Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Estudios en Ensambles de Redes Neuronales.

Tesis que presenta

Eduardo Filemón Vázquez Santacruz

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica
opción Computación

Director de la Tesis

Dr. Debrup Chakraborty

México, D.F. **26 de Noviembre de 2007**

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología por los recursos otorgados que facilitaron los estudios de maestría. Al personal del Departamento de Computación del Centro de Investigación y de Estudios Avanzados del IPN por los conocimientos y hospitalidad ofrecidos. Al personal de la Biblioteca de Ingeniería Eléctrica por facilitar el material necesario durante los estudios de maestría.

Al Dr. Debrup Chakraborty, por su singular motivación, asesoría y enseñanza durante el trabajo de tesis realizado.

A los Drs. Francisco Rodríguez Henríquez y Carlos A. Coello Coello, por sus valiosos comentarios durante la revisión del documento de tesis.

En especial a mis padres, quienes han fomentado parte de la integridad que me ha forjado como un ser humano comprometido con su alrededor.

Gracias a la vida que implica la posibilidad de dudar, de pensar y de imaginar.

Resumen

Presupóngase un conjunto de datos $\mathcal{L} = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \ldots, n; \mathbf{x}_i \in \Re^p, \mathbf{y}_i \in \Re^s\}$, generado a partir de una distribución de probabilidad P desconocida. Existen varios métodos disponibles en la literatura para estimar la relación entrada-salida que está presente en los datos del conjunto $\mathcal{L}$. En la mayoría de los métodos una función de predicción ϕ se construye usando el conjunto $\mathcal{L}$ y optimizando cierto criterio. La propiedad más importante que la función ϕ debe tener es la capacidad de predecir las salidas de los datos (generados con la misma distribución P) que no están presentes en el conjunto de entrenamiento $\mathcal{L}$. Esta propiedad es llamada habilidad de *generalización* de la función de predicción. El desarrollo de sistemas que puedan predecir con buenas propiedades de generalización es un área de investigación activa y existen muchas propuestas para hacerlo. Una de las técnicas para realizar esta tarea consiste en construir múltiples clasificadores usando $\mathcal{L}$ y combinar las salidas de éstos para obtener la predicción final. Este tipo de métodos son denominados "métodos de ensamble".

Una de la estrategias comunes para construir un clasificador a partir de un conjunto de datos dado $\mathcal{L}$, consiste en entrenar una red neuronal de "pro-alimentación" (por ejemplo, un perceptrón multicapa (MLP)) usando $\mathcal{L}$. Es posible entrenar varias redes neuronales a partir de $\mathcal{L}$ y posteriormente crear un ensamble con ellas. En esta tesis estudiamos algunos métodos para crear ensambles de redes neuronales.

Presentamos tres nuevos métodos para crear ensambles y también discutimos algunas variantes de estas metodologías. Primero discutimos una variante de la MLP llamada "red de cuello de botella", ésta ha sido ampliamente usada para la reducción de dimensión de datos. Presentamos una variante de la MLP de cuello de botella original, y proponemos una nueva estrategia para crear ensambles de redes neuronales usando proyecciones a partir de "redes de cuello de botella". También discutimos una metodología para crear clones a partir de una red individual entrenada. La generación de clones se realiza mediante la adición de ruido controlado a los parámetros de la red principal. Demostramos que un ensamble de clones puede dar mejor desempeño que la red principal. Finalmente mostramos un método para entrenar redes usando datos generados a partir de la estimación de densidad del conjunto de datos original $\mathcal{L}$. Hemos validado todos nuestros métodos con experimentos usando conjuntos de datos de clasificación estándares y nuestros métodos dan resultados aleantadores.

Abstract

Let us assume a data set $\mathcal{L} = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \ldots, n; \mathbf{x}_i \in \Re^p, \mathbf{y}_i \in \Re^s\}$, generated from an unknown but fixed probability distribution P. There are numerous methods available in the literature to estimate the input-output relationship present in the data points in $\mathcal{L}$. In most methods, a predictor function ϕ is constructed using the set $\mathcal{L}$ and optimizing certain criteria. The most important property that the predictor ϕ should have is that the predictor should be able to predict the outputs for points which are from the same distribution P but are not present in the training set $\mathcal{L}$. This property is called the *generalization* ability of the predictor. Constructing predictors with good generalization properties is an active area of research and there exist many proposals to do it. One of the techniques to do it is to construct multiple predictors using $\mathcal{L}$ and aggregating the outputs of these predictors for obtaining the final prediction. This class of methods are called *ensemble methods*.

One of the popular ways to construct a predictor from a given data $\mathcal{L}$ is to train a feed forward neural network (like a multilayered perceptron (MLP)) using $\mathcal{L}$. It is possible to train multiple neural networks from $\mathcal{L}$ and thus create an ensemble of neural networks. In this thesis we study certain methods to create neural network ensembles.

We present three new broad methods to create ensembles and we also discuss some variants of these broad methodologies. First, we discuss a MLP variant called the bottleneck network, which has been widely used for data dimensionality reduction. We present a variant of the original bottleneck MLP, and propose a new way to create neural network ensembles using bottleneck projections. Next, we discuss a methodology to create clones from a single trained network by adding controlled noise to the parameters of the parent network. We show that an ensemble of the clones can give better performance than the parent network. Finally, we show a method to train networks from data points generated from a kernel density estimate of the original data set $\mathcal{L}$. We validate all our methods with experiments using standard benchmark classification data sets, and we show that our methods provide encouraging results.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1. Panorama actual

El "aprendizaje de máquina" es un amplio campo de estudio de la inteligencia artificial que concierne al desarrollo de algoritmos y técnicas que permiten a las computadoras "aprender". El aprendizaje de una máquina se considera como la habilidad de un sistema para mejorar su desempeño considerando su experiencia previa [1].

El aprendizaje de máquina tiene un amplio espectro de aplicaciones que incluye procesamiento de lenguaje natural, clasificación de secuencias de ADN, reconocimiento del habla y de escritura a mano y reconocimiento de objetos, entre otras. Nuestro trabajo se enfoca en sistemas que son capaces de realizar tareas de sistemas desconocidos de una manera aproximada. Para obtener modelos de estos sistemas desconocidos se requiere de ciertas estrategias de identificación que nos aproximen a la definición real de éstos.

Los métodos de "identificación de sistemas" tienen como objetivo identificar modelos que se asemejen a sistemas desconocidos que son capaces de realizar cierta tarea de reconocimiento. La tarea de aproximar un modelo a un sistema real implica un proceso de extraer conocimiento mediante la experiencia previa, representarlo y usarlo en tareas de predicción/reconocimiento futuras. El proceso para obtener el conocimiento es denominado "aprendizaje"; en esta tesis nuestros sistemas consideran el "aprendizaje supervisado" que consiste en obtener un conocimiento atendiendo cierto tipo de guía. Una vez que se tiene un sistema listo para predecir/clasificar, éste será capaz de realizar su tarea usando datos deconocidos. Los sistemas de aprendizaje difieren de acuerdo al tipo de función que deban aprender: sistemas de aproximación de funciones (salidas numéricas) y sistemas clasificadores (salidas discretas). Nuestros estudios se realizan en torno a los sistemas que generan salidas discretas.

El problema fundamental del aprendizaje de máquina consiste en definir una "buena generalización". Dado un conjunto de entrenamiento $\mathcal{L}$, se busca una función capaz de relacionar las entradas conocidas con las salidas conocidas. Para esto puede usarse una función de aproximación de manera que el error no sea excesivo y que nuevos valores en la entrada sean relacionados con tal función. También se puede reproducir

el conjunto de entrenamiento sin error. Sin embargo, cuando el conjunto de entrenamiento se conforme de puntos experimentales, generalmente existirá cierto ruido en los datos y la reproducción exacta de tal conjunto no será una buena estrategia, dado que el ruido también será reproducido y por lo tanto es mejor considerar una aproximación. Así, los dos objetivos opuestos de la aproximación funcional son: minimización del error de entrenamiento y minimización del error en la correspondencia de entradas desconocidas.

Se presupone que un vector de entrada $\boldsymbol{X}$ tiene una relación funcional con la salida $\boldsymbol{Y}$; es decir, $\boldsymbol{Y}=S(\boldsymbol{X})$, donde S es la función desconocida. Esto se modela en la figura 1.1. El problema consiste en encontrar una función B_S que se aproxime a S usando el conjunto finito de datos de entrada-salida $\mathcal{L} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : i = 1 \ldots n\}$; denominamos a $\mathcal{L}$ como el conjunto de entrenamiento. El error originado por B_S usando los puntos que no pertenecen a $\mathcal{L}$, se conoce como "error de generalización".

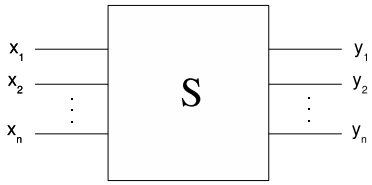


Figura 1.1: Función S

La mala generalización afecta negativamente el desempeño de un sistema de aprendizaje por lo que hoy en día es un reto en el área de aprendizaje de máquina. Actualmente existen varias técnicas para mejorar tal desempeño y una de ellas son precisamente los métodos de ensamble [1, 2].

Ha sido empíricamente mostrado que una mejor estimación (que implica un menor error de generalización) puede ser obtenida combinando las salidas de múltiples estimadores [3]. Así, los ensambles se han convertido en uno de los métodos de aprendizaje más poderosos, estableciéndose como un área de investigación muy activa en el campo del reconocimiento de patrones.

Un ensamble consiste en un conjunto de clasificadores entrenados individualmente cuyas predicciones son combinadas para clasificar nuevos datos [4] (un modelo de ensamble se puede apreciar en la figura 1.2). *Bagging* [5] y *Boosting* [6, 7] son los métodos más comunes para crear ensambles. *Bagging* muestrea un conjunto de datos aleatoriamente con una distribución de probabilidad uniforme mientras que *Boosting* lo hace mediante una distribución no uniforme. Este último procede por etapas, con una nueva red entrenada en cada una de ellas. La red inicial se entrena con igual énfasis sobre todos los patrones de entrenamiento. Al terminar cada etapa, los patrones clasificados erróneamente son identificados para incrementar su importancia en un nuevo conjunto de entrenamiento dado a la red de la siguiente fase. Al generar determinada cantidad de redes, éstas son combinadas por medio de una "votación ponderada" que se basa en el error de entrenamiento de cada clasificador.

Bagging frecuentemente es más preciso que un clasificador individual, sin embargo a veces es mucho menos preciso que *Boosting*, el cual puede crear ensambles cuya

precisión es menor que la de un clasificador único, especialmente cuando se usan redes neuronales como elementos del ensamble. Los análisis indican que el desempeño de los métodos tipo *Boosting* es dependiente de las características del conjunto de datos que estén siendo examinados. Además, existen resultados que muestran que los ensambles tipo *Boosting* usualmente son mejores que *Bagging* pero pueden sobreajustar conjuntos de datos ruidosos, lo cual decrementa su desempeño. Esto último es diferente para *Bagging*, ya que estudios recientes han demostrado que es un método con mayor resistencia al ruido [8].

Los métodos *Bagging* y *Boosting* cuentan con garantías de desempeño teóricas y resultados experimentales importantes. Sin embargo, necesitan que el conjunto de entrenamiento total esté disponible en cada proceso del aprendizaje y en ocasiones requieren acceso aleatorio a los datos. Se han desarrollado versiones en línea de estos métodos que implican únicamente un acceso a los datos de entrenamiento y que se ejecutan mucho más rápido que sus contrapartes por lotes. La diferencia entre las precisiones de los algoritmos de ensamble en línea y por lotes está en función de las diferencias entre las precisiones de los algoritmos de aprendizaje en línea y por lotes que se hayan usado. Los algoritmos en línea son útiles cuando los conjuntos de datos por lotes no pueden ser cargados totalmente en memoria [9]. *Boosting* ha sido calificado por el mismo Leo Breiman como el mejor clasificador existente indicando que su secreto radica en el hecho de que es capaz de crear los conjuntos subsecuentes de datos de entrenamiento con los que los clasificadores son entrenados para darle diversidad al ensamble [10].

En el intento por definir las razones que expliquen el desempeño exitoso de *Bagging* se han originado diversos trabajos que pretenden dar una respuesta [11, 5]. Friedman [12] indica que *Bagging* funciona porque reduce la varianza del error manteniendo sin cambios al sesgo, mientras que Grandvalet [13] muestra evidencia de que puede converger sin reducir la varianza. Domingos [11] da un enfoque Bayesiano e investiga dos hipótesis: *Bagging* funciona porque (1) es una aproximación al Promedio del Modelo

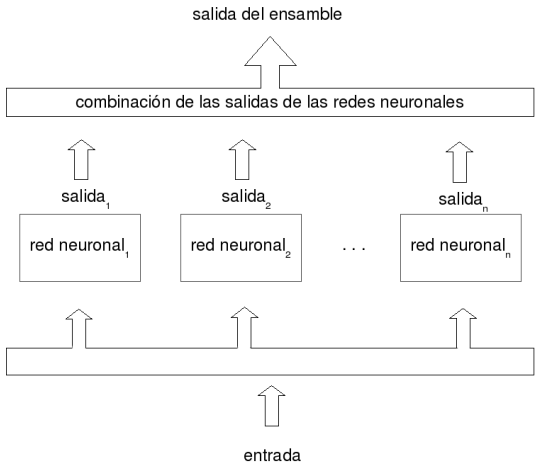


Figura 1.2: Ensamble de redes neuronales [1].

Bayesiano (*Bayesian Model Averaging*) [14] y (2) cambia la distribución del clasificador previamente combinado a una región más apropiada del espacio del modelo, aunque sus estudios soportan mejor su segunda hipótesis.

Existen estudios con ensambles de varios tipos de clasificadores. Uno de ellos implica crear ensambles simples de redes neuronales donde cada red usa el conjunto de entrenamiento completo y solamente difiere en las definiciones aleatorias de sus pesos iniciales. Los resultados obtenidos indican que esta técnica de ensamble es realmente efectiva, frecuentemente produciendo resultados tan buenos como *Bagging* [8]. Tales resultados muestran que los ensambles de redes neuronales son generalmente consistentes en términos de precisión. Sin embargo, el problema de la aplicación de estos métodos radica en que se debe entrenar cada clasificador del ensamble, lo cual implica un alto costo computacional.

La diversidad entre los miembros de un ensamble se considera un factor clave cuando se desea su combinación y hasta la fecha su medición no es directa porque no se ha aceptado de manera general una definición formal que permita hacerlo. A partir de experimentos recientes se han encontrado y estudiado ciertas estadísticas que miden la diversidad entre las salidas de los clasificadores binarios; esto ha establecido como incierta la utilidad de este factor para construir ensambles aún ante la existencia probada de conexiones entre la diversidad y la precisión en algunos casos especiales de combinación de clasificadores. Por lo tanto, no es claro el grado de influencia de este factor sobre el desempeño de los ensambles de clasificadores. Sin embargo, el estudio realizado en [15] indica que es correcta la motivación general para diseñar ensambles con buena diversidad, pero señala también que el problema que implica el medir esta diversidad y usarla efectivamente para construir mejores equipos de clasificadores está aún por resolverse.

A través de otro estudio de Kuncheva [16] se observaron las relaciones entre diferentes medidas de diversidad y algunos métodos de combinación de clasificadores. Los experimentos realizados no mostraron alguna relación consistente debido a que no se encontró una correlación muy alta en tales relaciones. Los resultados de éste y otros estudios han permitido saber que sigue siendo incierta la participación de las medidas de diversidad en el diseño de ensambles.

Los ensambles pueden integrarse por diferentes tipos de clasificadores. Los estudios considerados en este trabajo consisten en analizar ensambles cuyos miembros son redes neuronales. Las redes neuronales representan dificultades para *Bagging* y *Boosting* ya que estos clasificadores necesitan mucho tiempo de procesamiento y requieren seleccionar parámetros de entrenamiento. A continuación se mencionan algunos factores que se han considerado para incluir a las redes neuronales en nuestros estudios:

- Estudios empíricos previos han demostrado que las redes neuronales individuales producen clasificadores de precisión elevada [17].

- Las redes neuronales ofrecen un marco de trabajo natural para construir ensambles ya que son clasificadores cuyo método de aprendizaje es inestable (pequeños

cambios en el conjunto de entrada pueden generar grandes cambios en la predicción de las salidas) [18].

- Son apropiadas para aplicaciones en las que no se dispone *a priori* de un modelo identificable que pueda ser programado [18].

- Los ensambles de redes neuronales son aplicados en el reconocimiento digital de escritura a mano [19] y en otras áreas que requieren precisión elevada.

Las redes neuronales han despertado un renovado interés en los últimos 15 años ya que se han desarrollado nuevos métodos de aprendizaje capaces de tratar con problemas a gran escala. Después del trabajo de Rosenblatt y otros, no existía ningún algoritmo de aprendizaje eficiente para redes multicapa o de "pro-alimentación" (*feed-forward*), lo que dio origen a la conclusión temprana de que este campo de estudio estaba condenado a la desaparición. Pero en 1980 se propuso el algoritmo de "retro-propagación" (*backpropagation*) y surgieron topologías de redes alternativas dando origen a nueva actividad que puso a la neurocomputación de vuelta dentro de los principales campos de estudio de la ciencia de la computación.

El algoritmo de "retro-propagación" es el más utilizado para el aprendizaje de redes neuronales y es de cierto modo únicamente un método numérico de aproximación estadística. Tiene la desventaja de que llega a ser muy lento en regiones planas de la función de error. Se puede pensar que este tipo de problema puede ser resuelto cambiando de algoritmo de aprendizaje. Quizás exista un método de aprendizaje capaz de encontrar una solución con un número de pasos que sea polinomial en el número de pesos en la red, pero éste no se ha encontrado hasta la fecha. Así, es computacionalmente difícil encontrar los pesos apropiados para un problema de aprendizaje mediante el ajuste de un par entrada-salida. Esto significa que esta tarea pertenece a la clase de problemas NP-completos en el peor caso, por lo que no se conoce ningún algoritmo que requiera tiempo polinomial para resolverlo, y éste probablemente no existe. El esfuerzo computacional inmerso en el aprendizaje de la red crece exponencialmente con la cantidad de parámetros desconocidos [20]. Así, la investigación ha girado en torno a la generación de nuevos métodos de aprendizaje que tengan alguna mejora sobre el algoritmo de "retro-propagación". También, hoy en día se están tratando otros problemas diferentes a la determinación de los parámetros de la red, tales como la adaptación de la topología de la red.

Los cambios a los pesos de la red permiten definir adecuadamente un ajuste funcional y al mismo tiempo detectar la configuración óptima de parámetros de la red, lo que implica por un lado que la red realice una correspondencia de manera que las entradas conocidas coincidan con las salidas conocidas y además que la red sea capaz de generalizar. Para lograr esto, las redes de "pro-alimentación" son las más estudiadas.

Otra factor importante de los ensambles es su arquitectura. Ésta se define por la combinación de las arquitecturas de los clasificadores que lo integran; así, conforme éstas sean más complejas el aprendizaje del ensamble requerirá más esfuerzo

computacional. La arquitectura es un aspecto muy importante en el diseño de las redes neuronales para relacionar de manera óptima el desempeño y la complejidad computacional, por lo que para su definición se han creado algoritmos que consideran técnicas de poda de parámetros y de "análisis de sensibilidad" que permiten eliminar parámetros irrelevantes de las redes neuronales [21].

La sensibilidad es un aspecto importante que se considera en los experimentos de esta tesis, específicamente alrededor de la idea consistente en definir redes neuronales a partir de una red neuronal inicial. Estudios recientes en el perceptrón multicapa definen a la sensibilidad como el valor esperado de los errores en las salidas del perceptrón multicapa que han sido originados por las perturbaciones en los pesos y entradas de la red. Hoy se sabe que dicho valor se incrementa conforme aumentan también las perturbaciones en los parámetros y que la configuración de la red neuronal limita tal incremento por lo que cada neurona en cierta capa posee un valor máximo de sensibilidad [22].

Varias técnicas se han aplicado para generar ensambles de redes neuronales. Existen estudios sobre ensambles que implican aprendizaje evolutivo para determinar automáticamente la cantidad de clasificadores neuronales en un ensamble y para explotar la interacción entre la arquitectura y la combinación de las redes neuronales. Los resultados mostraron que el ensamble creado tiene una buena capacidad de generalización [23].

Se han desarrollado otras aplicaciones de algoritmos genéticos para buscar directamente un conjunto preciso y diverso de redes neuronales entrenadas. Una técnica denominada *ADDEMUP* crea primero una población inicial que usa operadores genéticos para crear continuamente nuevas redes, manteniendo al conjunto con redes neuronales tan precisas como se pueda y que al mismo tiempo tengan diferencias entre sí tanto como sea posible para mantener diversidad. Los experimentos indicaron que esta técnica, aplicada en problemas de ADN, es capaz de generar un conjunto de redes neuronales con mayor precisión que varias aproximaciones existentes [24]. Sin embargo, la aplicación de algoritmos genéticos para generar redes neuronales implica costos de cómputo muy altos.

1.2. Objetivo

Los estudios realizados en este trabajo contemplan el diseño e implementación de un nuevo método de ensamble de redes neuronales cuyo proceso de aprendizaje sea menos costoso que el de los métodos actuales sin que esto implique gran pérdida de precisión. Se pretende competir con *Bagging* porque es un método simple y con gran capacidad de desempeño que se ha usado en muchas situaciones para crear ensambles de redes neuronales.

Ante el paradigma actual que consiste en crear ensambles de clasificadores mediante el entrenamiento de cada clasificador miembro, nuestros estudios surgen para proponer un esquema donde el entrenamiento sea simulado y por ende el costo computacional sea menor al de los métodos actuales. Tal esquema implica crear ensambles

de redes neuronales perturbando los parámetros de una red para generar los clasificadores miembros. Cabe mencionar que cada clasificador que compone al ensamble en un principio es una copia de una red neuronal entrenada de manera que los valores de sus parámetros ya están bien definidos. Posteriormente, se busca obtener redes neuronales convenientes por medio de la perturbación a los parámetros de tales copias.

Nuestro trabajo considera también definir un nuevo método de ensamble de redes neuronales que contemple la generación de los datos de entrenamiento y prueba mediante estrategias de reducción de dimensión de los datos.

Los experimentos relacionados con la reducción de dimensiones implican poder aminorar el tiempo de aprendizaje de las redes neuronales sin gran pérdida de precisión. Se contemplan ciertas modificaciones a la red neuronal clásica de cuello de botella con el objetivo de aplicarla en tareas de clasificación. La reducción de dimensiones también es considerada como una estrategia para construir ensambles de redes neuronales más rápidos.

A lo largo de este trabajo presentamos comparaciones de desempeños de nuestros esquemas ante *Bagging* y otros métodos clásicos y se aprecian estadísticas que definen el desempeño de nuestros métodos.

1.3. Descripción de la tesis

A lo largo de los 6 capítulos de este documento se describen aspectos importantes de los ensambles de clasificadores que han sido observados durante nuestros estudios.

En este primer capítulo hemos dado un panorama general de lo que implica el problema de la generalización en el área de reconocimiento de patrones, se ha mencionado la importancia de los ensambles de clasificadores en tal área y se han dado algunas de sus características principales. También se han descrito algunas cualidades de las redes neuronales que motivan su inclusión en nuestros estudios.

En el capítulo 2 mencionamos aspectos importantes de los ensambles de redes neuronales y de la red neuronal multicapa (*multilayer perceptron*).

En el capítulo 3 se explica el trabajo desarrollado acerca de la reducción de dimensión que implica el entrenamiento de redes neuronales con cierta arquitectura. Se mencionan detalles de los esquemas que se proponen a partir de estos estudios. Se describen nuestras ideas, los esquemas implementados y la conveniencia de aplicar nuestros esquemas a problemas que se definen con gran cantidad de características.

En el capítulo 4 describimos los experimentos realizados para crear un nuevo método de ensamble. Se mencionan las diversas estrategias diseñadas alrededor de la idea de adicionar ruido a los parámetros que definen a una red neuronal. Se mencionan los problemas encontrados durante nuestros experimentos y se indican las causas que establecen el grado de éxito de nuestros esquemas.

En el capítulo 5 mencionamos aspectos importantes de otra técnica existente que proponemos usar para crear ensambles usando estimación de densidades (*kernel estimate*). La idea básica consiste en integrar un ensamble de redes neuronales entrenando

cada clasificador con un conjunto diferente de entrenamiento generado a partir de la adición de ruido a los datos de entrenamiento originales mediante la estimación de densidades.

Finalmente, en el capítulo 6 se mencionan las conclusiones del trabajo que se ha desarrollado.

Capítulo 2

Ensambles de redes neuronales

Cuando es necesario realizar decisiones importantes, frecuentemente es útil un grupo de personas, por lo que se tienen comités, parlamentos, jurados, etc. Esto indica que es generalmente aceptado que "dos cabezas piensan mejor que una" y que pueden no existir malas decisiones si se atienden varias opiniones para obtener un acuerdo.

La combinación de clasificadores en la década pasada comenzó a recibir mayor atención y actualmente es una derivación del reconocimiento de patrones inmersa en el paradigma del aprendizaje supervisado. Si se combinan convenientemente las salidas de ciertos clasificadores se obtiene una mayor precisión que la del mejor clasificador combinado. El enfoque del estudio de los ensambles ha evolucionado de las soluciones prácticas mediante heurísticas, a la explicación de por qué los métodos y estrategias funcionan tan bien y en cuáles casos algunos métodos son mejores que otros. Hoy en día existe dentro del área un consenso que indica que la diversidad es el factor más determinante para lograr una mejor precisión del ensamble de manera que el método de fusión tiene una importancia secundaria, aunque su adecuada elección puede mejorar el desempeño del ensamble [25].

2.1. Redes neuronales artificiales

Las redes neuronales artificiales (RNA) exhiben propiedades interesantes ya que intentan reflejar las capacidades de procesamiento de información de los sistemas nerviosos. Estas redes no operan secuencialmente y tienen una estructura multicapa jerárquica que permite transmitir la información desde una unidad a sus vecinos más distantes e inmediatos calculando los parámetros de la red neuronal por medio de la adaptación de valores y usando canales de entrada, un cuerpo celular y un canal de salida. Las sinapsis biológicas se simulan con puntos de contacto entre el cuerpo celular y las conexiones de salida o entrada asociando un "peso" a tales puntos [20].

El problema inmerso en los sistemas neuronales es la transmisión de la información que puede resolverse transformándola en una transmisión recurrente entre dos puntos. Las neuronas biológicas transmiten información usando una descarga de

señales eléctricas (denominada "potencial de acción") que se produce y transmite en la membrana celular viajando a través de los axones de manera auto-regenerativa. Así, la transmisión digital de información consiste en producir una nueva señal tras otra. Una neurona codifica su nivel de actividad ajustando la frecuencia de los impulsos generados la cual es mayor para estímulos grandes por lo que se puede decir que la información se transmite de célula a célula usando modulación de frecuencia para incrementar la precisión de la señal. Luego, las sinapsis indican la dirección para transmitir información y las señales fluyen de una célula a otra de una manera bien definida lo que es expresado en los modelos de redes neuronales artificiales empotrando las neuronas artificiales en un grafo dirigido. Así, las células procesan la información integrando señales de entrada y reaccionando a la inhibición (señales de entrada). El flujo de los transmisores a partir de una sinapsis excitatoria permite que la célula adjunta reaccione si se excede un umbral, lo que implica que suficientes canales deben ser abiertos de manera que la célula alcance su umbral de activación para que se genere un potencial de acción en el axón de esta célula. Posteriormente, las neuronas procesan la información en la membrana, la cual regula su procesamiento y transmisión. Todo tipo de información generada en una red neuronal es almacenada en las sinapsis [20].

A través de la modificación de la permeabilidad de la membrana, una célula puede ser entrenada para activarse más frecuentemente definiendo un umbral de activación menor. La información almacenada debe ser actualizada periódicamente para mantener la permeabilidad óptima de la membrana celular. Así, este tipo de almacenamiento se usa también en las RNA pues están conectadas por aristas con diferentes eficiencias de transmisión, de manera que la información que fluye a través de las aristas es afectada por una constante que refleja tal eficiencia.

Con la información anterior se puede modelar una "neurona artificial" integrada por dendritas, un cuerpo celular y un axón. Por lo tanto, en las RNA se pueden observar funciones localizadas en sus neuronas (nodos) y también es posible considerar ciertas reglas de composición inmersas implícitamente en los patrones de interconexión de los nodos, en la transmisión de la información y en la presencia o ausencia de ciclos.

En la figura 2.1 se aprecia la estructura de una neurona artificial con n entradas. Cada canal de entrada i transmite un valor real x_i. La función f en el cuerpo de la neurona puede ser seleccionada arbitrariamente. Los canales de entrada tienen un peso asociado de modo que la información de entrada x_i es multiplicada por el peso w_i. La información transmitida ingresa a la neurona (usualmente sólo sumando las diferentes señales) y entonces la función es evaluada.

Así, cada nodo se define como una función que transforma su entrada en una salida, lo que permite integrar una red de funciones con determinado patrón de interconexión como se modela en la figura 2.2. La red se aprecia como una función F evaluada en el punto (x, y, z). Sus nodos implementan las funciones f_1, f_2, f_3, f_4 que se combinan considerando la comparación de su entrada total contra su propio umbral para definir finalmente a la "función de red F" [20]. Esto muestra que diferentes conjuntos de pesos generan diferentes funciones de red de lo que se desprende que los elementos

relevantes que definen cualquier modelo de RNA son: la estructura de los nodos, la topología de la red y el algoritmo de aprendizaje usado para encontrar los pesos de la red.

Las RNAs relacionan sus entradas con determinadas salidas mediante un proceso de auto-organización. Si se desea relacionar una entrada real n-dimensional $(x_1, x_2, \ldots, x_n)$ a una salida real m-dimensional $(y_1, y_2, \ldots, y_m)$, entonces la red neuronal actúa como una "máquina de correspondencia" que modela una función $F : \mathbb{R}^n \to \mathbb{R}^m$. F se evalúa con una red de funciones que permite el flujo de información a través de sus aristas y contiene nodos que calculan valores que son transmitidos como argumentos de otros nodos para realizar nuevos cálculos. Finalmente, el último nodo genera cierto valor que será tomado como el resultado del cálculo total.

En nuestros experimentos se presupone que el total de aristas en una determinada dirección y la cantidad de aristas que ingresan en un nodo no están restringidos por algún límite. Las aristas que ingresan en un nodo indican los n argumentos que serán reducidos a un valor numérico único, de lo que se desprende que cada unidad se compone por una función de integración g que hace tal reducción y por la función de activación f que produce la salida del nodo tomando el valor de g como argumento.

También hemos experimentado con redes cuya topología se modifica para integrar ensambles. El modelo de RNA usado en nuestros estudios es el perceptrón multicapa por la precisión que ofrece y por el amplio rango de funciones que puede calcular.

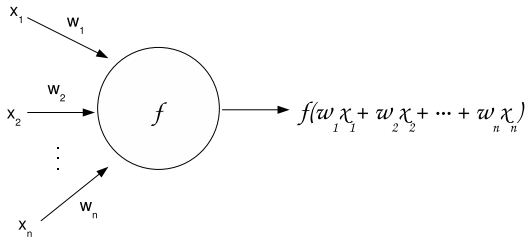


Figura 2.1: Una neurona artificial.

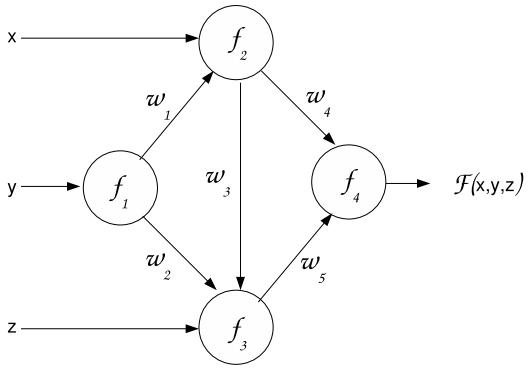


Figura 2.2: Modelo de una red neuronal artificial.

2.2. El perceptrón multicapa (MLP)

En 1958, Frank Rosenblatt propuso el modelo del perceptrón que define pesos numéricos con un patrón especial de interconexión e integra un proceso de aprendizaje que realiza la adaptación de los pesos mediante un algoritmo numérico. El modelo implica convertir el umbral de la neurona al peso -θ (denominado "sesgo") del canal de una entrada adicional conectado a la constante 1. De esta manera, el vector de entrada $(x_1, x_2, \ldots, x_n)$ se extiende con un 1 adicional obteniendo al vector $(n+1)$-dimensional $(x_1, x_2, \ldots, x_n, 1)$ denominado "vector de entrada extendido" y definiendo un vector de pesos extendido $(w_1, \ldots, w_n, w_{n+1})$ asociado al perceptrón donde $w_{n+1} = -\theta$. A partir de múltiples entradas, el perceptrón calcula una salida única formando una combinación lineal con sus pesos de entrada y estableciendo la salida con cierta función de activación no lineal como se expresa en la ecuación 2.1 [18].

Un perceptrón individual no es muy útil por su limitada habilidad de correspondencia y en conjunto incrementa su utilidad si se usa como bloque constructor de una estructura más larga y práctica denominada "red perceptrón multicapa" (*Multilayer Perceptron*). Un MLP consiste de un conjunto de nodos fuente que integran la capa de entrada, una o más capas ocultas de nodos de cómputo y una capa de nodos de salida. La señal de entrada se propaga a través de la red, capa por capa [18]. Así, mientras que las redes de capa única (compuestas de perceptrones paralelos) están limitadas en el tipo de correspondencias que pueden representar, el poder de una red MLP con únicamente una capa oculta, es muy grande. Algunos investigadores demostraron en 1989 [26] que las redes MLP son capaces de aproximar cualquier función continua $f : \mathbb{R}^n \to \mathbb{R}^m$ con precisión arbitraria.

El aprendizaje de una red MLP implica un proceso que requiere de un conjunto de pares entrada-salida de entrenamiento cuya dependencia será modelada. Tal proceso es una adaptación de los pesos y sesgos a sus valores óptimos para los pares dados considerando que el criterio a ser optimizado es el cuadrado del error de aproximación. Así, el aprendizaje consiste en aproximar una función desconocida S con una función de red F mediante el algoritmo de "retro-propagación" que consiste de dos etapas que se describen a continuación. En la etapa de "pro-alimentación" los pesos no se alteran y se calculan las salidas de cada neurona por medio de sumatorias de productos entre los pesos y entradas relacionadas con cada nodo. Inicia en la capa de entrada y termina en la capa de salida, calculando la señal de error para cada neurona de esta capa. La etapa de "retro-propagación" inicia en la capa de salida comunicando las señales de error capa por capa hacia las entradas de la red. Esto significa que se calculan las derivadas parciales de la función de costo (error) con respecto a los diferentes parámetros (es decir, el gradiente local para cada neurona) y se retro-propagan a través de la red. Este proceso modifica los pesos capa por capa y propaga tales cambios con el objetivo de acercar la respuesta de la red a la deseada. Así, tal adaptación de parámetros se realiza con cualquier algoritmo de optimización basado en el gradiente cuyo proceso es iterado hasta que los pesos converjan [18].

$$y = f(\sum_{i=1}^{n} w_i x_i + b) \tag{2.1}$$

El MLP permite crear representaciones compactas de correspondencias en problemas del mundo real y las neuronas que la integran no definen funciones lineales. Así, la salida de cada neurona se obtiene por su función de activación que define el nivel de activación de la neurona. Cuando estas funciones son no lineales, una red multicapa no es equivalente a cualquier arquitectura de capa única cuyos nodos tengan la misma función de activación. Se ha demostrado que una capa de neuronas no lineales seguida de una capa lineal puede aproximar cualquier función no lineal con precisión arbitraria [26], lo que significa que una red MLP es un aproximador de funciones universal. Las funciones de activación más usadas son la tangente hiperbólica *tanh(x)* y la función *sigmoide* definida en la ecuación 2.2.

$$f(x) = \frac{1}{(1 + exp(-x))} \tag{2.2}$$

La computación realizada por un perceptrón se entiende como una separación lineal del espacio de entrada; sin embargo, si se buscan los pesos apropiados para un perceptrón el proceso de búsqueda se puede observar mejor en un espacio de pesos. Así, si se desean obtener m pesos reales, el espacio de búsqueda es $\mathbb{R}^m$. Dadas n entradas, encontrar la separación lineal apropiada equivale a encontrar $n + 1$ parámetros libres (n pesos y el sesgo) que representan un punto en el espacio de pesos (n+1)-dimensional; tal punto define una combinación de pesos y una separación lineal específica del espacio de entrada. Esto significa que cada punto en el espacio de pesos puede ser asociado con un hiperplano en el espacio de entrada como se modela en la figura 2.3. Ocurre algo similar a partir del espacio de entrada hacia el de pesos. Es decir, un punto en el espacio de entrada define un plano de corte en el espacio de pesos, por lo que puntos en un espacio, son mapeados a planos en el otro espacio y viceversa; tal relación complementaria se denomina "dualidad" [20].

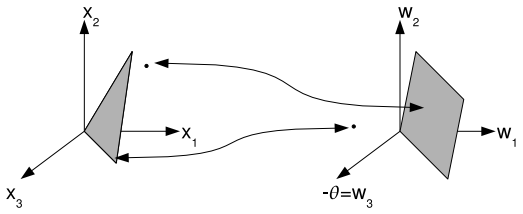


Figura 2.3: Dualidad del espacio de entrada y salida.

El algoritmo de aprendizaje que se usa en este trabajo implica presentar a cada red algunas muestras de la correspondencia entrada-salida deseada, de manera que la red "aprenda" a producir la respuesta deseada después de corregir sus parámetros iterativamente. Esto se modela en la figura 2.4.

Las neuronas y sus interconexiones definen la arquitectura de un MLP. Las arquitecturas con capas implican que el conjunto de unidades de cómputo se divide en L subconjuntos de manera que sólo las conexiones de las unidades del primer subconjunto se dirigen hacia las unidades del segundo, las del segundo subconjunto hacia las unidades del tercero y así sucesivamente. Las entradas se conectan únicamente con las unidades del primer subconjunto y las unidades del último subconjunto son las únicas conectadas a las salidas. Estos subconjuntos se denominan "capas" de red por lo que el conjunto de entradas define la capa de entrada. El correspondiente a las unidades de salida integra la capa de salida y el resto constituye el conjunto de capas ocultas. Las entradas I permiten el ingreso de la información y no realizan cálculos. En cambio, cada neurona de cómputo del conjunto N colecta la información que proviene de sus n líneas de entrada a través de su "función de integración" $\psi : \mathbb{R}^n \to \mathbb{R}$ para calcular el valor de "excitación" que se evalúa con la "función de activación" $\Phi : \mathbb{R} \to \mathbb{R}$. Los resultados se transmiten al conjunto O de salidas desde las cuales son leídos. El conjunto A de aristas ponderadas se distribuye entre la capa de entrada y la primera capa oculta, entre la última capa oculta y la de salida, entre las capas ocultas y posiblemente entre las unidades de cómputo, por lo que una arista dirigida se representa por tres partes: $u \in I \cup N$, $v \in N \cup O$ y $w \in \mathbb{R}$ [20].

La arquitectura usada en nuestros estudios establece que las unidades en una capa no están conectadas unas con otras. Sin embargo, todas las unidades de una capa están conectadas a todas las unidades en la siguiente capa de manera que si existen m unidades en la primera capa y n en la siguiente, se tendrán mn pesos entre las capas.

La capacidad de generalización de un MLP es mayor si el conjunto de entrenamiento es más grande; sin embargo, también puede suceder que la función se "sobreajuste" lo que significa que se aprenda el conjunto de entrenamiento perfectamente pero que se relacionen los puntos desconocidos erróneamente.

2.2.1. El algoritmo de "retro-propagación"

La red MLP calcula un rango de funciones más amplio que las redes de capa única y el esfuerzo computacional necesario para encontrar la combinación correcta de pesos se incrementa conforme se consideran más parámetros y topologías más complejas. El algoritmo de "retro-propagación" básicamente busca el mínimo de la función de error en el espacio de pesos usando el método del gradiente descendente de la función para

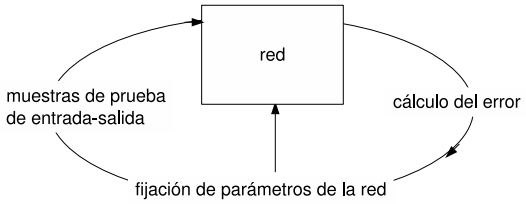


Figura 2.4: Proceso de aprendizaje en un sistema paramétrico

actualizar los pesos que en un principio eran aleatorios, de manera que la combinación de pesos que minimiza la función de error es una solución del problema de aprendizaje. En general, este algoritmo usa la función de activación *sigmoide* que se define como una función real $s_c : \mathbb{R} \to (0, 1)$ y cuya derivada con respecto a x se indica por la expresión 2.3.

$$\frac{d}{dx}s(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = s(x)(1 - s(x)) \tag{2.3}$$

Una función de activación diferenciable permite que la función de red sea también diferenciable (considerando que la función de integración es la suma de las entradas) dado que la red es una composición de funciones a partir de las entradas hasta el espacio de salida. Así, la función de error (expresada en la ecuación 2.5) es también diferenciable. Debido a que la función *sigmoide* siempre tiene una derivada positiva, la pendiente de la función de error genera una mayor o menor dirección descendente que puede ser seguida para moverse por la superficie de la función de error hasta alcanzar el mínimo global.

En este trabajo consideramos la función *sigmoide* de manera que dada una entrada $x_1, \ldots, x_n$, los pesos $w_1, \ldots, w_n$ y un sesgo -θ, se calcula la salida mediante la expresión 2.4. Un mayor valor de excitación origina una salida con valor cercano a 1 por lo que la serie continua de valores de salida del MLP se compara a una serie continua de clases que divide al espacio de entrada.

$$f(X) = \frac{1}{1 + e^{(\sum_{i=1}^{n} w_i x_i - \theta)}} \tag{2.4}$$

Dado un conjunto de entrenamiento $\{(x_1, t_1), \ldots, (x_p, t_p)\}$ integrado por p patrones de entrada y salida (representados por vectores de n y m dimensiones), se define una red neuronal con n unidades de entradas y m unidades de salida. En general, al presentar x_i a la RNA, ésta produce una salida o_i diferente al valor real t_i pero se desea que o_i y t_i sean lo más similar posible para todos los patrones. Entonces, el algoritmo de aprendizaje minimiza la función de error E de la red para el conjunto de entrenamiento que se expresa por la ecuación 2.5. Posteriormente, nuevos patrones de entrada desconocidos se presentan a la red esperando que ésta realice una estimación correcta.

$$E = \frac{1}{2}\sum_{i=1}^{p} \parallel o_i - t_i \parallel^2 \tag{2.5}$$

Para calcular E, cada unidad de salida j se conecta a un nodo que evalúa la función $\frac{1}{2}(o_i - t_i)^2$, donde o_{ij} y t_{ij} denotan el j-ésimo componente del vector de salida o_i y de la salida deseada t_i, respectivamente. Las salidas de los m nodos se suman para obtener E_i y esto se realiza para cada patrón t_i. Al final, el error E de la red sobre el conjunto de entrenamiento se obtiene sumando los errores $E_i + \cdots + E_p$.

Este algoritmo de aprendizaje se aplica en redes de "pro-alimentación" con más de una entrada. El peso w_{ij} tiene asociada la arista que va del i-ésimo nodo al j-ésimo

nodo de la red y puede ser tratado como un canal de entrada de la subred hecha por todos los caminos que inician en w_{ij} y que terminan en una única salida de la red. La información de entrada de tal subred en el paso de "pro-alimentación" es $o_i w_{ij}$ donde o_i es la salida almacenada de la unidad i. El paso de "retro-propagación" calcula el gradiente de E con respecto a esta entrada, $\frac{\partial E}{\partial w_{ij}} = o_i \frac{\partial E}{\partial o_i w_{ij}}$. Así, todas las subredes definidas por cada peso se pueden manejar simultáneamente considerando en cada nodo i los valores siguientes:

- La salida o_i del nodo en el paso de "pro-alimentación"
- El resultado acumulado del cálculo realizado durante la "retro-propagación" hasta este nodo. Este valor es denominado "error de retro-propagación" de manera que si éste es denotado en el j-ésimo nodo como δ_j, entonces se puede expresar la derivada parcial de E con respecto a w_{ij} como se define en la ecuación 2.6.

Una vez que se han calculado las derivadas parciales, se aplica el gradiente descendente para sumar el incremento definido por $\Delta w_{ij} = -\gamma o_i \delta_j$ a cada peso w_{ij}.

$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_i \tag{2.6}$$

Lo anterior es necesario para usar al algoritmo como método de aprendizaje el cual puede resumirse en cuatro pasos:

1. Computación en la etapa de "pro-alimentación"
2. Retro-propagación hacia la capa de salida
3. Retro-propagación hacia las capas ocultas
4. Actualización de pesos

Se termina cuando el valor de la función de error es lo suficientemente pequeño [18].

2.3. Cálculo del error de predicción

El principal punto concerniente a la aproximación de funciones, mediante técnicas que usen redes neuronales, es obtener una estimación del error de predicción cuando se presentan nuevos valores a una red. Se podría pensar que el error de generalización esperado de una red es la raíz cuadrada de la media del error cuadrático de entrenamiento E considerando que N es el tamaño del conjunto de entrenamiento, como se expresa con la ecuación 2.7 [20].

$$\tilde{E} = \sqrt{\frac{E}{N}} \tag{2.7}$$

Sin embargo, esto subestimaría el verdadero error de generalización ya que los parámetros de la red estarían ajustados para tratar exactamente con determinado conjunto de datos y podría existir un sesgo en favor de sus elementos.

Bootstrap

El método *Bootstrap* fue propuesto por Efron en 1979 [20] y trata con este problema. Su observación clave es que los datos existentes pueden ser usados para ajustar un clasificador de manera que es posible definir un indicador de la distribución de las entradas esperadas en un futuro. En el mundo real se podría usar la regresión lineal para calcular el error de generalización usando nuevos datos no incluidos en el conjunto de entrenamiento. Con *Bootstrap* se intenta imitar esta situación creando muestreos aleatorios a partir del conjunto de datos de entrenamiento inicial para crear conjuntos de entrenamiento diferentes.

Se presupone que se tiene un conjunto de datos $X = x_1, x_2, \ldots, x_n$ y una estadística $\hat{\theta}$ calculada con X que estima el valor real θ de la estadística sobre la población entera. Se desea conocer la confiablidad de $\hat{\theta}$ calculando su desviación estándar. La realidad implica que los datos son generados con una distribución de probabilidad desconocida F y en *Bootstrap* se asume que es posible aproximar esta distribución creando muestras aleatoriamente con reemplazo a partir de X. De esta manera, se genera un nuevo conjunto de datos X^* y se calcula un nuevo valor de las estadísticas denominado $\hat{\theta}^*$. Este procedimiento puede ser repetido varias veces con muchos conjuntos de datos generados aleatoriamente y se puede aproximar el valor de $\hat{\theta}$ usando $\hat{\theta}^*$. Esto indica que la variación de la densidad de los datos se puede aproximar realizando un muestreo con reemplazo a partir de los datos que se conocen. Así, *Bootstrap* usa a $\hat{F}$ como aproximación a la distribución F y si tal aproximación es lo suficientemente buena se puede derivar más información a partir de los datos generados. El procedimiento de *Bootstrap* se resume en la figura 2.5.

Este método ayuda a calcular una mejor estimación del error promedio esperado generando N conjuntos de entrenamiento diferentes. Cada uno de estos conjuntos se genera seleccionando m pares de entrada-salida de manera aleatoria y con reemplazo, a partir del conjunto de entrenamiento original. La red neuronal se entrena siempre usando el mismo algoritmo y criterio de parada, por lo que para cada red entrenada se calcula:

- La media del error cuadrático Q_i^* para el i-ésimo conjunto de entrenamiento generado con *Bootstrap*,
- La media del error cuadrático para los datos originales, denominada Q_i^0

La desviación estándar de los valores Q_i^* es una aproximación a la verdadera desviación estándar del ajuste de la función. Además, en general Q_i^* será menor que Q_i^0 porque el algoritmo de entrenamiento ajusta los parámetros de manera óptima para el conjunto de entrenamiento que se esté usando [20].

Figura 2.5: Algoritmo *Bootstrap*

Entrada. Conjunto de datos X
Salida. N réplicas, estimación de error estándar $\hat{s}_N$

i) Seleccionar N muestras independientes generadas con *Bootstrap* $x^{*1}, x^{*2}, \ldots, x^{*N}$ cada una consistente de n valores de datos seleccionados con reemplazo a partir de X.

ii) Evaluar la estadística S deseada correspondiente a cada muestra generada con *Bootstrap*

$$\hat{\theta}^*(b) = S(x^{*b}) \qquad b = 1, 2, \ldots, N$$

iii) Estimar el error estándar $\hat{s}_N$ con la desviación estándar muestral de las N réplicas

$$\hat{s}_N = \left(\frac{\sum_{b=1}^{N} [\hat{\theta}^*(b) - \tilde{\theta}]^2}{(N-1)} \right)^{1/2}, \text{ donde } \tilde{\theta} = \frac{\sum_{b=1}^{N} \hat{\theta}^*(b)}{N}$$

Validación cruzada

En métodos que aplican redes neuronales, la técnica de "validación cruzada" (*cross-validation*) ha sido usada con mucha frecuencia y consiste en que, dado un conjunto de entrenamiento T, algunos de los pares de entrada-salida están reservados y no se usan para entrenar la red neuronal (típicamente entre el 5 % y 10 % de los datos). La red entrenada es probada con los pares de entrada-salida reservados y el error promedio observado es tomado como una aproximación de la verdadera media del error cuadrático sobre el espacio de entrada. Este error estimado es una buena aproximación si ambos conjuntos de entrenamiento y prueba reflejan completamente la distribución de probabilidad de los datos en el espacio de entrada [20].

Los métodos *Bootstrap* y *cross-validation* permiten calcular intervalos confiables para las estadísticas de interés y al ser aplicados a redes neuronales implican el uso de mucho más tiempo debido a que se requiere el entrenamiento repetitivo de la red que por sí mismo ya consume una cantidad de tiempo considerable. Por lo tanto, estos métodos ofrecen cierta visión de la confiabilidad de los resultados de la red, lo cual es útil en este trabajo porque es necesario validar de la manera más realista posible el desempeño de nuestros esquemas.

2.4. Ensambles de clasificadores

Cualquier sistema de más de un clasificador puede ser denominado ensamble de clasificadores [27]. La capacidad de aproximación de un ensamble de redes neuronales es mucho mejor que la de alguna de las redes entrenadas; sin embargo, integrar una combinación de estos clasificadores requiere mucho poder de cómputo.

La combinación de clasificadores es actualmente una área activa de investigación en el aprendizaje de máquina y reconocimiento de patrones. Varios estudios de tipo teórico y empírico han sido publicados demostrando las ventajas del paradigma de la combinación sobre modelos de clasificadores únicos. Un buen ensamble se integra de clasificadores precisos y diversos. Un clasificador preciso se define como aquel que posee una mejor precisión que una conjetura aleatoria y la existencia de diversidad en un clasificador implica generar diferentes predicciones sobre nuevos puntos de datos de manera que se definen diferentes errores. Por lo tanto, el entrenamiento de un ensamble consiste en balancear entre la diversidad del error y la precisión individual [28].

La característica principal de un ensamble es la "redundancia" que surge debido a que cada clasificador puede realizar la misma tarea por sí mismo, pero el mejor desempeño de generalización es obtenido cuando se combinan. La estrategia de combinación para un conjunto de clasificadores es muy importante y puede decidir el desempeño del sistema completo. Existen varios métodos para realizar esta tarea, sin embargo dos de ellos se distinguen como los más usados para agregar las salidas de los ensambles de clasificadores. Por un lado se tiene la sumatoria ponderada linealmente de las salidas individuales de cada clasificador como lo indica la expresión 2.8, donde M indica el número de clasificadores, f_i es la salida del i-ésimo clasificador, y w_i es el peso de combinación correspondiente. Este método se conoce como "estrategia de fusión de suma".

$$f_{ens} = \sum_{i=1}^{M} w_i f_i \tag{2.8}$$

Otra estrategia de combinación ampliamente usada en problemas de clasificación define una "votación por mayoría" entre las salidas de cada miembro del ensamble [29].

Se considera un conjunto dado de m pares de entrada-salida $(x^1, t_1), \ldots, (x^m, t_m)$ y se presupone que N redes neuronales son entrenadas usando este conjunto. Suponiendo que se tiene un vector de entrada n-dimensional y una unidad de salida única, se puede denotar por f_i a la función de red computada por la i-ésima red, con $i = 1, \ldots, N$. Por lo tanto, la función de red f generada por el ensamble de redes se define por la expresión 2.9 [20].

$$f = \frac{1}{N} \sum_{i=1}^{N} f_i \tag{2.9}$$

El análisis para este promedio de funciones de red indica que si cada una de las aproximaciones está sesgada con respecto a una parte del espacio de entrada, un promedio sobre el ensamble de la redes puede reducir el error de predicción significativamente. Para cada función de red f_i podemos calcular un vector m-dimensional $\mathbf{e}^i$ cuyos componentes son el error de aproximación de la función f_i para cada par de

entrada-salida. Por lo tanto, el error de aproximación cuadrático Q de la función f del ensamble se define en la ecuación 2.10.

$$Q = \sum_{i=1}^{m} \left(t_i - \frac{1}{N} \sum_{j=1}^{N} f_j(x^i) \right)^2 \tag{2.10}$$

Un aspecto muy importante es que el error cuadrático total del ensamble es más pequeño que el promedio de los errores cuadráticos de las aproximaciones funcionales computadas. Sin embargo, este resultado se mantiene únicamente si se cumple que los residuales de error no están correlacionados, y esto se observa inmediatamente sin que N sea muy grande. Para algunos casos, si $N = 2$ o $N = 3$ se puede alcanzar una mejora significativa sobre las capacidades de aproximación de las redes combinadas. Además, si los errores cuadráticos no están correlacionados, se puede usar una combinación ponderada de las N funciones f_i como lo indica la expresión 2.11 [20].

$$f = \frac{1}{N} \sum_{i=1}^{N} w_i f_i \tag{2.11}$$

Cada red neuronal requiere de su propio conjunto de entrenamiento para dar diversidad al equipo de clasificadores, y cada una puede definirse con arquitecturas diferentes con el mismo objetivo. El hecho de que el conjunto de entrenamiento pueda o no ser aprendido exactamente por una red neuronal, depende de la cantidad de pesos (plasticidad) de la red y de la información empírica usada. Por lo tanto, se puede incrementar la plasticidad que ayudará a disminuir el error de entrenamiento pero esto puede incrementar el error sobre el conjunto de prueba. Además, el decremento excesivo de la plasticidad puede dar lugar a un error grande de entrenamiento y de prueba. Sin embargo, actualmente no existe un método general para determinar la cantidad de parámetros de la red ya que esto depende del problema que se trate [20]. Por lo tanto, la cantidad de grados de libertad de una función de red no debe exceder a los datos por sí mismos, ya que de otro modo se tendrá el riesgo de sobreajuste de la red [30].

La idea de un ensamble de redes neuronales puede decirse que surge con Nilson [31] quien ordenó un grupo perceptrones de una capa y combinó sus salidas con una segunda capa de votación. Posteriormente, surgen dos de las técnicas más ampliamente usadas para crear ensambles, *Bagging* y *Boosting* que son aplicadas con redes neuronales.

A partir de los trabajos de investigación realizados en la combinación de redes neuronales, hoy en día se tiene un conocimiento que nos indica aspectos importantes de este tipo de clasificadores. Iniciar cada red con diferentes pesos iniciales aleatorios incrementará la probabilidad de continuar en una trayectoria diferente a las de otras redes y es esta la manera más común para generar un ensamble aunque hoy en día éste es generalmente aceptado como el último método efectivo para alcanzar diversidad. Brown [28] menciona que se ha investigado la relación entre la inicialización de los vectores de peso finales y los tipos de soluciones encontradas aplicando el algoritmo

de "retro-propagación". En tales trabajos se variaron los vectores de pesos iniciales de las redes neuronales y después se entrenaron con un conjunto de datos fijo resultando redes neuronales diferentes en su estructura pero estadísticamente no independientes en su desempeño de generalización lo que significa que generaron patrones muy similares de generalización a pesar de haber sido derivadas a partir de diferentes vectores de pesos iniciales. Partridge [32, 33] realizó varios experimentos sobre conjuntos de datos muy grandes (>15000 patrones) concluyendo que después del tipo de red, la estructura del conjunto de entrenamiento y la cantidad de unidades ocultas, la inicialización aleatoria de los pesos es el último método más efectivo para generar diversidad. Otras personas compararon en [34] a los métodos de validación cruzada de 10 bloques, *Bagging* y la estrategia de inicializar los pesos de manera aleatoria encontrando que esta última no es suficiente para dar diversidad al ensamble.

Maclin y Shavlik [35] inicializan los pesos de las redes neuronales usando un aprendizaje competitivo para crear redes neuronales inicializadas lejos del origen del espacio de pesos lo cual incrementa el conjunto de mínimos locales alcanzables. Mostraron una mejora significativa de desempeño sobre el método estándar de inicialización en algunos problemas de la vida real. Finalmente, la técnica "Rápido aprendizaje de un ensamble" [36] entrena una red neuronal tomando M imágenes del estado de sus pesos durante el entrenamiento. Estas M imágenes se usan como M miembros del ensamble y, aunque el desempeño no fue muy bueno como al usar redes entrenadas separadamente, esto ofrece la ventaja de un tiempo de entrenamiento reducido como si únicamente fuera necesario entrenar una red.

Muchos métodos intentan producir redes diversas o complementarias proveyéndoles conjuntos de entrenamiento ligeramente diferentes. Éste es quizás el método de entrenamiento del ensamble más ampliamente investigado. Si se desea tener errores diversos en un ensamble, la manipulación de la arquitectura de la red puede ser una opción por lo que tiene sentido usar diferentes tipos de aproximadores de funciones para lograr el objetivo. Partridge [32, 33] indica que la variación de la cantidad de nodos ocultos es, después de los pesos iniciales, el último método para crear diversidad en este tipo de ensambles debido a las similitudes metodológicas en los algoritmos de aprendizaje supervisado. También encontró que la técnica de usar redes de diferente tipo (MLP y funciones base radiales) es más productiva que variar la cantidad de nodos ocultos.

Así, con el objetivo de construir clasificadores más precisos, se han usado algoritmos evolutivos para optimizar las topologías de la red que componen el ensamble [24], se han integrado ensambles monitoreando la diversidad durante el proceso como lo hace el algoritmo CNNE [37] y últimamente los ensambles híbridos están siendo estudiados combinando árboles de decisión, redes neuronales, clasificadores de k vecinos más cercanos y clasificadores de Bayes cuadráticos en un ensamble [38].

Capítulo 3

Red neuronal de cuello de botella modificada

Los experimentos que se describen en este capítulo consisten en definir una "red neuronal de cuello de botella modificada" para reducir la dimensión de los patrones de entrenamiento (en adelante MBNN por *Modified Bottleneck Neural Network*). Nuestro esquema se diferencia de una red de cuello de botella tradicional utilizada para reducir dimensiones, en que usa la información de clase y, por lo tanto, los datos transformados pueden ser usados convenientemente para un problema de clasificación. Dentro de este marco experimental, se propone una nueva técnica para crear ensambles de redes neuronales usando múltiples proyecciones obtenidas con diferentes MBNNs a partir del mismo conjunto de datos. Además, ya que la técnica anterior requiere mucho tiempo de cómputo, se propone una estrategia basada en la clonación de redes neuronales que permite obtener múltiples proyecciones en menos tiempo. En este capítulo se justifica la conveniencia del método propuesto a través de algunos experimentos sobre determinados problemas de clasificación.

La reducción de dimensiones es probablemente unas de las tareas más importantes en cualquier problema de reconocimiento de patrones. Este problema ha sido planteado desde diversos enfoques y usando varios tipos de herramientas. De manera sencilla, la reducción de dimensionalidad se puede definir de la siguiente manera: dado el conjunto de datos de entrada $X = \{x : x \in \Re^p\}$, se desea encontrar una transformación $\Phi : X \to X'$, donde $X' = \{x' : x' \in \Re^q\}$ y $q < p$. Típicamente el conjunto X' optimiza cierto criterio J que usualmente depende del problema que se esté resolviendo. Así, para un problema de clasificación, J puede ser una medida de separabilidad de clase o una tasa de las clasificaciones erróneas para un clasificador determinado.

Dos estrategias importantes de reducción de dimensiones son la selección y extracción de características. La primera consiste en seleccionar únicamente un subconjunto de p características presentes en los puntos de datos de X, mientras que en la segunda estrategia los datos son proyectados en algún espacio con dimensión menor. En el segundo tipo de transformación, las características originales presentes en los datos pueden perder su identidad individual y las características finalmente obtenidas en

X' pueden ser una combinación de todas las características presentes en los puntos de datos de X. Se debe tener claro que el término "extracción de características" es genérico y que significa extraer información de un conjunto dado de características e incluye las transformaciones que decrementan la dimensión de un conjunto dado de datos.

La reducción de dimensiones permite obtener información cuya representación es menos compleja de manera que puede ser aprendida por una máquina de aprendizaje cuya arquitectura también sea menos compleja. Esto permite un ahorro de cómputo en la construcción y uso de la máquina de aprendizaje. Además, la reducción de dimensiones puede dar lugar a precisiones de predicción mejoradas en escenarios donde el tamaño de muestra de entrenamiento es pequeño (lo cual es común en la mayoría de los escenarios de la vida real) [39]. El problema de la reducción de dimensiones ha sido muy tratado en la literatura y ha sido probado con varios paradigmas. Estudios previos en este contexto se enfocan principalmente alrededor de aproximaciones estadísticas como el Análisis de Componentes Principales (*Principal Components Analysis*, PCA) [40], el Análisis Discriminante Lineal (*Linear Discriminant Analysis*, LDA) [41], etc.

Estos métodos intentan reducir la dimensión del espacio de características a través de la creación de nuevos datos que son una combinación de los originales. Por lo tanto, PCA y otros métodos relacionados son técnicas de extracción de características que obtienen un nuevo conjunto a partir de los datos disponibles, y la dimensión del espacio extraído es menor que la del original. Existen muchas variantes de PCA que intentan resolver el problema de proyección usando varias modificaciones de la técnica principal de PCA [42] y su principal desventaja es que los nuevos datos pierden su identidad original y además su significado. Por otro lado, también se han realizado otros trabajos de selección de características usando técnicas estadísticas [43, 44].

Blum y Langley [45] han dado una excelente recopilación de las técnicas de selección relevantes en el aprendizaje de máquina. Estas técnicas se diferencian en la evaluación de subconjuntos de características y se pueden clasificar como técnicas de filtro y de envoltura. En las técnicas de filtro, el índice de evaluación de características es independiente del algoritmo principal de clasificación o de aproximación de funciones; mientras que en las técnicas de envoltura las características son evaluadas por el propio algoritmo principal.

Se considera que las técnicas de envoltura son mejores ya que la relevancia de una característica es generalmente dependiente de la tarea que se esté desarrollando y también de la herramienta que se esté usando para realizar tal tarea [46].

Existen varios algoritmos de selección de características que usan *soft computing* o herramientas de inteligencia computacional. Los métodos descritos en [47, 48] usan algoritmos genéticos para seleccionar los subconjuntos de características relevantes. Los métodos descritos en [49, 50, 51, 52] y varios otros usan redes neuronales para la selección de características. Las técnicas de selección han sido tratadas también usando métodos difusos y neuro-difusos [53, 54].

Nuestra contribución radica en usar la reducción de dimensionalidad para la tarea

de clasificación. A lo largo de este capítulo se discute una técnica para realizar tal reducción usando una red neuronal. Nuestro método es una técnica de extracción de características que proyecta un cierto conjunto de datos dado en un espacio con dimensión menor mientras que preserva la separabilidad de clase de los datos. Nuestra técnica depende de una arquitectura neuronal especial llamada "red neuronal de cuello de botella".

La red neuronal de cuello de botella tradicional ha sido previamente aplicada en la compresión de datos y no usa información de clase presente en un conjunto de datos [55, 56], por lo que la compresión alcanzada es de tipo no supervisada, lo cual puede no ser conveniente para problemas de clasificación. En este capítulo se discute una simple variante de la red de cuello de botella que utiliza la información de clase para clasificación: además, se discuten varias alternativas de aplicar los datos transformados para clasificar información. También se propone una nueva técnica para crear ensambles de redes neuronales considerando datos proyectados obtenidos de una red neuronal y probamos el método en algunos problemas de clasificación de la vida real.

3.1. La red neuronal de cuello de botella

La red neuronal de cuello de botella es una buena estrategia que ha sido usada para la compresión de datos. Se tiene un conjunto de datos $\Xi = \{\xi_1, \xi_2, \ldots, \xi_m\} \subset \Re^p$ que es considerado para entrenar un perceptrón multicapa (MLP) con p nodos en las capas de entrada y de salida, y con q ($q < p$) nodos en una capa oculta. En adelante nos referiremos a tal arquitectura con la notación $p : q : p$. Para el entrenamiento de esta red, se usan pares de entrada-salida de la forma (ξ_i, ξ_i) $(i = 1, 2, \ldots, n)$. Así, la red es entrenada para aprender una correspondencia de identidad, por lo que para una red propiamente entrenada la salida de ξ_i sería el mismo ξ_i. Entonces, la salida de la capa oculta para cada ξ_i sería una representación q-dimensional de ξ_i y por lo tanto se obtiene una reducción de dimensión. Este tipo de red ha sido previamente usada en varias aplicaciones [55, 56].

3.1.1. Red de cuello de botella modificada

Proponemos una variante de la red neuronal de cuello de botella que hemos denominado "red de cuello de botella modificada" (MBNN por *Modified Bottleneck Neural Network*). Establezcamos que $X = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, 2, \ldots, n\}$ es un conjunto de datos de entrenamiento donde $\mathbf{x}_i \in \Re^p$ y $\mathbf{y}_i \in \Re^s$; tal conjunto X será usado para entrenar un MLP. Se debe aclarar que no existe restricción alguna en la arquitectura del MLP excepto que al menos una de las capas ocultas (digamos, la r-ésima capa) contenga q nodos (donde $q < p$); a esta capa la hemos denominado "capa de cuello de botella". A diferencia de la red de cuello de botella tradicional, con MBNN no estamos entrenando a la red para aprender una correspondencia de identidad pero sí se intenta aprender la correspondencia real entrada-salida presente en los datos.

Definamos que B denota una MBNN. B no es diferente a un MLP ordinario en términos de estructura o algoritmo de aprendizaje, únicamente tiene la restricción en su arquitectura como se estableció anteriormente. La salida de B no se encuentra en su capa de salida y sí en su capa de cuello de botella. Por ejemplo, consideremos un conjunto de datos $S = \{(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in \Re^2\}$ que sirve para entrenar una MBNN como la que se muestra en la figura 3.1. La MBNN de la figura 3.1 tiene una arquitectura de 5:3:4:2 considerando a la primera capa oculta (la capa oculta que contiene 3 nodos) como la capa de cuello de botella. También se pudo haber seleccionado la segunda capa oculta como la capa de cuello de botella. La red es entrenada usando cualquier algoritmo de aprendizaje conveniente como el de "retro-propagación" o alguna de sus variantes. Para usar esta red, la salida es obtenida de la primera capa oculta, por lo que para cualquier vector de entrada $\xi \in \Re^5$, la salida de la red sería $\alpha \in \Re^3$.

En general, digamos que B_X es una MBNN entrenada con X y que la r-ésima capa de B_X es la capa de cuello de botella con q nodos, por lo que $\alpha_i \in \Re^q$ denota la salida de la r-ésima capa de B_X para un punto de datos x_i. A partir de una red neuronal entrenada se reunen las salidas de los nodos de la r-ésima capa de manera que la salida total acumulada sirve como una proyección en un espacio q-dimensional de los datos originales de dimensión p. Hemos denominado a estos datos como "datos reducidos" y posiblemente retienen las propiedades requeridas para el aprendizaje de la función que es representada por los datos originales. Esto es porque los datos reducidos representan una configuración interna de la información inmersa en la red, y realmente esta configuración ayuda a la red neuronal en el aprendizaje de la correspondencia apropiada presente en los datos.

El conjunto de datos reducidos obtenido a partir de la red entrenada puede ser usado posteriormente para entrenar una nueva red cuya arquitectura posiblemente sea menos compleja. Esto permitirá un tiempo de entrenamiento menor y posiblemente

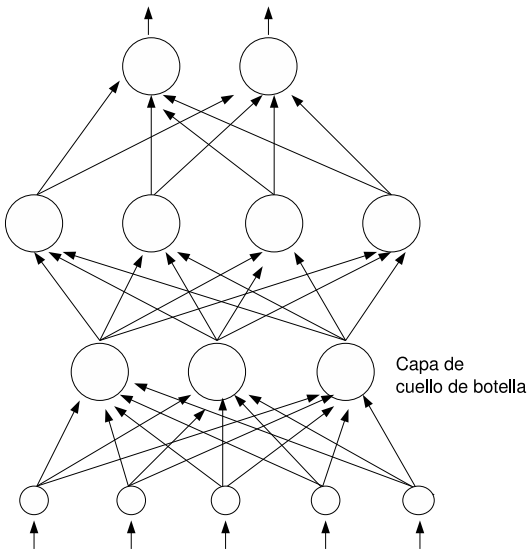


Figura 3.1: Una red de cuello de botella modificada.

una mejor generalización. A continuación se discuten algunas formas en las que pueden ser usados los datos reducidos que se han obtenido a partir de la MBNN.

3.2. ¿Cómo usar los datos reducidos?

Una MBNN transforma un conjunto de datos en un espacio de dimensión menor usando una transformación no lineal. Además, una característica importante de esta transformación es que usa la información de salida (por ejemplo, las etiquetas de clase en el caso de los datos de clasificación) para realizar la transformación. Los datos reducidos representan una configuración interna de una red entrenada la cual posteriormente los transforma en su salida. Así, esta configuración intermedia es conveniente para ser usada en cualquier otra red neuronal. A continuación se mencionan algunos escenarios en los que el conjunto de datos reducidos puede ser útil.

3.2.1. Como una transformación antes del entrenamiento de un MLP usado para la tarea de predicción

Se sabe que conforme crece la dimensión de los datos es necesaria una red más grande para aprender la correspondencia entre los datos. Esto implica que tal red debe tener más parámetros ajustables pero conforme el número de parámetros crece es mayor la posibilidad de que una red se sobreajuste. Como se ha mencionado, una red sobreajustada puede fallar en gran medida para generar resultados aceptables con los conjuntos de datos de prueba. Este problema llega a ser más agudo si el número de muestras de entrenamiento es pequeño por lo que para evitar esto, una práctica común y natural es aplicar una reducción de dimensiones sobre los datos antes de usarlos para entrenar una red. Así, reduciendo la dimensión de los datos originales, su correspondencia puede ser aprendida por una red menos compleja por lo que la posibilidad de sobreajuste llega a ser menor. La técnica más usada es aplicar Análisis de Componentes Principales (PCA) sobre los datos. Ésta es una técnica genérica para reducir dimensiones y proyecta los datos en un nuevo espacio donde tienen su máxima varianza. Pero esto puede no ser el mejor objetivo para la proyección cuando se sabe qué hacer con los datos. Cuando la tarea deseada es clasificación, la separabilidad de clase puede ser un mejor objetivo que maximizar la varianza. Además, si la herramienta elegida es un perceptrón multicapa que funciona de una manera altamente no lineal y produce límites de clase altamente no lineales, el uso de PCA (u otras técnicas de proyección relacionadas como LDA) puede no ser siempre útil. El conjunto de datos reducidos obtenido por una MBNN tendrá mejor representación que los generados por los otros métodos de proyección de datos conocidos en términos de capacidad de clasificación de una red neuronal. La razón de esto es que los datos reducidos generados por una MBNN definen una representación interna de una red neuronal que es significativa para la clasificación de datos.

Este método puede ser criticado porque es necesario entrenar una red para obtener los datos reducidos. Por ende, para los conjuntos de datos de dimensiones enormes,

la MBNN puede por sí misma sobreajustar los datos si necesitamos una red cuyo desempeño sea bueno sobre los datos de entrenamiento. Esto es verdad, sin embargo, dado que la MBNN no está siendo usada para propósitos de predicción; una MBNN sobreajustada puede no ser problemática. Así, extraemos los datos reducidos de baja dimensión a partir de una MBNN y entrenamos una nueva red (posiblemente mucho más pequeña que la propia MBNN), por lo tanto la red que nosotros usamos para la predicción tendrá mejor capacidad de generalización.

3.2.2. Para construir ensambles de redes neuronales

Los métodos de ensambles como *Bagging* [5] y *Boosting* [57] pueden incrementar considerablemente la habilidad de predicción de cualquier clasificador. Como ya hemos mencionado, los ensambles de redes neuronales tienen mejor desempeño que redes aisladas. *Bagging* es un método muy conveniente para redes neuronales en términos de desempeño, ya que *Bagging* puede decrementar la varianza de predicciones en los clasificadores, y las redes neuronales que son conocidas por ser "inestables"tienen una varianza elevada en predicción [5]. Para redes neuronales, *Bagging* implica crear múltiples muestras por medio de *Bootstrap* a partir de un conjunto de datos dado y usarlas para entrenar múltiples redes neuronales. El resultado final es obtenido por la agregación conveniente de las salidas de las redes que son miembros del ensamble. Así, un ensamble de redes neuronales creado con *Bagging* implica usar mucho tiempo de cómputo, ya que el entrenamiento de una red neuronal requiere un tiempo considerablemente alto. Usando una red neuronal de cuello de botella $\mathcal{BN}$ uno puede proyectar los datos a un espacio con menor dimensión y usar muestras *Bootstrap* de los datos proyectados para entrenar miembros individuales del ensamble. Esto puede dar ahorros considerables en tiempos de entrenamiento.

3.2.3. Métodos que usan estimación de densidad implícita o explícita

Existen varios trabajos que indican que una de las razones del sobreajuste de los MLPs es debido a que éstos son entrenados con una muestra de entrenamiento finita [58, 59, 60]. Un MLP se entrena con un algoritmo de aprendizaje convencional usando una muestra fija de puntos de entrenamiento en cada época. En [59] se indica que el sobreajuste se puede reducir si el conjunto de entrenamiento se expande indefinidamente y se propone entrenar una red con un nuevo conjunto de datos en cada época. Para lograr esto usaron un método para estimar la densidad de probabilidad de los datos con el objetivo de generar un nuevo conjunto de datos con la misma densidad de probabilidad en cada época; por lo tanto el MLP encuentra nuevos puntos en cada época. En [60], el método reportado en [59] fue modificado mejorando el procedimiento para la estimación de la densidad. En [58] se propuso otro método para expandir el conjunto de entrenamiento, el cual no depende de la estimación de densidad explícita pues usa una heurística de k-NN para generar nuevos puntos en cada época.

Todos estos métodos reportados en [58, 59, 60] pueden reducir en buen grado el sobreajuste en las redes MLP. Sin embargo, en [59, 60], donde un tipo de estimación de densidad de núcleo es considerada usando núcleos Gaussianos, la corrección de la estimación de densidad decrece con el incremento de la dimensión de los datos. En [58] se usa una técnica K-NN y también falla para datos con dimensión alta. Una solución fácil a este problema puede ser usar el conjunto de datos reducido obtenido de una MBNN (y no el conjunto de datos original) para el entrenamiento y estimación de la densidad.

3.2.4. En *Bagging* con múltiples proyecciones del mismo conjunto de datos

La idea detrás de *Bagging* es crear clasificadores independientes a partir del mismo conjunto de datos, por lo que cada muestra de *Bootstrap* actúa como un conjunto de datos diferente para cada clasificador. Aquí proponemos usar las diferentes proyecciones del mismo conjunto de datos para entrenar cada miembro del ensamble. Se propone usar la MBNN para proyectar los datos.

Primero entrenamos k MBNNs $B_1, B_2, \ldots, B_k$ con cierto conjunto de datos $X = (\boldsymbol{x}_i, \boldsymbol{y}_i) : i = 1, 2, \ldots, n, \boldsymbol{x} \in \Re^p, \boldsymbol{y} \in \Re^s$. Cada B_i puede tener diferentes arquitecturas con distintas cantidades de capas ocultas y de nodos ocultos. La restricción para cada B_i consiste en que la capa de entrada debe contener p nodos, la capa de salida debe contener s nodos y cualquier capa oculta debe contener q $(q < p)$ nodos. Entonces cada una de las redes permite obtener una proyección diferente de los datos originales X, posiblemente con diferentes dimensiones. Así, usando cada B_i obtenemos la transformación $B_i : X \rightarrow X_i'$. Posteriormente entrenamos k nuevas redes neuronales usando los conjuntos de datos X_i' para $i = 1, \ldots, k$. Creamos un ensamble de estas k redes, y agregamos la salidas finales usando una función de agregación adecuada.

3.2.5. En *Bagging* con múltiples proyecciones generadas con clonación de redes

El entrenamiento de las redes $B_1, \ldots, B_k$ implica mucho tiempo. Esto, prácticamente deja sin factibilidad la parte de reducción de dimensiones para construir un ensamble, por lo que se desea disminuir su costo computacional. Proponemos un nuevo método que permite aminorar el tiempo que se usa en la etapa de reducción y consiste en re-entrenar las redes neuronales que se han definido a partir de una red MBNN entrenada completamente.

Usando el conjunto de datos X definido previamente, se entrena una MBNN B_1. Posteriormente, se definen $C_1, \ldots, C_k$ redes neuronales con arquitecturas variables que definen sus pesos iniciales con los valores de los pesos de B_1 y con valores aleatorios para las posibles aristas nuevas de la arquitectura. Cada una de estas k redes se define con una arquitectura diferente de la siguiente manera: dada una arquitectura para B_1 con una capa oculta de q nodos, la red C_i se definirá por medio de una capa oculta

Tabla 3.1: Resumen de los conjuntos de datos

Datos	Fuente	Número de puntos	Número de características	Número de clases
Lung Cancer	[61]	32	56	3
Sonar	[61]	208	60	2
Iono	[61]	351	34	2
DNA	[61]	2000	180	3
Protein	[62]	698	125	4

con $q + a$ nodos donde a se selecciona aleatoriamente a partir del conjunto $\{1, 2, 3\}$. Por lo tanto, dado que la red C_i implica un conjunto de pesos posiblemente mayor, hemos establecido que la mayoría de los valores de sus pesos se definan a partir de los ya establecidos por la red B_1 y el resto con valores aleatorios. De esta manera, se define un conjunto C de clones de B_1 cuyas arquitecturas varían y posteriormente se re-entrena cada clon con una cantidad menor de épocas que las usadas para B_1 (entre el 5 % y 10 %). Es fácil entender que las nuevas redes definen diferentes proyecciones de los datos originales y que tales proyecciones representan diferentes dimensiones que son menores a la que originalmente define a los patrones de entrenamiento. Por lo tanto, esta estrategia permite establecer diferentes representaciones con menor dimensión a la de los datos originales. Esto se logra porque el re-entrenamiento de los clones genera diferentes proyecciones que contienen información muy similar.

El ahorro de tiempo usando este método es significativo comparado con el método anteriormente detallado que implica entrenar redes neuronales completamente. Además, el uso de esta técnica proporciona mayor información a las redes miembro del ensamble gracias a la variabilidad de las arquitecturas de los clones. Así, el entrenamiento de las redes neuronales del ensamble se realiza de una manera más rápida y más precisa que en la estrategia anterior.

3.3. Resultados experimentales

Estudiamos la conveniencia de este método aplicándolo a algunos problemas de clasificación de la vida real. Los conjuntos de datos usados son Sonar, Iono, Lung Cancer, DNA y Protein. El resumen de los conjuntos de datos se presenta en la tabla 3.1. En el caso de los datos de Proteínas, los datos originales contienen 313 puntos de entrenamiento y 385 puntos de prueba. Se aplica una validación cruzada de 10 bloques en estos experimentos.

En todas las simulaciones realizadas hemos usado las herramientas de redes neuronales incluidas en MATLAB. Usamos la función de aprendizaje *traingdx* como algoritmo de entrenamiento, que es una variante del algoritmo de "retro-propagación" con la habilidad de ajustar automáticamente la tasa de aprendizaje cuando sea necesario.

Tabla 3.2: Resultados en Lung Cancer

	Métodos	Desempeño (%)
1	MBNN Individual	80.167 ± 4.158
2	*Bagging* con MBNN	80.667 ± 5.230
3	*Bagging* de MP con MBNN	83.778 ± 3.674
4	PCA Individual	78.167 ± 3.639
5	*Bagging* con PCA	75.833 ± 4.245
6	RN Individual con datos completos	77.167 ± 5.650
7	*Bagging* con datos completos	82.440 ± 2.194

Tabla 3.3: Resultados en Sonar

	Métodos	Desempeño (%)
1	MBNN Individual	82.114 ± 3.048
2	*Bagging* con MBNN	82.193 ± 3.251
3	*Bagging* de MP con MBNN	85.793 ± 0.617
4	PCA Individual	77.679 ± 1.488
5	*Bagging* con PCA	78.786 ± 1.176
6	RN Individual con datos completos	79.621 ± 2.869
7	*Bagging* con datos completos	84.543 ± 0.889

Las tablas 3.2, 3.3, 3.4, 3.5 y 3.6 resumen los resultados obtenidos. Cada renglón en las Tablas se refiere a un método diferente y las columnas reportan el desempeño promedio (con desviación estándar), en porcentaje, sobre los datos de prueba. Cada resultado reportado implica una validación cruzada de 10 bloques repetida 20 veces.

Los resultados obtenidos al aplicar nuestras estrategias para generar nuevas proyecciones se muestran en las tablas 3.7, 3.8, 3.9, 3.10 y 3.11 con el objetivo de compararlas directamente. Se muestran únicamente los dos métodos construidos cuyos resultados, en precisión, son comparables con el método clásico *Bagging*. El tiempo de cada estrategia se muestra en segundos (s). Como se aprecia en estas tablas, el primero de nuestros métodos requiere mucho más tiempo que nuestra segunda estrategia. Además, con esta última podemos generar proyecciones con mayor información ya que se tienen diferentes reducciones de dimensión de los datos originales. A continuación se explica con más detalle cada método indicados en los renglones de las tablas. Los resultados de la mayoría de los métodos que se explican se muestran en las primeras 5 tablas de resultados. Únicamente los métodos "*Bagging* de MP con clonación MBNN" y "*Bagging* de MP con MBNN" (que también se considera en las primeras tablas) se comparan entre sí en la tablas posteriores porque el primero de estos métodos es una variante del segundo, con ciertas mejoras.

1. **MBNN Individual**: Entrenamos una MBNN con 10 nodos en la capa oculta para obtener datos reducidos de dimensión 10. Se entrenaron 20 redes diferentes con el mismo conjunto de datos reducidos, cada una teniendo la misma arqui-

Tabla 3.4: Resultados en Iono

	Métodos	Desempeño (%)
1	MBNN Individual	89.979 ± 0.999
2	*Bagging* con MBNN	90.294 ± 1.164
3	*Bagging* de MP con MBNN	90.834 ± 0.262
4	PCA Individual	86.725 ± 0.753
5	*Bagging* con PCA	86.158 ± 0.686
6	RN Individual con datos completos	87.839 ± 1.749
7	*Bagging* con datos completos	91.061 ± 0.960

Tabla 3.5: Resultados en DNA

	Métodos	Desempeño (%)
1	MBNN Individual	92.425 ± 0.675
2	*Bagging* con MBNN	92.490 ± 0.561
3	*Bagging* de MP con MBNN	94.585 ± 0.094
4	PCA Individual	90.720 ± 0.305
5	*Bagging* con PCA	90.600 ± 0.286
6	RN Individual con datos completos	89.300 ± 0.177
7	*Bagging* con datos completos	94.800 ± 0.262

Tabla 3.6: Resultados en Protein

	Métodos	Desempeño (%)
1	MBNN Individual	76.338 ± 1.451
2	*Bagging* con MBNN	76.400 ± 1.799
3	*Bagging* de MP con MBNN	79.558 ± 0.459
4	PCA Individual	63.714 ± 1.279
5	*Bagging* con PCA	64.400 ± 1.385
6	RN Individual con datos completos	69.763 ± 4.120
7	*Bagging* con datos completos	78.000 ± 0.590

tectura con 10 nodos en la única capa oculta. El renglón 1 de las tablas muestra el desempeño promedio de esas 20 redes.

2. ***Bagging* con MBNN**: Se aplicó *Bagging* con 10 miembros en el ensamble, usando los mismos datos reducidos de dimensión 10 obtenidos con el método del renglón uno. Usamos "votación por mayoría" para agregar las salidas de los 10 miembros diferentes. Se crearon 20 ensambles diferentes usando los mismos datos reducidos. El renglón 2 muestra el desempeño promedio de tales 20 ensambles.

3. ***Bagging* de MP con MBNN**: *Bagging* de MP significa *Bagging* con múltiples proyecciones. En este experimento entrenamos 10 MBNNs diferentes, cada una con 10 nodos en la capa de cuello de botella para obtener así 10 proyecciones diferentes de dimensión 10 de los datos originales. Con las proyecciones generadas entrenamos 10 redes diferentes con 10 nodos en una única capa oculta y agregamos sus resultados usando "votación por mayoría". Este experimento se realizó 20 veces y los resultados se resumen en el renglón 3 de las tablas correspondientes.

4. ***Bagging* de MP con clonación MBNN(i)**: Esta variante implica aplicar *Bagging* entrenando las redes que integran el ensamble usando diferentes proyecciones para cada una de ellas. Tales proyecciones se obtienen re-entrenando clones obtenidos a partir de una MBNN B_1 entrenada completamente; estos clones poseen arquitecturas ligeramente variables específicamente en su capa oculta. Se entrenó una MBNN con 10 nodos en la capa oculta para obtener una dimensión reducida (con tamaño 10) de los datos originales. Posteriormente, se crean 10 redes clones MBNN de B_1 con arquitecturas diferentes definidas con 10, 11, 12 o 13 nodos en la capa oculta. La cantidad de nodos agregados se indica por (i). Tales clones inicializan la mayoría de sus pesos usando los valores establecidos por el entrenamiento de B_1 y el resto se define de manera aleatoria. Finalmente se re-entrenan tales clones para obtener una proyección con cada uno. Las proyecciones obtenidas sirven para entrenar cada uno de los elementos del ensamble al momento de aplicar *Bagging*.

5. **PCA individual**: Se aplicó PCA usando los datos originales completos (con todas sus características) y tomamos los 10 componentes más significativos para reducir la dimensión de los datos originales a una dimensión de 10 nuevas características. El renglón 4 de las tablas muestra el mejor desempeño y el desempeño promedio de los 20 MLPs cuya arquitectura consiste de una única capa oculta de 10 nodos. Estas redes fueron entrenadas con los datos de 10 características obtenidos con PCA.

6. **Bagging con PCA**: Los datos de dimensión 10 obtenidos con PCA son también usados para entrenar un ensamble de 10 MLPs (cada uno con 10 nodos en su única capa oculta) por medio de *Bagging*. El renglón 5 de las tablas muestra el

Tabla 3.7: Otros resultados en Lung Cancer

	Métodos	Desempeño (%)	Tiempo de reducción (s)	Tiempo en *Bagging* (s)
1	*Bagging* de MP con MBNN	83.777±3.674	9.110±0.020	5.144±0.117
2	*Bagging* de MP con clonación MBNN (+1)	84.833±3.244	7.828±0	6.817±0.123
3	*Bagging* de MP con clonación MBNN (+2)	82.916±4.422	7.872±0	6.922±0.106

Tabla 3.8: Otros resultados en Sonar

	Métodos	Desempeño (%)	Tiempo de reducción (s)	Tiempo en *Bagging* (s)
1	*Bagging* de MP con MBNN	85.793±0.616	345.650±0	103.505±5.086
2	*Bagging* de MP con clonación MBNN (+1)	81.571±0.605	194.897±5.758	120.078±3.991
3	*Bagging* de MP con clonación MBNN (+2)	83.303±0.638	192.106±0	93.518±3.430

mejor desempeño y el desempeño promedio de los 20 ensambles creados con los datos de dimensión 10 obtenidos mediante PCA.

7. **Red neuronal individual con datos completos**: El renglón 6 muestra los resultados obtenidos al aplicar un MLP ordinario sobre los datos con todas las características.

8. ***Bagging* con datos completos**: El renglón 7 muestra los resultados generados al aplicar *Bagging* usando los conjuntos de datos con dimensión completa.

Los resultados muestran claramente que los datos reducidos retienen la separabilidad de clase para todos los conjuntos de datos con los que se experimentó. Para todos los conjuntos de datos, nuestro esquema "MBNN individual" da un mejor resultado que una red neuronal individual entrenada con los datos de dimensión completa. Los resultados obtenidos usando PCA para la reducción de las dimensiones son significativamente peores que los resultados obtenidos usando "MBNN individual". La aplicación de "*Bagging* de MP con MBNN" da resultados muy alentadores en todos los casos. De hecho, los resultados obtenidos por "*Bagging* de MP con MBNN" superan a todos los otros métodos y son completamente comparables con los resultados obtenidos por *Bagging* sobre el conjunto de datos con dimensión completa. La

Tabla 3.9: Otros resultados en Iono

	Métodos	Desempeño (%)	Tiempo de reducción (s)	Tiempo en *Bagging* (s)
1	*Bagging* de MP con MBNN	90.834±0.262	97.890±0	101.135±2.851
2	*Bagging* de MP con clonación MBNN (+1)	90.103±0.224	89.073±25.197	124.006±3.672
3	*Bagging* de MP con clonación MBNN (+2)	91.560±0.191	67.478±0	138.146±3.550

Tabla 3.10: Otros resultados en DNA

	Métodos	Desempeño (%)	Tiempo de reducción (s)	Tiempo en *Bagging* (s)
1	*Bagging* de MP con MBNN	94.600±0.094	2765.100±0	353.400±13.759
2	*Bagging* de MP con clonación MBNN (+1)	93.140±0.287	659.040±0	411.029±14.061
3	*Bagging* de MP con clonación MBNN (+2)	93.357±0.055	678.830±0	306.711±5.944

Tabla 3.11: Otros resultados en Protein

	Métodos	Desempeño (%)	Tiempo de reducción (s)	Tiempo en *Bagging* (s)
1	*Bagging* de MP con MBNN	79.558±0.459	774.930±0	194.964±8.139
2	*Bagging* de MP con clonación MBNN (+1)	76.525±0.131	406.235±0	333.422±11.286
3	*Bagging* de MP con clonación MBNN (+2)	76.088±0.180	454.787±0	326.144±11.589
4	*Bagging* de MP con clonación MBNN (+3)	77.292±0.122	308.475±0	265.669±1.899

clonación de una MBNN entrenada completamente evidentemente es una buena estrategia para obtener proyecciones diferentes de los datos originales porque implica menos tiempo que si se entrena cada red cuello de botella y porque significa más información retenida. Por lo tanto, la clonación de redes cuello de botella ofrece una alternativa importante para construir ensambles de redes neuronales en términos de precisión.

3.4. Conclusión

A lo largo de este capítulo hemos presentado una simple modificación de la red tradicional de cuello de botella para la reducción de dimensiones. También hemos mostrado algunas formas específicas para usar los datos reducidos obtenidos a partir de una MBNN. El método de *Bagging* de múltiples proyecciones parece trabajar muy bien como un método para crear ensambles de redes neuronales. La nueva estrategia para generar múltiples proyecciones con base en el entrenamiento de una MBNN resulta interesante por el ahorro de tiempo obtenido comparado ante el requerido por la estrategia de generación de proyecciones diferentes mediante el entrenamiento de varias MBNNs. Los métodos son probados en algunos problemas de clasificación de la vida real y los resultados obtenidos son alentadores.

Capítulo 4

Ensambles de redes neuronales usando adición de ruido

En este capítulo, se describe el trabajo realizado con el objetivo de definir un nuevo método para crear ensambles de redes neuronales usando adición de ruido sobre la arquitectura de la red neuronal. Ya hemos mencionado en el capítulo previo la conveniencia de *Bagging* para redes neuronales en términos de desempeño. Pretendemos crear ensambles con el mismo desempeño que *Bagging* sin requerir tanto tiempo. *Bagging* requiere entrenar múltiples redes neuronales para crear el ensamble por lo que tiene un costo temporal excesivo. Se presenta a continuación una serie de experimentos diseñados a partir de la idea original que consiste en generar diferentes redes a partir de una que ya ha sido entrenada de manera clásica. Las redes generadas se usan posteriormente para integrar el ensamble.

Un conjunto de entrenamiento $\mathcal{L}$ contiene los datos $\{(\mathbf{x}_n, \mathbf{y}_n), n = 1, ..., N\}$, donde $\mathbf{x}$ es un vector de características y $\mathbf{y}$ define la respuesta numérica o etiqueta de clase asociada al vector. Existen procedimientos disponibles en la literatura que usan este conjunto de entrenamiento $\mathcal{L}$ para formar una función clasificadora $\phi(\mathbf{W}, \mathbf{x})$ [5, 57, 2]. Donde $\mathbf{W}$ es un vector de parámetros que es definido a partir del uso de $\mathcal{L}$. Un procedimiento común para obtener el clasificador ϕ consiste en entrenar una red neuronal (como un MLP) con $\mathcal{L}$. En este caso, $\mathbf{W}$ consistiría de los parámetros (pesos y sesgos) del MLP que son aprendidos a partir de $\mathcal{L}$ mediante un procedimiento de optimización que selecciona al vector $\mathbf{W}$ de tal manera que minimiza el error sobre $\mathcal{L}$ [18]. Leo Brieman en [63] observó que los clasificadores de tipo red neuronal son inestables, es decir, no es necesario que para una red neuronal, pequeños cambios en la entrada produzcan pequeños cambios en la salida. Esta situación puede llegar a ser incluso peor cuando el tamaño del conjunto de entrenamiento no es adecuado. En [63] también se observó que además de las redes neuronales, otros métodos muy conocidos como los árboles de clasificación son también inestables. En [5] ha sido demostrado que *Bagging* puede mejorar la estabilidad y precisión de predicción siempre que existan cambios notables en los conjuntos de entrenamiento y que los clasificadores sean inestables.

Se han realizado numerosos intentos para incrementar el desempeño de los algoritmos de aprendizaje de máquina y una de las técnicas usadas son los métodos de creación de ensambles [1, 2]. Ya hemos mencionado que se ha mostrado de varias maneras que una mejor estimación que conlleva un menor error de generalización puede ser obtenida si agregamos las salidas de algunos clasificadores [3]. Mucha gente ha estudiado los métodos de ensamble [4, 5, 57] y los dos paradigmas más importantes que se mantienen son *Bagging* y *Boosting*. Nosotros pretendemos considerar la estrategia de *Bagging* para diseñar un nuevo esquema. Tal estrategia consiste en crear múltiples muestras con *Bootstrap* $\{\{\mathcal{L}\}^{\mathcal{B}}\}$ [5, 64] a partir de $\{\mathcal{L}\}$ para formar múltiples clasificadores a partir de ellas. Su salida final es obtenida a través de una agregación conveniente de las salidas de los clasificadores que depende del tipo de salida; es decir, si es una respuesta numérica o una etiqueta de clase. En términos de costo computacional, el uso de *Bagging* en el aprendizaje de múltiples clasificadores neuronales parece no ser óptimo dado que el entrenamiento neuronal es computacionalmente caro.

Las redes neuronales presentan problemas para los métodos actuales pero ha sido mostrado que estos sistemas, individualmente, nos dan clasificadores con alta precisión [17]. Estudios recientes indican que los ensambles pueden dar resultados más interesantes para mejorar el desempeño que ofrecen los métodos actuales [27, 65, 9, 66, 15, 16, 67], por lo que se han propuesto nuevos algoritmos para combinar clasificadores, se han realizado estudios de diversos tipos y se ha escrito mucho acerca de las características del ensamble [8, 57, 11, 4]. Hoy en día, el trabajo de investigación en torno a la construcción de ensambles se basa en la idea de entrenar un clasificador individual sobre un diferente subconjunto de datos o subconjunto de características de tal manera que cada uno defina salidas diferentes. Este mismo trabajo ha establecido un acuerdo que indica que los elementos del ensamble deben ser diferentes para permitir diversidad en la combinación sin tener definidos métodos formales que indiquen que esto sea totalmente cierto.

Hemos creado diferentes esquemas para construir ensambles de redes neuronales. En la sección 4.2 se describe el esquema que únicamente usa la técnica de adición de ruido. Nuestro método que considera un análisis de sensibilidad se detalla en la subsección 4.3.1. Además, en la subsección 4.3.2 se explica la generación de ensambles usando una heurística de poda de redes neuronales basada en análisis de sensibilidad. Finalmente, la sección 4.4 describe otros métodos para crear ensambles considerando y no la inclusión de dos redes base en la combinación de las salidas de los clasificadores. También, en esta sección consideramos un esquema que selecciona los clones contemplando la medida de error MSE.

4.1. Nuestra propuesta

Se propone un esquema que permita la factibilidad de los ensambles de redes neuronales en el contexto temporal. Nuestros estudios buscan mejorar el desempeño de un clasificador individual mediante la definición de un método de construcción de

ensambles cuyo desempeño sea comparable con el de *Bagging*.

1. Punto de partida: nuestro esquema requiere de un punto inicial a partir del cual se construya el ensamble. Este punto inicial es una red neuronal entrenada sobre un conjunto de datos de entrenamiento determinado; a tal red la hemos denominado "red base".

2. Operador de clonación: es necesario definir un operador que permita generar los elementos que integrarán el ensamble. Se desarrollaron varias estrategias para realizar esto a través de la adición de ruido con distribución normal a los parámetros (pesos y sesgos) de la red neuronal. Una de las estrategias usadas implica realizar un análisis de sensibilidad para modificar los parámetros.

3. Operador de selección: un problema inherente en la definición de nuestro esquema implica establecer si se debe o no considerar un criterio de selección para indicar los clones neuronales que integrarán el ensamble.

4. Operador de agregación: la estrategia de combinación de las salidas de los clasificadores se ha considerado también como una característica a definir en el método propuesto.

Durante nuestros estudios hemos diseñado varias estrategias para adicionar ruido a la red base con el objetivo de generar clones neuronales y construir un ensamble a partir de éstos. Nuestros experimentos sugieren que este método puede aminorar considerablemente el tiempo que se consume en *Bagging*. Nuestro método puede mejorar la precisión de la red base pero su desempeño es siempre un poco menor que el de *Bagging*.

A continuación se describen los tres enfoques generales que han sido estudiados. Se consideran los esquemas de "Adición Simple de Ruido", "Adición de Ruido Usando Análisis de Sensibilidad" y "Adición de Ruido a Dos Redes Base". Todos nuestros esquemas, excepto que se especifique explícitamente, contemplan las clasificaciones erróneas como medida de error para seleccionar las redes del ensamble y para obtener el desempeño final de la combinación.

4.2. Esquema mediante adición simple de ruido

Este enfoque corresponde a la idea original y será usado para mostrar la arquitectura de los esquemas que proponemos. Aunque todos nuestros esquemas poseen la misma estructura, los esquemas posteriores se describen brevemente.

4.2.1. La Estrategia

Se propone un esquema con el objetivo de crear ensambles de redes neuronales usando menos recurso temporal y sin perder la precisión de los métodos actuales. Esto

implica generar múltiples copias a partir de una única red entrenada para obtener nuevas redes. Supongamos que $\mathcal{N}(\mathbf{W})$ es un MLP entrenado usando un conjunto de entrenamiento $\mathcal{L}$ y lo hemos denominado como "red base". Aquí, $\mathbf{W}$ es un vector de todos los parámetros del MLP que se pueden aprender. Así, si la red contiene m pesos y n sesgos, entonces $\mathbf{W}$ tendrá $p = m + n$ elementos. Entonces, $\mathbf{W} = (w_1, w_2, ..., w_p)^T$ si se contempla al sesgo como un peso. Nuestra idea inicial indica que una perturbación pequeña del vector de parámetros generará una configuración diferente, cuyo desempeño sería comparable con el de la red base. Entonces, sería posible crear un ensamble de estas redes degradadas para definir un clasificador final.

Consideremos un conjunto de redes degradadas $S = \{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_p\}$, en el que cada $\mathcal{N}_i$ es una versión degradada de la red base $\mathcal{N}$. De todas estas p redes en S necesitamos seleccionar los mejores candidatos y esto se realiza considerando el error de entrenamiento de cada red dado que no conocemos el conjunto de prueba. Así, si se tiene una red degradada cuyo error es cercano al de la red base (atendiendo un umbral), entonces será insertada dentro del ensamble. Posteriormente, las redes son validadas usando los mismos datos de prueba.

Hemos presentado la idea general, y durante el trabajo realizado se desarrollaron algunas variantes.

4.2.2. Red neuronal base

Se requiere de un elemento inicial que permita la generación de los clasificadores que integrarán el ensamble. Este elemento es una red neuronal denominada *base* que simplemente es entrenada con un conjunto de datos determinado. Hemos usado una red MLP para definir nuestra base y una vez entrenada conocemos su vector $\mathbf{W}$. El entrenamiento de este clasificador implica cierto tiempo que depende también de la dificultad del conjunto de datos.

Entrenando la red base

La red base $\mathcal{N}(\mathbf{W})$ es entrenada por cualquier algoritmo de entrenamiento, ya sea aquellos que incluyan el gradiente descendente o cualquier otra variante de éste. Nuestros estudios incluyen esquemas que usan los métodos de entrenamiento *traingdx* y *trainlm*. El primero es una variante del algoritmo de "retro-propagación" con la habilidad para ajustar automáticamente la tasa de aprendizaje cuando sea necesario y el segundo realiza la actualización de parámetros mediante la optimización de Levenberg-Marquardt que da una solución al problema de minimización de una función sobre su espacio de parámetros.

4.2.3. Operador de clonación

A partir de $\mathcal{N}(\mathbf{W})$ se generan otras redes neuronales modificando el vector $\mathbf{W}$. La idea es perturbar la configuración de parámetros original de manera que las nuevas configuraciones obtenidas permitan definir diferentes redes neuronales con conoci-

miento propio. La intención es tener clasificadores distintos y para esto se consideraron varias estrategias definidas por la función $W' = f(W)$. A continuación se describen las estrategias para adicionar ruido más prometedoras. Todas ellas son muy similares, únicamente varían en la manera y lugar considerado para añadir el ruido a la configuración de la red base.

1. Se generan dos valores $r(w)^l, r(b)^l \sim N(0, \sigma^2)$ por cada capa l, para indicar un mismo ruido por capa que se sumará a sus pesos y sesgos respectivos. Para cada parámetro i en la capa l, se añade el ruido generado (en este caso definido por r) para la capa l atendiendo a las expresiones w_i^l'$= w_i^l + r(w)^l$ o w_i^l'$= w_i^l + r(b)^l$.

   a) Una variante de esta estrategia considera la misma generación de ruido pero esta vez la adición ocurre de manera proporcional al valor original del parámetro i en la capa l como sigue: w_i^l'$= w_i^l + w_i^l \times r$, donde $r \in \{r(w)^l, r(b)^l\}$.

2. Nuevamente se añade ruido de manera proporcional al valor del parámetro original. Un único valor $r \sim N(0, \sigma^2)$ se genera para todos los parámetros de la red con el objetivo de definir la proporción que se usará para adicionar ruido atendiendo la expresión w_i^l'$= w_i^l + w_i^l \times r$, para toda capa l y todo parámetro i.

3. Es una variante de la estrategia anterior, la cual consiste en que para cada parámetro i se genera un valor de $r_i \sim N(0, \sigma^2)$.

4. Para cada parámetro i de la red se genera un valor $r_i \sim N(0, \sigma^2)$ y se suma de manera directa como sigue: $w_i' = w_i + r_i$.

5. Para cada peso i se genera un valor $r_i \sim N(0, \sigma^2)$ que define la proporción del peso original que se adicionará como ruido. Tal valor r_i es útil para definir los nuevos valores de los pesos j que estén conectados al peso i. Para el sesgo de cada neurona de la capa oculta se define un valor r_i y para los sesgos en la capa de salida existe un único valor r.

6. Es una variante de la estrategia anterior que consiste en definir un valor r_i diferente para cada sesgo de la capa de salida.

En todas las estrategias mencionadas, σ es el parámetro importante que debe ser seleccionado. La cantidad de degradación que una red puede alcanzar depende fuertemente del valor de σ que se use. Un valor de σ grande destruirá la configuración de pesos de la red base y por lo tanto los clones serán muy diferentes de la red base. Si usamos un valor muy pequeño para σ entonces los clones serán muy similares y por lo tanto el ensamble tendrá muy poca o nula diversidad. Es probable que σ sea dependiente de los datos, pero no nos ha sido posible identificar un método general que pueda calcular un valor de σ conveniente a partir de un conjunto de datos. Nuestra experiencia con los experimentos en varios conjuntos de datos para clasificación, confirma que un valor pequeño de σ (del orden de 0.003) puede funcionar

adecuadamente para todos los conjuntos de datos. Mayores detalles relacionados con la selección de σ pueden consultarse en la subsección 4.5.3.

4.2.4. Operador de selección

Se sugiere que cada clasificador del ensamble se seleccione mediante un criterio que filtre sólo aquellos clones neuronales cuyo error de entrenamiento sea cercano al de la red neuronal base. Tal error lo hemos medido mediante la cantidad de clasificaciones erróneas, con SSE (*Sum Square Error*) o MSE (*Mean Square Error*). Se sugiere definir un umbral sobre el error de la red base y considerar a todos aquellos clones neuronales cuyo error esté bajo este umbral.

Este operador es fundamental para el funcionamiento del esquema pues la adición de ruido puede llevar a configuraciones que resulten en mejores desempeños y que no puedan ser consideradas mediante la estrategia de selección definida. Aquí existen varios detalles que contemplar ya que si se usa la medida de error de clasificaciones incorrectas los clones seleccionados son muy similares causando que no se tenga mucha diversidad pues los miembros del ensamble definen errores muy parecidos. Por otro lado, si se intenta obtener clones cuyos errores difieran más, entonces la selección es lenta. En contraste, si se contempla el uso de alguna medida de error como el *SSE* o *MSE*, la selección de clasificadores es mucho más rápida porque se usan medidas con mayor precisión real en contraste con la definición entera de las clasificaciones erróneas. Sin embargo, el problema surge nuevamente debido a que no existe la diversidad apropiada. Cualquier medida que se use origina un ensamble con desempeño final muy similar, en clasificaciones incorrectas, al de la red base y, por lo tanto, no es comparable con el que ofrecen los métodos clásicos para crear ensambles.

Se observó que mientras un clasificador tiene en promedio un valor *SSE* menor al de la red base, su cantidad de clasificaciones erróneas es, en promedio, ligeramente menor a la de la red base lo que implica que las configuraciones de parámetros de los clones pueden ser numéricamente diferentes pero los desempeños que representan son muy similares al de la red base.

La idea básica que consiste en la simple adición de ruido mediante alguna de las estrategias mencionadas no dio buenos resultados debido a que los clones generados poseen una definición muy similar y esto evita la diversidad necesaria en el ensamble. Si bien la red base en general siempre es superada por el ensamble de clones, su desempeño no compite con *Bagging*. El esquema propuesto usando la idea básica se define en la figura 4.1 y requiere de la estrategia de adición de ruido enumerada como 1. Este esquema se ha denominado como "Clonación con Simple Adición de Ruido (CSAR)". Así, la idea de una estrategia basada en la adición de ruido a los parámetros de la red base requiere de otra que satisfaga la necesidad de proveer de diversidad al ensamble que se construya.

Figura 4.1: Esquema "Clonación con Simple Adición de Ruido (CSAR)".

Entrada. $\mathcal{L}$, σ, *umbral*
Salida. Función f que relaciona las entradas I con sus salidas T
de un conjunto de prueba dado $\mathcal{P}$ que contiene el par de vectores (I, T).
Procedimiento. Creación de ensamble ϕ
{
Entrenar una red base $\mathcal{N}$ con $\mathcal{L}$ usando un algoritmo de entrenamiento conveniente
Usar los pesos definidos en $\mathcal{N}$ para generar n clones neuronales $\mathcal{N}'_1, \ldots, \mathcal{N}'_n$
Perturbar cada clon usando una estrategia de adición de ruido
Probar cada clon i $(i = 1, \ldots, n)$ usando $\mathcal{L}$
Se define el criterio de selección considerando el *umbral* alrededor
del error de entrenamiento $e_{\mathcal{N}}$:
$\| e_i - e_{\mathcal{N}} \| \leq umbral$; donde e_i es el error del clon i
Seleccionar aquellos clones que cumplan con el criterio
Probar cada red j $(j = 1, \ldots, M)$ del ensamble usando $\mathcal{P}$.
Agregar las M salidas para obtener la salida final del ensamble ϕ
}

4.3. Esquema mediante adición de ruido usando análisis de sensibilidad

Se podría pensar en estrategias más agresivas pero se debe ser cuidadoso para no degradar de manera inconveniente el conocimiento de la red base. Tales estrategias deben considerar la posibilidad latente de que al momento de añadir ruido se estén alterando valores importantes en la definición del conocimiento base. Para contemplar tal situación se necesita una técnica que defina los pesos más importantes o sensibles de la red base con el objetivo de realizar las modificaciones sobre el resto y así obtener los clones neuronales. El análisis de sensibilidad da una clara imagen de la red entrenada ya que permite extraer la relación "causa-efecto" entre los parámetros y las salidas de la red. En general, se estima la sensibilidad de la función de error en cada parámetro y si uno de ellos es insensible entonces se puede excluir de la arquitectura [21, 18, 68].

4.3.1. Análisis de Sensibilidad

Hemos usado esta técnica para añadir ruido a aquellos pesos que son menos sensibles en una red neuronal con la intención de obtener redes con desempeños diferentes. Como una variante de la aplicación de esta técnica, también se introduce la estrategia de adicionar un ruido mayor a los pesos menos sensibles aplicando ligeras perturbaciones a los otros. El análisis de sensibilidad explora la arquitectura de una red neuronal para determinar la importancia de los parámetros de la red y, con base en ella, adicio-

nar ruido. Los parámetros con valores de sensibilidad bajos pueden ser considerados insignificativos a la red y ser removidos, lo que implica una reducción de tamaño y complejidad de la arquitectura, y un menor tiempo de entrenamiento [21].

La figura 4.2 muestra la notación (en la arquitectura de la red neuronal) que hemos usado para encontrar el efecto sobre las salidas del sistema que se origine por los cambios realizados en los parámetros. La función *sigmoide* (expresión 4.1) define la función de activación en nuestros modelos de redes neuronales y su derivada expresada en la ecuación 4.2 es útil en nuestros cálculos. Para la capa oculta, la entrada net_j y salida b_j de una neurona se define por las expresiones 4.3 y 4.4. En el caso de la capa de salida, la entrada net_k y salida y_k están dadas por las expresiones 4.5 y 4.6.

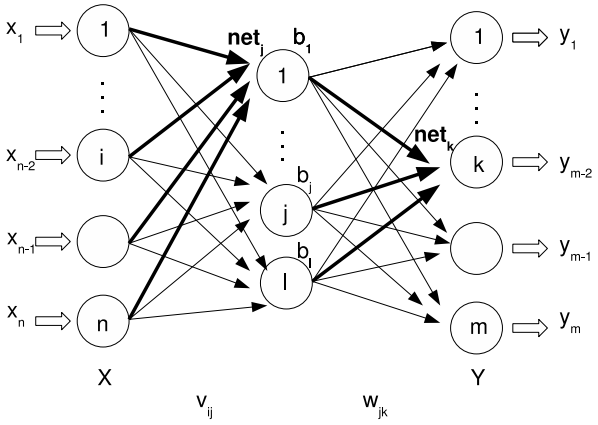


Figura 4.2: Sensibilidad de parámetros de la arquitectura de una red neuronal.

$$f = \frac{1}{1 + e^{-x}} \tag{4.1}$$

$$\frac{df}{dx} = f(1 - f) \tag{4.2}$$

$$net_j = \sum_{i=1}^{N} v_{ij} x_i + B_{j0} \tag{4.3}$$

$$b_j = f(net_j) \tag{4.4}$$

$$net_k = \sum_{j=1}^{L} w_{jk} b_j + B_{k0} \tag{4.5}$$

$$y_k = f(net_k) \tag{4.6}$$

Se considera que $i : 1, \ldots, N$; $j : 1, \ldots, L$; $k : 1, \ldots, M$, y dado que las sensibilidades de los pesos se obtienen mediante las derivadas parciales de la salida con respecto

a los pesos es necesario atender a las expresiones 4.7 y 4.8. Por lo tanto, si $o = N \times L$ y $s = L \times M$, entonces $t = o + s$, por lo que para cada patrón X de un conjunto de datos de entrada se tendrán t valores de sensibilidad. Las expresiones completas que definen las sensibilidades para la capa de entrada se declaran en la ecuación 4.9 y para la capa de salida se tiene la expresión 4.10. La deducción de nuestras fórmulas se desarrolló a partir de las expresiones presentadas en [21].

$$\frac{\partial y_k}{\partial v_{ji}} \tag{4.7}$$

$$\frac{\partial y_k}{\partial w_{kj}} \tag{4.8}$$

$$\begin{aligned} \frac{\partial y_k}{\partial v_{ij}} &= f'_{b_j} f'_{y_k} w_{jk} x_i \\ &= \left(f_{b_j}(1 - f_{b_j})\right) \left(f_{y_k}(1 - f_{y_k})\right) w_{jk} x_i \\ &= \left(f(net_j)(1 - f(net_j))\right) \left(f(net_k)(1 - f(net_k))\right) w_{jk} x_i \end{aligned} \tag{4.9}$$

$$\begin{aligned} \frac{\partial y_k}{\partial w_{jk}} &= f'_{y_k} b_j \\ &= \left(f_{y_k}(1 - f_{y_k})\right) b_j \\ &= \left(f(net_k)(1 - f(net_k))\right) f(net_j) \end{aligned} \tag{4.10}$$

Entonces se define una matriz de sensibilidad S_{o_k,θ_i} de la salida o_k con respecto al parámetro θ_i de la red de tal manera que las sensibilidades de los pesos por capa integran subconjuntos de esta matriz como se expresa en las ecuaciones 4.11 y 4.12.

$$S_{y_k,v_{ij}} \subset S_{o_k,\theta i} \tag{4.11}$$

$$S_{y_k,w_{jk}} \subset S_{o_k,\theta i} \tag{4.12}$$

Usando únicamente análisis de sensibilidad se definió el algoritmo dado en la figura 4.3. Una vez que se tiene la matriz de sensibilidades para cada patrón, se realiza un promedio de ellas. Usando la matriz promedio se define el valor correspondiente al parámetro más sensible y con base en su magnitud se eligen los parámetros que serán perturbados. También se consideró perturbar únicamente aquellos valores cercanos a cero. Sin embargo, esto no es suficiente para definir si un parámetro se poda o no ya que cada patrón tiene una matriz de sensibilidades asociada y, por lo tanto, es necesario contemplar a todas estas matrices al momento de decidir la relevancia de un parámetro con la varianza asociada. Los resultados obtenidos usando el algoritmo descrito en la tabla 4.3, en general mejoran muy poco a la red base por lo que se decidió contemplar alguna heurística definida para podar redes neuronales.

Figura 4.3: Algoritmo propuesto usando análisis de sensibilidad.

Entrada. $\mathcal{L}$, σ, *umbral*
Salida. Función f que relaciona las entradas I con sus salidas T
de un conjunto de prueba dado $\mathcal{P}$ que contiene el par de vectores (I, T).
Procedimiento. Creación de ensamble ϕ
{
Entrenar una red base $\mathcal{N}$ con $\mathcal{L}$ usando un algoritmo de entrenamiento conveniente
Identificar los parámetros insensibles de $\mathcal{N}$
Perturbar el valor de tales parámetros mediante adición de ruido
para generar n clones neuronales $\mathcal{N}'_1, \ldots, \mathcal{N}'_n$
Probar cada clon i $(i = 1, \ldots, n)$ usando $\mathcal{L}$
Se define el criterio de selección considerando el *umbral* alrededor
del error de entrenamiento $e_{\mathcal{N}}$:
$\| e_i - e_{\mathcal{N}} \| \leq umbral$; donde e_i es el error del clon i
Seleccionar aquellos clones que cumplan con el criterio
Probar cada red j $(j = 1, \ldots, M)$ del ensamble usando $\mathcal{P}$.
Agregar las M salidas para obtener la salida final del ensamble ϕ
}

4.3.2. Poda de redes neuronales

La idea inspirada en la poda de parámetros implica saber que la decisión de podar un parámetro debe basarse en otras medidas de relevancia del parámetro. Tal relevancia se puede calcular, y una heurística de poda se usa para decidir cuando un parámetro se considera irrelevante o no. Esta relevancia implica conocer la sensibilidad total del parámetro. Engelbrecht [21] define una heurística de poda basada en la información de la varianza del análisis de sensibilidad de una red. Tal heurística decide podar con base en el "coeficiente de nulidad de varianza del parámetro local". Estos coeficientes permiten manipular los parámetros para obtener una arquitectura de la red con menor complejidad que la original sin perder precisión al remover parámetros innecesarios. El análisis de sensibilidad no es una técnica propia para resolver el problema de diversidad. Sin embargo, dada su naturaleza decidimos incluirla en nuestros estudios para intentar construir clones diferentes entre sí.

Algoritmo implementado usando una heurística de poda de parámetros

El método de Engelbrecht [21] usa análisis de sensibilidad para cuantificar la relevancia de los parámetros. Su idea consiste en que a cada parámetro θ_i se asocia un coeficiente denominado "Nulidad de Varianza del Parámetro" para decidir si se puede o no remover. Este coeficiente cuantifica la nulidad estadística en la "varianza de sensibilidad de parámetro" para el parámetro θ_i de una red sobre los patrones $p = 1 \ldots, P$. La nulidad de varianza de θ_i se define en la expresión 4.13 donde $\sigma^2_{\theta_i}$ es la varianza de la sensibilidad de la salida de la red a perturbaciones en θ_i y σ^2_0 es

Figura 4.4: Algoritmo para crear ensambles basado en poda de parámetros.

Entrada. $\mathcal{L}$, σ, *umbral*

Salida. Función f que relaciona las entradas I con sus salidas T
de un conjunto de prueba dado $\mathcal{P}$ que contiene el par de vectores (I, T).

Procedimiento. Creación de ensamble ϕ
{
Entrenar una red base $\mathcal{N}$ con $\mathcal{L}$ usando un algoritmo de entrenamiento conveniente
Cuantificar la relevancia de los parámetros usando la
"medida de nulidad de varianza" (γ_{θ_i})
Examinar si la varianza en la sensibilidad de un parámetro
para diferentes patrones, es significativamente diferente de cero
Pequeño o nulo efecto del parámetro en la salida de la red neuronal:
si la varianza en las sensibilidades del parámetro es casi cero
y la sensibilidad en promedio es pequeña.
Probar hipótesis usando γ_{θ_i} para definir si un parámetro será podado
Generar n clones neuronales $\mathcal{N}'_1, \ldots, \mathcal{N}'_n$ perturbando los parámetros
identificados como menos sensibles de $\mathcal{N}$
Probar cada clon i $(i = 1, \ldots, n)$ usando $\mathcal{L}$
Se define el criterio de selección considerando el *umbral* alrededor
del error de entrenamiento $e_{\mathcal{N}}$:
$\| e_i - e_{\mathcal{N}} \| \leq umbral$; donde e_i es el error del clon i
Seleccionar aquellos clones que cumplan con el criterio
Probar cada red j $(j = 1, \ldots, M)$ del ensamble usando $\mathcal{P}$.
Agregar las M salidas para obtener la salida final del ensamble ϕ
}

un valor cercano a cero. La sensibilidad promedio de la salida de la red neuronal a perturbaciones de θ_i, para el patrón p, se indica en la expresión 4.14 y la sensibilidad promedio del parámetro θ_i está indicada por la expresión 4.15. Finalmente, la varianza de la sensibilidad de la salida de la red a perturbaciones en θ_i se expresa por la ecuación 4.16.

Nuestro esquema basado en poda de parámetros se define en el algoritmo de la figura 4.4 usando una variante del algoritmo de poda de Engelbretcht [21] y considerando la notación de la figura 4.5.

Figura 4.5: Notación

θ_i	Parámetro i de la red neuronal: pesos w_{ij} y sesgos b_j
P	Total de patrones
K	Total de salidas de la red neuronal
o_k	La salida k de la red neuronal
p	El patrón p
$S^{(p)}_{o_k,\theta_i}$	Sensibilidad de o_k a perturbaciones en θ_i para p

Figura 4.6: Importancia de un parámetro

Revisar que el valor esperado del valor de sensibilidad S_{o_k,θ_i} sea cero
$\left\langle S^2{}_{o_k,\theta_i} \right\rangle = \langle S_{ok,\theta_i} \rangle^2 + var(S_{o_k,\theta_i})$
Analizar si $\left\langle S^2{}_{o_k,\theta_i} \right\rangle = 0$
1) Probar la hipótesis
$H_0 : \langle S_{o_k,\theta_i} \rangle^2 = 0$
Si la hipótesis es aceptada, continuar con el paso 2);
de otro modo el parámetro no se puede podar
2) Probar la hipótesis
$H_0 : var(S_{o_k,\theta_i}) = 0$
Si se rechaza esta hipótesis, el parámetro es relevante y no puede ser podado

$$\gamma_{\theta_i} = \frac{(P-1)\sigma^2_{\theta_i}}{\sigma^2_0} \tag{4.13}$$

$$\aleph^{(p)}_{\theta_i} = \frac{\sum_{k=1}^{K} S^{(p)}_{o_k,\theta_i}}{K} \tag{4.14}$$

$$\bar{\aleph}_{\theta_i} = \frac{\sum_{p=1}^{P} \aleph^{(p)}_{\theta_i}}{P} \tag{4.15}$$

$$\sigma^2_{\theta_i} = \frac{\sum_{p=1}^{P} (\aleph^{(p)}_{\theta_i} - \bar{\aleph}_{\theta_i})^2}{P-1} \tag{4.16}$$

El algoritmo de poda usado ofrece la ventaja de definir con mayor precisión si un parámetro es o no importante mediante el procedimiento dado en la figura 4.6 [21].

Engelbrecht indica que no basta que $\langle S_{o_k,\theta i} \rangle^2 = 0$ ya que grandes valores de sensibilidad positivos y negativos se pueden cancelar unos con otros generando una suma de cero e indicando que θ_i es insignificativo (lo que es falso). Por esto es necesario que la relevancia de θ_i se defina en términos de la "nulidad de varianza del parámetro". Con la nulidad de varianza de un parámetro, para cada parámetro θ_i se prueba la

Figura 4.7: Algoritmo de poda de Engelbrecht.

1) Inicialización de la arquitectura de la red neuronal y parámetros de aprendizaje
2) Repetir
    a) Entrenar la red hasta que un indicador de poda es lanzado
    b) $\sigma_0^2 = 0.0001$;
    c) Para cada θ_i
        i) Para cada $p = 1, ..., P$, calcular $\aleph_{\theta_i}^{(p)}$
        ii) Calcular el promedio $\bar{\aleph}_{\theta_i}$
        iii) Calcular la varianza en la sensibilidad del parámetro usando $\sigma_{\theta_i}^2$
        iv) Calcular la variable de prueba γ_{θ_i}
    d) Aplicar la heurística de poda:
        i) Ordenar γ_{θ_i} en orden creciente
        ii) Encontrar γ_c
        iii)Para cada θ_i, si $\gamma_{\theta_i} \leq \gamma_c$, entonces podar θ_i
        iv) Si $\gamma_{\theta_i} > \gamma_c \; \forall \; \theta_i, \sigma_0^2 = \sigma_0^2 \times 10$ e ir a 2(c)(iv)
Hasta que ningún parámetro sea podado, o la red reducida no sea aceptada debido a un deterioro inaceptable en el desempeño de generalización
3) Entrenar la arquitectura final de la red podada

hipótesis nula de que la varianza en la sensibilidad del parámetro sea aproximadamente cero (expresada en 4.17). Pero la varianza no es exactamente cero, $\theta_0^2 \neq 0$, por lo que no se puede crear la hipótesis expresada en 4.18 y por lo tanto se selecciona para σ^2 un valor cercano a cero dando origen a la hipótesis alternativa dada en la ecuación 4.19.

$$H_0 : \sigma_{\theta_i}^2 = \theta_0^2 \tag{4.17}$$

$$\sigma_{\theta_i}^2 = 0 \tag{4.18}$$

$$H_1 : \sigma_{\theta_i}^2 < \theta_0^2 \tag{4.19}$$

Engelbrecht en [21] asume que bajo la hipótesis nula la medida de nulidad de varianza tiene una distribución $\chi^2(P-1)$ en el caso de P patrones y el valor crítico $\gamma_c = \chi_{v;1-\alpha}^2$ se obtiene de las tablas de distribución χ^2. En tal notación, $v = P - 1$ indica el número de grados de libertad y α el nivel de significancia. Si $\gamma_{\theta_i} \leq \gamma_c$, se acepta H_1 y se poda el parámetro θ_i. Los valores definidos para el algoritmo son $\sigma_0^2 = 0.0001$ y $\alpha = 0.01$ que indica que es suficiente rechazar incorrectamente la hipótesis 1 de cada 100 veces. Así, conforme α es más pequeño el algoritmo de poda es más estricto. El valor de σ_0^2 es muy importante ya que si es muy pequeño, ningún parámetro será podado y si es muy grande se podarán parámetros importantes.

El algoritmo de poda original, definido en la figura 4.7, revisa que el desempeño de la arquitectura definida después de podar los parámetros sea aceptable y si no lo es la arquitectura anterior es considerada nuevamente. Si el desempeño es aún aceptable,

Figura 4.8: Algoritmo de poda modificado.

1) Inicialización de la arquitectura de la red neuronal y parámetros de aprendizaje
2) Repetir
    a) Entrenar la red neuronal con determinada cantidad de épocas
    b) $\sigma_0^2 = 0.0001$;
    c) Para cada θ_i
        i) Para cada $p = 1, ..., P$, calcular $\aleph_{\theta_i}^{(p)}$
        ii) Calcular el promedio $\bar{\aleph}_{\theta_i}$
        iii)Calcular la varianza en la sensibilidad del parámetro usando $\sigma_{\theta_i}^2$
        iv) Calcular la variable de prueba γ_{θ_i}
    d) Aplicar la heurística de poda:
        i) Ordenar γ_{θ_i} en orden creciente
        ii) Encontrar γ_c
        iii)Para cada θ_i, si $\gamma_{\theta_i} \leq \gamma_c$, entonces podar θ_i
        iv) Si $\gamma_{\theta_i} > \gamma_c \; \forall \; \theta_i, \sigma_0^2 = \sigma_0^2 \times 10$ e ir a 2(c)(iv)
Hasta que ningún parámetro sea podado

el proceso de poda continúa. Además, cuando la red es podada, el modelo podado inicia un re-entrenamiento a partir de nuevos pesos aleatorios. Finalmente, si no hay más parámetros que se identifiquen para ser podados o si el modelo reducido no es aceptable, el proceso de poda termina. Todo el proceso mencionado implica mucho tiempo, incluso más que el proceso de entrenamiento de un clasificador neuronal. Por ende, se decidió modificar el algoritmo para obtener una respuesta rápida que implique una identificación de primera instancia de los pesos menos relevantes en la arquitectura de la red entrenada, por lo que posiblemente la definición de tales pesos puede no ser la óptima. Nuestra modificación se denomina "Algoritmo de poda modificado" y se muestra en la figura 4.8.

Nuestro esquema basado en poda de parámetros se detalla en la figura 4.4 y se resume mediante el algoritmo de la figura 4.9. Este esquema se ha denominado "Clonación usando Poda de Parámetros (CPP)".

4.4. Esquema usando clonación de dos redes base

Después de considerar el análisis de sensibilidad y observar que estas ideas no son suficientes, diseñamos otra estrategia cuyo objetivo nuevamente es incrementar la diversidad; ahora se contemplan dos redes base. Se entrenan dos redes base y se selecciona aleatoriamente una de ellas para generar los clones mediante alguna estrategia de adición de ruido. El criterio de selección implica considerar al clon neuronal cuyo error de entrenamiento sea menor a determinado umbral sobre el promedio de los errores de entrenamiento de ambas redes base. Se consideran las clasificaciones erróneas como medida de error. No se incluyen las redes base. Los resultados ob-

Figura 4.9: Esquema "Clonación usando usando Poda de Parámetros (CPP)"

Entrada. $\mathcal{L}$, σ, *umbral*
Salida. Función f que relaciona las entradas I con sus salidas T
de un conjunto de prueba dado $\mathcal{P}$ que contiene el par de vectores (I, T).

1) Entrenar una red base $\mathcal{N}$ con $\mathcal{L}$
2) $\sigma^2_{ruido} = 0.1$
3) $\sigma^2_0 = 0.00001$
4) Aplicar el "algoritmo de poda modificado" a la red base
5) Generar los $1 \leq i \leq n$ clones $\mathcal{N}'_i$
   a) Modificar para cada clon $\mathcal{N}'_i$ los pesos definidos en 4):
   $$w^{\mathcal{N}'_i}_{irrelevante} = w^{\mathcal{N}'_i}_{irrelevante} + w^{\mathcal{N}'_i}_{irrelevante} * N(0, \sigma^2_{ruido})$$
6) Probar cada clon usando $\mathcal{L}$
7) Seleccionar los clones convenientes.
8) Probar cada red del ensamble usando $\mathcal{P}$.
9) Agregar

tenidos son mejores que el esquema que considera una sola red base porque existe mayor diversidad gracias al entrenamiento de otra red. Se ha denominado a este esquema como "Clonación usando Dos Redes Base (CDRB)". Existe una variante de este esquema que incluye a las redes base en el ensamble y se denomina "Clonación y combinación de Dos Redes Base (CDRB+)".

Dado que todos nuestros esquemas usan las clasificaciones erróneas como medida de error para seleccionar los clones, existe otra variante del esquema que incluye las redes base dentro del ensamble y se denomina "Clonación usando Dos Redes Base con MSE (CDRB-MSE)". Usa la medida de error MSE para comparar el error sobre $\mathcal{L}$ de cada clon que es candidato a ser parte del ensamble. Al momento de validar el ensamble con los datos de prueba se usa la cantidad de clasificaciones incorrectas generadas por el ensamble para obtener su desempeño.

4.5. Resultados experimentales

Se han usado las herramientas disponibles en MATLAB para redes neuronales. Para definir nuestros esquemas se usan las funciones de aprendizaje de red *traingdx* y *trainlm* como algoritmos de entrenamiento. Se aplicó validación cruzada de 10 bloques para analizar con mayor precisión el desempeño de nuestros esquemas. Nuestros modelos se han aplicado a algunos problemas de clasificación de la vida real. Los conjuntos de datos usados a lo largo de nuestros experimentos son Glass, Sonar, Pima diabetes, Iono, Waveform, Breast Cancer, Bupa, Iris, DNA y Protein. El resumen de los conjuntos de datos se presentan en la tabla 4.1. La ecuación considerada durante todos nuestros experimentos para observar el desempeño de los clasificadores se indica en la expresión 4.20.

Tabla 4.1: Resumen de los conjuntos de datos

Datos	Fuente	Número de puntos	Número de características	Número de clases
Glass	[61]	214	9	7
Sonar	[61]	208	60	2
Pima diabetes	[61]	768	8	2
Iono	[61]	351	34	2
Waveform40	[61]	5000	40	3
Waveform21	[61]	5000	21	3
BreastCancer	[61]	684	9	2
DNA	[61]	3186	180	3
Iris	[61]	150	4	3
Bupa	[61]	345	6	2
Wine	[61]	178	13	3
Protein	[62]	698	125	4

$$precisión(\%) = \frac{(\text{total de puntos}) - (\text{clasificaciones incorrectas})}{(\text{total de puntos})} \times 100 \tag{4.20}$$

Las redes neuronales consideradas tienen una arquitectura definida de tres capas, una de entrada, la oculta y otra de salida. La capa oculta se integra de diez neuronas y el resto de las capas se define de acuerdo al conjunto de datos en análisis. Durante nuestros estudios se aplicaron tres tipos de medida de error: *SSE*, *MSE* y la cantidad de clasificaciones incorrectas. Para cada experimento con problemas de clasificación, se seleccionaron conjuntos de datos diferentes debido a que primeramente se intenta observar el desempeño del esquema que se diseña usando problemas simples con la idea de generalizar posteriormente. Sin embargo, la mayoría de los conjuntos de datos son probados con nuestros modelos. Al final, los resultados de nuestros mejores métodos son probados con problemas con diferente complejidad. Se muestra la precisión de nuestros esquemas en porcentaje (%) y el tiempo que requieren en segundos (s). Hemos estructurado la presentación de nuestros resultados de acuerdo a los esquemas generales que se estudiaron.

4.5.1. Esquema mediante adición simple de ruido

Primero se observó el desempeño de nuestro esquema "Clonación con Simple Adición de Ruido (CSAR)" aplicándolo al problema de aprender una curva senoidal con ruido. El análisis realizado sobre este problema abrió la posibilidad de la factibilidad de nuestro esquema en problemas de clasificación. La curva de seno con ruido se expresa en la ecuación 4.21.

$$f(x) = 0.4\ \text{sin}x + 0.5, x \in \Re \tag{4.21}$$

Generamos un conjunto de pares de entrada-salida $X = (x_i, y_i)$, $i = 1, \ldots, 150$ con x_i generada con distribución uniforme dentro del rango $[-\pi, \pi]$ y $y_i = f(x_i) + r_i$, donde $r_i \sim \mathcal{N}(0, \sigma^2)$ indica el ruido con varianza $\sigma^2 = 0.01$.

Cada dato de este conjunto tiene una salida y_i y una entrada x_i. Con este conjunto de datos entrenamos un MLP convencional y luego generamos clones neuronales considerando un valor de $\sigma^2 = 0.0015$ para la adición de ruido sobre los parámetros de la red base. También, generamos 50 pares de puntos sin ruido para crear un conjunto de datos de prueba y validar con éste a los miembros del ensamble.

El valor de σ de los clones se definió mediante experimentos que indican el comportamiento del valor del SSE conforme el valor de σ cambia. Las figuras 4.10 y 4.11 muestran estos resultados y nos permiten entender que cuando no existe perturbación, el ensamble nos dará el mismo resultado que la red base y que conforme la perturbación crece se aprecia un decremento en el SSE de tal manera que el ensamble mejora el desempeño de la red base. Sin embargo, si la perturbación es muy grande, el conocimiento de la red base en las copias se degrada de tal manera que el desempeño de cada clon es generalmente peor que la red base y por lo tanto el desempeño del ensamble empeora.

Las redes consideradas para este ensamble se definen por una capa oculta con 10 neuronas y únicamente 1 neurona en la salida. Se aplican 50 copias de la red base.

En la tabla 4.2 se muestran los resultados al integrar un ensamble mediante la selección de clones usando un umbral sobre el error de entrenamiento de la red base de 5 %. La tabla muestra resultados finales del promedio de 20 ensambles. El primer bloque implica $\sigma = 0.00001$, el segundo $\sigma = 0.0001$ y así sucesivamente hasta considerar el bloque de $\sigma = 0.1$. Conforme el valor de σ incrementa el algoritmo requiere más tiempo debido a que la adición de ruido destruye la configuración de pesos que define el conocimiento de la red base y esto provoca que sea difícil encontrar redes dentro del umbral deseado. Sin embargo, los resultados con un σ mayor son más aceptables y la razón es que existe mayor diversidad entre los clones. Además, conforme el valor de σ decrementa, los desempeños de los clasificadores y ensambles son muy similares debido a que la diversidad es casi nula. Se realizaron otros experimentos que buscan definir una mejor configuración de parámetros del ensamble. Se contempló extender el umbral para considerar a redes con mayor diversidad y los resultados en cuanto a recursos temporales son mejores logrando mantener la precisión adquirida con un umbral menor como se aprecia en la tabla 4.3. También, en la tabla 4.4 se aprecian otros resultados manteniendo fija la red base.

Estos resultados motivaron la investigación sobre la aplicación de nuestro esquema en problemas de clasificación.

El desempeño del esquema "Clonación con Simple Adición de Ruido (CSAR)" en problemas de clasificación se muestra en la tabla 4.5 y se compara contra *Bagging*. Se usa un valor para σ=0.0015. La tabla 4.5 claramente indica que este esquema puede mejorar el desempeño de la red base. Sin embargo el desempeño de este método es siempre un poco menor que el obtenido por *Bagging*. A pesar de esto, el factor

Tabla 4.2: Función seno con ruido (umbral de 5 %)

Método	σ	Error de clasificación	Tiempo de ensamble (s)	Mejora (%)
Red base	-	0.462 ± 0.273	-	-
Ensamble	0.00001	0.461 ± 0.274	1.235 ± 0.178	0.216
Red base	-	0.531 ± 0.220	-	-
Ensamble	0.0001	0.508 ± 0.213	1.731 ± 0.194	4.331
Red base	-	0.632 ± 0.391	-	-
Ensamble	0.001	0.578 ± 0.382	3.427 ± 2.370	8.544
Red base	-	0.588 ± 0.467	-	-
Ensamble	0.01	0.498 ± 0.449	18.068 ± 24.279	15.306
Red base	-	0.703 ± 0.521	-	-
Ensamble	0.1	0.568 ± 0.476	106.579 ±109.247	19.203

Tabla 4.3: Función seno con ruido (umbral de 10 %)

Método	σ	Error de clasificación	Tiempo de ensamble (s)	Mejora (%)
Red base	-	0.654 ± 0.478	-	-
Ensamble	0.00001	0.652 ± 0.478	1.253 ± 0.233	0.305
Red base	-	0.560 ± 0.460	-	-
Ensamble	0.0001	0.551 ± 0.458	1.279 ± 0.094	1.607
Red base	-	0.610 ± 0.357	-	-
Ensamble	0.001	0.559 ± 0.346	1.648 ± 0.419	8.360
Red base	-	0.665 ± 0.476	-	-
Ensamble	0.01	0.569 ± 0.443	4.863 ± 4.344	14.436

Tabla 4.4: Resultados sobre los datos de seno con ruido

Renglón No.	Umbral sobre el SSE de entrenamiento	SSE de datos de prueba de la red base	SSE de datos de prueba del ensamble	Mejora (%)
1	0.080	0.0325	0.0305	6.150
2	0.050	0.0325	0.0310	4.615
3	0.040	0.0325	0.0306	5.846
4	0.030	0.0325	0.0302	7.077
5	0.020	0.0325	0.0301	7.385
6	0.015	0.0325	0.0297	8.615

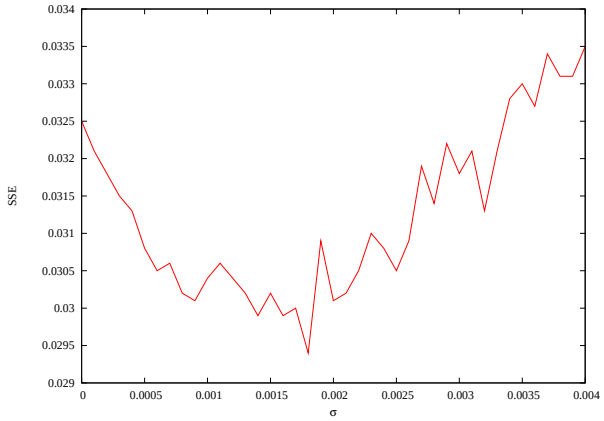


Figura 4.10: Valores de σ y SSE aplicando nuestro esquema.

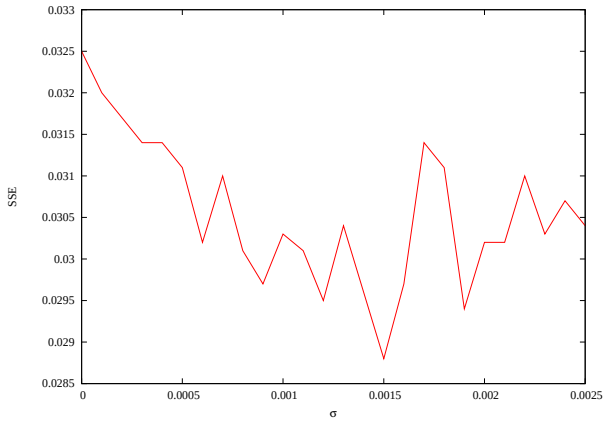


Figura 4.11: Acercamiento de la Figura 4.10.

Tabla 4.5: Desempeño del esquema "Clonación con Simple Adición de Ruido (CSAR)"

Método	Precisión de red neuronal base (%)	Precisión de ensamble (%)	Tiempo (s)
Iris			
Bagging	-	96.089 ± 5.660	134.015 ± 2.323
CSAR	91.266 ± 6.114	91.333 ± 6.812	18.922 ± 4.367
Glass			
Bagging	-	72.960 ± 8.051	66.114 ± 1.324
CSAR	67.871 ± 3.609	71.109 ± 9.757	45.410 ± 3.245
Waveform 40 características			
Bagging	-	85.407 ± 0.939	4077.500 ±166.793
CSAR	60.185 ± 5.994	71.016 ± 3.621	367.124 ±196.124
Waveform 21 características			
Bagging	-	84.107 ± 1.887	3669.750 ± 142.141
CSAR	62.745 ± 5.935	72.638 ± 3.799	294.792 ± 186.022
PimaDiabetes			
Bagging	-	75.113 ± 4.060	797.816 ± 8.127
CSAR	66.352 ± 2.104	68.417 ± 1.966	41.308 ±10.343
Wine			
Bagging	-	97.184 ± 1.886	345.792 ± 6.874
CSAR	83.901 ± 7.986	85.545 ± 7.636	29.991 ± 8.263

importante a observar es que el ahorro de tiempo que ofrece $CSAR$, es muy grande comparándolo con el recurso temporal que consume $Bagging$.

4.5.2. Esquema mediante adición de ruido usando análisis de sensibilidad

El esquema propuesto "Clonación usando Poda de Parámetros (CPP)" basado en el algoritmo de Engelbrecht [21] generó los resultados mostrados en la tabla 4.6. Se muestran resultados de conjuntos con dimensiones bajas que definen problemas simples. Este método implica un mayor costo computacional que la adición simple de ruido. Se usa el valor σ=0.0015. Como se puede apreciar en la tabla, los ensambles creados por CPP ofrecen únicamente una mejora marginal (o a veces ninguna) sobre la red base.

4.5.3. Buscando el valor de σ para problemas de clasificación

El problema de definir un valor para σ que permita adicionar un valor adecuado de ruido a los parámetros de la red base requiere de una estrategia de búsqueda. Esto

Tabla 4.6: Desempeño de "Clonación usando Poda de Parámetros (CPP)"

Métodos	Precisión de red base (%)	Precisión de ensamble (%)	Tiempo (s)
Breast cancer			
CPP	87.721 ± 1.813	68.947 ± 0.241	72.294 ± 11.178
Bupa			
CPP	63.344 ± 3.445	63.467 ± 3.132	5.400 ± 0.348
Iris			
CPP	93.407 ± 4.327	93.481 ± 4.421	23.307 ± 6.987
Wine			
CPP	86.097 ± 0.473	88.097 ± 0.632	38.347 ± 0.231

nos motivó a definir tal valor en un rango pequeño cercano a cero atendiendo los efectos de la degradación realizada conforme el valor de σ crece.

La figura A.1 muestran que nuestro esquema "Clonación con Simple Adición de Ruido (CSAR)" funciona para cierto rango de valores para σ. Esta figura muestra el uso de la estrategia de adición de ruido anteriormente enumerada como 2 que implica un valor de r para cada capa de sesgos y de pesos.

A través de la figura A.2 se observan los resultados aplicando nuestro esquema "Clonación con Simple Adición de Ruido (CSAR)" usando las estrategias de ruido que funcionaron mejor y que se han enumerado anteriormente como 1, 1a), 2, 5, 6. Para estos experimentos no se usa criterio de selección de clones ya que sólo deseamos conocer el comportamiento de nuestro esquema conforme σ crece. A través de esta información se observa que la idea implica mejorar el desempeño de la red base pero no es suficiente para generar ensambles con desempeños competentes con *Bagging*.

De acuerdo a lo observado en los experimentos anteriores, el valor de σ debe ser muy pequeño, alrededor de $\sigma = 0.0035$ (para problemas de clasificación) y parece funcionar de manera aceptable para la mayoría de los conjuntos de datos. Los resultados que a continuación se presentan, se realizaron usando diferentes valores para σ, $\sigma = 0.001, 0.002, 0.0035$.

4.5.4. Esquema usando clonación de dos redes base

Nuestros mejores resultados han sido obtenidos con los esquemas "Clonación usando Dos Redes Base (CDRB)" y "Clonación y combinación de Dos Redes Base (CDRB+)" que aplican la técnica "votación mayoritaria" para la agregación. Para la selección de los clones se usa la medida *SSE* y para definir el desempeño de cada ensamble se consideran las clasificaciones incorrectas. Además usa *Bootstrap* para generar los conjuntos de entrenamiento. Para cada problema se realizaron 20 iteraciones. Obtuvimos los desempeños de *Bagging* para ensambles de 15 miembros y podemos observar que su costo temporal es muy alto. Mediante las tablas 4.7, 4.8 y 4.9 se pueden apreciar los desempeños de los esquemas "Clonación usando Dos Redes Base

(CDRB)", "Clonación usando Dos Redes Base con MSE (CDRB-MSE)", "Clonación y combinación de Dos Redes Base (CDRB+)" y *Bagging*.

En las tablas 4.7, 4.8 se puede consultar el desempeño de nuestro esquema "Clonación usando Dos Redes Base (CDRB)" que no incluye las redes base en el ensamble; el desempeño de nuestro esquema "Clonación y combinación de Dos Redes Base (CDRB+)" que incluye tales redes base se puede observar en la tabla 4.9.

La estrategia implementada para adicionar ruido en estos esquemas es la que hemos enumerado como 1. Se usa un umbral del 5 % sobre el promedio de error de ambas redes neuronales para seleccionar aquellas redes cuyo error esté por debajo del umbral. Se usa una probabilidad de adición de ruido de 1 y para seleccionar una red base con el objetivo de definir un clon de ella se usa una probabilidad de 0.5.

La tabla A.1 muestra el desempeño de un ensamble integrado únicamente por las dos redes base entrenadas. Esto permite conocer si los clones que se definen con nuestro esquema son útiles para incrementar el desempeño del ensamble.

En el apéndice se pueden apreciar otros resultados obtenidos aplicando nuestros esquemas para crear ensambles de mayores tamaños.

4.5.5. La perturbación mejora el desempeño de la red base

Las redes perturbadas pueden tener mejores desempeños que la red base considerando un rango adecuado de σ. Las figuras 4.10 y 4.11 muestran que existe un valor de σ que permite tener un menor valor de SSE sobre los datos de prueba del seno con ruido. Encontramos que un valor adecuado para σ es 0.0015; sin embargo, tal valor no es el mismo para todos los problemas como se deseaba.

Las figuras A.1 y A.2 muestran que nuestro esquema de creación de ensambles mediante la adición de ruido mejora el desempeño de la red base en general pero no es suficiente para obtener desempeños considerablemente importantes.

4.6. Conclusión

Dado que las redes neuronales son inestables, si perturbamos la entrada obtendremos salidas diferentes. Esto es completamente cierto cuando se aplica un algoritmo de entrenamiento a partir de un conjunto de entrada y determinados parámetros iniciales. Aún cuando el clon no ha sido entrenado de manera diferente a otros, si perturbamos sus parámetros se pueden generar peores desempeños que la red original y la estrategia para identificar clones adecuados implica un problema cuya solución debe definirse. La perturbación permitirá definir una configuración que integre un clon diferente a la red base pero el tiempo que sea necesario es desconocido. Este es el gran problema al momento de seleccionar configuraciones perturbadas porque puede ser más costoso que el entrenamiento y esto no se desea.

La desventaja principal de los esquemas propuestos es la ausencia de diversidad en el ensamble. El acuerdo entre clasificadores es un factor importante, y conforme su magnitud sea baja, la diversidad será mayor. El panorama ideal contempla tener

Tabla 4.7: Desempeños de "Clonación usando Dos Redes Base (CDRB)", "Clonación usando Dos Redes Base con MSE (CDRB-MSE)" y *Bagging* (15 clasificadores). Parte 1

Métodos	Precisión (%)	Tiempo (s)
Protein		
Bagging	86.900 ± 1.100	6509.000 ± 3040.200
CDRB(σ=0.001)	83.350 ± 1.652	801.541 ± 20.256
CDRB(σ=0.002)	82.520 ± 3.101	801.630 ± 20.245
CDRB(σ=0.0035)	82.870 ± 2.656	801.640 ± 20.436
CDRB-MSE(σ=0.001)	83.003 ± 2.697	690.400 ± 19.196
CDRB-MSE(σ=0.002)	83.386 ± 2.328	735.447 ± 18.402
CDRB-MSE(σ=0.0035)	84.792 ± 1.918	730.427 ± 19.229
Iono		
Bagging	92.230 ± 0.656	99.630 ±18.630
CDRB(σ=0.001)	89.520 ± 1.008	13.940 ± 2.044
CDRB(σ=0.002)	90.290 ± 0.679	13.978 ± 2.034
CDRB(σ=0.0035)	89.310 ± 0.983	14.110 ± 2.057
CDRB-MSE(σ=0.001)	89.757 ± 0.963	10.785 ± 2.332
CDRB-MSE(σ=0.002)	89.042 ± 0.927	12.803 ± 2.039
CDRB-MSE(σ=0.0035)	89.807 ± 0.504	13.804 ± 2.124
Bupa		
Bagging	68.273 ± 1.671	79.090 ± 12.948
CDRB(σ=0.001)	63.590 ± 2.300	11.000 ± 1.105
CDRB(σ=0.002)	64.850 ± 3.311	11.030 ± 2.069
CDRB(σ=0.0035)	64.840 ± 2.404	11.947 ± 1.063
CDRB-MSE(σ=0.001)	64.475 ± 2.007	10.932 ± 2.090
CDRB-MSE(σ=0.002)	63.991 ± 2.337	9.892 ± 1.063
CDRB-MSE(σ=0.003)	65.827 ± 2.450	9.861 ± 2.011
CDRB-MSE(σ=0.0035)	64.994 ± 2.131	10.932 ± 1.085

Tabla 4.8: Desempeño de "Clonación usando Dos Redes Base (CDRB)", "Clonación usando Dos Redes Base con MSE (CDRB-MSE)" y *Bagging* (15 clasificadores). Parte 2

Métodos	Precisión (%)	Tiempo (s)
Sonar		
Bagging	84.900 ± 1.348	203.370 ± 13.927
CDRB(σ=0.001)	82.870 ± 1.794	27.930 ± 5.044
CDRB(σ=0.002)	82.590 ± 2.239	27.922 ± 5.464
CDRB(σ=0.0035)	81.710 ± 2.560	27.950 ± 4.552
CDRB-MSE(σ=0.001)	82.885 ± 3.321	28.011 ± 3.531
CDRB-MSE(σ=0.002)	82.842 ± 1.000	26.994 ± 3.666
CDRB-MSE(σ=0.0035)	82.121 ± 1.691	25.963 ± 1.247
Waveform		
Bagging	85.400 ± 0.939	4077.500±166.793
CDRB(σ=0.001)	79.100 ± 3.652	542.900± 53.402
CDRB(σ=0.002)	78.900 ± 1.659	545.800± 54.268
CDRB(σ=0.0035)	80.100 ± 1.712	548.230± 53.249
CDRB-MSE(σ=0.001)	78.100 ± 4.349	532.500± 50.169
CDRB-MSE(σ=0.002)	78.900 ± 4.414	522.300± 50.147
DNA		
Bagging	93.480 ± 0.200	577.320± 123.113
CDRB(σ=0.001)	92.490 ± 0.397	79.690 ± 10.548
CDRB(σ=0.002)	92.240 ± 0.560	79.480 ± 10.287
CDRB(σ=0.0035)	92.450 ± 0.261	80.250 ± 10.342
CDRB-MSE(σ=0.001)	92.510 ± 0.456	80.506 ± 9.479
CDRB-MSE(σ=0.002)	92.190 ± 0.351	83.573 ± 8.337
CDRB-MSE(σ=0.0035)	92.665 ± 0.359	82.493 ± 9.492

Tabla 4.9: Desempeño de "Clonación y combinación de Dos Redes Base (CDRB+)" y *Bagging* (15 clasificadores)

Métodos	Precisión (%)	Tiempo (s)
Protein		
Bagging	86.900 ± 1.100	6509.000 ±3040.200
CDRB+(σ=0.001)	83.000 ± 2.697	721.624 ± 18.145
CDRB+(σ=0.002)	83.380 ± 2.328	723.434 ± 19.402
CDRB+(σ=0.0035)	84.790 ± 1.918	723.002 ± 18.229
Iono		
Bagging	92.230 ± 0.656	99.630 ± 18.630
CDRB+(σ=0.001)	89.750 ± 0.963	11.785 ± 2.032
CDRB+(σ=0.002)	89.040 ± 0.927	12.380 ± 2.039
CDRB+(σ=0.0035)	89.800 ± 0.504	12.430 ± 2.024
Bupa		
Bagging	68.273 ± 1.671	79.090 ± 12.948
CDRB+(σ=0.001)	64.470 ± 2.000	9.930 ± 1.090
CDRB+(σ=0.002)	63.990 ± 2.337	8.890 ± 1.063
CDRB+(σ=0.0035)	64.990 ± 2.131	10.132 ± 1.085
Sonar		
Bagging	84.900 ± 1.348	203.370 ±13.927
CDRB+(σ=0.001)	82.880 ± 2.702	28.010 ± 5.050
CDRB+(σ=0.002)	82.840 ± 1.268	27.900 ± 5.060
CDRB+(σ=0.0035)	82.120 ± 1.537	26.960 ± 5.036
Waveform		
Bagging	85.400 ± 0.939	4077.500 ± 166.793
CDRB+(σ=0.001)	78.100 ± 3.306	471.500 ± 53.185
CDRB+(σ=0.002)	78.900 ± 3.653	468.300 ± 52.148
DNA		
Bagging	93.480 ± 0.200	577.320 ±123.113
CDRB+(σ=0.001)	92.510 ± 0.456	74.908 ± 10.479
CDRB+(σ=0.002)	92.190 ± 0.351	75.570 ± 10.337
CDRB+(σ=0.0035)	92.660 ± 0.359	74.490 ± 9.492

diversidad significativa de manera que no se comprometa la precisión individual de cada clasificador [27].

Hemos presentado una nueva estrategia para crear ensambles de redes neuronales y con esto hemos definido parte del trabajo relacionado con generar clasificadores sin requerir mucho tiempo. La generación rápida de ensambles de clasificadores es una meta ambiciosa que se complica cuando se usan redes neuronales. No tenemos una respuesta sólida ante la posibilidad de definir una estrategia para fijar el valor de σ que permita perturbar convenientemente la red base y que sea independiente de los datos como era el objetivo inicial. La diversidad es un factor determinante para el buen funcionamiento de nuestra estrategia y un método para obtenerla se convierte en otro de los problemas cuya solución no es trivial cuando se tiene la restricción del tiempo. Evidentemente nuestro ensamble trabaja ahorrando mucho tiempo de entrenamiento debido a que estamos simulando el proceso de aprendizaje a través de la perturbación de la red base pero el desempeño final no es comparable con el de *Bagging*. El costo implícito en el ahorro de tiempo es la escasa diversidad en nuestro ensamble.

En general, para todos los problemas se cumple que nuestro esquema mejora el desempeño de la red neuronal base gracias al rango de variabilidad de $\boldsymbol{W}$ y el problema radica cuando se desea ir más allá de ese rango para obtener diversidad. Esto indica que nuestra idea puede considerarse para realzar de una manera rápida el desempeño de una red base promedio considerando que la mejora que se obtendrá puede incrementar con otros métodos más costosos.

Normalmente el entrenamiento de las redes neuronales es no determinista porque la función de error contiene varios mínimos locales que pueden ser alcanzados cuando el aprendizaje se realiza por medio del gradiente descendente. Este hecho, aunado con la variabilidad de los conjuntos de datos de entrenamiento para cada clasificador del ensamble, integran los dos factores que hasta ahora son los más usados para construir combinaciones convenientes.

Capítulo 5

Ensambles de redes neuronales usando estimaciones de densidad

La diversidad de nuestros esquemas debe ser incrementada. Los clasificadores individuales con diversidad considerable pueden ser obtenidos de varias formas: i) usando algoritmos diferentes para aprender de los datos, ii) cambiando la estructura interna de un algoritmo dado, y iii) aprendiendo a partir de diferentes subconjuntos seleccionados adecuadamente del conjunto de datos original [69]. La última estrategia es la más usada y está fuertemente ligada a la inestabilidad del algoritmo de aprendizaje [5]. En el caso de las redes neuronales, la inestabilidad está dada de manera natural a partir de los datos inherentes y la aleatoriedad del proceso de entrenamiento. Así, la combinación de la fuerte inestabilidad del algoritmo de aprendizaje con la diversidad de los clasificadores ensamblados, contra la capacidad de una buena generalización individual requiere una selección adecuada de los miembros del ensamble. Esto se aprecia en las técnicas *Bagging* y *Boosting*.

Dado que existe la posibilidad de hacerlo mediante la generación de diferentes subconjuntos seleccionados adecuadamente a partir del conjunto de datos original, esta vez se contempla el conjunto de datos de entrada usando una técnica para obtener nuevos conjuntos de entrenamiento.

Se requiere definir una nueva estrategia para crear el conjunto de entrenamiento de cada red neuronal del ensamble. *Bagging* es un método que define estimadores mediante un entrenamiento que requiere múltiples réplicas del conjunto de entrenamiento que se usan como nuevos conjuntos de aprendizaje [5]. Tales réplicas son generadas mediante la técnica *Bootstrap*. Aquí se propone una nueva estrategia para sustituir la técnica anterior; esta vez se generan las réplicas usando estimaciones de densidad.

Las redes neuronales pueden entrenarse para realizar muchas tareas y una característica importante de su desempeño es su habilidad para responder de manera adecuada a datos de entrada no vistos previamente (generalización, ver capítulo 1). Para el problema de clasificación, esto significa que una red pueda clasificar correctamente muestras que no han sido usadas en el entrenamiento de la red. En los problemas de

regresión, la generalización implica la habilidad de interpolar de manera significativa entre las muestras de entrenamiento. Una generalización deficiente puede crecer en una situación que implique que una cantidad de parámetros deba ser aprendida y que sea muy alta comparada con la cantidad de muestras de entrenamiento. Para remediar esta situación se ha sugerido generar una fuente ilimitada de muestras de entrenamiento adicionando ruido aleatorio a los vectores de entrenamiento disponibles [59]. El problema entonces radica en seleccionar las características del ruido aditivo de tal manera que la mejora real en el desempeño de una red sea alcanzada.

5.1. Esquema propuesto

5.1.1. Estimaciones de densidad

El ruido aditivo que se usa en nuestros experimentos se interpreta como la generación de la estimación de densidad de probabilidad que describe la distribución del vector de entrenamiento. El trabajo realizado en [59] implica el uso de esta estrategia para presentar a la red un nuevo conjunto de datos durante cada época en la etapa de entrenamiento. Esta vez usaremos la estrategia de manera diferente, con el objetivo de incrementar la diversidad en el ensamble. Se considera el entrenamiento de cada red miembro a partir del conjunto de datos de entrenamiento generado con ruido aditivo sin modificarlo en cada época.

Se tiene un conjunto de entrenamiento inicial $\mathcal{I}$ de tamaño s que será expandido usando la estimación de núcleos para obtener un nuevo conjunto de entrenamiento $\mathcal{L}$ de tamaño S donde $s \leq S$. El tamaño del conjunto expandido puede ser tan grande como se quiera y en nuestro caso tiene el mismo tamaño que el conjunto original ya que se afecta cada patrón de $\mathcal{I}$ para generar un nuevo patrón que lo sustituirá para obtener un nuevo conjunto $\mathcal{L}$.

Los patrones de entrenamiento originales a los que se les adiciona ruido son respetados mientras se crean muestreos a partir de la estimación de densidad del vector de entrenamiento real. Una estimación de densidad se define como sigue. Se supone que X es un vector aleatorio que se define por valores en un espacio Euclideano R^d y supóngase también que la distribución de X es descrita por una función de densidad de probabilidad f. K es una densidad de probabilidad que define el "núcleo" y supongamos un valor $h > 0$. Si $X_1, \ldots, X_n$ es una muestra de n observaciones independientes de X, entonces la estimación de núcleo de f del punto n correspondiente a K y h, es la densidad dada por la expresión 5.1 [59].

$$f_{n,h}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h^d} K\left(\frac{x - X_i}{h}\right), x \in R^d \tag{5.1}$$

Las características del ruido aditivo se controlan por la selección de K y h. Sean n muestras de entrenamiento disponibles para una red, si se reemplaza una densidad de probabilidad original f por su estimación de núcleo $f_{n,h}$ del punto n, esto resulta en una función de error modificada a minimizar por la red. Por esto el ruido aditivo

debe ser seleccionado de manera que la función de pérdida modificada se aproxime a la función de pérdida original lo mejor posible.

Introducir ruido aditivo implica reemplazar un conjunto finito $\{x_1, \ldots, x_n\}$ de vectores muestreados a partir de la densidad original f por una fuente infinita de vectores de entrenamiento representada por $f_{n,h}$ donde h define el nivel de ruido.

Supongamos que el vector $w \in R^l$ contiene los pesos y sesgos de la red y definamos las dimensiones de los vectores de entrada y salida de la red como k y m, respectivamente. Entonces la red representa una correspondencia continua $g : R^k \times R^l \to R^m, y = g(x, w)$. Se asume que los pesos w están restringidos a un subconjunto $W \subset R^l$, que $g : R^k \times W \to R^m$ de manera que $w \mapsto g(x, w)$ es continua para cada $x \in R^k$, y que $x \mapsto g(x, w)$ es medible para cada $w \in W$. Dado esto, consideramos a g como una red neuronal [59].

Ahora supongamos que $x_1, \ldots x_n \in R^k$ son vectores de las clases $j_1, \ldots, j_n \in \{1, \ldots, m\}$. Entonces se tienen n vectores de entrenamiento $z_i = (x_i, e_{j_i}), i = 1, \ldots, n$ donde $e_j = \left(0, \ldots, 0, \overset{\text{j}}{1}, 0, \ldots\right)$ es el vector que define la clase j para un vector x. Se crean nuevos vectores de entrenamiento adicionando ruido a los vectores x_i. Supongamos que se tiene un vector aleatorio S de ruido con densidad de probabilidad K definida por la densidad normal $K(x) = (2\pi)^{-k/2} e^{(-\|x\|^2/2)}, x \in R^k$. El nivel de ruido para la clase j se controla con un parámetro $h^j > 0$ denotando $h = (h^1, \ldots, h^m)$. Consideremos el procedimiento:

1. Seleccionar $i \in \{1, \ldots, n\}$ con igual probabilidad para cada índice.

2. Generar una muestra s a partir de la densidad K.

3. Definir $x_h^{(n)} = x_i + h^{i_i} s, j_h^{(n)} = j_i$

Esta estrategia genera muestras $x_h^{(n)}$ y $j_h^{(n)}$ de un nuevo vector aleatorio $X_h^{(n)}$ y una nueva variable aleatoria discreta $J_h^{(n)}$. Los vectores de entrenamiento con ruido $(x_h^{(n)}, e_{j_h^{(n)}})$ pueden suponerse como instancias del vector aleatorio $Z_h^{(n)} = (X_h^{(n)}, Y_h^{(n)})$, y de esto se observa que $X_h^{(n)}$ tiene la densidad definida en la expresión 5.2.

$$f_{n,h}(x) = \frac{1}{n} \sum_{i=1}^{n} K_{h^j i} \left(x - x_i\right), x \in R^k \tag{5.2}$$

donde se usa la notación $K_\rho(x) = \rho^{-k} K(x/\rho), \rho > 0, x \in R^k$ [59].

Este procedimiento de re-muestreo puede ser considerado como una instancia del procedimiento de estimación denominado "*Bootstrap* suavizado" que consiste en obtener a $f_{n,h}$ a partir de $x_1, \ldots, x_n$. Cada parte de la densidad definida en 5.2 representa un "cúmulo" centrado en x_i cuya amplitud es controlada por el parámetro h^{j_i} (para cada punto i de clase j), por lo que h determina el nivel en el que las muestras discretas x_i son suavizadas.

La meta del entrenamiento de una red puede ser considerada como el encontrar una solución $\bar{w}$ con mínimos cuadrados al sistema de ecuaciones $y_i = g(x_i, w), i = 1, \ldots, n$. Ciertas situaciones son no determinadas por el hecho de que la cantidad de vectores de entrenamiento disponibles n es menor que la cantidad de pesos l, lo que hace aceptable que la adición de ruido pueda ser útil. Debido a la no unicidad de los vectores de minimización, y dependiendo del valor inicial de w, la minimización de la función de pérdida obtenida por una red puede producir vectores de peso que estén lejos de los vectores de minimización de la función de pérdida real. La adición de ruido alivia el problema de la no unicidad creando un sistema sobredeterminado cuando se trata de entrenar una red neuronal [59].

No se definen parámetros de suavizado constantes h^j y se permite que el suavizado varíe como una función del número de muestras disponibles de cada clase. Por lo tanto, se consideran m secuencias infinitas $(h_n^j)_n$, lo que se denota como $H_n^j := h^j_{N_n^j}$ y $H_n = (H_n^1, \ldots, H_n^m)$.

Se ha obtenido el valor del parámetro de suavizado maximizando la función de probabilidad validada por cruza. Este método permite manipular los datos disponibles $X_1, \ldots, X_n$. Se denota a $f_{n,h,i}$ como la estimación de núcleo de f obtenida de la i-ésima observación y queda definida por la expresión 5.3. Posteriormente, se considera la expresión de probabilidad validada por cruza (definida en 5.4) y se selecciona el valor de h que maximice $L(h)$. La expresión 5.3 puede considerarse como una medida del grado al que las muestras $X_j, j \neq i$, cuando se suavizan con h, "describen" la observación X_i. Así, un valor grande de $f_{n,h,i}(X_i)$ indica un buen ajuste y, por lo tanto, un buen valor de h; por el contrario, un valor pequeño indica un valor deficiente de h.

$$f_{n,h,i}(x) = \frac{1}{(n-1)\,h^d} \sum_{j=1}^{n} K_h\left(x - X_j\right), \text{ donde } j \neq i \tag{5.3}$$

$$L(h) = \Pi_{i=1}^{n} f_{n,h,i}(X_i) \tag{5.4}$$

La idea de la adición de ruido se considera para generar diferentes conjuntos de datos para el entrenamiento de los clasificadores que integran el ensamble. Posteriormente, se agregan las salidas de cada red neuronal usando el "promedio" de las salidas de los miembros o "votación por mayoría" para obtener la salida final del ensamble.

En el esquema *Bagging* cada red neuronal es entrenada usando muestras de entrenamiento finitas que pueden repetirse para cada clasificador lo cual implica que la red se concentra cada vez más en estos puntos de manera que se puede provocar una mala generalización. Una posible solución a esto sería disponer de conjuntos infinitos de puntos durante el entrenamiento de la red. Sin embargo, esta vez presentamos un esquema que genera los muestreos del conjunto de entrenamiento a través de la estimación de núcleos de los vectores originales para presentarlos en un conjunto de entrada finito para realizar el entrenamiento de cada red. Nuestro esquema se denomina "*Bagging* con Estimación de Densidades (*Bagging* con ED)".

Se pretende simular lo que ocurre en la realidad como lo intenta *Bootstrap* en *Bagging* clásico. Se sabe que los patrones de entrada poseen cierto ruido al presentarse a la red neuronal y mediante la estimación de núcleos se pretende emular tal situación.

5.2. Resultados experimentales

Se ha estudiado el desempeño de nuestro método aplicándolo a algunos problemas de clasificación del mundo real. Nuestros conjuntos de datos seleccionados son Sonar, Iono, Waveform, BreastCancer, Bupa, DNA, Iris y Protein. La descripción que define los aspectos más importantes de estos conjuntos de datos se presenta en la tabla 4.1 del capítulo previo.

De igual manera que nuestros experimentos anteriormente descritos, esta vez se han realizado simulaciones usando las herramientas de redes neuronales de MATLAB. Se usaron las funciones de aprendizaje de red *traingdx* y *trainlm* como algoritmos de entrenamiento. Los ensambles definidos consisten de 15 clasificadores. Se realizaron 20 iteraciones aplicando validación de cruza de 10 puntos. Se indica la precisión en porcentaje (%) y el tiempo en segundos (s).

Los valores h obtenidos para cada clase en los problemas estudiados se indican en la tabla 5.1.

La tabla 5.2 muestra los resultados obtenidos por nuestro esquema "*Bagging* con Estimación de Densidades (*Bagging* con ED)" usando las técnicas de agregación "votación por mayoría" y la del "promedio de las salidas".

Mostramos tiempo y precisión para cada problema de clasificación mencionado. También hacemos la comparación contra el desempeño de una red neuronal. Cada resultado reportado implica validación cruzada de 10 bloques (*10 fold cross validation*) repetida 20 veces.

A partir de la tabla 5.2, podemos observar que "*Bagging* con ED" puede mejorar la precisión de la red base en algunos de los conjuntos de datos, pero en otros casos ofrece un desempeño deficiente.

Tabla 5.1: Parámetro h

Conjunto de datos	Clase	Ancho de banda (h)
Iris	1	0.22000
	2	0.17000
	3	0.16000
Breast Cancer	1	0.51000
	2	1.59000
Bupa	1	0.06900
	2	0.06200
Protein	1	0.05725
	2	0.05750
	3	0.04050
	4	0.08450
Sonar	1	0.08950
	2	0.10900
Iono	1	0.54800
	2	0.13000
Dna	1	0.42100
	2	0.40000
	3	0.51000
Waveform 40	1	1.03000
	2	1.01750
	3	1.02200

Tabla 5.2: Desempeño de "*Bagging* con Estimación de Densidades (*Bagging* con ED)"

	Desempeños					
Conjuntos de datos	Con votación por mayoría		Con promedio de salidas		Red neuronal individual	
	Precisión	Tiempo (s)	Precisión	Tiempo (s)	Precisión	Tiempo (s)
BreastCancer	65.299 ± 0.196	29.812 ± 2.352	65.240 ± 0.220	31.117 ± 4.557	95.224 ± 1.455	25.707 ± 0.889
DNA	93.030 ± 1.401	193.571 ± 0.692	93.610 ± 0.366	191.603 ± 0.634	91.845 ± 0.213	23.908 ± 0.023
Iono	88.449 ± 0.603	36.858 ± 2.391	88.444 ± 0.603	37.471 ± 2.318	90.209 ± 1.067	4.463 ± 0.902
Bupa	58.963 ± 2.715	48.910 ± 25.374	58.048 ± 1.904	53.177 ± 0.325	68.170 ± 2.325	0.720 ± 0.208
Iris	96.800 ± 1.079	38.569 ± 21.438	97.466 ± 1.079	40.818 ± 20.611	95.400 ± 5.324	1.879 ± 0.421
Protein	83.290 ± 0.544	100.620 ± 0.695	83.099±0.382	133.614±106.407	76.134 ± 6.343	19.227 ± 1.535
Sonar	80.328 ± 1.595	134.530 ± 67.231	81.214 ± 1.151	133.955 ± 19.764	83.500 ± 6.452	6.958 ± 0.868
Waveform	83.500 ± 2.078	1188.100 ± 30.673	85.700 ± 0.298	1191.166 ± 23.809	62.770± 1.546	12.078 ± 0.822

5.3. Conclusión

Esta vez se considera aplicar diversidad al ensamble mediante la modificación de los patrones de entrada. Esta estrategia permite generar conjuntos de entrenamiento nuevos para cada MLP modificando los patrones de entrenamiento originales mediante la expansión del conjunto de entrenamiento disponible. La idea implica que cada clasificador del ensamble tendrá diferentes patrones que dificilmente se repetirán aunque la esencia de cada patrón es aproximadamente la misma. Nuestros experimentos no contemplan los mismos procedimientos realizados en [59] ya que tal trabajo considera que en cada época de entrenamiento se genera un nuevo conjunto mediante la técnica de adición de ruido, de manera que la red enfrenta diferentes patrones conforme se realiza el entrenamiento lo que permite evadir el problema del sobreajuste del MLP.

En nuestro caso, modificamos el conjunto de entrenamiento adicionando ruido con distribución normal con la intención de que cada clasificador se entrene con un diferente conjunto de entrada. Sin embargo, esta estrategia no basta para dar la suficiente diversidad al ensamble de manera que pueda tener resultados competitivos con *Bagging*. La pérdida de precisión del ensamble puede ser originada porque cada clasificador establece una correspondencia muy similar a partir de conjuntos que no son los originales. Tales correspondencias definen poca diversidad pues su imprecisión es similar ya que cada una se concentra en un conjunto ligeramente diferente. Así, es difícil encontrar miembros en el ensamble mucho mejores que otros y por lo tanto el desempeño de *Bagging* no es alcanzado en todos los problemas. Para los casos de *Protein* y *DNA* el desempeño de nuestro esquema es muy bueno pues la precisión es similar a la de *Bagging* mostrada en las tablas 4.7 y 4.8; sin embargo, el consumo de tiempo es mucho menor. Para los casos de *Bupa* y *BreastCancer* (cuyo desempeño no se muestra para *Bagging*) el desempeño es muy malo. Para el resto de los casos nuestro esquema se acerca al desempeño del método clásico (ver tablas 4.7 y 4.8).

Una dificultad con los métodos de estimación de densidad validados por cruza es probar que la estimación resultante sea consistente, lo que haría falta en nuestros experimentos. Nuevas ideas surgen a partir de todos los experimentos realizados, tales como la posible combinación de patrones originales y extendidos con la técnica *Bootstrap*, la cual podría resultar mejor que el esquema que hemos propuesto. En la práctica existe únicamente un número finito de muestras originales disponibles y consideraciones computacionales limitan el número de muestras a ser generadas a partir de la estimación de núcleo. Además, no hay garantía de encontrar el mínimo global de la función de pérdida [59]. Por lo tanto, es necesario encontrar métodos para seleccionar las características del ruido aditivo en problemas de clasificación con varias clases.

Capítulo 6

Conclusiones

Nuestros estudios incorporaron nuevas estrategias alrededor de la idea original. Se aplicó análisis de sensibilidad para variar la arquitectura de los clasificadores y se consideraron diversas alternativas de agregación de ruido para generar nuevas configuraciones de parámetros. Se usó una técnica diferente a *Bootstrap* que implica la estimación de densidades para generar el nuevo conjunto de entrenamiento de cada red neuronal que integra el ensamble.

El trabajo realizado motiva en la continuación de la investigación referente a aminorar el tiempo de aprendizaje de un ensamble de redes neuronales. El aprendizaje de las redes neuronales requiere de un algoritmo que manipule componentes aleatorios y esto define cierta dificultad para estructurar un ensamble completo mediante el esquema que se propone basado en la adición de ruido a los parámetros de una red entrenada. Actualmente, el área de investigación relacionada con la combinación de clasificadores posee dos principales paradigmas: *Bagging* y *Boosting*. Nuestros estudios toman el principio de agregar las salidas de los clasificadores que se encuentran inmersos en el primero de estos paradigmas, pero aplicamos una manera diferente para crear los clasificadores del ensamble.

Estudios posteriores deben atender nuevas estrategias, la definición de un esquema rápido necesita contemplar factores para crear diversidad de manera que el aprendizaje requiera menos tiempo que los métodos actuales. La adición de ruido parecía ser una posibilidad para este objetivo usando redes neuronales. Sin embargo, las estrategias usadas en este trabajo no permitieron obtener la diversidad deseada en el ensamble sin que las configuraciones resultantes se alejaran de las soluciones factibles.

Cuando se entrena una red neuronal se define una configuración de varios planos en el espacio de los pesos. Así, cada patrón tiene asociado un hiperplano en tal espacio. Cuando se realiza alguna perturbación, la configuración dada por los diferentes planos puede variar y, por lo tanto, la solución relacionada también, lo cual motiva seguir investigando en torno a este tema.

Se debe recordar que la meta de la generalización es aprender la función de generación de la salida y no los datos de salida en sí pues de otro modo puede ocurrir un sobreajuste de los datos [70].

La figura 6.1 indica la relación de cada patrón de entrada de dimensión n con una

configuración de $n + 1$ parámetros libres. Por lo tanto, cada punto en el espacio de entrada extendido de dimensión $n + 1$ puede asociarse a un hiperplano en el espacio de pesos de dimensión $n + 1$. La misma relación ocurre en la dirección inversa.

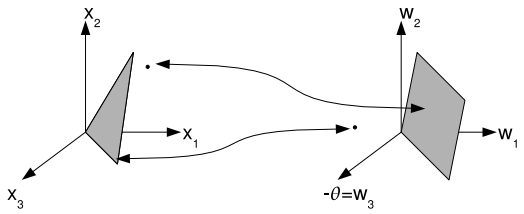


Figura 6.1: Dualidad del espacio de entrada y salida.

Por lo tanto, la red entrenada define un conjunto de relaciones entre los puntos de entrada y los planos en el espacio de pesos. Al realizar la perturbación se obtienen variaciones de las configuraciones originales de tal manera que si se desea generar varias redes entonces deben obtenerse múltiples configuraciones. El problema radica en que las configuraciones deben ser convenientes para obtener una solución factible y hasta el momento esto se ha logrado mediante la adaptación de los valores de los parámetros usando un algoritmo de aprendizaje. Las configuraciones que generamos definen redes muy similares o cuyas soluciones son inconvenientes; sin embargo, nuestro objetivo es obtener redes diferentes cuyas soluciones sean factibles. Los MLP tradicionales logran esto mediante la configuración aleatoria inicial de sus pesos y su algoritmo de entrenamiento.

El problema de la diversidad aunado con la restricción de tiempo fue enfocado a través de la modificación de los datos de entrada. Una nueva generación de subconjuntos de entrenamiento se aplicó a la generación de clones mediante la adición de ruido. Este intento no genera resultados competentes debido a que la función establecida por la red base no varía, sino que sigue siendo prácticamente la misma en todos los clones. De tal forma, aunque los patrones varían, el experimento indica que esta estrategia no genera redes lo suficientemente diversas.

En el ensamble clásico, cada red neuronal entrenada establece una diferente configuración final en el espacio de pesos y esto define una función diferente para generar una solución, lo que asegura que el ensamble posea clasificadores distintos con precisiones similares.

Esto lo pudimos apreciar a través de varios experimentos realizados. Entrenamos una red neuronal completamente y generamos muestras mediante *Bootstrap* para encontrar otras configuraciones a partir de la definición de los pesos de la red entrenada. Para eso se crearon clones de la red base y se re-entrenaron (que es la misma idea de agregar ruido pero con un algoritmo de adaptación de valores). Las redes clones generaron resultados muy similares ya que las configuraciones en el espacio de pesos definen funciones muy cercanas a la de la red entrenada. Sin embargo, usando el mismo principio, si entrenamos parcialmente una red neuronal base y a partir de ella generamos copias y re-entrenamos para generar otras redes, entonces obtenemos

mejores resultados que con el *Bagging* clásico en un tiempo menor, porque las funciones obtenidas son diferentes. Dado que esta última estrategia es prácticamente una versión de *Bagging*, no la consideramos como una nueva técnica para crear ensambles.

Otra idea fue afectar las entradas y para esto se contempló la estrategia consistente en usar estimación de núcleos para generar nuevos conjuntos de datos de entrenamiento sustituyendo la técnica *Bootstrap*.

Las estrategias que consideraban el uso de la perturbación para generar diferentes redes no permitieron dar diversidad factible al ensamble. Si la perturbación a los parámetros de la red es muy grande, el conocimiento de la red base puede ser degradado en gran escala. Esta situación es similar a la del entrenamiento parcial de la red con la diferencia de que en ésta el conocimiento no puede ser degradado ya que los pesos se modifican de una manera adaptativa mediante un algoritmo de entrenamiento. La estrategia que implica la aplicación de análisis de sensibilidad para conocer cuáles eran los pesos menos importantes de una red no permite resultados satisfactorios porque no se modifican los pesos de manera significativa por lo que la variación de ciertos pesos podría no cambiar el significado del resultado que la configuración original implica. Sin embargo, si tal perturbación es más grande entonces podría perderse también la precisión original y el tiempo necesario para encontrar una red factible a partir de las perturbaciones es muy alto, lo que nos pone en una posición similar a la de la simple agregación de ruido. El hecho de no poseer la diversidad suficiente se origina porque cada clasificador se equivoca en los mismos patrones, lo que no sucede al entrenar completamente a cada red neuronal pues existen errores sobre diferentes patrones.

Por otro lado, los métodos propuestos para la reducción de dimensiones que generan nuevas entradas a un ensamble, permiten identificar un área de estudio que consiste en minimizar el tiempo para crear tal reducción mediante redes cuello de botella. Los métodos que hemos propuesto para este problema indican soluciones competentes con los métodos del estado del arte en el contexto de precisión. Sin embargo, el tiempo inmerso en la reducción de dimensiones es muy alto y es en esta parte donde se requiere de un mayor esfuerzo para definir las reducciones con mayor rapidez.

Nuestros estudios muestran que existen varias estrategias para obtener ensambles de clasificadores. El problema relacionado con el costo computacional de un ensamble de redes neuronales evidentemente se debe a la excesiva cantidad de tiempo que se requiere para entrenar una red. De acuerdo a nuestra experiencia adquirida con este trabajo, la generación rápida de ensambles de redes neuronales requiere de mayor trabajo en la definición de algoritmos de aprendizaje más rápidos. En nuestro caso particular, la idea de generar conocimiento variado, a partir de uno ya existente, parece una idea brillante. Sin embargo, la restricción de costo temporal de este tipo de clasificadores limita severamente las posibilidades de aplicar estrategias no tan costosas como las actuales. Una muy buena opción es disminuir la dimensión de los datos antes de realizar el entrenamiento de las redes. Nuestro esquema de la "red de cuello de botella modificada" permite una reducción sin gran pérdida de

información. Esto ofrece la gran ventaja de poder entrenar redes neuronales para la tarea de clasificación en un tiempo mucho menor al requerido cuando se usan los datos no reducidos. Así, los ensambles creados a partir de diversas reducciones de los mismos datos, ofrecen muy buena precisión y su costo computacional es menor que los ensambles construidos de manera clásica. La desventaja de este método radica en el hecho de que se tienen que entrenar dos conjuntos de redes para obtener la salida final del ensamble.

La adición de ruido, como hemos mostrado, ofrece una opción para mejorar el desempeño de cualquier red neuronal aunque esta mejora puede ser muy pequeña. El análisis de la estructura de la red neuronal es una estrategia que, aunada a la anterior, podría generar ensambles con mayor diversidad. Sin embargo, identificar las zonas importantes de la configuración de una red neuronal requiere de cómputo excesivo por lo que en este trabajo descartamos su inclusión y optamos por una variante de ella. El entrenamiento de una red neuronal usando un conjunto de datos de entrenamiento distinto es el método más eficiente para obtener conocimiento diferente, y esto se observa con el desempeño de nuestro esquema que implica entrenar dos redes base. En este caso, los clones contribuyen muy poco (a veces casi nada) en el desempeño del ensamble. Nuestros esquemas ofrecen una alternativa para obtener de manera rápida una mayor precisión que la ofrecida por la red base (que puede ser cualquier red neuronal).

De acuerdo a lo observado en nuestro trabajo, la investigación puede continuar enfocada sobre dos puntos específicos: en el diseño de algoritmos de aprendizaje más rápidos (para redes neuronales) y en la definición de un método que defina una valor de σ adecuado para cada conjuunto de datos. El análisis de sensibilidad es una técnica muy poderosa para conocer detalles acerca de la estructura de las redes neuronales y consideramos que no puede faltar en el trabajo que se realice para definir ensambles de redes neuronales más rápidos. Disminuir el costo de este análisis es también una buena opción, sin embargo no es una estrategia directa en la definición del conocimiento de la red neuronal y puede ser usada como herramienta.

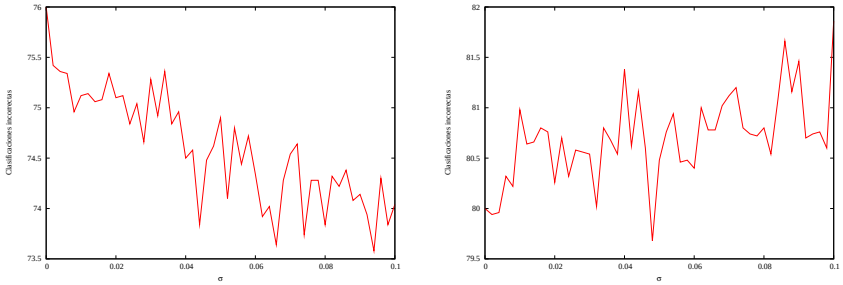
Apéndice A

Resultados obtenidos

Las tablas A.2, A.5 muestran resultados usando ensambles integrados por 50 y 100 clasificadores. A través de ellas se puede observar que nuestro esquema "Clonación usando Dos Redes Base (CDRB)" es una buena opción para definir un ensamble rápido a partir de dos configuraciones de parámetros obtenidas por el entrenamiento clásico.

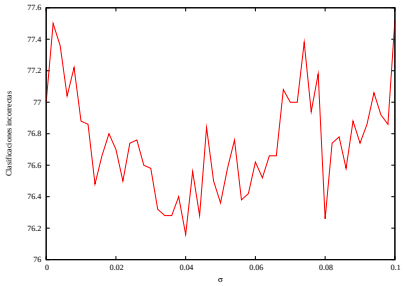
Las tablas A.3 y A.4 muestran los resultados obtenidos usando el esquema "Clonación usando Dos Redes Base con MSE (CDRB-MSE)". Tales resultados corresponden a ensambles de 50 y 100 miembros. Se puede observar que considerar la medida de error MSE para seleccionar los clones es mejor que hacerlo a través de las clasificaciones erróneas. Los resultados son, en general, mejores que los mostrados en las tablas A.2, A.5, A.6, A.7.

Las tablas A.6, A.7 muestran resultados de ensambles integrados por 50 y 100 redes neuronales. Estas tablas indican que los ensambles que integran las redes base como miembros mejoran por muy poco el desempeño de los ensambles que han sido formados únicamente por clones de las redes base. La razón es que cada clon es muy similar a algunas de las redes base. Por lo tanto, la inclusión de estas redes entrenadas podría ayudar por muy poco o en nada pues es posible que algunos clones sean mejores que ambas (al menos de manera marginal).

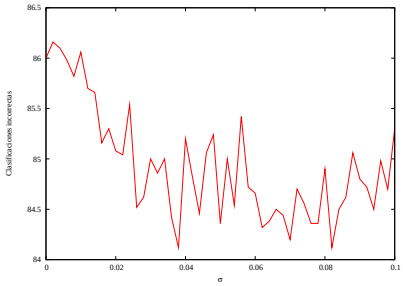


(a) Primero

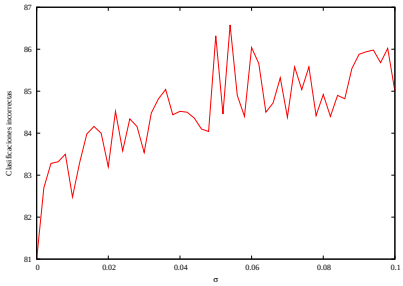
(b) Segundo



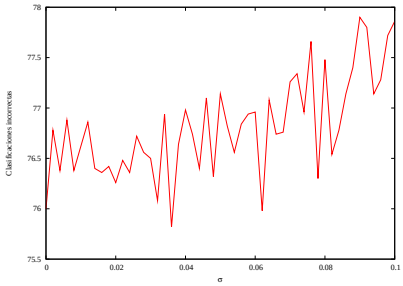
(c) Tercero



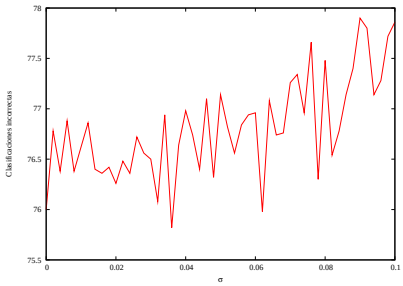
(d) Cuarto



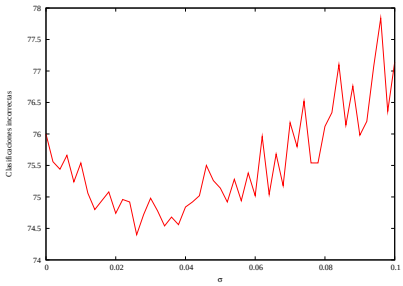
(e) Quinto



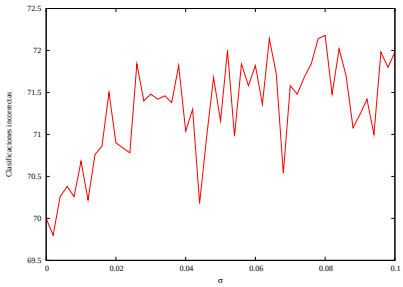
(f) Sexto



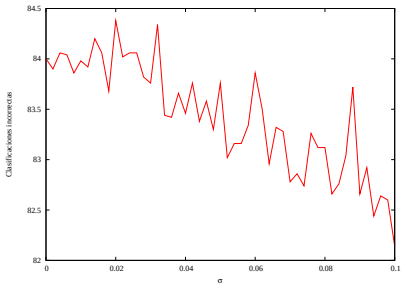
(g) Séptimo



(h) Octavo

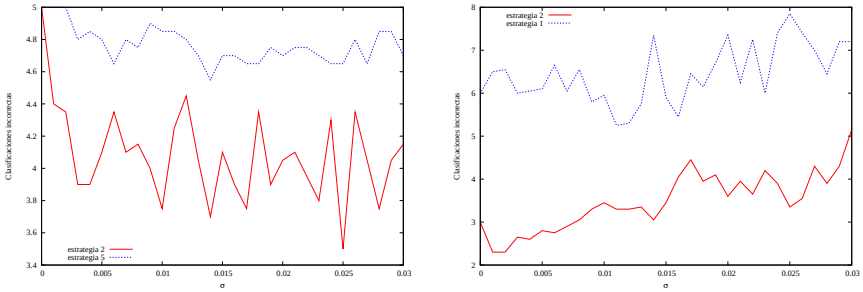


(i) Noveno



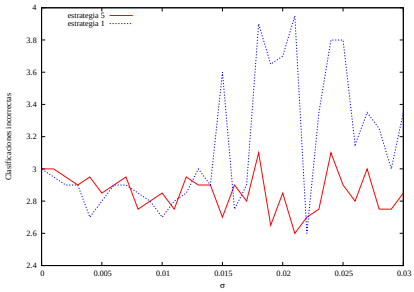
(j) Décimo

Figura A.1: Desempeño de "Clonación con Simple Adición de Ruido (CSAR)" (50 miembros) aplicado a los datos de Waveform de 40 características.

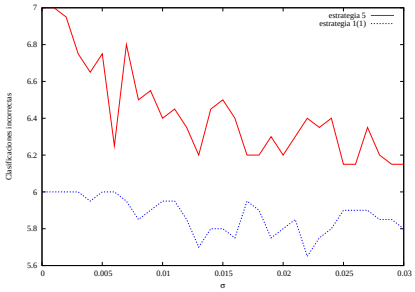


(a) Primero

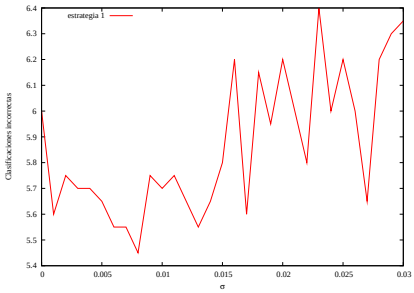
(b) Segundo



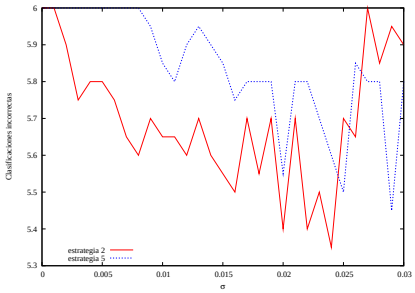
(c) Tercero



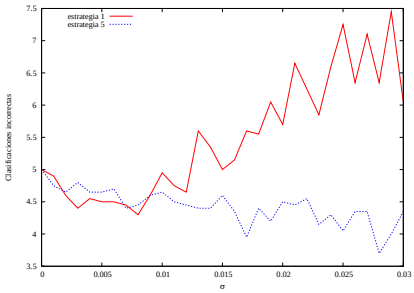
(d) Cuarto



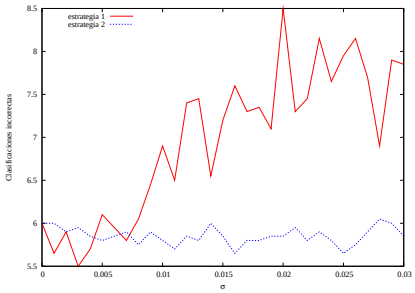
(e) Quinto



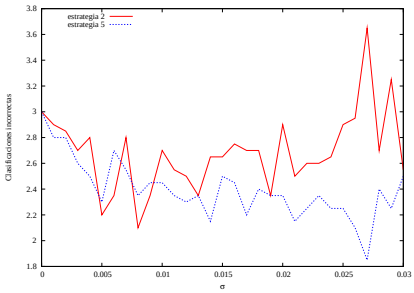
(f) Sexto



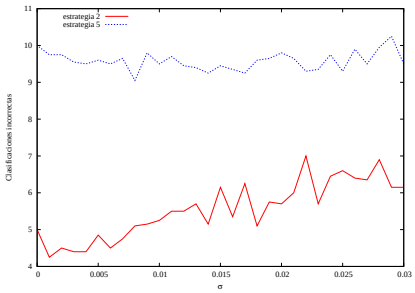
(g) Séptimo



(h) Octavo



(i) Noveno



(j) Décimo

Figura A.2: Desempeño de "Clonación con Simple Adición de Ruido (CSAR)" (50 miembros) aplicado a los datos de Sonar

Tabla A.1: Desempeño de ensamble de dos redes base

Precisión (%)	Tiempo (s)
Protein	
81.920 ± 1.986	680.051 ± 20.105
Iono	
89.160 ± 1.612	9.050 ± 2.101
Bupa	
62.890 ± 2.472	7.894 ± 1.003
Sonar	
81.110 ± 2.087	25.034 ± 5.003
Waveform	
79.200 ± 3.082	420.130 ± 53.002
Dna (3000 épocas)	
91.490 ± 0.369	70.141 ± 10.013

Tabla A.2: Desempeño de "Clonación usando Dos Redes Base (CDRB)" (50 clasificadores)

Clasificador	σ	Precisión (%)	Tiempo (s)
Protein			
Ensamble	0.001	82.740±2.825	2802.988±1.167
Ensamblados	0.001	81.440±0.775	-
Ensamble	0.002	83.860±1.454	2458.599 ±0.991
Ensamblados	0.002	82.080±0.724	-
Ensamble	0.0035	83.220±1.912	2531.579 ±1.101
Ensamblados	0.0035	81.550±0.725	-
Iono			
Ensamble	0.001	89.610±1.122	45.960 ±0.130
Ensamblados	0.001	88.760±0.345	-
Ensamble	0.002	89.430±1.552	45.162 ± 0.327
Ensamblados	0.002	88.840±0.281	-
Ensamble	0.0035	89.950±1.107	45.998 ±0.141
Ensamblados	0.0035	88.960±0.243	-
Bupa			
Ensamble	0.001	64.060±2.410	37.070±0.181
Ensamblados	0.001	64.160±0.471	-
Ensamble	0.002	64.590±3.200	31.359±0.239
Ensamblados	0.002	64.530±0.438	-
Ensamble	0.0035	63.830±2.852	38.349±0.142
Ensamblados	0.0035	63.600 ± 0.460	-
Sonar			
Ensamble	0.001	82.750±2.728	114.121±0.220
Ensamblados	0.001	81.215±0.559	-
Ensamble	0.002	83.270±2.056	90.243±0.122
Ensamblados	0.002	80.870±0.518	-
Ensamble	0.0035	81.950±1.491	111.185±0.260
Ensamblados	0.0035	80.200±0.527	-
Waveform			
Ensamble	0.001	77.200±4.234	1703.071±0.768
Ensamblados	0.001	70.770±0.876	-
Ensamble	0.002	79.200±4.125	1773.830± 0.789
Ensamblados	0.002	69.390±1.174	-
Ensamble	0.0035	79.510±4.140	1780.311±0.811
Ensamblados	0.0035	69.210±1.181	-
Dna			
Ensamble	0.001	92.400±0.468	289.354±1.092
Ensamblados	0.001	92.140±0.080	-
Ensamble	0.002	92.520±0.253	312.594±0.844
Ensamblados	0.002	92.140±0.069	-
Ensamble	0.0035	92.750±0.375	257.361±0.702
Ensamblados	0.0035	92.380±0.085	-

Tabla A.3: Desempeño de "Clonación usando Dos Redes Base con MSE (CDRB-MSE)" (50 clasificadores)

Método	Precisión (%)	Tiempo (s)
Protein		
CDRB-MSE(σ=0.001)	82.747 ± 3.356	2842.229 ± 0.992
CDRB-MSE(σ=0.002)	83.993 ± 1.984	2480.726 ± 1.017
CDRB-MSE(σ=0.0035)	83.067 ± 1.043	2655.626 ± 1.057
Iono		
CDRB-MSE(σ=0.001)	89.347 ± 1.141	41.134 ± 0.051
CDRB-MSE(σ=0.002)	89.979 ± 1.242	41.910 ± 0.192
CDRB-MSE(σ=0.0035)	90.065 ± 1.173	38.086 ± 0.186
Bupa		
CDRB-MSE(σ=0.001)	65.421 ± 1.535	35.587 ± 0.197
CDRB-MSE(σ=0.002)	64.711 ± 2.186	35.435 ± 0.237
CDRB-MSE(σ=0.0035)	63.926 ± 1.921	40.649 ± 0.115
Sonar		
CDRB-MSE(σ=0.001)	83.857 ± 1.082	107.730 ± 0.098
CDRB-MSE(σ=0.002)	82.885 ± 1.545	105.223 ± 0.102
CDRB-MSE(σ=0.0035)	82.107 ± 2.642	98.398 ± 0.056
Waveform		
CDRB-MSE(σ=0.001)	77.700 ± 7.197	1634.948 ± 0.673
CDRB-MSE(σ=0.002)	78.100 ± 6.060	1649.661 ± 0.750
CDRB-MSE(σ=0.0035)	78.933 ± 5.784	1674.324 ± 0.889
DNA		
CDRB-MSE(σ=0.001)	92.295 ± 0.300	262.733 ± 0.797
CDRB-MSE(σ=0.002)	92.310 ± 0.464	279.146 ± 0.976
CDRB-MSE(σ=0.0035)	92.420 ± 0.264	233.941 ± 0.568

Tabla A.4: Desempeño de "Clonación usando Dos Redes Base con MSE (CDRB-MSE)" (100 clasificadores)

Método	Precisión (%)	Tiempo (s)
Protein		
CDRB-MSE(σ=0.001)	82.236 ± 3.223	5977.207 ± 2.210
CDRB-MSE(σ=0.002)	82.172 ± 3.865	4852.300 ± 1.937
CDRB-MSE(σ=0.0035)	83.897 ± 1.895	4495.974 ± 1.257
Iono		
CDRB-MSE(σ=0.001)	89.900 ± 0.999	85.311 ± 0.279
CDRB-MSE(σ=0.002)	89.381 ± 0.936	84.867 ± 0.337
CDRB-MSE(σ=0.0035)	88.928 ± 1.032	76.057 ± 0.269
Bupa		
CDRB-MSE(σ=0.001)	64.285 ± 2.204	71.885 ± 0.284
CDRB-MSE(σ=0.002)	62.714 ± 1.663	70.125 ± 0.235
CDRB-MSE(σ=0.0035)	64.742 ± 2.159	80.688 ± 0.533
Sonar		
CDRB-MSE(σ=0.001)	83.171 ± 1.009	208.457 ± 0.405
CDRB-MSE(σ=0.002)	82.585 ± 1.540	210.025 ± 0.245
CDRB-MSE(σ=0.0035)	83.514 ± 1.890	207.422 ± 0.528
Waveform		
CDRB-MSE(σ=0.001)	79.800 ± 2.238	3305.864 ± 1.257
CDRB-MSE(σ=0.002)	79.300 ± 5.188	3454.390 ± 0.550
CDRB-MSE(σ=0.0035)	81.233 ± 1.323	3501.126 ± 0.993
DNA		
CDRB-MSE(σ=0.001)	92.410 ± 0.213	530.983 ± 1.178
CDRB-MSE(σ=0.002)	92.755 ± 0.207	563.316 ± 1.171
CDRB-MSE(σ=0.0035)	92.470 ± 0.488	468.349 ± 1.225

Tabla A.5: Desempeño de "Clonación usando Dos Redes Base (CDRB)" (100 clasificadores)

Clasificador	σ	Precisión (%)	Tiempo (s)
Protein			
Ensamble	0.001	82.420±4.736	4809.927 ±1.859
Ensamblados	0.001	78.910±1.101	-
Ensamble	0.002	83.830±2.093	4838.522 ±2.514
Ensamblados	0.002	82.400±0.455	-
Ensamble	0.0035	84.280±2.942	4837.847 ±2.148
Ensamblados	0.0035	82.720±0.323	-
Iono			
Ensamble	0.001	89.750±0.902	90.725 ±0.327
Ensamblados	0.001	88.750±0.276	-
Ensamble	0.002	89.890±1.378	101.027 ±0.197
Ensamblados	0.002	88.830±0.235	-
Ensamble	0.0035	89.900±1.294	90.570 ±0.264
Ensamblados	0.0035	88.080±0.410	-
Bupa			
Ensamble	0.001	65.880±2.712	72.805 ±0.283
Ensamblados	0.001	64.660±0.539	-
Ensamble	0.002	64.590±2.035	60.773 ±0.339
Ensamblados	0.002	64.160±0.566	-
Ensamble	0.0035	65.290±2.383	80.072 ±0.378
Ensamblados	0.0035	64.470±0.503	-
Sonar			
Ensamble	0.001	81.820±2.435	223.106 ±0.358
Ensamblados	0.001	80.310±0.576	-
Ensamble	0.002	83.390±1.363	183.193 ±0.243
Ensamblados	0.002	80.540±0.532	-
Ensamble	0.0035	82.490±0.943	221.702 ±0.379
Ensamblados	0.0035	80.870±0.533	-
Waveform			
Ensamble	0.001	78.800±4.220	3460.640 ±2.001
Ensamblados	0.001	70.360±0.942	-
Ensamble	0.002	79.100±3.870	3723.093±2.143
Ensamblados	0.002	70.760±0.922	-
Ensamble	0.0035	79.230±3.452	3666.150±2.351
Ensamblados	0.0035	71.030±0.894	-
Dna			
Ensamble	0.001	92.410±0.347	594.333±1.353
Ensamblados	0.001	92.130±0.139	-
Ensamble	0.002	92.530±0.312	630.814±1.361
Ensamblados	0.002	92.190±0.081	-
Ensamble	0.0035	92.330±0.251	512.405±1.131
Ensamblados	0.0035	92.150±0.150	-

Tabla A.6: Desempeño de "Clonación y combinación de Dos Redes Base (CDRB+)" (50 clasificadores)

Clasificador	σ	Precisión (%)	Tiempo (s)
Protein			
Ensamble	0.001	83.220±2.622	2561.931 ±1.015
Ensamblados	0.001	80.853±1.572	-
Ensamble	0.002	83.990±2.127	2478.267 ±1.179
Ensamblados	0.002	81.420±1.196	-
Ensamble	0.0035	83.060±1.601	2653.094 ±1.048
Ensamblados	0.0035	81.000±1.486	-
Iono			
Ensamble	0.001	89.340±1.141	41.042±0.051
Ensamblados	0.001	88.200±0.425	-
Ensamble	0.002	89.970±1.242	41.865±0.192
Ensamblados	0.002	88.420±0.494	-
Ensamble	0.0035	90.060±1.173	37.718 ±0.186
Ensamblados	0.0035	88.810±0.339	-
Bupa			
Ensamble	0.001	65.420±1.535	35.846±0.197
Ensamblados	0.001	65.060±0.533	-
Ensamble	0.002	64.710±2.186	35.435±0.237
Ensamblados	0.002	64.570±0.615	-
Ensamble	0.0035	63.920±1.921	40.611±0.115
Ensamblados	0.0035	63.850±0.689	-
Sonar			
Ensamble	0.001	83.850±2.199	107.730 ±0.135
Ensamblados	0.001	82.140±0.824	-
Ensamble	0.002	82.880±1.567	105.042 ±0.161
Ensamblados	0.002	81.100±0.631	-
Ensamble	0.0035	82.107±2.373	101.106±0.096
Ensamblados	0.0035	80.160±0.558	-
Waveform			
Ensamble	0.001	77.700±5.434	1646.869 ±0.664
Ensamblados	0.001	69.380±1.424	-
Ensamble	0.002	78.100±4.753	1656.757 ±0.746
Ensamblados	0.002	70.460±1.086	-
Ensamble	0.0035	78.650±4.413	1680.613±0.714
Ensamblados	0.0035	71.320±0.960	-
Dna			
Ensamble	0.001	92.290±0.300	262.733 ±0.797
Ensamblados	0.001	92.060±0.103	-
Ensamble	0.002	92.310±0.464	279.146 ±0.976
Ensamblados	0.002	92.020±0.094	-
Ensamble	0.0035	92.420±0.264	233.683 ±0.568
Ensamblados	0.0035	92.220±0.118	-

Tabla A.7: Desempeño de "Clonación y combinación de Dos Redes Base (CDRB+)" (100 clasificadores)

Clasificador	σ	Precisión (%)	Tiempo (s)
Protein			
Ensamble	0.001	82.230±3.209	5969.119 ±2.280
Ensamblados	0.001	78.950±1.444	-
Ensamble	0.002	82.172±3.026	4853.037 ±2.495
Ensamblados	0.002	75.530±3.010	-
Ensamble	0.0035	83.890±1.796	4494.359 ±1.893
Ensamblados	0.0035	78.300±3.448	-
Iono			
Ensamble	0.001	89.900±0.999	85.190 ±0.279
Ensamblados	0.001	88.980±0.275	-
Ensamble	0.002	89.380±0.936	84.862 ±0.337
Ensamblados	0.002	88.290±0.233	-
Ensamble	0.0035	88.920±1.032	75.263 ±0.269
Ensamblados	0.0035	87.900±0.310	-
Bupa			
Ensamble	0.001	64.280±2.204	72.295 ±0.284
Ensamblados	0.001	64.710±0.726	-
Ensamble	0.002	62.710±1.663	70.739 ±0.235
Ensamblados	0.002	62.470±0.607	-
Ensamble	0.0035	64.740±2.150	80.632 ±0.533
Ensamblados	0.0035	64.950±0.508	-
Sonar			
Ensamble	0.001	83.170±1.708	209.719 ±0.331
Ensamblados	0.001	80.380±0.653	-
Ensamble	0.002	82.580±1.994	210.122 ±0.233
Ensamblados	0.002	80.060±0.672	-
Ensamble	0.0035	83.510±1.579	207.291 ±0.426
Ensamblados	0.0035	81.030±0.588	-
Waveform			
Ensamble	0.001	79.800±1.927	3329.135 ±1.086
Ensamblados	0.001	71.920±1.075	-
Ensamble	0.002	79.300±2.397	3510.011 ± 0.882
Ensamblados	0.002	71.580±0.987	-
Ensamble	0.0035	79.500±2.213	3443.022 ± 0.901
Ensamblados	0.0035	71.750±0.995	-
Dna			
Ensamble	0.001	92.410±0.213	530.739 ±1.178
Ensamblados	0.001	92.110±0.067	-
Ensamble	0.002	92.750±0.207	563.316 ±1.171
Ensamblados	0.002	92.360±0.107	-
Ensamble	0.0035	92.470±0.488	412.998 ±1.225
Ensamblados	0.0035	92.010±0.106	-

Referencias

[1] Ying Zhao, Jun Gao, and Xuezhi Yang. A survey of neural network ensembles. In *Proceedings of International Conference on Neural Networks and Brain, 2005. ICNN and B '05.*, volume 1, pages 438–442, 13-15 Oct. 2005.

[2] Thomas G. Dietterich. Ensemble methods in machine learning. In *MCS '00: Proceedings of the First International Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, volume 1857, pages 1–15. Springer-Verlag, London, UK, 2000.

[3] N. Ueda and R. Nakano. Generalization error of ensemble estimators. *In Proceedings of International Conference on Neural Networks*, 1(3-6):90–95, June 1996.

[4] Peter Sollich and Anders Krogh. Learning with ensembles: How overfitting can be useful. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 190–196. The MIT Press, 1996.

[5] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

[6] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of 13th International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.

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

[8] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, August 1999.

[9] Nikunj C. Oza and Stuart Russell. Online bagging and boosting. In *Eighth International Workshop on Artificial Intelligence and Statistics*, pages 105–112, Key West, Florida. USA, January 2001. Morgan Kaufmann.

[10] L.I. Kuncheva. Diversity in multiple classifier systems (editorial). *Information Fusion*, 6(1):3–4, May 2004.

[11] Pedro Domingos. Why does bagging work? a bayesian account and its implications. In *Procedings of 3rd International Conference on Knowledge Discovery and Data Mining*, pages 155–158, Newport Beach, CA, 1997.

[12] Jerome H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[13] Yves Grandvalet. Bagging can stabilize without reducing variance. In Georg Dorffner, Horst Bischof, and Kurt Hornik, editors, *Lecture Notes In Computer Science; Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, volume 2130, pages 49–56. Springer-Verlag London, UK, 2001.

[14] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4):382–401, 1999.

[15] Ludmila I. Kuncheva and Christopher J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, May 2003.

[16] Catherine A. Shipp and Ludmila Kuncheva. Relationships between combination methods and measures of diversity in combining classifiers. *Information Fusion*, 3(2):135–148, 2002.

[17] R. Mooney, J. Shavlik, G. Towell, and A. Gove. An experimental comparison of symbolic and connectionist learning algorithms. In J. W. Shavlik and T. G. Dietterich, editors, *Proceedings of 11th International Joint Conference on Aritificial Intelligence*, pages 775–780, San Mateo, CA, 1990. Morgan Kaufmann.

[18] Simon Haykin. *Neural Networks - A Comprehensive Foundation (2nd Edition).* Prentice-Hall, 2nd edition, July 6 1998.

[19] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, October 1990.

[20] Raúl Rojas. *Neural networks: a systematic introduction.* Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[21] A.P. Engelbrecht. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Transactions on Neural Networks*, 12(6):1386–1399, Nov. 2001.

[22] Xiaoqin Zeng and D.S.Yeung. Sensitivity analysis of multilayer perceptron to input and weight perturbations. *IEEE Transactions on Neural Networks*, 12(6):1358–1366, Nov. 2001.

[23] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, November 2000.

[24] David W. Opitz and Jude W. Shavlik. Generating accurate and diverse members of a neural-network ensemble. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 535–541. The MIT Press, 1996.

[25] Ludmila I. Kuncheva. A theoretical study on six classifier fusion strategies. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 24(2):281–286, Feb 2002.

[26] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[27] Juan J. Rodríguez, Ludmila I. Kuncheva, and Carlos J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, Oct. 2006.

[28] G. Brown. *Diversity in Neural Network Ensembles*. PhD thesis, School of Computer Science, University of Birmingham, 2003.

[29] R. Battiti and A.M. Colla. Democracy in neural nets: Voting schemes for classification. *Neural Networks*, 7(4):691–707, 1994.

[30] M. P. Perrone and L. N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. In R. J. Mammone, editor, *Neural Networks for Speech and Image Processing*, pages 126–142. Chapman-Hall, 1993.

[31] Nils J. Nilsson. *Learning machines: Foundations of Trainable Pattern Classifying Systems*. McGraw–Hill, New York, 1965.

[32] Derek Partridge and W. B. Yates. Engineering multiversion neural-net systems. *Neural Computation*, 8(4):869–893, 1996.

[33] Y. Partridge, W. Yates, and D. Partridge. Use of methodological diversity to improve neural network generalisation. *Neural Computing and Applications*, 4(2):114–128, August 18 1995.

[34] P. W. Munro B. Parmanto and H. R. Doyle. Improving committee diagnosis with resampling techniques. In M.C. Mozer D.S. Touretzky and M.E. Hesselmo, editors, *In Advances in Neural Information Processing Systems*, volume 8, pages 832–888. MIT Press, Cambridge, MA, 1996.

[35] Richard Maclin and Jude W. Shavlik. Combining the predictions of multiple classifiers: Using competitive learning to initialize neural networks. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 524–531, Montreal, Canadá, 1995. Morgan Kautmann.

[36] A. Swann and N. Allinson. Fast committee learning: preliminary results. *Electronics Letters*, 34(14):1408 – 1410, 9 Jul 1998.

[37] M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *IEEE Transactions on Neural Networks*, 14(4):820–834, July 2003.

[38] William B. Langdon, S. J. Barrett, and B. F. Buxton. Combining decision trees and neural networks for drug discovery. In Evelyne Lutton, James A. Foster, Julian Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, *Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002*, volume 2278, pages 61–70. Springer-Verlag, Kinsale, Ireland, 3-5 2002.

[39] B. Chandrasekaran A.K. Jain. *Dimensionality and Sample Size Considerations in Pattern Recognition Practice*, volume 2, pages 835–855. Handbook of Statistics, North-Holland, Amsterdam, 1982.

[40] I.T.Jollife. *Principal Component Analysis (Second Edition).* Springer Verlag, New York, 1986.

[41] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 2nd edition, 28 September 1991.

[42] Mevlut Ture, Imran Kurt, and Zekeriya Akturk. Comparison of dimension reduction methods using patient satisfaction data. *Expert Systems and Applications*, 32(2):422–426, 2007.

[43] A. K. Jain and D. Zongker. Feature selection: evaluation, application and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):153–148, Feb 1997.

[44] P. Pudil J. Novovicova and J. Kittler. Divergence based feature selection for multimodal class densities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2):218–223, Feb 1996.

[45] Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. In *Artificial Intelligence*, volume 97, pages 245–271. Elsevier Science Publishers Ltd., Essex, UK, December 1997.

[46] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, December 1997.

[47] D.E. Brown F.Z. Brill and W.N. Martin. Fast genetic selection of features for neural network classifiers. *IEEE Transactions on Neural Networks*, 3(2):324–328, Mar 1992.

[48] E.D. Goodman L.A. Kuhn M.L. Raymer, W.F. Punch and A.K. Jain. Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(2):164–171, Jul 2000.

[49] M. Kabrisky D.W. Ruck, S.K. Rogers. Feature selection using a multilayered perceptron. *Journal of Neural Network Computation*, 2(2):40–48, 1990.

[50] R. Setiono and Huan Liu. Neural network feature selector. *IEEE Transactions on Neural Networks*, 8(3):654–662, May 1997.

[51] J.M. Steppe Jr. Integrated feature and architecture selection. *IEEE Transactions on Neural Networks*, 7(4):1007–1014, Jul 1996.

[52] A. Malinowski J.M. Zurada and S. Usui. Perturbation method for detecting redundant inputs of perceptron networks. *Neurocomputing*, 14:177–193, 1997.

[53] N.R. Pal R. De and S.K. Pal. Feature analysis: neural network and fuzzy set theoretic approaches. *Pattern Recognition*, 30(10):1579–1590, 1997.

[54] B. P. F. Lelieveldt M. R. Rezaee, B. Goedhart and J.H.C. Reiber. Fuzzy feature selection. *Pattern Recognition*, 32(12):2011–2019, 1999.

[55] D. Citterio M. Hagiwara K. Suzuki Y. Araki, T. Ohki. A new method for inverting feedforward neural networks. In *IEEE International Conference on Systems, Man and Cybernetics, 2003*, volume 2, pages 1612–1617. 5-8 Oct 2003.

[56] M. Marseguerra and A. Zoia. The autoassociative neural network in signal analysis: I. the data dimensionality reduction and its geometric interpretation. *Annals of Nuclear Energy*, 32(11):1191–1206, July 2005.

[57] Y. Freund and R. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, september 1999.

[58] D. Chakraborty and N.R. Pal. Expanding the training set for better generalization in mlp. In *Proceedings of International Conference on Communication, Devices and Intelligent Systems, CODIS-2004*, pages 454–457. Jadavpur University, India, 2004.

[59] P. Holmstrom, L. Koistinen. Using additive noise in back-propagation training. *IEEE Transactions on Neural Networks*, 3(1):24 – 38, Jan 1992.

[60] G.N. Karystinos and D. A. Pados. On overfitting, generalization, and randomly expanded training sets. *IEEE Transactions on Neural Networks*, 11(5):1050–1057, Sep 2000.

[61] A. Asuncion and D.J. Newman. *UCI Machine Learning Repository*. Irvine, CA: University of California, Department of Information and Computer Science, http://mlearn.ics.uci.edu/MLRepository.html.

[62] http://www.nersc.gov/ $\sim$ cding/protein.

[63] L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

[64] B. Efron and R.J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.

[65] Enrico Zio. A study of the bootstrap method for estimating the accuracy of artificial neural networks in predicting nuclear transient processes. *IEEE Transactions on Nuclear Science*, 53(3):1460–1478, June 2006.

[66] J. H. Friedman and B. E. Popescu. Importance sampled learning ensembles. Technical report, Stanford University, Department of Statistics, 2003.

[67] J. Wu Z.-H. Zhou and W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1-2):239–263, 2002.

[68] Sang-Hoon Oh and Youngiik Lee. Sensitivity analysis of single hidden-layer neural networks with threshold functions. *IEEE Transactions on Neural Networks*, 6(4):1005–1007, July 1995.

[69] P. M. Granitto, P. F. Verdes, and H. A. Ceccatto. Neural network ensembles: evaluation of aggregation algorithms. *Artificial Intelligence*, 163(2):139–162, April 2005.

[70] Z.S.H. Chan and N. Kasabov. Fast neural network ensemble learning via negative-correlation data correction. *IEEE Transactions on Neural Networks*, 16(6):1707–1710, 2005.