

INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

***“OBTENCIÓN DE LA CANTIDAD DE NEURONAS EN
LA CAPA OCULTA DE CLASIFICADORES
MULTICATEGORÍA MLP”***

TESIS

QUE PARA OBTENER EL GRADO DE

***DOCTOR EN CIENCIAS EN INGENIERÍA
ELECTRÓNICA***

PRESENTA:

RICARDO MAJALCA MARTÍNEZ

**DIRECTOR DE LA TESIS:
*DR. PEDRO RAFAEL ACOSTA CANO DE LOS RIOS***

CHIHUAHUA, CHIH., OCTUBRE 2019



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO





EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO®

Instituto Tecnológico de Chihuahua

"2019, Año del Caudillo del Sur, Emiliano Zapata"

Chihuahua, Chih. 5 de noviembre de 2019

**RICARDO MAJALCA MARTÍNEZ
PRESENTE**

Por este conducto le comunico que a propuesta del Jurado de Examen, la División de Estudios de Posgrado e Investigación ha concedido autorización para la impresión de su tesis para obtener el grado de Doctor en Ciencias en Ingeniería Electrónica, cuyo título es:

"Obtención de la cantidad de neuronas en la capa oculta de clasificadores multicategoría MLP"

La tesis presenta el siguiente contenido de capítulos:

- I Antecedentes
- II Estado de la técnica
- III Estructuras y técnicas utilizadas en este estudio
- IV Desarrollo
- V Validación
- VI Conclusiones

ATENTAMENTE

Excelencia en Educación Tecnológica®

"La técnica por el Engrandecimiento de México"

**MTRO. LUIS CARDONA CHACÓN
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN**



Ave Tecnológico No. 2909 Col. 10 de Mayo C.P. 31310, Chihuahua, Chih. México
Tel. 01 (614) 201 2000, (614) 413 5187, Ext. 2150 e-mail: dir_chihuahua@tecnm.mx

www.tecnm.mx | www.itchihuahua.edu.mx





Chihuahua, Chih. 31 de octubre de 2019

MTRO. LUIS CARDONA CHACÓN
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
PRESENTE

Por medio de la presente notificamos a usted que en cumplimiento de los requerimientos para la obtención del grado de Doctor en Ciencias en Ingeniería Electrónica, el documento de tesis del alumno **C. RICARDO MAJALCA MARTÍNEZ**, ha sido aprobado y aceptado para su impresión. El título de la tesis es:

"Obtención de la cantidad de neuronas en la capa oculta de clasificadores multicategoría MLP"

Por lo que proponemos, le sea concedida la autorización de impresión correspondiente.

Agradeciendo la atención a la presente, quedamos de usted:

ATENTAMENTE

Excelencia en Educación Tecnológica
"La técnica por el Engrandecimiento de México"

**DR. PEDRO RAFAEL ACOSTA CANO DE
LOS RÍOS**
DIRECTOR DE TESIS

DR. JOSÉ RIVERA MEJÍA
MIEMBRO DEL JURADO DE EXAMEN

DR. RICARDO BLANCO VEGA
MIEMBRO DEL JURADO DE EXAMEN

DR. ÓSCAR ARTURO CHÁVEZ LÓPEZ
MIEMBRO DEL JURADO DE EXAMEN

DR. JAVIER VEGA PINEDA
MIEMBRO DEL JURADO DE EXAMEN



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN
PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CHIHUAHUA
**DIVISIÓN DE ESTUDIOS DE POSGRADO
E INVESTIGACIÓN**





EDUCACIÓN

SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO®

Instituto Tecnológico de Chihuahua

"2019, Año del Caudillo del Sur, Emiliano Zapata"

CARTA CESIÓN DE DERECHOS

En la ciudad de Chihuahua el día 31 de octubre de 2019, el que suscribe **RICARDO MAJALCA MARTÍNEZ** del Doctorado en Ciencias en Ingeniería Electrónica, con número de control **D15060705**, adscrito a la **DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN**, del Instituto Tecnológico de Chihuahua, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección del **Dr. Pedro Rafael Acosta Cano de los Ríos** y cede los derechos del trabajo titulado *"Obtención de la cantidad de neuronas en la capa oculta de clasificadores multicategoría MLP"*, al Tecnológico Nacional de México y/o Instituto Tecnológico de Chihuahua para su difusión, divulgación, transmisión, reproducción, así como su digitalización con fines académicos y de investigación.

RICARDO MAJALCA MARTÍNEZ



Ave Tecnológico No. 2909 Col. 10 de Mayo C.P. 31310, Chihuahua, Chih. México
Tel. 01 (614) 201 2000, (614) 413 5187, Ext. 2150 e-mail: dir_chihuahua@tecnm.mx
www.tecnm.mx | www.itchihuahua.edu.mx



Ad majorem Dei gloriam Mariaquae

A mis padres Rubén Y María de Lourdes, cuyas oraciones y bendiciones siempre están conmigo. A mis hermanas Cecilia y Cristina con cuyo apoyo y soporte siempre cuento. A mi director de tesis, Dr. Pedro Acosta por ayudarme en ese largo camino recorrido. A Paty, siempre juntos, todo juntos, hoy y siempre.

Índice general

Índice de figuras	v
Índice de tablas	ix
1. Antecedentes	1
1.1. La cantidad de neuronas en la capa oculta	3
1.2. La información geométrica como valor heurístico	5
2. Estado de la técnica	6
2.1. Reglas de oro	8
2.2. Métodos de prueba y error	9
2.3. Reglas heurísticas	14
3. Estructuras y técnicas utilizadas en este estudio	16
3.1. Fundamentos de una red MLP	16
3.1.1. La neurona artificial	16
3.1.2. Capas de neuronas	20
3.1.3. Redes neuronales artificiales	21
3.1.3.1. Propagación hacia adelante	22
3.2. Entrenamiento en redes MLP	23
3.2.1. Aprendizaje en una red MLP	24
3.2.2. La regla delta generalizada	25
3.3. Diseño de la arquitectura	30
3.3.1. La cantidad de capas ocultas	30
3.3.2. El total de neuronas en la capa oculta	31
3.4. Redes MLP clasificadoras	31
3.4.1. La capa de entrada y la capa de salida	33

3.4.2.	La capa oculta y la regla de los hiperplanos	34
3.4.3.	MLP clasificador multiclase	35
3.4.4.	Clases empalmadas	37
3.5.	Geometría computacional	38
3.5.1.	Envoltentes convexas	39
3.5.2.	Triangulaciones de Delaunay	40
3.5.3.	Formas alfa	45
4.	Desarrollo	48
4.1.	La forma de una clase	48
4.1.1.	La forma alfa de una clase	49
4.1.2.	Aproximando la forma alfa de una clase	49
4.1.3.	Ejemplos prácticos de la aproximación a la forma alfa	57
4.1.3.1.	Ejemplo 4.1.3a	57
4.1.3.2.	Ejemplo 4.1.3b	57
4.1.3.3.	Ejemplo 4.1.3c	57
4.1.3.4.	Ejemplo 4.1.3d	58
4.2.	Los bordes fundamentales de una clase	59
4.2.1.	Usando el vértice más cercano en la otra clase	61
4.2.2.	El uso de programación lineal para la aproximación a la envolvente convexa	62
4.2.3.	Ejemplos sobre la extracción de los bordes fundamentales	64
4.2.3.1.	Ejemplo 4.2.3a	65
4.2.3.2.	Ejemplo 4.2.3b	65
4.2.3.3.	Ejemplo 4.2.3c	66
4.2.3.4.	Ejemplo 4.2.3d	67
4.2.3.5.	Ejemplo 4.2.3e	68
4.3.	El promedio entre clases	68
4.3.1.	Superficies de decisión	68
4.3.2.	El promedio entre clases como superficie de decisión	70
4.3.3.	Ejemplos sobre el cálculo del promedio entre clases	72
4.3.3.1.	Ejemplo 4.3.3a	72
4.3.3.2.	Ejemplo 4.3.3b	73
4.3.3.3.	Ejemplo 4.3.3c	73
4.3.3.4.	Ejemplo 4.3.3d	73

4.3.3.5.	Ejemplo 4.3.3e	74
4.4.	Ajustando el promedio entre clases mediante hiperplanos	74
4.4.1.	Ejemplos sobre el ajuste con hiperplanos	78
4.4.1.1.	Ejemplo 4.4.1a	79
4.4.1.2.	Ejemplo 4.4.1b	79
4.4.1.3.	Ejemplo 4.4.1c	80
4.4.1.4.	Ejemplo 4.4.1d	81
4.4.1.5.	Ejemplo 4.4.1e	82
4.5.	Algoritmo completo con el método propuesto para la obtención del tamaño de la capa oculta	82
4.6.	El caso multiclase	84
4.6.1.	El esquema OaO en una red SLFN	85
4.6.2.	Ejemplos del uso del esquema OaO en problemas multiclase	86
4.6.2.1.	Ejemplo 4.6.2a	86
4.6.2.2.	Ejemplo 4.6.2b	88
4.7.	El caso de las clases traslapadas	89
4.7.1.	Preprocesando los datos de entrenamiento	90
4.7.1.1.	Tres tipos de traslape	90
4.7.1.2.	La zona de conflicto	92
4.7.1.3.	La clase conflicto	93
4.7.2.	Ejemplos del proceso de aislamiento de la zona y clase conflicto	96
4.7.2.1.	Ejemplo 4.7.2a	97
4.7.2.2.	Ejemplo 4.7.2b	97
4.7.2.3.	Ejemplo 4.7.2c	98
4.7.3.	La clasificación con clases traslapadas	98
4.7.4.	Ejemplos de clasificación con clases traslapadas	103
4.7.4.1.	Ejemplo 4.7.4a	103
4.7.4.2.	Ejemplo 4.7.4b	104
4.7.4.3.	Ejemplo 4.7.4c	105
5.	Validación	107
5.1.	Técnicas de clasificación a comparar	107
5.1.1.	El clasificador kNN	107
5.1.2.	El clasificador bayesiano	108
5.1.3.	El clasificador basado en árboles de decisión	109

5.1.4.	El clasificador SVM	109
5.2.	Ejemplos de aplicación	110
5.2.1.	Ejemplos de aplicación sobre datasets artificiales	111
5.2.1.1.	Ejemplo 5.2.1a	112
5.2.1.2.	Ejemplo 5.2.1b	113
5.2.1.3.	Ejemplo 5.2.1c	114
5.2.1.4.	Ejemplo 5.2.1d	115
5.2.1.5.	Ejemplo 5.2.1e	116
5.2.1.6.	Ejemplo 5.2.1f	117
5.2.1.7.	Ejemplo 5.2.1g	118
5.2.1.8.	Ejemplo 5.2.1h	119
5.2.1.9.	Ejemplo 5.2.1i	120
5.2.2.	Ejemplos de aplicación sobre datasets reales	121
5.2.2.1.	El dataset TAE	121
5.2.2.2.	El dataset Crabs	122
5.2.2.3.	El dataset Iris	123
5.2.2.4.	El dataset Haberman's Survival Dataset	123
5.2.2.5.	El dataset Paychecks	124
5.2.2.6.	El dataset Ionosphere	125
5.2.2.7.	El dataset Wine	126
6.	Conclusiones	128
6.1.	Resultados	128
6.2.	Condiciones de uso	130
6.3.	Trabajo a futuro	131
7.	Referencias	133

Índice de figuras

1.1. Cantidad de capas ocultas y de neuronas en cada capa oculta	2
1.2. El único problema a resolver en una red SLFN, es la cantidad de neuronas en la capa oculta	3
2.1. La red SLFN a utilizar en esta investigación. Nótese la notación para la cantidad de elementos o neuronas en cada capa	6
3.1. Una neurona artificial	17
3.2. Una capa con $n = 4$ neuronas, con un patrón de entrada $\mathbf{X} \in \mathbb{R}^3$, y una respuesta $\mathbf{Y} \in \mathbb{R}^4$	21
3.3. Redes RNA, en general, y con su nomenclatura oficial	22
3.4. La conexión w_{jl}^i a corregir durante la explicación de la regla delta	26
3.5. Elementos en la red involucrados en el proceso de retropropagación	28
3.6. Una red MLP con una sola capa de neuronas ocultas	30
3.7. La red SLFN clasificadora. La capa de entrada tiene $n_E + 1$ elementos incluido un bias. La capa oculta tiene $n_O + 1$ neuronas incluido un bias. La capa de salida contiene n_S neuronas	34
3.8. La regla de los hiperplanos en un problema de clasificación binaria, en $\mathbb{R}^2$. . .	35
3.9. Problema de clasificación con tres clases	36
3.10. Los esquemas de clasificación multiclase	37
3.11. Clases empalmadas y no empalmadas, en $\mathbb{R}^2$	38
3.12. El concepto de envolvente convexa, para un grupo de datos en $\mathbb{R}^2$	40
3.13. El concepto de n-esfera circunscrita, en $\mathbb{R}^2$	42
3.14. Los círculos circunscritos sobre una triangulación de Delaunay en $\mathbb{R}^2$	42
3.15. El <i>flipping</i> de una arista para producir una triangulación de Delaunay válida, en un ejemplo en $\mathbb{R}^2$	43
3.16. La inserción de un nuevo punto en una triangulación previa, en un caso en $\mathbb{R}^2$.	44

3.17. La diferencia entre forma ideal, y envolvente convexa en $\mathbb{R}^2$	45
3.18. La n-esfera con distintos radios, en $\mathbb{R}^2$, mas bien un círculo	46
4.1. El concepto de la forma de una clase. En (a), la clase original. En (b), su forma. Nótese las muestras en los bordes de la clase	49
4.2. La forma alfa en dos y tres dimensiones. Nótese la forma poliédrica en color verde, de la forma alfa, delimitada por las muestras en los bordes de la nube . .	50
4.3. La nube original de puntos en $\mathbb{R}^2$	52
4.4. Cuatro casos de agrupamientos para la misma nube de la Figura 4.3	53
4.5. Las aproximaciones a la forma alfa, $\alpha'(C_1)$, en $\mathbb{R}^2$, de mayor a menor cantidad de agrupamientos, en comparación con la forma alfa, $\alpha(C_1)$, ideal	54
4.6. Cuatro casos de dispersión y extensión, en $\mathbb{R}^2$	55
4.7. La aproximación a la forma de una clase en el ejemplo 4.1.3a en $\mathbb{R}^2$	58
4.8. La aproximación a la forma de una clase en el Ejemplo 4.1.3b en $\mathbb{R}^2$	59
4.9. La aproximación a la forma de una clase en el Ejemplo 4.1.3c en $\mathbb{R}^2$	60
4.10. La aproximación a la forma de una clase en el Ejemplo 4.1.3d en $\mathbb{R}^3$	61
4.11. Una aproximación a la forma de una clase en $\mathbb{R}^2$	62
4.12. Los bordes fundamentales en dos clases en $\mathbb{R}^2$. Los bordes fundamentales son los vértices cuyo contorno aparece en color verde, en ambas clases	63
4.13. Estableciendo los bordes fundamentales con el método del vértice mas cercano	64
4.14. Bordes fundamentales en el ejemplo 4.2.3a	66
4.15. Bordes fundamentales en el ejemplo 4.2.3b	66
4.16. Bordes fundamentales en el ejemplo 4.2.3c	67
4.17. Bordes fundamentales en el Ejemplo 4.2.3d	67
4.18. Bordes fundamentales en el ejemplo 4.2.3e	68
4.19. Tres casos de problemas de clasificación binaria, en $\mathbb{R}^2$	69
4.20. Los bordes correspondientes entre los bordes fundamentales, en un problema en $\mathbb{R}^2$	70
4.21. El promedio aproximado entre clases, en $\mathbb{R}^2$	72
4.22. El promedio entre clases en $\mathbb{R}^2$ en el Ejemplo 4.3.3a	72
4.23. El promedio entre clases, en $\mathbb{R}^2$ del Ejemplo 4.3.3b	73
4.24. El promedio entre clases en $\mathbb{R}^2$ en el Ejemplo 4.3.3c	74
4.25. El promedio entre clases en $\mathbb{R}^2$ en el ejemplo 4.3.3d	75
4.26. El promedio entre clases en $\mathbb{R}^3$ en el Ejemplo 4.3.3e	76

4.27. Comparando a la verdadera, y muy ideal superficie de decisión y la aproximación mediante el promedio en $\mathbb{R}^2$	76
4.28. La tolerancia <i>tole</i> para ajustar un grupo de muestras a un hiperplano I	77
4.29. El ajuste mediante hiperplanos, cuatro de ellos, a las muestras del promedio y la superficie ideal en $\mathbb{R}^2$	79
4.30. La mínima distancia entre clases, la distancia entre el par de bordes correspondientes más cercanos, en un caso en $\mathbb{R}^2$	79
4.31. El ajuste con $n_H = 7$ hiperplanos en el Ejemplo 4.4.1a	80
4.32. El ajuste con $n_H = 3$ hiperplanos en el Ejemplo 4.4.1b	80
4.33. El ajuste con $n_H = 4$ hiperplanos en el Ejemplo 4.4.1c	81
4.34. El ajuste con $n_H = 1$ hiperplano en el Ejemplo 4.4.1d	81
4.35. El ajuste con $n_H = 4$ hiperplano en el Ejemplo 4.4.1e	82
4.36. La red SLFN clasificadora para el problema en el ejemplo 4.4.1a	83
4.37. La red SLFN clasificadora para el problema en el Ejemplo 4.4.1b	84
4.38. La red SLFN clasificadora para el problema en el Ejemplo 4.4.1c	85
4.39. La red SLFN clasificadora para el problema en el Ejemplo 4.4.1d	86
4.40. La red SLFN clasificadora para el problema en el Ejemplo 4.4.1e	87
4.41. la regla de los hiperplanos adaptada al esquema multiclase OaO	87
4.42. El esquema OaO adaptado a la regla de los hiperplanos, en un problema con $n_C = 3$ en $\mathbb{R}^2$	88
4.43. El problema de clasificación multiclase en el Ejemplo 4.6.2a. Las clases son C_1 en color rojo, C_2 en color azul, y c_3 en color verde. También, la separación de las clases en $n_H = 5$ hiperplanos	89
4.44. La red SLFN que separa las tres clases con $n_O = n_H = 6$ neuronas ocultas, deducidas según el esquema OaO y la regla de los hiperplanos en el ejemplo 4.6.2a	90
4.45. El problema de clasificación multiclase en el Ejemplo 4.6.2b. Las clases son C_1 en color rojo, C_2 , en color azul y c_3 en color verde. También, la separación de las clases en $n_H = 17$ hiperplanos	91
4.46. La red SLFN que separa las tres clases, con $n_O = n_H = 17$ neuronas ocultas deducida según el esquema OaO y la regla de los hiperplanos, en el ejemplo 4.6.2b	92
4.47. Los casos de dos clases empalmadas en $\mathbb{R}^2$	93
4.48. La zona de conflicto, en los tres casos de clases empalmadas, en $\mathbb{R}^2$	94
4.49. La separación final de las clases luego del preprocesamiento, en $\mathbb{R}^2$	95
4.50. Separando las clases conflicto del resto de ellas en el Ejemplo 4.7.2a	98
4.51. Separando las clases conflicto del resto de ellas en el Ejemplo 4.7.2b	99

4.52. Separando las clases conflicto del resto de ellas en el Ejemplo 4.7.2c	100
4.53. Los tres esquemas para clasificación en clases traslapadas, manipulando la clase conflictiva, en $\mathbb{R}^2$	100
4.54. El clasificador final, utilizado en este estudio, usando el esquema del mezclado	103
4.55. El problema de clasificación del ejemplo 4.7.4a, separado con clases C_1 , C_2 y C_{cnf} y la red SLFN para la primera etapa del clasificador, según el esquema del mezclado	104
4.56. El problema de clasificación en el Ejemplo 4.7.4b ya separado, con clases C_1 , C_2 y C_{cnf} y su red SLFN para la primera etapa del clasificador, según el esquema del mezclado	105
4.57. El problema de clasificación en el Ejemplo 4.7.4c ya separado con clases C_1 , C_2 y C_{cnf} y su red SLFN para la primera etapa del clasificador, según el esquema del mezclado	106
5.1. El problema original en $\mathbb{R}^2$ del Ejemplo 5.2.1a	112
5.2. El problema de clasificación para el Ejemplo 5.2.1b	113
5.3. El problema de clasificación para el ejemplo 5.2.1c	114
5.4. El problema de clasificación para el ejemplo 5.2.1d	115
5.5. El problema de clasificación para el Ejemplo 5.2.1e	116
5.6. El problema de clasificación para el ejemplo 5.2.1f	117
5.7. El problema de clasificación para el Ejemplo 5.2.1g	118
5.8. El problema de clasificación para el ejemplo 5.2.1h	119
5.9. El problema de clasificación, para el Ejemplo 5.2.1i	120

Índice de tablas

3.1. Funciones de activación populares	18
5.1. Tabla comparativa sobre métodos para obtener n_O en el Ejemplo 5.2.1a	112
5.2. Tabla comparativa de los cinco modelos de clasificador en el Ejemplo 5.2.1a . .	112
5.3. Comparación entre métodos para obtener n_O en el Ejemplo 5.2.1b	113
5.4. Comparación de los métodos clasificadores en el Ejemplo 5.2.1b	113
5.5. Comparación entre métodos para obtener n_O en el Ejemplo 5.2.1c	114
5.6. Comparación de los métodos clasificadores en el Ejemplo 5.2.1c	114
5.7. Comparación entre métodos para obtener n_O en el ejemplo 5.2.1d	115
5.8. Comparación de los métodos clasificadores en el Ejemplo 5.2.1d	115
5.9. Comparación entre métodos para obtener n_O en el Ejemplo 5.2.1e	116
5.10. Comparación de los métodos clasificadores en el Ejemplo 5.2.1e	116
5.11. Comparación entre métodos para obtener n_O , en el Ejemplo 5.2.1f	117
5.12. Comparación de los métodos clasificadores en el Ejemplo 5.2.1f	117
5.13. Comparación entre métodos para obtener n_O , en el Ejemplo 5.2.1g	119
5.14. Comparación de los métodos clasificadores en el Ejemplo 5.2.1g	119
5.15. Comparación entre métodos para obtener n_O en el Ejemplo 5.2.1h	120
5.16. Comparación de los métodos clasificadores en el Ejemplo 5.2.1h	120
5.17. Comparación entre métodos para obtener n_O , en el Ejemplo 5.2.1i	121
5.18. Comparación de los métodos clasificadores en el ejemplo 5.2.1i	121
5.19. Comparación entre métodos para obtener n_O , en el problema del dataset TAE .	122
5.20. Comparación de los métodos clasificadores en el problema del dataset TAE . .	122
5.21. Comparación entre métodos para obtener n_O , en el problema del dataset Crabs	122
5.22. Comparación de los métodos clasificadores en el problema del dataset Crabs . .	122
5.23. Comparación entre métodos para obtener n_O , en el problema del dataset Iris . .	123
5.24. Comparación de los métodos clasificadores en el problema del dataset Iris . . .	123

5.25. Comparación entre métodos para obtener n_O , en el problema del dataset Haberman's Survival	124
5.26. Comparación de los métodos clasificadores en el problema del dataset Haberman's Survival	124
5.27. Comparación entre métodos para obtener n_O , en el problema del dataset Paychecks	125
5.28. Comparación de los métodos clasificadores en el problema del dataset Paychecks	125
5.29. Comparación entre métodos para obtener n_O , en el problema del dataset Ionosphere	125
5.30. Comparación de los métodos clasificadores en el problema del dataset Ionosphere	125
5.31. Comparación entre métodos para obtener n_O , en el problema del dataset Wines	126
5.32. Comparación de los métodos clasificadores en el problema del dataset Wines .	126

Capítulo 1

Antecedentes

Una *red MultiLayered Perceptron (Multi-Layered Perceptron)*, es una red neuronal muy popular incluso actualmente luego de décadas de haber sido propuesta, luego de que nuevos paradigmas de máquinas que aprenden siguen surgiendo con el paso de los años. Una razón de esto es que la red MLP tiene una capacidad casi infinita de adaptarse a cualquier problema que le sea asignado [1]. La red MLP es aún hoy una red neuronal muy aplicada en una gran variedad de problemas, desde predicción de valores hasta clasificación de patrones. Las redes MLP tienen toda una filosofía detrás de ellas en cuanto a como una máquina que aprende debe organizarse y lo más importante, cómo debe aprender. Su descubrimiento fue el parte aguas de la investigación en redes neuronales a partir de mediados de la década de 1980 y ha dado lugar a un sin fin de investigaciones acerca de la mejora de estos esquemas, no pocas veces resultando en nuevas formas de redes neuronales y sus correspondientes métodos de aprendizaje sin que esto signifique que la original red MLP pierda popularidad y atención.

Sin embargo dos aspectos concernientes a la organización de la red MLP quedan aún abiertos, por un lado *la cantidad de capas ocultas* que una red MLP debe poseer y por otro, *la cantidad de neuronas en cada una de esas capas ocultas* [2]. El correcto desempeño de una red MLP tiene una relación muy directa con la correcta respuesta que estas dos aspectos de diseño a resolver. Una red neuronal es útil en tanto su desempeño sea adecuado. Una red MLP pobremente diseñada tiene un desempeño inadecuado, por no decir que resulta inútil del todo [3]. La Figura 1.1 a continuación es una representación gráfica acerca del concepto de cantidad de capas ocultas, y de la cantidad de neuronas en cada capa oculta. En la Figura 1.1 el concepto de total de capas ocultas aparece remarcado en un rectángulo con bordes rojos, en tanto que el concepto de total de neuronas en cada capa oculta aparece en un recuadro con bordes verdes alrededor de cada capa oculta.

I. Antecedentes

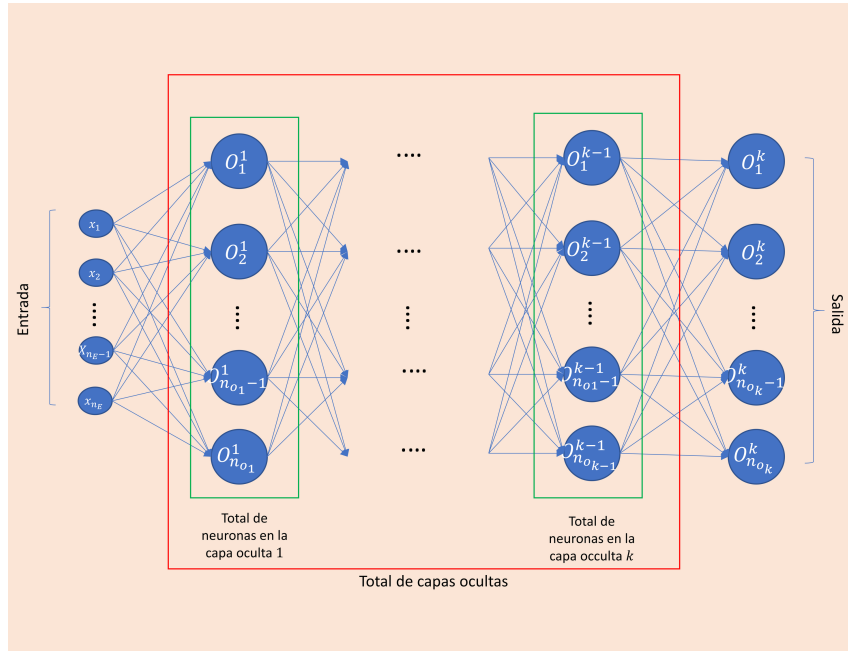


Figura 1.1: Cantidad de capas ocultas y de neuronas en cada capa oculta

En principio una red MLP con bastantes capas ocultas y una gran cantidad de neuronas en cada capa debería ser capaz de aprender cualquier relación que existiera entre un conjunto de datos de entrenamiento y sus respuestas deseadas. En la práctica sin embargo, existen muchas situaciones que deben ser atendidas para evitar ciertos efectos nocivos que esta filosofía pueda producir. A saber:

- (1) Una *gran cantidad de capas ocultas y de neuronas* puede producir una red que no generaliza lo que aprende (*overfitting* o sobreajuste) [1, 4].
- (2) Una gran cantidad de capas ocultas no es un aspecto que se lleve bien con la metodología de entrenamiento común en una red MLP, pues esta produce que las conexiones de las capas más cercanas a la de entrada no reciban ninguna modificación, lo que puede producir que el entrenamiento sea mas lento, o en el peor caso, inefectivo [1, 5].
- (3) Una *pobre cantidad de capas ocultas y de neuronas* puede producir una red que no logra aprender en lo absoluto (*underfitting* o subajuste) [1, 6].

Estos dos problemas se pueden minimizar si la red neuronal MLP tiene tanto la cantidad correcta de capas ocultas, como la cantidad correcta de neuronas en cada capa oculta que utilice [2]. Resulta entonces de suma importancia que la red MLP sea diseñada con una organización adecuada, con la cantidad correcta de capas ocultas y con la cantidad correcta de neuronas en cada capa oculta, para que la red MLP sea de alguna utilidad.

I. Antecedentes

El primer detalle de diseño el referente a la cantidad de capas de neuronas ocultas. En la mayoría de los casos una sola capa de neuronas ocultas bastaría para que la red aprendiese esa relación entre las muestras de entrenamiento y sus respectivas entradas [7], dado que la cantidad de neuronas en esta única capa de neuronas ocultas sea la adecuada. Una red MLP que utiliza solo una capa de neuronas ocultas habrá de ser denotada como red *single layered feedforward network* (SLFN), en español *red de conexiones hacia adelante con una sola capa*. El término una sola capa hace referencia a la única capa de neuronas ocultas. La Figura 1.2 muestra la red SLFN que será utilizada a lo largo del presente estudio, una red con una sola capa oculta. Ahora el único problema a resolver es la cantidad de neuronas ocultas, en la Figura 1.2 esquematizada mediante un recuadro con bordes color rojo.

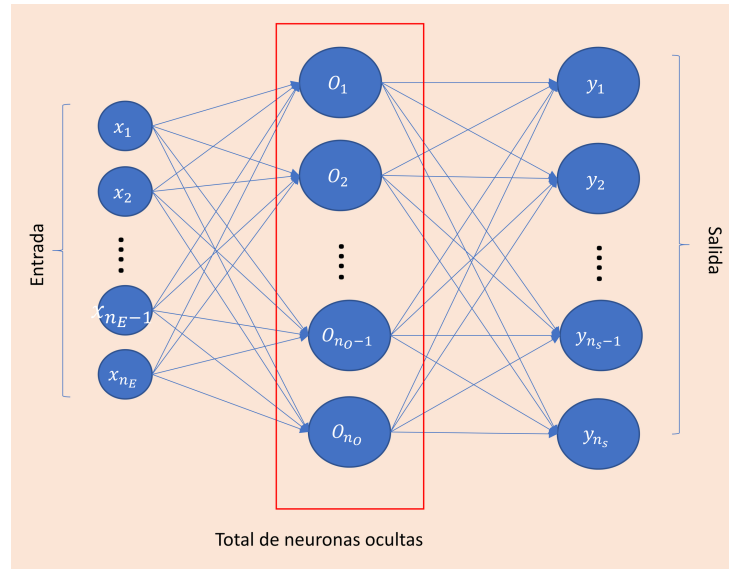


Figura 1.2: El único problema a resolver en una red SLFN, es la cantidad de neuronas en la capa oculta

1.1. La cantidad de neuronas en la capa oculta

Considerando que una sola capa de neuronas ocultas es suficiente resta el segundo aspecto de diseño, la cantidad de neuronas ocultas que esa única capa de neuronas ocultas debe poseer. Este aspecto a diferencia del anterior, no tiene una respuesta, una heurística o regla de oro, o principio formal, para alcanzar una respuesta o acuerdo general. En cambio, este aspecto de diseño sigue siendo un tema de investigación muy abierto.

Para la capa oculta de una red SLFN, que es el tipo de red MLP en el que esta investigación se centra, debe establecerse su cantidad de neuronas ocultas de manera que, por un lado, no

I. Antecedentes

posea un exceso de neuronas, porque se corre el riesgo de una red con overfitting, y por otro lado, el entrenamiento de tal red puede ser innecesariamente lento. Además no debe contener menos neuronas de las necesarias porque se tiene ahora el problema del underfitting. Acerca de esta cantidad de neuronas *la mínima cantidad de neuronas ocultas que hagan que la red SLFN aprenda adecuadamente, es el tamaño correcto* [8].

Es oportuno hacer la siguiente declaración: *no existe manera de determinar cuantas capas deberían existir en la única capa oculta en una red SLFN*. No lo existe de manera general y única, como en una fórmula o regla general. Sucede que la cantidad de neuronas en una capa oculta depende de distintos factores que escapan a un enfoque general. Por ejemplo, depende del método de aprendizaje de la red neuronal [9], pero principalmente, para una red SLFN, del tamaño de la capa oculta *es un detalle de diseño que tiene relación más con las particularidades del problema a resolver* [10], que con un aspecto general de diseño. Esto implica que el problema no es que no se pueda establecer un tamaño para la capa oculta, sino que en realidad existen muchas formas de hacerlo, todo relacionado con el problema a resolver.

Normalmente se siguen dos metodologías para llegar al tamaño de la capa oculta:

- (1) Seguir un *proceso de prueba y error*, un proceso de búsqueda del mejor tamaño, dentro de un espacio de posibles soluciones, hasta encontrar la mejor [11].
- (2) Usar *reglas de oro*, que son reglas sencillas que posiblemente produzcan un buen estimado para el tamaño de la capa oculta [12, 11].

La primer metodología produce una buena estimación acerca del tamaño de la capa oculta, puesto que la busca en un espacio de soluciones. Esta metodología tiene algunos inconvenientes: en general es lenta y como con cualquier proceso de búsqueda basado en derivadas o diferenciales, tiene el riesgo de quedar estancada en un óptimo local al buscar por el mejor valor de las conexiones o pesos [11], o de no encontrar la solución del todo, si no se tiene una buena política de toma de decisiones. La segunda metodología, la de las reglas de oro, es en definitiva una metodología que produce resultados rápidos, sin embargo tiene un enorme inconveniente: carece de formalidad matemática [13, 12]. De hecho ni siquiera usa la información local del problema para alcanzar una solución.

Un buen punto de equilibrio entre la metodología segura de los métodos de prueba y error y la metodología rápida en el uso de reglas de oro, consiste en usar *una regla heurística*, que a diferencia de las reglas de oro, sí usa información local del problema a resolver para alcanzar respuesta aceptable a cuantas neuronas deben existir en la única capa oculta.

1.2. La información geométrica como valor heurístico

¿Cual es la clase de información local del problema que puede servir para llegar a una buena estimación acerca del tamaño de la capa oculta? Existen muchas opciones sobre lo que puede servir como información heurística para encontrar el tamaño de la capa oculta. Por ejemplo, algunos usan la relación estadística que pueda existir entre las muestras de entrenamiento y sus respuestas deseadas [14]. O la relación entre la cantidad de conexiones en la red SLFN y la cantidad de información de entrenamiento [9]. Estas buscan minimizar el efecto del sobre ajuste, pensando que demasiadas conexiones implican una red muy grande, aunque no utilizan las características particulares del problema. En el *caso específico de que el problema que se debe resolver por la red SLFN sea un problema de clasificación de patrones*, algo que resulta realmente útil es la *caracterización geométrica de las clases involucradas*.

En esta investigación se propone un método para el uso de la caracterización de la frontera entre clases, aproximada mediante hiperplanos en el espacio del problema que es $\mathbb{R}^{n_E}$, para $n_E \geq 2$, siendo n_E la cantidad de elementos en el patrón de entrada a la red clasificadora, es decir las dimensiones del problema. Todo esto debe ser tal que sea la cantidad de estos los que establezcan la cantidad de neuronas para la única capa oculta de la red SLFN. Esta cantidad de hiperplanos se obtiene luego de caracterizar geométricamente a las clases involucradas y a la separación entre estas, incluso cuando estas clases ni siquiera estén separadas. Cuando se habla de una caracterización geométrica de las clases se habla de ser capaces de describir mediante geometrías, el alcance y límites de las clases, por ejemplo, a manera de poliedros en ese espacio $\mathbb{R}^{n_E}$, de manera que sea fácil establecer donde una clase termina y las otras comienzan, donde entre las clases, existe una frontera caracterizable por hiperplanos, otra descripción geométrica de un concepto tan general como el de frontera entre clases.

A toda esta metodología que esta investigación propone, desde la caracterización de las clases mediante poliedros, a la aproximación a la frontera entre clases mediante hiperplanos, a la manera en que esta frontera es localizada, se le llamará en adelante la *regla de los hiperplanos y su metodología*.

Capítulo 2

Estado de la técnica

Una red SLFN está formada siempre por tres capas: la *capa de entrada*, formada por n_E elementos de entrada, la *capa oculta* [15], formada por n_O neuronas, y la *capa de salida*, formada por n_S neuronas. Nótese que la capa de entrada no contiene neuronas, sino solamente los elementos mismos del patrón de entrada. Entre la capa de entrada y la capa oculta existe un grupo de conexiones $\mathbf{W}^1$, y entre la capa oculta y la capa de salida existe otro grupo de conexiones denominado $\mathbf{W}^2$, como lo muestra la Figura 2.1.

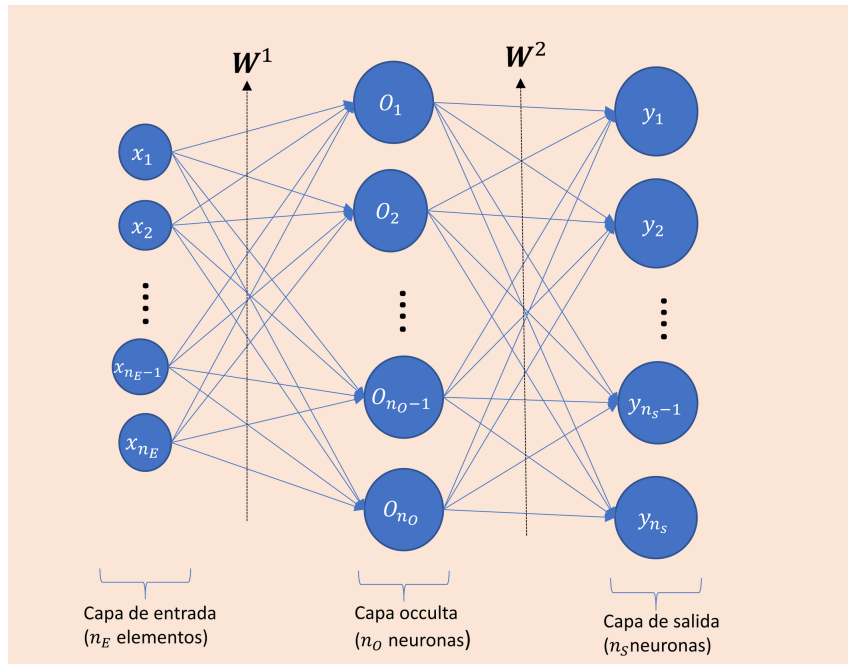


Figura 2.1: La red SLFN a utilizar en esta investigación. Nótese la notación para la cantidad de elementos o neuronas en cada capa

II. Estado de la técnica

Entonces ¿como se establece el tamaño de la capa oculta, n_O ? Normalmente se siguen dos metodologías muy generales:

$$\text{Métodos para establecer } n_O \left\{ \begin{array}{l} \text{Métodos de prueba y error} \\ \text{Métodos heurísticos} \left\{ \begin{array}{l} \text{Reglas de oro} \\ \text{Reglas heurísticas} \end{array} \right. \end{array} \right.$$

Es importante destacar una vez más que no existe forma de establecer de manera general, cuantas neuronas deben componer la capa oculta de una red SLFN, pues esto depende de muchos factores involucrados con el problema a clasificar, la información de entrada, la información de salida, la complejidad de la relación entre entradas y targets, el tamaño del conjunto de entrenamiento, las funciones de activación que las neuronas usan, el algoritmo de entrenamiento, la cantidad de ruido en la información de entrenamiento, entre otros [16, 15]. En todo caso, *el tamaño correcto de la capa oculta es dependiente del problema a resolver* [17]. De esta manera, los métodos comúnmente usados para establecer el tamaño de la capa oculta son de dos estilos: los de *prueba y error*, que no intentan adivinar o calcular el tamaño de la capa oculta, el valor para n_O , sino que lo *buscan exhaustivamente* [17, 18]. Lo bueno de los métodos de prueba y error, es que cuando encuentran un valor para n_O , *es un valor óptimo* pues lo encontraron luego de buscarlo en el espacio del problema. Lo malo, es que *como cualquier proceso de búsqueda basado en optimización, puede quedar estancada en óptimos locales*, además de que *el encontrar la solución óptima pueda tomar demasiado tiempo y gastar muchos recursos*. Los métodos heurísticos, en cambio *buscan determinar, usando información local del problema, un valor al menos adecuado* para el tamaño de la capa oculta [9]. Este valor quizá sea subóptimo, pero es un valor al menos adecuado incluso para iniciar el proceso de búsqueda de los métodos de prueba y error. Los métodos heurísticos *tienen la ventaja de ser algunas veces muy rápidos y fáciles de aplicar, algunos de ellos aplican en general*, sin importar realmente la naturaleza del problema a resolver. Tienen la desventaja, en ciertos casos, de carecer de formalismo matemático, en otras ocasiones, *exigen preprocesamiento de la información de entrenamiento, lo que los hace relativamente lentos* para arrojar resultados.

Los métodos heurísticos se dividen en dos grandes clases: las llamadas *reglas de oro*, que son fórmulas muy simples que hacen a la cantidad n_O dependiente únicamente de las cantidades n_E y n_S , puesto que ambas son información ya disponible del problema. Por otro lado están las llamadas *reglas heurísticas*, que en lugar de hacer depender a n_O únicamente de n_E y n_S , usan otro tipo de información más profunda del problema, más allá de las capas de entrada y salida de la red, para establecer de modo más confiable el valor del tamaño de la capa oculta.

2.1. Reglas de oro

Entre las *reglas de oro* más sobresalientes se tienen las siguientes [14, 19, 20]:

$$\text{Reglas de oro} \begin{cases} \text{Promedio: } n_O = \frac{n_E + n_S}{2} \\ \text{Dos tercios: } n_O = \frac{2n_E}{3} + n_S \\ \text{Piramidal } n_O = \sqrt{n_E n_S} \end{cases}$$

Existen bastantes variantes de las reglas de oro, pero lo que caracteriza a todas ellas es que usan la cantidad de elementos en la capa de entrada y la cantidad de neuronas en la capa de salida, para dar lugar al tamaño de la capa oculta, y es que la información de entrada define la complejidad del problema, mientras que la información de salida define la dificultad para separar las clases, si el problema es uno de clasificación [21]. Típicamente la relación que existe entre estos tres elementos es que el tamaño de la capa oculta se encuentra en algún lugar entre el tamaño de la capa de entrada, y el de la capa de salida. Estas reglas tienen la ventaja de que son sencillas de aplicar, son rápidas para producir un resultado, y se aplican a cualquier problema. Sin embargo tienen una gran desventaja: carecen por completo de una justificación matemática, y por tanto, no hay seguridad de obtener un valor realmente adecuado para el tamaño de la capa oculta [6]. Por otro lado, existen también las llamadas reglas heurísticas, estas son metodologías para llegar a una buena aproximación al tamaño óptimo de la capa oculta, pero a diferencia de las reglas de oro, llevan a cabo un análisis muy específico sobre el problema a resolver, detrás de sus formulaciones o algoritmos para establecer ese valor n_O .

Así como existen variantes de las reglas de oro, existen aún más variantes de reglas heurísticas para establecer el tamaño de la capa oculta de una red SLFN. A continuación se mencionan algunas de estas formas heurísticas de manejar la información del problema a resolver, para llegar a un tamaño para la capa oculta de la red.

- (1) Usando *selección de variables*: aquí lo que se busca es identificar, en el patrón de entrada $\mathbf{X} \in \mathbb{R}^{n_E}$, cuales de esos elementos x_i para $i = 1, 2, \dots, n_E$ en el vector de entrada son verdaderamente significativos, la cantidad de estos últimos, por ejemplo n_i , es el tamaño de la capa oculta, así pues $n_O = n_i$. Existen distintas maneras de obtener el valor n_i , por ejemplo usando bosques aleatorios [22], usando análisis de componentes principales [23], usando árboles de decisión [24], también pueden usarse técnicas de análisis de dependencias y correlaciones [10].
- (2) Usando *una relación conexiones/conjunto de entrenamiento* para, al menos, limitar el tamaño de la capa oculta. Por ejemplo, si n_W es el total de conexiones, la suma de $\mathbf{W}^1$ y $\mathbf{W}^2$,

y n_P la cantidad de pares patrón/target en C_E , que denota al *conjunto de entrenamiento* usado, entonces deberá cumplirse que $n_W \leq \frac{n_P}{30}$ [16].

- (3) Usando *una tolerancia al error de clasificación*, por ejemplo, si la tolerancia al error cuadrático medio es $tole \in \mathbb{R}$ entonces sería posible establecer que $n_O \leq \frac{n_P * tole}{n_P + n_S}$ [25].
- (4) Usando *la cantidad de clases a separar*, en caso de que la red SLFN fuera un clasificador, separando n_P clases C_i para $i = 1, 2, \dots, n_C$, entonces hacer $n_O = n_C$ [25].
- (5) Usando *validación cruzada* para encontrar el tamaño apropiado, luego de llevar a cabo una cierta cantidad de pruebas sobre diferentes arquitecturas candidatas [26]. Específicamente, usando validación cruzada *k-fold*. Este es un proceso parecido al siguiente. Establecer diferentes valores para n_O , como en $sizes = \{1, 2, \dots, tam_{max}\}$. Separar el conjunto C_E en k grupos. Para cada red SLFN con $n_O \in sizes$: entrenar la red usando $k - 1$ grupos como datos de entrenamiento, y el otro grupo restante como datos de prueba. Repetir lo anterior k veces. Seleccionar la red SLFN cuyo valor n_O tuvo el mejor desempeño.
- (6) Usando *factores de diseño*, si la red SLFN ha de funcionar como clasificador. Los factores de diseño sirven para identificar dentro de la información disponible a clasificar, que aspectos son verdaderamente significativos en lo que a distinguir las clases se refiere. Una de estas metodologías es el *método Taguchi* [27, 28] para identificar factores de diseño.
- (7) Usando alguna variante de los *algoritmos genéticos* para encontrar el tamaño adecuado de una red SLFN. Esto implica que el valor para n_O es encontrado luego de un proceso de búsqueda aleatoria. En esta clase de metodologías existen numerosas variantes, por ejemplo, un algoritmo genético en combinación con métodos de programación lineal [29], o como un simple algoritmo genético clásico, cuya población es conformada por redes SLFN con capas ocultas de distintos tamaños [30]. En [31], una combinación de algoritmos genéticos junto con una *evolución gramática* es usada para generar una red SLFN con un tamaño de capa oculta optimizado.

2.2. Métodos de prueba y error

Los métodos de prueba y error tienen un razonamiento muy sencillo: probar un tamaño de capa oculta entrenando la red con ese tamaño específico. Si el error es lo suficientemente pequeño, se considerará a este tamaño como valor para n_O , de lo contrario, se prueba con otro, llevando a cabo una vez más este mismo procedimiento [32, 33]. A modo de algoritmo, un método de prueba y error funciona como lo indica el Algoritmo 1 a continuación.

Algoritmo 1: Algoritmo para un método de prueba y error

Data: C_E :conjunto de entrenamiento
Result: n_O : el mejor valor para el tamaño de la capa oculta

- 1 Establecer un valor inicial para el $mejor_{ecm}$, de preferencia, alto;
- 2 Hacer $listo = False$;
- 3 Hacer $n_O = 0$;
- 4 **while** $listo == False$ **do**
- 5 Hacer $n_O = n_O + 1$;
- 6 Entrenar la red SLFN con este valor para n_O ;
- 7 Probar la red y calcular su ecm ;
- 8 **if** $ecm < mejor_{ecm}$ **then**
- 9 Hacer $mejor_{ecm} = ecm$;
- 10 **else**
- 11 Hacer n_O actual el valor final para el tamaño la capa oculta;
- 12 Hacer $listo = True$
- 13 **end**
- 14 **end**

Como el Algoritmo 1 muestra, el proceso de prueba y error es un proceso iterativo que consiste en que en cada iteración una versión de la red es construida y probada, si luego de esta prueba el desempeño de la red ya no puede ser mejorado, entonces se considera que el tamaño de la última red probada es el tamaño ideal. Nótese como el proceso se detiene cuando el desempeño de la red SLFN actual ya no es mejor al de la versión anterior, esto hace que los métodos de prueba y error queden, dependiendo de la forma en que se actualiza n_O , estancados en un óptimo local.

Por otro lado, existen abundantes variantes de los métodos de prueba y error. Todo consiste en aquello de seleccionar un valor para n_O , el tamaño de la capa oculta de la red SLFN en producción, justo en el paso 5 del algoritmo. Existen tres formas de elegir como llevar a cabo esta elección, lo que da lugar a tres maneras de implementar un método de prueba y error. Estos son resumidos en el siguiente cuadro sinóptico.

$$\text{Métodos de prueba y error} \left\{ \begin{array}{l} \text{Constructivos} \\ \text{De poda} \\ \text{Mixtos} \end{array} \right.$$

Los llamados *métodos constructivos* siempre eligen *agregar neuronas a la capa oculta*, lo que significa que en cada iteración hacen crecer la capa oculta. Los métodos constructivos sue-

len iniciar con un capa oculta muy pequeña como tamaño de capa oculta inicial. Por su parte, los llamados *métodos de poda* eligen *eliminar neuronas de la capa oculta*, es decir que hacen decrecer el tamaño de la capa oculta en cada iteración. Los métodos de poda suelen iniciar con una red cuyo tamaño inicial es muy grande como tamaño de capa oculta inicial. Luego están los *métodos mixtos*, estos en cada iteración dejan que la capa oculta crezca, para luego empezar a podarlos si cierta condición se cumple. Alternativamente pueden comenzar podando la capa oculta, para luego hacerla crecer si cierta condición es detectada. Los métodos mixtos suelen usar algunas otras herramientas como los algoritmos genéticos, para controlar el que el proceso no quede estancado en un óptimo local en etapas tempranas del proceso.

Con respecto a los métodos constructivos, a veces, también llamados *métodos de crecimiento*, la idea fundamental es agregar más neuronas a la capa oculta de la red en cada iteración, comenzando con una red SLFN cuya capa oculta posee un tamaño mas bien pequeño. Dos métodos sobresalen entre estos: el método *cascade correlation algorithm* ó método *CCA* [34], y el método *cascade 2* [34], en realidad una variante de CCA. Ambos métodos añaden una nueva neurona oculta en una nueva capa oculta, es decir que crecen tanto en capas como en neuronas en cada capa oculta; por lo que en realidad estas dos metodologías no son significativas para una red SLFN que se limita a una capa de neuronas ocultas. Existen otros métodos constructivos, como el método *dynamic node creation* o método *DNC* [35]. El método DNC agrega una nueva neurona oculta en una capa oculta ya existente, por lo que es más apropiada para construir redes SLFN, ya que se limita a agregar neuronas en la capa oculta que se le indique, por ejemplo a la única capa oculta de la red SLFN. Muy parecido a DNC es el método *FNNCA* [36], que también hace crecer la capa oculta de la red SLFNN, agregando un nodo a la vez y entrenando en cada ocasión a la red, probándola posteriormente, hasta que el desempeño de la red sea satisfactorio. Los métodos de crecimiento o constructivos requieren comenzar el proceso con una red SLFN más bien pequeña, en cuanto a su capa oculta, un problema con esto es que una red pequeña queda muy fácilmente atascado en un óptimo local [37].

Los métodos constructivos tienen algunas desventajas notables [38]:

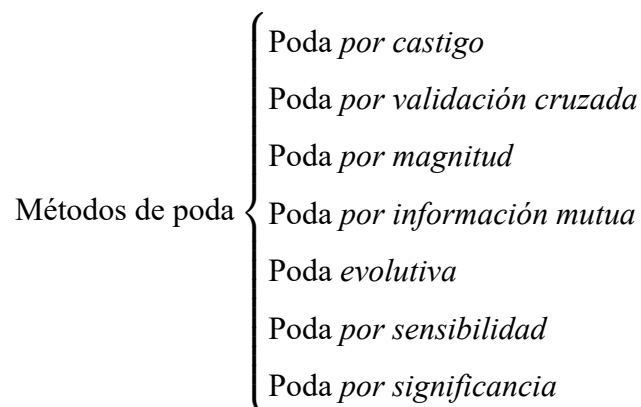
- (1) Requieren un tamaño inicial para la red SLFN que inicia todo el proceso. Este tamaño inicial no es trivial, más grande de lo necesario y la red encontrada no será la óptima.
- (2) Estos métodos, puesto que hacen crecer la red iterativamente para luego probarla, implican que el método de entrenamiento no se base en solo un método tal como el gradiente descendiente, sino una adaptación del mismo. Luego, no existe un único método de entrenamiento, sino que cada método de crecimiento usa el propio.
- (3) Algunos de estos métodos pueden producir redes MLP que, ni son SLFN y ni siquiera son

II. Estado de la técnica

redes con conexiones hacia adelante.

- (4) Algunos de estos métodos, al final terminan produciendo redes bastante complejas, más complejas de lo que una simple red SLFN como la planteada en este estudio necesita.
- (5) Y la que quizá es la desventaja más grande: puesto que pueden producir redes de tamaño inadecuado, la superficie de error de tales redes suele contener bastantes óptimos locales, mismos en los que el entrenamiento puede quedar estancado muy fácilmente.

Los métodos de poda, por su parte, se basan en la idea de eliminar neuronas de la capa oculta que comienza como una red SLFN con una capa oculta muy grande, más grande de lo necesario, eliminando luego neuronas en el proceso iterativo hasta que la red se encuentre con el tamaño adecuado. *En la mayoría de los casos lo que los métodos de poda eliminan son conexiones hacia y desde una neurona oculta*, de modo que eventualmente, algunas de esas neuronas quedan desconectadas hacia o desde la capa de entrada y la capa de salida, por lo que esta neurona debe ser eliminada de la capa oculta. La poda de una red SLFN comienza con una red SLFN con una capa oculta grande. Luego, es podada en busca de una capa oculta más pequeña. La poda puede ocurrir de distintas maneras. Según [37], existen distintas maneras de establecer que neuronas o conexiones deben ser eliminadas. Esto da lugar a que los métodos de poda se clasifiquen según lo indica el cuadro sinóptico a continuación:



En la *poda por castigo*, a la función objetivo a ser minimizada durante el entrenamiento, por ejemplo al error cuadrático medio *ecm*, se le agrega un término de castigo, y debido a este término aquellas conexiones sin mucho impacto serán anuladas durante el entrenamiento, véase [39, 40] para ejemplos de esto. En la *poda por validación cruzada*, una etapa de prueba para una nueva red podada compite con una versión anterior de la misma, con la capa oculta mas grande. El conjunto de entrenamiento es dividido en un subconjunto de prueba y otro de validación. El de validación es usado para comparar la versión podada de la red con la versión anterior. Solo en

II. Estado de la técnica

caso de que esta versión podada resulta mejor que la anterior, el proceso continua una iteración más. Véase [41] para un ejemplo de esto. En la poda por magnitud, luego de entrenada la red aquellas conexiones o pesos que sean menores a una tolerancia se consideran irrelevantes y son eliminados, véase por ejemplo, [40, 42] para un ejemplo de este estilo de poda.

Por su parte, en la poda basada en información mutua se busca relacionar, mediante un análisis de covarianzas entre las respuestas de las neuronas en la capa oculta, usando para ello una descomposición en valores singulares. El rango de la matriz de covarianzas establece el valor óptimo de neuronas en la capa oculta. Véase [43, 44] para ejemplos de esto. En la poda evolutiva se tiene un proceso de poda que es controlado por un algoritmo genético, es decir, la red SLFN es formada mediante un proceso de evolución. Véase [45] para un ejemplo de esto.

La poda por sensibilidad busca aquellas neuronas o conexiones cuyo efecto en la función objetivo, es decir, el error cuadrático medio, es casi nulo o imperceptible para luego podarlos de la capa oculta. Dos de los métodos de poda por sensibilidad más conocidos y utilizados son, por un lado el método *Optimal Brain Surgeon* o método *OBS*, [46], y el método *Optimal Brain Damage* o método *OBD* [47].

Parecido pero no igual, son los métodos por significancia. Estos también localizan conexiones hacia o desde la capa oculta, cuyo valor de significancia sea menor que un cierto umbral, para luego eliminarlos de la misma. Aquí, significancia tiene un sentido amplio, es cualquier medición tomada entre la información de entrada y las respuestas de las neuronas en la capa oculta. Por ejemplo, en [48] la significancia es calculada a partir de comparar mediciones estadísticas. Dado un patrón de prueba a la red SLFN, un patrón conocido de ruido, la respuesta de la capa oculta a tal patrón de prueba es calculada. Lo interesante en esto es que la respuesta de la capa oculta ofrece un valor de significancia para las conexiones de la red SLFN, hacia y desde la capa oculta. Aquellas conexiones con significancia más baja que un umbral son eliminadas de la red. En [49], se propone otro método para encontrar neuronas con poca significancia en una red MLP con una o mas capas ocultas. Aquí la significancia de las neuronas ocultas se obtiene a partir de la respuesta de la neurona a eliminar, usando para ello una función de activación sigmoideal junto con las conexiones de esta neurona con las neuronas en la siguiente capa de neuronas.

Los métodos de poda tienen una serie de desventajas, entre ellas se pueden enumerar las siguientes [38]:

- (1) Deben, necesariamente, iniciar con una red cuya capa oculta sea más grande de lo necesario. Pero ¿cual es ese tamaño? Aunque una capa oculta demasiado grande hará que el método tome más tiempo para encontrar una solución.
- (2) Dadas las conexiones iniciales, si el valor de estas conexiones es muy pequeño, no garan-

tiza que el método converja a una solución.

- (3) Todos los métodos de poda implican un tiempo de procesamiento relativamente largo, y requieren un buen espacio en memoria durante su procesamiento.

En cuanto a los métodos mixtos, estos llevan a cabo la construcción de una red SLFN en dos etapas, la primera haciendo crecer la red, y la segunda podando las neuronas insignificantes. Este tipo de enfoque requiere iniciar con una red con capa oculta de tamaño reducido. Existe la otra opción, la de iniciar con una red mas grande de lo necesario, luego podarla en una primera etapa, para pasar a hacerla crecer en una segunda etapa. Véase, por ejemplo en [50], un método basado en cascade correlation. Este método permite construir redes MLP con múltiples capas ocultas. Lo hace haciendo crecer las capas ocultas, en una primera etapa, y luego podando de las capas ocultas las neuronas que no resulten significantes, en lo que al error cuadrático medio se refiere. En [51], un método para hacer crecer primero, la capa oculta de una red SLFN, para podarlo posteriormente. Este método es particularmente útil para entrenar una red MLP usando patrones o muestras esparcidas en el espacio del problema. La red SLFN inicia con una capa oculta con un tamaño específico luego es entrenada de la manera usual, luego aquellas conexiones cuya magnitud sea menor a un umbral son eliminadas de la red.

Los métodos mixtos también presentan desventajas, por ejemplo [38]:

- (1) Deben comenzar con un tamaño adecuado para la red SLFN inicial.
- (2) Dependen, en gran medida, del algoritmo de aprendizaje o entrenamiento, para converger a una solución.
- (3) Ocasionalmente, cuando si convergen a una solución, esta resulta ser subóptima.

2.3. Reglas heurísticas

Parecidas a las reglas de oro en el sentido de que intentan establecer el tamaño de la capa oculta de la red SLFN mediante un único proceso que ocurre antes de entrenar y utilizar la red en cuestión, las reglas heurísticas se posicionan entre los métodos de prueba y error y las reglas de oro. A diferencia de las reglas de oro, las reglas heurísticas intentan extraer el máximo de la información disponible del problema a resolver, en particular para la presente investigación, del problema de clasificación de patrones a resolver. A diferencia de los métodos de prueba y error, no gastan tiempo buscando la mejor configuración entre diferentes opciones. Las reglas heurísticas tienen la particularidad de que requieren un *pre-procesamiento*. El pre-procesamiento se refiere

a una forma de *análisis previo de los datos de entrenamiento, en busca de cierta información que se cree resulta útil para algún propósito*, en el caso que atañe a esta investigación, directamente el tamaño de la capa oculta, o alguna otra forma de información que ayude a establecer su tamaño.

En este sentido, se tiene, en [52], una regla básica: no pueden existir mas neuronas en la capa oculta, que muestras de entrenamiento. Por otro lado, en [53] se propone que conocida la cantidad de conexiones en la red SLFN, deberían existir de cinco a diez veces más muestras de entrenamiento. En [54] se menciona que un enfoque basado en particionar el espacio de soluciones del problema usando árboles de decisión, puede ayudar. Por ejemplo, la cantidad de neuronas en la capa oculta equivaldría a la cantidad de árboles de decisión que particionaron el espacio estado. Otros prefieren un enfoque más estadístico, por ejemplo en [23] se usa la descomposición de valores propios de la matriz de covarianza de la respuesta de las neuronas en la capa oculta. En [13] usan programación evolutiva para establecer el tamaño de la capa oculta. En [11] usan la estrictamente decreciente relación entre el error de aproximación de muestras, y la cantidad de neuronas en la capa oculta, para guiar un proceso de búsqueda de la arquitectura óptima de una red SLFN, utilizando un algoritmo genético.

Capítulo 3

Estructuras y técnicas utilizadas en este estudio

Una red MLP (*MultiLayered Perceptrón*), o red neuronal multicapa, es una forma de red neuronal artificial ó una *máquina que aprende*. En el presente capítulo se describen y analizan los aspectos importantes de las estructuras y técnicas específicas consideradas en las que se apoya el desarrollo del método propuesto.

3.1. Fundamentos de una red MLP

Una red MLP *es una forma de red neuronal artificial que utiliza aprendizaje supervisado, por medio de una técnica de entrenamiento basada en desempeño y basada en una arquitectura multicapa con conexiones hacia adelante* [1]. Las siguientes subsecciones explican cada uno de estos conceptos.

3.1.1. La neurona artificial

El componente fundamental de las RNAS (redes neuronales artificiales), es la neurona artificial. Una *neurona artificial* o simplemente, una neurona. Esta es el elemento básico de procesamiento, y está compuesta por:

- (1) Un patrón de entrada $\mathbf{X} \in \mathbb{R}^{n_E}$.
- (2) Un patrón de conexiones $\mathbf{W} \in \mathbb{R}^{n_E}$.
- (3) Un acumulador de entrada $net = \mathbf{W}^T \mathbf{X} = \sum_{i=1}^{n_E} w_i x_i$.

III. Estructuras y técnicas utilizadas en este estudio

- (4) Una función de activación $f(.) : \mathbb{R}^{n_E} \rightarrow \mathbb{R}$.
- (5) Una respuesta $y \in \mathbb{R}$.

La Figura 3.1 representa los componentes de la neurona artificial, a modo de diagrama de componentes. La neurona genera en y la respuesta a un patrón de entrada $\mathbf{X}$. La respuesta se genera aplicando una función de activación $f(.)$ a la acumulación net de cada elemento de entrada en $\mathbf{W}$, modificado de acuerdo con el respectivo elemento en las conexiones $\mathbf{W}$, $y = f(net) = f(\mathbf{W}^T \mathbf{X}) = f(\sum_{i=1}^{n_E} w_i x_i)$. Es obvio que la respuesta y de la neurona al patrón de entrada $\mathbf{X}$, depende directamente de la función de activación usada en dicha neurona.

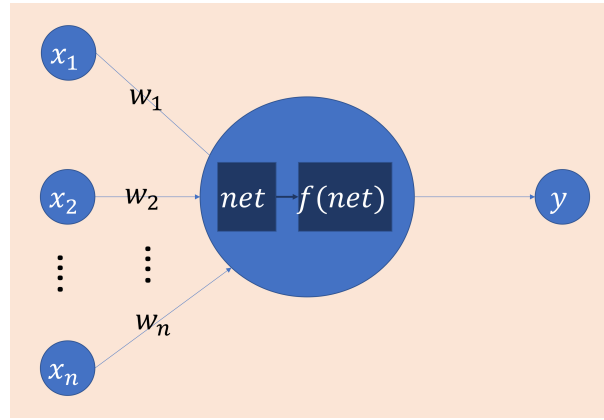


Figura 3.1: Una neurona artificial

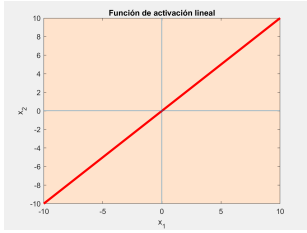
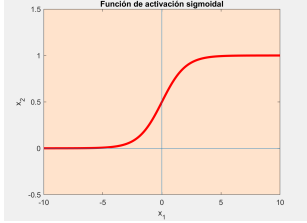
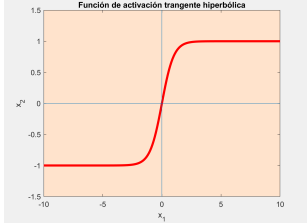
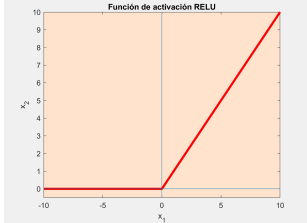
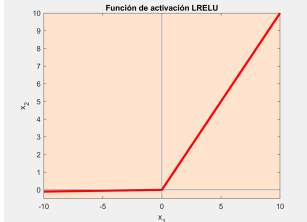
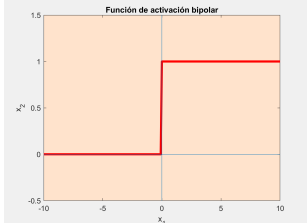
La función de activación dictamina lo que la neurona puede hacer [55], es decir que la neurona está limitada por las características de la función de activación. Las opciones que se tienen en la elección de la función de activación, se pueden clasificar en:

$$\text{Funciones de activación} = \begin{cases} \text{Funciones lineales} \\ \text{Funciones no lineales} \end{cases}$$

Por un lado, las *funciones lineales* tiene la ventaja de ser diferenciables y fáciles de implementar, pero tienen la desventaja de que son demasiado simples y no son capaces de capturar la complejidad en la relación entrada/salida de muchos problemas reales [56]. Las *funciones no lineales* no son necesariamente diferenciables ni simples de implementar, pero en cambio, son capaces de representar mejor la complejidad en la relación entrada/salida de muchos problemas reales [56, 57]. Aún así, las funciones no lineales varían mucho en cuanto a capacidades. No es extraño que la aplicación de las funciones lineales sea muy limitada en comparación con los de las funciones no lineales. La Tabla 3.1 a continuación muestra algunas de las funciones de activación disponibles en RNAs.

III. Estructuras y técnicas utilizadas en este estudio

Tabla 3.1: Funciones de activación populares

Función	Expresión	Gráfica
Lineal	$f(net) = net$	
Sigmoidal	$f(net) = \frac{1}{1+e^{-net}}$	
Tangente hiperbólica	$f(net) = \frac{\sinh(net)}{\cosh(net)}$	
RELU	$f(net) = \max(0, net)$	
LRELU	$f(net) = \begin{cases} a * net & \text{si } net \leq 0 \\ net & \text{si } net > 0 \end{cases}$	
Escalón unitario	$f(net) = \begin{cases} 0 & \text{si } net \leq 0 \\ 1 & \text{si } net > 0 \end{cases}$	

III. Estructuras y técnicas utilizadas en este estudio

En la Tabla 3.1 se presenta la *función lineal*. Esta es una función muy simple, es solo un seguidor del argumento net de la función. Su rango de salida es infinito, $-\infty < f(net) < \infty$. Esto provoca una respuesta excesiva en ciertos casos, además es demasiado simple como para capturar la complejidad de muchos sistemas. Es una *función monotónica*, mas su derivada no lo es. Esto se expresa como en la Ecuación 3.1:

$$f(net) = net \quad (3.1)$$

La función de activación lineal encuentra su mayor aplicación en la capa de salida en RNAs estilo MLP y en redes Adaline, en donde el argumento se espera acotado y procesado. La función *escalón unitario* o también conocida como función *bipolar*, tiene un rango acotado y discreto, $f(net) \in \{0, 1\}$. Es una función no diferenciable, pero monotónica [1, 57]. Esto se expresa como e la Ecuación 3.2.

$$f(net) = \begin{cases} 0 & \text{si } net \leq 0 \\ 1 & \text{si } net > 0 \end{cases} \quad (3.2)$$

El escalón unitario produce una respuesta perfecta para clasificadores binarios, por lo que es muy común en redes clasificadoras de este tipo [58, 57]. Luego se tiene la *función sigmoideal*, que es una función muy popular en RNAs. Es una función continua con un rango acotado $0 < f(net) < 1$. Además es una función monotónica, mas no así su derivada [1, 58]. Su expresión es como en la Ecuación 3.3.

$$f(net) = \frac{1}{1 + e^{-net}} \quad (3.3)$$

Puede decirse que la función sigmoideal es una versión continua del escalón unitario, pero sus características son muy diferentes. La *función tangente hiperbólica* es muy parecida a la función sigmoideal, pero con un rango más amplio, $-1 < f(net) < 1$. Se dice que es una función *centrada en cero*, a diferencia de las funciones sigmoideal y escalón unitario que se centran en $net = 0,5$ [1, 58]. Es una función monotónica con derivada no monotónica. Su comportamiento se expresa como en la Ecuación 3.4.

$$f(net) = \frac{\sinh(net)}{\cosh(net)} \quad (3.4)$$

Las funciones sigmoideal y tangente hiperbólica encuentran su aplicación principalmente en redes neuronales MLP. La *función ReLU* que es una función semi-lineal y se expresa como en la Ecuación 3.5.

$$f(net) = \max(0, net) \quad (3.5)$$

Esta función ReLU es una versión acotada de la función lineal, su rango es $0 \leq f(net) < \infty$. Es una función monotónica con derivada también monotónica. Es muy popular para redes neuronales profundas [58]. Existe una versión alterada de la función ReLU, conocida como *función LReLU* que se expresa como en la Ecuación 3.6.

$$f(net) = \begin{cases} a * net & \text{si } net \leq 0 \\ net & \text{si } net > 0 \end{cases} \quad (3.6)$$

Es una función parecida a ReLU excepto que la función LReLU admite respuestas negativas, a diferencia de la función ReLU. Es monotónica al igual que su derivada. En cuanto al factor $a \in \mathbb{R}$, el llamado *factor de fuga* se recomienda $a = 0,01$, aunque otros valores están permitidos [56]. Al igual que con la función ReLU, la función LReLU encuentra su mayor aplicación en redes neuronales profundas.

3.1.2. Capas de neuronas

Excepto por las redes Perceptrón y Adaline, ninguna red RNA se compone por una sola neurona artificial. Para que algo realmente útil ocurra es necesario agrupar más de una neurona en un arreglo conocido como *capa de neuronas*. Una capa de neuronas es un arreglo de n neuronas, organizadas de modo que todas ellas reciben el mismo patrón de entrada y cada una de ellas genera su propia respuesta aislada [59], como lo muestra la Figura 3.2.

La Figura 3.2 muestra una capa de neuronas compuesta por 4 neuronas, cada una de ellas toma el mismo patrón $\mathbf{X} \in \mathbb{R}^3$ de entrada, $\mathbf{X} = [x_1, x_2, x_3]^T$, y producen un patrón de respuesta $\mathbf{Y} \in \mathbb{R}^4$. Cada neurona n_i para $i = 1, 2, 3, 4$ en la Figura 3.2 produce su propia respuesta y_i , aunque estas se acumulan en un patrón de respuesta, en este caso $\mathbf{Y} \in \mathbb{R}^4$ $\mathbf{Y} = [y_1, y_2, y_3, y_4]^T$.

Si bien cada una de las neuronas en la capa de neuronas es independiente de las otras, *todas las neuronas en una misma capa usan la misma función de activación*, de modo que la respuesta de todas ellas sean igualmente significativas. Así si cada neurona utiliza la función de activación $f(\cdot)$, el patrón de respuesta $\mathbf{Y}$ suele expresarse como en la Ecuación 3.7.

$$\mathbf{Y} = f(\mathbf{net}) = f(\mathbf{WX}) \quad (3.7)$$

Se puede ver en la Ecuación 3.7, que $\mathbf{net}$ es la agregación de las respuestas de cada neurona en la capa, $\mathbf{net} = [net_1, net_2, net_3, net_4]^T$. El cálculo del valor neto de toda la capa se puede

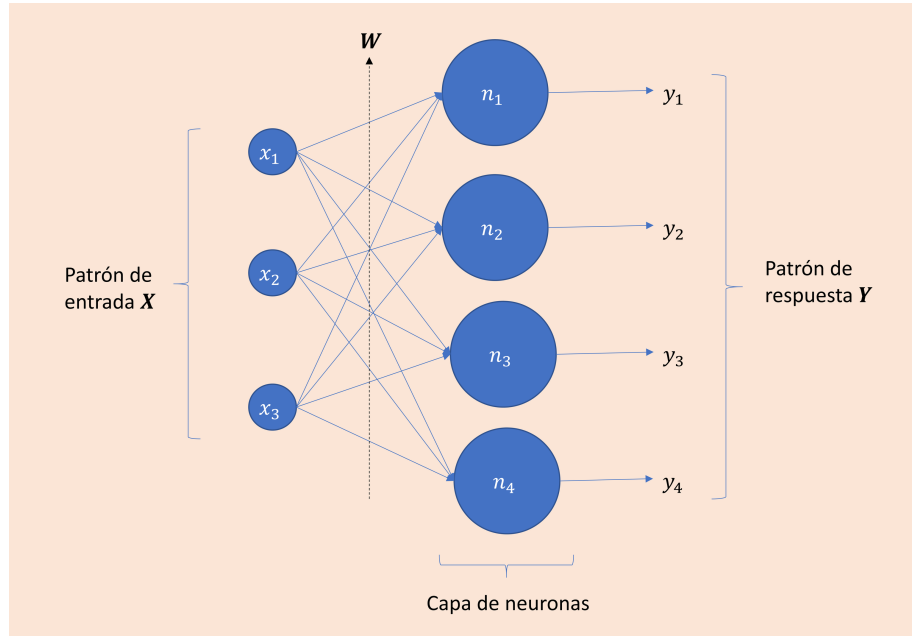


Figura 3.2: Una capa con $n = 4$ neuronas, con un patrón de entrada $\mathbf{X} \in \mathbb{R}^3$, y una respuesta $\mathbf{Y} \in \mathbb{R}^4$

obtener fácilmente multiplicando la matriz de conexiones $\mathbf{W}$, por el patrón de entrada a la capa $\mathbf{X}$, con el entendido de que si el patrón de entrada $\mathbf{X}$ tiene n_E elementos y la capa de neuronas ocultas tiene n_O elementos, entonces $\mathbf{W} \in \mathbb{R}^{n_O \times n_E}$, de modo que en la matriz de conexiones $\mathbf{W}$ existe un renglón por cada neurona en la capa oculta y cada uno de ellos son las conexiones de la i -ésima neurona en la capa de neuronas con el patrón de entrada $\mathbf{X}$, $\mathbf{W} = [\mathbf{W}_1^T, \mathbf{W}_2^T, \dots, \mathbf{W}_{n_O}^T]$, donde $\mathbf{W}_i$ para $i = 1, 2, \dots, n_O$ es el vector con las conexiones del patrón de entrada $\mathbf{X}$, a la i -ésima neurona n_i .

3.1.3. Redes neuronales artificiales

Una RNA utiliza un conjunto de *neuronas artificiales* organizadas a modo de capas, compartiendo información entre ellas para producir una respuesta $\mathbf{Y} \in \mathbb{R}^{n_S}$ a un patrón de entrada $\mathbf{X} \in \mathbb{R}^{n_E}$, en donde $n_E \in \mathbb{Z}$ es la cantidad de elementos en los patrones de entrada y $n_S \in \mathbb{Z}$ es la cantidad de elementos en los patrones de respuesta de la red.

En la Figura 3.3a se presenta una RNA con tres capas de neuronas y cada capa contiene solo tres neuronas. En cada capa las neuronas se nombran n_i^j , de modo que n_i^j se refiere a la i -ésima neurona en la j -ésima capa.

En la Figura 3.3b se muestra la denominación usual para las capas de una RNA. A la primer capa de neuronas de una RNA se le suele llamar la *capa de entrada*, que *contiene los elementos*

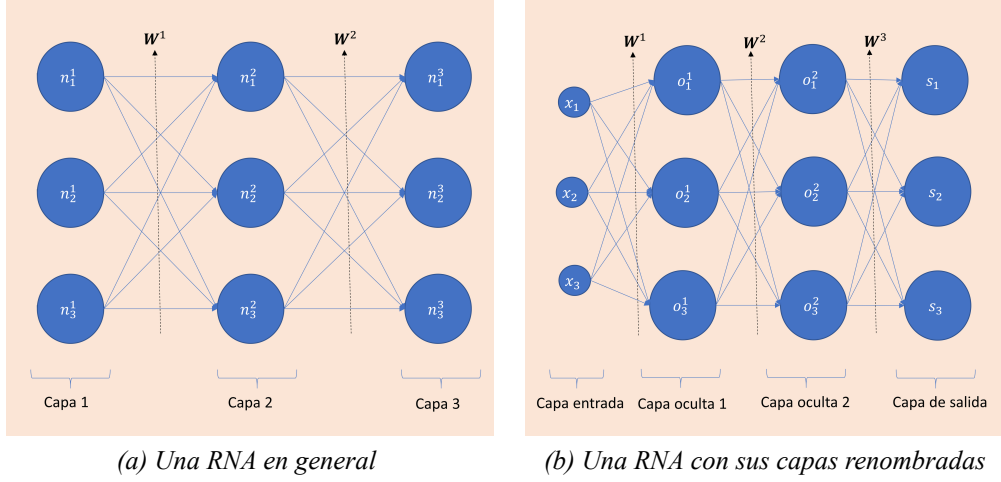


Figura 3.3: Redes RNA, en general, y con su nomenclatura oficial

del patrón de entrada. Así que esta capa de entrada no es una capa de neuronas como el resto de las capas de una RNA, sino que únicamente contiene los mismísimos elementos del patrón de entrada a la red. También se tiene una *capa de salida*. La capa de salida *contiene a las neuronas que producen la respuesta final de la red*. Entre la capa de entrada y la capa de salida, pueden existir una o más *capas de neuronas ocultas*. Las capas ocultas cumplen el papel de procesamiento en etapas, así la primer capa produce una primera transformación en el patrón de entrada, la segunda, una segunda transformación, así sucesivamente hasta llegar a la capa de salida, que lleva a cabo la última transformación y que produce la respuesta de la red.

3.1.3.1. Propagación hacia adelante

En la propagación hacia adelante la respuesta de la i -ésima capa de neuronas $\mathbf{O}^i$ depende de la $i - 1$ -ésima capa de neuronas. Comenzando en la capa de entrada cuyo índice es 0, su respuesta es simplemente $\mathbf{O}^0 = \mathbf{X}$. A partir de $i = 1$ la respuesta de la i -ésima capa de neuronas es como lo indica la Ecuación 3.8.

$$\mathbf{O}^i = f_i(\mathbf{net}^i) = f_i(\mathbf{W}^i \mathbf{O}^{i-1}) \quad (3.8)$$

En la Ecuación 3.8 las conexiones $\mathbf{W}^i \in \mathbb{R}^{n_{O_i} \times n_{O_{i-1}}}$ es el grupo de conexiones entre la capa $i - 1$ y la capa i , cada una de ellas conteniendo $n_{O_{i-1}}$ y n_{O_i} neuronas respectivamente. Considerando la última capa con índice k , ocurre que $\mathbf{Y} = f_k(\mathbf{W}^k \mathbf{O}^{k-1})$. A modo de algoritmo, esto ocurre como en el Algoritmo 2.

Así pues, las llamadas *redes de conexiones hacia adelante* tienen la particularidad de que la

Algoritmo 2: Algoritmo de propagación hacia adelante

Data: $\mathbf{X}$: patrón de entrada, k : número de capas

Result: $\mathbf{Y}$: respuesta de la red

```

1 Hacer  $\mathbf{O}^0 = \mathbf{X}$ ;
2 for  $i = 1$  to  $k$  do
3   | Hacer  $\mathbf{O}^i = f_i(\mathbf{W}^{i-1}\mathbf{O}^{i-1})$ ;
4 end
5 Hacer  $\mathbf{Y} = \mathbf{O}^k$ ;

```

información fluye como ha sido antes descrito: de capa a capa, únicamente entre capas consecutivas, y *sin retroalimentaciones* entre capas. Las redes MLP son redes con conexiones hacia adelante, por lo que este será el único estilo de RNA tratado en esta investigación [1, 59].

3.2. Entrenamiento en redes MLP

Existen algunas formas de RNAs populares actualmente, consideradas como un estilo de RNA separado de las redes MLP, pero que en realidad son una red MLP, por ejemplo, las SVM (*máquinas de vectores soporte*) [60], y las ELM (*máquinas de aprendizaje extrema*) [60]. Ambas son básicamente redes MLP excepto que se limitan a usar una sola capa oculta y su método de entrenamiento difiere bastante del de una red MLP tradicional. Las redes MLP se pueden definir listando los puntos que mejor establecen lo qué son y lo qué pueden hacer.

- (1) Son *redes con conexiones hacia adelante*, lo que implica que la información fluye únicamente de la capa de entrada, hacia la capa de salida [1, 57].
- (2) Usan *entrenamiento supervisado*, lo que implica que, a fin de entrenarlas, se requiere un conjunto de entrenamiento conformado por pares $C_E = \{\{\mathbf{X}^i, \mathbf{T}^i\}\}$, para $i = 1, 2, \dots, n_P$, en donde $\mathbf{X}^i$ es el i -ésimo patrón de entrenamiento, $\mathbf{T}^i$ es el i -ésimo patrón de respuesta deseada, y n_P es el total de pares en el conjunto de entrenamiento [1].
- (3) El método de entrenamiento es *entrenamiento basado en desempeño*, lo que implica que usan alguna medida de error, durante el ajuste de los parámetros internos de la red MLP, usando *cualquier variante del método de retropropagación* [61, 1].

En las siguientes secciones los dos últimos puntos son desarrollados e ilustrados. El de redes con conexiones hacia adelante ya fué expuesto en secciones anteriores.

3.2.1. Aprendizaje en una red MLP

Una red que utiliza aprendizaje supervisado y basado en desempeño aprende haciendo correcciones si la respuesta actual $\mathbf{Y} \in \mathbb{R}^{n_s}$ difiere de la respuesta deseada $\mathbf{T} \in \mathbb{R}^{n_s}$, entonces la red está cometiendo un error y los parámetros internos de la red $\mathbf{W}^i$ deben ser corregidos de acuerdo con ese error [61, 1, 62].

El entrenamiento de una red MLP es un proceso iterativo de optimización basado en el llamado *gradiente descendiente* para encontrar el mejor valor para las conexiones $\mathbf{W}^i$, de acuerdo con el error $\mathbf{e}$ que la red comete. El algoritmo de entrenamiento general está listado en el Algoritmo 3.

Algoritmo 3: Algoritmo de entrenamiento para una red MLP

Data: C_E : conjunto de entrenamiento
Result: $\mathbf{W}^i$: conexiones optimizadas, para $i = 1, 2, \dots, k$

```

1 Establecer un valor para tole, la tolerancia al error;
2 Hacer listo = False, la bandera de control de ciclo;
3 Hacer epocas = 0;
4 while listo == False do
5   Hacer ecm = 0 for cada par  $\{\mathbf{X}^i, \mathbf{T}^i\} \in C_E$ : do
6     Calcular la respuesta  $\mathbf{Y}^i$  de la red al patrón de entrada  $\mathbf{X}^i$ ;
7     Calcular el error  $\mathbf{e}^i = \mathbf{T}^i - \mathbf{Y}^i$ ;
8     for  $j = 1$  to  $k$  do
9       Corregir  $\mathbf{W}^j$  de acuerdo al error  $\mathbf{e}^i$ ;
10    end
11    Calcular  $ecm = ecm + \mathbf{e}^{iT} \mathbf{e}^i$ ;
12  end
13  Hacer  $ecm = \frac{ecm}{2 * n_P}$ ;
14  if  $ecm \leq tole$  then
15    Hacer listo = True;
16  else
17    if epocas > max then
18      Hacer listo = True;
19    else
20      Hacer epocas = epocas + 1;
21    end
22  end
23 end

```

El Algoritmo 3 deja ver que el proceso de entrenamiento busca el mejor valor posible para las conexiones $\mathbf{W}^i$, de acuerdo con el error que la red comete para con cada patrón $\mathbf{X}^j$ en C_E

para $j = 1, 2, \dots, n_P$. Ese error se mide en términos del ECM (*error cuadrático medio*).

En el Algoritmo 3 el patrón $\mathbf{Y}^j$ se refiere a la respuesta real que la red ha generado al patrón de entrada $\mathbf{X}^j$, a diferencia del patrón $\mathbf{T}^j$ que es la respuesta deseada al mismo patrón de entrada, el *target* de dicho patrón. Cada iteración del Algoritmo 3 presenta un patrón y calcula la respuesta real a tal patrón, luego calcula el error y corrige cada conexión en la red de acuerdo a este error. A cada una de estas iteraciones se les conoce como *épocas* de entrenamiento. Además en el Algoritmo 3 la ejecución de un ciclo o iteración más es controlado por la bandera $listo \in \{True, False\}$. Esta bandera booleana depende de que el ECM de la última época sea menor que la tolerancia $tole \in \mathbb{R}$. Esa tolerancia debe ser especificada antes de la primer época de entrenamiento, e indica el mínimo valor tolerable de ECM que la red puede considerar aceptable para concluir el entrenamiento y considerar que la red MLP ha sido exitosamente entrenada. El ECM de la última época se calcula con la expresión en la Ecuación 3.9.

$$ecm = \frac{\sum_{j=1}^{n_P} \mathbf{e}^{jT} \mathbf{e}^j}{2n_P} \quad (3.9)$$

En realidad en el Algoritmo 3, el entrenamiento también termina si se pasa del máximo de épocas permitida. De esta manera asegura que el entrenamiento eventualmente terminará aunque el valor de las conexiones no sean las óptimas.

3.2.2. La regla delta generalizada

Esta regla permite modificar las conexiones de acuerdo con el error que la red comete [62, 1]. Considérese una conexión en particular dentro del grupo de conexiones $\mathbf{W}^i$, para $i = 1, 2, \dots, k$, por ejemplo la conexión w_{jl}^i , que conecta a la j -ésima neurona en la capa de neuronas $i - 1$ con la l -ésima neurona en la capa de neuronas i . Véase la Figura 3.4. Para modificar esta conexión w_{jl}^i de acuerdo al error $\mathbf{e}^j$, se usa la regla de optimización del gradiente descendiente, expresado como en la Ecuación 3.10:

$$w_{jl}^i = w_{jl}^i + \alpha \delta_w \quad (3.10)$$

En la Ecuación 3.10 el factor α se conoce como *factor de aprendizaje*, y controla la magnitud de la modificación de la conexión w_{jl}^i entre dos iteraciones o épocas consecutivas. El factor δ_w se refiere a la dirección hacia donde ocurre la corrección de la conexión w_{jl}^i . El factor de aprendizaje $\alpha \in \mathbb{R}$ puede ser establecido a *prueba y error* y normalmente se recomienda $0 < \alpha < 1$. Para la dirección de cambio en la conexión δ_w se aplica el siguiente razonamiento: *la dirección debe ser contraria a la dirección del máximo cambio*, lo que formalmente se expresa como en la Ecuación

III. Estructuras y técnicas utilizadas en este estudio

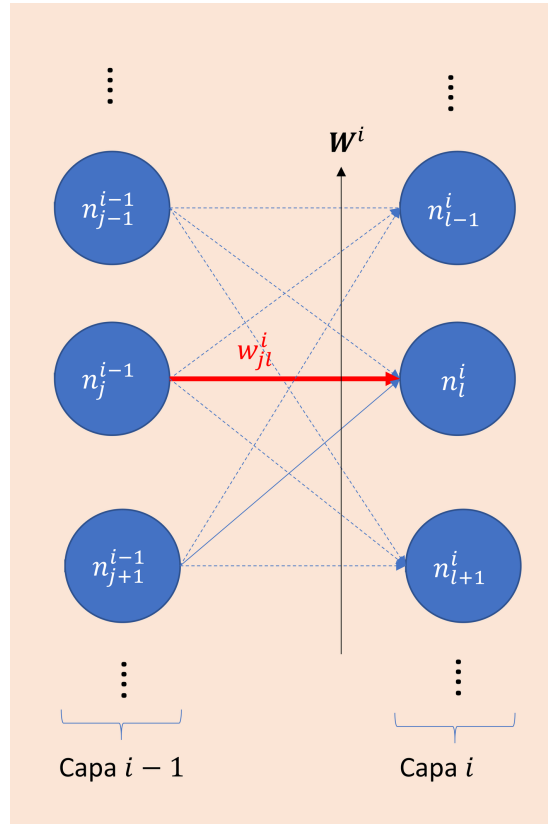


Figura 3.4: La conexión w_{jl}^i a corregir durante la explicación de la regla delta

3.11.

$$\delta_w = -\frac{d}{dw_{jl}^i}(ecm) \quad (3.11)$$

Y por tanto:

$$\begin{aligned}
 \delta_w = -\frac{d}{dw_{jl}^i}(ecm) &= -\frac{d}{dw_{jl}^i} \left(\frac{\sum_{j=1}^{n_P} e_l^{j^2}}{2n_P} \right) = \\
 &= -\frac{d}{dw_{jl}^i} \left(\frac{\sum_{j=1}^{n_P} (t_l^j - y_l^j)^2}{2n_P} \right) = \\
 &= -\frac{1}{n_P} (t_l^j - y_l^j) \left(-\frac{d}{dnet^i} (f_i(net^i)) \right) \left(\frac{d}{dw_{jl}^i} (net^i) \right) = \quad (3.12) \\
 &= \frac{1}{n_P} (t_l^j - y_l^j) f'_i(net^i) \frac{d}{dw_{jl}^i} \left(\sum_{j=1}^{n_P} w_{jl}^i x_j \right) = \\
 &= \frac{1}{n_P} e_l^j f'_i(net^i) x_j
 \end{aligned}$$

En la Ecuación 3.12 e_l^j se refiere al l -ésimo elemento del patrón de error $\mathbf{e}^j$, y_l^j se refiere al l -ésimo elemento del patrón de respuesta $\mathbf{Y}^j$ y t_l^j se refiere al l -ésimo elemento del patrón target $\mathbf{T}^j$. El factor f'_i en la Ecuación 3.12 es la derivada de la función de activación con respecto a net^i . Así pues, sustituyendo 3.12 en la Ecuación 3.10, se llega a la Ecuación 3.13.

$$w_{jl}^i = w_{jl}^i + \alpha \frac{e_l^j}{n_P} f'_i(net^i) x_j \quad (3.13)$$

La Ecuación 3.13 es comúnmente conocida como *regla delta*¹ puesto que es la derivada del error cuadrático medio con respecto a la conexión que se desea modificar, que es la conexión actual [62, 63, 1, 64].

Existe un problema con la regla delta: depende directamente del error $e_l^j = t_l^j - y_l^j$. Entonces para calcular la dirección de cambio δ_w , se debe contar con el target t_l^j , los cuales *solo existen para las neuronas en la capa de salida*.

Sin embargo, a una neurona en una capa oculta también se le puede calcular un error indirectamente, más bien aproximando su error *retropropagando el error de las neuronas de salida* [63]. El error en la capa de neuronas $i+1$ denotado matricialmente como $\mathbf{e}_{i+1}^j$ contiene los errores que cometen no solo las neuronas en esta capa, sino las neuronas en capas anteriores también, en particular, aquellas que se conectan directa e indirectamente con cada neurona en la capa $i+1$. Véase la Figura 3.5 para una representación gráfica de esto.

La expresión formal del proceso de retropropagación del error de una capa posterior a una anterior, se deduce de la regla delta de la Ecuación 3.11.

¹También conocida como regla Widrow-Hoff

III. Estructuras y técnicas utilizadas en este estudio

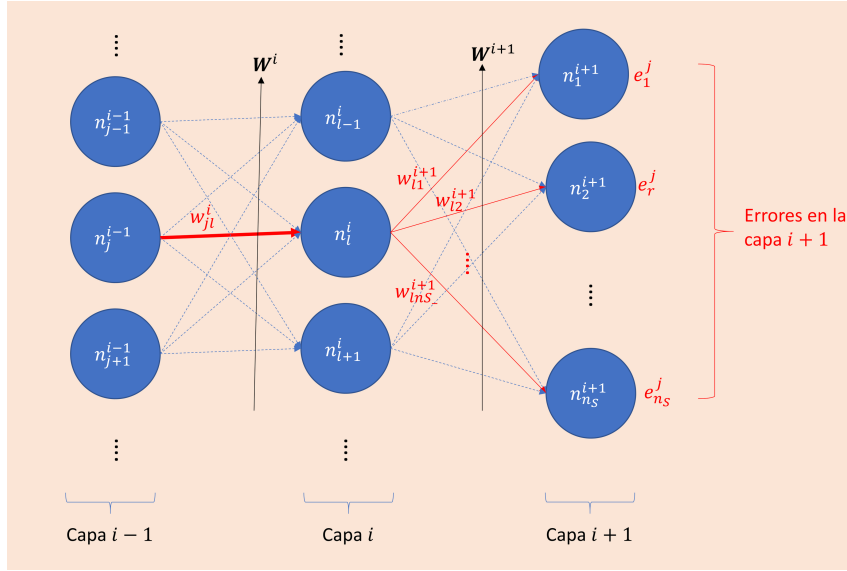


Figura 3.5: Elementos en la red involucrados en el proceso de retropropagación

$$\delta_w = -\frac{d}{dw_{jl}^i}(ecm) = -\frac{d}{dw_{jl}^i} \left(\frac{\sum_{j=1}^{n_P} \sum_{m=1}^{n_S} (t_m^j - y_m^j)^2}{2n_P} \right) \quad (3.14)$$

En la Ecuación 3.14 la doble sumatoria en derivación se refiere al valor del valor de error en la capa de salida, para el j -ésimo patrón de entrada. Ahora, en cada término de la sumatoria aparece el valor de la respuesta de cada neurona en la capa de salida, a ese mismo patrón de entrada. Aplicando la regla de la cadena, $\frac{d}{dw_{jl}^i}(ecm) = \frac{d}{dy_m^j}(ecm) \frac{d}{dw_{jl}^i}(y_m^j)$, de modo que se llega a lo expresado en la Ecuación 3.15.

$$\delta_w = -\frac{d}{dw_{jl}^i} = - \left(-\frac{\sum_{j=1}^{n_P} \sum_{m=1}^{n_S} (t_m^j - y_m^j)}{n_P} \frac{d}{dw_{jl}^i}(y_m^j) \right) \quad (3.15)$$

Ahora, como $y_m^j = f_{i-1}(net_m^{i+1})$, en donde net_m^{i+1} se refiere al valor neto de la neurona n_m^{i+1} , entonces se tiene lo expresado en la Ecuación 3.16.

$$\delta_w = -\frac{d}{dw_{jl}^i} = - \left(-\frac{\sum_{j=1}^{n_P} \sum_{m=1}^{n_S} (t_m^j - y_m^j)}{n_P} \frac{d}{dw_{jl}^i}(f_{i+1}(net_m^{i+1})) \right) \quad (3.16)$$

Aplicando nuevamente la regla de la cadena, se llega a la Ecuación 3.17.

$$\delta_w = -\frac{d}{dw_{jl}^i}(ecm) = \frac{\sum_{j=1}^{n_P} \sum_{m=1}^{n_S} (t_m^j - y_m^j)}{n_P} f'_{i+1}(net_m^{i+1}) \frac{d}{dw_{jl}^i}(net_m^{i+1}) \quad (3.17)$$

III. Estructuras y técnicas utilizadas en este estudio

Ya que $\frac{d}{dw_{jl}^i}(net_m^{i+1})$, entonces $net_m^{i+1} = \sum_{l=1}^{n_O} w_l^{i+1} o_l^i$, llegando así a la Ecuación 3.18.

$$\delta_w = -\frac{d}{dw_{jl}^i}(ecm) = \frac{\sum_{j=1}^{n_P} \sum_{m=1}^{n_S} (t_m^j - y_m^j)}{n_P} f'_{i+1}(net_m^{i+1}) \frac{d}{dw_{jl}^i} \left(\sum_{l=1}^{n_O} w_l^{i+1} o_l^i \right) \quad (3.18)$$

Como $o_l^i = f_i(net_l^i)$, es el caso de que $\frac{d}{dw_{jl}^i} (\sum_{l=1}^{n_O} w_l^{i+1} o_l^i) = \frac{d}{do_l^i} (\sum_{l=1}^{n_O} w_l^{i+1} o_l^i) \frac{d}{dw_{jl}^i} (o_l^i)$, y, de esta manera se tiene la expresión en la Ecuación 3.19.

$$\delta_w = -\frac{d}{dw_{jl}^i}(ecm) = \frac{\sum_{j=1}^{n_P} \sum_{m=1}^{n_S} (t_m^j - y_m^j)}{n_P} f'_{i+1}(net_m^{i+1}) \sum_{l=1}^{n_O} w_l^{i+1} \frac{d}{dw_{jl}^i} (o_l^i) \quad (3.19)$$

Ya que $\frac{d}{dw_{jl}^i} (o_l^i) = \frac{d}{dnet_l^i} (o_l^i) \frac{d}{dw_{jl}^i} (net_l^i)$, entonces se tiene lo expresado en la Ecuación 3.20.

$$\delta_w = -\frac{d}{dw_{jl}^i}(ecm) = \frac{\sum_{j=1}^{n_P} \sum_{m=1}^{n_S} (t_m^j - y_m^j)}{n_P} f'_{i+1}(net_m^{i+1}) \sum_{l=1}^{n_O} w_l^{i+1} f'_i(net_l^i) \frac{d}{dw_{jl}^i} (net_l^i) \quad (3.20)$$

Aquí el valor neto de la neurona net_l^i es dependiente de la conexión w_{jl}^i sobre la que la derivada del error cuadrático medio aplica, ya que $net_l^i = \sum_{j=1}^{n_{O_{i-1}}} w_{jl}^i x_j$, lo que permite, directamente calcular el factor $\frac{d}{dw_{jl}^i} (net_l^i)$, tal como lo indica la Ecuación 3.21

$$\delta_w = -\frac{d}{dw_{jl}^i}(ecm) = \frac{\sum_{j=1}^{n_P} \sum_{m=1}^{n_S} (t_m^j - y_m^j)}{n_P} f'_{i+1}(net_m^{i+1}) \sum_{l=1}^{n_O} w_l^{i+1} f'_i(net_l^i) \sum_{j=1}^{n_{O_{i-1}}} x_j \quad (3.21)$$

El factor x_j , para $j = 1, 2, \dots, n_{O_{i-1}}$, se refiere a los elementos del patrón de entrada a la capa i de neuronas, mientras que $n_{O_{i-1}}$ al total de neuronas en la capa de neuronas $i - 1$. La Ecuación 3.21 normalmente se denomina como la *regla delta generalizada* que permite aproximar el error de las neuronas en una capa cualquiera, a partir del error de las neuronas en la siguiente capa.

La regla delta generalizada permitió aplicar el aprendizaje basado en desempeño en redes neuronales complejas con al menos una capa oculta. El desarrollo de la regla delta generalizada, como fue expuesta en esta sección, se acredita a David Rumelhart y James McClelland, presentado por primera vez en 1986 [65]

3.3. Diseño de la arquitectura

Las siguientes dos subsecciones analizan con cuidado lo que se debe conocer para responder con calidad a los dos aspectos de diseño.

3.3.1. La cantidad de capas ocultas

Toda red MLP debe contener al menos una sola capa oculta. Existen redes basadas en una red MLP que se limitan a usar una sola capa oculta, por ejemplo las *máquinas de vectores de soporte* o máquinas *SVM* [59] por sus siglas en inglés, o las *máquinas de aprendizaje extremo* [59, 60], o máquinas *ELM* por sus siglas en inglés. Existen también redes basadas en redes MLP que usan múltiples capas ocultas, al menos dos de ellas. Este es el caso de las *redes de aprendizaje profundo* [56]. La cantidad de capas ocultas a utilizar puede ser establecida a través de un *proceso de prueba y error*, esto es, probar varias arquitecturas y seleccionar la que mejores resultados arrojó.

Sin embargo considerando el *principio de aproximación universal* [66, 67], se encontró que una red MLP con una sola capa oculta con la suficiente cantidad de neuronas, cada una con una función de activación sigmoideal es suficiente para la mayoría de los casos en donde una red MLP puede aplicar. La Figura 3.6 representa la red MLP usando una sola capa de neuronas ocultas, o red *SLFN* por las siglas en inglés de *Single Layer Feedforward Neural network*.

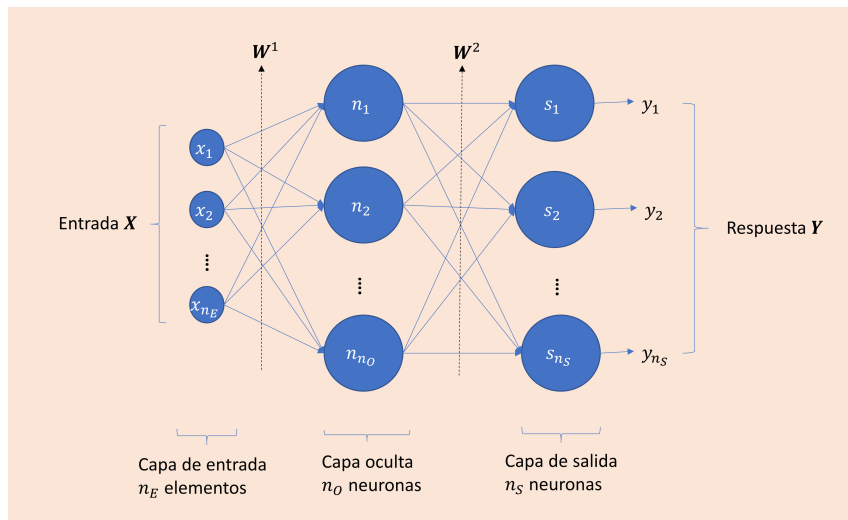


Figura 3.6: Una red MLP con una sola capa de neuronas ocultas

3.3.2. El total de neuronas en la capa oculta

A diferencia de lo que ocurre con la cantidad de capas ocultas *no existe ningún teorema o principio formal que defina de forma general la cantidad exacta de neuronas en la capa oculta*. Una red SLFN que usa entrenamiento supervisado basado en desempeño como el mostrado en la sección anterior, puede tener dos situaciones en cuanto a la cantidad de neuronas ocultas que se deben evitar:

- (1) Menos de las necesarias. En tal caso, la red SLFN no aprende nunca la relación entre los datos de entrada y salida, a partir del conjunto de entrenamiento. Se dice que existe un *subajuste a los datos de entrenamiento*.
- (2) Más de las necesarias. En este caso, la red SLFN aprende demasiado bien los datos de entrenamiento del problema para el que fue entrenado, pero es incapaz de funcionar correctamente con otros datos fuera de los datos de entrenamiento. En tal situación se habla de una *sobreajuste a los datos de entrenamiento*.

Una red SLFN con subajuste o con sobreajuste es igual de inútil en cualquier caso. Por tanto la cantidad de neuronas ocultas n_O es un parámetro de vital importancia en una red SLFN.

3.4. Redes MLP clasificadoras

Una red MLP con una sola capa oculta es capaz de funcionar en distintos ámbitos, la *clasificación de patrones* es uno de ellos. Un clasificador de patrones comúnmente conocido como *máquina clasificadora*. Además de las redes MLP existen formas alternas para implementar una máquina clasificadora, por ejemplo *máquinas de vectores soporte*, o máquinas *SVM*, *máquinas de aprendizaje extremo*, o máquinas *ELM*, existen los *clasificadores bayesianos*, o los *clasificadores kNN*, los *clasificadores basados en árboles de decisión* solo por mencionar algunos.

El siguiente listado expone algunos aspectos involucrados en el uso de redes SLFN y otras técnicas comunes para implementar clasificadores:

- (1) Las redes MLP *no son paramétricas* [68], luego, no asumen que los datos de entrenamiento obedezcan distribuciones específicas.
- (2) Las redes MLP *son capaces de aprender con conjuntos de entrenamiento muy grandes* [68].

III. Estructuras y técnicas utilizadas en este estudio

- (3) Las redes MLP *Son capaces de aprender relaciones patrón y target muy complicadas*, por ejemplo, no lineales [68].
- (4) Las redes MLP *pueden aprender incluso cuando la superficie de decisión, es decir, la frontera entre clases, en un problema de clasificación, es compleja*, por ejemplo, no lineal [69].
- (5) Las redes MLP *pueden generalizar de manera excelente*, es decir, rededir la respuesta a un patrón de entrada nunca antes visto. Lo anterior, dado un correcto entrenamiento [70].
- (6) Las redes MLP *no imponen, por si mismas, restricciones en la naturaleza de la información* de entrada [70].
- (7) A pesar de que el entrenamiento por retropropagación de errores, usando la regla delta generalizada antes vista, tiene ciertas desventajas, *el entrenamiento de una red MLP suele ser bastante más rápido* que, por ejemplo, el de una SVM [71].
- (8) Las redes MLP *pueden manejar de facto, la clasificación multiclase*, a diferencia de una red SVM, que es un clasificador binario por naturaleza [71].
- (9) Las redes MLP *pueden manejar de facto, información de muy alta dimensionalidad*, es decir, patrones de entrada con bastantes atributos o características, a diferencia de casi cualquier otro tipo de clasificador [71].
- (10) Las redes MLP, desde un principio, *contienen un conjunto fijo de parámetros, es decir, de conexiones*. Otras máquinas clasificadoras solo podrán conocer la cantidad de parámetros que contendrán, luego de haber sido entrenadas [72].
- (11) Las redes MLP *son una excelente opción en ambientes que requieren aprendizaje en tiempo real*, pues su forma de entrenamiento facilita el aprendizaje en línea, de forma directa [73].
- (12) Las redes MLP *comparten las mismas características deseables que los métodos estadísticos, más específicamente, comparten la misma matriz de covarianza gaussiana cruzada*, como función de densidad [74].
- (13) Las redes MLP *usando una función sigmoideal en las neuronas de la capa de salida, producen una respuesta que puede ser utilizadas para calcular la probabilidad posterior* de una muestra, en un sistema de predicción bayesiana [74].

Las siguientes subsecciones presentan los pormenores para lograr que la red SLFN cumpla con los requisitos para ser un buen clasificador.

3.4.1. La capa de entrada y la capa de salida

La red SLFN clasificadora contiene exactamente $n_E + 1$ elementos en la capa de entrada: n_E elementos x_i para $i = 1, 2, \dots, n_E$ que corresponden con cada elemento en el patrón de entrada, más un elemento extra fijo cuyo valor es la unidad y que funciona como un *bías* $b \in \mathbb{R}$, pues desliza al hiperplano asociado la neurona a la que se conecta fuera del origen. La capa de salida por su parte, es una capa de neuronas cuyas respuestas forman el patrón de respuesta de la red SLFN, y tiene tantas neuronas n_S como elementos aparecen en los targets o respuestas deseadas de la red. Esto implica que si la red SLFN ha de funcionar como clasificador multicategoría o multiclase, entonces $n_S = n_C$, en donde n_C es el total de clases a separar en el problema de clasificación.

Obsérvese que los targets o respuestas deseadas de la red, son patrones binarios, es decir, patrones cuyos elementos son ceros o unos. Entonces, si ocurre que $\mathbf{X} \in C_i$, únicamente el elemento $y_i \in \mathbf{Y}$ debe tener un valor unitario, el resto debe tener un valor, nulo, o cero, como en las expresiones en la Ecuación 3.22 y en la Ecuación 3.23.

$$y_i = 1 \leftrightarrow \mathbf{X} \in C_i \quad (3.22)$$

O lo que es lo mismo:

$$y_j = 0 \leftrightarrow \mathbf{X} \notin C_i \quad (3.23)$$

La respuesta del clasificador MLP es entonces un patrón binario en donde todos los elementos tienen un valor nulo, excepto el i -ésimo elemento que tiene un valor unitario, siempre que el patrón de entrada pertenezca a la i -ésima clase. A este tipo de codificación para los targets en la capa de salida se le conoce como *zero-hot* [75]. La Figura 3.7 muestra la composición gráfica de una red SLFN clasificadora.

La única cantidad cuyo valor no puede ser establecido de manera general y directa sin llevar a cabo un proceso más profundo de análisis, es la cantidad n_O . Este parámetro de la red SLFN es la razón de ser de la presente investigación. La siguiente subsección presenta la idea general utilizada aquí para determinar este valor para un problema de clasificación.

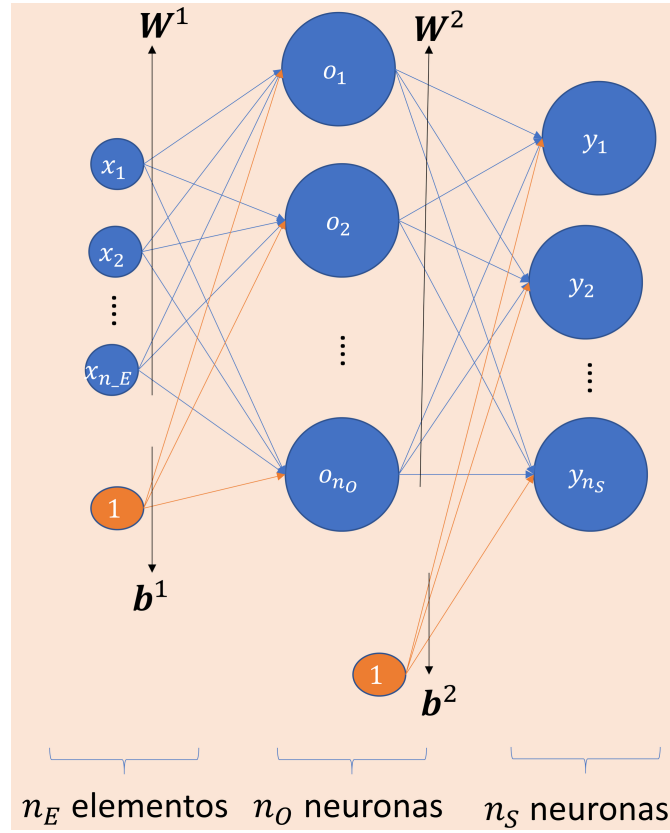


Figura 3.7: La red SLFN clasificadora. La capa de entrada tiene $n_E + 1$ elementos incluido un bias. La capa oculta tiene $n_O + 1$ neuronas incluido un bias. La capa de salida contiene n_S neuronas

3.4.2. La capa oculta y la regla de los hiperplanos

Como previamente se ha establecido se sabe por el principio de aproximación universal [66], que una red SLFN con una cantidad suficiente de neuronas en su única capa oculta puede aproximar cualquier función de manera satisfactoria. La clasificación de patrones es un caso especial de función a aproximar. Como se mencionó en el capítulo anterior, el valor n_O no puede ser establecido sin llevar a cabo un procedimiento que analice las condiciones particulares del problema a resolver, por ejemplo, usando *reglas heurísticas*, o implementando un proceso de *prueba y error* para encontrar mediante un proceso de búsqueda el mejor valor para n_O .

Existe una regla heurística que es muy útil en lo que a establecer un presunto tamaño para la capa oculta de una red SLFN clasificadora se refiere. Esta regla [1] permite establecer que el tamaño de la capa oculta, el valor n_O debe ser equivalente al número de hiperplanos que separan las clases n_H , en cuestión. Esta es una forma de regla heurística que es fácilmente aplicable a un problema de clasificación, siempre que se conozca de antemano el valor n_H , la cantidad de hiperplanos en $\mathbb{R}^{n_E}$ que separan las clases involucradas.

Esta regla heurística a la que habrá de referirse de aquí en adelante como la *regla de los hiperplanos*, es simple: supóngase un problema de clasificación binaria con $n_C = 2$. En este caso si las clases C_1 y C_2 se separan con n_H hiperplanos, entonces $n_O = n_H$. La Figura 3.8 muestra la idea principal de la regla de los hiperplanos. La determinación de la cantidad de hiperplanos que separan las clases para un caso general en cuanto a cantidad de clases y atributos de entrada, no es un problema trivial. Una metodología general para obtener la cantidad de hiperplanos es el objeto de esta investigación y será tratada en el siguiente capítulo.

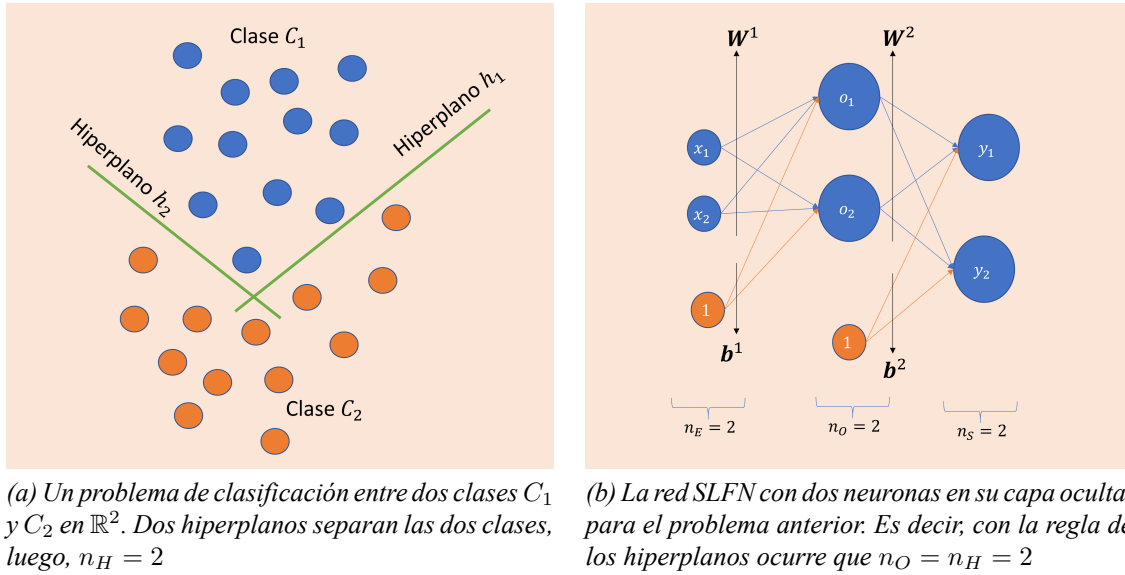


Figura 3.8: La regla de los hiperplanos en un problema de clasificación binaria, en $\mathbb{R}^2$

3.4.3. MLP clasificador multiclase

Un problema de clasificación entre n_C clases puede ser en *forma binaria*, con $n_C = 2$, separando únicamente a dos clases, o puede ser en *forma multiclase o multietiqueta, o multicategoría* en donde necesariamente $n_C > 2$, y separando a las clases C_i para $i = 1, 2, \dots, n_C$. El caso binario es el caso mas estudiado [76]. El caso multiclase es el menos estudiado en la literatura [77], a pesar de ser el caso más general en cuanto a clasificación de patrones.

Existen algunos esquemas para tratar el caso multiclase [77, 78]:

- (1) El *esquema One Against One (OaO)*: consiste en construir una red SLFN clasificadora que aprenda a distinguir cada par de clases diferentes, es decir, separar las muestras de cada par de clases distintas. Debido a esto, serán necesarios $n_C(n_C - 1)/2$ clasificadores diferentes.

III. Estructuras y técnicas utilizadas en este estudio

Cada uno de estos clasificadores se diseña y entrena por separado. De esta manera, un nuevo patrón o muestra $\mathbf{X} \in \mathbb{R}^{n_E}$ será asignado a la *clase más votada* de entre todos los clasificadores que lo reciben como entrada.

- (2) El *esquema One Against All (OaA)*: consiste en construir un clasificador que separa a las muestras de cada clase, de las muestras del resto de las clases. Por esto, en este esquema se requiere diseñar y entrenar n_C clasificadores diferentes, que separen las muestras de cada clase, del resto de las clases.

La Figura 3.9 muestra un problema de clasificación entre tres categorías, en color azul la clase C_1 , en anaranjado la clase C_2 y en verde la clase C_3 .

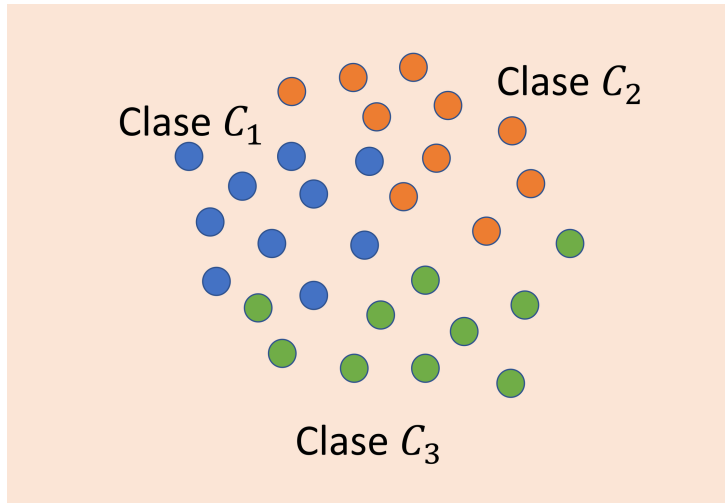


Figura 3.9: Problema de clasificación con tres clases

Para este problema de clasificación la Figura 3.10 muestra la implementación de los esquemas OaO y OaA de manera gráfica. Nótese que para el caso $n_C = 3$, el esquema OaO requiere $n_C(n_C - 1)/2 = 3(2)/2 = 3$ clasificadores, en tanto que el esquema OaA también requiere $n_C = 3$ clasificadores. El que en este caso el total de clasificadores en ambos esquemas fuera el mismo es mera coincidencia.

En general el esquema OaO requiere más clasificadores a diseñar y entrenar que los que requiere el esquema OaA. El esquema OaA implica reagrupar las muestras del resto de las clases en una sola clase antes de diseñar y entrenar al clasificador que separa a la primera del resto de las clases.

También debe tenerse en cuenta que la *etapa de votación* que sucede a la decisión de los clasificadores separados en ambos esquemas, es un proceso separado que ocurre para decidir

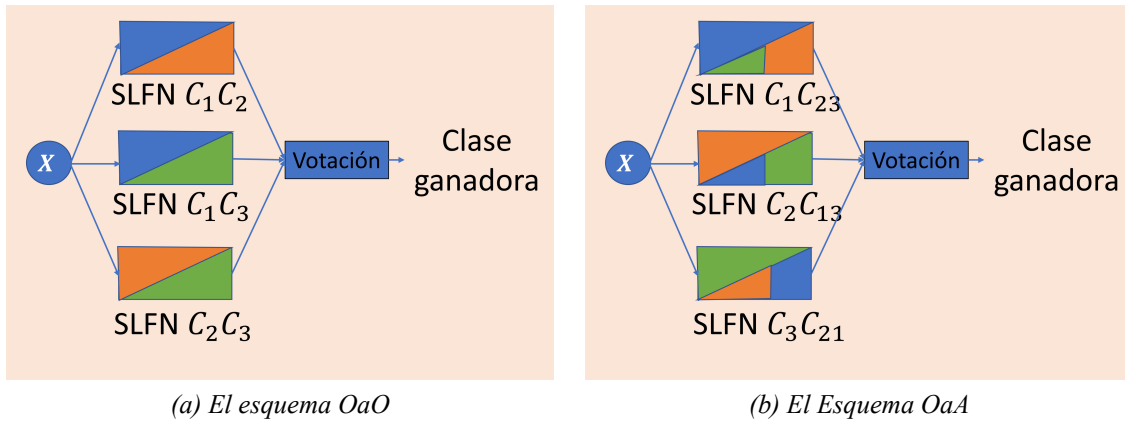


Figura 3.10: Los esquemas de clasificación multiclase

a que clase se asigna ese nuevo patrón a clasificar. La votación debe conciliar las decisiones separadas en una sola decisión final. Para tomar la decisión acerca de a cual de las clases habrá de ser asignado un patrón, tanto en OaO como en OaA existe una segunda etapa de votación que asignaría a la clase *más votada*, esto es, la clase cuyo clasificador binario tuvo la respuesta mayor.

3.4.4. Clases empalmadas

Normalmente cuando se habla de una red SLFN clasificadora se asume que las clases en juego se encuentran *separadas o no traslapadas*. Esto significa que *las muestras de dos clases diferentes son claramente distinguibles* y no existe manera de confundir las muestras de una clase con las de otra. En otras palabras, se puede afirmar que *en un problema con clases separadas, siempre es posible localizar una superficie de decisión entre las clases*, una frontera entre clases tal que todo a un lado de tal frontera pertenece a una clase y del otro lado no pertenecen a tal clase.

Por el contrario, cuando las clases en juego son tales que *al menos en alguna parte del espacio del problema, las muestras de dos o mas clases diferentes se confunden*, esto es, no se puede distinguir cuando son de una u otra clase, se dice que *se tiene un problema de clases empalmadas o traslapadas*. Véase la Figura 3.11 para una representación gráfica del hecho de que dos clases se encuentren traslapadas.

En la Figure 3.11a se tiene el caso de las clases no empalmadas o *clases separables*, como es más común llamar a este caso. Las clases separables implican que una frontera o superficie de decisión podría ser trazada sobre el espacio del problema. En la Figura 3.11b se tiene el caso

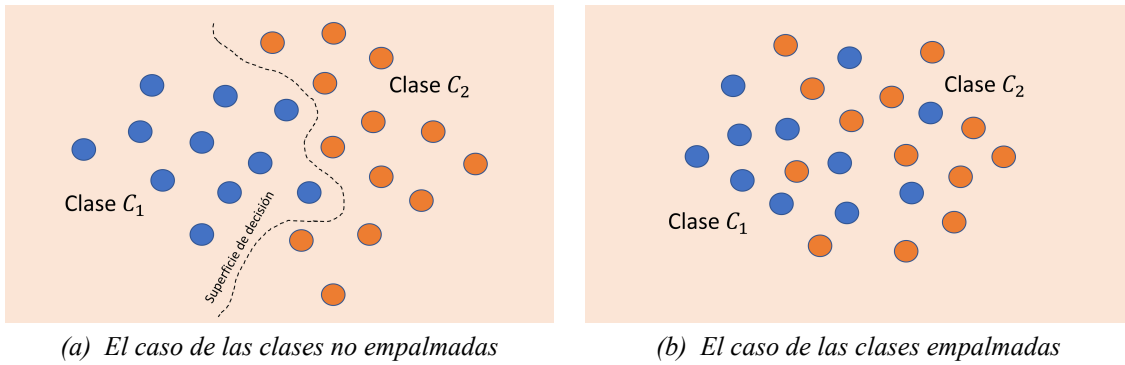


Figura 3.11: Clases empalmadas y no empalmadas, en $\mathbb{R}^2$

de las clases empalmadas. Aquí no es posible trazar una superficie de decisión, pues no existe forma de separar las muestras entre las clases, de manera que unas queden de un lado u otro de tal superficie de decisión. Existen distintos motivos para esto, por ejemplo:

- (1) Los atributos usados para describir las muestras no son los adecuados [79, 80].
- (2) Inadecuada preparación de los sensores durante la captura de las mediciones en las muestras [79].
- (3) Pobre muestreo durante la obtención de las muestras en las clases [81].

El problema con las clases empalmadas es que algunas formas de clasificadores exigen que las clases se encuentren separadas. Los clasificadores basados en perceptrones, las máquinas SVM y las redes MLP en general exigen que las clases sean separables a fin de entrenar exitosamente a la red, ya que su éxito se basa en localizar una superficie de decisión entre las clases.

3.5. Geometría computacional

Por definición la geometría computacional es el estudio del planteamiento de algoritmos computacionales en términos meramente geométricos [82]. La geometría computacional suele derivarse en dos formas o campos de estudio [83]:

$$\text{Geometría computacional} \begin{cases} \text{Geometría computacional combinatoria} \\ \text{Geometría computacional numérica} \end{cases}$$

La *geometría computacional combinatoria* se encarga de estudiar a un objeto geométrico como una entidad discreta. Por ejemplo, dada una nube de puntos en el espacio $\mathbb{R}^{n_E}$ ¿Cual es

el poliedro más pequeño que los contendría a todos ellos? En este caso el objeto geométrico es el poliedro y la entidad discreta que lo contiene podría ser una *envolvente convexa*. Por su parte la *geometría computacional numérica* busca representar a los objetos geométricos de modo que faciliten las operaciones numéricas requeridas por los cálculos geométricos, por ejemplo, describir una superficie en el espacio $\mathbb{R}^{n_E}$ como un *spline* o una *curva de Bezier*.

En esta investigación el interés está centrado en la geometría computacional combinatoria, pues ofrece medios para al menos aproximar en términos geométricos computacionales, conceptos relevantes al análisis de las clases involucradas en el proceso de clasificación. De particular importancia son dos conceptos geométricos: *envolventes convexas* y *formas alfa*.

3.5.1. Envolvertes convexas

Una de esas estructuras discretas más importantes en la geometría computacional, es la *envolvente convexa*, o *convex hull*, en inglés. Sea un conjunto finito de muestras, o puntos en el espacio $\mathbb{R}^{n_E}$, como lo pueden ser los patrones de entrenamiento de un dataset de un problema de clasificación. Llámese $\mathbf{Xs}$ a tal conjunto de muestras. Entonces una envolvente convexa es aquel polígono o poliedro más pequeño y *necesariamente convexo*, que contiene a todas las muestras en $\mathbf{Xs}$. Dos hechos destacan en esta definición:

- (1) El poliedro a todos los puntos del conjunto $\mathbf{Xs}$ en su interior.
- (2) Los vértices del poliedro son muestras del conjunto $\mathbf{Xs}$.

El convex hull, o envolvente convexa es una estructura discreta cuya importancia radica en que ofrece una interpretación geométrica del alcance de un grupo de datos, como lo son las muestras de una clase en un conjunto de entrenamiento o dataset. Si el grupo de datos es denotado como $\mathbf{Xs}$ su envolvente convexa se denota como $CH(\mathbf{Xs})$. La Figura 3.12 ofrece una descripción gráfica del concepto de envolvente convexa para el grupo de datos $\mathbf{Xs}$.

En la Figura 3.12a un grupo de datos o muestras de una clase $\mathbf{Xs} \in \mathbb{R}^2$ son ilustrados. En la Figura 3.12b la envolvente convexa para $\mathbf{Xs}$ es ilustrada como $CH(\mathbf{Xs})$. La envolvente convexa es el polígono delimitado por las líneas, y sus vértices son muestras del mismo grupo $\mathbf{Xs}$. $\mathbf{Xs}$ es un *conjunto convexo* si esto implica que para cualquier par de puntos $x \in Xs$ y $z \in Xs$ el segmento $xz \in Xs$.

La definición gráfica del concepto de envolvente convexa puede ser definida en términos más formales. Por ejemplo *la envolvente convexa, para un grupo de puntos o muestras $\mathbf{Xs} \in \mathbb{R}^{n_E}$ es la intersección de todos los medios espacios que contienen a todos esos puntos en $\mathbf{Xs}$* . Por

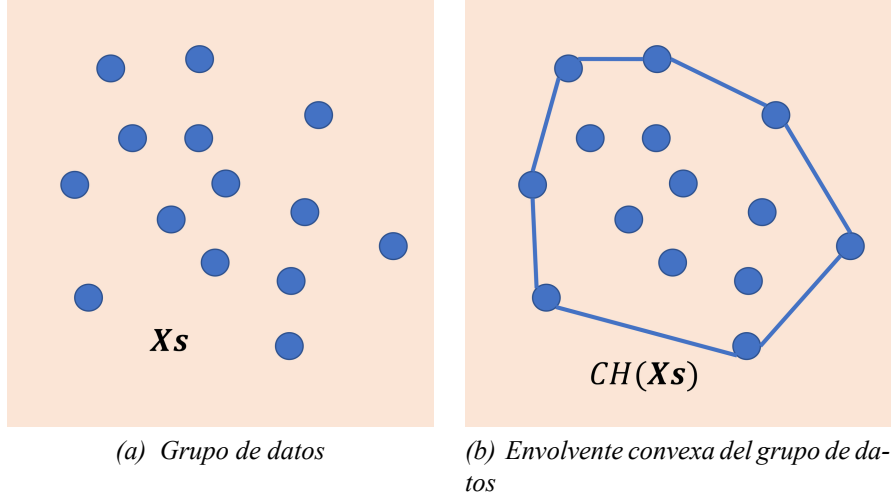


Figura 3.12: El concepto de envoltente convexa, para un grupo de datos en $\mathbb{R}^2$

medio espacio entiéndase todos los puntos o muestras a uno y otro lado del hiperplano que parte en dos al espacio de los puntos en $\mathbf{Xs}$. Nótese que la envoltente convexa es una superficie sólida y cerrada que contiene a todos los puntos del conjunto $\mathbf{Xs}$.

La envoltente convexa del conjunto $\mathbf{Xs}$, denotada como $CH(\mathbf{Xs})$ puede ser calculada de muy diferentes formas. Una de ellas, simple de plantear, es el llamado *algoritmo de Graham*, que funciona como lo indica el Algoritmo 4.

Este algoritmo tiene complejidad $O(n_P \log(n_P))$. Quizá su único inconveniente es el hecho de que implica ordenar los puntos en una colección. Al final lo importante de la envoltente convexa es que es una estructura que permite hacerse de una idea de la forma exclusivamente convexa, que tiene una agrupación de puntos o muestras.

3.5.2. Triangulaciones de Delaunay

Una *triangulación* de un polígono se refiere a la descomposición de la forma poligonal o poliédrica de la envoltente convexa $CH(\mathbf{Xs})$, como la unión de uno o más *simplexes*² tal que cualquier par de ellos que son colindantes, no intersectan sus interiores y la unión de todos ellos produce al poliedro $CH(\mathbf{Xs})$. Es el trazo de diagonales entre cualquier par de vértices o esquinas del polígono no adyacentes entre sí, de tal forma que estas diagonales no se intersecten [84]. Una triangulación puede ser llevada acabo de distintas maneras, pero no todas las triangulaciones son igual de útiles.

²Un simplex es el mismo concepto que el de un triángulo en $\mathbb{R}^2$, una pirámide en $\mathbb{R}^3$, pero en $\mathbb{R}^{n_E}$, para $n_E \geq 2$

Algoritmo 4: Algoritmo de Graham

Data: el conjunto $\mathbf{Xs}$
Result: la estructura $CH(\mathbf{Xs})$, la envolvente convexa de $\mathbf{Xs}$

- 1 Encontrar la muestra en $\mathbf{Xs}$, con la última coordenada x_{n_E} más baja. Llámese a esta muestra la muestra $\mathbf{x}_0$;
- 2 Ordenar el resto de las muestras de acuerdo con su ángulo polar, en dirección contraria a las manecillas del reloj, con respecto a $\mathbf{x}_0$, Llámese al más cercano la muestra $\mathbf{x}_1$, al segundo más cercano la muestra $\mathbf{x}_2$, y así sucesivamente;
- 3 Hacer un pila S vacía, e insertar los puntos $\mathbf{x}_0$, $\mathbf{x}_1$ y $\mathbf{x}_2$;
- 4 **for** $i = 3$ **to** n_P **do**
- 5 Hacer $listo = False$;
- 6 **while** $listo == False$ **do**
- 7 Calcular $angu = angulo(S(top), S(top - 1), \mathbf{x}_i)$;
- 8 **if** $angu \leq 0$ **then**
- 9 Eliminar $S(top)$;
- 10 Agregar $\mathbf{x}_i$ a la colección *puntos*;
- 11 **else**
- 12 Hacer $listo = True$;
- 13 **end**
- 14 **end**
- 15 Agregar *puntos* a la pila S ;
- 16 **end**

En este sentido, una de las formas de triangulación más útiles es la llamada *triangulación de Delaunay*. Una triangulación de Delaunay $TD(\mathbf{Xs})$ sobre un conjunto de datos $\mathbf{Xs} \in \mathbb{R}^{n_E}$ es de manera tal que los triángulos resultantes nunca son largos y estrechos. Esta característica es muy útil, pues asegura una mejor representación de la topología de la nube o grupo de puntos o muestras [82]. Esto se logra maximizando el mínimo ángulo de entre todos los triángulos que componen la triangulación.

Para llevar a cabo una triangulación de Delaunay se utilizan *n-esferas*³ *circunscritas*. Una *n-esfera* en el espacio $\mathbb{R}^{n_E}$, es posicionada sobre el simplex de manera que los vértices del simplex nunca residen adentro de la *n-esfera*, sino únicamente en los bordes, véase la Figura 3.13. La *n-esfera* es simplemente un círculo y los simplexes son simplemente triángulos.

En las triangulaciones de Delaunay ningún punto $\mathbf{x}_i \in \mathbf{Xs}$ permanece adentro de la *n-esfera* de alguno de los simplexes que componen la triangulación. A esto se le conoce como el *requerimiento o condición de Delaunay*. Las triangulaciones de Delaunay usan a todos los puntos o muestras de la nube o grupo de puntos, no únicamente a las muestras en los vértices o esquinas

³Una *n-esfera* es el equivalente al círculo en $\mathbb{R}^2$, o a la esfera en $\mathbb{R}^3$, pero en el espacio $\mathbb{R}^{n_E}$ para $n_E \geq 2$

III. Estructuras y técnicas utilizadas en este estudio

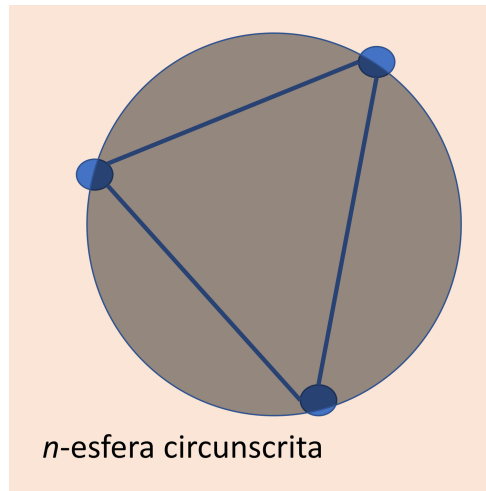


Figura 3.13: El concepto de n -esfera circunscrita, en $\mathbb{R}^2$

del poliedro de la envolvente convexa. Los puntos o muestras deben quedar circunscritos a los bordes u orillas de las n -esferas, y deben permanecer lejos de los interiores de dichas n -esferas. La Figura 3.14 muestra una triangulación de Delaunay sobre un conjunto de datos muy simple, junto con las n -esferas circunscritas en los simplexes de la triangulación.

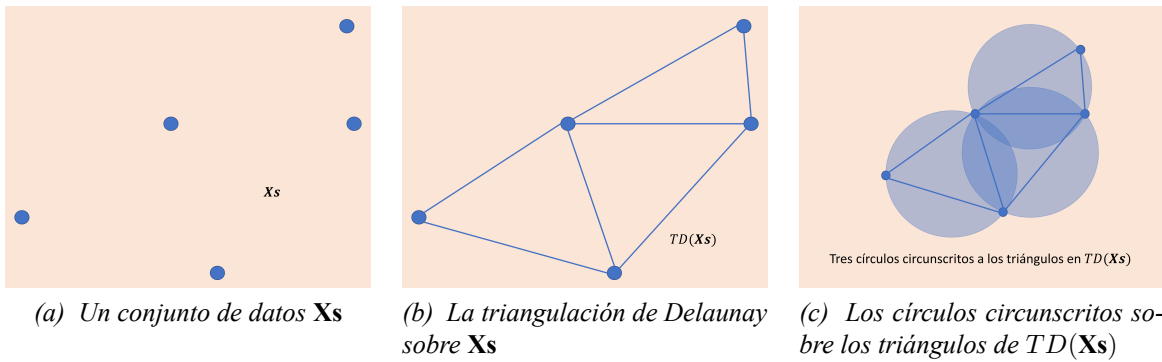


Figura 3.14: Los círculos circunscritos sobre una triangulación de Delaunay en $\mathbb{R}^2$

Dos simplexes vecinos, esto es, que comparten una arista en una triangulación como lo muestra la Figura 3.15 en un caso bidimensional, no siempre cumplen con la condición de Delaunay, pero algo se puede hacer.

Para un par de simplexes que comparten una arista en común, la llamada condición de Delaunay se puede plantear fácilmente a partir de los ángulos internos α y γ , según lo mostrado en la Figura 3.15a. Sucede que los triángulos ABC y ADC cumplen con la condición de Delaunay si $\alpha + \gamma \leq 180^\circ$. Es claro que los triángulos ABC y ADC en la Figura 3.15a no cumplen tal condición de Delaunay, y no es una triangulación de Delaunay válida. Sin embargo existe una

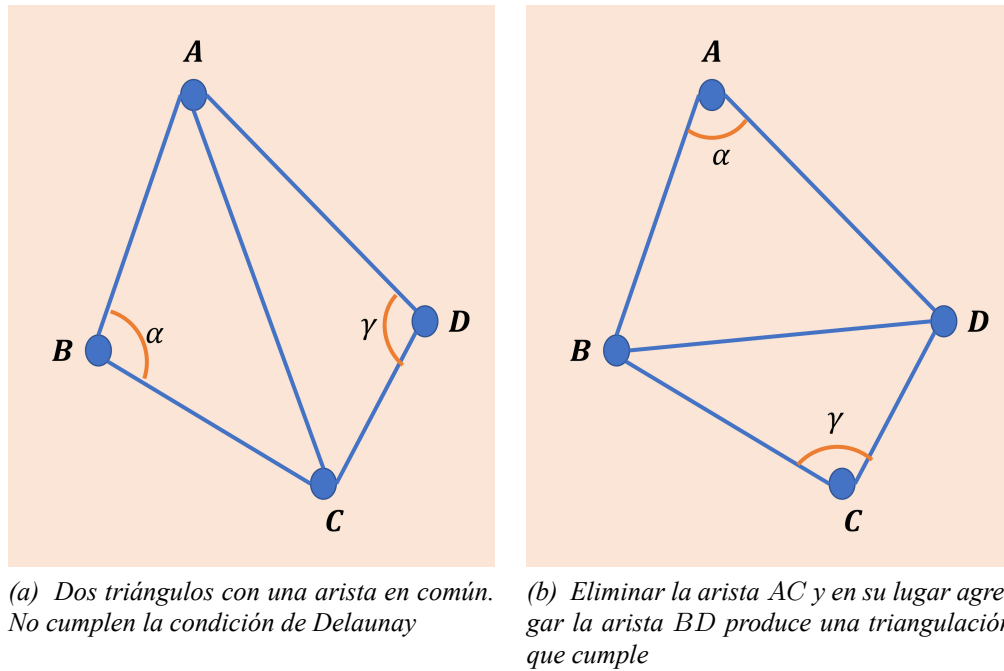
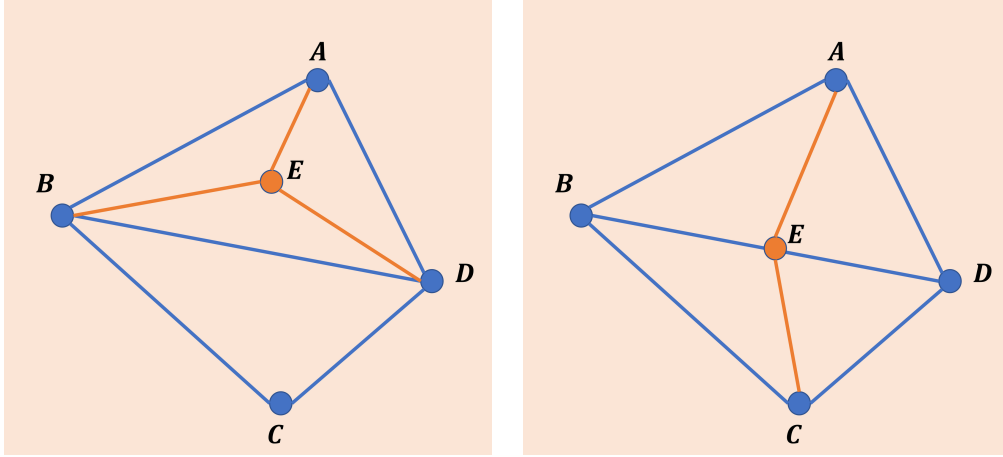


Figura 3.15: El flipping de una arista para producir una triangulación de Delaunay válida, en un ejemplo en $\mathbb{R}^2$

técnica simple llamada *flipping* que permite que dos simplexes que no cumplen con la condición, lo hagan, a partir de un simple intercambio de aristas: simplemente se elimina el arista común en cuestión AC , en este caso, y se intercambié por un nuevo arista, una que une a los vértices en donde se localizan los ángulos α y γ , por ejemplo los vértices B y D en la Figura 3.15a. De manera que ahora existirá un nuevo arista BD como lo muestra la Figura 3.15b. Ahora α es el ángulo formado por las aristas AB y AD en el nuevo triángulo ABD mientras que γ es el ángulo formado por las aristas CB y CD en el nuevo triángulo CBD . Estos serían los ángulos al frente del nuevo arista en común BD . Se puede verificar que la nueva triangulación si cumple con la condición de Delaunay, pues $\alpha + \gamma \leq 180^\circ$.

Existen distintos métodos para calcular una triangulación de Delaunay. Un método simple es el *método incremental*, que consiste en agregar puntos adentro de un simplex ya existente del conjunto original $\mathbf{X}$ s a partir de una primera triangulación original, y luego alterar la triangulación agregando nuevos simplexes, cuidando de no alterar las condiciones de Delaunay. Pueden presentarse dos casos:

- (1) Que no exista ningún problema: en este caso el simplex que contiene al punto se divide en tres o cuatro nuevos simplexes: una nueva arista es trazada, desde los vértices originales hacia el nuevo punto. Deberá verificarse si alguno de las aristas comunes entre los



(a) El caso sin problemas, tan solo se divide el simplex contenedor, en tres nuevos simplexes, o triángulos

(b) El caso degenerado, los dos simplexes o triángulos con arista en común, se divide, cada uno, en dos simplexes o triángulos nuevos

Figura 3.16: La inserción de un nuevo punto en una triangulación previa, en un caso en $\mathbb{R}^2$

simplexes resultantes debe ser sometido al flipping.

- (2) Que el nuevo punto se encuentre justo en alguna de las aristas del simplex que lo contiene: en este caso, un nuevo arista es trazado desde el vértice que contiene al ángulo al frente del arista en cuestión, dividiendo así el simplex original, en un dos nuevos simplexes. Téngase en cuenta que pueda ser el caso de que el nuevo punto quede en una arista que es común a dos simplexes previos. En este caso, el procedimiento de dividir al simplex en dos se aplica igual en ambos simplexes.

La Figura 3.16 proporciona una idea gráfica de este proceso. En la Figura 3.16a se tiene el caso sin problemas en donde el nuevo punto E queda adentro del triángulo ABD , por lo que este triángulo queda particionado ahora en tres nuevos triángulos ABE , ADE y BDE y ninguno de ellos requiere flipping especial. En la Figura 3.16b aparece el caso degenerado, aquí el nuevo punto E queda en el arista BD que es común a los triángulos ABD y BDC por lo que ahora cada uno de ellos será subdividido en dos triángulos, esto es, el triángulo ABD se divide en los triángulos AEB y AED ambos con el arista AE en común, mientras que el triángulo BDC se divide en los triángulos BEC y DEC ambos compartiendo el arista EC . No es necesario efectuar flipping sobre estas nuevas triangulaciones.

El Algoritmo 5 es el llamado algoritmo incremental para la generación de la triangulación de Delaunay a partir de la colección de muestras $\mathbf{Xs}$ original. Si el conjunto original $\mathbf{Xs}$ consta de n_P muestras este algoritmo incremental tiene una complejidad $O(n_P^2)$, por lo que es una buena

opción de implementación.

Algoritmo 5: El algoritmo incremental para producir la triangulación de Delaunay

Data: La nube o grupo de muestras $\mathbf{Xs}$

Result: La triangulación TD

```

1 Ordenar las muestras en  $\mathbf{Xs}$  de manera que los mas externos queden primero;
2 Hacer  $\mathbf{Pts} = x_1, x_2, \dots, x_{n_E}, x_{n_E+1}$ ;
3 Hacer la triangulación inicial  $TD = TD(\mathbf{Pts})$ ;
4 for  $i = n_E + 1$  to  $n_P$  do
5   | Hacer  $p = x_i$ ;
6   | Agregar  $p$  en  $TD$ , según sea el caso sin problemas, o el degenerado, para producir;
7   | Verificar si la triangulación  $TD$  resultante requiere flipping;
8 end
```

Existen otros algoritmos para la generación de la triangulación de Delaunay, como *divide y vencerás* y el *sweep hull*. El incremental tiene como ventaja su rapidéz, sobre todo cuando la cantidad de muestras en el conjunto o nube de puntos es pequeña.

3.5.3. Formas alfa

La envolvente convexa, o convex hull, es una posibilidad para describir al poliedro que contiene a todos los puntos de un conjunto de datos, pero esta estructura tiene forma convexa, esto sería una limitante en un conjunto de datos, pues es posible que existan conjuntos de datos cuya forma geométrica sea cóncava.

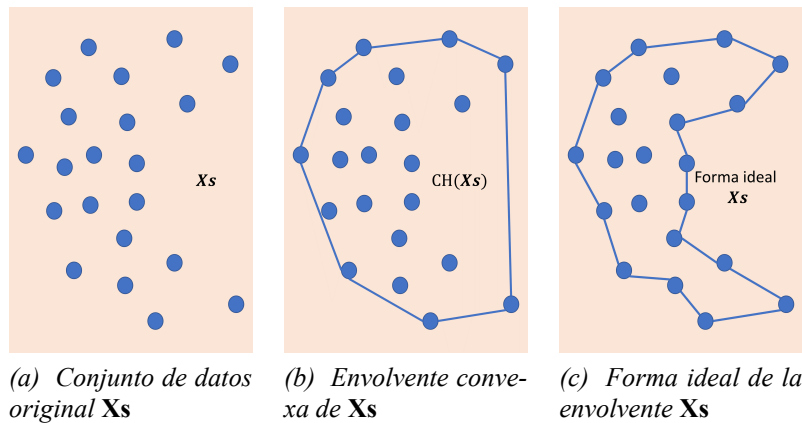


Figura 3.17: La diferencia entre forma ideal, y envolvente convexa en $\mathbb{R}^2$

En la Figura 3.17 se muestra gráficamente la diferencia entre la forma ideal de un conjunto de datos que pudiera no tener forma convexa, y la envolvente convexa de la misma. En la Figura

3.17a se aprecian los datos del conjunto original $\mathbf{Xs} \in \mathbb{R}^2$. En la Figura 3.17b aparece la envolvente convexa del mismo, en tanto que en la Figura 3.17c la forma ideal del conjunto $\mathbf{Xs}$. Es claro que en algunos casos como el mostrado en las figuras anteriores, la envolvente convexa no corresponde fielmente a la forma del conjunto de datos. La *forma alfa* del conjunto $\mathbf{Xs}$ denotada comúnmente como $\alpha(\mathbf{Xs})$ o *alpha shape*, en inglés, también denotada como $\alpha - shape$, es una colección finita de curvas aproximadas mediante segmentos lineales que en el espacio euclidiano aproximan el concepto de forma para un conjunto de puntos o muestras $\mathbf{Xs}$ en ese espacio $\mathbb{R}^{n_E}$ [85]. Así las formas alfa permiten una aproximación poliédrica no necesariamente convexa como se aprecia en la forma ideal, en la Figura 3.17c anterior.

Las formas alfa son estructuras discretas que permiten describir la forma de esa nube de puntos controlando el radio $r \in \mathbb{R}$ de una n-esfera ideal que recorre la nube de puntos $\mathbf{Xs}$ original. Ese recorrido es en el sentido de que la n-esfera rueda por la nube de puntos en lo que su volumen se lo permite. En lo que recorre la nube, aquellos puntos o muestras que entren en contacto con la n-esfera serán seleccionados para el resultado final. Si la n-esfera tuviese un radio r muy pequeño, definitivamente entraría al interior de la nube $\mathbf{Xs}$ tocando a todos los puntos, por lo que el resultado final sería el mismo grupo o nube de puntos $\mathbf{Xs}$ original. Si por otro lado, la n-esfera tuviese un radio r muy grande, definitivamente no entraría hacia el interior de la nube y únicamente los puntos o muestras más exteriores serían tocados y seleccionados para el resultado final, como el mostrado en la Figura 3.18.

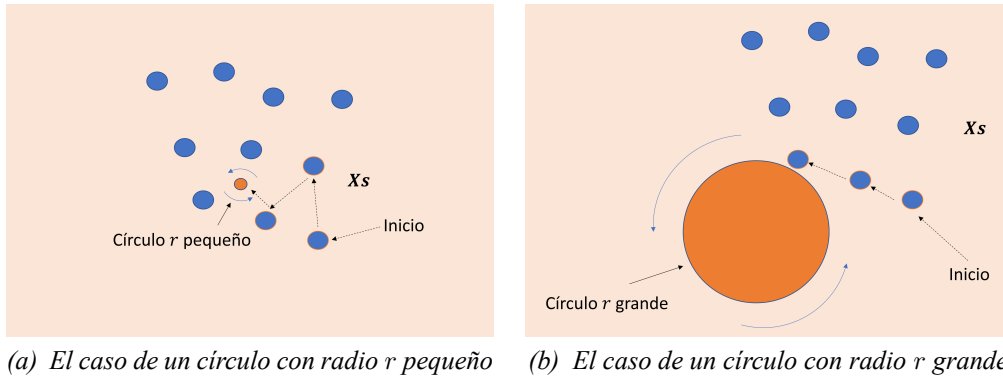


Figura 3.18: La n-esfera con distintos radios, en $\mathbb{R}^2$, mas bien un círculo

Existen muy variadas formas de calcular la forma alfa, pero una de ellas es fácil de plantear y se basa en la triangulación de Delaunay de la misma nube de puntos $\mathbf{Xs}$ a la que se le debe calcular la forma alfa. Se desarrolla un grafo y se deduce cuales son los vértices en los extremos, los contornos de toda la nube de puntos. El procedimiento se describe en el Algoritmo 6.

El Algoritmo 6 se vale únicamente de la triangulación de Delaunay de la nube de puntos

Algoritmo 6: Algoritmo para obtener la forma alfa, a partir de la triangulación de Delaunay

Data: $TD(\mathbf{Xs})$
Result: $\alpha(Xs)$

- 1 Crear un grafo G , a partir de $TD(\mathbf{Xs})$, de manera que los nodos o vértices corresponden con simplexes de la triangulación, y existe una arista entre dos nodos, si y sólo si sus correspondientes simplexes comparten una arista;
- 2 Calcular los componentes conectados del grafo G , por ejemplo, C_{TD} ;
- 3 Hacer $bordes = \{\}$;
- 4 **for** $p \in C_{TD}$ **do**
- 5 **if** $grado(p) < 3$ **then**
- 6 $bordes = bordes \cup \{p\}$;
- 7 **end**
- 8 **end**
- 9 Hacer $\alpha =$;
- 10 **for** $t \in bordes$ **do**
- 11 Extraer arista en t no compartida;
- 12 Extraer de esta arista sus vértices vs ;
- 13 Hacer $\alpha = \alpha \cup vs$;
- 14 **end**

original y no usa el parámetro α que es el radio de la n -esfera que recorre la nube de puntos. En lugar de ello el algoritmo se limita a buscar aquellos simplexes en la triangulación que se localizan en las orillas o bordes de la nube de puntos, y de ellos extrae el arista no compartido con ningún otro simplex de la triangulación, pues sus vértices son los puntos o muestras de la nube de puntos y localizados en los bordes, los que definen la forma geométrica de la forma alfa.

Lo hace a partir de un árbol cuyos nodos son aquellos simplexes en la triangulación, que tienen una arista en común con algún otro simplex, y cuando dos o más de estos existan, aparecerán en el grafo conectados por una arista. Después de esto una búsqueda de componentes conectados ocurre sobre el árbol para reconocer aquellos simplexes que se localizan en las orillas, pues corresponderán con los nodos del árbol cuyo grado es menor a tres.

Capítulo 4

Desarrollo

Aquí se presenta en detalle un método para determinar el tamaño de la capa oculta de una red SLFN para un problema de clasificación, usando para ello sólo las muestras de cada clase en $\{\mathbf{x}_j^i\}$ para $i = 1, 2, \dots, n_C$ y $j = 1, 2, \dots, n_{P_i}$ en el conjunto de entrenamiento CE . Este capítulo se divide en las siguientes secciones: una primer sección que presenta el concepto de *la forma de una clase*, tal como se utiliza en este estudio. Luego una segunda sección que analiza el concepto de *bordes fundamentales*, que localizan la superficie de decisión entre clases. Después llega una tercer sección sobre el *promedio entre clases* que aproxima la superficie de decisión, la frontera entre clases, de manera discreta. Luego una cuarta sección sobre *el ajuste con hiperplanos* de la aproximación a la frontera entre clases, que finalmente determina el valor n_H , la cantidad de hiperplanos que se ajustan a la frontera entre clases, lo que según la regla de los hiperplanos establece el valor final de n_O , el tamaño de la capa oculta de la red SLFN clasificadora.

4.1. La forma de una clase

Para establecer los hiperplanos que separan las clases será necesario contar primero con una descripción muy simplificada de la forma de las clases en el problema. La Figura 4.1 proporciona una idea gráfica del concepto de la forma de una clase.

Las muestras o patrones que siendo parte de la clase o *nube de puntos* de la clase, se localizan justo en los bordes de la misma habrán de ser llamados las *muestras borde de la clase*.

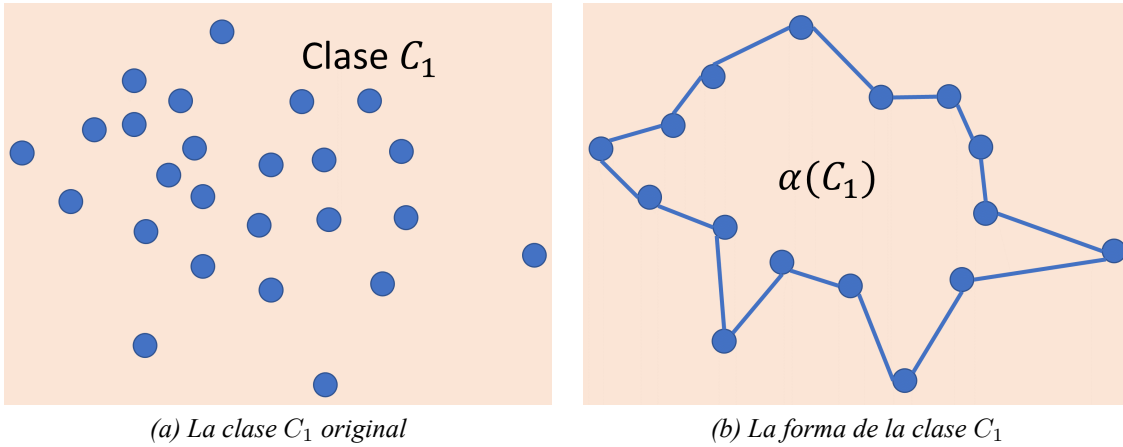


Figura 4.1: El concepto de la forma de una clase. En (a), la clase original. En (b), su forma. Nótese las muestras en los bordes de la clase

4.1.1. La forma alfa de una clase

La Figura 4.1b muestra la forma alfa de la nube de puntos mostrada en la Figura 4.1a. La forma alfa es una colección finita de segmentos lineales que juntos dan lugar a formas curvas sencillas en el espacio euclidiano [86]. Por ejemplo en la Figura 4.1b cada segmento lineal entre dos muestras borde, y todos los segmentos lineales en conjunto, dan lugar a una forma curva.

La Figura 4.2 muestra la forma alfa de una nube de puntos en $\mathbb{R}^2$ y $\mathbb{R}^3$. En la Figura 4.2a se muestra una nube de puntos en dos dimensiones, mientras que en la Figura 4.2c una nube de puntos en tres dimensiones.

La Figura 4.2b y la Figura 4.2d muestran la geometría poliédrica de una forma alfa. Este tipo de construcción resulta útil para determinar si alguna otra muestra cualquiera se localiza adentro de la nube de puntos. Del cálculo de la forma alfa se obtienen los vértices de la forma poliédrica de la forma de la nube de puntos.

4.1.2. Aproximando la forma alfa de una clase

En principio la forma alfa es una estructura discreta que es realizable para cualquier nube de puntos C_i , para $i = 1, 2, \dots, n_C$, siendo n_C el total de clases involucradas en el problema. En la práctica existe un serio problema que limita mucho la aplicabilidad de la forma alfa: *en dimensiones mayores el cálculo de la forma alfa se torna computacionalmente intratable.*

La razón de esto yace en el hecho de que casi todos los algoritmos para el cálculo de la forma alfa deben realizar un ordenamiento, y luego búsqueda sobre la triangulación de Delaunay de simplexes y aristas que cumplan con alguna condición, por ejemplo aquellos aristas vértices

IV. Desarrollo

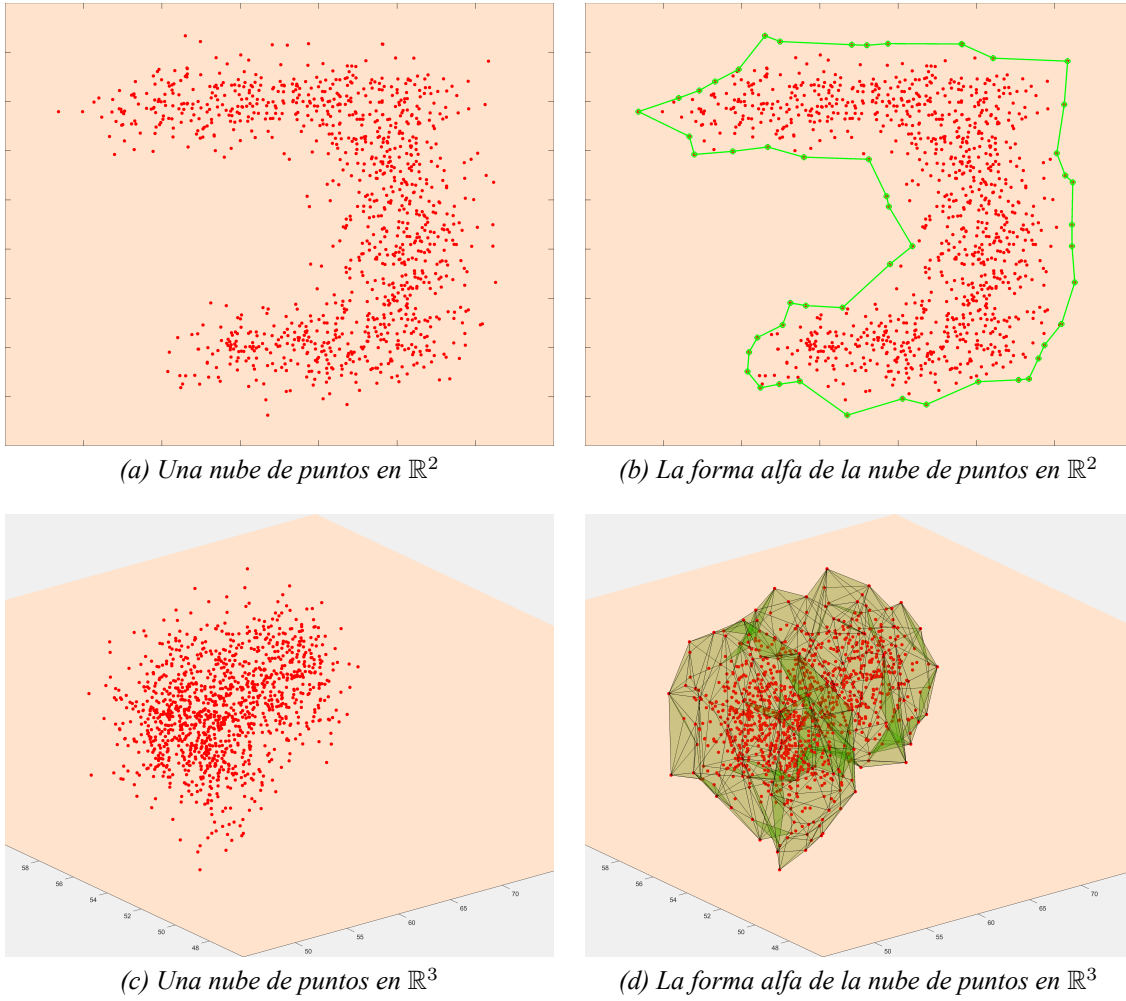


Figura 4.2: La forma alfa en dos y tres dimensiones. Nótese la forma poliédrica en color verde, de la forma alfa, delimitada por las muestras en los bordes de la nube

con algún grado de incidencias, lo que a su vez depende de la cantidad de dimensiones en el problema, de la cantidad n_E lo que hace que *el problema del cálculo de la forma alfa un problema combinatorio np* . Además si bien existen teoremas acerca de como aplicarles en dimensiones mayores a tres, no queda claro como la implementación debería ser. Debido a esto, al menos en la actualidad *solo se conocen implementaciones útiles del cálculo de la forma alfa para dos y tres dimensiones*¹, es decir para $n_E = 2$ y $n_E = 3$ únicamente.

Por esa razón algunas modificaciones al concepto de forma alfa y a la metodología para calcularles han sido propuestas. Así surgen entre otros, las *formas alfa con pesos* [88]. Utilizando

¹En su momento, el algoritmo para el cálculo de la forma alfa propuesto en 1983 [85] por Hebert Edelsbrunner aplicaba únicamente a dos dimensiones, sólo hasta 1994 fue extendido a tres dimensiones [87]

la triangulación de Delaunay sólo se consideran ciertos vértices y simplexes que cumplen con tener un cierto peso, lo que haría mucho más rápido el proceso de cálculo de la forma alfa. Existen también las *formas alfa con pesos anisotrópicos* [89] que a partir de la triangulación de Delaunay utiliza para construir los simplexes únicamente vértices que son elegidos a partir de una distribución no uniforme y usando un tensor métrico para ello. Sin embargo, para estas versiones más rápidas no se conoce como podrían ser implementadas de manera efectiva, en más de dos y tres dimensiones [90]. Esto último limita de sobremanera el uso de la forma alfa en cualquiera de sus versiones, para extraer las muestras o puntos en las orillas de la nube de puntos.

En esta investigación se propone una metodología eficiente para *aproximar la forma alfa* usando para ello la envolvente convexa o convex hull. Téngase en cuenta que a diferencia de la forma alfa, *el algoritmo para el cálculo de las envolventes convexas si es implementable en cualquier número de dimensiones*.

En realidad la idea de aproximar una forma compleja en general, usando la unión de formas más sencillas es bastante conocida. Por ejemplo, para formas tres dimensiones se han usado esferas [91] y también en [92]. Sin embargo estas metodologías además de ser complejas y combinatorias, sólo aplican a tres dimensiones. Para aproximar la forma alfa en cualquier dimensión utilizando envolventes convexas, se propone particionar o dividir la nube original de puntos C_i , para $i = 1, 2, \dots, n_C$ en una cantidad finita de agrupamientos G_j , para $j = 1, 2, \dots, k$. Luego para cada agrupamiento obtener su envolvente convexa $CH(G_j)$ de modo que la aproximación a la forma alfa $\alpha'(C_i)$ es la unión de esas envolventes convexas, tal como se muestra en la Ecuación 4.1.

$$\alpha'(C_i) = \bigcup_{j=1}^k CH(G_j) \quad (4.1)$$

Este procedimiento se aprecia gráficamente en la Figura 4.3. En este ejemplo bidimensional se ilustra el caso de una nube de puntos que es particionada en k agrupamientos, para cada uno se obtiene su envolvente convexa y se aproxima su forma alfa uniendo esas envolventes convexas. La nube original de puntos se muestra en la Figura 4.3a y su forma alfa ideal se muestra en la Figura 4.3b. La aproximación a la forma alfa depende del valor específico k . Un valor muy grande de k fragmentaría demasiado la nube de puntos, en un caso extremo $\lim_{k \rightarrow \infty} \alpha'(C_i) = C_i$ para $i = 1, 2, \dots, n_C$, es decir en un caso como $k \rightarrow \infty$ el particionamiento dejaría la nube de puntos exactamente igual.

La Figura 4.4 muestra el resultado de aplicar el proceso de aproximar la forma alfa a la

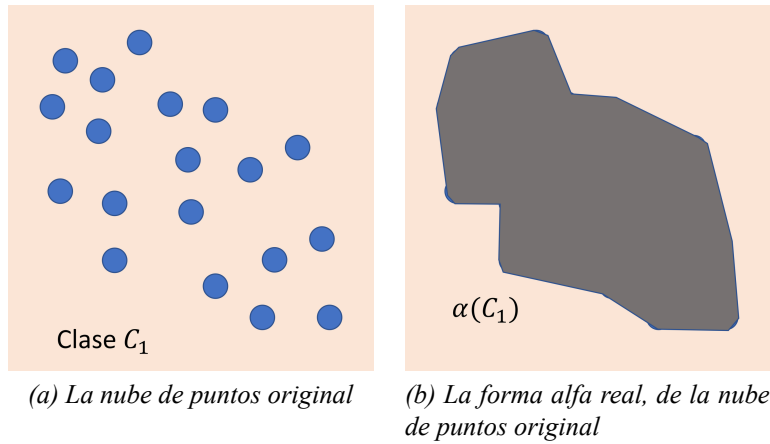


Figura 4.3: La nube original de puntos en $\mathbb{R}^2$

nube de puntos mostrada en la Figura 4.3, con $k = 4, k = 3, k = 2$ y $k = 1$. La unión de las envolventes convexas se muestra sombreando las áreas o volúmenes de los poliedros resultantes, representando las zonas que se consideran parte de la forma alfa. Así resulta evidente que entre más agrupamientos se usan, mayor separación entre poliedros existe y con ello, resultan muchos huecos o vacíos entre los poliedros, que en la forma alfa original no aparecen. Lo que implica que existirán zonas en las aproximaciones que no serán consideradas parte de la forma alfa, cuando en realidad en el caso ideal sí lo son.

Por otro lado, se puede apreciar que entre menos agrupamientos se utilizan la forma final de la aproximación tiende a ser un poliedro convexo, en el en que $k = 1$ la aproximación a la forma alfa es la envolvente convexa de la nube original de puntos. Esto permite concluir que entre menos agrupamientos se usan, existe la posibilidad de que existan algunas zonas que serán consideradas como parte de la forma alfa cuando en realidad no lo son. La Figura 4.5 muestra una comparativa de la forma alfa real de la nube de puntos en los dos últimos ejemplos, y las aproximaciones logradas con $k \in \{1, 2, 3, 4\}$, en dos dimensiones. Estas son las mismas aproximaciones mostradas en la Figura 4.4 pero alineadas para una más sencilla comparación.

La presencia de esos huecos o vacíos entre agrupamientos mina la aproximación adecuada de la forma alfa. Esta particularidad es más común cuando la nube de puntos es muy dispersa y sin suficiente población. Es decir que el problema de la forma alfa hueca suele aparecer en nubes de puntos o muestras que están muy dispersas, sin una clara tendencia central, y además cuando la cantidad de muestras no es muy grande. Es bajo estas condiciones que la forma alfa aproximada puede no ser muy similar a la forma alfa real.

Aquellas zonas de la forma alfa real mostradas en color azul en cualquiera de las gráficas en

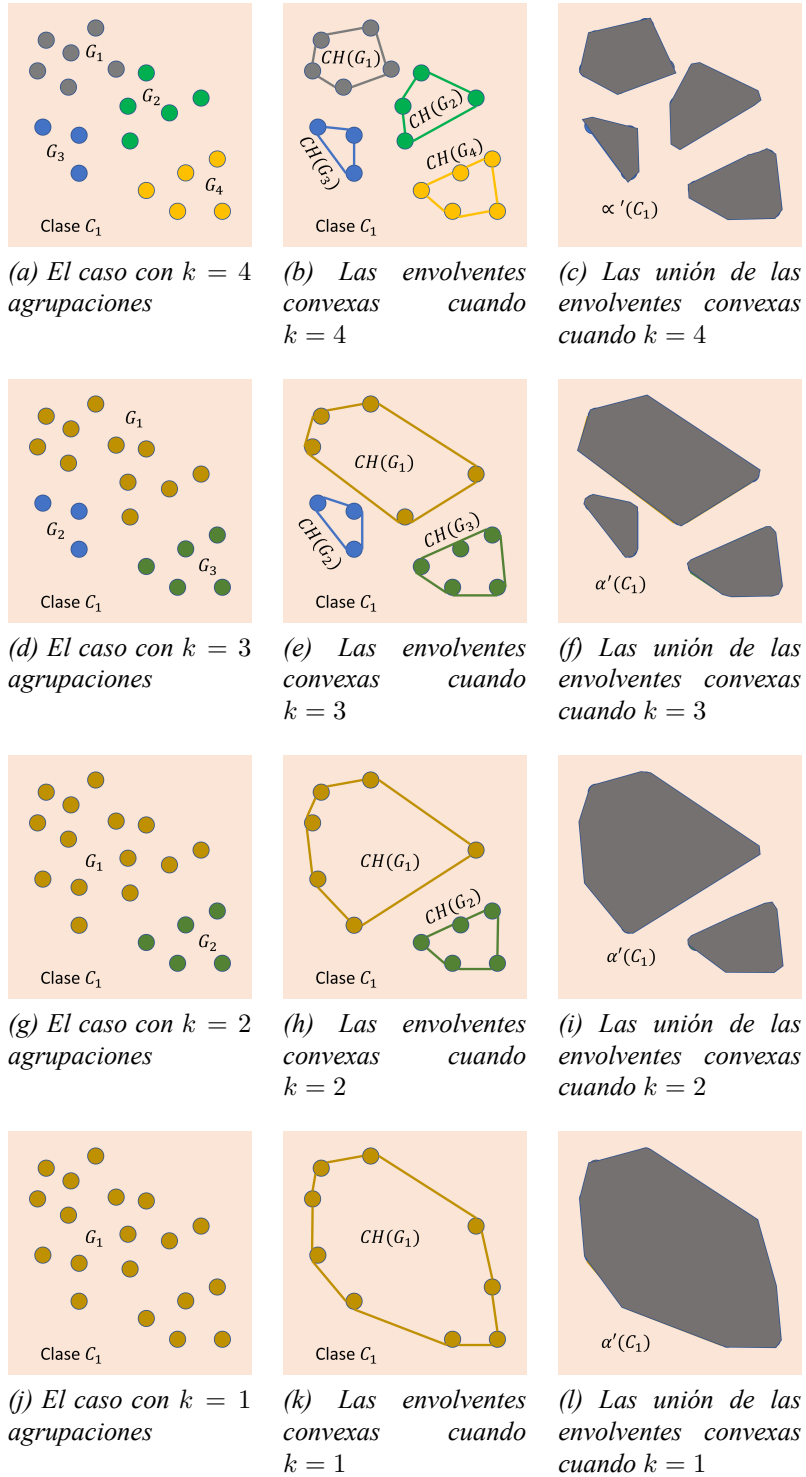


Figura 4.4: Cuatro casos de agrupamientos para la misma nube de la Figura 4.3

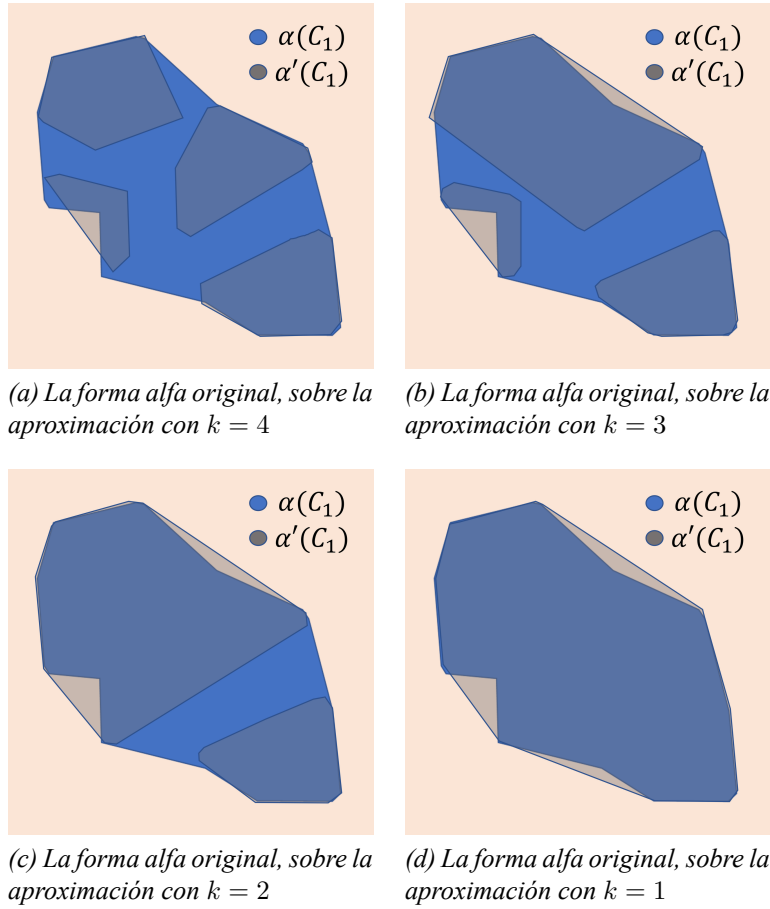


Figura 4.5: Las aproximaciones a la forma alfa, $\alpha'(C_1)$, en $\mathbb{R}^2$, de mayor a menor cantidad de agrupamientos, en comparación con la forma alfa, $\alpha(C_1)$, ideal

la Figura 4.5 y que no son llenadas por la aproximación a la forma alfa, mostrada en color gris en las figuras, son zonas de *falsos negativos*, en donde una muestra o punto no sería identificado como adentro de la forma alfa aproximada cuando en la forma alfa real si se encuentra adentro. Por otro lado, aquellas zonas de las formas alfa aproximadas que no son completadas por la forma alfa real son zonas de *falsos positivos*, cualquier muestra adentro de ellas sería identificado como parte de la forma alfa aproximada cuando en la forma alfa real no forman parte.

Así que la cantidad de particiones k a utilizar deberá depender de la cantidad de información que el dataset contiene. Se sabe que un método de particionamiento como *kmeans*, resulta particularmente efectivos en datasets grandes [93, 94]. Una regla muy aceptada es que un buen dataset con n_E dimensiones normalmente contiene una cantidad de muestras n_P , según la siguiente regla $n_P = 10 * n_E$ [95]. Si es el caso de que $n_P < 10 * n_E$ el dataset es pequeño. Si ocurre que $n_P > 10 * n_E$ el dataset es grande. De acuerdo con esto, el proceso de establecer el

valor k pasa por dos etapas:

- (1) Determinar si el dataset es pequeño o grande.
- (2) De acuerdo con lo anterior, el valor k se establece en $k = 1$ si el dataset es pequeño, en caso contrario, k es establecida llevando a cabo una prueba de particionamientos, hasta encontrar el mejor valor de k .

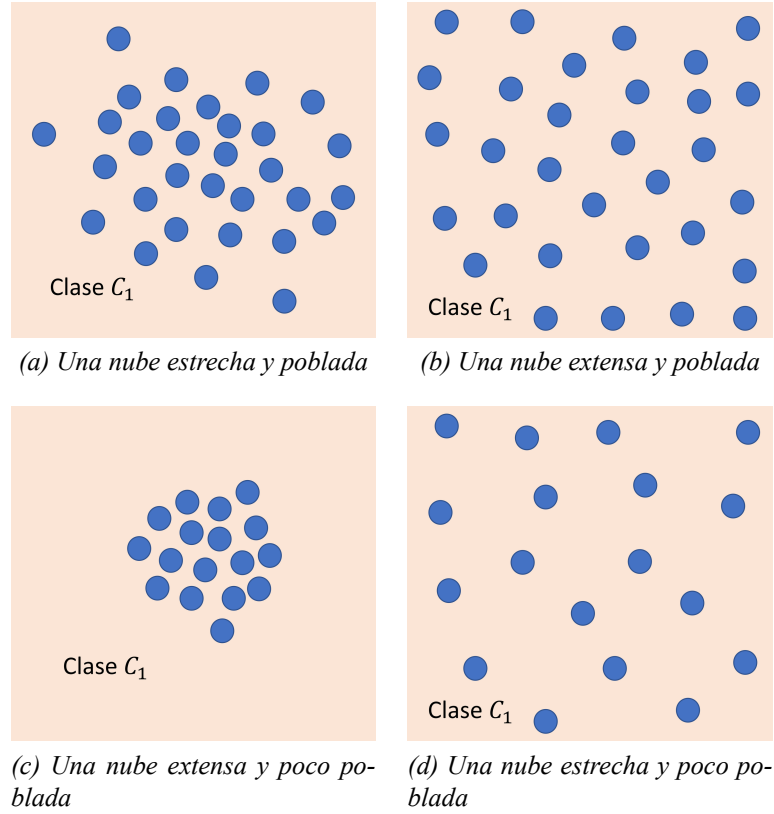


Figura 4.6: Cuatro casos de dispersión y extensión, en $\mathbb{R}^2$

Para establecer el total de particiones en las que la nube de puntos original se deberá separar, se usa el *criterio Calinski-Harabasz* [96]. De modo que usando este, será factible encontrar la cantidad óptima de agrupamientos para la nube de puntos C_i , para $i = 1, 2, \dots, n_C$.

El criterio Calinski-Harabasz es muy usado para analizar distintos casos de particionamientos del grupo o nube de puntos C_i . Usando diferentes valores para k , la cantidad de particiones o agrupamientos en que se descompone la nube de puntos, será posible encontrar el valor óptimo para ese mismo parámetro k . Es sumamente importante tener en cuenta que el criterio Calinski-Harabasz asume que los particionamientos ocurren usando la técnica *k-means*. El criterio es sim-

ple, dado un valor k específico el criterio Calinski-Harabsz mide su utilidad usando la siguiente relación:

$$calhar(k) = \left(\frac{B(k)}{W(k)} \right) \left(\frac{n_P - k}{k - 1} \right) \quad (4.2)$$

En la Ecuación 4.2, $B(k)$ se refiere a la varianza adentro de las particiones o agrupamientos, $W(k)$ se refiere a la varianza entre particiones o agrupamientos, n_P al total de muestras en el conjunto o nube de puntos C_i , y k a la cantidad de agrupamientos o particiones en que se divide la nube de puntos. Entonces distintos valores para k son analizados o probados, y se elige aquel valor de k que maximiza el índice $calhar(k)$.

Así pues, incluyendo el método de particiones, el algoritmo completo para llevar a cabo la aproximación a la forma alfa de una nube de puntos o muestras C_i , para $i = 1, 2, \dots, n_C$ es el siguiente.

Algoritmo 7: Algoritmo para la aproximación a la forma alfa

```

1 Function  $aShape(C)$ 
2   Establecer un valor para  $k_{max}$ , el máximo número de particiones;
3   Hacer  $max\_val = 0$ ;
4   for  $k = 1$  to  $k_{max}$  do
5     Calcular  $B(k)$ ;
6     Calcular  $W(k)$ ;
7     Calcular  $calhar(k) = \left( \frac{B(k)}{W(k)} \right) \left( \frac{n_P - k}{k - 1} \right)$  if  $max\_val < calhar(k)$  then
8       |  $max = k$ ;
9     end
10  end
11  Hacer  $k = max$ ;
12  Hacer  $Grupos = kmeans(C, k)$ ;
13  Hacer  $\alpha' = \{\}$ ;
14  for  $i = 1$  to  $k$  do
15    | Hacer  $\alpha' = \alpha' \cup CH(Grupo_i)$ ;
16  end
17  return  $\alpha'(C)$ 

```

El Algoritmo 7 primero establece el valor correcto para k , la cantidad de agrupamientos en los que la nube original de puntos se divide. Luego se particiona la nube original de puntos en k agrupamientos, usando el método de agrupamientos conocido como k -means. Luego para cada una de estas particiones se calcula su envolvente convexa $CH(G_i)$, y finalmente con esto, se aproxima la forma alfa como la unión de esas envolvente convexas de cada agrupamiento.

La aproximación $\alpha'(C_i)$ para $i = 1, 1, \dots, n_C$ no es una estructura que contiene todo lo que una forma alfa real contendría, sólo contiene información sobre los volúmenes en $\mathbb{R}^{n_E}$ que aproximan al volumen en la forma alfa real.

4.1.3. Ejemplos prácticos de la aproximación a la forma alfa

Antes de continuar con la aproximación a la frontera entre clases se presentan algunos ejemplos prácticos realizados en Matlab, acerca de la aproximación a la forma alfa de una clase representada por n_p muestras para lograr una clara representación, estos ejemplos se limitan a casos en $\mathbb{R}^2$ y $\mathbb{R}^3$. Se presentan en cada ejemplo la clase original, el valor resultante de k , para cada caso, el particionamiento de la clase y la resultante forma alfa aproximada.

4.1.3.1. Ejemplo 4.1.3a

Este ejemplo es el de una clase en $\mathbb{R}^2$ que consta de $n_p = 900$ muestras. Esta clase ha sido dividida en $k = 3$ grupos G_1 , G_2 y G_3 . Estos agrupamientos, sus envolventes convexas así como la unión de las mismas que aproximan la forma de la clase se muestran en la Figura 4.7.

Se puede ver en la Figura 4.7a a la clase original. Esta clase según la metodología antes analizada, queda particionada en $k = 3$ agrupamientos. Se puede observar que se conserva la forma general de la clase. Únicamente en el agrupamiento verde se puede observar un hueco relativamente importante en la esquina superior derecha.

4.1.3.2. Ejemplo 4.1.3b

La clase contiene $n_p = 1000$ muestras. En este caso el valor óptimo para la cantidad de muestras es $k = 4$. La Figura 4.8 muestra el procedimiento completo de aproximar la forma de la clase.

Como la Figura 4.8a muestra, en este ejemplo la clase tiene una forma más compleja, tiene una forma cóncava. Se puede observar en la Figura 4.8c y en la Figura 4.8d, que se respeta la forma cóncava original. Existen algunos huecos detectables en cada una de las envolventes convexas que componen la forma alfa.

4.1.3.3. Ejemplo 4.1.3c

En este ejemplo la clase consta de $n_P = 1000$ muestras. El particionamiento ha decidido para este ejemplo que $k = 5$. Como en el ejemplo anterior, la clase presenta una forma cóncava compleja. El resultado de todo el proceso aparece en la Figura 4.9 a continuación.

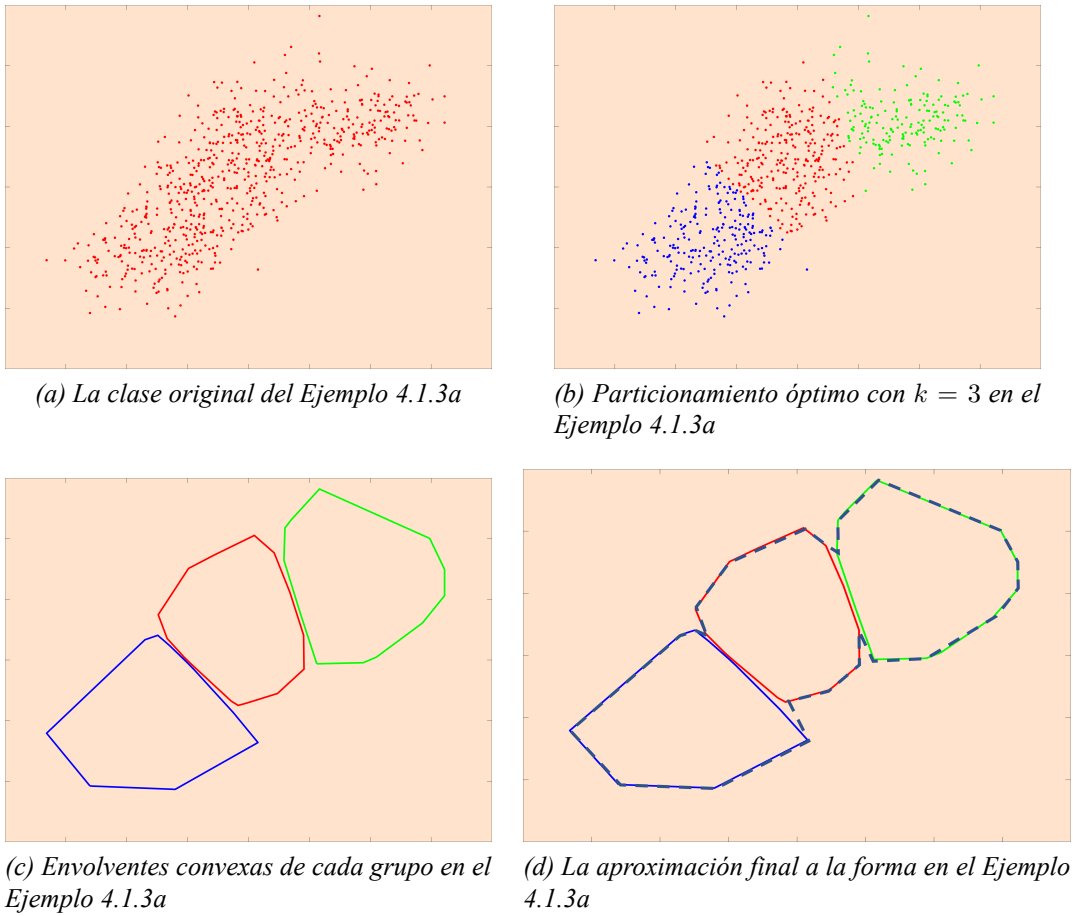


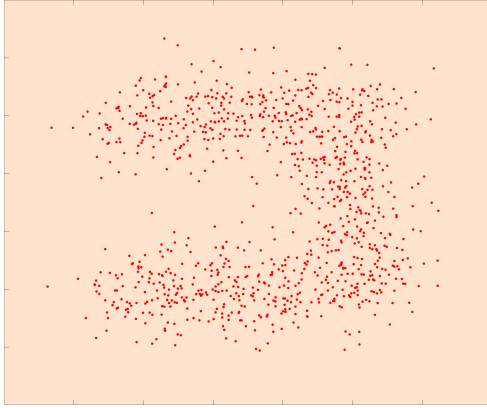
Figura 4.7: La aproximación a la forma de una clase en el ejemplo 4.1.3a en $\mathbb{R}^2$

La aproximación respeta la forma de la clase en general. Igualmente se pueden observar algunos huecos en las envoltentes convexas, sin afectar la forma básica de todo el conjunto.

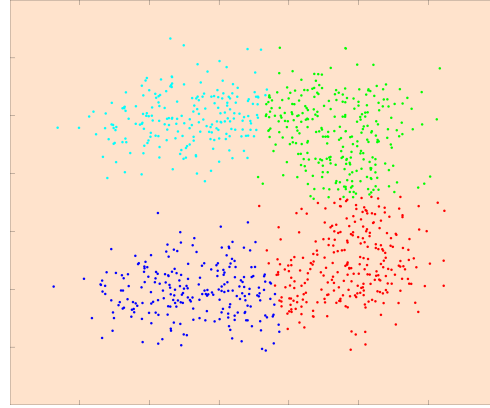
4.1.3.4. Ejemplo 4.1.3d

Este es un ejemplo de una clase en $\mathbb{R}^3$. La clase está formada por $n_P = 900$ muestras. La clase tiene una forma cóncava, por lo que la aproximación a la forma debería ser lo más fina posible. En este ejemplo la cantidad óptima de agrupamientos es $k = 4$. La siguiente Figura 4.10 ilustra todo el proceso de aproximación a la forma de la clase.

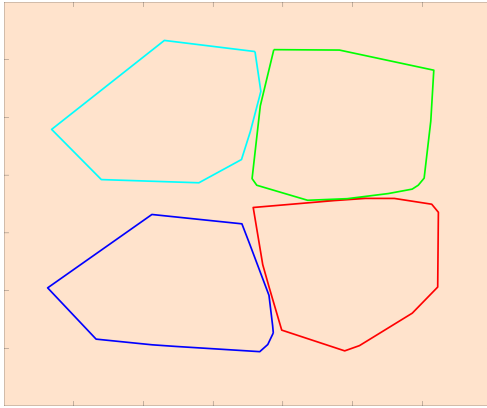
Se puede apreciar como el método propuesto funciona en tres dimensiones de la misma manera que lo hace en dos dimensiones. De hecho funciona igual en cualquier número de dimensiones. En la Figura 4.10a aparece la clase original. En la Figura 4.10b el particionamiento óptimo en $k = 4$ grupos G_i para $i = 1, 2, 3, 4$, cada grupo se presenta con un color diferente. Se puede



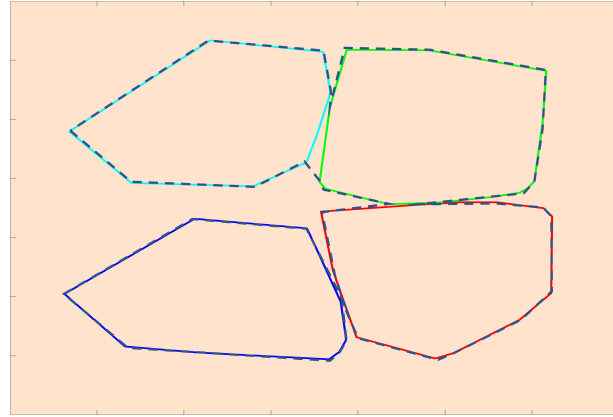
(a) La clase original en el Ejemplo 4.1.3b



(b) Particionamiento óptimo con $k = 4$ en el Ejemplo 4.1.3b



(c) Envoltentes convexas de cada grupo en el Ejemplo 4.1.3b



(d) La aproximación final a la forma en el Ejemplo 4.1.3b

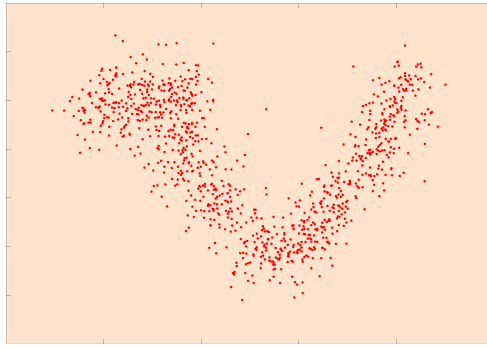
Figura 4.8: La aproximación a la forma de una clase en el Ejemplo 4.1.3b en $\mathbb{R}^2$

observar que la forma básica se conserva con un buen detalle en la aproximación a la forma alfa mostrada en la Figura 4.10d, aún y cuando existen algunos huecos entre las envoltentes convexas.

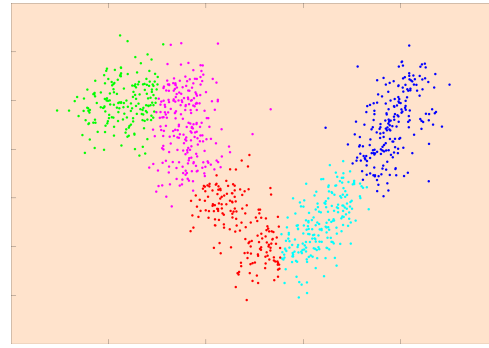
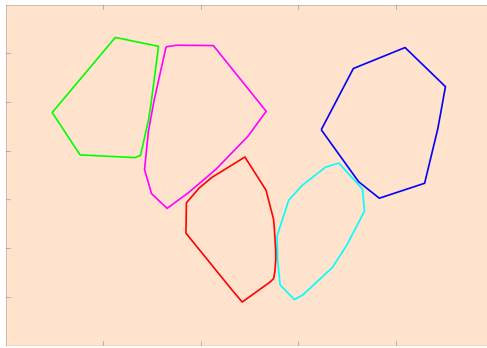
4.2. Los bordes fundamentales de una clase

En las envoltentes convexas $CH(G_j^i)$ para $j = 1, 2, \dots, k_i$ se tiene que cada una está delimitada por una serie de vértices $\mathbf{v}_l^{k_i}$ para $l = 1, 2, \dots, n_{v_k}$, siendo v_k el total de vértices en la envoltente convexa del agrupamiento G^j . Obsérvese en la Figura 4.11 los vértices en una aproximación a la forma de una clase.

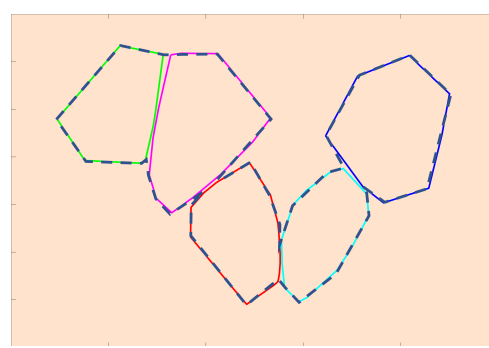
En lo que a esta investigación concierne los vértices que interesan son los llamados *bordes*



(a) La clase original en el Ejemplo 4.1.3c

(b) Particionamiento óptimo con $k = 5$ en el Ejemplo 4.1.3c

(c) Envoltentes convexas de cada grupo en el Ejemplo 4.1.3c



(d) La aproximación final a la forma en el Ejemplo 4.1.3c

Figura 4.9: La aproximación a la forma de una clase en el Ejemplo 4.1.3c en $\mathbb{R}^2$

fundamentales. Un vértice $\mathbf{v}_j^i$ se considera borde fundamental si y sólo si su presencia afecta la localización y la forma de la frontera entre clases o superficie de decisión entre clases. La Figura 4.12 ilustra el concepto de bordes fundamentales en un caso de clasificación entre dos clases en dos dimensiones.

0

En la Figura 4.12 los bordes fundamentales se muestran con un contorno en color verde. A fin de localizar a estos se propone un procedimiento exhaustivo, que uno a uno busca las muestras en las orillas de los poliedros de cada envoltente convexa que aproximan la forma alfa, para establecer cuales de ellos corresponden con aquellos en los bordes de la nube original. Una forma es revisar la recta de cada vértice de una clase, con respecto a los centroides de la otra clase, para detectar los vértices que corresponden con rectas que no cruzan por la forma alfa de la misma clase a la que pertenecen. Esto se aplica a cada clase.

En esta investigación se propone la siguiente metodología: usar el vértice $\mathbf{v}_k$ en una de las envoltentes convexas en la aproximación a la forma de la otra clase $\alpha'(C_k)$ para $k = 1, 2, \dots, n_C$

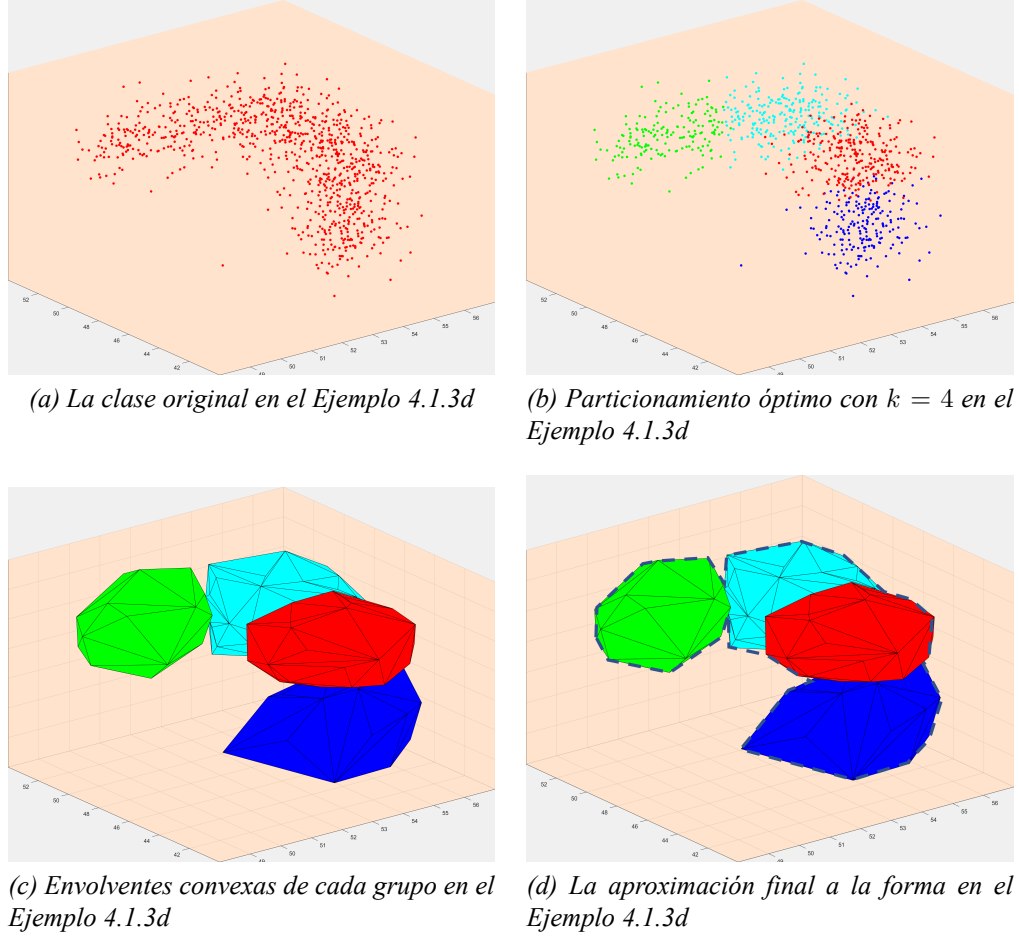


Figura 4.10: La aproximación a la forma de una clase en el Ejemplo 4.1.3d en $\mathbb{R}^3$

y $k \neq j$, más cercano al vértice $\mathbf{v}_i \in \alpha'(C_i)$. Esto se explica en detalle en la siguiente subsección.

4.2.1. Usando el vértice más cercano en la otra clase

El problema para encontrar los bordes fundamentales en clases con formas no convexas es que un borde fundamental puede encontrarse justo detrás de otras muestras en la misma clase, de modo que cuando la línea recta que lo une con el centroide de la otra clase es posicionada, siempre intersecta alguna zona de la propia clase en donde el borde fundamental reside, lo que lo deja fuera de elección según el método del centroide de la otra clase.

En casos como el anterior es mejor elegir otro destino para la línea que parte del vértice candidato a borde fundamental en la clase C_i , hacia la otra clase C_j . En lugar de buscar el centroide de la segunda clase como destino, busca para cada candidato el vértice $\mathbf{z}_j$ en la clase C_j más

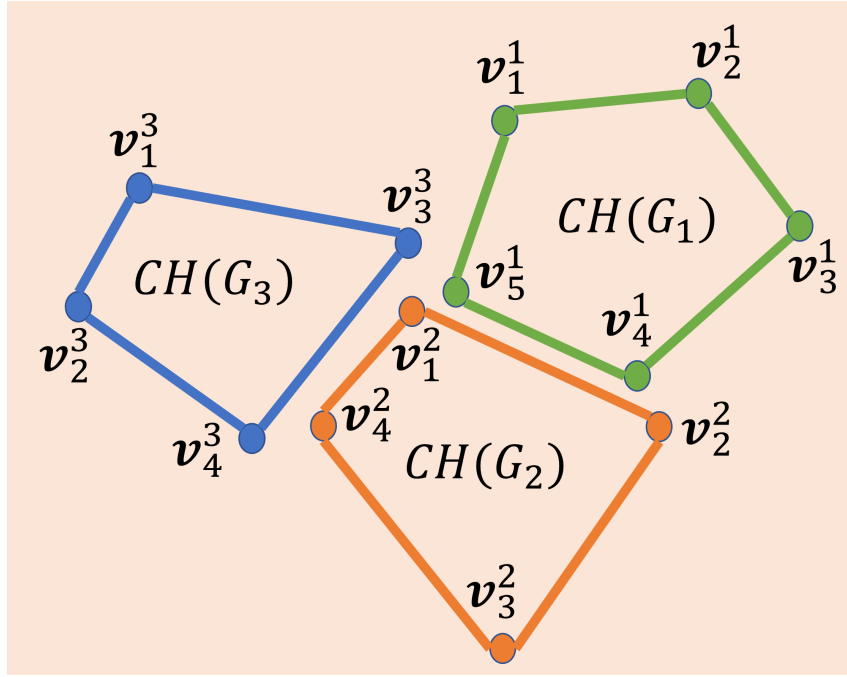


Figura 4.11: Una aproximación a la forma de una clase en $\mathbb{R}^2$

cercano al vértice candidato a borde fundamental.

La Figura 4.13 es la representación gráfica del método del vértice más cercano en la otra clase. Aquí en la forma aproximada $\alpha'_1(C_1)$ aparecen cuatro vértices candidatos a bordes fundamentales, denotados en la Figura 4.7b como $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ y $\mathbf{v}_4$. En esta situación el vértice $\mathbf{z}_1$ es el más cercano a $\mathbf{v}_1$, mientras que el vértice $\mathbf{z}_2$ es el más cercano a $\mathbf{v}_2$, a $\mathbf{v}_3$ y $\mathbf{v}_4$. Sin embargo los vértices $\mathbf{v}_2$ y $\mathbf{v}_3$ no pueden ser bordes fundamentales en C_1 , la línea recta que los une con el vértice más cercano en la clase C_2 intersecta la aproximación a la forma alfa α'_1 como se muestra en la Figura 4.13.

El Algoritmo 8 muestra el procedimiento para establecer los bordes fundamentales en dos clases, usando el vértice más cercano en la otra clase

4.2.2. El uso de programación lineal para la aproximación a la envolvente convexa

El procedimiento para obtener los bordes fundamentales descrito en la subsección anterior implica que se debe calcular la envolvente convexa $CH(g_j)$, y luego verificar si $\mathbf{v}_i \in CH(g_j)$. Esto no sería un problema para pocas dimensiones, pues Qhull y CGAL lo logran fácilmente. Cuando la clasificación ocurre en grandes dimensiones, incluso el cálculo de la envolvente con-

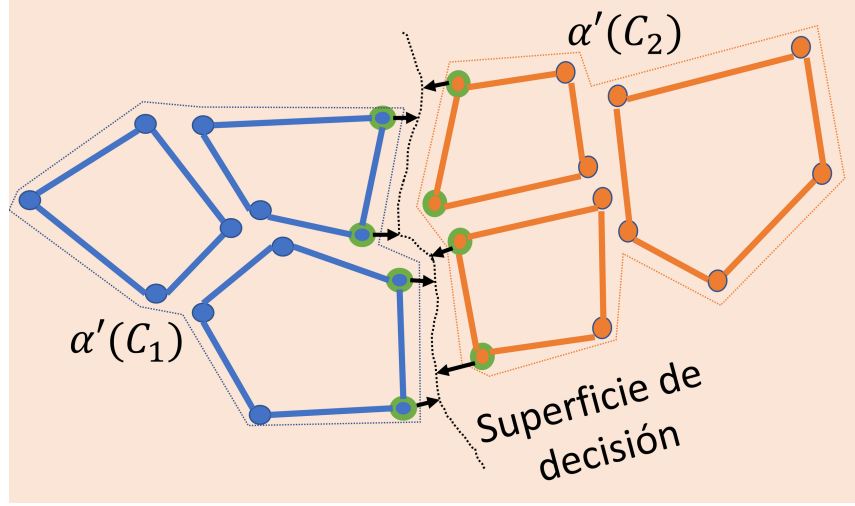


Figura 4.12: Los bordes fundamentales en dos clases en $\mathbb{R}^2$. Los bordes fundamentales son los vértices cuyo contorno aparece en color verde, en ambas clases

vexa, usando librerías de geometría computacional como Qhull y CGAL, se complica. Por ello, otro camino puede seguirse, por ejemplo, usar *programación lineal* y *optimización convexa*.

En un espacio $\mathbb{R}^{n_E}$, en donde existen k muestras $\mathbf{p}_i \in \mathbb{R}^{n_E}$ para $i = 1, 2, \dots, k$, agrupadas en un conjunto $\mathbf{P}$, como en $\mathbf{P} = \{\mathbf{p}_i\}$. Además, existe la envolvente convexa de este conjunto, denotada $CH(\mathbf{P})$. Dada una muestra $\mathbf{v} \in \mathbb{R}^{n_E}$ ¿Se localiza $\mathbf{v}$, adentro de la envolvente convexa $CH(\mathbf{P})$? Para decidir esto sin el uso de librerías de geometría computacional, se puede plantear la solución a esta pregunta como la solución a un problema de programación lineal [97]. La envolvente $CH(\mathbf{P})$, se puede expresar como en la expresión 4.3.

$$CH(\mathbf{P}) = \left\{ \mathbf{p} | \mathbf{p} = \sum_{i=1}^k \alpha_i \mathbf{p}_i \wedge \alpha_i \geq 0 \wedge \sum_{j=1}^k \alpha_j = 1 \right\} \quad (4.3)$$

De la Ecuación 4.3 se desprende el hecho de que cualquier muestra que se localice adentro de una envolvente convexa se debe descomponer como una combinación lineal de las mismas muestras que le dan lugar a la envolvente, con los debidos coeficientes α_i , para $i = 1, 2, \dots, k$. Luego, si $\mathbf{v} \in CH(\mathbf{P})$ entonces se debe cumplir la Ecuación 4.4.

$$\mathbf{v} = \sum_{i=1}^k \alpha_i \mathbf{p}_i \wedge \alpha_i \geq 0 \wedge \sum_{j=1}^k \alpha_j = 1 \quad (4.4)$$

De esta manera es necesario encontrar el conjunto de coeficientes α_i que dan lugar a as combinaciones lineales como en la Ecuación 4.3 y en la Ecuación 4.4. Encontrar tales coeficientes es una tarea resoluble eficientemente utilizando programación lineal. Esto ocurre como en el

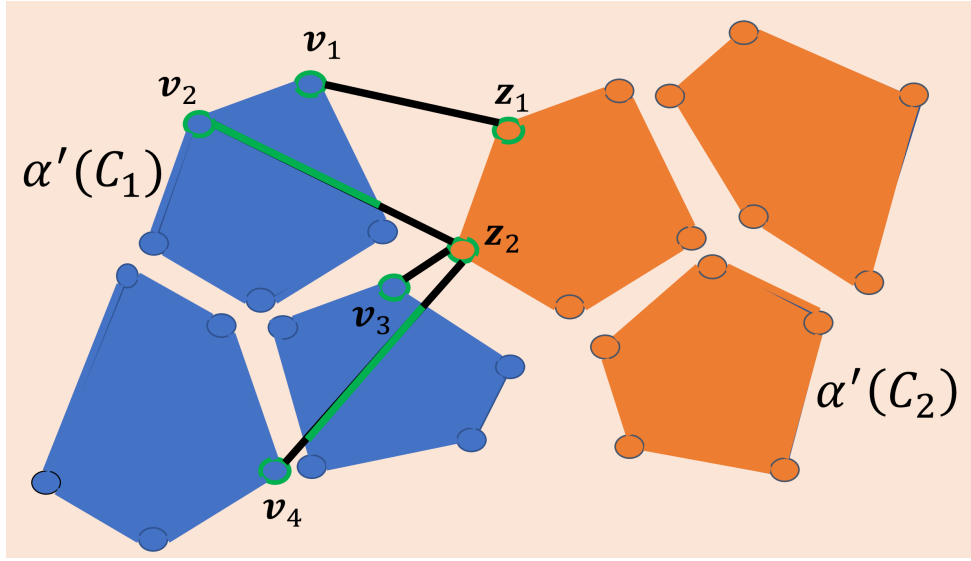


Figura 4.13: Estableciendo los bordes fundamentales con el método del vértice mas cercano

planteamiento descrito en la Ecuación 4.5.

$$\text{mín} \quad \mathbf{c}^T \boldsymbol{\alpha} \quad (4.5a)$$

$$\text{sujeto a:} \quad \mathbf{A}\boldsymbol{\alpha} = \mathbf{v} \quad (4.5b)$$

$$\mathbf{B}\boldsymbol{\alpha} = 1 \quad (4.5c)$$

$$\boldsymbol{\alpha} \geq 0 \quad (4.5d)$$

En la Ecuación 4.5, $\mathbf{c} \in \mathbb{R}^k$ es un vector arbitrario de costo, $\mathbf{A} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_k) \in \mathbb{R}^{k \times k}$ y $\mathbf{B} = (1, 1, \dots, 1) \in \mathbb{R}^{1 \times k}$, mientras que $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_k) \in \mathbb{R}^{1 \times k}$.

Si el problema establecido en el planteamiento anterior tiene solución entonces será un hecho que $\mathbf{v} \in CH(\mathbf{P})$. La ventaja de este enfoque es que no debe calcular la envolvente convexa en sí misma, todo se reduce a resolver un problema de programación lineal. Este proceso se facilita con un resolutor de problemas de optimización, por ejemplo *el toolbox de optimización de Matlab*.

4.2.3. Ejemplos sobre la extracción de los bordes fundamentales

A continuación se presentan algunos ejemplos de extracción de bordes fundamentales en problemas binarios de clasificación en $\mathbb{R}^2$ y $\mathbb{R}^3$, según el método expuesto en la última sección.

Algoritmo 8: Algoritmo para extracción de bordes fundamentales

```

1 Function fundamentales2( $\alpha'(C_1), \alpha'(C_2)$ )
2   hacer  $fundam_{C_1} = \{\}$ ;
3   for  $v \in C_1$  do
4     Hacer  $z$  el vértice en  $\alpha'(C_2)$  más cercano a  $v$ ;
5     Trazar la línea que une a  $v$  con su correspondiente  $z$ , llámese  $\vec{vz}$  a tal línea.;
6     Hacer  $inter = \{\}$ ;
7     for  $G \in \alpha'(C_1)$  do
8        $inter = inter \cup \vec{vz}$ ;
9     end
10    if  $inter = \{v\}$  then
11       $fundam_{C_1} = fundam_{C_1} \cup \{v\}$ ;
12    end
13  end
14  Hacer  $fundam_{C_2} = \{\}$ ;
15  for  $v \in C_2$  do
16    Hacer  $z$  el vértice en  $\alpha'(C_1)$  más cercano a  $v$ ;
17    Trazar la línea que une a  $v$  con su correspondiente  $z$ , llámese  $\vec{vz}$  a tal línea.;
18    Hacer  $inter = \{\}$ ;
19    for  $G \in \alpha'(C_2)$  do
20       $inter = inter \cup \vec{vz}$ ;
21    end
22    if  $inter = \{v\}$  then
23       $fundam_{C_2} = fundam_{C_2} \cup \{v\}$ ;
24    end
25  end
26  return  $\{fundam_{C_1}, fundam_{C_2}\}$ 

```

4.2.3.1. Ejemplo 4.2.3a

En este problema ambas clases tienen forma cóncava y son perfectamente separables.

La Figura 4.14a muestra la clase C_1 en color rojo, la clase C_2 , en color azul. La Figura 4.14b muestra los bordes fundamentales en ambas clases como fueron determinados usando el método propuesto.

4.2.3.2. Ejemplo 4.2.3b

En este ejemplo la particularidad es que una de las clases tiene una forma claramente cóncava y la otra notablemente convexa, además el problema de clasificación está bastante desbalanceado como lo ilustra la Figura 4.15. La clase C_2 tiene el doble de muestras 1200 que la clase minoritaria

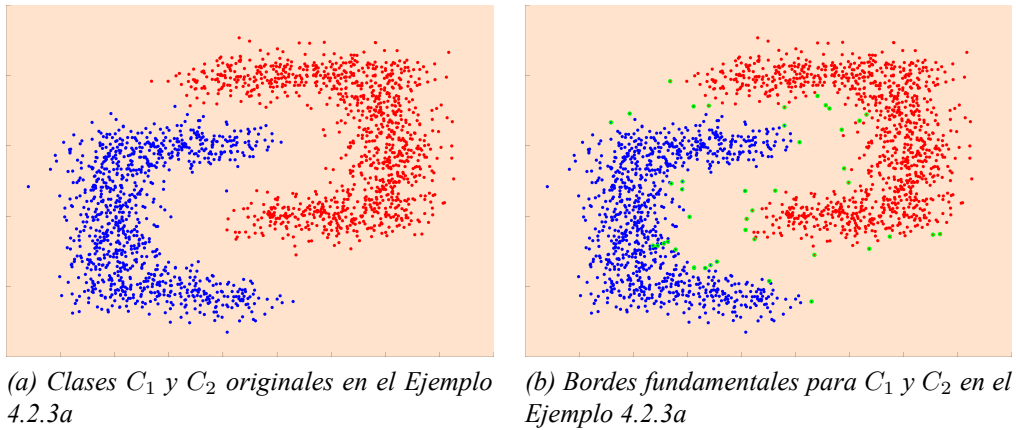


Figura 4.14: Bordes fundamentales en el ejemplo 4.2.3a

C_1 , que sólo posee 600 muestras.

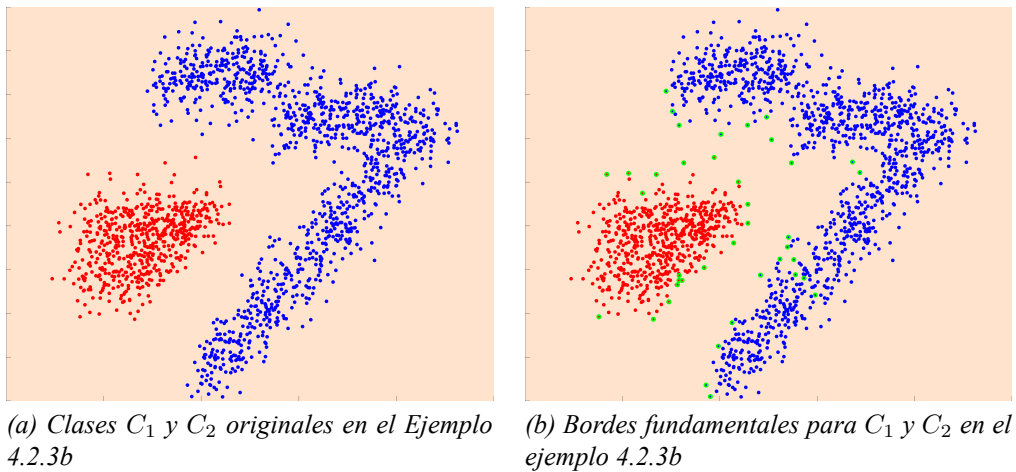


Figura 4.15: Bordes fundamentales en el ejemplo 4.2.3b

La Figura 4.15a muestra la distribución de las dos clases. La Figura 4.15b en donde la clase C_1 se muestra en color rojo, posee un total de 61 bordes fundamentales y la clase C_2 sólo 11 bordes fundamentales que aparecen con un contorno en color verde en ambas clases.

4.2.3.3. Ejemplo 4.2.3c

Este es un ejemplo entre dos clases C_1 y C_2 cuya forma es visiblemente convexa. Ambas clases se componen de 300 muestras. Además las clases C_1 y C_2 son linealmente separables. Las figuras a continuación muestra la distribución de ambas clases. La clase C_1 aparece en color rojo, la clase C_2 en color azul.

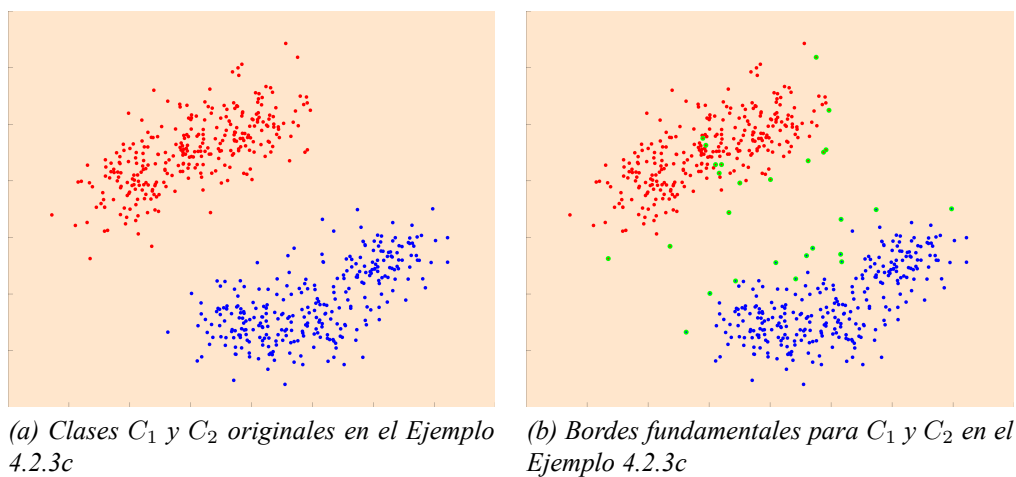


Figura 4.16: Bordes fundamentales en el ejemplo 4.2.3c

Desde un punto de vista de clasificación de patrones el problema de clasificación mostrado en la Figura 4.16a es bastante simple, pues las clases son linealmente separables. Los bordes fundamentales en cada clase son mostrados en la Figura 4.16b en color verde en cada clase.

4.2.3.4. Ejemplo 4.2.3d

Este es un problema de clasificación binaria en donde la clase C_1 tiene 1700 muestras y la clase C_2 tiene 2200 de ellas. Como lo muestra la Figura 4.17. Este problema si bien es perfectamente separable, muestra una frontera entre clases bastante compleja.

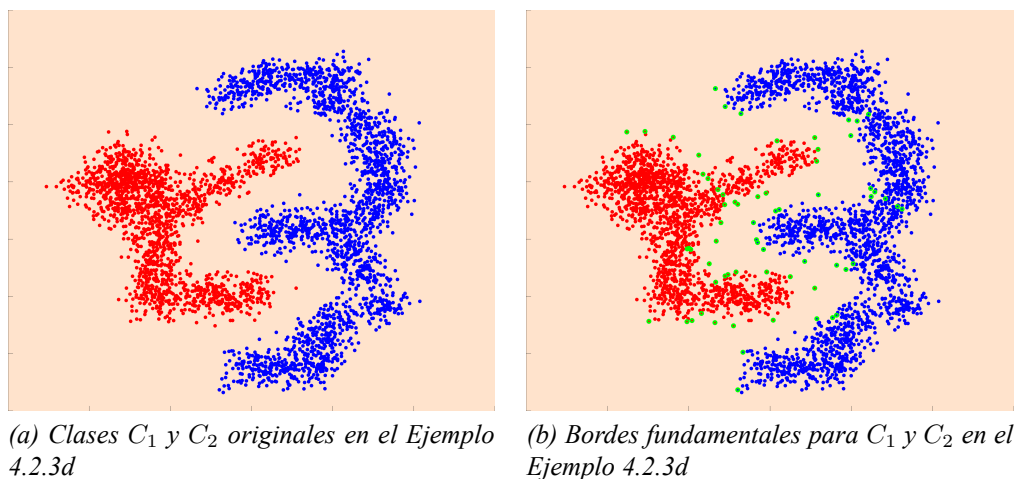


Figura 4.17: Bordes fundamentales en el Ejemplo 4.2.3d

En todos estos ejemplos los bordes fundamentales se han obtenido según el método del vér-

tice más cercano en la otra clase, para un problema de clasificación binario.

4.2.3.5. Ejemplo 4.2.3e

Este es un ejemplo de dos clases en $\mathbb{R}^3$. La clase C_1 contiene 1200 muestras y la clase C_2 contiene 1400 muestras. Las clases son separables, aunque no linealmente separables. La Figura 4.18a muestra la disposición de las clases así como la Figura 4.18b muestra los bordes fundamentales en cada una.

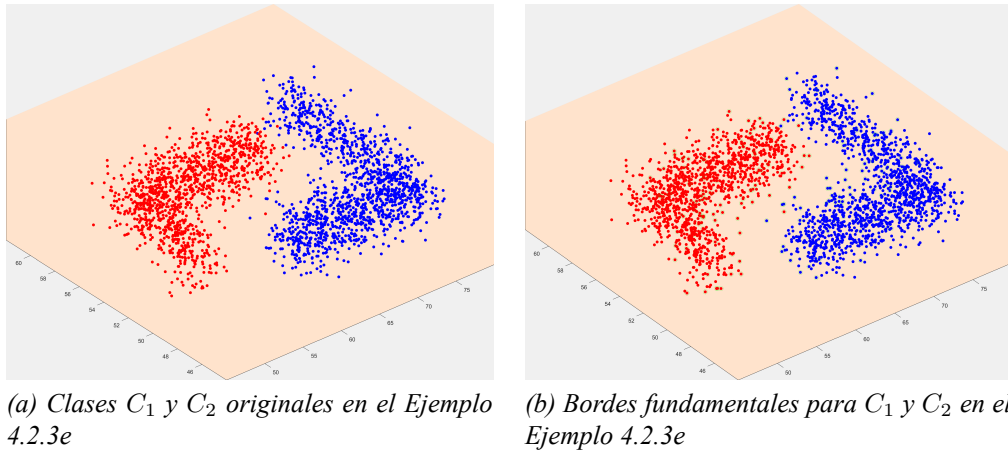


Figura 4.18: Bordes fundamentales en el ejemplo 4.2.3e

4.3. El promedio entre clases

Utilizando los bordes fundamentales, que han sido extraídos mediante el método expuesto en la subsección anterior, se obtiene el promedio entre clases, representando este, la superficie de decisión, misma que será aproximada mediante hiperplanos en subsecciones siguientes. La siguiente subsección revisa el concepto de superficie de decisión, para continuar en la siguiente subsección con la metodología para aproximarle usando el promedio entre clases.

4.3.1. Superficies de decisión

En un problema de clasificación, considerando un problema entre dos clases, *una superficie de decisión se refiere a la hipersuperficie que divide al espacio del problema en dos particiones, una por cada clase*. Así pues, suponiendo que la partición de la clase C_1 es V_1 y la partición de la

clase C_2 es V_2 , y la superficie de decisión es l , entonces es tres aspectos han de cumplirse, como en en listado a continuación.

- (1) Cualquier muestra $\mathbf{X} \in V_1$ es etiquetada como $\mathbf{X} \in C_1$.
- (2) Cualquier muestra $\mathbf{X} \in V_2$ es etiquetada como $\mathbf{X} \in C_2$.
- (3) Cualquier muestra $\mathbf{X} \in l$ no pertenece a ninguna de las dos clases, es *el caso indeterminado*.

La Figura 4.19 presenta los tres casos típicos de separabilidad² en dos dimensiones.

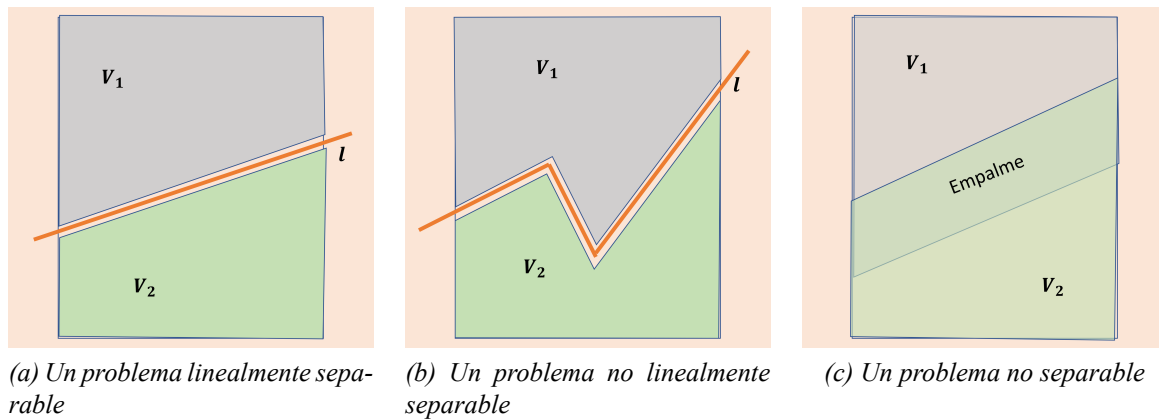


Figura 4.19: Tres casos de problemas de clasificación binaria, en $\mathbb{R}^2$

En la Figura 4.19a un caso de un problema de clasificación binaria es presentado. En este problema la superficie de decisión l se establece por completo mediante un solo hiperplano o línea recta en dos dimensiones. A este tipo de problemas en donde la superficie de decisión se caracteriza por completo por un solo hiperplano se les conoce como *problemas linealmente separables*, pues la superficie de decisión es una hipersuperficie lineal. En la Figura 4.19b se muestra otro caso de superficie de decisión, pero en este caso es una hipersuperficie compleja, pues tiene un patrón en zig-zag. Este es un caso de un *problema no linealmente separable* es decir, las clases aún se pueden separar mediante una hipersuperficie sólo que esta no es un único hiperplano. En la Figura 4.19c se presenta el caso de un *problema no separable*. Esto significa que no es posible establecer una hipersuperficie de decisión l de tal modo que todo a un lado pertenezca a una clase y del otro lado a la otra clase. En este caso de *clases empalmadas o traslapadas* al menos en alguna parte del problema, no es posible detectar si una muestra pertenece a una u otra clase. Este es el peor caso que un clasificador pueda enfrentar.

²Por separabilidad entiéndase la necesaria existencia de una superficie de decisión entre las clases, de tal modo que ambas clases queden separadas por la hipersuperficie

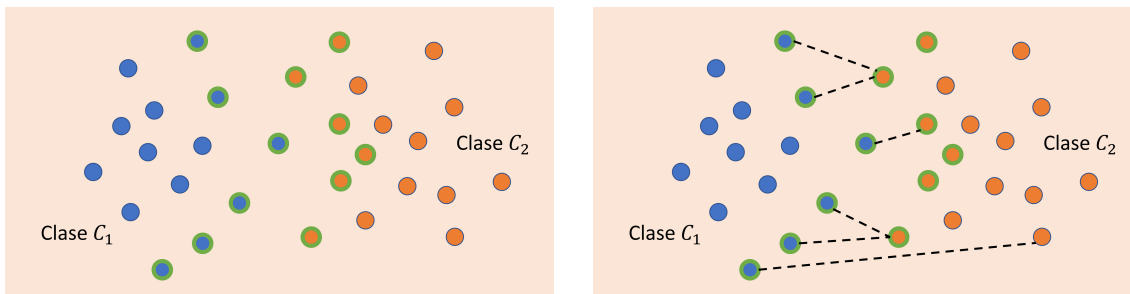
4.3.2. El promedio entre clases como superficie de decisión

Para determinar la superficie de decisión entre las dos clases se propone aproximar *la superficie de decisión usando el promedio de los bordes fundamentales de ambas clases*. Es común que una clase puede mostrar más bordes fundamentales que la otra, simplemente por que puede ser mas extensa o porque quizá, posea muchas más muestras que la otra. Lo que se hace en realidad no es promediar los bordes fundamentales de cada clase, sino sólo aquellos que son *correspondientes*, es decir *bordes fundamentales correspondientes*

Para obtener los bordes fundamentales correspondientes se toman los bordes fundamentales de aquella clase que tenga más bordes fundamentales, a la que se hace llamar la *clase pivotal* $\mathbf{B}_{piv}$. La otra clase se hace llamar la *clase no pivotal* $\mathbf{B}_{no-pivo}$. Así para cada borde fundamental $\mathbf{p} \in \mathbf{B}_{piv}$ asígnese el borde fundamental más cercano de la otra clase, $\mathbf{q} \in \mathbf{B}_{no-pivo}$ como borde correspondiente. Al final se tienen los patrones de bordes fundamentales correspondientes $\mathbf{P}$ y $\mathbf{Q}$, de modo que la superficie de decisión l se puede aproximar como en la Ecuación 4.6.

$$\mathbf{l} = \frac{\mathbf{P} + \mathbf{Q}}{2} \quad (4.6)$$

Obsérvese entonces que la aproximación a la superficie de decisión l es $\mathbf{l} \in \mathbb{R}^{n_E}$, siendo n_E las dimensiones del problema. Las muestras en $\mathbf{l}$ son puntos específicos en el espacio del problema $\mathbb{R}^{n_E}$ que representan una aproximación discreta a lo que idealmente seria la superficie de decisión, la frontera entre clases. La Figura 4.20 muestra una representación gráfica del concepto de bordes correspondientes.



(a) Los bordes fundamentales de dos clases separables

(b) Los pares de bordes correspondientes entre ambas clases

Figura 4.20: Los bordes correspondientes entre los bordes fundamentales, en un problema en $\mathbb{R}^2$

En la Figura 4.20a aparecen los bordes fundamentales de ambas clases. Aquí ambas clases tienen seis bordes fundamentales cada una y la clase pivotal puede ser cualquiera de las dos. En el ejemplo se elige la clase C_1 , cuyos patrones o muestras aparecen en color azul. Ahora se

busca para cada borde fundamental de esta clase, el borde fundamental más cercano de la clase C_2 , la asociación entre cada borde fundamental en C_1 con su correspondiente en C_2 se aprecia mediante líneas punteadas en la Figura 4.20b. Nótese que un mismo borde fundamental en la clase no pivotal puede ser el correspondiente de dos o más bordes fundamentales de la clase pivotal.

El Algoritmo 9 describe este proceso de aproximación a la superficie de decisión basado en los bordes fundamentales correspondientes. El algoritmo recibe como datos de entrada los bordes fundamentales de una y otra clase. Como resultado el algoritmo arroja las muestras del promedio resultante entre los bordes correspondientes. El procedimiento presentado en el Algoritmo 9 a manera de función, porque más adelante éste y los siguientes cuatro procesamientos serán citados de tal manera, especialmente en el algoritmo final para el cálculo de los hiperplanos separadores.

Algoritmo 9: Algoritmo para la aproximación a la superficie de decisión entre clases

```

1 Function promedio( $\mathbf{B}_1, \mathbf{B}_2$ )
2   Hacer  $\mathbf{C}_{piv}$  el conjunto entre  $\mathbf{B}_1$  y  $\mathbf{B}_2$  con la mayor cantidad de muestras;
3   Hacer  $\mathbf{C}_{no-piv}$  el conjunto entre  $\mathbf{B}_1$  y  $\mathbf{B}_2$  con la menor cantidad de muestras;
4   Hacer  $\mathbf{P} = \mathbf{C}_{piv}$ ;
5   Hacer  $\mathbf{Q} = \{\}$ ;
6   for  $\mathbf{p} \in \mathbf{C}_{piv}$  do
7     Hacer  $\mathbf{q} = \arg \min_q d_{min}(\mathbf{p}, \mathbf{q})$ ;
8     Hacer  $\mathbf{Q} = \mathbf{Q} \cup \{\mathbf{q}\}$ ;
9   end
10  Hacer  $\mathbf{AVG} = \frac{\mathbf{P} + \mathbf{Q}}{2}$ ;
11  return  $\mathbf{AVG}$ 

```

El arreglo con las muestras del promedio, en el Algoritmo 9 aparece con el nombre **AVG**, siendo la aproximación a la superficie de decisión. La Figura 4.21 muestra la disposición de las muestras del promedio entre clases.

En la Figura 4.21a aparecen los bordes fundamentales correspondientes asociados mediante una línea punteada. En la Figura 4.21b aparecen las muestras puntuales del promedio entre bordes fundamentales correspondientes. Estas muestras son solamente una aproximación a la superficie de decisión entre las clases. Mientras más muestras se tengan en el promedio entre bordes fundamentales correspondientes la aproximación a la superficie de decisión será mejor. Esto depende de dos aspectos, ambos listados a continuación.

- (1) La cantidad de muestras disponibles para cada clase, en el conjunto de entrenamiento.
- (2) La cantidad de bordes fundamentales en cada clase, en particular, de los de la clase pivotal.

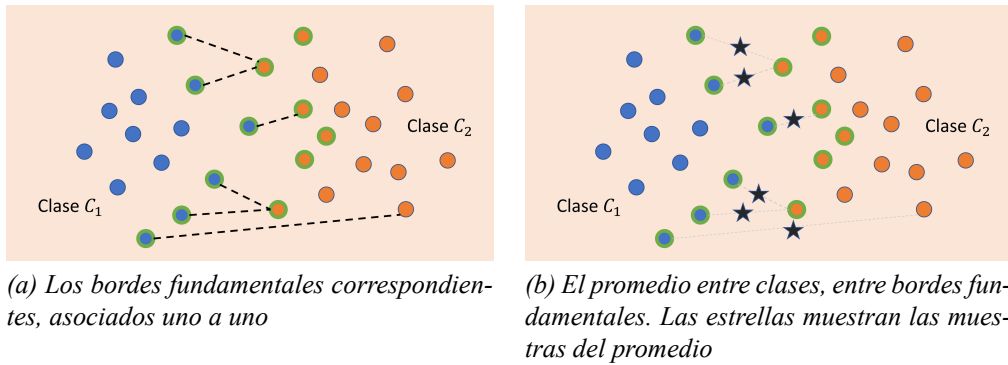


Figura 4.21: El promedio aproximado entre clases, en $\mathbb{R}^2$

4.3.3. Ejemplos sobre el cálculo del promedio entre clases

A continuación se proporcionan algunos ejemplos, en $\mathbb{R}^2$ y $\mathbb{R}^3$, sobre la aproximación a la superficie de decisión que separa dos clases, susando las muestras del promedio entre clases.

4.3.3.1. Ejemplo 4.3.3a

Este primer ejemplo es un problema de clasificación binario en $\mathbb{R}^2$. La clase C_1 contiene 1100 muestras y la clase C_2 contiene 1000 muestras. Las clases tienen una separación clara pero no lineal.

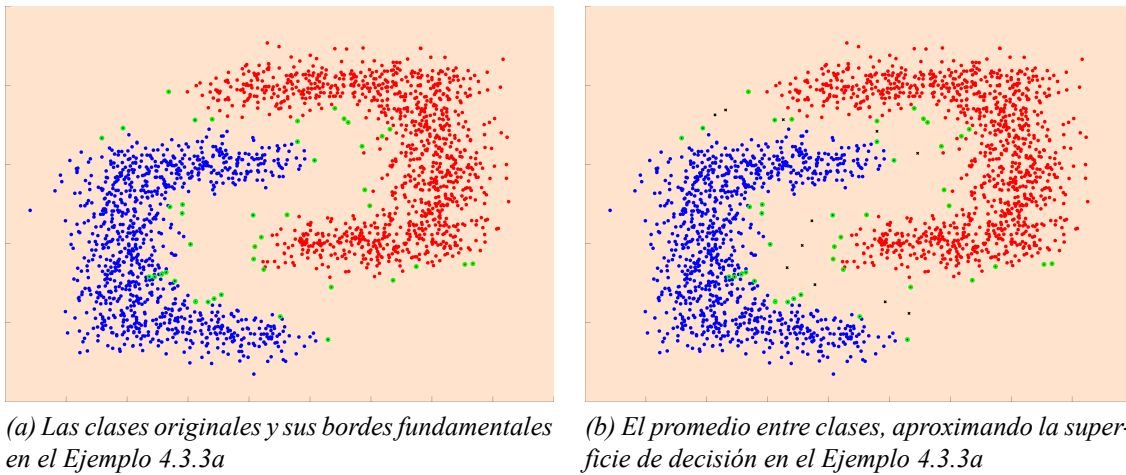


Figura 4.22: El promedio entre clases en $\mathbb{R}^2$ en el Ejemplo 4.3.3a

Se pueden apreciar en la Figura 4.22b las muestras del promedio entre clases como son producidas por el método del promedio recién analizado antes. Las muestras del promedio aparecen en color negro entre las clases del problema.

4.3.3.2. Ejemplo 4.3.3b

En este problema la clase C_1 contiene 600 muestras y la clase C_2 contiene 1200 muestras, así que este es un problema visiblemente desbalanceado. La Figura 4.23 muestra la distribución del promedio entre clases.

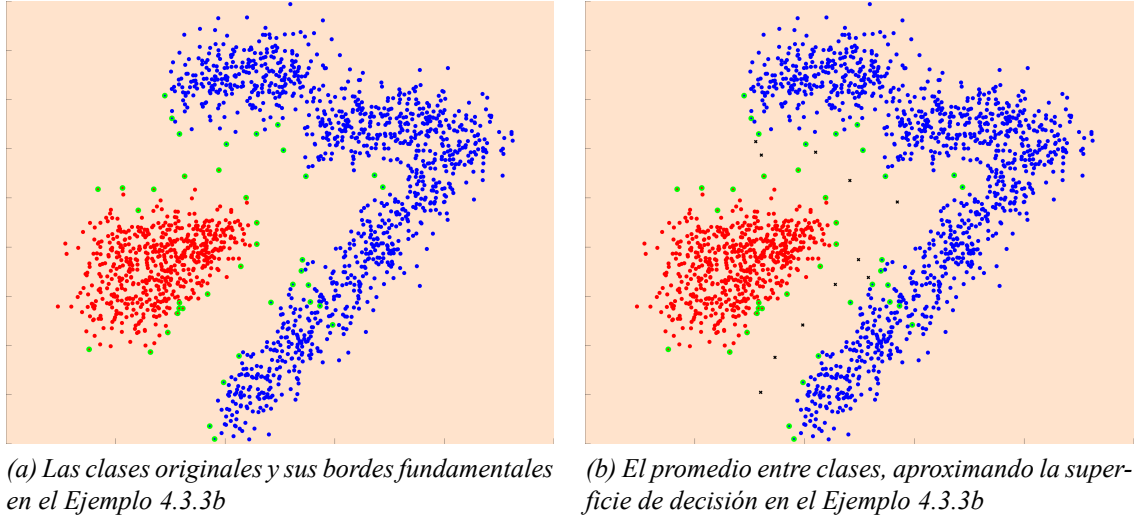


Figura 4.23: El promedio entre clases, en $\mathbb{R}^2$ del Ejemplo 4.3.3b

El promedio entre clases presenta una clara separación entre las dos clases sin un dominio de la clase mayoritaria, la clase C_2 en este ejemplo.

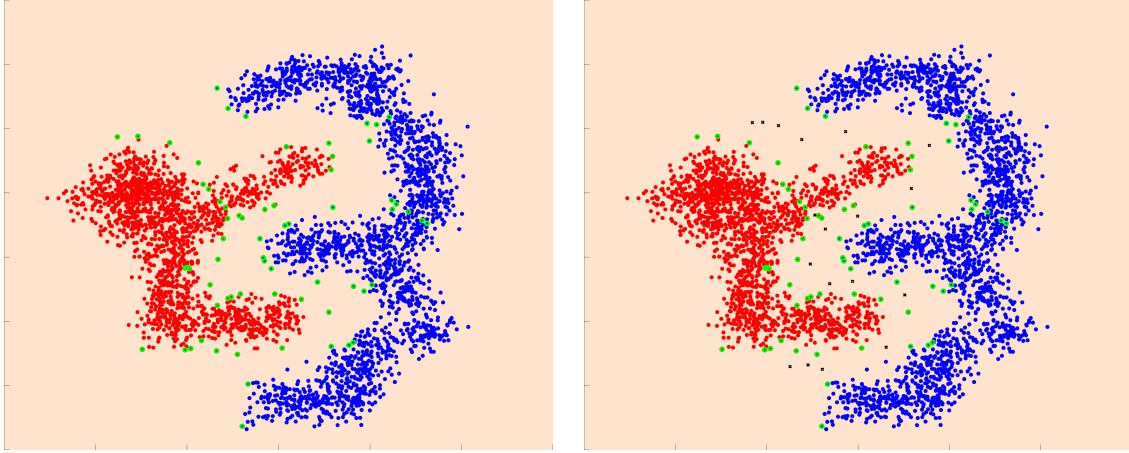
4.3.3.3. Ejemplo 4.3.3c

Ahora la clase C_1 contiene 1700 muestras y la clase C_2 contiene 2200 muestras. Estas clases son separables, pero su separación es de alguna forma complicada. La forma no convexa de las clases hace particularmente difícil localizar una frontera entre las clases. La Figura 4.24 muestra tal distribución de clases y su promedio.

Puede verse en la Figura 4.24b como las muestras del promedio en color negro, se localizan exactamente en medio de ambas clases, la forma tan particular de las clases realmente no afecta a la localización de un promedio.

4.3.3.4. Ejemplo 4.3.3d

La clase C_1 y la clase C_2 se componen de la misma cantidad de muestras, a saber 300. Además, y a diferencia del ejemplo anterior este es un problema relativamente sencillo, las clases son perfectamente separables, como lo muestra la Figura 4.25.



(a) Las clases originales y sus bordes fundamentales en el Ejemplo 4.3.3c

(b) El promedio entre clases, aproximando la superficie de decisión en el Ejemplo 4.3.3c

Figura 4.24: El promedio entre clases en $\mathbb{R}^2$ en el Ejemplo 4.3.3c

El promedio entre clases mostrado con muestras en color negro se localiza correctamente en medio de ambas clases, como lo ilustra la Figura 4.25b. También se puede apreciar como las clases de este problema son en realidad linealmente separables.

4.3.3.5. Ejemplo 4.3.3e

Este es un problema de clasificación binaria en $\mathbb{R}^3$. La clase C_1 contiene 1200 muestras y la clase C_2 contiene 1400 muestras. Las clases son separables aunque no linealmente. La Figura 4.26 muestra la disposición de ambas clases así como las muestras del promedio entre clases.

La Figura 4.26b muestra la distribución de las muestras del promedio en color negro, justo en medio de ambas clases.

4.4. Ajustando el promedio entre clases mediante hiperplanos

Es fácil observar que la aproximación mediante el promedio entre clases es una serie de puntos o muestras en el espacio del problema $\mathbb{R}^{n_E}$ y mostrado en la Figura 4.27b, mediante estrellas, es una simplificación discreta de lo que idealmente es la verdadera superficie de decisión de naturaleza continua, misma que aparece como una línea continua punteada en la Figura 4.27b superpuesta a las muestras del promedio entre clases. La regla de los hiperplanos consiste en establecer el tamaño de la capa oculta como equivalente a la mínima cantidad de hiperplanos que separan las clases involucradas en el problema.

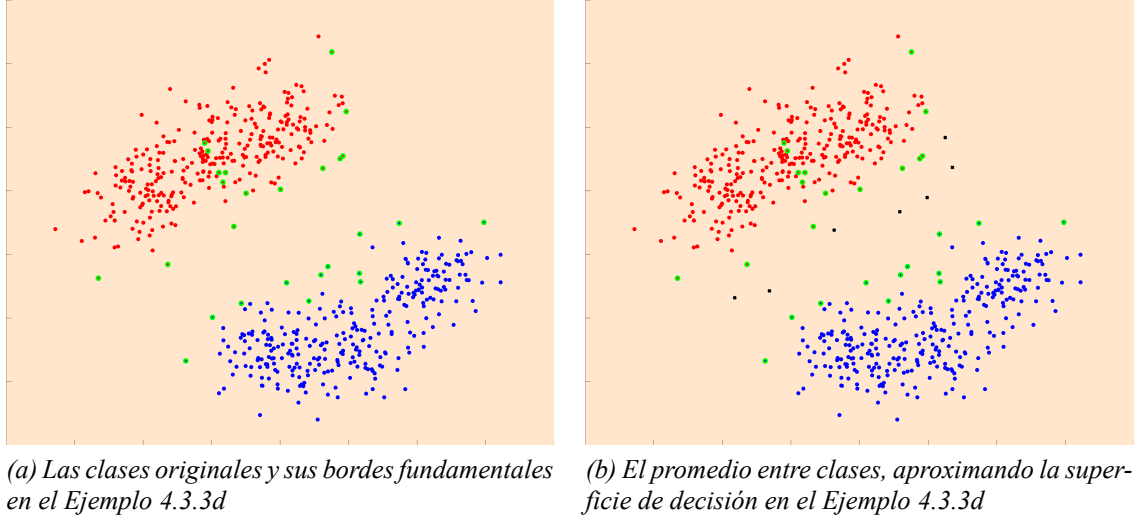


Figura 4.25: El promedio entre clases en $\mathbb{R}^2$ en el ejemplo 4.3.3d

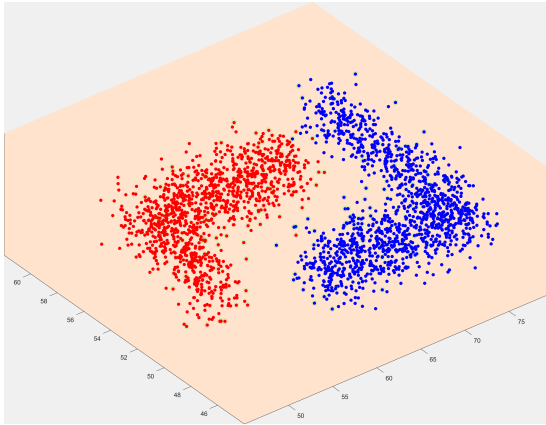
Así pues, se debe encontrar la cantidad de hiperplanos que se ajusten mejor al total de las muestras en el promedio entre clases **AVG** sin tocar las clases. Estos puntos o muestras en el promedio no están necesariamente ordenados, lo que complica el proceso de ajuste a los puntos del promedio. Lo requerido ahora es un algoritmo para ajustar una cantidad de hiperplanos a un conjunto finito de muestras en el espacio general $\mathbf{R}^{n_E}$. En este algoritmo se asumen tres importantes detalles, listados a continuación.

- (1) Los puntos a ajustar son $\mathbf{AVG} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_{n_{avg}}\}$, no están ordenados, aquí n_{avg} es el total de muestras o puntos en el arreglo del promedio entre clases.
- (2) El ajuste usa como *criterio de ajuste* un valor de tolerancia $tol \in \mathbb{R}$.
- (3) El ajuste se basa en un *algoritmo de fuerza bruta*, que de manera iterativa construye un hiperplano, repasa la colección entera de muestras en el promedio eligiendo aquellos que se ajusten al hiperplano actual con un mínimo de tolerancia, y luego eliminándolos de la colección hasta que no quede ningún punto muestra del promedio.

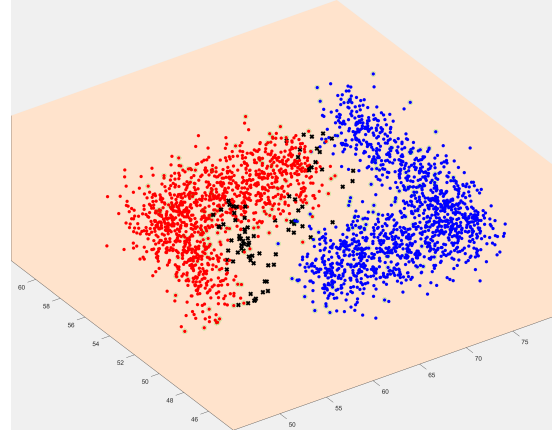
La tolerancia $tole$ debe ser lo más grande posible sin ocasionar colisiones del hiperplano con las clases, tratando de lograr *la cantidad* de hiperplanos ajustados. La tolerancia $tole$ es la distancia máxima perpendicular al hiperplano $\mathbf{l}$. La Figura 4.28 proporciona en un caso bidimensional una interpretación gráfica de la idea de la tolerancia $tole$ y el ajuste de un conjunto de muestras o puntos, al hiperplano $\mathbf{l}$.

En la Figura 4.28 las muestras que se consideran ajustadas al hiperplano $\mathbf{l}$ aparecen en color verde.

IV. Desarrollo

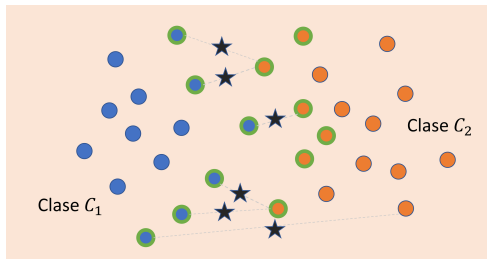


(a) Las clases originales y sus bordes fundamentales en el Ejemplo 4.3.3e

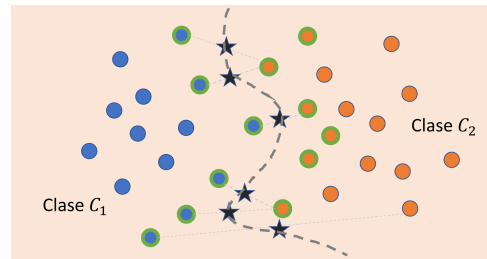


(b) El promedio entre clases, aproximando la superficie de decisión en el ejemplo 4.3.3e

Figura 4.26: El promedio entre clases en $\mathbb{R}^3$ en el Ejemplo 4.3.3e



(a) La aproximación discreta a la frontera entre clases



(b) El caso muy ideal de la verdadera frontera entre clases, la superficie de decisión

Figura 4.27: Comparando a la verdadera, y muy ideal superficie de decisión y la aproximación mediante el promedio en $\mathbb{R}^2$

El Algoritmo 10 describe el proceso de ajustar un conjunto de muestras en el arreglo o colección **AVG**, a una cierta cantidad de hiperplanos cubiertos con una tolerancia *tole*.

En el Algoritmo 10 la operación **Pts/P_{adj}** significa complemento del conjunto **Pts** sobre el conjunto **P_{adj}**. El resultado del algoritmo es la cantidad de hiperplanos n_H que se ajustan a la colección de muestras en el promedio entre clases.

El efecto de este procedimiento se puede ver en la Figura 4.29 demostrando el ajuste de cuatro hiperplanos con las muestras del promedio.

El caso mostrado en la Figura 4.29b es muy ideal, donde la tolerancia *tole* dio lugar al ajuste que allí se muestra. Dependiendo del valor de la tolerancia el ajuste puede quedar visiblemente diferente. En el ejemplo han sido necesario cuatro hiperplanos para separar las dos clases, estos son $\mathbf{l}_1$, $\mathbf{l}_2$, $\mathbf{l}_3$ y $\mathbf{l}_4$. Entonces, la capa oculta de la red SLFN que separe este problema tendrá cuatro

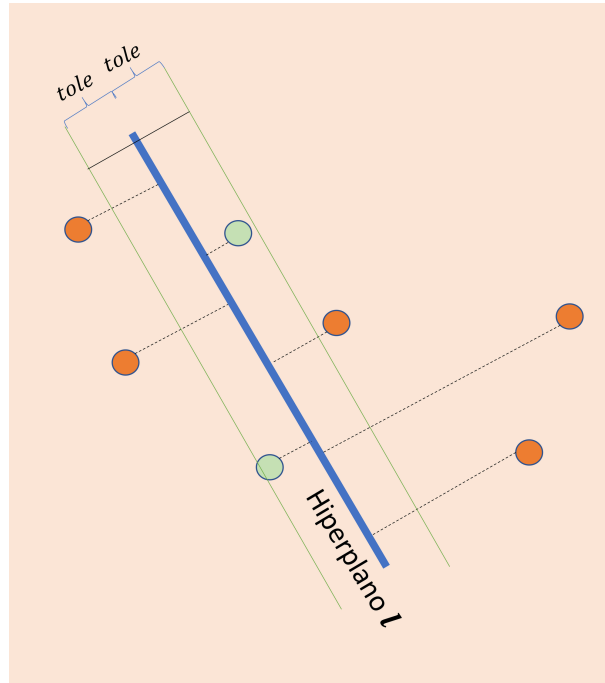


Figura 4.28: La tolerancia *tole* para ajustar un grupo de muestras a un hiperplano **L**

neuronas ocultas, es decir $n_O = n_H = 4$.

La tolerancia *tole* debería ajustarse a las condiciones locales de las clases que se pretende separar. Por ejemplo, considérense los tres criterios listados a continuación.

- (1) La separación que las clases tienen debe ser el criterio a considerar, para la tolerancia *tole*.
- (2) Entre más separadas las clases se encuentren, más grande debe ser la tolerancia *tole*.
- (3) Entre menos separadas, menor la tolerancia *tole*.

Es de esperar que dos clases muy distantes entre sí requieran mucho menos hiperplanos para separarse y dos clases muy próximas entre sí requieren muchos más hiperplanos para separarse.

Denotando la *mínima distancia entre clases* como d_{min} , la Ecuación 4.7 establece como la tolerancia *tole* se calcula a partir de la mínima distancia entre clases:

$$tole = \frac{d_{min}}{2} \quad (4.7)$$

Considerando los pares de bordes correspondientes **P** y **Q**, la *mínima distancia entre las clases* la proporcionan aquel par de bordes correspondientes cuya distancia es la menor, como lo ilustra la Figura 4.30 y se expresa como en la Ecuación 4.8.

Algoritmo 10: Algoritmo para el ajuste de una cantidad de hiperplanos a un conjunto de muestras

```

1 Function ajuste(AVG = {p1, p2, . . . , pnavg}, tole)
2   Hacer Pts = AVG;
3   Hacer listo=False;
4   Hacer nH = 0;
5   while listo == False do
6     if |Pts| < nE then
7       | listo = True;
8     else
9       Hacer Base conteniendo nH puntos de Pts, preferentemente cercanos;
10      Hacer Resto = Pts/Base;
11      Hacer lBase el hiperplano formado con las muestras en Base;
12      Hacer Padj = {};
13      for p ∈ Resto do
14        | ;
15      end
16      if d(lBase, p) ≤ tole then
17        | Hacer Padj = Padj ∪ {p};
18      end
19      if |Padj| > 0 then
20        | nH = nH + 1;
21        | Hacer Pts = Pts/Padj;
22      end
23    end
24  end
25  return nH

```

$$d_{min} = \min_{\forall \mathbf{p} \in \mathbf{P}, \forall \mathbf{q} \in \mathbf{Q}} d(\mathbf{p}, \mathbf{q}) \quad (4.8)$$

4.4.1. Ejemplos sobre el ajuste con hiperplanos

A continuación se presentan los cinco ejemplos de las dos últimas secciones, ajustando a las muestras del promedio, calculado en la sección anterior, con una cantidad de hiperplanos específica. Son, como ya se ha visto, ejemplos de problemas de clasificación binarios, en $\mathbb{R}^2$ y $\mathbb{R}^3$, exclusivamente.

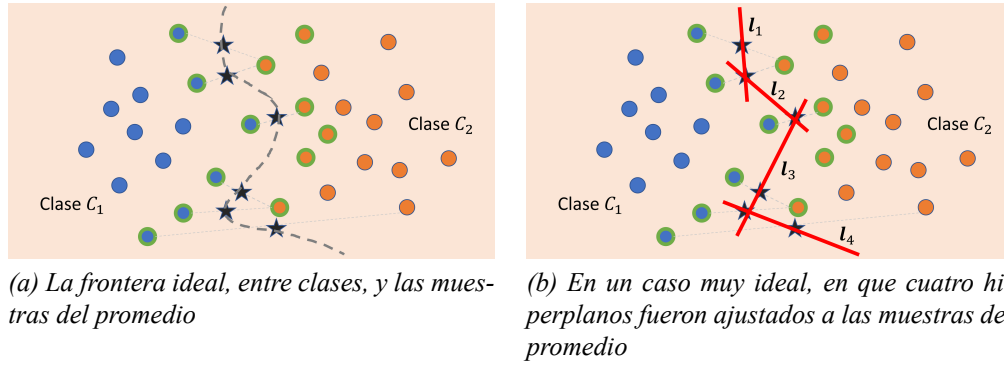


Figura 4.29: El ajuste mediante hiperplanos, cuatro de ellos, a las muestras del promedio y la superficie ideal en $\mathbb{R}^2$

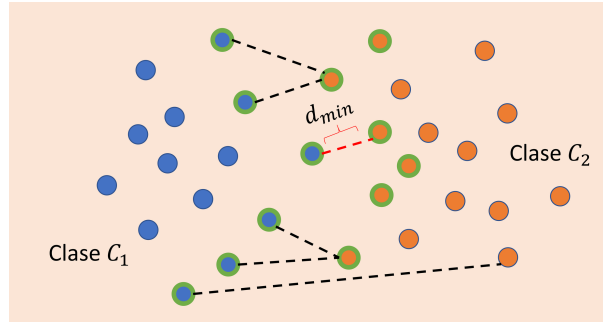


Figura 4.30: La mínima distancia entre clases, la distancia entre el par de bordes correspondientes más cercanos, en un caso en $\mathbb{R}^2$

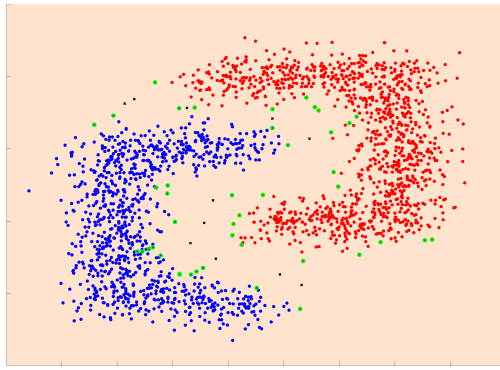
4.4.1.1. Ejemplo 4.4.1a

Este problema implica dos clases en $\mathbb{R}^2$. La clase C_1 contiene $n_{p_1} = 1100$ muestras y la clase C_2 contiene $n_{p_2} = 1000$ muestras. El ajuste, según lo establecido en la presente sección produce un ajuste a las muestras del promedio entre clases, con $n_H = 7$ hiperplanos, o líneas rectas en este caso.

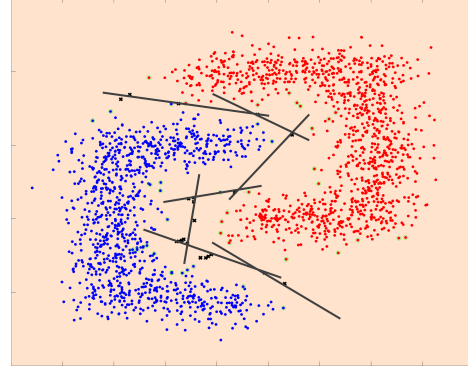
De acuerdo con el método propuesto las dos categorías se separan usando siete hiperplanos $n_H = 7$. La disposición de estos siete hiperplanos se aprecian en la Figura 4.31b. En este problema $tole = 0,5$ dado que $d_{min} = 1,0027$.

4.4.1.2. Ejemplo 4.4.1b

En este caso la clase C_1 consta de 600 muestras y la clase C_2 de 1200 muestras, así que es un problema de clasificación bastante desbalanceado. La Figura 4.32 proporciona una interpretación gráfica del problema.

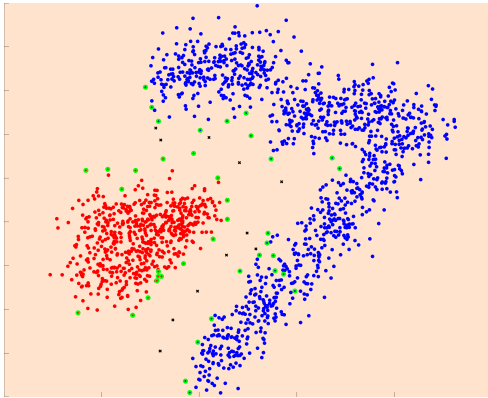


(a) Las dos clases y su promedio en el Ejemplo 4.4.1a

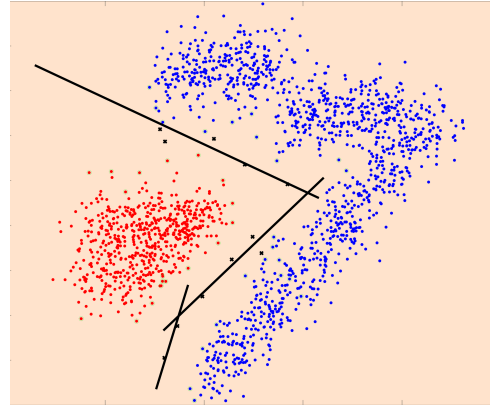


(b) El ajuste con $c_H = 7$ hiperplanos en el Ejemplo 4.4.1a

Figura 4.31: El ajuste con $n_H = 7$ hiperplanos en el Ejemplo 4.4.1a



(a) Las dos clases y su promedio en el Ejemplo 4.4.1b



(b) El ajuste con $c_H = 3$ hiperplanos en el Ejemplo 4.4.1b

Figura 4.32: El ajuste con $n_H = 3$ hiperplanos en el Ejemplo 4.4.1b

En la Figura 4.32b es claro como un total de $n_H = 3$ hiperplanos separan las dos clases en juego. En este caso la distancia entre clases $d_{min} = 1,11$, por lo que la tolerancia $tole = 0,55$. Un resultado muy interesante de este ejemplo es que siendo este un problema de clasificación muy desbalanceado, el método no se ve afectado por la abrumadora mayoría de las muestras en la clase mayoritaria, la clase C_2 .

4.4.1.3. Ejemplo 4.4.1c

En este ejemplo la clase C_1 contiene 1700 muestras y la clase C_2 contiene 2200 muestras. Las clases, como lo muestra la Figura 4.33a son separables, aunque su superficie de decisión es bastante compleja.

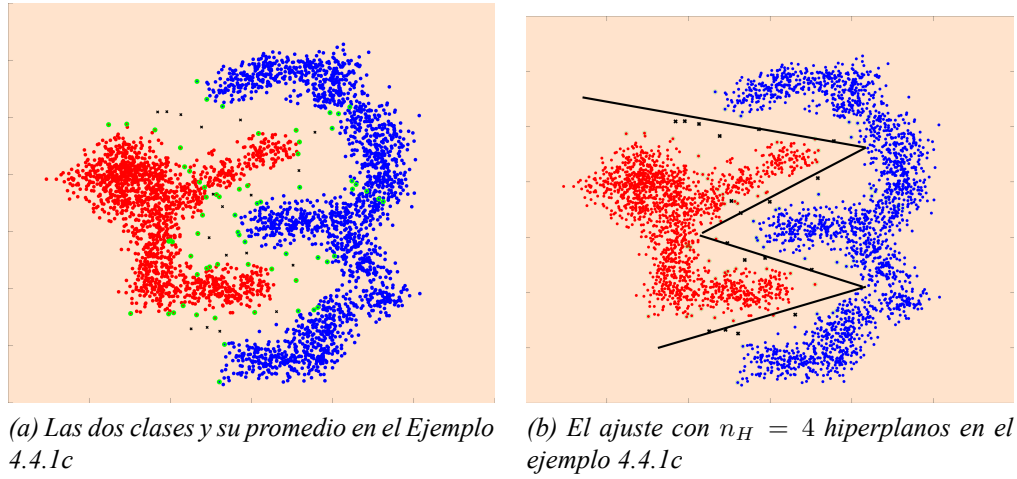


Figura 4.33: El ajuste con $n_H = 4$ hiperplanos en el Ejemplo 4.4.1c

En la Figura 4.33b se muestran cuatro hiperplanos separando perfectamente esas dos categorías.

4.4.1.4. Ejemplo 4.4.1d

Este es el ejemplo más sencillo de manipular, desde el punto de vista de un clasificador. Es un problema de clasificación binaria en $\mathbb{R}^2$. la clase C_1 contiene 300 muestras y la clase C_2 contiene 300 muestras. Las dos clases son linealmente separables, como la Figura 4.34 muestra.

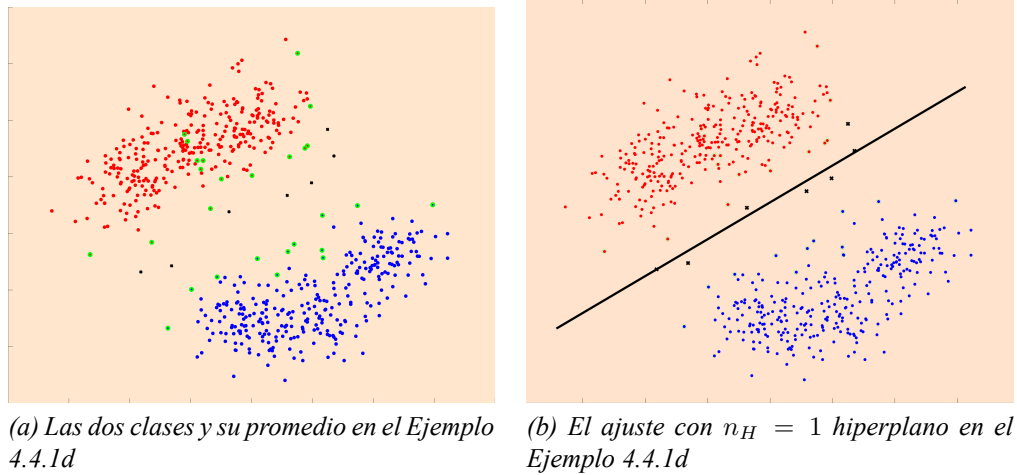


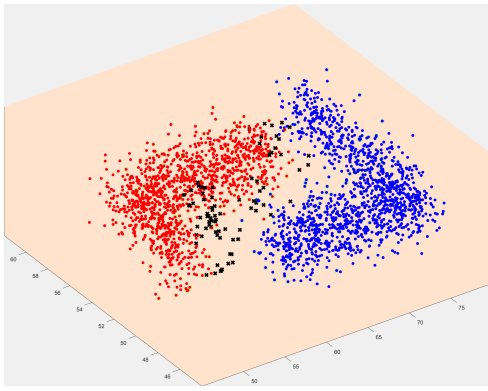
Figura 4.34: El ajuste con $n_H = 1$ hiperplano en el Ejemplo 4.4.1d

Obsérvese como el método propuesto, efectivamente, es capaz de ajustar un solo hiperplano,

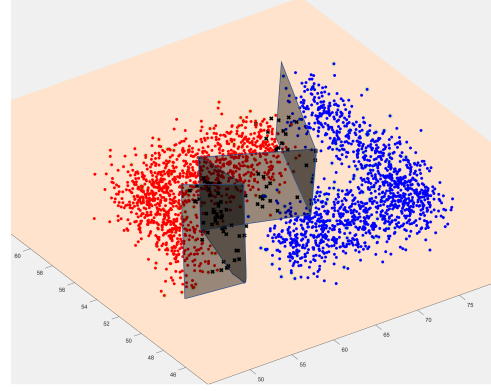
es decir, $n_H = 1$, en este caso, una sola línea recta, al conjunto de muestras del promedio entre clases, mostradas como cruces en color negro en la Figura 4.34a, puesto que el problema es linealmente separable. El ajuste de este único hiperplano se muestra en la Figura 4.34b. En este problema ocurrió que $d_{min} = 1,6$, por lo que $tole = 0,8$.

4.4.1.5. Ejemplo 4.4.1e

Este es un problema de clasificación en $\mathbb{R}^3$. Las clases son C_1 con 1200 muestras y la clase C_2 con 1400. Las clases son separables pero no linealmente, como la Figura 4.35a muestra.



(a) Las dos clases y su promedio en el Ejemplo 4.4.1e



(b) El ajuste con $n_H = 4$ hiperplano en el Ejemplo 4.4.1e

Figura 4.35: El ajuste con $n_H = 4$ hiperplano en el Ejemplo 4.4.1e

La Figura 4.35b muestra en una perspectiva tridimensional, la disposición de $n_H = 4$ hiperplanos que separan a las dos clases. Estos cuatro hiperplanos son el resultado de una distancia $d_{min} = 2,74$ entre clases, por lo que $tole = 1,37$.

4.5. Algoritmo completo con el método propuesto para la obtención del tamaño de la capa oculta

En el Algoritmo 11 se muestra el método propuesto, desde separar las muestras en el conjunto de entrenamiento en conjuntos separados para cada clase, hasta describir cada conjunto de manera geométrica, caracterizarlos a modo de poliedro y localizar sus bordes fundamentales, aproximar una frontera entre conjuntos para cada clase y ajustar dicho promedio usando uno o más hiperplanos.

Algoritmo 11: Algoritmo para el cálculo del tamaño de la capa oculta

```

1 Function damenO( $C_1, C_2$ )
2   Hacer  $a1 = aShape(C_1)$ ;
3   Hacer  $a2 = aShape(C_2)$ ;
4   Hacer  $[p, q] = fundamentales(a1, a2)$ ;
5   Hacer  $d_{min}$  = mínima distancia entre  $p$  y  $q$ ;
6   Hacer  $avg = promedio(p, q)$ ;
7   Hacer  $tole = \frac{d_{min}}{2}$ ;
8   Hacer  $n_H = ajuste(avg, tole)$ ;
9   Hacer  $n_O = n_H$ ;
10  return  $n_O$ 

```

De esta manera, las correspondientes redes SLFN para cada uno de los ejemplos en la sección anterior se muestran en las siguientes figuras. El Ejemplo 4.4.1a de la última subsección y mostrado en la Figura 4.36a resulta en una red SLFN como la mostrada en la Figura 4.36b.

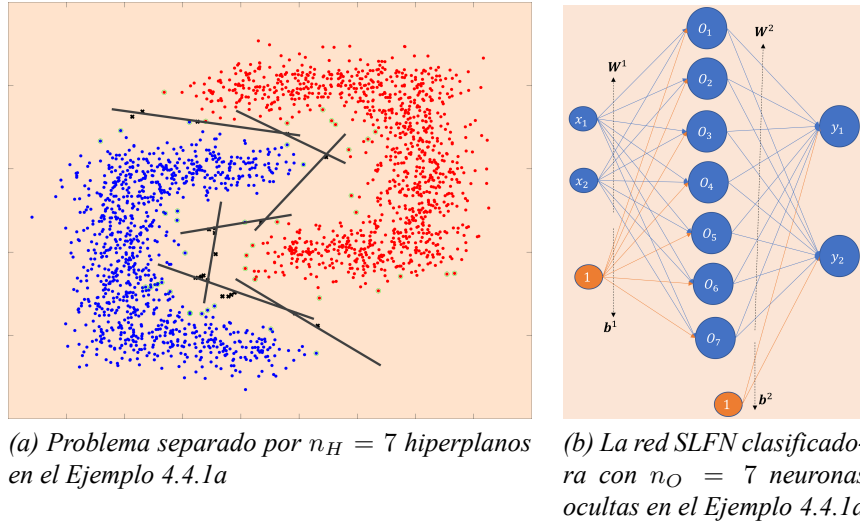


Figura 4.36: La red SLFN clasificadora para el problema en el ejemplo 4.4.1a

La red SLFN para el ejemplo 4.4.1b en la última subsección es mostrada en la Figura 4.37, mientras que en la Figura 4.37b se presenta la red SLFN con $n_O = 3$.

Por otro lado, para el ejemplo 4.4.1c de la última subsección de la sección anterior, ilustrado en la Figura 4.38a con $n_H = 4$, su red SLFN correspondiente se muestra en la Figura 4.38b con $n_O = 4$.

Para el ejemplo 4.4.1d que es linealmente separable y que se muestra en la Figura 4.39a con $n_H = 1$, la red perceptrón que clasifica este problema se muestra en la Figura 4.39b. Puesto que

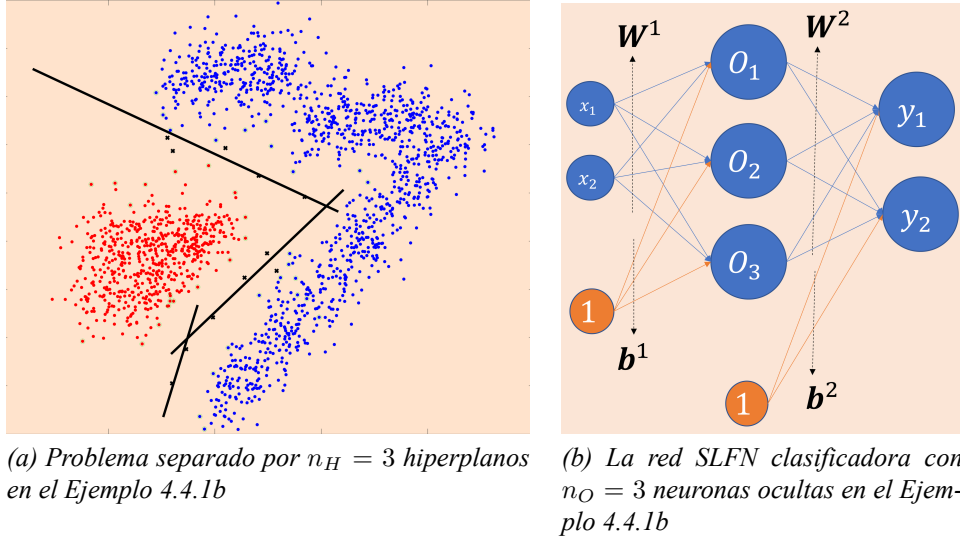


Figura 4.37: La red SLFN clasificadora para el problema en el Ejemplo 4.4.1b

el método de entrenamiento de un perceptrón suele ser más simple y rápido [1], se deduce que un perceptrón es bastante más efectivo con problemas linealmente separables. Así que para un mejor desempeño, si un problema de clasificación binaria es linealmente separable, es mejor una red perceptrón que una red SLFN con una sola neurona oculta.

Finalmente, para el ejemplo 4.4.1e, el problema requiere de $n_H = 4$ hiperplanos para separar ambas clases, como lo muestra la Figura 4.40a, por lo que la red SLFN clasificadora mostrada en la Figura 4.40b tiene $n_O = 4$ neuronas ocultas.

4.6. El caso multiclase

Los esquemas de clasificadores multiclase ya fueron debidamente detallados en el capítulo anterior. Aquí se analiza como pasar los esquemas de clasificación multiclase a un esquema de redes MLP clasificadoras, en particular en lo referente a establecer el tamaño de la capa oculta de la red usando para ello la regla de los hiperplanos. Hasta ahora, sólo ha sido desarrollada una metodología para encontrar n_H a partir de la aproximación a la frontera entre dos clases. Las siguientes subsecciones tratan con detalle como extenderlo a problemas multiclase, específicamente para un esquema OaO.

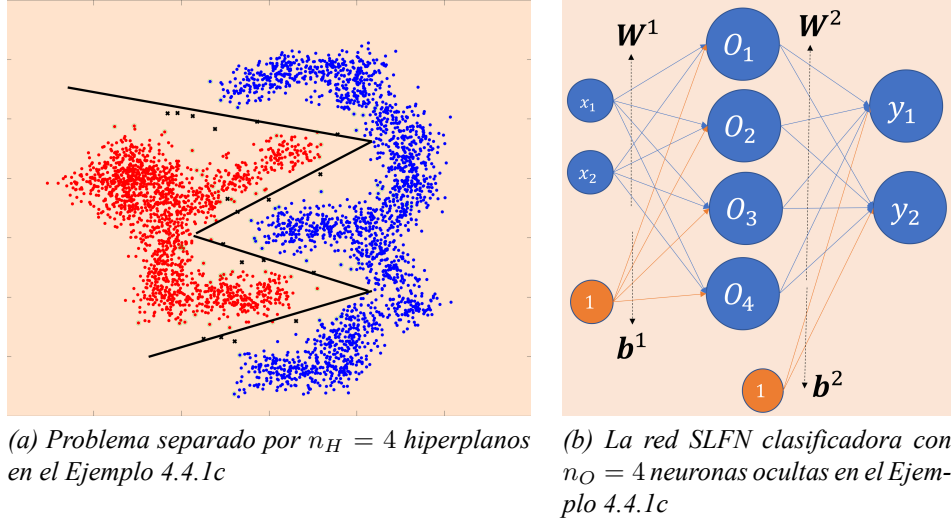


Figura 4.38: La red SLFN clasificadora para el problema en el Ejemplo 4.4.1c

4.6.1. El esquema OaO en una red SLFN

El esquema OaO es muy útil porque permite acomodar un problema multiclase como si este fuera una colección finita de problemas de clasificación binarios. Normalmente OaO luego de clasificar una nueva muestra $\mathbf{X} \in \mathbb{R}^{n_E}$ a una clase específica, por cada uno de los clasificadores separados, entra una segunda etapa de votación que decide mediante un proceso de votación entre los clasificadores separados a que clase asignará finalmente ese nuevo patrón, que sería la clase más votada. Utilizando hiperplanos para definir la cantidad de neuronas ocultas, en el caso del clasificador multiclase basado en redes SLFN, la primer fase consiste en decidir con cuantos hiperplanos se separa cada par de clases, por ejemplo $n_{H_{ij}}$ para las clases C_i y C_j , de manera que la segunda etapa se encarga ahora simplemente de acumular la cantidad de hiperplanos que separan cada par de clases en una sola cantidad de hiperplanos separadores, así pues, la Ecuación 4.9 expresa, para esta situación, el cálculo de la cantidad n_H .

$$n_H = \sum_{i=1}^{n_C} \sum_{j < i} n_{H_{ij}} \quad (4.9)$$

La Figura 4.41 muestra una representación gráfica del proceso OaO adaptado a la regla de los hiperplanos como recién fue explicado. La idea fundamental detrás de esta adaptación es que la cantidad de hiperplanos que separan a las n_C clases, no es más que la agregación de los hiperplanos que separan a cada par de clases separables. En forma geométrica esta misma adaptación se muestra en la Figura 4.42, para un problema sencillo que ilustra la idea de esta

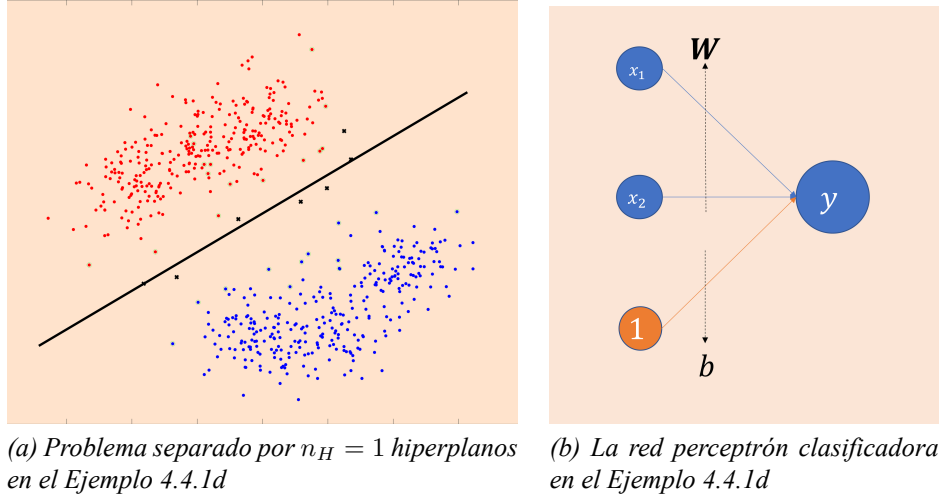


Figura 4.39: La red SLFN clasificadora para el problema en el Ejemplo 4.4.1d

adaptación del esquema multiclase OaO.

En la Figura 4.42a aparece el problema original con $n_C = 3$, con las clases C_1 en color azul, la clase C_2 en color anaranjado y la clase C_3 en color verde. En la Figura 4.42b aparecen las formas alfa aproximadas de cada una de las clases. En la Figura 4.42c aparece la separación de las clases C_2 y C_3 , en este caso un solo hiperplano separa estas dos clases, $n_{H_{23}} = 1$. En la Figura 4.42d aparece la separación entre las clases C_1 y C_3 , en este caso nuevamente un solo hiperplano separa ambas clases, $n_{H_{13}} = 1$. En la Figura 4.42e aparece la separación entre las clases C_1 y C_2 , ocurre que una vez más ambas clases se separan con un solo hiperplano $n_{H_{12}} = 1$. En la Figura 4.42f aparece la acumulación de los tres resultados anteriores, es decir que $n_H = n_{H_{12}} + n_{H_{13}} + n_{H_{23}} = 1 + 1 + 1 = 3$.

4.6.2. Ejemplos del uso del esquema OaO en problemas multiclase

En esta subsección se presentan una serie de ejemplos del uso del esquema OaO recién discutido, considerando la regla de los hiperplanos, en problemas de clasificación multiclase en $\mathbb{R}^2$ y $\mathbb{R}^3$.

4.6.2.1. Ejemplo 4.6.2a

Este primer ejemplos implica tres clases, C_1 , C_2 y C_3 en un espacio $\mathbb{R}^2$. Cada una de las clases consta de 700 muestras. Su disposición ese muestra en la Figura 4.43a.

Las clases en este problema se separan de la siguiente manera: las clases C_1 y C_2 por $n_{H_{12}} = 3$

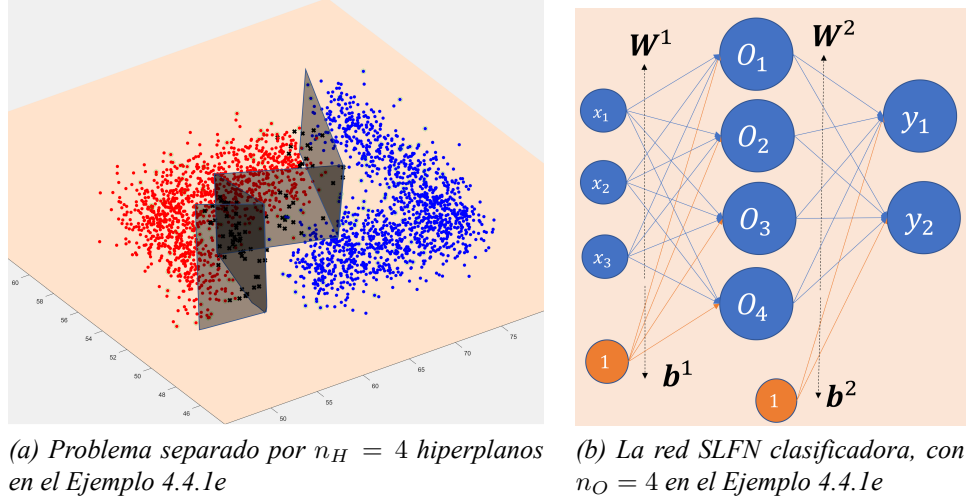


Figura 4.40: La red SLFN clasificadora para el problema en el Ejemplo 4.4.1e

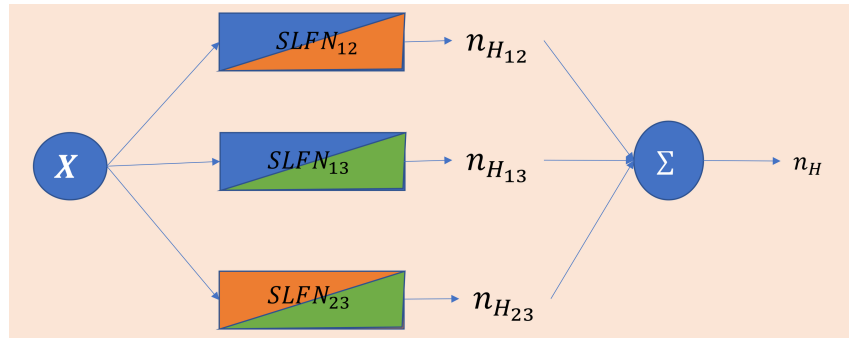


Figura 4.41: la regla de los hiperplanos adaptada al esquema multiclase OaO

hiperplanos, las clases C_1 y C_3 por $n_{H_{13}} = 2$ hiperplanos, mientras que las clases C_2 y C_3 lo hacen con $n_{H_{23}} = 1$ hiperplanos, es decir C_2 y C_3 son linealmente separables. La separación de estas clases según lo comentado, se ilustra en la Figura 4.43b.

Aplicando el esquema OaO la cantidad de hiperplanos necesarios para separar estas categorías requiere un total de $n_H = n_{H_{12}} + n_{H_{13}} + n_{H_{23}} = 3 + 2 + 1 = 6$ hiperplanos. La superficie de decisión que separa las clases C_1 y C_2 , denotada l_{12} se aproxima usando $n_{H_{12}} = 3$ hiperplanos, que en la Figura 4.43b se muestran en color azul. La superficie de decisión que separa las clases C_1 y C_3 , denotada l_{13} , se aproxima usando $n_{H_{13}} = 2$ hiperplanos que en la misma figura aparecen en color cian. La superficie de decisión que separa a las clases C_2 y C_3 , denotada l_{23} , se aproxima usando sólo un hiperplano, $n_{H_{23}} = 1$, en la misma figura esta superficie de decisión aparece en color morado. De lo anterior usando la regla de los hiperplanos, la red SLFN que separe estas tres clases se compone de $n_O = n_H = 6$ neuronas ocultas y tres neuronas de salida,

IV. Desarrollo

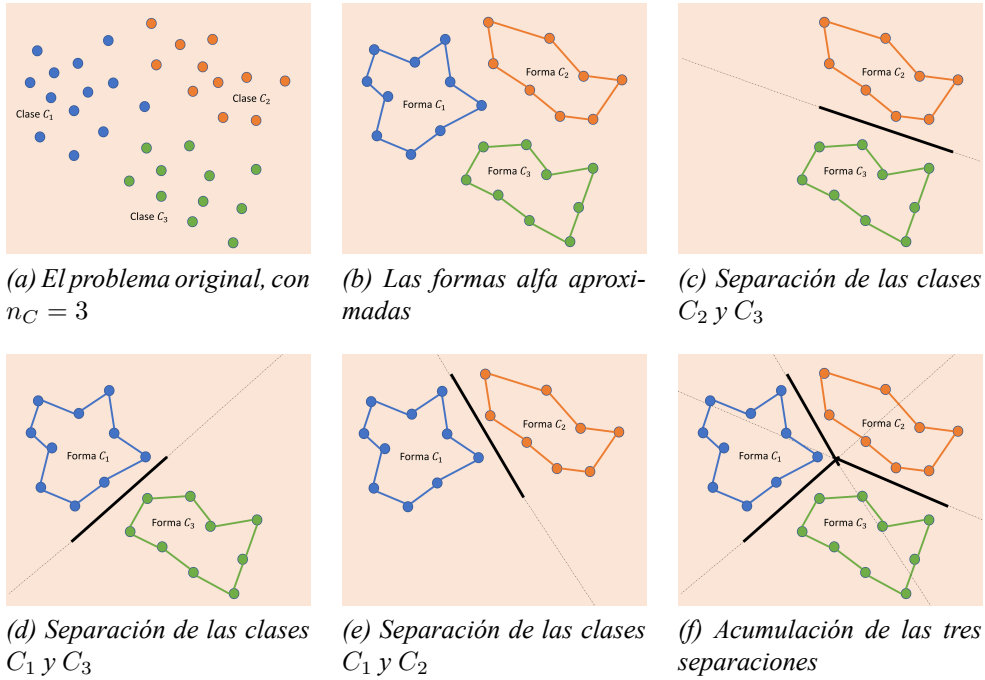


Figura 4.42: El esquema OaO adaptado a la regla de los hiperplanos, en un problema con $n_C = 3$ en $\mathbb{R}^2$

como lo muestra la Figura 4.44.

4.6.2.2. Ejemplo 4.6.2b

Este es un problema de clasificación entre $n_C = 3$ clases, en $\mathbb{R}^3$. Las clases en el problema son C_1 con 600 muestras, la clase C_2 con 800 muestras, y la clase C_3 con 1100 muestras. La distribución de estas tres clases se muestra en la Figura 4.45a. Las muestras de la clase C_1 aparecen en color rojo y las muestras en la clase C_2 en color azul, así como las muestras en la clase C_3 en color verde. Es visible como este problema es un problema de clasificación desbalanceado, siendo C_3 la clase mayoritaria y C_1 la clase minoritaria.

Al aplicar el esquema de separación OaO la clase C_1 se separa de la clase C_2 mediante $n_{h_{12}} = 5$ hiperplanos, que aproximan la superficie de decisión l_{12} . Las clases C_1 y C_3 se separan mediante $n_{H_{13}} = 6$ hiperplanos, que aproximan la superficie de decisión l_{13} . Y las clases C_2 y C_3 se separan mediante $n_{H_{23}} = 6$ hiperplanos, que aproximan la superficie de decisión l_{23} , por lo que al final la cantidad de hiperplanos necesarios para separar estas tres clases es $n_H = n_{H_{12}} + n_{H_{13}} + n_{H_{23}} = 5 + 6 + 6 = 17$ hiperplanos. La Figura 4.45b muestra la disposición de los hiperplanos, de acuerdo con la superficie de decisión que aproximan.

Obtener la cantidad de neuronas ocultas para el clasificador SLFN con la regla de los hiper-

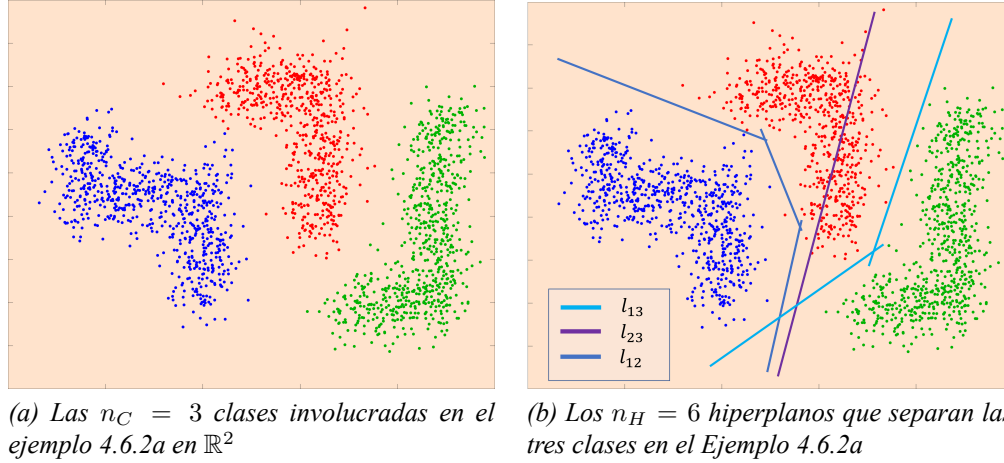


Figura 4.43: El problema de clasificación multiclase en el Ejemplo 4.6.2a. Las clases son C_1 en color rojo, C_2 en color azul, y C_3 en color verde. También, la separación de las clases en $n_H = 5$ hiperplanos

planos no implica ningún problema. Como lo muestra la Figura 4.46, la red SLFN clasificadora utilizaría una capa oculta con un total de $n_o = n_H = 17$ neuronas ocultas, según la regla de los hiperplanos.

Una particularidad interesante de este problema es que las clases si bien perfectamente separables, yacen muy cerca una de otra, por lo que la distancia entre clases es pequeña, y la tolerancia que define el ajuste de hiperplanos a las muestras del promedio entre clases es relativamente grande en este ejemplo.

4.7. El caso de las clases traslapadas

El problema de las clases empalmadas o traslapadas, que es otra nomenclatura para los problemas de clases no separables, es uno de los mayores detractores en cuanto al desempeño de muchos clasificadores [98], más si estos clasificadores dependen de localizar una frontera entre las clases, como las redes MLP [99], las máquinas SVM ³ [100], los clasificadores basados en árboles de decisión [80] al menos porque estos son muy sensibles a falsos positivos y falsos negativos, entre otros.

³Las máquinas SVM, sin embargo, poseen un parámetro llamado *variable slack*, que hasta cierto punto les permite manejar clases traslapadas ignorando aquellas muestras en las zonas de conflicto. Debe tenerse en cuenta que esto solo aplica hasta cierto punto

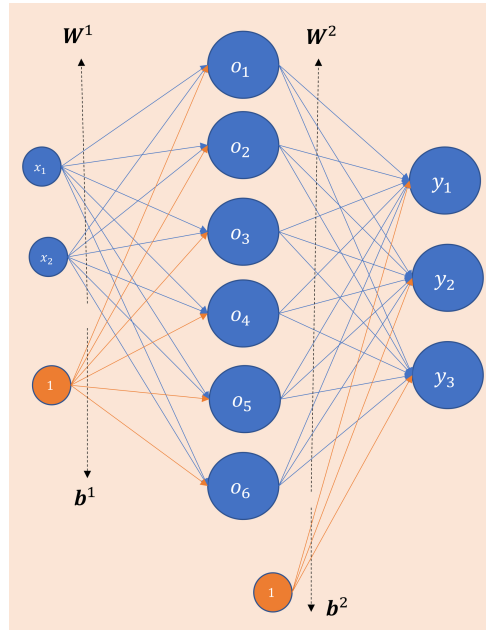


Figura 4.44: La red SLFN que separa las tres clases con $n_O = n_H = 6$ neuronas ocultas, deducidas según el esquema OaO y la regla de los hiperplanos en el ejemplo 4.6.2a

4.7.1. Preprocesando los datos de entrenamiento

La clasificación en problemas con clases no separables o clases traslapadas suele llevarse muy bien con métodos como los clasificadores bayesianos, puesto que estos no dependen de localizar una frontera entre las clases. Si los datos de las clases a separar en un problema de clasificación muestran un mayor o menor grado de empalme, es posible hacer algo para *forzar* a que una separación entre ellos pueda existir, para lo cual esos mismos datos deben ser sometidos a una etapa previa al entrenamiento del clasificador, en este caso, al entrenamiento de la red SLFN clasificadora.

Las principales causas de empalme están ligadas a que, por un lado, la elección de los atributos para describir a los individuos de la población de manera incorrecta o inconveniente. Otra razón común es un mal muestreo de los individuos que forman la población, por ejemplo una mala lectura debido a un mal funcionamiento en un sensor, o en el incorrecto proceso de muestreo.

4.7.1.1. Tres tipos de traslape

Los datos de un problema de clasificación pueden aparecer empalmados de tres maneras diferentes [101]:

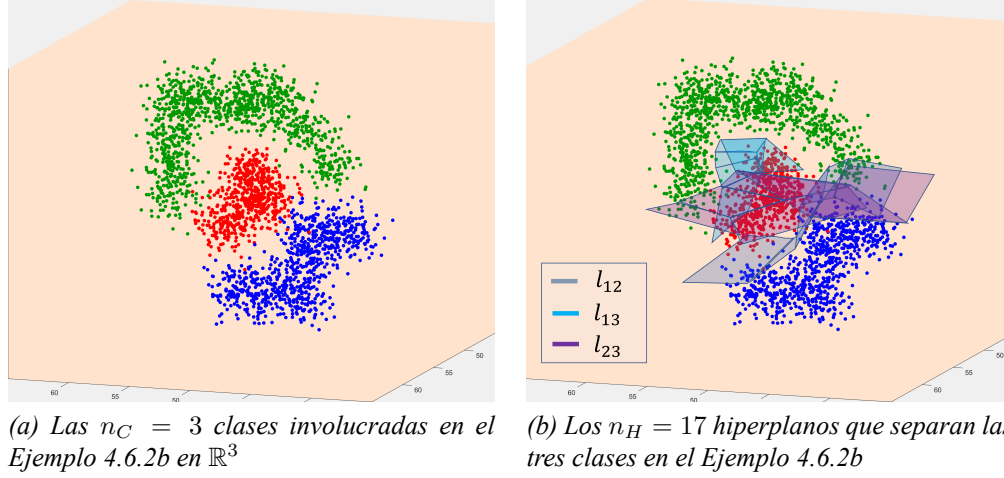


Figura 4.45: El problema de clasificación multiclase en el Ejemplo 4.6.2b. Las clases son C_1 en color rojo, C_2 , en color azul y C_3 en color verde. También, la separación de las clases en $n_H = 17$ hiperplanos

- (1) Ligeramente empalmados.
- (2) Moderadamente empalmados.
- (3) Severamente empalmados.

Véase la Figura 4.47 para una descripción gráfica de cada uno de estos tres casos de clases empalmadas. En la Figura 4.47a se muestra el caso ligeramente empalmado. En la Figura 4.47b el caso moderadamente empalmado. En la Figura 4.47c el caso severamente empalmado. Para cuantificar el empalme en una clase C_i para $i = 1, 2, \dots, n_C$, las muestras de la clase que se localizan en la zona del problema que le pertenece a otra clase, denotado como n_{emp} son contabilizadas, y una razón entre estas y el total de muestras en la clase denotada como n_{P_i} es llevada a cabo, según lo expresa la Ecuación 4.10.

$$r_{emp_i} = \frac{n_{emp}}{n_{P_i}} \quad (4.10)$$

Con lo anterior se puede definir que un problema de clasificación se encuentra ligeramente empalmado si $\max_{1 \leq i \leq n_C} r_{emp_i} \leq 30\%$. Si, por otro lado, ocurre que $30\% < \max_{1 \leq i \leq n_C} r_{emp_i} \leq 50\%$ entonces el problema tiene clases moderadamente empalmadas. Por último, si ocurre que $\max_{1 \leq i \leq n_C} r_{emp_i} < 50\%$ el empalme se considera severo. Lo anterior basado en la dificultad que tiene el distinguir una clase empalmada en otra. Por ejemplo, hasta un 20 % de empalme de una clase aún permite distinguirla del resto. Mientras que para un empalme de entre 20 % y 50 % si bien aún es posible distinguir una clase, ya se tiene una pérdida moderada de la información de esta, mas con lo que aún se tiene sería posible modelarla hasta cierto punto. En cambio un

IV. Desarrollo

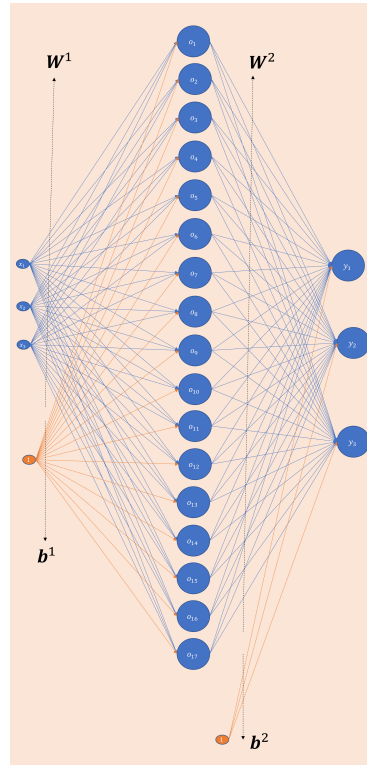


Figura 4.46: La red SLFN que separa las tres clases, con $n_O = n_H = 17$ neuronas ocultas deducida según el esquema OaO y la regla de los hiperplanos, en el ejemplo 4.6.2b

empalme mayor al 50 % ya hace difícil separar a la clase empalmada ya es difícil modelarle con lo que se tiene.

4.7.1.2. La zona de conflicto

Cuando se definió el concepto de empalme de acuerdo con la cantidad de muestras, que en una clase permanecen adentro de lo que se consideraría la zona de otra clase, se está implicando que existe una manera de contabilizar cuantas muestras de una clase invaden la zona de otra clase. Es decir que debería existir una manera de cuantificar *la intersección* de las formas de dos clases, para determinar en donde una invade la zona de otra. Este es un concepto geométrico acerca del empalme de clases. Y puesto que se cuenta con la forma geométrica o al menos una aproximación a la misma, no debería resultar difícil establecer tal concepto.

Llámesese *zona de conflicto* a la zona en particular en el espacio estado, en donde las formas de ambas clases se intersectan. El concepto de intersección de las formas de cada clase resulta muy útil en este contexto, pues permite darle una interpretación de geometría computacional al empalme o traslape de clases. Encontrar tal zona de conflicto es de particular importancia en

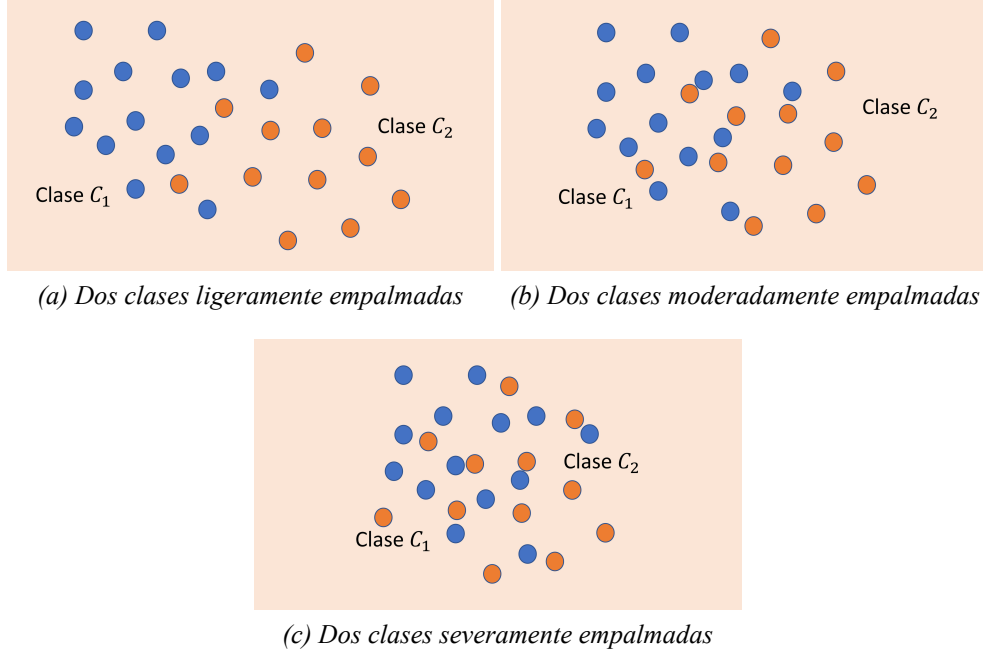


Figura 4.47: Los casos de dos clases empalmadas en $\mathbb{R}^2$

lo que al preprocesamiento de los datos de entrenamiento se trata, como se verá mas adelante. Entonces, la zona de conflicto Z_{confl} , se puede expresar como en la Ecuación 4.11.

$$Z_{confl} = \bigcap_{i=1}^{n_C} \alpha'(C_i) \quad (4.11)$$

Una interpretación gráfica del concepto de intersección entre formas de clases se muestra en la Figura 4.48 con un ejemplo para cada uno de los tipos de empalme. Esta figura muestra el mismo caso en dos dimensiones de los casos de clases empalmadas ilustradas en la Figura 4.47. La zona de conflicto se muestra sombreada en un tono transparente, justo en la intersección de las formas aproximadas de cada clase.

4.7.1.3. La clase conflicto

Ahora que la zona de conflicto se tiene perfectamente capturada, como el poliedro resultante de la intersección de las formas aproximadas de cada clase es fácil determinar cuales patrones o muestras en cada clase, se localizan adentro de esta zona de conflicto. La idea de capturar estas muestras o patrones es separarlos de sus respectivas clases y acumularlos en una nueva clase, la *clase de conflicto*, que será representada como la clase C_{confl} , como en la expresión 4.12.

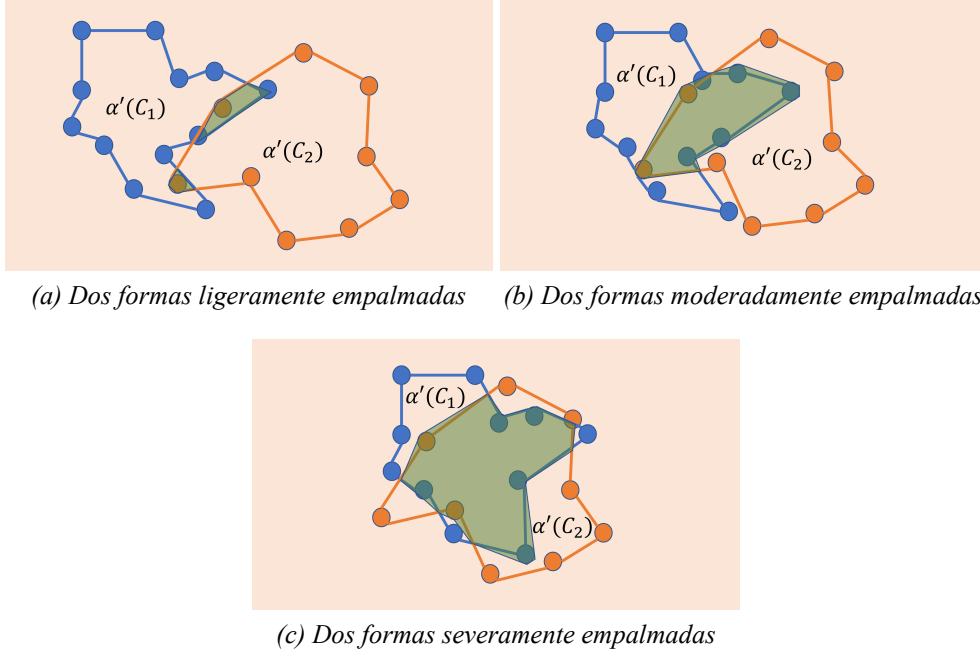


Figura 4.48: La zona de conflicto, en los tres casos de clases empalmadas, en $\mathbb{R}^2$

$$C_{confl} = \bigcup_{i=1}^{n_C} \{\mathbf{p} \in C_i \wedge \mathbf{p} \in Z_{confl}\} \quad (4.12)$$

Así pues, los patrones de cada clase adentro del poliedro que representa a Z_{confl} son separados y acumulados en la nueva clase C_{confl} , una vez que la clase conflicto ha tomado forma, las muestras conflictivas de cada clase deben eliminarse de sus clases originales, esto es $C_i = C_i / C_{confl}$ para $i = 1, 2, \dots, n_C$. La Figura 4.49 muestra el mismo caso que han ilustrado anteriormente la Figuras 4.47 y en la Figura 4.48, con la extracción de las muestras de la clase conflicto.

La Figura 4.49 muestra las clases C_1 en tono azul, la clase C_2 en anaranjado, y la nueva clase C_{confl} en color verde, para los tres casos de empalme. Al separar la clase conflicto de las n_C clases originales resulta un nuevo problema de clasificación con *tres clases que son totalmente separables*.

El preprocesamiento con clases empalmadas consiste detectar las muestras en cada clase en la zona de conflicto, y una vez localizadas, estas son separadas de sus clases originales y pasan a formar la nueva clase conflicto. El Algoritmo 12 describe este proceso.

En general este procesamiento tiene una consecuencia muy importante: agrega una nueva clase al problema, la clase conflicto C_{confl} , por lo que si el problema incorporaba originalmente a n_C clases ahora se compone de $n_C + 1$ clases.

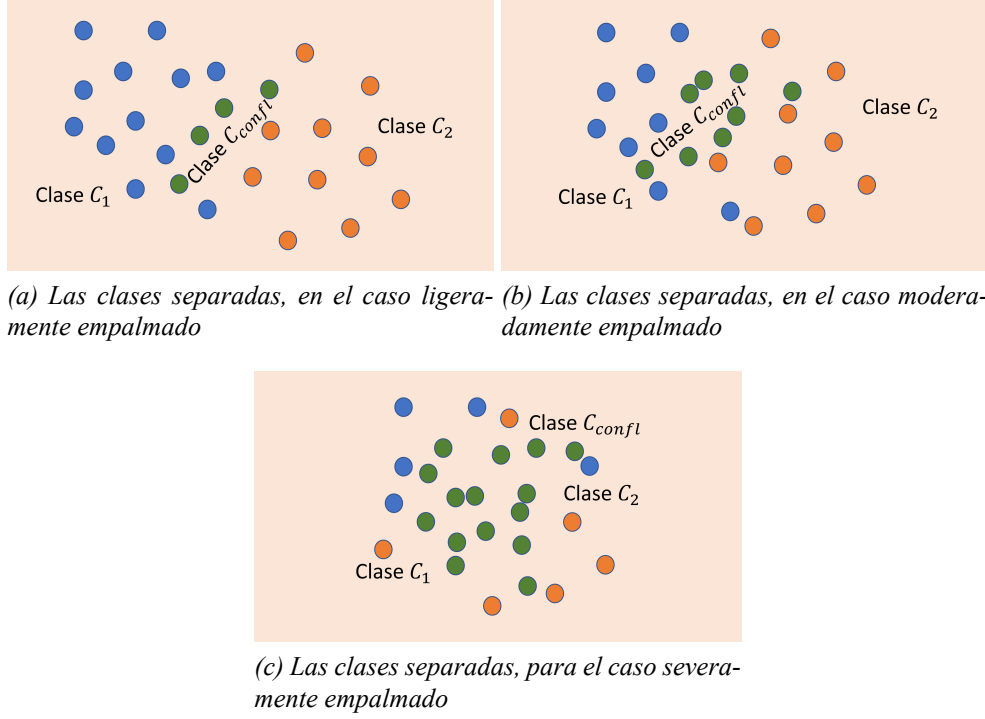


Figura 4.49: La separación final de las clases luego del preprocesamiento, en $\mathbb{R}^2$

Otra comentario oportuno tiene que ver con la operación de intersección entre la forma alfa aproximada $\alpha'(C_i)$ para $i = 1, 2, \dots, n_C$, con la zona de conflicto Z_{confl} . Esta operación quizá parezca trivial, pues es una intersección entre dos poliedros, y esto es verdad a menos que se observe un aspecto con más detalle: *la forma alfa aproximada no es un simple poliedro, es la unión de k de ellos*. Es decir que la forma alfa aproximada $\alpha'(C_i)$ es una agregación de poliedros más simples, las envolventes convexas $CH(G_j)$ para $j = 1, 2, \dots, k$, de los agrupamientos en los que la nube original C_i fue particionada. Esto implica que la intersección entre forma alfa aproximada $\alpha'(C_i)$ y zona de conflicto Z_{confl} es más compleja de lo que parece.

Para abordar este problema, la intersección por sí misma y la revisión sobre si una muestra o patrón $\mathbf{p}$ está en la clase C_i pueden llevarse a cabo en una misma operación, como lo demuestra a continuación el siguiente razonamiento: para un problema con n_C clases C_i para $i = 1, 2, \dots, n_C$ siendo $\alpha'(C_i)$ la forma alfa aproximada de cada una de ellas y cada forma alfa aproximada se compone de k_i agrupamientos, una muestra $\mathbf{p} \in C_i$ se localiza en la zona de conflicto Z_{confl} si esa misma muestra se encuentra adentro de uno de las k_j envolventes convexas que forman la forma alfa aproximada de otra clase C_j , para $i \neq j$.

El Algoritmo 13 describe exactamente este proceso, y aprovechando que las formas alfa de cada clase son sólo aproximaciones a partir de la unión de las envolventes convexas de cada uno

Algoritmo 12: Algoritmo para preprocesamiento de datos de entrenamiento

```

1 Function preProcesa( $C_1, C_2$ )
2   Calcular  $\alpha'(C_1)$ ;
3   Calcular  $\alpha'(C_2)$ ;
4   Hacer  $Z_{confl} = \alpha'(C_1) \cap \alpha'(C_2)$ ;
5   Hacer  $C_{confl} = \{\}$ ;
6   for  $p \in C_1$  do
7     if  $p \in Z_{confl}$  then
8       | Hacer  $C_{confl} = C_{confl} \cup \{p\}$ ;
9     end
10  end
11  for  $p \in C_2$  do
12    if  $p \in Z_{confl}$  then
13      | Hacer  $C_{confl} = C_{confl} \cup \{p\}$ ;
14    end
15  end
16  Hacer  $C_1 = C_1 / C_{confl}$ ;
17  Hacer  $C_2 = C_2 / C_{confl}$ ;
18  return  $\{C_1, C_2, C_{confl}\}$ 

```

de los agrupamiento en los que cada clase fue particionada, las operaciones de intersección y formado de la zona de conflicto se llevan a cabo juntas, mediante una verificación sobre si una muestra de una clase queda adentro de la envolvente convexa de uno de los grupos de otra clase, de ser así la clase conflicto queda inmediatamente establecida.

Debe observarse como este algoritmo toma con argumentos a las clases, no sólo sus formas alfa aproximadas, y como resultado arroja las clases sin empalmes y la nueva clase conflicto directamente. El Algoritmo 13 funciona para n_C clases, con $n_C \geq 2$.

4.7.2. Ejemplos del proceso de aislamiento de la zona y clase conflicto

A continuación se presentan algunos ejemplos del aislamiento de la zona de conflicto, en primer lugar, y la clase conflicto, como consecuencia, en problemas en $\mathbb{R}^2$ y $\mathbb{R}^3$. En estos ejemplos, sólo se consideran problemas de clasificación binarios, con clases empalmadas en distintos grados de empalme.

Algoritmo 13: Algoritmo mejorado para el preprocesamiento de datos de entrenamiento

```

1 Function preProcesa( $\{C_1, C_2, \dots, C_{n_C}\}$ )
2   for  $i = 1$  to  $n_C$  do
3     | Hacer  $\alpha'(C_i) = aShape(C_i)$ ;
4   end
5   Sea  $k_i$  el total de agrupamientos en los que se dividió a  $C_i$ , para  $i = 1, 2, \dots, n_C$ ;
6   Hacer  $C_{confl} = \{\}$ ;
7   for  $i = 1$  to  $n_C$  do
8     | for  $p \in C_i$  do
9       |   for  $j = 1$  to  $k_i$  do
10        |   | if  $p \in CH(G_j)$  then
11          |   |    $C_{confl} = C_{confl} \cup \{p\}$ ;
12          |   | end
13        |   end
14      | end
15    end
16    for  $i = 1$  to  $n_C$  do
17      | Hacer  $C_i = C_i / C_{confl}$ ;
18    end
19  return  $\{C_1, C_2, \dots, C_{confl}\}$ 

```

4.7.2.1. Ejemplo 4.7.2a

En este primer ejemplo un problema de clasificación binario en $\mathbb{R}^2$ es presentado. En este problema existen las clase C_1 y la clase C_2 , ambas contienen $n_{P_1} = n_{P_2} = 600$ muestras. El empalme entre ellas es moderado, pues cerca del 46 por ciento de las muestras en la clase C_1 , la clase con más muestras conflictivas se consideran en la zona de conflicto. Aplicando el preprocesamiento aquí expuesto, se obtienen las nuevas versiones sin empalme de C_1 y de C_2 , así como la clase conflicto C_{cnf} .

En la Figura 4.50a se muestra la disposición de las clases C_1 mostrada con muestras en color rojo, y de la clase C_2 mostrada con muestras en color azul. En la Figura 4.50b aparecen las clases C_1 y C_2 sin muestras conflictivas, y la clase conflicto C_{cnf} con muestras en color negro.

4.7.2.2. Ejemplo 4.7.2b

Este es otro ejemplo de separación de clases para clasificación binaria en $\mathbb{R}^2$. En este ejemplo ambas clases contienen la misma cantidad de muestras, 700. Las clase tienen un empalme moderado, como lo muestra gráficamente la Figura 4.51.

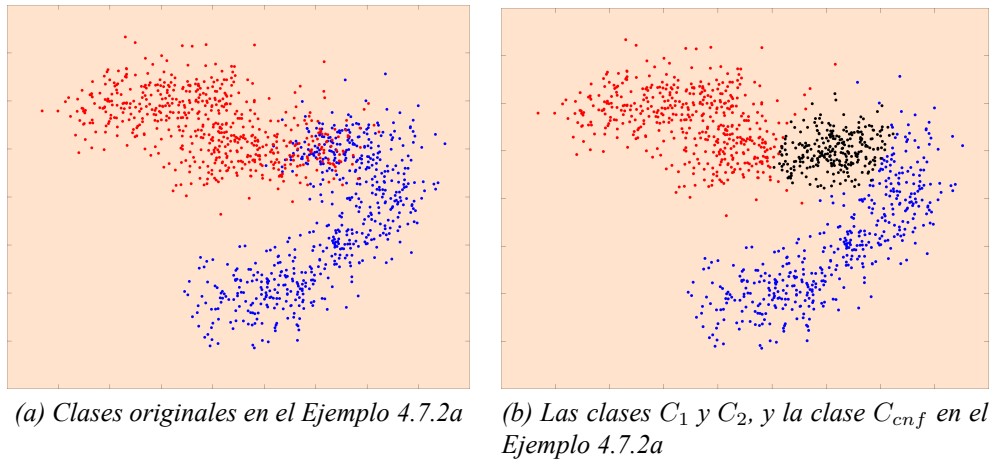


Figura 4.50: Separando las clases conflicto del resto de ellas en el Ejemplo 4.7.2a

La Figura 4.51a muestra las dos clases originales en su espacio bidimensional. La Figura 4.51b muestra la separación de las clases originales sin sus muestras conflictivas, y la clase conflicto resultante. Las muestras de la clase C_1 aparecen en color rojo, las de la clase C_2 en color azul, y las muestras de la clase C_{cnf} aparecen en color negro.

4.7.2.3. Ejemplo 4.7.2c

Este es un ejemplo en $\mathbb{R}^3$. Es un problema de clasificación binario en donde ambas clases C_1 y C_2 contienen 600 muestras, véase la Figura 4.52a. El empalme entre ambas clases es severo, pues la clase con la máxima cantidad de muestras en conflicto contiene cerca del 60 por ciento de sus muestras en la zona de conflicto.

La Figura 4.52b muestra la disposición de las clases C_1 y C_2 sin sus muestras conflictivas, y la clase C_{cnf} . Las muestras en la clase C_1 aparecen en color rojo, las muestras en la clase C_2 en color azul y las muestras en la clase C_{cnf} en color negro.

4.7.3. La clasificación con clases traslapadas

La clasificación en caso de clases empalmadas o traslapadas se lleva a cabo usando una de tres metodologías. Estas tres se basan en la idea de que *en clases traslapadas se puede lograr una separación si la clase conflicto es considerada una clase extra y separada de las clases originales*.

Esta filosofía se acopla a la perfección a un clasificador implementado a partir de una red SLFN, pues una frontera entre clases habrá de aparecer después de que esa clase conflicto sea

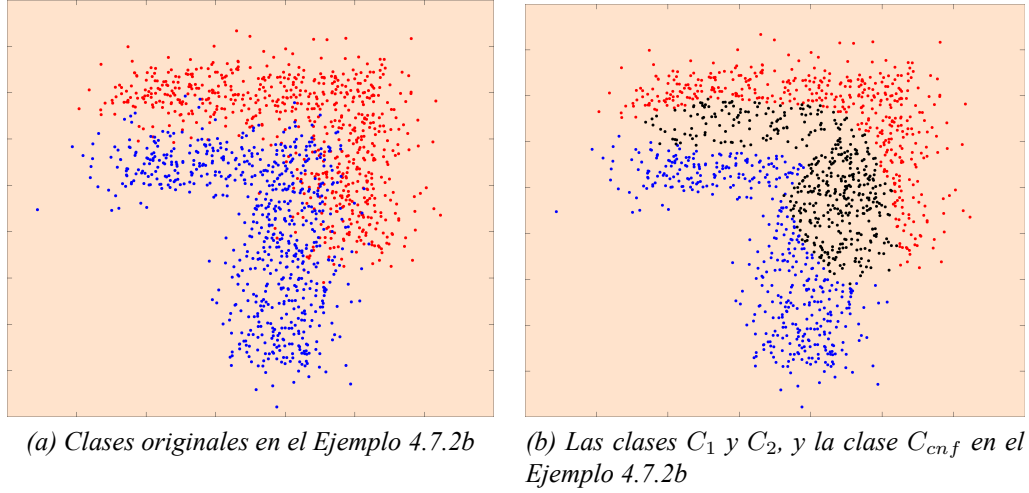


Figura 4.51: Separando las clases conflicto del resto de ellas en el Ejemplo 4.7.2b

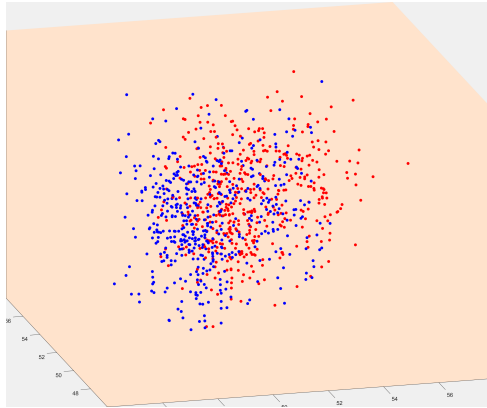
generada y tratada, utilizando uno de los tres esquemas mencionados en [102] y listados a continuación.

- (1) El *esquema del descarte*: la clase conflicto es, sencillamente, descartada del problema.
- (2) El *esquema del mezclado*: aquí, en una primera etapa, las muestras conflictivas en cada clase son separadas de estas, y acumuladas en una nueva clase, la clase conflicto, propiamente. Así que el problema pasa de uno de separar n_C clases a uno de $n_C + 1$ clases, las originales sin muestras conflictivas mas la nueva clase conflicto. En una segunda etapa y solo en el caso de que una nueva muestra sea clasificada como adentro de la clase conflicto, se usaría un segundo clasificador especializado en separar adentro de la clase conflicto, para con las clases originales.
- (3) El *esquema del separado*: el problema en una primera etapa separa la zona de conflicto de la zona sin conflicto, nótese que en esta primera etapa solo se usa un clasificador, uno que separa entre dos clases: conflicto y no conflicto. Luego en una segunda etapa, se separa para con las clases originales con cualquiera que fuera el resultado del primer clasificador.

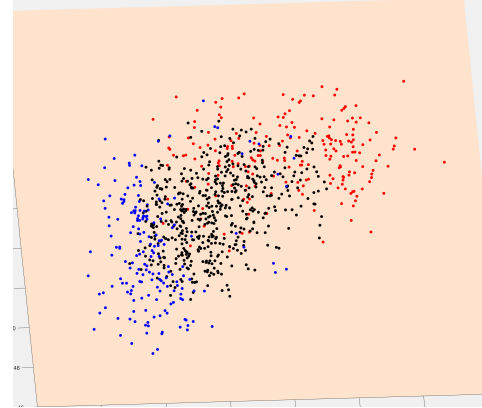
La Figura 4.53 muestra una representación gráfica de lo que estos tres esquemas para manejo de clase empalmadas proponen. Nótese como los esquemas del mezclado y del separado son esquemas que implican un clasificador en dos etapas separadas. El esquema del descarte en cambio, usa una sola etapa de clasificación.

El *esquema del descarte es por mucho el más simple*. Consta de un solo clasificador como lo muestra la Figura 4.53a que *aprende a separar las clases originales C_i para $i = 1, 2, \dots, n_C$* ,

IV. Desarrollo

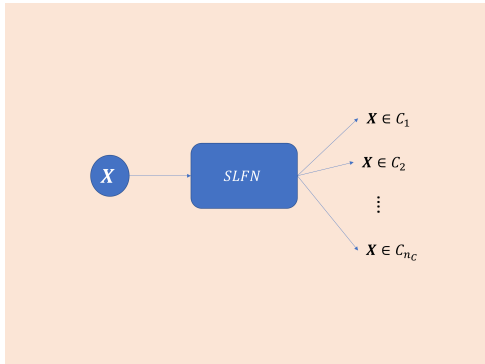


(a) Clases originales en el Ejemplo 4.7.2c

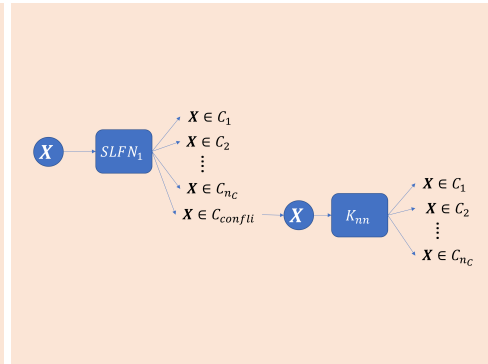


(b) Las clases C_1 y C_2 , y la clase C_{conf} en el Ejemplo 4.7.2c

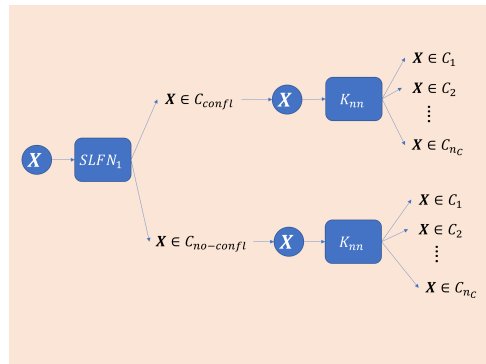
Figura 4.52: Separando las clases conflicto del resto de ellas en el Ejemplo 4.7.2c



(a) El esquema del descarte



(b) El esquema del mezclado



(c) El esquema del separado

Figura 4.53: Los tres esquemas para clasificación en clases traslapadas, manipulando la clase conflictiva, en $\mathbb{R}^2$

pero sin sus muestras conflictivas, es decir aquellas muestras de cada clase que pertenecían a la clase conflicto, pues esta última ha sido eliminada del problema. Lo que queda es un problema

perfectamente separable. El esquema del descarte, sin embargo, tiene un inconveniente: puesto que elimina la clase conflictiva del problema *no es recomendable usarlo en un problema con empalme moderado o severo*, pues ese descarte de la clase conflictiva perdería demasiada información y por lo tanto, generaría un problema demasiado simple que no representa en lo más mínimo al problema original.

El esquema del mezclado presentado a manera de diagrama de bloques en la Figura 4.53b es un esquema compuesto, en el sentido de que el proceso completo de clasificación se divide en dos etapas separadas e independientes, ambas listadas a continuación.

- (1) La primera etapa consiste en separar a las n_C clases originales C_i para $i = 1, 2, \dots, n_C$, cada una sin sus muestras conflictivas, y también a la clase conflictiva C_{confl} . En principio, esto no debe resultar difícil, pues este problema involucra a $n_C + 1$ clases separadas, luego, *se aplicará la red SLFN, para implementar esta primera etapa.*
- (2) La segunda etapa consiste en otro clasificador que se activa si y sólo si el resultado del primer clasificador es una muestra asignada a la clase conflicto, $\mathbf{X} \in C_{confl}$. Este segundo clasificador es uno especializado en resolver el problema en la zona de traslape, en donde las clases se empalman, por lo que no es recomendable usar una red SLFN en esta etapa. En este estudio, se usa un *clasificador kNN* para resolver la clasificación en la zona de empalme.

Entonces la red SLFN clasificadora sólo aplica en la primera etapa, el resultado final de la clasificación sólo es el mismo que el generado por la red SLFN, si la muestra nueva es asignada a una de las n_C clases originales. Si la red SLFN asigna una muestra a la clase conflicto, entonces la decisión final sobre a cual clase esta muestra debe ser asignada será tomada en la segunda etapa, donde, en este estudio, se usa un clasificador kNN especializado para la zona de empalme.

De la misma manera, compuesto por dos etapas separadas e independientes, es el caso del esquema del separado. En este caso mostrado esquemáticamente en la Figura 4.53c se utilizan tres clasificadores, uno de ellos que puede ser una red SLFN, se encarga de separar las clases conflicto C_{confl} de las clases no conflictivas, que básicamente consiste en la unión de todas las clases originales sin sus muestras conflictivas, $C_{no-confl} = \bigcup_{i=1}^{n_C} C_i$. El segundo y tercer clasificador separan de las clases conflictivas y no conflictivas, respectivamente, a las clases originales así pues, dos aspectos son muy importantes en esto, y se listan a continuación.

- (1) La primera etapa, con el *primer clasificador*, separa a las clases C_{confl} de la clase $C_{co-confl}$, *mismas que son separables*. Nótese que este clasificador es un clasificador binario siempre. Este primer clasificador se puede implementar usando una red SLFN.

IV. Desarrollo

- (2) En la *segunda etapa*, existen dos clasificadores separados e independientes que pueden activar simultáneamente, el primero está especializado en separar, de la zona de no conflicto, a las clases originales, este clasificador también se puede implementar a partir de una red SLFN, pero, también, se puede implementar a partir de otro estilo de clasificador, uno kNN, por ejemplo. Y el segundo clasificador es muy especializado, debe separar a las clases originales en la zona de conflicto. Aquí es útil utilizar un clasificador kNN para esta zona.

El clasificador en la Figura 4.53c muestra el diagrama a bloques del esquema de separado. El clasificador en la segunda etapa, que se encarga de separar a las clases originales en la zona de no conflicto. En esta figura se muestra implementado a partir de un clasificador kNN, pero puede ser implementado por una red SLFN, puesto que las clases originales sin sus muestras conflictivas, son separables.

Los esquemas del mezclado y del separado son esquemas compuestos, luego su implementación práctica resulta más compleja. Es por ello que *sólo se recomiendan para los casos con empalmes severos y moderados* [102]. En el caso de empalme ligero, es mucho más práctico recurrir al esquema del descarte.

En ambos casos del esquema del mezclado y del esquema del separado que se utilizan en este estudio son clasificadores kNN, por las ventajas mostradas en un problema de clasificación es multiclase y no separable [103]:

- (1) Son realmente *sencillos de implementar*.
- (2) Son *muy flexibles*, funcionan con patrones cuyos atributos pueden ser cualquier cosa, siempre y cuando una buena medida de distancia sea también proveída.
- (3) Pueden *manejar clasificación multiclase* sin ninguna complicación.
- (4) Pueden *manejar clasificación en clases no separables*, con el entendido de que algún cierto grado de error estará presente.

Tienen naturalmente, sus claras desventajas también [103]:

- (1) Para funcionar adecuadamente *requieren que los datos de entrenamiento sean numerosos*.
- (2) Puesto que su fundamento es la comparación de distancias entre las muestras en una vecindad cercana para con una nueva muestra a clasificar, *necesitan todas las muestras en el conjunto de entrenamiento, cada vez que hacen una nueva predicción*.
- (3) Lo que sea que se considere como *la medida de distancia*, deberá ser una buena medida de distancia, de lo contrario, las predicciones del clasificador resultarán bastante desviadas.

El clasificador completo en este estudio utiliza el esquema del mezclado, es uno en dos etapas, como en el caso de los esquemas del separado y del mezclado. En la primera etapa se utiliza el clasificador SLFN cuya capa oculta será calculada a partir de la separación de estas $n_C + 1$ clases. La segunda etapa utiliza un clasificador kNN especializado en la clase conflicto, como ya fue antes mencionado. La Figura 4.54 muestra a modo de diagrama a bloques, la construcción final del clasificador de patrones.

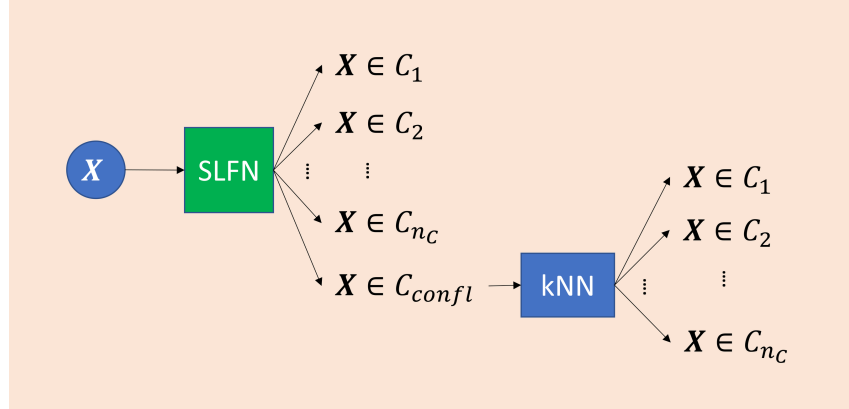


Figura 4.54: El clasificador final, utilizado en este estudio, usando el esquema del mezclado

4.7.4. Ejemplos de clasificación con clases traslapadas

En esta última subsección, se utilizan los mismos datos en los ejemplo utilizados en la última subsección 4.7.2. Se aplican las reglas presentadas en esta sección para la primera etapa en problemas de clasificación traslapados, utilizando el esquema del mezclado. Se muestra el diseño de la red SLFN clasificadora utilizada en la primera etapa.

4.7.4.1. Ejemplo 4.7.4a

Este es un problema de clasificación binaria entre dos clases C_1 y C_2 , con un empalme moderado, como lo muestra la Figura 4.55a. Usando el esquema del mezclado la red SLFN en la primera etapa del clasificador tiene una capa oculta con $n_O = 6$ neuronas. Esto se debe a que para separar las clases C_1 y C_{cnf} se requieren $n_{H_{1c}} = 3$ hiperplanos. Para separar las clases C_2 y C_{cnf} se requieren $n_{H_{2c}} = 3$ hiperplanos, y para separar las clases C_1 y C_2 se requieren $n_{H_{12}} = 2$ hiperplanos. Por lo que usando el esquema OaO, $n_H = n_{H_{1c}} + n_{H_{2c}} + n_{H_{12}} = 8$ hiperplanos separan las tres clases en el problema, entonces la red SLFN ha de contar con $n_O = 8$ neuronas ocultas.

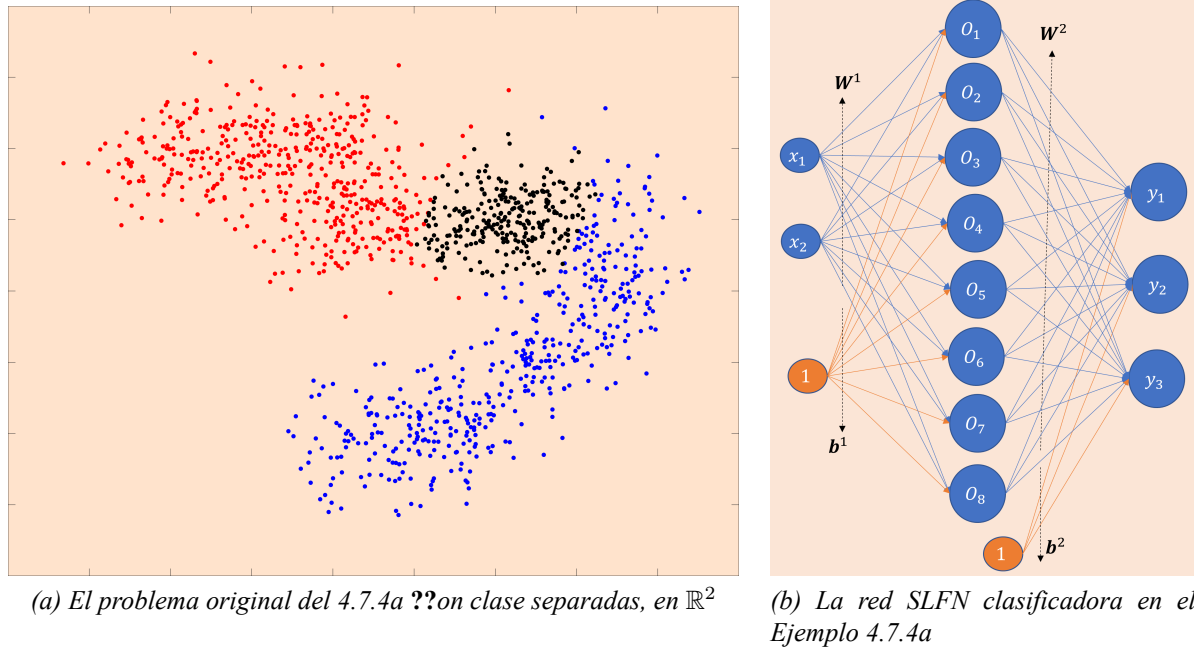


Figura 4.55: El problema de clasificación del ejemplo 4.7.4a, separado con clases C_1 , C_2 y C_{cnf} y la red SLFN para la primera etapa del clasificador, según el esquema del mezclado

En la Figura 4.55b la capa oculta muestra las $n_O = 8$ neuronas que se requieren para separar las tres clases en el problema.

4.7.4.2. Ejemplo 4.7.4b

Este ejemplo también es un problema de clasificación en $\mathbb{R}^2$. En esta ocasión la clases C_1 y C_2 contienen $n_{P_1} = n_{P_2} = 600$ muestras, como lo muestra la Figura 4.56a. Las clases como se muestran en la Figura 4.56 se encuentran moderadamente traslapadas o empalmadas, pues la clase con mayor cantidad de muestras en conflicto en este caso es la clase C_1 que contiene cerca del 36 % de sus muestras en la zona de conflicto. Para separar las clases C_1 y C_{cnf} se requieren, según la regla de los hiperplanos, $n_{H_{1c}} = 5$ hiperplanos, para separar las clases C_2 y C_{cnf} se requieren $n_{H_{2c}} = 6$ hiperplanos y para separar las clases C_1 y C_2 se requieren $n_{H_{12}} = 4$ hiperplanos. Por la agregación de hiperplanos propuesta en el esquema OaO, la separación total del problema requiere $n_H = n_{H_{1c}} + n_{H_{2c}} + n_{H_{12}} = 5 + 6 + 4 = 15$ hiperplanos, y por lo tanto $n_O = n_H = 15$ neuronas ocultas, en la red SLFN en la primera etapa del clasificador, como lo muestra la Figura 4.56b.

Este problema aún más que el anterior, implica clases C_1 , C_2 y C_{cnf} que si bien están perfectamente separadas, la separación entre ellas es pequeña y por lo tanto, generando $n_H = n_O = 15$

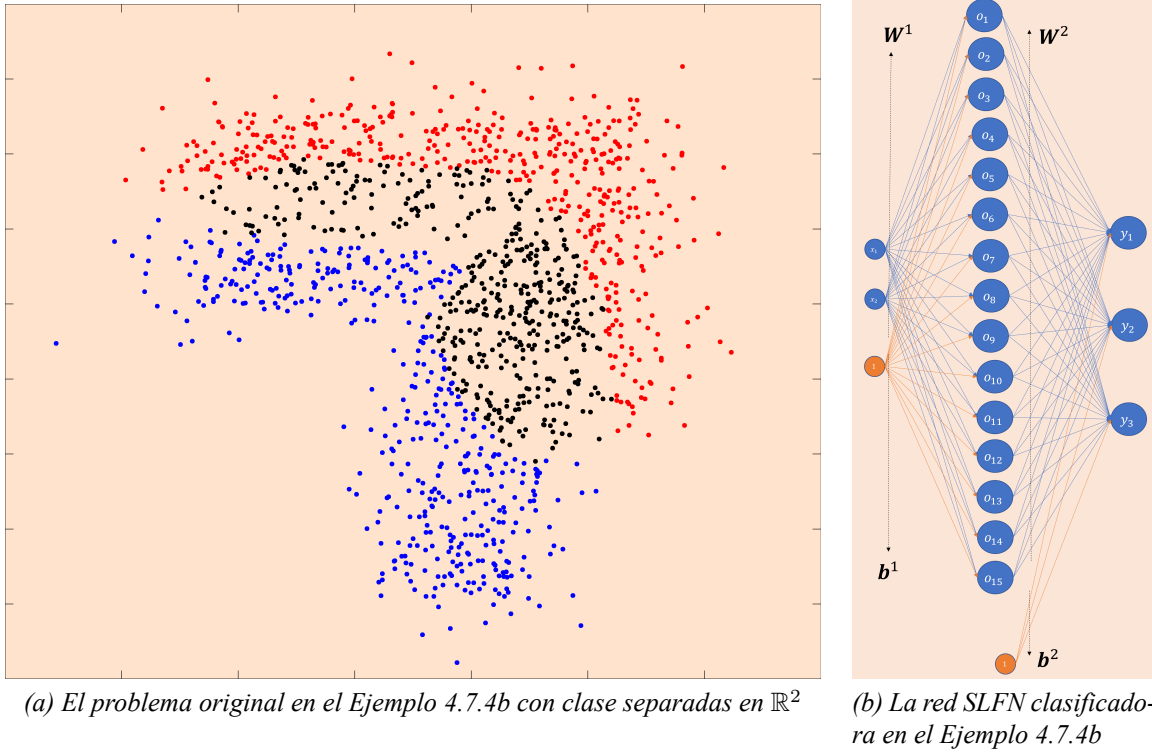


Figura 4.56: El problema de clasificación en el Ejemplo 4.7.4b ya separado, con clases C_1 , C_2 y C_{cnf} y su red SLFN para la primera etapa del clasificador, según el esquema del mezclado

hiperplanos.

4.7.4.3. Ejemplo 4.7.4c

Este es un problema de clasificación binario en $\mathbb{R}^3$. En este problema existen dos clases empalmadas, la clase C_1 con 500 muestras y la clase C_2 con 500. Este caso tiene un empalme severo, pues cerca del 59 % de las muestras en C_2 se encuentran en la zona conflictiva del problema. El problema aparece en la Figura 4.57a. De la manera acostumbrada las muestras de C_1 , sin muestras conflictivas aparecen en color rojo, las de C_2 en color azul y las de C_{cnf} en color negro.

Al aplicar el esquema de separación multiclase, según OaO, como fue expuesto antes, resulta que para separar las clases C_1 y C_{cnf} se requieren $n_{H_{1c}} = 6$ hiperplanos, para separar las clases C_2 y C_{cnf} se requieren $n_{H_{2c}} = 7$ hiperplanos, y para separar las clases C_1 y C_2 se requieren $n_{H_{12}} = 3$ hiperplanos. Al agregar todos estos se tiene que para separar el problema completo, serán necesarios $n_H = n_{H_{1c}} + n_{H_{2c}} + n_{H_{12}} = 6 + 7 + 3 = 16$ hiperplanos. Por la regla de los hiperplanos entonces, la red SLFN en la primera etapa del clasificador compuesto, según

IV. Desarrollo

el esquema del mezclado tendrá una capa oculta compuesta por $n_O = n_H = 16$ neuronas. La Figura 4.57b muestra esta red SLFN clasificadora.

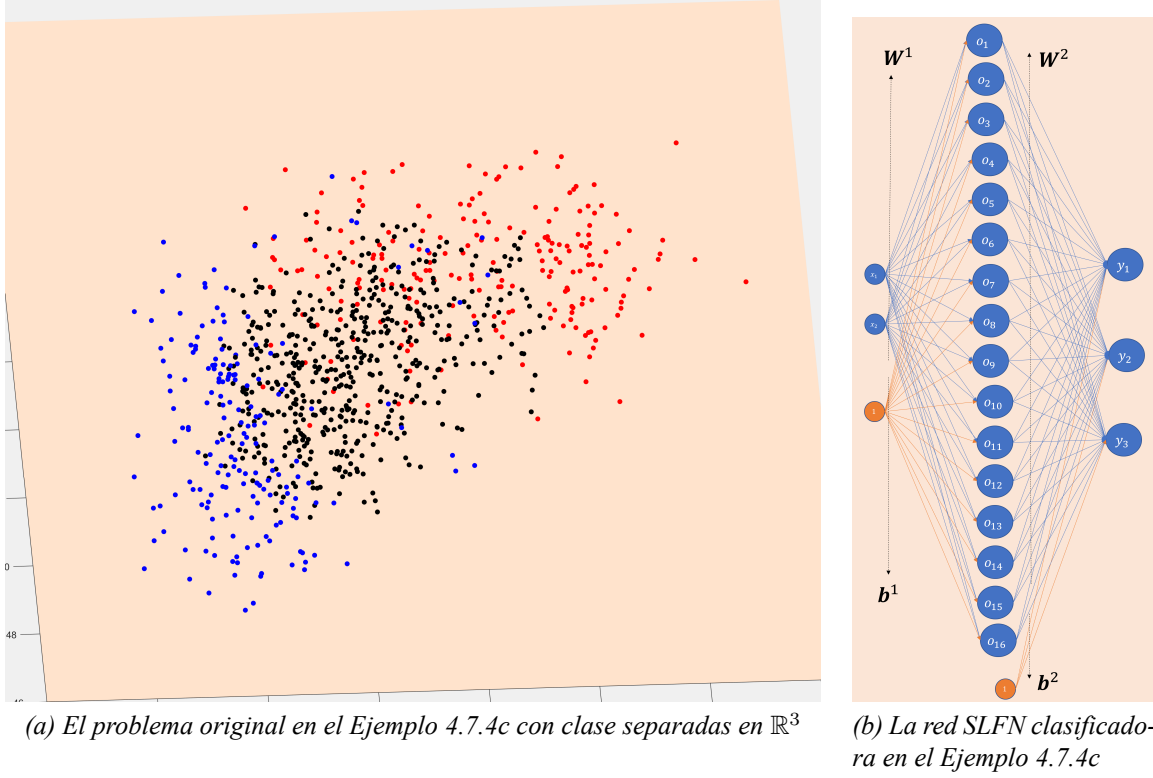


Figura 4.57: El problema de clasificación en el Ejemplo 4.7.4c ya separado con clases C_1 , C_2 y C_{cnf} y su red SLFN para la primera etapa del clasificador, según el esquema del mezclado

Este ejemplo muestra que su red SLFN clasificadora tiene una capa oculta grande, pues la cantidad de hiperplanos que separan las clases involucradas en el problema es alto dado lo ajustado que se encuentra cada par de clases, en el espacio $\mathbb{R}^3$ en donde se encuentran.

Capítulo 5

Validación

En este capítulo una serie de problemas de clasificación son presentados. La idea de este capítulo es finalmente evaluar el desempeño del clasificador SLFN, cuya única capa oculta es calculada usando la regla de los hiperplanos. La evaluación se realiza comparando el desempeño con respecto a otras técnicas de clasificación.

5.1. Técnicas de clasificación a comparar

Las técnicas clasificadoras a considerar en esta comparación con la red SLFN diseñada con el esquema propuesto aquí, son los cuatro listados a continuación.

- (1) Un *clasificador kNN* con un valor $k = 5$.
- (2) Un *clasificador bayesiano* con una parametrización normal.
- (3) Un *clasificador SVM* con kernel RBF.
- (4) Un clasificador basado en *árboles de decisión*.

5.1.1. El clasificador kNN

El clasificador kNN es una forma de clasificación sin supervisión que goza de gran popularidad por su simple implementación, y al menos en problemas de clasificación simples suele comportarse de excelente manera, además no asume ninguna distribución o parametrización en la información. Su regla para tomar una decisión sobre a qué clase C_i para $i = 1, 2, \dots, n_C$ debe asignar una nueva muestra $\mathbf{X} \in \mathbb{R}^{n_F}$ es muy simple: asignarla a la clase a la que la mayoría de sus *vecinos* pertenece. Esto implica que cada vez que una nueva muestra debe ser clasificada, el clasificador kNN debe localizar los *vecinos más cercanos* específicamente k de ellos.

Su principal desventaja sin embargo, es su modelo de aprendizaje conocido como *aprendizaje perezoso*. Sucede que los clasificadores kNN en realidad no aprenden nada de la información de entrenamiento [55]. Antes bien, usan toda la información de entrenamiento cada vez que deben tomar una nueva decisión para localizar los k vecinos más cercanos, lo que hace evidente que su uso en problemas con mucha información sea poco recomendable.

Además del conjunto de entrenamiento, el clasificador kNN posee un segundo parámetro que dictamina en gran medida su propio desempeño, y este se refiere al valor $k \in \mathbb{N}$. Este es tamaño de la vecindad, es decir la cantidad de vecinos que serán admitidos para la decisión final sobre una nueva muestra. Este parámetro debe utilizarse con el valor correcto, pues si es muy grande el procedimiento final será aún más lento, y si es muy pequeño el resultado de la clasificación suele quedar sobreajustado.

5.1.2. El clasificador bayesiano

El clasificador bayesiano o *clasificador NB* por las siglas en inglés de *Naive Bayes*, es también un clasificador muy popular. Tiene como principal ventaja el que es un clasificador simple de implementar, y su regla de decisión también es muy simple: asignar un nuevo patrón a la clase cuya probabilidad posterior sea mayor, es decir que asigna $\mathbf{X} \in \mathbb{R}^{n_E}$ a la clase C_i , para $i = 1, 2, \dots, n_C$, siempre que $p(\mathbf{X} \in C_i | \mathbf{d}) > p(\mathbf{X} \in C_j | \mathbf{d})$ para $i \neq j$ y en donde $\mathbf{d}$ es la evidencia del problema.

Una de las ventajas más evidentes del uso del clasificador bayesiano es que la evidencia $\mathbf{d}$ es fácil de obtener a partir de los datos de entrenamiento, así como también lo es el cálculo de las probabilidades involucradas.

Su principal desventaja es que el clasificador bayesiano supone que la información de entrenamiento se ajusta o sigue una distribución específica para así calcular las probabilidades necesarias en el uso de la regla de Bayes, esta presunción es demasiado, pues en la mayoría de los casos la información de entrenamiento no sigue una distribución de datos específica, lo que puede provocar que las predicciones de un clasificador bayesiano resulten poco confiables. Otra desventaja del clasificador bayesiano es que supone que los atributos de los patrones de entrada son independientes, esta es una de las razones por la que este tipo de clasificador es sencillo de implementar, y sin embargo, los atributos de un patrón de entrada suelen estar relacionados en menor o mayor grado.

5.1.3. El clasificador basado en árboles de decisión

Otro clasificador que goza de enorme popularidad es el clasificador basado en árboles de decisión. Un clasificador de este estilo toma una decisión acerca de la categoría a que asigna una nueva muestra de acuerdo con un particionamiento que lleva a cabo del espacio del problema, de tal forma que cada partimiento ocurre de acuerdo con una comparación de los atributos del problema y ciertos umbrales previamente establecidos para cada atributo, sobre el espacio estado. Eventualmente, durante uno de los particionamientos se encontrará la clase a la que el patrón pertenece.

El clasificador basado en un árbol de decisión se implementa como una simple serie de preguntas cuya respuesta es *si*, o *no*, una vez respondida una de ellas otra se presenta de inmediato, cada pregunta particiona el espacio del problema, hasta que eventualmente la pregunta original es respondida, y la clase de la nueva muestra queda expuesta. También cada pregunta equivale a un nodo del árbol, cada respuesta lleva hacia uno de dos hijos del nodo, dando lugar a un nuevo nivel de profundidad. De esta manera la raíz de árbol es el punto de inicio, es la primer pregunta de la serie. Las hojas del árbol en cambio, son las respuestas a la pregunta original, la respuesta acerca de la clase a la que pertenece el nuevo patrón. El entrenamiento de un clasificador basado en redes de decisión consiste en precisamente construir el árbol de decisión a partir de los datos de entrenamiento.

Los árboles de decisión como clasificadores tienen algunas claras ventajas [104]: son fáciles de implementar, son intuitivos, es decir, la forma en que razonan se parece mucho a la manera en que lo hace la mayoría de las personas, funcionan tanto con información categórica como numérica, además son capaces de funcionar bien con datasets grandes. Tienen sin embargo, algunas desventajas [104], por ejemplo tienden a sobreajustarse mucho a la información, por lo que no generalizan bien. Son muy poco robustos en el sentido de que cualquier cambio en la información de entrenamiento invalida a todo el árbol recién aprendido, tienden a favorecer a las clases mayoritarias. Por si fuera poco, el problema de construir un árbol óptimo es un problema *np*, es decir que es muy difícil conseguir un árbol óptimo, y se debe conformar con uno subóptimo en la mayoría de los casos.

5.1.4. El clasificador SVM

En realidad una máquina SVM es una red SLFN, excepto que su método de aprendizaje y parte de su funcionamiento interno difiere del de las redes MLP regulares. Las máquinas SVM, siglas en inglés de *Support Vector Machine*, son máquinas de aprendizaje bastante populares

actualmente. Las máquinas SVM son clasificadores por naturaleza son clasificadores binarios, específicamente.

Las máquinas SVM son clasificadores binarios, únicamente en esta modalidad pueden funcionar, y se entrenan usando optimización cuadrática, lo que les hace clasificadores muy confiables en lo que a aprender relaciones entrada y salida se refiere. Las máquinas SVM aprender a separar las muestras de una y otra clase, transformando el espacio original del problema $\mathbb{R}^{n_E}$ en otro espacio $\mathbb{R}^{n_{opt}}$ en donde las muestras de entrenamiento son linealmente separables, y en donde $n_{opt} > n_E$. Así pues, un hiperplano separa las dos clases.

Y este hiperplano separador en ese espacio aumentado, se localiza mediante los *vectores soporte*, aquellas muestras en el conjunto de entrenamiento, que localizan de manera única en donde se posiciona el hiperplano, ninguna otra muestra logra esto, sólo los vectores soporte. Si uno de estos vectores soporte se mueve se mueve también el hiperplano separador.

Las máquinas SVM tienen la ventaja de poseer un método de entrenamiento que siempre resulta en el valor óptimo de las conexiones, pues su superficie de desempeño a optimizar es siempre cuadrática. Además el tamaño de la capa oculta, uno de los problemas primordiales de las redes SLFN, queda establecida por la cantidad de vectores de soporte, y esta cantidad es uno de los productos del método de entrenamiento de las máquinas SVM. Sin embargo las máquinas SVM padecen algunas claras desventajas, por ejemplo, sólo son clasificadores binarios, para un problema multiclase se deben combinar más de dos máquinas SVM distintas. Otra desventaja es que no suelen comportarse bien en problemas con una gran cantidad de datos.

5.2. Ejemplos de aplicación

En esta sección, se presentan algunos problemas de clasificación, cada uno de ellos con diferentes características, y niveles de dificultad. La intención del presente capítulo, es presentar, y revisar, el desempeño del clasificador basado en una red SLFN cuya cantidad de neuronas la capa oculta fue determinada con la regla de los hiperplanos. El análisis del comportamiento de la red clasificador a SLFN ocurre en dos formas, como se lista a continuación.

- (1) Determinando, de diferentes maneras, la cantidad n_O , y cual es el desempeño de la red SLFN producida, en cada caso.
- (2) Comparando el desempeño del clasificador SLFN, que usa la regla de los hiperplanos para obtener n_O , con otros modelos de clasificadores.

En el primer caso lo que se compara es la exactitud de clasificación de la red SLFN, utilizando

otros métodos para establecer n_O . así como el tiempo requerido para obtener n_O . En este sentido, además del método aquí propuesto, se exponen otros tres métodos:

- (1) La regla de oro denominada *regla del promedio*, que establece que $n_O = \frac{n_E + n_S}{2}$.
- (2) La regla de oro denominada la *regla de los dos tercios*, que hace $n_O = \frac{2n_E}{3} + n_S$.
- (3) Y el llamado *método constructivo* de fuerza bruta.

En el segundo caso la exactitud de la red diseñada con el método propuesto se compara con cuatro clasificadores. Los clasificadores son, un clasificador kNN con $k = 5$, un clasificador bayesiano con parametrización normal, un clasificador basado en árboles de decisión, y una máquina SVM con un kernel RBF. En cada ejemplo los diferentes clasificadores se comparan usando los criterios listados a continuación.

- (1) La *exactitud* del clasificador, esto es, el porcentaje de muestras bien clasificadas.
- (2) El tiempo de que le toma al clasificador alcanzar su respuesta, medida en segundos.

5.2.1. Ejemplos de aplicación sobre datasets artificiales

Los siguientes nueve ejemplos ilustran la aplicación de la construcción de un clasificador basado en una red SLFN cuya capa oculta ha sido construida usando el método propuesto y su comparación con otros métodos tanto para construir la capa oculta, como de otros modelos de clasificación sobre conjuntos de datos que han sido generados artificialmente. Se dice que un dataset es artificial cuando ha sido generado buscando llenar ciertas características o atributos especiales, sea para simular un proceso con parámetros conocidos, o para formar un dataset con una forma o distribución muy específica de modo que bajo estas circunstancias sea más sencillo comparar el desempeño de los diferentes modelos que serán puestos a prueba. Estos nueve ejemplos han sido ya presentados en el capítulo anterior.

Para la implementación de cada uno de estos cinco modelos de clasificadores dos *toolboxes* de Matlab han sido utilizados, a saber, los listados a continuación

- (1) El toolbox *Statistics and Machine Learning Toolbox*.
- (2) El toolbox *Deep Learning Toolbox*.

El toolbox Statistics and Machine Learning Toolbox proporciona la facilidad para implementar clasificadores como el clasificador kNN, el clasificador bayesiano, el clasificador basado en árboles de decisión, y el clasificador SVM. Por otro lado el toolbox Deep Learning Toolbox proporciona la facilidad en la implementación del clasificador SLFN.

5.2.1.1. Ejemplo 5.2.1a

Este primer problema es un problema de clasificación en $\mathbb{R}^2$, entre dos clases, la clase C_1 con 1000 muestras de entrenamiento y la clase C_2 con 1100. La distribución de las clases aparece en la Figura 5.1. Dos tablas muestran una comparativa en cuanto a metodologías. Una de ellas con el comportamiento de la red SLFN, contrastando con tres técnicas alternativas para obtener el tamaño de la capa oculta. En la otra tabla, el comportamiento de la misma red SLFN como clasificadora, en comparación a las otras cuatro técnicas para implementar un clasificador.

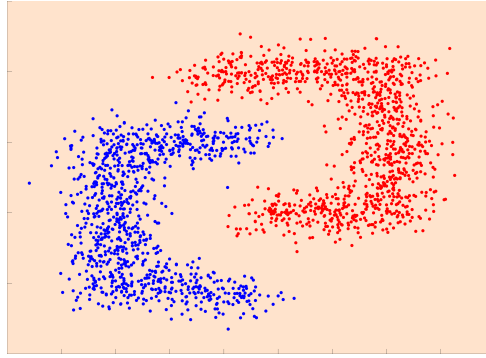


Figura 5.1: El problema original en $\mathbb{R}^2$ del Ejemplo 5.2.1a

Tabla 5.1: Tabla comparativa sobre métodos para obtener n_O en el Ejemplo 5.2.1a

Método	n_O	Tiempo	Exactitud
Promedio	2	0.001	85
Dos tercios	3	0.001	85
Constructivo	4	140	98
Hiperplanos	4	85	98

Tabla 5.2: Tabla comparativa de los cinco modelos de clasificador en el Ejemplo 5.2.1a

Método	Exactitud	Tiempo
Hiperplanos	98	0.13
Bayesiano	93	0.12
kNN	98	0.16
Árboles	95	0.14

En este ejemplo la red SLFN clasificadora utiliza $n_O = 4$ neuronas en su capa oculta, producto de aplicar la regla de los hiperplanos. Se puede observar como el desempeño de la red MLP clasificadora es similar, comparable al el clasificador kNN. La Tabla 5.1 muestra el comportamiento de la técnica de la regla de los hiperplanos, en comparación con las reglas de oro, la del promedio, y la de los dos tercios, y con el método constructivo, en lo que a establecer el tamaño de la capa oculta se refiere. Es notorio que por un lado, la regla de los hiperplanos y el método constructivo obtienen el mismo resultado en cuanto a exactitud, sin embargo su tiempo para la obtención del tamaño de la capa oculta es mayor al de las reglas de oro, esta comparación aparece en la Tabla 5.2.

5.2.1.2. Ejemplo 5.2.1b

Este problema de clasificación se mostró en el segundo ejemplo de las secciones 4,3 y 4,4. Es también un problema de clasificación binario. Aquí las clases C_1 contiene 600 muestras y la clase C_2 contiene 1200, en $\mathbb{R}^2$. Las clases aunque son separables muestran un muy claro desbalance entre ellas, siendo C_2 la clase mayoritaria. La Figura 5.2 muestra la distribución de estas dos clases en su espacio $\mathbb{R}^2$.

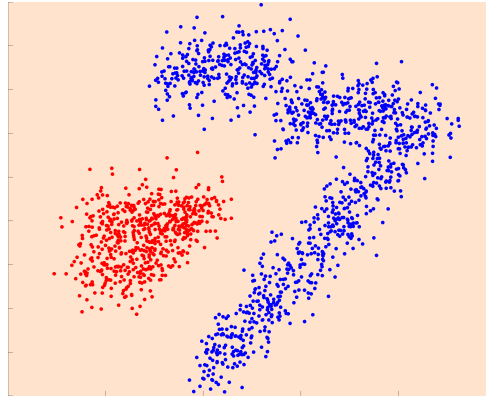


Figura 5.2: El problema de clasificación para el Ejemplo 5.2.1b

Este problema a pesar de presentar un claro desbalance, no es un problema para tres de los cuatro clasificadores basados en la red SLFN, como lo muestra la Tabla 5.3. En este caso el método constructivo supera al resto en cuanto al desempeño, aunque claramente es el que tardó más para obtener el tamaño de la capa oculta. La regla de los dos tercios y la regla de los hiperplanos tienen el mismo desempeño esta vez.

Tabla 5.3: Comparación entre métodos para obtener n_O en el Ejemplo 5.2.1b

Método	n_O	Tiempo	Exactitud
Promedio	2	0.001	85
Dos tercios	3	0.001	93
Constructivo	4	70	98
Hiperplanos	3	15	93

Tabla 5.4: Comparación de los métodos clasificadores en el Ejemplo 5.2.1b

Método	Exactitud	Tiempo
Hiperplanos	93	0.13
Bayesiano	88	0.12
kNN	95	0.16
Árboles	89	0.14

Por otro lado, en cuanto a los otros clasificadores, el clasificador kNN logra el mejor desempeño en lo que a exactitud se refiere. la regla de los hiperplanos tiene el segundo mejor desempeño. Esto se aprecia claramente en la Tabla 5.4.

5.2.1.3. Ejemplo 5.2.1c

Este es el tercer problema de clasificación presentado en las secciones 4,3 y 4,4. Es un problema de clasificación binaria en $\mathbb{R}^2$. Las clases C_1 que contiene 1700 muestras, y la clase C_2 que contiene $n_{P_2} = 2200$ de ellas. Las clases son separables con una superficie de decisión más bien compleja, como lo muestra la Figura 5.3.

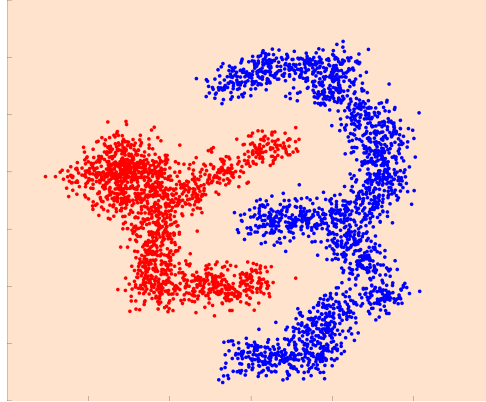


Figura 5.3: El problema de clasificación para el ejemplo 5.2.1c

En este problema entre los métodos para obtener el tamaño de la capa oculta, se tiene que los mejores métodos en cuanto a la exactitud de clasificación se logran con la regla de los hiperplanos, y con el método constructivo. En ambos casos toma mayor tiempo establecer tal valor, aunque significativamente menor con la regla de los hiperplanos. Esto lo muestra claramente la Tabla 5.5.

Tabla 5.5: Comparación entre métodos para obtener n_O en el Ejemplo 5.2.1c

Método	n_O	Tiempo	Exactitud
Promedio	2	0.001	85
Dos tercios	3	0.001	93
Constructivo	4	210	94
Hiperplanos	4	38	94

Tabla 5.6: Comparación de los métodos clasificadores en el Ejemplo 5.2.1c

Método	Exactitud	Tiempo
Hiperplanos	91	0.23
Bayesiano	83	0.2
kNN	93	0.37
Árboles	86	0.17

También como lo muestra la Tabla 5.6, el clasificador kNN tiene el mejor desempeño, apenas un poco mejor al obtenido por la red SLFN cuya capa oculta ha sido obtenida con la regla de los hiperplanos, su tiempo de ejecución sin embargo, ha sido mayor.

5.2.1.4. Ejemplo 5.2.1d

Este es en realidad el quinto problema presentado tanto en la sección 4.2 como en la sección 4.3. Es un problema de clasificación binario en $\mathbb{R}^3$. La clase C_1 cuenta con = 1200 muestras y la clase C_2 con 1400. Ambas clases son separables. La Figura 5.4 muestra la distribución de estas dos clases.

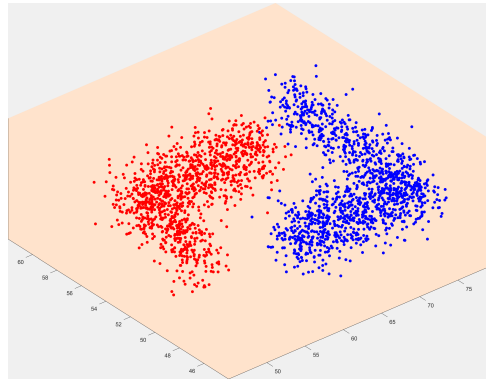


Figura 5.4: El problema de clasificación para el ejemplo 5.2.1d

La Tabla 5.7 muestra la comparación entre las cuatro técnicas usadas para establecer el tamaño de la capa oculta, es fácil ver como es el método constructivo quien tiene el mejor desempeño, apenas un poco mejor que el método de los hiperplanos, aunque el tiempo para obtener el tamaño de la capa oculta es evidentemente menor.

Tabla 5.7: Comparación entre métodos para obtener n_O en el ejemplo 5.2.1d

Método	n_O	Tiempo	Exactitud
Promedio	2	0.001	87
Dos tercios	3	0.001	87
Constructivo	6	265	96
Hiperplanos	4	56	94

Tabla 5.8: Comparación de los métodos clasificadores en el Ejemplo 5.2.1d

Método	Exactitud	Tiempo
Hiperplanos	94	0.28
Bayesiano	89	0.26
kNN	92	0.37
Árboles	90	0.20

En cuanto a la comparación con los otros clasificadores mostrada en la Tabla 5.8, se puede ver que en este caso el mejor desempeño lo obtuvo la red SLFN cuya capa oculta fue calculada con la regla de los hiperplanos. El clasificador kNN queda un poco más abajo, notando que en esta ocasión el desempeño de casi todos los clasificadores es parecido, no muy alejados en cuanto a desempeño, uno del otro.

5.2.1.5. Ejemplo 5.2.1e

Este es un ejemplo en $\mathbb{R}^2$ de un problema multiclase con $n_C = 3$, es el primer problema presentado en la sección 4.6. La clase C_1 contiene 700 muestras, la clase C_2 contiene también 700 muestras y la clase C_3 también contiene 700 muestras. Las clases son separables como lo muestra la Figura 5.5.

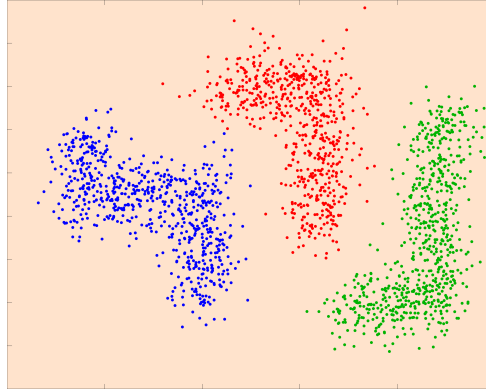


Figura 5.5: El problema de clasificación para el Ejemplo 5.2.1e

En la Tabla 5.9 se puede ver que el mejor desempeño lo obtiene la red SLFN cuya capa oculta fue calculada con un método constructivo. Aunque su mejor desempeño es apenas un poco mejor que para el método de la regla de los hiperplanos, misma que ha utilizado la regla OaO para llegar al tamaño de la capa oculta. A pesar de esto, el tiempo que le toma a la regla de los hiperplanos calcular el tamaño de la capa oculta es mucho menor.

Tabla 5.9: Comparación entre métodos para obtener n_O en el Ejemplo 5.2.1e

Método	n_O	Tiempo	Exactitud
Promedio	2	0.001	81
Dos tercios	4	0.001	85
Constructivo	8	750	97
Hiperplanos	6	120	95

Tabla 5.10: Comparación de los métodos clasificadores en el Ejemplo 5.2.1e

Método	Exactitud	Tiempo
Hiperplanos	95	0.28
Bayesiano	90	0.26
kNN	94	0.41
Árboles	92	0.20
SVM	92	0.30

En cuanto a la comparación en el desempeño de los otros cuatro métodos de clasificación utilizados que la red SLFN tiene el mejor desempeño de todos, mostrado en la Tabla 5.10 El clasificador kNN tiene un desempeño similar, aunque le toma más tiempo tiempo completar su tarea.

5.2.1.6. Ejemplo 5.2.1f

Este es el caso de un problema multiclase con $n_C = 3$, en $\mathbb{R}^3$. Es el segundo problema en la sección 4.6. La clase C_1 cuenta con 600 muestras, la clase C_2 cuenta con 800 muestras, y la clase C_3 cuenta con 1100 muestras. Las clases son separables como lo muestra la Figura 5.6. La clase C_1 aparece con muestras en color rojo, la clase C_2 con muestras en color azul, y la clase C_3 con muestras en color verde.

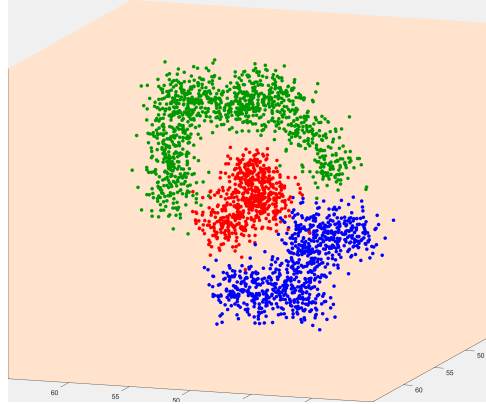


Figura 5.6: El problema de clasificación para el ejemplo 5.2.1f

Este problema de clasificación multiclase ha utilizado el esquema OaO para implementar la regla de los hiperplanos. La Tabla 5.11 muestra la comparación en el desempeño de las cuatro diferentes formas para calcular el tamaño de la capa oculta. Se aprecia nuevamente que el mejor desempeño lo obtiene el método constructivo, mismo que es apenas un poco mejor que el de la regla de los hiperplanos, aunque nuevamente el tiempo para obtener el tamaño de la capa oculta es mucho menor. Es obvio también que el tiempo que les ha tomado a estos dos métodos el calcular el valor n_O es bastante elevado en comparación con las reglas de oro.

Tabla 5.11: Comparación entre métodos para obtener n_O , en el Ejemplo 5.2.1f

Método	n_O	Tiempo	Exactitud
Promedio	3	0.001	76
Dos tercios	6	0.001	84
Constructivo	14	812	96
Hiperplanos	17	360	93

Tabla 5.12: Comparación de los métodos clasificadores en el Ejemplo 5.2.1f

Método	Exactitud	Tiempo
Hiperplanos	93	0.38
Bayesiano	92	0.52
kNN	94	0.46
Árboles	93	0.18
SVM	93	0.27

La Tabla 5.12 muestra el desempeño de los cinco modelos de clasificador considerados. Se puede ver que en este caso el clasificador kNN tiene el mejor desempeño, apenas por encima

de la máquina SVM y de la red SLFN, estas últimas tienen el mismo desempeño. En cuanto al clasificador diseñado con la regla de los hiperplanos, debe recordarse que la elevada cantidad obtenida de hiperplanos se debe en gran parte a que las clases permanecen muy cerca una de la otra, lo que hace que la tolerancia de ajuste para los hiperplanos con las muestras del promedio entre clases sea muy pequeña.

5.2.1.7. Ejemplo 5.2.1g

Este es el primer ejemplo de un problema no separable y se corresponde con el primer problema presentado en la sección 4.7. Es un problema de clasificación en $\mathbb{R}^2$ entre dos clases. La clase C_1 con 600 muestra y la clase C_2 con 600 muestras también. Estas dos clases no son separables y la clase C_1 contiene un empalme cercano al 46 %. Así que es un empalme moderado. La Figura 5.7 muestra la distribución de las clases, la clase C_1 aparece con sus muestras en color rojo y la clase C_2 aparece con sus muestras en color azul.

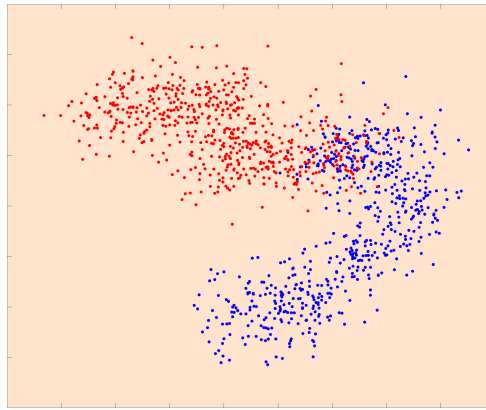


Figura 5.7: El problema de clasificación para el Ejemplo 5.2.1g

La Tabla 5.13 muestra la comparación entre los cuatro métodos considerados para obtener el tamaño de la capa oculta. Se puede apreciar una vez más como los métodos constructivos y de la regla de los hiperplanos son quienes mejor desempeño tienen en cuanto a la exactitud del clasificador final. Ambos métodos resuelven el mismo valor para n_O , el tamaño de la capa oculta.

Es claro, al observar la Tabla 5.14 que el clasificador kNN ha obtenido el mejor desempeño en lo que a exactitud del clasificador se refiere. Apenas un poco mejor que el clasificador SLFN que utiliza la regla de los hiperplanos para calcular el valor n_O .

Tabla 5.13: Comparación entre métodos para obtener n_O , en el Ejemplo 5.2.1g

Método	n_O	Tiempo	Exactitud
Promedio	2	0.001	81
Dos tercios	3	0.001	81
Constructivo	8	300	93
Hiperplanos	8	250	93

Tabla 5.14: Comparación de los métodos clasificadores en el Ejemplo 5.2.1g

Método	Exactitud	Tiempo
Hiperplanos	93	0.38
Bayesiano	88	0.46
kNN	94	0.40
Árboles	85	0.18
SVM	91	0.27

5.2.1.8. Ejemplo 5.2.1h

Este es el segundo problema en la sección 4.7. Es un ejemplo de clasificación binario en $\mathbb{R}^2$, en donde las clases involucradas C_1 y C_2 tienen la misma cantidad de muestras, que son 600 de ellas. Las dos clases se encuentran moderadamente empalmadas pues en el peor de los casos, el de la clase C_1 tiene cerca del 36 % de sus muestras como conflictivas. La distribución de ambas clases se muestra en la Figura 5.8, la clase C_1 aparece con muestras en color rojo y la clase C_2 con muestras en color azul.

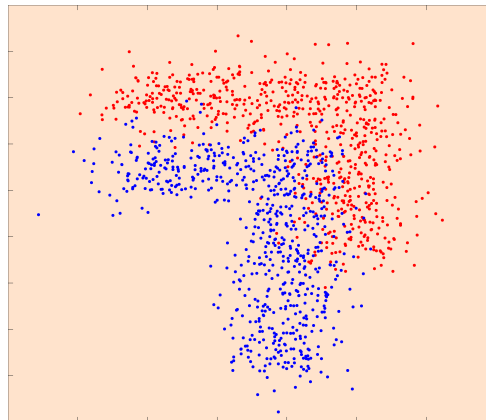


Figura 5.8: El problema de clasificación para el ejemplo 5.2.1h

Con las diferentes técnicas para obtener la cantidad n_O , las redes resultantes tienen el desempeño mostrado en la Tabla 5.15. Una vez más el mejor desempeño lo tiene el método constructivo, y no muy lejos se encuentra el método de la regla de los hiperplanos.

La Tabla 5.16 permite apreciar como es el clasificador kNN el que mejores resultados obtiene en este ejemplo. El clasificador SLFN y la máquina SVM obtienen un resultado equivalente apenas un poco menor al del clasificador kNN.

V. Validación

Tabla 5.15: Comparación entre métodos para obtener n_O en el Ejemplo 5.2.1h

Método	n_O	Tiempo	Exactitud
Promedio	2	0.001	81
Dos tercios	3	0.001	82
Constructivo	11	320	91
Hiperplanos	15	220	89

Tabla 5.16: Comparación de los métodos clasificadores en el Ejemplo 5.2.1h

Método	Exactitud	Tiempo
Hiperplanos	90	0.32
Bayesiano	85	0.46
kNN	92	0.32
Árboles	85	0.20
SVM	90	0.30

5.2.1.9. Ejemplo 5.2.1i

Este es un problema de clasificación entre dos clases en $\mathbb{R}^3$. Es el tercer problema en la sección 4.7. Las dos clases involucradas, C_1 y C_2 , se encuentran severamente empalmadas, pues cerca del 59% de las muestras en C_2 son conflictivas. La Figura 5.9 muestra la disposición de estas dos clases, en color rojo las muestras de C_1 , en color azul las muestras de la clase C_2 . Ambas clases contienen la misma cantidad de muestras, 500 de ellas.

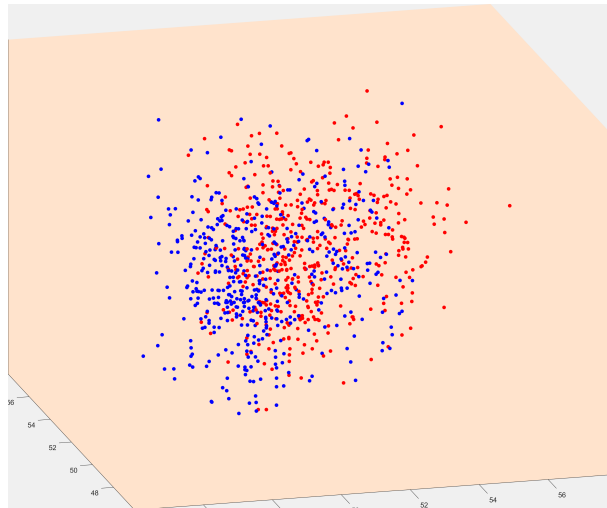


Figura 5.9: El problema de clasificación, para el Ejemplo 5.2.1i

Este problema de clasificación con clases empalmadas se somete primero a la prueba entre metodologías para establecer el tamaño de la capa oculta de una red SLFN, la cantidad n_O . Los resultados los muestra la Tabla 5.17. Se aprecia una vez más como el método constructivo obtiene el resultado con mejor exactitud, el método de la regla de los hiperplanos ha obtenido una capa oculta mas grande puesto que las clases involucradas incluida la clase conflicto, está muy cercana al resto de las clases.

La Tabla 5.18 muestra la segunda comparación, la de los cinco modelos de clasificador en

Tabla 5.17: Comparación entre métodos para obtener n_O , en el Ejemplo 5.2.1i

Método	n_O	Tiempo	Exactitud
Promedio	3	0.001	72
Dos tercios	6	0.001	82
Constructivo	13	346	91
Hiperplanos	18	250	87

Tabla 5.18: Comparación de los métodos clasificadores en el ejemplo 5.2.1i

Método	Exactitud	Tiempo
Hiperplanos	87	0.30
Bayesiano	80	0.38
kNN	92	0.32
Árboles	85	0.20
SVM	88	0.32

competencia. Se puede apreciar que el clasificador kNN tiene un desempeño apenas mejor que el de la red SLFN y que el de la máquina SVM.

5.2.2. Ejemplos de aplicación sobre datasets reales

Los ejemplos que a continuación se presentan corresponden a un *dataset real*, información recopilada sobre un proceso real, son datos generados durante el muestreo y recolección de información de un autor o autores que seguían un suceso específico y que decidieron hacer públicos su información recopilada. En la exposición de los siguientes problemas de clasificación, la distribución de las clases en su espacio no será ilustrada, al ser la mayoría de estos problemas que ocurren en espacios mayores al tridimensional, lo que dificulta en gran medida el contar con una ilustración intuitiva de las clases en su espacio.

5.2.2.1. El dataset TAE

El dataset TAE, siglas en inglés de *Teacher Assistant Evaluation dataset* [105], es un problema de clasificación en $\mathbb{R}^5$. Este dataset contiene las evaluaciones de $n_P = 150$ maestros o asistentes de maestro, tomadas en el departamento de estadística de la universidad de Winsconsin-Madison. Este dataset representa un problema de clasificación multiclase formado por las muestras de tres clases, la clase C_1 de los asistentes de maestro con buen desempeño, la clase C_2 fr los asistentes de maestro con desempeño promedio, y la clase C_3 de los asistentes de bajo desempeño. La clase C_1 contiene 49 muestras, la clase C_2 contiene 52 muestras y la clase C_3 contiene 49 muestras. Este es un problema de clasificación con un empalme moderado, pues la clase C_2 contiene cerca del 48 % de sus muestras en la zona de conflicto.

Al observar la Tabla 5.19 se puede concluir que tanto el método constructivo como la regla de los hiperplanos construyen una red SLFN con el mejor desempeño posible. En lo que a la comparación del método de la regla de los hiperplanos para producir la red SLF con otros modelos de clasificador, los resultados se muestran en la Tabla 5.20. Es evidente que en este problema

Tabla 5.19: Comparación entre métodos para obtener n_O , en el problema del dataset TAE

Método	n_O	Tiempo	Exactitud
Promedio	4	0.001	61
Dos tercios	6	0.001	65
Constructivo	8	364	83
Hiperplanos	8	290	83

Tabla 5.20: Comparación de los métodos clasificadores en el problema del dataset TAE

Método	Exactitud	Tiempo
Hiperplanos	83	0.30
Bayesiano	96	0.38
kNN	93	0.38
Árboles	80	0.20
SVM	58	0.36

ha sido el clasificador bayesiano quien el mejor desempeño logra. El clasificador SLFN tiene un desempeño más bajo pero aún respetable.

5.2.2.2. El dataset Crabs

Este es un dataset en $\mathbb{R}^6$. Es un problema de clasificación binario cuyas $n_P = 100$ muestras se corresponden con dos subespecies de la especie *Leptograpsus*, un cangrejo tropical [106]. Las dos clases involucradas son por una lado, la clase C_1 de la especie azul del *Leptograpsus*, y la clase C_2 de la especie anaranjada. Cada clase queda representada por 50 muestras. Este es un problema de clasificación moderadamente empalmado pues cerca del 47 % de las muestras en C_2 que es la clase con más muestras conflictivas.

Tabla 5.21: Comparación entre métodos para obtener n_O , en el problema del dataset Crabs

Método	n_O	Tiempo	Exactitud
Promedio	4	0.001	61
Dos tercios	6	0.001	65
Constructivo	8	370	83
Hiperplanos	8	306	83

Tabla 5.22: Comparación de los métodos clasificadores en el problema del dataset Crabs

Método	Exactitud	Tiempo
Hiperplanos	83	0.30
Bayesiano	86	0.38
kNN	93	0.38
Árboles	50	0.4
SVM	58	0.36

La Tabla 5.22 muestra la comparación entre los cinco modelos de clasificador considerados. Es interesante como el clasificador kNN tiene un desempeño que supera por mucho al resto de los clasificadores, si bien el clasificador SLFN muestra aún un desempeño superior al del clasificador basado en árboles de decisión y al clasificador SVM. La Tabla 5.21 muestra el desempeño del clasificador SLFN con una capa oculta calculada con cuatro diferentes métodos. Nuevamente se observa como el método constructivo y la regla de los hiperplanos obtienen el mismo resultado.

5.2.2.3. El dataset Iris

Iris es quizá el dataset más popular y conocido en el ambiente de los clasificadores de patrones. Su origen se remonta a la década de 1930, cuando Ronald Fisher, un biólogo y estadista inglés lo creó como herramienta para un estudio de análisis de discriminantes lineales [107]. Es un dataset que contiene la recopilación de $n_P = 150$ muestras en $\mathbb{R}^4$, sobre tres especies o clases de la flor iris. La clase C_1 contiene 50 muestras de la especie iris Setosa, la clase C_2 contiene 50 muestras de la especie iris Versicolor, y la clase C_3 contiene 50 muestras de la especie Iris Virginica. La versión que en este estudio se presenta es una versión modificada a la original, es la versión más común actualmente y se debe a [108].

El dataset Iris es un problema de clasificación multiclase severamente empalmado, pues la clase Iris Versicolor, es decir la clase C_2 , tiene cerca del 70 % de sus muestras en la zona de empalme. Comparando las cuatro diferentes técnicas para encontrar el valor n_O , el método constructivo resulta el mejor de todos, la regla de los hiperplanos le sigue, aunque su desempeño no es muy similar al del método constructivo como lo muestra la Tabla 5.23. Por otro lado, en cuanto al desempeño de los cinco modelos de clasificador, tanto la regla de los hiperplanos como el método de los árboles de decisión se quedan atrás de los otros tres modelos. Ahora, el mejor desempeño lo tiene una vez más el clasificador kNN como lo ilustra la Tabla 5.24.

Tabla 5.23: Comparación entre métodos para obtener n_O , en el problema del dataset Iris

Método	n_O	Tiempo	Exactitud
Promedio	3	0.001	60
Dos tercios	6	0.001	65
Constructivo	12	340	93
Hiperplanos	9	250	85

Tabla 5.24: Comparación de los métodos clasificadores en el problema del dataset Iris

Método	Exactitud	Tiempo
Hiperplanos	85	0.40
Bayesiano	93	0.85
kNN	98	0.62
Árboles	54	0.6
SVM	93	0.50

Una explicación plausible porque la red SLFN clasificadora tiene en este ejemplo un desempeño relativamente bajo, se debe a que en este ejemplo, el empalme es muy severo, por lo que el esquema del mezclado aún con su clasificador de segunda etapa no es capaz de recuperar la información completa, el aislamiento de la zona de conflicto se ve comprometido cuando una de las clases, en este caso la clase C_2 , casi desaparece por completo.

5.2.2.4. El dataset Haberman's Survival Dataset

Este es un dataset en $\mathbb{R}^4$. Contiene la información histórica acerca del tiempo de supervivencia al tratamiento de cáncer de mama en diferentes mujeres sometidas a una cirugía para tal

padecimiento en un periodo de tiempo que abarca de 1958 a 1970, en el hospital Billings de la Universidad de Chicago [108]. El problema es de clasificación binaria, siendo la clase C_1 , la de mujeres que no han sobrevivido más de cinco años, y la clase C_2 la de mujeres que sobrevivieron a los cinco años después de la operación. La clase C_1 está compuesta por 212 muestras y la clase C_2 por 357 muestras, por lo que es un problema desbalanceado, siendo C_2 la clase mayoritaria. El problema además tiene un empalme severo, pues cerca del 75 % de C_1 son muestras conflictivas.

La Tabla 5.25 muestra la comparación entre las cuatro técnicas para obtener el tamaño de la capa oculta. Se puede apreciar que la red SLFN construida usando la regla de los dos tercios y la regla de los hiperplanos tienen un comportamiento equivalente, aunque detrás del desempeño de la red SLFN construida mediante el método constructivo. La Tabla 5.26 muestra que excepto por el clasificador basado en árboles de decisión, el resto de ellos tiene un desempeño muy efectivo, si bien el clasificador kNN sigue superándolos a todos.

Tabla 5.25: Comparación entre métodos para obtener n_O , en el problema del dataset *Haberman's Survival*

Método	n_O	Tiempo	Exactitud
Promedio	3	0.001	90
Dos tercios	4	0.001	92
Constructivo	7	210	95
Hiperplanos	5	140	92

Tabla 5.26: Comparación de los métodos clasificadores en el problema del dataset *Haberman's Survival*

Método	Exactitud	Tiempo
Hiperplanos	92	0.32
Bayesiano	95	0.65
kNN	95	0.62
Árboles	12	0.67
SVM	92	0.32

5.2.2.5. El dataset Paychecks

Este también es un problema de clasificación binaria en $\mathbb{R}^4$, es un problema de clasificación entre las clases C_1 que contiene muestras de cheques falsos, y la clase C_2 que contiene muestras de cheques legítimos [109]. La clase C_1 contiene 600 muestras de cheques falsos y la clase C_2 contiene 732 muestras de cheques legítimos. El problema de clasificación en este dataset es uno moderadamente empalmado pues cerca del 32 % de las muestras en C_1 se localizan en la zona de conflicto del problema.

La Tabla 5.27 muestra la comparación en el desempeño de los diferentes métodos para calcular el tamaño de la red SLFN clasificadora en el problema actual. Se puede apreciar que en este caso, prácticamente cualquiera de las cuatro metodologías es efectiva, siendo como era de esperar, el método constructivo el que mejores resultados ha obtenido, seguido del método de la regla de los hiperplanos. Además con este método se obtiene la cantidad de neuronas más

Tabla 5.27: Comparación entre métodos para obtener n_O , en el problema del dataset Pay-checks

Método	n_O	Tiempo	Exactitud
Promedio	3	0.001	96
Dos tercios	5	0.001	97
Constructivo	12	210	100
Hiperplanos	9	130	99

Tabla 5.28: Comparación de los métodos clasificadores en el problema del dataset Pay-checks

Método	Exactitud	Tiempo
Hiperplanos	99	0.4
Bayesiano	91	0.45
kNN	100	0.5
Árboles	72	0.6
SVM	94	0.35

cercana a la del método constructivo. En la comparación de los cinco modelos de clasificador en este estudio, se puede apreciar como el clasificador kNN y la red SLFN cuya capa oculta fue construida con la regla de los hiperplanos los que mejores resultados obtuvieron.

5.2.2.6. El dataset Ionosphere

Este es un problema de clasificación binaria en $\mathbb{R}^{34}$. Aquí el problema es clasificar las señales que un arreglo de radares reciben, al hacer rebotar señales de origen sobre ciertas zonas en la ionosfera de la tierra. Es un problema de clasificación binaria entre dos categorías, la clase C_1 con señales que rebotan en la ionosfera de la tierra y regresan al radar, y la clase C_2 con señales que no rebotan en la ionosfera, sino que pasan a través de ella sin regresar al radar. La versión del dataset Ionosphere que aquí se utiliza es la referenciada en [108]. En este problema la clase C_1 está compuesta por 225 muestras y la clase C_2 por 126 muestras, se puede ver que es otro problema de clasificación desbalanceado. Una particularidad de este problema es que es linealmente separable, así que no contiene empalme entre clases.

Tabla 5.29: Comparación entre métodos para obtener n_O , en el problema del dataset Ionosphere

Método	n_O	Tiempo	Exactitud
Promedio	19	0.001	98
Dos tercios	22	0.001	98
Constructivo	1	130	100
Hiperplanos	1	110	100

Tabla 5.30: Comparación de los métodos clasificadores en el problema del dataset Ionosphere

Método	Exactitud	Tiempo
Hiperplanos	100	0.4
Bayesiano	97	0.37
kNN	100	0.9
Árboles	95	0.5
SVM	94	0.32

Al ser un problema linealmente separable, el método propuesto calculó la cantidad de hiperplanos necesarios para separar las clases C_1 y C_2 en sólo $n_H = 1$ hiperplanos. Compárese esto con la enorme cantidad de hiperplanos propuestos por las reglas de oro, esto debido que no

analizan las condiciones del problema, sino que sólo aplican una fórmula. Lo anterior implica que la red SLFN debería tener $n_O = n_H = 1$ neuronas en su única capa oculta y aunque podría usarse tal red SLFN y entrenarse, debido a que el problema es linealmente separable es preferible considerar una red perceptrón en lugar de una red SLFN con una neurona oculta. En la Tabla 5.29 en el renglón del clasificador SLFN se puede ver una sola neurona oculta, aunque en realidad una red perceptrón ha sido utilizada. Puede verse que casi todos los métodos de clasificación obtienen excelentes resultados, aunque el perceptrón y el clasificador kNN obtuvieron los mejores.

5.2.2.7. El dataset Wine

Este es un dataset en $\mathbb{R}^{13}$. Es un problema de clasificación multiclase pues $n_C = 3$. En este problema se debe separar a las muestras cuyos atributos representan la composición química de un vino, entre tres clases que denotan el viñedo de donde provienen los vinos de la misma región del norte de Italia. La clase C_1 denota a la región 1, la clase C_2 a la región 2, y la clase C_3 a la región 3. Para la clase C_1 se tienen 59 muestras, para la clase C_2 se tienen 71 muestras, y para la clase C_3 se cuenta con 48 muestras. Las clases presentan un empalme leve, pues cerca del 5 % de las muestras en C_2 , la clase con más muestras en conflicto, son muestras conflictivas. Podría considerarse que este problema tiene sus clases casi separables entre ellas, aún más, casi son linealmente separables entre si. La versión del dataset Wine aquí presentada es la versión en [108].

Tabla 5.31: Comparación entre métodos para obtener n_O , en el problema del dataset Wines

Método	n_O	Tiempo	Exactitud
Promedio	8	0.001	93
Dos tercios	11	0.001	94
Constructivo	3	170	97
Hiperplanos	3	100	97

Tabla 5.32: Comparación de los métodos clasificadores en el problema del dataset Wines

Método	Exactitud	Tiempo
Hiper.	97	0.4
Bayesiano	97	0.32
kNN	97	0.8
Árboles	97	4
SVM	94	0.26

La Tabla 5.31 muestra la comparación entre las cuatro metodologías para calcular el tamaño de la capa oculta de la red SLFN clasificadora. Se puede ver una vez más, como el método de la regla de los hiperplanos detecta perfectamente el hecho de que las clases involucradas son virtualmente separables, y la cantidad de hiperplanos detectados son la mínima cantidad necesaria en tal caso. Compárese esto con el tamaño calculado por las reglas de oro, que al sólo aplicar una fórmula, no perciben el hecho de que las clases son casi linealmente separables, y producen una capa oculta muy grande. La Tabla 5.32 muestra como cualquiera de los modelos

V. Validación

se desarrolla considerablemente bien en este problema. En particular la máquina SVM tienen un ligero peor desempeño.

Capítulo 6

Conclusiones

En esta investigación se ha presentado *un procedimiento para determinar la cantidad de hiperplanos que separan a un grupo de clases* en un problema de clasificación multiclase y con ello, *establecer el tamaño de la capa oculta de una red MLP con una sola capa de neuronas ocultas, o una red SLFN*. Esta metodología utiliza únicamente los patrones o muestras que son parte del conjunto de entrenamiento. Además se han propuesto los procedimientos que permiten aplicar a la regla de los hiperplanos en problemas de clasificación multiclase y con clases empalmadas. Y todo lo anterior usa como principal herramienta la geometría computacional, describiendo el problema de clasificación original a través de un grupo de descriptores geométricos como los hiperplanos, ajuste de hiperplanos y formas geométricas principalmente. Lo más importante de todo esto es que *esta metodología funciona en $\mathbb{R}^{n_E}$* , siendo n_E las dimensiones del problema y siendo este un parámetro arbitrario.

6.1. Resultados

De todo lo presentado e ilustrado en esta investigación se demuestra que:

- (1) Se puede representar al conjunto de muestras en términos de una clase como una descripción geométrica acerca de la forma de dicha clase, una *aproximación a la forma alfa*.
- (2) Se puede deducir a partir de la información de entrenamiento, *la mínima cantidad de hiperplanos que separan* a dos clases separables.
- (3) La mínima cantidad de hiperplanos separables se deduce al *ajustar una mínima cantidad de hiperplanos a las muestras del promedio entre clases*, que representa la superficie de decisión entre clases.

VI. Conclusiones

- (4) El promedio entre clases se obtiene de *promediar los bordes fundamentales* es decir, las muestras de entrenamiento en las orillas de la nube de puntos que representan a cada clase.

Para aproximar la forma alfa en un espacio $\mathbb{R}^{n_E}$ y recuperar los bordes fundamentales, una tarea primordial de la presente investigación, es necesario que:

- (1) La *forma alfa de la clase sea aproximada* a partir de la unión de k particiones.
- (2) El valor k que particiona a la clase se deduce a partir de un análisis de agrupamientos.
- (3) Los bordes fundamentales son los vértices en un agrupamiento, siempre que la línea que una a este vértice con su vértice más cercano en la otra clase no intersecte la forma aproximada de su respectiva clase.

Así, el promedio entre clases que representa la superficie de decisión entre ellas, se obtiene simplemente como el promedio entre los bordes fundamentales de cada clase. Ahora para ajustar la mínima cantidad de hiperplanos sobre las muestras en $\mathbb{R}^{n_E}$ del promedio entre clases, se compilan todas las muestras que se ajusten a un hiperplano formado con otras de las muestras en el mismo conjunto, usando una medida de tolerancia que se obtiene con la mínima distancia entre las clases involucradas, repitiendo este proceso una y otra vez hasta que ya no queden muestras del promedio sin haber sido ajustadas.

Para la generalización del esquema antes presentado a un problema de clasificación multiclase en un espacio $\mathbb{R}^{n_E}$, se adapta el esquema de clasificación multiclase One Against One OaO, de la siguiente manera:

- (1) *Separar por pares diferentes*, las clases involucradas en el problema.
- (2) Con el método descrito anteriormente *encontrar la mínima cantidad de hiperplanos que separan a cada par de clases diferentes*.
- (3) *Se acumula esta cantidad de hiperplanos en una cantidad global de hiperplanos*, siendo esta la cantidad de hiperplanos que separan a las clases involucradas.

Para aplicar el método aquí propuesto en problemas con clases empalmadas o traslapadas, es necesario una etapa de *pre procesamiento de la información de entrenamiento*, de modo que las *muestras conflictivas queden aisladas en una nueva clase*, la clase conflicto. Una vez aislada la clase conflicto la clasificación ocurre usando uno de tres esquemas:

- (1) El *esquema del descarte*: aplicable cuando el *empalme es leve*, entre clases.
- (2) El *esquema del mezclado*: aplicable cuando el *empalme es moderado o severo*.

(3) El *esquema del separado*: aplicable cuando el *empalme es moderado o severo*.

El grado de empalme entre clases es *leve cuando la clase con mayor cantidad de muestras conflictivas en el problema no supera el 20 %*, es *moderado cuando la cantidad de muestras conflictivas en la clase con la mayor cantidad de ellas no supera el 50 %*, y es *severo cuando supera el 50 %*. Lo anterior implica analizar cuantas muestras en cada clase se localizan adentro de la forma aproximada de las otras clases, lo que se logra utilizando una vez más geometría computacional, en problemas en $\mathbb{R}^{n_E}$, para $n_E \geq 2$.

El uso de los esquemas anteriores para manejo de problemas con clases empalmadas implica que el clasificador final resulte diferente de lo que originalmente se contempla: el esquema del descarte puede funcionar usando únicamente la red SLFN cuya capa oculta ha sido diseñada usando la regla de los hiperplanos con el método aquí propuesto. Sin embargo los esquemas del mezclado y del separado exigen que el clasificador final sea compuesto por más de un clasificador. En este estudio se ha decidido por el *esquema del mezclado en caso de que las clases muestren un empale moderado o severo*. El esquema del mezclado exige que la *clasificación ocurra en dos etapas: la primera clasifica las muestras en sus clases respectivas incluyendo a la clase conflictiva como una clase adicional al resto de las clases sin muestras conflictivas, esta primera etapa es la que usa a la red SLFN cuya capa oculta fue diseñada con el método aquí propuesto. La segunda etapa sólo se encarga de resolver los casos de muestras en la zona conflictiva para lo que usa un clasificador kNN*.

El método propuesto es aplicable en problemas de clasificación con clases separables y no separables, problemas de clasificación binaria y clasificación multiclase, y lo que es mas importante, *en cualquier número de dimensiones*, es decir, en problemas en $\mathbb{R}^{n_E}$ para $n_E \geq 2$.

6.2. Condiciones de uso

El método propuesto tiene aún ciertas limitantes o casos de uso que deben ser tomadas en cuenta. Por ejemplo:

- (1) La metodología funciona mejor en *clases autocontenidas*, es decir, que no existan agrupamientos de datos de la misma clase separados, por los datos de otra clase. Esto obliga a la metodología propuesta a tomar medidas muy particulares que eliminarían la generalidad con la que la regla de los hiperplanos funciona, posiblemente, generar clases artificiales adicionales para esa misma clase, dependiendo de los agrupamientos separados que existan.

- (2) El estado actual de las cosas, en la geometría computacional aún se tienen enormes limitaciones, en particular, en la disponibilidad de librerías o paquetes de geometría computacional, pues, en la mayoría de los casos, funcionan confiablemente sólo en problemas con dimensiones pequeñas, en dos y tres dimensiones, específicamente, por lo que este tipo de operaciones deben evitarse por completo. En su lugar, se debe limitar a usar funcionalidades que, al menos en principio, si se pueden aplicar en mayor número de dimensiones. Por ejemplo, aproximar las formas de las clases usando sólo envolventes convexas, en lugar de formas alfa. Debido a lo anterior, y en la medida de lo posible, se aconseja no utilizar las librerías como Qhull y CGAL, incluso para el cálculo de las envolventes convexas, y recurrir a otros esquemas para su implementación, por ejemplo, el uso de programación lineal y de optimización convexa.
- (3) Otro aspecto que se encuentra limitado, si se utilizan las librerías Qhull y CGAL, es la aproximación a la forma de una clase. Actualmente esto se logra uniendo una o más envolventes convexas, una por cada subgrupos de datos en que se divide la clase original, pero, incluso el cálculo de las envolventes convexas queda limitado para dimensiones mayores. Esto afecta, al final, a la aproximación a la forma alfa.
- (4) La cantidad de subgrupos en los que se debe dividir a las muestras de una clase, es un valor muy ambiguo, y no existe una forma de establecerlo de forma definitiva. Incluso recurriendo a procedimientos de optimización convexa para aproximar las envolventes convexas, como se hizo en la presente investigación. Experimentalmente, lo único que se puede afirmar, es que este valor debe mantenerse pequeño, especialmente cuando se usan librerías como Qhull y CGAL.

6.3. Trabajo a futuro

El uso de la regla de los hiperplanos con el método propuesto ha mostrado ser muy útil para considerar las condiciones locales de un problema de clasificación. En el estado actual de la implementación sin embargo, su mayor desventaja es su dependencia de las directivas y funcionalidades que las librerías de geometría computacional actuales. Por lo tanto algunas mejoras se podrían lograr si:

- (1) La forma alfa de un grupo de datos, en dimensiones mayores, se pudiera representar, o aproximar, como la unión de formas alfa en dimensiones más pequeñas, por ejemplo, formas alfa en dos dimensiones. Esto parece factible, pero un análisis más profundo de esto,

VI. Conclusiones

y las condiciones en que esto puede ocurrir, quedan fuera del alcance de la presente investigación.

- (2) El uso de una segunda capa oculta. Las redes MLP con una sola capa oculta pueden, en principio, aproximar cualquier función real, pero, es un hecho, que una segunda capa oculta es de gran ayuda en muchos sentidos, uno de ellos, es que permitiría al enfoque propuesto manejar los casos de clases no autocontenidas, sólo habría que establecer las condiciones en que esto puede ocurrir.
- (3) La cantidad de particiones o subgrupos en los que las muestras de una clase se dividen, debe ser refinado, debería desarrollarse una metodología, que, usando únicamente los datos de entrenamiento, arrojará un valor confiable para ellos, usando las estadísticas apropiadas. Si bien en este estudio se propone una metodología para hacerlo, dista mucho de ser la óptima.
- (4) Muchos de los problemas que actualmente se enfrentan, con respecto a la aproximación de la forma de una clase, se pueden evitar si la forma de clase ni se aproxima con envolventes convexas, o con la unión de formas alfa más simples, es decir, ni siquiera debería intentar aproximarse la forma de una clase. Utilizando una forma de clasificador, conocido como clasificador de una sola clase. Algo muy útil en un clasificador de una sola clase, sería el identificar, en un conjunto de entrenamiento, cuáles muestras pertenecen a tal clase, y cuáles no. Una forma de implementarlos, es una forma muy particular de máquina SVM, conocida como máquina SVDD. La gran ventaja de usar una máquina SVDD, es que, usando el kernel correcto, luego de ser entrenada, arroja en sus vectores de soporte las muestras en las orillas de la nube de puntos de la clase, es decir, arroja los bordes de la clase. Esto puede ser aprovechado para representar la forma de la clase, usando los vectores soporte que arroja la máquina SVDD.

Capítulo 7

Referencias

- [1] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural Network Design*. Boston, PWS Publishing Company, 2014.
- [2] R. Majalca and P. Acosta, “Clasificador Multicategoría Usando Redes MLP Con Una Sola Capa Oculta,” *Tecnociencia*, vol. 9, no. 3, pp. 148–159, 2018.
- [3] A. B. Siddiquee, M. E. H. Mazumder, and S. M. Kamruzzaman, “A constructive algorithm for feedforward neural networks for medical diagnostic reasoning,” *ArXiv*, vol. abs/1009.4564, 2010.
- [4] J. L. Subirats, L. Franco, and J. M. Jerez, “C-Mantec: A novel constructive neural network algorithm incorporating competition between neurons,” *Neural Networks*, vol. 26, pp. 130–140, 2012.
- [5] L. Jing, C. Ji-hang, S. Jing-yuan, H. Fei, J.-Y. Shi, H. Fei, S. Jing-yuan, and H. Fei, “Brief introduction of backpropagation (BP) neural network algorithm and its improvement,” *Advances in computer science and information engineering*, vol. 169, pp. 553–558, 2012.
- [6] K. G. Sheela and S. N. Deepa, “Review on methods to fix number of hidden neurons in neural networks,” *Mathematical Problems in Engineering*, vol. 2013, 2013.
- [7] R. Reed, “Pruning algorithms - a survey,” 1993.
- [8] Researchgate, “How to decide the number of hidden layers and nodes in a hidden layer?,” 2012.
- [9] S. Walczak and N. Cerpa, “Heuristic principles for the design of artificial neural networks,” *Information and Software Technology*, vol. 41, no. 2, pp. 107–117, 1999.

VII. Referencias

- [10] S. Belciug and F. Gorunescu, "Learning a single-hidden layer feedforward neural network using a rank correlation-based strategy with application to high dimensional gene expression and proteomic spectra datasets in cancer detection," *Journal of Biomedical Informatics*, vol. 83, no. July 2017, pp. 159–166, 2018.
- [11] K. Peng, S. S. Ge, and C. Wen, "An algorithm to determine neural network hidden layer size and weight coefficients," in *Proceedings of the 2000 IEEE International Symposium on Intelligent Control. Held jointly with the 8th IEEE Mediterranean Conference on Control and Automation (Cat. No.00CH37147)*, pp. 261–266, July 2000.
- [12] S. Chattopadhyay and G. Chattopadhyay, "Identification of the best hidden layer size for three-layered neural net in predicting monsoon rainfall in India," *Journal of Hydroinformatics*, vol. 10, no. 2, p. 181, 2008.
- [13] J. R. McDonnell and D. Waagen, "Determining neural network hidden layer size using evolutionary programming," p. 6, 10 1993.
- [14] D. Stathakis, "How many hidden layers and nodes?," *International Journal of Remote Sensing*, vol. 30, no. 8, pp. 2133–2147, 2009.
- [15] J. Cao and Z. Lin, "Extreme learning machines on high dimensional and large data applications: A survey," *Mathematical Problems in Engineering*, vol. 2015, 07 2015.
- [16] W. S. Sarle, "Neural Networks FAQ," 2000.
- [17] J. Heaton, "Programming neural networks in Java," *Btp://Www. Heatonresearch. Com*, 2004.
- [18] P. L. Narasimha, W. H. Delashmit, M. T. Manry, J. Li, and F. Maldonado, "An integrated growing-pruning method for feedforward network training," *Neurocomputing*, vol. 71, no. 13-15, pp. 2831–2847, 2008.
- [19] T. Masters, "Practical Neural Network Recipies in C++," 1993.
- [20] S. Trenn, "Multilayer perceptrons: Approximation order and necessary number of hidden units," *IEEE Transactions on Neural Networks*, vol. 19, no. 5, pp. 836–844, 2008.
- [21] T. Kavzoglu and P. M. Mather, "The use of backpropagating artificial neural networks in land cover classification," *International Journal of Remote Sensing*, vol. 24, no. 23, pp. 4907–4938, 2003.

VII. Referencias

- [22] S. Wang, C. Aggarwal, and H. Liu, “Using a Random Forest to Inspire a Neural Network and Improving on It,” *Proceedings of the Seventeenth SIAM International Conference on Data Mining*, no. January, pp. 1–9, 2017.
- [23] M. Girolami, “Principal Components Identify MLP Hidden Layer Size for Optimal Generalisation Performance,” in *Artificial Neural Nets and Genetic Algorithms*, 1998.
- [24] I. Ivanova and M. Kubat, “Initialization of neural networks by means of decision trees,” *Knowledge-Based Systems*, vol. 8, no. 6, pp. 333–344, 1995.
- [25] Quora, “How do I decide the number of nodes in a hidden layer of a neural network?,” 2017.
- [26] R. Setiono, “Feedforward neural network construction using cross validation,” *Neural Computation*, vol. 13, pp. 2865–2877, Dec 2001.
- [27] J. F. Khaw, B. Lim, and L. E. Lim, “Optimal design of neural networks using the Taguchi method,” *Neurocomputing*, vol. 7, no. 3, pp. 225–245, 1995.
- [28] G. E. Peterson, W. E. Bond, D. C. Clair, and S. R. Aylward, “Using Taguchi’s Method of Experimental Design to Control Errors in Layered Perceptrons,” *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 949–961, 1995.
- [29] M. Ettaouil and Y. Ghanou, “Neural architectures optimization and genetic algorithms,” *WSEAS Transactions on Computers*, vol. 8, no. 3, pp. 526–537, 2009.
- [30] M. Harvey, “Let’s evolve a neural network with a genetic algorithm—code included,” 2017.
- [31] F. Ahmadizar, K. Soltanian, F. Akhlaghiantab, and I. Tsoulos, “Artificial neural network development by means of a novel combination of grammatical evolution and genetic algorithm,” *Engineering Applications of Artificial Intelligence*, vol. 39, pp. 1–13, 2015.
- [32] G.-b. Huang, L. Chen, and C.-k. Siew, “Universal Approximation Using Incremental Constructive Feedforward Networks With Random Hidden Nodes,” vol. 17, no. 4, pp. 879–892, 2006.
- [33] G. B. Huang, “Learning capability and storage capacity of two-hidden-layer feedforward networks,” *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 274–281, 2003.

VII. Referencias

- [34] S. E. Fahlman and C. Lebiere, “The Cascade-Correlation Learning Architecture,” *Advances in neural information processing systems*, pp. 524–532, 1990.
- [35] T. Ash, “Dynamic Node Creation in Backpropagation Networks,” *Connection Science*, vol. 1, no. 4, pp. 365–375, 1989.
- [36] R. Setiono and L. C. Kwong Hui, “Use of a Quasi-Newton Method in a Feedforward Neural Network Construction Algorithm,” *IEEE Transactions on Neural Networks*, 1995.
- [37] M. Augasta and T. Kathirvalavakumar, “Pruning algorithms of neural networks—a comparative study,” *Central European Journal of ...*, vol. 3, no. 3, pp. 105–115, 2013.
- [38] P. Thomas and M.-C. Suhner, “A new multilayer perceptron pruning algorithm for classification and regression applications,” *Neural Process. Lett.*, vol. 42, pp. 437–458, Oct. 2015.
- [39] R. Setiono, “A penalty-function approach for pruning feedforward neural networks,” *Neural Comput.*, vol. 9, pp. 185–204, Jan. 1997.
- [40] W. Wan, S. Mabu, K. Shimada, K. Hirasawa, and J. Hu, “Enhancing the generalization ability of neural networks through controlling the hidden layers,” *Applied Soft Computing*, vol. 9, no. 1, pp. 404 – 414, 2009.
- [41] T. Q. Huynh and R. Setiono, “Effective neural network pruning using cross-validation,” in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 972–977 vol. 2, July 2005.
- [42] M. Hagiwara, “A simple and effective method for removal of hidden units and weights,” *Neurocomputing*, vol. 6, no. 2, pp. 207 – 218, 1994. Backpropagation, Part IV.
- [43] H. J. Xing and B. G. Hu, “Two-phase construction of multilayer perceptrons using information theory,” *IEEE Transactions on Neural Networks*, vol. 20, pp. 715–721, April 2009.
- [44] Z. Zhang and J. Qiao, “A node pruning algorithm for feedforward neural network based on neural complexity,” in *2010 International Conference on Intelligent Control and Information Processing*, pp. 406–410, Aug 2010.

VII. Referencias

- [45] D. Whitley and C. Bogart, "The evolution of connectivity: Pruning neural networks using genetic algorithms," in *Proceedings of the International Joint Conference on Neural Networks* (Washington, DC), pp. 134–137, IEEE Press, 1990.
- [46] B. Hassibi, D. G. Stork, G. J. Ivoff, S. Hill, and R. Suite, "Optimal Brain Surgeon and General Network Pruning," *IEEE International Conference on Neural Networks*, 1993.
- [47] Y. L. Cun, J. S. Denker, and S. a. Solla, "Optimal Brain Damage," *Advances in Neural Information Processing Systems*, vol. 2, no. 1, pp. 598–605, 1990.
- [48] L. M. Belue and K. W. Bauer, "Determining input features for multilayer perceptrons," *Neurocomputing*, vol. 7, no. 2, pp. 111 – 121, 1995.
- [49] M. Gethsiyal Augasta and T. Kathirvalavakumar, "A novel pruning algorithm for optimizing feedforward neural network of classification problems," *Neural Processing Letters*, vol. 34, no. 3, pp. 241–258, 2011.
- [50] J. P. Thivierge, F. Rivest, and T. R. Shultz, "A dual-phase technique for pruning constructive networks," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 1, pp. 559–564 vol.1, July 2003.
- [51] B. J. Briedis and T. D. Gedeon, "Using the Grow-And-Prune Network to Solve Problems of Large Dimensionality," 1998.
- [52] Z. Zhang, X. Ma, and Y. Yang, "Bounds on the number of hidden neurons in three-layer binary neural networks," *Neural Networks*, vol. 16, no. 7, pp. 995–1002, 2003.
- [53] K. Messer and J. Kittler, "Choosing an Optimal Neural Network Size to aid a Search Through a Large Image Database.," *Proceedings of the 9th British Machine Vision Conference*, pp. 235–244, 1998.
- [54] Researchgate, "What is the criteria to select optimal number of hidden layer neurons in MLP neural network ?," 2016.
- [55] Howard Schneider, *Master Machine Learning Algorithms Discover How They Work and Implement Them From Scratch*. 2016.
- [56] A. Fallis, *Deep Learning with Python*, vol. 53. 2013.
- [57] P. Kim, *MATLAB Deep Learning*. 2017.

VII. Referencias

- [58] J. Heaton, *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks*, vol. 3. 2015.
- [59] J. P. Mueller and L. Massaron, *Machine Learning for Dummies*, vol. 53. New Jersey: John Wiley & Sons Inc., 1st ed., 2013.
- [60] D. Conway and J. M. White, *Machine Learning for Hackers*. California: O'Reilly Media, first ed., 2012.
- [61] V. B. Rao, *C++ Neural Networks and Fuzzy Logic*. 1995.
- [62] B. M. del Brío and A. S. Molina, *Redes Neuronales y Sistemas Borrosos*. Madrid: Alfaomega, 3rd ed., 2007.
- [63] P. Isasi and I. M. Galván, *Redes de neuronas artificiales: un enfoque práctico*. Madrid: Pearson. Prentice Hall, 2004.
- [64] *Prepared exclusively for Ricardo Majalca*.
- [65] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [66] G. Cybenko, "Degree of approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 9, no. 3, pp. 303–314, 1989.
- [67] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [68] Reserachgate, "What are the advantages of using Artificial Neural Network compared to other approaches?," 2014.
- [69] Quora, "What are some advantages of using neural networks over decision trees?," 2014.
- [70] Deeplearningtrack, "Introduction to NEURAL NETWORKS, Advantages and Applications," 2017.
- [71] Stackoverflow, "SVM versus MLP (Neural Network): compared by performance and prediction accuracy," 2012.
- [72] Stackoverflow, "What are the advantages of Artificial Neural Networks over Support Vector Machines?," 2017.

VII. Referencias

- [73] Scikit Learn Developers, “Neural networks models (supervised),” 2017.
- [74] A. R. Webb and K. D. Copsey, *Statistical pattern recognition*. Wiley, 3rd ed., 2011.
- [75] P. Louridas and C. Ebert, *Machine Learning*. New Jersey: Apress, first ed., 2016.
- [76] R. O. Duda, P. E. Hart, and D. G. Stork, “Pattern classification,” 2001.
- [77] P. Honeine, Z. Noumir, and C. Richard, “Multiclass classification machines with the complexity of a single binary classifier,” *Signal Processing*, vol. 93, no. 5, pp. 1013–1026, 2013.
- [78] M. Aly and <. Edu>, “Survey on multiclass classification methods,” *Neural Netw*, no. November, pp. 1–9, 2005.
- [79] Wenyin Tang, K. Z. Mao, Lee Onn Mak, Gee Wah Ng, W. Tang, and K. Z. Mao, “Classification for overlapping classes using optimized overlapping region detection and soft decision,” *2010 13th International Conference on Information Fusion*, pp. 1–8, 2010.
- [80] J. Stefanowski, “Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data,” *Smart Innovation, Systems and Technologies*, vol. 13, pp. 277–306, 2013.
- [81] R. Rekha and R. Vidhyapriya, “Maximisation of arrhythmia classification accuracy by addressing class overlap and imbalance problem,” *International Journal of Biomedical Engineering and Technology*, vol. 26, no. 2, p. 197, 2018.
- [82] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, vol. 17. 2008.
- [83] Techopedia, “Computational Geometry,” 2018.
- [84] Geeksforgeeks, “Minimum cost polygon triangulation,” 2018.
- [85] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, “On the shape of a set of points,” *IEEE Transactions on information theory*, vol. 29, no. 4, pp. 551–559, 1983.
- [86] CGAL, “2D Alpha Shapes,” 2012.
- [87] H. Edelsbrunner and E. P. Mücke, “Three-dimensional alpha shapes,” *ACM Transactions on Graphics*, vol. 13, no. 1, pp. 43–72, 1994.

VII. Referencias

- [88] C. Varytimidis, K. Rapantzikos, and Y. Avrithis, “W α sh: Weighted α -shapes for local feature detection,” in *Computer Vision – ECCV 2012* (A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, eds.), (Berlin, Heidelberg), pp. 788–801, Springer Berlin Heidelberg, 2012.
- [89] M. Teichmann and M. Capps, “Surface reconstruction with anisotropic density-scaled alpha shapes,” in *Proceedings Visualization '98 (Cat. No.98CB36276)*, pp. 67–72, Oct 1998.
- [90] C. Varytimidis, K. Rapantzikos, Y. Avrithis, and S. Kollias, “A-Shapes for Local Feature Detection,” *Pattern Recognition*, vol. 50, pp. 56–73, 2016.
- [91] N. Amenta, S. Choi, and R. K. Kolluri, “The power crust, unions of balls, and the medial axis transform,” *Computational Geometry*, vol. 19, no. 2, pp. 127 – 153, 2001. Combinatorial Curves and Surfaces.
- [92] F. Cazals, T. Dreyfus, S. Sachdeva, and N. Shah, “Greedy geometric algorithms for collection of balls, with applications to geometric approximation and molecular coarse-graining,” *Comput. Graph. Forum*, vol. 33, pp. 1–17, Sept. 2014.
- [93] A. K. Formann and T. Kohlmann, “Latent class analysis in medical research,” *Statistical Methods in Medical Research*, vol. 5, no. 2, pp. 179–211, 1996. PMID: 8817797.
- [94] M. Wedel and W. Kamakura, *Market Segmentation: Conceptual and Methodological Foundations*, vol. 8. 01 2000.
- [95] Y. Abu-Mostafa, “Learning From Data,” 2017.
- [96] T. Calinski and J. Harabasz, “A dendrite method for cluster analysis,” *Communications in Statistics - Theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.
- [97] B. Boyd, V. Vandenberghe, and F. Faybusovich, *Convex Optimization*, vol. 51. 2006.
- [98] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, “Balancing Strategies and Class Overlapping,” *Advances in Intelligent Data Analysis VI*, pp. 24–35, 2005.
- [99] R. Kretschmar, N. Karayiannis, and F. Eggimann, “Feedforward neural network models for handling class overlap and class imbalance,” vol. 15, pp. 323–38, 11 2005.
- [100] N. Cristianini and B. Schölkopf, “Support Vector Machines and Kernel Methods: The New Generation of Learning Machines,” *Artificial Intelligence Magazine*, vol. 23, no. 3, pp. 31–42, 2002.

VII. Referencias

- [101] W. Tang, K. Mao, L. Mak, and G.-W. Ng, “Classification for overlapping classes using optimized overlapping region detection and soft decision,” pp. 1 – 8, 08 2010.
- [102] H. Xiong, J. Wu, and L. Liu, “Classification with Class Overlapping: A Systematic Study,” *The 2010 International Conference on E-Business Intelligence*, pp. 491–497, 2010.
- [103] A. Tomar, A. Nagpal, and A. P. J. Abdul, “Comparing Accuracy of K-Nearest-Neighbor and Support-Vector-Machines for Age Estimation,” *International Journal of Engineering Trends and Technology*, vol. 38, no. 6, pp. 326–329, 2016.
- [104] G. Casella, S. Fienberg, and I. Olkin, *Introduction to statistical learning with applications*. New York: Springer New York, 2015.
- [105] Wei-Yin Loh, “UCI Machine Learning Repository: Teaching Assistant Evaluation Data Set,” 1997.
- [106] N. A. Campbell and R. J. Mahon, “A multivariate study of variation in two species of rock crab of the Genus *Leptograpsus*,” *Australian Journal of Zoology*, vol. 22, no. 3, pp. 263–276, 1974.
- [107] R. Fishe, “The use of multiple measurements in taxonomic problems,” *Annals of Eugenics*, vol. 26, no. 1, 1936.
- [108] D. Dheeru and E. Karra Taniskidou, “{UCI} Machine Learning Repository,” 2017.
- [109] Volker Lohweg and Ostwestfalen-Lippe, “UCI Machine Learning Repository: banknote authentication Data Set,” 2013.