UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA CIVIL ELÉCTRICA

ESTUDIO DE LA UTILIZACIÓN DEL POTENCIAL DE INFORMACIÓN CRUZADO EN EL APRENDIZAJE CON ENSAMBLE DE REDES NEURONALES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

PABLO ANTONIO SAAVEDRA DOREN

PROFESOR GUÍA:
SR. PABLO ESTÉVEZ VALENCIA

MIEMBROS DE LA COMISIÓN:
SR. PABLO HUIJSE HEISE
SR. PABLO ZEGERS FERNÁNDEZ

SANTIAGO DE CHILE
2017

RESUMEN DE LA MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL ELÉCTRICO
POR: PABLO ANTONIO SAAVEDRA DOREN
FECHA: 11/09/2017
PROF. GUÍA: DR. PABLO ESTÉVEZ VALENCIA

ESTUDIO DE LA UTILIZACIÓN DEL POTENCIAL DE INFORMACIÓN CRUZADO EN EL APRENDIZAJE CON ENSAMBLE DE REDES NEURONALES

El propósito del presente trabajo es estudiar y proponer un método de aprendizaje para los Ensambles de Redes Neuronales basados en la maximización de la Información Mutua Cuadrática entre las salidas de los modelos que componen el Ensamble. En esencia el método propuesto es una función de costo que incluye un término de regularización basado en Información Mutua que se estima a partir del Potencial de Información Cruzado o CIP (Cross Information Potential), además el término de regularización busca favorecer la diversidad entre los modelos del Ensamble. Al método propuesto se le identifica en este trabajo como CIPL (Cross Information Potential Learning).

La hipótesis de trabajo es que la utilización de herramientas de Teoría de la Información en la definición de la función de costo de CIPL pueden ayudar a mejorar la precisión y la diversidad del Ensamble comparado con el método basado en correlación negativa propuesto por el método NCL (Negative Correlation Learning) además de ayudar a favorecer más aun la diversidad.

La metodología de trabajo incluye primeramente la implementación de una librería desarrollada en el lenguaje de programación Python para poder entrenar modelos de redes neuronales en forma paralela con el fin de poder probar el método de entrenamiento NCL y CIPL. Para evaluar el método de entrenamiento CIPL se realizan pruebas sobre problemas de regresión y clasificación típicos, parte de estas pruebas intentan determinar su comportamiento bajo condiciones de ruido y valores atípicos. Para el caso de CIPL se agregan pruebas sobre los diferentes hiperparámetros que tiene.

Los resultados obtenidos muestran que CIPL tiene un desempeño similar que NCL en problemas de clasificación, no así en regresión donde NCL es mucho mejor. En cuanto a los hiperparámetros de CIPL se destaca que la sinergia y la redundancia influyen directamente en la diversidad del Ensamble, incluso permiten obtener mejores niveles de diversidad que NCL.

La implementación de CIPL tiene problemas con los tiempos de entrenamiento que aumentan de forma exponencial con la cantidad de muestras y de modelos del Ensamble, por lo que requiere una optimización del código. Por otro lado, aunque la diversidad en el caso de CIPL mejora los resultados, no es posible cuantificar este efecto, por tanto se deja propuesto para trabajos futuros. Además, falta resolver problemas que tiene la implementación de CIPL cuando se trabaja con más de 2 clases.

Agradecimientos

Me es imposible al momento de pensar a quien agradecer y no considerar a mi madre en primer lugar, no solo por su rol como tal sino también como un ejemplo de cómo vivir con alegría a pesar de los problemas y dificultades que presenta la vida, por tanto, dedico este trabajo a Mónica del Carmen Doren Valdés mi madre.

Pablo Doren

Tabla de contenido

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

El Aprendizaje de Maquinas o Machine Learning en inglés es un área de Ciencias de la Computación que estudia la forma en como proveer la capacidad de aprender a las maquinas o dicho de otra forma "darles a los computadores la habilidad de aprender sin ser explícitamente programadas" (Arthur Samuel, 1959). El Aprendizaje de Maquinas ha suscitado gran interés desde hace bastante tiempo partiendo por los trabajos de Alan Turing en donde en uno de ellos titulado "Computing Machinery and Intelligence" (1950) plantea la posibilidad de que una maquina pueda simular la mente humana adulta a partir de la de un niño además en este mismo trabajo explica la dificultad de determinar sobre si las maquinas pueden pensar y reformular la pregunta por ¿las maquinas pueden hacer que nosotros pensemos que pueden pensar? [4] lo anterior significó un cambio de paradigma importante. Aunque desde mucho antes de la proposición de Turing ha existido el interés por recrear la inteligencia humana en máquinas no es sino hasta décadas posteriores que se han planteado los objetivos más ambiciosos al respecto. El término "Machine Learning" fue utilizado por primera vez el año 1980 en un paper que fue publicado como "Machine Learning: An Artificial Intelligence Approach" [5, prefacio] y desde entonces se ha utilizado como nombre identificador de las técnicas y algoritmos que permiten "aprender" a partir de un conjunto de datos y con el objetivo de generar predicciones o proponer decisiones respecto a los mismos.

Según el destacado profesor Tom M. Mitchell define el Aprendizaje de Maquinas en forma general de la siguiente forma: "se dice que un programa aprende de una experiencia E con respecto una tarea de tipo T y una métrica de desempeño P si su desempeño en la tarea T, medido por P, mejora con la experiencia E." [6, p. 2]. Las Maquinas de Aprendizaje son modelos diseñados para llevar a cabo una tarea T los cuales están parametrizados y por tanto el proceso de entrenamiento o aprendizaje puede ser visto como un problema de optimización sobre los parámetros del modelo usando como función objetivo la métrica P además los datos para el entrenamiento son obtenidos desde la experiencia E.

En este informe se habla de Maquina de Aprendizaje o Modelo de forma indistinta, ambas expresiones hacen alusión a cualquier estructura lógica o matemática que permita llevar a cabo una tarea particular como por ejemplo clasificación, identificación de patrones, regresión de funciones, extracción de características, etc. Es importante destacar que los modelos simplifican elementos de la realidad y por tanto solo constituyen una aproximación

de la misma. Sin embargo, para fines prácticos esta abstracción de la realidad es suficiente para resolver las tareas propuestas, en el capítulo 2 se profundiza en el concepto de modelo y los diferentes tipos de entrenamientos existentes.

El Aprendizaje de Maquinas ha tenido un gran impulso en estas últimas décadas debido en primera instancia a la necesidad creciente de procesar grandes volúmenes de datos en tiempos cada vez más cortos de forma que la capacidad humana se ha visto sobrepasada y por tanto se ha hecho necesario la introducción de técnicas que automaticen su procesamiento. Lo anterior junto con el aumento de la capacidad de computo de los aparatos electrónicos ha dado paso a un gran desarrollo de métodos de procesamiento automático de la información, en especial a partir de la primera década de este siglo se han desarrollado avances importantes en problemas de aprendizaje de máquinas relacionados con el procesamiento de imágenes, vídeo y sonido utilizando modelos cada vez más complejos (por ejemplo el Aprendizaje Profundo) que, aunque muchos de aquellos fueron propuestos hace décadas atrás, no es sino hasta estos días en donde se han podido trabajar efectivamente con estos.

Bajo un contexto de avances en el área del Aprendizaje de Maquinas este trabajo de Memoria intenta aportar en la linea del conocimiento acerca del Aprendizaje con Ensambles. Un Ensamble es una máquina de aprendizaje que se forma por la combinación de múltiples modelos con el fin generar un meta modelo que es mejor que el desempeño de los modelos individuales. Los Ensambles han demostrado ser una herramienta importante para mejorar los resultados de predicción de los modelos que lo constituyen y por tanto tiene especial interés en problemas en donde los modelos simples por si solos no pueden mejorar sus resultados.

El aporte de este trabajo es la propuesta de un método de entrenamiento de Ensambles que utiliza herramientas de Aprendizaje con Teoría de la Información o ITL del inglés Information Theoretical Learning. Específicamente se propone utilizar la herramienta del Potencial de Información Cruzado o CIP (Cross Information Potential) como un elemento base de la función objetivo del entrenamiento de un Ensamble. Sin entrar en mayores detalles CIP entrega una estimación de la Información Mutua que relaciona a más de una fuente de información [7], en el capítulo 2 se explica con mayor profundidad que es CIP.

Aunque en los Ensambles los modelos que lo componen pueden ser heterogéneos, es decir modelos de diferentes tipos y arquitecturas, este trabajo se centra en las Redes Neuronales como modelos base, aunque el método propuesto de entrenamiento también puede ser usado con otro tipo de modelos.

Para el análisis del método propuesto y determinar su desempeño se han dispuesto una serie de pruebas enfocadas en el paradigma de Aprendizaje Supervisado y especialmente en problemas de clasificación y regresión, de forma de simplificar el análisis.

1.1. Motivación

Si existiera una máquina de aprendizaje perfecta los Ensambles no tendrían sentido, pero dado que no existe una máquina que resuelva todos los problemas sin errores es necesario generar diferentes métodos de aprendizaje que permitan mejorar el desempeño de los modelos. Los Ensambles son una herramienta que permite combinar diferentes modelos para generar un modelo más robusto y de mejor desempeño que los modelos individuales que lo componen.

El simple hecho de combinar diferentes modelos no genera mejores resultados, es necesario cuidar que las contribuciones de los modelos individuales sea sinérgica evitando que las salidas de los mismos sean homogéneas, de lo contrario la contribución total de los modelos sería la misma que la entregada por solo un modelo lo que implica que el Ensamble no genera una mejora y por tanto no tendría sentido su utilización.

Evitar que los modelos sean homogéneos no es una tarea sencilla principalmente debido a que los modelos intentan resolver el mismo problema por tanto la correlación de las salidas de los modelos tiende a ser alta. Por tal motivo es necesario generar alguna estrategia que permita generar cierta diferenciación entre los modelos pero considerando que a su vez estos coincidan en varias predicciones. Al concepto intuitivo de diferenciación entre modelos se le conoce como Diversidad y es lo que se busca al momento de entrenar un Ensamble.

La Diversidad es un concepto que todavía no tiene una definición concreta pero es posible entender la intuición detrás de esta [8, 9]. La Diversidad se puede lograr de diferentes formas una de ellas es a través de la exploración del espacio transversal de hipótesis. Lo anterior se puede lograr a través de la inclusión de un término de regularización en la función objetivo de entrenamiento, la que considere la información de los otros modelos del Ensamble.

Un ejemplo de cómo favorecer la diversidad a través de un término de regularización es el método de entrenamiento de Ensambles llamado *Negative Correlation Learning* o NCL por sus siglas en inglés, el cual utiliza la correlación negativa entre las salidas de los modelos y el Ensamble [10].

Por otro lado, el Potencial de Información Cruzado es una medida que permite estimar la entropía cuadrática de Renyi entre variables aleatorias de diferentes dimensiones entre sí. Además, La entropía cuadrática de Renyi es una medida de información que puede ser usada como medida de similitud. Debido al uso de mayor información estadística, el Potencial de Información Cruzado es más robusto al ruido y a la presencia de valores atípicos (outliers) al compararlo con la correlación.

La motivación de este trabajo es estudiar como el Aprendizaje de Ensambles y el Potencial de Información Cruzado pueden ser mezclados para generar un término de regularización del tipo NCL con el objetivo de ayudar a mejorar la diversidad, la precisión y la generalización de un Ensamble.

1.2. Estado del Arte

El Aprendizaje con Ensambles tiene como fin combinar diferentes modelos para generar un meta-modelo que mejore los resultados individuales. Para ello existen diferentes métodos que se diferencian en como son entrenados los modelos individuales y como se genera la salida final del Ensamble. Para todos los métodos de Ensambles se busca que las contribuciones de los modelos sea heterogénea para que la contribución total de los modelos mejore en forma efectiva al ser combinados en el Ensamble, a esta propiedad intuitiva se le conoce como diversidad la cual no tiene una definición formal [8, 9].

Aunque no existe una regla general que explique que la diversidad mejora la precisión de los Ensambles [8] existen problemas particulares en los cuales es posible probar esta relación. En problemas de regresión, la Descomposición Ambigua [11] permite relacionar la diversidad y la reducción del error cuadrático de predicción de un Ensamble, por otro lado el dilema sesgo-varianza permite entender porqué la diversidad favorece la precisión de los Ensambles [2, p. 102] [12].

En problemas de clasificación no es tan fácil como en el caso de la regresión favorecer la diversidad principalmente debido a que no existe una única forma de cuantificar el error de predicción y por tanto no existe una medida intrínseca para cuantificar el error como en el caso de la regresión [9, p. 30]. Aunque existen trabajos que intentan aplicar los mismos conceptos de diversidad y regresión a problemas de clasificación [13, 14]. Otros estudios intentan relacionar la diversidad en clasificadores a partir de herramientas estadísticas considerando que las salidas de los modelos entregan información de las distribuciones de las muestras [15, 16].

Para favorecer la diversidad existen métodos de Ensambles que utilizan la manipulación de los datos, como por ejemplo los métodos de entrenamiento de Bagging y Boosting [17, 18]. En otros casos se agregan datos artificiales para generar diversidad [19] o se genera una extracción previa de características a partir de otros modelos como el caso del método Stacking [20]. Otros métodos utilizan la manipulación de la salidas de los modelos [21, 22]. También existen métodos que favorecen la diversidad a partir de la manipulación de cómo se recorre el espacio de hipótesis. Un ejemplo de esto último es NCL el cual a partir de la inclusión de un término de regularización que incluye la correlación negativa entre los modelos y la salida del Ensamble intenta favorecer la diversidad [10].

Con todo lo anterior se puede concluir que la diversidad actualmente es un problema abierto y que depende fuertemente de la arquitectura y los modelos que utiliza el Ensamble. En problemas de regresión resulta más directo determinar la relación entre diversidad y precisión de los modelos respecto a los problemas de clasificación, esto último supone la dirección actual de investigación respecto al Aprendizaje con Ensambles de diversos grupos de investigación relacionados con Sistemas de Múltiples Clasificadores.

1.3. Objetivos

1.3.1. Objetivos Generales

El objetivo general de esta memoria es desarrollar un método de aprendizaje para el entrenamiento de Ensambles con Redes Neuronales basado en conceptos de Teoría de la Información y específicamente en la Información Mutua Cuadrática.

También se busca de forma general estudiar como a partir de herramientas de Aprendizaje con Teoría de la Información o ITL es posible mejorar la diversidad entre los modelos de un Ensamble y como esto afecta a la precisión del mismo. Por otro lado, el estudio se acota al desempeño del método propuesto en problemas de clasificación y regresión, es decir a problema del paradigma de Aprendizaje Supervisado.

1.3.2. Objetivos Específicos

Para el desarrollo de esta Memoria y visto ya los objetivos generales se enumeran a continuación un conjunto de objetivos específicos que dividen de forma sistemática las etapas de desarrollo del diseño y estudio del método de aprendizaje propuesto:

1. Analizar el efecto de la utilización de Ensambles para mejorar los resultados de redes neuronales tipo MLP (Multi Layer Perceptron). A priori los resultados deben mejorar los resultados de una red neuronal por si sola volviéndola más robusta a los valores atípicos (outliers) y ruidos de los datos.

2. Implementar una librería que permita el entrenamiento de Ensambles y que incluya la posibilidad de entrenar de forma paralela los modelos es decir que en el proceso de entrenamiento individual de los modelos del Ensamble se incluya información de los otros modelos bajo la acción de los mismos datos.

3. Diseñar funciones de costos que utilicen información estadística extraída directamente de las muestras de entrenamiento, en particular con métodos de Teoría de la Información.

4. Diseñar un término de regularización que utilice la información estadística extraída directamente de las muestras de entrenamiento. Modelar la diversidad con métodos de Teoría de la Información, esto implica revisar cómo es posible promover que las salidas de los modelos de un Ensamble sean estadísticamente independientes, pero con el compromiso de mantener la precisión de los modelos.

5. Combinar las funciones de costo de regularización en un solo método de entrenamiento, de forma simultáneamente se mejore la precisión y la diversidad.

6. Analizar el desempeño del método propuesto en problemas de clasificación y regresión. Considerar ejemplos de clasificación de 2 a varias clases.

1.4. Hipótesis de Trabajo y Metodología

Este trabajo se sustenta en los buenos resultados del aprendizaje obtenidos con Correlación Negativa (NCL) que a partir de un término de regularización es capaz de favorecer de forma explícita la diversidad en un Ensamble. En NCL se utiliza un término de regularización basado en la correlación negativa para favorecer la diversidad entre los modelos del Ensamble.

La propuesta de este trabajo es utilizar el Potencial de Información Cruzada o Cross Information Potential (CIP) para poder estimar la Información Mutua Cuadrática que finalmente es utilizada como parte del término de regularización. Al igual que la correlación negativa en NCL, CIP puede ser usados como una medida de no similitud entre las salidas de los diferentes modelos de un ensamble.

El nuevo término basado en CIP al considerar en sus cálculos mayor cantidad de momentos estadísticos respecto a la correlación debería mejorar los resultados de predicción respecto al método propuesto en NCL. Por otro lado, CIP tiene propiedades que lo vuelven más robusto al ruido y valores atípicos de los datos, lo cual es una ventaja respecto a NCL, el cual tiene problemas intrínsecos al respecto [23].

Como ya se ha mencionado el estudio del método propuesto se centra en Ensambles de redes neuronales y en problemas de clasificación y regresión pertenecientes al paradigma de Aprendizaje Supervisado. El alcance del método propuesto se centra solo en la proposición de una función objetivo que incluya un término de regularización que mejore los resultados.

Como parte del trabajo, se incluye la implementación de una librería desarrollada en el lenguaje de programación Python que permita entrenar de forma paralela los distintos modelos de un Ensamble, por otro lado, esta librería incluye la posibilidad de utilizar aceleración por Hardware (GPU), de forma de poder entrenar desde redes neuronales simples a redes más complejas como las de Aprendizaje Profundo (Deep Learning).

1.5. Estructura de la Memoria

El presente informe de Memoria presenta una estructura secuencial partiendo desde los conceptos más básicos de Redes Neuronales, Aprendizaje con Ensambles y Teoría de la Información hasta llegar a plantear una propuesta de entrenamiento de Ensambles Neuronales con CIP. Cada capítulo incluye una pequeña introducción para informar de los objetivos del mismo además de ubicar al lector en el contexto del trabajo.

El Capítulo 2 presenta el marco teórico bajo el cual se sustenta este trabajo, en este se pretende entregar los conceptos básicos para entender el contexto de este trabajo. En este capítulo se explican conceptos claves: Redes Neuronales Artificiales, Aprendizaje con Ensambles, Diversidad, Teoría de la Información y CIP.

En el Capítulo 3 se presenta la propuesta de aprendizaje de Ensambles Neuronales utilizando CIP, además incluye los fundamentos teóricos. En este mismo capítulo se muestran los detalles de la implementación del método con una explicación detallada del código generado.

En el Capítulo 4 se presentan los resultados experimentales del método propuesto, en este mismo capítulo se incluye un análisis de los datos obtenidos de forma de poder concluir si la hipótesis de trabajo es correcta. En las pruebas realizadas se exponen y discuten los resultados obtenidos y se compara con otros métodos de Aprendizaje.

Finalmente, el Capítulo 5 se presentan las conclusiones del trabajo considerando los resultados de la propuesta de entrenamiento de Ensambles de Redes Neuronales. También se presenta una discusión acerca de los resultados y de los alcances del método. Finalmente, se entregan algunas recomendaciones para quienes deseen continuar el trabajo.

Capítulo 2

Marco Teórico

Este capítulo presenta las bases teóricas de los conceptos fundamentales y útiles para este trabajo, centradas fundamentalmente en el Aprendizaje de Máquinas y Ensambles de Redes Neuronales, Diversidad y Teoría de la Información aplicada al Aprendizaje de Maquinas. El objetivo de este capítulo es entregar al lector las herramientas suficientes para entender el contexto y los fundamentos matemáticos que dan sustento a la propuesta de Entrenamiento de Ensambles.

En primer lugar, se presenta una pequeña reseña del Aprendizaje de Maquinas con el fin de introducir los conceptos fundamentales. Luego se presenta una descripción general de los modelos de Redes Neuronales donde se incluye información acerca de sus fundamentos, estructuras y características. En esta parte también se incluye un repaso general al aprendizaje bajo el paradigma supervisado, así como información acerca de conceptos como generalización, sobreajuste, regularización, adaptación del factor de aprendizaje, etc.

En segundo lugar, se presenta el Aprendizaje con Ensambles donde se explica su motivación, funcionamiento y como se relaciona con las Redes Neuronales. Además, se incluye una sección para presentar y explicar el concepto de Diversidad y como esta se relaciona con el aprendizaje con Ensambles.

Finalmente se presenta el Aprendizaje con Teoría de la Información o ITL de sus siglas en inglés (Information Theorical Learning). En esta parte se presenta el fundamento teórico del uso de la Teoría de la Información para la resolución de problemas de Aprendizaje de Maquinas, además se pone especial interés en presentar el Potencial de Información Cruzado o CIP (Cross Information Potential) como una herramienta de ITL que permite estimar la información mutua entre un conjunto de variables aleatorias.

2.1. Paradigmas de Aprendizaje de Maquinas

El aprendizaje de máquinas busca generar modelos de la realidad a partir de datos de la misma, estos modelos permiten generar predicciones o extraer información del medio. Los modelos pueden ser de tipo probabilístico o determinista, estáticos o dinámicos, paramétricos o no-paramétricos, etc. Todos ellos necesitan ser entrenados para poder utilizados, este aprendizaje depende de los datos con los que se cuenten y los objetivos del modelo. Los principales paradigmas de aprendizaje son: Supervisado, No Supervisado y Reforzado, aunque también existen otros tipos sin embargo son combinaciones completas o parciales de estos tipos principales.

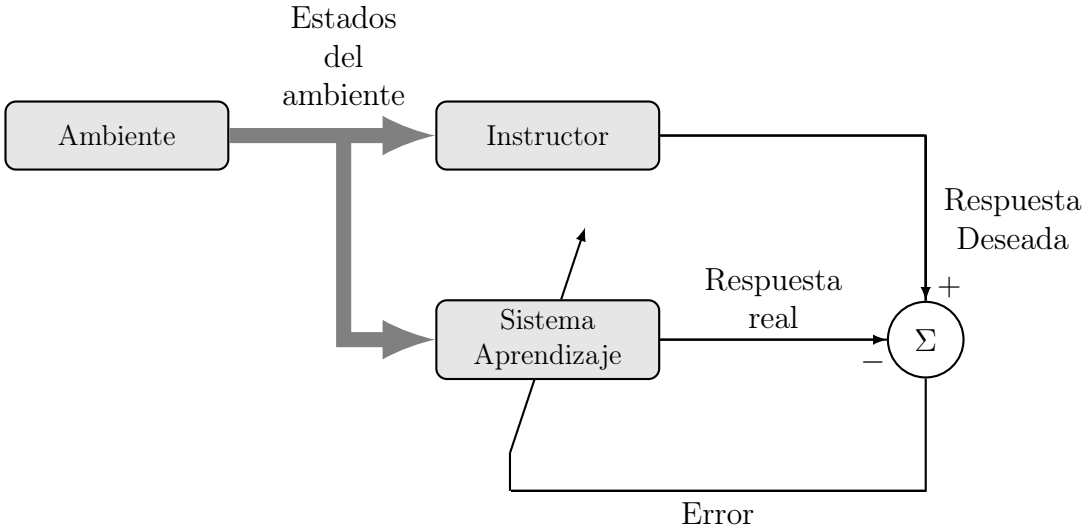


Figura 2.1: Diagrama de bloques del Aprendizaje Supervisado.

El Aprendizaje Supervisado (ver figura 2.1) permite generar un modelo que en términos matemáticos representa un mapa $f(\cdot)$ que relaciona los datos de entrada $\mathcal{X}$ y salida $\mathcal{Y}$ (ver ecuación (2.1)). Se habla de mapa y no de función debido a que es muy común que los elementos del conjunto de salida o imagen $\mathcal{Y}$ tengan más de un elemento en la pre-imagen $\mathcal{X}$, un caso típico son los modelos para problemas de clasificación donde los datos de entrada o características pueden ser de una alta dimensión mientras que las clases son solo un conjunto discreto pequeño.

$$\begin{aligned} f :& \mathcal{X} \to \mathcal{Y} \\ & x \mapsto y = f(x) \end{aligned} \tag{2.1}$$

Los problemas de Aprendizaje Supervisado deben su nombre a que durante el entrenamiento se supervisa que la salida del modelo entrenado sea similar a la salida deseada. La acción de supervisar implica corrección del modelo según objetivos del aprendizaje, esto puede ser por ejemplo disminuir el error a través de la minimización de una función objetivo. Lo importante de este paradigma es contar con ejemplos que sirven como información a priori de los datos de entrada, estos ejemplos pueden ser datos numéricos en el caso de problemas de regresión o etiquetas de clase en el caso de problemas de clasificación. Estos ejemplos se utilizan durante el entrenamiento para ajustar los parámetros internos del modelo.

Figura 2.2: Diagrama de bloques del Aprendizaje No Supervisado.

El segundo paradigma de aprendizaje es el No Supervisado (ver figura 2.2), el cual a diferencia del anterior no necesita de una acción supervisora durante el proceso de entrenamiento debido a que o no existe información a priori de los datos. En este paradigma se intentan buscar relaciones existentes en los datos de entrada. Los modelos o problemas típicos de este paradigma son de agrupación o *clustering* en inglés, es decir encontrar conjuntos de clases existentes en los datos. También es posible encontrar desarrollos relacionados con compresión o visualización de datos.

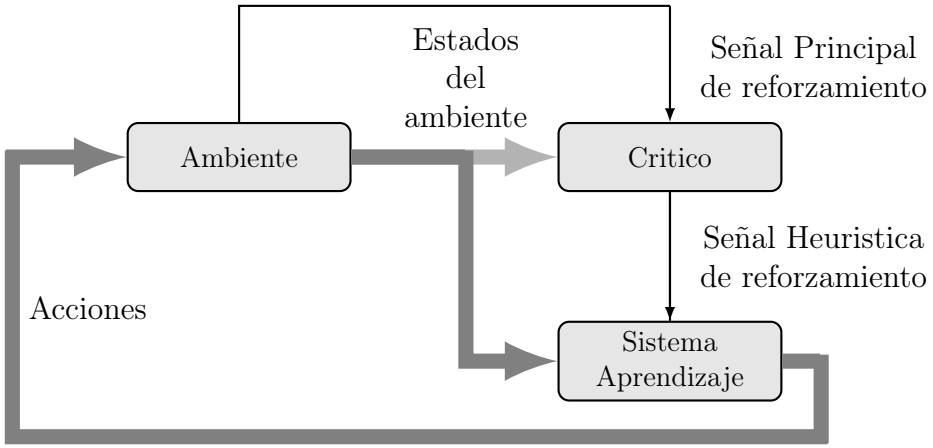


Figura 2.3: Diagrama de bloques del Aprendizaje Reforzado.

El tercer paradigma es el de Aprendizaje Reforzado (ver figura 2.3) que en principio es similar al Aprendizaje Supervisado pero que en vez de tener pares de entrada y salida como ejemplos o información a priori, este paradigma tiene una función de recompensa que permite obtener información y generar correcciones sobre el modelo. La función de recompensa premia cuando el modelo ha realizado una tarea correctamente y castiga en caso contrario. El objetivo del entrenamiento es maximizar la recompensa acumulada. En este paradigma se identifica a la fuente de información de entrada como estados del ambiente, además el modelo como un agente crítico que recibe estos estados dependiendo del entorno mismo y del conjunto de acciones que se lleven a cabo por parte del agente. En algunos casos el agente no cuenta con toda la información de entorno y por otro lado el conjunto de acciones se puede restringir. En los problemas típicos el medio ambiente se formula como un proceso de Markov y lo que se busca es una política que permita tomar el mejor curso de acción dependiendo del estado actual y la función de recompensa futura. Un aspecto importante de este aprendizaje es la exploración debido a que solo existe un conocimiento local del rendimiento y por tanto es necesario indagar en el espacio del entorno para encontrar una solución.

2.2. Redes Neuronales Artificiales

2.2.1. Fundamentos de las redes neuronales artificiales

El cerebro humano suscita gran interés en muchas áreas de la ciencia debido a su capacidad de procesar gran cantidad de información en fracciones de segundo junto con su capacidad de aprender. Por años los científicos han estudiado y desarrollado modelos que puedan ayudar a entender y replicar las capacidades del cerebro, parte de esos estudios han llevado a generar modelos simplificados partiendo como base de la unidad básica de procesamiento del cerebro: la neurona, a este respecto es importante destacar que el cerebro humano cuenta con aproximadamente 86.1×10^9 neuronas [24] y a su vez cada neurona tiene cientos de conexiones, lo anterior ejemplifica la tarea titánica que significa entender una red completa como el cerebro, por tanto los primeros estudios se basan en solo una neurona pero no por ello menos importante. McCulloch y Pitts en 1943 plantearon el modelo de la neurona como una maquina binaria con varias entradas y salidas [25], más tarde Hebb basándose en investigaciones psicofisiológicas definió en 1949 el primer acercamiento a como se genera el aprendizaje a nivel neuronal, planteando que este se localiza en las sinapsis o conexiones entre las neuronas, además de que la información se representa en el cerebro mediante un conjunto de neuronas activas o inactivas [26], esta forma de aprendizaje queda sintetizada en la regla de Hebb: el valor de una conexión sináptica se incrementa si las neuronas de ambos lados de dicha sinapsis se activan repetidas veces de forma simultánea. La Regla de Hebb se puede expresar en términos de neuronas artificiales como:

$$\Delta w_i = \eta x_i y, \tag{2.2}$$

donde η es el factor de aprendizaje, x_i es la entrada i-ésima, y es la salida deseada y Δw_i es la razón de cambio de los pesos sinápticos. La ecuación (2.2) es conocida como Regla de Hebb generalizada y permite actualizar los pesos sinápticos de la forma siguiente: $w_i(k) = w_i(k-1) + \Delta w_i(k)$. En la ecuación η es el factor de aprendizaje, $w_{i\in\{1,\dots,N\}}$ son los pesos sinápticos, $y = \sum_j w_j x_j$ es la salida de la red para el caso de una neurona lineal, más adelante se verá en mayor detalle el significado de cada uno de estos términos.

Luego del primer modelo neuronal tuvieron que pasar varios años para que se comenzara a trabajar en mayor profundidad en el tema de aprendizaje y aplicaciones. En 1956 la universidad de Dartmouth College, ubicada en Hanover, Nuevo Hampshire (Estados Unidos), se desarrolló la primera conferencia sobre Inteligencia Artificial en la cual se abordó el tema de la capacidad de las computadoras para simular el aprendizaje [27], luego comenzaron varios estudios de investigación para formalizar el modelo neuronal junto con estudiar sus capacidades [26].

Otro hito importante y fundamental es el que ocurrió en 1957 con el desarrollo del Perceptrón por el Frank Rosemblatt en el Laboratorio Aeronáutico de la Universidad de Cornell. El primer prototipo fue desarrollado mediante una computadora IBM 704 en 1957. El Perceptrón constituye el modelo fundamental de las redes neuronales actuales.

Luego pasaron algunos años de cierta inactividad en cuanto a adelantos en el estudio de redes neuronales debido a las limitaciones del Perceptrón puestas de manifiesto por el

trabajo de Minsky y Papert. En 1986 Rumelhart y McClelland ayudan a retomar el interés con su trabajo de Procesamiento Distribuido Paralelo (*Parallel Distributed Processing*) [28] y el desarrollo del algoritmo de Retropropagación o *Backpropagation*, lo que permitió el desarrollo de las redes de Perceptrón multicapa.

Actualmente las redes neuronales son utilizadas para abordar problemas relacionados con reconocimiento de patrones, procesamiento de imágenes, vídeos y voz, control automático, predicción de series, etc. en general todo tipo de aplicaciones que necesiten el aprendizaje a partir de datos.

2.2.2. Redes Neuronales Artificiales y Biológicas

La neurona biológica es una célula especializada con múltiples canales de entrada llamados dendritas y un canal de salida llamado axón (ver figura 2.4). Las neuronas se relacionan entre sí mediante las sinapsis, que permiten que una célula influya en la actividad de otras, proceso que involucra variaciones de concentraciones de sustancias químicas de iones de sodio y potasio entre otros, esto último modifica el potencial eléctrico de la célula con respecto al medio de forma que los estímulos que comunican son de tipo electro-químico.

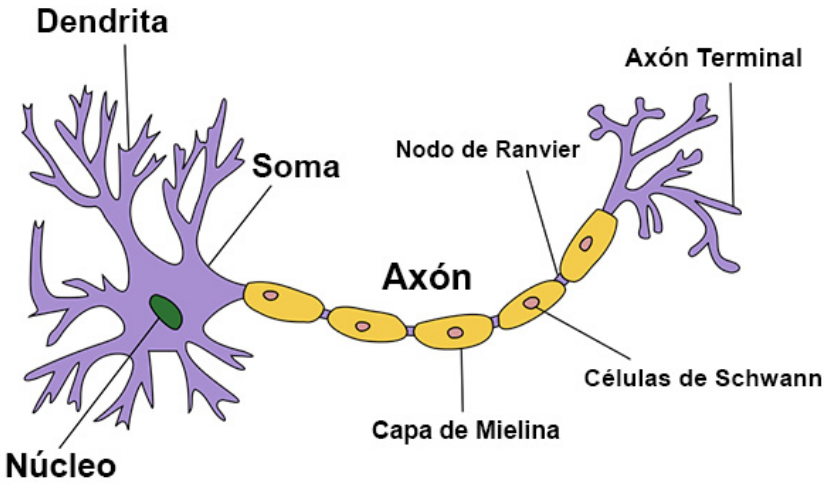


Figura 2.4: Ejemplo de una neurona (*Fuente imagen: Wikimedia Commons*).

El cerebro humano cuenta con miles de millones de neuronas y a su vez ciento de billones de conexiones entre ellas [29], además se organizan dependiendo de la especialización de los estímulos que reciben y su función. Una gran cantidad de estas neuronas son redundantes lo que permite que en el caso de daños otras puedan tomar el lugar de esta forma si el cerebro pierde parte de ellas las funciones que desempeñaban no terminen, esta característica se le conoce como Plasticidad Neuronal. Esta capacidad permite incluso la reorganización sináptica y la posibilidad de crecimiento de nuevas sinapsis a partir de una neurona o varias neuronas dañadas.

Las redes neuronales artificiales al igual que las redes biológicas son sistemas compuestos por varias unidades básicas interconectadas entre sí, en el caso de las redes neuronales artificiales la unidad básica es el Perceptrón, propuesto por Rosenblatt en 1958 [3]. El Perceptrón

es una abstracción matemática de como funciona una neurona biológica, en la figura 2.5 es posible ver un esquema de su estructura.

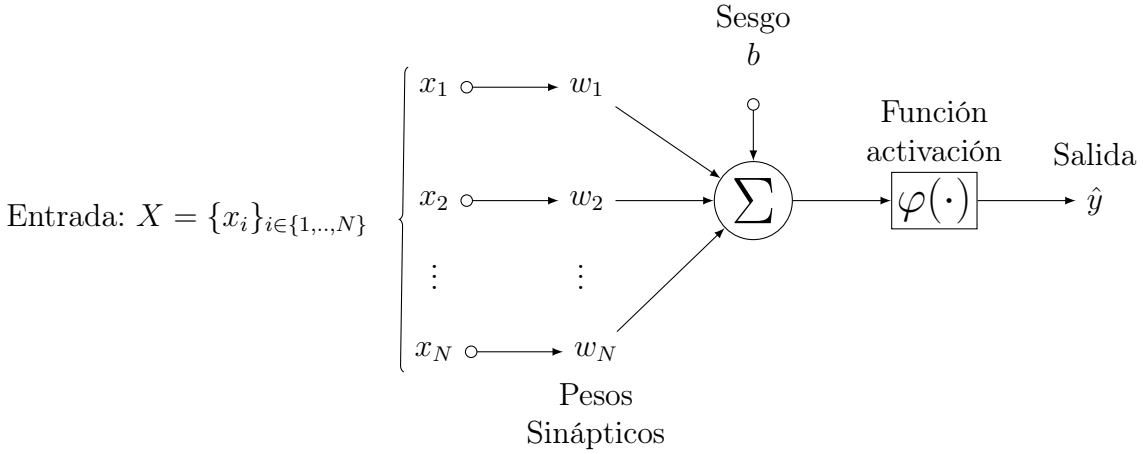


Figura 2.5: Ejemplo de la estructura de un Perceptrón planteado por Rosenblatt en 1958 [3].

El Perceptrón como modelo de una neurona considera al estímulo externo como un vector de entradas $X = \begin{bmatrix} x_1 & x_2 & ... & x_N \end{bmatrix}$ el cual a su vez es ponderado por pesos sinápticos $W = \{w_i\}_{i\in\{1,...,N\}}$ para luego a este resultado aplicarles una función no lineal $\varphi(\cdot)$ llamado comúnmente función de activación. Esto último simula cómo se genera el impulso nervioso en los seres vivos, el que se transmite solo si la integración de entradas supera cierto umbral. Por otro lado, la función de activación es seleccionada con el fin de limitar los valores de salida. En resumen el Perceptrón pondera de forma lineal la entrada o estímulo y aplica una función no lineal al resultado como se puede ver en la ecuación (2.3):

$$\hat{y}(X) = \varphi\left(W \cdot X + b\right) = \varphi\left(\sum_{i}^{N} w_i x_i + b\right), \tag{2.3}$$

se agrega el término constante b como sesgo y tiene el efecto de aumentar o disminuir la entrada final a la función de activación. Para algunos autores el sesgo se incluye dentro los pesos sinápticos por lo que se incluye una entrada aumentada como $X' = [X \ 1] = \begin{bmatrix} x_1 & x_2 & ... & x_N & 1 \end{bmatrix}$ con el objetivo de simplificar la notación. El Perceptrón es un modelo no lineal que para poder ser entrenado requiere modificar convenientemente sus parámetros W y b. Los pesos sinápticos W permiten ponderar cual parte de la entrada tiene mayor relevancia en la salida, un peso $w_i = 0$ implica que esa componente i de la entrada no tiene efecto en la salida.

El Perceptrón dada su estructura solo puede generar comportamientos linealmente separables lo cual fue la gran critica que recibió por parte de Minsky y Papert en 1969, lo anterior limita el conjunto de problemas que puede resolver el Perceptrón. Por lo tanto, si se requiere expandir el conjunto de problemas que se pueden abarcar es necesario reestructurar el esquema de un solo Perceptrón a una red donde cada unidad interactué entre si al igual que una neurona biológica, necesita de más de una para generar comportamientos más complejos.

Las redes neuronales típicas son representadas en la figura 2.6, estas estructuras se basan en un conjunto de Perceptrones dispuestos en *layers* o capas en donde las neuronas de capas

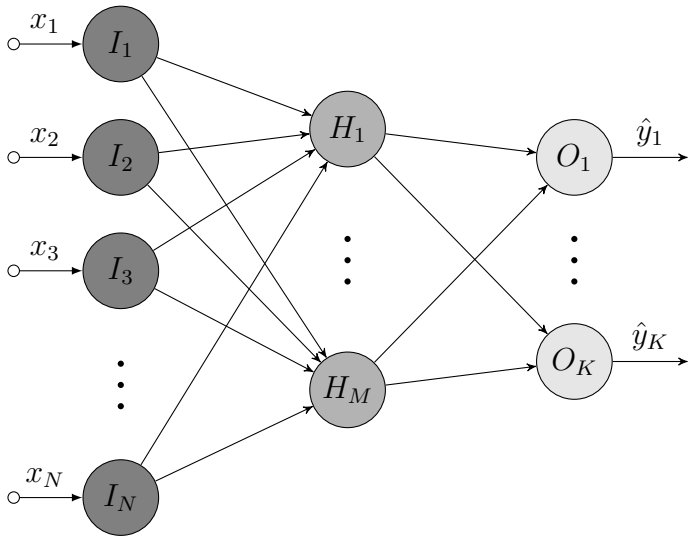


Figura 2.6: Ejemplo de una red neuronal Multi Capa o MLP por sus siglas en inglés, con una entrada $X \in R^N$, una capa oculta con $\{H_m(\cdot)\}_{m\in\{1,..,M\}}$ neuronas y una capa de salida con $\hat{Y} = \{\hat{y}_k\}_{k\in\{1,..,K\}}$ elementos.

contiguas están todas conectadas entre si pero no entre neuronas de una misma capa, a esta estructura se le llama Perceptrón Multi Capa o MLP por sus siglas en inglés (Multi Layer Perceptron). En esta estructura se distinguen al menos 3 capas importantes: la capa de entrada la cual recibe los datos de entrada ($I_{i\in\{1,..,N\}}$), la capa oculta ($H_{i\in\{1,..,M\}}$) la cual recibe las salidas de la capa de entrada y que además puede estar constituida por más de una capa y finalmente la capa de salida ($O_{i\in\{1,..,K\}}$) la cual recibe los datos de las capas ocultas y entrega la salida de la red.

Para ajustar los pesos sinápticos en una red MLP existen diversos algoritmos los cuales se detallan en la sección 2.3.6. Lo que es importante destacar aquí es el hecho que dependiendo del tipo de aprendizaje se lleva a cabo un algoritmo de entrenamiento. El algoritmo más conocido de entrenamiento de redes neuronales MLP es error *Backpropagation* (ver sección 2.3.6) el cual permitió renovar el interés de la comunidad científica en la redes neuronales.

2.2.3. Características de las Redes Neuronales Artificiales

Las redes neuronales artificiales cuentan con muchas ventajas debido a su versatilidad y a las características heredadas de su par biológico, una de las habilidades que más resalta es su poder de generalización es decir la capacidad de predecir resultados razonables para ciertas entradas sin haber usado estas últimas en el proceso de entrenamiento o aprendizaje. La generalización es una característica deseada en máquinas de aprendizaje debido a que generalmente no se tiene todos los datos para generar el entrenamiento de forma que para esos datos faltantes uno esperaría que la máquina de aprendizaje o modelo pudiera interpolar o predecir lo más cercano a la realidad.

La generalización no es fácil de conseguir debido a que durante el proceso de aprendizaje o entrenamiento es frecuente el uso de herramientas de optimización que si no se tiene especial

cuidado pueden generar un sobreajuste a los datos de entrenamiento, en particular al ruido presente. Esto implica una menor habilidad de generalización del modelo resultante, aunque existen métodos para evitar que esto suceda, por ejemplo la regularización (ver sección 2.3.4)

Otra característica importante heredada de las redes biológicas es la tolerancia a fallas, esta capacidad está relacionada con la Plasticidad Neuronal. La tolerancia a fallas implica que la red puede seguir operando a pesar de que exista alguna anomalía en ella, aunque es posible que se produzca una degradación del desempeño la red no deja de funcionar. Esta tolerancia a fallas se debe a que dada la estructura de la red el aprendizaje es distribuido, es decir en las neuronas no existen preferencias en cuanto a que datos aprender y por tanto todas intentan durante el entrenamiento ajustar sus pesos sinápticos para resolver todos los ejemplos del conjunto de entrenamiento.

El Perceptron Multi Capa o MLP es considerado como un aproximador universal de funciones, es decir es capaz de aproximar cualquier tipo de función con cualquier tipo de exactitud. Las redes neuronales pueden ser utilizadas tanto para problemas lineales como no lineales.

2.2.4. Tipos de Redes Neuronales

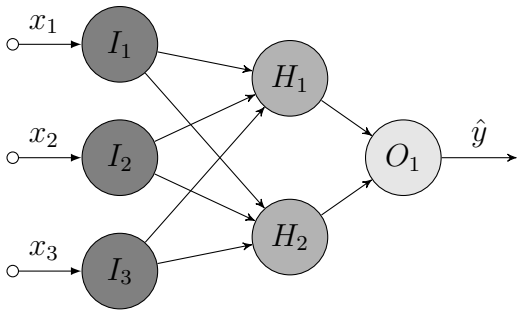
Existe una gran variedad de modelos de redes neuronales con distintos propósitos que se diferencian por su estructura interna y forma de entrenar u obtener sus parámetros. Es posible generar una primera clasificación de las mismas a partir del paradigma de aprendizaje al que pertenecen, es decir existen redes que se utilizan en el aprendizaje supervisado, no supervisado y reforzado como se puede ver en la tabla 2.1. Las redes neuronales del aprendizaje supervisado necesitan de un conjunto de entrenamiento que incluya ejemplos catalogados (es decir, un conjunto donde cada vector de entrada este relacionado con la salida deseada) previamente para poder generar los respectivos modelos, estas redes son principalmente usadas para tareas de regresión y clasificación. Las redes pertenecientes al paradigma de aprendizaje No Supervisado no necesitan que los datos estén catalogados por lo tanto el resultado de estas redes es encontrar similitudes entre los datos, estas redes son utilizadas principalmente para tareas de reconocimiento de patrones y *clustering* de datos.

Un segundo esquema de clasificación podría ser considerar cual es la regla de aprendizaje: corrección de error, competencia o minimización de una distribución de probabilidad (Boltzmann), etc. La corrección del error da cuenta de la minimización del error entre la salida predicha $\hat{y}$ y la salida real y. Si se utiliza una función convexa del error se garantiza la existencia de un mínimo local. La función más utilizada es la del error cuadrático medio o MSE (Mean Square Error):

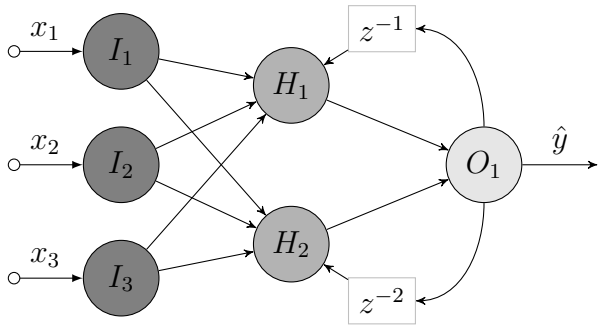
$$MSE = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}(x_i) - y_i)^T(\hat{y}(x_i) - y_i), \tag{2.4}$$

donde x_i y y_i representan el ejemplo i-ésimo de entrenamiento de entrada y salida, respectivamente. Las redes más conocidas que utilizan la corrección de error son el Perceptrón Multi Capa, Elman y Jordan, redes recurrentes, LVQ, etc. La competencia como regla de apren-

dizaje trata de generar durante el entrenamiento competencia entre grupos de neuronas, las redes más conocidas que utilizan esta regla son ART y Mapas de auto organización o Mapas de Kohonen (en inglés es conocido como SOM o Self-organizing map). Los mapas de Kohonen son de especial interés ya que permiten visualizar datos de alta dimensión debido a que al reducir la dimensión de un conjunto de datos los mapas generados conservan las relaciones topológicas. Los Mapas de Kohonen pertenecen al paradigma de aprendizaje no supervisado. Finalmente, las redes más conocidas que minimizan una distribución de probabilidad son las máquinas de Boltzmann, las que definen una función de energía a partir de la distribución de Boltzmann y el objetivo de entrenamiento es que después de pasado un tiempo la energía disminuye y se alcanza un equilibrio térmico, lo anterior es un símil de los fenómenos físicos de un sistema termodinámico.



(a) Feed Forward



(b) Recurrente

Figura 2.7: Ejemplo de una red neuronal tipo (a) feed-forward y (b) recurrente en donde ambas tienen una capa de entrada con 3 neuronas $\{I_1, I_2, I_3\}$, una capa oculta con 2 neuronas $\{H_1, H_2\}$ y una capa de salida con una sola neurona O_1. La diferencia radica en que la red recurrente incluye en la capa oculta entradas pasadas indicadas por los símbolos z^{-1} y z^{-2}.

También es posible clasificar las redes neuronales según la estructura que existe entre los elementos de la misma, estas estructuras se diferencian por la cantidad de capas ocultas que tienen y las conexiones entre ellas. Principalmente es posible diferenciar al menos 2 tipos: redes sin capa oculta y redes multi capa, a su vez es posible clasificarlas por las conexiones que existen entre sus neuronas en: redes pre alimentadas o feedforward en inglés y redes recurrentes (ver figura 2.7), las redes recurrentes permiten modelar datos con dinámica propia es decir que dependen del tiempo en que se tomaron las muestras, mientras que las redes feeforward se pueden considerar como modelos estáticos que dependen de la entrada actual para generar una predicción.

Algoritmos de aprendizaje más conocidos				
Paradigma	**Regla de Aprendizaje**	**Arquitectura**	**Algoritmo de Aprendizaje**	**Tareas**
Supervisado	Corrección de error	Perceptrón Perceptrón Multi Capa	Backpropagation, ADALINE, MADALINE	Clasificación de patrones, Aproximación de funciones, Predicción, control automático, etc.
		Elman y Jordan recurrentes	Backpropagation	Síntesis de series de tiempo
	Boltzmann	Recurrente	Algoritmo de aprendizaje Boltzmann	Clasificación de patrones
	Competitivo	Competitivo	LVQ	Categorización ínter-clases, compresión de datos
		Red ART	ARTMap	Clasificación de patrones, categorización ínter-clases
No supervisado	Relajamiento	Red de Hopfield	Aprendizaje de memoria asociativa	Memoria asociativa
		Multi Capa sin realimentación	Proyección de Sammon	Análisis de datos
	Competitiva	Competitiva	VQ	Categorización, compresión de datos
		SOM	Kohonen SOM	Categorización, análisis de datos
		Redes ART	ART1, ART2	Categorización
	Hebbian	Multi Capa sin realimentación	Análisis lineal de discriminante	Análisis de datos, clasificación de patrones
		Sin realimentación o competitivas	Análisis de componentes principales	Análisis de datos, compresión de datos

Tabla 2.1: Tabla con los algoritmos de aprendizaje más conocidos en el área de redes neuronales (Origen: tabla 2 Tesis Doctoral Juan José Montaño [1]).

2.3. Entrenamiento en Aprendizaje Supervisado

Los métodos de aprendizaje supervisado necesitan primeramente de un conjunto de ejemplos de entrada y salida los cuales representan el comportamiento que se intenta imitar por parte del modelo. Estos ejemplos forman un conjunto de pares de entrada y salida en donde los datos de entrada pueden representar los estados del ambiente (ver figura 2.1), elementos del dominio de una función que se requiera aprender (problemas de regresión) o un vector de características (problemas de clasificación). En síntesis los datos de entrada son los que recibe el modelo, en cuanto a los datos de salida estos representan la salida deseada, por tanto el objetivo de aprendizaje es que la salida del modelo (predicción) sea lo más parecida a la salida deseada.

El proceso de aprendizaje supervisado implica la minimización de una función objetivo o de costos la cual en general utiliza el error entre la predicción y la salida deseada. Por otro lado, también depende de los parámetros del modelo lo que implica que durante el proceso de optimización se minimiza la función de costo modificando los parámetros del modelo.

Los algoritmos más comunes para minimizar una función objetivo utilizan la dirección del gradiente, sin embargo también existen métodos que utilizan derivadas de orden superior como los métodos que utilizan el Hessiano, como por ejemplo BFGS (Broyden-Fletcher-Goldfarb-Shanno) u otros de la familia de métodos Cuasi-Newton. El problema de estos métodos es la utilización de mucha memoria y poder de cómputo para obtener la dirección de menor crecimiento, mientras que los métodos basados en el gradiente solo necesitan trabajar sobre un vector lo que minimiza los costos computacionales. Sin embargo, la desventaja de los métodos basados en el gradiente es que necesitan parámetros adicionales o hiperparámetros como el factor de aprendizaje (ver 2.3.2) lo que en muchas ocasiones si no es bien seleccionado dificulta el proceso de optimización.

Algoritmo 1 Gradiente descendente

Entrada: $epocas_{max}$ el número máximo de épocas de entrenamiento.
θ parámetros de la red.
η el factor de aprendizaje.
$S = (X, y)$ datos de entrenamiento.
$J(\theta, S)$ función de costo.

Salida: Los parámetros θ de la red actualizados.

1: Se inicializan los valores de θ_0
2: **para** $t = 1$ hasta $epocas_{max}$ **hacer**
3: Obtener el gradiente a partir de **todos** los datos de entrenamiento S
4: $\nabla J_t \leftarrow \nabla_{\theta_{t-1}} J(\theta_{t-1}, S)$ {Gradiente actual de la función de costo}
5: Actualizar parámetros de la red
6: $\theta_t = \theta_{t-1} - \eta \cdot \nabla J_t$
7: **fin para**
8: **devolver** θ_t {El último valor calculado}

2.3.1. Gradiente descendente

El algoritmo más popular para entrenar máquinas de aprendizaje es el de Gradiente Descendente o GD (Gradient Descent) este hecho se debe en gran medida a su simplicidad, fácil implementación y bajo costo computacional respecto a otros métodos. En la ecuación (2.5) se puede ver como el método actualiza los parámetros del modelo de aprendizaje:

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta_{t-1}), \tag{2.5}$$

en esta ecuación θ_t representa a los parámetros de la red calculados en la época t. En el método GD el factor de aprendizaje representado por η (ver sección 2.3.2) es importante debido a que requiere especial cuidado al momento de seleccionarlo si se quiere alcanzar la convergencia y evitar problemas de sobreajuste.

En la variante batch *offline* las muestras de ejemplo se presentan todas al mismo tiempo y por tanto la actualización de los parámetros se realiza solo una vez durante cada época, lo anterior se puede ver más claramente en la descripción del algoritmo 1. Este método garantiza la convergencia al mínimo global en el caso de que las superficies de solución generadas en el espacio de los parámetros de la red sean convexos, en el caso contrario solo garantiza convergencia a mínimos locales.

El problema del método batch es que es necesario una gran cantidad de memoria debido a que es necesario calcular el gradiente de todos los datos de entrenamiento, lo anterior lo hace un método lento e intratable en el caso de problemas donde el conjunto de entrenamiento sea tan grande que la memoria del computador sea insuficiente para contener a todos los datos.

Algoritmo 2 Gradiente descendente Estocástica

Entrada: $epocas_{max}$ el número máximo de épocas de entrenamiento.
θ parámetros de la red.
η el factor de aprendizaje.
$S = (X, y)$ datos de entrenamiento.
$J(\theta, S)$ función de costo.

Salida: Los parámetros θ de la red actualizados.

1: Se inicializan los valores de θ_0
2: **para** $t = 1$ hasta $epocas_{max}$ **hacer**
3: Mezclar al azar los elementos de S
4: $S \leftarrow mezclar(S)$
5: **para** $k = 1$ hasta n^o de datos de entrenamiento **hacer**
6: Obtener el gradiente a partir de **un solo dato** de entrenamiento: $S^{(k)}$
7: $\nabla J_t \leftarrow \nabla_{\theta_{t-1}} J(\theta_{t-1}, S^{(k)})$ {Gradiente actual de la función de costo}
8: Actualizar parámetros de la red
9: $\theta_t = \theta_{t-1} - \eta \cdot \nabla J_t$
10: **fin para**
11: **fin para**
12: **devolver** θ_t {El ultimo valor calculado}

Una alternativa al método GD es el de Gradiente Descendente Estocástico o SGD (Stochastic Gradient Descent) que a diferencia del método anterior utiliza un ejemplo a la vez

para actualizar los pesos sinápticos, en la ecuación (2.6) se puede ver como se actualizan los pesos en la red,

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta_{t-1}, S^{(k)}), \tag{2.6}$$

el aprendizaje con SGD puede ser utilizado en el aprendizaje en linea debido a que para realizar la actualización de los pesos solo se necesita un ejemplo. En este método se incluye una etapa que baraja al azar los datos de entrenamiento, ver descripción del algoritmo 2, para evitar presentar los datos en el mismo orden y generar ciclos que repiten la forma de actualización. Este método resulta habitualmente más rápido que GD y no tiene los problemas de memoria de este último, sin embargo el problema del método de SGD son las fluctuaciones sobre el gradiente generadas por las variaciones existentes en los datos de ejemplo. Aquello dificulta lograr alcanzar exactamente el mínimo local, por esa razón en este método es importante manejar correctamente el factor de aprendizaje sobre todo al finalizar las épocas de forma de mejorar los resultados de convergencia, en general se utiliza este método junto con métodos de adaptación del factor de aprendizaje.

Algoritmo 3 Gradiente descendente por mini-lotes

Entrada: $epocas_{max}$ el número máximo de épocas de entrenamiento.
θ parámetros de la red.
η el factor de aprendizaje.
$S = (X, y)$ datos de entrenamiento.
$J(\theta, S)$ función de costo.
n el largo del mini-lote.
Salida: Los parámetros θ de la red actualizados.

Se inicializan los valores de θ_0
para $t = 1$ hasta $epocas_{max}$ **hacer**
  Mezclar al azar los elementos de S
  $S \leftarrow mezclar(S)$
  Obtener los mini lotes de largo n del conjunto S
  $M \leftarrow lotes(S, n)$
  para $k = 1$ hasta n^o de lotes en M **hacer**
    Obtener el gradiente a partir del **mini lote**: $M^{(k)}$
    $\nabla J_t \leftarrow \nabla_{\theta_{t-1}} J(\theta_{t-1}, M^{(k)})$ {Gradiente actual de la función de costo}
    Actualizar parámetros de la red
    $\theta_t = \theta_{t-1} - \eta \cdot \nabla J_i$
  fin para
fin para
devolver θ_t {El ultimo valor calculado}

El método de SGD aunque corrige los problemas de espacio en memoria trae consigo problemas en la convergencia hacia el óptimo, una mejora al respecto es la que entrega el método de SGD con mini-lotes el cual genera pequeños subconjuntos o mini-lotes a partir del conjunto de datos de entrenamiento. Estos mini-lotes permiten actualizar los parámetros del modelo tal como se puede ver en la ecuación (2.7):

$$\theta_t = \theta_{t-1} - \eta \cdot \nabla_\theta J(\theta_{t-1}, S^{(k:k+n)}). \tag{2.7}$$

La ventaja de trabajar con mini-lotes es que permite disminuir la carga de memoria sobre el computador evitando el problema de entrenar con todos los datos de entrenamiento a la

vez como el método GD y además permite disminuir las fluctuaciones sobre la dirección del gradiente debido a la varianza de los datos, problema existente en el método de SGD. El algoritmo de entrenamiento se describe en el algoritmo 3. El método con mini-lotes añade un nuevo hiperparámetro: el número de elementos del lote, en general se utilizan valores entre 32 y 256 dependiendo de la cantidad de datos del conjunto de entrenamiento.

Un punto importante a destacar es que el método de GD es un método determinista mientras que los métodos de SGD con y sin mini-lotes corresponden a métodos estocásticos debido a que incluyen una etapa de mezcla de los datos en cada época para evitar ciclos de actualización y pueden explorar mucho mejor el espacio de parámetros.

2.3.2. Factor de aprendizaje

Para muchos el hiperparámetro más importante en el aprendizaje de máquinas es el factor de aprendizaje debido a que los resultados de convergencia de los métodos de optimización dependen de este factor. Este factor es un número real positivo que permite regular el ajuste de los pesos sinápticos a partir del valor actual del gradiente. Para normalizar los símbolos a partir de aquí en adelante se denotará con el símbolo η para identificar al factor de aprendizaje. En la ecuación (2.5) se puede ver cómo afecta este factor en la actualización de los parámetros de la red.

El factor de aprendizaje es vital para alcanzar la convergencia, en la figura 2.8 es posible ver los efectos de seleccionar diferentes factores de aprendizaje y como estos influyen en la convergencia. En redes MLP no existen reglas generales para seleccionar este parámetro, dado que depende de los datos y la estructura de red seleccionada, pero en general es habitual utilizar valores como: 1, 0.1, 0.01, 0.001, etc.

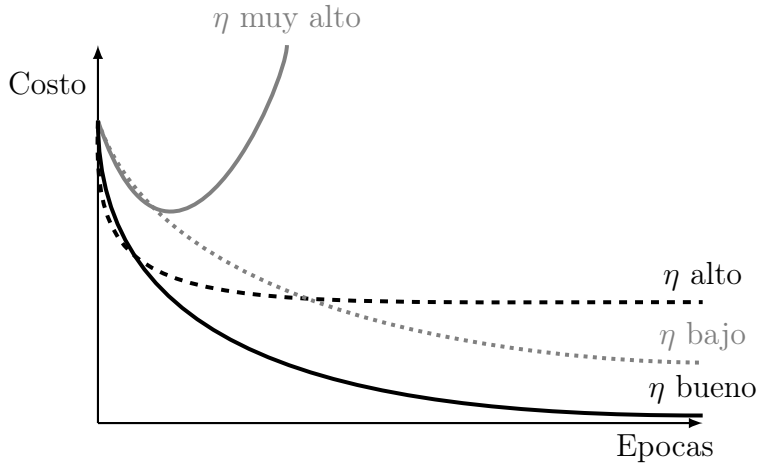


Figura 2.8: Efecto del factor de aprendizaje η en la convergencia al óptimo durante el entrenamiento. El costo es el valor de la función objetivo que se desea optimizar.

Aunque en general el factor de aprendizaje se mantiene constante durante el entrenamiento también existen métodos que permiten ir adaptando su valor a medida que avanzan las épocas. Algunos de estos métodos son funciones que dependen exclusivamente de la épo-

ca, otros utilizan el valor del gradiente como, por ejemplo: Adagrad [30], Adadelta [31] y Adam [32].

2.3.3. Generalización y sobreajuste

Una característica deseada al momento de entrenar un modelo es la generalización. Esta propiedad permite al modelo poder realizar predicciones razonables a partir de datos que no se encuentran originalmente en el conjunto de datos de entrenamiento. La generalización tiene sentido en problemas en donde no se tiene acceso completo al conjunto de todos los datos posibles del dominio del problema. En la práctica casi siempre se trabaja con un subconjunto reducido por lo tanto es necesario que este subconjunto sea representativo del problema para evitar que el modelo tenga un sesgo al momento de generar predicciones.

El sobreajuste se genera cuando un modelo se ajusta demasiado a los datos de entrenamiento, incluyendo ruido, de forma que el modelo pierde generalidad. Para favorecer la generalización y evitar el sobreajuste es importante controlar el Factor de Aprendizaje y la complejidad del modelo, para esto último existen técnicas como por ejemplo la regularización (ver sección2.3.4).

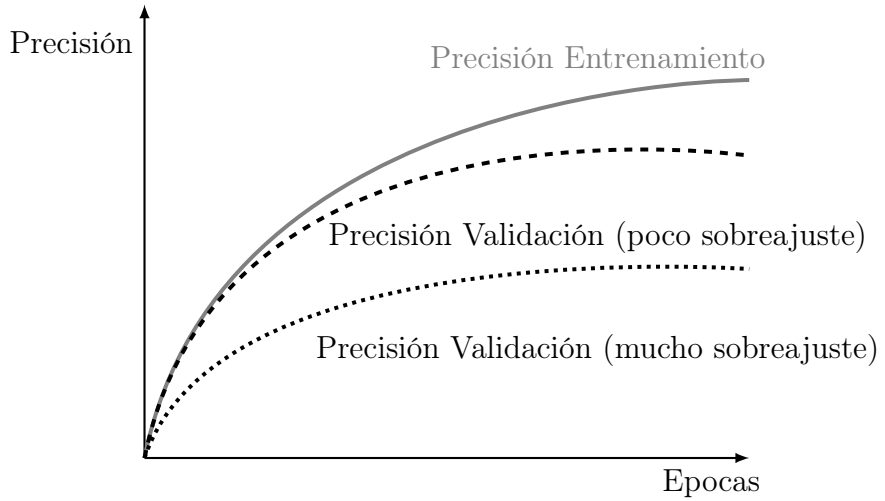


Figura 2.9: Efecto del sobreajuste sobre el conjunto de entrenamiento comparado con el conjunto de validación y una métrica de precisión. En este caso la métrica de precisión indica que mientras más alto sea su valor mayor es la precisión de la predicción.

Para evaluar los resultados del entrenamiento y generalización de un modelo se recurre a una división del conjunto de datos disponibles en 3 subconjuntos diferentes: entrenamiento, validación y prueba. El conjunto de entrenamiento permite realizar los ajustes de los parámetros del modelo. En conjunto de validación permite seleccionar el mejor modelo al comparar los resultados obtenidos con el conjunto de entrenamiento (ver figura 2.9), lo que permite tener información para poder modificar los parámetros del modelo o su estructura. El conjunto de prueba sirve para obtener resultados y métricas de cómo se comportará el modelo con datos reales fuera de los datos de ejemplo. La división de los datos de ejemplo no siempre es posible debido a la cantidad, sin embargo existen técnicas como la validación cruzada que permiten generar resultados sobre una cantidad reducida de datos de ejemplo.

2.3.4. Regularización

Los métodos de regularización introducen un término extra a la función objetivo con el fin de penalizar la complejidad del modelo obtenido. La justificación de aquello se debe a que en general los modelos más simples conservan los aspectos más relevantes de los datos y permiten generalizar de mejor manera [33], el caso contrario es el sobreajuste que implica solo un buen ajuste en los datos de entrenamiento, pero no sobre datos diferentes. La regularización favorece la generalización de los modelos a través de penalizar su complejidad, en general esto se logra generando una función que aplique una norma sobre los parámetros del modelo.

En el caso de las redes neuronales los métodos de regularización más utilizados son los de la norma L1 y L2 que se pueden apreciar en las ecuaciones (2.8) y (2.9) respectivamente,

$$L1(W) = \frac{1}{N_T} \sum_{i=1}^{L} \sum_{j=1}^{L(i)} |\omega_{i,j}| \tag{2.8}$$

$$L2(W) = \frac{1}{2N_T} \sum_{i=1}^{L} \sum_{j=1}^{L(i)} \omega_{i,j}^2. \tag{2.9}$$

donde el término $\omega_{i,j}$ es el peso sináptico de la neurona j en la capa i y W es la matriz que los contiene, L es el número de capas de la red, $L(i)$ es una función que devuelve el número de neuronas en la capa i, N_T es el número total de neuronas en la red. El término de regularización se agrega a la función objetivo como se puede ver en la ecuación (2.10), el parámetro λ_{reg} permite controlar el efecto de la regularización en el resultado de la optimización.

$$\min_{W,b} \sum_{i=1}^{N} J(x_i, y_i, W, b) + \lambda_{reg} L2(W) \tag{2.10}$$

2.3.5. Métodos para mejorar el proceso de optimización

Los métodos de optimización basados en el gradiente adolecen de problemas de convergencia ya sea debido a una mala asignación del factor de aprendizaje, existencia de mínimos locales que entorpecen la búsqueda de un mínimo global o incluso la velocidad de convergencia. A continuación, se presentan una serie de métodos que intentan aliviar estos problemas y que fueron implementados para generar las pruebas.

Momentum

Los métodos basados en Gradiente Descendente tienen dificultad en encontrar la dirección apropiada en superficies donde existan planicies, aquello implica que aumente el tiempo de convergencia, para evitar este problema se utiliza el método del Momentum [34] que tiene la forma siguiente:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta_{t-1}) \\ \theta_t &= \theta_{t-1} - v_t. \end{aligned} \tag{2.11}$$

Este método actualiza los pesos de la red no solo considerando el valor actual del gradiente sino también el valor anterior de actualización ponderado por el Factor de Momento γ. Esta fórmula permite que el valor de actualización tenga una especie de memoria la cual, al igual que el símil de momentum mecánico, acumula energía de forma que al momento de que se pase por un punto de la superficie como una planicie no queda atrapado, debido a la energía o momentum acumulado permite salir pronto de este lugar en las próximas iteraciones. El término γ es un valor real positivo menor a 1, además el término imita una especie de resistencia viscosa proporcional a la velocidad de cambio, aquello permite sintonizar el efecto del momentum sobre la actualización de los pesos, en general se utiliza el valor 0.9, el valor 0 implica que el efecto es nulo y se vuelve a tener una actualización solo por efecto del gradiente. Este método reduce las oscilaciones en la superficie del espacio de parámetros y por tanto reduce el tiempo de convergencia, además permite evitar la convergencia en mínimos locales, cuando es apropiado.

Nesterov (Gradiente acelerada)

Aunque el momentum permite evitar problemas con fluctuaciones en la superficie del espacio de parámetros no siempre es suficiente aquello, es posible que en el próximo punto ya no pueda ejercer ningún tipo de efecto, por lo tanto se necesita un conocimiento previo del punto en cuestión, para lo anterior se utiliza el método de Nesterov [35] el cual genera una predicción de donde podría estar el punto en la iteración siguiente, la forma de actualización de los parámetros del método de Nesterov se presenta en la ecuación (2.12):

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta_{t-1} - \gamma v_{t-1}) \\ \theta_t &= \theta_{t-1} - v_t. \end{aligned} \tag{2.12}$$

La predicción de la próxima corrección se hace a partir el término $\eta \nabla_\theta J(\theta_{t-1} - \gamma v_{t-1})$, el cual ayuda acelerar el tiempo de convergencia y además de evitar las fluctuaciones que podrían llevar a una convergencia en mínimos locales. Este método por lo general significa una mejora por sobre el método de Momentum.

Adagrad

Adagrad es un método de adaptación del factor de aprendizaje propuesto por Duchi et. al. [30], este método modifica o adapta el factor de aprendizaje por cada parámetro y en cada iteración de tiempo, es decir que a diferencia del método de Gradiente Descendente donde el Factor de Aprendizaje es constante e igual para todos los parámetros de la red, Adagrad modifica el factor en cada iteración y en forma diferente para cada parámetro de la red,

$$\begin{aligned} g_{t-1} &= \nabla_\theta J(\theta_{t-1}) \\ G_t &= G_{t-1} + g_{t-1} \odot g_{t-1} \\ \theta_t &= \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_{t-1}, \end{aligned} \tag{2.13}$$

como se puede ver en la ecuación (2.13) para adaptar el factor de aprendizaje se utiliza un factor de aprendizaje inicial η el cual es dividido por el factor $\frac{1}{\sqrt{G_t+\epsilon}}$ donde G_t es la suma

de los cuadrados de cada elemento del gradiente, por esa razón se utiliza el producto $\odot$ de Hadamard (multiplicación de elemento por elemento). El número ϵ permite evitar problemas numéricos como divisiones por cero, habitualmente se fija entre 1×10^{-8} y 1×10^{-4}. Un buen valor inicial para el Factor de Aprendizaje inicial η es 0.01, luego el mismo algoritmo se encarga de adaptarlo.

Un detalle en cuanto a Adagrad es el hecho que debido a que el factor G_t siempre está creciendo debido a que corresponde a la suma de números positivos la corrección del factor de aprendizaje puede verse como una función monótonamente decreciente.

El beneficio más evidente de Adagrad es que no es necesario sintonizar de forma manual el Factor de Aprendizaje, además permite adaptar este factor dependiendo de los datos de entrenamiento, esto usualmente mejora los resultados obtenidos con el método de Gradiente Descendente Estocástica.

RMSprop

RMSProp (o Root Mean Square Propagation en inglés) es un método que permite adaptar el Factor de Aprendizaje a través de la normalización de este factor utilizando la raíz cuadrada del promedio o RMS del gradiente, este método normaliza para cada peso de la red de forma diferente, tal como lo hace Adagrad pero utilizando el segundo momento del gradiente, es decir G_t, como se puede ver en la ecuación (2.14):

$$\begin{aligned} g_{t-1} &= \nabla_\theta J(\theta_{t-1}) \\ M_t &= \rho M_{t-1} + (1-\rho) g_{t-1} \\ G_t &= \rho G_{t-1} + (1-\rho) g_{t-1} \odot g_{t-1} \\ \theta_t &= \theta_{t-1} - \frac{\eta}{\sqrt{G_t - M_t^2 + \epsilon}} \odot g_{t-1}, \end{aligned} \tag{2.14}$$

algunos autores incluso agregan el primer momento M_t. En la ecuación (2.14) se incluye el factor ρ para generar la media móvil de los primeros y segundos momentos del gradiente, el valor típico de este hiperparámetro es 0.9. El factor $\sqrt{G_t - M_t^2 + \epsilon}$ que normaliza a η también puede ser considerado como el RMS del gradiente y por eso el nombre del método. Este método no tiene una publicación asociada, pero fue presentada en un curso de aprendizaje de máquinas realizado por Tijmen Tieleman y Geoffrey Hinton [36], conocidos por sus trabajos en el desarrollo de herramientas de entrenamiento de redes neuronales entre ellas Backpropagation. Este método ofrece la ventaja de que la adaptación del Factor de Aprendizaje no es tan agresiva como en Adagrad. Un detalle interesante es que RMSProp puede ser considerado como una generalización del método RProp o Resilient Propagation [37], el cual solo utiliza la dirección del gradiente y no su valor para realizar las actualizaciones de los pesos sinápticos y que tiene la característica de lograr la convergencia muy rápidamente.

Adadelta

Adadelta es un método de adaptación del Factor de Aprendizaje pero que incluye en el cálculo G_t el segundo momento del gradiente, este segundo momento es calculado a partir de

una media móvil de la misma forma que RMSProp. Además, se incluye el término X_t el cual representa el segundo momento de la actualización de los parámetros tal como se muestra en la ecuación (2.15). La inclusión del RMS de v_t permite que no sea necesario la utilización de un factor de aprendizaje, por otro lado este término al igual que con RMSProp permite suavizar el efecto de cambiar de forma agresiva el Factor de Aprendizaje por parte del método Adagrad, lo anterior facilita la convergencia y los tiempos de computo [31],

$$\begin{aligned} g_{t-1} &= \nabla_\theta J(\theta_{t-1}) \\ G_t &= \rho G_{t-1} + (1-\rho) g_{t-1} \odot g_{t-1} \\ X_t &= \rho X_{t-1} + (1-\rho) v_{t-1} \odot v_{t-1} \\ v_t &= \frac{\sqrt{X_t + \epsilon}}{\sqrt{G_t + \epsilon}} \odot g_{t-1} \\ \theta_t &= \theta_{t-1} - v_t. \end{aligned} \tag{2.15}$$

Para calcular la media de los segundos momentos del gradiente G_t y la actualización de los parámetros X_t se utiliza el método de la media móvil que tiene como parámetro de ajuste la constante ρ como se puede ver en la ecuación (2.15), el valor típico de ρ es 0.9.

Adam

Adam o *Adaptive Moment Estimation* [32] es un método de adaptación del Factor de Aprendizaje para cada parámetro de la red de forma diferente y simultanea que incluye dentro de su cálculo la media móvil del primer y segundo momento del gradiente, es decir M_t y G_t respectivamente, como se ve en la ecuación (2.16):

$$\begin{aligned} g_{t-1} &= \nabla_\theta J(\theta_{t-1}) \\ M_t &= \beta_1 M_{t-1} + (1-\beta_1) g_{t-1} \\ G_t &= \beta_2 G_{t-1} + (1-\beta_2) g_{t-1} \odot g_{t-1} \\ \hat{M}_t &= \frac{M_1}{1-\beta_1^t} \\ \hat{G}_t &= \frac{G_1}{1-\beta_2^t} \\ \theta_t &= \theta_{t-1} - \frac{\eta}{\sqrt{\hat{G}_t + \epsilon}} \odot \hat{M}_t. \end{aligned} \tag{2.16}$$

Para el cálculo de la media móvil se incluyen los hiperparámetros β_1 y β_2 los cuales habitualmente son muy cercanos a 1, típicamente 0.999. Además, se incluye una etapa que corrige posibles sesgos en los momentos generados por la inicialización de los vectores M_t y G_t. Para ello se divide a cada momento por $1 - \beta_i^2$.

La forma de actualizar los pesos de la red son similares a los vistos en RMSProp y Adadelta tal como se puede apreciar en la ecuación (2.16). En la práctica Adam resulta tener un mejor desempeño en cuanto a tiempo de convergencia y exactitud al momento de encontrar el óptimo al compararlo con otros métodos de adaptación del factor de aprendizaje [32].

2.3.6. Entrenamiento de Redes Neuronales

Las formas de entrenar una red neuronal varían dependiendo del tipo de paradigma de aprendizaje y los objetivos del mismo, pero es posible mencionar que en general se tratan de procesos de optimización que buscan encontrar la combinación de pesos sinápticos que minimicen algún funcional de costo dependiente de los objetivos de la red: reducción de error, minimización de energía (Boltzmann), etc.

La función objetivo en general es de tipo convexa para garantizar la existencia de mínimos, aunque no siempre es posible garantizar unicidad debido a las no linealidades de las funciones de activación. En el Aprendizaje Supervisado las funciones objetivas en general buscan minimizar el error entre la salida del modelo y la salida deseada, a su vez la función objetivo depende de los pesos sinápticos de la red, por tanto esta función está definida en un espacio $\mathbb{R}^N$ donde N es la cantidad de pesos sinápticos. El problema de optimización es encontrar un punto mínimo sobre la superficie generada por la función objetivo definida en el espacio de parámetros, lo anterior se puede resolver a partir de la dirección del gradiente negativo en la cual la función del error tendrá un menor crecimiento.

Durante el entrenamiento se presentan los ejemplos al modelo para obtener el error, luego se obtiene el gradiente para corregir los pesos sinápticos, esta corrección de los pesos se realiza para cada capa de la red a través del algoritmo de retro propagación del error o Backpropagation en inglés (ver 2.3.6). Es posible que para alcanzar el óptimo sea necesario más de una iteración por tanto otro parámetro importante es la cantidad de iteraciones máximas que se deben realizar, a las cuales durante el entrenamiento se les llama épocas.

Backpropagation

El método de Backpropagation (o retro-propagación en español) permite obtener una forma de actualizar los pesos de una red a partir de la propagación del error hacia las capas intermedias u ocultas [38]. Este método fue propuesto como una forma de actualizar los pesos sinápticos de las capas ocultas de una red con funciones de activación no lineales. El proceso cuenta con 2 etapas una de propagación hacia adelante la cual se encarga de generar la predicción de la red a partir de los datos de entrada, la segunda etapa es la de propagación hacia atrás la cual se encarga de propagar el error hacia las capas intermedias. Este error es el obtenido entre la predicción y la salida deseada. Por tanto, el método permite actualizar los pesos intermedios de la red a partir de solo el error de la red (ver algoritmo 4).

Algoritmo 4 Algoritmo Backpropagation (el símbolo $\odot$ representa el producto de matrices Hadamard, es decir multiplicación elemento por elemento).

Entrada: L el número de capas en la red neuronal.
$\theta = \{\omega^l, b^l\}_{l\in\{1,..,L\}}$ parámetros de la red.
(X, y) datos de entrada y salida respectivamente.
$J(e)$ función de costo.
Salida: El gradiente de la función de costo para cada capa.

Calcular la salida de todas las capas de la red
$a^1 \leftarrow X$
para $l = 2$ hasta L **hacer**
  $z^l \leftarrow w^l a^{l-1} + b^l$
  $a^l \leftarrow \varphi(z^l)$
fin para
Calcular el error de la capa de salida
$\delta^L \leftarrow \nabla_a J(a^L - y) \odot \varphi'(z^L)$
Propagar el error
para $l = L - 1$ hasta 2 **hacer**
  $\delta^l \leftarrow ((w^{l+1})^T \delta^{l+1}) \odot \varphi'(z^l)$
fin para
devolver $\frac{\partial J}{\partial \omega^l} = a^{l-1}\delta^l$ y $\frac{\partial J}{\partial b^l} = \delta^l$

2.4. Aprendizaje con Ensambles

Aprendizaje con Ensambles es un método de aprendizaje que utiliza un conjunto de modelos para generar un modelo final. Para ello combina de forma conveniente las distintas salidas ya sea a través de una función u otro modelo. Para poder formalizar el concepto de Ensamble primero es importante introducir el concepto de modelo o máquina de aprendizaje. Como se vio en la expresión (2.1) un modelo en el paradigma de Aprendizaje Supervisado puede ser visto como una relación entre un conjunto de entrada $\mathcal{X}$ y uno de salida $\mathcal{Y}$, el proceso de aprendizaje por tanto intenta buscar esta relación tal que cumpla con los requerimientos de optimización definidos por los objetivos de aprendizaje, por ejemplo la minimización del error cuadrático medio.

En el Aprendizaje Supervisado es posible definir una tupla que contiene los elementos de entrada y salida que se intentan modelar, es decir un conjunto $\mathcal{D}_{real} := \mathcal{X} \times \mathcal{Y}$ el cual representa todas las combinaciones posibles de elementos de entrada y salida de los estados de la realidad. Un detalle importante es reconocer que en un problema real estos datos son solo una representación simplificada de la realidad con el objeto de poder utilizarlos en computadores, por lo tanto cualquier inferencia que se pueda tomar de estos últimos debe considerar este hecho al momento de generar un análisis final. En general en los problemas prácticos no es posible trabajar con el conjunto completo $\mathcal{D}_{real}$ por lo que solo se utiliza un subconjunto del mismo al cual llamaremos conjunto de ejemplo $\mathcal{D} \subseteq \mathcal{D}_{real}$, este conjunto debe ser representativo de $\mathcal{D}_{real}$ de lo contrario cualquier conclusión sería solo aplicable al subconjunto $\mathcal{D}$.

Los modelos deben utilizar el conjunto de ejemplo $\mathcal{D}$ para poder aprender a predecir.

Para formalizar y sin pérdida de generalidad es posible definir el objetivo de aprendizaje como la minimización de una función de costo $J : \mathcal{D}_{real} \times \Theta \rightarrow \mathbb{R}$, donde el conjunto Θ es el espacio donde se definen los parámetros del modelo. Dada las definiciones anteriores es posible definir el proceso de aprendizaje de un modelo $h : \mathcal{X} \rightarrow \mathcal{Y}$ como:

$$h(\cdot) := h(\cdot,\ \theta^*) \quad : \quad \theta^* = \underset{\forall\theta\in\Theta}{\arg\min}\ J(\mathcal{D},\ \theta). \tag{2.17}$$

En el caso de Ensambles se utiliza más de un modelo de aprendizaje por tanto es conveniente definir el conjunto $\Gamma_M = \{h_i(\cdot)\}_{i\in\{1..M\}}$ el cual representa a un conjunto de modelos, los cuales pueden ser del mismo tipo o no. El conjunto Γ_M es a su vez subconjunto del espacio de Hipótesis $\mathcal{H}$ compuesto de todos los posibles modelos candidatos para resolver el problema de encontrar un modelo que se asemeje a la relación desconocida $f(x)$ (ver ecuación (2.1)). En general el espacio de hipótesis $\mathcal{H}$ queda definido por los parámetros del modelo.

Se define formalmente un Ensamble como:

$$H(x) = g(h_1(x), h_2(x), ..., h_M(x)), \tag{2.18}$$

donde el término $g(\cdot)$ puede ser una función u otro modelo que combina las salidas de los distintos modelos de Γ_M. El Ensamble por tanto es un modelo $H : \mathcal{X} \rightarrow \mathcal{Y}$ similar a la definición funcional de los modelos que utiliza.

Los modelos que utiliza un Ensamble pueden ser o no similares en sus tipos, por ejemplo, existen trabajos en donde se combinan Arboles de Decisión, Redes Neuronales, Maquinas de Soporte Vectorial, etc. Por otro lado, existen trabajos donde solo se utilizan modelos simples o débiles similares en cuanto a su tipo entre si, un ejemplo de aquello son los métodos de Boosting y Bagging. Las funciones o métodos de combinación de las salidas de los modelos se describirán en mayor detalle en la sección 2.4.4, sin embargo es posible adelantar que en general se utiliza el promedio en el caso de modelos de regresión y máxima votación o mayoría en el caso de los modelos de clasificación. Lo anterior se debe a que no existen razones concluyentes que inclinen la balanza sobre un método particular que funcione en todos los casos, por tanto la elección depende de cada problema. Además, existen métodos como Stacking [20] que generan un nuevo modelo a partir de las salidas de los modelos del Ensamble, este método ha demostrado buenos resultados en ciertos problemas pero al igual que en el caso de las funciones no para todos.

El sentido de utilizar Ensambles es la necesidad de mejorar los resultados de modelos ya existentes, se utiliza en casos donde los modelos simples por sí solos no son capaces de mejorar su predicción. Lo anterior implica que en el caso de tener un modelo simple con un error de predicción muy bajo o precisión casi perfecta la utilización de Ensambles probablemente no tenga sentido.

En los Ensambles una característica importante que resalta es el hecho de la existencia de redundancia, debido a que todos los modelos intentan resolver el mismo problema o al menos una parte de él. Lo anterior puede ser visto como positivo debido a que puede ayudar a mejorar el desempeño en cuanto a la generalización, debido a que durante el entrenamiento los modelos pudieron haber aprendido ciertas características diferentes entre ellas de forma que en complemento pueden predecir considerando un espacio sobre el conjunto de características mayor que el que cubren por sí solos.

La intuición respecto a porque los Ensambles pueden ayudar a mejorar el desempeño lo da el concepto de comité de expertos. La opinión de un experto puede representar la predicción de un modelo de forma que aunque el experto posiblemente acierte en varios casos sobre algún asunto pero no es infalible y por tanto se puede equivocar. Para evitar este problema considerar no solo la opinión de un experto sino más bien la de un comité de ellos de forma que la posibilidad de error disminuya. La analogía anterior, aunque sirve para entender como un Ensamble puede mejorar los resultados plantea interrogantes como ¿cuál es la mejor forma de tomar la decisión final? ¿cómo se logra que el aporte de los miembros del comité sea significativo para el resultado final? ¿es solo la cantidad de miembros la que define la precisión? de estas preguntas se intenta hacer cargo el concepto de diversidad, el cual es un concepto todavía no claro pero lo que intenta explicar es como las diferencias entre los modelos puedan ayudar a mejorar resultados y no solo generar redundancia, es una especie de complementariedad, para mayor detalles se incluye una sección dedicada al concepto de diversidad 2.5.

En los Ensambles se busca generar entre los modelos diversidad de forma de poder tener un aporte significativo en el resultado final de cada uno de ellos. Esta diversidad se puede llevar a cabo de forma explícita o implícita, además es posible fomentarla de diversas formas como por ejemplo durante el entrenamiento, la selección de las estructuras e inicialización de los modelos, la forma de combinar las salidas o el conjunto de entrenamiento que utilicen cada modelo, lo importante es evitar que los modelos sean homogéneos pero a su vez considerando que estos deben resolver un problema común. Existen diferentes formas de generar Ensambles y diversidad de los mismos, a continuación se presentan los métodos más conocidos.

2.4.1. Boosting

Boosting es un algoritmo perteneciente al Aprendizaje de Ensambles que permite convertir modelos débiles en modelos fuertes de aprendizaje. La definición de un modelo o máquina de aprendizaje débil es que tiene una baja correlación con la salida esperada, en casos de clasificación implica que es ligeramente mejor que un clasificador que solo prediga la clase en forma aleatoria, el caso contrario es un modelo fuerte el cual es capaz de alcanzar precisión de predicción importante. La primera implementación de Boosting fue propuesta por Robert Schapire en 1990 como respuesta a la pregunta que años antes propusieron Kearns and Valiant sobre la posibilidad de convertir a los clasificadores débiles en fuertes [18]. Los detalles del método se pueden ver en el algoritmo 5.

En Boosting los modelos son generados a partir de muestras del conjunto de entrenamiento que son seleccionadas según una distribución especifica generada a partir de los errores generados por los modelos actuales el Ensamble. El método Boosting puede considerarse un método de Ensambles paralelo debido a que los modelos en cierta forma son generados de forma paralela debido a la interacción de la distribución de las muestras.

Un algoritmo conocido de Boosting es Adaboost [39] el cual tiene como objetivo minimizar una función de costo exponencial relacionada con la distribución de las muestras y el error.

Algoritmo 5 Boosting

Entrada: M Número de modelos.
$h = \{h_1, .., h_M\}$ Conjunto de modelos.
N Número de ejemplos utilizados en el entrenamiento.
$d \subseteq \mathcal{D}$ Conjunto de entrenamiento.
$g(\cdot)$ Función para combinar los modelos.

Salida: Se devuelve el ensamble $H_{boost}(X)$.

1: Inicializar la distribución $W = \{w_1, .., w_N\}$ de las muestras
2: **para** $m = 1$ hasta M **hacer**
3: Entrenar un modelo h_m a partir del conjunto d y su distribución W.
4: Evaluar el error:
5: $\varepsilon_m \leftarrow P_{x \sim W}\left(\mathbb{I}(h_m(x) \neq y_i)\right)$
6: Actualizar la distribución W de las muestras a partir del error ε_m.
7: **fin para**
8: **devolver** $H_{boost}(x) = g(h_1(x), h_2(x), .., h_M(x))$

2.4.2. Bagging

Bagging viene de Booststrap aggregation es un algoritmo que permite generar un modelo fuerte a partir de un conjunto de modelos débiles de aprendizaje, fue propuesto por Leo Breiman en 1994 [17]. La descripción del método se puede ver en el algoritmo 6. En síntesis el método entrena cada modelo que lo compone a partir del muestreo uniforme del conjunto de entrenamiento utilizando reemplazo. Bagging se puede considerar como un método de Ensamble secuencial debido a la forma de obtener los modelos la que explota la dependencia entre ellos [2, p. 47] además de fomentar de forma implícita la diversidad entre sus modelos.

Algoritmo 6 Bagging

Entrada: M Número de modelos.
$h = \{h_1, .., h_M\}$ Conjunto de modelos. N Número de ejemplos utilizados en el entrenamiento.
$N' \leq N$ Número de ejemplos del bootstrap.
$d \subseteq \mathcal{D}$ Conjunto de entrenamiento.
$g(\cdot)$ Función para combinar los modelos.

Salida: Se devuelve el ensamble $H_{bag}(X)$.

1: **para** $m = 1$ hasta M **hacer**
2: Generar un conjunto d_m seleccionando N' muestras por reemplazo desde d.
3: Entrenar un modelo h_m con el conjunto d_m.
4: Agregar el modelo h_m al Ensamble.
5: **fin para**
6: **devolver** $H_{bag}(x) \leftarrow g(h_1(x), h_2(x), .., h_M(x))$

Bagging genera un conjunto d_m de largo N' para entrenar al modelo h_m, como se mencionó este conjunto se genera a partir de un muestreo uniforme desde el conjunto de entrenamiento d, el conjunto d_m es conocido como muestreo bootstrap [40]. La salida del modelo generado con Bagging se obtiene a partir de los promedios sobre todas las salidas de los modelos que lo constituyen. Como es posible notar los modelos h_m son entrenados de forma independiente,

además estos modelos pueden ser heterogéneos. Dadas la características de este método la probabilidad de que un mismo patrón aparezca en la muestra es de $(1-1/M)^M$ lo que converge a 0.632 cuando $M \to \infty$ (M número de modelos). Bagging permite mejorar la capacidad de generalización y ayuda a mejorar la estabilidad de los modelos. Uno de los algoritmos más conocido de este método es Random Forest el cual utiliza arboles de decisión como modelos bases, además en Random Forest se puede seleccionar las dimensiones [41].

2.4.3. Stacking

Stacking o también llamado Stacked generalization [20] es un método para generar Ensambles pero que en vez de simplemente utilizar una función para combinar las salidas de los modelos utiliza otro modelo de aprendizaje. Stacking funciona en etapas: primero se entrena una serie de modelos en forma independiente utilizando el mismo conjunto de entrenamiento, luego en la segunda etapa se genera un nuevo conjunto de entrenamiento a partir de las salidas de los modelos de nivel-0, en la tercera etapa y final se entrena un modelo llamado de nivel-1 a partir del nuevo conjunto de entrenamiento. El detalle de este método se puede ver en el algoritmo 7.

Algoritmo 7 Stacking

Entrada: M Número de modelos.
$h^0 = \{h_1^0, .., h_M^0\}$ Conjunto de modelos de nivel-0.
h^1 Modelo de nivel-1.
N Número de ejemplos utilizados en el entrenamiento.
$d \subseteq \mathcal{D}$ Conjunto de entrenamiento.
Salida: Se devuelve el ensamble $H_{stack}(X)$.

para $m = 1$ hasta M **hacer**
  Entrenar el modelo h_m con el conjunto d.
fin para
Generar el conjunto d' a partir de las salidas de los modelos.
$d' \leftarrow \emptyset$
para $i = 1$ hasta N **hacer**
  para $m = 1$ hasta M **hacer**
    $z_{i,m} \leftarrow h_m^0(x_i)$.
  fin para
  $d' \leftarrow d' \cup ((z_{i,1}, .., z_{i,M}), y_i)$.
fin para
Entrenar h^1 a partir del conjunto d'.
devolver $H_{stack}(x) \leftarrow h^1(h_1^0(x), h_2^0(x), .., h_M^0(x))$

Stacking mejora el desempeño de los modelos a nivel individual. Por otro lado se considera un método de Ensamble secuencial debido a que los modelos bases o de nivel-0 son entrenados de forma independiente. Un algoritmo conocido de tipo Stacking es Blending [42] el cual fue utilizado por 2 equipos durante una competencia de clasificación desarrollada por la empresa Netflix, uno de estos equipos fue el vencedor en aquella competencia.

2.4.4. Métodos de combinación

Una parte importante en la estructura de un Ensamble es la combinación de las distintas salidas de los modelos bases, mientras en el método de Stacking se utiliza otro modelo de aprendizaje para esta tarea la gran mayoría de los otros métodos necesitan de una función para ello. Existen varias razones para poder utilizar la combinación de diversos modelos [43], algunos de los más importantes son:

- Problemas estadísticos: El espacio de Hipótesis $\mathcal{H}$ en general tiende a ser más grande para ser explorado a través del limitado conjunto de entrenamiento, es decir que existen diferentes modelos u hipótesis que obtienen la misma precisión sobre el conjunto de entrenamiento, sin embargo al seleccionar un solo modelo de este conjunto puede correrse el riesgo que no funcione con menos datos. Este riesgo se disminuye si se utiliza una combinación de distintas hipótesis.
- Problemas computacionales: Muchos métodos de aprendizaje generan una búsqueda de óptimos que puede dar como resultado un óptimo local independiente de la cantidad de muestras que tenga el conjunto de entrenamiento, sin embargo, si se utiliza una combinación de distintos modelos que partan desde diferentes puntos de inicio es posible que disminuya el riesgo de caer en un óptimo local.
- Problemas de representación: Es posible que la hipótesis correcta que se desea encontrar (ver ecuación (2.1)) no pueda ser representado por ningún elemento del espacio de Hipótesis $\mathcal{H}$ utilizado, sin embargo, a través de la combinación de algunos de sus elementos se pueda al menos acercar a aquella.

La profesora Kuncheva muy destacada en el estudio de la diversidad y la combinación de modelos en su libro "Combining Pattern Classifiers" [44] recopila una serie de métodos para combinar distintos modelos poniendo especial interés en los métodos de clasificación. A continuación, se presentarán algunos de estos métodos.

Métodos para modelos de regresión

Para problemas de regresión el método de combinación más popular es el del promedio simple en donde todas las salidas de los distintos modelos que componen un Ensamble son promediadas tal como aparece en la ecuación (2.19):

$$H(X) = \frac{1}{M}\sum_{i=1}^{M} h_i(X), \tag{2.19}$$

más adelante se explica cómo este método a pesar de ser tan simple puede garantizar la reducción del error cuadrático medio o RMS del Ensamble respecto a los RMS individuales de los modelos que lo constituyen (ver sección 2.5.2). En esta función se combinan las salidas sin preferencia alguna sobre los modelos, todos los modelos aportan a la salida de la misma forma.

Otra forma de combinar las salidas de los modelos es a través de una suma ponderada tal como aparece en la ecuación (2.20):

$$H(X) = \sum_{i=1}^{M} w_i h_i(X), \tag{2.20}$$

es habitual asumir que los pesos w_i son positivos y que la suma de estos es uno, es decir:

$$w_i \geq 0 \forall i \in \{1, .., M\} \quad \text{y} \quad \sum_{i=1}^{M} w_i = 1, \tag{2.21}$$

las restricciones anteriores de los pesos w_i pueden ser vistos como una normalización y resultan ser útiles al momento de generar ciertas herramientas para el análisis de los ensambles (ver sección 2.5.1). La combinación de las salidas por el promedio puede ser vista como un caso particular donde los pesos son uniformes. Dadas las restricciones de la ecuación (2.21) es posible definir el error de un Ensamble como [45]:

$$err(H) = \sum_{i=1}^{M} \sum_{j=1}^{M} C_{ij} \tag{2.22}$$

donde C_{ij} se define como:

$$C_{ij} = \int (h_i(x) - y(x))(h_j(x) - y(x))p(x)dx, \tag{2.23}$$

donde $p(x)$ es la distribución de las muestras. Si se requiere minimizar el error $err(H)$, es decir disminuir el error de predicción del Ensamble se puede plantear el siguiente problema de optimización [2, p. 70]:

$$w = \arg\min_{w} \sum_{i=1}^{M} \sum_{j=1}^{M} w_i w_j C_{ij}, \tag{2.24}$$

finalmente, utilizando multiplicadores de Lagrange se obtiene el siguiente resultado [45]:

$$w_i = \frac{\sum_{j=1}^{M} C_{ij}^{-1}}{\sum_{i=1}^{M} \sum_{j=1}^{M} C_{ij}^{-1}}. \tag{2.25}$$

La ecuación (2.25) permite definir los pesos w_i, en la práctica C_{ij} se obtiene de forma empírica a partir del error promedio de los modelos.

Métodos para modelos de clasificación

Antes de mostrar los métodos de combinación para problemas de clasificación es importante definir las salidas de los modelos teniendo en consideración que éstas representan un conjunto discreto de etiquetas de clase. Para los problemas de clasificación multi-clase es típica la representación "One Hot Encoding" en donde las clases se representan como un vector de L elementos donde L representa la cantidad de clases y donde cada elemento del vector

representa una clase. En esta codificación solo puede existir un 1 en el vector mientras que los demás valores son cero, por tanto, la posición donde esté dispuesto el 1 es la clase que representa el vector. Para la salida de un modelo con esta codificación se define de forma general este vector como $h_i(x) = (h_i^1, .., h_i^L)$.

En muchas ocasiones los modelos intrínsecamente no entregan valores discretos de tal forma que en necesario codificar las clases posteriormente al entrenamiento, por otro lado, es posible utilizar datos no discretos como valores de confidencias de las clases, de forma que la información no se pierda en la codificación y pueda ser utilizada como por ejemplo determinar qué tan confiable es una predicción.

En los problemas de clasificación de 2 clases también es posible encontrar una codificación de clases en donde se utilice una sola salida y las clases sean determinadas por su cercanía con +1 y 0 o +1 y -1, lo selección anterior es típica de algoritmos como SVM y redes neuronales, por otro lado, también es posible utilizar algún umbral para realizar la clasificación de las salidas.

Dadas las codificaciones anteriores y sin pérdida de generalidad se utilizaran las codificaciones $h_i^j(x) \in \{0, 1\}$ para salidas discretas y $h_i^j(x) \in [0, 1]$ cuando se utilice la salida continua de un modelo, considerando a esta última como una especie de probabilidad a posteriori ($P(c_j|x)$) o confidencia de la clase c_j.

El primer método de combinación es el de votación por mayoría en donde la elección se realiza considerando un resultado mayor a la mitad de los votos, de lo contrario se rechaza la clasificación, lo anterior queda expresado en la ecuación (2.26),

$$H(x) = \begin{cases} c_j & \text{Si } \sum_{i=1}^{M} h_i^j(x) > \frac{1}{2} \sum_{k=1}^{L} \sum_{i=1}^{M} h_i^k(x) \\ \text{Se rechaza} & \sim . \end{cases} \tag{2.26}$$

Una variación del método anterior es el de votación simple en este caso no se rechaza ninguna muestra por falta de mayoría y la elección se realiza utilizando a la clase que recibe la votación más alta, por tanto, la salida del Ensamble es el que se muestra en la ecuación (2.27):

$$H(x) = c_{\arg\max_j \sum_{i=1}^{M} \sum_{j=1}^{M} h_i^j(x)}. \tag{2.27}$$

En los 2 métodos anteriores se considera que cada voto de los modelos del Ensamble pesa lo mismo, es decir no existen preferencias sobre los modelos que afecten la votación. Una alternativa a esto es el método de votación por pesos tal como se muestra en la ecuación (2.28):

$$H(x) = c_{\arg\max_j \sum_{i=1}^{M} \sum_{j=1}^{M} w_i h_i^j(x)} \tag{2.28}$$

en esta votación se le da mayor peso a los modelos que tengan un mejor desempeño durante el entrenamiento. Por ejemplo, los pesos pueden ser definidos de la misma forma que en el caso de regresión (ver ecuación (2.25)).

2.5. Diversidad

No existe una definición formal de diversidad entre distintos modelos [8], sin embargo es fácil entender de forma intuitiva el concepto en el ámbito del aprendizaje con Ensambles. La idea detrás de la diversidad es que los modelos que constituyen un Ensamble deben ser "distintos" entre ellos de forma que su aporte sea significativo y no solo una copia del resto. Pero ¿qué implica ser "distintos"? hay que entender que todos los modelos del Ensamble intentan resolver el mismo problema y en cierta forma siempre existirá cierta correlación en las salidas de estos, por tanto el ser "distintos" o diversos ayuda pero no está claro cual es el límite de aquello, por ejemplo si solo se tuviera 2 modelos en los Ensambles y solo interesara favorecer la diversidad, medida como la correlación negativa entre ellos, es posible que se terminaría con un modelo preciso y otro totalmente opuesto con grandes errores. Al final se logra el objetivo de diversidad pero no el de predicción, por tanto existe cierto compromiso entre diversidad y precisión que actualmente no está totalmente resuelto para problemas de aprendizaje en especial problemas de clasificación.

Para generar diversidad es posible utilizar diferentes caminos a continuación se presentan los más importantes:

- **Manipulación de los datos de ejemplo**: Esto implica que cada modelo del Ensamble es entrenado con un conjunto de datos diferente. Para ello es posible que se pueda realizar un muestreo aleatorio del mismo conjunto de entrenamiento, se utilicen conjuntos disjuntos o se agregue información extra al conjunto. Los métodos más conocidos que utilizan esta forma de generar diversidad son Bagging (muestreo bootstrap) y Boosting (muestreo secuencial).
- **Manipulación de parámetros de aprendizaje**: Es decir que el entrenamiento de cada modelo del Ensamble se realiza con distintos parámetros, por ejemplo la inicialización de los pesos de las redes neuronales. También es posible modificar hiperparámetros como las constantes de los términos de regularización o las de los métodos de adaptación del factor de aprendizaje, etc.
- **Manipulación de cómo se recorre el espacio de parámetros**: Esto se logra modificando la función objetivo de los modelos, para ello se puede utilizar un término de regularización como NCL (*Negative Correlation Learning*) [10], el cual utiliza la correlación negativa de las salidas de los modelos respecto a la salida final del Ensamble.
- **Manipulación de la representación de la salida del modelo**: Esto implica la utilización de diferentes representaciones de la salida de cada modelo, por ejemplo ECOC que emplea técnicas de corrección de errores en una señal para variar las salidas de los modelos [21].

Como ya se mencionó la diversidad no está definida formalmente, sin embargo, existen algunas formas de cuantificarla o al menos entender como favorecerla dentro de un Ensamble. A continuación, se presentan 2 tipos de descomposiciones muy utilizadas para entender la diversidad de los modelos de regresión. En el caso de los modelos de clasificación debido a las limitaciones impuestas por su conjunto objetivo discreto solo es posible un análisis menos general y por tanto solo se revisarán las métricas existentes para ello.

2.5.1. Descomposición Ambigua

Si en un modelo de regresión con Ensamble se combinan los modelos de forma ponderada con factores positivos y suman 1, es decir $\sum_{i=1}^{M} w_i = 1$ (ver sección 2.4.4), Krog y Vedelsby [11] probaron que el error cuadrático del Ensamble es menor o igual al promedio del error cuadrático de los componentes de este. Lo anterior se puede ver más claramente en la ecuación (2.29) en donde aparece un término al que Krogh y Vedelsby identificaron como término Ambiguo:

$$(H-y)^2 = \underbrace{\sum_{i=1}^{M} w_i(h_i-y)^2}_{\text{Error individual}} - \underbrace{\sum_{i=1}^{M} w_i(h_i-H)^2}_{\text{Término Ambiguo}}, \tag{2.29}$$

donde h_i es la salida del modelo i-ésimo y H es la salida ponderada de los M modelos, es decir: $H = \sum_{i=1}^{M} w_i h_i$. En la ecuación (2.29) se ha omitido la entrada X solo para simplificación y limpieza de la expresión, de todas formas, este término aplica para cualquier entrada ($\forall X \in \mathcal{X}$). Además, a la ecuación (2.29) se le conoce también como Descomposición Ambigua, la cual define el término ambiguo:

$$ambi(H) = \sum_{i=1}^{M} w_i(h_i-H)^2, \tag{2.30}$$

este término siempre es positivo y por tanto mientras más grande menor es el error cuadrático de predicción del modelo. Esto implica que para reducir el error de predicción es necesario que la diferencia entre la salida de los modelos individuales y la salida total del Ensamble aumente. Aunque lo anterior es correcto, a medida que los modelos se diferencian de la salida total del Ensamble lo hacen también de la salida esperada y por tanto el error individual aumenta, por tanto se puede deducir que existe un compromiso entre "diferencia" o diversidad de los modelos y la precisión del Ensamble.

La Descomposición Ambigua permite garantizar que en problemas de regresión con Ensambles siempre se tiene un mejor desempeño respecto a los modelos individuales, salvo el caso $h_i = H \ \forall i \in \{1, .., M\}$.

2.5.2. Descomposición Sesgo, Varianza y Covarianza

Otra forma de entender la diversidad en los Ensambles es a través de la Descomposición Sesgo, Varianza y Covarianza [12]. A diferencia de la Descomposición Ambigua en donde solo se calcula el error cuadrático en este caso se calcula el error cuadrático medio o RMS (Root Mean Square), además se considera una función convexa para combinar las salidas del Ensamble donde los pesos son uniformes, es decir el promedio simple de las salidas:

$$\mathbb{E}\{(H-y)^2\} = sesgo(H)^2 + \frac{1}{M} varianza(H) + \left(1 - \frac{1}{M}\right) covarianza(H) \tag{2.31}$$

donde:

$$sesgo(H) = \frac{1}{M}\sum_{i=1}^{M}(\mathbb{E}_i\{h_i\} - y)^2 \qquad varianza(H) = \frac{1}{M}\sum_{i=1}^{M}\mathbb{E}_i\{(h_i - \mathbb{E}_i\{h_i\})^2\}$$

$$covarianza(H) = \frac{1}{M(M-1)}\sum_{i=1}^{M}\sum_{j\neq i}^{M}\mathbb{E}_{i,j}\{(h_i - \mathbb{E}_i\{h_i\})(h_j - \mathbb{E}_j\{h_j\})\},$$

en esta descomposición el sesgo explica que tan cerca se está del objetivo (generalización de modelo). La varianza explica la variación de la estimación bajo diferentes modelos. El término de la covarianza explica cómo se relacionan los modelos dentro de un Ensamble. Si los modelos dentro de un Ensamble cometen los mismos errores sobre el conjunto de entrenamiento el término de la covarianza aumenta y por tanto también lo hace el RMS del Ensamble. Por tanto es preferible que los modelos cometan errores diferentes, de este comportamiento se puede inferir la importancia de la diversidad dado que puede mejorar los resultados de predicción. Además, sigue existiendo un compromiso de la covarianza respecto al sesgo y la varianza dado que disminuir la covarianza implica también aumentar de forma indirecta estos términos.

Finalmente es posible relacionar las dos descomposiciones (2.30) y (2.31), obteniendo la siguiente expresión para la esperanza del término Ambiguo [9]:

$$\mathbb{E}\{ambi(H)\} = varianza(H) - \frac{1}{M}varianza(H) - \left(1 - \frac{1}{M}\right)covarianza(H). \qquad (2.32)$$

De la relación (2.32) es posible advertir que en ambas descomposiciones aparece la varianza, de lo que se desprende la dificultad de afectar el término ambiguo sin afectar el sesgo y por tanto generar diversidad resulta en un problema difícil [2, p. 104].

2.5.3. Formas de medir la Diversidad

Las descomposiciones vistas en las secciones anteriores aplican solamente para problemas de regresión. El problema de hacerlas extensivas a problemas de clasificación radica en la dificultad de poder generalizar cuando el conjunto objetivo del aprendizaje es discreto y por otro lado este conjunto no es intrínsecamente ordinal, todo lo anterior dificulta la definición del concepto de covarianza en estos casos. Según la tesis del profesor Gavin Brown [9, p. 31] la mayor piedra de tope para entender la diversidad en los problemas de clasificación es la no existencia de un símil de la covarianza del error o la descomposición de sesgo-varianza-covarianza en estos problemas.

A pesar de la dificultad de poder cuantificar la diversidad en los problemas de clasificación existen varios trabajos que al menos intentan calificarlo de forma heurística, por ejemplo Sharkey [46] propone patrones de error en un Ensamble de redes neuronales que permiten entender el nivel de diversidad. Estos patrones se dividen en 4 niveles que describen la existencia de coincidencia de errores entre los modelos, a continuación se detallan los niveles:

- **Nivel 1**: No existe coincidencia de error. Además, siempre existe al menos un modelo que genera la predicción correcta para el conjunto de prueba, en este caso la votación máxima o mayoría entre los modelos genera siempre la mejor respuesta por parte del Ensamble.
- **Nivel 2**: Existen coincidencias de error pero la mayoría de los modelos predicen correctamente y al menos uno genera la predicción correcta para el conjunto de prueba. El Ensamble debe ser de 4 o más modelos para mantenerse en este nivel.
- **Nivel 3**: La mayoría de votos no siempre permite obtener la predicción correcta pero al menos un modelo del ensamble siempre predice correctamente dada una entrada del conjunto de prueba.
- **Nivel 4**: El Ensamble no siempre puede predecir correctamente.

Para Sharkey si un Ensamble muestra resultados de los niveles 2 o 3 es posible que se pueda subir de nivel a uno mejor, por ejemplo, si un Ensamble está en el nivel 2 es posible que exista un subconjunto de modelos que pertenezcan al nivel 1, del mismo modo para el caso del nivel 3 es posible que existan subconjuntos que pertenezcan a los niveles 2 y 1.

También existen métricas que permiten cuantificar la diversidad entre pares de modelos y sobre un Ensamble completo, en la literatura estas métricas se identifican como métricas entre parejas o no. Las métricas de parejas de modelos son calculadas a través de la tabla de contingencia (ver tabla 2.2) la cual es generada a partir de la cantidad de resultados positivos y negativos de predicción entre ambos modelos. En la tabla 2.2 el par de modelos está representado por h_i y h_j además "incorrecto" y "correcto" indican el estado del modelo, es decir si el modelo ha cometido error en la predicción o no respectivamente. Las entradas de la tabla indican la cantidad de muestras que coinciden con los estados de los modelos, además la suma $a + b + c + d = N$ indica la cantidad total de muestras utilizadas.

	h_i correcto	h_i incorrecto
h_j correcto	a	c
h_j incorrecto	b	d

Tabla 2.2: Tabla de Contingencia.

A partir de la tabla de contingencia es posible calcular prácticamente todas las métricas de pares de modelos, en los casos de métricas de no pares se utilizan generalmente métodos estadísticos como la entropía y la varianza de las salidas del Ensamble.

En la tabla 2.3 se presentan las métricas más utilizadas para cuantificar la diversidad, la columna "Conocimiento" indica si es necesario conocer las predicciones de los clasificadores individuales de un Ensamble, la columna "Simetría" indica si la métrica cumple la propiedad de simetría cuando se comparan pares de modelos, finalmente los símbolos $\uparrow$ / $\downarrow$ indican en qué dirección de crecimiento o decremento de la métrica se favorece la diversidad.

Métrica Diversidad	**Símbolo**	$\uparrow / \downarrow$	**Entre parejas**	**Conocimiento**	**Simetría**
Disagreement	dis	$\uparrow$	Si	No	Si
Q-statistic	Q	$\downarrow$	Si	No	Si
Coeficiente de Correlación	ρ	$\downarrow$	Si	No	Si
Kappa-statistic	κ_p	$\downarrow$	Si	No	Si
Doble falla	df	$\downarrow$	Si	Si	No
Interrater agreement	κ	$\downarrow$	No	Si	Si
Kohavi-Wolpert (varianza)	kw	$\uparrow$	No	Si	Si
Entropía (C&C's)	Ent_{cc}	$\uparrow$	No	No	Si
Entropía (S&K's)	Ent_{sk}	$\uparrow$	No	Si	Si
Dificultad	θ	$\downarrow$	No	Si	No
Diversidad Generalizada	gd	$\uparrow$	No	Si	No
Coincidencia de Falla	cfd	$\uparrow$	No	Si	No

Tabla 2.3: Detalle de las métricas más conocidas para cuantificar la diversidad en Ensambles (Extraído del libro Ensemble Methods [2, p. 109]).

2.6. Teoría de la Información para el aprendizaje (ITL)

ITL (*Information Theoretic Learning*) hace referencia a un conjunto de herramientas que utilizan descriptores de teoría de la información como la entropía y la divergencia para generar directamente de los datos estimadores que reemplacen a los descriptores estadísticos de segundo orden como la varianza y la covarianza. Las herramientas de ITL han sido utilizadas con éxito en problemas de filtros adaptativos lineales y no lineales, y en problemas de aprendizaje supervisado y no supervisado.

La ventaja de ITL respecto a otros criterios típicos como la norma L2 y MSE (medidas de correlación) es que no asume nada respecto a las funciones de densidad de probabilidad de los datos, sino que esta información es extraída directamente de los datos. Las medidas son estimadas a partir de entropía diferentes a las propuestas por Shannon y a métodos de estimación de distribuciones con ventanas de Parzen.

Antes de mostrar y explicar las herramientas de ITL que se utilizan en este trabajo se presenta un pequeño resumen de los conceptos básicos de Teoría de la Información.

2.6.1. Elementos de Teoría de la información

En teoría de la información un concepto muy importante es la Entropía la cual permite medir el nivel de incertidumbre de una fuente de información. En principio la Entropía surgió del problema de caracterizar el error de transmisión de un mensaje a través de un canal ruidoso. En 1928 Hartley propuso el concepto de cantidad de información para referirse a una medida que intenta cuantificar el número de posibles formas de seleccionar un símbolo a partir de un alfabeto (conjunto de símbolos). Sin embargo, fue Shannon quién propuso la definición que conocemos actualmente, en donde a diferencia de Hartley considera una información a priori de los símbolos, a saber, la probabilidad de ocurrencia en el mensaje.

La Entropía de Shannon puede ser vista como la cantidad de información existente en una señal o mensaje el cual puede ser caracterizado como una variable aleatoria discreta X, por tanto, la cantidad de información de los diferentes estados de X se define según Shannon como:

$$H(X) = \sum_x -p(x)log_2(p(x)) = \mathbb{E}\{-log_2(P(X))\}, \tag{2.33}$$

en donde $p(x) = P(X = x)$ representa la probabilidad del estado x y se utiliza el logaritmo de base 2 para representar la información como código binario, aunque esto último depende de la representación de la información que se utilice. Por otro lado la elección de la función logaritmo responde a principalmente a sus propiedades que se acercan a como intuitivamente se entiende que aumenta la información cuando se varia la magnitud del mensaje [47]. La Entropía es una cantidad adimensional y positiva, y si todos los estados de la señal tienen la misma probabilidad.

Otro criterio básico en Teoría de la Información es la Información Mutua la cual mide la dependencia entre 2 variables aleatorias, en particular como el conocimiento de una de las

variables reduce la incertidumbre respecto a la otra, la definición formal es la siguiente:

$$I(X_1,\ X_2) = \sum_{x_1,x_2} p(x_1,x_2) log\left(\frac{p(x_1,x_2)}{p(x_1)p(x_2)}\right), \tag{2.34}$$

donde $p(x_1)$ y $p(x_2)$ es la probabilidad de la ocurrencia de las variables aleatorias X_1 y X_2 respectivamente, $p(x_1,x_2)$ corresponde a la probabilidad conjunta. A partir de la definición de la Información Mutua es posible advertir la relación con la Entropía: $I(X,\ X) = H(X)$.

A continuación se presentan algunas ecuaciones importantes relacionadas con la Entropía y la Información mutua:

- **Definición Entropía conjunta**:

$$H(X,Y) = \sum_y \sum_x -p(x,y) log_2(p(x,y)) \tag{2.35}$$

- **Definición Información Mutua condicional**:

$$I(X,Y|Z) = H(X|Z) + H(X|Y,Z) \tag{2.36}$$

- **Propiedad Regla de la cadena**:

$$H(X_1,X_2,..,X_n) = \sum_{i=1}^{n} H(X_i|X_{i-1},..,X_1) \leq \sum_{i=1}^{n} H(X_i) \tag{2.37}$$

$$I(X_1,X_2,..,X_n;Y) = \sum_{i=1}^{n} I(X_i;Y|X_{i-1},..,X_1) \tag{2.38}$$

$$\tag{2.39}$$

- **Relación entre Información Mutua y Entropía**:

$$\begin{aligned} I(X,Y) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \end{aligned} \tag{2.40}$$

En la figura 2.10 se puede ver en un diagrama de Venn la relación existente entre las diferentes medidas de información más importantes.

Otra medida importante es la Entropía Relativa o Divergencia de Kullback-Leibler, con la cual es posible generar una medida de "distancia" entre 2 distribuciones de probabilidad, sin embargo, esta distancia es entre comillas debido a que no cumple con la desigualdad triangular, la definición matemática es:

$$D_{KL}(p||q) = \sum_x p(x) log\left(\frac{p(x)}{q(x)}\right) = \mathbb{E}_p\{log\left(\frac{p(X)}{q(X)}\right)\}. \tag{2.41}$$

De esta definición se puede inferir que $I(X,Y) = D_{KL}(p(x,y)||p(x)p(y))$. Esta medida es utilizada en algunos casos como función objetivo o de costo para el entrenamiento de una

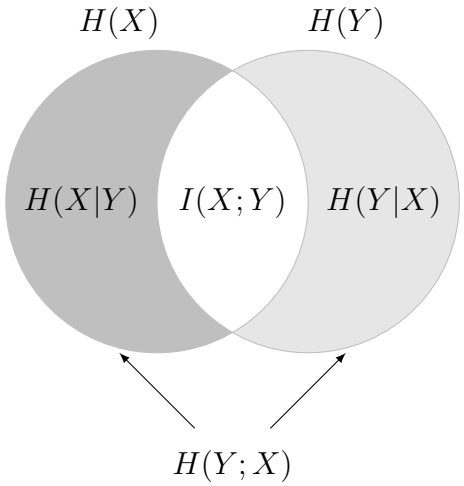


Figura 2.10: Diagrama de Venn que muestra la relación entre Entropía e Información Mutua.

Maquina de Aprendizaje en donde directamente la salida del modelo y la salida deseada son tratadas como una función de probabilidad.

La minimización de la Entropía Relativa de Kullback-Liebler busca que la diferencia entre las distribuciones sea mínima y por tanto igualar el comportamiento estadístico de las señales comparadas. En la practica la definición de la entropía de Shannon (2.33) no resulta fácil de estimar a partir de muestras, por tanto en ITL se utiliza la Entropía de Renyi:

$$H_{R\alpha}(X) = \frac{1}{1-\alpha} \log \left(\sum_{i=1}^{n} p_i^{\alpha} \right), \tag{2.42}$$

la que generaliza el concepto de entropía de Shannon. En la Entropía de Renyi (ver ecuación (2.42)) α es un valor positivo pero diferente de 1 debido a que se produce indeterminación en el término del denominador sin embargo en el límite cuando α tiende a 1 la Entropía de Renyi tiende a la Entropía de Shannon. También es posible definir la Entropía de Renyi para una distribución continua como se ve en la ecuación (2.43) donde $f_X(\cdot)$ es la función de distribución de la variable aleatoria X,

$$H_{R\alpha}(X) = \frac{1}{1-\alpha} \log \left(\int_{-\infty}^{\infty} f_X(x)^{\alpha} dx \right). \tag{2.43}$$

La Entropía de Renyi es una función monotónica de la norma-α de la distribución de probabilidad, donde la norma-α es $||P||^{\alpha} = \sum_{k=1}^{N} p_k^{\alpha}$ en donde $p_k \geq 0$ y $\sum_{k=1}^{N} p_k = 1$. La norma-α se puede ver como la distancia del punto $P = (p_1, p_2, .., p_N)$ al origen además se puede relacionar a la Entropía de Renyi por:

$$H_{R\alpha}(X) = \frac{1}{1-\alpha} \log \left(||P||^{\alpha} \right). \tag{2.44}$$

Para poder estimar la Entropía cuadrática de Renyi H_{R2} a partir de un conjunto de datos

$\{x\} = \{x_i\}_{i\in\{1,..,N\}}$ se utiliza el método de Ventanas de Parzen:

$$\begin{aligned} H_{R\alpha}(X|\{x\}) &= -\log\left(\int_{-\infty}^{\infty} f_X(x)^2 dx\right) \\ &= \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\kappa_h(x-x_i)\kappa_h(x-x_j)dx \end{aligned} \tag{2.45}$$

donde $\kappa_h(\cdot)$ es una función no negativa llamada kernel. Las ventanas de Parzen es un método no paramétrico para la estimación de la función de densidad de probabilidad $\hat{f}_h(\cdot)$ de una variable aleatoria x. Este método utiliza un kernel κ_h para llevar a cabo la estimación:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n}\kappa_h(x-x_i) = \frac{1}{nh}\sum_{i=1}^{n}\kappa\Big(\frac{x-x_i}{h}\Big), \tag{2.46}$$

donde el término h representa los parámetros del kernel. El parámetro h es también conocido como el tamaño la ventana o ancho de banda del kernel y tiene un efecto importante en los resultados de la estimación de la densidad de probabilidad. Si h es muy pequeño el resultado estará muy influenciado por las fluctuaciones de los datos generando una estimación ruidosa, por otro lado, si h es muy elevado es posible que la estimación este sobre suavizada, es decir que muchos de los datos sean filtrados generando una estimación incorrecta. Una buena primera aproximación al valor de h es la regla se Silverman [48] presentada en la siguiente ecuación:

$$h = \hat{\sigma}\left(\frac{4}{(2d+1)n}\right)^{\frac{1}{d+4}}, \tag{2.47}$$

donde d representa la dimensión de las muestras, n la cantidad de muestras y $\hat{\sigma}$ es la desviación estándar de las muestras. La ecuación (2.47) entrega un valor óptimo de h el cual permite minimizar el Error Medio Cuadrático Integrado (MISE) considerando un kernel Gaussiano.

2.6.2. Correntropía

La Correntropía es una medida de similitud entre variables aleatorias que incluye información de relación estadística, su nombre proviene de la correlación y la entropía. La correlación es una medida de relación lineal entre 2 variables estadísticas y la entropía es una medida de la incertidumbre de una fuente de información. La Correntropía también llamada Función de Correlación Generalizada utiliza una estimación de la distribución de los datos, la que se genera a partir del método de Ventanas de Parzen (ver ecuación (2.46)) [7].

La Correntropía define una función de correlación generalizada a partir de productos internos de vectores $\phi(x_{t_1})$ y $\phi(x_{t_2})$ en un espacio de características definido por el kernel κ, el producto interno es remplazado por la operación del kernel (truco del kernel). La auto Correntropía se define como:

$$V(x_{t_1}, x_{t_2}) = \mathbb{E}\{< \phi(x_{t_1}), \phi(x_{t_2}) >\} = \mathbb{E}\{G_h(x_{t_1} - x_{t_2})\}, \tag{2.48}$$

donde x_{t_1} e x_{t_2} son variables aleatorias de un proceso estocástico, $\mathbb{E}\{\cdot\}$ la esperanza estadística de la variable aleatoria x y G_h es el kernel Gaussiano de parámetro h. Cuando el kernel

es Gaussiano y se utiliza ventana de Parzen para obtener la densidad de probabilidad, la Correntropía se puede expandir con series de Taylor reduciendo a la forma que se muestra a continuación:

$$V(x_{t_1}, x_{t_2}) = \frac{1}{2\sqrt{\pi h}} \sum_{n=0}^{\infty} \frac{(-1)^n}{2^n h^{2n} n!} \mathbb{E}\left[||x_{t_1} - x_{t_2}||^{2n}\right], \tag{2.49}$$

donde h es el parámetro del kernel Gaussiano y $\mathbb{E}(\cdot)$ es la Esperanza Matemática. Es importante destacar que la Correntropía mide similitud a través del kernel debido a que esta operación representa el resultado de un producto interno, los cuales son una medida de similitud entre pares de vectores. Por otro lado la Correntropía puede ser definida para dos procesos estocásticos distintos x e y por ejemplo, de forma que la Correntropía cruzada según la ecuación:

$$V_{x,y,h}(t_1, t_2) = \mathbb{E}\{\kappa_h(x_{t_1} - y_{t_2})\}. \tag{2.50}$$

2.6.3. Potencial de Información

El Potencial de Información IP (Information Potential) es una medida de ITL que permite estimar la entropía cuadrática de Renyi, ver ecuación (2.45). IP puede ser utilizado como criterio de optimización del proceso de entrenamiento de un modelo donde se busca minimizar la entropía del error de predicción. Para estimar la Entropía Cuadrática de Renyi se utiliza IP junto con un kernel Gaussiano Multidimensional de la forma:

$$\kappa_\Sigma(x) = G(x, \Sigma) = \frac{1}{\sqrt{(2\pi)^M |\Sigma|}} exp\left(-\frac{1}{2} x^T \Sigma^{-1} x\right), \tag{2.51}$$

donde $x \in \mathbb{R}^M$ y Σ es la matriz de covarianza, entonces al reemplazar (2.51) en (2.45) se obtiene:

$$H_{R2}(X|\{x\}) = -log\left(\frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} G(x_i - x_j, 2\sigma^2 I)\right) = -log(V(\{x\})), \tag{2.52}$$

en donde se utiliza la propiedad de convolución de las Gaussianas $\int G(z - x_i, \Sigma_1) G(z - x_j, \Sigma_2) dz = G(x_i - x_j, \Sigma_1 + \Sigma_2)$ además se sustituye Σ por $\sigma^2 I$ donde I es la matriz identidad y σ actúa como parámetro del kernel Gaussiano. El término $V(\{x\})$ en la ecuación (2.52) se conoce como Potencial de Información el cual se define como la suma de las interacciones de todos los pares de datos o partículas,

$$V(x) = \frac{1}{N^2} \sum_i \sum_j V_{ij}, \tag{2.53}$$

donde la interacción entre partículas se define como $V_{ij} = G(x_i - x_j, 2\sigma^2 I)$. Por tanto la intensidad de las interacciones se debe al campo local generado por el kernel Gaussiano, además esta interacción puede ser controlada a través del tamaño del kernel σ. El Potencial de Información se puede entender siguiendo una analogía del mundo físico como la energía potencial de los datos generado por el kernel. Siguiendo esta analogía física se puede definir la Fuerza de la Información como la derivada del Potencial de Información, es decir:

$$IF(x_i, x_j) = \frac{\partial}{\partial x_i} G(x_i - x_j, 2\sigma^2 I) = -\frac{1}{2\sigma^2} G(x_i - x_j, 2\sigma^2 I)(x_i - x_j). \tag{2.54}$$

Sumando todas las contribuciones de Fuerza de la Información o IF (Information Force) del conjunto de muestras sobre x_i se obtiene el efecto total del Potencial IP sobre este dato. En problemas de entrenamiento de máquinas de aprendizaje en donde se requiere maximizar la Entropía cuadrática la que corresponde al negativo del logaritmo del Potencial de Información (ver ecuación (2.52)) entonces maximizar la entropía equivale a minimizar el Potencial de Información.

2.6.4. Potencial de Información Cruzado

El Potencial de Información Cruzado o CIP (Cross Information Potential) permite calcular la interacción entre Potenciales de Información basados en variables aleatorias. CIP permite estimar la Información Mutua Cuadrática usando las Divergencias de Cauchy-Schwartz y Euclideana. Antes de presentar como CIP se relaciona con estas medidas de divergencias se presenta la forma del Potencial de Información Cruzado:

$$V_c(\mathbf{y}) = \frac{V_{nc}(\mathbf{y})^2}{V_J(\mathbf{y}) \prod_{k=1}^{M} V^k(\{y_k\})}, \tag{2.55}$$

donde $\mathbf{y} = \{\mathbf{y}_1, \mathbf{y}_2, .., \mathbf{y}_M\}$ es un vector de variables aleatorias y cada término se define como:

$$V_J(\mathbf{y}) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\prod_{k=1}^{M} G(y_{ik} - y_{jk}, 2\sigma^2 I) \right) \quad , \tag{2.56}$$

$$V_j^k = V^k(y_j, \{y_k\}) = \frac{1}{N} \sum_{i=1}^{N} G(y_{ik} - y_{jk}, 2\sigma^2 I) \qquad \text{Donde } k = \{1, .., M\}, \tag{2.57}$$

$$V^k = V^k(\{y_k\}) = \frac{1}{N} \sum_{i=1}^{N} V^k(y_j, \{y_k\}) \qquad \text{Donde } k = \{1, .., M\}, \tag{2.58}$$

$$V_{nc}(\mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} \left(\prod_{k=1}^{M} V^k(y_j, \{y_k\}) \right), \tag{2.59}$$

donde el término V_J es el Potencial de Información Conjunto o JIP (Joint Information Potential), V_j^k es el Potencial de Información Marginal Parcial o PMIP (Partial Marginal IP), V_{nc} es el Potencial de Información Cruzado no normalizado o UCIP (Unnormalized Cross IP) y V^k es un promedio de los PMIP para un indice k particular. La divergencia de Cauchy-Schwartz y Euclideana se definen como sigue:

$$D_{CS}(f, g) = log \left(\frac{\int f(x)^2 dx \int g(x)^2 dx}{\left(\int f(x) g(x) dx \right)^2} \right) \qquad D_{ED}(f, g) = \int (f(x) - g(x))^2 dx, \tag{2.60}$$

donde $f(\cdot)$ y $g(\cdot)$ son funciones de distribución de probabilidad.

Mientras la Entropía cuadrática se relaciona con la IP, la Información Mutua Cuadrática o QMI (Quadratic Mutual Information) se relaciona con CIP. La Información Mutua Cuadrática se puede utilizar para implementar criterios de optimización de máxima transferencia

de información y de entropía mínima. La Información Mutua Cuadrática puede ser definida utilizando las divergencias de Cauchy-Schwartz D_{CS} y Euclidiana D_{ED} de la forma siguiente:

$$\begin{aligned} I_{CS}(\mathbf{y}) &= D_{CS}\left(f_Y y_1, .., y_M, \prod_{i=1}^{M} f_Y(y_i)\right) \\ I_{ED}(\mathbf{y}) &= D_{ED}\left(f_Y y_1, .., y_M, \prod_{i=1}^{M} f_Y(y_i)\right), \end{aligned} \tag{2.61}$$

a su vez las divergencias pueden ser estimadas a partir de la definición de CIP:

$$\begin{aligned} \hat{I}_{CS}(\mathbf{y}) &= \hat{D}_{CS}\left(f_Y y_1, .., y_M, \prod_{i=1}^{M} f_Y(y_i)\right) = -log(V_c(\mathbf{y})) \\ &= log(V_J(\mathbf{y})) - 2log(V_{nc}(\mathbf{y})) + log(V_M(\mathbf{y})) = -log(V_c(\mathbf{y})), \\ \hat{I}_{ED}(\mathbf{y}) &= \hat{D}_{ED}\left(f_Y y_1, .., y_M, \prod_{i=1}^{M} f_Y(y_i)\right) = V_{ED}(\mathbf{y}) \\ &= V_J(\mathbf{y}) - 2V_{nc}(\mathbf{y}) + V_M(\mathbf{y}), \end{aligned} \tag{2.62}$$

donde $V_M(\mathbf{y}) = \prod_{k=1}^{M} V^k$, $V_c(\cdot)$ y $V_{ED}(\cdot)$ son los Potenciales de Información Cruzados generados a partir de las distancias de Cauchy-Schwarz y Euclidiana respectivamente. Es importante destacar que los elementos de $\mathbf{y}$ pueden ser variables aleatorias multi-dimensionales y además estas dimensiones pueden ser diferentes entre ellas.

CIP puede ser utilizado en procesos de entrenamiento donde se busca transferir la máxima información entre la entrada y la salida de un modelo, esto se logra maximizando la información mutua o minimizando CIP. Este criterio puede ser usado en el paradigma No Supervisado debido a que la Información Mutua no necesita de un conocimiento previo de la salida. En el caso del paradigma de Aprendizaje Supervisado también es posible que se maximice la información mutua entre la salida del modelo y la salida deseada [49].

2.6.5. Otros Criterios ITL para el aprendizaje

Como se mencionó en un principio de la sección 2.6, ITL sustituye las medidas estadísticas típicas de la Teoría de la Información. Los estimadores ITL son más robustos a ruido y *outliers* en los datos. Esto último se debe a que en ITL el criterio de ajuste considera una estimación de la función de distribución de probabilidad o pdf (Probability Distribution Function) de las muestras y no solo el error individual de éstas.

En el paradigma de Aprendizaje Supervisado un criterio de optimización que permite ajustar parámetros de un modelo es la Minimización de la Entropía del Error o MEE (Minimization of the Error Entropy). Este criterio nace de la utilización de la Entropía de Renyi cuadrática estimada a partir del Potencial de Información:

$$\theta^* = \arg\min_{\theta} H_{R2}(e) = \arg\max_{\theta} V(e), \tag{2.63}$$

donde θ es el conjunto de parámetros del modelo que se desea ajustar en el entrenamiento y e es el error entre la salida deseada y la salida predicha por el modelo. Este criterio de optimización permite no solo capturar la interacción individual del error de las muestras sino también el comportamiento estadístico completo del error. Minimizar la entropía del error implica una reducción de la incerteza del error de forma que toda la información de las muestras está contenida en los parámetros θ del modelo.

En la práctica MEE busca que la función de densidad de probabilidad se transforme en una función delta. Un problema del MEE es que se genera un sesgo en el error debido a que la entropía es invariante a la media de la variable aleatoria [50], por tanto, aunque la incerteza del error se reduzca, éste puede no estar centrada en cero como es el objetivo, una solución simple es agregar el sesgo a la salida del modelo para corregir el problema [51].

Otro problema de MEE y compartido por otros estimadores en ITL es que genera una carga computacional mayor a otros criterio típicos como MSE, mientras que con MSE el orden de complejidad es de $O(N)$ en MEE es $O(N^2)$ (donde N es la cantidad de muestras) debido que es necesario computar todas las interacciones entre las muestras, aunque hay aproximaciones como la transformada rápida de Gauss [52].

Otro criterio importante en ITL es Maximizar la Correntropía o MCC (Maximum Correntropy Criterion). Este criterio busca aumentar del valor de la pdf del error en 0 menos fuerte que MEE. MCC no tiene el problema del sesgo y por otro lado su tiempo de computo es menor dado que solo necesita el error de las muestras individuales, lo que implica que utiliza menos información que MEE. La definición del criterio MCC se encuentra expresada en la ecuación siguiente:

$$\theta^* = \arg\operatorname*{máx}_{\theta} \hat{p}_E(e) = \arg\operatorname*{máx}_{\theta} \frac{1}{N}\sum_{i=1}^{N} G(e - e_i, \sigma). \tag{2.64}$$

Capítulo 3

Diseño e Implementación

En la primera parte de este capítulo es una introducción al entrenamiento de Ensambles, donde se presentan algunos detalles importantes relacionados con el Aprendizaje supervisado, poniendo énfasis en los algoritmos utilizados en esta memoria. Por otra parte, se discute las diferencias existentes al momento de generar diversidad en problemas de clasificación y regresión. Bajo el contexto anterior se incluye una sección que presenta como relacionar la diversidad en Ensambles y la Teoría de la Información. Todo esto con el fin de servir de introducción al método de entrenamiento propuesto.

También en esta primera parte se presenta el método de entrenamiento de Ensambles NCL el que se utilizará en el capítulo de análisis como método de comparación. Se presenta el método NCL debido a que tanto este como la propuesta de este trabajo intentan favorecer la diversidad a partir de un término de regularización.

En la segunda parte de este capítulo se presenta la propuesta del método de entrenamiento de Ensambles utilizando criterios de ITL específicamente el Potencial de Información Cruzado o CIP. El método propuesto busca favorecer la diversidad a partir de un término de regularización basado en CIP. En esta parte se incluye una primera comparación entre NCL y CIP para poder revisar las diferencias y ventajas claves que genera la inclusión de elementos de Teoría de la Información.

Al finalizar el capítulo se incluye una descripción general de la librería DeepEnsemble construida durante este trabajo. Esta librería fue desarrollada con la finalidad de generar un entrenamiento en paralelo de diferentes modelos neuronales. Esta librería fue desarrollada en el lenguaje de programación Python además utiliza como librería fundamental Theano a fin de poder obtener de forma fácil los distintos gradientes. Además, la librería Theano permite compilar el código para poder utilizar aceleración por hardware (GPU).

3.1. Entrenamiento de Ensambles y Diversidad

El entrenamiento de los ensambles depende de la estructura y de la forma de cómo se favorece la diversidad (ver sección 2.5), en cuanto a las estructuras estas dependen de cómo es tratada la información de entrada a los modelos y como son procesadas para obtener el resultado final. La estructura base de Ensamble utilizada en ente trabajo es la que se puede ver en la figura 3.1 la cual es una estructura paralela, aunque también existen otras estructuras como árboles o grafos (Random Forest, Stacking, etc.), pero que no son consideradas para simplificar el trabajo.

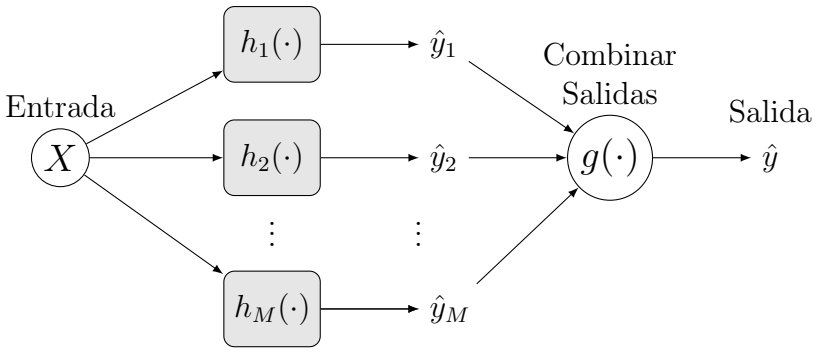


Figura 3.1: Estructura general de un Ensamble.

La estructura de un Ensamble define un orden secuencial en el que se deben entrenar los modelos, por ejemplo en Stacking (ver sección 2.4.3). Es necesario entrenar los modelos del nivel-0 antes que el de nivel-1, dado que este último necesita como entrada las salidas de los modelos anteriores. Por otro lado también existen métodos que requieren que los modelos sean entrenados de forma paralela, debido a que necesitan información de los demás modelos para llevar a cabo su entrenamiento, e.g. NCL (ver sección 3.1.1).

Un elemento importante en la estructura de un Ensamble es la función que combina las distintas salidas de los modelos que componen el Ensamble. Pero, aunque esta función afecta directamente a la función objetivo del entrenamiento varios estudios han demostrado de forma empírica que la elección de esta función no afecta de forma significativa los resultados de predicción. En el trabajo de Kuncheva del año 2002 [53] se compararon 6 funciones de combinación distintos: promedio, mínimo, máximo, mediana, votación máximo y oracle. Los resultados arrojaron que si la probabilidad a posteriori de clasificación se distribuye de forma uniforme los métodos de mínimo y máximo lograban un mejor desempeño, pero si se distribuyen de forma normal todos los métodos presentan el mismo desempeño. El mismo año Kuncheva probó empíricamente que no existe una clara preferencia sobre el método de combinación [54], además se demostró de forma experimental que el método de combinación basado en pesos (ver ecuación (2.20)) muestra un mejor desempeño respecto a la media especialmente en problemas donde las clases están no balanceadas [55]. Visto lo anterior queda claro que el desempeño de un Ensamble bajo una función de combinación específica depende de las muestras y por tanto no es posible establecer una clara preferencia.

En cuanto a la diversidad algunos métodos como Bagging y Boosting (ver secciones 2.4.2 y 2.4.1 respectivamente) la favorecen a partir de la manipulación del conjunto de entrenamiento, otros métodos manipulan la descripción de la salida de los modelos (ECOC [21]) y otros

utilizan funciones de regularización para modificar el recorrido del espacio de hipótesis (NCL). En todos los casos anteriores al intentar favorecer la diversidad se altera la forma de entrenar a los modelos.

Los métodos que buscan favorecer la diversidad a partir de la manipulación de los datos de entrada y salida lo hacen de forma implícita debido a que su objetivo no es optimizar directamente una métrica relacionada con la diversidad. En este caso la diversidad se alcanza debido a que los modelos resultantes tienden a ser diferentes entre sí porque fueron generados con muestras diferentes.

En el caso de manipulación del recorrido del espacio de hipótesis, la diversidad puede ser favorecida de forma explícita a partir de una métrica que durante el proceso de entrenamiento es optimizada. Esta métrica no es única debido a que como se mencionó en la sección 2.5 no existe una definición formal de la diversidad y por tanto es difícil generar una métrica general que sirva para todos los problemas.

Las métricas que intentan favorecer la diversidad y al mismo tiempo resolver un problema de Aprendizaje Supervisado deben enfrentar el dilema entre precisión y diversidad. Mientras que la diversidad busca que los aportes de los modelos del Ensamble sean diferentes entre sí el objetivo de la precisión es la minimización del error de predicción lo que genera que los modelos ya que tienden a ser similares en sus salidas, este dilema todavía es un problema abierto y que es de profunda investigación [8, 9].

Una dificultad importante al momento de favorecer la diversidad en Ensambles se produce en el caso de los problemas de clasificación, mientras que en problemas de regresión existen métricas muy bien fundamentadas que permiten en cierta forma medir la diversidad (ver sección 2.4.4) en el caso de la clasificación esto no existe. Esta diferencia entre regresión y clasificación se debe a que en los problemas de regresión la salida es continua y por tanto ordinal, es decir se puede definir una relación de orden entre 2 salidas, mientras que en el caso de los problemas de clasificación la salida de los modelos son discretas y no ordinales dado que no existe una relación de orden implícita entre las distintas clases [9, p. 33]. Por tanto el error de predicción y la función objetivo dependen fuertemente de la forma en como son representadas las clases, lo que a su vez genera que no se pueda obtener una expresión general de la diversidad como en el caso de la regresión [16]. Tumer y Ghosh [56] relacionaron el error de clasificación con las correlaciones entre las salidas de los predictores individuales. Este trabajo demostró que era posible descomponer el error de un conjunto de modelos combinados linealmente en componentes de precisión y diversidad. Sin embargo, en trabajos posteriores resulto imposible generar un error del Ensamble donde se utilice como método de combinación la mayoría de votos (ver ecuación (2.26)) y que esta se descomponga de manera similar en términos de precisión y diversidad aditivas. Esto último se explica porque Tumer y Ghosh utilizaron una combinación lineal sobre una función objetivo MSE mientras que las funciones típicas de combinación de clasificación resultan ser no lineales (por ejemplo, la mayoría de votos) y la función de costo es una función de cero-uno (es o no es de una clase determinada) por tanto resulta más complicado en estos casos obtener una relación de diversidad y error.

La dificultad de poder establecer una definición de diversidad sobre los problemas de clasificación ha generado el nacimiento de múltiples métricas que intentan cuantificar la

diversidad (ver tabla 2.3), sin embargo, cada una tiene sus limitaciones y no es posible decidir cuál de todas presenta un mejor desempeño para todo tipo de problema. Resumiendo lo dicho anteriormente en el Aprendizaje con Ensambles mientras los métodos de regresión presentan una forma fácil de interpretar la diversidad y por tanto favorecerla, los métodos de clasificación aún presentan dificultades a la hora de interpretar la diversidad y por tanto es necesario hacer un análisis en cada caso al momento de entrenar de forma de favorecer la diversidad.

3.1.1. Aprendizaje con Correlación Negativa (NCL)

Impuesto por Liu junto al Profesor Xin Yao en 1999 [10] es un método de aprendizaje basado en Correlación Negativa, conocido como NCL por sus siglas en inglés (Negative Correlation Learning). NCL busca fomentar la diversidad en un Ensamble incluyendo un término de regularización [33] en la función objetivo,

$$\begin{aligned} p_i &= (h_i - H)\sum_{j\neq i}^{M}(h_j - H); \quad \text{con } H = \frac{1}{M}\sum_{i=1}^{M} h_i \\ &= (h_i - H)\cdot -(h_i - H) \\ &= -(h_i - H)^2 \end{aligned} \tag{3.1}$$

donde p_i corresponde a la correlación negativa del modelo h_i respecto a los demás modelos del Ensamble ($\{h_{j\neq i} \ : \ \forall j \in \{1,..,M\}\}$) y H corresponde a la salida total del Ensamble. El término p_i es similar al término ambiguo (ver ecuación (2.29)) de la descomposición Ambigua propuesta por Krogh y Vedelsby [11].

El término p_i se agrega a la función de costo MSE (Minimum Square Error) de cada red de forma que la función de costo final asociada al i-ésimo modelo del Ensamble queda como:

$$\begin{aligned} J_i(X) &= \sum_{k=1}^{N}(h_i(x_k) - y)^2 + \lambda p_i \\ &= \sum_{k=1}^{N}(h_i(x_k) - y)^2 - \lambda\sum_{k=1}^{N}(h_i(x_k) - H(x_k))^2, \end{aligned} \tag{3.2}$$

donde la constante λ permite controlar el efecto del término de correlación negativa sobre la función de costo total. En el primer trabajo donde se presentó el método NCL [10] se propuso que la constante λ debe ser un número positivo entre 0 y 1 sin embargo no se presentó ningún respaldo a tal afirmación. Posteriormente Gavin Brown planteó límites para λ fundamentado en criterios de optimización (signo de la segunda derivada de la función de costo) [57]. Estos límites restringen el valor de λ a un valor no mayor a $\lambda_{upper} = \frac{M}{M-1}$ donde M es el número de modelos en el Ensamble, al revisar este límite se puede ver que a medida que crece el número de modelos λ_{upper} tiende a 1.

Como ya se mencionó NCL busca favorecer la diversidad a partir de maximizar la correlación negativa de las salidas de los modelos. Esto en principio parece razonable pero al igual que en el problema del dilema de sesgo y varianza que dice que al disminuir el sesgo asociado

a la precisión de predicción la varianza aumenta, al aumentar la correlación negativa entre los modelos aumenta la complejidad del Ensamble y por tanto su diversidad (aumento de la varianza a partir del término ambiguo p_i). Aquello tiene como consecuencia que el sesgo del Ensamble disminuye pero también se vuelve más sensible a las muestras de entrenamiento y por tanto se produce sobreajuste. En NCL no existe un mecanismo que dé cuenta de forma implícita del compromiso entre sesgo y varianza, aunque a partir de la constante λ se puede regular la acción del término p_i.

Aunque en un principio NCL fue propuesto con solo un término de regularización posteriormente el mismo profesor Xin Yao quien es co-autor de la propuesta de NCL presenta un trabajo junto con Huanhuan Chen que incluye un término de regularización con la norma $L2$ (ver ecuación (2.9)), este último método se llama Regularized Negative Correlation Learning o RNCL [23], el que mejora los problemas que tiene NCL con el sobreajuste durante el entrenamiento y por tanto lo vuelve más robusto al ruido. La inclusión del término con la norma $L2$ sobre los parámetros del modelo evita que se produzca sobreajuste y regula la complejidad de los modelos generados.

Dada la forma que tiene el término de regularización propuesto por NCL, el entrenamiento de los modelos del Ensamble debe ser realizado de forma paralela, lo que permite manejar de forma explícita la diversidad entre los modelos, algo que no es posible con otros métodos.

3.1.2. Diversidad y Teoría de la Información

En clasificación con Ensambles uno de los problemas abiertos más interesante es poder cuantificar la diversidad en términos generales dado que depende fuertemente de la función de costos y de la función que combina los distintos modelos. Lo anterior se debe a que estas funciones definen la forma como se cuantifica el error y como intervienen los modelos en el resultado final del Ensamble. Aunque no existe una definición formal de la diversidad si existen variadas métricas para medirla en problemas de clasificación (ver tabla 2.3). La mayoría de estas intenta extraer información estadística sobre las salidas de los modelos de forma de independizarse de la estructura del Ensamble.

Gavin Brown en 2009 propuso ver el problema de la diversidad en los Ensambles como un problema desde el punto de vista de la Teoría de la Información [16]. Para Brown el objetivo de aprendizaje en un Ensamble se puede ver como un problema de decodificar un mensaje Y a partir de un conjunto de señales de entrada $X_{1:M}$, las que en un Ensamble representarían las salidas de los modelos. Para decodificar se utiliza la función $g(\cdot)$ que corresponde a la función que combina las salidas de los modelos. A partir de las definiciones anteriores la estimación del mensaje es $\hat{Y} = g(X_{1:M})$ en donde las variables $X_{1:M}$ se consideran aleatorias y por tanto el objetivo de aprendizaje se puede expresar como un problema de minimización de la probabilidad de error de estimación, es decir:

$$\min_{\theta} p(\hat{Y} \neq Y), \tag{3.3}$$

donde θ representa los parámetros de los modelos. En el mismo trabajo Brown propuso límites para esta probabilidad de error [16] basado en las inecuaciones de Fano, 1961 [58] y

Hellman-Raviv 1970 [59] tal como se muestra en la ecuación siguiente (3.4):

$$\underbrace{\frac{H(Y) - I(X_{1:M}; Y) - 1}{log(|Y|)}}_{\text{(Fano, 1961)}} \leq p(g(X_{1:M}) \neq Y) \leq \underbrace{\frac{H(Y) - I(X_{1:M}; Y)}{2}}_{\text{(Hellman-Raviv, 1970)}}, \tag{3.4}$$

donde $H(\cdot)$ es la Entropía de Shannon (Ecuación (2.33)) e $I(X_{1:M}; Y)$ es la Información Mutua entre las variables aleatorias $X_{1:M}$ e Y (Ecuación (2.34)). Tanto la inecuación de Fano como Hellman-Raviv dependen de la Información Mutua (Ecuación (2.40)) por tanto si se quiere minimizar la probabilidad de error es necesario maximizar $I(X_{1:M}; Y)$.

Es importante destacar que la probabilidad de error depende en última instancia de la función $g(\cdot)$ por tanto si se quiere reducir el error también es importante diseñar convenientemente esta función.

Una expansión para el $I(X_{1:M}; Y)$ es la que se muestra a continuación [60]:

$$I(X_{1:M}; Y) = \sum_{T \subseteq S} I(\{T \cup Y\}), \qquad |T| \geq 1, \tag{3.5}$$

donde $S = X_1, .., X_M$ es el conjunto de modelos del Ensamble e Y es la salida esperada. La operación $\sum_{T \subseteq S}$ significa que se deben sumar todos los posibles subconjuntos T generados a partir del conjunto S. La ecuación (3.5) se puede reordenar de la siguiente forma:

$$\begin{aligned} I(X_{1:M}; Y) = & \overbrace{\sum_{i \in \{1,..,M\}} I(\{X_i, Y\})}^{\text{Relevancia}} - \overbrace{\sum_{|X|=2} I(\{X\})}^{\text{Redundancia}} + \overbrace{\sum_{|X|=2} I(\{X\}|Y)}^{\text{Sinergia}} \\ & - \sum_{|X|=3} I(\{X\}) + \sum_{|X|=3} I(\{X\}|Y) \\ & - \quad ... \\ & - \sum_{|X|=M} I(\{X\}) + \sum_{|X|=M} I(\{X\}|Y). \end{aligned} \tag{3.6}$$

Esta descomposición de la Información Mutua da lugar a 3 expresiones que representan conceptos importantes: relevancia [61, 62] redundancia [63, 64] y complementariedad o sinergia [61, 65].

La relevancia es una medida de la información que tienen los modelos respecto a la salida deseada. Esta puede ser asociada con la precisión, aunque no sea exactamente aquello [16].

La redundancia es una medida de la dependencia entre los distintos modelos del ensamble. Puede ser asociada a la diversidad debido a que solo considera la interacción entre las salidas de los modelos.

La sinergia o complementariedad es una especie de redundancia condicional que da cuenta de cómo la suma de los modelos pueden entregar mayor información acerca de la salida esperada si solo se consideran en forma individual [60]. En el trabajo de Zhou en 2010b se relacionó la diversidad directamente con los términos de redundancia y sinergia [66].

Trabajar con la ecuación (3.6) no resulta conveniente debido a la cantidad de cálculos necesarios que aumentan de forma exponencial por cada modelo extra. Sin embargo, se puede realizar una aproximación usando solo los primeros términos que cumplan $|T| \leq 2$, es decir:

$$I(X_{1:M}; Y) \approx \sum_{i=1}^{M} I(X_i; Y) + \sum_{j=1}^{M} \sum_{k=j+1}^{M} I(\{X_j, X_k, Y\}), \tag{3.7}$$

donde el segundo término se puede descomponer a su vez en 2 expresiones tal como en el caso de la ecuación (3.6) para obtener los términos de redundancia y sinergia. El resultado de esta descomposición puede ser expresado tal como sigue:

$$I(X_{1:M}; Y) \approx \sum_{i=1}^{M} J_m, \tag{3.8}$$

donde

$$J_m = \underbrace{I(X_m; Y)}_{\text{Relevancia}} - \underbrace{\sum_{k=m+1}^{M} I(X_m; X_k)}_{\text{Redundancia}} + \underbrace{\sum_{k=m+1}^{M} I(X_m; X_k|Y)}_{\text{Sinergia}}. \tag{3.9}$$

En la ecuación (3.9) se pueden ver los términos de relevancia, redundancia y sinergia asociados al m-ésimo modelo respectivamente. Esta descomposición es la contribución del modelo m-ésimo a la aproximación con $|T| \leq 2$ de la información mutua (ver ecuación (3.7)).

En la definición de J_m se puede incluir 2 constantes que permitan manejar el efecto de la redundancia y sinergia, de la misma forma que Brown define la función objetivo First Order Utility [60]:

$$J_m = \underbrace{I(X_m; Y)}_{\text{Relevancia}} - \beta \underbrace{\sum_{k=m+1}^{M} I(X_m; X_k)}_{\text{Redundancia}} + \lambda \underbrace{\sum_{k=m+1}^{M} I(X_m; X_k|Y)}_{\text{Sinergia}}. \tag{3.10}$$

Brown utilizó la ecuación (3.10) como una generalización de diversos métodos de selección de características que utilizan la información mutua.

3.2. Método de Entrenamiento Propuesto: CIPL

El método CIPL parte desde la ecuación (3.10) propuesta por Brown en su trabajo "*A New Perspective for Information Theoretic Feature Selection*" [60]. Sin embargo, en esta memoria es aplicado al entrenamiento de ensambles.

La dificultad de utilizar directamente la información mutua en (3.10), es que en problemas prácticos no se cuenta con la información estadística de los datos (e.g. función de probabilidad), por tanto, es necesario estimarla de forma empírica.

Aunque existen diversas formas de estimar la información mutua, en este trabajo se propone reemplazarla por la información mutua cuadrática (ver ecuación (2.62)) basadas en las divergencias de Cauchy-Schwartz D_{CS} y Euclidiana D_{ED}. La razón del reemplazo, es que es posible estimar QMI directamente desde los datos, además permite trabajar con variables aleatorias de diferentes dimensiones entre sí.

El reemplazo de la información mutua por QMI es una propuesta heurística, dado que no existen pruebas que permitan asegurar que las relaciones obtenidas con la información mutua se cumplan con QMI [7, p. 52]. Sin embargo, la relación de orden respecto a la información se mantiene [67, 68]. Por lo tanto, aunque con QMI se puede asegurar la maximización de la información entre pares de variables aleatorias, eso no significa que el resultado sea similar al obtenido con la ecuación (2.62), la cual utiliza la Información Mutua.

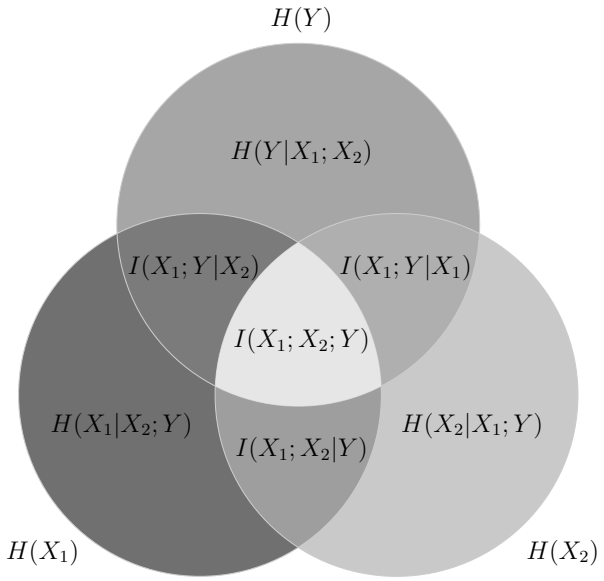


Figura 3.2: Diagrama de Venn de la Información Mutua entre 3 variables aleatorias.

Antes de estimar QMI con el Potencial de Información Cruzado, es necesario expresar los términos la sinergia de la ecuación (2.62) como $I(X_1;X_2|Y) = I(X_1;X_2) - I(X_1;X_2;Y)$ (ver figura 3.2), de forma que la ecuación queda como

$$J_m = I(X_m;Y) - \beta \sum_{k=m+1}^{M} I(X_m;X_k) + \lambda \sum_{k=m+1}^{M} I(X_m;X_k) - I(X_m;X_k;Y). \tag{3.11}$$

En la ecuación (3.11) es importante destacar que si β y λ son iguales, entonces tanto la Sinergia como la Redundancia pueden ser mezcladas en un solo término, pero si se trabajara directamente con este término se perdería la posibilidad de estudiar el efecto de la Sinergia y la Redundancia por separado.

Para estimar QMI se utiliza el Potencial de Información Cruzado junto con las divergencias de Cauchy-Schwartz y Euclidiana, de forma que la función de costo heurística propuesta queda como

$$\begin{aligned} J_m^{CS} &= I_{CS}(X_m;Y) - \beta \sum_{k=m+1}^{M} I_{CS}(X_m;X_k) + \lambda \sum_{k=m+1}^{M} I_{CS}(X_m;X_k) - I_{CS}(X_m;X_k;Y) \\ &\approx -log(V_{CS}(X_m;Y)) + \beta \sum_{k=m+1}^{M} log(V_{CS}(X_m;X_k)) - \lambda \sum_{k=m+1}^{M} log\left(\frac{V_{CS}(X_m;X_k)}{V_{CS}(X_m;X_k;Y)}\right) \end{aligned} \tag{3.12}$$

$$\begin{aligned} J_m^{ED} &= I_{ED}(X_m;Y) - \beta \sum_{k=m+1}^{M} I_{ED}(X_m;X_k) + \lambda \sum_{k=m+1}^{M} I_{ED}(X_m;X_k) - I_{ED}(X_m;X_k;Y) \\ &\approx V_{ED}(X_m;Y)) - \beta \sum_{k=m+1}^{M} V_{ED}(X_m;X_k) + \lambda \sum_{k=m+1}^{M} V_{ED}(X_m;X_k) - V_{ED}(X_m;X_k;Y) \end{aligned} \tag{3.13}$$

donde V_{CS} y V_{ED} son los Potenciales de Información Cruzado de las divergencias de Cauchy-Schwartz y Euclidiana respectivamente. Es importante volver a recordar que las ecuaciones (3.12) y (3.13) nacen de una propuesta heurística y por tanto es necesario probar su validez de forma empírica.

Para simplificar la notación de las funciones de costo J_m, J_m^{CS} y J_m^{ED} se definen los siguientes términos:

Para MI Shannon J_m:

$$\begin{aligned} C_m(X_m,Y) &= I(h_m(\{d\},\theta_m);\{y\})) \\ R_m(X_{1:M}) &= \sum_{k=m+1}^{M} I(h_m(\{d\},\theta_m);h_k(\{d\},\theta_k)) \\ S_m(X_{1:M},Y) &= \sum_{k=m+1}^{M} I(h_m(\{d\},\theta_m);h_k(\{d\},\theta_k)) - I(h_m(\{d\},\theta_m);h_k(\{d\},\theta_k);\{y\}) \end{aligned} \tag{3.14}$$

Para QMI Cauchy-Schwarz J_m^{CS}:

$$\begin{aligned} C_m^{CS}(X_m,Y) &= -log(V_{CS}(h_m(\{d\},\theta_m);\{y\}))) \\ R_m^{CS}(X_{1:M}) &= -\sum_{k=m+1}^{M} log(V_{CS}(h_m(\{d\},\theta_m);h_k(\{d\},\theta_k))) \\ S_m^{CS}(X_{1:M},Y) &= -\sum_{k=m+1}^{M} log\left(\frac{V_{CS}(h_m(\{d\},\theta_m);h_k(\{d\},\theta_k)))}{V_{CS}(h_m(\{d\},\theta_m);h_k(\{d\},\theta_k);\{y\})}\right) \end{aligned} \tag{3.15}$$

Para QMI Euclidiana J_m^{ED}:

$$
\begin{aligned}
C_m^{ED}(X_m, Y) &= V_{ED}(h_m(\{d\}, \theta_m); \{y\})) \\
R_m^{ED}(X_{1:M}) &= \sum_{k=m+1}^{M} V_{ED}(h_m(\{d\}, \theta_m); h_k(\{d\}, \theta_k)) \\
S_m^{ED}(X_{1:M}, Y) &= \sum_{k=m+1}^{M} V_{ED}(h_m(\{d\}, \theta_m); h_k(\{d\}, \theta_k)) - V_{ED}(h_m(\{d\}, \theta_m); h_k(\{d\}, \theta_k); \{y\}),
\end{aligned}
\tag{3.16}
$$

donde el índice m indica que se está trabajando con el modelo m-ésimo, los conjuntos $\{d\}$ y $\{y\}$ indican a los conjuntos de entrenamiento de entrada y salida respectivamente, el término $h_m(\{d\}, \theta_m)$ representa la salida o predicción del modelo m-ésimo con parámetros θ_m y evaluado en el conjunto de entrenamiento $\{d\}$. Los términos $C_m(X_m, Y)$, $R_m(X_{1:M})$ y $S_m(X_{1:M}, Y)$ representan la aproximación de la relevancia, redundancia y sinergia respectivamente, en el caso de Cauchy-Schwartz y Euclidiana no necesariamente representan específicamente estos términos.

El término $C_m(X_m, Y)$ está asociado a la relevancia que visto desde un punto de vista práctico se relaciona con la precisión de predicción del modelo m-ésimo dado que busca maximizar la la información mutua entre el modelo y la salida deseada.

El término $R_m(X_{1:M})$ está asociado a la redundancia que dada su construcción intenta reducir la información mutua entre pares de salidas de modelos, este término se puede relacionar con el concepto de diversidad. El término $S_m(X_{1:M}, Y)$ asociado a la sinergia es la redundancia condicional que intenta minimizar la relación redundante entre pares de modelos pero considerando también la salida deseada Y.

El objetivo del aprendizaje para el modelo m-ésimo del Ensamble queda expresado como

$$
\begin{aligned}
\theta_m^* &= \arg\min_{\theta} \; J_m^{CIP} \\
&= \arg\min_{\theta} \; -C_m(X_m, Y) + \beta R_m(X_{1:M}; Y) - \lambda S_m(X_{1:M}, Y),
\end{aligned}
\tag{3.17}
$$

donde J_m^{CIP} puede representar a las funciones de costo J_m^{CS} o J_m^{ED}. El método de entrenamiento propuesto a partir de esta función de costo se le identifica a partir de ahora en este trabajo como el nombre CIPL de las siglas en ingles Cross Information Potential Learning, para hacer referencia a este más fácilmente. La descripción en detalle del método completo se puede ver en el algoritmo 8.

CIPL requiere de 5 hiperparámetros, 2 de los cuales están asociados al número de épocas: L y L_0, y los otros 3 asociados a la función de costo: β, λ y σ. Tanto β como λ permiten sintonizar el efecto de la redundancia y la sinergia en el costo final. El parámetro σ es el tamaño del kernel Gaussiano utilizado en el cálculo de CIP, el valor de este parámetro puede ser obtenido a partir de la regla de Silverman (2.47).

Un detalle importante a destacar al revisar el algoritmo 8 de CIPL es la inclusión de una etapa de entrenamiento con MSE durante las primeras épocas, esto se debe a que CIP es invariante a la media de los datos. Por tanto si durante las primeras épocas del entrenamiento

Algoritmo 8 Entrenamiento Ensamble con CIPL

Entrada: M Número de modelos.
$h = \{h_1, .., h_M\}$ Conjunto de modelos.
L Número total de épocas de entrenamiento con CIP.
$L_0 < L$ Número de épocas de entrenamiento con MSE.
$d \subseteq \mathcal{D}$ Conjunto de entrenamiento.
$g(\cdot)$ Función para combinar los modelos.
β, λ, σ Hiperparámetros.
Salida: Se devuelve el ensamble $H_{CIP}(d)$.

```
Inicializar el Ensamble H_CIP(d) = g(h_1(d), h_2(d), .., h_M(d))
para n = 1 hasta L hacer
    si n < L_0 entonces
        Entrenar el ensamble H_CIP con la función objetivo basada en MSE.
    si no
        Entrenar el ensamble H_CIP con la función objetivo basada en CIP.
    fin si
fin para
devolver H_CIP(d) = g(h_1(d), h_2(d), .., h_M(d))
```

se producen sesgos CIP no es capaz de corregirlos por sí solo, por esta razón se incluye un pre-entrenamiento con RMS, existen otras técnicas para corregir el sesgo como por ejemplo restar el sesgo a la salida del modelo, sin embargo, se optó por esta estrategia para no complicar la implementación de los modelos. En cuanto a la utilización de MSE solo responde a la simplicidad, sin embargo es posible utilizar cualquier función de costo que permita alcanzar un grado de convergencia suficiente para evitar los problemas de sesgo producidos por CIP.

CIPL puede ser utilizado tanto en problemas de regresión como de clasificación, no existe restricción respecto a los datos de entrada o salida. Por otro lado, debido a que el método solo define la función de costo tampoco existe restricciones sobre los modelos que pueden ser utilizados en el Ensamble, sin embargo en este trabajo solo se utilizan redes neuronales dado los alcances de la memoria.

Haciendo un símil entre CIPL y NCL se puede decir que el término C_m se parece al término MSE debido a que al minimizarlo se consigue que aumente la precisión de predicción de los modelos individuales, en cuanto a los términos R_m y S_m se asemejan al término de regularización p_i de NCL, dado que estos términos permiten controlar la diversidad [66]. Es importante destacar que CIPL al igual que el método NCL necesita que los modelos sean entrenados de forma simultánea debido a que la función de costo J_m^{CIP} necesita de la información de las salidas de los otros modelos para sus cálculos.

3.2.1. Annealing

A modo de mejorar la convergencia del algoritmo CIP se puede utilizar un método de actualización tipo annealing para el tamaño del kernel Gaussiano durante el proceso de entrenamiento. La idea es variar el tamaño del kernel desde un valor por sobre el óptimo (Silverman, (2.47)) y luego disminuirlo hasta un valor menor a este. Esto permite a medida que se avanza en el proceso de entrenamiento que la estimación generada por CIP sea más sensible a los errores de estimación y se logre un mejor ajuste de optimización. En un principio la búsqueda del óptimo es más gruesa y a medida que se avanza en las iteraciones del entrenamiento la búsqueda es cada vez más fina.

En la ecuación (3.18) se puede ver una función de actualización del tamaño del kernel o annealing:

$$\sigma_i = \sigma^+ \cdot \left(\frac{\sigma^-}{\sigma^+}\right)^{i/i_{max}} \tag{3.18}$$

donde σ^+ y σ^- son los valores superior e inferior del tamaño del kernel respectivamente, i es el valor de la iteración actual dentro del entrenamiento, i_{max} es el total de iteraciones que se deben realizar y finalmente σ_i es el valor del tamaño del kernel obtenido con annealing. Este valor es utilizado para calcular las funciones de costo definidas en CIP. Los valores típicos para σ^+ y σ^- son 1.5 y 0.5 veces el óptimo de Silverman (ver sección (2.47)) [69, p. 52].

3.2.2. Diversidad en CIPL

En CIPL los términos que se pueden asociar a la diversidad son la redundancia $R_m(X_{1:M})$ y la sinergia $S_m(X_{1:M}; Y)$. En el caso de la redundancia se busca minimizar la información mutua entre los pares de salidas de los modelos del Ensamble, lo que implica que estos sean menos redundantes debido a que aumenta la incertidumbre (entropía) de una salida sobre la otra. Lo anterior parece razonable para favorecer la diversidad, pero hay que considerar que los modelos intentan resolver el mismo problema, por tanto las salidas tienden a ser similares, por ejemplo, en el Aprendizaje Supervisado las salidas tiendan a ser redundantes de forma natural.

Aunque minimizar la redundancia favorece que los modelos sean diferentes, también puede perjudicar la precisión de los modelos (dilema diversidad-precisión), por tanto lo que se busca es que los modelos sean en parte redundantes y que las diferencias entre ellos produzcan sinergías.

Para evitar que al minimizar el término $R_m(X_{1:M})$ se perjudique la precisión, se utiliza una constante β que lo pre-multiplique con el fin de controlar su efecto en el resultado final del costo. En este trabajo no se propone una forma de escoger β mas que ensayo y error.

El término $S_m(X_{1:M}, Y)$ de sinergia busca maximizar la redundancia condicional entre los pares de salidas de los modelos. Esto implica que aumente la información mutua total, lo que es consecuente con el objetivo de disminuir la probabilidad de error de predicción. Pero a pesar de que lo anterior es correcto, también al maximizar este término y considerando modelos homogéneos en su estructura se produce una disminución de la diversidad. Maximizar la

sinergia tiene como efecto maximizar también la redundancia como se puede ver en la ecuación (3.20):

$$\sum_{k=m+1}^{M} I(X_m; X_k; Y) \leq \sum_{k=m+1}^{M} I(X_m; X_k) = R_m(X_{1:M}) \tag{3.19}$$

$$S_m(X_{1:M}, Y) \leq 2 \cdot R_m(X_{1:M}), \tag{3.20}$$

por tanto, al maximizar la sinergia se disminuye la diversidad generada por la minimización de la redundancia, sin embargo también permite aumentar la precisión, lo anterior da cuenta del compromiso entre redundancia y sinergia.

Del mismo modo que la redundancia, para poder manejar la acción del término $S_m(X_{1:M}, Y)$ al momento de entrenar los modelos, se propone pre-multiplicarlo por una constante llamada λ. En este trabajo no se propone una forma de escoger λ mas que ensayo y error.

3.2.3. Los Hiperparámetros de CIPL

CIPL cuenta con 4 parámetros extras que es necesario establecer antes de comenzar el entrenamiento, a continuación, se detalla cada uno de estos:

- β: Es una constante que regula el efecto de la redundancia entre los modelos. Minimizar la redundancia produce que la correlación entre las salidas de los modelos disminuya lo que es similar al efecto esperado cuando se quiere favorecer la diversidad. En este trabajo no se propone una forma de escoger β mas que ensayo y error.
- λ: Es una constante que regula el efecto de la redundancia condicional. Maximizar la redundancia condicional tiene como efecto mejorar la precisión en desmedro de la diversidad generada por la redundancia (ver ecuación (3.20)). En este trabajo no se propone una forma de escoger λ mas que ensayo y error.
- σ: Es el tamaño del kernel y dado que se trabaja con un kernel Gaussiano es posible utilizar la regla de Silverman (ver ecuación (2.47)). Si el valor de esta constante es muy grande el resultado de la estimación del Potencial de Información Cruzado se aproxima a operaciones sobre el error cuadrático entre las salidas de los modelos del Ensamble, aquello tiene como efecto que la estimación es menos sensible. En el caso de que σ sea muy pequeño la estimación de la función de distribución es más ruidosa y por tanto más sensible a las variaciones de los datos. Por todo lo anterior, se sugiere trabajar con valores cercanos a σ_S, obtenido con la regla de Silverman sobre la señal deseada Y. Si se utiliza annealing se sugiere un rango de valores entre 0.5 y 1.5 veces σ_S.
- L_o: Este parámetro indica cuantas épocas se requiere entrenar con MSE antes de pasar a utilizar la función de costo J_m^{CIP}, la idea es que los modelos del Ensamble alcancen un error de predicción pequeño de forma que luego el entrenamiento con CIPL no se generen sesgos propios de la estimación del Potencial de Información. Es importante destacar que se seleccionó a MSE como la primera función de costo durante el entrenamiento debido a que no tiene restricciones sobre la salida. Sin embargo, se puede utilizar cualquier función de costo que no tenga problemas de sesgo.

3.3. Diseño de librería DeepEnsemble

La librería DeepEnsamble fue desarrollada con el propósito de entrenar en forma paralela los distintos modelos de redes neuronales en un Ensamble. La librería fue desarrollada en el lenguaje de programación Python en su tercera versión. Además para el cálculo de las gradientes se utiliza la librería Theano [70] la cual también permite utilizar aceleración de hardware si se cuenta con una tarjeta de vídeo Nvidia, esto último es especialmente importante para reducir los tiempos de computo en problemas de Aprendizaje Profundo (Deep Learning).

La librería DeepEnsamble está orientada para trabajar con Ensambles de redes neuronales y en problemas de Aprendizaje Supervisado para ello se implementaron 3 algoritmos de entrenamiento: GD, SGD y SGD con mini-lotes (ver sección 2.3.1). También se agregaron distintos métodos para mejorar el desempeño de optimización: Adagrad, Adadelta, RMSProp, Nesterov y Momento (ver sección 2.3.5).

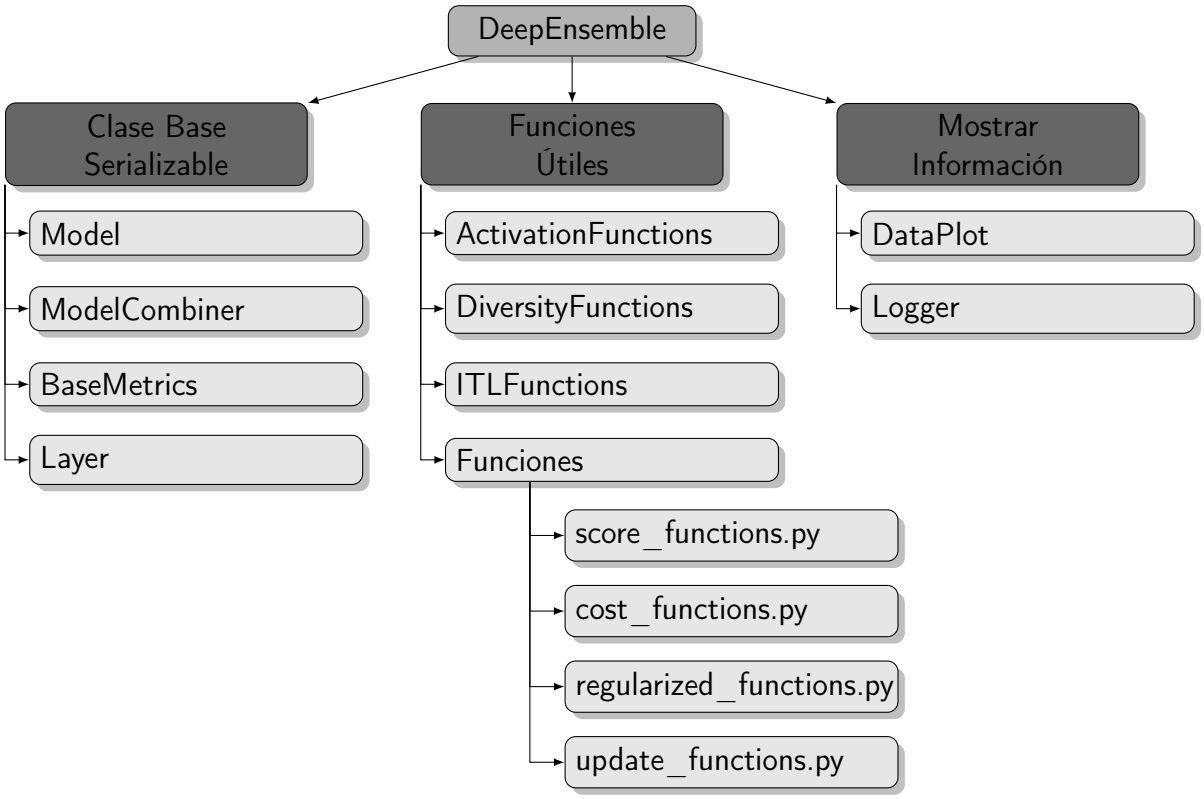


Figura 3.3: Diagrama de las clases y funciones útiles de la librería DeepEnsemble.

En la figura 3.3 se puede ver la estructura que tiene la librería DeepEnsemble. Todas las clases relacionadas con los modelos y métricas heredan de la clase *Serializable* la cual permite guardar y cargar datos desde el disco duro.

La clase *Model* es la clase base desde la que se heredan los modelos implementados: redes neuronales y Ensambles. Los modelos de redes neuronales son construidos de forma secuencial integrando diferentes tipos de capas, las diferentes capas implementadas heredan de *Layer* la clase base. *ModelCombiner* es la clase base de las funciones que permiten combinar las

salidas de diferentes modelos en un Ensamble. *BaseMetrics* es la clase base de las métricas calculadas en tiempo de entrenamiento.

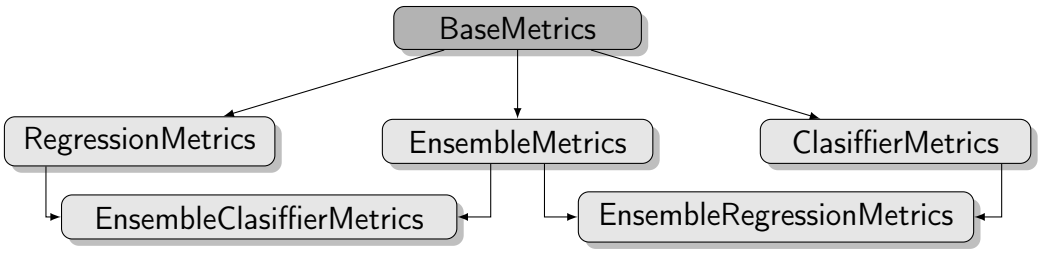


Figura 3.4: Diagrama de las clases de métricas de la librería DeepEnsemble.

Como se mencionó antes, la librería DeepEnsemble está orientada al entrenamiento de modelos bajo el paradigma de Aprendizaje Supervisado por tanto las métricas se centran solo en problemas de regresión y clasificación. Estas métricas están separadas en los casos donde los modelos sean Ensambles o no tal como se muestra en el diagrama de la figura 3.4. Lo anterior se debe a que existen métricas como la diversidad que solo aplican a Ensambles. Las clases de métricas guardan información acerca de los costos en cada época del entrenamiento junto con información del desempeño, en el caso de los Ensambles es capaz de guardar información por separado tanto del Ensamble total como de los modelos individuales.

La librería cuenta con 2 clases que permiten mostrar y controlar información, la primera de ellas es la clase *DataPlot* la cual permite mostrar gráficos generados a partir de la librería Matplotlib [71]. La clase *Logger* controla los datos que se muestran en consola además de permitir guardar los datos en un archivo dado que hereda de la clase *Serializable*.

A parte de las clases antes descritas DeepEnsemble implementa una serie de funciones y clases útiles relacionadas con el entrenamiento de los modelos:

Funciones Útiles

- *score_functions*: Conjunto de funciones para calcular métricas de desempeño de los modelos.
- *cost_functions*: Conjunto de funciones de costos para usar durante el entrenamiento de los modelos.
- *regularized_functions*: Conjunto de funciones de regularización para usar durante el entrenamiento de los modelos.
- *update_functions*: Conjunto de funciones para actualizar los parámetros de los modelos durante el entrenamiento de los modelos.

Clases Útiles

- *ActivationFunctions*: Clase que contiene varias funciones típicas de activación para ser usadas al construir una red neuronal: tangente hiperbolica, sigmoide, softmax, etc.

- *DiversityFunctions*: Clase que contiene varias funciones y métricas relacionadas con la diversidad: bías, ambiguedad, varianza, etc.
- *ITLFunctions*: Clase que contiene varias funciones relacionadas con ITL (ver sección 2.6).

3.3.1. Estructura de los modelos

En DeepEnsemble existen 3 modelos como se ve en el diagrama de la figura 3.5, donde cada uno hereda de la clase abstracta *Model*. El modelo *Sequential* permite generar redes neuronales apilando diferentes capas de neuronas. Los modelos *Wrapper* son modelos que permiten generar una interfaz entre los modelos de la librería Scikit [72] y los de DeepEnsemble, por tanto, es posible trabajar no solo con redes neuronales sino también con otros como SVM, Random Forest, KNN, etc. El modelo *Ensemble* permite combinar diferentes modelos en uno solo, estos modelos pueden ser de tipo *Sequential* o *Wrapper*, para poder generar un Ensamble es necesario agregar los modelos y seleccionar una función para combinarlos.

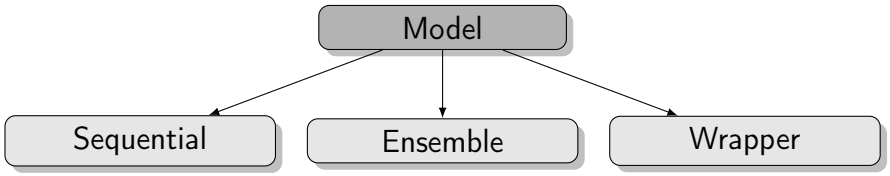


Figura 3.5: Diagrama de las clases de modelos en la librería DeepEnsemble.

El modo de generar modelos en DeepEnsemble se inspira principalmente en Keras [73] y Lasagne [74] 2 librerías de Python que son ampliamente utilizadas para construir redes neuronales de Aprendizaje Profundo. Para generar las redes neuronales en DeepEnsemble es necesario crear un objeto de tipo *Sequential* para luego ir agregando capas a la red, las capas que se pueden utilizar son las que aparecen en la figura 3.6:

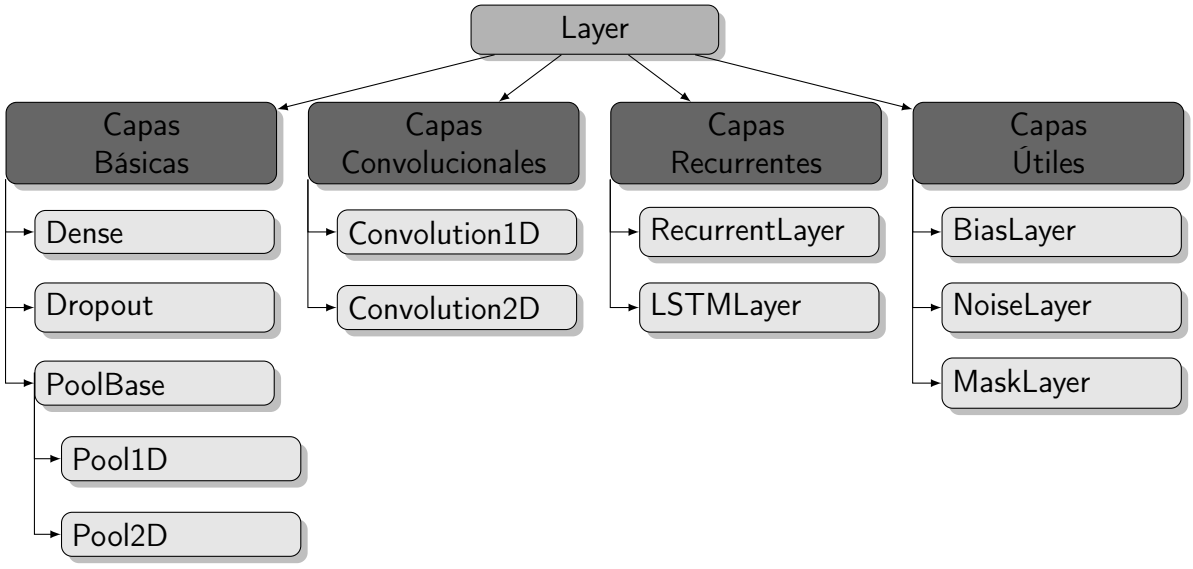


Figura 3.6: Diagrama de las clases de capas de la librería DeepEnsemble.

La capa básica para generar redes en donde todas las neuronas esta conectadas es *Dense*, las demás capas básicas son las típicas en redes de Aprendizaje Profundo: *Dropout*, *Pool1D* y *Pool2D*, también están implementadas las capas para generar redes convolucionales o CNN (Convolutional Neural Network): *Convolution1D* y *Convolution2D*.

Para generar redes recurrentes se pueden utilizar las capas: *RecurrentLayer* y *LSTMLayer*, la primera permite generar redes recurrentes simples y la segunda capa es para generar redes recurrentes tipo LSTM [75]. Finalmente, en cuanto a las capas existen algunas que son de propósito general que permiten agregar ruido, enmascarar salidas y agregar sesgo a la salida estas corresponden a las capas *NoiseLayer*, *MaskLayer* y *BiasLayer* respectivamente.

Los Ensambles utilizan una estructura similar a la de la figura 3.1 en donde los modelos deben ser combinados a través de una función. Para esto último se utilizan las clases del diagrama de la figura 3.7.

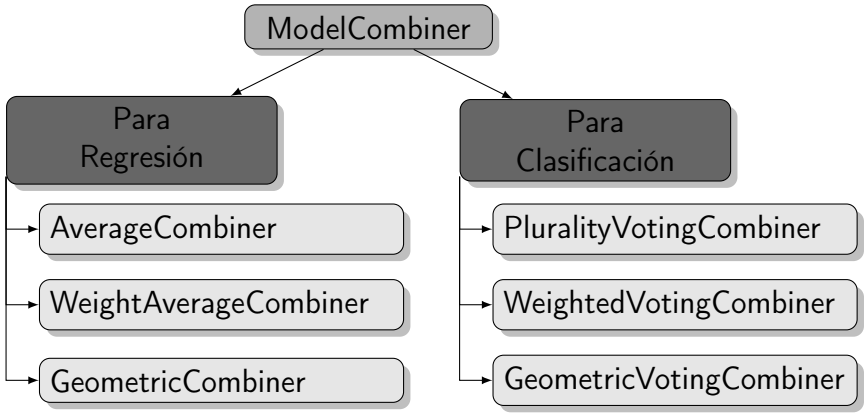


Figura 3.7: Diagrama de las clases para combinar modelos en la librería DeepEnsemble.

Estas clases están separadas según el tipo de problema para el cual se entrena el modelo: regresión o clasificación, en los métodos de clasificación son similares a los de regresión durante el entrenamiento sin embargo incluyen una etapa en el proceso de predicción donde traducen la salida numérica del modelo por la clase correspondiente la cual es seleccionada según el criterio de votación máxima. Los sufijos "Average" corresponden a que la salida del Ensamble se calcula como el promedio simple de los modelos, "Weight" corresponde a que la salida del Ensamble es la suma de las salidas ponderada por pesos calculados a partir de la ecuación (2.23), y los sufijos "Geometric" corresponden a que la salida del Ensamble es calculada a partir de la media geométrica de las salidas de los modelos. Es importante destacar que las votaciones siempre entregan una clase por tanto para el caso en donde la votación entre los modelos de un Ensamble no entrega una mayoría la clase entrega como resultado la primera clase por defecto.

3.3.2. Ejemplo de uso

Debido a que los objetivos del presente trabajo no están centrados en la creación de una librería no es necesario ahondar en los detalles de la librería DeepEnsemble, sin embargo daremos una pequeña reseña de ejemplos para entender de forma sucinta como funciona debido a que tomo gran parte del tiempo del desarrollo de este trabajo.

El modo de crear redes neuronales en DeepEnsemble se inspira principalmente en las librerías Keras [73] y Lasagne [74] en donde las capas se definen en clases externas y luego son anexadas en forma secuencial al objeto que representa a la red. En DeepEnsemble se debe definir una objeto de la clase *Sequential* para comenzar a construir la red neuronal, luego se definen las diferentes capas de la red que son integradas a través del método *add_layer* tal como se muestra en el ejemplo del código 3.1 donde se genera una red neuronal de tipo MLP (Multi Layer Perceptor), la definición de la estructura de la red se realiza en las lineas 4-10 del código 3.1.

Código 3.1: Ejemplo de cómo crear una Red Neuronal tipo MLP.

```
1 # Crear Red MLP para clasificación
2 net = Sequential("mlp", "classifier", classes_names)
3 # Definir Capa de entrada y oculta
4 net.add_layer(Dense( n_input=10, # Número de entradas
5  n_output=5,  # Número de Neuronas
6  activation=ActivationFunctions.sigmoid))
7
8 # Definir capa de salida
9 net.add_layer(Dense( n_output=2,  # Número de Salidas
10  activation=ActivationFunctions.sigmoid))
11
12 # Definir función de costo
13 net.append_cost(mse, name='MSE')  # MSE
14
15 # Definir método de actualización de parámetros
16     net.set_update(adagrad, name='Adagrad',  # Método ADAGRAD
17      initial_learning_rate=0.01)
18
19 # Compilar red MLP
20 net.compile()
21
22 # Entrenar red MLP y obtener las métricas
23 # (X, y) conjunto de entrenamiento
24 metrics = net.fit(X, y, max_epoch=300, batch_size=40)
```

Para la definición de un modelo en DeepEnsemble es necesario definir no solo la estructura sino también las funciones de costo (linea 13 del código 3.1) y la función de actualización de parámetros (linea 16 y 17 del código 3.1). Antes de pasar al entrenamiento es necesario compilar el modelo (ver linea 20 del código 3.1). Este procedimiento se debe realizar debido a que la librería Theano compila su código internamente en el lenguaje de C para optimizar y

mejorar el tiempo de ejecución. Para el entrenamiento se utiliza el método *fit* el cual permite definir algunos parámetros del entrenamiento como el número de épocas máximo, el tamaño del mini-lote, el conjunto de entrenamiento, el porcentaje de muestras dedicadas al conjunto de validación, etc. El método de entrenamiento por defecto es el de gradiente descendente estocástica con mini-lote (ver sección (2.3.1)) sin embargo también es posible cambiar aquello a través los parámetros que se entregan al método *fit*, por ejemplo si se requiere implementar solo GD se puede incluir los parámetros *update_sets=False* y *minibatch=False* para evitar la mezcla del conjunto de muestras y la utilización de mini-lotes.

Para construir un modelo de Ensamble primero se crea una instancia de la clase *Ensemble* y luego se agregan los modelos que contendrá. La definición de los modelos es similar a lo visto en el código 3.1 sin embargo es importante destacar que en la definición de un Ensamble no se especifica las dimensiones de entrada y salida puesto que esto queda definido por el primer modelo que se añade al ensamble. Luego los siguientes modelos deben tener los mismos números de entrada y salida de lo contrario se produce un error al momento de añadir el modelo. La definición del entrenamiento es similar al caso del modelo MLP del código 3.1. Sin embargo, es importante definir previamente el método de combinación tal como se muestra en la linea 21 del código 3.2, donde se muestra un ejemplo de creación de un Ensamble que define una función de votación para definir la clase resultante.

Código 3.2: Ejemplo de como crear un Ensamble de Redes Neuronales tipo MLP.

```
1 # Crear Ensamble para clasificación
2 ensemble = EnsembleModel(name="Ensemble")
3
4 # Generar 10 redes neuronales tipo MLP
5 for i in range(10):
6     net = Sequential("net%d_ens" % i)
7     # Definir Capa de entrada y oculta
8     net.add_layer(Dense( n_input=10, # Número de entradas
9      n_output=10,  # Número de Neuronas
10     activation=ActivationFunctions.sigmoid))
11        # Definir capa de salida
12    net.add_layer(Dense( n_output=2,  # Número de Salidas
13     activation=ActivationFunctions.sigmoid))
14    # Definir función de costo
15    net.append_cost(kullback_leibler, name='KL')  # Kullback Leibler
16    # Definir método de actualización de parámetros
17    net.set_update(sgd, name='SGD',  # Método SGD
18     learning_rate=0.1)
19
20 # Método para combinar las redes neuronales
21 ensemble.set_combiner(PluralityVotingCombiner())
22
23 # Función de regularización para NCL
24 ensemble.add_cost_ensemble(fun_cost=neg_corr, name="NCL", lamb=0.8)
25 # Compilar Ensemble
26 ensemble.compile()
```

```
27
28 # Entrenar Ensemble y obtener las métricas
29 # (X, y) conjunto de entrenamiento
30 metrics = ensemble.fit(X, y, max_epoch=300, batch_size=40)
```

El ejemplo del código 3.2 también incluye en la linea 24 la integración de una función de costo que involucra a todos los modelos que en este caso utiliza la función de costo de Correlación Negativa con un parámetro $\lambda = 0.8$, el método add_cost_ensemble es importante debido a que permite que el entrenamiento se desarrolle de forma paralela si la función de costo requiere información de los demás modelos del Ensamble.

También es posible trabajar con redes neuronales del Aprendizaje Profundo, a continuación, se presenta un ejemplo de cómo crear una red convolucional o CNN para el problema de clasificación de dígitos representados por una imagen de 28×28 en escala de grises [76]:

Código 3.3: Ejemplo de como crear una red convolucional CNN para el reconocimiento de dígitos MNIST.

```
1 # Obtener base de datos MNIST ...
2
3 # 8 x 12 filtros convolucionales de [5 x 5]
4 net = Sequential("mlp", "classifier", classes_names)
5 net.add_layer(Convolution2D(input_shape=(None, 1, 28, 28), num_filters=8,
6  filter_size=(5, 5)))
7 net.add_layer(MaxPool2D(pool_size=(2, 2)))
8 net.add_layer(Convolution2D(num_filters=12, filter_size=(5, 5)))
9 net.add_layer(MaxPool2D(pool_size=(2, 2)))
10 net.add_layer(Dropout(p=0.5))
11 net.add_layer(Dense(n_output=20, activation=ActivationFunctions.sigmoid))
12 net.add_layer(Dense(n_output=10, activation=ActivationFunctions.softmax))
13 net.append_cost(neg_log_likelihood, name='Neg Log Likelihood')
14 net.set_update(sgd, name='SGD', learning_rate=0.001)
15 net.compile() # Compilar la red
16
17 # Entrenar la red CNN
18 # (X, y) conjunto de entrenamiento
19 metrics = net.fit(X, y, max_epoch=300, batch_size=200)
```

Para las redes convolucionales se siguen los mismos pasos que en la creación de las redes neuronales simples la única diferencia lo marcan las capas, en el caso del código de ejemplo 3.3 la red tiene 2 capas convolucionales de 2 dimensiones con un filtro de $5x5$, la primera capa tiene 8 filtros y la segunda 12, además se agrega a la salida de cada capa convolucional una capa de pooling que reduce la dimensiones de la imagen filtrada a la mitad. Para reducir los problemas de sobreajuste se agrega una capa de Dropout que apaga la mitad de las salidas anteriores, finalmente se agregan 2 capas de neuronas todas conectadas con el fin de generar la clasificación final de los dígitos.

Con DeepEnsemble también es posible entrenar Redes Neuronales Recurrentes simples y

del tipo LSTM [75] estas redes permiten trabajar con información calculada previamente se utilizan en problemas donde los datos que se intentan modelar tienen dinámica. Varias redes recurrentes pueden ser combinadas para formar un Ensamble tal como aparece en el código 3.4:

Código 3.4: Ejemplo de cómo crear un Ensamble de Redes Neuronales Recurrentes.

```
# Crear Ensamble para regresión
ensemble = EnsembleModel(name="Ensemble")

# Crear 4 Redes Neuronales Recurrentes para el Ensamble
for i in range(4):
    net = Sequential("net%d_ens" % i)  # Crear RNN
        # Agregar una capa recurrente
    net.add_layer(RecurrentLayer(n_input=X_train.shape[1],
     n_recurrent=n_neurons, activation=ActivationFunctions.tanh))
        # Agregar una capa neuronal no recurrente
    net.add_layer(Dense(n_output=y.shape[1],
     activation=ActivationFunctions.sigmoid))
    net.append_cost(mse, name='MSE')  # MSE como función de costo
    # SGD como método de actualización de parámetros
    net.set_update(sgd, name='SGD', learning_rate=0.01)
        # Agregar la red recurrente al Ensamble
    ensemble.append_model(net)

# Método para combinar las redes neuronales
ensemble.set_combiner(AverageCombiner())
ensemble.compile()  # Compilar el Ensamble

# Entrenar Ensamble y obtener las métricas
# (X, y) conjunto de entrenamiento
metrics = ensemble.fit(X, y, max_epoch=300, batch_size=40)
```

Finalmente es importante destacar que en los problemas de clasificación la representación interna de las clases son del tipo one hot encoding, es decir, que cada muestra es representada por un vector con un número de elementos igual a la cantidad de clases y en donde solo existe un "1 " y los demás son ceros. La ubicación del "1" indica la clase que representa el vector.

3.3.3. Requerimientos e Instalación

Para poder instalar la librería DeepEnsemble es necesario primeramente instalar en el computador alguna versión de Python 3 y la herramienta *pip* para poder instalar los diferentes módulos adicionales. A continuación, se muestra una lista con los paquetes o módulos que son requisito para poder instalar la librería DeepEnsemble:

Theano 0.8.0, *mock*, *numpydoc*, *pep8 1.6.2*, *pytest*, *pytest cov*, *pytest pep8*, *Jinja2 2.7.3*, *Sphinx 1.2.3*, *sphinx_rtd_theme*, *sphinxcontrib napoleon*, *numpydoc*, *sklearn*, *matplotlib* y *pandas*.

Antes de usar la aceleración de hardware provista por la librería Theano es necesario realizar una configuración por tanto se siguiere revisar el link del programa `http://deeplearning.net/software/theano/install.html`. Las demás librerías no requieren en principio una configuración extra.

La librería se DeepEnsemble se puede instalar a través de la herramienta *pip* escribiendo en alguna consola del sistema el comando:

```
pip install https://github.com/pdoren/DeepEnsemble
```

también es posible descargar directamente el código fuente a través de git para realizar modificaciones, el comando para descargar el código es el siguiente:

```
git clone https://github.com/pdoren/DeepEnsemble.git
```

Finalmente, luego de descargar el código y realizar alguna modificación se puede instalar en el sistema utilizando el siguiente comando:

```
cd DeepEnsemble
pip install -r requirements.txt
```

El código de DeepEnsemble está disponible de manera gratuita en la plataforma Github bajo la licencia MIT (`https://github.com/pdoren/DeepEnsemble/blob/master/LICENSE`) además la versión v0.1 fue utilizada para todas las pruebas de este trabajo.

Capítulo 4

Análisis de Resultados

En este capítulo se presentan una serie de pruebas con CIPL realizadas sobre diferentes problemas de regresión y clasificación con el objetivo de evaluar su desempeño. Estas pruebas fueron realizadas con la librería DeepEnsemble. A modo de comparar los resultados de desempeño de CIPL las pruebas son realizadas de forma paralela sobre otros modelos incluido el modelo base de comparación que es NCL.

Las pruebas se centran en evaluar principalmente la precisión de predicción de los Ensambles junto con la relación de diversidad generada entre los modelos del mismo. Respecto a la diversidad se presentan métricas generadas sobre pares de modelos y sobre el Ensamble completo. Por otro lado, se incluyen gráficos de cómo evoluciona la diversidad y la precisión del Ensamble durante el entrenamiento.

En este capítulo también se incluye un análisis sobre el comportamiento de CIPL bajo distintas configuraciones de parámetros los cuales incluyen los hiperparámetros λ, β y σ además de la cantidad de neuronas por cada modelo junto con la cantidad de modelos en el Ensamble. Estas pruebas buscan determinar cómo afecta cada uno de los hiperparámetros sobre la precisión y la diversidad, y si existen interrelaciones entre ellos.

En el análisis de los distintos modelos evaluados se incluye una prueba de comportamiento bajo condiciones de ruido y existencia de valores atípicos con el objetivo de cuantificar que tan robustos son bajo condiciones negativas.

Finalmente, después de concluir las pruebas se presenta un análisis comparativo entre el método NCL y CIPL con el fin de determinar debilidades y fortalezas de cada método.

4.1. Bases de Datos

Para entrenar y analizar cada uno de los modelos se dividen los ejemplos de las bases de datos en 70 % para el conjunto de entrenamiento y 30 % para el conjunto de prueba o test. El conjunto de entrenamiento a su vez se divide en 90 % para el entrenamiento y 10 % para la validación. Además, para obtener los resultados comparativos entre diferentes tipos de modelos se utiliza la validación cruzada de 10 iteraciones. A continuación se presentan las distintas bases de datos utilizadas.

4.1.1. Problemas de Regresión

Para probar el método CIPL en problemas de regresión se utilizan 2 base de datos las cuales fueron antes utilizadas en la tesis doctoral de Gavin Brown en 2004 [9].

Jacobs

En este problema de regresión los datos son generados a partir de la siguiente función:

$$h_J(\mathbf{x}) = \frac{1}{13}\left[10sin(\pi x_1 x_2) + 20\left(x_3 - \frac{1}{2}\right)^2 + 10x_4 + 5x_5\right] - 1, \tag{4.1}$$

donde $\mathbf{x} = [x_1, ..., x_5]$ es el vector de entrada y cada componente es generada por una variable aleatoria de distribución uniforme definida en el intervalo $[0, 1]$. La función $h(\cdot)$ tiene como imagen el intervalo $[-1, 1]$. Esta base de datos es utilizada en los trabajos [77] y [78].

Friendman #1

En este problema de regresión los datos son generados a partir de la siguiente función:

$$h_F(\mathbf{x}) = 10sin(\pi x_1 x_2) + 20\left(x_3 - \frac{1}{2}\right)^2 + 10x_4 + 5x_5, \tag{4.2}$$

donde $\mathbf{x} = [x_1, ..., x_{10}]$ es el vector de entrada y cada componente de este es obtenido como el valor de una variable aleatoria de distribución uniforme en el intervalo $[0, 1]$. Aunque el vector de entrada de la función (4.2) tiene 10 elementos solo se usan los primeros 5 elementos, por tanto, existen 5 características irrelevantes y por tanto se espera que los modelos al ser entrenados descarten estos valores.

4.1.2. Problemas de Clasificación

Para los problemas de clasificación se han seleccionado 3 bases de datos típicas para evaluar el desempeño de los modelos, 2 de estas bases de datos corresponden a clasificación

binaria y la restante tiene 6 clases. Todas las bases de datos utilizadas fueron obtenidas de la página `http://mldata.org` estas también pueden ser obtenidas directamente del repositorio UCI Machine Learning [79].

Australian Credit Card

Esta base de datos corresponde a un problema de solicitud de la tarjeta de crédito, visto como un problema de clasificación en donde las clases corresponden a si la solicitud es aceptada o no. La base de datos cuenta con 690 muestras, 307 (44.5 %) para solicitud aceptada y 383 (55.5 %) para solicitud no aceptada. Cada muestra contiene 14 características que están normalizadas al rango $[0, 1]$, las características originales son continuas y nominales con distintas cantidades de categorías lo que lo hace un problema interesante por la variedad de tipos de características.

German Numer

Esta base de datos corresponde a un problema de evaluación de riesgo de crédito visto como un problema de clasificación en donde las clases corresponden a si el cliente es bueno o malo. La base de datos cuenta con 1000 muestras, 700 (70 %) para calificar como buen cliente y 300 (30 %) para malos clientes. Cada muestra contiene 24 características que están normalizadas al rango $[0, 1]$. Las características originales son continuas y nominales con distintas cantidades de categorías.

Satimage

La base de datos proviene de imágenes del satélite Landsat MSS que a su vez consta de cuatro imágenes digitales de la misma escena en bandas espectrales diferentes, 2 de ellos son en la región visible (correspondiente aproximadamente al verde y regiones rojas del espectro visible) y dos están cerca del infrarrojo. Cada píxel es una palabra binaria de 8 bits, con 0 correspondiente a negro y 255 a blanco. La resolución espacial de un píxel es $80 \times 80[m^2]$, cada imagen contiene 2340×3380 píxeles.

Las muestras corresponden a una pequeña sub-área de una escena que consta de 82×100 píxeles donde cada línea de datos corresponde a un barrio cuadrado de 3x3 de píxeles completamente contenidos dentro de la sub-área 82×100. Cada línea contiene los valores de píxel en las cuatro bandas espectrales (Convertido a ASCII) de cada uno de los 9 píxeles en el barrio 3x3 y un número que indica la etiqueta de clasificación del píxel central. La base de datos cuenta con 10870 muestras las cuales están divididas en 6 clases: tierra roja (2605, 22.0 %), cosecha de algodón (1182, 10.9 %), suelo gris (2319, 21.3 %), suelo gris húmedo(1041, 9.6 %), suelo con rastrojo de vegetación (1117, 10.3 %), suelo gris muy húmedo (2546, 23.4 %), cada muestra a su vez cuenta con 36 características (= 4 bandas x 9 píxeles) que son normalizadas al rango $[0, 1]$.

4.2. Pruebas de validación del método CIPL

Debido a la dificultad de implementar el método CIPL es necesario asegurar el correcto funcionamiento de la librería antes de usarla. Para ello se hicieron 3 pruebas para validar la implementación del método CIPL. Estas pruebas también tienen el propósito de demostrar que el método propuesto es coherente en su objetivo de maximizar la Información Mutua.

Las primeras pruebas consideran un problema de una red neuronal multicapa de forma que sea simple la obtención de algunos resultados. Las últimas pruebas incluyen Ensambles compuesto de modelos similares a los utilizados anteriormente con el objetivo de validar la implementación del método en Ensambles.

En esta primera prueba se busca probar que la función de costo implementada es la correcta, para ello se compara la Información Mutua de Shannon (2.34) con las funciones de estimación de la Información Mutua Cuadrática calculadas con las divergencias Euclidiana y Cauchy-Schwartz, ver ecuación (2.62). La información Mutua de Shannon se estima a partir de ventanas de Parzen,

$$
\begin{aligned}
I(x,y) \approx I_p(x,y) &= \sum \hat{p}_{XY}(x,y) log\left(\frac{\hat{p}_{XY}(x,y)}{\hat{p}_X(x)\hat{p}_Y(y)}\right) \\
p_X(x) &= \frac{1}{N}\sum_{j=1}^{N} G(x-x_j,\sigma^2 I) \Rightarrow \hat{p}_X(x) = \frac{p_X(x)}{\sum_{x_i \in X} p_X(x_i)} \\
p_Y(y) &= \frac{1}{N}\sum_{j=1}^{N} G(y-y_j,\sigma^2 I) \Rightarrow \hat{p}_X(x) = \frac{p_Y(y)}{\sum_{y_i \in Y} p_Y(y_i)} \\
p_{XY}(x,y) &= \frac{1}{N^2}\sum_{j=1}^{N}\sum_{i=1}^{N} G(x-x_j,2\sigma^2 I) G(y-y_i,2\sigma^2 I) \\
\hat{p}_{XY}(x,y) &= \frac{p_{XY}(x,y)}{\sum_{x_i,y_i \in X,Y} p_{XY}(x_i,y_i)},
\end{aligned}
\tag{4.3}
$$

para poder compararla con las informaciones mutuas cuadráticas que también utilizan en sus cálculos las ventanas de Parzen.

En la figura 4.1 se muestra las curvas de las 3 informaciones mutuas obtenidas del mismo conjunto de ejemplos. Para obtener las curvas se utilizan 2 variables aleatorias a las cuales se les calcula su información mutua para distintas ocurrencias, de forma de generar la cantidad suficiente de puntos. Las variables aleatorias utilizadas son X e Y, que distribuyen de forma uniforme con valores discretos $\{0,1\}$ y parámetro 0.5. Para cada punto se calcula la información mutua de 2 series de datos formados de $N = 200$ elementos obtenidos de las ocurrencias de las variables aleatorias X e Y, a su vez este punto es asociado a la precisión o similitud entre ambas series.

En un primer análisis cualitativo de las curvas de la figura 4.1 se puede apreciar que I_{ED} es una cota inferior de la Información Mutua I. Además, en este ejemplo las 3 informaciones mutuas son convexas al menos en este ejemplo, aunque es importante destacar que se sabe que eso no siempre pasa con I_{CS} [80]. En un segundo análisis se puede ver que las curvas son

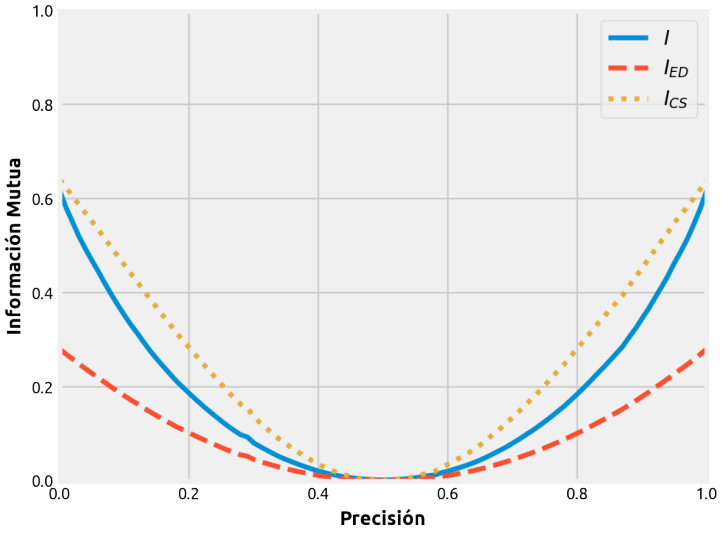


Figura 4.1: Prueba de validación Función de Costo CIPL, I corresponde a la Información Mutua de Shannon estimada con ventanas de Parzen, I_{CS} y I_{ED} son las Informaciones Mutuas Cuadráticas estimadas con el Potencial de Información Cruzado. La precisión se refiere al porcentaje de similitud entre las series de datos obtenidas de las ocurrencias de las variables aleatorias X e Y.

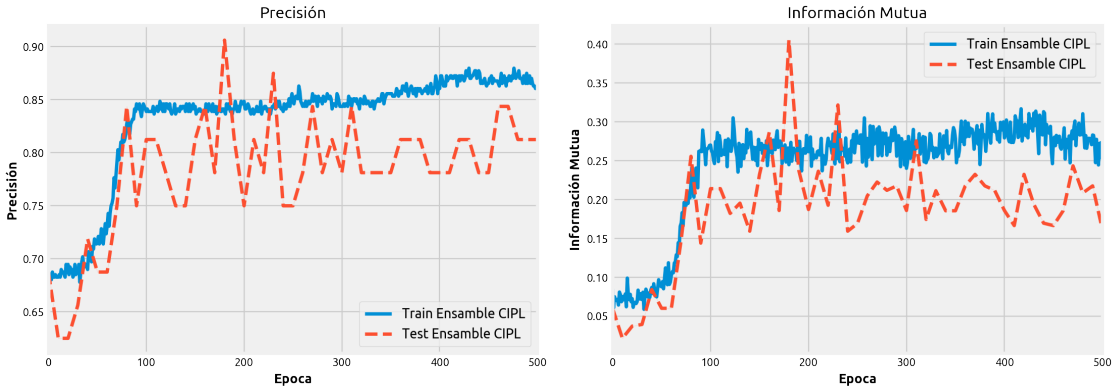
coherentes con el comportamiento esperado para la información mutua, por ejemplo, cuando la precisión es 0.5, o lo que es lo mismo que ambas series de ocurrencias de X e Y difieren en la mitad de sus datos, significa que la información de una respecto a otra es mínima e igual a 0, y en los casos extremos donde ambas series son iguales o totalmente diferentes los valores de la información mutua son máximos.

La segunda prueba busca demostrar que la maximización de la función de costo CIPL permite maximizar la Información Mutua de Shannon entre los distintos modelos del Ensamble. Para ello se utiliza la base de datos de Australian Credit Card para generar un entrenamiento con CIPL sobre una red neuronal tipo MLP con solo una capa oculta de 10 neuronas y donde la capa de salida codifica las 2 clases con un vector de 2 elementos (one hot encoding). Las funciones de activación para todas las capas es la función sigmoide, por otro lado para ambos métodos CIPL los parámetros utilizados son: $\beta = 0.1$ y $\lambda = 0.5$.

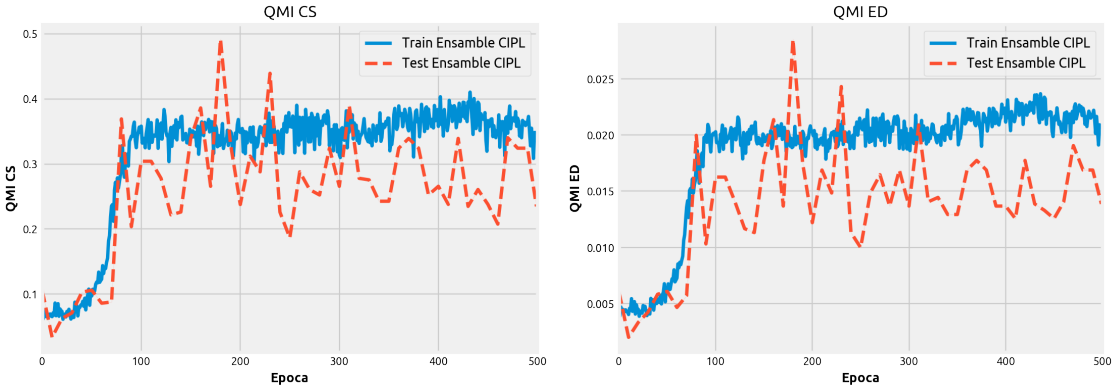
El método de optimización utilizado es el de gradiente descendente por mini-lotes donde cada lote tiene 32 elementos y la tasa de aprendizaje es de -0.03 (el valor negativo es por la maximización en SGD), este valor es negativo debido a que se requiere es maximizar la información mutua.

Para presentar los resultados se muestra un gráfico de la progresión de la precisión alcanzada por el método CIPL en el entrenamiento junto con gráficos de las informaciones mutuas entre la salida esperada y la salida final de la red MLP, A continuación, se presenta los resultados de entrenamiento obtenidos con CIPL calculado con Cauchy-Schwarz 4.2.

Al revisar los gráficos 4.2 se puede ver por un lado que la maximización de la Información Mutua Cuadrática tanto de Cauchy-Schwarz y Euclidiana implica una maximización de la



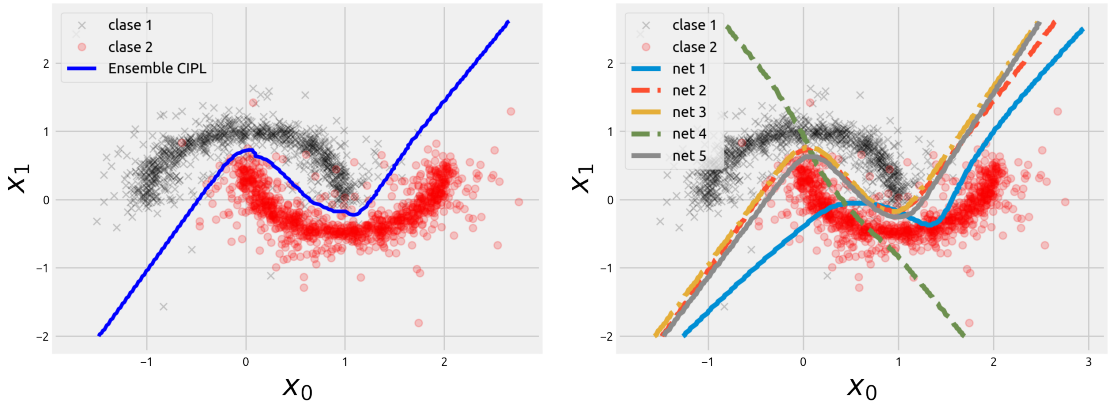
(a) Precisión (b) I Shannon



(c) I_{CS} (d) I_{ED}

Figura 4.2: Gráficos de entrenamiento de una red MLP donde se pueden ver los progresos por época de la precisión del modelo junto con los valores de la estimación de la Información Mutua de Shannon y la Información Mutua Cuadrática de Cauchy-Schwartz (QMI CS) y Euclidiana (QMI ED). Las informaciones Mutuas son calculadas entre la salida esperada y la salida del modelo MLP.

Información Mutua de Shannon. Además, cualitativamente las informaciones mutuas tienen curvas similares pero que difieren en escala.



(a) Ensamble

(b) Modelos por separado

Figura 4.3: Ejemplo de clasificación utilizando un Ensamble de 3 redes neuronales tipo MLP entrenadas con CIPL CS (Cauchy-Schwarz).

Como tercera prueba se toma un ejemplo muy simple de clasificación para determinar si el código optimiza correctamente cuando se utiliza CIPL en Ensambles. El problema de clasificación es de 2 clases y 2 características. Además el conjunto de datos fue generado con la función make_moons de la librería Sklearn la cual genera 2 medias lunas como se ve en la figura 4.3. El Ensamble este contiene 5 redes neuronales del tipo MLP con 4 neuronas cada uno y una sola capa oculta, además la función de activación en las capas es la función tangente hiperbólica y en la capa de salida es la función Sigmoide. El ensamble no tiene problemas en adaptarse a los datos, la precisión alcanzada sobre el conjunto de prueba es de 97.68 %, mientras que la máxima precisión de los modelos del ensamble fue 97.45 % (net 3) y la mínima fue 47.23 % (net 5). Por tanto, con esta prueba simple se puede ver que la implementación de CIPL realiza correctamente el ajuste.

4.3. Resultados

Los resultados que se presentan a continuación incluyen primeramente información del entrenamiento junto con datos de cómo se comporta el error de predicción de CIPL. Para poder tener una referencia del desempeño de CIPL se incluyen en algunas pruebas 3 modelos para comparar: una red neuronal tipo MLP, un Ensamble entrenado con MSE y un Ensamble con NCL. Todos estos modelos, incluido los ensambles y la red MLP, tienen la misma cantidad de neuronas, de forma que puedan ser comparables dado que su complejidad está es su forma de entrenamiento y no en su estructura.

En el caso de los ensambles utilizados estos tienen la misma función para combinar la salida de sus modelos, es decir, el promedio simple para los problemas de regresión y votación por mayoría para problemas de clasificación.

En cuanto a los gráficos de entrenamiento algunos es importante aclarar que son obtenidos partir del entrenamiento de un solo modelo el cual es el que mejor resultado obtuvo, por otro lado en las tablas se agrega la información del promedio de 10 modelos obtenidos del método de validación cruzada de 10 iteraciones (10-fold).

4.3.1. Problemas de Regresión

El desempeño en regresión es evaluado a partir del RMS del error entre la salida del modelo y la salida esperada. Además se agrega el progreso de la Información Mutua de Shannon, estimada con ventanas de Parzen, para verificar que se cumple la condición de maximización de esta, tal como propone CIPL.

Aparte de los resultados de entrenamiento se presentan 3 pruebas extras para evaluar el efecto de CIPL al cambiar sus parámetros y el tamaño del kernel σ, además de una prueba para observar el efecto de ruido en el modelo. La primera prueba es para observar cómo afecta al desempeño el cambio de parámetros β (sinergia) y λ (redundancia) en CIPL. En este caso la variación se restringe de forma que ambos parámetros tienen el mismo valor, esto último para simplificar los cálculos y poder presentar los datos en un plano. Además, el valor de estos parámetros se restringe al rango $[-1, 1]$, este rango es arbitrario, pero busca simplificar el proceso de obtención de datos. La segunda prueba busca ver cómo afecta la variación del tamaño del kernel sobre el desempeño del modelo entrenado con CIPL, el valor de la variación es respecto a Silverman σ_s y este varia en el rango $[0.01 \cdot \sigma_s, 100 \cdot \sigma_s]$. La tercera prueba muestra como el desempeño afecta a la precisión del modelo cuando los datos incluyen ruido de forma de concluir que tan robusto es el modelo ante esta situación. El ruido es de tipo aditivo Gaussiano $\mathcal{N}(\mu = 0.2, \sigma^2_{ruido})$, se agrega un sesgo debido a que los resultados con solo una varianza no muestran diferencias significativas.

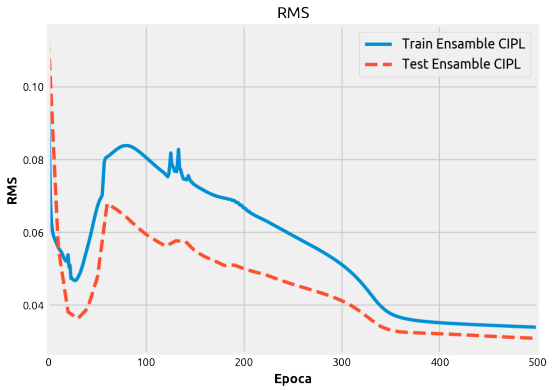
Jacobs

El modelo de Ensamble con CIPL que mejor resultado entrego es el que utiliza la diver-

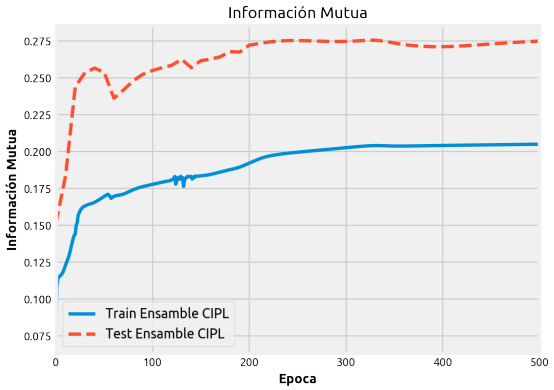
gencia Euclidiana con los parámetros $\beta = 0.2$ y $\lambda = 0.6$ y el factor de aprendizaje $\eta = -0.08$. En cuanto al tamaño del kernel este es $\sigma = \sigma_s$, por otro lado dado los resultados probados no se utiliza Annealing 3.2.1 durante el entrenamiento, es decir el tamaño del kernel queda fijo durante todo el entrenamiento, el cual es $\sigma_s = 0.1171$.

Los modelos utilizados para la comparación son una red neuronal de tipo MLP con una sola capa oculta, 3 ensambles compuestos de 4 redes neuronales de tipo MLP también con una sola capa oculta. Un ensamble es entrenado con mínimos cuadrados, uno con NCL y otro con CIPL. Los 4 modelos contienen la misma cantidad de neuronas incluso la red MLP, esta cantidad es de 36 donde cada una tiene como función de activación la función de tangente hiperbólica en la capa oculta y de salida.

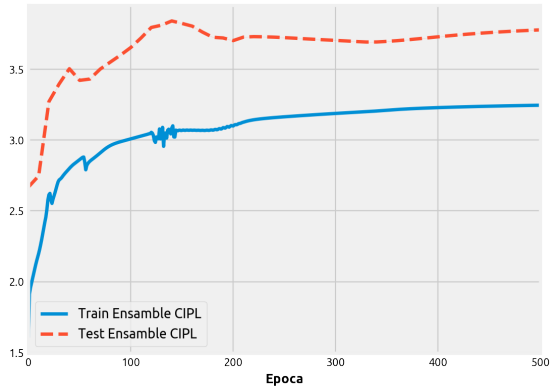
Los modelos fueron entrenados con un factor de aprendizaje de $\eta = -0.1$, en cuanto al parámetros de NCL el que mejor resultados entrega es $\lambda = 0.1$.



(a) RMS



(b) Información Mutua



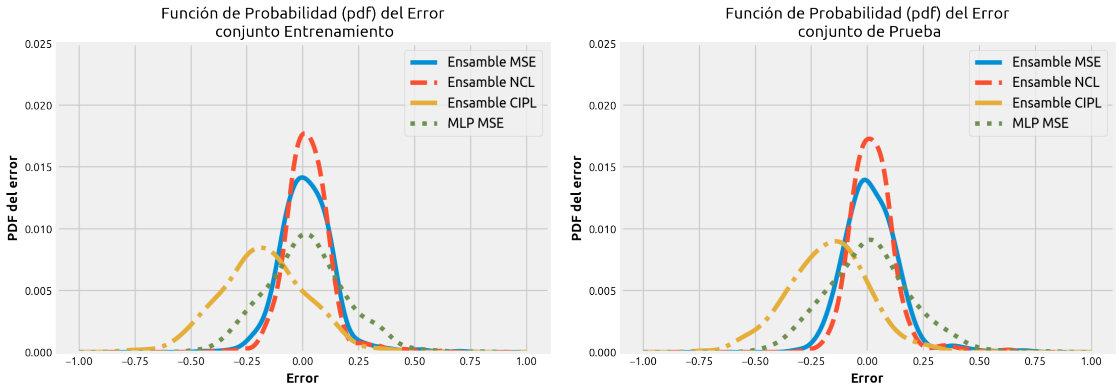
(c) Función de costo CIPL

Figura 4.4: Resultado del entrenamiento del mejor modelo CIPL obtenido. En el gráfico (a) se puede ver el progreso del desempeño RMS, (b) la Información Mutua de Shannon y (c) la maximización de la función de costo CIPL. En todos los gráficos se muestra el progreso de las métricas durante el entrenamiento y sobre los conjuntos de entrenamiento y test. La Información Mutua de Shannon es estimada con ventanas de Parzen y se calcula sobre los datos de salida del ensamble y la salida esperada.

Como se puede ver en la figura 4.4 el mejor Ensamble entrenado con CIPL es capaz

de reducir el error de regresión al mismo tiempo que maximiza la información mutua. En cuanto al desempeño final el CIPL obtuvo un RMS promedio de 0.0319 en el conjunto de entrenamiento y 0.0346 en el conjunto de prueba (ver tabla 4.1).

En la figura 4.5 se puede ver la función de probabilidad o pdf del error obtenido por cada modelo sobre los conjuntos de entrenamiento y prueba. Además, estos resultados corresponden al mejor modelo obtenido de cada tipo evaluado. Al revisar la pdf del error de CIPL de la figura 4.5 se puede ver que existe un sesgo respecto a los demás, por tanto se puede mejorar el desempeño restando ese sesgo a la salida del modelo.



(a) Conjunto Entrenamiento

(b) Conjunto de prueba

Figura 4.5: Función de probabilidad o pdf del error de los distintos modelos analizados. Las pdfs se obtuvieron para los conjuntos de entrenamiento y prueba respectivamente.

El método que peor desempeño es CIPL, como se puede ver en la tabla 4.1, donde los valores de esta tabla fueron obtenidos utilizando validación cruzada de 10 iteraciones.

Método	**Promedio RMS**	**min**	**max**
Ensamble CIP	0.0368 ± 0.0251	0.0346	0.0534
Ensamble NCL	0.0023 ± 0.0043	0.0018	0.0206
Ensamble MSE	0.0135 ± 0.0081	0.0087	0.0183
MLP MSE	0.0155 ± 0.0196	0.0124	0.0460

Tabla 4.1: Tabla con los resultados de desempeño de los distintos modelos evaluados, estos datos se obtuvieron a partir de la base de datos Jacobs y con la aplicación del método de validación cruzada de 10 iteraciones (10-fold). Se resalta en gris el modelo NCL que obtuvo el mejor desempeño en promedio.

Al revisar los resultados individuales de cada sub-modelo del Ensamble entrenado con CIPL (ver figura 4.6), se puede advertir que a nivel individual los modelos presentan sesgos diferentes que degeneran el resultado final del ensamble al ser combinados con solo la función de promedio simple. Es posible mejorar los resultados corrigiendo este problema antes de combinar los modelos.

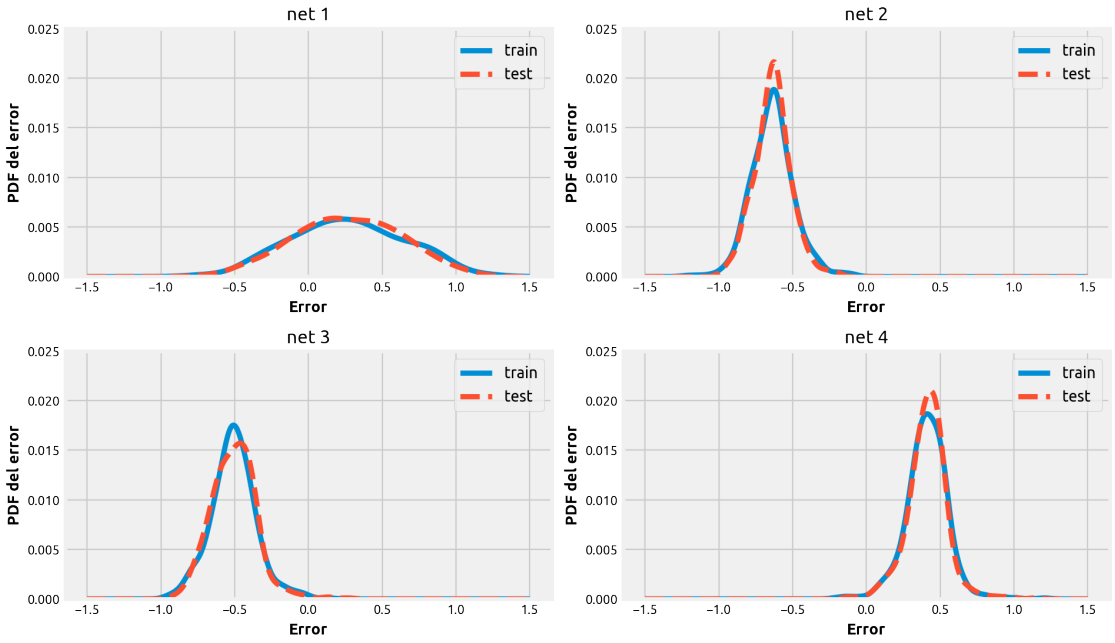


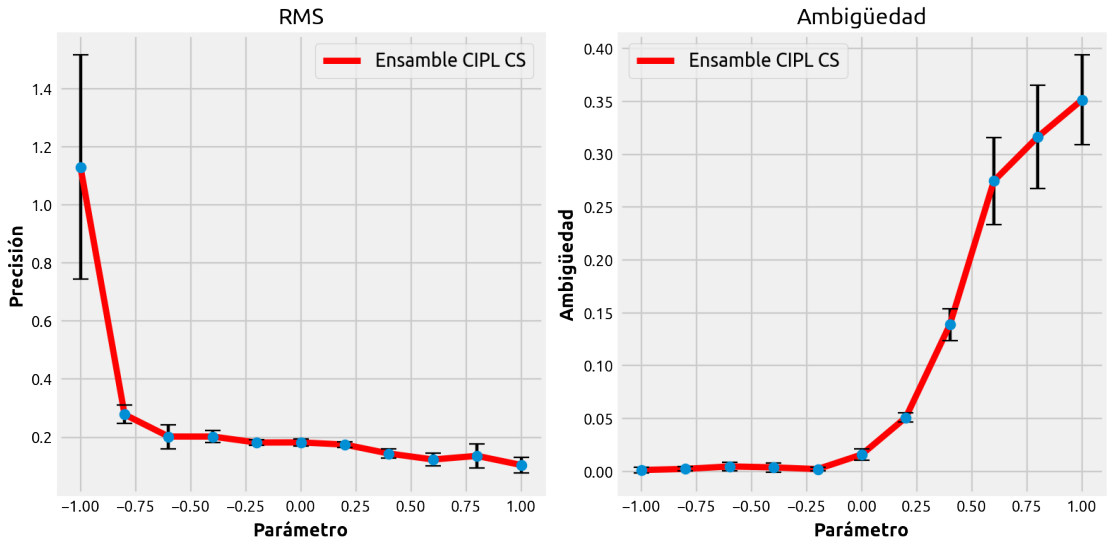
Figura 4.6: Función de probabilidad o pdf del error de los distintos sub-modelos del Ensamble entrenado con CIPL y que mejor resultado obtuvo. Las pdfs se obtuvieron para los conjuntos de entrenamiento (train) y prueba (test).

Prueba variación de parámetros

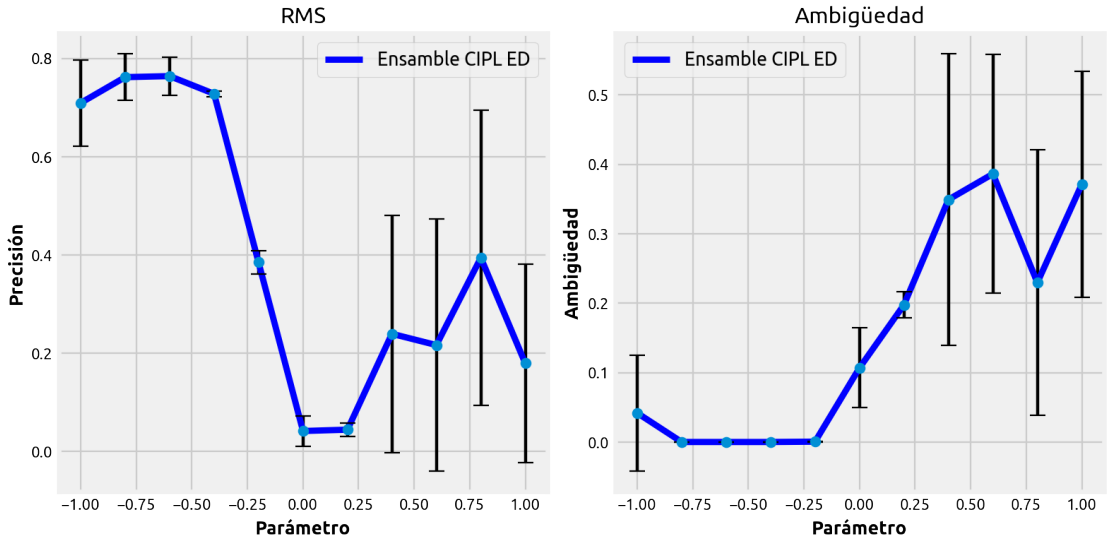
En la figura 4.7 se puede ver como se comporta el desempeño al variar los parámetros del Ensamble CIPL en la misma cantidad $\lambda = \beta$. En el caso de CIPL con Cauchy-Schwartz este muestra un comportamiento bien definido, donde al aumentar el valor de los parámetros aumenta la precisión del modelo (disminución del RMS). Además, para valores negativos de los parámetros la ambigüedad es prácticamente 0, pero aumenta para valores más grandes. Pasa algo similar con CIPL Euclidiana pero con menor definición (mayor desviación estándar en los puntos obtenidos). No es concluyente al revisar los gráficos, de la figura 4.7, que el aumento de la ambigüedad vaya de la mano con la disminución del RMS, pero se ve una tendencia.

Prueba variación del tamaño del kernel

El resultado obtenido en esta prueba se muestra en la figura 4.8, donde se aprecia que en los resultados de CIPL con Cauchy-Schwartz el error de predicción aumenta y la ambigüedad aumenta para valores pequeños del tamaño del kernel. En CIPL con divergencia Euclidiana la tendencia no es muy clara, la ambigüedad se mantienen casi constante.

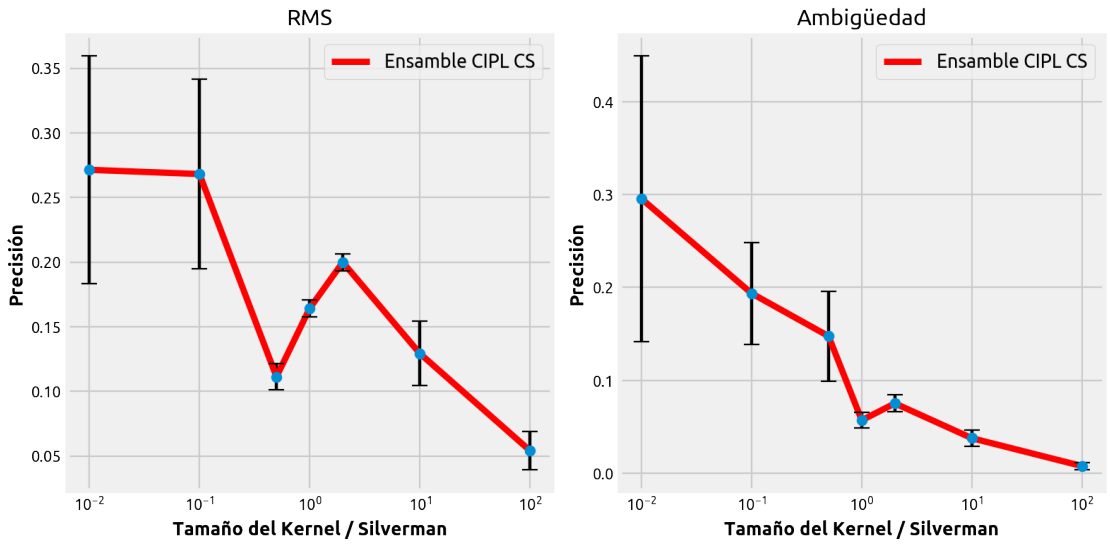


(a) CIPL Cauchy-Schwartz

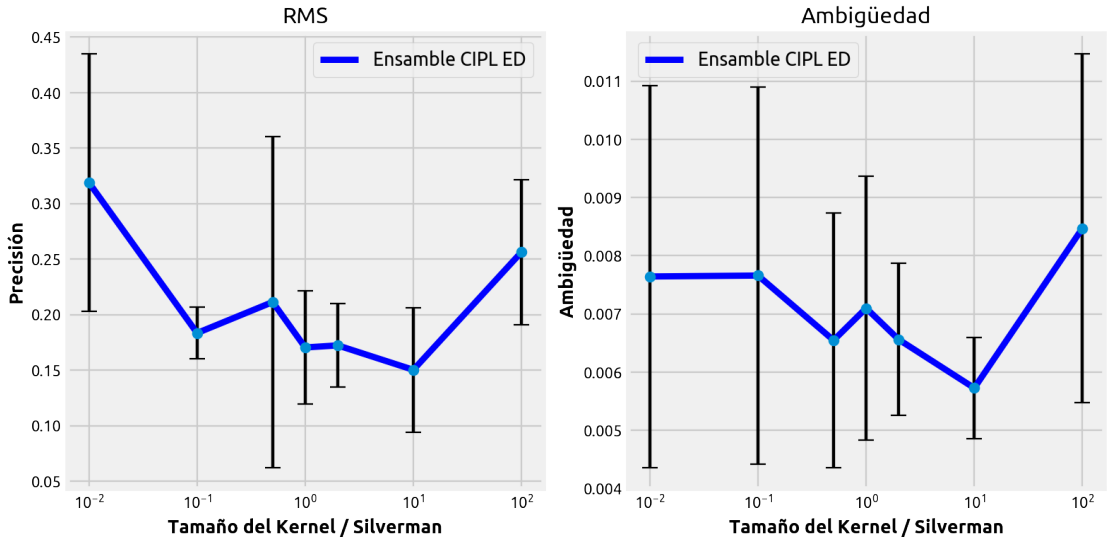


(b) CIPL Euclidiana

Figura 4.7: Gráficos que muestran como varia el desempeño (RMS) y la diversidad (ambigüedad) al variar el valor de los parámetros del Ensamble CIPL en la misma cantidad, es decir $\lambda = \beta$. Se muestra el desempeño de CIPL obtenido con divergencia (a) Cauchy-Schwartz y (b) Euclidiana. Las barras verticales en las curvas representan la desviación estándar de cada medición, además cada punto de la curva fue obtenido a partir de 10 mediciones.



(a) CIPL Cauchy-Schwartz



(b) CIPL Euclidiana

Figura 4.8: Gráficos que muestran como varia el desempeño (RMS) y la diversidad (ambigüedad) al variar el valor del tamaño del kernel del Ensamble CIPL.

Prueba de los modelos con ruido

En resultado de esta prueba se muestra en la figura 4.9 donde el ensamble CIPL con divergencia Euclidiana muestra un comportamiento singular a medida que disminuye el ruido, mejora su resultado. esto solo se puede explicar al sesgo $\mu = 0.2$ que se agrega en el ruido.

Por otro lado, el que mejor resultado entrega para ruidos altos es el ensamble entrenado con NCL seguido por CIPL con divergencia Euclidiana. Es importante destacar que en la figura 4.9 no se incluye el resultado de CIPL con Cauchy-Schwartz debido a que el desempeño es mucho peor que los otros modelos y por lo mismo no es comparable.

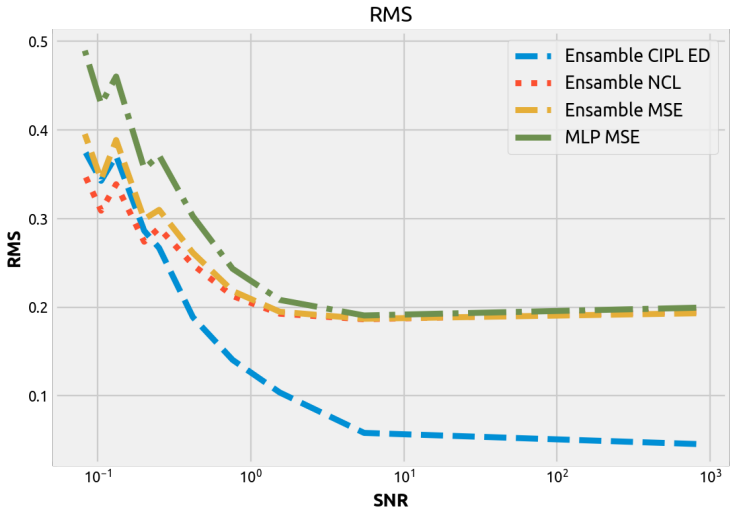


Figura 4.9: Gráfico que muestra como varia el desempeño evaluado con RMS de los distintos modelos de regresión cuando a los datos de entrada se les agrega ruido (Menos es mejor). El ruido es de tipo aditivo Gaussiano $\mathcal{N}(\mu = 0.2, \sigma^2_{ruido})$. El SNR es calculado a partir del ruido y los datos de entrada del modelo: $\text{SNR} = \sigma^2_{\text{entrada}}/\sigma^2_{\text{ruido}}$.

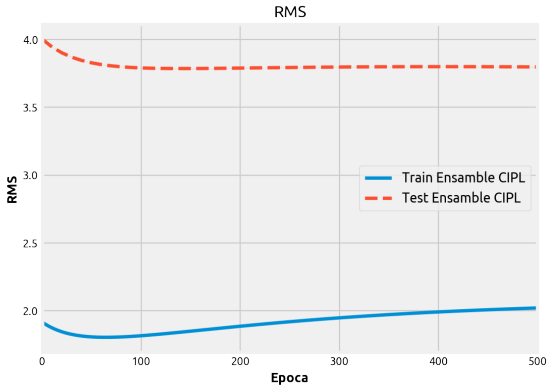
Friendman #1

El modelo de Ensamble con CIPL que mejor resultado entrego es el que utiliza la divergencia Euclidiana con los parámetros $\beta = 0.1$ y $\lambda = 0.1$ y el factor de aprendizaje $\eta = -0.1$. En cuanto al tamaño del kernel este el de Silverman $\sigma_s = 1.498$ y no se utiliza Annealing 3.2.1 durante el entrenamiento.

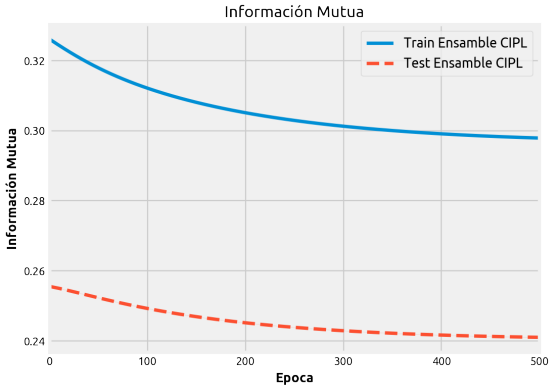
Los modelos utilizados para la comparación son similares a los utilizados en el problema de regresión Jacobs. La cantidad de neuronas total de cada modelo es de 45, además cada neurona tiene como función de activación en la capa oculta la función de tangente hiperbólica y en la capa de salida es lineal.

Los 4 modelos (MLP MSE, Ensamble MSE, Ensamble NCl y Ensamble CIPL) fueron entrenados con un factor de aprendizaje de $\eta = 0.1$. El parámetros de NCL que mejor resultados entrega es $\lambda = 0.4$.

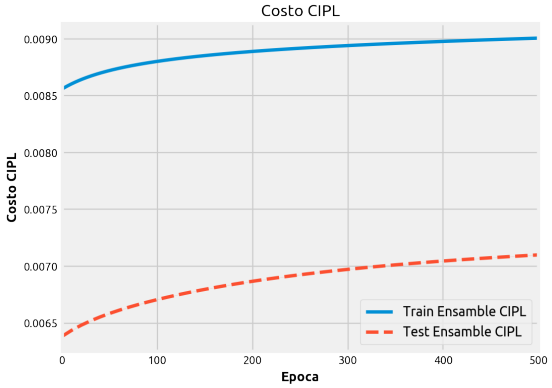
El resultado del entrenamiento del Ensamble entrenado con CIPL se presenta a continuación en la figura 4.10:



(a) RMS



(b) Información Mutua



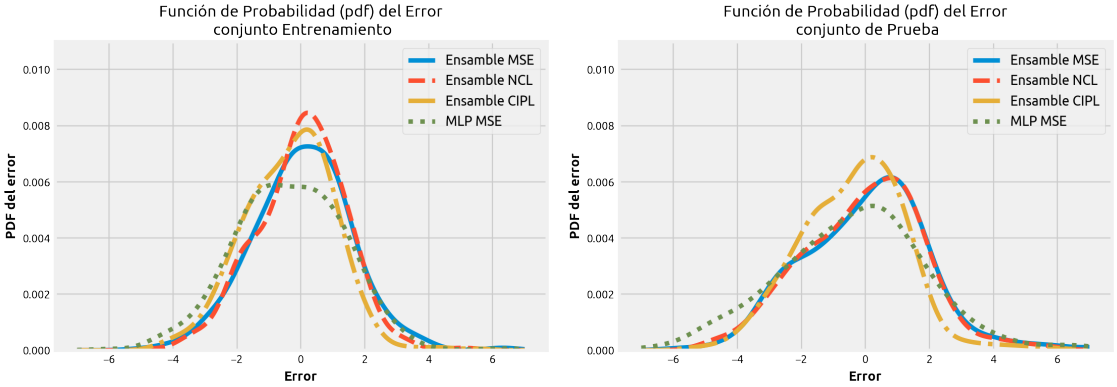
(c) IFunción de costo CIPL

Figura 4.10: Resultado del entrenamiento del mejor modelo CIPL obtenido. En el gráfico (a) se puede ver el progreso del desempeño RMS, (b) la Información Mutua de Shannon y (c) la maximización de la función de costo CIPL. En todos los gráficos se muestra el progreso de las métricas durante el entrenamiento y sobre los conjuntos de entrenamiento y test. La Información Mutua de Shannon es estimada con ventanas de Parzen y se calcula sobre los datos de salida del ensamble y la salida esperada.

Como se puede ver en la figura 4.10 el mejor Ensamble entrenado con CIPL no es es capaz de reducir el error de regresión y además la maximización de la función de costo CIPL no maximiza la Información Mutua de Shannon estimada con ventanas de Parzen. Lo anterior implica que la aproximación heurística de CIPL y propuesta en la ecuación (3.13) no siempre funciona.

También es importante mencionar que al revisar los resultados individuales de los modelos se observa el mismo problema que en el caso del problema Jacobs, es decir, los sub-modelos muestran sesgos distintos que deben ser corregidos antes de combinarlos. Por otro lado, en la figura 4.10 las curvas de desempeño entre los conjuntos de entrenamiento y prueba exhiben una gran diferencia que puede ser explicado por sobreajuste, sin embargo a pesar de reducir el factor de aprendizaje el comportamiento siguió siendo el mismo.

En la figura 4.11 se puede ver la función de probabilidad o pdf del error obtenido por cada modelo sobre los conjuntos de entrenamiento y prueba. Además, estos resultados corresponden al mejor modelo obtenido de cada tipo evaluado.



(a) Conjunto Entrenamiento (b) Conjunto de prueba

Figura 4.11: Función de probabilidad o pdf del error de los distintos modelos analizados. Las pdfs se obtuvieron para los conjuntos de entrenamiento y prueba respectivamente.

En cuanto al desempeño final el CIPL obtuvo un RMS promedio de 4.1233 en el conjunto de entrenamiento y 5.3460 en el conjunto de prueba (ver tabla 4.2).

Método	**Promedio RMS**	**min**	**max**
Ensamble CIP	5.3460 ± 4.1289	2.8324	8.1273
Ensamble NCL	4.1930 ± 2.2451	3.4142	5.8997
Ensamble MSE	5.1020 ± 3.5552	3.8311	7.0015
MLP MSE	8.7051 ± 5.0196	5.3118	10.6703

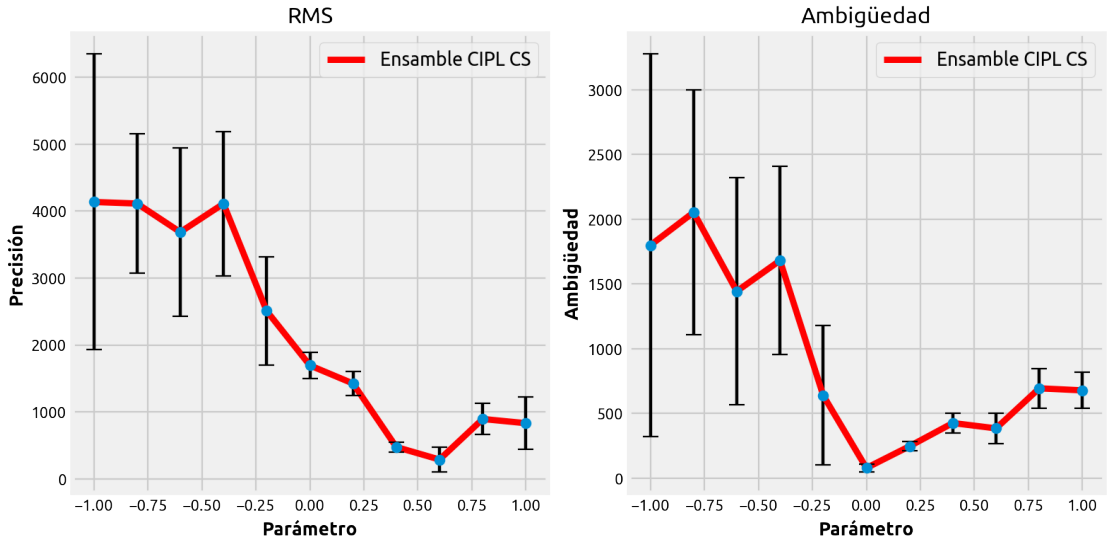
Tabla 4.2: Tabla con los resultados de desempeño de los distintos modelos evaluados, estos datos se obtuvieron a partir de la base de datos Friedman y con la aplicación del método de validación cruzada de 10 iteraciones (10-fold). Se resalta en gris el modelo NCL que obtuvo el mejor desempeño en promedio.

Prueba variación de parámetros

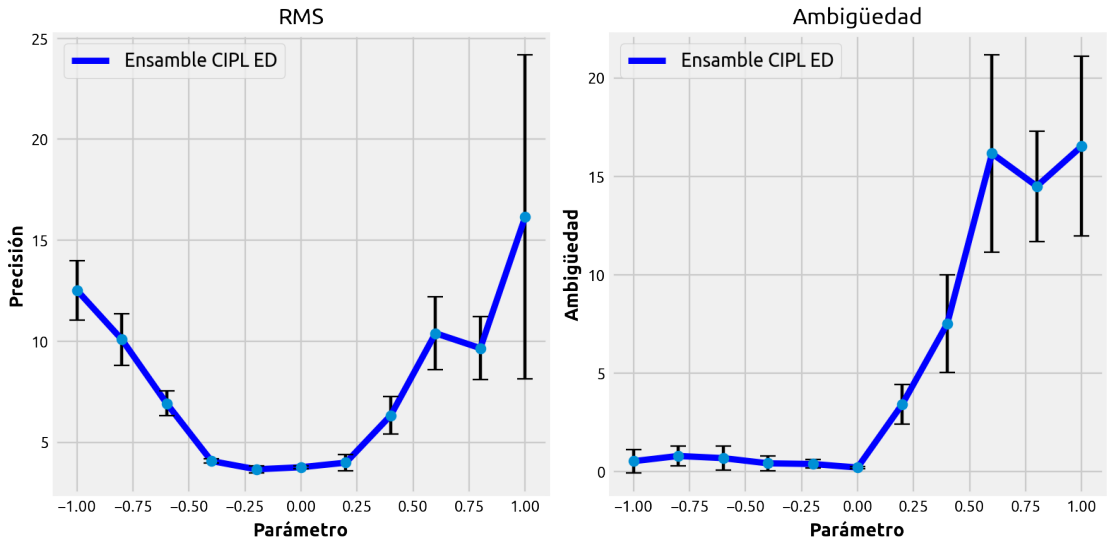
En la figura 4.12 se puede ver como se comporta el desempeño al variar los parámetros del Ensamble CIPL en la misma cantidad $\lambda = \beta$. En el caso de CIPL con Cauchy-Schwartz este muestra que el desempeño o precisión (RMS) y la ambigüedad disminuyen al aumentar el valor de los parámetros, esto es diferente a lo que paso en el caso del problema de regresión Jacobs. Por otro lado, el error de precisión con Cauchy-Schwartz es muy grande en comparación con CIPL Euclidiana.

En cuanto a CIPL Euclidiana alcanza un mínimo en RMS en valores cercanos a cero para los parámetros, pero la ambigüedad (diversidad) aumenta al aumentar el valor de los parámetros. Al comparar los problemas de Friendman y Jacobs se puede ver que para valores positivos λ y β, el aumento en la diversidad medido a través de la ambigüedad implica una

disminución de la precisión del modelo, lo que es coherente con la idea de diversidad. En el caso de valores negativos no se ve un comportamiento definido.



(a) CIPL Cauchy-Schwartz

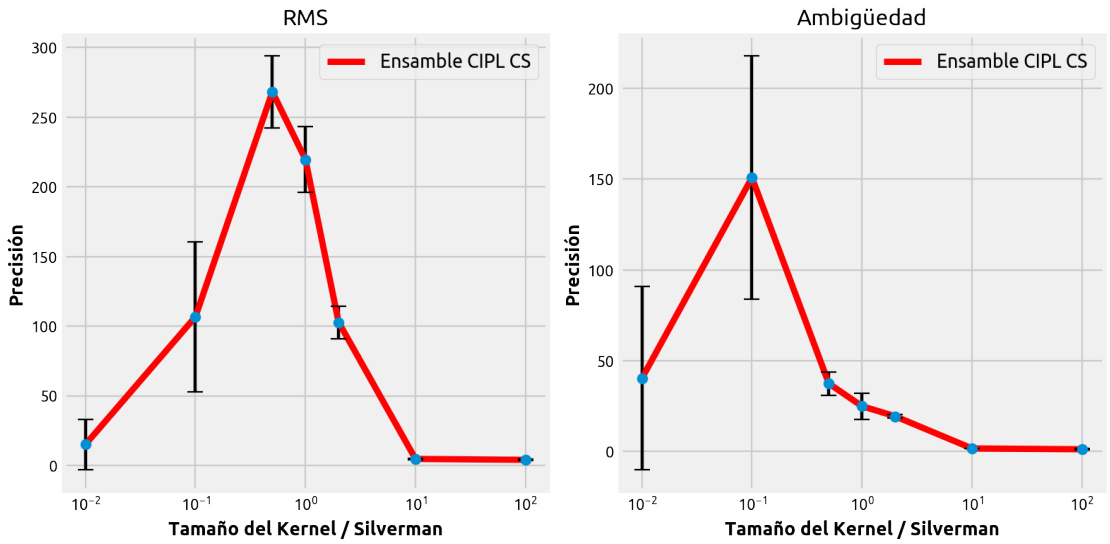


(b) CIPL Euclidiana

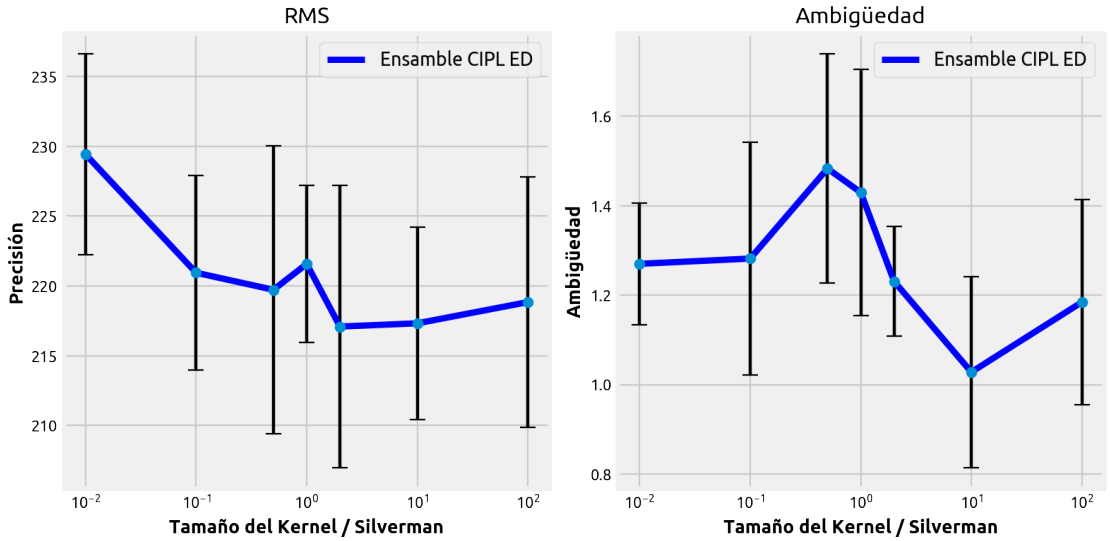
Figura 4.12: Gráficos que muestran como varia el desempeño (RMS) y la diversidad (ambigüedad) al variar el valor de los parámetros del Ensamble CIPL en la misma cantidad, es decir $\lambda = \beta$. Se muestra el desempeño de CIPL obtenido con divergencia (a) Cauchy-Schwartz y (b) Euclidiana. Las barras verticales en las curvas representan la desviación estándar de cada medición, además cada punto de la curva fue obtenido a partir de 10 mediciones.

Prueba variación tamaño del kernel

El resultado obtenido en esta prueba es el que se muestra en la figura 4.13 donde se aprecia un comportamiento diferente que en el problema de regresión de Jacobs, a valores positivos grandes disminuye la ambigüedad y también el desempeño o precisión de los modelos.



(a) CIPL Cauchy-Schwartz



(b) CIPL Euclidiana

Figura 4.13: Gráficos que muestran como varia el desempeño (RMS) y la diversidad (ambigüedad) al variar el valor del tamaño del kernel del Ensamble CIPL

Prueba de los modelos con ruido

En resultado de esta prueba se muestra en la figura 4.14 donde el ensamble CIPL con divergencia Euclidiana muestra un comportamiento similar que en el caso de Jacobs, a medida que disminuye el ruido, mejora su resultado. esto solo se puede explicar por el sesgo $\mu = 0.2$ que se agrega en el ruido. Es importante destacar que en la figura 4.14 no se incluye el resultado de CIPL con Cauchy-Schwartz debido a que el desempeño es mucho peor que los otros modelos y por lo mismo no es comparable.

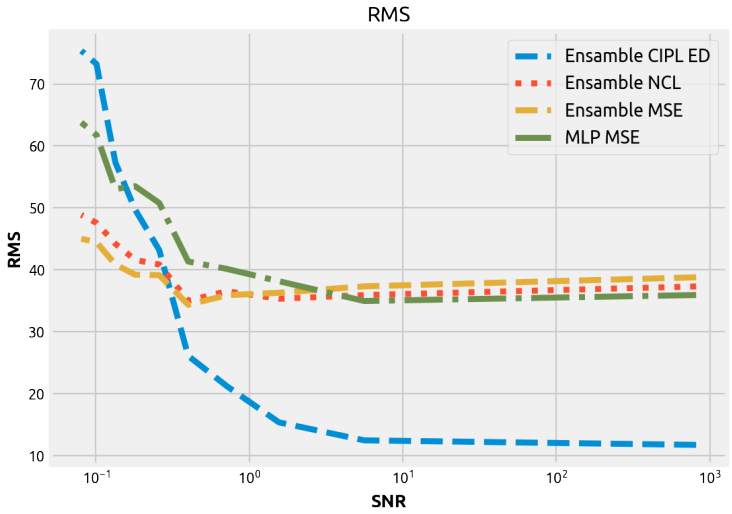


Figura 4.14: Gráfico que muestra como varia el desempeño evaluado con RMS de los distintos modelos de regresión cuando a los datos de entrada se les agrega ruido (Menos es mejor). El ruido es de tipo aditivo Gaussiano $\mathcal{N}(\mu = 0.2, \sigma^2_{ruido})$. El SNR es calculado a partir del ruido y los datos de entrada del modelo: $\text{SNR} = \sigma^2_{\text{entrada}}/\sigma^2_{\text{ruido}}$.

4.3.2. Problemas de Clasificación

Al igual que en la revisión de los problemas de regresión, se incluye información del entrenamiento junto con pruebas para comparar el desempeño de los modelos de ensambles entrenados con CIPL. En cuanto a la salida de los modelos para representar las clases se utiliza la codificación "One Hot Encoding", además para combinar las salidas de los modelos del esnsamble se utiliza la función de votación por mayoría.

Para todos los problemas de clasificación se utiliza el método de Annealing 3.2.1 durante el entrenamiento debido a los buenos resultados obtenidos lo que difiere del caso de los problemas de regresión.

Para la medición de la diversidad durante el entrenamiento se utiliza la métrica Diversidad Generalizada (GD, *Generalized Diversity* en inglés), propuesta por Partridge y Krzanowski

1997 [81] [66] que se define como:

$$GD = 1 - \frac{\sum_{j=1}^{M} \frac{j(j-1)}{M(M-1)} T_j}{\sum_{j=1}^{M} \frac{j}{M} T_j}, \tag{4.4}$$

donde GD es la Diversidad Generalizada, M es la cantidad de modelos que tiene el Ensamble y T_j es la probabilidad de que j modelos clasifiquen incorrectamente un ejemplo. La idea detrás de esta métrica es que la diversidad máxima se alcanza cuando la mala clasificación de un modelo va acompañada de la buena clasificación de los demás, por otro lado, la mínima diversidad es cuando todos los modelos se equivocan. El rango donde se define GD es $[0, 1]$, siendo la máxima diversidad cuando $GD = 1$ y mínima cuando $GD = 0$.

Se incluyen 2 pruebas para evaluar el desempeño de CIPL al modificar sus parámetros y al agregar ruido a los datos de entrada del modelo. Para revisar cómo afecta al desempeño el cambio de parámetros, se utiliza las mismas restricciones del problema de regresión de Jacobs, es decir, se restringe la variación de los parámetros β y λ de forma que ambos tienen el mismo valor para simplificar los cálculos y poder mostrar el resultado en el plano 2D. Además su valor se encuentra en el rango $[-1, 1]$, esta prueba es similar en todos los siguientes problemas de clasificación. Para la prueba con ruido se utiliza un ruido aditivo gaussiano que distribuye $\mathcal{N}(\mu = 0.2,\, \sigma^2_{ruido})$.

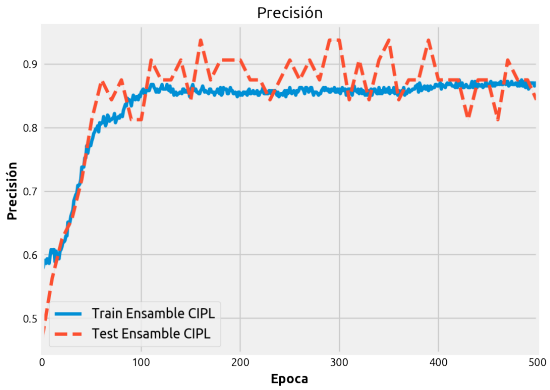
Australian Credit Card

El modelo de Ensamble con CIPL que mejor resultado entrego es el que utiliza la divergencia Cauchy-Schwarz con los parámetros $\beta = 0.1$ y $\lambda = 0.5$, además el factor de aprendizaje $\eta = -0.03$. Para el Annealing se consideró: $\sigma^- = 0.5 \cdot \sigma_s$ y $\sigma^+ = 1.5 \cdot \sigma_s$ donde σ_s corresponde al tamaño del kernel calculado con Silverman: $\sigma_s = 0.4882$.

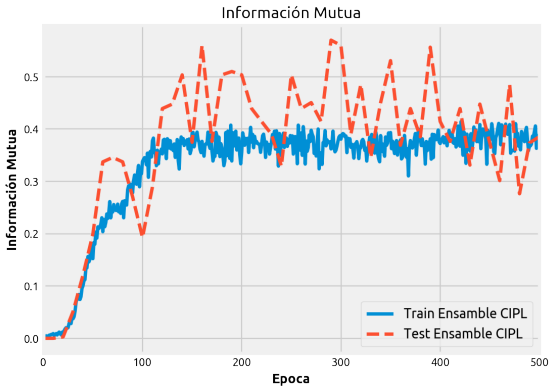
Los demás modelos fueron entrenados con un factor de aprendizaje de $\eta = 0.3$, en cuanto al parámetro de NCL el que mejor resultados entrega es $\lambda = 0.6$. Todos los modelos utilizados tienen 32 neuronas, además cada neurona tiene como función de activación en la capa oculta y la capa de salida la función sigmoide.

El resultado del entrenamiento del Ensamble entrenado con CIPL se presenta en la figura 4.15, donde se muestra que durante el entrenamiento CIPL maximiza tanto el desempeño (precisión) como la información mutua. En cuanto al desempeño final CIPL obtuvo un promedio de 0.8779 con el conjunto de entrenamiento y 0.8551 con el conjunto de prueba, tal como se puede ver en la tabla 4.3.

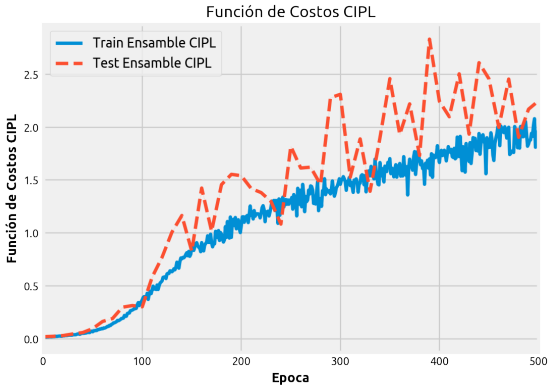
En la figura 4.16 se puede ver la función de probabilidad o pdf del error de cada mejor modelo obtenido. Se puede apreciar que con CIPL a pesar de no reducir el error de predicción de la clase 1 (salida 1) si pudo reducir mucho mejor el error de predicción de la clase 2 (salida 2) respecto a los otros métodos. Al revisar los resultados de precisión promedios en la tabla 4.3 se aprecia que CIPL aventaja de forma marginal a los demás modelos pero con una alta varianza respecto a los otros métodos.



(a) RMS



(b) Información Mutua

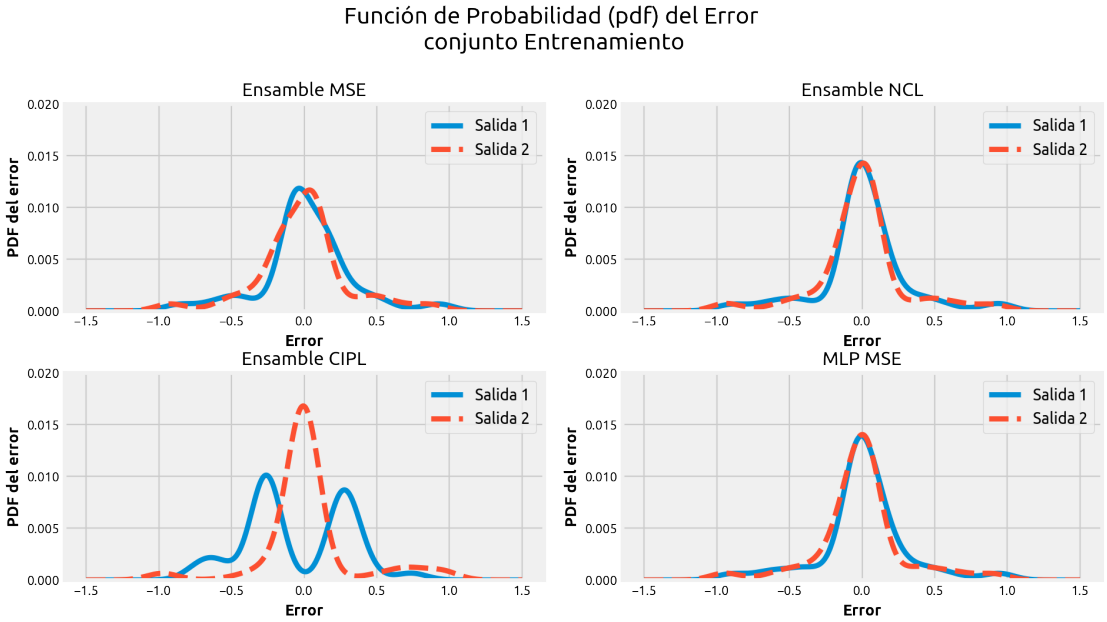


(c) Función de costo CIPL

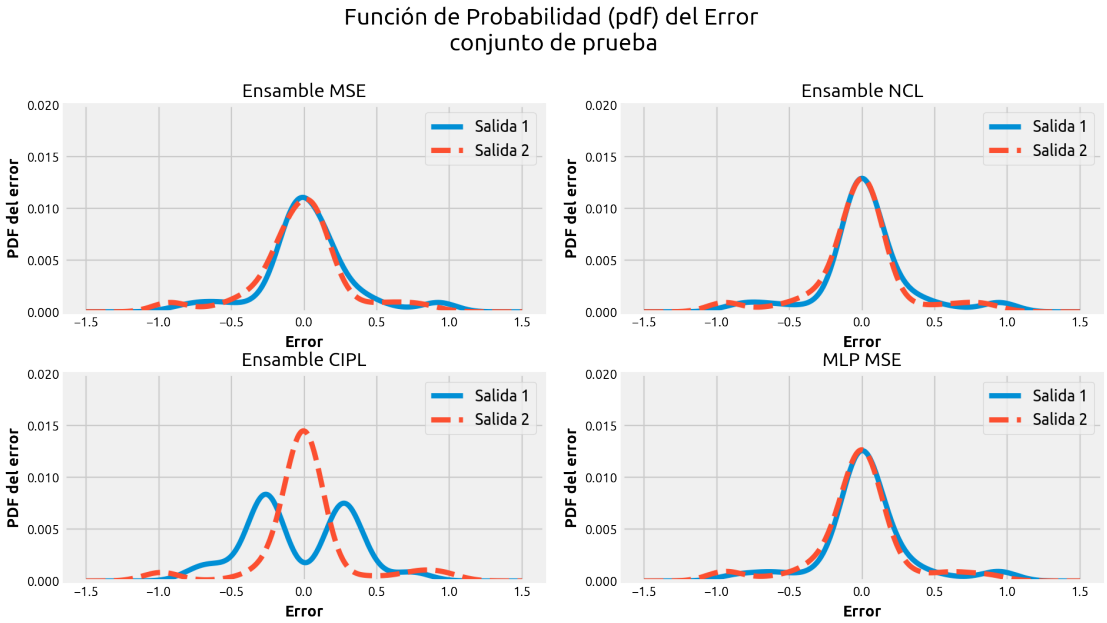
Figura 4.15: Resultado del entrenamiento del mejor modelo CIPL obtenido. En el gráfico (a) se puede ver el progreso de la precisión del clasificador, (b) la Información Mutua de Shannon y (c) la maximización de la función de costo CIPL de Cauchy-Schwartz. En todos los gráficos se muestra el progreso de las métricas durante el entrenamiento y sobre los conjuntos de entrenamiento y prueba (test). La Información Mutua de Shannon es estimada con ventanas de Parzen y se calcula sobre los datos de salida del ensamble y la salida esperada.

Método	**Promedio Precisión**	**min**	**max**
Ensamble CIP	0.8551 ± 0.1740	0.7830	0.8812
Ensamble NCL	0.8401 ± 0.0534	0.8153	0.8834
Ensamble MSE	0.8401 ± 0.0678	0.8200	0.8834
MLP MSE	0.8498 ± 0.0987	0.8001	0.8756

Tabla 4.3: Tabla con los resultados de desempeño de los distintos modelos evaluados con la base de datos Australian Credit Card y con la aplicación del método de validación cruzada de 10 iteraciones (10-fold). La fila en gris señala el modelo con mejor precisión.



(a) Conjunto Entrenamiento

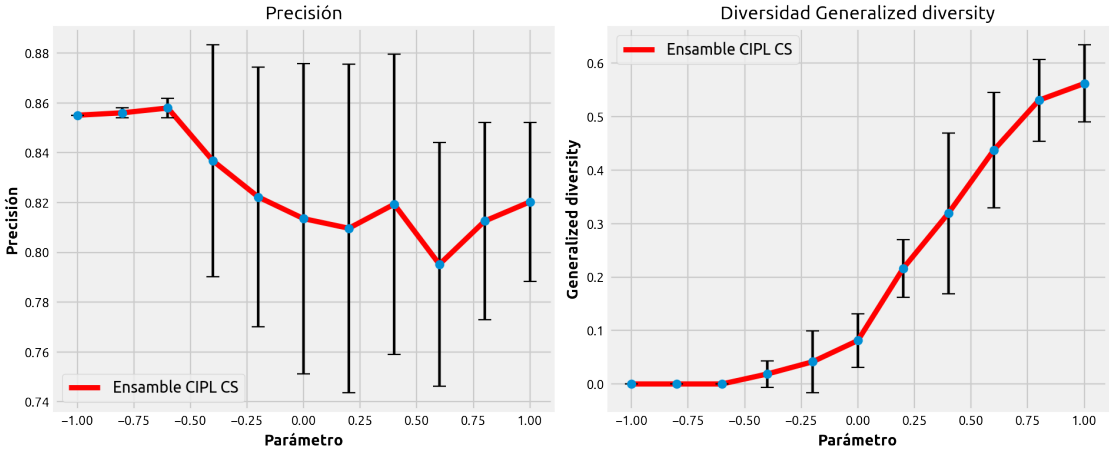


(b) Conjunto de prueba

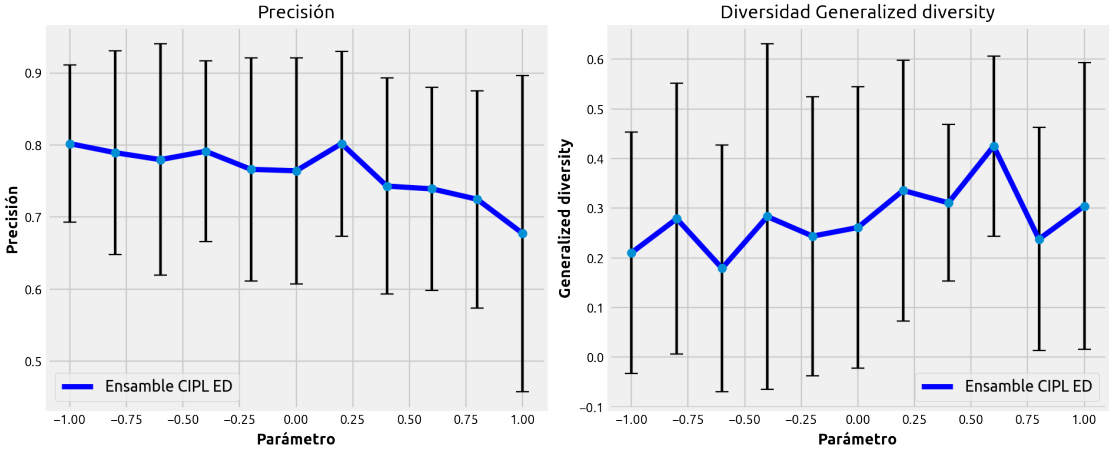
Figura 4.16: Función de probabilidad o pdf del error de los distintos modelos analizados. Las pdfs se obtuvieron para los conjuntos de entrenamiento y prueba respectivamente. Las salidas enumeradas corresponden a la salida de las clases correspondientes (One Hot Encoding).

Prueba variación de parámetros

La figura 4.17 muestra que para CIPL con Cauchy-Schwartz aunque los valores promedios de precisión son mayores para valores negativos de los parámetros la Diversidad Generalizada es cercana a cero, o lo que es lo mismo que los modelos del ensamble son exactamente iguales. Por otro lado, para valores positivos, aunque el desempeño disminuye la diversidad aumenta, de forma que el aumento de la sinergia y la redundancia tiene un efecto directo en la diversidad del Ensamble. Para CIPL con divergencia Euclidiana el efecto anterior no es claro.



(a) CIPL Cauchy-Schwartz



(b) CIPL Euclidiana

Figura 4.17: Gráficos que muestran como varia el desempeño (precisión) y la diversidad (Generalized Diversity) al variar el valor de los parámetros del Ensamble CIPL en la misma cantidad, es decir $\lambda = \beta$. Se muestra el desempeño de CIPL obtenido con divergencia (a) Cauchy-Schwartz y (b) Euclidiana. Las barras verticales en las curvas representan la desviación estándar de cada medición, además cada punto de la curva fue obtenido a partir de 10 mediciones.

Prueba de los modelos con ruido

En la figura 4.18 se puede ver que prácticamente todos los modelos tienen el mismo comportamiento con el ruido, es decir, reducción de su desempeño.

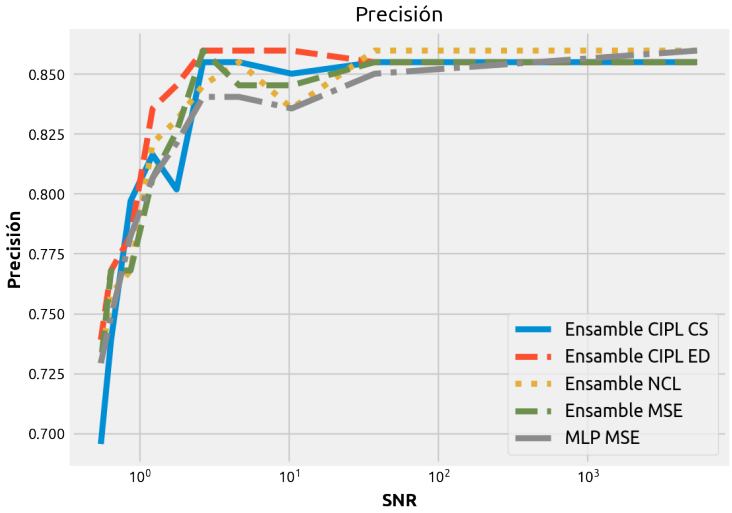


Figura 4.18: Gráfico que muestra como varia el desempeño o precisión de los distintos modelos de clasificación cuando a los datos de entrada se les agrega ruido (más es mejor). El ruido es de tipo aditivo Gaussiano $\mathcal{N}(\mu = 0.2, \sigma^2_{ruido})$. El SNR es calculado a partir del ruido y los datos de entrada del modelo: $\text{SNR} = \sigma^2_{\text{entrada}}/\sigma^2_{\text{ruido}}$.

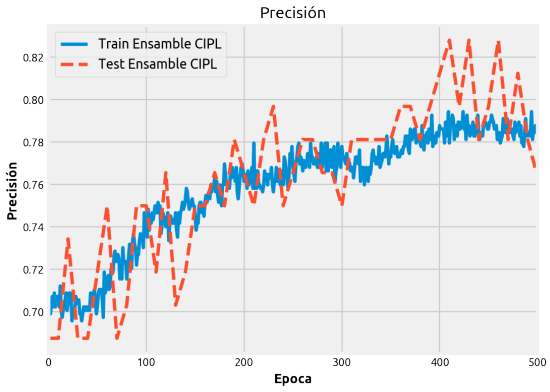
German Numer

El modelo de Ensamble con CIPL que mejor resultado entregó es el que utiliza la divergencia Cauchy-Schwarz con los parámetros $\beta = 0.1$ y $\lambda = 0.5$ y el factor de aprendizaje $\eta = -0.2$. Para el Annealing se consideró: $\sigma^- = 0.8 \cdot \sigma_s$ y $\sigma^+ = 2.0 \cdot \sigma_s$ donde σ_s corresponde al tamaño del kernel calculado con Silverman: $\sigma_s = 0.4523$.

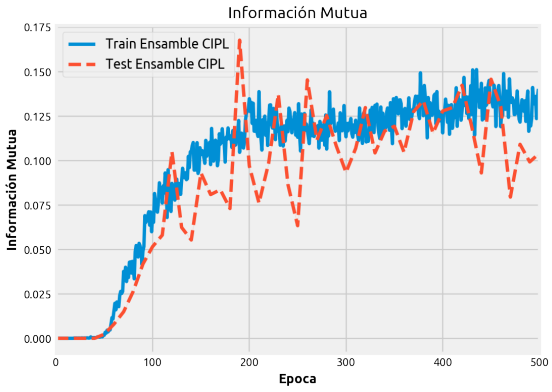
Todos los modelos utilizados tienen 40 neuronas, además cada neurona tiene como función de activación en la capa oculta y la capa de salida a la función sigmoide. En cuanto a la cantidad de modelos por ensamble, esta es de 4 redes MLP de 10 neuronas cada uno. Además, los modelos fueron entrenados con un factor de aprendizaje de $\eta = 0.2$, en cuanto al parámetros de NCL el que mejor resultados entrega es $\lambda = 0.8$.

Como se puede ver en la figura 4.19 durante el entrenamiento CIPL maximiza tanto el desempeño (precisión) como la Información Mutua de Shannon, en cuanto al desempeño final CIPL obtuvo un promedio de precisión de 0.7756 con el conjunto de entrenamiento y 0.7645 con el conjunto de prueba, tal como se puede ver en la tabla 4.4.

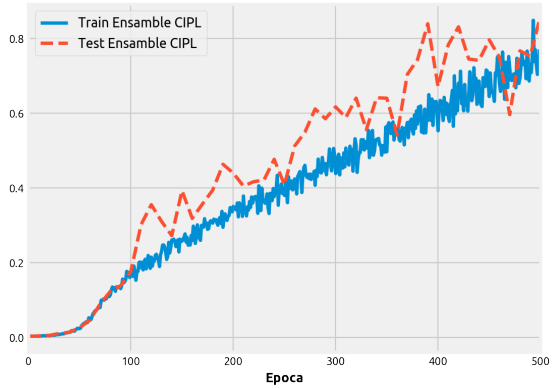
En la figura 4.21 se muestra la función de probabilidad o pdf del error de cada uno de los modelos analizados, los cuales son los que mejor resultados mostraron para cada tipo. Al revisar este resultado se puede ver que CIPL a diferencia de los demás puede al menos centrar



(a) Precisión



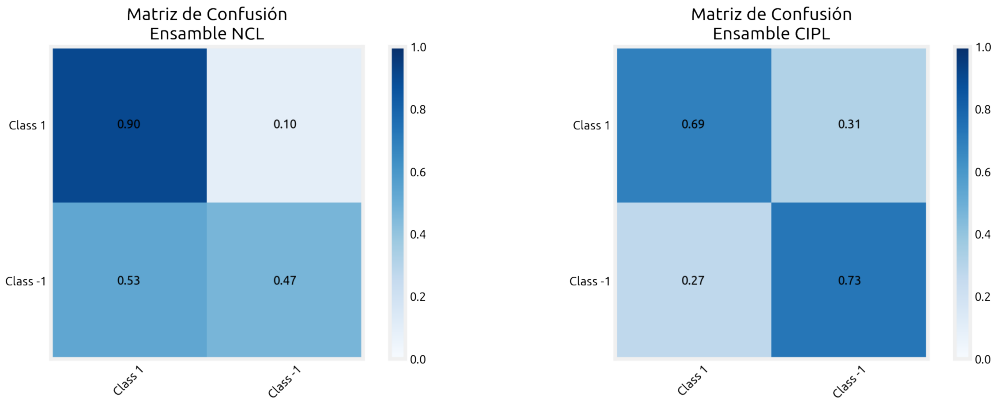
(b) Información Mutua



(c) Función de Costo CIPL

Figura 4.19: Resultado del entrenamiento del mejor modelo CIPL obtenido. En el gráfico (a) se puede ver el progreso de la precisión del clasificador, (b) la Información Mutua de Shannon y (c) la maximización de la función de costo CIPL de Cauchy-Schwartz. En todos los gráficos se muestra el progreso de las métricas durante el entrenamiento y sobre los conjuntos de entrenamiento y prueba (test). La Información Mutua de Shannon es estimada con ventanas de Parzen y se calcula sobre los datos de salida del ensamble y la salida esperada.

el error de una de las clases en cero (clase -1 correspondiente a la salida 1 del modelo). Por otro lado, al comparar las matrices de confusión de la figura 4.20 , se observa que el método NCL logra un alto desempeño en la clase 1 en desmedro de la clase -1 (esto pasa de la misma forma con MLP y el Ensamble MSE), esto es diferente en el caso de CIPL el que logra diferenciar mejor ambas clases. Al respecto es importante destacar que la clase 1 corresponde al 70 % de los ejemplos, por esa razón es importante revisar con cuidado los datos de la tabla 4.4, dado que la precisión es calculada sin considerar el desbalance de los ejemplos de las clases.



(a) Matriz Confusión NCL

(b) Matriz Confusión CIPL

Figura 4.20: Función de probabilidad o pdf del error de los distintos modelos analizados. Las pdfs se obtuvieron para los conjuntos de entrenamiento y prueba respectivamente.

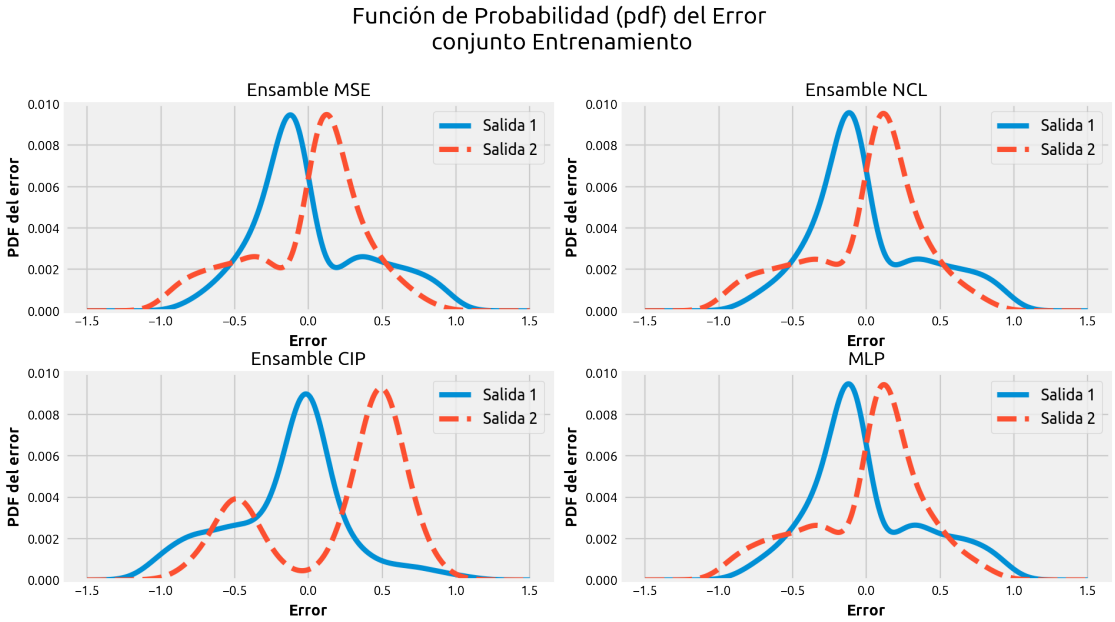
A continuación, se presenta la tabla 4.4 con los datos de comparación utilizando validación cruzada, estos resultados muestran que CIPL aventaja de forma marginal a los demás modelos.

Método	**Promedio Precisión**	**min**	**max**
Ensamble CIP	0.7645 ± 0.0267	0.7155	0.7767
Ensamble NCL	0.7534 ± 0.0123	0.7345	0.7890
Ensamble MSE	0.7534 ± 0.0143	0.7345	0.7645
MLP MSE	0.7534 ± 0.0067	0.7345	0.7401

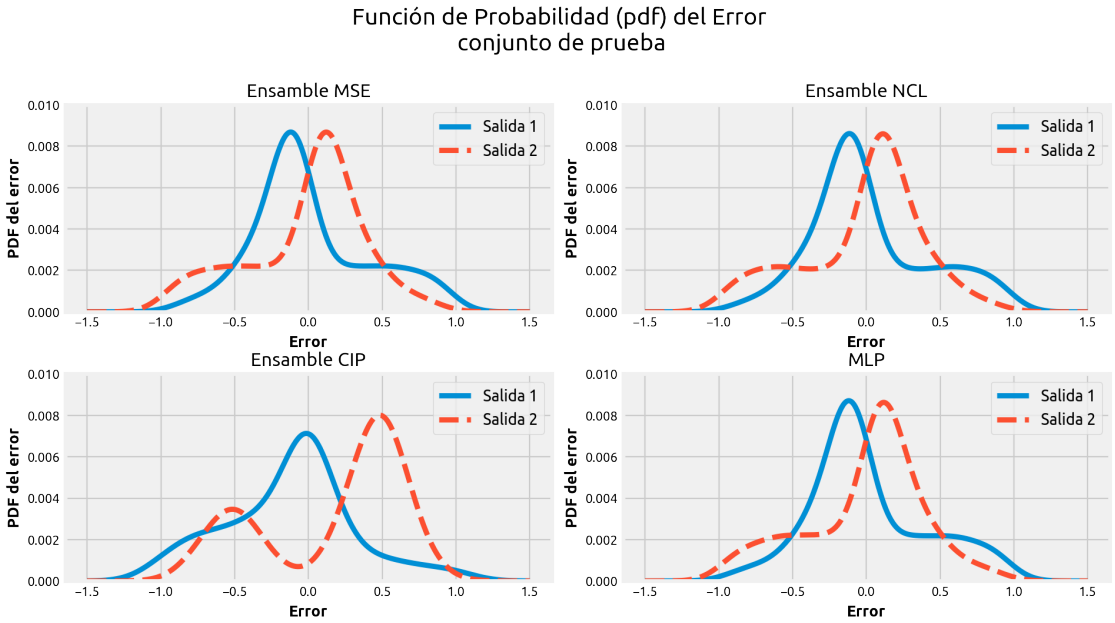
Tabla 4.4: Tabla con los resultados de desempeño de los distintos modelos evaluados con la base de datos German Numer y con la aplicación del método de validación cruzada de 10 iteraciones (10-fold). La fila en gris señala el modelo con mejor precisión.

Prueba variación de parámetros

La figura 4.22 muestra que para CIPL con Cauchy-Schwartz aunque los valores promedios de precisión son mayores para valores negativos de los parámetros la Diversidad Generalizada es cercana a cero. Por otro lado, para valores positivos, aunque el desempeño disminuye la diversidad aumenta, de la misma forma que en el caso del problema de Australian Credit Card. Para CIPL con divergencia Euclidiana el efecto anterior no es claro, se observa que estos valores son prácticamente constantes.

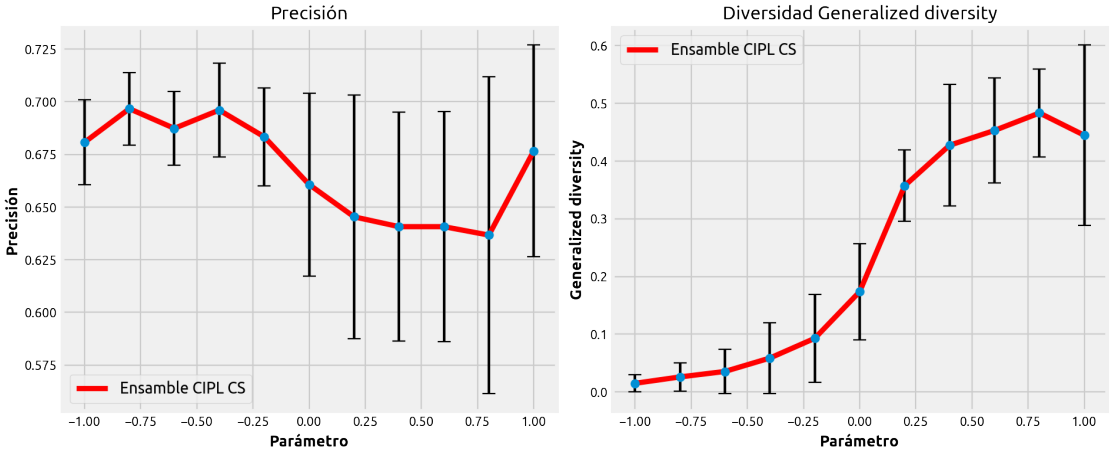


(a) Conjunto Entrenamiento

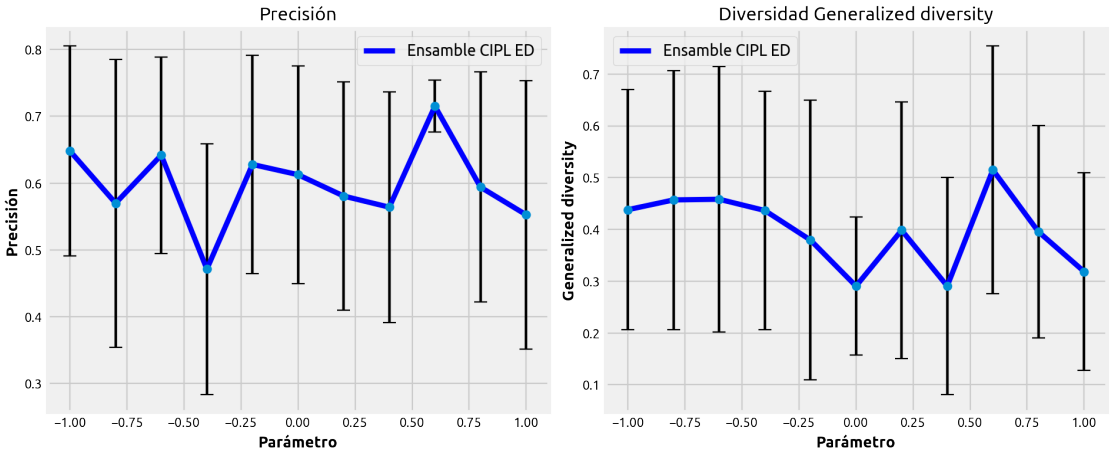


(b) Conjunto de prueba

Figura 4.21: Función de probabilidad o pdf del error de los distintos modelos analizados. Las pdfs se obtuvieron para los conjuntos de entrenamiento y prueba respectivamente. Las salidas enumeradas corresponden a la salida de las clases correspondientes (One Hot Encoding).



(a) CIPL Cauchy-Schwartz



(b) CIPL Euclidiana

Figura 4.22: Gráficos que muestran como varia el desempeño (precisión) y la diversidad (Generalized Diversity) al variar el valor de los parámetros del Ensamble CIPL en la misma cantidad, es decir $\lambda = \beta$. Se muestra el desempeño de CIPL obtenido con divergencia (a) Cauchy-Schwartz y (b) Euclidiana. Las barras verticales en las curvas representan la desviación estándar de cada medición, además cada punto de la curva fue obtenido a partir de 10 mediciones.

Prueba de los modelos con ruido

En la figura 4.23 se puede ver que prácticamente todos los modelos tienen el mismo comportamiento con el ruido, es decir, reducción de su desempeño. Sin embargo, con CIPL se observa que existe una reducción en la precisión. En este problema CIPL no se muestra robusto al ruido, aunque CIPL con Cauchy-Schwartz presenta un mejor comportamiento respecto a CIPL con divergencia Euclidiana.

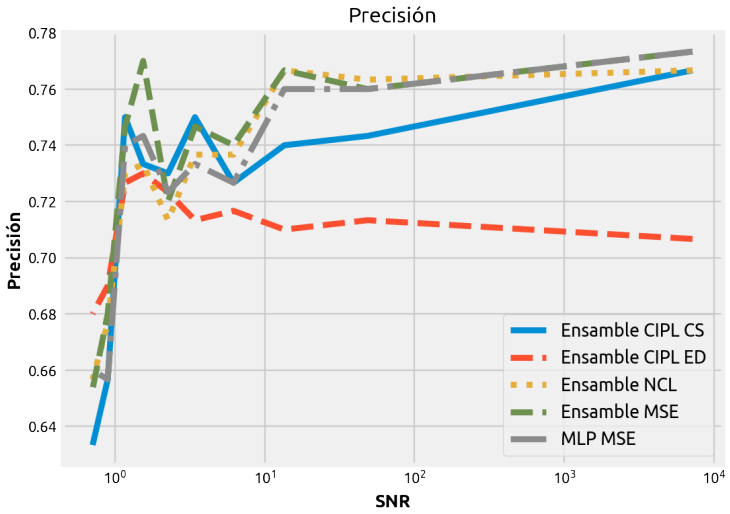


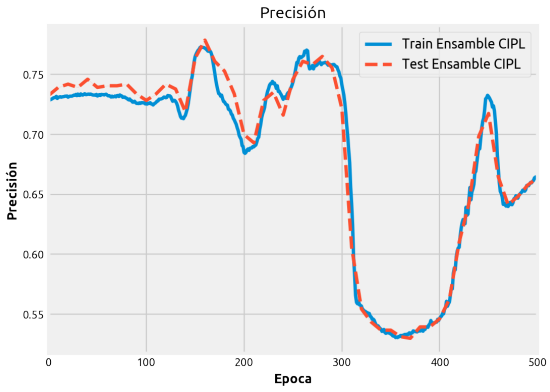
Figura 4.23: Gráfico que muestra como varia el desempeño o precisión de los distintos modelos de clasificación cuando a los datos de entrada se les agrega ruido (más es mejor). El ruido es de tipo aditivo Gaussiano $\mathcal{N}(\mu = 0.2, \sigma^2_{ruido})$. El SNR es calculado a partir del ruido y los datos de entrada del modelo: $\text{SNR} = \sigma^2_{\text{entrada}}/\sigma^2_{\text{ruido}}$.

Satimage

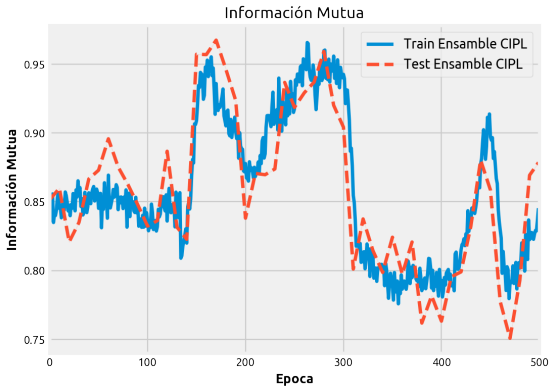
El modelo de Ensamble con CIPL que mejor resultado entrego es el que utiliza la divergencia Cauchy-Schwarz con los parámetros $\beta = 0.1$ y $\lambda = 0.1$ y el factor de aprendizaje $\eta = -0.1$. Para el Annealing se consideró: $\sigma^- = 0.5 \cdot \sigma_s$ y $\sigma^+ = 1.5 \cdot \sigma_s$ donde σ_s corresponde al tamaño del kernel calculado con Silverman: $\sigma_s = 1.874$.

Todos los modelos utilizados tienen 84 neuronas, además cada neurona tiene como función de activación en la capa oculta y la capa de salida a la función sigmoide. En cuanto a la cantidad de modelos por ensamble, esta es de 4 redes MLP de 21 neuronas cada uno. Además, los modelos fueron entrenados con un factor de aprendizaje de $\eta = 0.1$, en cuanto al parámetros de NCL el que mejor resultados entrega es $\lambda = 0.6$.

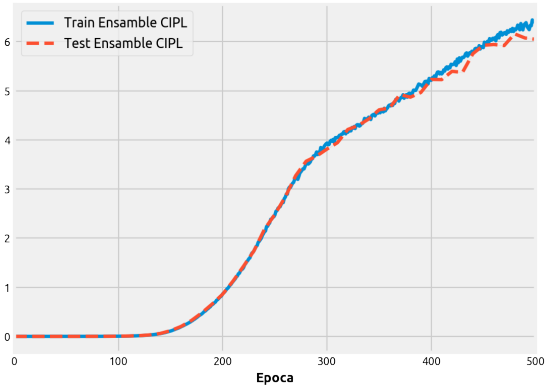
Como se puede ver en la figura 4.24 durante el entrenamiento CIPL no puede maximizar la Información Mutua de Shannon a pesar de maximizar la función de costo CIPL si es maximizada. Por otro lado, la precisión alcanzada es muy baja, en promedio la precisión con CIPL es de 0.7589 con el conjunto de entrenamiento y 0.7001 con el conjunto de prueba, tal como se puede ver en la tabla 4.5.



(a) Precisión



(b) Información Mutua



(c) Función de Costo CIPL

Figura 4.24: Resultado del entrenamiento del mejor modelo CIPL obtenido. En el gráfico (a) se puede ver el progreso de la precisión del clasificador, (b) la Información Mutua de Shannon y (c) la maximización de la función de costo CIPL de Cauchy-Schwartz. En todos los gráficos se muestra el progreso de las métricas durante el entrenamiento y sobre los conjuntos de entrenamiento y prueba (test). La Información Mutua de Shannon es estimada con ventanas de Parzen y se calcula sobre los datos de salida del ensamble y la salida esperada.

Durante el entrenamiento CIPL no solo no logra mantener un buen desempeño (precisión) sino también no es capaz de maximizar la información mutua, esto no es culpa de los datos sino del método CIPL, dado que con otros métodos se logra obtener resultados cercanos al 90 % de precisión. Además, al revisar la función de costo CIPL durante el entrenamiento, ver figura 4.24 (c), la maximización de CIPL se logra pero no así la maximización de la Información Mutua.

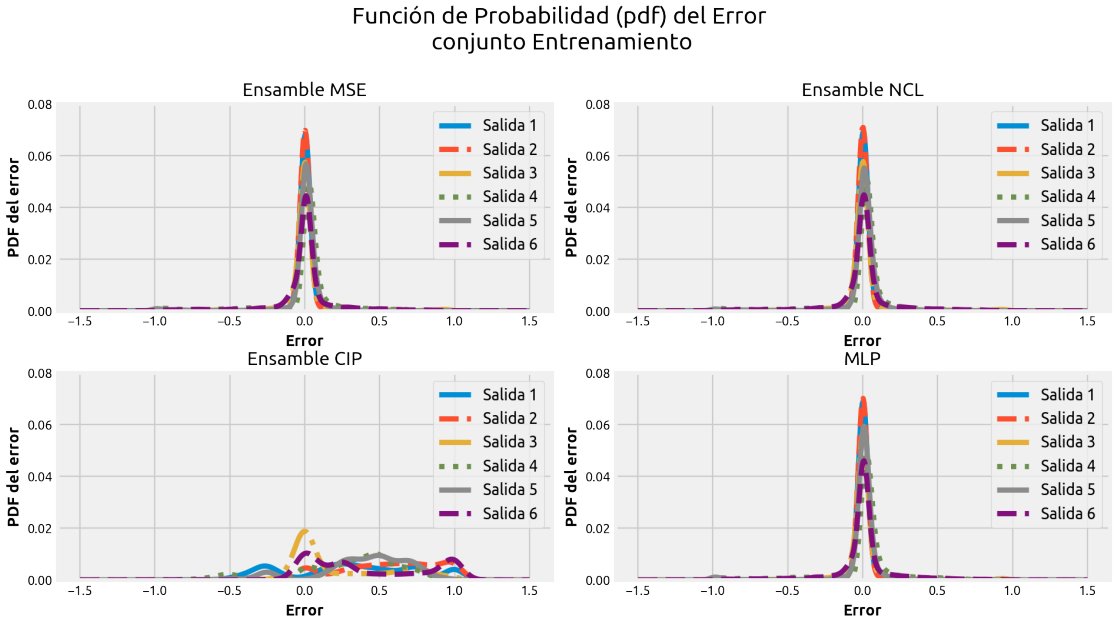
Al revisar otros problemas de clasificación de más de 2 clases se observo que CIPL tiene problemas al maximizar la Información Mutua de Shannon. Para poder comparar el desempeño de los demás modelos se presenta la distribución del error de predicción de cada modelo en la figura 4.25. Al comparar los resultados de la pdf del error se observa que CIPL tiene un pésimo desempeño respecto a los demás.

Método	**Promedio Precisión**	**min**	**max**
Ensamble CIP	0.7001 ± 0.2012	0.50397	0.7377
Ensamble NCL	0.9634 ± 0.0165	0.9356	0.9745
Ensamble	0.9546 ± 0.0189	0.9355	0.9745
MLP RMS	0.9455 ± 0.0159	0.9355	0.9745

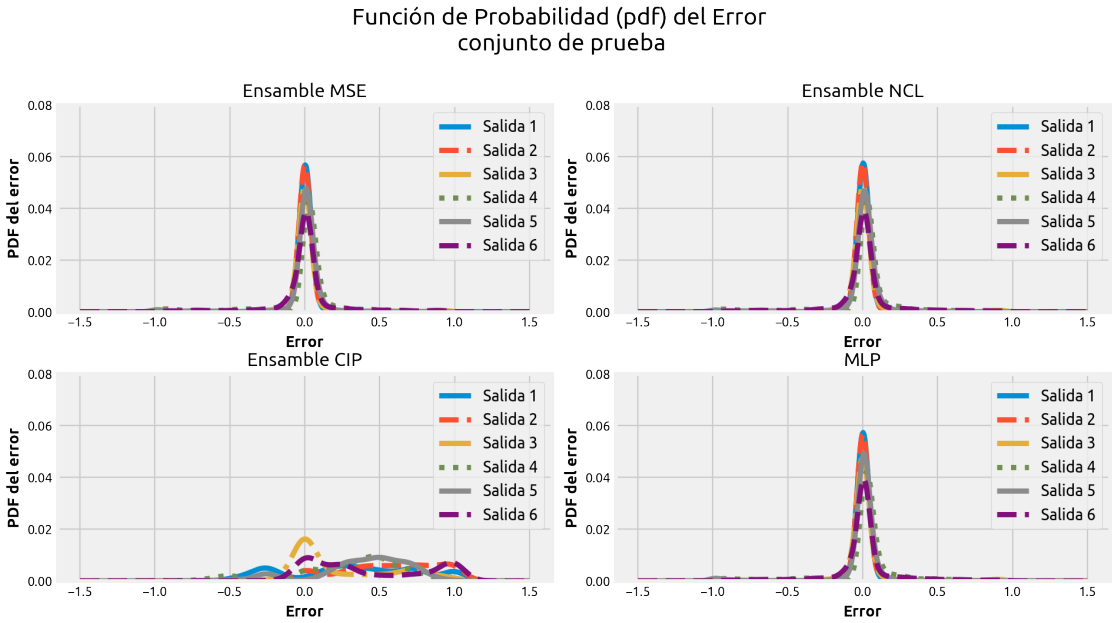
Tabla 4.5: Tabla con los resultados de desempeño de los distintos modelos evaluados con la base de datos Satimage y con la aplicación del método de validación cruzada de 10 iteraciones (10-fold). La fila en gris señala el modelo con mejor precisión.

Prueba variación de parámetros

En la figura 4.26 se puede ver que para valores negativos de los parámetros el desempeño del Ensamble CIPL con Cauchy-Schwartz es muy bajo y la diversidad alta, lo cual no es extraño considerando que en este caso los sub-modelos tienen un mayor margen para ser diferentes (más de 2 clases y error grande en la predicción). Por otro lado, para valores positivos aunque aumenta la precisión de predicción no es así con la diversidad como en los otros casos. En cuanto a CIPL con divergencia Euclidiana se observa un comportamiento constante al modificar los parámetros, es similar a lo observado en los otros casos.

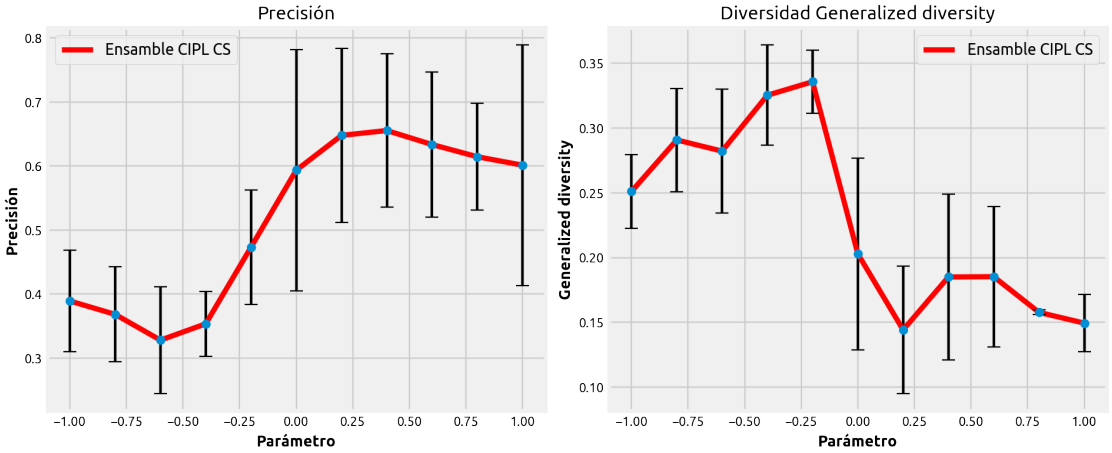


(a) Conjunto Entrenamiento

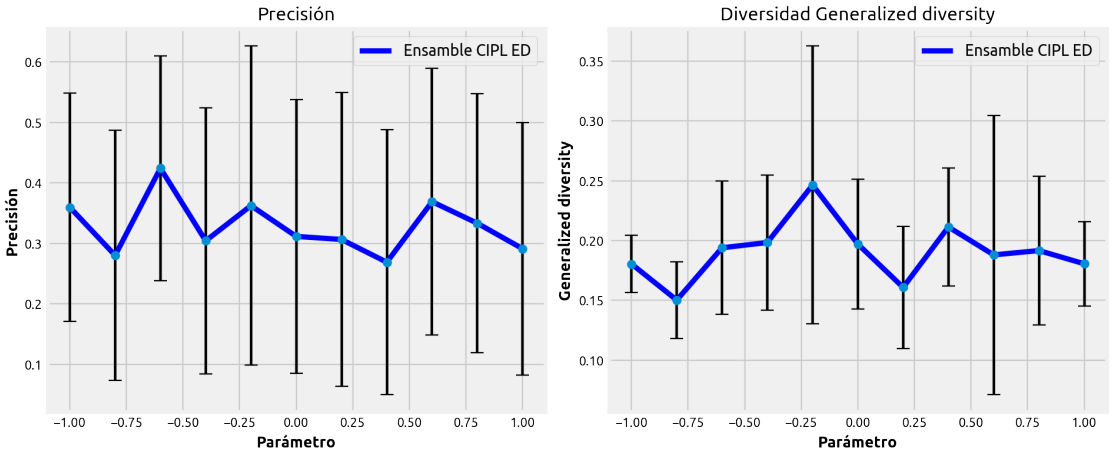


(b) Conjunto de prueba

Figura 4.25: Función de probabilidad o pdf del error de los distintos modelos analizados. Las pdfs se obtuvieron para los conjuntos de entrenamiento y prueba respectivamente. Las salidas enumeradas corresponden a la salida de las clases correspondientes (One Hot Encoding).



(a) RMS



(b) Ambigüedad

Figura 4.26: Gráficos que muestran como varia el desempeño (precisión) y la diversidad (Generalized Diversity) al variar el valor de los parámetros del Ensamble CIPL en la misma cantidad, es decir $\lambda = \beta$. Se muestra el desempeño de CIPL obtenido con divergencia (a) Cauchy-Schwartz y (b) Euclidiana. Las barras verticales en las curvas representan la desviación estándar de cada medición, además cada punto de la curva fue obtenido a partir de 10 mediciones.

Prueba de los modelos bajo ruido

En la figura 4.27 se muestra que prácticamente todos los modelos tienen el mismo comportamiento con el ruido. pero que con CIPL este comportamiento es mucho menos notorio por la baja precisión alcanzada por ambos métodos de CIPL.

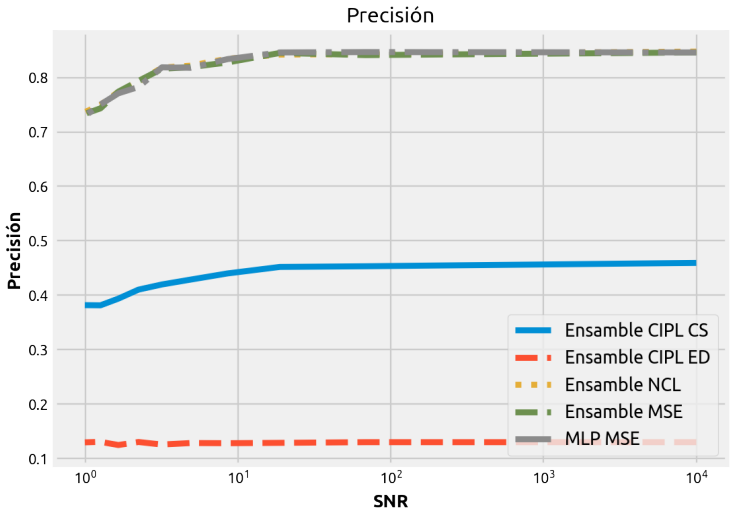


Figura 4.27: Gráfico que muestra como varia el desempeño o precisión de los distintos modelos de clasificación cuando a los datos de entrada se les agrega ruido (más es mejor). El ruido es de tipo aditivo Gaussiano $\mathcal{N}(\mu = 0.2, \sigma^2_{ruido})$. El SNR es calculado a partir del ruido y los datos de entrada del modelo: $\text{SNR} = \sigma^2_{\text{entrada}}/\sigma^2_{\text{ruido}}$.

4.4. Análisis

En el análisis del comportamiento de CIPL se consideran propiedades intrínsecas de este criterio, como son sus hiperparámetros, además de propiedades estructurales del Ensamble, como la cantidad de modelos diferentes que este puede contener.

4.4.1. Efecto de los hiperparámetros

Al revisar los datos obtenidos se desprende que para los hiperparámetros β y λ en el rango $[-1, 1]$ CIPL CS mejora su desempeño en valores positivos, mientras que en el caso de CIPL ED su mejora es marginal. En cuanto a la diversidad, está mejora en valores positivos también, aunque con mucha más fuerza en CIPL CS, incluso en problemas de clasificación se puede ver una tendencia de alcanzar un máximo de diversidad (medido con GD (4.4)) cuando β y λ se aproximan a 1.

En cuanto a valores negativos de β y λ, según lo visto en las pruebas, generan modelos con menor diversidad, pero su precisión no se ve afectada mayormente, esto se puede deber a

que, aunque los modelos tienden a ser similares por la acción de la sinergia y la redundancia, la relevancia ayuda a mantener la precisión.

Para el tamaño del kernel su reducción aumenta la ambigüedad alcanzada en el caso de CIPL aplicado a regresión, lo que al mismo tiempo empeora el desempeño (medido con RMS) del modelo. La reducción del tamaño del kernel implica que los modelos se hacen más diferentes entre sí pero a costa de aumentar el error de predicción del Ensamble, lo que va en contra de la idea intuitiva de la diversidad que busca que la diferencia de los modelos implique una mejora general del resultado del Ensamble. En el caso de aumentar el tamaño del kernel aunque se logra mejorar el desempeño, la variación respecto a lo alcanzado por la etapa previa de pre-entrenamiento generada con MSE es pequeña, lo que implica que existe un compromiso entre el tamaño del kernel y su desempeño, lo cual concuerda con la teoría.

4.4.2. Efecto de variar la cantidad de modelos y neuronas

Para revisar el efecto de variar la cantidad de modelos y neuronas se utilizan los Ensambles entrenados con NCL y CIPL para comparar sus resultados. Además, para obtener cada punto de los gráficos, las pruebas fueron realizadas utilizando validación cruzada de 10 iteraciones (10-fold) para evitar problemas por la selección de los datos. Se uso la base de datos Australian Credit Card, además se mantienen las configuraciones de parámetros presentadas en la sección de resultados para cada método.

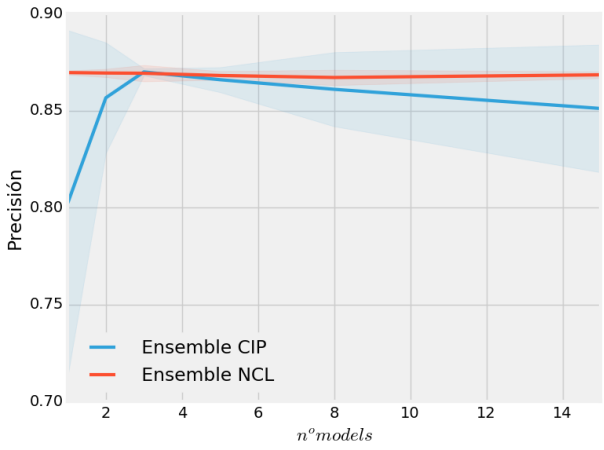


Figura 4.28: Gráficos que compara la precisión de los Ensambles NCL y CIP (Cauchy-Schwarz) cuando varia la cantidad de modelos que estos contienen. La sombra sobre cada curva representa la dispersión de resultados observada.

Primeramente, se revisa el efecto en el desempeño de la variación de modelos en el Ensamble, el resultado obtenido se muestra en la figura 4.28 donde se puede apreciar que el aumento de la cantidad de modelos no implica un aumento importante en el desempeño del Ensamble entrenado con NCL. Sin embargo, aunque con CIPL tiene un desempeño menor en promedio, se obtiene mejores resultados entre 3 y 4 modelos por Ensamble. Además, en CIPL es más evidente que el uso de más modelos implica mejores resultados, aunque es importante también destacar que existe un límite superior desde el cual el desempeño comienza

a decrecer.

Para el caso de la variación de la cantidad de modelos en el Ensamble se realiza una prueba donde se utilizan distintos modelos que difieren en la cantidad de neuronas que tiene cada modelo. La cantidad de modelos es de 3 para reducir la carga computacional. El resultado de esta prueba se puede ver en la figura 4.29.

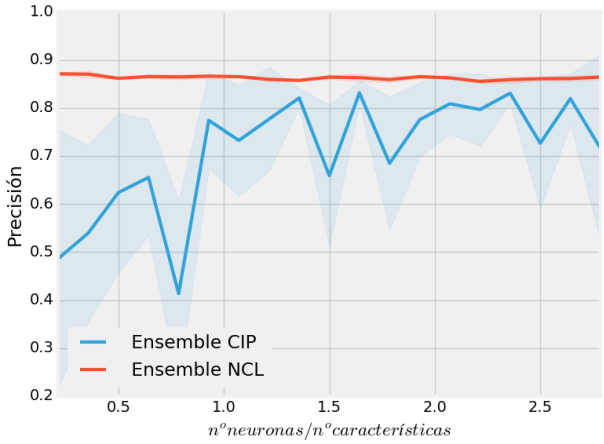


Figura 4.29: Gráficos que comparan la precisión de los Ensambles NCL y CIP (Cauchy-Schwarz) Al variar la cantidad de neuronas utilizadas en cada modelo que los conforman. La sombra sobre cada curva representa la dispersión de resultados observada.

En esta prueba pasa lo mismo con NCL que el anterior, el aumento de la complejidad del modelo no implica aumento de la precisión de este, sin embargo, para CIPL el aumento de neuronas implica una mejora en el desempeño del modelo, aunque esta tendencia no es constante. En resumen, en esta prueba NCL no mejora sus resultados aumentando su complejidad a diferencia de CIPL que, si lo logra, en especial aumentando la cantidad de modelos, aunque existe un límite.

4.4.3. Comportamiento con ruido y valores atípicos

Las pruebas de ruido en los modelos realizados sobre los mejores modelos de cada tipo evaluado en este trabajo no son concluyentes, por una parte, muestran que los modelos se ven afectados de la misma forma por el ruido, es decir, que mientras más ruido contengan los datos de entrada menor es el desempeño de estos.

Para revisar cómo afecta el cambio del tamaño del kernel en el desempeño del Ensamble entrenado con CIPL se realiza una prueba que compara distintos modelos entrenados con distintos tamaños del kernel, la base de datos utilizada es la de Autralian Credit Card, el resultado es el que se muestra en la figura 4.30. El ruido utilizado es de tipo aditivo Gaussiano $\mathcal{N}(\mu = 0, \sigma^2_{ruido})$.

Al revisar el resultado se puede ver que al aumentar el tamaño del kernel durante el entrenamiento permite que los modelos generados puedan ser más robustos al ruido en especial para CIPL Cauchy-Schwarz, esto concuerda con lo esperado, dado que el aumento del tamaño

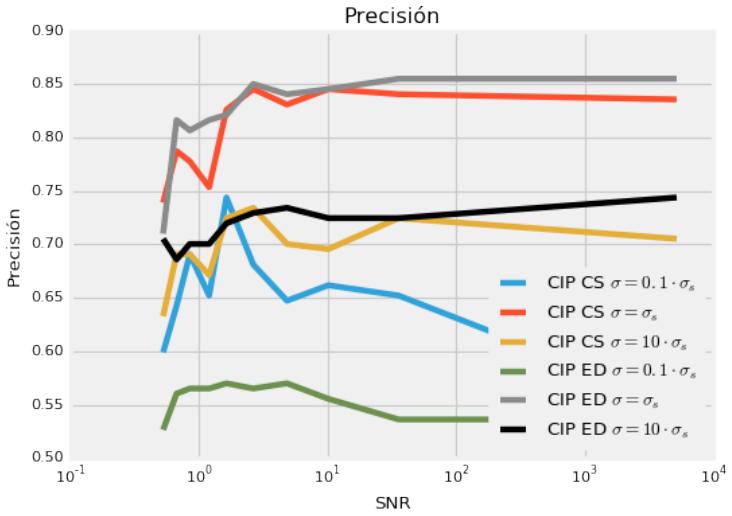


Figura 4.30: Gráficos que compara la precisión de los Ensambles CIPL bajo ruido en la entrada de los datos y cuando se varia el tamaño del kernel. El ruido es aditivo Gaussiano $\mathcal{N}(\mu = 0, \sigma^2_{ruido})$. El SNR es calculado a partir del ruido y los datos de entrada del modelo: SNR $= \sigma^2_{\text{entrada}}/\sigma^2_{\text{ruido}}$. En el gráfico σ_s corresponde al tamaño del kernel calculado con Silverman.

del kernel con respecto a Silverman permite que la estimación de la Información Mutua sea menos sensible al ruido en los datos.

Capítulo 5

Conclusiones

En este trabajo de investigación no solo se revisó el Potencial de Información Cruzado (CIP) sino también otras medidas propuestas por la Teoría de la Información para el Aprendizaje (ITL). En un principio se revisaron métodos como la Correntropía 2.6.2, el Potencial de Información (IP) 2.6.3 y MEE 2.6.5, todas herramientas del Aprendizaje con Teoría de la Información (ITL). Al final del proceso de exploración de alternativas se optó por el Potencial de Información Cruzado (CIP) 2.6.4 dada la capacidad de manejar más de 2 fuentes de información, lo que lo hizo ideal para el caso de los Ensambles.

Después del proceso de implementación de CIPL se realizaron algunas pruebas para determinar su comportamiento logrando identificar 2 mejoras importantes, la primera de ellas es la inclusión de una etapa de pre-entrenamiento para ayudar a la convergencia del método. El problema de optimizar una función de costo basada en la Información Mutua para el aprendizaje supervisado, es que existen 2 condiciones finales en la optimización: un aumento o disminución del error de predicción, ambas condiciones aumentan la Información Mutua y para métodos como el de gradiente descendente 2.3.1 no existe forma de discriminar entre ambos. Para ayudar a la convergencia correcta se utiliza MSE en las primeras épocas de entrenamiento para propiciar que la optimización se genere en la dirección correcta.

La inclusión de Annealing 3.2.1 para ir modificando el tamaño del kernel durante el entrenamiento, mejoró los resultados especialmente en problemas de clasificación. Se observó que CIPL en el proceso de optimización podía quedar "estancado" en un estado en que la variación del gradiente era muy pequeña de forma que no se modificaban significativamente los parámetros del Ensamble, al incluir Annealing se favorece la exploración durante el proceso de optimización debido al cambio de sensibilidad sobre los datos que se obtiene con la reducción del tamaño del kernel.

La hipótesis de trabajo propuesta es que el uso de CIP podría mejorar en primer lugar la diversidad de los modelos de un Ensamble y al mismo tiempo el desempeño de éste. Al revisar los resultados es posible decir que esta hipótesis no se logró comprobar. Los problemas de clasificación aunque mostraron resultados comparables a NCL no son mejores que éste. En cuanto a CIPL en regresión se pudo observar problemas de sesgos que impiden mejorar los resultados, se llega a la conclusión que primero se debe compensar el sesgo en cada modelo antes de combinarlos.

En clasificación se pudo ver que el uso de la sinergia como la redundancia, manejadas a partir de los parámetros β y λ respectivamente, ayudan a manejar la diversidad en el ensamble. Es importante destacar que durante las pruebas se utilizó un rango limitado de variación de ambos parámetros ($[-1, 1]$), pero eso fue establecido de forma arbitraría, aunque sería recomendable utilizar algún tipo de normalización para formalizar este rango.

Otra hipótesis de trabajo era la de que CIPL es robusto a ruido (Gaussiano), esto no se pudo comprobar de forma concluyente, los ensambles generados por CIPL son tan robustos como los otros modelos utilizados en este trabajo. Por otro lado, si se pudo comprobar que el tamaño del kernel si tiene un efecto sobre esta capacidad pero que al mismo tiempo puede generar modelos de menor desempeño, por tanto, depende de cada problema el uso o no del tamaño del kernel.

Resumiendo, CIPL en problemas de clasificación se puede convertir en una herramienta interesante para continuar su investigación, considerando sus propiedades respecto a la información mutua y el manejo de la diversidad que no depende de la salida final del ensamble, sino de la relación entre las salidas de los modelos. Por otro lado, CIPL se puede presentar de 2 formas, dependiendo de la divergencia utilizada para la estimación de CIP: Cauchy-Schwarz (CS) y Euclidiana (ED), la primera obtiene mejores resultados en problemas de clasificación y la segunda en problemas de regresión. CIPL CS es más sensible al ruido en los datos en comparación con CIPL ED. Respecto a la diversidad, con CIPL CS se puede manejar la diversidad a través de la variación de los parámetros asociados a la sinergia y la redundancia. Por otro lado, en CIPL ED no se observa una relación clara entre diversidad y los términos de sinergia y redundancia.

Finalmente, se puede decir que el valor de este trabajo es el haber podido realizar y evaluar un entrenamiento de Ensamble de redes neuronales donde la función de costo contiene información no solo de los modelos individuales, sino también de las interacciones de los demás modelos. Además, queda disponible en GitHub para futuros trabajos la librería DeepEnsemble, desarrollada durante este trabajo.

5.1. Comparación entre NCL y el método CIPL

Al revisar los resultados que comparan NCL con CIPL es posible decir que el primero tiene un mejor desempeño en problemas de regresión, incluso mejora los resultados obtenidos con un Ensamble entrenado solo con MSE. CIPL por otro lado, tiene los peores resultados. Una respuesta a este hecho es que NCL intenta minimizar directamente el error de predicción, mientras que CIPL la reduce de forma indirecta a través de la maximización de la información mutua entre la salida deseada y la del Ensamble. Además CIPL contienen sesgos diferentes en cada sub-modelo que perjudican su resultado sin no son corregidos antes de ser combinados.

CIPL en problemas de clasificación muestra un mejor desempeño, aunque no muy grande, respecto a NCL, en este caso esto se puede atribuir a que la estrategia de maximizar la información mutua funciona mejor cuando los datos de salida son de tipo discreto, además el uso de la información mutua implica que CIPL no solo usa información del error de predicción sino también información estadística de los mismos. A pesar de lo anterior, CIPL presenta

problemas al momento de trabajar con más de 2 clases a diferencia de NCL.

En cuanto a la diversidad generada por estos 2 métodos es posible decir que ambos logran su cometido. Sin embargo, con CIPL se puede tener un control mayor dado que la diversidad la puede manejar durante el entrenamiento y con la información de todos los modelos y no solo entre los modelos individuales y la salida del ensamble como en NCL. Por otro lado, en NCL el término de la ambigüedad que utiliza depende del error de predicción solamente, mientras que en CIPL depende de la distribución estadística de las salidas individuales de los modelos, por tanto, tiene mayor información.

5.2. Trabajos Futuros

Uno de los primeros trabajos propuestos es revisar la implementación del Potencial de Información para revisar la forma de mejorar su desempeño en problemas de clasificación en casos de más de 2 clases. Además, es importante ver como optimizar la librería DeepEnsemble de forma de poder reducir el tiempo de compilación y ejecución. Un añadido interesante respecto a la libraría sería implementar la opción para aprendizaje no supervisado, actualmente solo puede entrenar modelos basados en redes neuronales para el paradigma supervisado.

En cuanto al método CIPL propiamente tal, es posible mejorar el proceso de optimización dado que el método de gradiente descendente 2.3.1 al maximizar CIP no discrimina entre la maximización de la Información Mutua en dirección de reducir el error de predicción o aumentarlo, por tal motivo se puede diseñar un método de optimización como PSO (Optimización por enjambre de partículas) que tenga en cuenta la dirección que mejora el desempeño, esto mejoraría el proceso de convergencia al óptimo que se quiere y además no se necesitaría un proceso de pre-entrenamiento con MSE dado que CIPL no requeriría estar cerca del óptimo.

Bibliografía

[1] Juan José Montaño. *Redes Neuronales Artificiales aplicadas al Análisis de Datos*. PhD thesis, Universitat De Les Illes Balears, Facultad de Psicología, 2002.

[2] Z.H. Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Serie. Taylor & Francis, 2012.

[3] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[4] Stevan Harnad. The annotation game: On turing (1950) on computing, machinery, and intelligence (published version bowdlerized). 2008.

[5] C. Sammut and G.I. Webb. *Encyclopedia of Machine Learning*. Encyclopedia of Machine Learning. Springer US, 2011.

[6] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[7] J.C. Principe. *Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives*. Information Science and Statistics. Springer New York, 2010.

[8] 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, 2003.

[9] Gavin Brown. *Diversity in neural network ensembles*. PhD thesis, University of Birmingham, 2004.

[10] Yong Liu and Xin Yao. Ensemble learning via negative correlation. *Neural Networks*, 12(10):1399–1404, 1999.

[11] Anders Krogh, Jesper Vedelsby, et al. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, 7:231–238, 1995.

[12] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[13] Gavin Brown and Ludmila I Kuncheva. "good" and "bad" diversity in majority vote ensembles. In *International Workshop on Multiple Classifier Systems*, pages 124–133. Springer, 2010.

[14] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.

[15] Kagan Tumer and Joydeep Ghosh. Robust order statistics based ensembles for distributed data mining. Technical report, DTIC Document, 2001.

[16] Gavin Brown. An information theoretic perspective on multiple classifier systems. In *International Workshop on Multiple Classifier Systems*, pages 344–353. Springer, 2009.

[17] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

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

[19] Prem Melville and Raymond J Mooney. Creating diversity in ensembles using artificial data. *Information Fusion*, 6(1):99–111, 2005.

[20] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[21] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286, 1995.

[22] Eun Bae Kong and Thomas G Dietterich. Error-correcting output coding corrects bias and variance. In *ICML*, pages 313–321, 1995.

[23] Huanhuan Chen and Xin Yao. Regularized negative correlation learning for neural network ensembles. *IEEE Transactions on Neural Networks*, 20(12):1962–1979, 2009.

[24] Frederico A.C. Azevedo, Ludmila R.B. Carvalho, Lea T. Grinberg, José Marcelo Farfel, Renata E.L. Ferretti, Renata E.P. Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of Comparative Neurology*, 513(5):532–541, 2009.

[25] R.F. López and J.M.F. Fernández. *Las Redes Neuronales Artificiales*. Metodología y Análisis de Datos en Ciencias Sociales. Netbiblo, 2008.

[26] S. Lingireddy and G.M. Brion. *Artificial Neural Networks in Water Supply Engineering*. American Society of Civil Engineers, 2005.

[27] S. Kumar. *Neural Networks: A Classroom Approach*. Computer engineering series. McGraw-Hill, 2004.

[28] Parallel Distributed Processing, David E Rumelhart, et al. Vol. 1. *Chapter*, 8:319–362, 1986.

[29] J.A. Anderson. *An Introduction to Neural Networks*. A Bradford book. MIT Press, 1995.

[30] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[31] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[32] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[33] A.N. Tikhonov and V.I.A. Arsenin. Solutions of ill-posed problems. 1977.

[34] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, jan 1999.

[35] Y Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). *Doklady ANSSSR*, 269(1):543–547, 1983.

[36] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

[37] Martin Riedmiller and Heinrich Braun. A fast adaptive learning algorithm. *Proceedings of the International Symposium on Computer and Information Science VII*, 1992.

[38] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[39] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[40] Bradley Efron and Robert J Tibshirani. An introduction to the bootstrap chapman & hall. *New York*, 436, 1993.

[41] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[42] Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. Feature-weighted linear stacking. *arXiv preprint arXiv:0911.0460*, 2009.

[43] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[44] Ludmila I Kuncheva. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.

[45] MP Perrone, LN Cooper, and RJ Mammone. Neural networks for speech and image processing. *When Networks Disagree: Ensemble Methods for Hybrid Neural Networks, Chapman Hall*, 1993.

[46] Amanda JC Sharkey and Noel E Sharkey. Combining diverse neural nets. *Knowledge Engineering Review*, 12(3):231–247, 1997.

[47] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[48] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[49] Xiao-Tong Yuan and Bao-Gang Hu. Robust feature extraction via information theoretic learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1193–1200. ACM, 2009.

[50] Deniz Erdogmus and Jose C Principe. Comparison of entropy and mean square error criteria in adaptive system training using higher order statistics. In *Proc. ICA*, pages 75–80, 2000.

[51] Deniz Erdogmus and Jose C Principe. An error-entropy minimization algorithm for supervised training of nonlinear adaptive systems. *IEEE Transactions on Signal Processing*, 50(7):1780–1786, 2002.

[52] Leslie Greengard and John Strain. The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

[53] Ludmila I Kuncheva. A theoretical study on six classifier fusion strategies. *IEEE Transactions on pattern analysis and machine intelligence*, 24(2):281–286, 2002.

[54] Ludmila I Kuncheva. Switching between selection and fusion in combining classifiers: an experiment. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 32(2):146–156, 2002.

[55] Giorgio Fumera and Fabio Roli. Performance analysis and comparison of linear combiners for classifier fusion. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 424–432. Springer, 2002.

[56] Kagan Tumer and Joydeep Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection science*, 8(3-4):385–404, 1996.

[57] Gavin Brown and Jeremy Wyatt. Negative correlation learning and the ambiguity family of ensemble methods. In *International Workshop on Multiple Classifier Systems*, pages 266–275. Springer, 2003.

[58] Robert M Fano and David Hawkins. Transmission of information: A statistical theory of communications. *American Journal of Physics*, 29(11):793–794, 1961.

[59] Chernoff Bound. Probability of error, equivocation, and the. *IEEE Transactions on Information Theory*, 16(4), 1970.

[60] Gavin Brown. A new perspective for information theoretic feature selection. In *AISTATS*, pages 49–56, 2009.

[61] Ivan Kojadinovic. Relevance measures for subset variable selection in regression problems based on k-additive mutual information. *Computational Statistics & Data Analysis*, 49(4):1205–1227, 2005.

[62] David A Bell and Hui Wang. A formalism for relevance and its application in feature subset selection. *Machine learning*, 41(2):175–195, 2000.

[63] Lei Yu and Huan Liu. Efficient feature selection via analysis of relevance and redundancy. *Journal of machine learning research*, 5(Oct):1205–1224, 2004.

[64] Willfried Wienholt and Bernhard Sendhoff. How to determine the redundancy of noisy chaotic time series. *International Journal of Bifurcation and Chaos*, 6(01):101–117, 1996.

[65] Aleks Jakulin and Ivan Bratko. Quantifying and visualizing attribute interactions. *arXiv preprint cs/0308002*, 2003.

[66] Zhi-Hua Zhou and Nan Li. Multi-information ensemble diversity. In *International Workshop on Multiple Classifier Systems*, pages 134–144. Springer, 2010.

[67] PA Bromiley, NA Thacker, and E Bouhova-Thacker. Shannon entropy, renyi entropy, and information. *Statistics and Inf. Series (2004-004)*, 2004.

[68] Rupert L Frank and Elliott H Lieb. Monotonicity of a relative rényi entropy. *Journal of Mathematical Physics*, 54(12):122201, 2013.

[69] Pablo Vera. *Proyección de Datos Multidimensionales utilizando Teoría de la Información*. PhD thesis, PhD thesis, University of Chile, Santiago, Chile, 2010.

[70] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[71] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[72] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[73] François Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[74] Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, diogo149, Brian McFee, Hendrik Weideman, takacsg84, peterderivaz, Jon, instagibbs, Dr. Kashif Rasul, CongLiu, Britefury, and Jonas Degrave. Lasagne: First release., August 2015.

[75] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[76] Yann LeCun, Corinna Cortes, and Christopher JC Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.

[77] Robert A Jacobs. Bias/variance analyses of mixtures-of-experts architectures. *Neural computation*, 9(2):369–383, 1997.

[78] Yong Liu. *Negative correlation learning and evolutionary neural network ensembles*. PhD thesis, PhD thesis, University College, The University of New South Wales, Australian Defence Force Academy, Canberra, Australia, 1998.

[79] David J Newman, Seth Hettich, Cason L Blake, and Christopher J Merz. {UCI} repository of machine learning databases. 1998.

[80] J.N. Kapur. *Measures of Information and Their Applications*. Wiley, 1994.

[81] Derek Partridge and Wojtek Krzanowski. Distinct failure diversity in multiversion software. *Res. Rep*, 348:24, 1997.