Trabajo Fin de Máster
Máster en Ingeniería Industrial

Codificación de variables categóricas en aprendizaje automático

Autor: Adrián Rocha Íñigo

Tutores: Amparo Núñez Reyes

Fernando Pavón Pérez

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Máster
Máster en Ingeniería Industrial

Codificación de variables categóricas en aprendizaje automático

Autor:
Adrián Rocha Íñigo

Tutores:
Amparo Núñez Reyes
Profesora Titular

Fernando Pavón Pérez
CEO Gamco S.L

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Máster: Codificación de variables categóricas en aprendizaje automático

Autor: Adrián Rocha Íñigo
Tutores: Amparo Núñez Reyes
Fernando Pavón Pérez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Este trabajo ha sido posible gracias al proyecto *SEKAS – Arquitectura de conocimiento con auto-aprendizaje y ajuste autónomo para aplicaciones de seguridad* del *Ministerio de Ciencia, Innovación y Universidades* (Nº dossier: RTC-2017-6178-8).

En primer lugar, me gustaría dar las gracias a todos los buenos profesores por cuyas manos he pasado. Su dedicación, entrega y amor por la enseñanza conforman la base de este trabajo y mis estudios en ingeniería. En especial, le agradezco a Amparo Núñez la confianza depositada en mí para llevar a cabo esta tarea, y la ayuda que siempre me ha proporcionado cuando la he necesitado.

También a Fernando Pavón y Adrián García, quienes, con infinita paciencia, me brindaron su valioso tiempo y conocimientos para ayudarme a vislumbrar los primeros trazos de este inmenso mundo de la inteligencia artificial.

Por último, y más importante, gracias a mis padres, por hacerme la persona que soy hoy, por enseñarme a aspirar cada vez más alto y, porque sin sus esfuerzos, nada de esto sería posible. A mi hermana, por ser el mejor ejemplo a seguir. Y a mi pareja, porque su apoyo incondicional es un pilar fundamental.

Adrián Rocha Íñigo
Máster en Ingeniería Industrial

Sevilla, 2020

Resumen

Las variables categóricas aparecen con gran frecuencia en los conjuntos de datos de problemas reales. Sin embargo, debido a su naturaleza, no pueden ser tratadas directamente por un número significativo de técnicas de aprendizaje automático. Para afrontar esta situación y poder sacar provecho de la información que estas variables condensan, se van a estudiar diversas técnicas de codificación que permiten transformarlas en otras variables continuas que sí son más aptas para ser empleadas en la resolución de problemas. De esta manera, el propósito de este trabajo consiste en el análisis y comparación de dichas codificaciones.

Abstract

The categorical variables frequently appear in real problems' datasets. However, due to their nature, they cannot be directly addressed by a significant number of automatic learning techniques. To handle this situation and to be able to take advantage of the information that these variables condense, we will study various coding techniques that allow them to be transformed into other continuous variables which are more suitable to be used in problem solving.

Índice

Índice de Figuras

Índice de Tablas

Notación

sim	Similitud
$dist$	Distancia
$\mathbb{R}$	Conjunto de los números reales
$\mathbb{N}$	Conjunto de los número naturales
$\mathscr{D}$	Conjunto de datos
d	Número de muestras en el conjunto de datos
$\mathbf{X}^k$	Vector de características de la muestra k-ésima
X_i^k	Elemento i-ésimo del vector de características para la muestra k-ésima
m	Tamaño del vector de características
$\mathscr{X}_i$	Conjunto de posibles valores para el elemento i-ésimo del vector de características
p_i	Tamaño del conjunto de posibles valores de la variable i-ésima
${}_zx_i$	Elemento genérico z-ésimo del conjunto $\mathscr{X}_i$
$\mathbf{Y}^k$	Vector de salida de la muestra k-ésima
Y_j^k	Elemento j-ésimo del vector de características para la muestra k-ésima
$\hat{\mathbf{Y}}^k$	Valor predicho de salida de la muestra k-esima
n	Tamaño del vector de salida
$\mathscr{Y}_j$	Conjunto de posibles valores para el elemento j-ésimo del vector de salida
q_j	Tamaño del conjunto de posibles valores de la variable de salida j-ésima
${}_sy_j$	Elemento genérico del conjunto $\mathscr{Y}_j$
Φ	Aplicación de codificación de variables categóricas
${}_z\Phi_i$	Vector de codificación de la categoría genérica ${}_zx_i$
a_i	Tamaño del vector de codificación de la categoría genérica ${}_zx_i$
${}_z\phi_{i,t}$	Elemento t-ésimo del vector ${}_z\Phi_i$
$\rightarrow$	Correspondencia funcional
e	Número e
:	Tal que
$\mid$	Tal que
$\forall$	Para todo
$\in$	Pertenece
$\cup$	Unión de conjuntos
$\cap$	Intersección de conjuntos
$\leq$	Menor o igual

$\geq$	Mayor o igual
$=$	Igual
$\neq$	Diferente
$\sum_a^b$	Sumatorio de a a b
$minimo\{A\}$	Valor mínimo del conjunto A
$\arg\max\{A\}$	Argumento que maximiza la expresión A
$\log_a(b)$	Logaritmo en base a de b
$\ln a$	Logaritmo neperiano de a
$P(A)$	Probabilidad de A
$P(A\|B)$	Probabilidad condicional de A dado B
f_a	Frecuencia absoluta del suceso a
$E(A)$	Esperanza matemática de A
$E(A\|B)$	Esperanza condicional de A dado B
μ	Media aritmética
σ	Desviación típica
δ_{ab}	Delta de Kronecker entre las variables a y b
δ_{ab}^c	Vector de dimensión c de deltas de Kronecker
$\Lambda()$	Función de cálculo de pesos de ponderación
${}^{s}\overline{\phi}^{y}$	Valor medio de codificación para la salida s-ésima
θ	Vector con todas las variables de codificación
$WoE(H;E)$	Peso de la evidencia a favor de H dado E
$sim(a,b)$	Similitud entre las categorías a y b
$d(a,b)$	Distancia entre las observaciones a y b
$\mathscr{K}$	Conjunto de observaciones que conforman la vecindad
k	Número de vecinos que conforman la vecindad
VN	Número de verdaderos negativos
VP	Número de verdaderos positivos
FN	Número de falsos negativos
FP	Número de falsos positivos
SEN	Sensibilidad
PFA	Porcentaje de falsas alarmas
EFF	Eficiencia
$MAPE$	Error porcentual medio absoluto

1 Introducción

En los últimos años el término Inteligencia Artificial ha cobrado cada vez más relevancia y casi se ha convertido en una constante en nuestro día a día. Si bien es cierto que este hecho puede ser debido a algunas campañas de marketing que publicitan productos que aparentemente integran dicha tecnología con el objetivo de atraer al consumidor, no podemos dudar que este campo perteneciente a las Ciencias de la Computación ha progresado y evolucionado vertiginosamente desde sus inicios hasta la actualidad, otorgando en numerosas situaciones sorprendentes resultados previamente inalcanzables.

La primera vez que se empleó el término IA fue en 1956 durante la Conferencia de Dartmouth por John McCarthy, quien la define como: *"It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable."* [1]

No obstante, las investigaciones en esta línea ya habían comenzado antes, como el caso de Warren McCulloch y Walter Harry Pitts, quienes en 1943 definieron el primer modelo matemático de una red neuronal artificial [2], dando nombre a la que se conoce hoy en día como neurona de McCulloch-Pitts.

Desde entonces hasta la actualidad, numerosos avances han aparecido en áreas que tenemos muy presentes en nuestra vida cotidiana como, por ejemplo: el reconocimiento de caracteres, personas y objetos mediante cámaras; la existencia de *chatbots* en páginas web para prestar soporte al usuario; los asistentes virtuales capaces de mantener una comunicación oral con las personas mientras cumplen diferentes tareas, los cuales han logrado bastante auge en el hogar y la automoción; y la conducción autónoma, un campo en continua evolución donde aún queda mucho por mejorar.

De igual modo, la Inteligencia Artificial ha influido en campos como la medicina [3], las finanzas [4], la economía [5], el transporte [6], las comunicaciones [7], el ocio [8], la industria y la robótica [9], entre otros muchos sectores más.

En un gran número de las aplicaciones previamente mencionadas existe un denominador común, el uso de cantidades inmensas de datos. Mediante estos se puede llegar a generar conocimiento con el objetivo de construir recursos con la capacidad de originar nuevos datos, extraer conclusiones, tomar decisiones o crear nuevos recursos, todo ello con la finalidad de alcanzar el objetivo del problema que se trata de resolver de la forma más óptima posible.

Es por este uso masivo de datos y sus implicaciones, que importantes empresas han ampliado su línea de negocio en la recopilación, tratamiento, empleo y distribución de ellos; a la par que diversos organismos han comenzado a regular en este ámbito para velar por la privacidad y derechos de la población.

1.1 Motivación y objetivos del proyecto

Tras las líneas anteriores se puede apreciar la importancia que tienen los datos y es por ello que se realiza el presente trabajo. Una vez se tiene acceso a ellos, bien por mecanismos propios de adquisición, o bien por obtenerlos a partir de un tercero, es necesario tratarlos y adaptarlos para hacer posible la aplicación de las técnicas conocidas que nos pueden conducir a resolver el problema afrontado.

En concreto, el objetivo de los problemas que se van a tratar en este texto será la asignación de una clase, o el cálculo de alguna magnitud, a las distintas unidades de información de las que se dispone, haciendo uso para ello de los datos conocidos de cada unidad.

Un ejemplo en el ámbito universitario podría ser la clasificación de los alumnos en una de las siguientes clases: aprobar el examen final de una asignatura al primer intento, o no superarlo. Con este objetivo se podría usar información como la edad, sexo, centros previos de estudio, historial de asignaturas cursadas con las respectivas notas obtenidas, situación laboral, créditos restantes y superados, etc. Otro ejemplo podría ser el uso de los indicadores obtenidos tras una analítica médica, incluyendo incluso la evolución de estos en el tiempo, para asignar a un paciente un porcentaje de predisposición de padecer una enfermedad.

En el primer caso, una unidad de información, que a partir de ahora se llamará muestra u observación, es cada alumno, mientras que en el segundo ejemplo es cada paciente. Cabe destacar que en el primer problema se le asigna a cada muestra una de las dos clases, mientras que en el segundo se está estimando el valor de una variable continua. Cada uno de los datos que se conoce de una muestra pasará a usarse como variable del problema y se hará uso de la palabra característica o atributo para hacer referencia a ella.

De este modo, los repositorios de datos están formados principalmente por variables estadísticas, las cuales adoptan un valor para cada una de las muestras, individuos o elementos. Además, puede darse el caso de que también se disponga del registro de estas variables en diferentes instantes, o su evolución durante un intervalo de tiempo, dando lugar a un histórico de datos.

A continuación. se desarrolla la doble clasificación de las variables [10] [11]. Para mayor claridad, en la figura 1.1 se representa de forma esquemática junto a varios ejemplos.

1. Primera clasificación:

- ***Variables categóricas:*** Son variables que se consideran clasificadoras o calificadoras, ya que catalogan los elementos en grupos, conjuntos, clases o categorías. Se conocen como *dicotómicas* aquellas que solo pueden adoptar dos valores diferentes, y como *politómicas* las que poseen un número de valores mayor a dos. Concretamente, el número de valores que puede adoptar una variable categórica se conoce como cardinalidad. A su vez, se subdividen en:

  - *Nominales:* Entre sus valores no se puede establecer un orden o jerarquía.
  - *Ordinales:* Entre sus valores sí se puede establecer un orden o jerarquía, pero no una magnitud de diferencia entre ellos. Es decir, se emplea una escala ordinal.
  - *Discretas o de conteo:* Son el resultado de contar el número de veces que ocurre un suceso y, por tanto, las categorías están representadas por números enteros.
- ***Variables continuas:*** Son aquellas cuyos posibles valores están recogidos en una escala métrica continua de medición, indicando así cuantas veces contiene la unidad básica de medida de forma entera y fraccionada.

2. Segunda clasificación:

- ***Variables cualitativas:*** A este grupo pertenecen las variables que describen características que pueden presentarse en los elementos que conforman el conjunto de datos. La frecuencia es la única propiedad que se puede aportar directamente a partir de estas. Concretamente, agrupa las variables categóricas nominales y ordinales.
- ***Variables cuantitativas:*** Estas son las variables que se expresan de forma numérica y que, por lo general, están asociadas a una magnitud y unidades concretas. Además de la frecuencia, es posible el cálculo de una amplia gama de propiedades matemáticas. Engloba las variables continuas y las categóricas discretas.

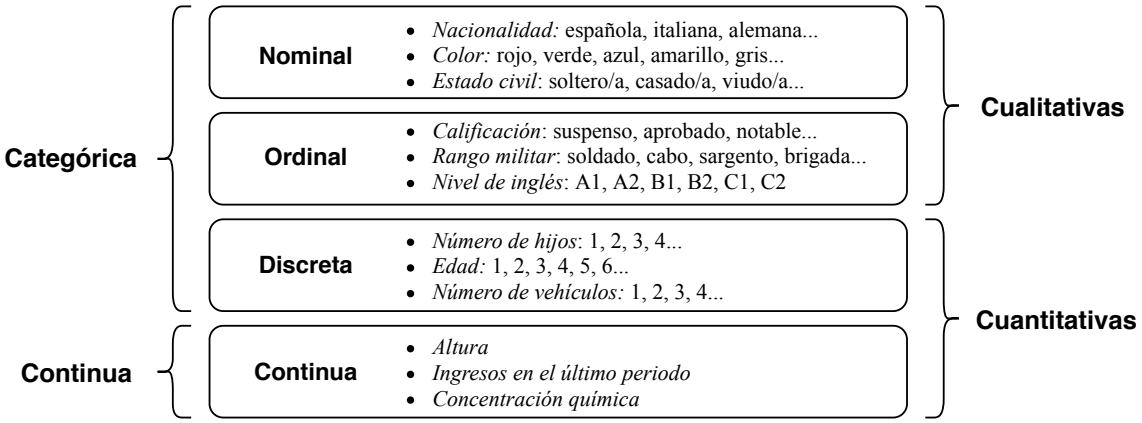

Figura 1.1 Clasificación de las variables.

El hecho de que una variable pertenezca a un tipo u otro depende tanto de la naturaleza de esta como de la forma en la que se expresa. Por ejemplo, la acidez de una disolución se puede expresar en función del pH, es decir, un valor perteneciente a una escala continua y, por tanto, sería una variable continua. Sin embargo, también se podría expresar según si la disolución es básica, neutra o ácida, por lo que sería una variable categórica ordinal.

Otro ejemplo se puede extraer de la medida de la masa de una población. Lo común sería expresarla en kilogramos, por lo que de nuevo estaríamos antes una variable continua. No obstante, si se usa una balanza con precisión de decigramos y expresamos los resultados en estas mismas unidades se podría considerar como una variable discreta, ya que conocemos el incremento mínimo que se produce entre dos valores consecutivos y nunca se obtendría uno nuevo entre ellos dos.

Del último ejemplo conviene destacar un hecho importante, si bien es cierto que aumentando o disminuyendo la precisión de la medida se puede obtener una variable continua o discreta, respectivamente, la variable continua con mayor precisión es capaz de condensar y manifestar más información que la discreta. Por este motivo, muchos textos indican que las variables continuas poseen más información, puesto que su transformación a variable categórica de cualquier tipo, la cual es unidireccional, siempre conlleva pérdidas.

Esto no quiere decir que las variables categóricas no sean importantes o que la información que condensan no sea relevante para conseguir alcanzar el objetivo del problema planteado. Sin embargo, el aprovechamiento de estas muchas veces es limitado debido a que la mayor parte de las técnicas y algoritmos desarrollados a lo largo de los años, para generar conocimiento y utilizarlo, se basan en cálculos numéricos y propiedades matemáticas que no se pueden extrapolar directamente a las variables cualitativas.

Por este motivo, el presente trabajo tiene como objetivo establecer métodos para aprovechar mejor la información que contienen las variables categóricas. No será desarrollando procedimientos que las usen directamente, sino mediante técnicas de codificación que permitan obtener, a partir de ellas, variables continuas o discretas con mayor significado, es decir, variables cuantitativas. De esta forma, será posible usar todos los algoritmos que se conocen para este tipo de variables.

Si por ejemplo existiese una variable llamada clase social con tres posibles valores: baja, media y alta; la transformación consistiría en obtener una o más variables nuevas que tomarían valores numéricos para cada uno de los valores anteriores. Otro caso podría ser la variable día del mes, que aun siendo una variable cualitativa discreta que toma valores enteros comprendidos en el rango $[1,31]$ y que, por tanto, sí se le podría aplicar las técnicas conocidas, sus valores numéricos son realmente etiquetas y nada indica que estas sean las más adecuadas para extraer toda la información que condensa la variable. Por este motivo, también es una posible candidata para aplicarle una transformación similar a la del ejemplo previo.

Para ilustrar los ejemplos previos se incluyen las tablas 1.1 y 1.2, en las que se han utilizado valores arbitrarios para simular la codificación de las variables propuestas. Para la clase social la transformación ha dado lugar a dos variables nuevas, y para el día del mes solo a una.

Tabla 1.1 Ejemplo ficticio para la codificación de la variable *clase social*.

Variable de codificación	**Categorías**		
	Baja	Media	Alta
$C_{ClaseSocial,1}$	0.877	0.455	0.261
$C_{ClaseSocial,2}$	-0.683	0.038	-0.248

Tabla 1.2 Ejemplo ficticio para la codificación de la variable *día de la semana*.

Variable de codificación	**Categorías**						
	1	2	3	4	...	30	31
$C_{DiaSemana}$	0.627	0.960	0.811	-0.936	...	-0.129	0.362

1.2 Estado del arte

Numerosas aproximaciones se han desarrollado en torno a la cuestión planteada en las líneas anteriores. La de aplicación más directa y a la que se suele recurrir en las primeras iteraciones del proceso de resolución de los problemas enfrentados es la de utilizar las variables categóricas casi tal y como aparecen en los repositorios de datos. Es decir, para las discretas se usa su representación numérica, incluso en los casos que sean solamente etiquetas; los valores de las variables ordinales se sustituyen por sus correspondientes cardinales; y en el caso de las nominales, cada categoría es sustituida de forma aleatoria, o como el autor crea más conveniente, por un número entero [12]. De esta forma todas las variables serían cuantitativas.

Otra aproximación sería el uso de algoritmos basados en lógica. Entre ellos destacan los árboles de decisión, de los cuales Murthy [13] ofrece una descripción de sus características, métodos de construcción automáticos, aplicaciones e inconvenientes que pueden surgir durante su uso.

A grandes rasgos, un árbol de decisión es un método de representación de reglas que son aplicadas de forma jerárquica y secuencial con el objetivo de fraccionar el conjunto de datos. Se componen de nodos, los cuales representan variables; y de ramas, que unen los anteriores y están asociadas a los valores que puede tomar la variable de su nodo de origen, figura 1.2. Durante la clasificación, cada muestra comienza por el nodo de cabecera y desciende por el árbol tomando las ramas correspondientes al valor de sus variables hasta que acaba en un nodo terminal, el cual asignará una clase a dicha muestra y, por tanto, quedará clasificada. Este método presenta problemas cuando las variables poseen gran cardinalidad, debido al incremento en el número de ramas; y también cuando existen variables continuas, ya que habría que establecer umbrales durante su discretización para poder incluirlas en el árbol.

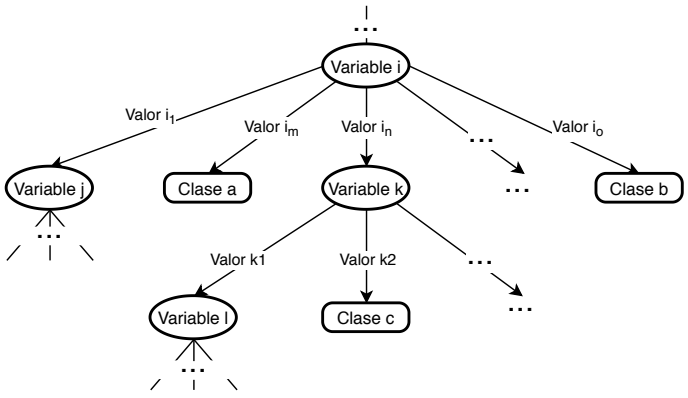

Figura 1.2 Fragmento de un árbol de decisión.

Otro tipo de algoritmo basado en lógica consiste en los conjuntos de reglas de aprendizaje. El procedimiento consiste en la elaboración de grupos disjuntos de reglas que aplicadas sobre las variables de una muestra ofrecen como resultado una clase. Estos grupos pueden ser construidos a partir de un árbol de decisión [14] o ser extraídos directamente de los datos [15]. De nuevo, nos enfrentamos al problema de usar variables continuas en el conjunto de reglas y, a pesar de que con pequeñas modificaciones pueden ser empleadas, son varios los autores que defienden un mejor rendimiento cuando se discretizan [16].

Existe otro tipo de algoritmos importantes en la minería de datos y descubrimiento de conocimiento diseñados para realizar clasificaciones, *clustering* o detección de *outliers*, entre otras aplicaciones, que se basan en el concepto de similitud entre muestras del conjunto de datos mediante el cálculo de las distancias entre ellas. Ejemplos de estas técnicas podrían ser los algoritmos de aprendizaje supervisado conocidos como el método de los k vecinos más cercanos, k-NN [17], y también las máquinas de vectores de soporte, SVMs [18].

Mientras que en las técnicas mencionadas calcular las distancias entre muestras resulta sencillo cuando las variables son continuas: distancia euclídea, distancia de Mahalanobis o distancia de Manhattan, entre muchas otras; no ocurre lo mismo para las variables categóricas. Es por ello que muchos autores proponen sustituir el cálculo de distancias por funciones de cálculo de similitud entre los valores de una variable categórica. De esta forma, se hace posible la aplicación directa de estas técnicas que se fundamentan en la medida de distancias. En [19, 20] se puede consultar una amplia colección de funciones de similitud junto a sus rendimientos en diversas aplicaciones. Por otro lado, también es posible usar funciones de distancia heterogéneas que admiten tanto variables categóricas como continuas [21].

De entre las funciones de medida de similitud destaca la conocida como *overlap* [22] por su sencillez de uso. Cuando compara dos muestras, esta función asigna un 1 por cada variable cuyos valores coinciden en ambas muestras y un 0 por cada una en las que hay discrepancia. Por tanto, la similitud entre ambas muestras resulta proporcional al número de variables categóricas cuyos valores coinciden. Evidentemente, cuando las funciones devuelven una similitud y no una distancia es necesario realizar una conversión mediante, por ejemplo, la expresión (1.1) [19].

$$sim = \frac{1}{1 + dist} \tag{1.1}$$

Existen otras técnicas de cálculo de similitud entre valores cualitativos que se basan en la representación escrita de las categorías. Por ejemplo, si aplicásemos dichas técnicas a una variable que recoge el lugar de nacimiento, los valores *"Jubrique"* y *"Ubrique"* tendrían mayor similitud entre ellas que con *"Olvera"*. Un estudio de este recurso lo podemos encontrar en la obra de Gomma y Fahmy [23], donde recopilan un buen número de métodos para calcular la similitud descrita.

Es cierto que estas técnicas se aproximan más al procesamiento del lenguaje natural que a la codificación de variables categóricas, ya que la similitud entre las expresiones escritas de los valores de una variable no aporta nada en la mayoría de los casos. No obstante, sí tienen una importante aplicación en el tratamiento inicial de los datos. En concreto, ayudan a disminuir la cardinalidad de las variables categóricas cuando en un conjunto de datos una variable adopta diferentes representaciones para la misma categoría [24]. De esta forma, si se considera una variable para la ocupación laboral, los valores *"administración"*, *"administrativo"* y *"admón."* pueden ser agrupados bajo la misma categoría.

Existen otros métodos basados en *hashing* [25] o *clustering* [26] para reducir la cardinalidad y, por tanto, facilitar la codificación posterior haciendo que sean necesarios menos recursos computacionales e incluso mejorando los resultados.

Por último, aproximaciones centradas en el tema principal de este trabajo, es decir, la codificación numérica de las categorías de una variable, serán tratadas en apartados posteriores de este texto junto a los resultados de su aplicación en problemas reales.

1.3 Contenido del trabajo

Tras la presentación que se ha realizado de las variables categóricas, los tipos que existen y el tratamiento habitual que reciben, el contenido restante del trabajo se estructura de la siguiente forma.

En el capítulo 2 se detallan más de 10 métodos de codificación, incluyendo la descripción matemática de estos y varios ejemplos. En el capítulo 3 se presentan las técnicas empleadas para resolver los problemas con los que se compararán posteriormente las codificaciones, en concreto, el método de los k vecinos más cercanos para clasificación y una red neuronal de base radial para regresión. En el capítulo 4 se estudian los resultados de las codificaciones en dos problemas con datos reales: la predicción del nivel económico de una población, y la estimación del volumen de llamadas en un centro de atención al cliente. En el capítulo 5 se recogen unas breves conclusiones sobre el trabajo.

También se incluye un anexo a modo de librería de funciones con la implementación en MATLAB® de los métodos de codificación empleados.

2 Codificación de variables categóricas

A lo largo de este capítulo se va a desarrollar una recopilación de métodos para llevar a cabo la codificación de variables categóricas. Se realizará una descripción de sus fundamentos y, en un capítulo posterior, se valorarán los resultados de la aplicación de los métodos sobre datos de problemas reales.

Antes de empezar, se va a definir una serie de conceptos comunes a todas las codificaciones que ayudarán a formular y precisar cada una de ellas. El conjunto de muestras u observaciones que conforman los datos del problema a resolver se define como $\mathscr{D} = \{\mathbf{X}^k, \mathbf{Y}^k\}_{k=1...d}$, donde sus dos elementos son los vectores:

$$\begin{aligned} \mathbf{X}^k &= \left[X_1^k, X_2^k, \ldots, X_i^k, \ldots, X_m^k\right] \\ \mathbf{Y}^k &= \left[Y_1^k, Y_2^k, \ldots, Y_j^k, \ldots, Y_n^k\right] \end{aligned} \tag{2.1}$$

Donde:

- k : Hace referencia a la muestra k-ésima del conjunto de datos.
- d : Número total de muestras que conforman el conjunto de datos.
- $\mathbf{X}^k$: Vector de características o atributos de la muestra k-ésima.
- m : Dimensión del vector de características. Es un valor constante para todas las muestras.
- X_i^k : Elemento i-ésimo del vector de características de la muestra k-ésima. Si esta variable es continua, puede adoptar cualquier valor del conjunto de los números reales, $\mathbb{R}$. No obstante, si la variable es categórica, sus posibles valores pertenecen al conjunto $\mathscr{X}_i$ de tamaño p_i. El elemento z-ésimo de este conjunto se denota mediante la letra minúscula ${}_z x_i$.
- $\mathbf{Y}^k$: Vector de salida de la muestra k-ésima.
- n : Dimensión del vector de salida. Es un valor constante para todas las muestras.
- Y_j^k : Elemento j-ésimo del vector de salida de la muestra k-ésima. Si esta variable es continua puede adoptar cualquier valor del conjunto de los números reales, $\mathbb{R}$. No obstante, si la variable es categórica, sus posibles valores pertenecen al conjunto $\mathscr{Y}_j$ de tamaño q_j. El elemento s-ésimo de este conjunto se denota mediante la letra minúscula ${}_s y_j$.

Las codificaciones de las variables se van a definir como una aplicación $\Phi : \mathscr{X}_i \to \mathbb{R}^{a_i}$, donde a_i representa la dimensión resultante y, por tanto, el número de variables nuevas por cada original codificada. Este valor dependerá de la transformación realizada. De esta forma, la codificación de la categoría z-ésima de la variable i-ésima queda como $\Phi\left({}_z x_i\right) = {}_z\Phi_i = \left[{}_z\phi_{i,1},\ {}_z\phi_{i,2},\ \ldots,\ {}_z\phi_{i,t},\ \ldots,\ {}_z\phi_{i,a_i}\right]$, siendo ${}_z\phi_{i,t}$ un elemento genérico del vector. Evidentemente, la misma aplicación que define la codificación se puede emplear con las variables del vector de salida $\mathbf{Y}$.

En la situaciones en las que algún índice de la notación matemática no sea necesario será omitido. Esto ocurre cuando, por ejemplo, el vector de salida o el de entrada tienen un único elemento, $Y_1^k = Y^k$ y $X_1^k = X^k$, respectivamente.

Algunas codificaciones requieren el conocimiento de las variables de salida para poder ser aplicadas y, como se describirá en líneas posteriores, hacen uso de todas las observaciones de las que se tiene esta información. No obstante, es recomendable dividir al menos en dos partes el conjunto de muestras con salida conocida: uno de entrenamiento para calcular tanto la codificación como el modelo predictivo, y otro de validación para posteriormente comprobar la aptitud de los resultados.

Para concluir, cabe mencionar que en la mayoría de problemas el vector de salida está formado por un único elemento, $n = 1$, y podemos distinguir dos situaciones según su naturaleza. Por un lado, estaríamos ante un problema de *regresión* si este elemento fuese una variable continua, por ejemplo: estimación de las ventas de un ejercicio económico, predicción del volumen de tráfico, etc. Por otro lado, se trataría de un problema de *clasificación* si la variable fuese categórica, ya que el objetivo consistiría en determinar a qué clase o categoría pertenece cada observación. Algunos casos de ejemplo podrían ser: determinar si una persona es afín a adquirir un producto, catalogar la música en géneros musicales o determinar si una imagen es apta para todos los públicos.

A continuación, se procede con la exposición de las técnicas para llevar a cabo la codificación.

2.1 Codificación por etiquetado

Se va a presentar brevemente esta codificación ya que, aunque no es demasiado rigurosa, es la que se emplea normalmente en las primeras iteraciones durante la resolución del problema [12, 27]. Esto se debe principalmente a que resulta intuitiva, es de las ideas iniciales que suelen surgir y, además, es sencilla de implementar.

El tratamiento que hace de las variables categóricas depende en esencia de la naturaleza de estas. Por un lado, cuando se trata de una variable discreta, hace uso de su misma etiqueta numérica para representar cada categoría. Así, por ejemplo, las variables número de vehículos propios o puntos restantes del permiso de conducir serían codificadas de esta manera, haciendo uso en el modelo predictivo de los propios números naturales que representan las categorías.

Por otro lado, cuando la variable es ordinal, la codificación consiste en la traducción de ese orden en una enumeración, donde cada categoría es asociada con un número natural respetando el orden implícito existente entre categorías. Normalmente se comienza por el 0 o 1, y se continúa con el resto de números consecutivos. Por ejemplo, los niveles de dolor leve, moderado y fuerte se podrían asociar con los números 1, 2 y 3, respectivamente.

Por último, si la variable es nominal, también se realiza una enumeración similar, aunque en esta ocasión será de forma prácticamente aleatoria debido a la ausencia de una pauta. Así, para la variable color, la asociación de amarillo con 1, azul con 2 y rojo con 3, es totalmente arbitraria y resulta en una posible codificación de dicha variable.

En resumen, en todos los casos se realiza una aplicación biyectiva $\Phi : \mathscr{X}_i \to \mathbb{N}$ que consiste en una enumeración del conjunto de categorías posibles.

Los resultados obtenidos con este método se pueden mejorar mediante el uso de la razón y un análisis detallado del conjunto de datos y sus variables categóricas. De esta forma, mediante un proceso de prueba y error, se persigue modificar las asignaciones previas hasta conseguir dicho fin.

Un hipotético ejemplo podría ser la variable día de la semana en un problema de predicción de la demanda eléctrica. En vez de relacionar los días con los números del 1 al 7, se podrían asociar de distinta forma teniendo en cuenta la variabilidad de la actividad humana a lo largo de la semana: de lunes a jueves se podría identificar con el 1, los viernes y sábado con el 4, y los domingos con el 3. Esto podría dar lugar a un mejor desempeño.

Sin embargo, hay que tener en cuenta que esta tarea puede resultar no ser sencilla y que, además, no es seguro que se logre mejorar el desempeño. A esto hay que añadir que, a pesar del tiempo que es posible que conlleve conseguirlo, la codificación final no es extrapolable a otros problemas, donde habría que repetir el proceso de análisis y prueba.

2.2 Codificación one-hot

Esta codificación, también conocida como variables *dummy*, es una de las más empleadas tanto en la resolución de problemas como en la investigación. Esto se debe a su fácil integración y a los buenos resultados que normalmente produce cuando las categorías de la variable que se codifica son mutuamente excluyentes. En obras como la de Cohen et al. [28] podemos encontrar un exhaustivo análisis estadístico de este método, y en otros muchos artículos científicos hacen uso de él para resolver problemas reales, por ejemplo, sobre la predicción de la demanda de bicicletas en una empresa de transporte [29].

La base del método es sencilla. En primer lugar, a partir de una variable categórica X_i se crean p_i nuevas variables ϕ_i, y cada una de estas nuevas se asocia de forma individual a un elemento del conjunto $\mathscr{X}_i$. En segundo lugar, todas las variables de codificación toman el valor 0, no obstante, en cada muestra k, la variable ${}_z\phi_{i,t}$ asociada al valor que toma la variable original, ${}_zx_i^k$, es igual a 1.

A modo de ejemplo, se recoge en la tabla 2.1 el resultado de la codificación de la variable ficticia color, la cual puede adoptar cinco valores diferentes: azul, rojo, verde, blanco y negro.

Algunos autores proponen usar solo $p_i - 1$ variables en la codificación para evitar que aparezca redundancia[1] y mejorar algunas propiedades estadísticas [28]. Para conseguirlo, a una de las categorías se le asigna el vector nulo sin ninguna variable igual a 1. No obstante, con esta práctica se pierde una característica importante de la codificación que, por lo general, es conveniente para las técnicas de aprendizaje automático que se basan en el cálculo de distancias: produce un espacio

[1] En algunos modelos predictivos es necesario que no existan combinaciones lineales en la entrada para asegurar un buen desempeño.

Tabla 2.1 Ejemplo de codificación one-hot.

Vector de codificación	Categorías				
	Azul	Rojo	Verde	Blanco	Negro
${}_{Azul}\Phi_{Color}$	1	0	0	0	0
${}_{Rojo}\Phi_{Color}$	0	1	0	0	0
${}_{Verde}\Phi_{Color}$	0	0	1	0	0
${}_{Blanco}\Phi_{Color}$	0	0	0	1	0
${}_{Negro}\Phi_{Color}$	0	0	0	0	1

ortogonal donde cada categoría de un atributo se asocia con una dimensión, provocando que todas las categorías sean equidistantes con una distancia igual a $\sqrt{2}$ entre cualquier par de valores. En la figura 2.1 se representa esta propiedad, para ello, se simula una variable con solo tres categorías y se representan los tres vectores que produce la codificación junto a la distancia entre ellos.

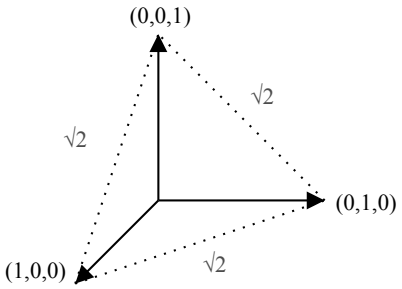

Figura 2.1 Espacio ortogonal generado por la codificación one-hot.

Un caso excepcional aparece cuando se aplica la codificación *one-hot* a una variable categórica de cardinalidad 2. Si se realiza tal y como se ha expuesto, daría lugar a dos variables nuevas, lo cual no es necesario. Con una sola variable, que toma el valor 0 para una categoría y 1 para la otra, se cumpliría la equidistancia entre ellas.

Esta técnica no está exenta de inconvenientes, siendo el más evidente el número de variables nuevas que aparecen. Esto tiene implicaciones directas en la cantidad de memoria necesaria para almacenar toda la información del conjunto de datos, problemática que se acentúa cuando crece la cifra de variables categóricas de gran cardinalidad. No obstante, con las capacidades de almacenamiento de los ordenadores de hoy en día no supone un gran problema. Donde sí afecta realmente es en el número de cálculos que los algoritmos posteriores tendrán que realizar sobre los datos, lo que se traduce directamente en mayor tiempo de ejecución. Por ejemplo, si se trabaja con un conjunto $\mathscr{D}$ de 10 atributos donde 3 son categóricos con cardinalidad 5, 6 y 3; el tiempo requerido será evidentemente mayor cuando se emplea un espacio de dimensión 21 en vez de 10.

Para paliar este efecto se pueden usar matrices dispersas. Estas no se almacenan de la forma tradicional, sino que guardan los pares valor y posición de los elementos diferentes a 0, asumiendo que los restantes son todos nulos. Así es posible reducir el tamaño de los datos cuando existe un gran número de ceros, y además, bajo determinadas condiciones, agilizar los cálculos [30].

No obstante, es más frecuente emplear otro tipo de recursos para realmente reducir la dimensionalidad del problema resultante. Por ejemplo, se podría realizar un análisis de las categorías de $\mathscr{X}_i$ con el objetivo de encontrar aquellas con menor frecuencia de aparición, posteriormente, agrupar estas bajo una misma categoría codificada en una única variable $\phi_{i,t}$. Así se consigue reducir el número de variables afectando a un número reducido de muestras. Esta agrupación también se podría realizar unificando categorías similares bajo una única que sea representativa de las que agrupa, bien de forma manual por inspección y estudio de los datos, o bien recurriendo a medidas de similitud entre categorías [19] o mediante técnicas de *clustering*. Otra opción sería aplicar procedimientos para reducir la dimensionalidad una vez se han codificado las variables: análisis de componentes principales (PCA), análisis discriminante lineal (LDA) o análisis de la correlación canónica (CCA).

El otro inconveniente de este método se presenta cuando aparecen nuevos valores de una variable categórica. Es posible que durante las fases de extracción de conocimiento y generación de modelos se use un conjunto de datos que no represente toda la cardinalidad de las variables categóricas, lo que puede provocar que cuando se apliquen los modelos obtenidos sobre futuras muestras aparezcan nuevas categorías que no tengan cabida en ninguna variable de codificación ϕ. Como consecuencia, puede que sea necesario volver a pasar por las primeras etapas y procesar de nuevo los datos para poder incluir dicha categoría. Para evitar esta situación, se puede prever una variable cuyo valor sea 1 para todas las categorías nuevas que no habían sido consideradas, lo cual ayuda a atajar el problema ahorrando tiempo, a pesar de no ser lo óptimo.

2.2.1 Codificación one-cold

Existe un método muy similar al anterior llamado *one-cold*. La diferencia reside en que actúa de forma inversa: todas las variables son 1 excepto la asociada al valor que toma la variable categórica en cada muestra. El efecto que consigue es el mismo, dos categorías diferentes vuelven a estar separadas una distancia de $\sqrt{2}$, por lo que los resultados que produce deberían ser similares; aunque es posible que aparezcan discrepancias por la etapa de normalización de datos que se aplica siempre durante el tratamiento de estos.

Todas las particularidades de *one-hot* descritas en las líneas anteriores son también de aplicación en esta situación, sin embargo, por motivos obvios, el uso de matrices dispersas no resulta ahora de gran utilidad.

A modo de ejemplo, se realiza en la tabla 2.2 la codificación *one-cold* de la misma variable color de la tabla 2.1.

Tabla 2.2 Ejemplo de codificación one-cold.

Vector de codificación	Categorías				
	Azul	Rojo	Verde	Blanco	Negro
${}_{Azul}\Phi_{Color}$	0	1	1	1	1
${}_{Rojo}\Phi_{Color}$	1	0	1	1	1
${}_{Verde}\Phi_{Color}$	1	1	0	1	1
${}_{Blanco}\Phi_{Color}$	1	1	1	0	1
${}_{Negro}\Phi_{Color}$	1	1	1	1	0

2.2.2 Codificación rank-hot

En las dos codificaciones anteriores se está eliminando por completo cualquier representación implícita de una ordenación entre categorías. Sin embargo, la codificación *rank-hot* sí refleja un orden entre los valores que puede adoptar la variable categórica, lo cual puede ser beneficioso en caso de ser ordinal [31].

Para ello, se usa un vector de codificación de tamaño igual a la cardinalidad de la variable categórica menos uno. Luego, a cada categoría, se le asigna un número entero para establecer la ordenación, se debe empezar por cero y usar números consecutivos. Posteriormente, para codificar una categoría, se igualan a 1 las variables del vector de codificación cuya posición sea igual o menor al entero asociado a dicha categoría. Al resto de variables que no cumplen esta condición se les asigna un 0.

Con *rank-hot* se consigue que la distancia entre dos muestras dependa de la categoría que adopta cada una. Si son iguales, será 0, y sin son diferentes, la distancia será mayor mientras más separadas estén las categorías en la ordenación establecida.

De nuevo, se realiza con este método la codificación de la variable color, tabla 2.3. Para el ejemplo, se ha realizado una ordenación totalmente aleatoria de los colores siguiendo la disposición que aparece en la tabla.

Tabla 2.3 Ejemplo de codificación rank-hot.

Variable de codificación	Categorías				
	Azul	Rojo	Verde	Blanco	Negro
$\phi_{Color,1}$	0	1	1	1	1
$\phi_{Color,2}$	0	0	1	1	1
$\phi_{Color,3}$	0	0	0	1	1
$\phi_{Color,4}$	0	0	0	0	1

2.3 Codificación binaria

El método de codificación binaria guarda ciertas similitudes con el presentado previamente. En concreto, las variables de codificación que genera también toman solo el valor uno o cero. Además, el valor a_i, es decir, el tamaño del vector de codificación, también es mayor a 1; aunque como veremos, crece de forma más moderada con respecto a la cardinalidad.

Para aplicar esta codificación sobre una variable categórica genérica, X_i, es necesario primero establecer un orden entre las categorías que puede adoptar. La ordenación se puede realizar aplicando cualquier criterio y, según el problema, tendrá efectos diferentes o incluso arbitrarios: por frecuencia absoluta de cada categoría, por orden alfabético, por orden de aparición, por alguna relación con la variable de salida como la probabilidad condicional, de forma aleatoria, etc. Una vez se tienen las categorías ordenadas, se les asigna un número entero siguiendo el orden obtenido y empezando por el 0. Por último, esta asignación se transforma a base binaria y cada uno de los dígitos pasará a ser una variable de codificación $\phi_{i,t}$.

Para mayor claridad se añade un ejemplo en la tabla 2.4, donde se recoge la codificación binaria de la variable categórica estado civil, la cual puede ser: soltero/a, casado/a, pareja de hecho, separado/a, divorciado/a o viudo/a. Las categorías se han ordenado de forma totalmente arbitraria siguiendo el mismo orden en el que se han enumerado.

Tabla 2.4 Ejemplo de codificación binaria.

Variable de codificación	**Categorías**					
	Soltero/a	Casado/a	Pareja de hecho	Separado/a	Divorciado/a	Viudo/a
$\phi_{EstadoCivil,1}$	0	0	0	0	1	1
$\phi_{EstadoCivil,2}$	0	0	1	1	0	0
$\phi_{EstadoCivil,3}$	0	1	0	1	0	1

El número de variables nuevas requeridas para la codificación de X_i, que también es dependiente de la cardinalidad de la variable que se está tratando, es igual al número menor de bits necesarios para representar de forma individual todas las categorías. Este valor se puede obtener mediante:

$$a_i = \text{mínimo}\,\{\mathbb{N} \mid \geq \log_2(p_i)\} \tag{2.2}$$

La ampliación de la dimensión del espacio de características es menor que con la codificación previa gracias a la presencia del logaritmo. Este hecho queda reflejado en la figura 2.2, donde se representa la dimensión del vector Φ_i en función de la cardinalidad de la variable codificada, quedando en evidencia el beneficio de la evolución logarítmica frente a la lineal.

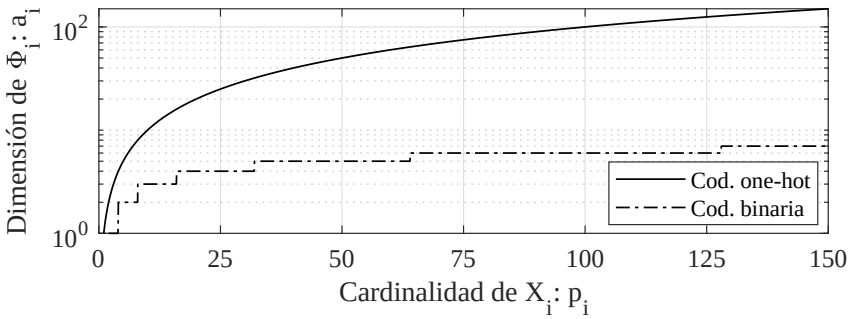

Figura 2.2 Comparación de la codificación binaria y *one-hot* respecto al número de variables nuevas.

2.3.1 Código Gray

Cuando se aplica la transformación a binario es muy posible que no se aproveche el potencial de la ordenación que se realiza previamente. Por ello, puede ser interesante pasar a base 2 haciendo uso del código Gray, ya que este tiene la ventaja de que dos números consecutivos solo se diferencian en un dígito.

Para pasar de binario a Gray tan solo es necesario realizar una operación XOR desechando el acarreo del número a convertir consigo mismo desplazado un bit a la derecha [32]. En la tabla 2.5 se recoge la transformación para 4 bits.

Tabla 2.5 Correspondencia entre código binario y Gray.

Base 10	Binario	Gray	Base 10	Binario	Gray
0	000	000	4	100	110
1	001	001	5	101	111
2	010	011	6	110	101
3	011	010	7	111	100

2.4 Codificación mediante entity embeddings

La técnica de codificación recogida en esta sección se ha extraído del artículo *Entity Embeddings of Categorical Variables* de los autores Cheng Guo y Felix Berkhahn [33]. En el documento describen como emplear el método conocido como *entity embeddings*, ampliamente usado en el procesamiento del lenguaje natural (NLP), para llevar a cabo de forma automática la representación de variables categóricas en un espacio multidimensional, de manera que, categorías similares se encuentran próximas entre sí y, al mismo tiempo, alejadas de otras categorías con un menor grado de similitud. Bajo este contexto, la similitud entre categorías es evaluada en función de la relación de estas con la variable de salida.

El hecho de que los valores similares queden próximos entre sí al realizar el mapeo evidencia las relaciones intrínsecas entre categorías, ayudando a las técnicas más comunes de aprendizaje automático a resolver los problemas. Además, al permitir el cálculo de distancias, esta técnica puede ser también empleada para visualizar datos o realizar *clustering*.

La codificación está sumamente ligada a las redes neuronales. En realidad, es presentada como una forma efectiva de poder hacer uso de ellas en problemas que contienen variables categóricas, ya que sus capacidades para hacer frente a estas variables son limitadas por tratarse de modelos continuos. Al mismo tiempo que las redes resuelven el problema, generan la codificación de las variables categórica, la cual puede ser fácilmente extraída y usada en otros tipos de modelos predictivos.

El método toma como base la codificación *one-hot*, la cual genera un espacio de dimensión igual a la cardinalidad de la variable codificada, tal como se describió en la sección 2.2. Partiendo de este punto, la nueva codificación trata de condensar dicho espacio en otro continuo de dimensión más reducida aprovechando las posibles relaciones que existen entre categorías. Tras esto, representa cada valor categórico como un vector en el dominio del espacio condensado.

La idea de los autores para llevar a cabo esta tarea consiste en la modificación de la capa de entrada de la red, tal y como se refleja en la figura 2.3.

Ahora, las variables de entrada están conectadas a la primera capa mediante un codificador *one-hot*. Como se puede observar, la capa está segmentada en grupos independientes y, cada uno de estos, tiene el mismo número de neuronas que la cardinalidad de la variable que trata. La codificación

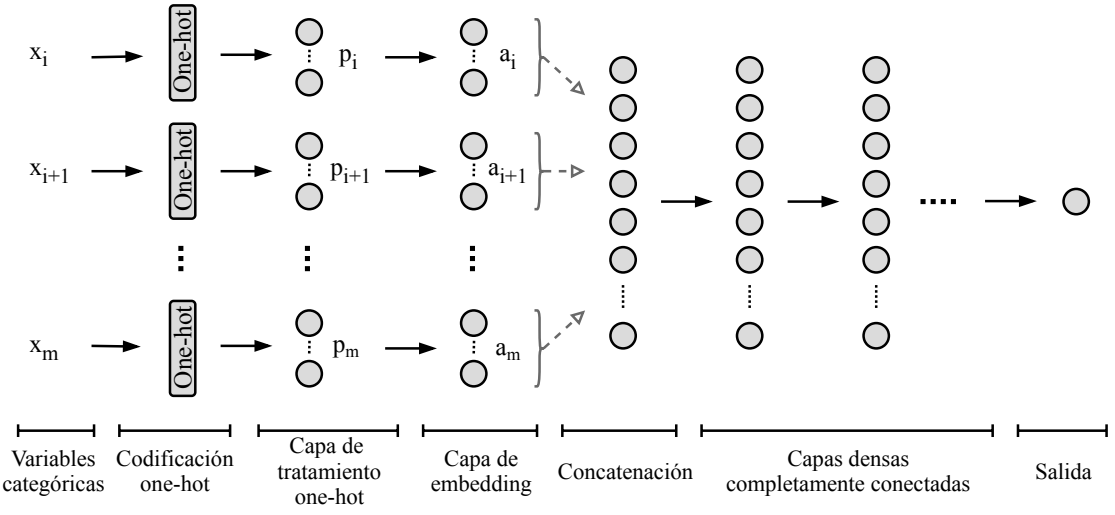

Figura 2.3 Estructura de la red neuronal para el tratamiento y codificación de variables categóricas.

se puede definir matemáticamente como una aplicación $f : X_i \rightarrow \delta_{{}_z x_i\,\alpha}^{p_i}$, donde el resultado es un vector de dimensión igual a la cardinalidad, p_i, y sus componentes son deltas de Kronecker, $\delta_{{}_z x_i\,\alpha}$. El subíndice α es diferente para cada elemento del vector y puede adoptar los mismos valores que X_i, $\alpha \in \mathscr{X}_i$. En definitiva, el resultado es el mismo que el presentado en *one-hot*, por lo que, por cada categoría a la entrada, todas las neuronas de un grupo reciben un cero a excepción de la asociada a dicha categoría, que recibe un 1.

$$\delta_{{}_z x_i\,\alpha} = \begin{cases} 1 & si \quad \alpha = {}_z x_i \\ 0 & si \quad \alpha \neq {}_z x_i \end{cases} \tag{2.3}$$

Tras esta capa que recibe las variables codificadas se añade una nueva formada por neuronas lineales conocida como capa de *embedding*. Al igual que ocurría con la primera, las neuronas de esta capa se encuentran divididas en tantos grupos como variables categóricas. Además, las conexiones de neuronas de ambas capas solo se producen entre las que pertenecen al grupo asociado a la misma variable. En una etapa posterior, todas las neuronas de la capa de *embedding* son concatenadas para unirse con la primera de las capas densas de la red completamente conectada. Si en el problema existiesen también variables continuas, sus correspondientes entradas a la red se podrían añadir en esta concatenación.

Cuando una entrada toma el valor ${}_z x_i$, se puede calcular la salida de las neuronas de la segunda capa pertenecientes al grupo asociado a esa entrada mediante la expresión 2.4. En esta, el término ω_α hace referencia a la matriz de pesos de ponderación de las uniones de la capa de codificación con la capa de *embedding*. Al ser solamente un elemento del vector $\delta_{{}_z x_i\,\alpha}^{p_i}$ igual a 1, el resultado es el vector de pesos que une la neurona de la capa de codificación asociada a la categoría ${}_z x_i$ con las neuronas de la capa de *embedding*.

$${}_z\Phi_i = \sum_\alpha \omega_\alpha \cdot \delta_{{}_z x_i\,\alpha}^{p_i} = \omega_{{}_z x_i} \tag{2.4}$$

Tal como se ha adelanto con la nomenclatura en la expresión 2.4, estos pesos entre las dos capas descritas constituyen la codificación de las categorías. El cálculo de estos es directo mediante la aplicación de las técnicas convencionales de entrenamiento supervisado. De este modo, al mismo tiempo que la red es entrenada y las capas más ocultas aprenden a resolver el problema, la capa de *embedding* entiende y condensa las relaciones implícitas entre categorías, quedando el resultado reflejado en los pesos.

Un hecho muy importante es que el número de neuronas en cada grupo de la capa de *embedding* es un parámetro que se puede ajustar. Como consecuencia, el número de pesos entre ambas capas es variable y, por tanto, el tamaño del vector de codificación ${}_z\Phi_i$ también. Concretamente, su dimensión a_i es igual al tamaño del grupo de neuronas que codifica la variable en cuestión. Para evitar la aparición de dependencias lineales, el número de neuronas está acotado superiormente por la cardinalidad de la variable menos 1, $p_i - 1$.

Una aplicación de este método de codificación se puede encontrar en el artículo de Bo Wang et al., en el cual tratan de predecir el flujo de tráfico en un sistema de bicicletas compartidas ubicado en Sozhou usando solo variables categóricas en una red neuronal [34]. En esta situación, el tamaño que recomiendan para la capa de *embedding* es igual a la raíz cuadrada de la cardinalidad, $\sqrt{p_1}$.

Por último, es importante destacar que para aplicar esta técnica es necesario conocer la salida de un subconjunto representativo de muestras para poder entrenar la red, requisito que no era necesario con *one-hot*. No obstante, posee la ventaja de que el tamaño del vector de codificación es un parámetro independiente de la cardinalidad y, sobre todo, variable según se crea conveniente. Esto permite reducir el tamaño del espacio de características, lo que ahorra memoria y acelera los modelos predictivos.

2.5 Codificación basada en estadísticas de la variable de salida

Este método corresponde con el presentado por Daniele Micci-Barreca en su trabajo *A Preprocessing Scheme for High-Cardinality Categorical Attributes in Classification and Prediction Problems* [35]. El autor desarrolla una técnica sencilla para la codificación de variables categórica que no se ve afectada por variables de gran cardinalidad.

Se basa principalmente en el método de Bayes empírico, de forma que cada categoría de una variable pasa a estar representada por una probabilidad estimada de la variable de salida del problema: cuando se trata de un problema de clasificación, se emplea la probabilidad a posteriori de la salida condicionada por el valor de la variable categórica; cuando el problema es de regresión, se usa el valor numérico esperado de salida condicionado nuevamente por el valor categórico que se está codificando. De esta manera, las variables categóricas se convierten en variables cuasi-continuas, lo que facilita la integración de estas en modelos predictivos.

Barreca también describe una modificación para tratar variables cuyas categorías guardan una estructura jerárquica implícita, sin embargo, no se recoge en este texto por ser solo aplicable en determinados casos concretos.

2.5.1 Aplicación en clasificación con salida dicotómica

Aunque en líneas posteriores se mostrará que la siguiente asunción no es un requisito necesario, se va a asumir que el problema sobre el que se aplica esta técnica reparte las observaciones entre dos clases, es decir, el vector de salida es realmente un único valor dicotómico, $Y^k \in \{{}_1Y = 1, {}_2Y = 0\}$, donde la representación 0 y 1 se ha tomado por comodidad.

Bajo esta situación, el método realiza la codificación de una variable categórica X_i mapeando cada valor de su conjunto $\mathscr{X}_i$ con un escalar. Esto quiere decir que la dimensión del vector de codificación es 1, $a_i = 1$, puesto que solo aparece una variable de codificación por cada variable categórica original del problema, gracias a lo cual, no se altera la dimensión del espacio de características.

El escalar usado para mapear una categoría representa la probabilidad estimada de que la salida sea igual a 1 condicionada a que la variable original adopte el valor categórico a codificar, matemáticamente:

$$ {}_z\Phi_i = P(Y = 1 | X_i = {}_zx_i) \tag{2.5} $$

Es evidente que, para poder calcular esta probabilidad, es necesario disponer previamente de un conjunto de datos ya clasificado. Si el número de muestras con $X_i = {}_zx_i$ para todo ${}_zx_i \in \mathscr{X}$ es suficientemente grande, la probabilidad se puede calcular simplemente como el cociente del número de observaciones con $Y = 1$ y $X_i = {}_zx_i$ entre el número total de observaciones con la categoría ${}_zx_i$:

$$ {}_z\Phi_i = P(Y = 1 | X_i = {}_zx_i) = \frac{f_{(Y=1 \cap X_i = {}_zx_i)}}{f_{(X_i = {}_zx_i)}} \tag{2.6} $$

No obstante, en los conjuntos de datos reales es muy probable que algunas categorías no aparezcan un número suficiente de veces, sobre todo cuando las variables tienen gran cardinalidad, por lo que la probabilidad calculada según la ecuación 2.6 no sería fiable. Por este motivo, el autor propone el cálculo de la probabilidad necesaria en la codificación mediante la suma ponderada de dos probabilidades: la a posteriori, calculada con la misma expresión 2.6; y la a priori, siendo la probabilidad de obtener la salida $Y = 1$ calculada sobre el conjunto completo con independencia de las variables de entrada, hipótesis nula:

$$ {}_z\Phi_i = \Lambda(f_{X_i = {}_zx_i}) \frac{f_{(Y=1 \cap X_i = {}_zx_i)}}{f_{(X_i = {}_zx_i)}} + (1 - \Lambda(f_{X_i = {}_zx_i})) \frac{f_{Y=1}}{d} \tag{2.7} $$

Por la forma en la que está definida la expresión 2.7, parece razonable que la función de ponderación $\Lambda(f)$ sea monótona creciente y, además, una condición que sí debe cumplir es que se encuentre acotada en el rango $[0, 1]$. Así, cuando la cantidad de observaciones con $X_i = {}_zx_i$ sea representativa, el peso de la probabilidad a posteriori será cercano a 1 y el de la a priori será aproximadamente 0, y viceversa. Esto permite usar la misma expresión en la codificación de todas las categorías independientemente del número de veces que aparezcan en el conjunto de datos.

Barreca propone usar para la ponderación la familia de curvas con forma de "S" definida en la ecuación 2.8. El parámetro λ_1 corresponde con el valor de f para el cual se cumple $\Lambda(f) = 0.5$, y el segundo parámetro λ_2 controla la pendiente de la curva en el punto de inflexión. En la figura 2.4 se han representado en diferentes tonos de grises las curvas obtenidas variando λ_2, mientras que la influencia de λ_1 queda reflejada mediante el eje de abscisas al presentar f en relación a dicho parámetro.

$$\Lambda(f) = \frac{1}{1 + e^{-\frac{f-\lambda_1}{\lambda_2}}} \tag{2.8}$$

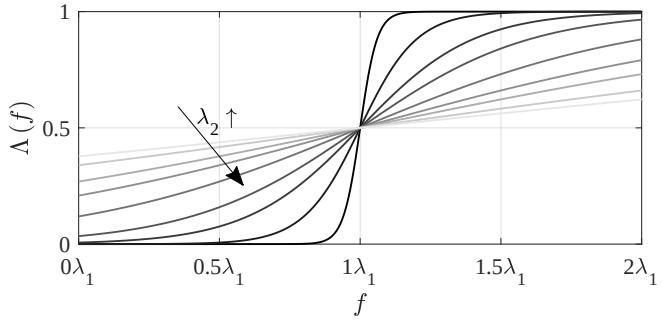

Figura 2.4 Familia de curvas S para la ponderación de las probabilidades.

En esta codificación, cabe destacar el hecho de que ya está de por sí normalizada, lo que puede ser beneficioso en algunos modelos predictivos: los resultados obtenidos por las expresiones 2.6 y 2.7 se encuentran acotados en el rango $[0, 1]$. Por otro lado, cuenta con la ventaja de no presentar problemas cuando trata de codificar una categoría que no ha aparecido previamente, puesto que le será asignada la probabilidad a priori y tratada por el mismo modelo.

2.5.2 Aplicación en clasificación con salida politómica

Cuando la salida del problema es una variable categórica de cardinalidad q mayor a 2 es necesario modificar la forma de proceder. El vector de variables de codificación ${}_z\Phi_i$ ya no estará formado por un único elemento, sino que su tamaño a_i será igual a q. De esta manera, cada variable nueva de codificación está asociada a una categoría de salida y representa la probabilidad estimada de obtener dicha salida condicionada a la categoría de la variable original:

$${}_z\phi_{i,s} = P(Y = {}_sy | X_i = {}_zx_i) \qquad \forall {}_sy \in \mathscr{Y} \tag{2.9}$$

Esta probabilidad puede ser estimada haciendo uso de una expresión similar a la 2.7, que será aplicada una vez por cada valor de la variable de salida:

$${}_z\phi_{i,s} = \Lambda(f_{X_i={}_zx_i}) \frac{f_{(Y={}_sy \cap X_i={}_zx_i)}}{f_{(X_i={}_zx_i)}} + (1 - \Lambda(f_{X_i={}_zx_i})) \frac{f_{Y={}_sy}}{d} \qquad \forall {}_sy \in \mathscr{Y} \tag{2.10}$$

Una ventaja de este método frente a otros que también conllevan la ampliación del espacio de características es que el incremento no está ligado a la cardinalidad de las variables de entrada, sino a la cardinalidad de la de salida, la cual normalmente no suele ser elevada en los problemas de clasificación. Gracias a esto, el número final de variables será menor.

Por ejemplo, en un problema donde el tamaño del conjunto $\mathscr{Y}$ es 4 y el vector de características tiene dimensión 40, siendo 6 de ellas variables categóricas, el espacio de características para el modelo predictivo pasaría a tener 58 variables tras la codificación. Además, como existe una dependencia lineal entre las 4 variables de codificación que aparecen por cada categórica es posible eliminar 1 de ellas sin pérdida de información, por lo que el número final se podría reducir a 52.

2.5.3 Aplicación en regresión

En un problema de regresión, donde la variable de salida es continua, difícilmente se pueden aplicar las ecuaciones previas. Por este motivo, se realiza un cambio en el punto de vista para realizar el mapeo: en vez de estimar probabilidades de la variable de salida, se pasa a realizar estimaciones del valor esperado de la salida.

Esto no produce cambios significativos en la expresión matemática a emplear, donde ahora, la probabilidad a priori pasa a estar representada por la esperanza de la salida, y la a posteriori, por la esperanza de la salida condicionada a la variable categórica:

$$\begin{aligned} {}_z\Phi_i &= \Lambda(f_{X_i={}_zx_i})E(Y|X_i={}_zx_i) + (1-\Lambda(f_{X_i={}_zx_i}))E(Y) = \\ &= \Lambda(f_{X_i={}_zx_i})\frac{\sum\limits_{\forall k:(X_i^k={}_zx_i)} Y^k}{f_{(X_i={}_zx_i)}} + (1-\Lambda(f_{(X_i={}_zx_i)}))\frac{\sum\limits_{k=1}^{d} Y^k}{d} \end{aligned} \tag{2.11}$$

Es fácil comprobar que la ecuación 2.11 produce los mismos resultados que la 2.7 cuando $Y^k \in \{{}_1Y=1, {}_2Y=0\}$.

2.6 Codificación usando optimal scaling

Esta técnica es presentada por Samuel E. Buttrey en su artículo *Nearest-neighbor classification with categorical variables* [36]. En él describe un método que permite mapear las categorías de una variable en la recta de los número reales de una forma óptima. Entendiéndose por óptima como aquella en la que se maximiza el ratio de la suma total de cuadrados entre la suma de cuadrados dentro de una clase, sumando todas las clases. Según el autor, el valor que se maximiza proviene del propio discriminante lineal de Fisher. Aunque en el artículo se propone el uso de la codificación con un clasificador de tipo *k-nn*, el resultado se puede usar en cualquier otro modelo predictivo una vez se tiene la correspondencia numérica de las categorías.

El autor defiende que el número asignado a cada categoría aporta información de cómo de cerca están unas de otras en relación a mejorar la clasificación. Así, cuando las categorías que adoptan dos muestras presentan similitud entre sí, las observaciones quedarán situadas próximas en el eje de dicha variable, lo que ayudará a realizar la clasificación.

Para comenzar la explicación, se va a considerar que el vector de características está compuesto por una única variable categórica, $\mathbf{X} = X_i = X$. Además, por como se ha descrito la técnica, cada categoría es codificada en un único número real, por lo que el tamaño del vector de codificación también es uno, $\Phi({}_z x) = {}_z\Phi = {}_z\phi$.

En primer lugar, se va a asumir sin pérdida de generalidad, que la suma para todas las observaciones del valor numérico de codificación de la variable categórica es igual a 0:

$$\sum_{k=1}^{d} \Phi(x^k) = 0 \tag{2.12}$$

Luego, el valor medio de codificación de un valor concreto de salida se puede calcular como la suma del valor de codificación de todas las muestras con dicha salida entre el número total de muestras con esa salida:

$${}_s{}^y\overline{\phi} = \frac{\sum\limits_{\forall {}_z x \in \mathscr{X}} \Phi({}_z x)\, f_{(X={}_z x \cap Y={}_s y)}}{f_{(Y={}_s y)}} \tag{2.13}$$

Además, el término que se busca maximizar, la suma de cuadrados de los valores de codificación, TSS, entre la suma de cuadrados dentro de una clase adicionando todas las clases, WSS, se calcula mediante las expresiones:

$$TSS = \sum_{k=1}^{d} [\Phi(x^k)]^2 = \sum_{\forall {}_z x \in \mathscr{X}} f_{(X={}_z x)}\, [\Phi({}_z x)]^2 \tag{2.14}$$

$$WSS = \sum_{\forall {}_s y \in \mathscr{Y}} \sum_{\forall {}_z x \in \mathscr{X}} f_{(X={}_z x \cap Y={}_s y)} [\Phi({}_z x) - {}_s{}^y\overline{\phi}]^2 = TSS - \sum_{\forall {}_s y \in \mathscr{Y}} f_{(Y={}_s y)} [{}_s{}^y\overline{\phi}]^2 \tag{2.15}$$

El autor usa también la restricción de $WSS = 1$ a la hora de maximizar el ratio TSS/WSS. De esta forma, consigue que si todos los valores numéricos resultantes de la codificación de las categorías son multiplicados por la misma constante, el ratio permanezca invariable. Llamando θ al vector formado por la codificación de cada una de las categorías, $\theta = [\Phi({}_1 x), \Phi({}_2 x)), ..., \Phi({}_z x), ..., \Phi({}_{p_i} x)]$, el problema puede ser resuelto mediante la expresión:

$$\mathbf{W}\theta - \lambda[\mathbf{W}\theta - \mathbf{U}\theta] = 0 \qquad \rightarrow \qquad \mathbf{U}\theta = \mu\mathbf{W}\theta\ , \quad \mu = \frac{(\lambda - 1)}{\lambda} \tag{2.16}$$

Donde $\mathbf{U}$ es una matriz de dimensión $p\,x\,p$ y $\mathbf{W}$ es otra matriz con el mismo tamaño pero con la peculiaridad de ser diagonal. Ambas se pueden calcular como:

$$\mathbf{W}_{uv} = \begin{cases} f_{(X={}_u x)} & si \quad u = v \\ 0 & si \quad u \neq v \end{cases} \tag{2.17}$$

$$\mathbf{U}_{uv} = \sum_{\forall_s y \in \mathscr{Y}} \frac{f_{(X=_u x \cap Y=_s y)}\, f_{(X=_v x \cap Y=_s y)}}{f_{(Y=_s y)}} \tag{2.18}$$

Si se premultiplica la segunda ecuación de la expresión 2.16 por la matriz cuyo cuadrado es la inversa de $\mathbf{W}$ resulta:

$$\mathbf{W}^{-1/2}\mathbf{U}\mathbf{W}^{-1/2}\left[\mathbf{W}^{1/2}\theta\right] = \mu\left[\mathbf{W}^{1/2}\theta\right] \tag{2.19}$$

Revelando que los vectores $\mathbf{W}^{1/2}\theta$ son los autovectores de la matriz simétrica positiva semidefinida $\mathbf{W}^{-1/2}\mathbf{U}\mathbf{W}^{-1/2}$. Por tanto, la codificación de cada categoría puede ser obtenida resolviendo un problema de autovalores.

El primer autovalor de este problema es siempre 1 y su autovector asociado θ también tiene todos los elementos iguales a 1. A pesar de que este par es solución de la primera ecuación de la expresión 2.16, no lo es de la segunda ecuación, puesto que no existe una solución finita para λ cuando $\mu = 1$. Por tanto, esta solución es ignorada y será el segundo autovalor el correspondiente a la mejor solución, la que maximiza el ratio TSS/WSS. De esta forma, el segundo autovector es el vector θ cuyos elementos representan los valores numéricos buscados para codificar las categorías de la variable x.

Esta codificación presenta un problema cuando, tras tratar los datos y entrenar un modelo predictivo, aparece una observación con una categoría que no estaba recogida en los datos iniciales. La única forma de poder representar la nueva categoría junto a las demás en la misma recta real es realizando todo del proceso de nuevo, es decir, volviendo a calcular la codificación y entrenando un nuevo modelo.

2.6.1 Extensión a problemas con múltiples variables categóricas de entrada

Cuando el vector de entrada tiene más de una variable categórica, $m > 1$, se puede realizar un desarrollo similar al anterior. El resultado conduce a la misma expresión que la expuesta en 2.16, pero la definición de sus elementos es diferente para poder tener en cuenta las categorías existentes entre todas las variables.

En esta ocasión, la matriz $\mathbf{W}$ es una matriz cuadrada por bloques de dimensión igual al número total de categorías teniendo en cuenta todas las variables. El bloque $\mathbf{W}_{UV}$ de dimensión igual a $p_U \, x \, p_V$ se puede calcular como una tabla de tabulación cruzada entre las variables X_U y X_V. La matriz $\mathbf{U}$ también se puede definir por bloques $\mathbf{U}_{UV}$ del mismo tamaño que los anteriores. Se calculan como:

$$[\mathbf{W}_{\underline{UV}}]_{uv} = f_{(X_{\underline{U}}=_u x \cap X_{\underline{V}}=_v x)} \tag{2.20}$$

$$[\mathbf{U}_{\underline{UV}}]_{uv} = \sum_{\forall_s y \in \mathscr{Y}} \frac{f_{(X_{\underline{U}}=_u x \cap Y=_s y)} f_{(X_{\underline{V}}=_v x \cap Y=_s y)}}{f_{(Y=_s y)}} \tag{2.21}$$

De nuevo, la resolución del problema de autovalores de la expresión 2.16 permite obtener el vector θ. El primer autovalor y autovector vuelven a ser $\mu = 1$ y $\theta = \mathbf{1}$, por lo que la solución deseada que maximiza el ratio TSS/WSS corresponde al segundo autovector. Este tiene ahora tantas componentes como categorías hay entre todas las variables y cada elemento del vector es el número a usar para codificar cada categoría.

Una peculiaridad a tener en cuenta es que ninguna de las dos matrices tiene rango completo. Esto puede provocar problemas de inestabilidad en los algoritmos de la resolución del problema de autovalores. Una solución que propone el autor es modificar ambas matrices sumando una cantidad constante y muy pequeña a los elementos de la diagonal principal.

Al codificar cada categoría con un único número real se manifiesta la ventaja de esta codificación. Cada variable categórica es sustituida por una única variable continua, por lo que no se incrementa el tamaño del espacio de características. Aunque a priori parezca que la cardinalidad de las variables no tiene efecto en la codificación, no hay que olvidar que hay que resolver un problema de autovalores cuyo tamaño aumenta proporcionalmente con la cardinalidad total.

2.7 Codificación usando el peso de la evidencia

La expresión peso de la evidencia, más conocida como *weight of evidence (WOE)*, ha sido utilizada a lo largo de los años en muchos contextos diferentes: publicaciones científicas, literatura política y jurídica, evaluación de riesgos, etc; lo que ha ocasionado que no haya consenso en su definición y contenido [37]. Actualmente, el término es ampliamente usado en el campo de la puntuación crediticia y la regresión logística, para llevar a cabo la discretización de variables continuas [38].

El peso de la evidencia se podría ver de forma cuantitativa como la robustez de unos hechos para defender una hipótesis o, lo que es lo mismo, el poder predictivo de una variable independiente sobre la salida. En la obra de I. J. Wod [39] se desarrolla una descripción matemática basada en probabilidades hasta llegar a la expresión 2.22, donde según el autor, E corresponde con un evento, evidencia o resultado, y H con una teoría o hipótesis.

$$WoE(H;E) = \ln\left(\frac{P(E|H)}{P(E|\overline{H})}\right) \tag{2.22}$$

En otros textos, debido a la relación de esta expresión con el crédito financiero, se puede encontrar reflejada como en 2.23. La distribución de buenos hace referencia a aquellos clientes que cumplen su obligación respecto a un préstamo y la distribución de malos a los que no.

$$WoE = \ln\left(\frac{Distribución\ de\ Buenos}{Distribución\ de\ Malos}\right) \tag{2.23}$$

Sabemos que la función logarítmica es estrictamente creciente, y que en 1 su valor es 0. Por ello, cuando el peso de la evidencia es cercano a 0, la distribución de buenos y malos es muy parecida, lo que significa que el poder predictivo es nulo. Por otro lado, un valor positivo quiere decir que la probabilidad de un evento respecto a una hipótesis es mayor que la probabilidad del mismo evento

bajo la hipótesis opuesta y, a mayor diferencia, mayor valor de *WoE*. Del mismo modo, se puede deducir el resultado contrario cuando el peso de la evidencia es negativo.

Cuando dos hipótesis tienen un valor similar de *WoE* se puede deducir que ambas tienen la misma capacidad de predicción sobre un evento. Este es el motivo por el cual se utiliza dicho criterio para discretizar el rango de una variable continua en intervalos semejantes. No obstante, la cuestión de interés es justamente la contraria, pasar de una variable categórica a otra continua. Para ello, se va a utilizar el propio valor del peso de la evidencia como la variable continua de codificación, ${}_z\phi_i$. En esta ocasión, la evidencia o resultado E es una categoría de la variable codificada, ${}_zx_i$, y la hipótesis H, la salida de cada observación. Matemáticamente se calcula mediante la expresión 2.24, donde el factor 100 es opcional y su efecto es limitado tras la normalización de los datos.

$$ {}_z\phi_i = WoE(Y; {}_zx_i) = 100\, ln\left(\frac{P(X_i = {}_zx_i \mid Y = {}_{Good}y)}{P(X_i = {}_zx_i \mid Y = {}_{Bad}y)}\right) = 100\, ln\left(\frac{\frac{f_{(X_i={}_zx_i \cap Y={}_{Good}y)}}{f_{(Y={}_{Good}y)}}}{\frac{f_{(X_i={}_zx_i \cap Y={}_{Bad}y)}}{f_{(Y={}_{Bad}y)}}}\right) \tag{2.24} $$

Una restricción que presenta este método se deduce de la expresión previa. La variable de salida debe ser categórica dicotómica, para así poder distinguir entre observaciones *"buenas"* y *"malas"*. Esta nomenclatura puede resultar difícil de extrapolar a problemas no relacionados con el crédito, por ello, resulta más fácil pensar en la no ocurrencia de un evento y en que sí suceda este, respectivamente. Por lo general, las observaciones con presencia del evento o *malas* suelen estar asociadas con la salida de menor frecuencia, ya que, por ejemplo, en un banco son menos los clientes que no pagan que los que sí lo hacen. En este caso, el impago es el evento.

Otra cuestión a tener en cuenta es qué sucede cuando alguna categoría no tiene representación en una de las dos salidas. En esta situación, el numerador o denominador sería igual a 0 y, como consecuencia, el peso de la evidencia tendería a menos infinito o más infinito, respectivamente. Para solventar este contratiempo se puede sumar una cantidad constante muy próxima a 0 al numerador y al denominador, para que así nunca se anulen completamente sin influir en el resto de resultados.

Para mayor claridad se añade el ejemplo de la tabla 2.6. En este se tratan los datos ficticios de una empresa que oferta un servicio bajo demanda con suscripción mensual. Se identifican dos tipos de clientes, los que mantienen su suscripción más de tres meses, buenos, y los que prescinden del servicio antes de ese tiempo, malos. El objetivo es realizar la codificación mediante el peso de la evidencia de la variable categórica situación laboral de cardinalidad igual 5.

Tabla 2.6 Ejemplo de codificación WoE.

Categoría $[{}_zx_i]$	**Buenos** $[f_{(X_i={}_zx_i \cap Y={}_{Good}y)}]$	**Malos** $[f_{(X_i={}_zx_i \cap Y={}_{Bad}y)}]$	**Dist. buenos** $[P(X_i = {}_zx_i \mid Y = {}_yGood)]$	**Dist. malos** $[P(X_i = {}_zx_i \mid Y = {}_{Bad}y)]$	**WoE** $[{}_z\phi_i]$
Jubilado/a	27	61	0.045	0.230	-163.2
Estudiante	84	86	0.140	0.325	-084.1
Desempleado/a	117	9	0.195	0.034	174.8
Autónomo/a	249	55	0.415	0.208	069.3
Asalariado/a	123	54	0.205	0.204	000.6
Total	600	265			

2.7.1 Factor de Bayes

Realmente, el cálculo del peso de la evidencia no es más que el logaritmo neperiano del factor de Bayes [40]. Este deriva directamente del teorema de Bayes, y también es usado como medida cuantitativa de la robustez de la evidencia. En esencia, el factor de Bayes es una comparación de cómo de bien dos hipótesis complementarias predicen los datos.

Por ello, parece buena idea emplear también este cociente de probabilidades para codificar las categorías. Matemáticamente queda como:

$$ {}_z\phi_i = 100 \left(\frac{P(X_i = {}_zx_i \mid Y = {}_{Good}y)}{P(X_i = {}_zx_i \mid Y = {}_{Bad}y)} \right) = 100 \left(\frac{\frac{f_{(X_i={}_zx_i \cap Y={}_{Good}y)}}{f_{(Y={}_{Good}y)}}}{\frac{f_{(X_i={}_zx_i \cap Y={}_{Bad}y)}}{f_{(Y={}_{Bad}y)}}} \right) \tag{2.25} $$

Indistintamente de la expresión que se utilice, 2.24 o 2.25, este método de codificación resulta en una única variable nueva por cada variable original categórica, por lo que, de nuevo, no se incrementa el tamaño del espacio de características. El inconveniente vuelve a ser la necesidad de conocer la salida de un subconjunto de muestras y, además, que esta debe ser categórica dicotómica.

2.8 Codificación por similitud

En el artículo *Similarity encoding for learning with dirty categorical variables* [24], los autores mencionan sin entrar en detalle una forma de codificar las categorías de una variable haciendo uso de la similitud entre ellas.

La idea consiste en usar para cada variable categórica X_i un vector de codificación ${}_z\Phi_i$ de tamaño igual a la cardinalidad de la variable, $a_i = p_i$. Luego, asociar cada elemento de dicho vector, ${}_z\phi_{i,t}$, con una categoría de la variable original, ${}_zx_i$. Posteriormente, calcular cada elemento del vector como la similitud entre la categoría a codificar, z, y la categoría asociada al elemento.

$$ \begin{aligned} {}_z\Phi_i = \left[{}_z\phi_{i,1},\ {}_z\phi_{i,2},\ \dots,\ {}_z\phi_{i,t},\ \dots,\ {}_z\phi_{i,a_i} \right] = \\ = \left[sim({}_zx_i, {}_1x_i),\ sim({}_zx_i, {}_2x_i),\ \dots, sim({}_zx_i, {}_zx_i),\ \dots, sim({}_zx_i, {}_{a_i}x_i) \right] \end{aligned} \tag{2.26} $$

Para el cálculo de la similitud existen numerosas expresiones, y el uso de una u otra dará lugar a diferentes resultados. En el artículo de Boriah et al. [19] se puede consultar una recopilación de 14 funciones para calcular la similitud y, de entre todas ellas, se ha escogido la siguiente:

$$ sim({}_rx_i, {}_zx_i) = \begin{cases} 1 & si \quad {}_rx_i = {}_zx_i \\ \frac{1}{1+\ln \frac{d}{f_{(X_i={}_rx_i)}} \cdot \ln \frac{d}{f_{(X_i={}_zx_i)}}} & si \quad {}_rx_i \neq {}_zx_i \end{cases} \tag{2.27} $$

El uso de la expresión previa hace que este método guarde cierta semejanza con *one-hot*. En concreto, por cada categoría existe una variable de codificación que toma el valor de 1 cuando la variable original es igual a dicha categoría. La diferencia radica en que ahora el resto de variables de codificación no son 0, sino que adoptan un valor entre 0 y 1 en función de la similitud entre categorías.

Para evidenciar este hecho se adjunta la tabla 2.7 con la codificación de la misma variable que se usó de ejemplo en *one-hot*, tabla 2.1. Se comprueba como la diagonal sigue estando formada por unos, pero el resto de elementos ya no son ceros. La simetría de la tabla se debe a la propiedad conmutativa de la función de cálculo de la similitud.

Tabla 2.7 Ejemplo de codificación por similitud.

Vector de codificación	Categorías				
	Azul	Rojo	Verde	Blanco	Negro
${}_{Azul}\Phi_{Color}$	1	0.353	0.179	0.657	0.965
${}_{Rojo}\Phi_{Color}$	0.353	1	0.576	0.335	0.634
${}_{Verde}\Phi_{Color}$	0.179	0.576	1	0.295	0.523
${}_{Blanco}\Phi_{Color}$	0.657	0.335	0.295	1	0.205
${}_{Negro}\Phi_{Color}$	0.965	0.634	0.523	0.205	1

2.9 Codificación basada en la frecuencia

Esta última codificación se basa en la única propiedad que se puede medir de forma directa en las variables categóricas: la frecuencia de aparición de cada categoría.

Su aplicación resulta tan sencilla como sustituir cada valor categórico por su frecuencia relativa. Matemáticamente queda como:

$$ {}_{z}\Phi_i = \frac{f_{(X_i =_z x_i)}}{d} \tag{2.28} $$

La ventaja de esta codificación es que se realiza con una única variable continua, independientemente de la cardinalidad y sin necesidad de conocer la salida. Además, al utilizar la frecuencia relativa, la codificación está normalizada en el intervalo $[0,1]$.

No obstante, la sencillez del método deriva en que solo es idóneo cuando la salida del problema guarda cierta relación con la frecuencia. En otros casos, el resultado será inservible, como por ejemplo, en la variable día de la semana en una serie temporal continua: debido a que todos sus valores se repiten de forma cíclica a lo largo del tiempo, todos tienen la misma frecuencia relativa.

3 Técnicas de análisis predictivo empleadas

Para resolver los dos tipos de problemas descritos en la introducción, clasificación y regresión, y de esta manera poder comparar el desempeño de las codificaciones previas, se va a hacer uso de recursos pertenecientes al campo del aprendizaje automático, en concreto, técnicas de análisis predictivo. Mediante la aplicación de estas es posible aprovechar la información condensada en acontecimientos o casos conocidos para generar modelos predictivos. Estos últimos nos permiten pronosticar eventos y resultados que son futuros o desconocidos.

Para realizar la tarea de predicción se ha desarrollado a lo largo de los años un gran número de técnicas, y el desempeño que produce una u otra está estrechamente vinculado con la naturaleza de los datos y del problema tratado.

En concreto, se va a usar el método conocido como *k* vecinos más cercanos en el problema de clasificación que se va a tratar. Para el problema de regresión se usará una red neuronal de función de base radial. A continuación, se realiza una descripción de ambas técnicas.

3.1 Método de los *k* vecinos más cercanos

Esta técnica fue descrita la primera vez por E. Fix y J. L. Hodges para llevar a cabo tareas de clasificación [41], siendo más conocida por la abreviatura *k-NN* con origen en su nombre en inglés, *k-nearest neighbors*. Más tarde, T. M. Cover y P. Hart continuaron con el estudio de las propiedades del método y acotaron inferior y superiormente el error obtenido en la clasificación [17].

Su principio de funcionamiento es muy sencillo y se basa en el cálculo de distancias. Primero, se parte de un conjunto de muestras cuya salida, también denominada clase, es conocida. Luego, durante la clasificación, una muestra con salida desconocida es clasificada en función de las salidas de las observaciones con las que guarda mayor relación, es decir, las que se encuentran más próximas. En concreto, el parámetro *k* que aparece en el nombre es el número de vecinos de la muestra desconocida que se tienen en cuenta durante la clasificación, y su finalidad es filtrar posibles muestras anómalas.

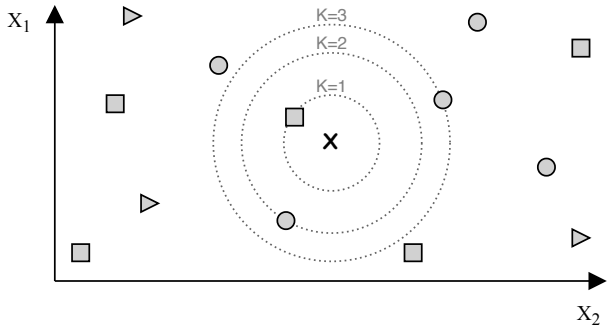

Figura 3.1 Ejemplo gráfico de clasificación *k-nn*.

El método es más sencillo de explicar con un ejemplo gráfico. En la figura 3.1 se representa un hipotético problema cuyo espacio de características es de dimensión 2, por lo que las variables pueden ser representadas en un plano. Todas las muestras con salida conocida son situadas en el plano en función del valor que toman sus variables de entrada, y la salida de cada una es ilustrada en función del símbolo que las representa.

Posteriormente, cuando se realiza la clasificación de una muestra desconocida, representada con una cruz, tan solo es necesario observar su entorno. Si k es uno, la salida predicha es igual a la de las muestras representadas con un cuadrado, puesto que la observación más cercana pertenece a esa clase. Si k es dos, la vecindad está formada por una observación con la clase representada por el cuadrado y otra por el círculo; ante este empate habría que establecer algún criterio de prioridad para seleccionar una de las dos, por ejemplo, la más cercana, que seguiría siendo el cuadrado como en la situación anterior. Por último, si k es tres, la salida representada por el círculo es mayoritaria, por lo que es esta la que se le asigna a la muestra clasificada.

En definitiva, para un tamaño de vecindad genérico k, la salida predicha de una muestra, $\hat{y}^i$, será igual a la más repetida de entre las k observaciones más cercanas con salida conocida. Matemáticamente se puede expresar como queda recogido en la ecuación 3.1, donde $\mathscr{K}$ es el conjunto de las k muestras conocidas más cercanas a la que está siendo clasificada y δ es una delta de Kronecker cuyo valor es uno cuando las dos salidas que compara son iguales.

$$\hat{y}^i = \arg\max_{y \in \mathscr{Y}} \sum_{j \in \mathscr{K}} \omega_j \delta_{y,\, y^j} \tag{3.1}$$

Además, el término ω_j se puede usar para ponderar la importancia de cada uno de los vecinos que intervienen en la clasificación. Lo más habitual es que el factor de ponderación sea función de la distancia entre la muestra nueva y su vecino, incrementando su valor mientras más próximos estén. De esta forma se consigue evitar la situación de empate del ejemplo para el caso $k = 2$. Una posibilidad es usar como peso la inversa de la distancia, normalizada con la inversa de todas las distancias con el vecindario. Así, al estar invertida, el peso es mayor conforme más cercas están y, al estar normalizada, la suma de todos los pesos es 1, lo cual siempre es deseable en todas las ponderaciones. Otras formas de calcular los pesos se pueden encontrar en la obra de Dudani [42].

$$\omega_j = \frac{d(i,j)^{-1}}{\sum_{p \in \mathscr{K}} d(i,p)^{-1}} \tag{3.2}$$

Como es evidente, para poder llevar a cabo la clasificación es necesario calcular las distancias entre la muestra nueva y las muestras de salida conocida, para así determinar cuáles de estas últimas son las más cercanas. Además, este cálculo también es requerido cuando la ponderación depende de la proximidad. Se utilizará la notación $d(a,b)$ para hacer referencia a la distancia entre dos observaciones cualesquiera y para su cálculo se podría usar, entre otras, la distancia de Mahalanobis o la de Minkowski. Esta última se expresa como:

$$d(a,b) = \left(\sum_{i=1}^{m} w_i \left(|X_i^a - X_i^b| \right)^p \right)^{1/p} \tag{3.3}$$

La métrica resultante depende del valor p y se pueden destacar tres casos, figura 3.2: cuando es 1 se tiene la distancia de Manhattan; cuando es 2, la euclídea; y cuando tiende a infinito, la de Chebyshev. Además, se ha añadido otro factor de ponderación w_i para alterar la importancia relativa que tiene cada una de las variables que definen el espacio de características. otro aspecto muy importante es la necesidad de normalizar los datos, ya que, si no se hace, las variables cuyos valores tengan los mayores órdenes de magnitud cancelarán el efecto del resto de variables en el cálculo de la distancia.

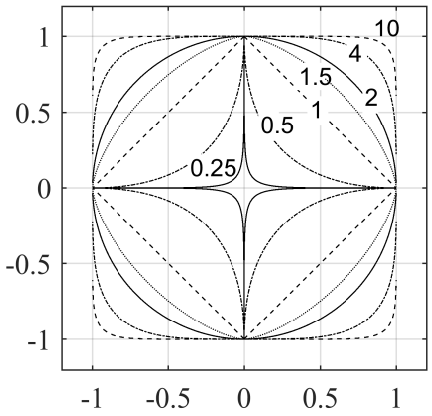

Figura 3.2 Conjunto de puntos con una distancia de Minkowski unidad respecto al origen para diferentes valores de p .

Además de llevar a cabo la clasificación, el método $k-nn$ es capaz de otorgar un valor que refleja la confianza de la predicción mediante un análisis de la vecindad [43, 44]. Gracias a esta característica, es posible rechazar predicciones cuya probabilidad de acierto es muy reducida o, al contrario, aprobar aquellas cuyo grado de confianza es muy elevado. Esta característica resulta de gran utilidad en problemas donde el coste asociado a un fallo en la clasificación es muy elevado y, por tanto, solo interesa actuar cuando es superado un umbral de confianza.

Por otro lado, para poder hacer uso de este clasificador con variables categóricas es necesario tratar los datos mediante codificaciones como las desarrolladas en el capítulo previo. Otra posibilidad es usar funciones de cálculo de distancias que admiten variables categóricas como las que se mencionaron en el estado del arte [19, 20, 21].

Para acabar, es cierto que la etapa de entrenamiento de este método consiste principalmente en almacenar las muestras con entrada y salida conocida para, posteriormente, ser usadas en el cálculo de distancias durante la clasificación. No obstante, es posible mejorar el resultado global mediante la

aplicación de técnicas más avanzadas que permiten aumentar la tolerancia del clasificador ante casos anómalos, disminuir el error, agilizar la clasificación, reducir la cantidad de memoria y los recursos computacionales necesarios, hacer frente a conjuntos desbalanceados en los que la predominancia de una clase provoca que la clasificación sea sesgada, etc. Algunos ejemplos de estas técnicas son:

- **Selección de prototipos**: Puede resultar beneficioso usar solo un subconjunto de las muestras conocidas en vez de todas de las que se dispone [45].
- **Entrenamiento de prototipos**: Además de usar solo un subconjunto de las muestras conocidas, se procesan y modifican las elegidas durante un algoritmo de entrenamiento [46], por ejemplo, *Learning Vector Quantization (LVQ)*.
- **División del espacio**: Para conocer cuáles son los vecinos más cercanos es necesario calcular la distancia a todas las observaciones conocidas. Sin embargo, existen métodos que dividen el espacio de características en regiones, o que almacenan los datos en estructuras especiales, con el objetivo de evitar tener que calcularlas todas [47, 48].
- **Entrenamiento del cálculo de distancias**: Además de existir numerosas funciones para calcular la proximidad, se han desarrollado algoritmos que aprenden que características son más importantes y lo reflejan en la función de distancia mediante pesos que ponderan las variables de entrada [49].

3.2 Redes neuronales de base radial

Este tipo de redes, conocidas en inglés como *Radial Basis Function Network (RBFN)*, aparecieron con el objetivo de llevar a cabo interpolación o aproximación de funciones. No obstante, su aplicación se ha extendido también en problemas de clasificación y regresión [50], siendo este último el uso que se le va a dar en este texto.

Estas redes tienen la peculiaridad de estar formadas por una capa de entrada, una única capa oculta y la de salida. Esta estructura, ilustrada en la figura 3.3, tiene la ventaja de disminuir el número de capas y neuronas totales en la red, lo que permite reducir los requisitos computacionales y los tiempos de entrenamiento. A continuación, se describe cada una de las capas.

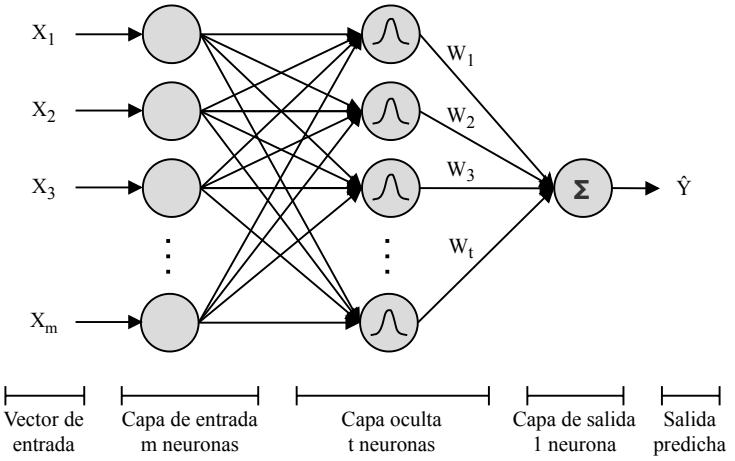

Figura 3.3 Estructura de una red neuronal de base radial.

La primera capa recibe el vector de entrada y tiene tantas neuronas como variables. Las salidas de estas neuronas no están ponderadas con ningún peso y se encuentran directamente conectadas con las neuronas de la capa oculta.

Luego, cada una de las neuronas de la capa oculta implementa una función de base radial. Estas funciones tienen la peculiaridad de depender únicamente de la distancia de la entrada a un centro. Una de las más utilizadas es la función gaussiana, expresión 3.4, que tras adaptarla a esta aplicación queda como la expresión 3.5. Esta última depende de dos parámetros. Uno es el vector $\mathbf{X}^P$, el cual constituye el centro de la función y, por lo general, es una observación con salida conocida del conjunto de datos denominada prototipo. El otro parámetro es β y permite controlar la pendiente de la curva resultante.

$$g(x) = a \exp\left(-\frac{(x-b)^2}{2c^2}\right), \qquad C > -1 \tag{3.4}$$

$$g(\mathbf{X}) = \exp\left(-\beta \left(d_{(\mathbf{X}, \mathbf{X}^P)}\right)^2\right), \qquad \beta > 0 \tag{3.5}$$

La distancia euclídea es la que se usa con mayor frecuencia en la aplicación de la función 3.5. Además, teniendo en cuenta que el exponente siempre es un número negativo, el rango está acotado entre 0 y 1. De esta forma, mientras más similares sean el vector de entrada y el prototipo, menor será la distancia entre ellos y, por tanto, más cercano a 1 el valor de la función. Por contra, mientras mayor sea la discrepancia entre ambos vectores, más tenderá la función a 0.

Esta respuesta se representa mediante el ejemplo de la figura 3.4. Se han usado vectores de un único elemento para facilitar la visualización y el centro de la campana está en 3, es decir, $X^P = 3$. Se han empleado varios valores de β para mostrar la influencia de este parámetro: cuando crece, la campana se estrecha; y cuanto más se aproxima a 0, más se suaviza la pendiente, ensanchando la campana y ofreciendo una transición más suave entre el valor máximo y mínimo.

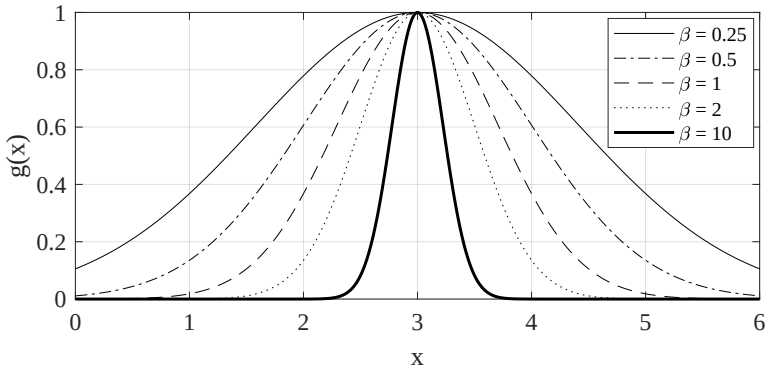

Figura 3.4 Familia de curvas centradas en 3 para diferentes valores de β de la expresión 3.5.

Obviamente, cada una de las neuronas de esta capa implementa su propia función con diferentes parámetros, ofreciendo así respuestas distintas ante el mismo vector de entrada. El número de neuronas es otro parámetro de diseño de la red: un número demasiado elevado puede provocar sobreajuste, mientras que un número reducido puede inhabilitar la red para producir predicciones correctas.

Por último, la capa de salida está formada por una única neurona. Su función es realizar la suma ponderada de todas sus entradas, las cuales están unidas a la salida de las neuronas de la capa oculta. La ponderación viene dada por los pesos asociados a estas conexiones, W. Toda la red se podría condensar en la expresión 3.6, donde $g_n(\mathbf{X})$ hace referencia a la función de base radial asociada a la neurona n-ésima.

$$\hat{Y} = \sum_{n=1}^{t} W_n \, g_n(\mathbf{X}) \tag{3.6}$$

Una vez construida la red es necesario llevar a cabo el aprendizaje de esta [51]. Durante este proceso, aprende de las muestras con salida conocida para, posteriormente, poder realizar predicciones minimizando el error cometido. En el aprendizaje se calcula [52]: el tamaño de la capa oculta, los pesos de unión entre las dos últimas capas, los prototipos que actúan de centro en cada neurona con función de base radial y el parámetro que define la anchura de cada campana.

Una forma típica de seleccionar los prototipos para el centro de las neuronas es escogerlos al azar de entre todas las muestras con salida conocida. Otra alternativa más elaborada son las técnicas de *clustering* [53]; mediante la aplicación de estas, las observaciones similares se pueden agrupar y se obtiene un representante de cada grupo adecuado para ser usado como prototipo de cada neurona. Algunos algoritmos conocidos son *k-means* [54] o *k-Harmonic* [55]. También se puede recurrir al método conocido como mínimos cuadrados ortogonales [56], el cual construye la red de forma iterativa y al mismo tiempo otorga un valor a los pesos de las conexiones.

Otras alternativas para el cálculo de los pesos son el uso de métodos de descenso del gradiente [57] mediante los que se busca reducir el error cometido, la aplicación de algoritmos de esperanza-maximización [58] o el uso de lógica difusa [59].

Para concluir, los comentarios que se realizaron en el método *k-nn* sobre el uso de variables categóricas y la necesidad de normalizar los datos también se aplican en esta ocasión. El motivo es que ambas técnicas tienen como base el cálculo de distancias.

4 Desempeño de las codificaciones propuestas

En este capítulo se van a recoger los resultados obtenidos en dos problemas reales haciendo uso de las codificaciones previas. Uno tratará la clasificación de personas según su nivel económico y, el otro, la predicción del volumen de llamadas. Para cada problema se expone una descripción de los datos, los algoritmos y técnicas empleados, y la evaluación de los resultados.

4.1 Predicción del nivel económico

4.1.1 Descripción de los datos

En este problema se va a realizar la clasificación en dos grupos de los miembros de una población según la cantidad de ingresos que generan en un año, en concreto, si esta cantidad supera o no el umbral de 50000 $. El conjunto de datos usado se puede encontrar en el repositorio público *CensusIncome* [60].

Estos datos se encuentran divididos en dos conjuntos y, tras unirlos y eliminar los registros incompletos, se dispone de la información de un total de 45222 personas. El número de variables, contando la de salida, es 15 y hay tanto categóricas como continuas. A continuación, se describen brevemente:

- **age**: Edad en años.
- **capital-gain**: Ganancia de capital.
- **capital-loss**: Pérdida de capital.
- **hours-per-week**: Número de horas a la semana.
- **fnlwgt**: Pesos calculados por el creador del repositorio. Personas con valores similares poseen características demográficas semejantes.
- **education-num**: Nivel de educación.
- **workclass**: Tipo de trabajo que desempeña.
  Private (33307), Self-emp-not-inc (3796), Local-gov (3100), State-gov (1946), Self-emp-inc (1646), Federal-gov (1406), Without-pay (21).

- **education**: Tipo de estudios.
  HS-grad (14783), Some-college (9899), Bachelors (7570), Masters (2514), Assoc-voc (1959), 11th (1619), Assoc-acdm (1507), 10th (1223), 7th-8th (823), Prof-school (785), 9th (676), 12th (577), Doctorate (544), 5th-6th (449), 1st-4th (222), Preschool (72).
- **marital-status**: Estado civil.
  Married-civ-spouse (21055), Never-married (14598), Divorced (6297), Separated (1411), Widowed (1277), Married-spouse-absent (552), Married-AF-spouse (32).
- **occupation**: Profesión.
  Craft-repair (6020), Prof-specialty (6008), Exec-managerial (5984), Adm-clerical (5540), Sales (5408), Other-service (4808), Machine-op-inspct (2970), Transport-moving (2316), Handlers-cleaners (2046), Farming-fishing (1480), Tech-support (1420), Protective-serv (976), Priv-house-serv (232), Armed-Forces (14).
- **native-country**: País de origen.
  United-States (41292), Mexico (903), Philippines (283), Germany (193), Puerto-Rico (175), Canada (163), El-Salvador (147), India (147), Cuba (133), England (119), China (113), Jamaica (103), South (101), Italy (100), Dominican-Republic (97), Japan (89), Guatemala (86), Vietnam (83), Columbia (82), Poland (81), Haiti (69), Portugal (62), Iran (56), Taiwan (55), Greece (49), Nicaragua (48), Peru (45), Ecuador (43), France (36), Ireland (36), Thailand (29), Hong (28), Cambodia (26), Trinadad&Tobago (26), Yugoslavia (23), Outlying-US(Guam-USVI-etc) (22), Laos (21), Scotland (20), Honduras (19), Hungary (18), Holand-Netherlands (1).
- **relationship**: Relación familiar.
  Husband (18666), Not-in-family (11702), Own-child (6626), Unmarried (4788), Wife (2091), Other-relative (1349).
- **race**: Raza.
  White (38903), Black (4228), Asian-Pac-Islander (1303), Amer-Indian-Eskimo (435), Other (353).
- **sex**: Sexo.
  Male (30527), Female (14695).
- **income**: Nivel de ingresos.
  <=50K (34014), >50K (11208).

Las 6 primeras variables se han tratado como continuas, a pesar de que la edad y el nivel de educación podrían considerarse como discretas. No obstante, no se ha hecho así puesto que las 8 variables siguientes sí son categóricas y ya representan un número significativo para hacer las pruebas. En estas se ha indicado las categorías que pueden adoptar junto a la frecuencia absoluta de aparición en los datos. La última variable restante es dicotómica y corresponde con la salida del problema. El tratamiento de esta no supone ningún cambio importante, ya que ambas categorías se usan simplemente como etiquetas para representar cada clase y no influyen en los cálculos.

Como se puede observar, el número de personas que no superan el umbral de ingresos es el triple de los que sí, por lo que estamos ante un problema desbalanceado. Para sortear esta dificultad se ha balanceado el conjunto de datos eliminando muestras al azar de la clase predominante hasta tener el mismo número de registros de ambos tipos, quedando así 22416 muestras para usar, siendo cada mitad de un tipo. Posteriormente, como es típico en estas aplicaciones, se ha dividido ese número de muestras en dos conjuntos disjuntos también exactamente balanceados. El primero, denominado *CE*, está destinado al entrenamiento del modelo predictivo y posee el 80 % de las muestras; el segundo, *CP*, se emplea para la validación del modelo y se compone del 20 % restante.

4.1.2 Procedimiento para la resolución

Para realizar la clasificación se ha usado el algoritmo *k-nn* descrito en el capítulo previo. El parámetro k se ha fijado en 1, por lo que solo se tiene en cuenta el vecino más cercano, lo que evita tener que establecer una ponderación entre ellos. Se ha empleado la distancia euclídea, en la cual tampoco se ha ponderado la importancia relativa de cada variable. Eso sí, todas ellas han sido normalizadas en unidades tipificadas mediante la expresión genérica 4.1, donde x es el valor a normalizar, μ la media de la variable y σ la desviación típica.

$$z = \frac{x - \mu}{\sigma} \tag{4.1}$$

Se han hecho pruebas con diferentes cantidades de prototipos elegidos al azar, siendo cada mitad de una clase. Además, estos prototipos se han entrenado mediante el algoritmo conocido como *learning vector quantization (LVQ)*. Hay estudios que anuncian buenos resultados con la combinación de este entrenamiento y el parámetro $k = 1$ [61].

4.1.3 Desempeño de las codificaciones

Para la evaluación de los resultados se ha optado por usar los ratios que derivan directamente de la matriz de confusión clásica de una clasificación binaria [62], tabla 4.1. Considerando que lo que se busca detectar son las personas con ingresos altos, los 4 elementos que aparecen en la tabla se pueden definir para este problema como:

- **Verdaderos Negativos (VN)**: Número de personas con predicción correcta de ingresos menores de 50000 $.
- **Verdaderos Positivos (VP)**: Número de personas con predicción correcta de ingresos mayores de 50000 $.
- **Falsos Negativos (FN)**: Número de personas con predicción errónea de ingresos menores de 50000 $.
- **Falsos Positivos (FP)**: Número de personas con predicción errónea de ingresos mayores de 50000 $.

Tabla 4.1 Matriz de confusión.

	Predicción	
Realidad	Negativo	Positivo
Negativo	VN	FP
Positivo	FN	VP

A partir de estos, se calculan las tres métricas que se van a utilizar. La primera es la sensibilidad o ratio de verdaderos positivos, que corresponde con la proporción de individuos con ingresos mayores al umbral que son correctamente identificados de entre todos los que verdaderamente cumplen dicha condición, expresión 4.2. La segunda métrica es el porcentaje de falsas alarmas, y

representa la proporción de individuos detectados de forma errónea con altos ingresos respecto al total de bajos ingresos, expresión 4.3. La última es la exactitud, que se puede definir como la proporción de predicciones correctas respecto al total de muestras, expresión 4.4.

$$SEN = \frac{VP}{VP+FN} \tag{4.2}$$

$$PFA = \frac{FP}{FP+VN} \tag{4.3}$$

$$EFF = \frac{VP+VN}{VP+FN+FP+VN} \tag{4.4}$$

A continuación, se describen las pruebas que se han realizado con las codificaciones estudiadas.

- **Etiquetado Frec.**: Las categorías de cada variable se han sustituido por números enteros consecutivos en función del número de veces que se repiten. A la de mayor frecuencia le corresponde el 1; y *n* a la que menos aparece, siendo *n* igual a la cardinalidad de la variable.
- **One-hot**: Esta codificación es aplicada tal como se describe en la sección 2.2.
- **One-hot 100**: Igual que la anterior, pero sin usar las variables de codificación que representan categorías con frecuencia menor a 100. Esto se hace con el objetivo de reducir la cardinalidad afectando al menor número posible de observaciones.
- **One-cold**: Esta codificación es aplicada tal como se describe en la sección 2.2.1.
- **Rank-hot**: Esta codificación es aplicada tal como se describe en la sección 2.2.2. Las categorías se ordenan haciendo uso del número entero de *Etiquetado Frec.*
- **Binaria**: Esta codificación es aplicada tal como se describe en la sección 2.3. Las categorías se ordenan haciendo uso del número entero de *Etiquetado Frec.*
- **Gray**: Esta codificación es aplicada tal como se describe en la sección 2.3.1. Las categorías se ordenan haciendo uso del número entero de *Etiquetado Frec.*
- **Entity Embeddings**: Esta codificación es aplicada como se describe en la sección 2.4; con la salvedad de que se ha entrenado una red por cada variable categórica, y no una única con todas las variables del problema. El tamaño de la capa de *embeddings* en cada variable es igual a la mitad de la cardinalidad de esta.
- **Estadística Sal.**: Esta codificación es aplicada como se describe en la sección 2.5. Para la ponderación se ha usado la constante $\Lambda = 1$, por lo que solo se tiene en cuenta la probabilidad a posteriori.
- **Estadística Sal. 100**: Igual que la anterior, pero considerando ambas probabilidades. Para ello, la ponderación es calculada con los parámetros $\lambda_1 = 500$ y $\lambda_2 = 100$.
- **WoE**: Esta codificación es aplicada tal como se describe en la sección 2.7, considerando como muestras "buenas" aquellas con bajos ingresos.
- **Bayes**: Esta codificación es aplicada tal como se describe en la sección 2.7.1, considerando como muestras "buenas" aquellas con bajos ingresos.
- **Similitud**: Esta codificación es aplicada tal como se describe en la sección 2.8.
- **Frec. Relativa**: Esta codificación es aplicada tal como se describe en la sección 2.9.

Tabla 4.2 Resultados de las codificaciones en *CensusIncome*.

Id.	Codificación	Nº prototipos	CE			CP		
			EFF	SEN	PFA	EFF	SEN	PFA
1	Etiquetado Frec.	50	0.7720	**0.8531**	0.3091	0.7654	**0.8480**	0.3171
2	One-hot	50	0.7901	0.8057	0.2256	0.7835	0.7943	0.2273
3	One-hot 100	50	**0.7989**	**0.8170**	0.2193	**0.7950**	**0.8078**	**0.2179**
4	One-cold	50	0.7832	0.8004	0.2340	0.7770	0.7988	0.2449
5	Rank-hot	50	0.7835	0.7975	0.2305	0.7639	0.7839	0.2562
6	Binaria	50	0.7965	0.7980	**0.2051**	**0.7970**	0.8015	**0.2075**
7	Gray	50	0.7922	0.8117	0.2273	0.7729	0.7948	0.2490
8	Entity Embeddings	50	0.7902	0.8011	0.2207	0.7803	0.7957	0.2350
9	Estadística Sal.	50	**0.8012**	**0.8425**	0.2400	**0.7959**	**0.8363**	0.2445
10	Estadística Sal. 100	50	**0.8022**	0.8100	**0.2057**	**0.7982**	0.8029	**0.2066**
11	WoE	50	0.7975	0.8041	**0.2092**	0.7907	0.8002	**0.2188**
12	Bayes	50	0.7945	0.8020	**0.2130**	0.7887	0.7993	0.2219
13	Similitud	50	**0.8025**	**0.8326**	0.2276	0.7941	**0.8218**	0.2336
14	Frec. Relativa	50	0.7843	0.8079	0.2394	0.7758	**0.8078**	0.2562
15	Etiquetado Frec.	75	0.7877	**0.8328**	0.2574	0.7767	0.8191	0.2657
16	One-hot	75	0.7992	**0.8348**	0.2363	0.7817	**0.8209**	0.2576
17	One-hot 100	75	0.7972	**0.8347**	0.2404	0.7887	0.8169	0.2395
18	One-cold	75	0.7964	**0.8337**	0.2410	0.7839	**0.8277**	0.2598
19	Rank-hot	75	0.7930	0.8316	0.2457	0.7846	0.8200	0.2508
20	Binaria	75	**0.8019**	0.8163	**0.2124**	0.7941	0.8106	0.2224
21	Gray	75	0.7971	0.8227	0.2285	0.7812	0.8137	0.2512
22	Entity Embeddings	75	0.7852	0.8135	0.2430	0.7826	0.8097	0.2445
23	Estadística Sal.	75	**0.8049**	0.8250	0.2153	**0.8002**	0.8178	**0.2174**
24	Estadística Sal. 100	75	**0.8058**	0.8240	**0.2125**	**0.7961**	0.8101	**0.2179**
25	WoE	75	**0.8079**	0.8213	**0.2055**	**0.8067**	**0.8227**	**0.2093**
26	Bayes	75	0.7979	0.8161	0.2204	**0.7972**	**0.8223**	0.2278
27	Similitud	75	0.7986	0.8037	**0.2065**	0.7894	0.7970	**0.2183**
28	Frec. Relativa	75	0.7905	0.8288	0.2478	0.7738	0.8182	0.2706
29	Etiquetado Frec.	100	0.7852	0.8305	0.2601	0.7702	0.8101	0.2697
30	One-hot	100	0.8019	0.8322	0.2285	0.7779	0.8151	0.2594
31	One-hot 100	100	0.7995	0.8225	0.2236	0.7907	0.8155	0.2341
32	One-cold	100	0.7949	**0.8368**	0.2471	0.7882	**0.8327**	0.2562
33	Rank-hot	100	0.7927	0.8204	0.2349	0.7799	0.8069	0.2472
34	Binaria	100	**0.8034**	0.8235	**0.2168**	0.7918	0.8155	**0.2318**
35	Gray	100	0.8020	0.8297	0.2258	0.7862	0.8088	0.2364
36	Entity Embeddings	100	0.7912	0.8341	0.2517	0.7848	0.8277	0.2580
37	Estadística Sal.	100	**0.8078**	0.8318	**0.2162**	**0.8045**	0.8290	**0.2201**
38	Estadística Sal. 100	100	**0.8065**	0.8215	**0.2085**	**0.8058**	0.8205	**0.2088**
39	WoE	100	**0.8072**	**0.8353**	0.2209	**0.7988**	**0.8313**	0.2336
40	Bayes	100	0.7969	0.8144	**0.2207**	0.7864	0.8020	**0.2291**
41	Similitud	100	0.8010	**0.8457**	0.2437	**0.7986**	**0.8412**	0.2440
42	Frec. Relativa	100	0.7935	**0.8465**	0.2595	0.7774	**0.8340**	0.2792

En la tabla 4.2 se han recogido los resultados numéricos obtenidos con cada codificación. Las filas se han divido en tres grupos para separar las clasificaciones con diferente número de prototipos: 50, 75 y 100. Además, para cada grupo y columna, se han resaltado los 4 mejores valores.

En la clasificación con 50 prototipos destaca la codificación *Estadística Sal. 100* por los buenos valores que presentan los tres indicadores. También se obtienen resultados notables con *WoE*, *Binaria* y *One-hot 100*.

La codificación que más resalta cuando se hace uso de 75 prototipos es *WoE*, seguida de cerca por *Estadística Sal* y *Estadística Sal. 100*.

Por último, con 100 prototipos, la codificación *Estadística Sal. 100* es la que genera los mejores resultados. El resto presentan como mínimo 2 puntos más en el porcentaje de falsas alarmas sin mejorar considerablemente los otros dos indicadores.

Para terminar, se representan todos los resultados anteriores en una gráfica con el ratio *PFA* en el eje de abscisas y *SEN* en el de ordenadas, lo que se conoce como espacio ROC, *Receiver Operating Characteristic*. En este tipo de gráficas, las clasificaciones serán mejores mientras más cerca estén del punto $(0,1)$, es decir, mientras más arriba y a la izquierda se sitúen. Es más, si la clasificación se posiciona cerca de la diagonal definida por los puntos $SEN = PFA$, se puede decir que es tan mala como una clasificación aleatoria.

En la figura 4.1 se representan las clasificaciones del conjunto de entrenamiento, y en figura 4.2 las del conjunto de prueba. En ambas se comprueba como las clasificaciones con identificador 25, 37 y 39 son las mejores, correspondientes a *WoE*, *Estadística Sal.* y *WoE*, respectivamente.

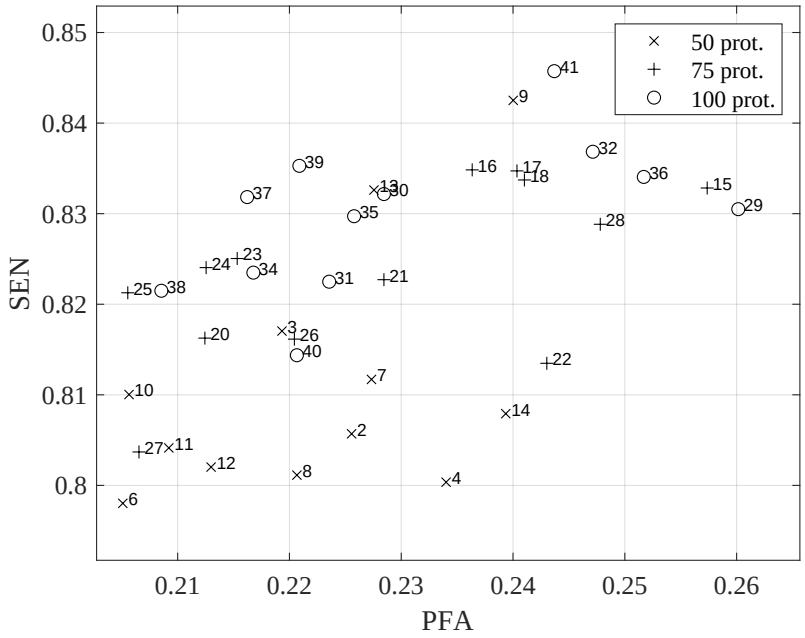

Figura 4.1 Representación de las clasificaciones del *CE* en el plano *PFA*-*SEN*.

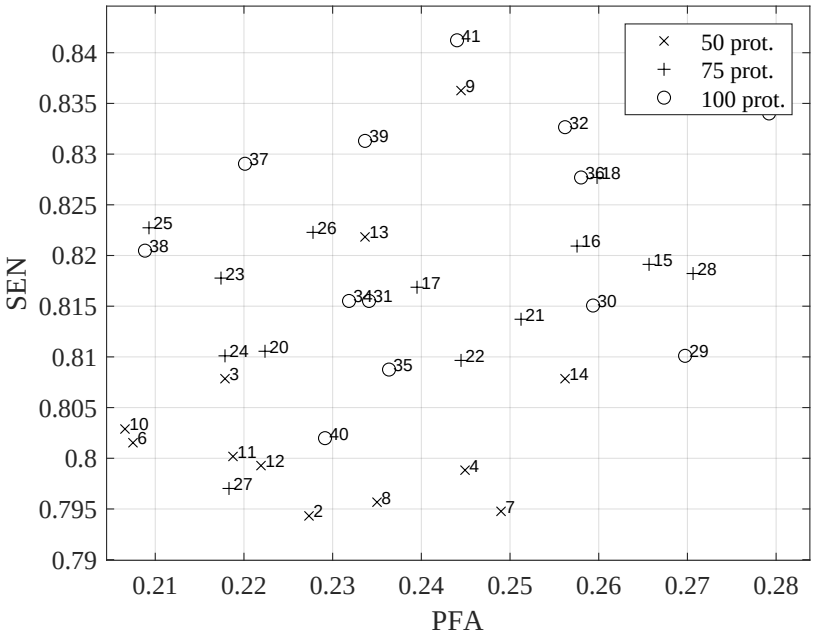

Figura 4.2 Representación de las clasificaciones del *CP* en el plano *PFA*-*SEN*.

4.2 Predicción del volumen de llamadas

4.2.1 Descripción de los datos

En este problema se va a tratar la predicción de la cantidad de llamadas que va a recibir un centro de atención al cliente en dos horizontes de predicción diferentes: a la hora siguiente de la actual, y una semana más tarde a la misma hora de la actual. Por tanto, es un problema de regresión y, en concreto, de serie temporal, debido a la dependencia con el tiempo. Estos datos ya han sido utilizados previamente para estudiar la selección de modelos y la elección de variables a utilizar [63, 64].

El conjunto almacena el registro desde julio de 2001 hasta marzo de 2002 de una serie de datos del centro de atención actualizados cada hora. En total se compone de 5760 observaciones. De entre todas las variables, las entradas al modelo que se van a usar son:

- El volumen de llamadas una semana antes a la misma hora que la actual.
- El volumen de llamadas una semana antes una hora más tarde que la actual.
- El volumen de llamadas una semana antes dos horas más tarde que la actual.
- El volumen medio de llamadas de las 7 horas consecutivas centradas en la semana antes una hora más tarde que la actual.
- El volumen de llamadas del primer día anterior al actual que sea del mismo tipo que el día del instante a predecir. Se consideran tres tipos de días: 1, de lunes a viernes; 2, sábado; y 3, domingo o festivo.
- El día de la semana en el instante a predecir.
  Lunes, Martes, Miércoles, Jueves, Viernes, Sábado, Domingo.
- La naturaleza del día en el instante a predecir.
  Laborable, Festivo.

En este caso, solo las dos últimas variables son categóricas, por lo que serán las únicas a las que se les aplique la codificación. Por otro lado, la salida del problema es el volumen de llamadas en el instante a predecir, tal y como se ha anunciado previamente.

De nuevo, el conjunto de datos original ha sido dividido en dos. El 80 % inicial para el entrenamiento, *CE*, y el 20 % final restante para la validación, *CP*.

En la figura 4.3 se ilustra la evolución del volumen de llamadas durante 4 semanas consecutivas, y se puede apreciar la existencia de un patrón que se repite a lo largo del tiempo. Los días están claramente marcados por un gran pico en el número de llamadas y las noches se reflejan por un descenso a 0. El lunes es el día que más llamadas se recibe y, hasta el viernes, el valor máximo diario no sufre gran variación. No obstante, los sábados y domingos se produce un gran descenso. Además, los dos máximos relativos presentes todos los días se producen casi siempre a las 12 y 18 horas.

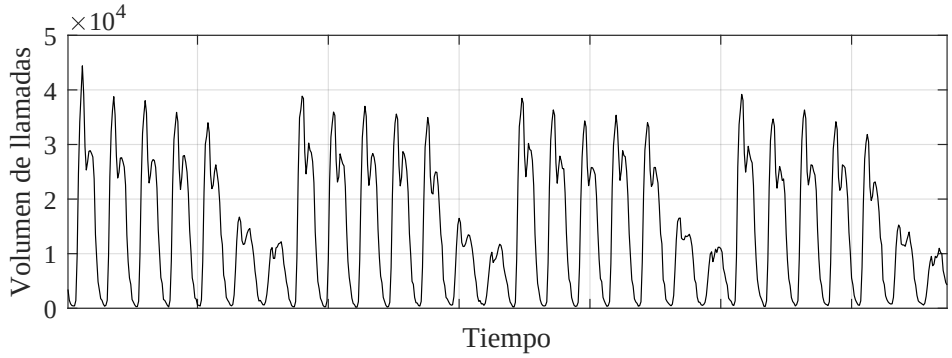

Figura 4.3 Evolución del volumen de llamadas entrantes durante tres semanas.

4.2.2 Procedimiento para la resolución

Para realizar las predicciones se ha empleado una red neuronal de base radial como las descritas en el capítulo previo. Se han usado 30 neuronas en la capa oculta, y la función que implementan es una gaussiana basada en la distancia euclídea. Para el entrenamiento de la red se ha empleado el algoritmo conocido como *least mean square (LMS)* [65]. Además, todos los datos han sido normalizados haciendo uso de la misma expresión 4.1 usada en el problema previo.

4.2.3 Desempeño de las codificaciones

Para evaluar y comparar los resultados se ha empleado el error porcentual medio absoluto (MAPE), expresión 4.5. El valor d es el tamaño del conjunto sobre el que se está evaluando el error, Y el valor real de la salida e $\hat{Y}$ el valor predicho de salida.

$$MAPE = \frac{1}{n}\sum_{k=1}^{d}\left(\frac{|Y^k - \hat{Y}^k|}{Y^k}\right) \tag{4.5}$$

A continuación, se describen las pruebas que se han realizado con las codificaciones estudiadas. Hay que tener que no todas ellas son de aplicación debido a que la variable de salida es continua.

- **Etiquetado**: Las variables que representan el día de la semana y la naturaleza del día se combinan en una sola con los siguientes valores: 0.8 los días de lunes a viernes, 0.4 los sábados, y 0.25 los domingos y los días festivos.
- **Etiquetado Natural**: Los días de la semana son sustituidos por números enteros consecutivos del 1 al 7 siguiendo el orden natural de la semana: el lunes es el 1 y el domingo el 7. Los días laborables se representan con el 1 y los festivos con el 2.
- **One-hot**: Esta codificación es aplicada tal como se describe en la sección 2.2.
- **One-cold**: Esta codificación es aplicada tal como se describe en la sección 2.2.1.
- **Rank-hot**: Esta codificación es aplicada como se describe en la sección 2.2.2, para ello, se hace uso del orden natural de los días.
- **Binaria**: Esta codificación es aplicada como se describe en la sección 2.3, para ello, se hace uso del orden natural de los días.
- **Gray**: Esta codificación es aplicada como se describe en la sección 2.3.1, para ello, se hace uso del orden natural de los días.
- **E. Embeddings 1**: Esta codificación es aplicada como se describe en la sección 2.4; con la salvedad de que se ha entrenado una red por cada variable categórica, y no una única con todas las variables del problema. El tamaño de la capa de *embeddings* en cada variable es igual a la mitad de la cardinalidad de esta.
- **E. Embeddings 2**: Esta codificación es aplicada como se describe en la sección 2.4; con la salvedad de que se ha entrenado una red por cada variable categórica, y no una única con todas las variables del problema. El tamaño de la capa de *embeddings* en cada variable es igual a la cardinalidad de esta.
- **Estadística Sal.**: Esta codificación es aplicada como se describe en la sección 2.5. Para la ponderación se ha usado la constante $\Lambda = 1$, por lo que solo se tiene en cuenta la probabilidad a posteriori.

En la tabla 4.3 se recogen los resultados numéricos obtenidos. Las primeras 10 filas están referidas a la predicción del volumen de llamadas a la hora siguiente, y las otras 10 restantes, a la próxima semana. Se han resaltado las dos mejores opciones para cada tipo de predicción y conjunto.

Las codificaciones por *Etiquetado* y *Estadística Sal.* destacan sobre todas las demás. El error cometido con estas es considerablemente inferior que con el resto.

Es cierto que ambas producen resultados similares, pero hay que tener en cuenta que la forma en la que codifican las variables es muy diferente. En *Etiquetado*, los valores numéricos son obtenidos mediante el análisis manual de los datos seguido de un proceso de prueba y error. En cambio, en *Estadística Sal.*, la codificación se realiza de forma automática.

En la figura 4.4 se representa el volumen de llamadas predicho con la codificación *Estadística Sal.* y el volumen real. Los valores están normalizados, y el intervalo temporal mostrado corresponde al conjunto de pruebas. Se observa como las diferencias más notables se producen en los instantes de mayor actividad del día, estando casi siempre el volumen predicho por encima del real.

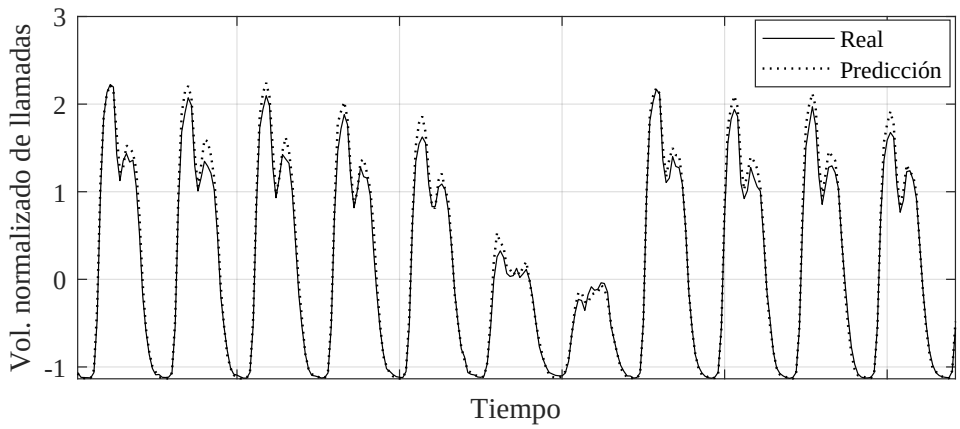

Figura 4.4 Comparación del volumen de llamadas real y predicho.

Tabla 4.3 Resultados de las codificaciones en la predicción del volumen de llamadas.

Id.	Codificación	Tipo de predicción	CE	CP
			MAPE	MAPE
1	Etiquetado	Horaria	**0.1490**	**0.1339**
2	Etiquetado Natural	Horaria	0.2193	0.2302
3	One-hot	Horaria	0.2862	0.2988
4	One-cold	Horaria	0.2862	0.2988
5	Rank-hot	Horaria	0.2944	0.8072
6	Binaria	Horaria	0.2242	0.2319
7	Gray	Horaria	0.2486	0.2640
8	E. Embeddings 1	Horaria	0.2810	0.2872
9	E. Embeddings 2	Horaria	0.2156	0.1954
10	Estadística Sal.	Horaria	**0.1432**	**0.1278**
11	Etiquetado	Semanal	**0.1508**	**0.1191**
12	Etiquetado Natural	Semanal	0.1620	0.1323
13	One-hot	Semanal	0.2710	0.2237
14	One-cold	Semanal	0.2710	0.2237
15	Rank-hot	Semanal	0.2536	0.6773
16	Binaria	Semanal	0.2267	0.1970
17	Gray	Semanal	0.1934	0.1596
18	E. Embeddings 1	Semanal	0.2324	0.2005
19	E. Embeddings 2	Semanal	0.3127	0.2847
20	Estadística Sal.	Semanal	**0.1532**	**0.1202**

5 Conclusiones

El objetivo del trabajo consistía en el estudio de la representación numérica de las variables categóricas mediante técnicas de codificación, con el objetivo de poder extraer la información que estas condensan.

Como resultado, se han descrito y probado 13 codificaciones. Es cierto que no hay ninguna que destaque en gran medida sobre las demás, pero entre todas ellas es posible afrontar situaciones de diferente índole: problemas con salida discreta, problemas con salida continua, variables categóricas de gran cardinalidad, variables categóricas con un orden implícito entre sus categorías, variables categóricas cuyos valores se pueden agrupar en conjuntos semejantes, etc.

Además, en uno de los problemas tratados, concretamente el de las llamadas del centro de atención al cliente con la codificación basada en estadísticas de la variable de salida, se ha comprobado que el uso de estas técnicas puede igualar el resultado obtenido mediante la sustitución de las categorías por los números resultantes de un tedioso y lento estudio racional de los datos. Lo que significa que esta tarea puede ser sorteada mediante la aplicación de un algoritmo automático, facilitando así el desarrollo y aplicación de modelos de aprendizaje automático en problemas con variables categóricas.

Por otro lado, las funciones en MATLAB® para realizar las codificaciones se han programado para que sean aplicables a cualquier problema, y no únicamente a los tratados en las pruebas. Por tanto, se ha desarrollado una librería de funciones que puede ser aplicada con el único requisito de organizar los datos de una forma concreta.

5.1 Líneas futuras

La continuación de este trabajo se puede abordar de diferentes formas. La más inmediata consiste en investigar y estudiar otros métodos de codificación no recogidos en este texto, o poner a prueba los ya presentados con nuevos problemas.

Otra alternativa sería el análisis de la combinación de codificaciones. Es decir, en vez de utilizar la misma para todas las variable categóricas del problema, utilizar en cada variable aquella que sea más conveniente. Es más, la determinación de cuál es la codificación más idónea en cada una también puede ser objeto de estudio para continuar el trabajo.

Asimismo, puede ser útil reflexionar sobre técnicas para reducir la dimensionalidad de las codificaciones que provocan un incremento en el tamaño del vector de entrada. De este modo, se mejoraría el tratamiento de las variable categóricas de gran cardinalidad y, por tanto, el desempeño de los modelos predictivos.

Por otro lado, podría ser interesante cambiar el enfoque en el tratamiento de las variables categóricas. En este trabajo se ha realizado mediante la codificación de las categorías, pero el estudio de otras alternativas también es atractivo. Por ejemplo, mediante funciones de cálculo de distancia adaptadas a variables categóricas, o mediante la combinación de modelos predictivos para resolver un problema. Esta última idea consiste en usar dos o más modelos, cada uno especializado en un tipo de variable, y combinar la salida de todos para obtener el resultado final del problema.

Anexo A

Códigos implementados

En este anexo se recogen las funciones implementadas en MATLAB® para llevar a cabo las codificaciones estudiadas previamente. En la cabecera de cada una se puede consultar una breve descripción de la función y de sus entradas y salidas.

Todas las funciones tienen dos entradas principales. La primera se llama *datos*, y corresponde con una matriz con una fila por cada observación y una columna por cada variable. La variable de la primera columna debe ser un identificador de cada observación, y la última columna, la salida del problema. En esta matriz, los valores de las variables categóricas son representados por números enteros.

La segunda entrada principal es *features*. Esta consiste en una estructura de dimensión igual al número de columnas de la matriz de entrada *datos*, sin contar la primera con los identificadores. Sus campos son los siguientes:

- **nombre**: Cadena de texto con el nombre de la variable.
- **tipo**: *'%f'* si la variable es continua o *'%s'* si la variable es categórica.
- **columna**: Entero que indica la columna de la matriz *datos* en la que se encuentra la variable en cuestión.
- **valores**: Cell de dimensión igual a la cardinalidad de la variable. En cada componente hay una cadena de texto con la representación textual de cada categoría. Este campo está vacío si la variable es continua.
- **nValores**: Entero con la cardinalidad de la variable. Este campo está vacío si la variable es continua.
- **frecuencia**: Vector de dimensión igual a la cardinalidad de la variable. En cada componente se indica el número de veces que aparece cada categoría. Este campo está vacío si la variable es continua.
- **frecuenciaRel**: Igual que la anterior pero con la frecuencia relativa.
- **representacion**: Vector de dimensión igual a la cardinalidad de la variable. En cada componente se indica el número entero con el que se representa cada categoría en la matriz *datos*. Este campo está vacío si la variable es continua.

A.1 Codificaciones

A.1.1 En clasificación y regresión

Codificación one-hot

Código A.1 onehot.m.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Este script transforma un archivo de datos ya procesado por main_cargarDatos
%de tal forma que solo incluye las variables categóricas y les realiza una
%codificación one hot.
%
%Uso:
%  [datosCod,nombresCabecera,cabecera,features]=onehot(datos,features)
%
%Entradas:
%  datos: Matriz de datos de las muestras dadas por el preprocesamiento
%     realizado en main_cargarDatos.m
%
%  features: Estructura con la información de cada variable dada por el
%     preprocesamiento realizado en main_cargarDatos.m
%
%Salidas:
%  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
%    como número de valores diferentes haya entre todas las variables
%    categóricas. Más una columna para los índices.
%
%  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
%     la variable de cada columna de la matriz datosCod.
%
%  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
%     cual cada elemento contiene la tranformación númerica que se le
%     asigna a cada valor de las variables categóricas.
%
%  features: Sruct [1xN] donde cada elemento corresponde a una
%     característica. Sus campos son:
%     - nombre: String con el nombre de la característica.
%     - tipo: String con el tipo de la característica.
%     - nValores: Entero con el número de valores diferentes que puede
%        adoptar cada característica. En el caso de ser continua está
%        vacío.
%     - valores: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x nValores] que contiene los
%        diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
%        de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
%     - columna: Vector [1 x nValores] donde los elementos indican la
%        columna de datosCod que codifica cada valor que puede
%        adoptar la variable categórica.
```

```
%
%Notas:
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [datosCod,nombresCabecera,cabecera,features]=onehot(datos,features)
% Se extrae la columna de los índices
indices=datos(:,1);
datos(:,1)=[];

% Se determinan las dimensiones de la matriz de entradas
[nMuestras,nVariables]=size(datos);

% Se determinan las columnas que corresponden a cada valor de las variables
N=1;
for ii=1:nVariables
   if strcmp(features(ii).tipo,'%s')
      features(ii).columna=N:(features(ii).nValores+N-1);
      N=N+features(ii).nValores;
   elseif strcmp(features(ii).tipo,'%f')
      features(ii).columna=[];
   end
end
N=N-1;

% Se iniciliza la matriz de datosCod
datosCod=-1*ones(nMuestras,N);

% Para los valores de cada variable se construye la matriz de datos codificados
for ii=1:nVariables
   % Si la variable es categórica
   if strcmp(features(ii).tipo,'%s')
      % Se construye la codificación para dicha variable
      datosCod(:,features(ii).columna)=full(ind2vec(datos(:,ii)',features(ii).nValores)↩
          )';
   end

end

% Se añade el índice de cada muestra como primera columna
datosCod=[indices datosCod];

% Se actualiza el campo columna de la estructura features debido a la
% inclusión del índice
for iVariable=1:nVariables
   features(iVariable).columna=features(iVariable).columna+1;
end

% Se procede a calcular las salidas cabecera y nombresCabecera
nombresCabecera=cell(1+N,1);
cabecera=cell(1,N+1);

% La primera posición corresponde al índice
nombresCabecera{1}={'1 --> Índice'};
cabecera{1}='Int';

% Se completa el resto de variables
for iVariable=1:nVariables
   if strcmp(features(iVariable).tipo,'%s')
      for iValor=1:features(iVariable).nValores
         nombresCabecera{features(iVariable).columna(iValor)}=...
```

```
            sprintf('%d --> is_%s',features(iVariable).columna(iValor),features(iVariable).valores{iValor});
        cabecera{features(iVariable).columna(iValor)}='Bool';
    end
  end
end

end
```

Codificación one-cold

Código A.2 onecold.m.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Este script transforma una base de conocimiento ya procesada por main_cargarDatos
%de tal forma que solo incluye las variables categóricas y les realiza una
%codificación one cold.
%
%Uso:
%  [datosCod,nombresCabecera,cabecera,features]=onehot(datos,features)
%
%Entradas:
%  datos: Matriz de datos de las muestras dadas por el preprocesamiento
%     realizado en main_cargarDatos.m
%
%  features: Estructura con la información de cada variable dada por el
%     preprocesamiento realizado en main_cargarDatos.m
%
%Salidas:
%  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
%    como número de valores diferentes haya entre todas las variables
%    categóricas. Más una columna para los índices.
%
%  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
%     la variable de cada columna de la matriz datosCod.
%
%  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
%     cual cada elemento contiene la tranformación númerica que se le
%     asigna a cada valor de las variables categóricas.
%
%  features: Sruct [1xN] donde cada elemento corresponde a una
%     característica. Sus campos son:
%     - nombre: String con el nombre de la característica.
%     - tipo: String con el tipo de la característica.
%     - nValores: Entero con el número de valores diferentes que puede
%        adoptar cada característica. En el caso de ser continua está
%        vacío.
%     - valores: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x nValores] que contiene los
%        diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
%        de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
```

```
47  %     - columna: Vector [1 x nValores] donde los elementos indican la
48  %        columna de datosCod que codifica cada valor que puede
49  %        adoptar la variable categórica.
50  %
51  %Notas:
52  %
53  %
54  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55
56  function [datosCod,nombresCabecera,cabecera,features]=onecold(datos,features)
57  % Se extrae la columna de los índices
58  indices=datos(:,1);
59  datos(:,1)=[];
60
61  % Se determinan las dimensiones de la matriz de entradas
62  [nMuestras,nVariables]=size(datos);
63
64  % Se determinan las columnas que corresponden a cada valor de las variables
65  N=1;
66  for ii=1:nVariables
67     if strcmp(features(ii).tipo,'%s')
68        features(ii).columna=N:(features(ii).nValores+N-1);
69        N=N+features(ii).nValores;
70     elseif strcmp(features(ii).tipo,'%f')
71        features(ii).columna=[];
72     end
73  end
74  N=N-1;
75
76  % Se iniciliza la matriz de datosCod
77  datosCod=-1*ones(nMuestras,N);
78
79  % Para los valores de cada variable se construye la matriz de datos codificados
80  for ii=1:nVariables
81     % Si la variable es categórica
82     if strcmp(features(ii).tipo,'%s')
83        % Se construye la codificación para dicha variable
84        datosCod(:,features(ii).columna)=abs(full(ind2vec(datos(:,ii)',features(ii).↩
               nValores))'-1);
85     end
86
87  end
88
89  % Se añade el índice de cada muestra como primera columna
90  datosCod=[indices datosCod];
91
92  % Se actualiza el campo columna de la estructura features debido a la
93  % inclusión del índice
94  for iVariable=1:nVariables
95     features(iVariable).columna=features(iVariable).columna+1;
96  end
97
98  % Se procede a calcular las salidas cabecera y nombresCabecera
99  nombresCabecera=cell(1+N,1);
100 cabecera=cell(1,N+1);
101
102 % La primera posición corresponde al índice
103 nombresCabecera{1}={'1 --> Índice'};
104 cabecera{1}='Int';
105
106 % Se completa el resto de variables
107 for iVariable=1:nVariables
```

```
108     if strcmp(features(iVariable).tipo,'%s')
109        for iValor=1:features(iVariable).nValores
110           nombresCabecera{features(iVariable).columna(iValor)}=...
111              sprintf('%d ---> notis_%s',features(iVariable).columna(iVariable).valores{iValor});
112           cabecera{features(iVariable).columna(iVariable).columna(iValor)}='Bool';
113        end
114     end
115  end
116
117  end
```

Codificación rank-hot

Código A.3 rankhot.m.

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % (c) Copyright 2009. GAMCO S.L.
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  % Proyecto: Investigación
5  % Módulo:
6  % Autor/es: Adrián Rocha
7  % Fecha: 29 de noviembre de 2019
8  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9  % Modificaciones (autor, fecha, modificaciones)
10 % <version>, <Autor>, <fecha>, <modificaciones>
11 %
12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 %
14 %Este script transforma una base de conocimiento ya procesada por main_cargarDatos
15 %de tal forma que solo incluye las variables categóricas y les realiza una
16 %codificación rank hot.
17 %
18 %Uso:
19 %  [datosCod,nombresCabecera,cabecera,features]=rankhot(datos,features)
20 %
21 %Entradas:
22 %  datos: Matriz de datos de las muestras dadas por el preprocesamiento
23 %     realizado en main_cargarDatos.m
24 %
25 %  features: Estructura con la información de cada variable dada por el
26 %     preprocesamiento realizado en main_cargarDatos.m
27 %
28 %Salidas:
29 %  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
30 %    como número de valores diferentes haya entre todas las variables
31 %    categóricas. Más una columna para los índices.
32 %
33 %  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
34 %     la variable de cada columna de la matriz datosCod.
35 %
36 %  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
37 %     cual cada elemento contiene la tranformación númerica que se le
38 %     asigna a cada valor de las variables categóricas.
39 %
40 %  features: Sruct [1xN] donde cada elemento corresponde a una
41 %     característica. Sus campos son:
42 %     - nombre: String con el nombre de la característica.
43 %     - tipo: String con el tipo de la característica.
```

```
%     - nValores: Entero con el número de valores diferentes que puede
%       adoptar cada característica. En el caso de ser continua está
%       vacío.
%     - valores: Para las continuas está vacío, pero en las
%       categóricas es un cell [1 x nValores] que contiene los
%       diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%       categóricas es un vector [1 x nValores] que contiene los
%       enteros usados para sustituir los diferentes valores que puede
%       adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
%       categóricas es un vector [1 x nValores] que contiene el número
%       de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
%     - columna: Vector [1 x nValores] donde los elementos indican la
%       columna de datosCod que codifica cada valor que puede
%       adoptar la variable categórica.
%
%Notas:
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [datosCod,nombresCabecera,cabecera,features]=rankhot(datos,features)
% Se extrae la columna de los índices
indices=datos(:,1);
datos(:,1)=[];

% Se determinan las dimensiones de la matriz de entradas
[nMuestras,nVariables]=size(datos);

% Se determinan las columnas que corresponden a cada valor de las variables
N=1;
for ii=1:nVariables
   if strcmp(features(ii).tipo,'%s')
      features(ii).columna=N:(features(ii).nValores+N-2);
      N=N+features(ii).nValores-1;
   elseif strcmp(features(ii).tipo,'%f')
      features(ii).columna=[];
   end
end
N=N-1;

% Se iniciliza la matriz de datosCod
datosCod=-1*ones(nMuestras,N);

% Para los valores de cada variable se construye la matriz de datos codificados
for ii=1:nVariables
   % Si la variable es categórica
   if strcmp(features(ii).tipo,'%s')
      % Se construye la codificación para dicha variable
      auxCod = zeros(nMuestras,features(ii).nValores);

      for iValor = features(ii).representacion
         indicesValor=(datos(:,ii)==iValor);
         auxCod(indicesValor,1:iValor)=1;
      end

      datosCod(:,features(ii).columna)= auxCod(:,2:end);
   end

end
```

```
106
107 % Se añade el índice de cada muestra como primera columna
108 datosCod=[indices datosCod];
109
110 % Se actualiza el campo columna de la estructura features debido a la
111 % inclusión del índice
112 for iVariable=1:nVariables
113    features(iVariable).columna=features(iVariable).columna+1;
114 end
115
116 % Se procede a calcular las salidas cabecera y nombresCabecera
117 nombresCabecera=cell(1+N,1);
118 cabecera=cell(1,N+1);
119
120 % La primera posición corresponde al índice
121 nombresCabecera{1}={'1 ---> Índice'};
122 cabecera{1}='Int';
123
124 % Se completa el resto de variables
125 for iVariable=1:nVariables
126    if strcmp(features(iVariable).tipo,'%s')
127       for iValor=1:features(iVariable).nValores-1
128          nombresCabecera{features(iVariable).columna(iValor)}=...
129             sprintf('%d ---> rhot_%s',features(iVariable).columna(iValor),features(↩
                  iVariable).valores{iValor});
130          cabecera{features(iVariable).columna(iValor)}='Bool';
131       end
132    end
133 end
134
135 end
```

Codificación binaria

Código A.4 binary.m.

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %
3  %Este script transforma un conjunto de datos ya procesados por main_cargarDatos
4  %de tal forma que solo incluye las variables categóricas y les realiza una
5  %codificación binaria.
6  %
7  %Antes de realizar la codificación binaria se le resta 1 a los datos,
8  %puesto que estos empiezan por uno y si no se hiciera se desaprovecharía un
9  %bit.
10 %
11 %Uso:
12 %  [datosCod,nombresCabecera,cabecera,features]=binary(datos,features)
13 %
14 %Entradas:
15 %  datos: Matriz de datos de las muestras dadas por el preprocesamiento
16 %     realizado en main_cargarDatos.m
17 %
18 %  features: Estructura con la información de cada variable dada por el
19 %     preprocesamiento realizado en main_cargarDatos.m
20 %
21 %Salidas:
22 %  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
23 %    como número de bits sean necesarios para la codificación binaria e
```

```
%    independiente de los valores que puede adoptar cada variable. Más
%    una columna para los índices.
%
%  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
%     la variable de cada columna de la matriz datosCod.
%
%  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
%     cual cada elemento contiene la tranformación númerica que se le
%     asigna a cada valor de las variables categóricas.
%
%  features: Sruct [1xN] donde cada elemento corresponde a una
%     característica. Sus campos son:
%     - nombre: String con el nombre de la característica.
%     - tipo: String con el tipo de la característica.
%     - nValores: Entero con el número de valores diferentes que puede
%        adoptar cada característica. En el caso de ser continua está
%        vacío.
%     - valores: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x nValores] que contiene los
%        diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
%        de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
%     - columna: Vector [1 x ceil(log2(nValores))] donde los elementos
%        indican las columnas de datosCod que codifican los valores de
%        la variable en cuestión.
%     - representacionBinaria: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene enteros
%        cuya representación en binario sustituye los diferentes valores
%        que puede adoptar la característica.
%
%Notas:
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [datosCod,nombresCabecera,cabecera,features]=binary(datos,features)
% Se extrae la columna de los índices
indices=datos(:,1);
datos(:,1)=[];

% Se determinan las dimensiones de la matriz de entradas
[nMuestras,nVariables]=size(datos);

% Se determinan las columnas que corresponden a cada valor de las variables
% También se añade la representación en decimal cuya representación binaria
% se usará para cada valor de las variables
N=1;
for ii=1:nVariables

   if strcmp(features(ii).tipo,'%s')
      nColumnas=ceil(log2(features(ii).nValores));
      features(ii).columna=N:(nColumnas+N-1);
      N=N+nColumnas;
      features(ii).representacionBinaria=features(ii).representacion-1;
   elseif strcmp(features(ii).tipo,'%f')
```

```
86          features(ii).columna=[];
87          features(ii).representacionBinaria=[];
88       end
89
90    end
91    N=N-1;
92
93    % Se iniciliza la matriz de datosCod
94    datosCod=-1*ones(nMuestras,N);
95
96    % Con los valores de cada variable se construye la matriz de datos codificados
97    for ii=1:nVariables
98       % Si la variable es categórica
99       if strcmp(features(ii).tipo,'%s')
100         % Se transforman los valores de la variable a binario
101         auxCod=dec2bin(datos(:,ii)-1);
102
103         % Cada columna de la matriz resultado se contruye con un bit
104         % resultante de la transformación. Se comienza por el más
105         % significativo.
106         for iBit=1:size(auxCod,2)
107            datosCod(:,features(ii).columna(iBit))=str2num(auxCod(:,iBit));
108            % str2double no tiene el comportamiento deseado
109         end
110
111      end
112
113   end
114
115   % Se añade el índice de cada muestra como primera columna
116   datosCod=[indices datosCod];
117
118   % Se actualiza el campo columna de la estructura features debido a la
119   % inclusión del índice
120   for iVariable=1:nVariables
121
122      features(iVariable).columna=features(iVariable).columna+1;
123
124   end
125
126   % Se procede a calcular las salidas cabecera y nombresCabecera
127   nombresCabecera=cell(1+N,1);
128   cabecera=cell(1,N+1);
129
130   % La primera posición corresponde al índice
131   nombresCabecera{1}={'1 --> Índice'};
132   cabecera{1}='Int';
133
134   % Se completan cabecera y nombresCabecera para el resto de variables
135   for iVariable=1:nVariables
136      % Si la variable es categórica
137      if strcmp(features(iVariable).tipo,'%s')
138
139         % iBit es un índice del bit en la codificación de cada variable
140         iBit=1;
141         for iCol=features(iVariable).columna
142            % Se completan ambas salidas para cada columna resultante de
143            % la codificación de la variable
144            nombresCabecera{iCol}=sprintf('%d --> %dbit_%s',iCol,iBit,features(iVariable)↩
                   .nombre);
145            cabecera{iCol}='Bool';
146
```

```
147             % Se incrementa el índice para el siguiente bit de esa misma
148             % variable
149             iBit=iBit+1;
150         end
151
152     end
153
154 end
155
156 end
```

Código Gray

Código A.5 gray.m.

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %
3  %Este script transforma un conjunto de datos ya procesados por main_cargarDatos
4  %de tal forma que solo incluye las variables categóricas y les realiza una
5  %codificación binaria gray.
6  %
7  %Antes de realizar la codificación se le resta 1 a los datos,
8  %puesto que estos empiezan por uno y si no se hiciera se desaprovecharía un
9  %bit.
10 %
11 %Uso:
12 %  [datosCod,nombresCabecera,cabecera,features]=gray(datos,features)
13 %
14 %Entradas:
15 %  datos: Matriz de datos de las muestras dadas por el preprocesamiento
16 %     realizado en main_cargarDatos.m
17 %
18 %  features: Estructura con la información de cada variable dada por el
19 %     preprocesamiento realizado en main_cargarDatos.m
20 %
21 %Salidas:
22 %  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
23 %    como número de bits sean necesarios para la codificación binaria e
24 %    independiente de los valores que puede adoptar cada variable. Más
25 %    una columna para los índices.
26 %
27 %  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
28 %     la variable de cada columna de la matriz datosCod.
29 %
30 %  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
31 %     cual cada elemento contiene la tranformación númerica que se le
32 %     asigna a cada valor de las variables categóricas.
33 %
34 %  features: Sruct [1xN] donde cada elemento corresponde a una
35 %     característica. Sus campos son:
36 %     - nombre: String con el nombre de la característica.
37 %     - tipo: String con el tipo de la característica.
38 %     - nValores: Entero con el número de valores diferentes que puede
39 %        adoptar cada característica. En el caso de ser continua está
40 %        vacío.
41 %     - valores: Para las continuas está vacío, pero en las
42 %        categóricas es un cell [1 x nValores] que contiene los
43 %        diferentes valores que puede adoptar la característica.
44 %     - representacion: Para las continuas está vacío, pero en las
```

```
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
%        de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
%     - columna: Vector [1 x ceil(log2(nValores))] donde los elementos
%        indican las columnas de datosCod que codifican los valores de
%        la variable en cuestión.
%     - representacionBinaria: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene enteros
%        cuya representación en gray sustituye los diferentes valores
%        que puede adoptar la característica.
%
%Notas:
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [datosCod,nombresCabecera,cabecera,features]=gray(datos,features)
% Se extrae la columna de los índices
indices=datos(:,1);
datos(:,1)=[];

% Se determinan las dimensiones de la matriz de entradas
[nMuestras,nVariables]=size(datos);

% Se determinan las columnas que corresponden a cada valor de las variables
% También se añade la representación en decimal cuya representación binaria
% se usará para cada valor de las variables
N=1;
for ii=1:nVariables
   if strcmp(features(ii).tipo,'%s')
      nColumnas=ceil(log2(features(ii).nValores));
      features(ii).columna=N:(nColumnas+N-1);
      N=N+nColumnas;
      features(ii).representacionBinaria=features(ii).representacion-1;
   elseif strcmp(features(ii).tipo,'%f')
      features(ii).columna=[];
      features(ii).representacionBinaria=[];
   end
end
N=N-1;

% Se iniciliza la matriz de datosCod
datosCod=-1*ones(nMuestras,N);

% Con los valores de cada variable se construye la matriz de datos codificados
for ii=1:nVariables
   % Si la variable es categórica
   if strcmp(features(ii).tipo,'%s')
      % Se transforman los valores de la variable a binario
      auxCod=dec2bin(datos(:,ii)-1);
      % Se transforman los valores binarios a gray
      auxCod = bin2gray(auxCod);

      % Cada columna de la matriz resultado se contruye con un bit
      % resultante de la transformación. Se comienza por el más
      % significativo.
      for iBit=1:size(auxCod,2)
         datosCod(:,features(ii).columna(iBit))=auxCod(:,iBit);
```

```
107         end
108      end
109
110   end
111
112   % Se añade el índice de cada muestra como primera columna
113   datosCod=[indices datosCod];
114
115   % Se actualiza el campo columna de la estructura features debido a la
116   % inclusión del índice
117   for iVariable=1:nVariables
118      features(iVariable).columna=features(iVariable).columna+1;
119   end
120
121   % Se procede a calcular las salidas cabecera y nombresCabecera
122   nombresCabecera=cell(1+N,1);
123   cabecera=cell(1,N+1);
124
125   % La primera posición corresponde al índice
126   nombresCabecera{1}={'1 --> Índice'};
127   cabecera{1}='Int';
128
129   % Se completan cabecera y nombresCabecera para el resto de variables
130   for iVariable=1:nVariables
131      % Si la variable es categórica
132      if strcmp(features(iVariable).tipo,'%s')
133
134         % iBit es un índice del bit en la codificación de cada variable
135         iBit=1;
136         for iCol=features(iVariable).columna
137            % Se completan ambas salidas para cada columna resultante de
138            % la codificación de la variable
139            nombresCabecera{iCol}=sprintf(...
140               '%d --> %dgray_%s',iCol,iBit,features(iVariable).nombre);
141            cabecera{iCol}='Bool';
142
143            % Se incrementa el índice para el siguiente bit de esa misma
144            % variable
145            iBit=iBit+1;
146         end
147
148      end
149   end
150
151   end
152
153   function gray = bin2gray(binstr)
154   % Transforma un vector fila de caracteres representando un número
155   % binario a un vector numérico con el mismo número pero en gray
156   % Si la entrada es una matriz de string, considera cada columna como un
157   % bit y cada fila como un número diferente.
158   bin=zeros(size(binstr));
159   for i=1:size(binstr,2)
160      bin(:,i)=str2num(binstr(:,i));
161   end
162   gray = bitxor(bin(:,2:end),bin(:,1:(end-1)));
163   gray = [bin(:,1) gray];
164
165   end
```

Codificación basada en la frecuencia

Código A.6 frequency.m.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Este script transforma un conjunto de datos ya procesados por main_cargarDatos
%de tal forma que solo incluye las variables categóricas y les realiza una
%codificación basada en la frecuencia relativa de cada categoría.
%
%Uso:
%  [datosCod,nombresCabecera,cabecera,features]=frequency(datos,features)
%
%Entradas:
%  datos: Matriz de datos de las muestras dadas por el preprocesamiento
%     realizado en main_cargarDatos.m
%
%  features: Estructura con la información de cada variable dada por el
%     preprocesamiento realizado en main_cargarDatos.m
%
%Salidas:
%  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
%    como variables categóricas exitan en la matriz de entrada datos. Más
%    una columna para los índices.
%
%  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
%     la variable de cada columna de la matriz datosCod.
%
%  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
%     cual cada elemento contiene la transformación numérica que se le
%     asigna a cada valor de las variables categóricas.
%
%  features: Sruct [1xN] donde cada elemento corresponde a una
%     característica. Sus campos son:
%     - nombre: String con el nombre de la característica.
%     - tipo: String con el tipo de la característica.
%     - nValores: Entero con el número de valores diferentes que puede
%        adoptar cada característica. En el caso de ser continua está
%        vacío.
%     - valores: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x nValores] que contiene los
%        diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
%        de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
%     - columna: Entero que indica la columna de datosCod que contiene la
%        codificación de la variable categórica. Para las continuas está
%        vacío.
%Notas:
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [datosCod,nombresCabecera,cabecera,features]=frequency(datos,features)
```

```
% Se extrae la columna de los índices
indices=datos(:,1);
datos(:,1)=[];

% Se determinan las dimensiones de la matriz de entradas
[nMuestras,nVariables]=size(datos);

% Se determina la columna de la matriz de salida datosCod que tendrá la
% codificación de cada variable categórica original.
N=1;
for iVar=1:nVariables
   if strcmp(features(iVar).tipo,'%s')
      features(iVar).columna=N;
      N=N+1;
   elseif strcmp(features(iVar).tipo,'%f')
      features(iVar).columna=[];
   end
end
N=N-1;

% Se iniciliza la matriz de datosCod
datosCod=-1*ones(nMuestras,N);

% Se pasa a realizar el cálculo de la codificación
% Para cada una de las variables
for iVar=1:nVariables
   % Si la variable es categórica
   if strcmp(features(iVar).tipo,'%s')

      % Para cada uno de los valores que puede adoptar la variable
      for iValor=1:features(iVar).nValores

         % Se determina el índice de las muestras con dicho valor
         iMuestrasValor=datos(:,iVar)==features(iVar).representacion(iValor);

         % Se atualiza la salida datosCod haciendo uso de la
         % frecuencia relativa de aparición de dicho valor
         datosCod(iMuestrasValor,features(iVar).columna)=features(iVar).frecuenciaRel(↩
             iValor);
      end

   end
end

% Se añade el índice de cada muestra como primera columna
datosCod=[indices datosCod];

% Se actualiza el campo columna de la estructura features debido a la
% inclusión del índice
for iVar=1:nVariables
   features(iVar).columna=features(iVar).columna+1;
end

% Se procede a calcular las salidas cabecera y nombresCabecera
nombresCabecera=cell(1+N,1);
cabecera=cell(1,N+1);

% La primera posición corresponde al índice
nombresCabecera{1}={'1 --> Índice'};
cabecera{1}='Int';

% Se completan cabecera y nombresCabecera para el resto de variables
```

```
for iVar=1:nVariables
   % Si la variable es categórica
   if strcmp(features(iVar).tipo,'%s')
      % Se inicializa cada componente de la cabecera con un cell de
      % tamaño igual al número de valores
      col=features(iVar).columna;
      cabecera{col}=cell(features(iVar).nValores,1);

      % En cada componente del cell creado se indica la transformación
      % numérica llevada a cabo
      for iValor=1:features(iVar).nValores
         cabecera{col}{iValor}=[features(iVar).valores{iValor} ' --> ' num2str(
             features(iVar).frecuenciaRel(iValor))];
      end

      % Se completa nombresCabecera con un nombre para cada nueva
      % variable
      nombresCabecera{col}=[num2str(col) ' --> freq_' features(iVar).nombre];

   end
end


end
```

A.1.2 En clasificación

Codificación mediante entity embeddings

Código A.7 entityEmbedding.m.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Este script transforma un conjunto de datos ya procesados por main_cargarDatos
%de tal forma que solo incluye las variables categóricas y les realiza una
%codificación basada en entity embedding
%
%Para cada una de las variables, el tamaño de la capa oculta y, por tanto,
%el de variables resultantes para codificar la original, será indicado en
%la entrada hiddenLayerSizes. Si esta entrada está vacía, se usará como
%tamaño el resultado de redondear la semisuma de 1 más el número de
%categorías de la variable.
%
%Uso:
%
%  [datosCod,nombresCabecera,cabecera,features]=entityEmbedding(datos,features,
    hiddenLayerSizes,idCod)
%
%Entradas:
%  datos: Matriz de datos de las muestras dadas por el preprocesamiento
%     realizado en main_cargarDatos.m
%
%  features: Estructura con la información de cada variable dada por el
%     preprocesamiento realizado en main_cargarDatos.m
%
%  hiddenLayerSizes: Vector de enteros de tamaño igual al número de
%     variables categóricas que indica el tamaño de la capa oculta de la
```

```
26 %     red usada para codificar cada una de las variables. Si está vacío
27 %     se calcula mediante la expresión previa.
28 %
29 %Salidas:
30 %  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
31 %    como neuronas totales se hayan usado en la capa oculta de todas las
32 %    variables categóricas. Más una columna para los índices.
33 %
34 %  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
35 %     la variable de cada columna de la matriz datosCod.
36 %
37 %  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
38 %     cual cada elemento contiene la transformación numérica que se le
39 %     asigna a cada valor de las variables categóricas.
40 %
41 %  features: Sruct [1xN] donde cada elemento corresponde a una
42 %     característica. Sus campos son:
43 %     - nombre: String con el nombre de la característica.
44 %     - tipo: String con el tipo de la característica.
45 %     - nValores: Entero con el número de valores diferentes que puede
46 %        adoptar cada característica. En el caso de ser continua está
47 %        vacío.
48 %     - valores: Para las continuas está vacío, pero en las
49 %        categóricas es un cell [1 x nValores] que contiene los
50 %        diferentes valores que puede adoptar la característica.
51 %     - representacion: Para las continuas está vacío, pero en las
52 %        categóricas es un vector [1 x nValores] que contiene los
53 %        enteros usados para sustituir los diferentes valores que puede
54 %        adoptar la característica.
55 %     - frecuencias: Para las continuas está vacío, pero en las
56 %        categóricas es un vector [1 x nValores] que contiene el número
57 %        de veces que aparece cada valor.
58 %     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
59 %     - columna: Entero que indica las columnas de datosCod que contienen la
60 %        codificación de la variable categórica. Para las continuas está
61 %        vacío.
62 %     - codificacion: Para las continuas está vacío, pero en las
63 %        categóricas es un cell [1 x length(columna)] que contiene los
64 %        números con los que se sustituye cada posible valor de la variable.
65 %
66 %Notas:
67 %
68 %
69 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
70
71 function [datosCod,nombresCabecera,cabecera,features]=entityEmbedding(datos,features,↩
       hiddenLayerSizes,idCod)
72
73 % Se extrae la columna de los índices
74 indices=datos(:,1);
75 datos(:,1)=[];
76
77 % Se determinan las dimensiones de la matriz de entradas
78 [nMuestras,nVariables]=size(datos);
79
80 % Se determinan las columnas de la matriz de salida datosCod que tendrán la
81 % codificación de cada variable categórica original.
82 % El número de columnas esta especificado por hiddenLayerSizes.
83 % Si este vector está vacío se usa el número medio entre el tamaño de la
84 %  capa de entrada y de salida.
85 N=0;
86 for iVar=1:nVariables
```

```
87      if strcmp(features(iVar).tipo,'%s')
88         if ~isempty(hiddenLayerSizes)
89            tam=hiddenLayerSizes(1);
90            hiddenLayerSizes(1)=[];
91         else
92            tam=round((1+features(iVar).nValores)/2);
93         end
94         features(iVar).columna=(1:tam)+N;
95         N=features(iVar).columna(end);
96      elseif strcmp(features(iVar).tipo,'%f')
97         features(iVar).columna=[];
98      end
99   end
100
101  % Se iniciliza la matriz de datosCod
102  datosCod=-1*ones(nMuestras,N);
103
104  % Se realiza la codificación one-hot de la salida
105  t=full(ind2vec(datos(:,end)',features(end).nValores));
106
107  % Se pasa a realizar el cálculo de la codificación
108  % Para cada una de las variables
109  for iVar=1:nVariables
110     % Si la variable es categórica
111     if strcmp(features(iVar).tipo,'%s')
112        % Se inicializa el campo de codificacion
113        features(iVar).codificacion=cell(1,features(iVar).nValores);
114
115        % Se realiza la codificación one-hot de la variable
116        x=full(ind2vec(datos(:,iVar)',features(iVar).nValores));
117
118        % Se inicializa la red con el tamaño de la capa oculta adecuado
119        net = patternnet(length(features(iVar).columna), 'trainscg');
120
121        % Se determina la división de los datos
122        net.divideParam.trainRatio = 70/100;
123        net.divideParam.valRatio = 15/100;
124        net.divideParam.testRatio = 15/100;
125
126        % Se entrena la red
127        net = train(net,x,t);
128
129        % Se recogen los pesos necesarios para la codificación
130        weights=net.IW{1}';
131        clear net x
132
133        % Para cada una de las categorías de la variable
134        for iValor=1:features(iVar).nValores
135           % Se rellena el campo codificacion de la estructura
136           features(iVar).codificacion{iValor}=weights(iValor,:);
137
138           % Se completa la salida datosCod
139           % Se buscan las muestras de cada categoría y se rellena cada
140           %  una de las columnas
141           indices_valor=features(iVar).representacion(iValor)==datos(:,iVar);
142           for iCol=1:length(features(iVar).columna)
143              datosCod(indices_valor,features(iVar).columna(iCol))=features(iVar).↩
                      codificacion{iValor}(iCol);
144           end
145        end
146
147     % Si la variable no es categórica
```

```
148     elseif strcmp(features(iVar).tipo,'%f')
149        % El campo codificación estará vacío
150        features(iVar).codificacion=[];
151     end
152  end
153
154  % Se libera memoria
155  clear indices_valor weights t datos
156
157  % Se añade el índice de cada muestra como primera columna
158  datosCod=[indices datosCod];
159  clear indices
160
161  % Se actualiza el campo columna de la estructura features debido a la
162  % inclusión del índice
163  for iVar=1:nVariables
164     features(iVar).columna=features(iVar).columna+1;
165  end
166
167  % Se procede a calcular las salidas cabecera y nombresCabecera
168  nombresCabecera=cell(1+N,1);
169  cabecera=cell(1,N+1);
170
171  % La primera posición corresponde al índice
172  nombresCabecera{1}={'1 --> Índice'};
173  cabecera{1}='Int';
174
175  % Se completan cabecera y nombresCabecera para el resto de variables
176  for iVar=1:nVariables
177     % Si la variable es categórica
178     if strcmp(features(iVar).tipo,'%s')
179        % Para cada columna de la matriz datosCod
180        for iCol=1:length(features(iVar).columna)
181           col=features(iVar).columna(iCol);
182           cabecera{col}=cell(features(iVar).nValores,1);
183
184           % Se describe el valor que adopta cada categoría
185           for iValor=1:features(iVar).nValores
186              cabecera{col}{iValor}=[features(iVar).valores{iValor} ' --> ' num2str(↩
                    features(iVar).codificacion{iValor}(iCol))];
187           end
188
189           % Se completa nombresCabecera con un nombre para cada nueva
190           % variable
191           nombresCabecera{col}=[num2str(col) ' --> ee' num2str(idCod) '_' num2str(iCol)↩
                  'w_' features(iVar).nombre];
192        end
193
194     end
195  end
196
197
198  end
```

Codificación basada en estadísticas de la variable de salida dicotómica

Código A.8 probabilityBinaryTarget.m.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Este script transforma un conjunto de datos ya procesados por main_cargarDatos
%de tal forma que solo incluye las variables categóricas y les realiza una
%codificación basada en la probabilidad a priori y posteriori de la salida
%condicionada por el valor de la variable categórica.
%
%Uso:
%  [datosCod,nombresCabecera,cabecera,features]=probabilityBinaryTarget(datos,features,↩
    salida1)
%
%Entradas:
%  datos: Matriz de datos de las muestras dadas por el preprocesamiento
%     realizado en main_cargarDatos.m
%
%  features: Estructura con la información de cada variable dada por el
%     preprocesamiento realizado en main_cargarDatos.m
%
%  salida1: Valor de la variable de salida que se considera para el
%     cálculo de las probabilidades.
%
%  lambda: Función anónima que se usará para el cálculo de los pesos de
%     las probabilidades a priori y a posteriori. Su única variable de
%     entrada es el número de veces que aparece un determinado valor de
%     una variable categórica.
%
%  idCod: Entero usado para ser añadido en nombresCabecera ya que la
%     codificación no es única (depende de la función lambda).
%
%Salidas:
%  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
%    como variables categóricas exitan en la matriz de entrada datos. Más
%    una columna para los índices.
%
%  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
%     la variable de cada columna de la matriz datosCod.
%
%  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
%     cual cada elemento contiene la tranformación númerica que se le
%     asigna a cada valor de las variables categóricas.
%
%  features: Sruct [1xN] donde cada elemento corresponde a una
%     característica. Sus campos son:
%     - nombre: String con el nombre de la característica.
%     - tipo: String con el tipo de la característica.
%     - nValores: Entero con el número de valores diferentes que puede
%        adoptar cada característica. En el caso de ser continua está
%        vacío.
%     - valores: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x nValores] que contiene los
%        diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
```

```
56 %        categóricas es un vector [1 x nValores] que contiene el número
57 %        de veces que aparece cada valor.
58 %     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
59 %     - columna: Entero que indica la columna de datosCod que contiene la
60 %        codificación de la variable categórica. Para las continuas está
61 %        vacío.
62 %     - codificacion: Para las continuas está vacío, pero en las
63 %        categóricas es un vector [1 x nValores] que contiene el número
64 %        con el que se sustituye cada posible valor de la variable.
65 %
66 %Notas:
67 %
68 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
69
70 function [datosCod,nombresCabecera,cabecera,features]=probabilityBinaryTarget(datos,↩
       features,salida1,lambda,idCod)
71 % Se extrae la columna de los índices
72 indices=datos(:,1);
73 datos(:,1)=[];
74
75 % Se determinan las dimensiones de la matriz de entradas
76 [nMuestras,nVariables]=size(datos);
77
78 % Se determina la columna de la matriz de salida datosCod que tendrá la
79 % codificación de cada variable categórica original.
80 N=1;
81 for iVar=1:nVariables
82    if strcmp(features(iVar).tipo,'%s')
83       features(iVar).columna=N;
84       N=N+1;
85    elseif strcmp(features(iVar).tipo,'%f')
86       features(iVar).columna=[];
87    end
88 end
89 N=N-1;
90
91 % Se iniciliza la matriz de datosCod
92 datosCod=-1*ones(nMuestras,N);
93
94 % Se determinan los índices de las muestras con salida igual a salida1
95 iSalida1=datos(:,end)==salida1;
96
97 % Se determina la probabilidad a priori
98 prioriP=sum(iSalida1)/nMuestras;
99
100 % Se pasa a realizar el cálculo de la codificación
101 % Para cada una de las variables
102 for iVar=1:nVariables
103    % Si la variable es categórica
104    if strcmp(features(iVar).tipo,'%s')
105       % Se inicializa el campo de codificacion
106       features(iVar).codificacion=-1*ones(1,features(iVar).nValores);
107
108       % Para cada uno de los valores que puede adoptar la variable
109       for iValor=1:features(iVar).nValores
110          valor=features(iVar).representacion(iValor);
111
112          % Se determina el índice de las muestras con dicho valor
113          iMuestrasValor=datos(:,iVar)==valor;
114          % Se calcula la probabilidad a posteriori
115          posteriorP=sum(and(iMuestrasValor,iSalida1))/sum(iMuestrasValor);
116          % Se calcula la codificación mediante el uso de las dos
```

```
117          % probabilidades y la función lambda
118          cod=posteriorP*lambda(sum(iMuestrasValor)) + prioriP*(1-lambda(sum(↩
                 iMuestrasValor)));
119          % Se actualiza el campo codificacion para dicho valor
120          features(iVar).codificacion(iValor)=cod;
121
122          % Se atualiza la salida datosCod haciendo uso de la
123          % codificación calculada
124          datosCod(iMuestrasValor,features(iVar).columna)=cod;
125       end
126
127    % Si la variable no es categórica
128    elseif strcmp(features(iVar).tipo,'%f')
129       % El campo codificación estará vacío
130       features(iVar).codificacion=[];
131    end
132 end
133
134 % Se añade el índice de cada muestra como primera columna
135 datosCod=[indices datosCod];
136
137 % Se actualiza el campo columna de la estructura features debido a la
138 % inclusión del índice
139 for iVar=1:nVariables
140    features(iVar).columna=features(iVar).columna+1;
141 end
142
143 % Se procede a calcular las salidas cabecera y nombresCabecera
144 nombresCabecera=cell(1+N,1);
145 cabecera=cell(1,N+1);
146
147 % La primera posición corresponde al índice
148 nombresCabecera{1}={'1 --> Índice'};
149 cabecera{1}='Int';
150
151 % Se completan cabecera y nombresCabecera para el resto de variables
152 for iVar=1:nVariables
153    % Si la variable es categórica
154    if strcmp(features(iVar).tipo,'%s')
155       % Se inicializa cada componente de la cabecera con un cell de
156       % tamaño igual al número de valores
157       col=features(iVar).columna;
158       cabecera{col}=cell(features(iVar).nValores,1);
159
160       % En cada componente del cell creado se indica la transformación
161       % numérica llevada a cabo
162       for iValor=1:features(iVar).nValores
163          cabecera{col}{iValor}=[features(iVar).valores{iValor} ' --> ' num2str(↩
                 features(iVar).codificacion(iValor))];
164       end
165
166       % Se completa nombresCabecera con un nombre para cada nueva
167       % variable
168       nombresCabecera{col}=[num2str(col) ' --> pbt' num2str(idCod) '_' features(iVar).↩
              nombre];
169
170    end
171 end
172 end
```

Codificación usando el peso de la evidencia

Código A.9 weightOfEvidence.m.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Este script transforma un conjunto de datos ya procesados por main_cargarDatos
%de tal forma que solo incluye las variables categóricas y les realiza una
%codificación basada en Weight of Evidence (WoE).
%
%Uso:
%
%  [datosCod,nombresCabecera,cabecera,features]=weightOfEvidence(datos,features,↩
    salidaGood)
%
%Entradas:
%  datos: Matriz de datos de las muestras dadas por el preprocesamiento
%     realizado en main_cargarDatos.m
%
%  features: Estructura con la información de cada variable dada por el
%     preprocesamiento realizado en main_cargarDatos.m
%
%  salidaGood: Valor de la salida en la matriz datos que se considerará
%     para el cálculo de los valores Goods.
%
%Salidas:
%  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
%    como variables categóricas exitan en la matriz de entrada datos. Más
%    una columna para los índices.
%
%  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
%     la variable de cada columna de la matriz datosCod.
%
%  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
%     cual cada elemento contiene la transformación numérica que se le
%     asigna a cada valor de las variables categóricas.
%
%  features: Sruct [1xN] donde cada elemento corresponde a una
%     característica. Sus campos son:
%     - nombre: String con el nombre de la característica.
%     - tipo: String con el tipo de la característica.
%     - nValores: Entero con el número de valores diferentes que puede
%        adoptar cada característica. En el caso de ser continua está
%        vacío.
%     - valores: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x nValores] que contiene los
%        diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
%        de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
%     - columna: Entero que indica la columna de datosCod que contiene la
%        codificación de la variable categórica. Para las continuas está
%        vacío.
%     - codificacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
```

```
%        con el que se sustituye cada posible valor de la variable.
%
%Notas:
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [datosCod,nombresCabecera,cabecera,features]=weightOfEvidence(datos,features,salidaGood)

   % Se extrae la columna de los índices
   indices=datos(:,1);
   datos(:,1)=[];

   % Se determinan las dimensiones de la matriz de entradas
   [nMuestras,nVariables]=size(datos);

   % Se determina la columna de la matriz de salida datosCod que tendrá la
   % codificación de cada variable categórica original.
   N=1;
   for iVar=1:nVariables
      if strcmp(features(iVar).tipo,'%s')
         features(iVar).columna=N;
         N=N+1;
      elseif strcmp(features(iVar).tipo,'%f')
         features(iVar).columna=[];
      end
   end
   N=N-1;

   % Se iniciliza la matriz de datosCod
   datosCod=-1*ones(nMuestras,N);

   % Se determinan los índices de las muestras con salida igual a salidaGood
   iSalidaGood=datos(:,end)==salidaGood;

   % Se pasa a realizar el cálculo de la codificación
   % Para cada una de las variables
   for iVar=1:nVariables
      % Si la variable es categórica
      if strcmp(features(iVar).tipo,'%s')
         % Se inicializa el campo de codificacion
         features(iVar).codificacion=-1*ones(1,features(iVar).nValores);

         % Para cada uno de los valores que puede adoptar la variable
         for iValor=1:features(iVar).nValores
            valor=features(iVar).representacion(iValor);

            % Se determina el índice de las muestras con dicho valor
            iMuestrasValor=datos(:,iVar)==valor;

            % Se calcula la codificación mediante la expresión de WoE
            % Se le suma 0.000001 para evitar la división entre 0 y
            %  logaritmo = -inf.
            DistrGoods=sum(and(iMuestrasValor,iSalidaGood));
            DistrGoods = DistrGoods./sum(iSalidaGood)+0.00001;
            DistrBads=sum(and(iMuestrasValor,not(iSalidaGood)));
            DistrBads = DistrBads./sum(not(iSalidaGood))+0.000001;
            cod=100*log(DistrGoods/DistrBads);

            % Se actualiza el campo codificacion para dicho valor
```

```
117             features(iVar).codificacion(iValor)=cod;
118 
119             % Se atualiza la salida datosCod haciendo uso de la
120             % codificación calculada
121             datosCod(iMuestrasValor,features(iVar).columna)=cod;
122          end
123 
124       % Si la variable no es categórica
125       elseif strcmp(features(iVar).tipo,'%f')
126          % El campo codificación estará vacío
127          features(iVar).codificacion=[];
128       end
129    end
130 
131    % Se añade el índice de cada muestra como primera columna
132    datosCod=[indices datosCod];
133 
134    % Se actualiza el campo columna de la estructura features debido a la
135    % inclusión del índice
136    for iVar=1:nVariables
137       features(iVar).columna=features(iVar).columna+1;
138    end
139 
140    % Se procede a calcular las salidas cabecera y nombresCabecera
141    nombresCabecera=cell(1+N,1);
142    cabecera=cell(1,N+1);
143 
144    % La primera posición corresponde al índice
145    nombresCabecera{1}={'1 --> Índice'};
146    cabecera{1}='Int';
147 
148    % Se completan cabecera y nombresCabecera para el resto de variables
149    for iVar=1:nVariables
150       % Si la variable es categórica
151       if strcmp(features(iVar).tipo,'%s')
152          % Se inicializa cada componente de la cabecera con un cell de
153          % tamaño igual al número de valores
154          col=features(iVar).columna;
155          cabecera{col}=cell(features(iVar).nValores,1);
156 
157          % En cada componente del cell creado se indica la transformación
158          % numérica llevada a cabo
159          for iValor=1:features(iVar).nValores
160             cabecera{col}{iValor}=[features(iVar).valores{iValor} ' --> ' num2str(↩
                   features(iVar).codificacion(iValor))];
161          end
162 
163 
164          % Se completa nombresCabecera con un nombre para cada nueva
165          % variable
166          nombresCabecera{col}=[num2str(col) ' --> woe_' features(iVar).nombre];
167 
168       end
169 
170    end
171 
172 
173    end
```

Factor de Bayes

Código A.10 probabilityRatio.m.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Este script transforma un conjunto de datos ya procesados por main_cargarDatos
%de tal forma que solo incluye las variables categóricas y les realiza una
%codificación basada en el ratio de probabilidades de valores buenos y malos.
%
%Uso:
%
%  [datosCod,nombresCabecera,cabecera,features]=probabilityRatio(datos,features,↩
    salidaGood)
%
%Entradas:
%  datos: Matriz de datos de las muestras dadas por el preprocesamiento
%     realizado en main_cargarDatos.m
%
%  features: Estructura con la información de cada variable dada por el
%     preprocesamiento realizado en main_cargarDatos.m
%
%  salidaGood: Valor de la salida en la matriz datos que se considerará
%     para el cálculo de los valores Goods.
%
%Salidas:
%  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
%    como variables categóricas exitan en la matriz de entrada datos. Más
%    una columna para los índices.
%
%  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
%     la variable de cada columna de la matriz datosCod.
%
%  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
%     cual cada elemento contiene la transformación numérica que se le
%     asigna a cada valor de las variables categóricas.
%
%  features: Sruct [1xN] donde cada elemento corresponde a una
%     característica. Sus campos son:
%     - nombre: String con el nombre de la característica.
%     - tipo: String con el tipo de la característica.
%     - nValores: Entero con el número de valores diferentes que puede
%        adoptar cada característica. En el caso de ser continua está
%        vacío.
%     - valores: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x nValores] que contiene los
%        diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
%        de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
%     - columna: Entero que indica la columna de datosCod que contiene la
%        codificación de la variable categórica. Para las continuas está
%        vacío.
%     - codificacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
```

```
%        con el que se sustituye cada posible valor de la variable.
%
%Notas:
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [datosCod,nombresCabecera,cabecera,features]=probabilityRatio(datos,features,salidaGood)

   % Se extrae la columna de los índices
   indices=datos(:,1);
   datos(:,1)=[];

   % Se determinan las dimensiones de la matriz de entradas
   [nMuestras,nVariables]=size(datos);

   % Se determina la columna de la matriz de salida datosCod que tendrá la
   % codificación de cada variable categórica original.
   N=1;
   for iVar=1:nVariables
      if strcmp(features(iVar).tipo,'%s')
         features(iVar).columna=N;
         N=N+1;

      elseif strcmp(features(iVar).tipo,'%f')
         features(iVar).columna=[];
      end
   end

   N=N-1;

   % Se iniciliza la matriz de datosCod
   datosCod=-1*ones(nMuestras,N);

   % Se determinan los índices de las muestras con salida igual a salidaGood
   iSalidaGood=datos(:,end)==salidaGood;

   % Se pasa a realizar el cálculo de la codificación
   % Para cada una de las variables
   for iVar=1:nVariables

      % Si la variable es categórica
      if strcmp(features(iVar).tipo,'%s')
         % Se inicializa el campo de codificacion
         features(iVar).codificacion=-1*ones(1,features(iVar).nValores);

         % Para cada uno de los valores que puede adoptar la variable
         for iValor=1:features(iVar).nValores
            valor=features(iVar).representacion(iValor);

            % Se determina el índice de las muestras con dicho valor
            iMuestrasValor=datos(:,iVar)==valor;

            % Se calcula la codificación mediante el ratio de
            % probabilidades.
            % Se le suma 0.000001 para evitar la división entre 0.
            Goods=sum(and(iMuestrasValor,iSalidaGood));
            Goods=Goods./sum(iSalidaGood);
            Bads=sum(and(iMuestrasValor,not(iSalidaGood)));
            Bads=Bads./sum(not(iSalidaGood))+0.000001;
```

```
117             cod=Goods/Bads;
118 
119             % Se actualiza el campo codificacion para dicho valor
120             features(iVar).codificacion(iValor)=cod;
121 
122             % Se atualiza la salida datosCod haciendo uso de la
123             % codificación calculada
124             datosCod(iMuestrasValor,features(iVar).columna)=cod;
125          end
126 
127       % Si la variable no es categórica
128       elseif strcmp(features(iVar).tipo,'%f')
129          % El campo codificación estará vacío
130          features(iVar).codificacion=[];
131       end
132 
133    end
134 
135    % Se añade el índice de cada muestra como primera columna
136    datosCod=[indices datosCod];
137 
138    % Se actualiza el campo columna de la estructura features debido a la
139    % inclusión del índice
140    for iVar=1:nVariables
141       features(iVar).columna=features(iVar).columna+1;
142    end
143 
144    % Se procede a calcular las salidas cabecera y nombresCabecera
145    nombresCabecera=cell(1+N,1);
146    cabecera=cell(1,N+1);
147 
148    % La primera posición corresponde al índice
149    nombresCabecera{1}={'1 --> Índice'};
150    cabecera{1}='Int';
151 
152    % Se completan cabecera y nombresCabecera para el resto de variables
153    for iVar=1:nVariables
154       % Si la variable es categórica
155       if strcmp(features(iVar).tipo,'%s')
156          % Se inicializa cada componente de la cabecera con un cell de
157          % tamaño igual al número de valores
158          col=features(iVar).columna;
159          cabecera{col}=cell(features(iVar).nValores,1);
160 
161          % En cada componente del cell creado se indica la transformación
162          % numérica llevada a cabo
163          for iValor=1:features(iVar).nValores
164             cabecera{col}{iValor}=[features(iVar).valores{iValor} ' --> ' num2str(↩
                   features(iVar).codificacion(iValor))];
165          end
166 
167          % Se completa nombresCabecera con un nombre para cada nueva
168          % variable
169          nombresCabecera{col}=[num2str(col) ' --> pr_' features(iVar).nombre];
170 
171       end
172 
173    end
174 
175 
176    end
```

Codificación por similitud

Código A.11 similarity.m.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Este script transforma una base de conocimiento ya procesada por main_cargarDatos
%de tal forma que solo incluye las variables categóricas y les realiza una
%codificación similarity.
%
%Uso:
%  [datosCod,nombresCabecera,cabecera,features]=similarity(datos,features)
%
%Entradas:
%  datos: Matriz de datos de las muestras dadas por el preprocesamiento
%     realizado en main_cargarDatos.m
%
%  features: Estructura con la información de cada variable dada por el
%     preprocesamiento realizado en main_cargarDatos.m
%
%Salidas:
%  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
%    como número de valores diferentes haya entre todas las variables
%    categóricas. Más una columna para los índices.
%
%  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
%     la variable de cada columna de la matriz datosCod.
%
%  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
%     cual cada elemento contiene la tranformación númerica que se le
%     asigna a cada valor de las variables categóricas.
%
%  features: Sruct [1xN] donde cada elemento corresponde a una
%     característica. Sus campos son:
%     - nombre: String con el nombre de la característica.
%     - tipo: String con el tipo de la característica.
%     - nValores: Entero con el número de valores diferentes que puede
%        adoptar cada característica. En el caso de ser continua está
%        vacío.
%     - valores: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x nValores] que contiene los
%        diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencia: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
%        de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
%     - columna: Vector [1 x nValores] donde los elementos indican la
%        columna de datosCod que codifica cada valor que puede
%        adoptar la variable categórica.
%
%Notas:
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [datosCod,nombresCabecera,cabecera,features]=similarity(datos,features)
```

```
% Se extrae la columna de los índices
indices=datos(:,1);
datos(:,1)=[];

% Se determinan las dimensiones de la matriz de entradas
[nMuestras,nVariables]=size(datos);

% Se determinan las columnas que corresponden a cada valor de las variables
N=1;
for ii=1:nVariables
   if strcmp(features(ii).tipo,'%s')
      features(ii).columna=N:(features(ii).nValores+N-1);
      N=N+features(ii).nValores;
   elseif strcmp(features(ii).tipo,'%f')
      features(ii).columna=[];
   end
end
N=N-1;

% Se iniciliza la matriz de datosCod
datosCod=-1*ones(nMuestras,N);

% Para los valores de cada variable se construye la matriz de datos codificados
for ii=1:nVariables
   % Si la variable es categórica
   if strcmp(features(ii).tipo,'%s')
      % Se construye la codificación para dicha variable
      auxCod = zeros(nMuestras,features(ii).nValores);

      vecSim = zeros(1,features(ii).nValores);
      for iValMuestra = features(ii).representacion
         for iValSim = features(ii).representacion
            if iValMuestra == iValSim
               vecSim(iValSim) = 1;
            else
               vecSim(iValSim) = 1./(1 + log(nMuestras./features(ii).frecuencia(↩
                   iValMuestra))*log(nMuestras./features(ii).frecuencia(iValSim)));
            end
         end
         auxCod(datos(:,ii)==iValMuestra,:) =...
            repmat(vecSim,features(ii).frecuencia(iValMuestra),1);
      end

      datosCod(:,features(ii).columna)= auxCod;
   end

end

% Se añade el índice de cada muestra como primera columna
datosCod=[indices datosCod];

% Se actualiza el campo columna de la estructura features debido a la
% inclusión del índice
for iVariable=1:nVariables
   features(iVariable).columna=features(iVariable).columna+1;
end

% Se procede a calcular las salidas cabecera y nombresCabecera
nombresCabecera=cell(1+N,1);
cabecera=cell(1,N+1);

% La primera posición corresponde al índice
```

```
nombresCabecera{1}={'1 --> Índice'};
cabecera{1}='Int';

% Se completa el resto de variables
for iVariable=1:nVariables
   if strcmp(features(iVariable).tipo,'%s')
      for iValor=1:features(iVariable).nValores
         nombresCabecera{features(iVariable).columna(iValor)}=...
            sprintf('%d --> sim_%s',features(iVariable).columna(iValor),features(↩
                iVariable).valores{iValor});
         cabecera{features(iVariable).columna(iValor)}='Float';
      end
   end
end

end
```

A.1.3 En regresión

Codificación mediante entity embeddings

Código A.12 entityEmbedding.m.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%Este script transforma un conjunto de datos ya procesados por main_cargarDatos
%de tal forma que solo incluye las variables categóricas y les realiza una
%codificación basada en entity embedding
%
%Para cada una de las variables, el tamaño de la capa oculta y, por tanto,
%el de variables resultantes para codificar la original, será indicado en
%la entrada hiddenLayerSizes. Si esta entrada está vacía, se usará como
%tamaño el resultado de redondear la semisuma de 1 más el número de
%categorías de la variable.
%
%Uso:
%
%  [datosCod,nombresCabecera,cabecera,features]=entityEmbedding(datos,features,salida,↩
    hiddenLayerSizes,idCod)
%
%Entradas:
%  datos: Matriz de datos de las muestras dadas por el preprocesamiento
%     realizado en main_cargarDatos.m
%
%  features: Estructura con la información de cada variable dada por el
%     preprocesamiento realizado en main_cargarDatos.m
%
%  salida: Vector columna con la variable de salida para cada una de las muestras
%     en datos. Si no se conoce una componente debe ser NaN.
%
%  hiddenLayerSizes: Vector de enteros de tamaño igual al número de
%     variables categóricas que indica el tamaño de la capa oculta de la
%     red usada para codificar cada una de las variables. Si está vacío
%     se calcula mediante la expresión previa.
%
%  idCod: Entero usado para ser añadido en nombresCabecera ya que la
%     codificación no es única (depende de hiddenLayerSizes).
```

```
%
%Salidas:
%  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
%    como neuronas totales se hayan usado en la capa oculta de todas las
%    variables categóricas. Más una columna para los índices.
%
%  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
%     la variable de cada columna de la matriz datosCod.
%
%  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
%     cual cada elemento contiene la transformación numérica que se le
%     asigna a cada valor de las variables categóricas.
%
%  features: Sruct [1xN] donde cada elemento corresponde a una
%     característica. Sus campos son:
%     - nombre: String con el nombre de la característica.
%     - tipo: String con el tipo de la característica.
%     - nValores: Entero con el número de valores diferentes que puede
%        adoptar cada característica. En el caso de ser continua está
%        vacío.
%     - valores: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x nValores] que contiene los
%        diferentes valores que puede adoptar la característica.
%     - representacion: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene los
%        enteros usados para sustituir los diferentes valores que puede
%        adoptar la característica.
%     - frecuencias: Para las continuas está vacío, pero en las
%        categóricas es un vector [1 x nValores] que contiene el número
%        de veces que aparece cada valor.
%     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
%     - columna: Entero que indica las columnas de datosCod que contienen la
%        codificación de la variable categórica. Para las continuas está
%        vacío.
%     - codificacion: Para las continuas está vacío, pero en las
%        categóricas es un cell [1 x length(columna)] que contiene los
%        números con los que se sustituye cada posible valor de la variable.
%
%Notas:
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [datosCod,nombresCabecera,cabecera,features]=entityEmbedding(datos,features,↩
    salida,hiddenLayerSizes,idCod)

% Se extrae la columna de los índices
indices=datos(:,1);
datos(:,1)=[];

% Se determinan las dimensiones de la matriz de entradas
[nMuestras,nVariables]=size(datos);

% Se determinan las columnas de la matriz de salida datosCod que tendrán la
% codificación de cada variable categórica original.
% El número de columnas esta especificado por hiddenLayerSizes.
% Si este vector está vacío se usa el número medio entre el tamaño de la
%  capa de entrada y de salida.
N=0;
for iVar=1:nVariables
   if strcmp(features(iVar).tipo,'%s')
      if ~isempty(hiddenLayerSizes)
```

```
95          tam=hiddenLayerSizes(1);
96          hiddenLayerSizes(1)=[];
97       else
98          tam=round((1+features(iVar).nValores)/2);
99       end
100      features(iVar).columna=(1:tam)+N;
101      N=features(iVar).columna(end);
102   elseif strcmp(features(iVar).tipo,'%f')
103      features(iVar).columna=[];
104   end
105end
106
107% Se iniciliza la matriz de datosCod
108datosCod=-1*ones(nMuestras,N);
109
110% Se pasa a realizar el cálculo de la codificación
111% Para cada una de las variables
112for iVar=1:nVariables
113   % Si la variable es categórica
114   if strcmp(features(iVar).tipo,'%s')
115      % Se inicializa el campo de codificacion
116      features(iVar).codificacion=cell(1,features(iVar).nValores);
117
118      % Se realiza la codificación one-hot de la variable
119      x=full(ind2vec(datos(:,iVar)',features(iVar).nValores));
120
121      % Se inicializa la red con el tamaño de la capa oculta adecuado
122      net = fitnet(length(features(iVar).columna), 'trainscg');
123
124      % Se determina la división de los datos
125      net.divideParam.trainRatio = 70/100;
126      net.divideParam.valRatio = 15/100;
127      net.divideParam.testRatio = 15/100;
128
129      % Se entrena la red con las muestras con salida conocida
130      net = train(net,x(:,~isnan(salida)),salida(~isnan(salida))');
131
132      % Se recogen los pesos necesarios para la codificación
133      weights=net.IW{1}';
134      clear net x
135
136      % Para cada una de las categorías de la variable
137      for iValor=1:features(iVar).nValores
138         % Se rellena el campo codificacion de la estructura
139         features(iVar).codificacion{iValor}=weights(iValor,:);
140
141         % Se completa la salida datosCod
142         % Se buscan las muestras de cada categoría y se rellena cada
143         %  una de las columnas
144         indices_valor=features(iVar).representacion(iValor)==datos(:,iVar);
145         for iCol=1:length(features(iVar).columna)
146            datosCod(indices_valor,features(iVar).columna(iCol))=features(iVar).↩
                  codificacion{iValor}(iCol);
147         end
148      end
149
150   % Si la variable no es categórica
151   elseif strcmp(features(iVar).tipo,'%f')
152      % El campo codificación estará vacío
153      features(iVar).codificacion=[];
154   end
155end
```

```
156
157 % Se libera memoria
158 clear indices_valor weights t datos
159
160 % Se añade el índice de cada muestra como primera columna
161 datosCod=[indices datosCod];
162 clear indices
163
164 % Se actualiza el campo columna de la estructura features debido a la
165 % inclusión del índice
166 for iVar=1:nVariables
167    features(iVar).columna=features(iVar).columna+1;
168 end
169
170 % Se procede a calcular las salidas cabecera y nombresCabecera
171 nombresCabecera=cell(1+N,1);
172 cabecera=cell(1,N+1);
173
174 % La primera posición corresponde al índice
175 nombresCabecera{1}={'1 --> Índice'};
176 cabecera{1}='Int';
177
178 % Se completan cabecera y nombresCabecera para el resto de variables
179 for iVar=1:nVariables
180    % Si la variable es categórica
181    if strcmp(features(iVar).tipo,'%s')
182       % Para cada columna de la matriz datosCod
183       for iCol=1:length(features(iVar).columna)
184          col=features(iVar).columna(iCol);
185          cabecera{col}=cell(features(iVar).nValores,1);
186
187          % Se describe el valor que adopta cada categoría
188          for iValor=1:features(iVar).nValores
189             cabecera{col}{iValor}=[features(iVar).valores{iValor} ' --> ' num2str(↩
                   features(iVar).codificacion{iValor}(iCol))];
190          end
191
192          % Se completa nombresCabecera con un nombre para cada nueva
193          % variable
194          nombresCabecera{col}=[num2str(col) ' --> ee' num2str(idCod) '_' num2str(iCol)↩
                'w_' features(iVar).nombre];
195       end
196
197    end
198 end
199
200
201 end
```

Codificación basada en estadísticas de la variable de salida continua

Código A.13 probabilityContinuousTarget.m.

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %
3 %Este script transforma un conjunto de datos ya procesados por main_cargarDatos
4 %de tal forma que solo incluye las variables categóricas y les realiza una
5 %codificación basada en la probabilidad a priori y posteriori de la salida
6 %condicionada por el valor de la variable categórica.
```

```
7  %
8  %Uso:
9  %  [datosCod,nombresCabecera,cabecera,features]=probabilityContinuousTarget(datos,↩
       features,salida,lambda,idCod)
10 %
11 %Entradas:
12 %  datos: Matriz de datos de las muestras dadas por el preprocesamiento
13 %     realizado en main_cargarDatos.m
14 %
15 %  features: Estructura con la información de cada variable dada por el
16 %     preprocesamiento realizado en main_cargarDatos.m
17 %
18 %  salida: Vector columna con la variable de salida para cada una de las muestras
19 %     en datos. Si no se conoce una componente debe ser NaN.
20 %
21 %  lambda: Función anónima que se usará para el cálculo de los pesos de
22 %     las probabilidades a priori y a posteriori. Su única variable de
23 %     entrada es el número de veces que aparece un determinado valor de
24 %     una variable categórica.
25 %
26 %  idCod: Entero usado para ser añadido en nombresCabecera ya que la
27 %     codificación no es única (depende de la función lambda).
28 %
29 %Salidas:
30 %  datosCod: Matriz [MxN'] con tantas filas como muestras y tantas columnas
31 %    como variables categóricas exitan en la matriz de entrada datos. Más
32 %    una columna para los índices.
33 %
34 %  nombresCabecera: Cell [N'x1] donde cada elemento indica el nombre de
35 %     la variable de cada columna de la matriz datosCod.
36 %
37 %  cabecera: Cell[1xN'] donde cada elemento contiene otro cell en el
38 %     cual cada elemento contiene la tranformación númerica que se le
39 %     asigna a cada valor de las variables categóricas.
40 %
41 %  features: Sruct [1xN] donde cada elemento corresponde a una
42 %     característica. Sus campos son:
43 %     - nombre: String con el nombre de la característica.
44 %     - tipo: String con el tipo de la característica.
45 %     - nValores: Entero con el número de valores diferentes que puede
46 %        adoptar cada característica. En el caso de ser continua está
47 %        vacío.
48 %     - valores: Para las continuas está vacío, pero en las
49 %        categóricas es un cell [1 x nValores] que contiene los
50 %        diferentes valores que puede adoptar la característica.
51 %     - representacion: Para las continuas está vacío, pero en las
52 %        categóricas es un vector [1 x nValores] que contiene los
53 %        enteros usados para sustituir los diferentes valores que puede
54 %        adoptar la característica.
55 %     - frecuencias: Para las continuas está vacío, pero en las
56 %        categóricas es un vector [1 x nValores] que contiene el número
57 %        de veces que aparece cada valor.
58 %     - frecuenciaRel: Igual que el anterior pero en tanto por cien.
59 %     - columna: Entero que indica la columna de datosCod que contiene la
60 %        codificación de la variable categórica. Para las continuas está
61 %        vacío.
62 %     - codificacion: Para las continuas está vacío, pero en las
63 %        categóricas es un vector [1 x nValores] que contiene el número
64 %        con el que se sustituye cada posible valor de la variable.
65 %
66 %Notas:
67 %
```

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [datosCod,nombresCabecera,cabecera,features]=probabilityContinuousTarget(datos,features,salida,lambda,idCod)
% Se extrae la columna de los índices
indices=datos(:,1);
datos(:,1)=[];

% Se determinan las dimensiones de la matriz de entradas
[nMuestras,nVariables]=size(datos);

% Se determina la columna de la matriz de salida datosCod que tendrá la
% codificación de cada variable categórica original.
N=1;
for iVar=1:nVariables
   if strcmp(features(iVar).tipo,'%s')
      features(iVar).columna=N;
      N=N+1;
   elseif strcmp(features(iVar).tipo,'%f')
      features(iVar).columna=[];
   end
end
N=N-1;

% Se iniciliza la matriz de datosCod
datosCod=-1*ones(nMuestras,N);

% Se determina la probabilidad a priori
prioriP=mean(salida,'omitnan');

% Se pasa a realizar el cálculo de la codificación
% Para cada una de las variables
for iVar=1:nVariables
   % Si la variable es categórica
   if strcmp(features(iVar).tipo,'%s')
      % Se inicializa el campo de codificacion
      features(iVar).codificacion=-1*ones(1,features(iVar).nValores);

      % Para cada uno de los valores que puede adoptar la variable
      for iValor=1:features(iVar).nValores
         valor=features(iVar).representacion(iValor);

         % Se determina el índice de las muestras con dicho valor
         iMuestrasValor=datos(:,iVar)==valor;
         % Se calcula la probabilidad a posteriori
         posteriorP=mean(salida(iMuestrasValor),'omitnan');
         % Se calcula la codificación mediante el uso de las dos
         % probabilidades y la función lambda
         cod=posteriorP*lambda(sum(iMuestrasValor)) + prioriP*(1-lambda(sum(iMuestrasValor)));
         % Se actualiza el campo codificacion para dicho valor
         features(iVar).codificacion(iValor)=cod;

         % Se atualiza la salida datosCod haciendo uso de la
         % codificación calculada
         datosCod(iMuestrasValor,features(iVar).columna)=cod;
      end

   % Si la variable no es categórica
   elseif strcmp(features(iVar).tipo,'%f')
      % El campo codificación estará vacío
```

```
128         features(iVar).codificacion=[];
129      end
130   end
131
132   % Se añade el índice de cada muestra como primera columna
133   datosCod=[indices datosCod];
134
135   % Se actualiza el campo columna de la estructura features debido a la
136   % inclusión del índice
137   for iVar=1:nVariables
138      features(iVar).columna=features(iVar).columna+1;
139   end
140
141   % Se procede a calcular las salidas cabecera y nombresCabecera
142   nombresCabecera=cell(1+N,1);
143   cabecera=cell(1,N+1);
144
145   % La primera posición corresponde al índice
146   nombresCabecera{1}={'1 --> Índice'};
147   cabecera{1}='Int';
148
149   % Se completan cabecera y nombresCabecera para el resto de variables
150   for iVar=1:nVariables
151      % Si la variable es categórica
152      if strcmp(features(iVar).tipo,'%s')
153         % Se inicializa cada componente de la cabecera con un cell de
154         % tamaño igual al número de valores
155         col=features(iVar).columna;
156         cabecera{col}=cell(features(iVar).nValores,1);
157
158         % En cada componente del cell creado se indica la transformación
159         % numérica llevada a cabo
160         for iValor=1:features(iVar).nValores
161            cabecera{col}{iValor}=[features(iVar).valores{iValor} ' --> ' num2str(↩
                   features(iVar).codificacion(iValor))];
162         end
163
164         % Se completa nombresCabecera con un nombre para cada nueva
165         % variable
166         nombresCabecera{col}=[num2str(col) ' --> pct' num2str(idCod) '_' features(iVar).↩
               nombre];
167
168      end
169   end
170
171
172   end
```

Bibliografía

[1] J. McCarthy, "What is artificial intelligence?" http://www-formal.stanford.edu/jmc/whatisai/node1.html, 2007, accedido 21-09-2020.

[2] W. S. McCulloch and W. H. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," in *EMBODIMENTS OF MIND*, 2016, pp. 19–38.

[3] E. J. Topol, "High-performance medicine: the convergence of human and artificial intelligence," *Nature Medicine*, vol. 25, no. 1, pp. 44–56, Jan 2019. [Online]. Available: https://doi.org/10.1038/s41591-018-0300-7

[4] A. Agrawal, J. Gans, and A. Goldfarb, *Prediction Machines: The Simple Economics of Artificial Intelligence*. Harvard Business Review Press, 2018. [Online]. Available: https://books.google.es/books?id=wJY4DwAAQBAJ

[5] R. R. Trippi and E. Turban, *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance*. USA: McGraw-Hill, Inc., 1992.

[6] M. Dougherty, "A review of neural networks applied to transport," *Transportation Research Part C: Emerging Technologies*, vol. 3, no. 4, pp. 247 – 260, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0968090X95000098

[7] Y. Xu, J. Du, L.-R. Dai, and C.-H. Lee, "A regression approach to speech enhancement based on deep neural networks," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 23, no. 1, p. 7–19, Jan. 2015. [Online]. Available: https://doi.org/10.1109/TASLP.2014.2364452

[8] D. Fu and R. Houlette, "Putting ai in entertainment: an ai authoring tool for simulation and games," *IEEE Intelligent Systems*, vol. 17, no. 4, pp. 81–84, July 2002.

[9] R. Murphy, *Introduction to AI Robotics, second edition*, ser. Intelligent Robotics and Autonomous Agents series. MIT Press, 2019. [Online]. Available: https://books.google.es/books?id=TmquDwAAQBAJ

[10] L. Achaerandio, "Iniciación a la práctica de la investigación," *Guatemala: Universidad Rafael Landivar. Recuperado de: http://www.academia.edu/13574235/iniciacion_a_la_practica_de_la_investigacion*, pp. 81–87, 2010.

[11] M. Martín, *Fundamentos de estadística en ciencias de la salud*, ser. Manuals ; 56. Bellaterra, Barcelona: Universitat Autònoma de Barcelona. Servei de Publicacions, 2010.

[12] W. Duch, K. Grudzinski, and G. Stawski, "Symbolic features in neural networks," in *In Proceedings of the 5th Conference on Neural Networks and Their Applications*. Citeseer, 2000.

[13] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data Mining and Knowledge Discovery*, vol. 2, no. 4, pp. 345–389, Dec 1998. [Online]. Available: https://doi.org/10.1023/A:1009744630224

[14] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[15] J. Fürnkranz, "Separate-and-conquer rule learning," *Artificial Intelligence Review*, vol. 13, pp. 3–54, 1999.

[16] A. An and N. Cercone, "Discretization of continuous attributes for learning classification rules," in *Methodologies for Knowledge Discovery and Data Mining*, N. Zhong and L. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 509–514.

[17] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[18] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[19] S. Boriah, V. Chandola, and V. Kumar, "Similarity measures for categorical data: A comparative evaluation," in *Proceedings of the 2008 SIAM international conference on data mining*. SIAM, 2008, pp. 243–254.

[20] A. Ahmad and L. Dey, "A method to compute distance between two categorical values of same attribute in unsupervised learning for categorical data set," *Pattern Recognition Letters*, vol. 28, no. 1, pp. 110 – 118, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865506001759

[21] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *Journal of artificial intelligence research*, vol. 6, pp. 1–34, 1997.

[22] C. Stanfill and D. Waltz, "Toward memory-based reasoning," *Commun. ACM*, vol. 29, no. 12, p. 1213–1228, Dec. 1986. [Online]. Available: https://doi.org/10.1145/7902.7906

[23] W. Gomaa and A. Fahmy, "A survey of text similarity approaches," *international journal of Computer Applications*, vol. 68, 04 2013.

[24] P. Cerda, G. Varoquaux, and B. Kégl, "Similarity encoding for learning with dirty categorical variables," *CoRR*, vol. abs/1806.00979, 2018. [Online]. Available: http://arxiv.org/abs/1806.00979

[25] K. Q. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. J. Smola, "Feature hashing for large scale multitask learning," *CoRR*, vol. abs/0902.2206, 2009. [Online]. Available:

http://arxiv.org/abs/0902.2206

[26] S. Guha, R. Rastogi, and K. Shim, "Rock: A robust clustering algorithm for categorical attributes," *Information Systems*, vol. 25, no. 5, pp. 345 – 366, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437900000223

[27] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *Journal of Big Data*, vol. 7, pp. 1–41, 2020.

[28] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken, *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[29] M. Zeng, T. Yu, X. Wang, V. Su, L. T. Nguyen, and O. Mengshoel, "Improving demand prediction in bike sharing system by learning global features," in *KDD 2016*, 2016.

[30] R. Bank and C. Douglas, "Sparse matrix multiplication package (smmp)," *Advances in Computational Mathematics*, vol. 1, 05 2001.

[31] S. C. Lowe. (2016, mar) Rank-hot encoder for ordinal features. [Online]. Available: https://scottclowe.com/2016-03-05-rank-hot-encoder/

[32] R. W. Doran, "The gray code." *J. UCS*, vol. 13, no. 11, pp. 1573–1597, 2007.

[33] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," *CoRR*, vol. abs/1604.06737, 2016. [Online]. Available: http://arxiv.org/abs/1604.06737

[34] B. Wang, K. Shaaban, and I. Kim, "Reveal the hidden layer via entity embedding in traffic prediction," *Procedia Computer Science*, vol. 151, pp. 163 – 170, 2019, the 10th International Conference on Ambient Systems, Networks and Technologies (ANT 2019) / The 2nd International Conference on Emerging Data and Industry 4.0 (EDI40 2019) / Affiliated Workshops. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050919304867

[35] D. Micci-Barreca, "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems." *SIGKDD Explorations*, vol. 3, pp. 27–32, 07 2001.

[36] S. E. Buttrey, "Nearest-neighbor classification with categorical variables," *Computational Statistics & Data Analysis*, vol. 28, no. 2, pp. 157 – 169, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167947398000322

[37] D. L. Weed, "Weight of evidence: a review of concept and methods," *Risk Analysis: An International Journal*, vol. 25, no. 6, pp. 1545–1557, 2005.

[38] G. Zeng, "A necessary condition for a good binning algorithm in credit scoring," *Applied Mathematical Sciences*, vol. Vol. 8, pp. 3229–3242, 07 2014.

[39] I. J. WOD, "Weight of evidence: A brief survey," in *Bayesian Statistics 2*, J. M. Bernardo, M. H. Degroot, and D. V. L. A. F. M. Smith, Eds. Amsterdam: Elsevier Science Publisher B.V., 9 1983, pp. 249–270.

[40] S. Goodman, "Toward evidence-based medical statistics. 2: The bayes factor." *Annals of internal medicine*, vol. 130 12, pp. 1005–1013, 1999.

[41] E. Fix and J. L. Hodges, "Discriminatory analysis. nonparametric discrimination: Consistency properties," *International Statistical Review / Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989. [Online]. Available: http://www.jstor.org/stable/1403797

[42] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, no. 4, pp. 325–327, 1976.

[43] C. Dalitz, "Reject options and confidence measures for knn classifiers," *Schriftenreihe des Fachbereichs Elektrotechnik und Informatik der Hochschule Niederrhein*, vol. 8, pp. 16–38, 01 2009.

[44] J. Arlandis, J. C. Perez-Cortes, and J. Cano, "Rejection strategies and confidence measures for a k-nn classifier in an ocr task," in *Object recognition supported by user interaction for service robots*, vol. 1, 2002, pp. 576–579 vol.1.

[45] S. Garcia, J. Derrac, J. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 417–435, 2012.

[46] C.-L. Liu and M. Nakagawa, "Evaluation of prototype learning algorithms for nearest-neighbor classifier in application to handwritten character recognition," *Pattern Recognition*, vol. 34, no. 3, pp. 601 – 615, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320300000182

[47] B. S. Kim and S. B. Park, "A fast k nearest neighbor finding algorithm based on the ordered partition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 761–766, 1986.

[48] P. J. Grother, G. T. Candela, and J. L. Blue, "Fast implementations of nearest neighbor classifiers," *Pattern Recognition*, vol. 30, no. 3, pp. 459 – 465, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320396000982

[49] G. Demiroz and H. A. Güvenir, "Genetic algorithms to learn feature weights for the nearest neighbor algorithm," *Bilkent University*, 08 1996.

[50] Y. Arora, A. Singhal, and A. Bansal, "A study of applications of rbf network," *International Journal of Computer Applications*, vol. 94, pp. 17–20, 2014.

[51] F. Schwenker, H. Kestler, and G. Palm, "Three learning phases for radial-basis-function networks." *Neural networks : the official journal of the International Neural Network Society*, vol. 14, pp. 439–58, 06 2001.

[52] C. Dash, A. K. Behera, S. Dehuri, and S.-B. Cho, "Radial basis function neural networks: A topical state-of-the-art survey," *Open Computer Science*, vol. 6, 01 2016.

[53] K.-L. Du, "Clustering: A neural network approach," *Neural Networks*, vol. 23, no. 1, pp. 89 – 107, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S089360800900207X

[54] R. Rollet, G. B. Benie, W. Li, S. Wang, and J.-M. Boucher, "Image classification algorithm based on the rbf neural network and k-means," *International Journal of Remote Sensing*, vol. 19, no. 15, pp. 3003–3009, 1998. [Online]. Available: https://doi.org/10.1080/014311698214398

[55] Z. Zainuddin and W. Lye, "Improving rbf networks classification performance by using k-harmonic means," *World Academy of Science, Engineering and Technology*, vol. 62, pp. 983–986, 2010.

[56] S. Chen, C. F. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on neural networks*, vol. 2, no. 2, pp. 302–309, 1991.

[57] S.-F. Su, C.-C. Chuang, C.-W. Tao, J.-T. Jeng, and C.-C. Hsiao, "Radial basis function networks with linear interval regression weights for symbolic interval data," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 1, pp. 69–80, 2011.

[58] R. Langari, Liang Wang, and J. Yen, "Radial basis function networks, regression weights, and the expectation-maximization algorithm," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 27, no. 5, pp. 613–623, 1997.

[59] D. Zhang, L.-F. Deng, K. Cai, and A. So, "Fuzzy nonlinear regression with fuzzified radial basis function network," *IEEE Transactions on Fuzzy Systems*, vol. 13, pp. 742–760, 2005.

[60] R. Kohavi and B. Becker, "UCI machine learning repository - census income data set," 1994. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/census+income

[61] T. Kohonen, "Improved versions of learning vector quantization," in *1990 IJCNN International Joint Conference on Neural Networks*, 1990, pp. 545–550 vol.1.

[62] R. Kohavi and F. Provost, "Glossary of terms," *Machine Learning*, vol. 2, pp. 271–274, 01 1998.

[63] M. R. Arahal, M. Berenguel, E. F. Camacho, and F. Pavón, "Selección de variables en la predicción de llamadas en un centro de atención telefónica," *Revista Iberoamericana de Automática E Informática Industrial RIAI*, vol. 6, no. 1, pp. 94–104, 2009.

[64] M. R. Arahal, P. F. Pavón, and E. F. Camacho, "Models for incoming calls forecasting in a customer attention center," *IFAC Proceedings Volumes*, vol. 36, no. 16, pp. 205–210, 2003.

[65] M. Y. Mashor, "Hybrid training algorithm for rbf network," *International Journal of the computer, the Internet and Management*, vol. 8, no. 2, pp. 50–65, 2000.