.1.5 (Semianillo en X). *$S \subseteq 2^X$ es semianillo si verifica:*

i $\emptyset \in S$

ii $A, B \in S$ *entonces* $A \cap B \in S$

iii $A, B \in S$ *entonces existen* $A_1, \ldots, A_n \in S$ *verificándose* $A \setminus B = \sum_{i=1}^{n} A_i$

Damos un ejemplo de semianillo:

Ejemplo:

Definición. *Sean X_1, X_2 conjuntos, Σ_i σ-álgebra sobre X_i.*

Dados $A_1 \in \Sigma_1, A_2 \in \Sigma_2$ arbitrarios, definimos el rectángulo de lados A_1 y A_2 como:

$$Rec(A_1, A_2) = A_1 \times A_2$$

La clase de rectángulos $Rec = \{Rec(A_1, A_2) : A_i \in \Sigma_i\}$ es un semianillo en $X_1 \times X_2$, ya que dados $R_1 = A_1 \times A_2 \in Rec, R_2 = B_1 \times B_2 \in Rec$ arbitrarios:

1. $\emptyset \in \Sigma_i$, y $\emptyset \times \emptyset = \emptyset$
2. $R_1 \cap R_2 = (A_1 \cap B_1) \times (A_2 \cap B_2)$, donde $A_i \cap B_i \in \Sigma_i$ por ser Σ_i cerrada bajo intersecciones.
3. $R_1 \setminus R_2 = \{(x, y) : (x, y) \in A_1 \times A_2, (x, y) \notin B_1 \times B_2)\}$

   Es decir $R_1 \setminus R_2 = Rec(A_1 \setminus B_1, A_2) \cup Rec(B_1, A_2 \setminus B_2)$. Además $A_i \setminus B_i \in \Sigma_i$ por ser Σ_i cerrada bajo diferencias.

Definición 3.1.6 (Anillo en X). *$R \subseteq 2^X$ es anillo si verifica:*

i $\emptyset \in R$

ii $A, B \in R$ *entonces* $A \cup B \in R$

iii $A, B \in R$ *entonces* $A \setminus B \in R$

Proposición 3.1.7. *Toda σ-álgebra es anillo. Todo anillo es semianillo.*

Demostración. Que σ-álgebra es más fuerte que anillo quedó probado en la proposición 3.1.3.

Sea ahora $R \subseteq X$ anillo y veamos que es semianillo.

Como $A \cap B = A \setminus (A \setminus B)$, se deduce la segunda condición de la definición de semianillo.

Sean $A, B \in R$, tomando $A_1 = A \setminus B \in R$, $A_n = \emptyset \in R$ para todo $n > 1$, entonces se verifica la tercera condición de semianillo. □

> **Contraejemplo:**
>
> Veamos un contraejemplo de que no todo semianillo es anillo:
>
> $\Sigma = \{\emptyset, [0,1], [1,2], [0,2]\}$ es σ-álgebra sobre $[0,2]$, y $[0,1]^2$ y $[1,2]^2$ son rectángulos en $[0,2]^2$, pero $[0,1]^2 \cup [1,2]^2$ no puede ser escrito como producto de $A, B \in \Sigma$.
>
> Y uno de que no todo anillo es σ-álgebra:
>
> Sea $S = \{A \subseteq \mathbb{N} : |A| < \infty\}$. Entonces se puede comprobar fácilmente que es anillo (y semianillo) de $\mathbb{N}$, pero no es cerrado para complementarios porque $\mathbb{N} = \emptyset^c$ no es finito.

Definición 3.1.8. *Sea $\mathcal{A} \subseteq 2^X$. Llamamos anillo generado por $\mathcal{A}$ y lo notamos $(\mathcal{A})$ al menor anillo que contiene a $\mathcal{A}$:*

$$(\mathcal{A}) = \bigcap_{\substack{R \text{ anillo en } X \\ \mathcal{A} \subseteq R}} R$$

Es trivial probar que la definición es correcta ($(\mathcal{A})$ es anillo), ya que dados dos conjuntos $B, C \in (\mathcal{A})$, entonces $A, B \in R$ para todo R anillo conteniendo a A. Además la intersección es no vacía, porque $2^X \supseteq \mathcal{A}$ y 2^X es anillo.

Proposición 3.1.9. *Sea S un semianillo en X. Entonces:*

$$(S) = \left\{ \sum_{i=1}^{n} A_i : n \geq 1, A_i \in S \right\}$$

Demostración. Sea $R = \{\sum_{i=1}^{n} A_i : n \geq 1, A_i \in S\}$. Es claro que $S \subseteq R$.

Sean $A = \sum_{i=1}^{n} A_i, B = \sum_{j=1}^{p} B_j \in R$ no nulos. Entonces $A \cap B = \sum_{i,j}(A_i \cap B_j) \in R$, por ser $A_i \cap B_j \in S$. Por tanto R es cerrado para intersección de elementos. Además:

$$A \setminus B = \cup_{i=1}^{n} A_i \bigcap \cap_{j=1}^{p} B_j^c = \bigcap_{j=1}^{p} \left(\sum_{i=1}^{n} (A_i \cap B_j^c) \right) = \bigcap_{j=1}^{p} \left(\sum_{i=1}^{n} \sum_{k=1}^{k(i,j)} C_{ij,k(i,j)} \right)$$

donde se ha aplicado que si $A_i, B_j \in S$ existen $C_{ij,1}, \ldots, C_{ij,k(i,j)} \in S$ verificando $A \setminus B_j = \sum_{k=1}^{k(i,j)} C_{ij,k(i,j)}$. Luego aplicando que R es cerrado para intersecciones, llegamos también a que lo es para diferencias.

Por último, como $A \cup B = A \setminus B + B$, entonces $A \cup B \in R$, y R es anillo, con $(S) \subseteq R$. Por otro lado, (S) debe contener por definición de anillo las uniones de elementos de S, en particular $R \subseteq (S)$. Luego $(S) = R$. □

Proposición 3.1.10. *Sea S un semianillo en X. Entonces:*

$$\Sigma(S) = \Sigma((S))$$

Demostración. $S \subseteq (S)$, luego $\Sigma(S) \subseteq \Sigma((S))$

Por otro lado, $\sum_{i=1}^{n} A_i \in \Sigma(S)$ para $A_i \in S$, luego $(S) \subseteq \Sigma(S)$, y por tanto deducimos $\Sigma((S)) \subseteq \Sigma(S)$. □

3.2 MEDIDAS Y EXTENSIÓN DE MEDIDAS

Definición 3.2.1 (Medida). *Sea $\mathcal{A} \subseteq 2^X$. Llamamos medida sobre $\mathcal{A}$ a cualquier función $\mu : \mathcal{A} \to [0, +\infty]$ verificando:*

i $\mu(\emptyset) = 0$

ii *μ es σ-aditiva: dados $A_n \in \mathcal{A}$ disjuntos tales que $\sum_{i=1}^{n} A_i \in \mathcal{A}$, entonces*

$$\mu\left(\sum_{i=n}^{+\infty} A_n \right) = \sum_{i=n}^{+\infty} \mu(A_n)$$

Nótese que la σ-aditividad implica aditividad finita, esto es, dados $A_i \in \mathcal{A}, i = 1, \ldots, n$, disjuntos, tales que $\sum_{i=1}^{n} A_i \in \mathcal{A}$, entonces $\mu\left(\sum_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} \mu(A_i)$.

Teorema 3.2.2. *Sea S semianillo en X, $\mu : S \to [0, +\infty]$ medida sobre S. Entonces existe una única medida sobre (S), $\bar{\mu} : (S) \to [0, +\infty]$ verificando $\bar{\mu}_{|S} = \mu$.*

Demostración. Recordamos que $(S) = \{\sum_{i=1}^{n} A_i : n \geq 1, A_i \in S\}$ por la proposición 3.1.9.

Es claro que $\bar{\mu}$ debería cumplir: $\bar{\mu}\left(\sum_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} \bar{\mu}(A_i) = \sum_{i=1}^{n} \mu(A_i)$ con $A_i \in S$. Por tanto, dado $A \in (S)$, tomamos $A_i \in S$ verificando $A = \sum_{i=1}^{n} A_i$, y definimos:

$$\bar{\mu}(A) := \sum_{i=1}^{n} \mu(A_i)$$

Veamos que $\bar{\mu}$ está bien definida, esto es, dado $\sum_{i=1}^{n} A_i = \sum_{j=1}^{k} B_j = A$, veamos que se verifica $\sum_{i=1}^{n} \mu(A_i) = \sum_{j=1}^{k} \mu(B_j)$. Como $A_i = A_i \cap A = A_i \cap \left(\cup_{j=1}^{k} B_j\right) = \bigcup_{j=1}^{k} A_i \cap B_j$, donde $A_i \cap B_j \in S$ por ser S semianillo. Además como los A_i son disjuntos, los $A_i \cap B_j$ también. Aplicando que μ es medida sobre S:

$$\mu(A_i) = \sum_{j=1}^{k} \mu(A_i \cap B_j) \implies \sum_{i=1}^{n} \mu(A_i) = \sum_{i=1}^{n} \sum_{j=1}^{k} \mu(A_i \cap B_j)$$

Análogamente podemos probar:

$$\sum_{j=1}^{k} \mu(B_j) = \sum_{i=1}^{n} \sum_{j=1}^{k} \mu(A_i \cap B_j)$$

Luego $\bar{\mu}$ está bien definida. Además $\bar{\mu}(\emptyset) = 0$ por ser $\mu(\emptyset) = 0$. Falta probar la σ-aditividad de $\bar{\mu}$ y por construcción, habríamos llegado al resultado buscado.

Sean $A_n = \sum_{i=1}^{k_n} A_i^n \in (S)$ disjuntos tal que $A_i^n \in S$ y $A = \sum_{n=1}^{+\infty} A_n \in (S)$. Por ser $A \in (S)$, tendríamos que podemos reescribir $A = \sum_{j=1}^{k} B_j$ con $B_j \in S$. Fijamos un B_p y tenemos:

$$B_p = A \cap B_p = \bigcup_{n=1}^{+\infty} \left(\bigcup_{i=1}^{k_n} A_i^n \cap B_p \right) = \bigcup_{n=1}^{+\infty} \bigcup_{i=1}^{k_n} \left(A_i^n \cap B_p\right)$$

Es decir, hemos reescrito B_p como una unión numerable disjunta de $A_i^n \cap B_p \in S$.

Por σ-aditividad en S deducimos $\mu(B_p) = \sum_{n=1}^{+\infty} \sum_{i=1}^{k_n} \mu(A_i^n \cap B_p)$. Además, como podemos reescribir $A_i^n = \sum_{j=1}^{k} A_i^n \cap B_j$, tendríamos:

$$\begin{aligned} \mu(A) &= \sum_{j=1}^{k} \sum_{n=1}^{+\infty} \sum_{i=1}^{k_n} \mu(A_i^n \cap B_j) = \sum_{n=1}^{+\infty} \sum_{i=1}^{k_n} \sum_{j=1}^{k} \mu(A_i^n \cap B_j) \\ &= \sum_{n=1}^{+\infty} \sum_{i=1}^{k_n} \mu(A_i^n) = \sum_{n=1}^{+\infty} \mu\left(\sum_{i=1}^{k_n} A_i^n \right) = \sum_{n=1}^{+\infty} \mu(A_n) \end{aligned}$$

□

Definición 3.2.3 (Medida exterior). *Sea $\mu^* : 2^X \to [0, +\infty]$ verificando:*

i $\mu^*(\emptyset) = 0$

ii Monotonía: $A \subseteq B$ entonces $\mu^*(A) \leq \mu^*(B)$

iii σ-subaditividad: $\mu^*\left(\bigcup_{n=1}^{+\infty} A_n\right) \leq \sum_{n=1}^{+\infty} \mu^*(A_n)$

Entonces μ^* se dice medida exterior sobre X.

Definición 3.2.4. *Sea μ^* una medida exterior sobre X. Se define la σ-álgebra asociada a μ^* como:*

$$\Sigma(\mu^*) = \{A \subseteq X : \mu^*(T) = \mu^*(T \cap A) + \mu^*(T \cap A^c), \forall T \subseteq X\}$$

Queda comprobar $\Sigma(\mu^*)$ es efectivamente una σ-álgebra.

Teorema 3.2.5. *Sea $\mu^* : 2^X \to [0, +\infty]$ una medida exterior sobre X. Entonces $\Sigma(\mu^*)$ es una σ-álgebra sobre X y $\mu^*_{|\Sigma(\mu^*)}$ es una medida sobre $\Sigma(\mu^*)$*

Demostración. Empezamos viendo que $\Sigma = \Sigma(\mu^*)$ es una σ-álgebra.

$$\mu^*(T) = \mu^*(T \cap X) + \mu^*(T \cap \emptyset) = \mu^*(T), \quad \forall T \subseteq X$$

Luego $X, \emptyset \in \Sigma$. También es trivial ver que dado $A \in \Sigma$, entonces $A^c \in \Sigma$.

Veamos que Σ es cerrada para intersecciones. Sean $A, B \in \Sigma$. Usaremos:

$$\begin{aligned} T \cap A^c &= T \cap (A \cap B)^c \cap A^c \\ T \cap A \cap B^c &= T \cap (A \cap B)^c \cap A \end{aligned} \qquad (*)$$

Como $A \in \Sigma$, tenemos $\mu^*(T) = \mu^*(T \cap A) + \mu^*(T \cap A^c)$. Por otro lado, como $B \in \Sigma$, tenemos $\mu^*(T \cap A) = \mu^*(T \cap A \cap B) + \mu^*(T \cap A \cap B^c)$. Es decir:

$$\begin{aligned} \mu^*(T) &= \mu^*(T \cap A^c) + \mu^*(T \cap A \cap B) + \mu^*(T \cap A \cap B^c) \\ &\underset{(*)}{=} \mu^*(T \cap (A \cap B)^c \cap A^c) + \mu^*(T \cap (A \cap B)^c \cap A) \\ &\quad + \mu^*(T \cap A \cap B) = \mu^*(T \cap (A \cap B)^c) + \mu^*(T \cap A \cap B) \end{aligned}$$

donde en la última igualdad se ha usado que $A \in \Sigma$.

Como podemos escribir $A \setminus B = A \cap B^c$, y $A \cup B = (A^c \cap B^c)^c$ con $A, B \in \Sigma$ que hemos probado que es cerrado para intersecciones y complementarios, Σ es también cerrado para uniones y diferencias.

Dados $A_n \in \Sigma$, como podemos escribir $\bigcup_{n\geq 1} A_n = \sum_{n=1}^{+\infty} B_n$ con los B_n disjuntos, definidos como $B_1 = A_1$, y $B_{n+1} = A_{n+1} \setminus \left(\cup_{i=1}^{n} A_i\right)$, podemos limitarnos a estudiar uniones disjuntas.

De hecho, dados $B, C \in \Sigma$ disjuntos:

$$\begin{aligned}\mu^*(T \cap (B \cup C)) &= \mu^*(T \cap (B \cup C) \cap B) + \mu^*(T \cap (B \cup C) \cap B^c) \\ &= \mu^*(T \cap B) + \mu^*(T \cap C)\end{aligned}$$

Sean B_n conjuntos disjuntos de Σ, y llamamos $B = \sum_{n=1}^{+\infty} B_n$, $\bar{B}_k = \sum_{n=1}^{k} B_n$. Hemos visto que Σ es cerrada para uniones finitas, luego $\bar{B}_k \in \Sigma$ para todo $k \in \mathbb{N}$. Vamos a ver que también lo es para uniones numerables. Sea $T \in 2^X$.

Si $\mu^*(T) = +\infty$, entonces $\mu^*(T) \leq \mu^*(T \cap B) + \mu^*(T \cap B^c) = +\infty$, usando monotonía. Se da la igualdad.

Si $\mu^*(T) < +\infty$, tenemos $\mu^*(T) \geq \mu^*(T \cap \bar{B}_k) = \sum_{n=1}^{k} \mu^*(T \cap B_n)$ para k arbitrario, y por tanto $\mu^*(T) \geq \sum_{n=1}^{+\infty} \mu^*(T \cap B_n)$. Fijado $\varepsilon > 0$ existe $M \in \mathbb{N}$ verificando que para todo $N \geq M$:

$$\sum_{n=1}^{+\infty} \mu^*(T \cap B_n) \leq \sum_{n=1}^{N} \mu^*(T \cap B_n) + \varepsilon = \mu^*(T \cap \bar{B}_N) + \varepsilon$$

Es decir, tendríamos:

$$\begin{aligned}\mu^*(T) = \mu^*(T \cap B^c + T \cap B) &\leq \mu^*(T \cap B^c) + \mu^*(T \cap B) \\ &\leq \mu^*(T \cap B^c) + \sum_{n=1}^{+\infty} \mu^*(T \cap B_n) \\ &\leq \mu^*(T \cap \bar{B}_N^c) + \mu^*(T \cap \bar{B}_N) + \varepsilon = \mu^*(T) + \varepsilon\end{aligned}$$

Pero $\varepsilon > 0$ era arbitrario, luego $\mu^*(T) = \mu^*(T \cap B^c) + \mu^*(T \cap B)$ y tomando $T = B$, se deduce también que $\mu^*(B) = \sum_{n=1}^{+\infty} \mu^*(B_n)$.

Es decir, Σ es σ-álgebra. y $\mu|_\Sigma$ es medida sobre Σ. □

Teorema 3.2.6 (Teorema de extensión de Caratheodory). *Sea $S \subseteq 2^X$ un semianillo, $\mu : S \to [0, +\infty]$ medida en S. Entonces existe una medida $\bar{\mu} : \Sigma(S) \to [0, +\infty]$ verificando $\bar{\mu}_{|S} = \mu$*

Demostración. Si existiese $\bar{\mu}$ debería verificar, por monotonía, $\bar{\mu}(T) \leq \sum_{n=1}^{+\infty} \bar{\mu}(A_n)$, donde $T \subseteq \bigcup_{n\geq 1} A_n$. Esto nos da una idea de la prueba.

Por el teorema 3.2.2, podemos considerar μ como la extensión al anillo $R = (S)$.

Definimos, para cualquier $T \in 2^X$:

$$\mu^*(T) := \text{ínf}\left\{\sum_{n=1}^{+\infty}\mu(A_n) : \bigcup_{n\geq 1} A_n \supseteq T, A_n \in R \quad \forall n \in \mathbb{N}\right\}$$

donde por convención adoptamos ínf$\emptyset := +\infty$. Además, como $\mu \geq 0$, entonces $\mu^* \geq 0$, y como $\mu^*(\emptyset) = 0$, entonces $\mu(\emptyset) = 0$. Vamos a ver que μ^* es medida exterior sobre X.

Dado $B \subseteq A \subseteq X$, tenemos que si $\bigcup_{n\geq 1} A_n \supseteq A$ con $A_n \in R$ entonces $\bigcup_{n\geq 1} A_n \supseteq B$ y de la definición de μ^* deducimos $\mu^*(A) \leq \mu^*(B)$. Hemos probado la monotonía de μ^*.

Sean ahora $A_n \subseteq X$, con $n \in \mathbb{N}$.Si existe A_k tal que $\mu^*(A_k) = +\infty$, entonces $\mu^*\left(\bigcup_{n\geq 1} A_n\right) \geq \mu^*(A_k) = +\infty$, por monotonía, y por otro lado $\sum_{n=1}^{+\infty}\mu^*(A_n) \geq \mu^*(A_k) = +\infty$. Es decir:

$$\mu^*\left(\bigcup_{n\geq 1} A_n\right) = \sum_{n=1}^{+\infty}\mu^*(A_n) = +\infty$$

Caso de que para todo n se tenga $\mu^*(A_n) < +\infty$, fijamos $\varepsilon > 0$. Dado $n \in \mathbb{N}$, por ser $\mu^*(A_n)$ un ínfimo, debe existir $\{A_p^n\}_{p\geq 1} \subseteq R$ verificando $\cup_{p\geq 1} A_p^n \supseteq A_n$ y $\sum_{p=1}^{+\infty}\mu^*(A_p^n) \leq \mu^*(A_n) + \frac{\varepsilon}{2^n}$.

Así, tenemos $\bigcup_{n\geq 1} A_n \subseteq \bigcup_{n\geq 1}\cup_{p\geq 1} A_p^n$ unión numerable de conjuntos A_p^n de S, luego:

$$\mu^*\left(\bigcup_{n\geq 1} A_n\right) \leq \sum_{n=1}^{+\infty}\sum_{p=1}^{+\infty}\mu^*(A_p^n) \leq \sum_{n=1}^{+\infty}\mu^*(A_n) + \sum_{n=1}^{+\infty}\frac{\varepsilon}{2^n} = \sum_{n=1}^{+\infty}\mu^*(A_n) + \varepsilon$$

Pero como $\varepsilon > 0$ era arbitrario:

$$\mu^*\left(\bigcup_{n\geq 1} A_n\right) \leq \sum_{n=1}^{+\infty}\mu^*(A_n)$$

Esto prueba la subaditividad, por lo que μ^* es medida exterior en X, y podemos tomar $\widetilde{\mu}$ medida sobre $\Sigma(\mu^*)$ por el teorema 3.2.5. Falta comprobar que $\bar{\mu} = \widetilde{\mu}_{|\Sigma(S)}$ es medida sobre $\Sigma(S)$. Basta ver que $S \subseteq \Sigma(\mu^*)$, y tendríamos $\Sigma(S) \subseteq \Sigma(\mu^*)$.

Fijamos $A \in R$. Se cumple que para todo $T \in 2^X$ arbitrario, por subaditividad:

$$\mu^*(T) \leq \mu^*(T \cap A) + \mu^*(T \cap A^c)$$

Sea ahora $T_n \in R$ con $T \subseteq \bigcup_{n\geq 1} T_n$. Entonces $T \cap A \subseteq \bigcup_{n\geq 1} T_n \cap A$ y $T \cap A \subseteq \bigcup_{n\geq 1} T_n \cap A^c$ con $A \cap T_n, A^c \cap T_n \in R$. Nótese que aquí estamos usando explícitamente que R es anillo. Así:

$$\begin{aligned}\mu^*(T) &\leq \mu^*(T \cap A) + \mu^*(T \cap A^c) \\ &\leq \sum_{n=1}^{+\infty} \mu(T_n \cap A) + \sum_{n=1}^{+\infty} \mu(T_n \cap A^c) \\ &= \sum_{n=1}^{+\infty} \mu(T_n \cap A) + \mu(T_n \cap A^c) = \sum_{n=1}^{+\infty} \mu(T_n)\end{aligned}$$

Fijado $\varepsilon > 0$ podemos buscar $T_n \in R$ verificando $\sum_{n=1}^{+\infty} \mu(T_n) \leq \mu^*(T) + \varepsilon$, por estar definido μ^* como un ínfimo. Por ser $\varepsilon > 0$ arbitrario, hemos llegado a:

$$\mu^*(T) = \mu^*(T \cap A) + \mu^*(T \cap A^c) \Rightarrow A \in \Sigma(\mu^*)$$

□

3.3 ESPACIO PROBABILÍSTICO

Definición 3.3.1 (Espacio medible). *Sea X un conjunto, Σ una σ-álgebra sobre X. A la tupla (X, Σ) la llamamos espacio medible. A los elementos de Σ los llamamos conjuntos Σ-medibles.*

Definición 3.3.2 (Espacio de medida). *Sea (X, Σ) espacio medible, y $\mu : \Sigma \to [0, +\infty]$ medida. A la tupla (X, Σ, μ) la llamamos espacio de medida.*

Definición 3.3.3 (Espacio de probabilidad). *Sea (X, Σ, P) espacio de medida. Entonces lo llamamos espacio de probabilidad sii $P(\Sigma) \subseteq [0, 1]$.*

Definición 3.3.4 (Distribución de probabilidad). *Sea (X, Σ, P) espacio de probabilidad. Llamamos a la tupla $\mathcal{D} = (\Sigma, P)$ distribución sobre X. Si $\mathcal{D}$ es distribución sobre X, lo notamos $x \sim \mathcal{D}$*

Definición 3.3.5 (Espacio de probabilidad producto). *Sean (X_i, Σ_i, P_i) con $i = 1, \ldots, n$ espacios de probabilidad. Entonces decimos que:*

$$(X_1 \times \ldots \times X_n, \Sigma_1 \otimes \ldots \otimes \Sigma_n, P)$$

es espacio de probabilidad producto asociado a Σ_i y a P_i, $i = 1, \ldots n$, si verifica:

i $\Sigma_1 \otimes \ldots \otimes \Sigma_n = \Sigma\left(\{A_1 \times \ldots \times A_n : A_i \in \Sigma_i\}\right)$

ii $P(A_1 \times \ldots \times A_n) = \prod_{i=1}^{n} P_i(A_i)$, *con* $A_i \in \Sigma_i$ *arbitrarios.*

$\mathcal{D}_1 \otimes \ldots \otimes \mathcal{D}_n := (\Sigma_1 \otimes \ldots \otimes \Sigma_n, P)$ *recibe el nombre de distribución de probabilidad producto.*

En el caso $\Sigma_1 = \ldots = \Sigma_n$*, notaremos* Σ^n *a la* σ*-álgebra producto.*

El siguiente teorema nos dice que siempre podemos definir un espacio de probabilidad producto.

Teorema 3.3.6. *Sean* (X_i, Σ_i, P_i) *con* $i = 1, \ldots n$ *espacios de probabilidad. Entonces existe un espacio probabilidad producto.*

$$(X_1 \times \ldots \times X_n, \Sigma_1 \otimes \ldots \otimes \Sigma_n, P)$$

Demostración. Sea S la clase de rectángulos de la forma $A_1 \times A_2 \times \ldots \times A_n$ con A_i conjunto Σ_i-medible. S es semianillo. Lo probamos para $n = 2$ en la sección 3.1 (por inducción es extensible a n arbitrario). Haremos la demostración para $n = 2$. Una obvia inducción en n nos da el resultado prara n mayor que 2.

Veamos que $P : S \to [0, +\infty]$ definida como $P(A \times B) = P_1(A)P_2(B)$ es medida sobre S. Claramente $P(\emptyset) = P_1(\emptyset)P_2(\emptyset) = 0$. Falta comprobar que dados $A_n \times B_n \in S$ verificando $\sum_{n=1}^{+\infty} A_n \times B_n \in S$, es decir, $\sum_{n=1}^{+\infty} A_n \times B_n = A \times B$, con $A \in \Sigma_1$, $B \in \Sigma_2$, entonces

$$P(A \times B) = \sum_{n=1}^{+\infty} P_1(A_n)P_2(B_n)$$

Para cualquier $(x, y) \in X_1 \times X_2$, se tiene:

$$\mathbb{1}_{A\times B}(x, y) = \mathbb{1}_A(x) \cdot \mathbb{1}_B(y) = \sum_{n=1}^{+\infty} \mathbb{1}_{A_n \times B_n}(x, y) = \sum_{n=1}^{+\infty} \mathbb{1}_{A_n}(x)\mathbb{1}_{B_n}(y)$$

Fijado $y \in X_2$ $\{\sum_{i=1}^{n} \mathbb{1}_{A_i}\mathbb{1}_{B_i}(y)\}_n \nearrow \mathbb{1}_A\mathbb{1}_B(y)$ puntualmente en X_1. Por el teorema de la convergencia monótona, integrando en X_1:

$$\sum_{n=1}^{+\infty} P_1(A_n)\mathbb{1}_{B_n}(y) = P_1(A)\mathbb{1}_B(y)$$

Como $\{\sum_{i=1}^{n} P_1(A_i)\mathbb{1}_{B_i}\} \nearrow P_1(A)\mathbb{1}_B$ puntualmente en $y \in X_2$, integrando ahora en X_2 y aplicando teorema de convergencia monótona de nuevo, llegamos a:

$$\sum_{n=1}^{+\infty} P_1(A_n)P_2(B_n) = P(A)P(B)$$

En virtud del teorema 3.2.6 de extensión de Caratheodory, como tenemos S semianillo en $X_1 \times X_2$ y P una medida en S, existe el espacio de probabilidad producto. □

Se puede probar que el espacio probabilístico producto sólo hay uno, pero no usaremos este hecho. Obsérvese que cuando tengamos un espacio de probabilidad (Σ, X) y consideramos que podemos extraer m muestras del mismo de manera independiente, estaremos midiendo conjuntos $A = A_1 \times A_2 \times \ldots \times A_n$, con $A_i \in \Sigma$. Para estos sucesos siempre tendremos $\overline{P}(A) = \prod_{i=1}^{n} P(A_i)$ sea cual sea la distribución producto $(\Sigma^n, \overline{P})$, pero conviene tener presente que es única.

3.4 DESIGUALDADES DE CONCENTRACIÓN

Para esta sección supondremos conocimiento previo de variables aleatorias. Sea (X, Σ, P) un espacio de probabilidad en lo que sigue.

Lema 3.4.1 (Desigualdad de Markov). *Sea $W \geq 0$ una variable aleatoria. Entonces para todo $a > 0$ se verifica:*

$$P[W \geq a] \leq \frac{\mathbb{E}(W)}{a}$$

Demostración. Se verifica $a\mathbb{1}_{[W \geq a]} \leq W$. Tomando esperanzas en ambos lados (ya que la esperanza es no decreciente):

$$aP[W \geq a] = a\mathbb{E}(\mathbb{1}_{[W \geq a]}) = \mathbb{E}(a\mathbb{1}_{[W \geq a]}) \leq \mathbb{E}(W)$$

de donde se deduce el resultado, dividiendo por a. □

Lema 3.4.2 (Lema de Hoeffding). *Sea W una variable aleatoria con $a \leq W \leq b$ y $\mathbb{E}(W) = 0$. Entonces para todo $\lambda > 0$:*

$$\mathbb{E}(e^{\lambda W}) \leq e^{\frac{\lambda^2 (b-a)^2}{8}}$$

Demostración. Supongamos $a = 0$ o $b = 0$. Entonces la desigualdad se cumpliría ya que se tendría $P[W = 0] = 1$. Por reducción al absurdo, en el caso $a = 0$ si $P[W \geq 0] = 1$ y si se tuviera $P[W > 0] > 0$ entonces $\mathbb{E}(W) > 0$. El caso $b = 0$ es análogo.

Con $a = 0$ o $b = 0$, tendríamos así $\mathbb{E}(e^{\lambda W}) = \mathbb{E}(1) = 1 = e^0 \leq e^{\frac{\lambda^2 (b-a)^2}{8}}$, por ser la exponencial monótona.

Supongamos pues en lo que sigue $a, b \neq 0$. Deberá ser $a < 0 < b$ por un argumento similar al primer caso.

Por convexidad de $x \mapsto e^{\lambda x}$ con $a \leq x \leq b$, tenemos que:

$$e^{\lambda W} \leq \frac{b - W}{b - a} e^{\lambda a} + \frac{W - a}{b - a} e^{\lambda b}$$

Tomando esperanzas en ambos términos:

$$\mathbb{E}\left(e^{\lambda W}\right) \leq \frac{b-\mathbb{E}(W)}{b-a}e^{\lambda a} + \frac{\mathbb{E}(W)-a}{b-a}e^{\lambda b} = \frac{b}{b-a}e^{\lambda a} - \frac{a}{b-a}e^{\lambda b}$$

Notando $t = \lambda(b-a), s = \frac{-a}{b-a}$, podemos definir $\varphi : \mathbb{R} \to \mathbb{R}$ dada por $\varphi(t) = -st + \log(1-s+se^t)$ y se tiene:

$$e^{\varphi(t)} = \frac{b}{b-a}e^{\lambda a} - \frac{a}{b-a}e^{\lambda b}$$

Basta por tanto probar que $e^{\varphi(t)} \leq e^{t^2/8}$.

Se tiene $1-s+se^t = 1 + \frac{a}{b-a} - \frac{a}{b-a}e^t = \frac{b}{a-b} - \frac{a}{b-a}e^t > 0$, por ser $a < 0 < b$. Luego $\varphi \in \mathcal{C}^\infty$. Asimismo:

$$\begin{aligned}
&\varphi(0) = 0\\
&\varphi'(0) = -s + \frac{se^t}{1-s+se^t}\bigg|_{t=0} = -s + \frac{s}{1-s+s} = 0\\
&\varphi''(t) = \frac{se^t}{1-s+se^t}\left(1 - \frac{se^t}{1-s+se^t}\right) = x(1-x) \leq \frac{1}{4}
\end{aligned}$$

Donde en la última desigualdad hemos usado que el polinomio $x - x^2$ tiene un máximo en $x = \frac{1}{2}$.

Así, tomando $f(t) = \varphi(t) - \frac{1}{8}t^2$, tenemos que $f(0) = 0 = f'(0)$ y $f''(t) \leq 0$. Es decir, $f'(t) \geq 0$ para todo $t < 0$ y $f'(t) \leq 0$ para todo $t > 0$, luego f tiene un máximo en $t = 0$. Por tanto $f \leq 0$ y $\varphi(t) \leq \frac{1}{8}t^2$. □

Lema 3.4.3 (Desigualdad de Hoeffding). *Sean $W_1, \ldots, W_m$ variables aleatorias i.i.d. (independiente e idénticamente distribuidas), tales que $a_i \leq W_i \leq b_i$, y sea $\overline{W} = \frac{1}{m}\sum_{i=1}^m W_i$. Entonces dado $\varepsilon > 0$ arbitrario:*

$$P\left[\left|\overline{W} - \mathbb{E}(\overline{W})\right| \geq \varepsilon\right] \leq 2e^{-2m^2\frac{\varepsilon^2}{\sum_{i=1}^m (b_i-a_i)^2}}$$

En el caso particular $a_i = a$ y $b_i = b$ para todo $i = 1, \ldots, m$:

$$P\left[\left|\overline{W} - \mathbb{E}(\overline{W})\right| \geq \varepsilon\right] \leq 2e^{-2m\left(\frac{\varepsilon}{b-a}\right)^2}$$

Demostración. Llamamos $V_i = W_i - \mathbb{E}(W_i)$. Nótese que $\overline{V} = \overline{W} - \mathbb{E}(W)$ Aplicando monotonía de la función exponencial y la desigualdad de Markov 3.4.1, tenemos que para todo $\varepsilon, \lambda > 0$:

$$P[\overline{V} \geq \varepsilon] = P[e^{\lambda\overline{V}} \geq e^{\lambda\varepsilon}] \leq e^{-\lambda\varepsilon}\mathbb{E}(e^{\lambda\overline{V}}) \qquad (*)$$

Además como las W_i eran independientes, $e^{\lambda V_i}$ lo son, y se tiene:

$$\mathbb{E}(e^{\lambda V}) = \mathbb{E}\left(\prod_{i=1}^{m} e^{\lambda V_i/m}\right) = \prod_{i=1}^{m} \mathbb{E}(e^{\lambda V_i/m})$$

Por otro lado tenemos:

$$\begin{aligned} &\mathbb{E}(V_i) = \mathbb{E}(W_i) - \mathbb{E}(\mathbb{E}(W_i)) = \mathbb{E}(W_i) - \mathbb{E}(W_i) = 0 \\ &a_i - \mathbb{E}(W_i) \leq V_i \leq b_i - \mathbb{E}(W_i) \qquad (**) \end{aligned}$$

Aplicando a cada V_i/m el lema 3.4.2, desde $(*)$ y $(**)$ llegamos a:

$$\mathbb{E}(e^{\lambda V}) \leq e^{-\lambda\varepsilon} \cdot e^{\frac{\lambda^2 \sum_{i=1}^{m}(b_i-a_i)^2}{8m^2}} = e^{\frac{\lambda^2 \sum_{i=1}^{m}(b_i-a_i)^2}{8m^2} - \lambda\varepsilon}$$

y el polinomio $q(\lambda) = \frac{\lambda^2}{8m^2}\sum_{i=1}^{m}(b_i - a_i)^2 - \lambda\varepsilon$ se minimiza en $\lambda = \frac{4m^2\varepsilon}{\sum_{i=1}^{m}(b_i-a_i)^2}$. Deducimos:

$$P[\overline{V} \geq \varepsilon] \leq e^{-2m^2 \frac{\varepsilon^2}{\sum_{i=1}^{n}(b_i-a_i)^2}}$$

Análogamente se probaría:

$$P[-\overline{V} \geq \varepsilon] \leq e^{-2m^2 \frac{\varepsilon^2}{\sum_{i=1}^{n}(b_i-a_i)^2}}$$

y por subaditividad:

$$P[|\overline{V}| \geq \varepsilon] \leq P[\overline{V} \geq \varepsilon] + P[-\overline{V} \geq \varepsilon] \leq 2e^{-2m^2 \frac{\varepsilon^2}{\sum_{i=1}^{n}(b_i-a_i)^2}}$$

□

4

APRENDIZAJE PAC

En este capítulo proporcionamos una introducción y una motivación al *framework* teórico que usaremos. Daremos definiciones básicas como dominio, conjunto de etiquetas, error empírico, etc. que usaremos asiduamente en capítulos posteriores, así como las definiciones de PAC y APAC cognoscibilidad, describiendo la relación entre ambas.

4.1 EJEMPLO PRÁCTICO

Pensemos en un ejemplo: tenemos clientes de un banco que quieren solicitar un préstamo, y a todos se les categoriza el tamaño del préstamo que quieren solicitar (cómo de grande es su importe) y su nivel de ingresos. Estas variables se miden en una escala de 0 a 1 donde 1 es el nivel máximo. Queremos etiquetar a cada cliente como $1 \equiv$ conceder préstamo o $0 \equiv$ no concederlo.

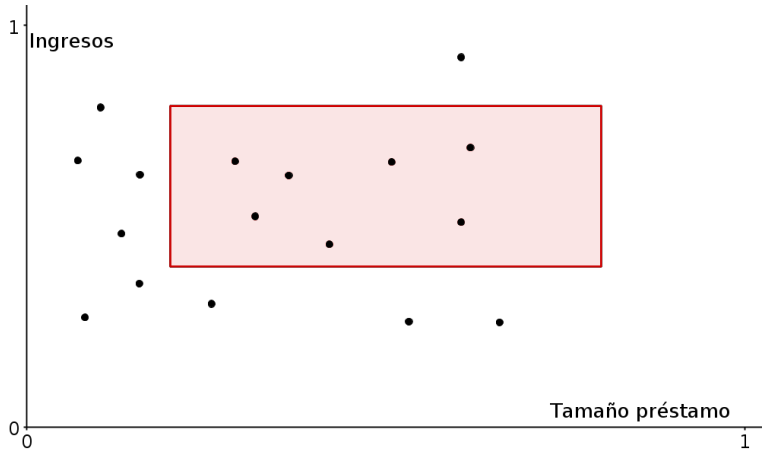
Dado un histórico de $m \in \mathbb{N}$ clientes a los que se les concedieron y devolvieron o no préstamos, tenemos una tupla $((x_1, y_1), \dots (x_m, y_m))$ donde $x_i = (x_i^{(1)}, x_i^{(2)}) \in [0,1]^2, y_i \in \{0,1\}$. Asimismo, $x_i^{(1)}$ representa el tamaño del préstamo solicitado y $x_i^{(2)}$ representa el nivel ingresos mensuales. Tenemos a los clientes clasificados en función de si devolvieron los préstamos o no.

Queremos encontrar una función que ofrezca una predicción sobre cualquier cliente, para minimizar posibles pérdidas del banco, es decir, buscamos una $f : [0,1]^2 \to \{0,1\}$ que llamaremos predicción.

Asumimos que los datos de los clientes de que disponemos no van a tener una distribución determinada y van a ser idéntica e independientemente distribuidos (i.i.d.). El histórico siempre va a crecer, y no sabemos cómo va a hacerlo, queriendo aprovechar al máximo la información del mismo.

También asumiremos que tenemos una clase de predicciones $H \subseteq \{0,1\}^{[0,1]^2}$, de entre las que existe una óptima. Por ejemplo, podríamos limitar la clase de predicciones sobre la que buscamos como subrectángulos de $[0,1]^2$, esto es, funciones:

$$h_{a,b,c,d} = \mathbb{1}_{[a,b]\times[c,d]}, \qquad [a,b] \times [c,d] \subseteq [0,1]^2$$



También necesitaremos definir una medida de acierto: ¿cómo de buena es una predicción?.

4.2 DEFINICIONES BÁSICAS

Damos unas notaciones/definiciones básicas que utilizaremos de aquí en adelante, en base a lo descrito en 4.1:

- **Dominio**: X. Llamamos **instancia** a $x \in X$
- **Conjunto de etiquetas**: Y. Consideramos $Y = \{0, 1\}$, lo que nos restringe de momento al paradigma de clasificación binaria. En ocasiones también usaremos $Y = \{-1, 1\}$ para las etiquetas. Al 1 se le llama clase positiva, y al 0 o -1 clase negativa.
- **Verdadero etiquetado**: Asumimos la existencia de una función $f : X \to Y$ que proporciona la verdadera etiqueta de todas las instancias.
- **Generación de instancias**: Se tiene $x \sim \mathcal{D} = (\Sigma, \underset{x\sim\mathcal{D}}{P})$. La distribución de probabilidad nos da información sobre la probabilidad de extraer cada posible instancia desde $x \in X$.
- **Conjunto de entrenamiento**: $S = ((x_1, y_1), \ldots, (x_m, y_m))$, $S \in (X \times Y)^m$. Nótese que llamarlo conjunto puede dar lugar a confusión, puesto que se trata de una tupla. También notaremos $S_x = (x_1, \ldots x_m)$.

De momento asumiremos que las etiquetas del conjunto de entrenamiento se corresponden con el verdadero etiquetado: $y_i = f(x_i)$, por lo que no podemos tener una instancia con etiquetas diferentes.

La elección de S_x es idéntica e independientemente distribuida, esto es $x_i \sim \mathcal{D}$ para todo $i = 1, \ldots, m$. Lo notamos $S_x \sim \mathcal{D}^m$, o $S \sim \mathcal{D}^m$, por abuso de notación. Por lo visto en el capítulo anterior, en 3.3, dicha probabilidad existe.

- **Hipótesis/clasificador/predicción**: cada posible aplicación perteneciente a $\{h, h : X \to Y\} := 2^X$.

- **Error del clasificador**: Definimos el error del clasificador, suponiendo $\{x \in X : h(x) \neq f(x)\} := [h \neq f] \in \Sigma$ como:

$$L_{\mathcal{D},f}(h) := \underset{x \sim \mathcal{D}}{P}[h \neq f]$$

- **Algoritmo de aprendizaje**: Llamamos algoritmo de aprendizaje a cualquier aplicación que tome conjuntos de entrenamiento y devuelva hipótesis sobre el problema:

$$A : \bigcup_{m \in \mathbb{N}} (X \times Y)^m \to 2^X$$

Asumimos que el algoritmo no tiene acceso a la función de verdadero etiquetado $f : X \to Y$ ni a la distribución $\mathcal{D}$.

Es decir, nos centraremos en el problema de la clasificación, que se trata de un aprendizaje supervisado (conocemos las etiquetas para el conjunto de entrenamiento), *batch* (recibimos todo el conjunto de entrenamiento, y no sucesivas porciones del mismo en contraposición al aprendizaje por lotes) y pasivo (el algoritmo no tiene interacción con el usuario).

4.2.1 *Relación entre hipótesis y conjuntos*

En clasificación binaria, existe una biyección canónica entre la clase de hipótesis y el conjunto potencia de X, donde a cada hipótesis se le asigna su clase positiva:

$$\begin{array}{rcl} \{h, h : X \to Y\} & \longrightarrow & \mathcal{P}(X) \\ h & \longmapsto & X_h := \{x \in X : h(x) = 1\} \end{array}$$

Esta relación es biyección claramente, lo que justifica que identifiquemos $\{h, h : X \to Y\}$ con 2^X, donde 2^X suele ser una notación emplea-

da para referirse a $\mathcal{P}(X)$. Ello nos permite trabajar con una aplicación $h : X \to Y$ o con su conjunto asociado.

4.2.2 *Minimización del riesgo empírico*

Definición 4.2.1 (Riesgo empírico). *Definimos el riesgo empírico o error empírico de un clasificador $h : X \to Y$ como:*

$$L_S(h) = \frac{|\{i \in 1, \dots, m : h(x_i) \neq y_i\}|}{m}$$

Es decir, es el error del clasificador h sobre el conjunto de entrenamiento S. Nótese que si $[h \neq f]$ como hemos supuesto al principio, fijado una tamaño de muestra m y $h \in 2^X$, $S \mapsto L_S(h)$ es una variable aleatoria, y por tanto $L_S(h)^{-1}(]-\infty, a])$ sería un conjunto Σ^m medible

Proposición 4.2.2. *Sea $H \subseteq 2^X$. Sea $\mathcal{D}$ distribución sobre $\mathcal{X}$. Sea $f \in H$ fija. Entonces para cualquier $h \in H$:*

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m}(L_S(h)) = L_{\mathcal{D},f}(h)$$

Demostración. Llamamos $p = \mathop{P}_{x \sim \mathcal{D}}[f \neq h]$.

Nótese que h puede cometer como mucho m errores.

$$\begin{aligned}
\mathop{\mathbb{E}}_{S \sim \mathcal{D}^m}(L_S(h)) &= \sum_{k=0}^{m} \frac{k}{m} \binom{m}{k} p^k (1-p)^{m-k} \\
&= \sum_{k=1}^{m} \binom{m-1}{k-1} p^k (1-p)^{m-k} \\
&= p \cdot \sum_{k=0}^{m-1} \binom{m-1}{k} p^k (1-p)^{m-1-k} \\
&= p(1 + (1-p))^{m-1} = p
\end{aligned}$$

□

Definición 4.2.3 (Minimizador del riesgo empírico, ERM). *Decimos que un algoritmo $A : \bigcup_{m \in \mathbb{N}} (X \times Y)^m \to 2^X$ es un ERM (Empirical Risk Minimizer) si busca una hipótesis cuyo error empírico sea mínimo:*

$$A(S) \in \mathop{\arg\min}_{h \in 2^X} L_S(h)$$

Definimos $ERM(S)$ como todas las hipótesis producidas por algoritmos que son ERM sobre el conjunto S. Cuando notemos $\mathcal{Q}(ERM(S))$

con $\mathcal{Q}$ un predicado sobre elementos de 2^X, nos refereriremos a que la propiedad se verifica para todos los elementos de $ERM(S)$.

Trivialmente $L_{\mathcal{D},f}(f) = 0$ y $L_S(f) = 0$ para cualquier $S \in (X \times Y)^m$ conjunto de entrenamiento. Esto último implica que $L_S(A(S)) = 0$ para A un algoritmo ERM.

Aunque $A(S)$ minimice el error sobre el conjunto de entrenamiento, esto no significa que el error $L_{\mathcal{D},f}(A(S))$ sea mínimo. Lo ilustramos en el siguiente ejemplo:

Ejemplo:

Sea $X = \mathbb{R}$, $\mathcal{D}$ la distribución uniforme sobre $[0,2] \subset \mathbb{R}$, y la función $f = \mathbb{1}_{[0,1]}$.

Sea $S = ((x_1, y_1), \dots (x_m, y_m))$ un conjunto de entrenamiento de tamaño m sin elementos repetidos y el clasificador:

$$h_S(x) = \begin{cases} y_i & \text{si } \exists i \in \{1 \dots m\} : x = x_i \\ 0 & \text{si } \nexists i \in \{1 \dots m\} : x = x_i \end{cases}$$

No es difícil ver tomando $A(S) = h_S$, A es un ERM. Pero se verifica $\underset{x \sim \mathcal{D}}{P}[h_S \neq f] = 1/2$. $A(S)$ tiene el mismo nivel de acierto que el clasificador idénticamente 1. A este fenómeno lo denominamos *overfitting*.

Definición 4.2.4 (Overfitting). *Dado $h : X \to Y$ clasificador, diremos que produce overfitting sobre el conjunto de entrenamiento S si se cumple $L_S(h) << L_{\mathcal{D},f}(h)$.*

Necesitamos incorporar, por tanto, *conocimiento previo* al problema, de manera que revisitamos la definición de riesgo empírico.

4.2.3 *ERM con sesgo inductivo*

Se intenta corregir el riesgo de producir *overfitting* restringiendo el espacio de búsqueda, esto es, la clase de hipótesis $H \subseteq 2^X$ desde la que el algoritmo puede escoger un $h : X \to Y$. Llamamos a esto *sesgo inductivo*, puesto que se asume una determinada clase de funciones en función de las características del problema y del conocimiento previo que tenemos del mismo.

Definición 4.2.5 (ERM con sesgo inductivo). *Decimos que un algoritmo $A : \bigcup_{m \in \mathbb{N}} (X \times Y)^m \to 2^X$ es un ERM con sesgo inductivo hacia H y lo*

abreviamos ERM_H, si busca una hipótesis en H cuyo error empírico sea mínimo, es decir:

$$A(S) \in \arg\min_{h \in H} L_S(h)$$

Se define ERM_H y $ERM_H(S)$ de manera análoga a ERM y $ERM(S)$.

Dada $H \subseteq 2^X$, podemos tomar la clase de funciones iguales probabilístamente a algún elemento de H:

$$\lfloor H \rfloor_{\mathcal{D}} = \{f \in 2^X : \exists h \in H \text{ verificando } \underset{x \sim \mathcal{D}}{P}[f \neq h] = 0\}$$

Definición 4.2.6 (Hipótesis de factibilidad). *Diremos que H verifica hipótesis de factibilidad respecto a $f \in 2^X$ y $\mathcal{D}$ distribución sobre X si $f \in \lfloor H \rfloor_{\mathcal{D}}$.*

Por definición: $\underset{x \sim \mathcal{D}}{P}[f \neq h] = 0$ sii $L_{D,f}(h) = 0$ sii $L_{D,h}(f) = 0$. También es inmediato afirmar que $H \subseteq \lfloor H \rfloor_{\mathcal{D}}$ para cualquier $\mathcal{D}$.

Proposición 4.2.7. *Dada H verificando factibilidad respecto a $f \in 2^X$ y $\mathcal{D}$ distribución sobre X, entonces siendo $\bar{h} \in H$ tal que $L_{D,f}(\bar{h}) = 0$, se verifica:*

i. $\underset{S \sim \mathcal{D}^m}{P}\left[L_S(\bar{h}) = 0\right] = 1.$

ii. $\underset{S \sim \mathcal{D}^m}{P}\left[L_S(ERM_H(S)) = 0\right] = 1.$

iii. $L_{\mathcal{D},f}(A(S)) = L_{\mathcal{D},\bar{h}}(A(S))$ *para A algoritmo ERM_H arbitrario.*

Demostración. Fijamos A un ERM_H. Sabemos que por hipótesis de factibilidad existe $\bar{h} \in H$ tal que $L_{\mathcal{D},f}(\bar{h}) = \underset{x \sim \mathcal{D}}{P}[\bar{h} \neq f] = 0$

Entonces:

$$\underset{S \sim \mathcal{D}^m}{P}[L_S(\bar{h}) > 0] = \underset{S \sim \mathcal{D}^m}{P}[\exists i : \bar{h}(x_i) \neq f(x_i)] \leq \sum_{i=1}^{m} \underset{x \sim \mathcal{D}}{P}[\bar{h} \neq f] = 0$$

Y como $[L_S(\bar{h}) = 0] \subseteq [L_S(A(S)) = 0]$ por ser $A(S) \in \arg\min_{h \in \mathcal{H}} L_S(h)$, deducimos:

$$1 = \underset{S \sim \mathcal{D}^m}{P}[L_S(\bar{h}) = 0] \leq \underset{S \sim \mathcal{D}^m}{P}[L_S(A(S)) = 0] \leq 1$$

Veamos $L_{\mathcal{D},f}(A(S)) = L_{\mathcal{D},\bar{h}}(A(S))$:

$$\begin{aligned}
\underset{x\sim\mathcal{D}}{P}[A(S) \neq f] &= \underset{x\sim\mathcal{D}}{P}\Big([A(S) \neq f \wedge f = \bar{h}] \cup [A(S) \neq f \wedge f \neq \bar{h}]\Big) \\
&= \underset{x\sim\mathcal{D}}{P}[A(S) \neq f \wedge f = \bar{h}] + \underset{x\sim\mathcal{D}}{P}[A(S) \neq f \wedge f \neq \bar{h}] \\
&= \underset{x\sim\mathcal{D}}{P}[A(S) \neq \bar{h} \wedge f = \bar{h}] + 0 = \underset{x\sim\mathcal{D}}{P}[A(S) \neq \bar{h}]
\end{aligned}$$

□

Nótese que para definir ERM y ERM_H necesitamos poder tomar arg mín siempre. Parece una condición fuerte, pero en nuestro caso, como $L_S(f)$ siendo S de tamaño m puede tomar finitos valores $0, \frac{1}{m}, \ldots, \frac{m-1}{m}, 1$, el mínimo se alcanza. El valor $L_{\mathcal{D},f}(ERM_H(S))$ depende por tanto del conjunto de entrenamiento S, y la elección del mismo está sometida al azar. Ahí entrará nuestra definición de aprendizaje PAC, midiendo conjuntos probabilísticamente.

4.3 PAC COGNOSCIBILIDAD

Proporcionamos a continuación una definición que materializa el concepto de que un algoritmo pueda aprender de los datos. Las siglas PAC derivan de Probablemente Aproximadamente Correcto. Conviene tener en cuenta para la siguiente definición la biyección canónica 4.2.1, que nos permite asegurar la medibilidad de las hipótesis incluyéndo sus conjuntos asociados en la σ-álgebra.

Definición 4.3.1 (PAC cognoscible). *Una clase de funciones $H \subseteq 2^X$ es PAC cognoscible sii existen $A : \bigcup_{m\in\mathbb{N}} (X \times Y)^m \to H$ y $m_H :]0,1[^2 \to \mathbb{N}$ verificando que para todo $0 < \varepsilon, \delta < 1$, para toda distribución $\mathcal{D} = (\Sigma, \underset{x\sim\mathcal{D}}{P})$ sobre X tal que $H \subseteq \Sigma$ y para toda función de verdadero etiquetado $f \in \lfloor H \rfloor_{\mathcal{D}}$, dado $m \geq m_H(\varepsilon, \delta)$ entonces:*

$$\underset{S\sim\mathcal{D}^m}{P}\Big[L_{\mathcal{D},f}(A(S)) \leq \varepsilon\Big] \geq 1 - \delta$$

A recibe el nombre de *algoritmo de aprendizaje*, m_H se llama *complejidad muestral*, $(1-\delta)$ es la *confianza* de la predicción y $(1-\varepsilon)$ la *exactitud*. Estos dos últimos parámetros explican los calificativos aproximadamente (≡ confianza) y correcto (≡ exactitud).

En lo que sigue supondremos que las distribuciones de las que se habla, $\mathcal{D}$, incluyen en la σ-álgebra las propias hipótesis, y que tienen las garantías de medibilidad necesarias sobre $\Big[L_{\mathcal{D},f}(A(S)) \leq \varepsilon\Big]$.

Nótese que la definición de PAC cognoscibilidad implica hipótesis de factibilidad de H respecto a la función de verdadero etiquetado f y la distribución $\mathcal{D}$. El último apartado de la proposición 4.2.7 nos permite dar una caracterización con $f \in H$ estrictamente:

Caracterización 4.3.2 (PAC cognoscible). *Una clase de funciones $H \subseteq 2^X$ es PAC cognoscible sii existen una función $m_H :]0,1[^2 \to \mathbb{N}$, llamada complejidad muestral, y $A : \bigcup_{m \in \mathbb{N}} (X \times Y)^m \to H$ verificando que para todo $0 < \varepsilon, \delta < 1$, para toda distribución $\mathcal{D}$ sobre X y para toda función de verdadero etiquetado $f \in H$, dado $m \geq m_H(\varepsilon, \delta)$ entonces:*

$$\underset{S \sim \mathcal{D}^m}{P}\left[L_{\mathcal{D},f}(A(S)) \leq \varepsilon\right] \geq 1 - \delta$$

Consideraremos m_H única en el sentido de que para cada (ε, δ) nos devuelva el menor natural verificando las hipótesis del enunciado.

Lema 4.3.3 (Desigualdad fundamental). *Dados $m \in \mathbb{N}, \varepsilon \in]0,1[$ se verifica $(1-\varepsilon)^m \leq e^{-\varepsilon m}$*

Demostración. Basta hacer la demostración para $m = 1$.

Tomando $h(\varepsilon) = e^{-\varepsilon} + \varepsilon - 1$, se tiene que h es derivable y $h'(\varepsilon) = -e^{-\varepsilon} + 1 \geq 0$. Como $h(0) = 0$, se cumple $h \geq 0$. □

Teorema 4.3.4 (Las clases finitas son PAC cognoscibles). *Sea $H \subseteq 2^X$ finito. Entonces H es PAC cognoscible para cualquier ERM_H con:*

$$m_H(\varepsilon, \delta) \leq \left\lceil \frac{1}{\varepsilon} \log\left(\frac{|H|}{\delta}\right) \right\rceil$$

Demostración. Fijamos $\varepsilon, \delta \in]0,1[$. Sea una distribución $\mathcal{D}$ sobre X, $m \in \mathbb{N}$ y una función de verdadero etiquetado $f \in H$, tomamos la clase de hipótesis "malas", que será numerable al ser $|H| < \infty$:

$$H_B = \{h \in H : L_{\mathcal{D},f}(h) > \varepsilon\}$$

Sea A un ERM_H, entonces:

$$\underset{S \sim \mathcal{D}^m}{P}[L_{\mathcal{D},f}(A(S)) > \varepsilon] \leq \underset{S \sim \mathcal{D}^m}{P}[\exists h \in H_B : L_S(h) = 0] \leq \sum_{h \in H_B} P[L_S(h) = 0]$$

La segunda desigualdad viene dada por subaditividad, puesto que:

$$\underset{S \sim \mathcal{D}^m}{P}[\exists h \in H_B : L_S(h) = 0] = \underset{S \sim \mathcal{D}^m}{P}\left(\bigcup_{h \in H_B} [L_S(h) = 0]\right)$$

Además, fijada $h \in H_B$, como $L_{\mathcal{D},f}(h) > \varepsilon$:

$$\begin{aligned} &\underset{S\sim\mathcal{D}^m}{P}[L_S(h) = 0] = \underset{S\sim\mathcal{D}^m}{P}[h(x_i) = f(x_i), i = 1, \dots, m\}] \\ &= \prod_{i=1}^{m} \underset{x\sim\mathcal{D}}{P}[h = f] = \prod_{i=1}^{m}(1 - L_{\mathcal{D},f}(h)) \leq (1-\varepsilon)^m \underset{4\cdot3\cdot3}{\leq} e^{-\varepsilon m} \end{aligned}$$

Las dos desigualdades probadas, junto a la hipótesis del enunciado, y usando $H_B \subseteq H$ dan lugar a:

$$\underset{S\sim\mathcal{D}^m}{P}[L_{\mathcal{D},f}(h_S) > \varepsilon] \leq |H|e^{-\varepsilon m}$$

Y basta tomar $m \in \mathbb{N}$ tal que $|H|e^{-\varepsilon m} \leq \delta$, sii $m \geq \frac{1}{\varepsilon}\log\left(\frac{|H|}{\delta}\right)$. □

¿Hay ejemplos de clases infinitas PAC cognoscibles? La respuesta es afirmativa. Veamos un ejemplo.

Ejemplo:

Definición 4.3.5 (Clasificadores de rectángulo). *La clase de clasificadores de rectángulo en el plano se define por:*

$$H^2_{rec} = \{h_{a,b,c,d} : a \leq b, c \leq d\}$$

donde $h_{a,b,c,d} = \mathbb{1}_{[a,b]\times[c,d]}$

Proposición 4.3.6. *H^2_{rec} es PAC cognoscible*

Demostración. Haremos la prueba tratando los clasificadores como conjuntos, en virtud de la biyección canónica 4.2.1.

Sea A el algoritmo que devuelve el rectángulo más pequeño que engloba a todos los ejemplos positivos de un conjunto de entrenamiento S. Claramente A es un ERM_H.

Sea $R = [a,b] \times [c,d]$ el rectángulo que proporciona el verdadero etiquetado. Fijamos $\varepsilon, \delta \in]0,1[$.

Supongamos $\mathcal{B} = [L_{\mathcal{D},R}(A(S)) > \epsilon] \neq \emptyset$. Sea $S \in \mathcal{B}$ y $A(S) = [a^*, b^*] \times [c^*, d^*]$. Tomamos los rectángulos:

$$\begin{aligned} R_1^S &= [a, b^*] \times [c, d], \qquad & R_2^S &= [a^*, b] \times [c, d] \\ R_3^S &= [a, b] \times [c, d^*], \qquad & R_4^S &= [a, b] \times [c^*, d] \end{aligned}$$

verificándose $R_j^S \subseteq R$ para todo j. Además debe existir $L_{\mathcal{D},R}(R_j^S) \geq \frac{\varepsilon}{4}$, para algún $j \in \{1, \dots 4\}$, ya que en caso contrario, por subaditividad tendríamos $L_{\mathcal{D},R}(A(S)) < \epsilon$. Además podemos suponer

s.p.g. $(R_i^S)^c = R \setminus R_i^S$ puesto que un clasificador $A(S)$ sólo fallará en el espacio restante entre R y $A(S)$.

Seleccionamos $R_i \in \{R_i^S : S \in \mathcal{B}, L_{\mathcal{D},R}(R_i^S) \geq \frac{\varepsilon}{4}\}$, $i = 1, \ldots, 4$. suponiendo s.p.g. que existen los cuatro (caso opuesto bastaría hacer la demostración para los rectángulos que sí existiesen). Fácilmente se comprueba que $S \in \mathcal{B} \Rightarrow A(S) \cap R_i^c = \emptyset$ para algún $i = 1, \ldots, 4$, usando de nuevo reducción al absurdo y subaditividad.

La demostración acaba probando que:

$$\begin{aligned} \underset{S\sim\mathcal{D}^m}{P}[\exists i : A(S) \cap R_i^c = \emptyset] &\leq \sum_{i=1}^{4} \underset{S\sim\mathcal{D}^m}{P}[A(S) \cap R_i^c = \emptyset] \\ &\leq 4\left(1 - \frac{\varepsilon}{4}\right)^m \underset{4\cdot3\cdot3}{\leq} 4e^{-\varepsilon m/4} \end{aligned}$$

Juntando toda esta información:

$$\underset{S\sim\mathcal{D}^m}{P}[L_{\mathcal{D},R}(A(S)) > \epsilon] \leq \underset{S\sim\mathcal{D}^m}{P}[\exists i : A(S) \cap R_i^c = \emptyset] \leq 4e^{-\varepsilon m/4}$$

y tomando $m > \frac{4}{\varepsilon} log\left(\frac{4}{\delta}\right)$ llegaríamos al resultado buscado. □

4.4 APAC COGNOSCIBILIDAD

Nótese que las condiciones exigidas en PAC cognoscibilidad son muy fuertes: cumplir la hipótesis de factibilidad y que la hipótesis devuelta deba estar en H. Relajaremos esta definición con el concepto de PAC agnóstico (APAC). Vamos a relajar tanto la definición de cognoscibilidad como el marco teórico de aprendizaje.

Sea Z un conjunto, $\mathcal{D} = (\Sigma, \underset{z\sim\mathcal{D}}{P})$ distribución sobre Z en lo que sigue.

Definición 4.4.1 (Función de pérdida). *Sea H un conjunto arbitrario, se denomina función de pérdida de H sobre Z a cualquier función de la forma:*

$$l : H \times Z \to \mathbb{R}_0^+$$

Modificamos los siguientes puntos de las definiciones básicas en 4.2:

- **Verdadero etiquetado**. No asumimos que exista.
- **Generación de instancias**. Tenemos una distribución $\mathcal{D}$ sobre $Z = (X, Y)$ donde X es el dominio y Y el conjunto de etiquetas.

- **Error del clasificador**. Lo redefinimos como:

$$L_{\mathcal{D}}(h) := \mathbb{E}_{z\sim\mathcal{D}}(l(h,z))$$

- **Error empírico**. Lo redefinimos como:

$$L_S(h) := \frac{1}{m}\sum_{i=1}^{m} l(h,z_i)$$

Definición 4.4.2 (APAC cognoscible). *Una clase H es agnósticamente PAC (APAC) cognoscible respecto a Z y respecto a una función de pérdida $l : H \times Z \to \mathbb{R}^+$ si existe una función $m_H :]0,1[^2 \to \mathbb{N}$ y un algoritmo $A : \bigcup_{m\in\mathbb{N}} Z^m \to Y^X$ verificando que si $0 < \varepsilon, \delta < 1$, entonces para toda distribución $\mathcal{D}$ sobre Z se cumple que dado $m \geq m_H(\varepsilon,\delta)$:*

$$\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) \leq \inf_{h\in H} L_{\mathcal{D}}(h) + \varepsilon\right] \geq 1-\delta$$

Nótese que en contraste con nuestra definición de PAC cognoscibilidad, el algoritmo A podría devolver una hipótesis que no estuviera en H. Cuando permitimos que el algoritmo A devuelva una función $h \notin H$, de manera que $h \in H'$ y $H \subset H'$ una clase de funciones donde la función de pérdida es extensible de manera natural, el aprendizaje recibe el nombre de *aprendizaje impropio*. Si suponemos $A(S) \in H$ estaremos usando *aprendizaje propio*.

Análogamente al aprendizaje PAC, en las definiciones supondremos implícitamente que l o $L_{\mathcal{D}}$ proporcionan las suficientes garantías de medibilidad.

4.4.1 *Particularización a distintos paradigmas*

Clasificación binaria

Tendremos $Y = \{0,1\}$ o $Y = \{-1,1\}$ conjunto de etiquetas, un conjunto X y $Z = X \times Y$, y nuestras clases de hipótesis H estarán contenidas en Y^X, de manera que al aprender distribuciones sobre Z podríamos tener etiquetas distintas para una misma instancia $x \in X$.

Definición 4.4.3 (Función de pérdida $0-1$). *Llamamos función de pérdida $0-1$ a una función de pérdida de $H \subseteq Y^X$ sobre Z, l, verificando:*

$$l(h,(x,y)) = \begin{cases} 1 & \text{si } h(x) \neq y \\ 0 & \text{en otro caso} \end{cases}$$

para cualquier $h \in H$, cualquier $(x,y) \in Z$

Consideraremos la función de pérdida $0-1$ en clasificación binaria de ahora en adelante, a no ser que se especifique lo opuesto.

Con estos conceptos revisitados, podríamos asegurar que la hipótesis que menor error comete para clasificación binaria y la función de pérdidad $0-1$ es el llamado *clasificador de Bayes*:

$$f_{\mathcal{D}}(x) = \begin{cases} 1 & P[y=1|x] >= 0{,}5 \\ 0 & \text{caso opuesto} \end{cases}$$

Pero evidentemente asumimos que el algoritmo no tiene acceso a la distribución, sino solo a los datos preetiquetados de S, y que por tanto no puede devolver el clasificador de Bayes directamente.

Particularizamos a continuación la definición de APAC anterior a clasificación binaria con función de pérdida $0-1$ y aprendizaje impropio (ya que la función de pérdida $0-1$ está definida en 2^X).

Definición 4.4.4 (APAC cognoscible - paradigma binario). *Una clase de funciones $H \subseteq 2^X$ es APAC cognoscible sii existen $m_H :]0,1[^2 \to \mathbb{N}$ y un algoritmo $A : \bigcup_{m\in\mathbb{N}} Z^m \to 2^X$ verificando que para todo $0 < \varepsilon, \delta < 1$ y para toda distribución $\mathcal{D}$ sobre Z, dado $m \geq m_H(\varepsilon, \delta)$:*

$$\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) \leq \inf_{h\in H} L_D(h) + \varepsilon\right] \geq 1-\delta$$

4.4.2 *Clasificación multiclase*

Dado Y, verificando $|Y| > 2$, $Z = X \times Y$ y $H \subseteq Y^X$, al problema lo llameremos de clasificación multiclase.

Una posible función de pérdida para este problema sería también la función de pérdida $0-1$.

4.4.3 *Regresión*

Tenemos un problema de regresión si tomamos $Z = X \times \mathbb{R}$ y $H \subseteq \mathbb{R}^X$ y $l(h,(x,y)) = (h(x)-y)^2$ función de pérdida cuadrática.

4.4.4 *k-clustering*

Tenemos un problema de clústering si $Z = \mathbb{R}^d$ y los elementos de H son funciones constantes $h = (c_1, \dots c_k)$ con $c_i \in \mathbb{R}^d$. Una posible función de pérdida podría ser $l((c_1, \dots c_k), z) = \text{mín}_{i=1,\dots,k} ||c_i - z||^2$.

4.5 RELACIÓN ENTRE APAC Y PAC COGNOSCIBILIDAD

Aunque pueda parecer contraintuitivo, las clases APAC cognoscibles están contenidas en las PAC cognoscibles. De hecho, tenemos una relación tal cual se muestra en la figura siguiente:

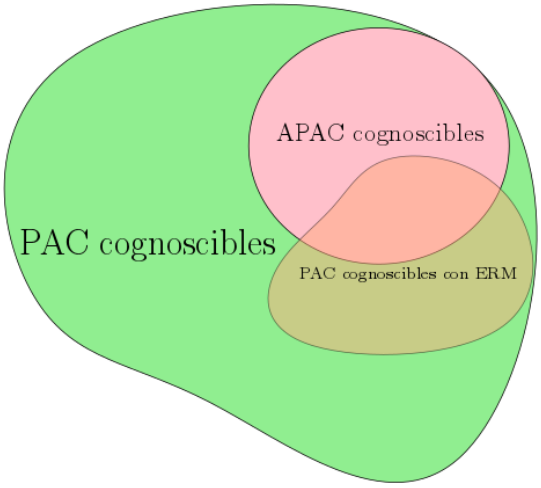


Figura 1.: Relación entre clases PAC

Proposición 4.5.1. *Si $H \subseteq 2^X$ es APAC propiamente cognoscible con funciones de pérdida $0-1$, entonces H es PAC cognoscible.*

Demostración. Sea $f \in H$ y una distribución $\mathcal{D} = (\Sigma, P)$ sobre X verificando que $H \subseteq \Sigma$. Podemos ver $\mathcal{D}$ como una distribución que asigna a cada $x \in X$ la etiqueta $f(x)$. Por definición $\text{ínf}_{h \in H} L_{\mathcal{D}}(h) = 0$. Para terminar, dado $S = (z_1, \dots, z_m)$ con $z_i = (x_i, y_i)$:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{x \sim \mathcal{D}}\Big(l\big(h, (x, f(x))\big)\Big) = \underset{x \sim \mathcal{D}}{P}[f \neq h] = L_{\mathcal{D},f}(h)$$

donde $[f \neq h] = [f \cap h]^c \in \Sigma$ por hipótesis ($f, h \in H \subseteq \Sigma$).

Por otro lado:

$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m} l(h, z_i) = \frac{|\{i \in \{1, \dots, m\} : h(x_i) \neq f(x_i)(= y_i)\}|}{m}$$

Además $l(h,\cdot)$ es Σ medible ya que $l(h,\cdot)^{-1}(\{0\}) = [f = h] \in \Sigma$ y $l(h,\cdot)^{-1}(\{1\}) = [f \neq h]$. Por hipótesis existiría un algoritmo A y $m_H(\epsilon,\delta)$ tal que para todo $m \geq m_H(\epsilon,\delta)$:

$$\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) \leq \varepsilon\right] \geq 1-\delta$$

Luego H sería PAC cognoscible. □

5

APRENDIZAJE UNIFORME

Haremos un breve recorrido por el concepto estadístico de aprendizaje uniforme (clases de Glivenko-Cantelli) y la implicación que tiene en teoría PAC. Probaremos que las clases finitas son APAC cognoscibles.

5.1 CLASES DE GLIVENKO-CANTELLI

Definición 5.1.1. *Un conjunto de entrenamiento S es ε-representativo de una clase de hipótesis H respecto a una distribución $\mathcal{D}$ y una función de pérdida l (recordemos que L se define en términos de l), si $\forall h \in H$ se tiene:*

$$|L_S(h) - L_{\mathcal{D}}(h)| \leq \varepsilon$$

Intuitivamente, cualquier ERM_H sobre S sería un buen algoritmo de aprendizaje si S es ε representativo para cualquier $\varepsilon \in]0,1[$ y cualquier distribución. Lo formalizamos con la siguiente proposición.

Proposición 5.1.2. *Si S es ε-representativo de H con respecto a una distribución $\mathcal{D}$ y una función de pérdida l, entonces se tiene:*

$$L_{\mathcal{D}}(ERM_H(S)) \leq \inf_{h \in H} L_{\mathcal{D}}(h) + 2\varepsilon$$

Demostración. Sea A un ERM_H en lo que sigue.

$L_{\mathcal{D}}(A(S)) \leq L_S(A(S)) + \varepsilon$ por ser S ε-representativo. Pero por definición de ERM_H, $L_S(A(S)) + \varepsilon \leq \text{ínf}_{h \in H} L_S(h) + \varepsilon$ y aplicando ε-representatividad de nuevo:

$$\inf_{h \in H} L_S(h) + \varepsilon \leq \inf_{h \in H} L_{\mathcal{D}}(h) + 2\varepsilon$$

En resumen: $L_{\mathcal{D}}(A(S)) \leq \text{ínf}_{h \in H} L_{\mathcal{D}}(h) + 2\varepsilon$, para $A \in ERM_H$ arbitrario. □

Definición 5.1.3 (Clase de Glivenko-Cantelli). *Decimos que una clase de hipótesis H es de Glivenko-Cantelli, respecto a un dominio Z, y a una función de pérdida l, si existe una función $m_H^{CU} :]0,1[^2 \to \mathbb{N}$ verificando*

que para todo $0 < \delta, \varepsilon < 1$ *y para toda distribución* $\mathcal{D}$ *sobre* Z*, siendo* $m \geq m_H^{CU}(\varepsilon, \delta)$*, entonces:*

$$\underset{S\sim\mathcal{D}^m}{P}[\forall h \in H, |L_S(h) - L_{\mathcal{D}}(h)| \leq \varepsilon] \geq 1 - \delta$$

Análogamente a lo que ocurría en la definición 4.3.2, podemos considerar m_H^{CU} única, en el sentido de que para cada $0 < \delta, \varepsilon < 1$, $m_H^{CU}(\varepsilon, \delta)$ es el mínimo natural que satisface las hipótesis. También suponemos suficientes garantías de medibilidad sobre $l, L_S, L_{\mathcal{D}}$.

5.2 GLIVENKO-CANTELLI Y APAC COGNOSCIBILIDAD

Teorema 5.2.1. *Sea* H *una clase de hipótesis de Glivenko-Cantelli, respecto a* Z *y función de pérdida* l*. Entonces es APAC cognoscible con cualquier algoritmo* ERM_H *y complejidad muestral* $m(\varepsilon, \delta) \leq m_H^{UC}\left(\frac{\varepsilon}{2}, \delta\right)$*.*

Demostración. Sea A un ERM_H arbitrario, y $\mathcal{D}$ una distribución arbitraria sobre Z. Fijamos $m \geq m_H^{UC}\left(\frac{\varepsilon}{2}, \delta\right)$.

Sea $S = (z_1, \dots z_m)$ un conjunto de entrenamiento, verificando que:

$$\forall h \in H, |L_S(h) - L_{\mathcal{D}}(h)| \leq \frac{\varepsilon}{2} \qquad (*)$$

Entonces S es ε representativa para $\mathcal{D}$ y l, y por proposición 5.1.2:

$$L_{\mathcal{D}}(A(S)) \leq \inf_{h\in H} L_{\mathcal{D}}(h) + \varepsilon \qquad (**)$$

Pero como $(*)$ ocurre con probabilidad (sobre S) mayor o igual a $1 - \delta$, entonces $(**)$ ocurre con probabiliad mayor o igual a $1 - \delta$ □

Proposición 5.2.2. *Sea* H *una clase de hipótesis finita,* Z *un dominio y sea* $l : H \times Z \to [a, b]$ *una función de pérdida. Entonces* H *es de Glivenko-Cantelli respecto a* Z *y* l *con:*

$$m_H^{CU}(\varepsilon, \delta) \leq \left\lceil \frac{log(2|H|/\delta)(b-a)^2}{2\varepsilon^2} \right\rceil$$

Demostración. Fijamos $0 < \delta, \varepsilon < 1$.

Necesitamos encontrar $m \in \mathbb{N}$ verificando:

$$\underset{S\sim\mathcal{D}^m}{P}[\exists h \in H : |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon] < \delta$$

Partimos de la siguiente desigualdad, que usaremos más adelante, obtenida por subaditividad:

$$\underset{S\sim\mathcal{D}^m}{P}[\exists h \in H, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon] \leq \sum_{h\in H} \underset{S\sim\mathcal{D}^m}{P}[|L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon] \tag{$*$}$$

Fijamos $h \in H$ en lo que sigue.

Dado un conjunto de entrenamiento $S = (z_1, \dots z_m)$, recordamos que $L_{\mathcal{D}}(h) = \underset{z\sim\mathcal{D}}{\mathbb{E}}(l(h,z))$ y que $L_S(h) = \frac{1}{m}\sum_{i=1}^m l(h, z_i)$, donde $z_i \sim \mathcal{D}$ son i.i.d. y por tanto:

$$\underset{S\sim\mathcal{D}^m}{\mathbb{E}}(L_S(h)) = \frac{1}{m}\sum_{i=1}^m \underset{z_i\sim\mathcal{D}}{\mathbb{E}}(l(h,z_i)) = \underset{z\sim\mathcal{D}}{\mathbb{E}}(l(h,z)) = L_{\mathcal{D}}(h)$$

Llamando $W_i : Z \to \mathbb{R}$, $W_i(z_i) = l(h, z_i)$, tenemos $a \leq W_i \leq b$ con las W_i independiente e idénticamente distribuidas. Estamos en condiciones de aplicar la desigualdad 3.4.3 de Hoeffding.

Por tanto, desde la parte derecha de ($*$):

$$\underset{S\sim\mathcal{D}^m}{P}\left[\left|\frac{1}{m}\sum_{i=1}^m W_i - L_{\mathcal{D}}(h)\right| > \varepsilon\right] = P[|L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon] \leq 2e^{-2m\left(\frac{\varepsilon}{b-a}\right)^2}$$

Así:

$$P[\exists h \in H, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon] \leq |H|2e^{-2m\left(\frac{\varepsilon}{b-a}\right)^2}$$

Despejando m para que $|H|2e^{-2m\left(\frac{\varepsilon}{b-a}\right)^2} < \delta$ llegamos al resultado buscado. □

Recordemos que hasta ahora el resultado que habíamos obtenido en el teorema 4.3.4 era el carácter PAC cognoscible de las clases de hipótesis finitas. El teorema que sigue generaliza dicho resultado para cualquier función de pérdida acotada.

Corolario 5.2.3. *Sea H una clase de hipótesis finita, Z un dominio y sea $l : H \times Z \to [a, b]$ una función de pérdida. Entonces H es APAC cognoscible con complejidad muestral:*

$$m_H(\varepsilon, \delta) \leq \left\lceil \frac{2log(2|H|/\delta)(b-a)^2}{\varepsilon^2} \right\rceil$$

Demostración. Estamos en condiciones de aplicar la proposición 5.2.2. H sería de Glivenko-Cantelli respecto a Z, l. Luego por teorema 5.2.1 llegamos al resultado buscado. □

Por tanto, si podríamos haber deducido el teorema 4.3.4 a partir del anterior, ¿por qué no probar directamente este último?. Asintóticamente, para errores muy pequeños $\epsilon \sim 0$, el resultado último nos proporciona una cota mucho mayor que 4.3.4, pues depende de $\frac{1}{\epsilon^2}$, mientras que el primero dependía de $\frac{1}{\epsilon}$. Para PAC cognoscibilidad el teorema 4.3.4 proporciona un mejor resultado que este último.

6

NO FREE LUNCH

Veremos que dado un algoritmo de aprendizaje PAC no resulta óptimo para aprender todas las distribuciones. Lo vemos para clasificación binaria. Consideramos en este capítulo funciones de pérdida $0-1$, puesto que trabajaremos con paradigma PAC.

6.1 TEOREMA DE NO FREE LUNCH

Lema 6.1.1. *Dados $w_1, \dots w_T \in \mathbb{R}$. Entonces:*

$$\min_{i=1,\dots,T} w_i \leq \frac{1}{T}\sum_{i=1}^{T} w_i \leq \max_{i=1,\dots,T} w_i$$

Demostración. Trivial. □

Teorema 6.1.2 (No Free Lunch). *Sea un dominio X, un conjunto de etiquetas $Y = \{0,1\}$, $H = 2^X$. Sea $A : \bigcup_{m\in\mathbb{N}} (X \times Y)^m \to 2^X$ un algoritmo. Dado $m \leq \frac{|X|}{2}$. Entonces existe $\mathcal{D}$ sobre $X \times \{0,1\}$ verificando:*

i Existe una función $f : X \to \{0,1\}$ con $L_{\mathcal{D}}(f) = 0$

ii $\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) > \frac{1}{8}\right] \geq \frac{1}{7}$

Demostración. Sea un conjunto de instancias $C = \{x_1, \dots, x_{2m}\}$. Hay $T = 2^{2m}$ posibilidades de etiquetado del conjunto, esto es, T posibles hipótesis, $f_i : C \to \{0,1\}$, que vamos a extender a X llamándolas $\bar{f}_i$ de forma que $\bar{f}_{i|C} = f_i$ y $\bar{f}_i(x) = 0 \quad \forall x \in X \setminus C$. Vamos a tomar para cada una de ellas una distribución $\mathcal{D}_i = (2^X, P_i)$ definida por:

$$\forall (x,y) \in X \times \{0,1\}, \qquad \underset{z\sim\mathcal{D}_i}{P_i}[z = (x,y)] = \begin{cases} 1/|C| & x \in C, y = f_i(x) \\ 0 & \text{si no} \end{cases}$$

Claramente $L_{\mathcal{D}_i}(f_i) = 0$. Vamos a probar que:

$$\exists i \in \{1, \dots T\} : \quad \underset{S\sim\mathcal{D}_i^m}{\mathbb{E}}[L_{\mathcal{D}_i}(A(S))] \geq \frac{1}{4}$$

Fijamos $i \in \{1, \dots, T\}$. Hay $k = (2m)^m$ posibles tuplas de tamaño m, $S_j, j = 1, \dots k$, tomadas desde C. Siendo $S_j = (x_1, \dots, x_m)$ notamos $S_j^i = ((x_1, f_i(x_1)), \dots, (x_m, f_i(x_m)))$. Cada S_j^i tiene la misma probabilidad de ser nuestro conjunto de entrenamiento (extracción de m valores con reemplazamiento desde el conjunto C), verificándose:

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}_i^m}[L_{\mathcal{D}_i}(A(S))] = \frac{1}{k} \sum_{j=1}^{k} L_{\mathcal{D}_i}(A(S_j^i))$$

Recordando que hemos llamado $k = (2m)^m$, $T = 2^{2m}$, y usando conmutatividad de sumatorios, se tiene:

$$\begin{aligned} \max_{i \in \{1, \dots T\}} \frac{1}{k} \sum_{j=1}^{k} L_{\mathcal{D}_i}(A(S_j^i)) &\underset{6.1.1}{\geq} \frac{1}{T} \sum_{i=1}^{T} \frac{1}{k} \sum_{j=1}^{k} L_{\mathcal{D}_i}(A(S_j^i)) \\ &= \frac{1}{k} \sum_{j=1}^{k} \frac{1}{T} \sum_{i=1}^{T} L_{\mathcal{D}_i}(A(S_j^i)) \\ &\underset{6.1.1}{\geq} \min_{j \in \{1, \dots k\}} \frac{1}{T} \sum_{i=1}^{T} L_{\mathcal{D}_i}(A(S_j^i)) \qquad (*) \end{aligned}$$

Fijamos ahora $j \in \{1, \dots, k\}$:

Sean $\{v_r\}_{r=1}^{p}$ los elementos de C no presentes en el conjunto de entrenamiento S_j. Claramente, como $|C| = 2m$ y S_j puede tener elementos repetidos, se tiene $p \geq m$. Asimismo:

$$L_{\mathcal{D}_i}(A(S_j^i)) = \frac{1}{|C|} \sum_{x \in C} \mathbb{1}_{[A(S_j^i)(x) \neq f_i(x)]} = \frac{1}{2m} \sum_{x \in C} \mathbb{1}_{[A(S_j^i)(x) \neq f_i(x)]}$$

Deducimos:

$$\begin{aligned} \frac{1}{T} \sum_{i=1}^{T} L_{\mathcal{D}_i}(A(S_j^i)) &\geq \frac{1}{T} \sum_{i=1}^{T} \frac{1}{2m} \sum_{x \in C} \mathbb{1}_{[A(S_j^i)(x) \neq f_i(x)]} \\ &\geq \frac{1}{T} \frac{1}{2p} \sum_{r=1}^{p} \sum_{i=1}^{T} \mathbb{1}_{[A(S_j^i)(v_r) \neq f_i(v_r)]} \\ &\underset{6.1.1}{\geq} \frac{1}{T} \frac{1}{2} \min_r \sum_{i=1}^{T} \mathbb{1}_{[A(S_j^i)(v_r) \neq f_i(v_r)]} \qquad (**) \end{aligned}$$

Dado un v_r cualquiera, $v_r \notin \left\{ x_k : S_j = (x_1, \dots, x_m) \right\}$. Luego existen $f_i, f_{i'}$ que se diferencian justo en el elemento v_r. Por tanto, como $S_j^i = S_j^{i'}$, tendremos que o bien $A(S_j^i)(v_r) \neq f_i(v_r)$ o bien $A(S_j^{i'})(v_r) \neq f_{i'}(v_r)$:

$$\frac{1}{T} \frac{1}{2} \sum_{i=1}^{T} \mathbb{1}_{[A(S_j^i)(v_r) \neq f_i(v_r)]} = \frac{1}{T} \frac{1}{2} \frac{T}{2} = \frac{1}{4} \qquad (***)$$

Y uniendo (∗), (∗∗) y (∗∗∗) tendríamos:

$$\max_{i\in\{1,\dots,T\}} \frac{1}{k}\sum_{j=1}^{k} L_{\mathcal{D}_i}(A(S_j^i)) \geq \frac{1}{4}$$

Sea $k = \arg\max_{i\in\{1,\dots,T\}} \frac{1}{k}\sum_{j=1}^{k} L_{\mathcal{D}_i}(A(S_j^i))$.

Si $\mathcal{D} = \mathcal{D}_k$ cumple la parte 2 del enunciado del teorema, podemos tomar $f = f_k$ que cumple la primera parte del enunciado.

Como $L_{\mathcal{D}}(A(\cdot))$ puede ser vista como una variable aleatoria donde $S \sim \mathcal{D}^m$ y que toma valores en $[0,1]$, tenemos que tomando $W = 1 - L_{\mathcal{D}}(A(\cdot)) \geq 0$, $a = \frac{7}{8}$, estamos en condiciones de aplicar la desigualdad 3.4.1 de Markov:

$$\underset{S\sim\mathcal{D}^m}{P}\left[\frac{1}{8} \geq L_{\mathcal{D}}(A(S))\right] \leq \frac{3}{4}\cdot\frac{8}{7} = \frac{6}{7}$$

donde $\underset{S\sim\mathcal{D}^m}{\mathbb{E}}(W) = \underset{S\sim\mathcal{D}^m}{\mathbb{E}}(1 - L_{\mathcal{D}}(A(\cdot))) = 1 - \underset{S\sim\mathcal{D}^m}{\mathbb{E}}(L_{\mathcal{D}}(A(\cdot))) \leq \frac{3}{4}$

Es decir:

$$\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) > \frac{1}{8}\right] \geq \frac{1}{7}$$

□

Como consecuencia del teorema, podemos decir que no hay un algoritmo de aprendizaje óptimo para todas las distribuciones, puesto que para un número de puntos $m \leq \frac{|X|}{2}$ y la distribución dada por el teorema anterior, el algoritmo ERM_H con $H = \{f\}$ cometería error nulo, mientras que el que habíamos fijado obtendría con cierta probabilidad un fallo superior a $\frac{1}{8}$.

Corolario 6.1.3. *Sea X un dominio con $|X| = \infty$, entonces 2^X no es PAC cognoscible.*

Demostración. Supongamos que lo fuese. Sea A el algoritmo que hace a 2^X PAC cognoscible. Entonces todo $m \in \mathbb{N}$, como $m \leq \frac{|X|}{2}$, el teorema anterior nos diría que existe una distribución $\mathcal{D}$ verificando:

$$\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) > \frac{1}{8}\right] \geq \frac{1}{7} \Leftrightarrow \underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) \leq \frac{1}{8}\right] \leq \frac{6}{7}$$

Luego no puede existir $m_H\left(\frac{1}{8},\delta\right)$ con $\delta < \frac{1}{7}$. □

6.2 EQUILIBRIO ERROR-VARIANZA

Definición 6.2.1. *Llamamos error de aproximación a* $err_H := \inf_{h \in H} L_{\mathcal{D}}(h)$. *LLamamos error de estimación a* $err_{est} := L_{\mathcal{D}}(A(S)) - err_H$ *para un cierto algoritmo* A *y conjunto de entrenamiento* S *arbitrario.*

El equilibrio error-varianza hace referencia a un concepto intuitivo que se desprende de la demostración del teorema 6.1.2. La clave de dicha demostración ha sido que estabamos considerando como conjunto de clasificadores todo el espacio 2^X. Así, intuitivamente, cuanto más complejo o grande sea el conjunto H, mejor aproximaremos el verdadero o mejor clasificador, pero más número de muestras necesitaremos y más costoso será aproximarlo.

Cuando crece err_H, decrece err_{est} y viceversa. Por tanto, el *machine learning* intenta buscar un equilibrio entre bondad de los clasificadores que se prueban y la complejidad de la clase a la que pertenecen los mismos.

7

TEOREMA FUNDAMENTAL DE PAC

Este capítulo introduce el concepto de fragmentación y de dimensión Vapnik-Chervonenkis (VC), un término muy importante en combinatoria y en geometría computacional. A partir de dichas nociones obtendremos el lema de Sauer-Shelah, la herramienta que nos falta para poder demostrar el teorema fundamental del aprendizaje PAC, que relaciona estadística, a través de clases de Glivenko-Cantelli, teoría PAC y geometría computacional, a través de la dimensión VC.

7.1 FRAGMENTACIÓN

Definición 7.1.1. *Sea $H \subseteq 2^X$, y $C \subseteq X$. Llamamos restricción de H a C:*

$$H_C := \{h_{|C} : h \in H\}$$

En notación conjuntista, existe biyección entre H_C y C_H donde:

$$C_H := \{C_h : h \in H\} \subseteq \mathcal{P}(X)$$

Definición 7.1.2 (Conjunto fragmentado). *Sea $H \subseteq 2^X$ y sea $A \subseteq X$. Decimos que H fragmenta a A si para toda función $g : A \to \{0,1\}$ existe $h \in H$ verificando $h_{|A} = g$.*

Recordando la biyección canónica entre hipótesis y conjuntos 4.2.1, podemos aportar la siguiente caracterización de fragmentación.

Proposición 7.1.3 (Caracterizaciones de conjunto fragmentado). *Sea $H \subseteq 2^X$ y sea $A \subseteq X$ finito. Entonces equivalen:*

i H *fragmenta a* A.

ii *Para todo* $B \subseteq A$, *existe* $h \in H$ *verificando* $X_h \cap A = B$

iii $A_H = \mathcal{P}(A)$.

iv $|A_H| = 2^{|A|}$

v $|H_A| = 2^{|A|}$

Demostración. Trivial desde 4.2.1 y 7.1.2. □

Cuando demostrábamos el teorema No Free Lunch 6.1.2, pedíamos que la clase de clasificadores H fuese todo 2^X. Observando la demostración, solo asumiamos la existencia de un conjunto $C \subseteq X$ sobre el que pudiéramos definir todas las posibles hipótesis $C \to \{0,1\}$. Revisitamos por tanto el enunciado de dicho teorema.

Teorema 7.1.4 (Teorema de No Free Lunch revisitado). *Sea un dominio X, un conjunto de etiquetas $Y = \{0,1\}$, $H \subseteq 2^X$, $A : \bigcup_{m\in\mathbb{N}} (X \times Y)^m \to 2^X$ un algoritmo. Dado $m \leq \frac{|X|}{2}$, supongamos que existe $C \subseteq X$ de tamaño $2m$ fragmentado por H. Entonces existe $\mathcal{D}$ distribución sobre $X \times \{0,1\}$ verificando:*

i *Existe una función $f : X \to \{0,1\}$ con $L_{\mathcal{D}}(f) = 0$*

ii $\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) > \frac{1}{8}\right] \geq \frac{1}{7}$

7.2 DIMENSIÓN VC

Definición 7.2.1 (Dimensión VC). *Sea $H \subseteq 2^X$. Definimos su dimensión Vapnik-Chervonenkis, que abreviaremos dimensión VC, como:*

$$VC(H) := \text{máx}\{|A| : A \subseteq X, A \text{ es fragmentado por } H\}$$

Ejemplo:

Sea $H_{thr} = \{h_a = \mathbb{1}_{]-\infty,a]} : a \in \mathbb{R}\}$ clase de hipótesis sobre $\mathbb{R}$.

Dado un conjunto $C = \{\alpha\}$, podemos tomar h_α y $h_{\alpha-1}$, que nos dan todos los posibles etiquetados de C. Sin embargo, dado un conjunto de tamaño 2, $C = \{\alpha, \beta\}$, donde podemos suponer s.p.g. $\alpha < \beta$. Entonces no podemos encontrar $h_b \in H$ verificando $h_b(\alpha) = 0$ y $h_b(\beta) = 1$, ya que esto implicaría que $b \geq \beta$ y por tanto entraría en contradicción con que $h_b(\alpha) = 0$.

Hemos probado $VC(H_{thr}) = 1$.

Ejemplo:

Sea $H^1_{rec} = \{h_{a,b} = \mathbb{1}_{[a,b]} : a, b \in \mathbb{R}\}$ clase de hipótesis sobre $\mathbb{R}$.

Dado un conjunto $C = \{\alpha, \beta\}$, con $\alpha < \beta$, las hipótesis $h_{\alpha+\delta_1,\beta-\delta_2}$ con $\delta_i \in \left\{0, \frac{\beta-\alpha}{2}\right\}$ nos dan todos los posibles etiquetados de C. Sin embargo, dado un conjunto de tamaño 3, $C = \{\alpha, \beta, \theta\}$, donde podemos suponer $\alpha < \beta < \theta$, no podemos encontrar $h_b \in H$ verificando $h_b(\alpha) = 1$, $h_b(\theta) = 1$ y $h_b(\beta) = 0$.

Por tanto $VC(H^1_{rec}) = 2$.

Ejemplo:

Sea H^2_{rec} la clase de hipótesis definidas en 4.3.5.

Podemos fragmentar el conjunto $C = \{(0,1),(0,-1),(1,0),(-1,0)\}$

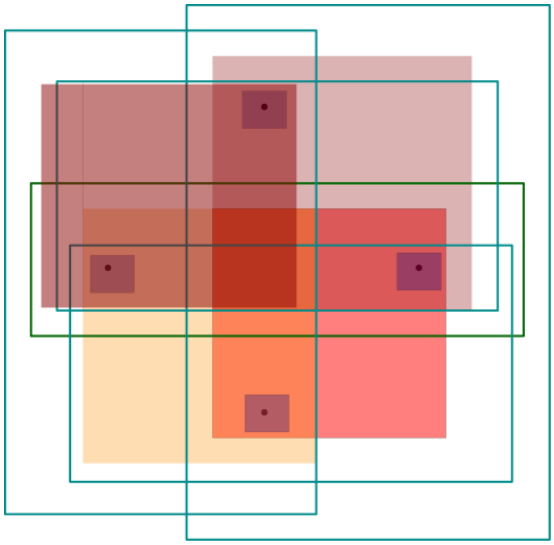


Figura 2.: Rectángulos que framentan C

Luego $VC(H^2_{rec}) \geq 4$. Veamos que $VC(H^2_{rec}) = 4$.

Sea $C \subseteq \mathbb{R}^2$ con $|C| = 5$. Entonces podemos tomar $z_1 \in C$ punto de menor abscisa, $z_2 \in C$ punto de mayor abscisa, $z_3 \in C$ punto de mayor ordenada, $z_4 \in C$ punto de menor ordenada. Se podría tener s.p.g. $z_i = z_j$ para algún $i \neq j$. Si tenemos un rectángulo que contiene a dichos puntos, debe necesariamente contenerlos a todos, luego no existe uno que contenga esos cuatro puntos y no el restante.

Proposición 7.2.2 (Propiedades de la dimensión VC). *Sean $H, H_1, H_2 \subseteq 2^X$. Se cumple:*

i $VC(H) \leq \log_2(|H|)$.

ii *Si* $H_1 \subseteq H_2$ *entonces* $VC(H_1) \leq VC(H_2)$.

iii $VC(H)$ *no depende del número de parámetros de los clasificadores de* H.

Demostración.

i Equivalentemente, veamos $2^{VC(H)} \leq |H|$.

Si $VC(H) < \infty$, entonces H fragmenta un conjunto de tamaño $VC(H)$, a saber, A, y por tanto deben existir tantos elementos en H al menos como subconjuntos de A. El número de subconjuntos de A viene dado por $2^{|A|} = 2^{VC(H)}$.

Si $VC(H) = \infty$, entonces $|H| = \infty$, ya que para cada $n \in \mathbb{N}$ podríamos fragmentar con H un conjunto de tamaño n, y por tanto H debería tener al menos 2^n elementos.

ii Si H_1 fragmenta un conjunto de tamaño $VC(H_1)$, entonces H_2 también fragmenta ese mismo conjunto puesto que $H_1 \subseteq H_2$. Por tanto $VC(H_1) \leq VC(H_2)$.

iii En los ejemplos anteriores hemos visto como $VC(H)$ coincidía con el número de parámetros de los que dependían los clasificadores de H: $VC(H_{thr}) = 1$, $VC(H^1_{rec}) = 2$, $VC(H^2_{rec}) = 4$. Veamos un ejemplo donde no se cumple esto.

Sea $H_{sin} = \{h_\theta : \theta \in \mathbb{R}\}$ donde $h_\theta : \mathbb{R} \to \{0,1\}$ está definida por $h_\theta(x) = \lceil -0{,}5 sen(\theta x) \rceil$. Cada $h \in H_{sin}$ depende de un único parámetro θ.

Sea $d \in \mathbb{N}$. Tomamos d puntos dados por $x_i = 0.x_{i,1}x_{i,2}\ldots x_{i,2^d}x_{i,2^d+1}$ donde cada $x_i, i = 1,\ldots,d$ viene determinado por una fila de la matriz $\left(x_{i,j}\right)$ donde la columna j-ésima codifica el número $j-1$ en binario leído de arriba a abajo, para $j \neq 2^d + 1$, y la columna (2^d+1)-ésima es constantemente 1.

$$d \left\{ \begin{pmatrix} 0 & 0 & \ldots & 1 & 1 \\ 0 & 0 & \ldots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \ldots & 1 & 1 \end{pmatrix} \right. \\ \underbrace{}_{0} \quad \underbrace{}_{1} \qquad\qquad \underbrace{}_{2^d-1}$$

Así, dada una asignación de d etiquetas, debe codificar un número en binario entre 0 y $2^d - 1$, a saber, la columna k ésima de la matriz. Tomamos el clasificador $h(x) = \lceil -0{,}5 sen(10^k \pi x) \rceil$ que verificará que su asignación de etiquetas es justamente la columna k ésima.

La última columna constantemente 1 puede explicarse en que daremos $x_{i,1}\ldots x_{i,k}$ medias vueltas a la circunferencia unidad y recorreremos una fracción $0.x_{i,k+1}\ldots x_{i,2^d}1$ no nula de otra media vuelta a la circunferencia. Si $x_{i,k} = 1$ entonces $h(x_i) = 1$, y si $x_{i,k} = 0$ entonces $h(x_i) = 0$.

Como $d \in \mathbb{N}$ era arbitrario, $VC(H_{sin}) = \infty$, y por tanto hemos encontrado una clase de hipótesis uniparámetrica con dimensión $VC(H_{sin}) \neq 1$.

□

Corolario 7.2.3. *Sea $H \subseteq 2^X$. Si $VC(H) = \infty$, entonces H no es PAC cognoscible.*

Demostración. Si $X < \infty$, entonces $|H| \leq 2^{|X|} < \infty$, y $VC(H) \leq \log_2(|H|) < \infty$. Luego $|X| = \infty$.

Por reducción al absurdo. Sea A el algoritmo de aprendizaje. Para todo $m \in \mathbb{N}$ par arbitrario, como $m \leq \frac{|X|}{2}$, existe C con $|C| = m$ fragmentado por H, el teorema 7.1.4 nos diría que existe una distribución $\mathcal{D}$ verificando:

$$\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) > \frac{1}{8}\right] \geq \frac{1}{7}$$

Luego no puede existir $m_H\left(\frac{1}{8}, \delta\right)$ con $\delta < \frac{1}{7}$. □

Nótese la similitud de este corolario con 6.1.3.

7.2.1 *Función de crecimiento*

Definición 7.2.4 (Función de crecimiento). *Sea H una clase de hipótesis. Definimos como función de crecimiento de H:*

$$\Pi_H : \mathbb{N} \longrightarrow \mathbb{N}, \qquad \Pi_H(m) = \underset{C\subseteq X,\ |C|=m}{\text{máx}} |H_C|$$

Esta función está bien definida puesto que fijado $m \in \mathbb{N}$ y $C \subseteq X$ con $|C| = m$, se tiene siempre que $|H_C| \leq 2^C$, identificando elementos de H_C con subconjuntos de C.

Lema 7.2.5 (Lema de Sauer-Shelah-Perles). *Sea $H \subseteq 2^X$ con $VC(H) \leq d < \infty$. Entonces para $m \in \mathbb{N}$ verificando $d \leq m$:*

$$\Pi_H(m) \leq \sum_{i=0}^{d} \binom{m}{i} \leq m^d + 1$$

Demostración. Supongamos s.p.g. $VC(H) = d$.

Empezamos probando que una clase $\mathcal{F}$, con $|\mathcal{F}| < \infty$, fragmenta al menos $|\mathcal{F}|$ conjuntos. Lo hacemos por inducción sobre el tamaño de $X_{\mathcal{F}}$ que es un conjunto finito por ser $\mathcal{F}$ clase finita.

Si su tamaño es 1, fragmenta al conjunto vacío.

Supuesto que se verifica la hipótesis para tamaños menores que $k-1$ y sea $|X_{\mathcal{F}}| = k$. Escogemos entonces $x \in X$ verificando que x pertenece a algunos conjuntos de $X_{\mathcal{F}}$ pero no a todos (debe existir, sino tendríamos que $X_{\mathcal{F}}$ contiene un único conjunto).

$$\text{Definimos: } \begin{array}{ll} A & = \{S \in X_{\mathcal{F}} : x \in S\} \\ A' & = \{S \setminus \{x\} : S \in A\} \quad \text{donde } X_{\mathcal{F}} = A + B. \\ B & = \{S \in X_{\mathcal{F}} : x \notin S\} \end{array}$$

Claramente $|A| = |A'|$ y por hipótesis de inducción A' fragmenta $p \geq |A|$ conjuntos, y B fragmenta $|B|$ conjuntos. Por otro lado, si un conjunto es fragmentado por A y B a la vez, entoces no contiene a x, luego es fragmentado por A' y por B a la vez, y podremos tener a lo sumo p conjuntos de este tipo. De hecho, dado S conjunto fragmentado por A y B, se tendría $x \notin S$, claramente, y $S \cup \{x\}$ sería fragmentado por $A + B$, pero no por A o B por separado.

Así, $\mathcal{F}$ fragmenta al menos $|A| + p + |B| - p = |A| + |B| = |X_{\mathcal{F}}|$ conjuntos.

Fijamos ahora un $C \subseteq X$ con $|C| = m$. Entonces $|H_C| < \infty$. Luego:

$$|H_C| \leq |\{B \subseteq C : H_C \text{ fragmenta } B\}| = |\{B \subseteq C : H \text{ fragmenta } B\}|$$

Por tanto $|\{B \subseteq C : H \text{ fragmenta } B\}| \leq \sum_{i=0}^{d} \binom{m}{i}$ al ser $VC(H) = d$ y el número de subconjuntos de C de cardinal menor o igual a d viene dado por $\sum_{i=0}^{d} \binom{m}{i}$.

Veamos ahora $\sum_{i=0}^{d} \binom{m}{i} \leq m^d + 1$. La parte izquierda de la desigualdad nos da todos los posibles subconjuntos desde C. Excepto el conjunto vacío, todos los demás subconjuntos pueden obtenerse extrayendo elementos con repetición desde C, y hay m^d posibles extracciones de este tipo. □

Corolario 7.2.6. *Sea $H \subseteq 2^X$. Si existe $d \in \mathbb{N}$ tal que $\Pi_H(m) > \sum_{i=0}^{d} \binom{m}{i}$, entonces $VC(H)$ es al menos $d+1$.*

Demostración. Por reducción al absurdo desde el lema 7.2.5, si $\Pi_H(m) > \sum_{i=0}^{d} \binom{m}{i}$ entonces no puede ser $VC(H) \leq d$. □

7.3 TEOREMA FUNDAMENTAL DEL APRENDIZAJE PAC

Lema 7.3.1. *Sea $a > 0$. Entonces $x \geq 2a\log(a) \Rightarrow x \geq a\log(x)$ para cualquier $x > 0$.*

Demostración. Tomamos $f(x) = x - a\log(x)$. Se tiene $f'(x) = 1 - \frac{a}{x}$ y $f'(x) = 0 \Leftrightarrow x = a$. Como $\lim_{x\to+\infty} f(x) = +\infty$, f debe tener un mínimo en a, por haber un único punto crítico.

Si $a \leq e$, entonces se tiene $f(a) = a - a\log(a) \geq a - a \geq 0$ y la desigualdad se cumple.

Si $a > e$, entonces $2a\log(a) \geq a$ y:

$$\begin{aligned} f(2a\log(a)) &= 2a\log(a) - a\log(2a\log(a)) \\ &= 2a\log(a) - a\log(a) - a\log(2\log(a)) \\ &= a\log(a) - a\log(2\log(a)) = a\log\left(\frac{a}{2\log(a)}\right) \end{aligned}$$

Tomando $g(a) = a - 2\log(a)$, se tiene $g'(a) = 1 - \frac{2}{a} = 0$ sii $a = 2$, y $g(2) = 2 - \log(4) \geq 2 - \log(e^2) = 0$. Luego $g(a) \geq 0$ y $\frac{a}{2\log(a)} \geq 1$, con lo que $f(2a\log(a)) \geq 0$, y se deduce la igualdad al ser f creciente para $x \geq a$. □

Lema 7.3.2. *Sea $a \geq \frac{1}{2}$, $b > 0$. Entonces $x \geq 4a\log(2a) + 2b \Rightarrow x \geq a\log(x) + b$.*

Demostración. Como $a \geq \frac{1}{2}$, entonces $x \geq 4a\log(2a) \geq 0$ y $x \geq 2b$.

Por otro lado, como $b > 0$, entonces $2b > 0$ y $x \geq 4a\log(2a)$. Usando el lema 7.3.1 deducimos que $x \geq 2a\log(x)$.

Sumando desigualdades: $2x \geq 2a\log(x) + 2b \Leftrightarrow x \geq a\log(x) + b$. □

Teorema 7.3.3 (Teorema fundamental de aprendizaje PAC). *Sea $H \subseteq 2^X$. Consideramos funciones de pérdida $0-1$. Entonces equivalen:*

i *H es de Glivenko-Cantelli.*

ii *H es APAC cognoscible por cuaquier algoritmo ERM_H.*

iii *H es APAC cognoscible.*

iv *H es PAC cognoscible.*

v *H es PAC cognoscible por cualquier algoritmo ERM_H.*

vi *$VC(H) < \infty$.*

Demostración. El teorema 5.2.1 demuestra (i) ⇒ (ii) ⇒ (iii).

Bajo las condiciones de función de pérdida $0-1$, (iii) $\Rightarrow$ (iv) y por proposición 4.5.1 (ii) $\Rightarrow$ (v).

Del corolario 7.2.3 deducimos, por contradicción, (iv) $\Rightarrow$ (vi) y (v) $\Rightarrow$ (vi). Falta ver (vi) $\Rightarrow$ (i).

Sea $Z = X \times \{0,1\}$. Fijamos $\epsilon, \delta \in]0,1[$ y una distribución $\mathcal{D}$ sobre Z que cumpla suficientes condiciones de medibilidad sobre la función de pérdida. Llamamos:

$$Q = \{S \in Z^m : \exists h \in H \text{ con } |L_S(h) - L_{\mathcal{D}}(h)| \geq \varepsilon\}$$
$$R = \{(S_1, S_2) \in Z^m \times Z^m : \exists h \in H \text{ con } |L_{S_1}(h) - L_{S_2}(h)| \geq \frac{\varepsilon}{2}\}$$

Entonces para $m \geq \frac{4}{\varepsilon^2}$ se verifica $\underset{S\sim\mathcal{D}^m}{P}(Q) \leq 2 \cdot \underset{\substack{S_1\sim\mathcal{D}^m \\ S_2\sim\mathcal{D}^m}}{P}(R)$. Veámoslo.

Aplicando desigualdad de Hoeffding 3.4.3, sabemos que fijado $h \in H$ tenemos:

$$\underset{S_2\sim\mathcal{D}^m}{P}\left[|L_{\mathcal{D}}(h) - L_{S_2}(h)| \leq \frac{\varepsilon}{2}\right] \geq (1 - 2e^{-m\varepsilon^2/2}) \underset{m\geq 4/\varepsilon^2}{\geq} (1 - 2e^{-2}) \geq \frac{1}{2} \tag{†}$$

Dado $h \in H$ verificando $|L_{\mathcal{D}}(h) - L_{S_1}(h)| \geq, \varepsilon\ |L_{\mathcal{D}}(h) - L_{S_2}(h)| \leq \frac{\varepsilon}{2}$ entonces por desigualdad triangular tenemos $|L_{S_1}(h) - L_{S_2}(h)| \geq \frac{\varepsilon}{2}$. Así:

$$\underset{\substack{S_1\sim\mathcal{D}^m \\ S_2\sim\mathcal{D}^m}}{P}(R) \geq \underset{\substack{S_1\sim\mathcal{D}^m \\ S_2\sim\mathcal{D}^m}}{P}\left[\exists h \in H : |L_{\mathcal{D}}(h) - L_{S_1}(h)| \geq \varepsilon, |L_{\mathcal{D}}(h) - L_{S_2}(h)| \leq \frac{\varepsilon}{2}\right]$$
$$\underset{(†)}{\geq} \frac{\underset{S_1\sim\mathcal{D}^m}{P}(Q)}{2} = \frac{\underset{S\sim\mathcal{D}^m}{P}(Q)}{2} \tag{∗}$$

Sea en lo que sigue $S = (S_1, S_2) \sim \mathcal{D}^{2m}$.

Consideramos ahora un conjunto de permutaciones Γ sobre $\{1, \ldots, 2m\}$ verificando que para todo $i \in \{1, \ldots, m\}$ entonces o bien $\tau(i) = i + m, \tau(i + m) = i$ o bien $\tau(i) = i, \tau(i + m) = i$. Es decir son permutaciones que intercambian (o no) posiciones de S_1 y S_2. Asignaremos a Γ una distribución uniforme U^Γ y llamaremos $P_\tau := \underset{\tau\sim U^\Gamma}{P}$. Se verificará:

$$\underset{S\sim\mathcal{D}^{2m}}{P}(R) = \underset{S\sim\mathcal{D}^{2m}}{\mathbb{E}}\left(P_\tau[\tau(S) \in R]\right) \leq \underset{S\in Z^{2m}}{\max} P_\tau[\tau(S) \in R] \tag{∗∗}$$

Veámoslo. Usando que S es una muestra i.i.d. escogida desde $\mathcal{D}^{2m}$:

$$\underset{S\sim\mathcal{D}^{2m}}{P}(R) = \underset{S\sim\mathcal{D}^{2m}}{P}[S \in R] = \underset{S\sim\mathcal{D}^m}{P}[\tau(S) \in R], \quad \forall \tau \in \Gamma$$

Por otro lado, fijada $\tau \in \Gamma$:

$$\underset{S\sim\mathcal{D}^{2m}}{P}(R) = \underset{S\sim\mathcal{D}^{2m}}{\mathbb{E}}(\mathbb{1}_R(S)) = \underset{S\sim\mathcal{D}^{2m}}{\mathbb{E}}(\mathbb{1}_R(\tau(S))) = \frac{1}{|\Gamma|}\sum_{\tau\in\Gamma}\underset{S\sim\mathcal{D}^{2m}}{\mathbb{E}}(\mathbb{1}_R(\tau(S))) =$$

$$\underset{\text{(linealidad de } \mathbb{E})}{=} \underset{S\sim\mathcal{D}^{2m}}{\mathbb{E}}\left(\frac{1}{|\Gamma|}\sum_{\tau\in\Gamma}[\mathbb{1}_R(\tau(S))]\right) = \underset{S\sim\mathcal{D}^{2m}}{\mathbb{E}}\Big(P_\tau[\tau(S)\in R]\Big)$$

Ahora la desigualdad $\underset{S\sim\mathcal{D}^{2m}}{\mathbb{E}}\Big(P_\tau[\tau(S)\in R]\Big) \leq \text{máx}_{S\in Z^{2m}}\, P_\tau[\tau(S)\in R]$ es trivial sin más que asegurar que existe el máximo en $S \in Z^{2m}$ de $P_\tau[\tau(S)\in R]$ puesto que P_τ toma finitos valores sobre su σ álgebra correspondiente.

Por último veamos que:

$$\underset{S\in Z^{2m}}{\text{máx}}\, P_\tau[\tau(S)\in R] \leq 2\Pi_H(2m)e^{-\varepsilon^2 m/8} \qquad (***)$$

Sea $S = \{(x_1,y_1),\dots(x_{2m},y_{2m})\}$ un conjunto de entrenamiento que maximice $P_\tau[\tau(S)\in R]$. Llamando $C = \{x_1,\dots x_{2m}\}$, entonces $H_C = \{h_1,\dots,h_t\}$, con $t \leq \Pi_H(2m)$, por definición de Π_H.

Dado $\tau \in \Gamma$, entonces $\tau(S) \in R$ sii para algún $h \in H_C$ se tiene (con l función de pérdida $0-1$):

$$\frac{1}{m}\left|\sum_{i=1}^{m} l\big(h,(x_{\tau(i)},y_{\tau(i)})\big) - \sum_{i=m+1}^{2m} l\big(h,(x_{\tau(i)},y_{\tau(i)})\big)\right| \geq \frac{\varepsilon}{2}$$

Fijado $h_j \in H_C$ llamamos $u_k^j = l\big(h_j,(x_k,y_k)\big)$ para $k = 1,\dots,2m$, $j = 1,\dots,t$.

Se tiene que $u^j_{\tau(i)} - u^j_{\tau(m+i)}$ vale o $u_i^j - u_{m+1}^j$ o $u_{m+1}^j - u_i^j$ con igual probabilidad, por haber asignado sobre Γ una distribución uniforme. Por tanto considerando una distribución uniforme $U^{\pm}$ sobre $\{1,-1\}^m$.

$$P_\tau\left(\frac{1}{m}\left|\sum_{i=1}^{m}(u^j_{\tau(i)} - u^j_{\tau(i+m)})\right| \geq \frac{\varepsilon}{2}\right) = \underset{(\beta_1,\dots,\beta_m)\sim U^{\pm}}{P}\left(\frac{1}{m}\left|\sum_{i=1}^{m}(u_i^j - u_{i+m}^j)\beta_i\right| \geq \frac{\varepsilon}{2}\right)$$

Dado $\beta = (\beta_1,\dots\beta_m) \in \{1,-1\}^m$, entonces si $\sum_{i=1}^m (u_i^j - u_{i+m}^j)\beta_i = \alpha$, tomando $-\beta$, se tiene que $\sum_{i=1}^m (u_i^j - u_{i+m}^j)(-\beta_i) = -\alpha$. Por tanto podemos agrupar las permutaciones de dos en dos, con valores opuestos, luego:

$$\underset{\beta\sim U^{\pm}}{\mathbb{E}}\left[\frac{1}{m}\sum_{i=1}^{m}(u_i^j - u_{i+m}^j)\beta_i\right] = 0$$

Estamos en condiciones de aplicar desigualdad 3.4.3 de Hoeffding, con $W_i = (u_i^j - u_{i+m}^j) \cdot \beta_i$, donde $-1 \leq W_i \leq 1$. Deducimos:

$$\underset{\beta \sim U^{\pm}}{P}\left(\frac{1}{m}\left|\sum_{i=1}^{m}(u_i^j - u_{i+m}^j)\beta_i\right| \geq \frac{\varepsilon}{2}\right) \leq 2e^{-2m(\epsilon/4)^2} = 2e^{-\epsilon^2 m/8}$$

Como habíamos fijado un $h_j \in H_C$,

$$\begin{aligned} P_\tau[\tau(S) \in R] &= P_\tau\left[\exists h_j \in H_C : \frac{1}{m} : \left|\sum_{i=1}^{m}(u^j_{\tau(i)} - u^j_{\tau(i+m)})\right| \geq \frac{\varepsilon}{2}\right] \\ &\leq 2|H_C|e^{-\varepsilon^2 m/8} \leq 2\Pi_H(2m)e^{-\varepsilon^2 m/8} \end{aligned}$$

En definitiva, uniendo (∗), (∗∗) y (∗∗∗) hemos probado que:

$$\underset{S \sim \mathcal{D}^m}{P}\left[|L_S(h) - L_{\mathcal{D}}(h)| \geq \varepsilon\right] \leq 4\Pi_H(2m)e^{-\varepsilon^2 m/8}$$

para $m \geq \frac{4}{\varepsilon^2}$, que era condición suficiente para que se verificara (∗).

Ya estamos en condiciones de probar (vi) ⇒ (i). Sea $VC(H) = d < \infty$. Por lema 7.2.5 de Sauer se tiene $\Pi_H(2m) \leq (2m)^d + 1 \leq 2(2m)^d$. Podemos suponer s.p.g. que $d > 0$.

$$\underset{S \sim \mathcal{D}^m}{P}\left[|L_S(h) - L_{\mathcal{D}}(h)| \geq \varepsilon\right] \leq 4\Pi_H(2m)e^{-\varepsilon^2 m/8} \leq 8(2m)^d e^{-\varepsilon^2 m/8}$$

Necesitamos $8(2m)^d e^{-\varepsilon^2 m/8} \leq \delta$. Tomando logaritmos en ambos miembros deducimos:

$$\begin{aligned} \log(8) + d\log(2m) &\leq \log(\delta) + \frac{2\varepsilon^2 m}{16} \Leftrightarrow \\ \Leftrightarrow \underbrace{\frac{16}{\varepsilon^2}\Big(log(8) - log(\delta)\Big)}_{b} + \underbrace{\frac{16d}{\epsilon^2}}_{a}\log(2m) &\leq \underbrace{2m}_{x} \end{aligned}$$

$a \geq \frac{1}{2}$ por ser $d \geq 1$, y podemos aplicar el lema 7.3.2, que nos asegura que si $2m \geq 4a\log(2a) + 2b$, se verifica la anterior desigualdad. Es decir, nos vale:

$$m_H^{CU}(\epsilon, \delta) \leq \left\lceil \frac{32d}{\epsilon^2}\log\left(\frac{32d}{\epsilon^2}\right) + \frac{16}{\epsilon^2}\log\left(\frac{8}{\delta}\right)\right\rceil$$

□

8

APRENDIZAJE NO UNIFORME

Introduciremos el concepto de aprendizaje no-uniforme, relajando la definición de agnósticamente PAC cognoscible. También proporcionaremos una noción de codificación para las clases de hipótesis numerables, y otro paradigma para aprenderlas, que nos llevará al principio de la *navaja de Occam* en el contexto del aprendizaje automático.

Definición 8.0.1 (Aprendizaje no-uniforme). *Una clase de hipótesis H sobre $Z = \mathcal{X} \times \mathcal{Y}$ es no-uniformemente PAC cognoscible si existe un algoritmo A y una función $m_H^{NU} :]0,1[^2 \times H \to \mathbb{N}$ verificando que dados $0 < \varepsilon, \delta < 1$ y $h \in H$, entonces para toda distribución $\mathcal{D}$ sobre Z y $m \geq m_H^{NU}(\varepsilon, \delta, h)$ se verifica:*

$$\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(h) + \varepsilon\right] \geq 1 - \delta$$

8.1 MINIMIZACIÓN DEL RIESGO ESTRUCTURAL

Hasta ahora hemos traducido el conocimiento previo sobre el problema como una restricción global en la clase de hipótesis para la minimización del riesgo empírico. Ahora generalizaremos esto aún más y estableceremos la suposición de que $H = \cup_{n\in\mathbb{N}} H_n$, donde cada clase H_n tendrá asignado un peso $w(n)$.

Proposición 8.1.1. *Sea $w : \mathbb{N} \to [0,1]$ verificando $\sum_{n=1}^{\infty} w(n) \leq 1$, $H \subseteq 2^X$ que puede ser escrita como $H = \cup_{n\in\mathbb{N}} H_n$, donde cada H_n es una clase de Glivenko-Cantelli. Escogemos:*

$$\varepsilon_n : \mathbb{N}\times]0,1[\to]0,1[, \qquad \varepsilon_n(m,\delta) \in \{\varepsilon \in]0,1[: m_{H_n}^{UC}(\varepsilon,\delta) \leq m\}$$

Entonces para todo $\delta \in]0,1[$ y para toda distribución $\mathcal{D}$ se verifica:

$$\underset{S\sim\mathcal{D}^m}{P}\left[\forall n \in \mathbb{N}, \forall h \in H_n, |L_{\mathcal{D}}(h) - L_S(h)| \leq \varepsilon_n(m, w(n)\delta)\right] \geq 1 - \delta$$

En particular, tomando $n(h) \in \{n : h \in H_n\}$ para cada $h \in H$:

$$\underset{S\sim\mathcal{D}^m}{P}\left[\forall h \in H, |L_{\mathcal{D}}(h) - L_S(h)| \leq \varepsilon_{n(h)}(m, w(n(h))\delta)\right] \geq 1 - \delta$$

Demostración. Fijado n, por ser H_n de Glivenko-Cantelli:

$$\underset{S\sim\mathcal{D}^m}{P}\left[\forall h \in H_n, |L_{\mathcal{D}}(h) - L_S(h)| \leq \varepsilon_n(m, w(n)\delta)\right] \geq 1 - w(n)\delta$$

Por tanto:

$$\begin{aligned}
&\underset{S\sim\mathcal{D}^m}{P}\left[\forall n \in \mathbb{N}, \forall h \in H_n, |L_{\mathcal{D}}(h) - L_S(h)| \leq \varepsilon_n(m, w(n)\delta)\right] \\
&= 1 - \underset{S\sim\mathcal{D}^m}{P}\left[\exists n \in \mathbb{N}, \exists h \in H_n, |L_{\mathcal{D}}(h) - L_S(h)| > \varepsilon_n(m, w(n)\delta)\right] \\
&\geq 1 - \sum_{n=1}^{\infty} w(n)\delta \geq 1 - \delta
\end{aligned}$$

□

Este teorema motiva las definiciones de riesgo estructural y minimizador de dicho riesgo.

Definición 8.1.2 (Riesgo estructural). *Sea $H = \cup_{n\in\mathbb{N}} H_n$ donde H_n es de Glivenko-Cantelli y tenemos una función de peso $w : \mathbb{N} \to [0,1]$, $\sum_n w(n) \leq 1$. Dado $h \in H$, escogemos:*

$$\varepsilon_n(m,\delta) = \min_{\varepsilon\in]0,1[}\{m_{H_n}^{UC}(\varepsilon,\delta) \leq m\}, \quad n(h) = min\{n : h \in H_n\}$$

Dado $S \in (X \times Y)^m$, se llama riesgo estructural de $h \in H$, respecto a ε_n, w:

$$L_S(h) + \varepsilon_{n(h)}\left(m, w(n(h))\frac{1}{m}\right)$$

Por sencillez de las demostraciones, supondremos que existe el mínimo de $\{m_{H_n}^{UC}(\epsilon,\delta)\}$ y por tanto podemos definir ε_n.

Definición 8.1.3 (Minimizador del riesgo estructural, SRM). *Decimos que un algoritmo $A : \bigcup_{m\in\mathbb{N}} (X \times Y)^m \to H$ es un SRM (Structural Risk Minimizer) si busca una hipótesis cuyo error estructural sea mínimo, suponiendo que dicho mínimo tiene sentido. Dado $S \in (X \times Y)^m$ devolverá:*

$$A(S) \in \arg\min_{h\in H} L_S(h) + \varepsilon_{n(h)}\left(m, w(n(h))\frac{1}{m}\right)$$

Es conocido que $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$. Usando este hecho y el anterior resultado, podemos dar condiciones suficientes para que un *SRM* haga a una clase no-uniformemente cognoscible.

Teorema 8.1.4. *Sea H una clase de hipótesis verificando $H = \cup_{n\in\mathbb{N}} H_n$ donde cada H_n es de Glivenko-Cantelli. Sea $w : \mathbb{N} \to [0,1]$ dada por $w(n) = \frac{6}{(\pi n)^2}$. Entonces H es no-uniformemente cognoscible para cualquier algoritmo A que sea un SRM. Además se verifica:*

$$m_H^{NU}(\varepsilon,\delta,h) \leq \left\{ m_{H_{n(h)}}^{CU}\left(\varepsilon/2, \frac{3\delta}{(\pi n(h))^2}\right), \left\lceil \frac{2}{\delta} \right\rceil \right\}$$

Demostración. Fijamos $\bar{h} \in H$ y $0 < \delta, \varepsilon < 1$. Sea:

$$m \geq \text{máx} \left\{ m_{H_{n(\bar{h})}}^{CU}\left(\varepsilon/2, \frac{3\delta}{(\pi n(\bar{h}))^2}\right), \left\lceil \frac{2}{\delta} \right\rceil \right\}$$

Claramente $\varepsilon_{n(\bar{h})}\left(m, w(n(\bar{h}))\frac{1}{m}\right) \leq \varepsilon/2$.

Usando el hecho de que $\sum_{n\geq 1} w(n) = 1$, por proposición 8.1.1 se verifica:

$$\underset{S\sim\mathcal{D}^m}{P}\left[\forall h \in H | L_{\mathcal{D}}(h) - L_S(h)| \leq \varepsilon_{n(h)}\left(m, w(n(h))\frac{1}{m}\right)\right] \geq 1 - \frac{1}{m} \geq 1 - \frac{\delta}{2} \qquad (*)$$

Por ser A un SRM, entonces dado $S \in (X \times Y)^m$:

$$\begin{aligned} L_S(A(S)) + \varepsilon_{n(A(S))}\left(m, w(n(A(S)))\frac{1}{m}\right) &\leq L_S(\bar{h}) + \varepsilon_{n(\bar{h})}\left(m, w(n(\bar{h}))\frac{1}{m}\right) \\ &\leq L_S(\bar{h}) + \frac{\varepsilon}{2} \qquad (**) \end{aligned}$$

A partir de $(*)$ y $(**)$, deducimos:

$$\begin{aligned} &\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) \leq L_S(\bar{h}) + \varepsilon/2\right] \geq 1 - \frac{\delta}{2} \\ \Leftrightarrow &\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) > L_S(\bar{h}) + \varepsilon/2\right] \leq \frac{\delta}{2} \qquad (\dagger) \end{aligned}$$

Además, usando convergencia uniforme, con probabilidad menor o igual que $\delta/2$:

$$L_S(\bar{h}) > L_{\mathcal{D}}(\bar{h}) + \varepsilon/2 \qquad (\dagger\dagger)$$

Uniendo $(\dagger)$ y $(\dagger\dagger)$ deducimos, por subaditividad:

$$\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) > L_S(\bar{h}) + \varepsilon/2 \bigvee L_S(\bar{h}) > L_{\mathcal{D}}(\bar{h}) + \varepsilon/2\right] \leq \delta$$

Es decir:

$$\underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) \leq L_{\mathcal{D}}(\bar{h}) + \varepsilon\right] \geq 1 - \delta$$

□

Teorema 8.1.5 (Caracterización de aprendizaje no-uniforme). *Una clase de hipótesis $H \subseteq 2^X$ es no-uniformemente cognoscible sii $H = \bigcup_{n\in\mathbb{N}} H_n$ donde cada H_n es APAC cognoscible.*

Demostración. Supongamos $H = \cup_{n\in\mathbb{N}} H_n$ donde cada H_n es agnósticamente PAC cognoscible. Por teorema fundamental de aprendizaje PAC, 7.3.3, cada clase H_n tiene la propiedad de convergencia uniforme, y podemos aplicar el teorema anterior.

En el sentido opuesto, sea H no-uniformemente cognoscible usando un cierto algoritmo A. Fijamos $\delta < \frac{1}{7}$ y definimos:

$$H_n = \{h \in H : m_H^{NU}(1/8, \delta, h) \leq n\}$$

para todo n natural. Claramente $H = \cup_{n\in\mathbb{N}} H_n$.

Supongamos $VC(H_n) = \infty$ para algún $n \in \mathbb{N}$. Entonces por teorema de No Free Lunch, 7.1.4, tendríamos que existe $\mathcal{D}$ una distribución sobre X verificando:

$$\exists \bar{h} \in H_n : L_{\mathcal{D}}(\bar{h}) = 0 \qquad \text{y} \qquad \underset{S\sim\mathcal{D}^m}{P}\left[L_{\mathcal{D}}(A(S)) > \frac{1}{8}\right] \geq \frac{1}{7}$$

lo cual para $m > m_H^{NU}(\frac{1}{8}, \delta, \bar{h})$ constituiría una contradicción. □

Corolario 8.1.6. *La noción de cognoscibilidad no-uniforme es más fuerte que la APAC cognoscibilidad.*

Demostración. Dada H APAC cognoscible, por el teorema anterior es no-uniformemente cognoscible.

Por otro lado $H = \{h : \mathbb{R} \to \{-1, 1\} : h^{-1}(1) \text{ es finito}\}$ es no-uniformemente cognoscible, pero no es APAC cognoscible. Veámoslo.

Escribimos $H_n = \{h \in H : |h^{-1}(1)| \leq n\}$. Es claro que $H = \bigcup_{n\in\mathbb{N}} H_n$. Además, $VC(H_n) = n$, porque no podríamos fragmentar conjuntos de tamaño mayor que n, al ser $h^{-1}(1)$ de cardinal n como mucho. Es trivial ver que sí podemos fragmentar conjuntos de tamaño n. Luego por teorema fundamental del PAC, 7.3.3, cada H_n sería APAC cognoscible, y el teorema anterior nos dice que H es no-uniformemente cognoscible.

Pero $VC(H) = \infty$, porque $VC(H) \geq VC(H_n) = n$ para $n \in \mathbb{N}$ arbitrario, luego H no es APAC cognoscible. □

8.2 MINIMIZACIÓN DE LONGITUD DESCRIPTIVA

Definición 8.2.1 (Lenguaje de descripción de hipótesis). *Sea H una clase de hipótesis y Γ un conjunto finito de símbolos, que llamaremos alfabeto. Notaremos al lenguaje generado por Γ:*

$$\Gamma^* = \{(\gamma_1, \ldots, \gamma_m) : \gamma_i \in \Gamma, m \in \mathbb{N}\}$$

donde la longitud de una palabra $\gamma = (\gamma_1, \ldots, \gamma_m) \in \Gamma$ será m y lo notaremos $|\gamma| = m$.

Un lenguaje de descripción para H será una asignación $d : H \to \Gamma^*$, libre de prefijos, esto es, dados $h, h' \in H$ distintos, con $|d(h)| \leq |d(h')|$, los $|d(h)|$ primeros símbolos de $d(h')$ no podrán ser $d(h)$ (al menos no en ese orden).

Dado un lenguaje de descripción de H, a saber, d, notaremos $|h| = |d(h)|$.

Por simplicidad, consideraremos $\Gamma = \{a, b\}$ en lo que sigue.

Lema 8.2.2 (Desigualdad de Kraft). *Si $\mathcal{L} \subseteq \Gamma^*$ es un conjunto de palabras provenientes de un lenguaje libre de prefijos, entonces:*

$$\sum_{\alpha \in \mathcal{L}} \frac{1}{2^{|\alpha|}} \leq 1$$

Demostración. Como las palabras del lenguaje son libres de prefijos, podríamos representar $\mathcal{L}$ con un árbol binario, donde cada palabra sería el camino más corto desde una hoja a la raíz del árbol ($\emptyset$). Recíprocamente, cualquier árbol binario estaría representando un lenguaje libre de prefijos, si consideramos la misma regla para formar palabras.

Hacemos la demostración sobre la altura del árbol, identificando cada hoja de un árbol, (T, E), con una palabra.

> Para un árbol de altura 0, tendríamos la palabra vacía, que tendría longitud 0, y por tanto $\frac{1}{2^0} = 1$.
>
> Supuesto que se cumple para árboles de hasta altura n, y sea un árbol de altura $n+1$, a saber (T, E). Sean J sus hojas. Si podásemos las ramas de longitud $n+1$, obtendríamos un árbol binario de altura n, con hojas J' por lo que cumpliría:
>
> $$\sum_{l' \in J'} \frac{1}{2^{|l'|}} \leq 1$$
>
> Pero la rama correspondiente a una hoja l' podría dividirse como mucho en dos ramas para volver a tener la rama original l. Si se

dividiese en una, tendríamos $|l| > |l'|$ y por tanto $\frac{1}{2^{|l|}} \leq \frac{1}{2^{|l'|}}$. Si se dividiese en dos, tendríamos dos factores que valdrían la mitad, y al sumarse, $\frac{1}{2^{|l'|}}$. Si no se dividiese en ninguna, no se alteraría el factor correspondiente.

□

Proposición 8.2.3. *Sea $H \subseteq 2^X$ una clase de hipótesis numerable, un lenguaje de descripción $d : H \to \{a,b\}^*$. Entonces para cualquier $m \in \mathbb{N}$, cualesquiera $0 < \delta, \epsilon < 1$ y una distribución $\mathcal{D}$ sobre X arbitraria, se cumpliría:*

$$\underset{S\sim\mathcal{D}^m}{P}\left[\forall h \in H, L_{\mathcal{D}}(h) \leq L_S(h) + \sqrt{\frac{|h| + \log(2/\delta)}{2m}}\right] \geq 1 - \delta$$

Demostración. Sea $H = \bigcup_{n\in\mathbb{N}}\{h_n\}$.

Por proposición 5.2.2 tendríamos que $m^{CU}_{\{h_n\}} \leq \left\lceil \frac{\log(2/\delta)}{2\epsilon^2} \right\rceil$ para cada $n \in \mathbb{N}$ arbitrario, respecto de la función de pérdida $0-1$.

Luego $\varepsilon_n(m,\delta) = \sqrt{\frac{\log(2/\delta)}{2m}}$ y el *SRM* estaría bien definido. Tomando $w(n) = \frac{1}{2^{|h_n|}}$, donde $|h_n|$ es la longitud dada por el lenguaje de descripción d, el lema 8.2.2 de Kraft nos diría $\sum_{n\geq 1} w(n) \leq 1$.

Aplicando la proposición 8.1.1 tendríamos que:

$$\underset{S\sim\mathcal{D}^m}{P}\left[\forall h \in H, L_{\mathcal{D}}(h) \leq L_S(h) + \sqrt{\frac{-\log(1/2^{|h|}) + \log(2/\delta)}{2m}}\right] \geq 1 - \delta$$

Pero como $-\log(1/2^{|h|}) = |h|\log(2) \leq h$, y $\sqrt{(\cdot)}$ es creciente, por subaditividad:

$$\underset{S\sim\mathcal{D}^m}{P}\left[\forall h \in H, L_{\mathcal{D}}(h) \leq L_S(h) + \sqrt{\frac{|h| + \log(2/\delta)}{2m}}\right] \geq 1 - \delta$$

□

Este teorema, adaptación de 8.1.1 al caso de H numerable y existencia de un lenguaje de descripción, motiva que adaptemos también el *SRM* a dicho paradigma.

Definición 8.2.4 (Minimizador de la longitud descriptiva). *Decimos que un algoritmo $A : \bigcup_{m\in\mathbb{N}} (X \times Y)^m \to H$ es un DLM (Description Length Minimizer) si dado $S \in (X \times Y)^m$ entonces:*

$$A(S) \in \arg\min_{h\in H} L_S(h) + \sqrt{\frac{|h| + \log(2m)}{2m}}$$

El teorema 8.1.4 nos dice que para una clase H numerable y d su lenguaje de descripción, un DLM hace a H no-uniformemente cognoscible.

8.2.1 *Navaja de Occam*

De la proposición 8.2.3 se desprende que tomando dos $h, h' \in H$ con igual $L_S(h)$, sería preferible tomar siempre la hipótesis de mínima longitud de descripción $|h|$. Este principio es conocido como *navaja de Occam*, y establece que una hipótesis sencilla (de descripción corta) tiende a ser más válida que una complicada (de descripción más larga).

8.3 APRENDIZAJE PAC VS APRENDIZAJE NO-UNIFORME

En términos de error, ambos paradigmas proporcionan una cota del error cometido, que se cumple con cierta probabilidad. Ahora bien, en paradigma no-uniforme no podemos conocer a priori cuántos ejemplos son necesarios para aprender la mejor hipótesis, mientras que en aprendizaje PAC sí le pedimos esta condición al algoritmo y la función de complejidad muestral.

Puesto que ambos paradigmas proporcionan cotas para err_{est}, sabemos qué parte del error corresponde a la aproximación y cuál a la estimación. Si el error de aproximación es muy alto, siempre podemos intentar aprender con otra clase H en PAC o modificar el esquema de asignación de pesos $w(n)$ en aprendizaje no-uniforme.

La pregunta ¿cómo aprender? es altamente dependiente del problema. Ambos paradigmas nos permiten usar conocimiento previo para que un problema sea resoluble: PAC restringiendo H y aprendizaje no-uniforme escogiendo $w(n)$. El aprendizaje no-uniforme es más flexible que el aprendizaje PAC, pero proporciona menos garantías en base al número de ejemplos que necesitamos para aprender.

Parte II

INFORMÁTICA

9

CLASIFICACIÓN BINARIA DESBALANCEADA

9.1 PROBLEMA DE DESBALANCEO

Se considerará en toda la sección de informática clasificación binaria, donde tenemos un dominio X, y un conjunto de etiquetas $Y = \{-1, 1\}$. Llamamos $Z = X \times Y$. También consideraremos que los datos siguen una distribución dada por una función de probabilidad prefijada pero no conocida por el algoritmo, a saber, $\mathcal{D} = (\Sigma, P)$.

Por simplicidad, podemos considerar siempre $\Sigma = \mathcal{P}(X)$, por lo que identificaremos en lo que sigue $\mathcal{D} \equiv P$. El problema de clasificación binaria consiste en buscar, dado $S = \{(x_1, y_1), \dots (x_m, y_m)\}$ un conjunto de entrenamiento, un clasificador de alguna clase H, h, verificando que $L_P(h)$ sea lo más pequeño posible. Nótese que estamos relajando las hipótesis de 4.4.1 al considerar la distribución de probabilidad como prefijada.

Definición 9.1.1 (Instancias positivas, negativas). *LLamamos:*

- *Instancias positivas a* $Z^+ = \{(x, y) \in Z : y = 1\}$.
- *Instancias negativas a* $Z^- = \{(x, y) \in Z : y = -1\}$.

Una notación equivalente se aplica a S, conjunto de entrenamiento. Además llamamos $P^+ = P(Z^+)$ *y* $P^- = P(Z^-)$.

Definición 9.1.2 (Desbalanceo entre clases). *Decimos que un problema de clasificación binaria es desbalanceado (entre clases) si se verifica que tomando* $N^- = |S^-|$ *y* $N^+ = |S^+|$, *entonces* $N^+ < N^-$.

Llamamos ratio de desbalanceo a $r(S) = \frac{N^+}{N^-}$.

Por convención se consideran problemas de desbalanceo donde $r(S) \ll 1$, tal y como se afirma en [He and Garcia, 2009]. Asimismo, en estos problemas nos interesa especialmente mejorar el error de los clasificadores sobre la clase minoritaria o positiva.

En innumerables ocasiones, al realizar análisis de datos, no se tiene en cuenta el desbalanceo existente entre clases, lo cual hace que los algoritmos no reflejen adecuadamente la distribución de las clases.

Para ciertos conjuntos de datos, donde el número de instancias positivas con respecto a las negativas es muy pequeño, un clasificador podría simplemente etiquetar todas las instancias como negativas, y obtener un error global pequeño, pero no acertaría en ninguna instancia positiva. Si los datasets empleados representaran enfermedades, por ejemplo, como es el caso de muchos de los disponibles en el repositorio [KEEL], algunos de los cuales se han integrado dentro del paquete, nos interesaría encontrar mecanismos para disminuir el error sobre la clase positiva (la de las instancias correspondientes a la enfermedad).

El desbalanceo puede ser:

i **Intrínseco**, cuando $P^+ < P^-$, y por tanto la cantidad de datos de cada clase que extraemos viene verdaderamente determinada por la distribución real de los datos.

ii **Extrínseco**, si $P^+ \geq P^-$ pero sin embargo para nuestro conjunto de entrenamiento S, tenemos que $N^+ < N^-$. Esto quiere decir que el desbalanceo no viene determinado por la verdadera distribución de los datos, sino por factores como por ejemplo la complejidad para extraer datos de alguna de las clases.

También podemos decir que existe desbalanceo *intra clases* cuando además la clase minoritaria está repartida en varias regiones no conexas, a saber $S^+ \supset A_1, \ldots, A_m$ donde $|A_1| \leq \ldots \leq |A_m|$, con algún menor o igual estricto. Esto dificulta aún más la correcta clasificación de dichas instancias, porque aparecen zonas con instancias positivas pobremente representadas. A estas zonas de baja representación las llamamos instancias raras. Nótese que hay que distinguir el concepto de instancia rara de otro concepto que surge en ciencia de datos: el concepto de ruido, instancias aisladas completamente.

La aparición de instancias raras no sólo se debe al desbalanceo de clases, sino que se puede deber a un amplio rango de factores que afectan a la complejidad de los datos: *overlapping* ($S_x^- \cap S_x^+ \neq \emptyset$), falta de representatividad en los datos, *small disjuncts*, etc. En general, los clasificadores intentan aprender a partir de una clase creando reglas (conceptos) disjuntas que agrupen a instancias. Como consecuencia de la infrarrepresentación de instancias y del desbalanceo *intra clases*, podemos tener reglas que cubran una pequeña porción de las instancias de la clase positiva. Estas instancias conforman lo que llamamos un *small disjunct*. Los *small disjuncts* no sólo afectan a la clase minoritaria, sino que pueden darse también dentro de la mayoritaria, aunque la mayor densidad de datos de esta clase hace que el efecto no sea tan agravado o sea una situación menos frecuente. De hecho hay estudios como [Jo and Japkowicz, 2004] que sostienen que no só-

lo el desbalanceo de clases es el responsable de la pérdida de acierto de los clasificadores, sino que los *small disjuncts* son un factor muy a tener en cuenta en la degradación del rendimiento de los mismos.

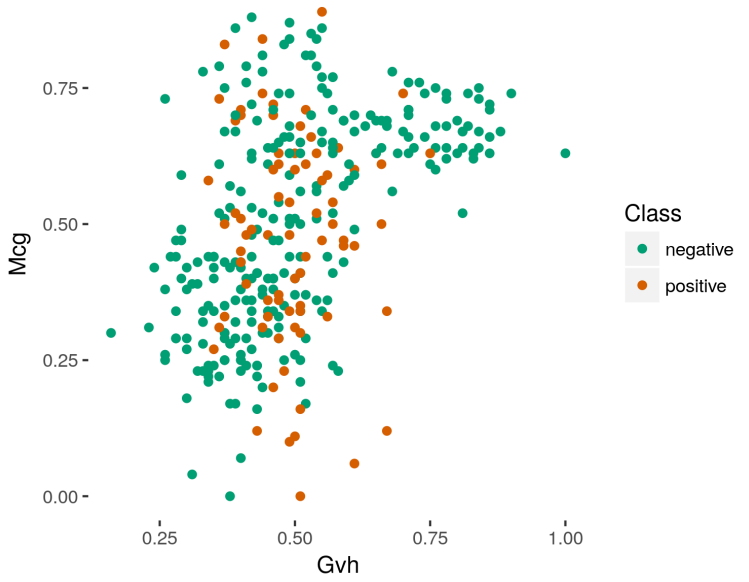


Figura 3.: Ejemplo de dataset con desbalanceo

9.2 TÉCNICAS PARA APRENDIZAJE CON DESBALANCEO

9.2.1 *Oversampling/Undersampling*

El *oversampling* consiste en generar instancias sintéticas etiquetadas como positivas, E, normalmente respetando la distribución de S^+, intentando generar conexos a partir de instancias raras, etc. Se lleva a cabo la tarea de clasificación usando el conjunto de entrenamiento $S' = S \cup E$.

El *undersampling* consiste en escoger un subconjuto $E \subseteq S^-$, de manera que $S' = S \setminus E$ y tenemos un problema de clasificación con el conjunto de entrenamiento S'. Normalmente E se genera de manera que se intentan eliminar instancias que aportan información redundante dentro del conjunto de instancias negativas.

Suponemos también que tanto el *oversampling* como el *undersampling* se realiza de manera que $r(S') > r(S)$.

La gran ventaja de los métodos de *sampling* respecto a otros como *cost-sensitive* es su flexibilidad, puesto que podemos aplicárselos a cualquier clasificador.

Limpieza de instancias en oversampling

Estas técnicas se emplean para limpiar los bordes de las zonas que conectan a ambas clases. El método más reseñable son los *enlaces Tomek*. Se dice que dos instancias x_i, x_j forman un *enlace Tomek* si $d(x_i, x_j) = \arg\min_{x_p, x_q} d(x_p, x_q)$. Por tanto si dos instancias corresponden a un *enlace Tomek* significará, a no ser que correspondan a ruido, que están en la frontera entre las dos clases. Por tanto dichos enlaces pueden usarse para una vez se ha hecho *oversampling* de la clase minoritaria, ir eliminando iterativamente todos los *enlaces Tomek* cuyas dos instancias pertenezcan a clases distintas, hasta que los únicos enlaces que resten pertenezcan a la misma clase.

También destaca el método ENN (*Edited Nearest Neighbour*), que consiste en eliminar todas aquellas instancias tales que al calcular sus k vecinos más cercanos, haya una mayoría pertenecientes a la otra clase de entre los mismos.

9.2.2 *Aprendizaje cost sensitive*

El *framework* de aprendizaje *cost sensitive* implica que no a todas las instancias le asignamos el mismo error al aprender (como podría ser una función de pérdida $0-1$ como las descritas en 4.4.3). Una técnica para usar funciones de pérdida aplicadas al problema de desbalanceo consiste en usar una matriz $\{C_{i,j}\}_{i,j\in\{-1,1\}}$, donde $C_{i,j}$ reprensenta el coste de asignar a la una instancia de clase i, la clase j, siendo $C_{i,i} = 0$; es habitual tomar $C_{1,-1} > C_{-1,1}$ para dar más importancia a la clase positiva.

Dentro de este paradigma destacan métodos de *boosting* y de *ensamble* de clasificadores. También es habitual encontrarnos algoritmos clásicos modificados para implementar *cost sensitive* hacia la clase minoritaria.

9.3 MEDIDAS DE LA BONDAD DEL APRENDIZAJE

VP	FP
FN	VN

Cuadro 1.: Matriz de confusión

Para definir medidas de bondad necesitaremos tener en cuenta la matriz de confusión, donde VP (verdaderos positivos) representará el número de instancias positivas de S clasificadas como positivas; FP (falsos positivos) representará las instancias verdaderamente negativas pero que han sido clasificadas como positivas; FN (falsos negativos) reprensetará instancias positivas que han sido clasificadas como negativas; y VN (verdaderos negativos) reprensentará instancias negativas correctamente clasificadas como negativas. Obsérvese además $N^+ = VP + FN$ y $N^- = FP + VN$, con $N = N^+ + N^-$.

La medida de bondad habitual en clasificación es el acierto o exactitud: $e = \frac{VP+VN}{N}$. Esta medida presenta el problema en clasificación desbalanceada de que podríamos tener una exactitud muy alta, pero un acierto nulo sobre la clase minoritaria. Necesitamos medidas que prioricen el acierto sobre la clase positiva, o al menos lo ponderen más, en clasificación desbalanceada.

A tal efecto, podemos definir la precisión como $p = \frac{VP}{VP+FP}$, esto es, el nivel de acierto sobre las instancias clasificadas como positivas. También tenemos en cuenta la sensibilidad, $s = \frac{VP}{N^+}$ que cuantifica las instancias positivas verdaderamente etiquetadas como tales, es decir, lo completa que ha sido la clasificación en términos de la clase minoritaria, reteniendo la información necesaria para clasificar las instancias positivas como tales.

También se define el F-score como $F_\beta = \frac{1+\beta^2}{\frac{\beta^2}{s}+\frac{1}{p}}$, una media ponderada de la sensibilidad y la precisión, donde cuantificamos la precisión con un factor β de importancia sobre la sensibilidad. De hecho en el caso $\beta = 1$ tenemos una media armónica de precisión y sensibilidad. También puede expresarse como $F_\beta = \frac{(1+\beta^2)VP}{(1+\beta^2)VP+\beta^2 FN+FP}$.

Análogamente al F-score podemos definir la media geométrica de la precisión y la sensibilidad $G = \sqrt{s \cdot p}$, que cuantificaría ambas medidas por igual, siendo menos sensible a que una de ellas sea exageradamente buena y la otra no.

Como última medida que vamos a considerar, tenemos el AUC (*Area Under the Curve*). Para entender lo que es el AUC necesitamos antes

definir la curva *ROC* (*Receiver Operating Characteristic curve*), que se obtiene al representar en el eje de abcisas la tasa de falsos positivos o (1-especificidad) $\frac{FP}{N^-}$, y en el eje de ordenadas la tasa de verdaderos positivos o sensibilidad del clasificador, de manera que nos quedan puntos en el espacio $[0,1] \times [0,1]$. La curva *ROC* se obtiene al unir todos los puntos de los que dispongamos para un clasificador (por ejemplo si hacemos una validación cruzada, o si consideramos distribuciones acumuladas sobre los datos). Intuitivamente, cuanto más cerca del punto $(0,1)$ (acertamos todos los ejemplos verdaderamente positivos, sin ningún falso positivo), mucho mejor *AUC* obtendremos. La bisectriz $y = x$ marca el clasificador aleatorio (equivalente a lanzar una moneda al aire) que asigna la mitad de instancias a cada clase.

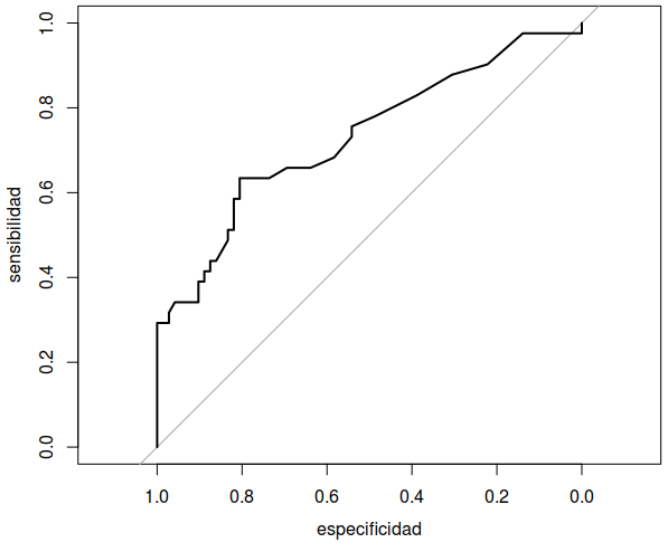


Figura 4.: Ejemplo de curva ROC

9.4 ESTADO DEL ARTE DEL OVERSAMPLING

A día de hoy, el algoritmo SMOTE, propuesto en [Chawla et al., 2002] sigue siendo una de las referencias en el campo del *oversampling* para clasificación desbalanceada, dada su idea simple pero eficaz de rellenar zonas visuales no cubiertas con instancias minoritarias a partir de instancias generadas en la recta que une otras dos minoritarias. Damos una breve descripción del algoritmo.

Algoritmo 1 Algoritmo de *oversampling* SMOTE

Entrada: $S^+ = \{x_1, \ldots, x_m\}$, instancias positivas
Entrada: T, número de instancias sintéticas deseado
Entrada: k, parámetro de KNN para crear instancias sintéticas

Inicializar $S' = \emptyset$, instancias sintéticas
Calcular los k vecinos más cercanos en S^+ para cada $x \in S^+$
Toma $M = \lceil T/|S^+| \rceil$

for $x \in S^+$ **do**
  for $m = 1, \ldots, M$ **do**
    Escoger un vecino más cercano aleatorio para x, a saber, y
    Seleccionar r en $[0, 1]$ de manera uniforme
    $S' = S' \cup \{x + r(y - x)\}$
  end for
end for
Hacer $S' = T$ instancias escogidas aleatoriamente desde S'

return S', ejemplos positivos sintéticos

10

ALGORITMOS IMPLEMENTADOS

10.1 ALGORITMO MWMOTE

El algoritmo 1 presentado anteriormente presenta el problema de que no incluye detección de las instancias en la frontera entre ambas clases, que son en muchas ocasiones las más complicadas de aprender y las que más información aportan, debido a la habitual existencia de *overlaping* entre clases en dichas zonas. Tras SMOTE surgió Borderline-SMOTE que detectaba instancias en la frontera bajo la tesis de que para una instancia minoritaria en la frontera, al efectuar el cálculo de sus k vecinos más cercanos, más de la mitad serían instancias negativas, dado que la clase mayoritaria tiene mayor número de instancias.

La última afirmación presenta problemas básicos como que instancias ruidosas podrían ser consideradas como fronterizas, o que el valor de k no puede ser el mismo para todas las zonas del espacio, puesto que habrá zonas mucho más infrarrepresentadas que otras. También podemos añadir que si generamos instancias minoritarias entre dos clústers de instancias positivas (esto podría pasar para dos clústeres muy infrarrepresentados y un k alto), la instancia que se generaría quedaría fuera del espacio de dichas regiones, podiendo ocasionar ruido en regiones de instancias negativas.

El algoritmo MWMOTE (*Majority Weighted Minority Oversampling Technique*) surge como una modificación de SMOTE para poner solución a los anteriores problemas. Sea $S^+ = \{x_1, \dots x_m\}$, clase minoritaria y $S^- = \{y_1, \dots y_m\}$, clase mayoritaria, con $S = S^+ \cup S^-$ el conjunto de entrenamiento. Se usará KNN con distancia euclídea, donde notaremos $d(x, y)$ a la distancia euclídea entre x e y. Notaremos para una instancia $x \in S$, $NN^k(x) \subseteq S$ a su k-KNN vecindario, esto es, el conjunto de las k más cercanas (con distancia euclídea) a x. Notaremos $NN^k_+(x)$ a su k-KNN vecindario positivo, esto es las k instancias más cercanas pertenecientes a S^+. Análogamente definimos $NN^k_-(x) \subseteq S^-$ como el k-vecindario negativo de una instancia.

Algoritmo 2 Algoritmo de *oversampling* MWMOTE

Entrada: $S^+ = \{x_1, \dots x_m\}$, clase minoritaria
Entrada: $S^- = \{y_1, \dots y_m\}$, clase mayoritaria
Entrada: T, número de instancias sintéticas deseado
Entrada: k_1, parámetro de KNN para filtrar ruido de S^+
Entrada: k_2, parámetro de KNN para calcular frontera $U \subseteq S^-$
Entrada: K_3, parámetro de KNN para calcular frontera $V \subseteq S^+$
Entrada: α, umbral de tolerancia en la cercanía a la frontera
Entrada: C, ponderación de la cercanía a la frontera
Entrada: C_{clust}, parámetro del *clustering* para determinar el número de clústers

Inicializar $S' = \emptyset$, instancias sintéticas
Para cada $x \in S^+$, calcular su k_1 KNN vecindario, $NN^{k_1}(x)$
Hacer $S_f^+ = S^+ - \{x \in S^+ : NN^{k_1}(x) \cap S^+ = \emptyset\}$
Calcular frontera negativa $U = \bigcup_{x \in S_f^+} NN_-^{k_2}(x)$
Calcular frontera positiva $V = \bigcup_{x \in U} NN_+^{k_3}(x)$
Para cada $x \in V$, calcular $P(x) = \sum_{y \in U} I_{\alpha,C}(x, y)$
Normalizar para cada $x \in V$, $P(x) = \frac{P(x)}{\sum_{z \in V} P(z)}$
Calcular $T_{clust} = C_{clust} \cdot \frac{1}{|S_f^+|} \sum_{x \in S_f^+} \text{mín}_{y \in S_f^+, y \neq x} d(x, y)$
Calcular $L_1, \dots L_M \subseteq S^+$ clústers para S^+, con umbral T_{clust}

for $t = 1, \dots, T$ **do**
    Escoger $x \in V$ de acuerdo a la probabilidad $P(x)$
    Seleccionar $y \in L_k$ uniformemente donde $L_k \ni x$ (L_k es el clúster de x)
    Seleccionar r en $[0, 1]$ de manera uniforme
    $S' = S' \cup \{x + r(y - x)\}$
end for

return S', ejemplos positivos sintéticos

El conjunto $S_f^+ \subseteq S^+$ se construye para filtrar instancias ruidosas (aquellas que no tienen ninguna otra de la clase positiva alrededor suyo). U representa la frontera de instancias negativas (habrá que tomar preferiblemente un k_2 pequeño, ya que la densidad de representación de la clase negativa es mucho mayor que la de la positiva). La construcción de V, o frontera de instancias positivas, debe hacerse con un k_3 grande por un motivo antagónico al ya expresado para U.

MWMOTE tiene 3 objetivos fundamentales: dar mayor peso a instancias minoritarias fronterizas, a instancias pertenecientes a clústeres de baja representación, y a instancias minoritarias muy cerca de zonas con muchas instancias mayoritarias.

Definimos el peso informativo de x respecto a y como $I_{\alpha,C}(x,y) = C_f(x,y) \cdot D_f(x,y)$, donde si $x \notin NN_+^{k_3}(y)$, entonces $I_{\alpha,C}w(x,y) = 0$. Caso opuesto, se toma:

$$f(x) = \begin{cases} x & ,x \leq \alpha \\ \alpha & \text{en otro caso} \end{cases} , \qquad C_f(x,y) = \frac{C}{\alpha} \cdot f\left(\frac{d}{d(x,y)}\right)$$

C_f es un factor que mide la cercanía a y, es decir, mide la cercanía entre dos instancias fronterizas. Se toma $D_f(x,y) = \frac{C_f(x,y)}{\sum_{z \in V} C_f(z,y)}$, factor de densidad, de manera que una instancia perteneciente a un clúster más compacto tendrá mayor $\sum C_f(z,y)$ que una perteneciente a un clúster muy disperso. El último de los objetivos de detección de instancias de MWMOTE se cumple al ponderar cada instancia x por la suma $\sum_{y \in U} I_{\alpha,C}(x,y)$.

Por último queda explicar el algoritmo de *clustering* que se usa: un algoritmo de *clustering* jerárquico aglomerativo con enlace medio (esto es, se define $dist(L_i, L_j) = \frac{1}{|L_i||L_j|} \sum_{x \in L_i} \sum_{y \in L_j} d(x,y)$.

Algoritmo 3 Algoritmo de *clustering* jerárquico

Entrada: $S^+ = \{x_1, \ldots x_m\}$, clase minoritaria

1: Inicializar $L_i = \{x_i\}, i = 1, \ldots, m$
2: **while** ínf$_{i,j,i \neq j}\, dist(L_i, L_j) \leq T_{clust}$, y quede más de un clúster **do**
3: Sean $(i,j) =$ arg mín$_{i,j,i \neq j}\, dist(L_i, L_j)$
4: $L_i = L_i \cup L_j$. Elimina L_j
5: **end while**
6: **return** Clústeres $L_1, \ldots L_k, k \leq m$

Nótese que a mayor valor del parámetro C_{clust} menor número de clústeres tendremos, y mayor número de instancias tendrán cada uno.

10.2 ALGORITMOS RACOG Y WRACOG

Sea un subconjunto S^+ de ejemplos de clase positiva de un conjunto de entrenamiento, donde cada $(x,y) \in S^+$ verifica que $x = (w_1, \ldots, w_d)$ es $d-$dimensional. Los algoritmos RACOG y wRACOG parten de la base de que lo que vamos a aproximar es una distribución discreta, y podemos obtener instancias siguiendo una distribución conjunta $P(W_1, \ldots, W_d)$.

El problema es que calcular la anterior distribución es muy costoso, dado que tendríamos un número de posibles valores para cada instancia de:

$$|\{\text{Posibles valores para } W_1\}| \cdots |\{\text{Posibles valores para} W_d\}|$$

Si consiguiéramos aproximar $P(W_1, \ldots, W_d)$ como producto de distribuciones marginales, esto es, $P(W_1, \ldots W_d) \approx \prod_{i=1}^{d} P(W_i \mid W_{n(i)})$ donde cada $n(i) \in \{1, \ldots, d\}$, entonces calcular dicha distribución sería menos costoso. Para expresar $P(W_1, \ldots, W_d)$ como producto de marginales, es decir, para decidir qué valor le damos a cada $n(i)$, se usa el algoritmo 4 de Chow-Liu, que minimiza la distancia de Kullback-Leibler entre dos distribuciones. Esta distancia proporciona una medida de la aproximación de una distribución de probabilidad Q a la verdadera distribución P, usando teoría de la información. Está definida como la esperanza de la diferencia logarítmica entre ambas distribuciones:

$$D_{KL}(P \parallel Q) = \sum_i P(i) \left(\log P(i) - \log Q(i)\right)$$

Debemos recordar que la definición de información mutua para dos variables aleatoria W_i, W_j, discretas, se define como:

$$I(W_i, W_j) = \sum_{w_1 \in W_1} \sum_{w_2 \in W_2} p(w_1, w_2) \log \left(\frac{p(w_1, w_2)}{p(w_1) p(w_2)} \right)$$

Algoritmo 4 Algoritmo de Chow-Liu de construcción de un árbol maximizando la suma de información mutua de los arcos

Entrada: $S = \{x_i = (w_1^{(i)}, \ldots w_d^{(i)})\}_{i=1}^{m}$, instancias

1: **for** Cada pareja i, j **do**
2: Calcular la información mutua $I(w_i, w_j)$
3: **end for**
4: Construir G el árbol generador de peso máximo mediante el algoritmo de Kruskal
5: **return** G

Una vez construido el árbol, es fácil, mediante el siguiente algoritmo obtener un grafo dirigido, en que a cada nodo (variable en la distribución), tiene un único padre. Lo hacemos recorriendo el árbol desde la raíz y actualizando el conjunto de nodos que pueden ir siendo padres de otros nodos, H.

Algoritmo 5 Construcción del grafo a partir del árbol de Chow-Liu

Entrada: $G = (E, V)$ grafo no dirigido

Inicializar $visitados = \emptyset$
Inicializar $G' = (E', V)$, con $E' = \{(p, q)\}$ con $(p, q) \in E$ arbitrario. p será la raíz del árbol
Hacer $E = E \setminus \{(p, q)\}$ y $H = \{p, q\}$ conjunto de padres
while Mientras $E \neq \emptyset$ y $H \neq \emptyset$ **do**
    Toma $h \in H$, y actualiza $H = H \setminus \{h\}$
    Calcula $J = \{(u, v) \in E : u = h \vee v = h\}$
    Calcula $J' = \{(u, v) : [(u, v) \in J, u = h] \vee [(v, u) \in J, u = h]\}$
    $E = E \setminus J$
    $E' = E' \cup J'$
    $H = H \cup \{v : (u, v) \in J'\}$
end while

return G'

Por tanto queda un algoritmo para aproximar la distribución de los ejemplos positivos tal que así:

Algoritmo 6 Algoritmo AproximarDistribución

Entrada: $S = \{x_i = (w_1^{(i)}, \ldots, w_d^{(i)})\}_{i=1}^m$, instancias

Calcular $G' = (E', V')$ árbol de dependencia según Chow Liu
Construir G un grafo no dirigido con algoritmo 5 desde G', donde E son los arcos, r la raíz. Definimos $P(W_r|n(r)) := P(W_r)$
for $(u, v) \in E$ **do**
    Hacer $n(v) = u$
    Calcular $P(W_v \mid W_u)$
    Calcular $P(W_u), P(W_v)$
end for
return $\{P(W_v \mid W_u), P(W_u), P(W_v)\}_{(u,v)\in E}$

Nótese que tenemos la distribución almacenada como marginales, y que por tanto necesitamos aún una forma de extraer muestras desde ella. Se usará para la construcción de ejemplos mediante dicha distribución un método de Monte Carlo llamado GibbsSampler. Se parte de cada ejemplo, y se construye un nuevo ejemplo basado en el anterior y la distribución conjunta aproximada. El método de Monte Carlo con dicho algoritmo consiste en ir construyendo m cadenas de Markov, una para cada instancia, de manera que iteramos una vez cada vez que llamamos al método.

Algoritmo 7 Algoritmo GibbsSampler

Entrada: $S = \{x_i = (w_1^{(i)}, \ldots, w_d^{(i)})\}_{i=1}^m$, instancias
Entrada: $\{P(W_v \mid W_u), P(W_u)\}_{(u,v)\in E}$
1: **for** $i = 1, \ldots, m$ **do**
2: **for** $k = 1, \ldots, d$ **do**
3: $\bar{w}_k^{(i)} \sim P(W_k \mid \bar{w}_1^{(i)}, \ldots, \bar{w}_{k-1}^{(i)}, w_{k+1}^{(i)} \ldots, w_d^{(i)})$
4: **end for**
5: **end for**
6: **return** $S = \{\bar{x}_i = (\bar{w}_1^{(i)}, \ldots \bar{w}_d^{(i)})\}_{i=1}^m$, conjunto de instancias generado desde S y P

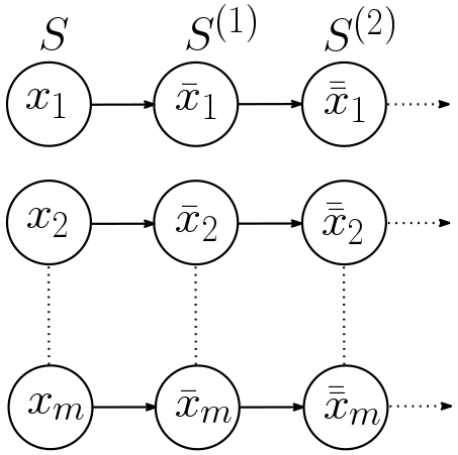


Figura 5.: Cadena de Markov obtenida por GibbsSampler

Nótese además que fijado $k \in \{1, \ldots, d\}$, tenemos:

$$P(W_k \mid \bar{w}_1^{(i)}, \ldots, \bar{w}_{k-1}^{(i)}, w_{k+1}^{(i)} \ldots, w_d^{(i)}) = \frac{P(W_k, \bar{w}_1^{(i)}, \ldots, \bar{w}_{k-1}^{(i)}, w_{k+1}^{(i)} \ldots, w_d^{(i)})}{P(\bar{w}_1^{(i)}, \ldots, \bar{w}_{k-1}^{(i)}, w_{k+1}^{(i)} \ldots, w_d^{(i)})}$$

Como el denominador es constante porque $\bar{w}_1^{(i)}, \ldots, \bar{w}_{k-1}^{(i)}, w_{k+1}^{(i)} \ldots, w_d^{(i)}$ son valores fijos para cualquier valor que pueda tomar W_k, no nos hace falta calcularlo. Además, como el numerador lo calculamos como $\prod_{i=1}^d P(W_i \mid W_{n(i)})$, también encontramos una serie de términos multiplicando que no debemos calcular, todos aquellos que cumplan $i \neq k \wedge n(i) \neq k$.

Una última observación: el cálculo de $P(W_k \mid W_{n(k)} = w_{n(k)})$ es directo. Pero el cálculo de $P(W_k = w_k \mid W_{n(k)})$ donde $n(i) = d$ debe hacerse usando el teorema de Bayes:

$$P(W_i = w_i \mid W_{n(i)} = a) = \frac{P(W_{n(i)} = a \mid W_i = w_i)}{P(W_{n(i)} = a)} \cdot P(W_i = w_i)$$

10.2.1 *RACOG*

RACOG (*Rapidly Converging Gibbs*) consiste en ir construyendo una cadena de Markov para cada una de las m instancias minoritarias, de manera que descartamos las β primeras instancias producidas para cada ejemplo del conjunto positivo, y a partir de entonces cada α iteraciones se va escogiendo una. Esto permite, según [Das et al., 2015], perder dependencia de los valores iniciales en la cadena (parámetro β o *burnin*), así como reducir la dependencia en la generación de sucesivas instancias (parámetro α o *lag*).

Algoritmo 8 Algoritmo de *oversampling* RACOG

Entrada: $S = \{x_1, \dots x_m\}$, ejemplos positivos
Entrada: β, burnin
Entrada: α, lag
Entrada: T, número de instancias sintéticas a generar

$P = \text{AproximarDistribución}(S)$
$S' = \emptyset$
$M = \lceil \frac{T}{m} \rceil \cdot \alpha + \beta$

for $t = 1, \dots, M$ **do**
  $S = \text{GibbsSampler}(S, P)$

  if $t > \beta$ **and** $t \mod (\alpha) = 0$ **then**
    $S' = S' \cup S$
  end if
end for

$S' =$ Escoger T instancias aleatorias de entre S'
return S', ejemplos positivos sintéticos

Traducido a código:

```
probDist <- .genDistribution(minority)

for(k in seq_len(iterations)){
  # Generate new sample using Gibbs Sampler
  minority <- gibbsSampler(probDist, minority)

  if(k > burnin && k%%lag == 0)
    newSamples <- rbind.data.frame(newSamples, minority)
}
```

Código 10.1: Cuerpo del algoritmo RACOG

10.2.2 *wRACOG*

El algoritmo RACOG presenta un problema: depende de los parámetros de *burnin*, *lag* y es el usuario quien decide el número de instancias que desea. El algoritmo wRACOG (*wrapper-based RACOG*), por el contrario, encapsula las iteraciones de GibbsSampler de manera que se proporcionan un conjunto de entrenamiento y otro de validación. A cada iteración, se efectúa una iteración del GibbsSampler, se añaden las instancias mal clasificadas por el modelo obtenido por un *wrapper* de entre las muestras sintéticas al conjunto de sintéticas, y se genera un nuevo *train* y un nuevo modelo. Se deja de iterar cuando se verifica un criterio sobre el conjunto de validación.

Algoritmo 9 Algoritmo de *oversampling* wRACOG

Entrada: $S_{train} = \{z_1 = (x_1, y_1), \ldots, z_m = (x_m, y_m)\}$, conjunto de *train*
Entrada: S_{val}, conjunto de validación
Entrada: $wrapper$, clasificador
Entrada: T, número de iteraciones a considerar
Entrada: α, parámetro de tolerancia

$S = S^+_{train}$
$P =$ AproximarDistribución(S)
Obtener *modelo* con $wrapper$ y S_{train}
Inicializar nuevas muestras $S' = \emptyset$
Inicializar $\tau = (\underset{1)}{+\infty}, \ldots, \underset{T)}{+\infty})$

while Desviación estándar de $\tau \geq \alpha$ **do**
  $S =$ GibbsSampler(S, P)
  $S_{misc} =$ instancias mal clasificadas de S por *modelo*
  Actualizar nuevas instancias, $S' = S' \cup S_{misc}$
  Actualizar *train*, $S_{train} = S_{train} \cup S_{misc}$
  Obtener *modelo* con $wrapper$ y S_{train}
  Hacer $s =$ sensibilidad de la predicción de *modelo* sobre S_{val}
  Hacer $\tau = (\tau_2, \ldots, \tau_T, s)$
end while

return S', ejemplos positivos sintéticos

10.2.3 *Críticas a los algoritmos*

Existe una crítica clara a los algoritmos: al aproximar distribuciones discretas, no se puede esperar que tengan buen comportamiento sobre atributos continuos.

Asimismo, como requieren acceso a la distribución precalculada, deberíamos tener alguna forma de tener acceso $\mathcal{O}(1)$ a los valores para dicha distribución. El problema es que R, el lenguaje donde se ha implementado, no proporciona estructura de tablas *hash*, a pesar de existir un paquete [1] que las simula, pero que no proporciona acceso a datos $\mathcal{O}(1)$. Esto supone que la implementación no puede hacerse eficiente a no ser que programemos los algoritmos 6, 8 y 7 en otro lenguaje que se llame desde R. El problema con esta última aproximación hubiera sido que *wrapper* en el algoritmo 9 es una función que se le pasa al método y no podríamos mezclar dos lenguajes en el bucle principal.

10.3 ALGORITMO RWO

El algoritmo RWO (*Random Walk Oversampling*) consiste en construir instancias sintéticas inspirándonos en el teorema central del límite, de manera que intentamos dejar invariante el límite en distribución de los datos de entrenamiento.

Teorema 10.3.1. *Sean* $\{W_1, \ldots, W_m\}$ *un conjunto de variables aleatorias i.i.d., con* $\mathbb{E}(W_i) = \mu$ *y* $Var(W_i) = \sigma^2 < \infty$. *Entonces:*

$$\lim_m P\left[\frac{\sqrt{m}}{\sigma}\left(\underbrace{\frac{1}{m}\sum_{i=1}^{m} W_i}_{\overline{W}} - \mu\right) \leq z\right] = \phi(z)$$

donde ϕ *es la función de distribución de la normal* $N(0,1)$.

Es decir $\frac{\overline{W}-\mu}{\sigma/\sqrt{m}} \to N(0,1)$ *en probabilidad.*

Supongamos que las muestras son de dimensión d. Fijamos una columna $j \in \{1, \ldots d\}$. Supuesto un conjunto de instancias de la clase positiva $\{x_i = (w_1^{(i)}, \ldots, w_d^{(i)})\}_{i=1}^m$, supondremos que la columna $j-$ésima de los datos está generada según una variable aleatoria W_j de media μ_j y desviación típica $\sigma_j < \infty$.

Sean $\mu_j', \sigma_j'^2$ la media y varianza sesgada muestrales, respectivamente. Las instancias sintéticas que generaremos serán de la forma $w_j'(i) = w_j^{(i)} - \frac{\sigma_j'}{\sqrt{m}} \cdot r$, con $r \sim N(0,1)$, para $i = 1, 2, \ldots, m$. Nótese la similitud de esta fórmula con 10.3.1, con la salvedad de que usamos los valores para la variable W_j que conocemos en lugar de μ_j, que es descono-

1 https://cran.rstudio.com/web/packages/hash/

cida; y $\sigma_j'^2$ que es la varianza muestral no corregida, en lugar de la verdadera varianza muestral σ_j^2, que tampoco la conocemos.

Teorema 10.3.2. *La esperanza de la media muestral de las instancias* $\{w_j'(i)\}_{i=1}^m$ *sintéticas es* μ_j*. La esperanza de la varianza muestral tiende en m a* σ_j^2*.*

Demostración. Se tiene una variable aleatoria $W_j' = W_j - \frac{\sigma_j'}{\sqrt{m}}R$ donde $R \sim N(0,1)$.

Para la media:

$$\mathbb{E}(W_j') = \mathbb{E}(W_j) - \frac{\sigma_j'}{\sqrt{m}}\mathbb{E}(R) = \mu_j$$

Puesto que $\mathbb{E}(W_j) = \mu_j$ y $\mathbb{E}(R) = 0$. Usando linealidad de la esperanza se prueba que $\mathbb{E}\left(\frac{1}{m}\sum_{i=1}^m w_j'(i)\right) = \mu_j$.

Ahora dado j arbitrario:

$$\mathbb{E}\left(\left(W_j - R\sqrt{m}\sigma_j' - \mu_j\right)^2\right) = \mathbb{E}\left((W_j - \mu_j)^2\right) - \mathbb{E}\left(2(W_j - \mu_j)\frac{r_i}{\sqrt{m}}\sigma_j'\right) + \\ + \mathbb{E}\left(\frac{R^2}{m}\sigma_j'^2\right) = \sigma_j^2 + 0 + \frac{1}{m}\sigma_j'^2$$

Sea $\tau_m = \frac{1}{m}\sum_{i=1}^m \left(w_j'(i) - \frac{1}{m}\sum_{i=1}^m w_j'(i)\right)^2$ la varianza muestral. Es conocido que $\mathbb{E}(\tau_m) = \frac{m-1}{m}Var(W_j') = \frac{(m-1)}{m}\left(\sigma_j^2 + \frac{1}{m}\sigma_j'^2\right)$

Luego $\lim_{m\to+\infty}\mathbb{E}(\tau_m) = \sigma_j^2$. □

El algoritmo incluye una distinción para los atributos no numéricos, donde se establece una distribución uniforme para todos los valores que posean dichos atributos, y se le asigna a la instancia generada. En pseudocódigo obtenemos:

Algoritmo 10 Algoritmo de *oversampling* RWO

Entrada: $S = \{x_i = (w_1^{(i)}, \dots w_d^{(i)})\}_{i=1}^{m}$, ejemplos positivos
Entrada: T, número de instancias sintéticas deseado
Inicializar $S' = \emptyset$
for Cada atributo $j = 1, \dots, d$ **do**
  if El atributo $j-$ésimo es numérico **then**
    Calcular $\sigma'_j = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(w_j^{(i)} - \frac{\sum_{i=1}^{m} w_j^{(i)}}{m}\right)^2}$
  end if
end for
Hacer $M = \lceil T/m \rceil$
for $t = 1, \dots, M$ **do**
  for $i = 1, \dots, m$ **do**
    for $j = 1, \dots, d$ **do**
      if El atributo $j-$ésimo es numérico **then**
        Escoger $r \sim N(0, 1)$
        $w_j = w_j^{(i)} - \frac{\sigma'_j}{\sqrt{m}} \cdot r$
      else
        Escoger w_j de manera uniforme sobre $\{w_j^{(1)}, \dots w_j^{(m)}\}$
      end if
    end for
    $S' = S' \cup \{(w_1, \dots w_d)\}$
  end for
end for
$S' =$ Escoger T instancias aleatorias de entre S'
return S', ejemplos positivos sintéticos

Implementado en R, el cuerpo principal del algoritmo, omitiendo otros detalles de la implementación, queda:

```
newSamples <- lapply(minority, function(x){
  # If attribute is continuous, generate new minority
  # sample preserving mean and variance of existent samples
  scaleFactors <- stats::rnorm(nrow(minority) * iterations,
                               mean = 0, sd = 1)

  if(class(x) == "numeric"){
    variance <- (m-1)/m * stats::var(x)
    x - variance/sqrt(m) * scaleFactors

    # Else if attribute is not numeric, make a roulette out
```

```
    # of possible values for the attribute and their frequency
  } else{
    dist <- table(x)
    distValues <- names(dist)
    distProbs <- unname(dist)
    sample(distValues, length(x) * iterations, replace = T, prob
        = distProbs)
  }
})
```

Código 10.2: Cuerpo del algoritmo RWO

10.3.1 *Críticas al algoritmo*

Se usa la varianza muestral sesgada, y por tanto en el teorema 10.3.2 lo único que podemos afirmar es que $\mathbb{E}(\tau_m) \underset{m\to\infty}{\longrightarrow} \sigma_j^2$. Si se hubiera seleccionado el estimador insesgado para la varianza muestral, $\overline{\tau}_m = \frac{1}{m-1}\sum_{i=1}^{m}\left(w_j'(i) - \frac{1}{m}\sum_{i=1}^{m} w_j'(i)\right)^2$, tendríamos $\mathbb{E}(\overline{\tau}_m) = \sigma_j^2$.

10.4 ALGORITMO PDFOS

10.4.1 *Motivación*

Dada una función de distribución de X variable aleatoria, $F(x)$, si esta función es derivable casi seguramente entonces podemos tomar la función de densidad como la derivada de la función de distribución, casi seguramente, es decir:

$$f(x) = \lim_{h\to 0}\frac{F(x+h) - F(x-h)}{2h} = \lim_{h\to 0}\frac{P(x-h < X \leq x+h)}{2h}$$

Si tenemos muestras aleatorias de X, a saber, $X_1, \ldots X_n$ y $x_1, \ldots x_n$ una realización muestral, entonces un estimador para f sería:

$$\widehat{f}(x) = \frac{1}{2hn}\left[\text{Número de } x_1, \ldots, x_n \text{ que se quedan en]x-h, x+h[}\right]$$

Es decir, definiendo $\omega(x) = \begin{cases} \frac{1}{2} & , |x| < 1 \\ 0 & \text{en otro caso} \end{cases}$

y $w_h(x) = w\left(\left|\frac{x}{h}\right|\right)$, podríamos reformular $\widehat{f}$ como:

$$\widehat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n}\omega_h(x - x_i)$$

Es decir, supuesto que las observaciones $x_1, \ldots, x_n$ se distancian múltiplos de $2h$ (caen en el centro de intervalos de longitud $2h$), habríamos construido $\widehat{f}$ a base de un histograma donde cada barra tiene ancho $2h$ y altura $\frac{1}{2nh} \cdot \Big[[\text{Número de muestras } x_1, \ldots, x_n \text{ en el intervalo}]\Big]$. Al parámetro h se le llama parámetro de *bandwidth*, por referencia al significado que tiene en el caso de los histogramas.

En el caso multidimensional, tendríamos:

$$\widehat{f}(x) = \frac{1}{nh^d}\sum_{i=1}^{n}\omega_h(x - x_i)$$

10.4.2 *Generalización a funciones kernel*

Tomar $w = \frac{1}{2}\mathbb{1}_{]-1,1[}$ presenta el problema de que $\widehat{f}$ será una función a saltos, y no será continua. Surge una generalización al tomar ω como funciones verificando $w \geq 0$, $\int_\Omega \omega(x)dx = 1$, siendo Ω el dominio de X. Se suelen considerar además funciones simétricas $w(x) = w(-x)$ para cualquier x en el dominio.

El estimador que se suele usar para evaluar la bondad de $\widehat{f}$ es el error cuadrático medio integrado (MISE):

$$MISE(h) = \mathop{\mathbb{E}}_{x_1,\ldots,x_n}\int(\widehat{f}(x) - f(x))^2 dx$$

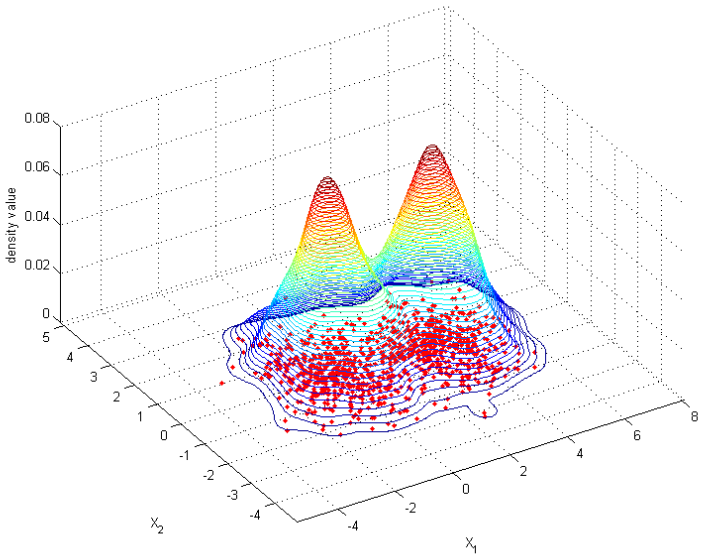


Figura 6.: Ejemplo de estimación de densidad con funciones kernel [2]

10.4.3 *Funciones kernel Gaussianas multivariantes*

El algoritmo PDFOS (*Probability Distribution density Function estimation based Oversampling*) se basa en tomar funciones kernel Gaussianas multivariantes. Recordamos la definición de la función de densidad de la distribución d-Gaussiana multivariante de media 0 y matriz de covarianza Ψ:

$$\phi^{\Psi}(x) = \frac{1}{\sqrt{(2\pi \cdot det(\Psi))^d}} exp\left(-\frac{1}{2} x \Psi^{-1} x^T\right)$$

Dada $S = \{x_i = (w_1^{(i)}, \ldots, w_d^{(i)})\}_{i=1}^m$, la clase minoritaria, calcularemos el estimador no sesgado para la covarianza:

$$U = \frac{1}{m-1} \sum_{i=1}^{m} (x_i - \overline{x})(x_i - \overline{x})^T, \qquad \text{siendo } \overline{x} = \frac{1}{m} \sum_{i=1}^{m} x_i$$

Las funciones kernel que tomaremos serán de la forma: $\phi = \phi^U$, y tendremos que ajustar el *bandwidth* h de $\phi_h(x) = \phi^U\left(\frac{x}{h}\right)$ para minimizar el MISE. A tales efectos, hay que buscar el mínimo de la función de validación cruzada:

$$M(h) = \frac{1}{m^2 h^d} \sum_{i=1}^{m} \sum_{j=1}^{m} \phi_h^*(x_i - x_j) + \frac{2}{m h^d} \phi_h(0) \tag{1}$$

2 Imagen de dominio público tomada de Wikimedia Commons

donde $\phi_h^* \approx \phi_{h\sqrt{2}} - 2\phi_h$.

Nótese que $x \mapsto \phi_h(x - x_i)$ es la función de densidad de una normal con matriz de covarianza h^2U y centrada en x_i.

Una vez minimizada la función de validación cruzada M, el esquema de generación de instancias se basará en dada una instancia $x_i \in S^+$, tomar $x_i + hRr$, donde $r \sim N^d(0,1)$ y $U = R \cdot R^T$. Ilustramos el porqué de esta última afirmación: análogamente al caso unidimensional, donde se pueden generar instancias desde $N(\mu, \sigma)$ tomando $\mu + \sigma r$ con $r \sim N(0,1)$, en el caso multivariante dada $U = R^T \cdot R$, se pueden generar instancias siguiendo una distribución $N^d(\mu, U)$, tomando $\mu + R \cdot r$, donde $r \sim N^d(0,1)$.

En [Gao et al., 2014] se usa para descomponer $U = R^T \cdot R$ la descomposición de Choleski [3], que sólo sirve para matrices definidas positivas.

Lema 10.4.1. *$\sum_{i=1}^m (x_i - \overline{x})(x_i - \overline{x})^T$ es una matriz semi-definida positiva.*

Demostración. Dado $y \in \mathbb{R}^d$, se tiene:

$$y^T \left(\sum_{i=1}^m (x_i - \overline{x})(x_i - \overline{x})^T \right) y = \sum_{i=1}^m (\underbrace{(x_i - \overline{x})^T y}_{z})^T ((x_i - \overline{x})^T y)$$

$$\underset{z \text{ es un vector}}{=} \sum_{i=1}^m ((x_i - \overline{x})^T y)^2 \geq 0$$

□

Luego al estimador S que calculamos para la varianza le podemos calcular la descomposición de Choleski cuando sea definido positivo ($y^T \cdot S \cdot y > 0$ de manera estricta para todo $0 \neq y \in \mathbb{R}^d$). Pero no está garantizado que siempre sea definido positivo.

3 https://en.wikipedia.org/wiki/Cholesky_decomposition

Algoritmo 11 Algoritmo de *oversampling* PDFOS

Entrada: $S = \{x_i = (w_1^{(i)}, \dots w_d^{(i)})\}_{i=1}^m$, ejemplos positivos
Entrada: T, número de instancias sintéticas deseado
1: Inicializamos $S' = \emptyset$
2: Búsqueda de $h =$ que minimice $M(h)$
3: Calcular U la matriz de covarianza insesgada de S
4: Calcular descomposición de Choleski de U, donde $U = R^T \cdot R$, y R triangular superior
5:
6: **for** $i = 1, \dots, T$ **do**
7: Escoger $x \in S$
8: Escoger r siguiendo una normal multivariante, $r \sim N^d(0, 1)$
9: $S' = S' \cup \{x + hrR\}$
10: **end for**
11:
12: **return** S', ejemplos positivos sintéticos

En [Gao et al., 2014] se hacía referencia a emplear un algoritmo *grid* (dividir un intervalo plausible en valores y buscar el que más pequeña hace la función de validación cruzada) para extraer el mejor valor de h. Se ha preferido tomar como primera aproximación el valor propuesto por [Silverman, 1986]:

$$h_{Silverman} = \left(\frac{4}{m(d+2)}\right)^{\frac{1}{d+4}}$$

donde d es la dimensión de los datos, y m el número de instancias minoritarias.

A partir de dicho valor se ha seguido un algoritmo de gradiente descendente para intentar buscar un mínimo local. Para ello se ha reformulado (1) y se ha derivado, pudiéndose comprobar como tanto (2) como (3) serían calculables con una única función:

$$\begin{aligned} M(h) &= \frac{1}{m^2 h^d} \sum_{i=1}^{m} \sum_{j=1}^{m} \phi_h^*(x_i - x_j) + \frac{2}{m h^d} \phi_h(0) \\ &= \frac{1}{m^2 h^d} \sum_{i=1}^{m} \sum_{j=1, j \neq i}^{m} \phi_h^*(x_i - x_j) + \frac{1}{m h^d} \phi_{h\sqrt{2}}(0) \\ &= \frac{2}{m^2 h^d} \sum_{j>i}^{m} \phi_h^*(x_i - x_j) + \frac{1}{m h^d} \phi_{h\sqrt{2}}(0) \end{aligned} \tag{2}$$

$$\begin{aligned} \frac{\partial M}{\partial h}(h) = \frac{2}{m^2 h^d} \sum_{j>i}^{m} \phi_h^*(x_i - x_j) \Big(&- d h^{-1} + h^{-3} (x_i - x_j)^T U (x_i - x_j) \Big) \\ &- \frac{d h^{-1}}{m h^d} \phi_{h\sqrt{2}}(0) \end{aligned} \tag{3}$$

El algoritmo de gradiente descendente dirigirá la búsqueda del valor de h hacia mínimos locales.

Algoritmo 12 Algoritmo de búsqueda con gradiente descendente

Hacer $M_{best} = \infty$
Hacer $h = h_{Silverman}$
Hacer $h_{best} = h$
Hacer $\delta = h$
Hacer $\alpha = 0{,}001$
Sea γ una constante suficientemente pequeña

while $\delta > \alpha$ y número de iteraciones menor que $m \cdot d$ **do**
  if $M(h) < M_{best}$ **then**
    Hacer $M_{best} = M(h)$ y $h_{best} = h$
  end if
  Hacer $h_{-1} = h$
  Actualizar h como $h - \gamma \cdot M'(h)$
  $\delta = h - h_{-1}$
end while
return h

10.4.4 *Críticas al algoritmo*

Podemos efectuar dos críticas principales al algoritmo, no estando reflejadas las siguientes consideraciones en el *paper* de sus autores:

- No toda matriz de covarianza es invertible, siendo el caso del dataset `ecoli1` incluido en el paquete de software desarrollado, por ejemplo, una matriz de covarianza no invertible. En nuestra implementación se ha decidido interrumpir el proceso en caso de que se encuentre una matriz no invertible, avisando al usuario a tales efectos. Podría estudiarse cómo encajaría dentro del algoritmo sustituir la inversa convencional por la inversa generalizada [4].

- El método de Choleski sólo se puede aplicar en los casos en los que la matriz es definida positiva, por lo que al implementarse el algoritmo nos hemos preocupado de usar el método de Choleski con pivotado (lo usa la función `rmvnorm` de [`mvtnorm`] para generar las instancias, para lo cual hubo que revisar el código de dicha función). El método de Choleski con pivotado sí que puede emplearse en el caso de matrices semidefinidas positivas no definidas positivas.

4 https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Degenerate_case

10.5 ALGORITMO NEATER

El algoritmo NEATER (*filteriNg of ovErsampled dAta using non cooperaTive gamE theoRy*) es un algoritmo de filtrado de instancias no representativas basado en teoría de juegos. Daremos a continuación unos preliminares de teoría de juegos para poder comprender mejor el algoritmo.

10.5.1 *Teoría de juegos*

Sea una tupla (P, S, f), donde $P = \{1, \ldots, n\}$, conjunto de jugadores. Tendremos $S_i = \{1, \ldots, k_i\}$ conjunto de posibles estrategias para el jugador i-ésimo, donde $S = S_1 \times \ldots \times S_n$. Dado $s = (s_1, \ldots s_n) \in S$ arbitrario, podremos asignarle una recompensa a cada jugador que dependerá de la estrategia que ha seguido y de la estrategia del resto de jugadores por lo que tendremos:

$$\begin{array}{rcl} f : S & \longrightarrow & \mathbb{R}^n \\ s & \longmapsto & (f_1(s), \ldots, f_n(s)) \end{array}$$

Notaremos $s_{-i} = (s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n)$ y $f_i(s_i, s_{-i}) = f_i(s)$.

Definición 10.5.1. *Un equilibrio de Nash estratégico es una tupla $(s_1, \ldots s_n)$ que verifica $f_i(s_i, s_{-i}) \geq f_i(s'_i, s_{-i})$ para cualquier otra $s' \in S$, para todo $i = 1, \ldots, n$.*

Es decir, una estrategia de Nash maximiza la recompensa para todos los jugadores.

Tendremos probabilidades para escoger estrategias para cada jugador, esto es $\delta_i \in \Delta_i = \{(\delta_i^{(1)}, \ldots, \delta_i^{(k_i)}) \in (R^+)_i^k : \sum_{j=1}^{k_i} \delta_i^{(j)} = 1\}$. Llamamos $\Delta_1 \times \ldots \times \Delta_n = \Delta$ y a $\delta = (\delta_1, \ldots, \delta_n) \in \Delta$ lo llamaremos perfil de estrategia. Asociado a un perfil de estrategia δ, notaremos a la recompensa total esperada para el jugador i-ésimo como:

$$u_i(\delta) = \sum_{s \in S} \delta_i^{(s_i)} f_i(s)$$

A u_i es la recompensa asociada al perfil de estrategia δ para el jugador i-ésimo, y dada $\delta \in \Delta$ notaremos $\delta_{-i} = (\delta_1, \ldots, \delta_{i-1}, \delta_{i+1}, \ldots, \delta_n)$, y $u_i(\delta_i, \delta_{-i}) = u_i(\delta)$.

Definición 10.5.2. *Un equilibrio probabilístico de Nash es un perfil de estrategia $x = (\delta_1, \ldots, \delta_n)$ que verifica $u_i(\delta_i, \delta_{-i}) \geq u_i(\delta'_i, \delta_{-i})$ para cualquier otra $\delta' \in \Delta$, para todo $i = 1, \ldots, n$.*

Teorema 10.5.3. *Todo juego (P, S, f) con $|P| < \infty$ y $|S| < \infty$ tiene un equilibrio probabilístico de Nash.*

10.5.2 *Aplicación al problema de clasificación desbalanceada*

En nuestro caso los jugadores serán todos los posibles puntos del conjunto de entrenamiento unido a las instancias sintéticas $S \cup S'$. Cada jugador podrá escoger entre dos estrategias $\{0, 1\}$ donde 0 será pertenencia a la clase mayoritaria y 1 pertenencia a la clase minoritaria. Habrá dos clases de jugadores, aquellos cuya estrategia ya es fija (es decir, conocemos su clase), que serán los de S, donde un jugador i de S siempre jugará a la estrategia 0, esto es $\delta_i = (0, 1)$ siempre si es una instancia de la clase negativa; y jugará a la estrategia 1, esto es $\delta_i = (1, 0)$ siempre, si es una instancia de la clase positiva.

Consideraremos que a una instancia sólo le afecta para su recompensa asociada a una estrategia la suya propia y la de sus k vecinos más cercanos, calculados en $S \cup S'$. Así para cada instancia $x_i \in S'$, tendremos $u_i(\delta) = \sum_{j \in NN^k(x)} (x_i^T w_{ij} x_j)$ donde $w_{ij} = g\left(d(x_i, x_j)\right)$ tal que g es decreciente (esto es, a mayor distancia, menor recompensa). En nuestro caso, hemos tomado $g(z) = \frac{1}{1+z^2}$, con d la distancia euclídea.

A cada paso se actualizarán los perfiles de estrategia de la clase minoritaria, donde para cada $x_i \in S'$ se hace:

$$\begin{aligned}
\delta_i(0) &= \left(\frac{1}{2}, \frac{1}{2}\right) \\
\delta_{i,1}(n+1) &= \frac{\alpha + u_i((1,0))}{\alpha + u_i(\delta(n))} \delta_{i,1}(n) \\
\delta_{i,2}(n+1) &= 1 - \delta_{i,1}(n+1)
\end{aligned}$$

Es decir, se va premiando a cada paso de las dos estrategias posibles aquella que está reportando más recompensa, a base de sustraérselo a la otra estrategia. Este proceso tiene garantizada una convergencia, por teoría de *dinámica de replicador*, de teoría de juegos evolutiva.

Algoritmo 13 Algoritmo de limpieza de instancias NEATER

Entrada: $S = \{z_1 = (x_1, y_1), \dots z_n = (x_n, y_n)\}$, dataset original
Entrada: $S' = \{\bar{z}_1 = (\bar{x}_1, \bar{y}_1), \dots \bar{z}_m = (\bar{x}_m, \bar{y}_m)\}$, ejemplos positivos
Entrada: k, número de vecinos más cercano para KNN.
Entrada: T, número de iteraciones deseadas.
Entrada: α, factor de suavizado.

Inicializar $E = \emptyset$
Para cada $x_i \in S'$, calcular su vecindario $NN^k(x_i) \subseteq S \cup S'$
Inicializar:

- $i = 1, \dots, n$, entonces $\delta_i = \begin{cases} (1,0) & y_i = 1 \\ (0,1) & y_i = -1 \end{cases}$
- $i = n+1, \dots, n+m$, entonces $\delta_i = (0{,}5, 0{,}5)$

for $t = 1, \dots, T$ **do**
  for $i = 1, \dots, m$ **do**
    Calcular recomp. total $u_i = \sum_{x_j \in NN^k(x_i)} g(d(\bar{x}_i, x_j)) \cdot \delta_i \cdot \delta_j^T$
    Calcular recomp. positiva $u = \sum_{x_j \in NN^k(x_i)} g(d(\bar{x}_i, x_j)) \cdot (1,0) \cdot \delta_j^T$
    Calcular $\alpha = (\alpha + u)/(\alpha + u_{n+1})$
    Actualizar $\delta_{n+1} = (\alpha, 1 - \alpha)$
  end for
end for

for $i = 1, \dots, m$ **do**
  if $\delta_{i1} > 0{,}5$ **then**
    $E = E \cup \{(\bar{x}_i, 1)\}$
  end if
end for

return $E \subseteq S'$, conjunto de instancias positivas filtrado

11

PAQUETE IMBALANCE

En este capítulo presentamos el paquete de R desarrollado: `imbalance`, así como las tecnologías y metodologías empleadas en su desarrollo.

11.1 ¿QUÉ ES R? ¿POR QUÉ R?

[R] es un lenguaje pensado para estadística computacional y todo lo que ello implica: manipulación de datos, visualización, variedad de modelos estadísticos, etc. R puede ser considerado una implementación renovada del antiguo lenguaje S, para estadística. Destaca sobretodo por su amplia gama de paquetes, disponible en repositorios como CRAN [1] o Bioconductor [2], este último orientado a bioinformática.

R sobresale por su extensibilidad, su facilidad de uso, su extensa documentación (para que un paquete pueda ser publicado en CRAN tiene como requisito fundamental estar bien documentado), su fácil acceso a ella, y su heterogénea comunidad de usuarios, siendo empleado por estadistas, bioinformáticos, científicos de datos, informáticos o incluso investigadores asociados a medicina o biología.

R, a diferencia de otros lenguajes, no tiene un estándar y una implementación independientes, puesto que lo que se asume como estándar del lenguaje es la propia implementación GNU-R (implementación libre), a pesar de existir otras como `pqR` [3] (*Pretty Quick R*) o `FastR` [4]. R no es rápido, puesto que es un lenguaje interpretado y el *core* de GNU-R no tiene como objetivo fundamental hacerlo más rápido, sino proporcionar un lenguaje estable.

La filosofía principal de la programación en R es:

- **Programación funcional**: los paquetes desarrollados sólo pueden exportar funciones (técnicamente también ojetos `S3`, `S4` o `RC`, pero incluso los mismos trabajan con un tipo de orientación a objetos peculiar, donde la función recibe al objeto, y este só-

1 https://cran.r-project.org/
2 https://www.bioconductor.org
3 www.pqr-project.org
4 https://github.com/allr/purdue-fastr

lo se usa para resolver sobre qué método concreto de los que concretan la función se *despacha* el objeto).

- **Inmutabilidad**: cualquier función del lenguaje no puede modificar un objeto que existe en un entorno o capa superior desde donde ha sido llamada la función (técnicamente puede hacerlo, pero con metaprogramación, no con evaluación convencional).
- **Vectorización**: `R` incluye una serie de funcionales (funciones que toman otras como entrada) que vectorizan operaciones (hacen que se ejecuten de manera más eficiente, aprovechando la estructura del objeto) sobre matrices, vectores, datasets, listas, etc. Cuando puede usarse vectorización, es mucho más rápida que los bucles convencionales.

Se ha escogido `R` como herramienta para desarrollar el software dado su arraigado uso en estadística y en ciencia de datos. Aunque otros lenguajes como `Python` proporcionan herramientas tremendamente potentes para hacer ciencia de datos, `R` parece estar menos orientado a programadores puros y más a obtener resultados científicos a partir de lo programado. Se quería proporcionar una implementación de los algoritmos que ofreciera un balance entre eficiencia y facilidad de uso, y `R` parecía la mejor opción para alcanzar dicho objetivo. Además, la diseminación de los algoritmos principales (no únicamente en preprocesamiento de datos, sino en ciencia de datos en general) en varios paquetes (entre los de tratatamiento de datos no balanceados cabe citar a [`smotefamily`] o [`ROSE`] en `R`) hace que no haya una fuerte dependencia de un único paquete de software, como ocurriría con [`scikit-learn`] en `python`.

11.2 INSTALACIÓN

11.2.1 *Instalación de `R`*

La instalación de `R` en distribuciones Linux basadas en `Debian` puede efectuarse de la forma:

```
sudo apt install r-base rbase-dev
```

En distribuciones basadas en `Arch Linux`:

```
sudo pacman -S r
```

También es recomendable, aunque no necesario, instalar `RStudio` [5].

5 https://www.rstudio.com

11.2.2 *Instalación del software*

La instalación del paquete `imbalance` desarrollado se puede llevar a cabo una vez abierto desde una terminal `R` o `RStudio` usando el paquete [`devtools`]:

```
install.packages("devtools")
devtools::install_github("ncordon/imbalance")
```

Posterior a este paso, sólo falta cargar el paquete:

```
library(imbalance)
```

La instalación del paquete `devtools` (primera línea) no es necesaria si ya se tiene instalado.

Cualquier cuestión de documentación puede ser consultada en el anexo del presente trabajo, en la página de documentación online `https://ncordon.github.io/imbalance`, pestaña *Reference*; o usando la ayuda de manual de `R`, una vez cargado el paquete, con el nombre de la función precedido de `?` (ej. `?rwo`).

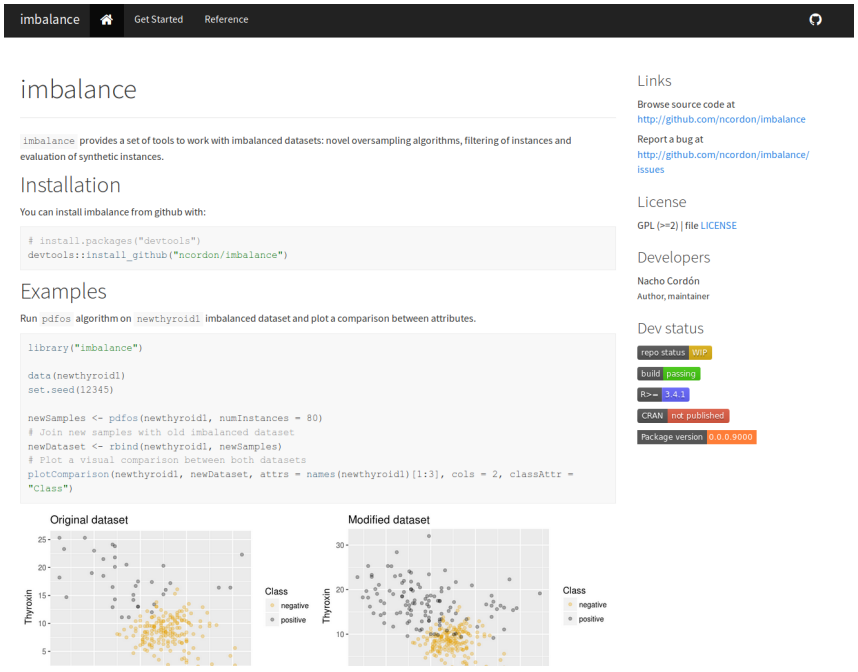


Figura 7.: Web de documentación del proyecto

Los métodos que exporta el paquete y que pueden usarse una vez se carga son:

- `mwmote`
- `racog`
- `wracog`
- `rwo`
- `pdfos`
- `neater`
- `plotComparison`

11.3 ESTRUCTURA DEL PAQUETE

El paquete, cuyo código fuente está disponible en un repositorio de Github [6], bajo licencia GPL2 y posteriores, presenta la siguiente estructura de directorios y archivos:

- `R`: En esta carpeta se encuentra código fuente de los algoritmos, cada uno en un archivo homónimo al nombre del algoritmo. También se encuentran los siguientes archivos:
  - `data.R`: contiene documentación en forma de comentarios de `Roxygen2` de la estructura de los datasets incluidos con el paquete, todos ellos extraídos del repositorio [KEEL].
  - `RccpExports.R`: usado por la librería `Rcpp` para almacenar wrappers de llamadas a funciones de `C++` que se encuentran en la carpeta `src` de la raíz del paquete.
  - `zzz.R`: (nombre del archivo por convenio), que se usa para liberar las librerías dinámicas generadas con `C++` en caso de que se libere el `Namespace` asociado al paquete.
  - `imbalance.R`: contiene la documentación del paquete para la ayuda `help`, accesible a través de:

    ```
    help(package="imbalance")
    ```

  - `utils.R`: contiene funciones auxiliares llamadas desde los algoritmos, y la función `plotComparison` para evaluar visualmente los resultados de un *oversampling* o *undersampling*.

6 https://github.com/ncordon/imbalance

- `data-raw`: hay un script `rkeel-data.R` en esta carpeta que contiene el procedimiento con el que se han leído y limpiado los datos desde archivos en formato KEEL. Es habitual distribuir este tipo de archivos en los fuentes de los paquetes, aunque las *builds* de los mismos no los contienen, por estar añadida esta carpeta al archivo `.Rbuildignore`.

- `data`: contiene archivos de datos `.rda` con los datasets con el tipo de datos correcto para cada columna y con el atributo de clase "Class" con posibles valores *positive* o *negative*, respetando su formato original. Ha sido necesario consultar no sólo el repositorio KEEL, sino también el repositorio de *machine learning* de [UCI]. Para acceder a la documentación sobre el formato de cada uno de ellos basta escribir en la consola de `R` el nombre del dataset precedido de `?` (ej. `?newthryoid1`). Los siguientes datasets están disponibles: `ecoli1`, `glass0`, `haberman`, `iris0`, `newthryoid1`, `wisconsin`, `yeast4`.

- `src`: contiene código auxiliar en `C++` usado para acelerar el cuerpo de los algoritmos homónimos. El archivo `RcppExports.cpp` contiene cabeceras de funciones que genera automáticamente el paquete `Rcpp`.

- `tests`: contiene los tests que pasa el paquete en la integración continua. Cada vez que una modificación es subida al repo se comprueba que ninguna función de las exportadas por el paquete falle, que las funciones comprueben correctamente los parámetros de entrada, que el paquete pase los tests en formato CRAN, etc.

11.4 METODOLOGÍA DE DESARROLLO

Los algoritmos han sido fundamentalmente programados en `R`, de manera que todos tienen una estructura de la forma:

```
samples ← cleanAndFormatSamples(samples)

# Cuerpo del algoritmo

samples ← originalFormat(samples)
return(samples)
```

Asimismo, se ha usado la librería [Rcpp], que proporciona una API de integración con `C++`, de manera que no hay que emplear manualmente llamadas a `.Call` en `R` y facilita la transformación de tipos

entre ambos lenguajes. También ofrece azúcar sintáctico inspirado en `R` disponible al programar `C++` apoyándonos en `Rcpp`. Además, para el desarrollo de dichos programas auxiliares hemos tenido que usar la librería [`Armadillo`], que proporciona una extensa variedad de operaciones matriciales y vectoriales en `C++`, y cuya sintaxis está inspirada en `Octave` [7].

Por último, cabe comentar que se ha empleado `Travis CI` [8] como servicio de integración continua con el repo de Github. De esta forma se han automatizado ejecuciones de tests unitarios y generación de documentación web. El archivo de configuración empleado para el servicio de integración continua está disponible en la raíz del repo, con nombre `.travis.yml`.

11.5 EJEMPLO DE USO DEL SOFTWARE

Ejecución del algoritmo PDFOS sobre el dataset `newthryoid1` y evaluación de resultados visualmente con la función `plotComparison`:

```
1  library("imbalance")
2
3  data(newthyroid1)
4  set.seed(12345)
5
6  newSamples <- pdfos(newthyroid1, numInstances = 80)
7  # Join new samples with old imbalanced dataset
8  newDataset <- rbind(newthyroid1, newSamples)
9  # Plot a visual comparison between both datasets
10 plotComparison(newthyroid1, newDataset,
11                attrs = names(newthyroid1)[1:3],
12                cols = 2, classAttr = "Class")
```

7 https://www.gnu.org/software/octave
8 https://travis-ci.org

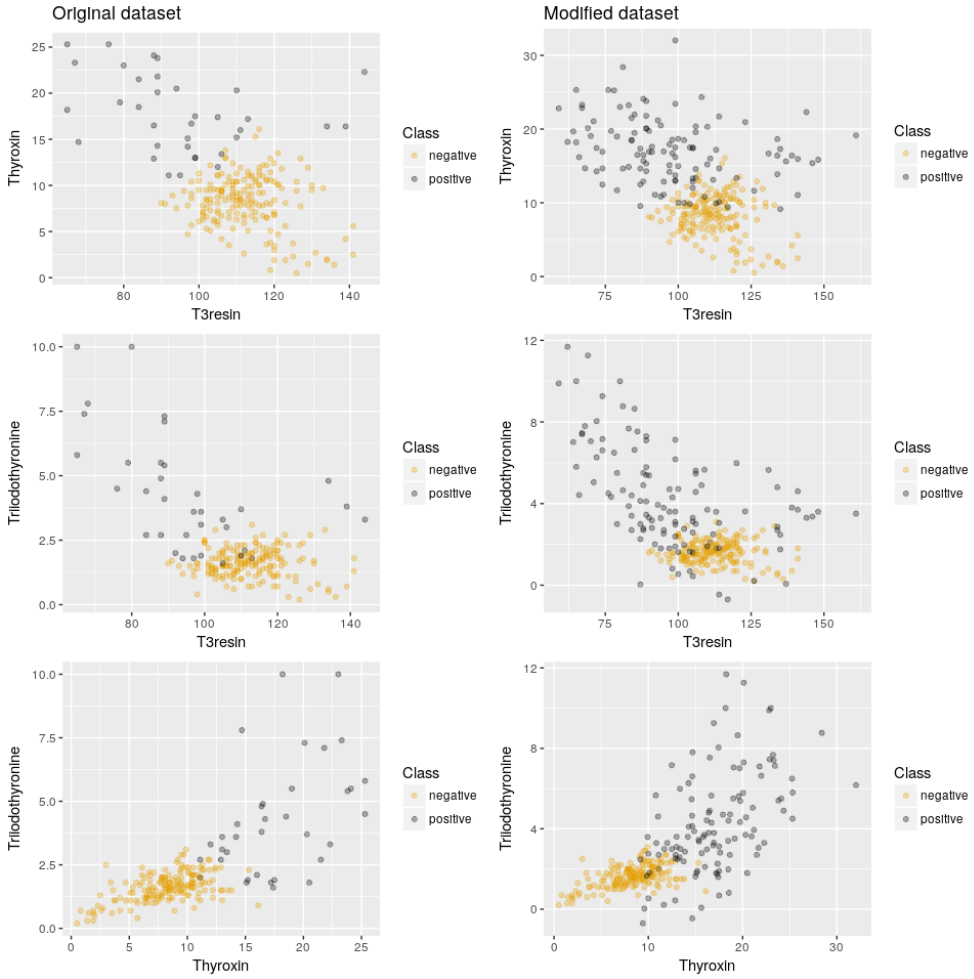


Figura 8.: PDFOS sobre los 3 primeros atributos de `newthryoid1`

Limpiado de las instancias generadas mediante NEATER:

```
filteredSamples ← neater(newthyroid1, newSamples,
                         iterations = 500)
#> [1] "14 samples filtered by NEATER"
filteredNewDataset ← rbind(newthyroid1, filteredSamples)
plotComparison(newthyroid1, filteredNewDataset,
               attrs = names(newthyroid1)[1:3])
```

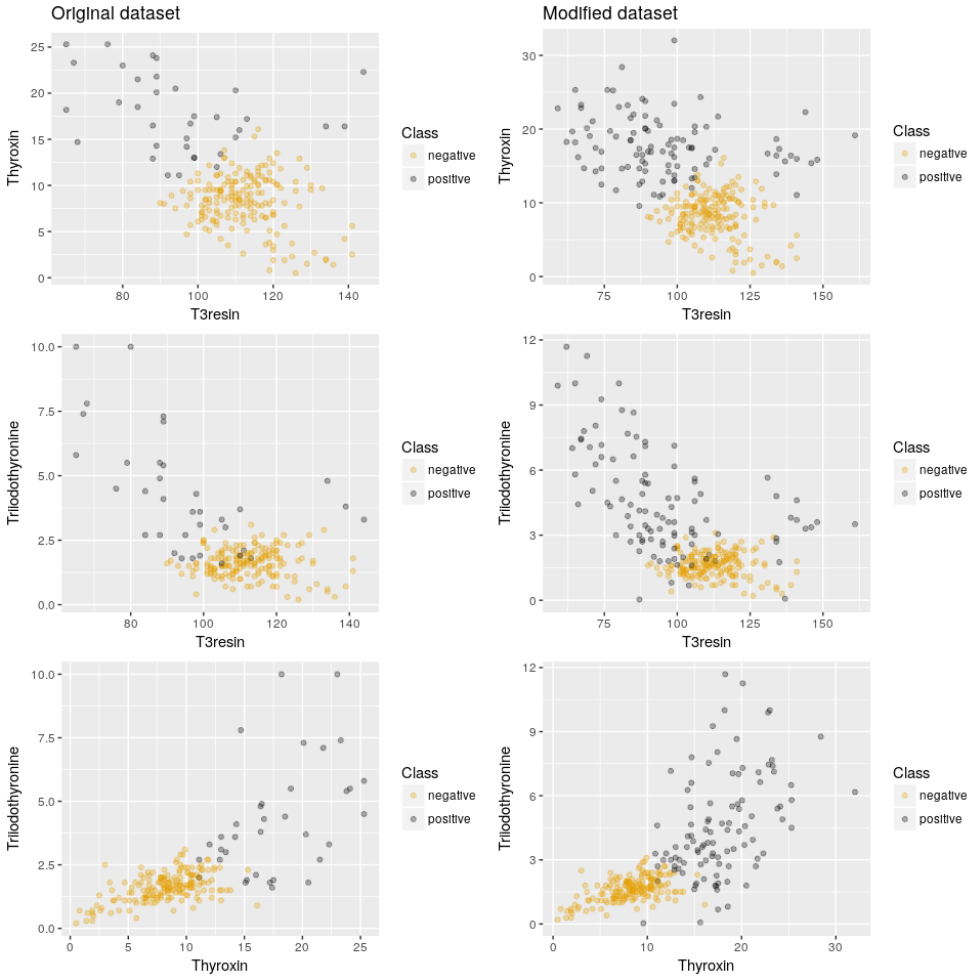


Figura 9.: Evaluación de NEATER sobre las instancias anteriores

12

EXPERIMENTACIÓN

En este capítulo proporcionamos una sencilla experimentación de los algoritmos implementados, aplicándolos al concepto de *small disjuncts* que se mencionó en 9.1, problema que surge como consecuencia del desbalanceo *intra clases*.

12.1 MARCO DE EXPERIMENTACIÓN

Se estudiarán los algoritmos MWMOTE, RWO, PDFOS sobre todos los datasets incluidos en el paquete `imbalance`. También se aplicará el algoritmo wRACOG sobre el único dataset de los del paquete enteramente discreto: `wisconsin`, puesto que en el resto de casos, con atributos continuos, no tendría sentido aplicar el algoritmo. También se hará una segunda experimentación, añadiendo filtrado con NEATER a cada uno de los algoritmos y datasets a los que pueden aplicarse.

Trabajaremos con la implementación de árboles C4.5 de [RWeka]: `J48`, usando árboles sin poda (opción `-U`), y sin mínimo de ejemplos por hoja (opción `-M=1`). Definiremos un *small disjunct* como una hoja del árbol con 3 o menos ejemplos. Nótese que no hay definición formal para el concepto, pues este se define como reglas que cubren pocos ejemplos. Calcularemos para cada dataset el número de *small disjuncts* (de ambas clases, puesto que se explicó en 9.1 que es un problema que puede afectar a ambas clases), así como la media del número de ejemplos que cubren las hojas del árbol.

Si alguno de los algoritmos de *oversampling* introdujera más *small disjuncts* en lugar de eliminarlos, la media de cobertura de las hojas bajaría, y el número de *small disjuncts* crecería. Por tanto teorizamos que si eliminamos desbalanceo en los datasets, el número de *small disjuncts* decrecerá, y la media de ejemplos por hoja aumentará. También trabajaremos con la sensibilidad, medida que se explicó en la sección 9.3.

Se dividirá cada conjunto en 3 particiones estratificadas por clase (esto es, la proporción de ejemplos de ambas clases será aproximadamente igual que en el conjunto sin particionar), y se intentará conseguir un ratio de desbalanceo del 80 % con cada uno de los algoritmos.

Se hará una media de las medidas descritas para cada dataset, y compararemos los resultados con los obtenidos para las 3 particiones sin ningún algoritmo aplicado. Para aplicar el algoritmo wRACOG se usará como conjunto de entrenamiento una partición y como conjunto de validación la siguiente.

La experimentación es completamente reproducible, y se encuentra disponible en la carpeta `experimentacion` del repositorio que aloja al presente trabajo [1]. Ahí pueden encontrarse dos archivos: `aux.R` y `small-disjuncts.R`. Basta ejecutar el segundo archivo para reproducir la experimentación.

12.2 CONTENIDO DE aux.r

`RWeka` no proporciona ningún método para obtener la cobertura de las hojas de los árboles que construye, así que ha sido necesario un método para *parsear* la impresión que se hace del árbol en forma de cadena de caracteres

```
leavesCoverage ← function(tree, classAttr){
  # retrieve tree in string format
  strgraph ← tree$classifier$graph()

  # select leaves corresponding to specified classAttr
  listmatches ← unlist(strsplit(strgraph,
                        paste(classAttr, "\\(", sep = " ")))
  whichNumeric ← grep("^[[:digit:]]+", listmatches, perl = TRUE)
  listmatches ← listmatches[whichNumeric]

  # extract coverage number
  coverages ← lapply(listmatches, function(x){
    y ← sub(").*", "", x)
    sub("/.*", "", y)
  })
  coverages ← as.numeric(unlist(coverages))
  coverages[coverages != 0]
}
```

Código 12.1: función `leavesCoverage` en `aux.R`

Asimismo, en dicho archivo también podemos encontrar las funciones `makePartition`, que recibe un `dataset` y un número de particiones `numPartitions` y genera particiones estratificadas a partir del da-

1 https://github.com/ncordon/tfg

taset; la función `infoSmallDisjuncts` que recibe un `dataset` y devuelve el cálculo de las dos medidas descritas, y la función `getResults`, que recibe un parámetro *booleano* indicando si queremos aplicar el filtro NEATER o no. Las funciones para particionar y obtener resultados fijan la semilla 12345 por defecto, con objeto de que la experimentación sea reproducible.

```
1  infoSmallDisjuncts <- function(dataset){
2    # make an unpruned C4.5 tree out of dataset
3    tree <- RWeka::J48(Class ~ ., dataset,
4                       control = Weka_control(U = list(TRUE),
5                                              M = list(1)))
6
7    positiveCoverage <- leavesCoverage(tree, classAttr = "positive")
8    negativeCoverage <- leavesCoverage(tree, classAttr = "negative")
9    coverages <- c(positiveCoverage, negativeCoverage)
10   sensitivity <- computeSensitivity(tree, dataset)
11
12   list(numSmallDisjuncts = length(which(coverages <= 3)),
13        meanCoverage = mean(coverages),
14        sensitivity = sensitivity)
15 }
```

Código 12.2: función `infoSmallDisjuncts` en `aux.R`

12.3 CONTENIDO DE small-disjuncts.r

Este archivo es el fichero principal de la experimentación. Carga librerías, el fichero auxiliar `aux.R`, los datos de `imbalance`, los particiona, crea un *wrapper* para `wRACOG` basado en `J48` y obtiene los resultados de la experimentación.

12.4 RESULTADOS

En los resultados, `none` representa el dataset original sin tratamiento de *oversampling*. En la columna correspondiente al algoritmo PDFOS faltan algunos datos debido a que las matrices de covarianza de los datos particionados resultan no invertibles, tal y como se explicó en la sección 10.4.4. La columna de wRACOG sólo incluye el único dataset discreto, `wisconsin`.

12.4.1 *Sin filtrado de instancias*

	none	mwmote	wracog	rwo	pdfos
ecoli1	2.67	3.00		17.67	
glasso	2.33	5.00		3.67	
haberman	1.33	6.33		3.33	4.33
iriso	0.00	0.00		0.00	0.00
newthyroid1	1.67	1.33		1.67	1.00
wisconsin	27.67	62.33	36.00	65.67	65.67
yeast4	3.00	16.67		5.00	

Cuadro 2.: Media de *small disjuncts* tras *oversampling*

	none	mwmote	wracog	rwo	pdfos
ecoli1	24.83	20.39		14.13	
glasso	13.30	8.76		11.07	
haberman	45.05	14.86		20.00	22.31
iriso	25.00	30.33		30.33	30.33
newthyroid1	19.89	30.00		20.40	23.40
wisconsin	6.74	3.88	5.68	3.65	3.65
yeast4	77.83	30.69		55.21	

Cuadro 3.: Tamaño medio de coberturas tras *oversampling*

	none	mwmote	wracog	rwo	pdfos
ecoli1	0.76	1.00		0.96	
glasso	0.97	0.98		1.00	
haberman	0.54	0.85		0.76	0.68
iriso	1.00	1.00		1.00	1.00
newthyroid1	0.97	1.00		0.99	1.00
wisconsin	1.00	1.00	1.00	1.00	1.00
yeast4	0.33	1.00		0.99	

Cuadro 4.: Media de sensibilidad tras *oversampling*

12.4.2 *Con filtrado de instancias*

	none	mwmote	wracog	rwo	pdfos
ecoli1	2.67	2.67		17.67	
glasso	2.33	3.67		3.33	
haberman	1.33	7.33		3.00	5.33
iriso	0.00	0.00		0.00	0.00
newthyroid1	1.67	1.33		2.33	1.00
wisconsin	27.67	62.33	34.33	65.00	63.67
yeast4	3.00	9.67		4.00	

Cuadro 5.: Media de *small disjuncts* tras *oversampling* y filtrado

	none	mwmote	wracog	rwo	pdfos
ecoli1	24.83	19.59		14.10	
glasso	13.30	9.80		14.77	
haberman	45.05	9.37		14.32	12.55
iriso	25.00	30.33		30.33	30.33
newthyroid1	19.89	28.83		21.19	23.85
wisconsin	6.74	3.88	5.78	3.67	3.71
yeast4	77.83	35.66		52.80	

Cuadro 6.: Tamaño medio de coberturas tras *oversampling* y filtrado

	none	mwmote	wracog	rwo	pdfos
ecoli1	0.76	0.98		0.96	
glasso	0.97	0.98		0.99	
haberman	0.54	0.83		0.85	0.87
iriso	1.00	1.00		1.00	1.00
newthyroid1	0.97	1.00		0.99	1.00
wisconsin	1.00	1.00	1.00	1.00	1.00
yeast4	0.33	0.99		0.99	

Cuadro 7.: Media de sensibilidad tras *oversampling* y filtrado

Se puede observar cómo nuestros algoritmos consiguen en algunos casos reducir el número de hojas de pequeña cobertura existente en los datasets, y aumentar los tamaños medios de las mismas. En especial se desprende que el algoritmo NEATER mejora los resultados obtenidos sin filtrado. Hemos usado en todos ellos los parámetros por defecto, y podrían obtenerse mejores resutlados si intentásemos optimizar los mismos. Además, la medida de *small disjuncts* que está siendo empleada es bastante arbitraria, y podría intentarse encontrar

una más adaptada a cada dataset: hojas que tuviesen una cobertura inferior al 80 % de la media, por ejemplo.

A esto hay que sumar que en algunos casos `J48` puede estar creando una única hoja donde clasifique a todos los ejemplos dentro de la misma clase, con lo cual obtendríamos un tamaño de cobertura bastante grande. Ello no quiere decir que si el *oversampling* disminuye el tamaño medio de cobertura, su funcionamiento no sea el correcto. Prueba de esto es que los algoritmos implementados aumentan notablemente la sensibilidad de los modelos, lo cual indica que consiguen su objetivo de mejorar la clasificación de instancias positivas.

Parte III

CONCLUSIONES

13

CONCLUSIONES Y VÍAS FUTURAS

En el presente trabajo se han descrito una serie de formalizaciones sobre la teoría del aprendizaje estadístico, que dan lugar a teoremas tan complejos como el teorema fundamental del aprendizaje PAC o el teorema de No Free Lunch. Ha constituido un área difícil de estudiar puesto que requiere conocimientos de muchas disciplinas, aunque se ha alcanzado un buen nivel de asimilación de la teoría presentada.

También se ha conseguido un objetivo fundamental, como era el desarrollo de un software de la temática a tratar, dando unas nociones básicas sobre diversos algoritmos noveles que tratan el problema de la clasificación no balanceada. Se ha comprobado a través de la lectura de *papers* sobre la temática que este es un tema muy estudiado, no sólo a nivel de informática, sino también de matemática teórica.

Como principales vías futuras que continúen la línea de lo presentado hasta ahora, cabría proponer:

- Caracterizar el problema de la clasificación no balanceada a través de los conceptos de dimensión Vapnik-Chervonenkis. Trabajos como [He and Garcia, 2009] aseguran que dicho problema ha sido muy tratado con esta teoría, aunque de manera indirecta, a través de modificación de clasificadores SVM. Tras la asimilación de la teoría aquí descrita, sería deseable poder llegar a una definición a través de dimensión VC que capturara el concepto de *instancias raras*.

- Leer e implementar [Pourhabib et al., 2015], un excelente *paper* de la revista *Journal of Machine Learning Research* que trata el problema del *oversampling* desde una perspectiva muy matemática, llevándose el espacio de dimensiones a un espacio Hilbert, usando funciones kernel, y empleando originales ideas como que los datos sintéticos, al proyectarlos a un espacio de menor dimensión, deberían quedar cerca de los de la clase minoritaria. Este *paper* se ha leído varias veces e intentado implementar el algoritmo que propone para este trabajo, pero contiene muchas referencias cruzadas a libros y otros artículos de matemáticas y su dificultad quedaba fuera del alcance por el momento. Ahora que por ejemplo se dispone de más conocimiento sobre funciones kernel, porque ha sido necesario para comprender el algo-

ritmo PDFOS, sería deseable intentar una lectura más profunda y la implementación.

- Actualmente `imbalance` no está alojado en el repositorio CRAN de `R`. De aquí a unas semanas se espera enviar el software para someterlo a revisión y que pueda ser colgado en dicho repositorio. Esto implicaría que los usuarios no tendrían que instalarse una versión en desarrollo desde Github, y que podría instalarse haciendo simplemente desde `R`:

```
install.packages("imbalance")
```

  Aparte, supondría un reconocimiento a que el paquete es usable, estable y tiene una documentación apropiada.

- Los artículos de Wikipedia en inglés [1] y en español [2] sobre la temática del aprendizaje PAC son escuetos, y en el caso del artículo en español está marcado como "mala traducción". Un objetivo deseable sería ampliar y corregir dichos artículos, añadiendo buenas referencias, siguiendo la política de conocimiento libre, que posibilita un acceso abierto a la cultura y el conocimiento.

1 https://en.wikipedia.org/wiki/Probably_approximately_correct_learning
2 https://es.wikipedia.org/wiki/Aprendizaje_PAC

ANEXO

A

DOCUMENTACIÓN DEL PAQUETE

`mwmote`	*Majority weighted minority oversampling technique for imbalance dataset learning*

Description

Modification for SMOTE technique which overcomes some of the problems of the SMOTE technique when there are noisy instances, in which case SMOTE would generate more noisy instances out of them.

Usage

```
mwmote(dataset, numInstances, kNoisy = 5, kMajority = 3,
       kMinority, threshold = 5, cmax = 2, cclustering = 3,
       classAttr = "Class")
```

Arguments

`dataset`	`data.frame` to treat. All columns, except `classAttr` one, have to be numeric or coercible to numeric.
`numInstances`	Integer. Number of new minority examples to generate.
`kNoisy`	Integer. Parameter of euclidean KNN to detect noisy examples as those whose whole kNoisy neighbourhood is from the opposite class.
`kMajority`	Integer. Parameter of euclidean KNN to detect majority borderline examples as those who are in any kMajority-neighbourhood of minority instances. Should be a low integer.

`kMinority` Integer. Parameter of euclidean KNN to detect minority borderline examples as those who are in the KMinority-neighbourhood of majority borderline ones. It should be a large integer. By default if not parameter is fed to the function, $|S^+|/2$ where S^+ is the set of minority examples.

`threshold` Numeric. A positive real indicating how much we measure tolerance of closeness to the boundary of minority boundary examples. A large integer indicates more margin of distance for a example to be considerated important boundary one.

`cmax` Numeric. A positive real indicating how much we measure tolerance of closeness to the boundary of minority boundary examples. The larger this number, the more we are valuing boundary examples.

`cclustering` Numeric. A positive real for tuning the output of an internal clustering. The larger this parameter, the more area focused is going to be the oversampling.

`classAttr` `character`. Indicates the class attribute from `dataset`. Must exist in it.

Value

A `data.frame` with the same structure as `dataset`, containing the generated synthetic examples.

Examples

```
data(iris0)
set.seed(12345)

# Generates new minority examples
newSamples <- mwmote(iris0, 100, classAttr = "Class")
```

---

racog *Rapidly converging Gibbs algorithm.*

Description

Allows you to treat imbalanced discrete numeric datasets by generating synthetic minority examples, approximating their probability distribution.

Usage

```
racog(dataset, numInstances, burnin = 100, lag = 20,
      classAttr = "Class")
```

Arguments

`dataset`	`data.frame` to treat. All columns, except `classAttr` one, have to be numeric or coercible to numeric.
`numInstances`	Integer. Number of new minority examples to generate.
`burnin`	Integer. It determines how many examples generated for a given one are going to be discarded firstly. By default, 100.
`lag`	Integer. Number of iterations between new generated example for a minority one. By default, 20.
`classAttr`	`character`. Indicates the class attribute from `dataset`. Must exist in it.

Details

Approximates minority distribution using Gibbs Sampler. Dataset must be discretized and numeric. In each iteration, it builds a new sample using a Markov chain. It discards first `burnin` iterations, and from then on, each `lag` iterations, it validates the example as a new minority example. It generates $d(iterations - burnin)/lag$ where d is minority examples number.

Value

A `data.frame` with the same structure as `dataset`, containing the generated synthetic examples.

Examples

```
data(iris0)
set.seed(12345)

# Generates new minority examples
newSamples <- racog(iris0, 100, classAttr = "Class")
```

`wracog` *Wrapper for rapidly converging Gibbs algorithm.*

Description

Generates synthetic minority examples by approximating their probability distribution until sensitivity of `wrapper` over `validation` cannot be further improved. Works only on discrete numeric datasets.

Usage

```
wracog(train, validation, wrapper, slideWin = 10,
       threshold = 0.02, classAttr = "Class", ...)
```

Arguments

`train`	`data.frame`. A initial dataset to generate first model. All columns, except `classAttr` one, have to be numeric or coercible to numeric.
`validation`	`data.frame`. A dataset to compare results of consecutive classifiers. Must have the same structure of `train`.
`wrapper`	An `S3` object. There must exist a method `trainWrapper` implemented for the class of the object, and a `predict` method implemented for the class of the model returned by `trainWrapper`.
`slideWin`	Number of last sensitivities to take into account to meet the stopping criteria. By default, 10.

`threshold`	Threshold that the last `slideWin` sensitivities mean should reach. By default, 0.02.
`classAttr`	`character`. Indicates the class attribute from `train` and `validation`. Must exist in them.
`...`	further arguments for `wrapper`.

Details

Until the last `slideWin` executions of `wrapper` over `validation` dataset reach a mean sensitivity lower than `threshold`, the algorithm keeps generating samples using Gibbs Sampler, and adding misclassified samples with respect to a model generated by a former train, to the train dataset. Initial model is built on initial `train`.

Value

A `data.frame` with the same structure as `train`, containing the generated synthetic examples.

Examples

```
data(haberman)
set.seed(12345)
myWrapper <- structure(list(), class="C50Wrapper")
trainWrapper.C50Wrapper <- function(wrapper, train, trainClass)
    {
  C50::C5.0(train, trainClass)
}

trainFold <- sample(1:nrow(haberman), nrow(haberman)/2, FALSE)
newSamples <- wracog(haberman[trainFold, ],
                     haberman[-trainFold, ],
                     myWrapper, classAttr = "Class")
```

`rwo` *Random walk oversampling*

Description

Generates synthetic minority examples for a dataset trying to preserve the variance and mean of the minority class. Works on every type of dataset.

Usage

```
rwo(dataset, numInstances, classAttr = "Class")
```

Arguments

`dataset`	`data.frame` to treat. All columns, except `classAttr` one, have to be numeric or coercible to numeric.
`numInstances`	Integer. Number of new minority examples to generate.
`classAttr`	`character`. Indicates the class attribute from `dataset`. Must exist in it.

Details

Generates `numInstances` new minority examples for `dataset`, adding to the each numeric column of the j-th example its variance scalated by the inverse of the number of minority examples and a factor following a $N(0,1)$ distribution which depends on the example. When the column is nominal, it uses a roulette scheme.

Value

A `data.frame` with the same structure as `dataset`, containing the generated synthetic examples.

Examples

```
data(iris0)
set.seed(12345)

newSamples ← rwo(iris0, 100, classAttr = "Class")
```

`pdfos` *Probability density function estimation based oversampling*

Description

Generates synthetic minority examples for a numerical dataset approximating a Gaussian multivariate distribution which best fits the minority data.

Usage

```
pdfos(dataset, numInstances, classAttr = "Class")
```

Arguments

`dataset`	`data.frame` to treat. All columns, except `classAttr` one, have to be numeric or coercible to numeric.
`numInstances`	Integer. Number of new minority examples to generate.
`classAttr`	`character`. Indicates the class attribute from `dataset`. Must exist in it.

Details

To generate the synthetic data, it approximates a normal distribution with mean a given example belonging to the minority class, and whose variance is the minority class variance multiplied by a constant; that constant is computed so that it minimizes the mean integrated squared error of a Gaussian multivariate kernel function.

Value

A `data.frame` with the same structure as `dataset`, containing the generated synthetic examples.

Examples

```
data(iris0)
set.seed(12345)

newSamples <- pdfos(iris0, 100, classAttr = "Class")
```

`neater` *Fitering of oversampled data based on non-cooperative game theory*

Description

Filters oversampled examples from a binary class `dataset` using game theory to find out if keeping an example is worthy enough.

Usage

```
neater(dataset, newSamples, k = 3, iterations = 100,
       smoothFactor = 1, classAttr = "Class")
```

Arguments

`dataset`	The original `data.frame`. All columns, except `classAttr` one, have to be numeric or coercible to numeric.
`newSamples`	A `data.frame` containing the samples to be filtered. Must have the same structure as `dataset`.
`k`	Integer. Number of nearest neighbours to use in KNN algorithm to rule out samples. By default, 3.
`iterations`	Integer. Number of iterations for the algorithm. By default, 100.
`smoothFactor`	A positive `numeric`. By default, 1.
`classAttr`	`character`. Indicates the class attribute from `dataset` and `newSamples`. Must exist in them.

Details

Uses game theory and Nash equilibriums to calculate the minority examples probability of trully belonging to the minority class. It discards examples which at the final stage of the algorithm have more probability of being a majority example than a minority one.

Value

Filtered samples as a `data.frame` with same structure as `newSamples`.

Examples

```
data(iris0)
set.seed(12345)

newSamples <- smotefamily::SMOTE(iris0[,-5], iris0[,5])$syn_
    data
# SMOTE overrides Class attr turning it into class
# and dataset must have same class attribute as newSamples
names(newSamples) <- c(names(newSamples)[-5], "Class")

neater(iris0, newSamples, k = 5, iterations = 100,
       smoothFactor = 1, classAttr = "Class")
```

`plotComparison` *Plots comparison between the original and the new balanced dataset.*

Description

It plots a grid of one to one variable comparison, placing the former dataset graphics next to the balanced one, for each pair of attributes.

Usage

```
plotComparison(dataset, anotherDataset, attrs, cols = 2,
               classAttr = "Class")
```

Arguments

`dataset`	A `data.frame`. The former imbalanced dataset.
`anotherDataset`	A `data.frame`. The balanced dataset. `dataset` and `anotherDataset` must have the same columns.
`attrs`	Vector of `character`. Attributes to compare. The function generates each posible combination of attributes to build the comparison.
`cols`	Integer. It indicates the number of columns of resulting grid. Must be an even number. By default, 2.
`classAttr`	`character`. Indicates the class attribute from `dataset`. Must exist in it.

Value

Plot of 2D comparison between the variables.

Examples

```
data(iris0)
set.seed(12345)

rwoSamples <- rwo(iris0, numInstances = 100)
rwoBalanced <- rbind(iris0, rwoSamples)
plotComparison(iris0, rwoBalanced, names(iris0), cols = 2,
    classAttr = "Class")
```

BIBLIOGRAFÍA

[Almogahed and Kakadiaris 2014] Almogahed, B. A. ; Kakadiaris, I. A.: NEATER: Filtering of Over-Sampled Data Using Non-Cooperative Game Theory. In: *Soft Computing* 19 (2014), Nr. 11, p. 3301–3322. – URL https://doi.org/10.1007/s00500-014-1484-5 4

[Barua et al. 2014] Barua, Sukarna ; Islam, Md. M. ; Yao, Xin ; Murase, Kazuyuki: MWMOTE-Majority Weighted Minority Oversampling Technique for Imbalanced Data Set Learning. In: *IEEE Transactions on Knowledge and Data Engineering* 26 (2014), Nr. 2, p. 405–425. – URL https://doi.org/10.1109/tkde.2012.232 4

[Chawla et al. 2002] Chawla, N. V. ; Bowyer, K. W. ; Hall, L. O. ; Kegelmeyer, W. P.: SMOTE: Synthetic Minority Over-sampling Technique. In: *Journal of Artificial Intelligence Research* 16 (2002), p. 321–357. – URL https://doi.org/doi:10.1613/jair.953 67

[Das et al. 2015] Das, Barnan ; Krishnan, Narayanan C. ; Cook, Diane J.: RACOG and wRACOG: Two Probabilistic Oversampling Techniques. In: *IEEE Transactions on Knowledge and Data Engineering* 27 (2015), Nr. 1, p. 222–234. – URL https://doi.org/10.1109/tkde.2014.2324567 4, 75

[Eddelbuettel and François 2011] Eddelbuettel, Dirk ; François, Romain: Rcpp: Seamless R and C++ Integration. In: *Journal of Statistical Software* 40 (2011), Nr. 8, p. 1–18. – URL http://www.jstatsoft.org/v40/i08/ 93

[Fetaya 2016] Fetaya, Ethan: *Introduction to Statistical Learning Theory*. 2016. – URL http://www.wisdom.weizmann.ac.il/~ethanf/teaching/ItSLT_16 4

[Gao et al. 2014] Gao, Ming ; Hong, Xia ; Chen, Sheng ; Harris, Chris J. ; Khalaf, Emad: PDFOS: Pdf Estimation Based Over-Sampling for Imbalanced Two-Class Problems. In: *Neurocomputing* 138 (2014), p. 248–259. – URL https://doi.org/10.1016/j.neucom.2014.02.006 4, 83, 84

[Genz et al. 2017] Genz, Alan ; Bretz, Frank ; Miwa, Tetsuhisa ; Mi, Xuefei ; Leisch, Friedrich ; Scheipl, Fabian ; Hothorn, Torsten: *mvtnorm: Multivariate Normal and t Distributions*, 2017. – URL https://CRAN.R-project.org/package=mvtnorm. – R package version 1.0-

6 85

[Gillespie and Lovelace 2017] Gillespie, Colin ; Lovelace, Robin: *Efficient R programming*. O'Reilly, 2017. – URL https://csgillespie.github.io/efficientR/ 5

[He and Garcia 2009] He, Haibo ; Garcia, E.A.: Learning From Imbalanced Data. In: *IEEE Transactions on Knowledge and Data Engineering* 21 (2009), Nr. 9, p. 1263–1284. – URL https://doi.org/10.1109/tkde.2008.239 4, 62, 104

[Hornik et al. 2009] Hornik, Kurt ; Buchta, Christian ; Zeileis, Achim: Open-Source Machine Learning: R Meets Weka. In: *Computational Statistics* 24 (2009), Nr. 2, p. 225–232 97

[J.Alcalá et al. 2010] J.Alcalá ; A.Fernández ; J.Luengo ; J.Derrac ; S.García ; L.Sánchez ; F.Herrera: Keel Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Anlysis Framework. In: *Multiple-Valued Logic and Soft Computing* 17 (2010), p. 255–287. – URL http://sci2s.ugr.es/keel 63, 92

[Jo and Japkowicz 2004] Jo, Taeho ; Japkowicz, Nathalie: Class Imbalances Versus Small Disjuncts. In: *ACM SIGKDD Explorations Newsletter* 6 (2004), Nr. 1, p. 40. – URL https://doi.org/10.1145/1007730.1007737 63

[Lichman 2013] Lichman, M.: *UCI Machine Learning Repository*. 2013. – URL http://archive.ics.uci.edu/ml 93

[Loève 1977] Loève, M.: *Probability Theory I*. Springer New York, 1977 (Graduate Texts in Mathematics) 4

[Lunardon et al. 2014] Lunardon, Nicola ; Menardi, Giovanna ; Torelli, Nicola: ROSE: a Package for Binary Imbalanced Learning. In: *R Journal* 6 (2014), Nr. 1, p. 82–92 90

[Pedregosa et al. 2011] Pedregosa, F. ; Varoquaux, G. ; Gramfort, A. ; Michel, V. ; Thirion, B. ; Grisel, O. ; Blondel, M. ; Prettenhofer, P. ; Weiss, R. ; Dubourg, V. ; Vanderplas, J. ; Passos, A. ; Cournapeau, D. ; Brucher, M. ; Perrot, M. ; Duchesnay, E.: Scikit-learn: Machine Learning in Python. In: *Journal of Machine Learning Research* 12 (2011), p. 2825–2830 90

[Pourhabib et al. 2015] Pourhabib, A. ; Mallick, B. K. ; Ding, Yu: Absent Data Generating Classifier for Imbalanced Class Sizes. In: *Journal of Machine Learning Research* 16 (2015), p. 2695–2724. – URL http://jmlr.org/papers/volume16/pourhabib15a/pourhabib15a.pdf 104

[Probability.net] Probability.net: *Caratheodory extension*. – URL http://www.probability.net/WEBcaratheodory.pdf 4

[R Core Team 2013] R Core Team: *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing (event), 2013. – URL http://www.R-project.org/. – ISBN 3-900051-07-0 89

[Sanderson and Curtin 2016] Sanderson, Conrad ; Curtin, Ryan: Armadillo: a template-based C++ library for linear algebra. In: *Journal of Open Source Software* 1 (2016), p. 26. – URL http://arma.sourceforge.net/armadillo_joss_2016.pdf 94

[Shalev-Shwartz and Ben-David 2014] Shalev-Shwartz, Shai ; Ben-David, Shai: *Understanding Machine Learning*. Cambridge University Press, 2014. – URL http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning 4

[Silverman 1986] Silverman, B. W.: *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986. – ISBN 0412246201 4, 84

[Siriseriwan 2016] Siriseriwan, Wacharasak: *smotefamily: A Collection of Oversampling Techniques for Class Imbalance Problem Based on SMOTE*, 2016. – URL https://CRAN.R-project.org/package=smotefamily. – R package version 1.0 90

[Wickham 2015] Wickham, Hadley: *R packages*. O'Reilly, 2015. – URL http://r-pkgs.had.co.nz/ 4

[Wickham and Chang 2017] Wickham, Hadley ; Chang, Winston: *devtools: Tools to Make Developing R Packages Easier*, 2017. – URL https://CRAN.R-project.org/package=devtools. – R package version 1.13.3 91

[Wikipedia 2017a] Wikipedia: *Hoeffding's inequality — Wikipedia, The Free Encyclopedia*. 2017. – URL https://en.wikipedia.org/wiki/Hoeffding's_inequality. – Recurso web; accedido 04-08-2017 4

[Wikipedia 2017b] Wikipedia: *Hoeffding's lemma — Wikipedia, The Free Encyclopedia*. 2017. – URL https://en.wikipedia.org/wiki/Hoeffding's_lemma. – Recurso web; accedido 04-08-2017 4

[Wikipedia 2017c] Wikipedia: *Markov's inequality — Wikipedia, The Free Encyclopedia*. 2017. – URL https://en.wikipedia.org/wiki/Markov's_inequality. – Recurso web; accedido 18-07-2017 4

[Zhang and Li 2014] Zhang, Huaxiang ; Li, Mingfang: RWO-Sampling: A Random Walk Over-Sampling Approach To Imbalan-

ced Data Classification. In: *Information Fusion* 20 (2014), p. 99–116. – URL https://doi.org/10.1016/j.inffus.2013.12.003 4