UNIVERSIDAD PRIVADA ANTENOR ORREGO

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE COMPUTACIÓN Y SISTEMAS

"PRINCIPAL COMPONENT ANALYSIS (PCA) PARA MEJORAR LA PERFORMANCE DE APRENDIZAJE DE LOS ALGORITMOS SUPPORT VECTOR MACHINE (SVM) Y RED NEURONAL MULTICAPA (MLNN)"

TESIS PARA OBTENER EL TÍTULO PROFESIONAL DE INGENIERO DE COMPUTACION Y SISTEMAS

LINEA DE INVESTIGACIÓN: *Machine Learning.*

AUTORES :

Br. AGUILAR GUTIÉRREZ, LUIS ANTONIO.

Br. VÁSQUEZ VALDIVIA, YNDRA OLENCA.

ASESOR :

Dr. LUIS VLADIMIR URRELO HUIMAN.

CO-ASESOR :

Ms. TEOBALDO HERNAN SAGASTEGUI CHIGNE.

Trujillo – Perú

2016

“PRINCIPAL COMPONENT ANALYSIS (PCA) PARA MEJORAR LA PERFORMANCE DE APRENDIZAJE DE LOS ALGORITMOS SUPPORT VECTOR MACHINE (SVM) Y RED NEURONAL MULTICAPA (MLNN)”

Presentado por **:**

Br. AGUILAR GUTIÉRREZ, LUIS ANTONIO.

Br. VÁSQUEZ VALDIVIA, YNDRA OLENCA.

Aprobado por **:**

Ing. Armando Javier Caballero Alvarado
CIP: 149181
Presidente

Ing. Wilder Adán Namay Zevallos
CIP: 130945
Secretario

Ing. Carlos Alberto Gaytan Toledo
CIP: 84519
Vocal

Ing. Luis Vladimir Urrelo Huiman
CIP: 88212
Asesor

DEDICATORIA

Dedicamos este trabajo a las generaciones futuras; esperando que les sirva de aliciente para seguir investigando y mejorando cada día más y más; porque creemos y, es esa nuestra firme convicción de que, debemos usar la ciencia y/o tecnología para poder crear cada día un mundo más digno e igual para todos; tal vez no logramos ver ese mundo en nuestro tiempo; pero quiero que sepan que dimos lo mejor de nosotros para verlo hecho realidad; finalmente encomendamos a todos y cada uno de ustedes esa labor, ese sueño de ver un mundo mejor para todos.

Los Autores.

AGRADECIMIENTO

Agradecemos a Dios, sin el cual nada de esto hubiera sido posible; a toda nuestra familia por brindarnos su apoyo y consejo; a todos nuestros profesores por ayudarnos tanto en nuestra formación académica como personal; especialmente un agradecimiento a la distancia al Profesor Andrew Ng; quien fue el que despertó nuestra curiosidad por el área de Machine Learning

Los Autores.

RESUMEN

“PRINCIPAL COMPONENT ANALYSIS (PCA) PARA MEJORAR LA PERFORMANCE DE APRENDIZAJE DE LOS ALGORITMOS SUPPORT VECTOR MACHINE (SVM) Y RED NEURONAL MULTICAPA (MLNN)”

Desarrollado por :

Br. Aguilar Gutiérrez, Luis Antonio.

Br. Vásquez Valdivia, Yndra Olenca.

Esta tesis explora el problema de data sets con un número alto de atributos; y el impacto que generan en la performance de aprendizaje de los algoritmos Support Vector Machine (SVM) y Redes Neuronales Multicapa (MLNN).

Para poder resolver este problema, proponemos la siguiente hipótesis: "La aplicación de Principal Component Analysis (PCA) sobre el data set; mejorará la performance de aprendizaje de los algoritmos Support Vector Machine (SVM) y Redes neuronales Multicapa (MLNN).

De acuerdo con nuestra hipótesis; tenemos el siguiente objetivo general: "Mejorar la performance de aprendizaje de los algoritmos Support Vector Machine (SVM) y Redes Neuronales Multicapa (MLNN) a través de la aplicación de Principal Component Analysis (PCA) sobre el data set".

Para poder implementar los algoritmos (SVM, MLNN y PCA); usamos el data set QSAR biodegradation, de obtenido del repositorio gratuito Machine Learning (UCI), asimismo, todo la implementación de los algoritmos fue realizada usando Matlab 2014a. Una vez que los algoritmos fueron implementados, empezamos la prueba de la hipótesis; para ello creamos dos dataset, uno aplicando PCA y el otro sin aplicarle PCA; luego medimos la performance de aprendizaje de los algoritmos SVM y MLNN contra sus contrapartes sin PCA; al final, los resultados mostraron que ambos algoritmos SVM y MLNN ganaron una mejora significativa en sus performances de aprendizaje en contraste con simplemente entrenar los algoritmos sin aplicar PCA al data set.

ABSTRACT

"PRINCIPAL COMPONENT ANALYSIS (PCA) TO IMPROVE THE LEARNING PERFORMANCE OF ALGORITHMS SUPPORT VECTOR MACHINE (SVM) AND MULTILAYER NEURAL NETWORK (MLNN)"

Developed by :

Br. Aguilar Gutiérrez, Luis Antonio.
Br. Vásquez Valdivia, Yndra Olenca.

This thesis explores the problem of data sets with a high number of attributes, and its impact on the learning performance of the algorithms Support Vector Machine (SVM) and Multilayer Neural Network (MLNN).

In order to solve this problem we propose the following hypothesis: "The application of Principal Component Analysis (PCA) over the data set; will improve the learning performance of the algorithms Support Vector Machine (SVM) and Multilayer Neural Network (MLNN)"

According with our hypothesis; we have the following general objective: "Improve the learning performance of the algorithms Support Vector Machine (SVM) and Multilayer Neural Network through the application of Principal Component Analysis (PCA) over the data set". In order to implement the algorithms (SVM, MLNN and PCA), we used the QSAR biodegradation dataset, obtained from the Free Machine Learning Repository (UCI), also all the development of the algorithms was done using Matlab 2014a. Once the algorithms were developed, we begin with the test of our hypothesis, to do so, we create two sets, one applying PCA to the dataset, and the other without applying it, then we measure the learning performance of the algorithms SVM and MLNN against themselves on both datasets (one applying PCA and the other not), at the end, the results show us that both algorithms SVM and MLNN gain a major improvement in their learning performance compared to simple train the algorithms without applying PCA to the dataset.

CONTENIDO

Índice de Figuras

Índice de Tablas

Índice de Ecuaciones

Capítulo I: Introducción

Este capítulo define aspectos básicos sobre los cuales se cimienta la investigación de la Tesis; los que se centran en analizar y/o definir la problemática, así como definir los objetivos de investigación, los cuales nos sirven como guía durante todo el proceso de desarrollo de la Tesis; finalmente este capítulo presenta un apartado donde se expone brevemente el contenido de los demás capítulos.

1.1. Planteamiento del Problema:

Dentro del área de Machine Learning; la performance de aprendizaje de un algoritmo es medida en base a varios indicadores (recall, precision, accuracy, f – score, entre otros); asimismo la performance se encuentra ligada a otros factores como: la selección del algoritmo, selección de parámetros, validaciones, entre otros; los cuales se ven afectados por las características que presente el data set con el que el algoritmo va a ser entrenado. (Awad & Khanna, 2015)

Otro factor de complejidad con respecto al data set es el número de las dimensiones del mismo; en aplicaciones reales; los data sets suelen contar con dimensiones grandes rodeando incluso los 100000 a más (aplicaciones industriales, bioinformática, etc.); esto supone un gran problema; ya que no todas las dimensiones aportan la misma cantidad de información para el aprendizaje del algoritmo; lo cual se traduce en una performance de aprendizaje baja; incluso después de haber realizado el pre – procesamiento con las técnicas antes mencionadas (Chellappa & Theodoridis, 2014).

Un ejemplo concreto donde se puede observar estos problemas, incluye a los algoritmos Support Vector Machine (SVM) y Multilayer Neural Network (MLNN); ya que debido al número elevado de dimensiones, ambos algoritmos presentan problemas de aprendizaje y/o performance; por un lado SVM presenta problemas con el tiempo de entrenamiento; ya que, al haber data sets grandes; el tiempo de entrenamiento crece exponencialmente $O(N^2)$ con respecto al data set (M Awad & Khan, 2004); mientras que MLNN experimenta una serie de comportamientos inciertos con respecto a sus

funciones locales (como kernles gaussianos) al lidiar con dimensiones densas (Verleysen, François, Simon, & Wertz, 2003)

Finalmente existen varios métodos por los cuales se puede reducir la complejidad de las dimensiones; ayudando de esta manera a que los algoritmos puedan mejorar sus tiempos de convergencia; uno de estos métodos es llamado Principal Component Analysis (PCA) (Mariette Awad & Khanna, 2015)

1.2. Delimitación de la investigación:

Este estudio está enfocado en el uso de ***Principal Componente ANALYSIS*** (PCA) como herramienta después del pre–procesamiento de los datos para el mejoramiento de la performance de aprendizaje de los algoritmos ***Support Vector Machine*** (SVM) y ***Redes Neuronales Multicapa*** (MLNN), concretamente, se usará una red multicapa; la elección de los algoritmos se debe que éstos dos algoritmos poseen una alta performance, y son usados frecuentemente en diversas aplicaciones; con respecto a la data; se hace uso de un data set gratuito sobre *Modelos para el estudio de las relaciones entre la estructura química y la biodegradación de moléculas* (Mansouri, Ringsted, Ballabio, Todeschini, & Consonni, 2013) obtenido de *UCI MACHINE LEARNING REPOSITORY*, el cual cuenta con 1055 muestras (después del pre–procesamiento) y 41 atributos o características, incluyendo por supuesto, el vector predictor $\vec{y}$**;** la elección del data set se hizo en base al número de dimensiones y observaciones del mismo; ya que reflejan el problema dimensional que se quiere resolver.

1.3. Características del Problema:

- Numero de dimensiones elevadas, relaciones internas entre atributos y varianza en pruebas comunes de aplicación de los algoritmos SVM y MLNN.
- Performance de aprendizaje baja de los algoritmos SVM y MLNN.
- Tiempo de procesamiento largo.

1.4. Análisis de características problemáticas:

- *Numero de dimensiones elevadas, relaciones internas entre atributos y/o varianza:* Las dimensiones elevadas de los data set son muy comunes en muchas aplicaciones; y constituyen a su vez un elemento fundamental cuando se requiere capturar valores de sistemas continuos; tales como en fábricas, en control de calidad, y también en el área de bioinformática; ya que un data set que almacena las características de un compuesto o molécula, tiende a exhibir un número elevado de dimensiones. Por otro lado también se encuentran marcadas las relaciones internas que existen entre los diversos atributos del data set, las cuales junto con la varianza, definen los aspectos de la propia información del data set.

- *Performance de aprendizaje baja de los algoritmos:* Este es un problema que toma mucha importancia; ya que una mala performance de aprendizaje significa que el algoritmo no ha sido capaz de aprender los patrones de los datos; con lo cual no puede ser puesto en producción; ya que una performance de aprendizaje baja implica que el algoritmo es incapaz de realizar una correcta generalización con datos futuros.

- *Tiempo de procesamiento largo:* Debido a presentar dimensiones y/o observaciones de magnitudes grandes; se requiere más tiempo para aprender las diversas hipótesis de aprendizaje; asimismo los tiempos empleados para la validación y el test se incrementan notablemente; ya que es computacionalmente más flexible trabajar con un test de validación de 350 observaciones y 20 características que con uno de 5000 observaciones y 45000 características.

1.5. Definición del problema:

El número de dimensiones de un data set implica trabajo para los algoritmos de aprendizaje que se empleen; a su vez, dado que no todas las dimensiones aportan el mismo nivel de información para el entrenamiento; éstas causan una degradación de la performance de los algoritmos SVM y MLNN.

1.6. Enunciado del Problema:

¿Cómo mejorar la performance de aprendizaje de los Algoritmos Support Vector Machine (SVM) y Red Neuronal Multicapa (MLNN)?

1.7. Hipótesis:

"La aplicación de Principal Component ANALYSIS (PCA) mejorará la Performance de aprendizaje de los Algoritmos Support Vector Machine (SVM) y Red Neuronal Multicapa (MLNN)"

1.8. Antecedentes del problema:

Principle components analysis and Support Vector Machine based Intrusion Detection System (Heba, Darwish, Hassanien, & Abraham, 2010)

En este estudio, se hace uso de SVM y PCA en el área de Sistema de detección de intrusiones (IDS); el cual es un tema fundamental en para asegurar la protección, recursos e infraestructura de una red. Durante el estudio se usa el data set NSL-KDD, sobre el cual se diseña el sistema; el cual consiste de 3 etapas: (1) pre – procesamiento, donde se realiza el mapeo y escalamiento de atributos del data set; para luego pasar a la (2) selección de características, en la cual se aplica PCA sobre el data set para seleccionar aquellos atributos que posean una mayor relevancia; es decir contengan más varianza; para finalmente pasar a hacer uso de la (3) detección de intrusiones; en donde finalmente se aplica SVM para poder clasificar las intrusiones a la red. Como se puede observar; en el paso (2), se aplica PCA al data set, con lo cual se logra disminuir los atributos del data set; mejorando de esta manera el tiempo de aprendizaje del algoritmo SVM, así como también su performance de aprendizaje se reduce.

Principal Component Analysis and Neural Networks for Predicting the Pile Capacity Using SPT (Benali, 2013)

En este estudio, se plantea el uso de PCA y Redes Neuronales para poder predecir la capacidad de carga de un pilote; para lo cual se elabora un cuadro estadístico mostrando las variables que influyen a la capacidad de carga del pilote (data set);en este enfoque primero se aplica PCA al data set antes del proceso de aprendizaje de la

red Neuronal; siendo observado que; a través del uso de PCA (Principal component Analysis) sobre las variables del data Set; se logran extraer los ratios de aquellas variables que más influencia tienen sobre la solución del problema; ya que todas ellas (variables) están correlacionadas entre sí; con lo cual se logra disminuir la complejidad del problema al crear un input (espacio vectorial de características) más conciso para la Red Neuronal; ya que ingresan como parámetros sólo en aquellas variables con alto impacto; de este modo se pudo estimar al final los parámetros óptimos para la predicción.

Integration of FCM, PCA and Neural Networks for classification of ECG Arrhythmias (Patra, Das, & Pradhan, 2009)

En este estudio se realiza una integración de varios modelos; con el objetivo de poder realizar una mejor clasificación de las arritmias ECG; para lo cual se usa un método basado en la integración de varios algoritmos; los cuales implican fuzzy c–means (FCM), principal component ANALYSIS (PCA) y redes neuronales (NN); el método propuesto consta de 4 fases; las cuales son: sampleo y pre–procesamiento de los datos de ECG (1), reducción de datos del data set (2), cálculo del vector de características (3); y finalmente clasificación de los datos usando la red neuronal (4); se puede apreciar como a través del paso (2) y (3) de la metodología se hace uso de PCA para poder extraer el vector de características; el cual reúne a los datos más representativos; son estos datos los que entran como entrada para la red neuronal; con lo cual se reduce el tiempo de procesamiento de la misma; a la vez que se aprovecha mejor su complejidad sin el consumo excesivo de recursos.

Comparison of Support Vector Machine and Back Propagation Neural Network in Evaluating the Enterprise Financial Distress (Lee & To, 2010)

Este estudio realiza una comparación entre lo performance de aprendizaje de los modelos SVM y Redes neuronales; para lo cual ambos modelos son aplicados al problema de las evaluaciones financieras; el cual genera pérdidas a las instituciones que ofrecen el préstamo cuando su cliente u organización no disponen de la capacidad de pagar después de haber realizado el préstamo. Durante la comparación de modelos

se puede apreciar cómo se realizan tareas de pre – procesamiento sobre el data set; más específicamente, escalamiento de atributos; para después pasar al entrenamiento de ambos algoritmos obteniendo SVM una performance más alta que las redes neuronales por un 4%.

1.9. Justificación del proyecto:

Support Vector Machine (SVM), Redes Neuronales Multicapa (NN) y Principal Component Analysis (PCA) son 3 algoritmos ampliamente usados para diversas aplicaciones; tanto en el ámbito académico como aplicativo (industrial, bioinformática, marketing, etc.). Al mejorar la performance de aprendizaje; se logra aumentar la capacidad de generalización de los modelos (SVM y MLNN); esto se traduce en la mejora de la performance (disminución de los errores al realizar las tareas de aprendizaje) al trabajar con nuevas muestras de datos; obteniendo resultados más fiables dentro de la tarea de aprendizaje que se esté llevando a cabo (predicción, clasificación, regresión, entre otros) lo cual genera que la eficiencia de los sistemas (a los cuales estén embebidos SVM y MLNN) mejore en términos de: capacidad para lograr reconocer de forma más fiable los patrones, tiempo de respuesta más rápido y precisión aumenten considerablemente; asimismo dichas aplicaciones tienen un gran uso dentro de la industria (control de sistemas de producción, monitoreo de aplicaciones de pozos petroleros, sistemas de reconocimiento de peatones, bioinformática, marketing, detección de fraudes bancarios, entre otros); como también en aplicaciones médicas (predicción de enfermedades, monitoreo de pacientes, clasificación de compuestos, entre otros).

1.10. Objetivos:

1.10.1. General:

Incrementar la performance de aprendizaje de los algoritmos Support Vector Machine (SVM) y Redes Neuronales Multicapa (MLNN) aplicando Principal Component Analysis (PCA).

1.10.2. Específicos:

1.10.2.1. Obtener y depurar el data set de QSAR biodegradation del repositorio de datos de Machine Learning (UC Irvine Machine Learning Repository) para su posterior utilización.

1.10.2.2. Elaborar la descripción general, resumen de características y el diccionario de datos sobre el dataset QSAR biodegradation con el objetivo de tener un mejor entendimiento de los atributos y/o características del mismo.

1.10.2.3. Implementar Principal Component Analysis (PCA) para pre – procesar el data set QSAR biodegradation usando MATLAB versión 2014a.

1.10.2.4. Desarrollar, entrenar, y/o implementar una red neuronal multicapa (MLNN) usando la data generada por PCA; usando MATLAB versión 2014a.

1.10.2.5. Entrenar, y/o implementar el algoritmo Support Vector Machine (SVM) en MATLAB versión 2014a, usando la librería SVMLIB (Chang & Lin, 2011) usando la data generada por PCA; usando MATLAB versión 2014a.

1.10.2.6. Desarrollar, entrenar, y/o implementar una red neuronal multicapa (MLNN) usando la data original sin aplicar PCA; usando MATLAB versión 2014a.

1.10.2.7. Entrenar, y/o implementar el algoritmo Support Vector Machine (SVM) en MATLAB versión 2014a, usando la librería SVMLIB (Chang & Lin, 2011) usando la data original sin aplicar PCA; usando MATLAB versión 2014a.

1.10.2.8. Realizar las comparaciones de performance de aprendizaje usando las métricas de Machine Learning (matriz de confusión, presicion, recall, F – Micro averaged Score y curvas ROC) en los algoritmos SVM y NN con y sin PCA.

A continuación se muestra la distribución general de la tesis; la cual se estructura por los siguientes capítulos:

Capítulo I: Introducción: Muestra la definición del problema de investigación, delimitación, los objetivos y finalmente un resumen del contenido de todos los capítulos de la tesis.

Capítulo II: Fundamento Teórico: El capítulo contempla toda la teoría usada como base; la cual abarca desde las definiciones puntuales de Machine Learning (ML); hasta la descripción, funcionamiento y/o implementación de los algoritmos: Principal Component Analysis (PCA), Support Vectonar Machine (SVM) y Multilayer Neural Network (MLNN); los cuales son aplicados durante el desarrollo de la tesis.

Capítulo III: Metodología: El capítulo muestra la estructura de la cual está compuesta la Metodología usada durante el desarrollo y/o implementación de los algoritmos Principal Component Analysis (PCA), Support Vector Machine (SVM) y Multilayer Neural Network (MLNN); así como la estructura de los pasos a seguir para realizar las comparaciones de performance de aprendizaje entre los algoritmos; lo cual sirve para realizar la contrastación final de la hipótesis.

Capítulo IV: Desarrollo: El capítulo muestra la implementación de los diversos algoritmos a través de la metodología definida en el *Capítulo III*.

Capítulo V: Discusión: En este capítulo se muestra formalmente la contrastación de la hipótesis; para lo cual se usa la metodología descrita en el *Capítulo III*.

Conclusiones: En este capítulo se muestran las conclusiones más relevantes de la investigación llevada a cabo; producto del análisis de los Capítulos *IV* y *V*.

Recomendaciones: Finalmente se realizan las recomendaciones de acuerdo todo el desarrollo y/o conclusiones de la tesis.

Capítulo II: Fundamento Teórico

En este capítulo se aborda el fundamento teórico necesario para poder llevar a cabo tanto el desarrollo (implementación) que se muestra en el *Capítulo IV*; así como también la contrastación de la hipótesis mostrada en el *Capítulo V*. Para lo cual se estructura el fundamento teórico partiendo desde las definiciones formales de Machine Learning (ML); para luego pasar a discutir formalmente los algoritmos que serán usados en el desarrollo del *Capítulo IV*; también se mencionan las diversas métricas que serán usadas en la contrastación de la hipótesis en el *Capítulo V*; finalmente se muestra el framework de trabajo de los sistemas de Machine Learning; el cual es usado para definir la metodología de trabajo mostrada en el *Capítulo III*.

2.1. Data Set:

Se denomina data set al conjunto de datos (por lo general en un formato estándar de filas y columnas); el cual es usado por un algoritmo para poder aprender patrones de dichos datos.

Para representar el data set, construiremos la matriz de diseño; para lo cual tomaremos el vector $\vec{x}$; donde agruparemos todas las características (vector de características); dando como resultado (Flach, 2012):

$$\vec{x} = \begin{bmatrix} x^{(0)} \\ x^{(1)} \\ \vdots \\ x^{(m)} \end{bmatrix}$$

Donde m representa el número total de muestras; una vez obtenido este vector construiremos la matriz X que agrupará a las características; para lo cual usaremos la transpuesta de cada vector $\left(x^{(i)}\right)^T$; donde i representa un índice; a su vez también construimos el vector *m-dimensional* $\vec{y}$; el cual agrupa los targets o labels[1], obteniendo la siguiente representación (Shalev-shwartz & Ben-david, 2014):

$$X = \begin{bmatrix} - \left(x^{(1)}\right)^T - \\ - \left(x^{(2)}\right)^T - \\ \vdots \\ - \left(x^{(m)}\right)^T - \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

[1] Para el caso de aprendizaje supervisado; ya que para el no-supervisado los targets no son dados de forma explícita

Donde:

$X =$ Data set.

$y =$ Target u outputs.

$\left[x^{(i)}, y^{(i)}\right] =$ Representa una muestra del data set.

$m =$ Representa el número de muestras del data set.

2.2. Machine Learning (ML):

Es una rama de la inteligencia artificial que sistemáticamente aplica algoritmos para sintetizar las relaciones subyacentes entre los datos y la información. Por ejemplo, los sistemas de ML pueden ser entrenados en sistemas automáticos de reconocimiento de voz (tal como el Iphone Siri) para convertir información acústica en una secuencia de datos habla en una estructura semántica expresada en una forma de cadenas de palabras.

Por lo tanto el objetivo de Machine Learning es predecir eventos o escenarios futuros que son desconocidos para la computadora; otra de las definiciones aceptas de Machine Learning es la citada por *ArJuevesr Samuel* en 1959 "es el campo de estudio que le da a las computadoras la habilidad para aprender sin ser explícitamente programadas" (Samuel, 1959); asimismo Tom M. Mitchell hace una definición más técnica diciendo que "Se dice que un programa de computadora aprende de una experiencia **E** con respecto a alguna clase de tarea **T** y una medida de performance **P**, si su performance en la tarea **T**, medida por **P**, mejora con la experiencia **E**"; como se puede observar esta última definición es más precisa con respecto a los componentes de un "algoritmo de aprendizaje"; ya que todos operan bajo la misma mecánica; independientemente del orden. Sin embargo, vale recalcar que ambas definiciones y hasta ahora todo lo relacionado a inteligencia artificial se encuentra ligado al famoso Test de Alan Turing (Turing, 1950); el cual establece un método por el cual se puede llamar "inteligente" a una inteligencia artificial (software, hardware o ambos); el cual precisa que; una persona no pueda reconocer las respuestas que recibe de una máquina y otro ser humano (Mariette Awad & Khanna, 2015).

2.3. Clasificación de Machine Learning:

Machine Learning se clasifica de acuerdo al tipo de aprendizaje; de los cuales tenemos (Stephen, 2014) :

2.3.1. Aprendizaje supervisado:

Dado un conjunto de datos de entrada X, compuesto por el vector $\vec{x} = \{x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(m)}\}$ (también llamado vector de características); donde m indica el número total de muestras; y un vector de salida (output o labels) $\vec{y} = \{y^{(1)}, y^{(2)}, y^{(3)} \dots y^{(m)}\}$; la tarea del aprendizaje supervisado, consiste en encontrar una función h; tal que $h: X \to y$; dicha función asume el nombre de hipótesis.

2.3.2. Aprendizaje no supervisado:

Dado un conjunto de datos de entrada X, compuesto por el vector $\vec{x} = \{x^{(1)}, x^{(2)}, x^{(3)} \dots x^{(m)}\}$; la tarea del aprendizaje no supervisado, consiste en encontrar una función h; tal que $h: X \to y$; cabe destacar que, a diferencia del aprendizaje supervisado; la función h tiene que realizar el mapeo sin el vector de salida $\vec{y}$ (Geras, 2011).

2.3.3. Aprendizaje por refuerzo:

Este tipo de aprendizaje se centra en la interacción, y el control de entornos estocásticos con conocimiento parcial; para lo cual se usa el modelo de ***proceso de decisión de Markov***; el cual define cuatro funciones (T, R, π, V) sobre el conjunto finito de estados S y acciones A; donde la función de transición T establece la probabilidad de distribución en el estado sucesor s_{t+1} de la siguiente manera $P(s_{t+1} = s' | s_t = s, a_t = a)$, al tiempo que, la recompensa r_t es establecida por la función R de la forma $P(r_t = r | s_t = s, a_t = a)$, asimismo la función de política π mapea los estados de las acciones, y finalmente la función V provee el valor de la recompensa esperada r cuando el proceso empezó en el estado s y el agente siguió la política π; finalmente a través de la interacción de las funciones y los estados, se logra

encontrar un conjunto óptimo de políticas π; con lo cual el agente resuelve el problema (Daw, 2003).

2.3.4. Aprendizaje evolutivo:

Este enfoque de aprendizaje se centra en tomar modelos de aprendizaje biológicos; los cuales adapta a los problemas de aprendizaje; algunos de estos modelos son: inteligencia de enjambre, algoritmos genéticos, sistemas artificiales inmunes (Mo, 2009), etc.; cabe destacar que en la mayoría de los modelos se suele usar una función fitness f; la cual se encarga de medir la performance de la solución en cada generación de los algoritmos (Stephen, 2014).

2.4. Algoritmos de Aprendizaje:

Como hemos visto existen diversos tipos de algoritmos de aprendizaje; los cuales están clasificados principalmente por el tipo de aprendizaje con el que se desea lidiar: supervisado, no supervisado, por refuerzo y finalmente evolutivo. Básicamente la idea de un algoritmo de aprendizaje (modelo) es encontrar una hipótesis h; la cual es una función que realiza la tarea[2] del modelo (Stephen, 2014) ; en los apartados siguientes nos estaremos centrando en los algoritmos que vamos a usar a lo largo de la investigación.

2.5. Redes Neuronales:

Una red neuronal artificial (artificial neural network) es una modelo computacional que está basado en la estructura de sus contrapartes biológicas en el cerebro, contando con unidades básicas llamadas análogamente neuronas, las cuales están interconectadas con otras neuronas formando una red de complejidad entre ellas. (Shalev-shwartz & Ben-david, 2014)

[2] Dependiendo del caso; para el aprendizaje supervisado, estaríamos hablando que h realiza un mapeo (para la regresión o predicción) o clasificación (binaria o multi – clases).

2.5.1. Neurona:

Uno de los primeros modelos de neurona se remonta a la década de 1940, cuando el psicólogo Donald Hebb (Hebb, 1949) introdujo lo que se conoce ahora como ***aprendizaje hebbiano***; el cual está basado en la propiedad de plasticidad de las neuronas, que estipula: *cuando una neurona situada a cualquier lado de la sinapsis es estimulada sincronizada y recurrentemente, la fuerza de dicha sinapsis se incrementa de manera proporcional a las respectivas salidas disparadas por las neuronas*; este proceso se define formalmente como como:

$$w_{ij}(t+1) = w_{ij}(t) + \eta_{ij} x_i(t) x_j(t) \qquad \text{Eq. (1)}$$

Donde t representa el *tiempo* de interacción, w_{ij} representa la conexión de los pesos entre las ith e jth neuronas, x_i es la salida (output) de la ith neurona y η_{ij} es el ratio de aprendizaje especifico a la sinapsis (Mariette Awad & Khanna, 2015).

Cabe destacar que el ***aprendizaje hebbiano*** es *considerado* como aprendizaje no – supervisado; ya que, en este esquema, los pesos w son actualizados de acuerdo a los demás pesos de las neuronas a las que se encuentre conectado; la primera implementación de este algoritmo se llevó a cabo en 1954 en el Instituto Tecnológico de Massachusetts usando maquinas computacionales (Farley & Clark, 1954).

2.5.2. Perceptron:

En la década de 1950 se introdujo el perceptron; el cual era un modelo de neurona de dos capas para el reconocimiento de patrones, usando operaciones de adición y sustracción (Rosenblatt, 1958). El funcionamiento del perceptron se basa en restringir la información a una señal digital simple de 0's o 1's correspondientes al estado activo o inactivo de la sinapsis; debido a esta abstracción, el perceptron puede ser representado por la siguiente función (Chellappa & Theodoridis, 2014) :

$$\mathbb{R}^n \ni x \mapsto H(\theta^T x - b) \in \mathbb{R} \qquad \text{Eq. (2)}$$

Donde $\theta \in \mathbb{R}$ representa los parámetros o pesos de la neurona, $b \in \mathbb{R}$ representa el sesgo o bias y H es la función de activación ***Heaviside***; la cual se define como (Chellappa & Theodoridis, 2014):

$$H(x) = \begin{Bmatrix} 1 & if\ 0x \geq 0, \\ 0 & otro\ caso \end{Bmatrix} \qquad \text{Eq. (3)}$$

2.5.3. Parámetros de una Neurona:

Una neurona consta de dos parámetros: estos son: (i) los valores de las conexiones entre neuronas (pesos) y (ii) el valor bias (o sesgo). Usualmente se suele representar a los pesos por w; sin embargo otra representación es usando la letra del alfabeto griego ϑ (que es la que vamos a adoptar); y finalmente al valor bias se le representa por b (Chellappa & Theodoridis, 2014)

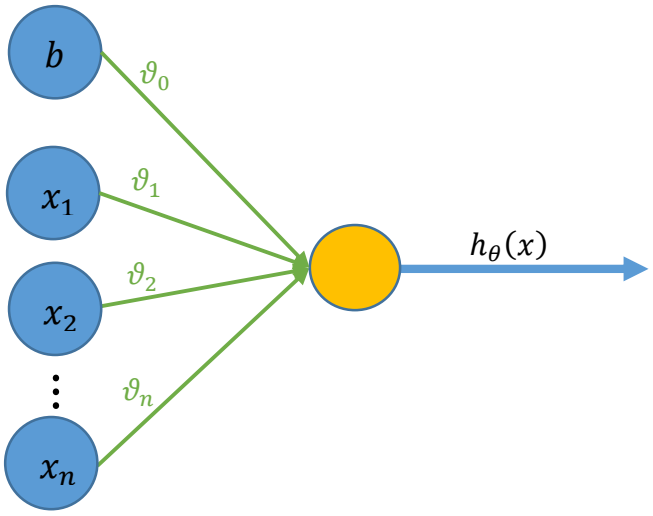


Figura N° 1: Representación de una Neurona

Donde:

$$h_\Theta(x) = \frac{1}{1 + e^{-\Theta^T x}} \qquad \text{Eq. (4)}$$

Representa la función de activación sobre los impulsos recibidos por la neurona; los cuales se computan como $\Theta^T x$. Cabe mencionar que para el caso

de la *figura N° 1*; es posible representar los parámetros θ de forma vectorial; tal que (Nielsen, 2015):

$$\theta = \begin{bmatrix} \theta^{(0)} \\ \theta^{(1)} \\ \theta^{(2)} \\ \vdots \\ \theta^{(n)} \end{bmatrix}$$

Siendo $\theta^{(0)}$ la conexión de la neurona respecto al parámetro bias. A su vez es posible realizar lo mismo con el parámetro bias; tal que (Nielsen, 2015):

$$x = \begin{bmatrix} x^{(0)} \\ x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(n)} \end{bmatrix}$$

Donde $x^{(0)}$ representa el parámetro bias; el cual está compuesto por $1's$. Más adelante veremos cómo podemos llevar ambos conceptos a una forma de matriz para cuando existan neuronas agrupadas por capas (Nielsen, 2015).

2.5.4. Inicialización de los parámetros:

La idea general que se *tiene* es que, los parámetros (pesos) deben ser iniciados aleatoriamente en un rango cercano a 0 siendo tanto positivos como negativos; pero sin llegar a *ser* 0 con el objetivo de romper la simetría. Alternativamente existen otros métodos con los que se pueden inicializar los parámetros de una red; siendo uno de ellos (Stephen, 2014):

- Usando el número de neuronas por capa $\Theta = [-\varepsilon, \varepsilon]$:

$$\varepsilon = \frac{\sqrt{6}}{\sqrt{L_{in} + L_{out}}} \qquad \text{Eq. (5)}$$

Donde:

L_{in}: *Numero* de neuronas en la capa de entrada.

L_{out}: Numero de neuronas en la capa de salida.

2.5.5. Función de activación:

Las funciones de activación son aquellas que determinan la intensidad del impulso de una neurona; la cual viaja a las neuronas adyacentes. Algunas de las funciones de activación son:

Función sigmoidea:

$$g(z) = \frac{1}{1 + e^{-z}} \quad \text{Eq. (6)}$$

Función Identidad:

$$g(z) = z \quad \text{Eq. (7)}$$

Función tangente hiperbólica:

$$g(z) = \frac{e^{\alpha Z} - e^{-\alpha Z}}{e^{\alpha Z} + e^{-\alpha Z}} \quad \text{Eq. (8)}$$

Función arco – tangente:

$$g(z) = tan^{-1}(z) \quad \text{Eq. (9)}$$

Cabe resaltar que teóricamente cualquier función diferencial $g(z)$ se puede utilizar como una función de activación, teniendo en cuenta por supuesto que su contradominio este entre valores que permitan definir la activación de y la no activación; dos de las funciones más usadas son la función identidad y la función sigmoidea (Cesar & da Fontoura Costa, 1997).

2.5.6. Redes multicapa:

Una estructura de red multicapa agrupa varias neuronas en estructuras denominadas capas L^i; donde i representa el número de capas que tiene la red (Chellappa & Theodoridis, 2014).

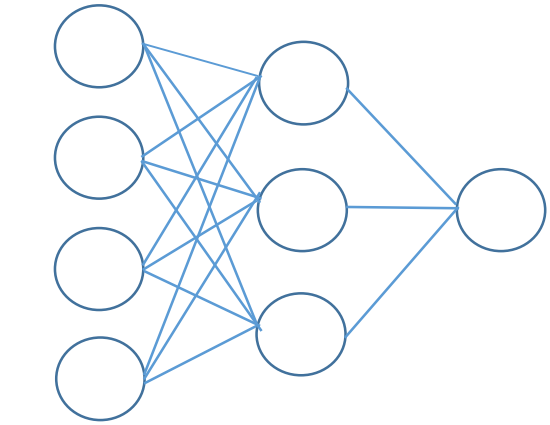


Figura N° 2: Estructura de una red multicapa

Usualmente se suelen considerar 3 capas; las cuales son (Stephen, 2014):

a) ***Capa de Entrada:*** En esta parte van las neuronas que van a representar a los atributos del data set.

b) ***Capa Oculta:*** Esta capa añade más complejidad a la red; por lo tanto ésta puede lidiar con problemas más complejos o con espacios de búsqueda más grandes.

c) ***Capa de Salida:*** Esta capa tiene un número de neuronas que es igual al número de targets o clases que se quieren clasificar[3].

Cabe resaltar que, ésta no es la única estructura; ya que una red neuronal puede estar construida por un número específico de capas; de acuerdo al contexto del problema; es así que, el número de capas ocultas puede variar desde 0 ... n capas.

2.5.7. Función Costo:

Antes de poder continuar con la implementación de la red neuronal; es importante definir una función que nos permita medir la performance del aprendizaje de la red; dicha función recibe el nombre de *función costo* debido a que computa el error de la hipótesis h con respecto a los labels y; tal que (Nielsen, 2015):

$$C(x) = \frac{1}{2}\|h(x) - y\|^2 \qquad \text{Eq. (10)}$$

[3] Para el caso del aprendizaje supervisado; y asumiendo que la tarea sea de clasificación.

Teniendo esta definición; vamos a definir a nuestra función costo mostrada en (10) por $J(x)$; la cual recibirá como entrada los parámetros de la red definidos como θ; los cuales podemos agrupar en una matriz Θ; teniendo como resultado (Ng, 2013b):

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right)\log\left(1 - \left(h_\Theta\left(x^{(i)}\right)\right)_k\right)\right] \quad \text{Eq. (11)}$$

Ahora vamos a agregar el parámetro de regularización λ; el cual se define como (Ng, 2013a):

$$\lambda = \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_l+1}\left(\Theta_{ji}^{(l)}\right)^2 \quad \text{Eq. (12)}$$

Teniendo definido el parámetro de regularización; vamos a agregar Eq. (*12*) en Eq. (*11*), obteniendo la función costo final $J(\Theta)$ con respecto a los parámetros de Θ[4], obteniendo (Ng, 2013b):

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right)\log\left(1 - \left(h_\Theta\left(x^{(i)}\right)\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{S_l}\sum_{j=1}^{S_l+1}\left(\Theta_{ji}^{(l)}\right)^2 \quad \text{Eq. (13)}$$

Nótese que si deseamos ignorar el parámetro de regularización λ; nos bastara por definir $\lambda = 0$.

2.5.8. Propagación hacia adelante (Fordward Propagation):

Dada una arquitectura de redes en multicapas; la propagación hacia adelante se define como el proceso de enviar las señales de activación desde las neuronas de entrada a través de las diversas neuronas en las capas ocultas que

[4] En este caso estamos considerando a todos los $\vartheta_0 \in \Theta$ como conexiones con el parámetro bias.

existan hasta llegar a las neuronas en la capa de salida (Chellappa & Theodoridis, 2014; Stephen, 2014). Para ilustrar mejor este proceso; usaremos la red multicapa mostrada en la *Figura N° 3*.

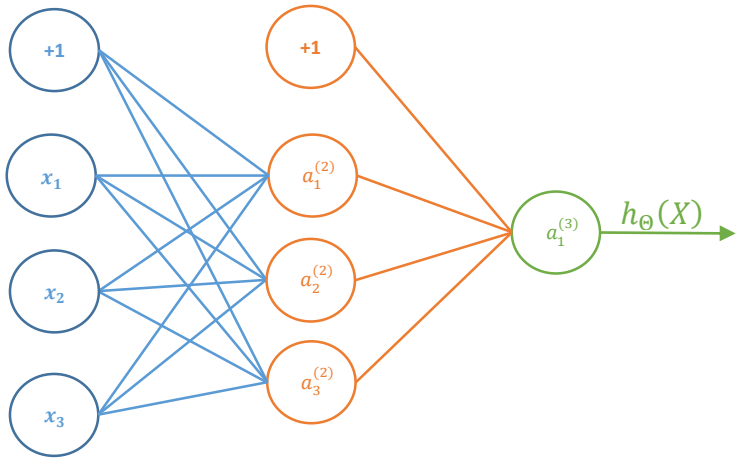


Figura N° 3: Proceso feedforward en una red multicapa

Donde:

$a_i^{(j)}$ = Activación de la unidad i en la capa j

$\Theta^{(j)}$ = Matriz de pesos[5] de la capa j a la capa $j+1$

Habiendo definido lo anterior; pasaremos a describir el proceso de propagación hacia adelante; tomando como muestra el diagrama de la red presentado por la *Figura N° 3*; podemos computar la activación en cada neurona de la siguiente manera (Ng, 2013b):

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) \qquad \text{Eq. (14)}$$
$$a_1^{(3)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$

Donde $g(z)$ viene a ser la función de activación sigmoidea definida en Eq. (*6*)

[5] Esta matriz es la encargada del control del mapeo de una capa a otra

Finalmente tenemos que computar el resultado dado por la hipótesis $h_\Theta(x)$ de la siguiente manera (Ng, 2013b):

$$h_\Theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right) \quad \text{Eq. (15)}$$

2.5.9. Algoritmo de propagación hacia atrás (Backpropagation):

La idea del algoritmo de propagación hacia atrás (backpropagation) consiste en propagar el error con respecto a la activación de la capa de salida y el label hacia las capas ocultas; sin tomar en cuenta la primera capa (dado que esta capa está compuesta por los valores del data set); dicho de esta manera; por cada nodo i en la capa l computaremos el error δ; el cual mide que tan responsable ha sido el nodo i con respecto al error en nuestra salida. Para los nodos ubicados en la capa de salida; podemos computar fácilmente el error entre la activación de la red y el label**;** mientras que para los nodos en las capas ocultas computaremos el error $\delta_i^{(l)}$ basado en el promedio ponderado de los términos del error de los nodos que usen $a_i^{(l)}$como entrada (Mariette Awad & Khanna, 2015; Ng, 2013a; Nielsen, 2015)**.**

A continuación se muestra como se realiza el cálculo del error δ:

- Para la capa de salida:

$$\delta_j^{(L)} = a_j^{(L)} - y_j \quad \text{Eq. (16)}$$

- Para las demás capas hasta δ_j^2

$$\delta_j^{(L-1)} = \left(\Theta^{(L-1)}\right)^T \delta^{L+1} \odot g'\left(z^{(L-1)}\right) \quad \text{Eq. (17)}$$

Donde $g'(z)$ es la derivada de la función sigmoidea; y se define como:

$$g'(z) = \frac{d}{dz} g(z) = g(z)(1 - g(z)) \quad \text{Eq. (18)}$$

Finalmente, antes de pasar a mostrar el algoritmo completo; vamos a definir:

$\Delta_{ij}^{(l)}$: Representa un acumulador

m: Número de observaciones del data set

$D_{ij}^{(l)}, j \neq 0$: Representa las derivadas paras los parámetros Θ

$D_{ij}^{(l)}, j = 0$: Representa las derivadas paras los parámetros bias b

λ: Representa el parámetro regularizador.

Una vez especificados los conceptos anteriores; el algoritmo de propagación hacia atrás (backpropagation) se define como (Ng, 2013a; Nielsen, 2015):
Sean los datos de aprendizaje $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Algoritmo 1: Algoritmo de Propagación hacia atrás (backpropagation)

Asignar $\Delta_{ij}^{(l)} = 0, (\forall\, l, i, j)$

For $i = 1\ to\ m$ ***do***

Asignar $a^{(i)} = x^{(i)}$

Forward propagation para computar $a^{(l)}, para\ l = 2,3 \dots, L$

Usar $y^{(i)}$ para computar $\delta^{(L)} = a^{(L)} - y^{(i)}$

Computar $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

end **for**

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \quad si\ j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \quad si\ j = 0$

2.5.10. Algoritmo Gradiente descendente (gradient descent):

El algoritmo del gradiente descendente es un método de optimización de primer orden; lo que significa que trabaja usando sólo gradientes (o sub – gradientes); el cual; dado un espacio de búsqueda $\mathbb{R}^n$; nos permite encontrar el valor mínimo local asumiendo el caso de funciones complejas; ya que para funciones simples es posible encontrar el mínimo global. El uso de este algoritmo durante el entrenamiento de las redes neuronales es con el objetivo de encontrar los valores de Θ que minimicen $J(\Theta)$; para lo cual a su vez se

hace uso del algoritmo de propagación hacia atrás (backpropagation); ya que e'ste es el encargado de computar los gradientes que son usados por el algoritmo del gradiente descendente en orden de encontrar un mínimo local aceptable. La versión del algoritmo que veremos a continuación se denomina gradiente descendente por lotes (batch gradient descent); y recibe este nombre debido a que esta versión opera sobre todo el data set; contrario a gradiente descendente estocástico (Stochastic gradient descent); el cual opera sobre cada una de los elementos $\{(x^{(i)}, y^{(i)}), \dots, (x^{(m)}, y^{(m)})\}$ del data set (Hastie & Tibshirani, 2015)

Antes de mostrar la implementación del gradiente descendente; vamos a definir:

- $W^{(l)} =$ Los parámetros (pesos) de la red.
- $\Delta W^{(l)} =$ Una matriz de *dimensiones* iguales a $W^{(l)}$.
- $b^{(l)} =$ Un *vector* de que contiene los parámetros referidos al bias.
- $\Delta b^{(l)} =$ Un *vector* de dimensiones iguales a $b^{(l)}$.
- $\alpha =$ El ratio de aprendizaje; este parámetro define el tamaño de los "pasos" que tomará el gradiente.
- $\nabla_{W^{(l)}} J(W, b; x, y) =$ Como la derivada parcial de $W^{(l)}$ respecto al costo J
- $\nabla_{b^{(l)}} J(W, b; x, y) =$ Como la derivada parcial de $b^{(l)}$ respecto al costo J
- $\lambda =$ Como el parámetro de regularización.
- $m =$ Como el número de muestras del data set.

Finalmente Δ no denota multiplicar por $W^{(l)}$. Una vez definidos estos conceptos; el gradiente descendente por lotes se define como (Hastie & Tibshirani, 2015):

Algoritmo 2: Algoritmo Gradiente Descendente (gradient descent)

---

1. Asignamos $\Delta W^{(l)} := 0, \Delta b^{(l)} := 0, \forall\, l$
2. ***For*** $i = 1\ hasta\ m$ ***do***
    a. Usar backpropgation para computar:

(i) $\nabla_{W^{(i)}} J(W, b; x, y)$

(ii) $\nabla_{b^{(i)}} J(W, b; x, y)$

b. Asignar:

(i) $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W, b; x, y)$

(ii) $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y)$

3. Actualizar los parámetros de acuerdo a:

(i) $W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$

(ii) $b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta W^{(l)} \right]$

2.5.11. Entrenamiento de una Red Neuronal:

Finalmente vamos a definir los pasos que se requieren para diseñar y/o entrenar una red neuronal usando los algoritmos de propagación hacia atrás (backpropagation) y gradiente descendente

Sea un data set definido por: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

El proceso de entrenamiento se define como (Ng, 2013b; Nielsen, 2015):

1. Iniciar aleatoriamente los parámetros (pesos)
2. Implementar la propagación hacia adelante para obtener $h_\Theta(x^{(i)})$ para cualquier $x^{(i)}$
3. Implementar la función costo $J(\Theta)$
4. Implementar backpropagation para computar las derivadas parciales $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
5. $for\ i = 1\ \ to\ \ m$
   5.1. Realizar propagación hacia delante y atrás (forward y backpropagation) usando $(x^{(i)}, y^{(i)})$
   5.2. Computar las activaciones $a^{(l)}$ y también deltas $\delta^{(l)}$ para $l = 2, \dots, L$

6. Usar el *gradiente* descendente o cualquier otro método avanzado de optimización junto con backpropagation para intentar minimizar la función costo $J(\Theta)$ en relación a los parámetros Θ.

2.6. Support Vector Machine (SVM):

SVM es un algoritmo de aprendizaje correspondiente a la categoría general de *métodos de kernel*, el cual puede ser usando tanto para clasificación (clasificación binaria y multi clasificación), como también para regresión; este método fue propuesto en 1992, en el trabajo realizado por Boser, Guyon y Vapnik denominado: *"A Training Algorithm for Optimal Margin Classifiers"* (Boser, Guyon, & Vapnik, 1992). SVM es usado en diversas áreas, presentando un alto uso en aplicaciones de clasificación con data sets de bioinformática (Ben-Hur & Weston, 2010), esto, debido a su alta precesión, su habilidad para lidiar con datos altamente dimensionales, tales como *representación de genes*, y su flexibilidad a la hora de modelar diversas fuentes de datos (Schölkopf, Tsuda, & Vert, 2004).

2.6.1. Kernels:

Un kernel se define como un algoritmo que, depende de la data sólo a través de productos vectoriales de la misma; cuando esta condición se cumple; es posible reemplazarla por una *función kernel*; la cual computa el producto vectorial en un posible *espacio de características* altamente dimensional (Ben-Hur & Weston, 2010).

El enfoque de trabajo de los kernels se basa en 4 aspectos claves:

(i) Los datos son embebidos dentro de un espacio vectorial llamado *espacio de características.*

(ii) Las relaciones lineales son buscadas en el espacio de características.

(iii) Los algoritmos son implementados de tal manera que, sólo es necesario el producto interno entre vectores en el espacio de características.

(iv) El producto interno de vectores puede ser directa y eficientemente computado desde los datos usando una función kernel.

Teniendo esto en mente, podemos pasar a la definición formal de kernel; la cual satisface las 4 caracterices antes mencionadas.

Se denomina kernel a una función K tal que, para todos los x,z de un conjunto no vacío X (el cual no debe ser un espacio vectorial), satisface la siguiente definición (Chellappa & Theodoridis, 2014):

$$K(x,z) = \langle \phi(x), \phi(z) \rangle \qquad \text{Eq. (19)}$$

Donde ϕ realiza un mapeo del conjunto X a un espacio de **Hilbert** F; el cual es usualmente llamado *espacio de características* (Chellappa & Theodoridis, 2014), siendo ϕ definido como :

$$\phi: x \in X \mapsto \phi(x) \in F \qquad \text{Eq. (20)}$$

Cabe mencionar que, dos de las grandes ventajas de poder trabajar con kernels son: (i) capacidad para generar clasificadores no lineales usando métodos definidos para clasificadores lineales y a su vez (ii) el uso de kernles nos permite aplicar un clasificador a los datos que no poseen una representación fija con respecto a sus dimensiones en el espacio vectorial (Ben-Hur & Weston, 2010).

2.6.2. Tipos de Kernels:

Algunos de los Kernels más usados son (Chih-Wei Hsu, Chih-Chung Chang, 2008):

Kernel Lineal:

$$K(x_i, x_j) = x_i^T x_j \qquad \text{Eq. (21)}$$

Kernel Polinomio:

$$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0 \qquad \text{Eq. (22)}$$

Kernel Gaussiano de función de base radial (RBF) o kernel Gaussiano:

$$k(x,z) = exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right), \sigma > 0 \qquad \text{Eq. (23)}$$

Kernel Sigmoideo:

$$K(x_i, x_j) = tanh(\gamma x_i^T x_j + r) \qquad \text{Eq. (24)}$$

Donde γ, r, d junto con σ son parámetros pertenecientes a cada kernel. Cabe mencionar que estos no son los únicos tipos de kernles; sin embargo son los que mayormente se suelen implementar, ya que, librerías como LIBSVM (Chang & Lin, 2011), suelen presentar los kernels antes mencionados. Por otro lado uno puede implementar una función que actué como kernel, independientemente de la librería o software que se use, sin embargo, para verificar que una función está actuando como kernel se puede usar el enfoque de: construir un espacio de características para los cuales el valor de la función para dos entradas, corresponde a: realizar un mapeo de características y luego computar su producto interno (Chellappa & Theodoridis, 2014).

2.6.3. Clasificación binaria usando SVM:

Para poder definir la clasificación binaria, vamos a partir de las siguientes definiciones:

Sea m el número de instancias de entrenamiento de un data set X, donde cada entrada $x_i \in X$ tiene n atributos y dos clases definidas por $y_i = -1,1$; el data set se puede representar por (Fletcher, 2009):

$$\{x_i, y_i\} \quad donde \quad i = 1 \dots m, y_i \in \{-1,1\}, x \in \mathcal{R}^n \qquad \text{Eq. (25)}$$

Para este caso, vamos a asumir que nuestros datos son linealmente separables; esto es; podemos dividir los datos usando una línea, siendo esta capaz de dividir al gráfico de x_1 vs x_2, separando las dos clases; siempre y cuando $n = 2$; y un hipeplano para el gráfico de $x_1, x_2, \dots, x_n$ cuando $n > 2$.

A su vez, el hiperplano, puede ser descrito por: $w \cdot x + b = 0$, donde:

- w es normal al hiperplano.

- $\frac{b}{\|w\|}$ es la distancia perpendicular desde el origen al hiperplano.

Se denomina SVM a los puntos del data set (*Figura N° 04*) que están ubicados tanto en $w^T x^{(i)} + b = 1$ y $w^T x^{(i)} + b = -1$ respectivamente; estos son llamados vectores de soporte; y son los puntos que se ubican al final de cada margen (Fletcher, 2009).

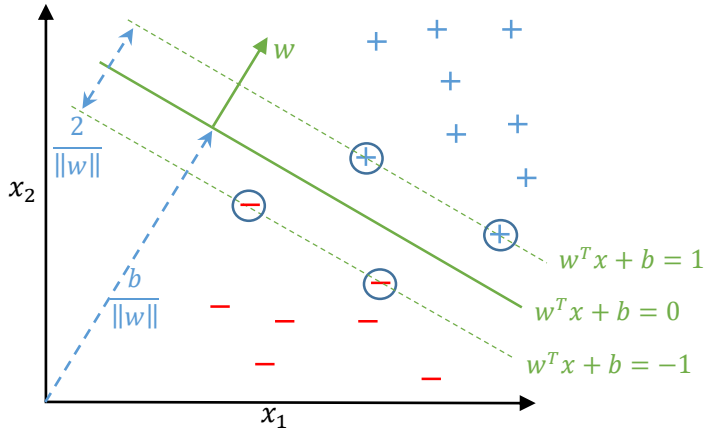


Figura N° 4: Interpretación geometría de SVM
Fuente: (Flach, 2012; Fletcher, 2009)

Tal como podemos apreciar en la *figura N° 4*, el hiperplano $\mathcal{H}^{(1)}$ descrito por $w^T x + b = 0$ divide (clasifica) las clases en dos sectores; para este caso asumiremos que los "+" representan clases positivas (+1) y los "-" clases negativas (-1), sin embargo, como podemos asegurar que el hiperplano $\mathcal{H}^{(1)}$ es el que mejor clasifica a las clases; dicho de otra manera sea $\mathcal{H} = \{h_1, h_2, \dots, h_n\}$, ¿cómo podemos saber cuál de todos es el que mejor clasifica los datos?

Una manera en la que podemos asegurar que el hiperplano $\mathcal{H}$ está realizando una buena clasificación es, a través del margen definido por $\frac{2}{\|w\|}$; ya que por medio de éste definiremos una distancia a cada lado del hiperplano $\mathcal{H}$, separando en forma más eficiente las clases; teniendo esto en mente, podemos clasificar las clases de la siguiente manera (Fletcher, 2009):

$$w^T x^{(i)} + b \geq +1 \quad si\ y^{(i)} = +1 \qquad \text{Eq. (26)}$$

$$w^T x^{(i)} + b \leq -1 \quad si\ y^{(i)} = -1 \qquad \text{Eq. (27)}$$

Las ecuaciones Eq. (*26*) y Eq. (*27*) pueden combinarse en una sola de la siguiente manera:

$$y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \qquad \text{Eq. (28)}$$

Teniendo todo esto definido, podemos ver qué; la manera en la que podemos maximizar el margen; el cual estará sujeto a la restricción impuesta por Eq. (*28*), es a través de la minimización de $\|w\|$; el cual puede ser definido de forma equivalentemente como: $\frac{1}{2}\|w\|^2$.

Finalmente la definición formal de SVM aplicada a la clasificación binaria se define como (Chellappa & Theodoridis, 2014; Fletcher, 2009):

$$\begin{aligned} &\min_{w,b} \quad \frac{1}{2}\|w\|^2 \\ &sujeto\ a \quad y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 \\ &\qquad i = 1 \ldots m \end{aligned} \qquad \text{Eq. (29)}$$

Más adelante veremos cómo extender este concepto para lidiar con problemas de multi – clasificación.

2.6.4. Multi – clasificación usando SVM:

Anteriormente se definió SVM para problemas de clasificación binaria; sin embargo SVM también se puede aplicar para problemas de multi – clasificación (problemas que involucren más de dos clases[6]); en orden de mostrar la implementación de SVM para multi – clases, vamos a partir introduciendo una variable de holgura ξ (como se observa en la *Figura N° 5*) a las restricciones para la clasificación binaria, definidas por Eq. (*26*) y Eq. (*27*), obteniendo las siguientes restricciones:

$$w^T x^{(i)} + b \geq +1 - \xi_i \quad si\ y^{(i)} = +1 \qquad \text{Eq. (30)}$$

$$\begin{aligned} &w^T x^{(i)} + b \leq -1 + \xi_i \quad si\ y^{(i)} = -1 \\ &\xi_i \geq 0, \quad i = 1 \ldots m \end{aligned} \qquad \text{Eq. (31)}$$

[6] También llamadas labels o targets.

A su vez, podemos re – definir las restricciones de las Eq. (*30*) y Eq. (*31*) como:

$$y^{(i)}\left(w^T\phi\left(x^{(i)}\right)+b\right)\geq 1-\xi_i,\ \ i=1,\dots,m$$
$$\xi_i\geq 0,\ \ i=1,\dots,m \qquad \text{Eq. (32)}$$

Donde ϕ representa una función kernel. Como podemos observar en la *figura N° 5* la variable de holgura ξ nos permite trabajar con una *margen suave*, ya que:

Para $0<\xi\leq 1$ se define un punto que está en el lado correcto del hiperplano, pero se encuentra en medio del margen; y para $\xi>1$ representa un punto mal clasificado; en ambos casos las restricciones originales definidas en Eq. (*26*) y Eq. (*27*) han sido *relajadas*; lo cual implica que se pagará una penalización; la cual veremos implementada más adelante.

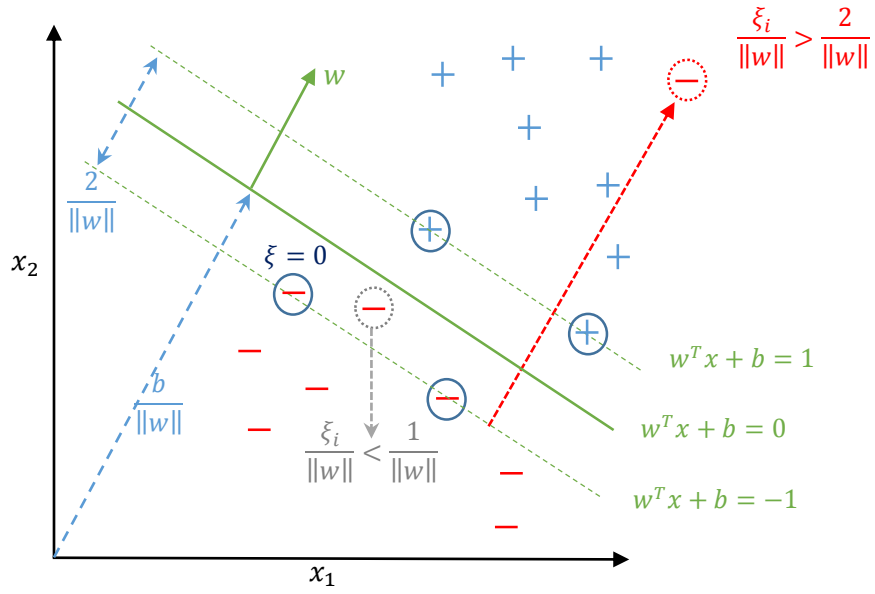


Figura N° 5: Interpretación geometría de SVM con variables de holgura

Fuente: (Flach, 2012; Fletcher, 2009)

Finalmente el problema consiste de igual manera en minimizar el margen; para lo cual re – definimos Eq. (*32*), agregándole las variables de holgura ξ_i, obteniendo así (Chellappa & Theodoridis, 2014; Fletcher, 2009):

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i$$
$$sujeto\ a \quad y^{(i)}\left(w^T\phi\left(x^{(i)}\right)+b\right) \geq 1-\xi_i,\ \ i=1,\dots,m \qquad \text{Eq. (33)}$$
$$\xi_i \geq 0,\ \ i=1,\dots,m$$

Donde el parámetro C controla la compensación entre la variable de holgura ξ y el tamaño del margen (Chellappa & Theodoridis, 2014)**.**

2.6.5. Regresión usando SVM:

En los problemas de regresión, dado un data $X \in \mathcal{R}^n$ y un label $y_i \in \mathcal{R}$; el cual posee valores numéricos, donde $(i = 1 \dots m)$; el objetivo es encontrar una función $f(x^{(i)})$ que pueda mapear los valores aproximados a y_i (Stephen, 2014)

Llevando este concepto a SVM podemos definir la forma estándar usada para la regresión usando como (Chellappa & Theodoridis, 2014):

$$\min\ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\left|y^{(i)} - f\left(x^{(i)}\right)\right|_\epsilon \qquad \text{Eq. (34)}$$

Donde el parametro C actúa como compensador entre el margen y la perdida empírica y ϵ actúa como el sesgo del clasificador (Chellappa & Theodoridis, 2014).

2.7. Principal Component Analysis (PCA):

PCA es un método correspondiente al análisis multivariable; a su vez que es uno de los más conocidos y más antiguos; fue propuesto por Pearson en 1901 en su trabajo titulado: *"On Lines and Planes of Closest Fit to Systems of Points in Space"*(Pearson, 1901); en este trabajo Pearson introduce al PCA en un contexto biológico; a su vez el concepto siguió siendo desarrollado por Hotelling en su trabajo: "*Analysis of a complex of statistical variables into principal components"*(Hotelling, 1933); es

desde ese punto en adelante que muchos trabajos más han sido propuestos dando forma a la idea de PCA que conocemos ahora.

PCA tiene por objetivo proyectar características dimensionales altas d dentro de un sub grupo de menor tamaño k, manteniendo la mayor **varianza** de los datos en el proceso; dos de las aplicaciones más usadas de PCA corresponden a:

(i) La compresión de datos; donde la tarea consiste en representar todos los datos con un número reducido de dimensiones a su vez que, se asegura la mínima distorsión de los mismos (debido a la proyección de los datos a una dimensión menor).
(ii) Extracción de características, donde el objetivo consiste en reducir las dimensiones de los datos mientras se retiene el mayor contenido de los mismos (Chellappa & Theodoridis, 2014).

Una de las aplicaciones usadas por (i) es la comprensión y restauración de imágenes, mientras que, con respecto a la aplicación (ii), se puede usar la data reducida para ser procesada por diversos algoritmos; reduciendo así el tiempo de procesamiento de los mismo. En orden de realizar la proyección; PCA computa nuevas características $Pc's$ (componentes principales); para lo cual utiliza métodos de transformaciones ortogonales para poder proyectar las características de una dimensión d a una dimensión k; donde $k < d$ (Mariette Awad & Khanna, 2015); siendo uno de los métodos ampliamente utilizados SVD (Shlens, 2005).

2.7.1. Valor singular de descomposición (SVD):

SVD puede ser definido de los siguientes puntos de vista (Baker, 2013):

(i) Un método para transformar variables correlacionadas dentro de un conjunto de variables sin correlación; las cuales muestran las relaciones entre las variables de los datos originales.
(ii) Un método para identificar y ordenar las dimensiones en las cuales los puntos de los datos exhiben (proyectan) una mayor varianza.
(iii) Un método para encontrar la mejor aproximación a los datos originales usando menos dimensiones.

Matemáticamente hablando SVD toma uno de los teoremas del algebra lineal; el cual estipula que una matriz rectangular X puede ser divida en el producto de tres matrices: una matriz ortogonal U, una matriz diagonal Σ y la transpuesta de una matriz ortogonal V; tal que:

$$X = U\Sigma V^T \qquad \text{Eq. (35)}$$

Donde $U^T = I, V^T V = I$; las columnas de U son vectores propios (eigenvectors) ortonormales de XX^T; a su vez las columnas de V representan los valores propios (eigenvalues) de $X^T X$, finalmente Σ representa una matriz diagonal de las raíces cuadradas de los valores propios (eigenvalues) de U o V ordenados de manera descendente.

2.7.2. Algoritmo PCA:

En orden de realizar la reducción de dimensiones; PCA opera de la siguiente manera (Mariette Awad & Khanna, 2015):

1. Pre – procesamiento:
   1.1. Normalización:
   (i) Iniciamos obteniendo la media de los datos definida por:

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \qquad \text{Eq. (36)}$$

   (ii) Reemplazaremos cada $x_j^{(i)}$ en los datos por: $x_j^{(i)} - \mu_j$

   1.2. Escalamiento
   (iii) Ahora obtendremos la varianza de los datos definida por:

$$\sigma_j^2 = \frac{1}{m}\sum_{i}\left(x_j^{(i)}\right)^2 \qquad \text{Eq. (37)}$$

(iv) Una vez obtenida la varianza; la usaremos para remplazar cada $x_j^{(i)}$ por $x_j^{(i)}/\sigma_j$

2. Computar la matriz de covarianza Σ

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x_i)(x_i)^T \qquad \text{Eq. (38)}$$

3. Computar los vectores propios (eigenvectors) U de la matriz Σ y la matriz diagonal S usando la definición de SVD descrita la Eq. (*35*); con lo cual obtenemos:

   Para U:

$$U = \begin{bmatrix} \vdots & \vdots & & \vdots \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ \vdots & \vdots & & \vdots \end{bmatrix} \in \mathbb{R}^{n\times n}$$

   Para S:

$$S = \begin{bmatrix} S_{(1,1)} & 0 & \cdots & 0 \\ 0 & S_{(2,2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & S_{(n,n)} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

4. Seleccionar de U los top[7] *k – vectores propios (eigenvectors)* de tal modo que la perdida de la varianza este en relación a:

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01 \qquad \text{Eq. (39)}$$

   Con respecto a la Eq. (*39*); es posible re – definirla en términos de la matriz diagonal S obtenida por la Eq. (*35*); con lo que obtendremos:

$$1 - \frac{\sum_{i=1}^{k} S_{(i,i)}}{\sum_{i=1}^{n} S_{(i,i)}} \leq 0.01 \qquad \text{Eq. (40)}$$

[7] Referido a los k vectores propios (eigenvectors) con los valores propios (eigenvalues) más largos.

Alternativamente también podemos representar la Eq. (*40*) en términos del porcentaje de la varianza retenida de la siguiente manera:

$$\frac{\sum_{i=1}^{k} S_{(i,i)}}{\sum_{i=1}^{n} S_{(i,i)}} \geq 0.99 \qquad \text{Eq. (41)}$$

Cabe mencionar que con estos procesos Eq. (*39*), Eq. (*40*) y Eq. (*41*) se está asegurando que se retenga la mayor varianza posible de los datos originales.

2.8. MSE – Error Cuadrático Medio:

El error cuadrático medio es sin duda el más importante criterio utilizado para evaluar el desempeño de un indicador o un estimador (la distinción sutil entre predictores y estimadores es que las variables aleatorias se predicen y constantes se calculan). El error cuadrático medio también es útil para transmitir los conceptos de sesgo, precisión y exactitud en la estimación estadística. Para examinar un error cuadrático medio, se necesita un objetivo de la estimación o predicción, y un predictor o estimador que es una función de los datos (Toh & Romay, 2014). Dicho esto; el error cuadrático medio se define como:

$$\frac{1}{2m}\sum_{i=1}^{m}\left(h\left(x^{(i)}\right) - y^{(i)}\right)^2 \qquad \text{Eq. (42)}$$

Donde:

$m =$ Número de observaciones del data set

$y^{(i)} =$ Valor esperado

$x^{(i)} =$ Representa una instancia del data set; tal que $x^{(i)} \in X^{(n)}$

$h\left(x^{(i)}\right) =$ Aplicación de la hipótesis sobre la instancia $x^{(i)}$

2.9. Curvas de Aprendizaje:

Usamos curvas de aprendizaje para poder detectar si nuestro algoritmo sufre de (Dietterich & Kong, 1995) :

- **Varianza Alta (High Variance):** La variazan alta (high variance) se presenta cuando el algoritmo logra obtener una alta exactitud en los datos de aprendizaje; a este fenómeno se le conoce también como "overfit" de los datos. Gráficamente cuando el algoritmo presenta "overfit" de sus parámetros se suele observar una distribución de los datos de la siguiente manera:

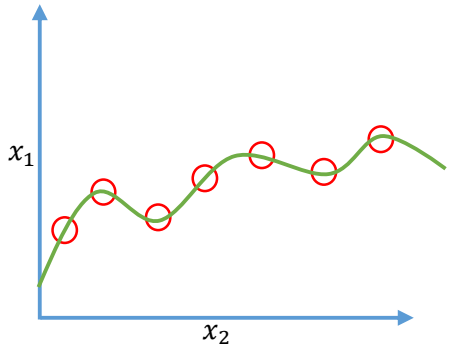


Figura N° 6: Predictor con overfit

- **Sesgo alto (High Bias):** Cuando un algoritmo sufre de sesgo alto (high bias) implica que el algoritmo no puede modelar correctamente la distribución de los datos para poder predecir las clases; esto se suele traducir en qué algoritmo presenta "underfit" con respecto a los datos. Gráficamente cuando el algoritmo presenta "underfit" de sus parámetros se suele observar una distribución de los datos de la siguiente manera:

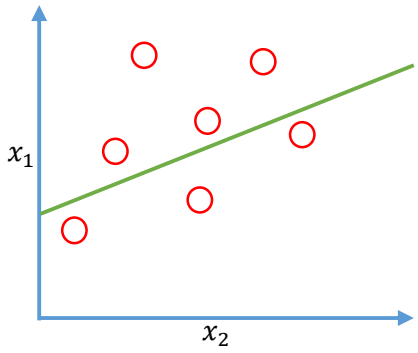


Figura N° 7: Predictor con underfit

Para poder realizar el diagnóstico de cuál de los problemas antes mencionados (high bias, high variance) sufre nuestro algoritmo; vamos a usar la ecuación del Error cuadrático Eq. (*42*); tal que:

Para los datos de entrenamiento como:

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \qquad \text{Eq. (43)}$$

Para los datos del cross validation como:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right) - y_{cv}^{(i)}\right)^2 \qquad \text{Eq. (44)}$$

Una vez definidas las Eq. (*43*) y Eq. (*44*); procedemos a realizar la gráfica de las curvas de aprendizaje; las cuales suelen presentar las siguientes formas; dependiendo del problema que presente el algoritmo:

- ***Curva para identificar sesgo alto (High bias)*:**

  Típicamente; cuando nuestro sistema presenta un sesgo alto; es común observar una curva donde tanto el error en ambos conjuntos de entrenamiento (train y validation) comienzan a disminuir; hasta casi llegar a juntarse; claro está, en la práctica se suelen observar la presencia de picos; sin embargo la distribución general se asemeja con la representación de la figura Figura N° 8

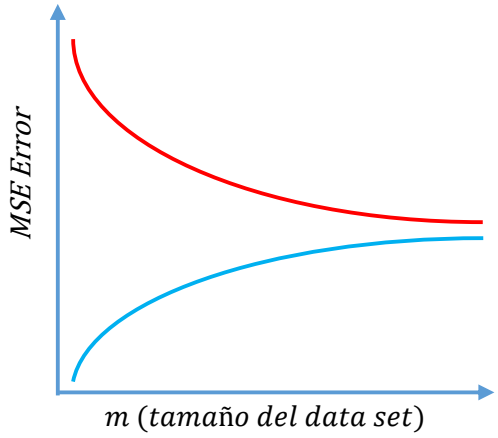


Figura N° 8: Curva de Aprendizaje con sesgo alto (high bias)

  Cuando nos encontramos con este tipo de problema; algunas formas de solucionarlo consisten en las siguientes:

  - Agregar más características (dimensiones)
  - Agregar características polinomiales
  - Disminuir el valor del parámetro de regularización λ

- ***Curva para identificar varianza alta (High variance)***

  Cuando nuestro sistema presenta una varianza alta; se puede observar que tanto el error en ambos conjuntos (train y validation) deja de disminuir en cierto punto y se mantiene a través de toda la gráfica con pequeñas variaciones; tal como se observa en la Figura N° 9.

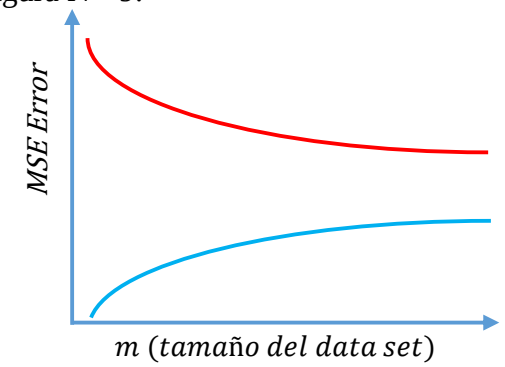


Figura N° 9: Curva de Aprendizaje con varianza alto (high variance)

Algunas de las acciones que podemos realizar cuando estamos lidiando con un sistema que presenta varianza alta son:

- Agregar más datos de entrenamiento
- Reducir las características (ayudar a reducir el "overfit" del algoritmo)
- Incrementar el valor del parámetro regularizador λ

2.10. Matriz de confusión:

En el área de Machine Learning; la matriz de confusión es un diseño de tabla específica; la cual permite la visualización de los resultados de aprendizaje de un algoritmo, típicamente se aplica al aprendizaje supervisado; donde cada columna representa la clase real (label); mientas que las finales representan las clases predichas por el algoritmo (o vice – versa); esto nos permite visualizar si el sistema está mal clasificando las clases. A continuación se muestra la implementación de una matriz de confusión para dos clases A y B (Asch, 2013):

LABELS / **PREDICCIONES**	**CLASE A**	**CLASE B**
CLASE A	TP	FP
CLASE B	FN	TN

Tabla N° 1: Modelo de Matriz de confusión

Donde:

TP = Verdadero positivo; el cual representa el número de clases "positivas" que han sido clasificadas correctamente.

TN = Verdadero negativo; representa el número de clases "negativas" que han sido clasificadas correctamente.

FP = Falso positivo; representa el número de clases que siendo "negativas" han sido clasificadas como "positivas"

FN = Falso negativo; representa el número de clases que siendo "positivas" han sido clasificadas como "negativas"

2.11. Métricas de Performance:

A continuación vamos a mencionar algunas de las métricas usadas para medir la performance de los algoritmos (modelos) de Machine Learning

2.11.1. Precisión:

La precisión puede ser definida desde dos perspectivas:

(i) Desde el punto de vista de **Recuperación de Información** (information retrieval); se denomina precisión como el número de documentos relevantes regresados por la búsqueda, dividido entre el número total de documentos retornados por la búsqueda.

(ii) En clasificación se define la precisión de una clase como el número de verdaderos positivos dividido por el número total de elementos clasificados como positivos

Teniendo esas definiciones; la precisión se define como (Özgür, Özgür, & Güngör, 2005):

$$\pi = \frac{TP}{TP + FP} \qquad \text{Eq. (45)}$$

Donde:

$TP =$ Número de Verdaderos Positivos.

$FP =$ Número de Falsos Positivos.

2.11.2. Recall (sensibilidad):

Al igual que con la precisión; recall (sensibilidad) puede ser definido desde dos perspectivas:

(i) Desde el punto de vista de **Recuperación de Información** (information retrieval); se denomina recall como el número de documentos relevantes regresados por la búsqueda, dividido entre el número existe de documentos relevantes.

(ii) En clasificación se define recall como el número de verdaderos positivos dividido por el número total de elementos que actualmente pertenecen a la clase positiva

Teniendo esas definiciones; recall se define como (Özgür et al., 2005)(Özgür et al., 2005):

$$\rho = \frac{TP}{TP + FN} \qquad \text{Eq. (46)}$$

Donde:

$TP =$ Número de Verdaderos Positivos.

$FN =$ Número de Falsos Negativos.

2.11.3. F – Micro averaged Score:

Para poder definir F – Micro averaged Score; vamos a realizar primero un re – definición de la precisión y recall definidas en las Eq. (*45*) y Eq. (*46*) respectivamente; obteniendo:

$$\pi = \frac{TP}{TP + FP} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m} (TP_i + FP_i)} \qquad \text{Eq. (47)}$$

$$\rho = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m} (TP_i + FN_i)} \qquad \text{Eq. (48)}$$

Donde m es el número de categorías. Una vez re – definidas la precisión y recall; usaremos las Eq. (*47*) y Eq. (*48*) para obtener la definición formal de Micro F – Score como (Özgür et al., 2005):

$$F(micro - averaged) = \frac{2\pi\rho}{\pi + \rho} \qquad \text{Eq. (49)}$$

Finalmente cabe mencionar que F – Micro averaged Score; asigna igual peso a cada documento; y es por tanto considerado como el promedio sobre todos los pares de documentos / categorías. A su vez tiende a ser dominado por la performance del clasificador en categorías comunes (Özgür et al., 2005)

2.11.4. Curva ROC (Receiver Operating Characteristics):

Una curva ROC es una técnica gráfica para la visualización, organización y selección de clasificadores basada en su performance (Fawcett, 2006). Dentro del Área de Machine Learning, uno de los primeros en adoptar y/o aplicar los gráficos ROC fue Spackman; el cual demostró el valor de las curvas ROC en la evaluación y comparación de algoritmos (Spackman, 1989). El espacio de los gráficos ROC está conformado por dos dimensiones; donde los verdaderos positivos son representados en eje Y; mientras que los falsos positivos son representados en el eje X; el gráfico ROC muestra la compensación relativa entre beneficios (verdaderos positivos) y costos (falsos positivos) (Fawcett, 2006). Para poder crear un gráfico de curva ROC; se hace necesario construir la matriz de confusión de los modelos; una vez obtenidos los valores de la matriz; procedemos a computar los valores de los ejes X, Y para cada uno de los modelos en evaluación los cuales son los ratios de falsos positivos y verdaderos positivos; éstos se obtienen de (Fawcett, 2006):

$$fp\_rate \approx \frac{negativos\_incorrectamente\_clasificados}{total\_negativos} \qquad \text{Eq. (50)}$$

$$tp\_rate \approx \frac{positivos\_clasificados\_correctamente}{total\_positivos} \qquad \text{Eq. (51)}$$

Donde:

$fp\_rate =$ falsos negativos

$tp\_rate =$ verdaderos positivos

Una vez obtenidos estos valores; para poder identificar la performance de los algoritmos dentro de un gráfico ROC; es necesario tener en cuenta 3 coordenadas dentro del espacio del gráfico; las cuales son importantes; ya que nos sirven de guía para poder medir la performance de los algoritmos; dichas coordenadas se distribuyen de la siguiente manera (Fawcett, 2006):

(i) El punto izquierdo inferior se ubica en las coordenadas (0,0), éste representa la estrategia de nunca emitir una clasificación positiva; los clasificadores que caen en ésta coordenada, no presentan errores de falsos positivos, y a su vez muestran una performance nula con respecto a verdaderos positivos.

(ii) El punto superior derecho se ubica en las coordenadas (1,1), éste punto representa la estrategia contraria a la de las coordenadas (0,0); y representa la estrategia de realizar clasificaciones positivas de manera incondicional; los clasificadores que caen en este cuadrante; muestran una performance alta de verdaderos positivos; sin embargo hacen pasar por verdaderos positivos a falsos negativos.

(iii) Finalmente el punto ubicado en las coordenadas (0,1), representa la clasificación perfecta; esto es; el número de falsos positivos es 0 (el algoritmo no mal clasifica a ninguna de las clases), mientras que el número de verdaderos positivos es 1 (el algoritmo detecta correctamente a todas las clases).

Para ilustrar mejor estos conceptos, vamos a proponer el siguiente ejemplo: Supongamos que hemos entrenado 4 clasificadores (A, B, C, D y E) para detectar las clases $y = \{0,1\}$; una vez que hemos entrenado y validado los modelos respectivamente; pasamos a realizar el gráfico ROC; obteniendo lo siguiente (Fawcett, 2006):

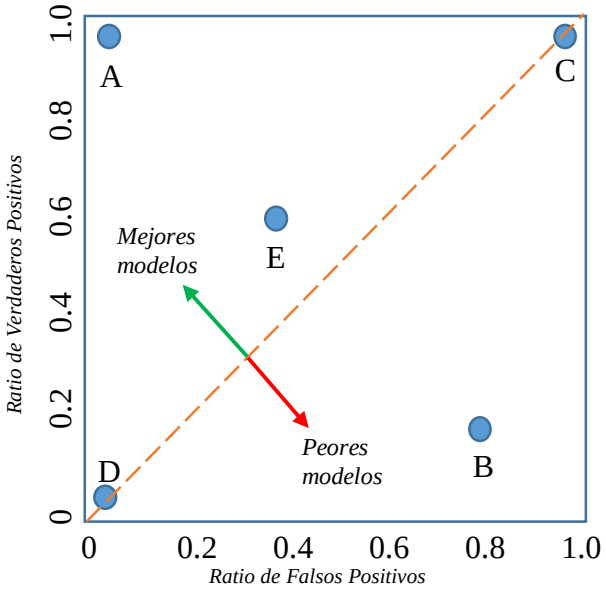


Figura N° 10: Gráfico ROC
Fuente: (Fawcett, 2006)

De acuerdo al análisis realizado anteriormente acerca de las coordenadas; podemos realizar varias observaciones (Fawcett, 2006):

(i) El modelo ***C*** se ubica en el punto (1,1); este modelo presenta el problema de que a todas las clases de y las hace pasar por positivas; incluyendo a las negativas.

(ii) El modelo ***E*** se ubica en el punto (0.4, 0.6); este modelo presenta un índice de performance de aprendizaje media – baja; ya que tiene una capacidad para detectar a las clases correctamente (verdaderos positivos); sin embargo presenta también ciertos problemas con respecto al número de falsos positivos; esto significa que, dentro de las clases que este modelo predice adecuadamente; un porcentaje de ellas están siendo

mal clasificadas como positivas. Usualmente esto es un indicador de que se debe reentrenar el modelo.

(iii) El Modelo ***B***, ubicado en el punto (0.8, 0.2), presenta una performance de aprendizaje muy baja; ya que la mayoría de sus clases detectadas pertenecen a falsos positivos; mientras que sólo una pequeña porción son verdaderos positivos (clases clasificadas correctamente); para este caso al igual que el anterior, se hace necesario realizar el reentrenamiento total del modelo.

(iv) El modelo ***D*** se ubica en el punto (0,0); y, presenta la performance de aprendizaje más baja de todos los 5 modelos.

(v) Finalmente el mejor modelo de todos (el que presenta la performance de aprendizaje más alta) es ***A***; ya que este se ubica en el punto (0,1); presentando un porcentaje de falsos negativos de 0.

Como hemos podido apreciar; existe un sesgo en el gráfico; el cual divide a los modelos en dos áreas; los modelos que se encuentran ubicados hacia la parte izquierda superior; presentan mejores características de aprendizaje; mientras que los modelos que se ubican hacia la parte derecha (superior e inferior) presentan una performance de aprendizaje baja. Cabe aclarar que; en aplicaciones reales; los mejores modelos no suelen estar siempre de manera rígida en el punto (0,1) como se mostró en el ejemplo anterior; sino que; estos se aproximan a ese cuadrante por medio de una curva (de ahí el nombre de curva ROC); podremos observar en mayor detalle ese comportamiento en el CAPÍTULO V.

2.12. Framework de proyectos con Machine Learning:

J. Bell nos ofrece un framework *"El Ciclo de Machine Learning"* donde se detalla el ciclo de vida que suelen tener los proyectos de Machine Learning (Bell, 2014):

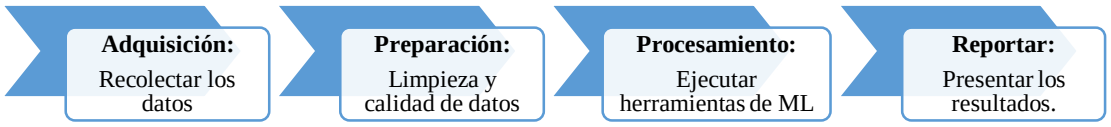


Figura N° 11: El Ciclo de Machine Learning

Fuente: (Bell, 2014)

A su vez M. Stephen define un framework más detallado al que denomina *"El proceso de Machine Learning";* este framework abarca más fases que el propuesto por J. Bell, sin embargo se pueden apreciar ciertas similitudes entre las fases (Stephen, 2014):

Figura N° 12: El proceso de Machine Learning

Fuente: (Stephen, 2014)

Finalmente M. Awad y R. Khanna definen un framework específico aplicado al aprendizaje supervisado (Mariette Awad & Khanna, 2015):

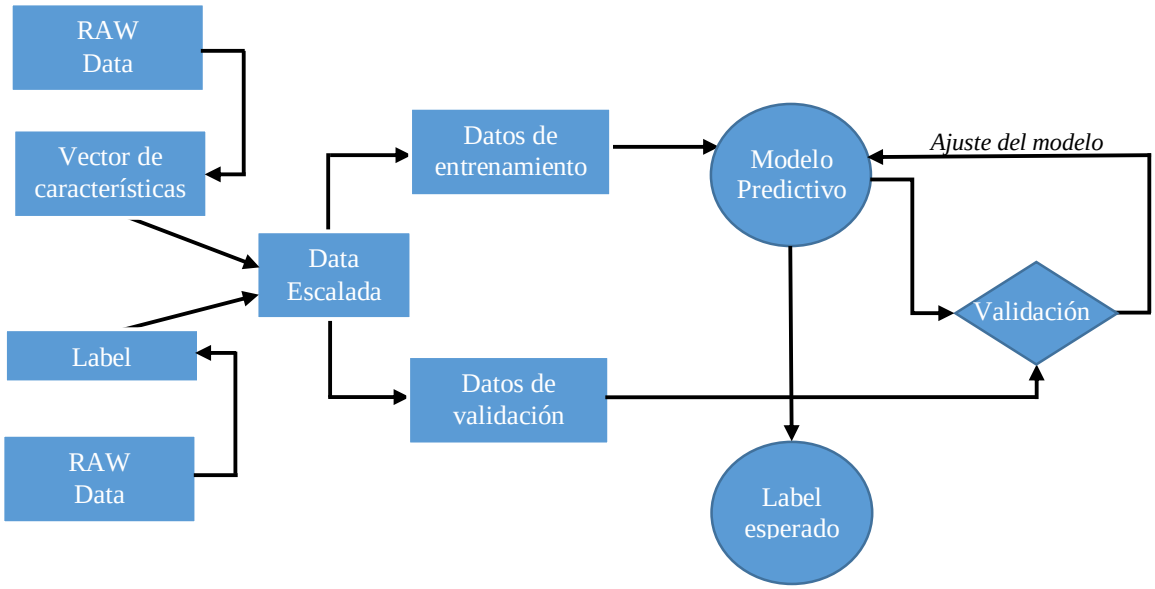


Figura N° 13: Flujo de alto nivel de aprendizaje supervisado

Fuente: (Mariette Awad & Khanna, 2015)

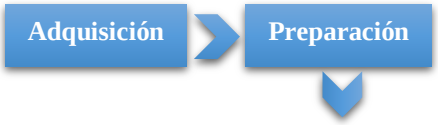
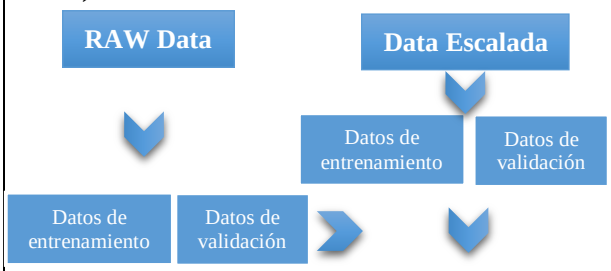
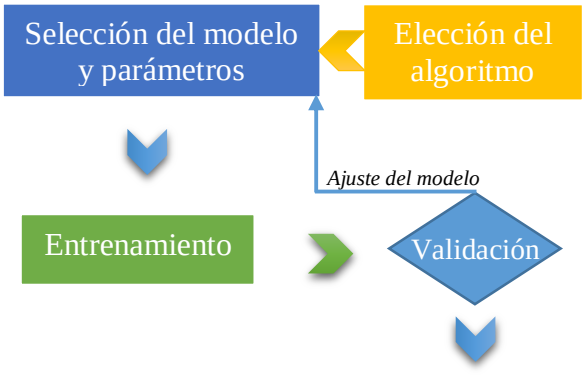
Capítulo III: Metodología

En este Capítulo se muestra el desarrollo de la metodología propuesta; la cual consta de 5 secciones en total; las cuales se agrupan en dos secciones principales; las 4 primeras están enfocadas en la gestión del data set, así como en la implementación y validación de los algoritmos; mientras que la última se enfoca en definir las métricas que serán usadas para medir la performance de aprendizaje de los algoritmos. Durante el desarrollo del *Capítulo IV* se implementan las secciones 1 – 4; mientras que en el *Capítulo V* se implementa la sección 5

3.1. Metodología Propuesta:

La metodología propuesta se compone de 5 secciones en total; y se encuentra dividida en dos secciones generales; la primera de ellas abarca el framework de desarrollo e implementación de proyectos (sistemas) de Machine Learning (ML); y comprende desde los puntos 1 – 4 de la misma; mientas que la segunda se centra en la definición y/o uso de métricas propias de Machine Learning (ML) para poder medir la performance de aprendizaje de los algoritmos; y comprende el punto 5; ésta última es usada para poder realizar la contrastación de la hipótesis.

La metodología propuesta para demostrar el aporte de PCA a la performance de aprendizaje de los algoritmos SVM y MLNN se basa en el framework "El Ciclo de Vida de Machine Learning" (Bell, 2014), "El proceso de Machine Learning" (Stephen, 2014) y "Flujo de alto nivel de aprendizaje supervisado" (Mariette Awad & Khanna, 2015); con lo cual estamos tomando ambas metodologías para de este modo poderlas ajustar a nuestro problema específico; obteniendo lo siguiente:

<table>
<tr><th colspan="2">DESARROLLO DE LA METODOLOGÍA</th></tr>
<tr><th>Frameworks Bases</th><th>Metodología Propuesta</th></tr>
<tr><td colspan="2">DESARROLLO E IMPLEMENTACIÓN DE PROYECTOS DE MACHINE LEARNING</td></tr>
<tr><td>“El ciclo de Machine Learning” (Bell, 2014)
 
</td><td>1. Obtener el data set:
1.1. Buscar y/o obtener fuente de datos
1.2. Pre – procesar los datos (limpieza)
1.3. Descripción de características
1.4. Crear Diccionario de Datos.</td></tr>
<tr><td>“Flujo de alto nivel de aprendizaje supervisado” (Mariette Awad & Khanna, 2015)
 
</td><td>2. Dividir el Data set en dos grupos:
2.1. Grupo con PCA (training set, test set, cross validation set)
2.2. Grupo sin PCA (training set, test set, cross validation set)</td></tr>
<tr><td rowspan="2">“El proceso de Machine Learning” (Stephen, 2014), “Flujo de alto nivel de aprendizaje supervisado” (Mariette Awad & Khanna, 2015)
 
</td><td>3. Entrenar y validar algoritmos usando data set sin PCA (npca[8]):
3.1. Entrenar y validar SVM_NPCA (usando test y cross validation set)
3.2. Entrenar y validar MLNN_NPCA (usando test y cross validation set)</td></tr>
<tr><td>4. Entrenar y validar algoritmos usando data set con PCA (pca[9])
4.1. Entrenar y validar SVM_PCA (usando test y cross validation set)
4.2. Entrenar y validar MLNN_PCA (usando test y cross validation set)</td></tr>
<tr><td colspan="2">MEDIDA DE PERFORMANCE DE APRENDIZAJE</td></tr>
</table>

[8] Referido a RAW Data sin procesamiento de PCA
[9] Referido a Data que ha sido procesada usando PCA

"El ciclo de Machine Learning" (Bell, 2014) **Reportar**	5. Comparar performance de aprendizaje entre algoritmos usando las métricas de Machine Learning SVM_NPCA vs SVM_PCA y MLNN_NPCA vs MLNN_PCA

Tabla N° 2: Metodología de Desarrollo Propuesta

Capítulo IV: Desarrollo

En este Capítulo se comienza a realizar la implementación de los algoritmos SVM, MLNN y PCA respectivamente; para lo cual se hace uso de las secciones 1 – 4 de la metodología descrita en el *Capítulo III*; comenzando con las tareas aplicadas al dataset; para, finalmente pasar a la implementación y validación formal de los algoritmos (SVM, MLNN y PCA); la implementación de la sección 5 no es realizada en éste Capítulo; ya que ésta se enfoca en las mediciones de la performance de aprendizaje; y para ello es necesario que primero se implementen y validen los algoritmos; no obstante la sección 5 de la metodología es implementada en el *Capítulo V*.

4.1. Data set:

Primeramente obtendremos el data set *QSAR biodegradation* (Mansouri et al., 2013) de la fuente de repositorios gratuitos *Machine Learning Repository*.

4.1.1. Descripción del Data set:

En el data set *QSAR biodegradation*; los datos se han utilizado para desarrollar modelos QSAR (relaciones cuantitativas estructura-actividad) para el estudio de las relaciones entre la estructura química y la biodegradación de moléculas; (1055) valores experimentales de Biodegradación de los productos químicos se obtuvieron de la página web del National Institute of Technology and Evaluation of Japan (NITE). Los modelos de clasificación se desarrollaron con el fin de discriminar moléculas biodegradables listas (356) y no listas (699) mediante tres métodos de modelado diferentes: k Nearest Neighbours, Partial Least Squares Discriminant Analysis and Support Vector Machine. Los detalles de los atributos (descriptores moleculares) seleccionados en cada modelo, pueden ser encontrados en el trabajo: *Mansouri, K., Ringsted, T., Ballabio, D., Todeschini, R., Consonni, V. (2013). Quantitative Structure - Activity Relationship models for ready biodegradability of chemicals. Journal of Chemical Information and Modeling, 53, 867-878.*

4.1.2. Descripción de características:

A continuación se muestra una descripción detallada de las características del Data set:

Características del Data set	Multivariable	**Número de Instancias**	1055	**Número de Clases**	2
Características de los atributos	Entero, real	**Número de Atributos**	41	**Área**	N/A
Tareas Asociadas	Clasificación	**Valores perdidos**	N/A[10]	**Fecha de Donación:**	2013 – 06 – 21

Tabla N° 3: Descripción de características del Data Set

4.1.3. Diccionario de datos:

N°	NOMBRE DE LA VARIABLE	DESCRIPCIÓN	VALORES
01	SpMax_L	Valor propio (eigenvalue) interlineado de la matriz de Laplace	2 … a … 6.4960
02	J_Dz(e)	Índice de Balaban de la matriz Barysz, ponderada por la electronegatividad de Sanderson	0.8039… a … 9.1775
03	nHM	Número de átomos pesados	0 … a … 12
04	F01[N-N]	Frecuencia de N-N a distancia topológica de 1	0 ... a ... 3
05	F04[C-N]	Frecuencia de C-N a distancia topológica 4	0 ... a ... 36
06	NssssC	Número de átomos de tipo 'ssssC'	0 ... a ... 13
07	nCb-	Número de benceno sustituido C(sp^2)	0 ... a ... 18
08	C%	Porcentaje de átomos C	0 ... a ... 60.7
09	nCp	Número de terminales primarias C(sp^3)	0 ... a ... 24
10	nO	Número de átomos de oxigeno	0 ... a ... 12
11	F03[C-N]	Frecuencia de C-N a distancia topológica 3	0 ... a ... 44
12	SdssC	Suma de dssC E-estados	-5.256 ... a ... 4.722
13	HyWi_B(m)	Índice de Hyper – Wiener (función log) de la matriz de carga ponderada por masa.	1.544 ... a ... 5.701
14	LOC	Índice céntrico de Lopping	0 ... a ... 4.491
15	SM6_L	Momento espectral de orden 06 de la matriz de Laplace	4.174 ... a ... 12.609
16	F03[C-O]	Frecuencia de C-O a distancia topológica 3	0 ... a ... 40
17	Me	Media atómica de la electronegatividad de Sanderson (escalada en átomo de carbono)	0.957... a ... 1.311
18	Mi	Media de la ionización potencial de primer orden (escalada en átomo de carbono)	1.022 ... a ... 1.377
19	nN-N	Número de N Hidracinas	0 ... a ... 2
20	nArNO2	Número de nitro grupos (aromático)	0 ... a ... 3

[10] Debido a que no existieron valores nulos; no fue necesaria la realización de la Limpieza.

21	nCRX3	Número de CRX3	0 ... a ... 3
22	SpPosA_B(p)	Suma normalizada espectral positiva de la matriz de carga ponderada por polarizabilidad	0.863 ... a ... 1.641
23	nCIR	Número de circuitos	0 ... a ... 147
24	B01[C-Br]	Presencia/ausencia de C-Br a distancia topológica de 1	0 ... a ... 1
25	B03[C-Cl]	Presencia/ausencia de C-Cl a distancia topológica de 3	0 ... a ... 1
26	N-073	Ar_2NH/ Ar_3N/ Ar_2N-Al/ R..N..R	0 ... a ... 3
27	SpMax_A	Valor propio (eigenvalue) líder de la matriz adyacente (índice Lovasz – Pelikan)	1... a ... 2.859
28	Psi_i_1d	Índice de estado intrínseco de pseudoconectividad – tipo 1d	-1.099 ... a ... 1.073
29	B04[C-Br]	Presencia/ausencia de C-Br a distancia topológica de 4	0 ... a ... 1
30	SdO	Suma de dO E – estados	0 ... a ... 71.167
31	TI2_L	Segundo índice de Mohar de la matriz de Laplace	0.444 ... a ... 17.537
32	nCrt:	Número de anillos terciarios C(sp^3)	0 ... a ... 8
33	C-026	R—CX—R	0 ... a ... 12
34	F02[C-N]	Frecuencia de C-N a distancia topológica 2	0 ... a ... 18
35	nHDon	Número de átomos donadores para enlaces de Hidrogeno (N y O)	0 ... a ... 7
36	SpMax_B(m)	Valor propio (eigenvalue) líder de la matriz de carga ponderada por masa.	2.267... a ... 10.695
37	Psi_i_A	Índice de estado intrínseco de pseudoconectividad tipo S promedio	1.467 ... a ... 5.825
38	nN	Número de átomos de nitrógeno	0 ... a ... 8
39	SM6_B(m)	Momento espectral de orden 6 de la matriz de carga ponderada por masa	4.917... a ... 14.70
40	nArCOOR	Número de ésteres (aromático)	0 ... a ... 4
41	nX	Número de átomos de halógeno	0 ... a ... 27
42	exp_class[11]	Variable predictora de los dos estados de biodegradación de las moléculas: - Listo para la biodegradación (RB) - No listo para la biodegradación (NRB)	1 … a … 2 **Valor 1:** No listo para la biodegradación **Valor 2:** Listo para la biodegradación

Tabla N° 4: Diccionario de Datos

[11] En orden de poder trabajar con valores numéricos; se realizó el mapeo de la variable predictiva exp_class a una clase numérica

4.2. División del Data set:

Usualmente se suelen dividir los datos en:

- **Train Set:** Estos datos serán usados para entrenar el algoritmo; de estos datos será que aprenda.
- **Cross Validation Set:** Usado para realizar validaciones de los algoritmos; las cuales implican: selección optima de parámetros.
- **Test Set:** Usado para medir la performance del algoritmo; el Test set se usa para mostrar la verdadera exactitud del algoritmo.

PARA LA EJECUCIÓN SIN PCA		
Train Set	***Cross Validation Set***	***Test Set***
Se tomó el 60% de los datos totales.	Se tomó el 20% de los datos restantes	Se tomó el 20% de los datos restantes
PARA LA EJECUCIÓN CON PCA		
Train Set	***Cross Validation Set***	***Test Set***
Se tomó el 60% de los datos totales.	Se tomó el 20% de los datos restantes	Se tomó el 20% de los datos restantes

Tabla N° 5: Esquema de división del data set

Cabe resaltar que durante la división del data set; también se re – distribuyo las observaciones de forma aleatoria; con el fin de romper cualquier simetría que hubiera en los datos.

4.3. Entrenamiento de Algoritmos sin aplicación de PCA:

En esta sección comenzaremos con el entrenamiento de los algoritmos Support Vector Machine (SVM) y Redes Neuronales Multicapa (MLNN); para lo cual usaremos los datos en su estado RAW; esto es sin ningún tipo de alteración; para más adelante poder contrastar con un data set aplicando PCA.

4.3.1. Aplicación de Support Vector Machine sin PCA:

Para la implementación de SVM, usaremos la librería ya desarrollada y ampliamente usada, llamada SVMLIB (Chang & Lin, 2011); la cual implementa todos los procedimientos para el algoritmo Support Vector Machine descritos en

el fundamento teórico. A través de las secciones siguientes mostraremos cuales fueron las fases realizadas para el entrenamiento del algoritmo.

4.3.1.1. Selección de parámetros C, γ:

La librería SVMLIB (Chang & Lin, 2011) presenta opciones de modelos como: regresión, clasificación binaria, multi – clasificación, etc., para esta sección nos vamos a centrar en obtener los parámetros adecuados para el costo definido por C y para γ.

Primeramente el costo definido por C representa que tanto se va a ampliar o reducir el margen definido por ξ; en otras palabras este parámetro nos dará una penalización para los márgenes.

Por otro lado el parámetro γ corresponde a la función kernel gaussiano definida en Eq. (*23*); como vimos en el fundamento teórico, la función kernel se encarga del mapeo de características a un espacio vectorial; cabe destacar que de acuerdo a la función kernel que elijamos; tendremos que buscar un parámetro adecuado para dicha función.

La forma de búsqueda que vamos a realizar para encontrar estos parámetros consiste en la realización de una validación cruzada (cross validation); para lo cual vamos a usar dos conjuntos de datos definidos en la Tabla N° 5 (ejecución sin PCA); estos conjuntos serán: los datos de entrenamiento (60%) y los datos de validación cruzada (20%); para comenzar a realizar nuestra validación; vamos a definir un espacio de búsqueda de los parámetros C, γ, para lo cual tomaremos como referencia (Chih-Wei Hsu, Chih-Chung Chang, 2008):

Para γ:

$$\gamma = [2^{-15}, 2^{-13}, \dots, 2^{3}]$$

Para C:

$$C = [2^{-5}, 2^{-3}, \dots, 2^{15}]$$

Una vez realizada las definiciones para C, γ, es necesario aclarar que el espacio de búsqueda consistirá de los exponentes tanto para C como γ. Lo siguiente que vamos a realizar será entrenar el algoritmo varias

veces (hasta acabar el espacio de búsqueda para C, γ) usando la configuración de parámetros antes mostrada; la idea aquí es encontrar un valor tanto para C, γ tal que se obtenga el menor error cuadrático medio definido por la Eq. (*42*).

Una vez realizada la validación cruzada (cross validation); obtenemos los valores para el parámetro C ; de la Figura N° 14 se obtiene que el valor más bajo es cuando el exponente de C es igual a 1; esto es: $C = 2^1 = 2$

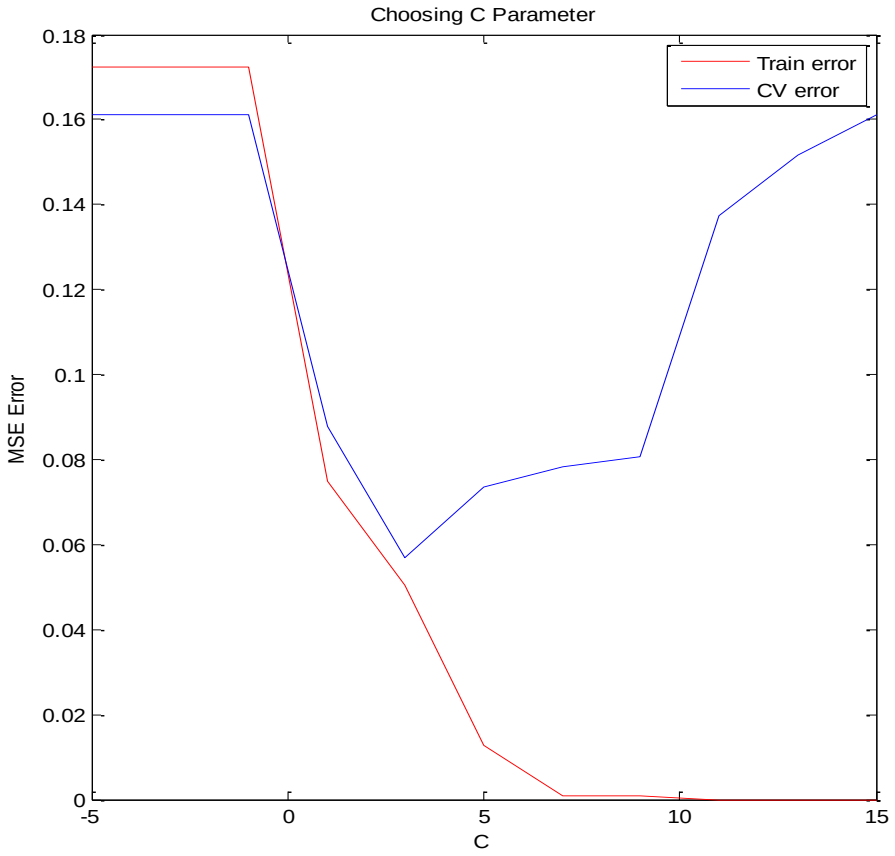


Figura N° 14: Selección del parámetro costo (C)

A su vez para el parámetro γ obtenemos que el valor para el exponente según la Figura N° 15 es -9; esto significa que $\gamma = 2^{-9} = 0.001953$

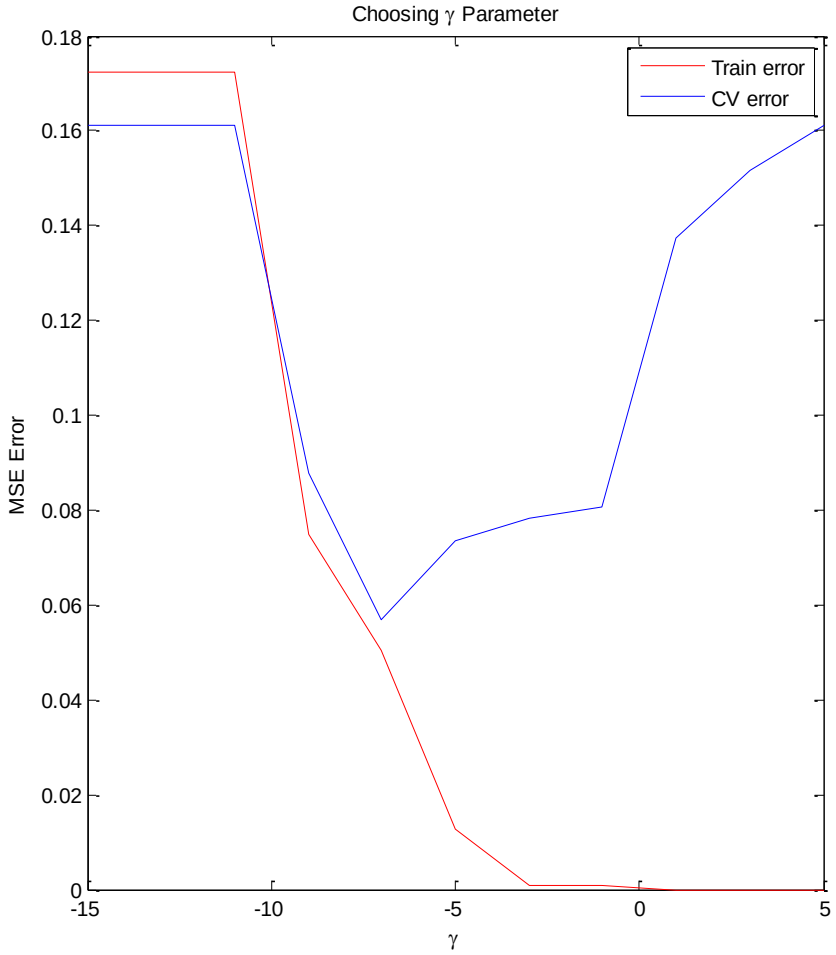


Figura N° 15: Selección del parámetro (y) del kernel

4.3.1.2. Entrenamiento del algoritmo SVM:

Una vez obtenidos los parámetros C,γ; y definida la función kernel; podemos procedemos a entrenar el algoritmo usando la siguiente configuración:

N° de iteraciones	Costo C	Función Kernel	Parámetro γ	Tipo de Problema
Las iteraciones se manejaron de forma automática por la librería, con la condición de detenerse cuando no se pueda minimizar más el error	2	Para el kernel usaremos la función gaussiana definida por la Eq. (23)	0.001953	Multi – clasificación

Tabla N° 6: Configuración de parámetros de Entrenamiento de SVM sin PCA

Los procesos de esta fase pueden ser observados en el ***Anexo A***.

4.3.1.3. Resultados de SVM:

Después de entrenar el algoritmo con la selección de parámetros C,γ obtenemos la siguiente tabla de Exactitud de aprendizaje:

RESULTADOS DEL ALGORITMO SUPPORT VECTOR MACHINE		
Exactitud en Datos de Entrenamiento (%)	*Exactitud en Datos de Validación Cruzada (%)*	*Exactitud en Datos de Prueba (%)*
85.1501	89.0995	84.8341

Tabla N° 7: Resultados del entrenamiento de SVM sin PCA

4.3.1.4. Curvas de Aprendizaje – SVM:

Una vez realizado el entrenamiento pasamos a visualizar las curvas de aprendizaje para el algoritmo Support Vector Machine (SVM); para lo cual vamos a usar las Eq. (*43*) y Eq. (*44*) definidas en el fundamento teórico; las cuales vamos a aplicar a los datos de entrenamiento y a los datos de validación cruzada respectivamente; a su vez definiremos las Eq. (*43*) y Eq. (*44*) en términos del número de datos que vayamos usando; obteniendo como resultado el siguiente gráfico:

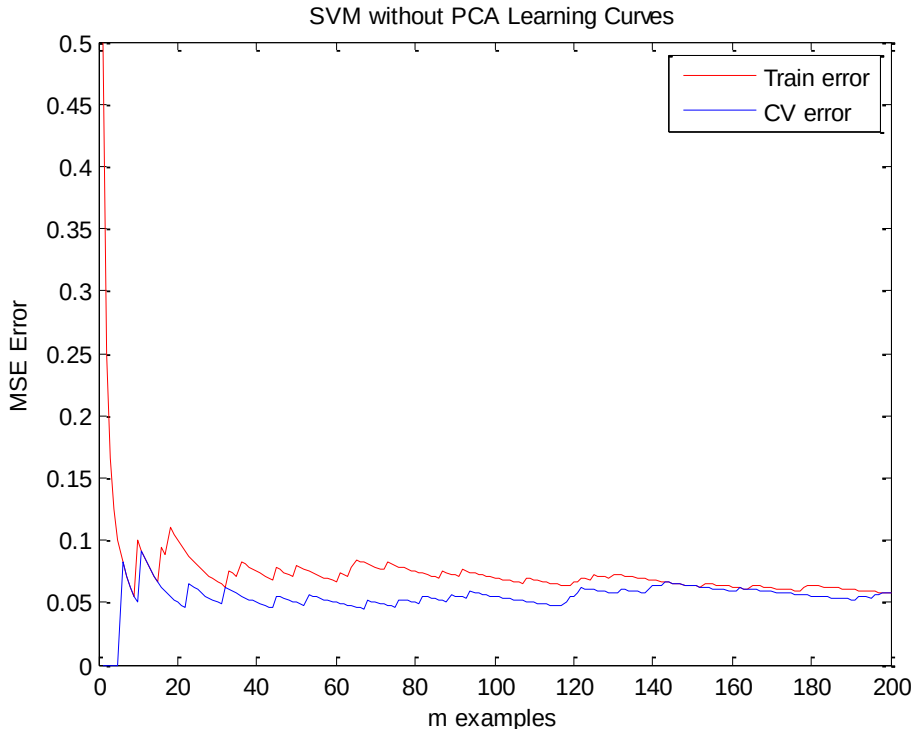


Figura N° 16: Curva de Aprendizaje para SVM sin PCA

De la Figura N° 16 podemos observar que nuestro algoritmo no presenta problemas de varianza ni de sesgo altos; ya que la curva del error de CV (cross validation) tiene a disminuir conforme se aumentan los datos.

Cabe resaltar que para la realización de las curvas de aprendizaje para SVM; estamos usando un total de 200 ejemplos; esto es $m = 200$ tanto para los datos de entrenamiento como para los de validación cruzada.

4.3.2. Aplicación de Redes Neuronales Multicapa sin PCA:

Una vez finalizado en entrenamiento de SVM sin el uso de PCA; pasamos a efectuar el entrenamiento de la Red Neuronal; para lo cual primero definiremos la arquitectura de la red neuronal.

4.3.2.1. Arquitectura de la Red Neuronal Multicapa sin usar PCA:

Para la estructura de la Red Neuronal sin usar PCA; se consideró: (i) una capa de entrada de 41 neuronas, (ii) una primera capa oculta compuesta por 3 neuronas, (iii) una segunda capa oculta compuesta por 5 neuronas y (iv) una capa de salida compuesta por 2 neuronas que corresponden a las dos clases predictoras.

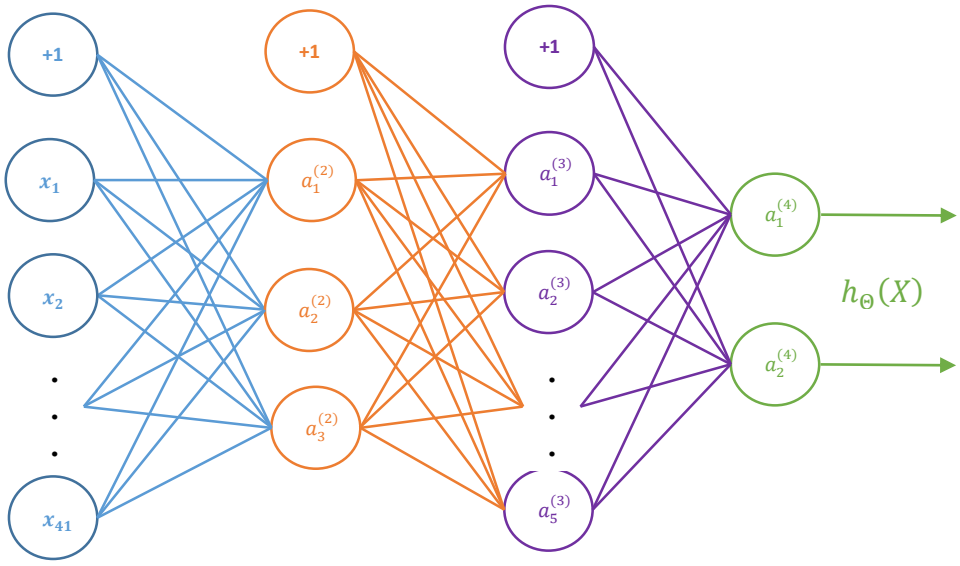


Figura N° 17: Estructura de la Red Neuronal Multicapa sin aplicar PCA

La razón de que el número de neuronas de la capa de entrada sea 41 es que, esta capa representa el número de atributos del data set, el cual sin procesar por PCA tiene ese número de entradas.

4.3.2.2. Iniciación aleatoria de los parámetros:

Una vez elegida la arquitectura de la red neuronal; vamos a pasar a iniciar la distribución aleatoria de los pesos; esto con el objetivo de poder romper la simetría; para lo cual usaremos la Eq. (*5*) definida en el fundamento teórico; con lo cual obtenemos los siguientes parámetros que se encuentran entre el rango de $[-\epsilon, \epsilon]$:

Para la capa de entrada y primera capa oculta:

$$\Theta^{(1)} \in \mathcal{R}^{3x42}$$

Para la primera capa oculta y segunda capa oculta:

$$\Theta^{(2)} \in \mathcal{R}^{5x4}$$

Para la segunda capa oculta y capa de salida:

$$\Theta^{(3)} \in \mathcal{R}^{2x6}$$

4.3.2.3. Selección del parámetro de regularización λ:

Para buscar el parámetro de regularización optimo; vamos a realizar una validación cruzada (cross validation); para lo cual vamos a usar los dos conjuntos de datos definidos en la Tabla N° 5 (ejecución sin PCA); estos conjuntos serán: los datos de entrenamiento (60%) y los datos de validación cruzada (40%); para comenzar a realizar nuestra validación; vamos a definir un espacio de búsqueda del parámetro λ:

$$\lambda = [0{,}0.24, \ldots, 10]$$

Una vez realizada la definición del espacio de búsqueda para λ, comenzaremos a entrenar el algoritmo varias veces (hasta acabar el espacio de búsqueda para λ) usando la configuración de parámetros y arquitectura de la red antes definidas; la idea aquí consiste encontrar un valor para λ tal que el error cuadrático medio definido en la Eq. (*42*) sea

mínimo. Una vez concluida la validación cruzada, obtenemos el siguiente gráfico:

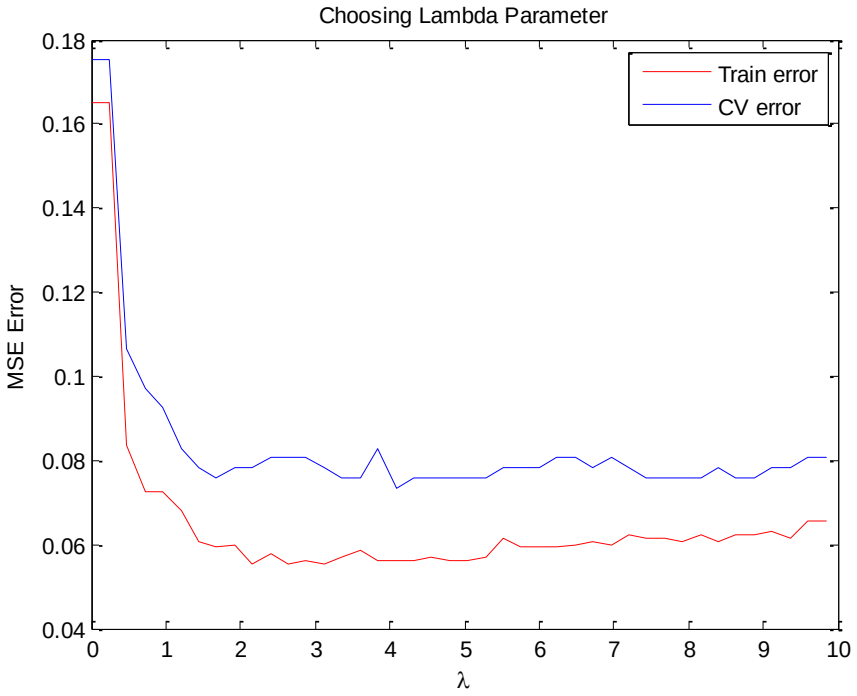


Figura N° 18: Selección del parámetro de regularización

De la Figura N° 18 obtenemos que el valor óptimo del parámetro de regularización es $\lambda = 4.08$.

4.3.2.4. Entrenamiento de la Red Neuronal:

Para iniciar el entrenamiento vamos a hacer uso de las ecuaciones descritas en el fundamento teórico para los algoritmos de gradiente descendente y propagación hacia atrás (1.5.10) y (1.5.9) respectivamente.

Para la implementación del algoritmo de gradiente descendente utilizaremos una librería desarrollada denominada fmincg; la cual va a recibir las derivadas proporcionadas por nuestro algoritmo de propagación hacia atrás para poder encontrar el valor mínimo local de la función; que en este caso representa el valor mínimo de los parámetros Θ con respecto a la función costo J. A su vez realizamos un bucle consistente de 150 iteraciones; el resultado de este proceso se muestra en el ***Anexo B***.

4.3.2.5. Curvas de Aprendizaje – MLNN:

Para el caso de la Red neuronal realizamos 2 curvas de aprendizaje; una de ellas usando el valor de la función costo $J(\theta)$; con el objetivo de ver el error con respecto al costo definido por la Eq. (*13*) del fundamento teórico; pero sin incluir el parámetro de regularización λ; ya que deseamos obtener el error únicamente en base a la función costo; esto nos da como resultado lo siguiente:

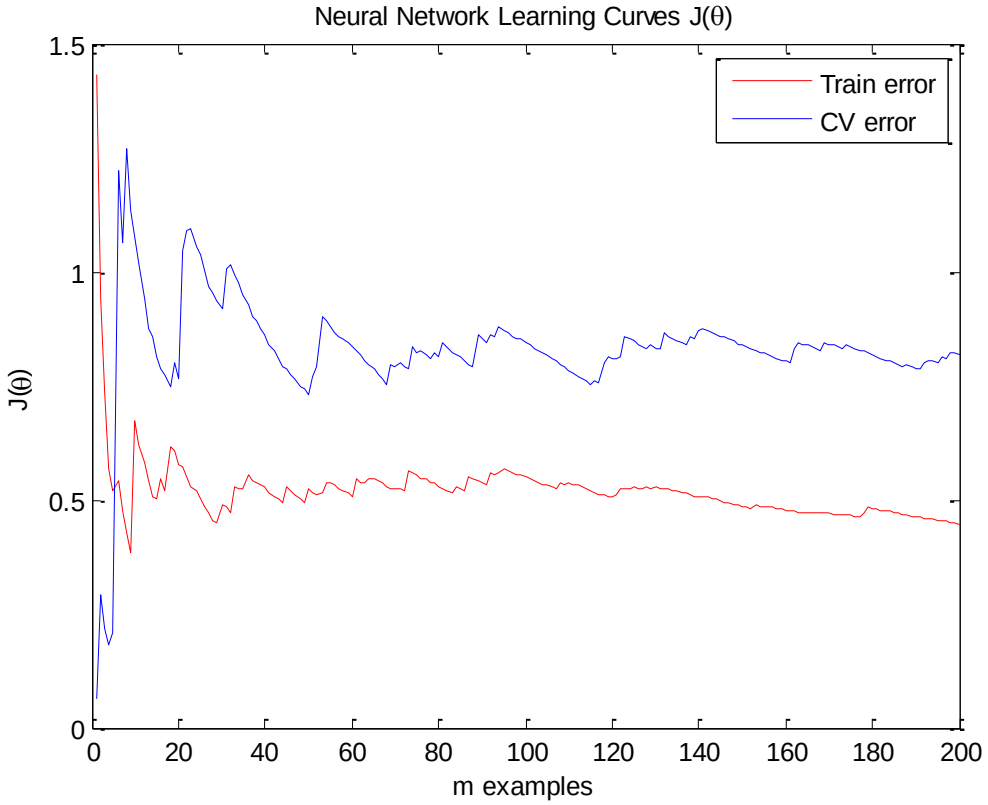


Figura N° 19: Curva de Aprendizaje para la Red en base a la función costo J

La segunda curva de aprendizaje corresponde a la aplicación de las Eq. (*43*) y Eq. (*44*) de las curvas de aprendizaje, definidas en el fundamento teórico; con lo cual obtenemos:

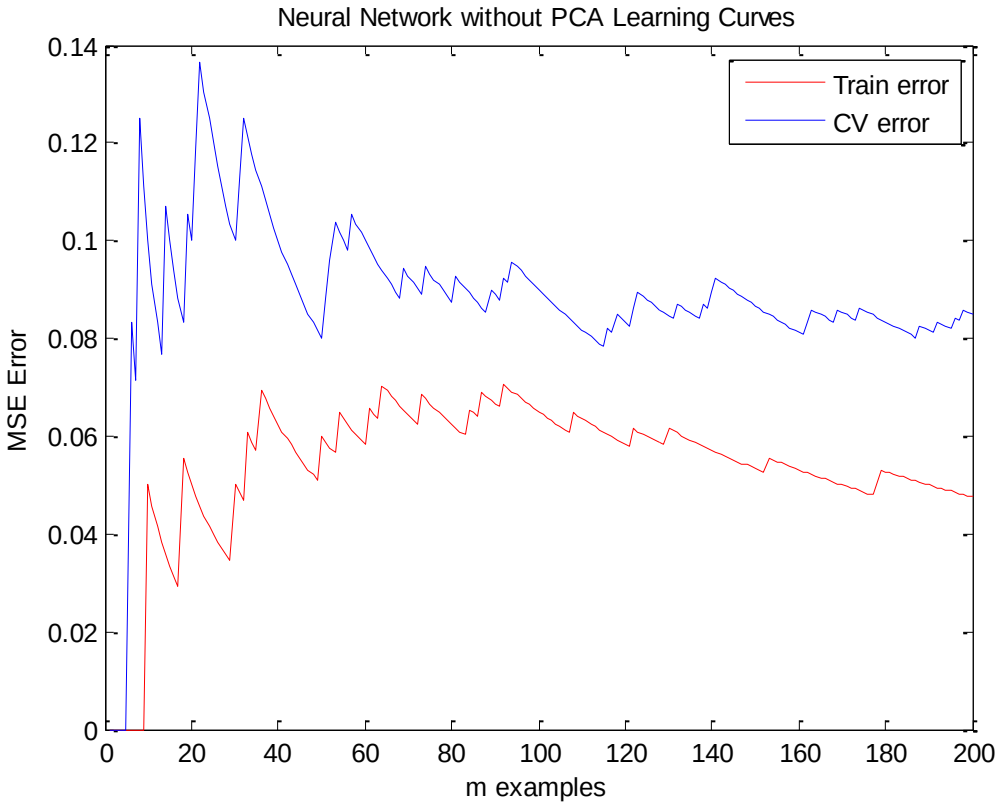


Figura N° 20: Curva de Aprendizaje para la Red Neuronal sin PCA

Observando tanto la primera curva como la segunda nos indican que nuestro algoritmo presenta una varianza alta (high variance). Más adelante veremos cómo se corrige este problema cuando realicemos la aplicación de PCA. Finalmente Cabe mencionar que para ambos casos se trabajó con un número de ejemplos de 200.

4.3.2.6. Resultados:

Una vez terminado todo el entrenamiento; obtenemos los siguientes resultados:

RESULTADOS DEL ALGORITMO DE REDES NEURONALES MULTICAPA		
Exactitud en Datos de Entrenamiento (%)	Exactitud en Datos de Validación Cruzada (%)	Exactitud en Datos de Prueba (%)
87.677725	83.886256	85.781991

Tabla N° 8: Resultados del entrenamiento de la Red sin aplicar PCA

A su vez obtenemos también la distribución final de nuestros parámetros; los cuales se muestran a continuación:

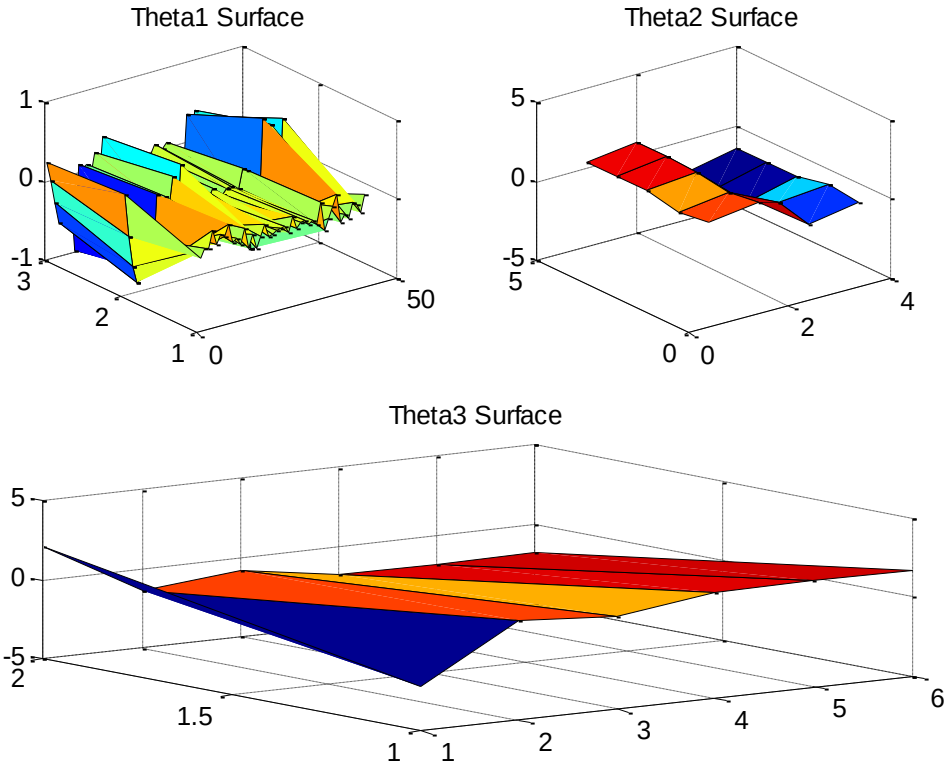


Figura N° 21: Distribución de los parámetros

Como podemos observar en la Figura N° 21 los parámetros de $\Theta^{(1)}$ muestran una distribución más compleja en relación a $\Theta^{(2)}$ y $\Theta^{(3)}$; esto se debe a que $\Theta^{(1)}$ recibe de forma directa los vectores $x^{(i)}$; que en este caso son 41, dicho de otro modo $\Theta^{(1)}$ mapea las características, haciendo de este modo que la distribución para $\Theta^{(2)}$ y $\Theta^{(3)}$ sea más lineal; finalmente $\Theta^{(3)}$ presenta una distribución lineal; esto debido a que $\Theta^{(3)}$ se encarga de la clasificación de las dos clases.

4.4. Entrenamiento de Algoritmos usando PCA:

En esta sección vamos a realizar el entrenamiento de los algoritmos de SVM y MLNN aplicando PCA sobre el data set.

4.4.1. Aplicación de Principal Component Analysis sobre el data set:

Comenzaremos con aplicar PCA sobre los datos de entrenamiento solamente para una vez obtenidas las dimensiones deseadas; podamos aplicar una transformación a los datos de validación cruzada y test, tomando como referencia los datos de entrenamiento que han sido alterados por PCA; ya que, de aplicar PCA a todos los datos unidos; estaríamos generando que los resultados finales del test estén en función de PCA y no del algoritmo en sí. Antes de comenzar con el proceso vamos a mostrar la distribución de los datos originales.

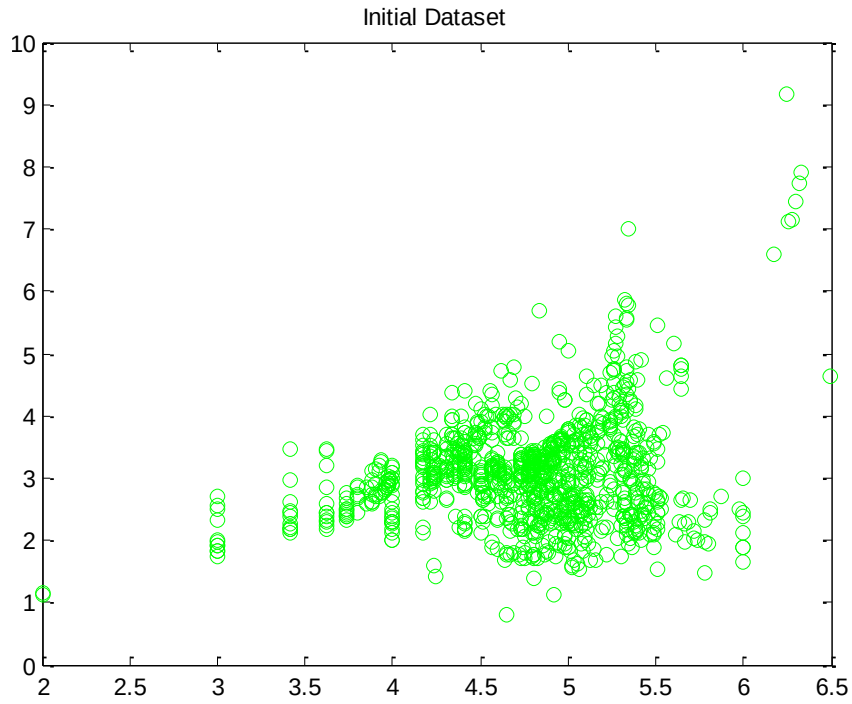


Figura N° 22: Distribución original de los datos del data set

La Figura N° 22 muestra los datos iniciales; veremos cómo al finalizar el proceso de PCA; esta distribución se verá ampliamente afectada.

Primeramente vamos a comenzar por el proceso de pre – procesamiento de los datos; para lo cual vamos a usar las formulas definidas en Eq. (*36*) y Eq. (*37*); una vez realizado este proceso vamos a computar la matriz de covarianza utilizando Eq. (*38*); una vez realizado este proceso procederemos con la computación de los vectores propios y matriz diagonal utilizando la Eq. (*35*) de SVD.

A continuación se muestra un resumen de los 41 componentes principales obtenidos; así como el porcentaje de los datos que retienen y el porcentaje de perdida; ambos en función de la cantidad de componentes.

Número de Componentes Principales	Porcentaje de Datos retenidos	Porcentaje de Datos Perdidos
Comp. 01	17.7608	0.82239
Comp. 02	30.7761	0.69224
Comp. 03	41.3764	0.58624
Comp. 04	49.2419	0.50758
Comp. 05	55.9185	0.44081
Comp. 06	61.8422	0.38158
Comp. 07	65.795	0.34205
Comp. 08	69.261	0.30739
Comp. 09	72.4558	0.27544
Comp. 010	75.3579	0.24642
Comp. 011	78.1539	0.21846
Comp. 012	80.7816	0.19218
Comp. 013	83.0354	0.16965
Comp. 014	85.2174	0.14783
Comp. 015	87.2371	0.12763
Comp. 016	88.9163	0.11084
Comp. 017	90.3578	0.096422
Comp. 018	91.4839	0.085161
Comp. 019	92.5518	0.074482
Comp. 020	93.552	0.06448
Comp. 021	94.3912	0.056088
Comp. 022	95.1455	0.048545
Comp. 023	95.7657	0.042343
Comp. 024	96.3505	0.036495
Comp. 025	96.8589	0.031411
Comp. 026	97.3401	0.026599
Comp. 027	97.772	0.02228
Comp. 028	98.1773	0.018227
Comp. 029	98.5223	0.014777
Comp. 030	98.7771	0.012229
Comp. 031	99.0229	0.0097707
Comp. 032	99.2479	0.0075209
Comp. 033	99.4182	0.0058184
Comp. 034	99.572	0.0042795
Comp. 035	99.7066	0.0029342
Comp. 036	99.8054	0.0019462
Comp. 037	99.8781	0.001219
Comp. 038	99.9271	0.00072867
Comp. 039	99.9715	0.00028521
Comp. 040	99.9926	7.4003e-05
Comp. 041	100	0

Tabla N° 9: Lista de Componentes Principales totales

Usando la información mostrada en la Tabla N° 9; es posible construir una representación gráfica para poder visualizar mejor la el porcentaje de datos retenidos por cada componente principal; con lo cual obtenemos:

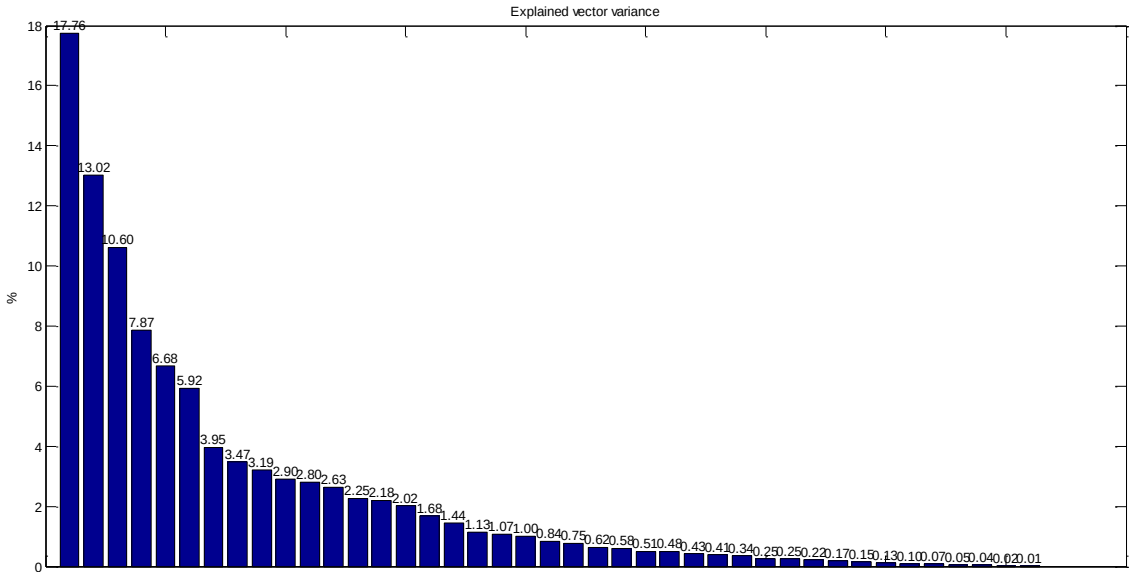


Figura N° 23: Porcentaje de Varianza retenida por componente

Ahora, para seleccionar coherentemente el número de dimensiones k vamos a usar las Eq. (*40*) y Eq. (*41*); junto con la información mostrada por la Figura N° 23; con lo que obtenemos la siguiente tabla de componentes principales:

Número de Componentes Principales	Porcentaje de Datos retenidos	Porcentaje de Datos Perdidos
Comp. 01	17.7608	0.82239
Comp. 02	30.7761	0.69224
Comp. 03	41.3764	0.58624
Comp. 04	49.2419	0.50758
Comp. 05	55.9185	0.44081
Comp. 06	61.8422	0.38158
Comp. 07	65.795	0.34205
Comp. 08	69.261	0.30739
Comp. 09	72.4558	0.27544
Comp. 010	75.3579	0.24642
Comp. 011	78.1539	0.21846
Comp. 012	80.7816	0.19218
Comp. 013	83.0354	0.16965
Comp. 014	85.2174	0.14783
Comp. 015	87.2371	0.12763
Comp. 016	88.9163	0.11084
Comp. 017	90.3578	0.096422
Comp. 018	91.4839	0.085161
Comp. 019	92.5518	0.074482

Comp. 020	93.552	0.06448
Comp. 021	94.3912	0.056088
Comp. 022	95.1455	0.048545
Comp. 023	95.7657	0.042343
Comp. 024	96.3505	0.036495
Comp. 025	96.8589	0.031411
Comp. 026	97.3401	0.026599
Comp. 027	97.772	0.02228
Comp. 028	98.1773	0.018227
Comp. 029	98.5223	0.014777
Comp. 030	98.7771	0.012229
Comp. 031	99.0229	0.0097707

Tabla N° 10: Lista de componentes principales seleccionados

Siendo de este modo 31 el número de dimensiones seleccionadas; ya que para k=31 se cumple para la Eq. (*40*) que el porcentaje de datos perdidos se encuentra entre: $0.0097707 \leq 0.01$. A su vez para la Eq. (*41*) que indica el porcentaje de datos retenidos se cumple para k=31 que: $99.0229 \geq 0.99$. De este modo se está asegurando que se retenga aproximadamente 99.0229 % de los datos originales con una pérdida del 0.0097707%. Una vez seleccionadas las dimensiones; obtenemos una nueva distribución de los datos:

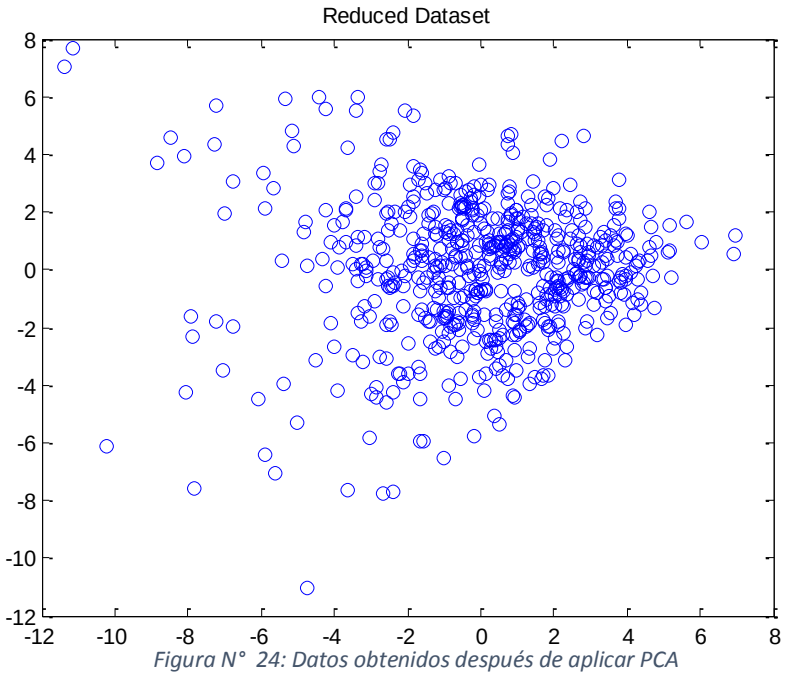


Figura N° 24: Datos obtenidos después de aplicar PCA

Estos datos representan los mismo datos que en la Figura N° 22; con la salvedad de que el número de dimensiones ahora es 31 y no 41. A su vez

también tenemos la distribución de los datos de acuerdo a sus 3 componentes principales; lo cual se visualiza en:

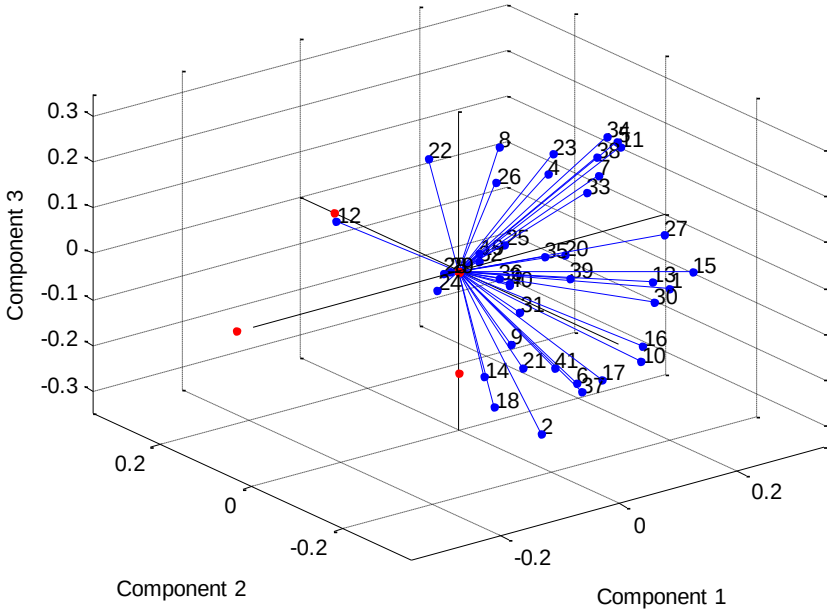


Figura N° 25: Distribución de los datos en 3 componentes

4.4.2. Support Vector Machine aplicando PCA:

Una vez finalizada la reducción de dimensiones vamos a implementar SVM usando como datos de entrenamiento, validación y pruebas los datos transformados por PCA en el apartado anterior.

4.4.2.1. Selección de parámetros C, γ:

El método de selección que vamos a usar consistirá en realizar una validación cruzada; usando para ellos los datos proporcionados por PCA (60% de entrenamiento y 40% de validación); nuestro objetivo en esta sección es encontrar los valores C, γ que minimicen el error cuadrático medio definido en Eq. (*42*).

Para ello definimos el espacio de búsqueda de los exponentes de C, γ de la siguiente manera (Chang & Lin, 2011):

Para γ:

$$\gamma = [2^{-15}, 2^{-13}, \dots, 2^{3}]$$

Para C:

$$C = [2^{-5}, 2^{-3}, \dots, 2^{15}]$$

Una vez realizada la validación cruzada (cross validation); obtenemos los valores para el parámetro C ; de la Figura N° 26 se obtiene que el valor más bajo es cuando el exponente de C es igual a 3; esto es: $C = 2^3 = 8$

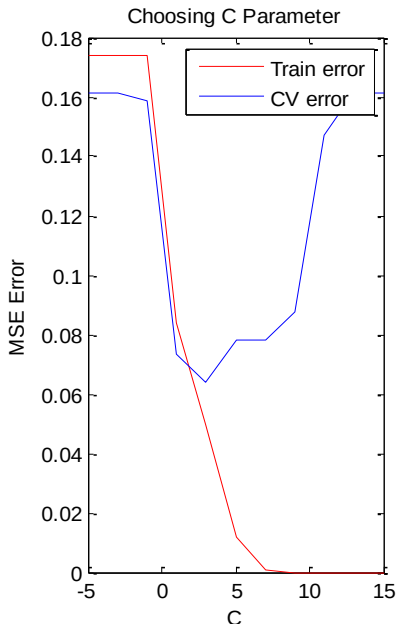


Figura N° 26: Selección del parámetro C usando PCA sobre los datos

A su vez para el parámetro γ obtenemos que el valor para el exponente según la Figura N° 27 es -7; esto significa que $\gamma = 2^{-7} = 0.007813$

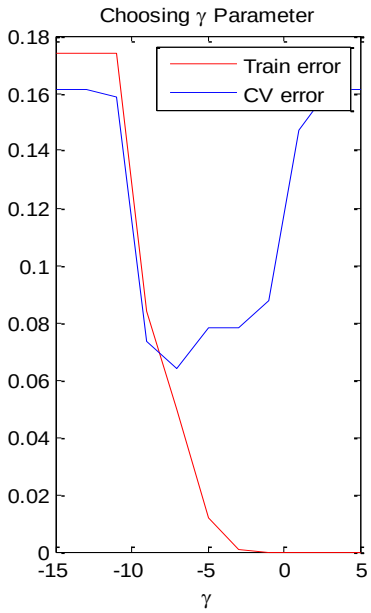


Figura N° 27: Selección del parámetro y usando PCA sobre los datos

4.4.2.2. Entrenamiento del algoritmo SVM usando PCA en los datos:

Una vez obtenidos los parámetros C,γ; y definida la función kernel; podemos procedemos a entrenar el algoritmo usando la siguiente configuración:

N° de iteraciones	Costo C	Función Kernel	Parámetro γ	Tipo de Problema
Las iteraciones se manejaron de forma automática por la librería, con la condición de detenerse cuando no se pueda minimizar más el error	8	Para el kernel usaremos la función gaussiana definida por la Eq. (23)	0.007813	Multi – clasificación

Tabla N° 11: Configuración de SVM aplicando PCA a los datos

El resultado completo del proceso de entrenamiento se muestra en el ***Anexo C***.

4.4.2.3. Resultados de SVM:

Después de entrenar el algoritmo con la selección de parámetros C,γ obtenemos la siguiente tabla de Exactitud de aprendizaje:

RESULTADOS DEL ALGORITMO SUPPORT VECTOR MACHINE USANDO DATA PROCESADA POR PCA		
Exactitud en Datos de Entrenamiento (%)	*Exactitud en Datos de Validación Cruzada (%)*	*Exactitud en Datos de Prueba (%)*
90.0474	91.9431	90.9953

Tabla N° 12: Resultados del entrenamiento de SVM aplicando PCA

4.4.2.4. Curvas de Aprendizaje – SVM:

Finalmente observarnos que nuestra curva de aprendizaje no presenta ningún problema de sesgo o varianza y a su vez; el error obtenido en la validación cruzada ha disminuido con respecto a la curva mostrada sin usar PCA; ya que el error ahora se encuentras entre $0.03 - 0.05$

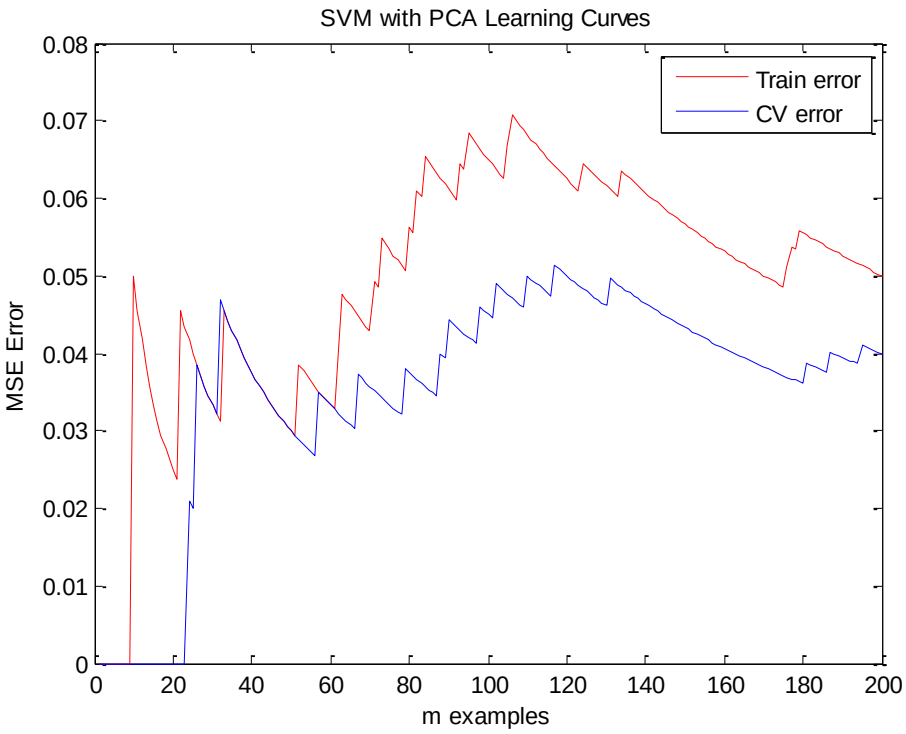


Figura N° 28: Curva de Aprendizaje de SVM aplicando PCA a los datos

Cabe mencionar que igual para la curva sin usar PCA; se está tomando un total de 200 ejemplos.

4.4.3. Redes Neuronales Multicapa aplicando PCA:

4.4.3.1. Estructura de la Red Neuronal Multicapa usando PCA:

Para la estructura de la Red Neuronal con PCA; se consideró: (i) una capa de entrada de 31 neuronas, (ii) una primera capa oculta compuesta por 6 neuronas,

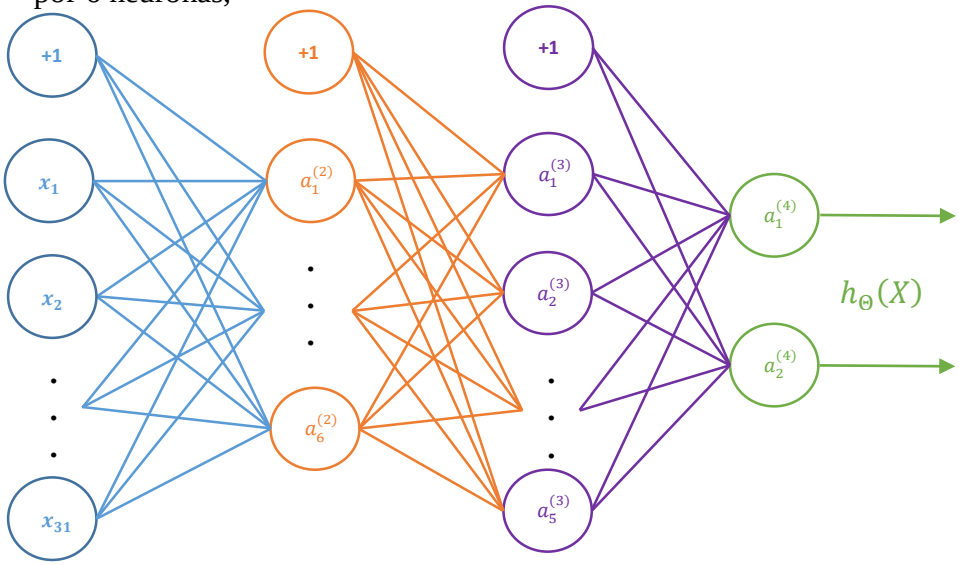


Figura N° 29: Estructura de la Red Neuronal Multicapa aplicando PCA

(iii) una segunda capa oculta compuesta por 5 neuronas y (iv) una capa de salida compuesta por 2 neuronas que corresponden a las clases predictoras.

4.4.3.2. Iniciación aleatoria de los parámetros:

Una vez elegida la arquitectura de la red neuronal; vamos a pasar a iniciar la distribución aleatoria de los pesos; esto con el objetivo de poder romper la simetría y así poder definir un espacio de búsqueda con mejores condiciones; para lo cual usaremos la Eq. (*5*) definida en el fundamento teórico; con lo cual obtenemos los siguientes parámetros que se encuentran entre el rango de $[-\epsilon, \epsilon]$:

Para la capa de entrada y primera capa oculta:

$$\Theta^{(1)} \in \mathcal{R}^{6x32}$$

Para la primera capa oculta y segunda capa oculta:

$$\Theta^{(2)} \in \mathcal{R}^{5x7}$$

Para la segunda capa oculta y capa de salida:

$$\Theta^{(3)} \in \mathcal{R}^{2x6}$$

4.4.3.3. Selección del parámetro de regularización λ:

Para buscar el parámetro de regularización optimo; vamos a realizar el mismo procedimiento que usamos para el caso de la Red Neuronal sin la aplicación de PCA; el cual consiste en una validación cruzada (cross validation); para lo cual vamos a usar los dos conjuntos de datos definidos en la Tabla N° 5 (ejecución con PCA); estos conjuntos serán: los datos de entrenamiento; los cuales representan el 60% de los datos; y los datos de validación cruzada que representan el 40%. Para comenzar a realizar nuestra validación; vamos a definir un espacio de búsqueda del parámetro λ; el cual consistirá de:

$$\lambda = [0,0.24, \dots, 10]$$

Una vez realizada la definición del espacio de búsqueda para λ, comenzaremos a entrenar el algoritmo varias veces (hasta acabar el espacio de búsqueda para λ) usando la configuración de parámetros y arquitectura de la red antes definidas en **(4.4.3.1)**; la idea consiste en encontrar un valor para λ tal que el error cuadrático medio definido en la Eq. (*42*) sea mínimo. Una vez concluida la validación cruzada, obtenemos el siguiente gráfico:

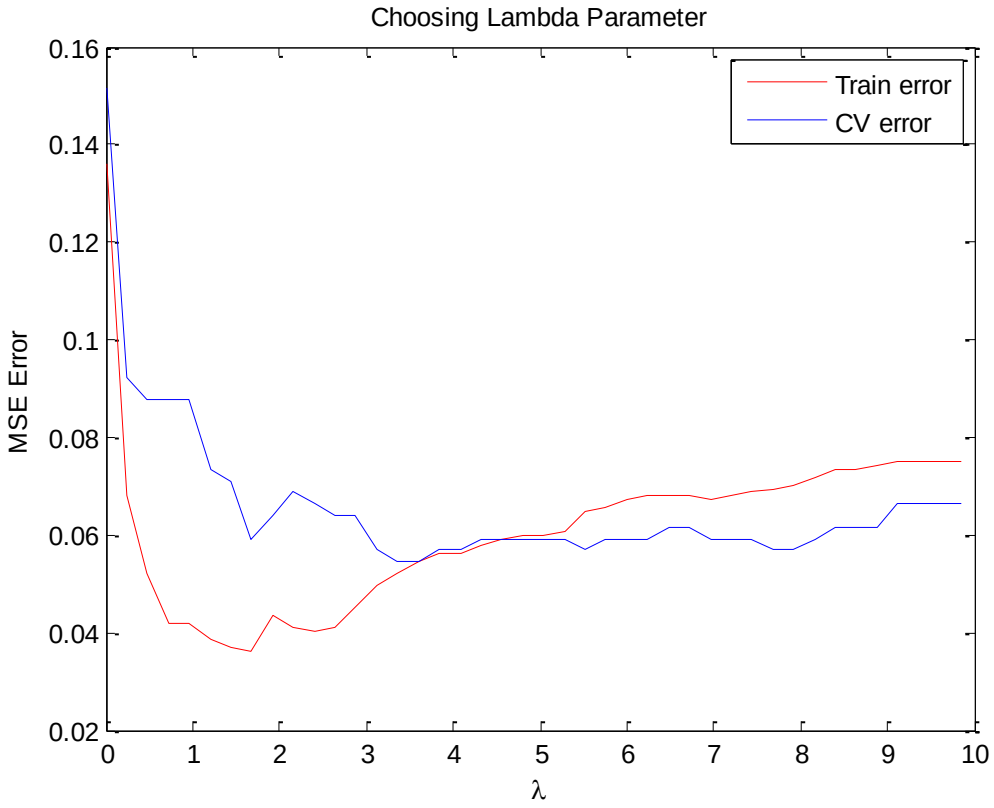


Figura N° 30: Selección del parámetro de regularización de la Red con PCA

De la Figura N° 30 obtenemos que el valor óptimo del parámetro de regularización es $\lambda = 3.36$.

4.4.3.4. Entrenamiento de la Red Neuronal:

Para iniciar el entrenamiento vamos a hacer uso de las ecuaciones descritas en el fundamento teórico para los algoritmos de gradiente descendente y propagación hacia atrás (1.5.10) y (1.5.9) respectivamente.

Para la implementación del algoritmo de gradiente descendente utilizaremos una librería desarrollada denominada *fmincg*; la cual va a recibir las derivadas proporcionadas por nuestro algoritmo de propagación hacia atrás para poder encontrar el valor mínimo local de la función; que en este caso, representa el valor mínimo de los parámetros Θ con respecto a la función costo J. A su vez realizamos un bucle consistente de 150 iteraciones; el resultado de este proceso se muestra en el ***Anexo D***.

4.4.3.5. Curvas de Aprendizaje – MLNN usando PCA:

Para el caso de la Red neuronal entrenada con datos procesados pro PCA; vamos a realizamos 2 curvas de aprendizaje; en la primera de ellas usaremos el valor de la función costo $J(\theta)$ definida en la *Eq. (13)* como índice de error; sin embargo, no se incluirá el parámetro de regularización λ; ya que deseamos obtener el error únicamente en base a la función costo; esto nos da como resultado la siguiente curva:

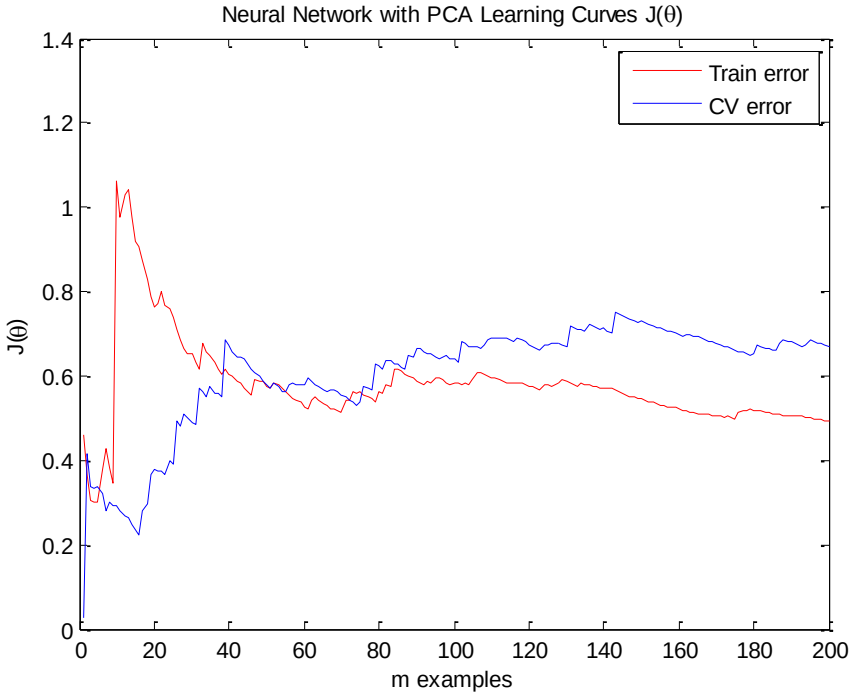


Figura N° 31: Curva de Aprendizaje $J(\theta)$ para MLNN aplicando PCA

La segunda curva de aprendizaje corresponde a la aplicación de las *Eq. (43)* y *Eq. (44)*, definidas en el fundamento teórico; con lo cual obtenemos:

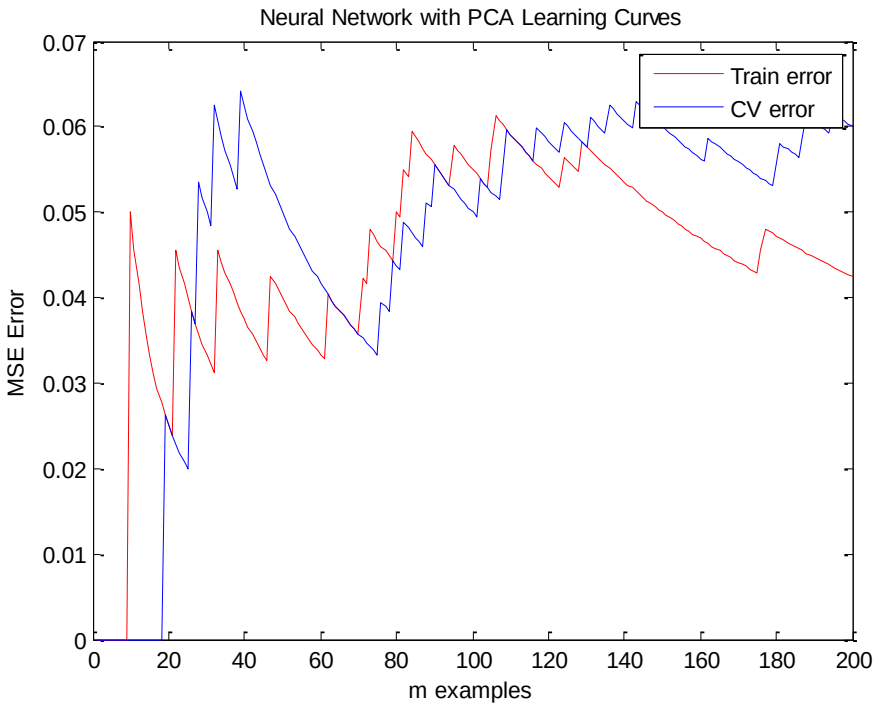


Figura N° 32: Curva de Aprendizaje MSE-Error para MLNN con PCA

Observando tanto la primera curva como la segunda nos indican que nuestro algoritmo presenta un leve problema de sesgo alto (high bias); sin embargo tanto en la Figura N° 31 y Figura N° 32 se puede apreciar que el margen del sesgo está comprendido entre los puntos [0.6 – 0.7] y [0.05 – 0.06] respectivamente; lo cual indica un sesgo mínimo; a su vez, se puede observar en la Figura N° 32 como el sesgo tiende a disminuir a medida que se agreguen más datos de ejemplo; por lo que el algoritmo presenta una mejora con respecto a las curvas definidas sin aplicar PCA al data set; ya que en las curvas anteriores el algoritmo padecía de una varianza alta. Finalmente Cabe mencionar que para ambos casos se trabajó con un número de 200 ejemplos.

4.4.3.6. Resultados:

Una vez terminado todo el entrenamiento; obtenemos los siguientes resultados:

RESULTADOS DEL ALGORITMO DE REDES NEURONALES MULTICAPA		
Exactitud en Datos de Entrenamiento (%)	Exactitud en Datos de Validación Cruzada (%)	Exactitud en Datos de Prueba (%)
90.363349	88.151659	89.099526

Tabla N° 13: Resultados del entrenamiento de MLNN aplicando PCA

A su vez obtenemos también la distribución final de nuestros parámetros; los cuales se muestran a continuación

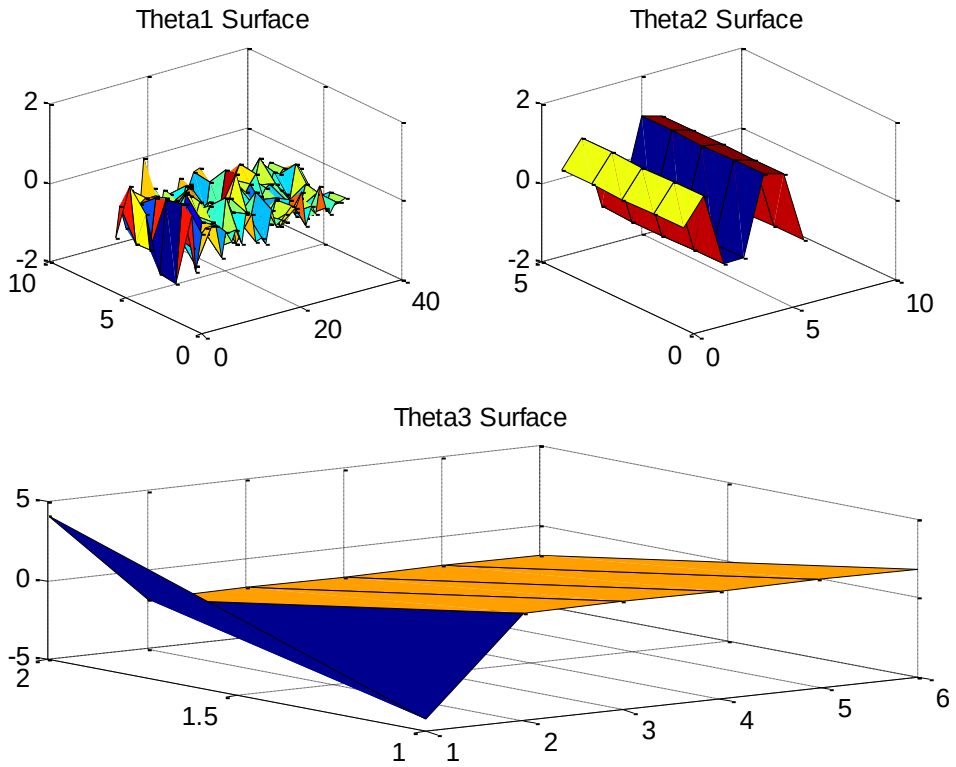


Figura N° 33: Distribución final de los parámetros después del entrenamiento

Como podemos observar en la Figura N° 33 los parámetros de $\Theta^{(1)}$ muestran una distribución aún más compleja comparada con el parámetro $\Theta'^{(1)}$ obtenido después de entrenar la red sin la aplicación de PCA.

En relación a $\Theta^{(2)}$ y $\Theta^{(3)}$; se puede observar un comportamiento lineal; esto se debe a que $\Theta^{(1)}$ mapea las características, haciendo de este modo que la distribución para $\Theta^{(2)}$ y $\Theta^{(3)}$ sea lineal; finalmente $\Theta^{(3)}$ presenta una distribución más lineal con respecto a su contraparte sin PCA $\Theta'^{(3)}$.

Capítulo V: Discusión

En este Capítulo se discuten y/o analizan los resultados obtenidos en el Capítulo IV, asimismo se realiza el análisis de los objetivos específicos acorde con los resultados obtenidos en el desarrollo. Finalmente se lleva a cabo la contrastación formal de la hipótesis; para lo cual se hace uso de la sección 5 de la metodología definida en el Capítulo III así como de las métricas de Machine Learning.

5.1. Discusión de Objetivos:

5.1.1. *Obtener y depurar el data set de qsar biodegradation del repositorio de datos de machine learning (uc irvine machine learning repository) para su posterior utilización*:

Los resultados obtenidos de este objetivo se traducen en la obtención en formato *csv* de todo el data set; incluyendo los valores predictivos y; debido a que éstos se encontraban en un formato *csv*; tuvieron que ser cambiados a formato *mat* en matlab para un manejo más práctico (el resultado completo se muestra en el Capítulo IV: DataSet).

5.1.2. *Elaborar la descripción general, resumen de características y el diccionario de datos sobre el data set QSAR biodegradation con el objetivo de tener un mejor entendimiento de los atributos y/o características del mismo:*

Este objetivo dio como resultado información acerca de las características del data set, como el número de atributos y observaciones; también se definió el diccionario de datos; en el cual se pudo notar que el vector y estaba en un formato cualitativo; por lo que se tuvo que mapear a un formato cuantitativo para que de esta manera pudiera ser procesado por los algoritmos (el resultado completo se muestra en el Capítulo IV: apartados: 4.1.1 – 4.1.3).

5.1.3. *Implementar Principal Component Analysis (PCA) para pre – procesar el data set QSAR biodegradation usando MATLAB versión 2014a:*

Este objetivo implementó toda la teoría descrita en el apartado del CAPÍTULO II acerca de PCA; los resultados obtenidos consistieron en una nueva versión del dataset; aquí, podemos apreciar que al aplicar PCA al dataset se redujo el

número de dimensiones de un total de 41 a 31; logrando retener un 99.0229% de los datos; y perdiendo un aproximado de 0.0097707%; éstos valores están acorde con los márgenes descritos en las Eq. (*41*) y Eq. (*40*) respectivamente; por lo que se pudo pasar a la siguiente fase sin problemas. Finalmente se obtuvieron dos datasets; uno con las dimensiones reducidas a 31 (con la aplicación de PCA) y otro con sus 41 dimensiones originales (sin aplicar PCA); estos datasets fueron usados por los algoritmos SVM y MLNN a través de los objetivos (5.1.4), (5.1.5), (5.1.6) y (5.1.7) respectivamente (el desarrollo y/o resultados son detallados en el Capítulo IV: Aplicación de Principal Component Analysis sobre el Data set).

5.1.4. *Desarrollar, entrenar, y/o implementar una red neuronal multicapa (MLNN) usando la data generada por PCA; usando MATLAB versión 2014a:*

En este objetivo se llevó a cabo todo el proceso de implementación de la red neuronal; incluyendo aspectos de entrenamiento y/o validación. Al analizar los resultados podemos notar que el número de capas de entrada se redujo al número de dimensiones del data set (procesado con PCA); esto sucedió debido a que las capas de entrada siempre se alinean con el número de atributos del data set; con esto se logró reducir en buena medida la complejidad del data set; ya que, sólo se conservaron los atributos que guardaban mayor varianza. Finalmente; cuando el algoritmo estuvo implementado, se observó un costo de *7.434114e-01*; el cual es menor en comparación de *8.075014e-01* obtenido por el mismo algoritmo entrenado por el data set original (sin aplicar PCA). La implementación y/o resultados obtenidos son detallados en el Capítulo IV: Redes Neuronales Multicapa Aplicando PCA.

5.1.5. *Entrenar, y/o implementar el algoritmo Support Vector Machine (SVM) en MATLAB versión 2014a, usando la librería SVMLIB (Chang & Lin, 2011) usando la data generada por PCA; usando MATLAB versión 2014a:*

En este objetivo se llevó a cabo todo el proceso de implementación del algoritmo SVM; incluyendo aspectos de entrenamiento y/o validación. El desarrollo del algoritmo fue más sencillo en relación con las redes neuronales;

ya que usamos la librería SVMLIB (Chang & Lin, 2011); la cual acelera bastante el tiempo de implementación. Por otro lado, cuando finalizo la implementación del algoritmo SVM; se obtuvo una exactitud de 90.9953%; la cual es más alta en comparación a su contraparte sin PCA: 84.8341%; no obstante, aun habiendo obtenido esos resultados, no estábamos en condición de afirmar que el algoritmo SVM entrenado aplicando PCA al data set era mejor; ya que, podría darse el caso de que el algoritmo reconociera a todas sus clases como positivas o negativas; con lo que obtendría una exactitud del 100%; por esta razón, no podemos confiarnos del todo de los resultados obtenidos por la exactitud; más adelante en el apartado ***5.4. Contrastación de la hipótesis*** analizaremos en más detalle las demás métricas que usaremos para poder confirmar o descartar esta asunción. Para un mayor detalle de la implementación y/o resultados obtenidos, consultar el Capítulo IV: Support Vector Machine Aplicando PCA.

5.1.6. *Desarrollar, entrenar, y/o implementar una red neuronal multicapa (MLNN) usando la data original sin aplicar PCA; usando MATLAB versión 2014a:*

En este objetivo se llevó a cabo todo el proceso de implementación de la red neuronal; incluyendo aspectos de entrenamiento y/o validación. Al analizar los resultados podemos notar que el número de capas de entrada es más alto que su contraparte con PCA (41 capas de entrada); esto sucedió debido a la relación que existe entre las capas de entrada y los atributos del data set; en esta situación la complejidad del data set se mantuvo; y no era posible discernir que atributos presentaban mayor varianza; sin embargo éste es un escenario común cuando se trabajan con algoritmos de aprendizaje. Finalmente; cuando el algoritmo MLNN estuvo implementado, se observó un costo de *8.075014e-01*; el cual es más alto que se contraparte con PCA: *7.434114e-01*; esto podría ser un indicador de que el algoritmo MLNN sin PCA, presenta algunas deficiencias; sin embargo en este punto es sólo una asunción que será validada o refutada el apartado ***5.4. Contrastación de la hipótesis***. La implementación y/o resultados son mostrados en el Capítulo IV: Aplicación de Redes Neuronales Multicapa sin PCA.

5.1.7. *Entrenar, y/o implementar el algoritmo Support Vector Machine (SVM) en MATLAB versión 2014a, usando la librería SVMLIB (Chang & Lin, 2011) usando la data original sin aplicar PCA; usando MATLAB versión 2014a:*

En este objetivo se llevó a cabo todo el proceso de implementación del algoritmo SVM usando la librería SVMLIB (Chang & Lin, 2011); incluyendo aspectos de entrenamiento y/o validación. Al igual que se contraparte con PCA; el desarrollo del algoritmo SVM fue más sencillo y rápido en relación con las redes neuronales. Cuando finalizo la implementación; se obtuvo una exactitud de 84.8341%; la cual es relativamente más baja en comparación a su contraparte sin PCA: 90.9953%; no obstante, al igual que pasaba con el algoritmo SVM aplicando PCA; no nos podíamos guiar sólo por la exactitud; ya que podría haberse dado el caso de que; en efecto, este algoritmo tuviera un índice de falsos positivos más bajo que el algoritmo SVM con PCA; lo cual le daría una performance de aprendizaje más alta. Más adelante en el apartado ***5.4. Contrastación de la hipótesis*** mostraremos en más detalle el análisis de estos resultados. Para un mayor detalle de la implementación y/o resultados obtenidos de este objetivo, consultar el Capítulo IV: Aplicación de Support Vector Machine sin PCA.

5.1.8. *Realizar las comparaciones de performance de aprendizaje usando las métricas de Machine Learning (matriz de confusión, presicion, recall, F – Micro averaged Score y curvas ROC) en los algoritmos SVM y NN con y sin PCA:*

Los resultados obtenidos en este objetivo muestran un incremento significativo en la performance de aprendizaje de los algoritmos SVM y MLNN al aplicar PCA al data set. Este objetivo será discutido con mayor detalle en el apartado ***5.4. Contrastación de la hipótesis***; dentro de este mismo Capítulo.

5.2. Diseño de Contrastacion de Hipótesis:

En esta sección vamos a formular el diseño de contrastación de la hipótesis; para lo cual vamos a usar las métricas más usadas en el área de Machine Learning.

Dada nuestra hipótesis como:

"La aplicación de Principal Component ANALYSIS (PCA) mejorará la Performance de aprendizaje de los Algoritmos Support Vector Machine (SVM) y Red Neuronal Multicapa (MLNN)"

Y el enunciado del problema definido:

¿Cómo mejorar la performance de aprendizaje de los Algoritmos Support Vector Machine (SVM) y Red Neuronal Multicapa (MLNN)?

Definimos la contratación de la Hipótesis como:

La evaluación y/o análisis de las métricas obtenidas al comparar cada algoritmo sin PCA con su respectiva contraparte con PCA.

Para lo cual primero computaremos lo siguiente:

- *Matriz de confusión*

Para de este modo poder aplicar los indicadores definidos en el fundamento teórico:

- *Precisión*
- *Recall*
- *F – Score*

Finalmente realizaremos un análisis gráfico de los resultados por medio de:

- *Curva ROC*

5.3. Cálculo de Métricas:

5.3.1. Diseño de la Matriz de confusión (Confussion Matrix):

A continuación vamos a definir la matriz de confusión para nuestro sistema; la cual cuenta con dos clases: RB (listo para la biodegradación) mapeada como 2 y NRB (no lista para la biodegradación) mapeada como 1; a su vez las columnas representarán a las clases del sistema; mientras que las filas serán las clases correspondientes a las predicciones de los algoritmos (SVM y MLNN).

LABELS / PREDICCIONES	LISTO PARA BIODEGRADACIÓN (RB)	NO LISTO PARA LA BIODEGRADACIÓN (NRB)
Listo para Biodegradación (RB)	TP	FP
No listo para la biodegradación (NRB)	FN	TN

Tabla N° 14: Diseño de la Matriz de confusión General del Sistema

Una vez definida nuestra matriz de confusión; la vamos a aplicar a los 4 casos que hemos obtenido; los cuales son:

5.3.1.1. Matriz de confusión SVM sin aplicar PCA al data set:

LABELS / PREDICCIONES	LISTO PARA BIODEGRADACIÓN (RB)	NO LISTO PARA LA BIODEGRADACIÓN (NRB)
Listo para Biodegradación (RB)	126	12
No listo para la biodegradación (NRB)	20	53

Tabla N° 15: Matriz de confusión SVM sin aplicar PCA al data set

5.3.1.2. Matriz de confusión SVM aplicando PCA al data set:

LABELS / PREDICCIONES	LISTO PARA BIODEGRADACIÓN (RB)	NO LISTO PARA LA BIODEGRADACIÓN (NRB)
Listo para Biodegradación (RB)	138	5
No listo para la biodegradación (NRB)	14	54

Tabla N° 16: Matriz de confusión SVM aplicando PCA al data set

5.3.1.3. *Matriz de confusión MLNN sin aplicar PCA al data set:*

LABELS / PREDICCIONES	LISTO PARA BIODEGRADACIÓN (RB)	NO LISTO PARA LA BIODEGRADACIÓN (NRB)
Listo para Biodegradación (RB)	121	17
No listo para la biodegradación (NRB)	13	60

Tabla N° 17: Matriz de confusión MLNN sin aplicar PCA al data set

5.3.1.4. *Matriz de confusión MLNN aplicando PCA al data set:*

LABELS / PREDICCIONES	LISTO PARA BIODEGRADACIÓN (RB)	NO LISTO PARA LA BIODEGRADACIÓN (NRB)
Listo para Biodegradación (RB)	132	11
No listo para la biodegradación (NRB)	12	56

Tabla N° 18: Matriz de confusión MLNN aplicando PCA al data set

De las matrices obtenidas; podemos observar que cuando se aplicó PCA al data set, tanto los algoritmos SVM y MLNN presentaron una disminución significativa en el número de falsos positivos y falsos negativos; en este punto de la contrastación de la hipótesis, y, basándonos en los resultados de las matrices; podríamos inferir que, ambos algoritmos con PCA presentan mejoras con respecto a su performance de aprendizaje; ya que, la disminución de FP y FN; se traduce en un incremento en la detección de las clases ***RB*** y ***NRB***; sin embargo; necesitamos realizar un análisis más profundo para estar del todo seguros; por lo que ahora se procederá a realizar el análisis de las métricas de performance de aprendizaje; para poder corroborar o descartar la inferencia que hemos realizado.

5.3.2. Precisión:

Dado que estamos trabajando con dos clases (RB y NRB); y para mostrar un análisis más completo, vamos a aplicar la métrica a ambas clases.

5.3.2.1. *Precisión para MLNN aplicando PCA al data set:*
Para la clase RB:

$$\pi_{nn\_pca}^{(RB)} = \frac{132}{132 + 11} = 0.9231$$

Para la clase NRB:

$$\pi_{nn\_pca}^{(NRB)} = \frac{56}{56 + 12} = 0.8235$$

5.3.2.2. *Precisión para MLNN sin aplicar PCA al data set:*
Para la clase RB:

$$\pi_{nn}^{(RB)} = \frac{121}{121 + 17} = 0.8768$$

Para la clase NRB:

$$\pi_{nn}^{(NRB)} = \frac{60}{60 + 13} = 0.8219$$

5.3.2.3. *Precisión para SVM aplicando PCA al data set:*
Para la clase RB:

$$\pi_{svm\_pca}^{(RB)} = \frac{138}{138 + 5} = 0.9650$$

Para la clase NRB:

$$\pi_{svm\_pca}^{(NRB)} = \frac{54}{54 + 14} = 0.7941$$

5.3.2.4. *Precisión para SVM sin aplicar PCA al data set:*
Para la clase RB:

$$\pi_{svm}^{(RB)} = \frac{126}{126 + 12} = 0.9130$$

Para la clase NRB:

$$\pi_{svm}^{(NRB)} = \frac{53}{53 + 20} = 0.7260$$

5.3.3. Recall (sensibilidad):

Al igual como con la precisión, dado que estamos trabajando con dos clases (RB y NRB); y para mostrar un análisis más completo; vamos a aplicar sensibilidad (recall) a ambas clases.

5.3.3.1. Recall para MLNN aplicando PCA al data set:

Para la clase RB:

$$\rho_{nn\_pca}^{(RB)} = \frac{132}{132 + 12} = 0.9167$$

Para la clase NRB:

$$\rho_{nn\_pca}^{(NRB)} = \frac{56}{56 + 11} = 0.8358$$

5.3.3.2. Recall para MLNN sin aplicar PCA al data set:

Para la clase RB:

$$\rho_{nn}^{(RB)} = \frac{121}{121 + 13} = 0.9030$$

Para la clase NRB:

$$\rho_{nn}^{(NRB)} = \frac{60}{60 + 17} = 0.7792$$

5.3.3.3. Recall para SVM aplicando PCA al data set:

Para la clase RB:

$$\rho_{svm\_pca}^{(RB)} = \frac{138}{138 + 14} = 0.9079$$

Para la clase NRB:

$$\rho_{svm\_pca}^{(NRB)} = \frac{54}{54 + 5} = 0.9153$$

5.3.3.4. *Recall para SVM sin aplicar PCA al data set:*

Para la clase RB:

$$\rho_{svm}^{(RB)} = \frac{126}{126 + 20} = 0.8630$$

Para la clase NRB:

$$\rho_{svm}^{(NRB)} = \frac{53}{53 + 12} = 0.8154$$

5.3.4. F Micro averaged Score:

Una vez realizado el cálculo de la precisión y recall de cada uno de nuestros algoritmos (SVM y MLNN); vamos a obtener los valores de *F Micro averaged Score*; para lo cual emplearemos una tabla por cada algoritmo con el fin de poder calcular las sumatorias respecto a las clases tanto para la precisión como recall.

5.3.4.1. *F Micro averaged Score: MLNN aplicando PCA al data set:*

A continuación se muestra la tabla de variables para poder calcular la métrica F Micro averaged Sccore:

LABEL	TP	FP	FN
RB	132	11	12
NRB	56	12	11
TOTAL	188	23	23

Tabla N° 19: Cálculo de variables F Micro averaged Score para el algoritmo MLNN aplicando PCA

Una vez obtenida la tabla; procedemos a calcular los valores de π, ρ respectivamente:

$$\pi = \frac{TP}{TP + FP} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m} (TP_i + FP_i)} = \frac{188}{188 + 23} = 0.8910$$

$$\rho = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m} (TP_i + FN_i)} = \frac{188}{188 + 23} = 0.8910$$

Finalmente; teniendo los valores de π, ρ; procedemos a calcular la métrica F micro averaged Score:

$$F_{micro_{aveg}}(nn\_pca) = \frac{2\pi\rho}{\pi + \rho} = 0.8910$$

5.3.4.2. *Micro F averaged Score: MLNN sin aplicar PCA al data set:*

LABEL	TP	FP	FN
RB	121	17	13
NRB	60	13	17
TOTAL	181	30	30

Tabla N° 20: Cálculo de variables F Micro averaged Score para el algoritmo MLNN sin PCA

$$\pi = \frac{TP}{TP + FP} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m}(TP_i + FP_i)} = \frac{181}{181 + 30} = 0.8578$$

$$\rho = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m}(TP_i + FN_i)} = \frac{181}{181 + 30} = 0.8578$$

$$F_{micro_{aveg}}(nn) = \frac{2\pi\rho}{\pi + \rho} = 0.8578$$

5.3.4.3. *Micro F averaged Score: SVM aplicando PCA al data set:*

LABEL	TP	FP	FN
RB	138	5	14
NRB	54	14	5
TOTAL	192	19	19

Tabla N° 21: Cálculo de variables F Micro averaged Score para el algoritmo SVM aplicando PCA

$$\pi = \frac{TP}{TP + FP} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m}(TP_i + FP_i)} = \frac{192}{192 + 19} = 0.91$$

$$\rho = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m}(TP_i + FN_i)} = \frac{192}{192 + 19} = 0.91$$

$$F_{micro_{aveg}}(svm\_pca) = \frac{2\pi\rho}{\pi + \rho} = 0.91$$

5.3.4.4. *Micro F averaged Score: SVM sin aplicar PCA al data set:*

LABEL	TP	FP	FN
RB	126	12	20
NRB	53	20	12
TOTAL	179	32	32

Tabla N° 22: Cálculo de variables F Micro averaged Score para el algoritmo MLNN sin PCA

$$\pi = \frac{TP}{TP + FP} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m}(TP_i + FP_i)} = \frac{179}{179 + 32} = 0.8483$$

$$\rho = \frac{TP}{TP + FN} = \frac{\sum_{i=1}^{m} TP_i}{\sum_{i=1}^{m}(TP_i + FN_i)} = \frac{179}{179 + 32} = 0.8483$$

$$F_{micro_{aveg}}(svm) = \frac{2\pi\rho}{\pi + \rho} = 0.8483$$

5.4. Contrastacion de Hipótesis:

Una vez obtenidas las métricas para los algoritmos SVM y MLNN; vamos a realizar el análisis y/o comparación de las mismas; para lo cual dividiremos a los algoritmos en 2 grupos: (1) aquellos a los que se aplicó PCA al data set y (2) los que trabajaron con los datos en su estado normal, para posteriormente comparar las métricas de los algoritmos.

5.4.1. Análisis de la precisión y recall para SVM:

Para poder realizar el análisis de la performance de aprendizaje entre los algoritmos SVM con y sin PCA; vamos a construir la siguiente tabla:

PERFORMANCE DE APRENDIZAJE (π, ρ) – SVM							
APLICANDO PCA AL DATA SET				**SIN APLICAR PCA AL DATA SET**			
Support Vector Machine (SVM)				Support Vector Machine (SVM)			
Precisión		*Recall*		*Precisión*		*Recall*	
$\pi_{svm\_pca}^{(RB)}$	$\pi_{svm\_pca}^{(NRB)}$	$\rho_{svm\_pca}^{(RB)}$	$\rho_{svm\_pca}^{(NRB)}$	$\pi_{svm}^{(RB)}$	$\pi_{svm}^{(NRB)}$	$\rho_{svm}^{(RB)}$	$\rho_{svm}^{(NRB)}$
0.9650	0.7941	0.9079	0.9153	0.9130	0.7260	0.8630	0.8154

Tabla N° 23: Comparación de Performance de Aprendizaje SVM (π, ρ)

Teniendo estas métricas; podemos proceder con el análisis; para el caso de la clase ***RB*** el algoritmo SVM con PCA logra clasificar correctamente (π) a un

mayor porcentaje de elementos de la clase ***RB***, en comparación con el algoritmo SVM sin PCA: $96.50\% > 91.30\%$; podemos observar una marcada diferencia de **5.20%** entre el porcentaje π de ambos algoritmos, lo cual implica que el algoritmo SVM con PCA tiene una mejora significativa del **5.20%** en su capacidad para clasificar a los elementos de la clase ***RB*** en contraste con el algoritmo SVM sin PCA. Ahora procederemos a analizar los resultados para la métrica ρ; el algoritmo SVM con PCA logra recuperar un 90.79% de elementos de la clase ***RB*** del data set; mientras que el algoritmo SVM sin PCA recupera el 86.30% de elementos del data set; podemos notar que el algoritmo SVM con PCA presenta un porcentaje mayor con respecto a los elementos recuperados de la clase ***RB*** que el algoritmo SVM sin PCA: $90.79\% > 86.30\%$; existiendo una diferencia significativa de **4.49%** entre la métrica ρ de ambos algoritmos, esta diferencia nos indica que el algoritmo SVM con PCA logra incrementar su capacidad para recuperar elementos de la clase ***NRB*** del data set en un **4.49%** en contraste a su contraparte sin PCA.

De igual manera al realizar el análisis para la clase ***NRB*** notamos que el algoritmo SVM con PCA logra clasificar correctamente (π) a un mayor porcentaje de elementos de la clase ***NRB***, en contraste con el algoritmo SVM sin PCA: $79.41\% > 72.60\%$; podemos apreciar que al igual que con la clase ***RB***, existe una marcada diferencia entre los algoritmos de **6.81%**; lo cual implica que el algoritmo SVM con PCA tiene una mejora significativa del **6.81%** en su capacidad para clasificar a los elementos de la clase ***NRB*** en contraste con el algoritmo SVM sin PCA. Ahora procederemos a analizar los resultados para la métrica ρ; el algoritmo SVM con PCA logra recuperar un 91.53% de elementos pertenecientes a la clase ***NRB*** del data set; mientras que el algoritmo SVM sin PCA recupera el 81.54% de elementos de la clase ***NRB***; podemos apreciar que el algoritmo SVM con PCA presenta un porcentaje mayor de elementos recuperados de la clase ***NRB*** con respecto al algoritmo SVM sin PCA: $91.53\% > 81.54\%$; existiendo una remarcada diferencia de **9.99%** entre la métrica ρ de ambos algoritmos, esta diferencia nos indica que

el algoritmo SVM con PCA logra incrementar su capacidad para recuperar elementos de la clase ***NRB*** del data set en un **9.99%** en contraste a su contraparte sin PCA.

Una vez realizado el análisis de las métricas π, ρ para los algoritmos SVM, podemos observar que el algoritmo SVM con PCA presenta valores superiores con respecto a π, ρ tanto para la clase ***RB*** como ***NRB*** en comparación al algoritmo SVM sin PCA; lo que se traduce en la mejora de la performance de aprendizaje por parte del algoritmo SVM con PCA, por consiguiente el algoritmo SVM con PCA clasifica mejor a ambas clases (RB, NRB), en comparación al algoritmo SVM sin PCA.

5.4.2. Análisis de la precisión y recall para MLNN:

Para poder realizar el análisis de la performance de aprendizaje entre los algoritmos MLNN con y sin PCA; vamos a construir una tabla; en donde, resumimos las métricas $\boldsymbol{\pi, \rho}$ obtenidas para las clases RB y NRB respectivamente de la siguiente manera:

PERFORMANCE DE APRENDIZAJE ($\boldsymbol{\pi, \rho}$) – MLNN							
APLICANDO PCA AL DATA SET				**SIN APLICAR PCA AL DATA SET**			
Red Neuronal Multicapa (MLNN)				Red Neuronal Multicapa (MLNN)			
Precisión		*Recall*		*Precisión*		*Recall*	
$\pi_{nn\_pca}^{(RB)}$	$\pi_{nn\_pca}^{(NRB)}$	$\rho_{nn\_pca}^{(RB)}$	$\rho_{nn\_pca}^{(NRB)}$	$\pi_{nn}^{(RB)}$	$\pi_{nn}^{(NRB)}$	$\rho_{nn}^{(RB)}$	$\rho_{nn}^{(NRB)}$
0.9231	0.8235	0.9167	0.8358	0.8768	0.8219	0.9030	0.7792

Tabla N° 24: Comparación de Performance de Aprendizaje MLNN (π, ρ)

De los resultados adjuntos; al realizar el análisis para la clase ***RB***, el algoritmo MLNN con PCA logra clasificar correctamente (π) a un porcentaje mayor de elementos en comparación con el algoritmo MLNN sin PCA: $92.31\% > 87.68\%$; podemos observar una marcada diferencia de **4.63%** entre el porcentaje π de ambos algoritmos, lo cual implica que el algoritmo MLNN con PCA tiene una mejora significativa del **4.63%** en su capacidad para clasificar a los elementos de la clase ***RB*** en contraste con el algoritmo MLNN

sin PCA. Ahora procederemos a analizar los resultados para la métrica ρ; el algoritmo MLNN con PCA logra recuperar un 91.67% de elementos de la clase ***RB*** del data set; mientras que el algoritmo MLNN sin PCA logra recuperar un 90.30%, podemos apreciar que $91.67\% > 90.30\%$ por un porcentaje de **1.37%**, esta diferencia nos indica que el algoritmo MLNN con PCA logra incrementar su capacidad para recuperar elementos de la clase ***RB*** del data set en un **1.37%** en contraste a su contraparte sin PCA.

Al analizar los resultados para la clase ***NRB***; observamos que el algoritmo MLNN con PCA logra clasificar correctamente (π) a un porcentaje de 82.35% de elementos pertenecientes a la clase ***NRB***, mientras que el algoritmo MLNN sin PCA clasifica correctamente a un 82.19%; podemos apreciar que: $82.35\% > 82.19\%$, existiendo una diferencia de aproximadamente **0.16%** entre la precisión de ambos algoritmos, lo cual nos indica que el algoritmo MLNN con PCA tiene una mejora del **0.16%** en su capacidad para clasificar a los elementos de la clase ***RB*** en contraste con el algoritmo SVM sin PCA. Ahora procederemos a analizar los resultados para la métrica ρ; el algoritmo MLNN con PCA logra recuperar un porcentaje de 83.58% de elementos de la clase ***NRB*** del data set; mientras que el algoritmo MLNN sin PCA, recupera un porcentaje de 77.92%; podemos observar que: $83.58\% > 77.92\%$, existiendo una diferencia significativa del **5.66%** entre la métrica ρ de los algoritmos; esta diferencia nos indica que el algoritmo MLNN con PCA logra incrementar su capacidad para recuperar elementos de la clase ***NRB*** del data set en un **5.66%** en contraste a su contraparte sin PCA.

Una vez realizado el análisis de las métricas π, ρ para los algoritmos MLNN, podemos observar que el algoritmo MLNN con PCA presenta valores superiores con respecto a π, ρ tanto para la clase ***RB*** como ***NRB*** en comparación al algoritmo MLNN sin PCA; lo que se traduce en la mejora de la performance de aprendizaje por parte del algoritmo MLNN con PCA, por

consiguiente el algoritmo MLNN con PCA clasifica mejor a ambas clases (RB, NRB), en comparación al algoritmo MLNN sin PCA.

5.4.3. Análisis de F micro averaged score para MLNN:

Ahora procederemos con el análisis de las medidas obtenidas de *F Micro averaged Score* para el algoritmo MLNN; para poder realizar el análisis contrastaremos la métrica *F Micro averaged Scor*e obtenida por el algoritmo MLNN que fue entrenado usando el data set procesado por PCA contra el algoritmo MLNN que fue entrenado usando el data set sin ninguna alteración.

PERFORMANCE DE APRENDIZAJE (F MICRO AVG SCORE) – MLNN	
APLICANDO PCA AL DATA SET	**SIN APLICAR PCA AL DATA SET**
Red Neuronal Multicapa (MLNN)	Red Neuronal Multicapa (MLNN)
$F_{micro_{aveg}}(nn\_pca)$	$F_{micro_{aveg}}(nn)$
0.8910	0.8578

Tabla N° 25: Comparación de la performance de aprendizaje (F MICRO AVG SCORE) de algoritmos MLNN

Al analizar los resultados; observamos que el algoritmo MLNN que fue entrenado con el data set procesado por PCA; presenta un porcentaje superior en contraste con el otro algoritmo MLNN: $89.10\% > 85.78\%$; al tomar en cuenta que la métrica F micro averaged Score es calculada globalmente sobre todas las clases; podemos afirmar que el algoritmo MLNN con PCA presenta una performance de aprendizaje superior con respecto al algoritmo que no usa PCA; lo que se traduce en una capacidad más alta por parte del algoritmo MLNN con PCA para poder clasificar correctamente a los elementos de las clases ***RB*** y ***NRB*** en contraste con el algoritmo MLNN que no uso PCA en su data set.

5.4.4. Análisis de F micro averaged score para SVM:

Ahora procederemos con el análisis de las medidas obtenidas de *F Micro averaged Score* para el algoritmo SVM; para poder realizar el análisis contrastaremos la métrica *F Micro averaged Scor*e obtenida por el algoritmo SVM que fue entrenado usando el data set procesado por PCA contra el algoritmo SVM que fue entrenado usando el data set sin ninguna alteración.

PERFORMANCE DE APRENDIZAJE (F MICRO AVG SCORE) – SVM	
APLICANDO PCA AL DATA SET	**SIN APLICAR PCA AL DATA SET**
Support Vector Machine (SVM)	Support Vector Machine (SVM)
$F_{micro_{aveg}}(svm\_pca)$	$F_{micro_{aveg}}(svm)$
0.91	0.8483

Tabla N° 26: Comparación de la performance de aprendizaje (F MICRO AVG SCORE) de algoritmos SVM

Al analizar los resultados; observamos que el algoritmo SVM que fue entrenado con el data set procesado por PCA; presenta un porcentaje muy superior en contraste con el otro algoritmo SVM: $91.00\% > 84.83\%$; para este caso en particular; podemos notar que el impacto que tuvo el aplicar PCA al data set se tradujo en una mejora significativa de la performance de aprendizaje del algoritmo SVM; asimismo, al tomar en cuenta que la métrica F micro averaged Score es calculada globalmente sobre todas las clases; podemos corroborar la mejora de la performance de aprendizaje del algoritmo SVM con PCA; ya que, posee una capacidad más alta para poder clasificar correctamente a los elementos de las clases ***RB*** y ***NRB*** en contraste con el algoritmo SVM que no uso PCA en su data set.

5.4.5. Análisis gráfico de la performance de aprendizaje de los algoritmos SVM y MLNN utilizando curvas ROC:

Finalmente vamos a realizar un último análisis; para lo cual usaremos las curvas ROC; el objetivo en esta sección es mostrar la performance de aprendizaje final de cada uno de los algoritmos que aplicaron PCA a su data set; en contraste con los que no lo usaron, también mostraremos la comparación de la performance de aprendizaje entre todos los algoritmos para lograr determinar el mejor algoritmo acorde con la curva ROC. El criterio que

vamos a usar para poder determinar si un algoritmo tiene la mejor performance de aprendizaje se regirá por las siguientes dos condiciones:

(1) índice de verdaderos positivos alto e

(2) índice de falsos positivos bajo

5.4.5.1. Curva ROC para los algoritmos SVM:

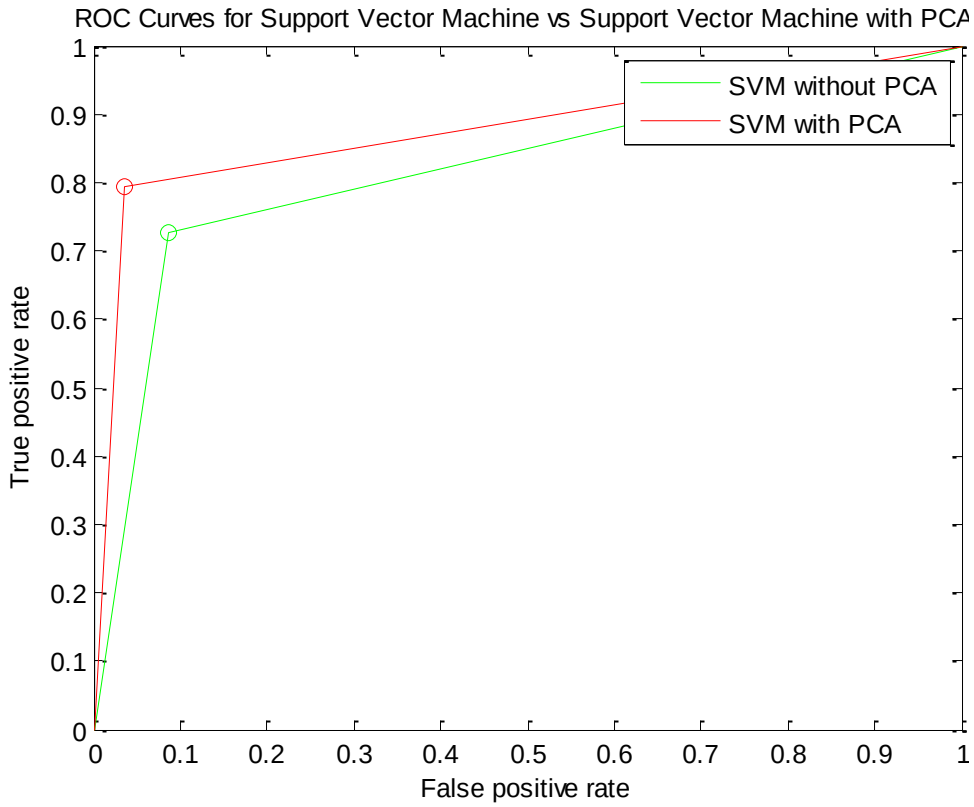


Figura N° 34: Curva ROC para los algoritmos SVM con y sin PCA

Para poder determinar cuál de los dos algoritmos SVM tiene la mejor performance de aprendizaje; tenemos que buscar el algoritmo que reúna las dos condiciones descritas anteriormente. Al observar detenidamente las curvas para ambos algoritmos; podemos notar que, el que mejor cumple con las condiciones (1) y (2) es el algoritmo SVM con PCA; por ende; la performance de este algoritmo es superior con respecto al algoritmo SVM sin PCA.

5.4.5.2. Curva ROC para los algoritmos MLNN:

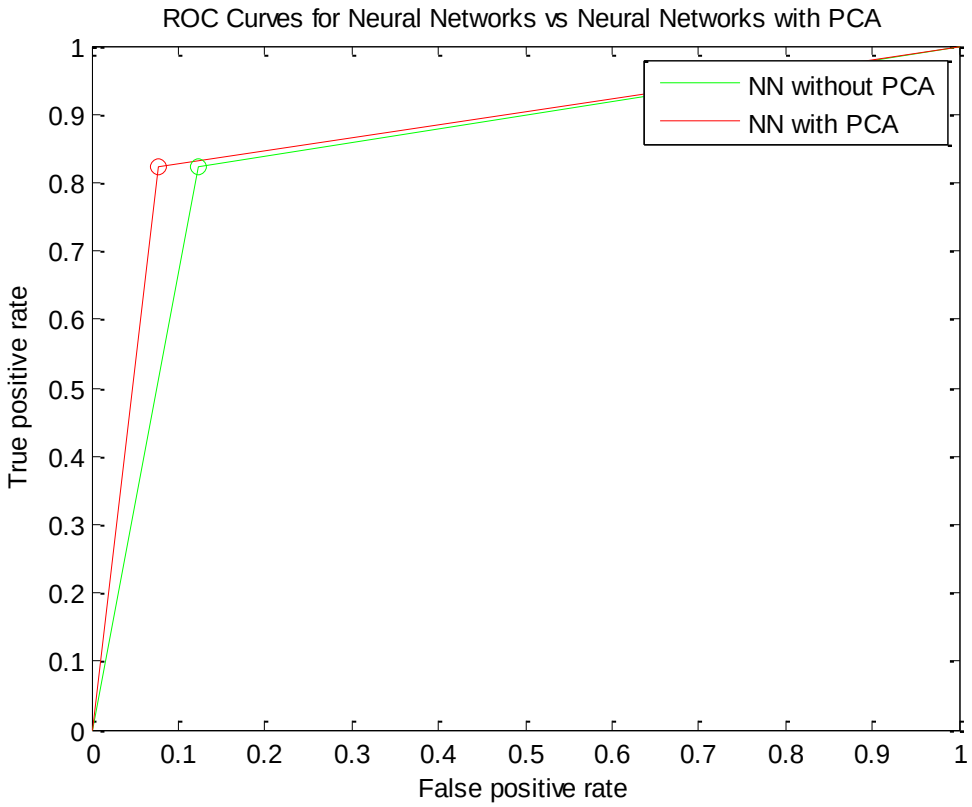


Figura N° 35: Curva ROC para los algoritmos MLNN con y sin PCA

Al observar detenidamente las curvas para ambos algoritmos; podemos notar que, el que mejor cumple con las condiciones (1) y (2) es el algoritmo MLNN con PCA; por ende; la performance de este algoritmo es superior con respecto al algoritmo MLNN sin PCA.

5.4.5.3. Curva ROC para los algoritmos MLNN y SVM:

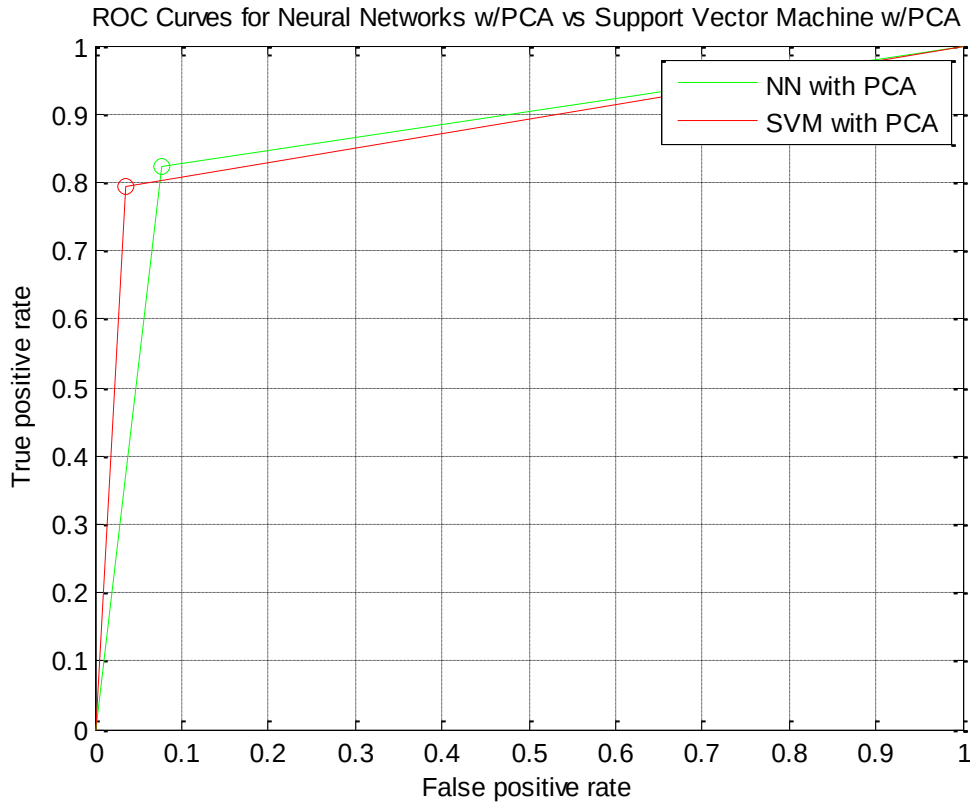


Figura N° 36: Curva ROC para los algoritmos MLNN y SVM con PCA

Ahora que ya hemos determinado que los algoritmos SVM y MLNN que aplicaron PCA a su data set tienen una performance de aprendizaje más alta; es hora de comparar la performance de aprendizaje entre estos dos algoritmos; a simple vista; podemos notar que ninguno de los algoritmos cumple a la vez las condiciones (1) y (2); sino que; cada uno de ellos cumple sólo con una de las mismas a la vez; sin embargo el algoritmo con la performance de aprendizaje superior es SVM; ya que, su índice de falsos positivos es más bajo que el algoritmo MLNN; el cual presenta un índice más alto de verdaderos positivos; pero a su vez incrementa el número de falsos positivos con lo que su capacidad de clasificación se ve menguada.

5.4.5.4. Curva ROC para los todos los algoritmos MLNN y SVM con y sin PCA:

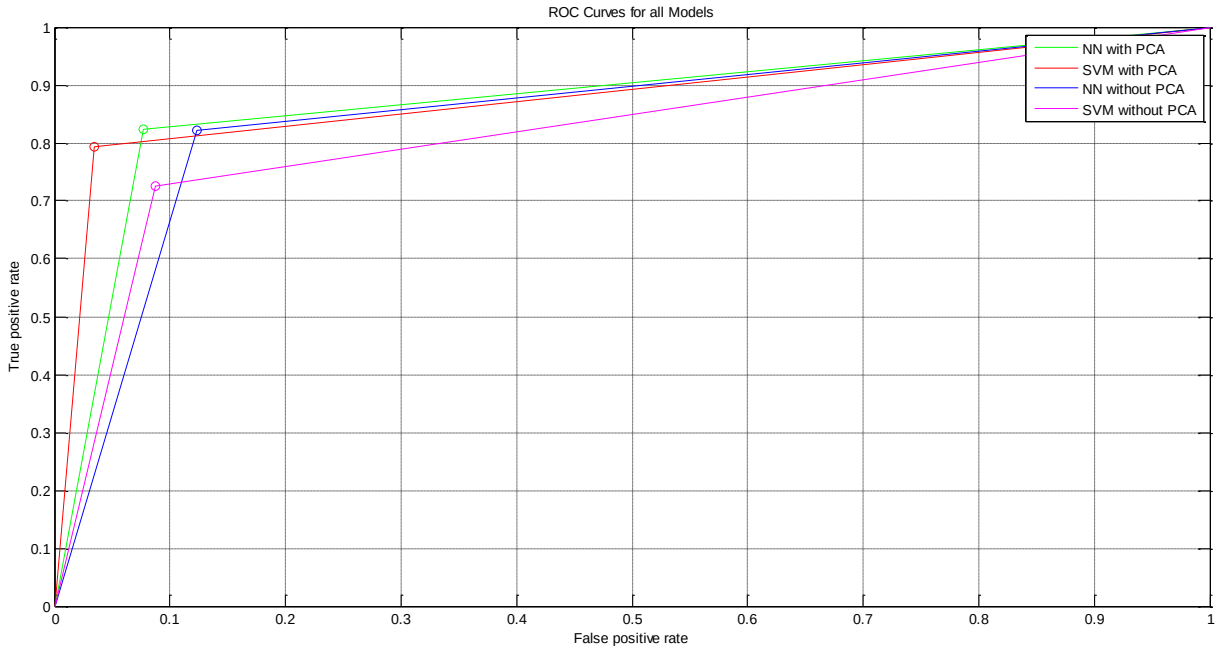


Figura N° 37: Curva ROC para los todos los algoritmos MLNN y SVM con y sin PCA

Finalmente al comparar la performance de aprendizaje de todos los algoritmos; podemos notar que el mejor de ellos es SVM con PCA; ya que presenta el índice de falsos positivos más bajo y también mantiene un índice de verdaderos positivos alto; seguido se encuentra el algoritmo MLNN con PCA; el cual presenta un índice de verdaderos positivos mayor al del algoritmo SVM con PCA; no obstante su índice de falsos positivos es mayor que el algoritmo SVM con PCA; en el tercer lugar tenemos al algoritmo SVM sin PCA; el algoritmo no presenta un índice de verdaderos positivos muy bajo; no obstante su índice de falsos positivos es mayor que el algoritmo MLNN con PCA; y, finalmente; el puesto de algoritmo con la performance de aprendizaje más baja le pertenece al algoritmo MLNN sin PCA; el cual, si bien es cierto presenta un índice de verdaderos positivos que rivaliza con su contraparte con PCA; también muestra el mayor índice de falsos negativos de todos los

algoritmos; con lo cual; cualquier clasificación que haga éste algoritmo tiene una mayor probabilidad a estar mal clasificada.

Conclusiones

Después de la finalización del trabajo de investigación se pudo concluir que:

1. Al obtener el data set QSAR biodegradation se pudo observar que; había un número mayor de observaciones correspondientes a la clase RB en contraste con la clase NRB: $699 > 356$.

2. Al realizar la descripción general de características del data set se pudo observar que los valores predictivos estaban en un formato nominal; por lo que se tuvo que realizar un mapeo de dichos valores a un formato ordinal con el fin de que se puedan llevar a cabo las respectivas fases de entrenamiento y/o validación de los algoritmos SVM y MLNN. Asimismo la aplicación de herramientas para el análisis de los data sets como los diccionarios de datos, análisis de características; entre otras; tienen vital importancia; ya que muestran información acerca de la estructura del data set; con lo cual; se pueden adaptar y/o entrenar mejor los algoritmos.

3. Principal Component Analysis (PCA) logró mejorar significativamente la performance de aprendizaje de los algoritmos SVM y MLNN; cuando se aplicó sobre el data set QSAR biodegradation, asimismo, referente a la implementación de PCA, se hizo uso de la función SVD de Matlab para poder llevar a cabo el proceso de descomposición de las matrices.

4. El algoritmo de Red Neuronal obtenido a partir de la data que fue procesada por PCA presentaba una estructura compuesta de 4 capas; 1 de entrada, 2 ocultas y 1 de salida; siendo la distribución del número de neuronas correspondientes a cada capa como: 31, 6, 5 y 2 neuronas respectivamente; cabe resaltar que el número de neuronas de la capa de entrada disminuyo debido a que las dimensiones del data set fueron reducidas por PCA. Asimismo; al analizar las curvas de aprendizaje de este algoritmo; no se detectó problemas de varianza alta, ni sesgo alto.

5. Al finalizar el entrenamiento del algoritmo SVM aplicando PCA al data set se puede observar un cambio en el comportamiento de la curva de aprendizaje del algoritmo; a simple inspección se podría decir que el algoritmo presenta una varianza alta, sin embargo, al observar la escala en la cual se encuentra la curva se puede apreciar que se ubica en un intervalo menor $[0: 0.08]$ con respecto a $[0: 0.5]$ del algoritmo sin PCA; este hecho nos indica que el error tanto para los datos de entrenamiento como cruzados ha disminuido considerablemente.

6. El algoritmo de Red Neuronal obtenido a partir de la data que no fue procesada por PCA presentaba una estructura compuesta de 4 capas; 1 de entrada, 2 ocultas y 1 de salida; siendo la distribución del número de neuronas correspondientes a cada capa como: 41, 3, 5 y 2 neuronas respectivamente. Asimismo; al observar las curvas de aprendizaje para este algoritmo; se pudo detectar un problema de varianza alta.

7. Al finalizar la implementación del algoritmo SVM sin aplicar PCA; se pudo notar que la exactitud del algoritmo respecto a los datos de prueba era del 84.83%, el cual es ligeramente menor en comparación a la exactitud obtenida por algoritmo MLNN sin aplicar PCA 85.78%; no obstante al analizar más detenidamente las medidas de precisión y recall para ambos algoritmos se pudo apreciar que el algoritmo SVM tiene una mayor facilidad para clasificar correctamente a elementos de la clase NRB; en comparación con el algoritmo MLNN que presenta una afinidad mayor para reconocer a elementos de la clase RB; esto se debe a que el recall obtenido por el algoritmo SVM es mayor con respecto al algoritmo MLNN para la clase NRB $81.54\% > 77.92\%$; no obstante el algoritmo MLNN presenta un recall mayor con respecto a la clase RB $90.30\% > 86.30\%$; lo cual explica que el algoritmo MLNN pueda recuperar más elementos de la clase RB.

8. Al comparar las métricas π, ρ entre los algoritmos MLNN (con y sin PCA); se pudo notar una mejora con respecto a π del *4.63%* y *1.37%* para las clases ***RB*** y ***NRB*** respectivamente mientras que para ρ se presentaron mejoras del *0.16%* y *5.66%* para las clases ***RB*** y ***NRB*** respectivamente; podemos apreciar que tanto la mejora de π, ρ con

respecto a las clases ***NRB*** y ***RB*** no es muy significativa, éste escenario puede ser atribuido a factores como: número reducido de instancias de la clase ***NRB***, lo que dificulta su clasificación y por ende impacta en la métrica π, número de iteraciones de entrenamiento y método del gradiente descendente los cuales tienen un impacto sobre la métrica ρ para la clase ***RB***; no obstante, tanto para la clase ***RB*** y ***NRB*** se pudo apreciar una mejora significativa en las métricas π, ρ. De igual manera al comparar las métricas π, ρ entre los algoritmos SVM (con y sin PCA); se pudo notar una mejora con respecto a π del *5.20%* y *6.81%* para las clases ***RB*** y ***NRB*** respectivamente mientras que para ρ se presentaron mejoras del *4.49%* y *9.99%* para las clases ***RB*** y ***NRB*** respectivamente; podemos apreciar que tanto la mejora de π, ρ con respecto a ambas clase ***RB*** y ***NRB*** presentan valores altos, sobresaliendo el valor de ρ para la clase ***NRB***. Podemos concluir que basados en las métricas π, ρ tanto MLNN como SVM han mejorado su performance de aprendizaje al utilizar el data set procesado por PCA.

9. Al observar los resultados obtenidos por la métrica F micro averaged Score para los algoritmos MLNN; se pudo apreciar que; una mejora en la performance de aprendizaje del *3.32%*, asimismo al evaluar los resultados de la métrica F micro averaged Score para los algoritmos SVM se puedo observar una mejora significativa con respecto a la performance de aprendizaje del algoritmo del *6.17%;* más alta aún que el algoritmo MLNN.

10. Finalmente al realizar las comparaciones entre algoritmos usando la curva ROC se pudo determinar qué; el mejor algoritmo de todos fue SVM con PCA; mientras que el algoritmo con la performance de aprendizaje más baja fue MLNN sin PCA; esto, debido en parte a que; generalmente los algoritmo MLNN que son entrenados con el método del gradiente descendente presentan problemas con los mínimos locales; mientras que SVM no.

Recomendaciones

1. Al momento de obtener un data set para realizar alguna tarea de clasificación; se debe asegurar que; exista suficiente información acerca de las clases que se desean clasificar; ya que, el no tener suficientes observaciones de alguna clase, puede conllevar a que el algoritmo no sea capaz de generalizar adecuadamente dicha o dichas clases.

2. Realizar siempre la descripción de características del data set; ya que, esta información nos brinda un panorama más general acerca de la estructura de los datos. De igual manera cuando trabajamos con data sets; es importante que se preste atención a la distribución de los mismos; ya que; si se logran corregir las anomalías en el data set antes de realizar el entrenamiento; se obtendrán mejores resultados.

3. Para poder aplicar PCA se debe evaluar el dataset con el que se está trabajando; ya que, si éste presenta un gran número de dimensiones; éstas aumentarán el tiempo de procesamiento de los algoritmos (aumentando también los recursos computacionales); en este escenario en particular, la aplicación de PCA puede ser considerada; no obstante siempre debemos de tener en cuenta que PCA realiza una reducción de las dimensiones; y lo por tanto, un porcentaje (ya sea mínimo) de los datos se pierden.

4. Con respecto al algoritmo MLNN con PCA; el cual hace uso del método del gradiente descendente, se recomienda usar librerías para la computación numérica que puedan encontrar los mínimos locales usando gradientes.

5. Se recomienda añadir el análisis de curvas de aprendizaje para los algoritmos; ya que, nos brindan información valiosa a la hora de analizar el comportamiento en términos de varianza y bias altos.

6. Con respecto al algoritmo MLNN entrenado con el data set sin ser procesado por PCA se recomienda: aumentar las observaciones del data set o incrementar el valor del parámetro regularizador λ para lograr una mejor performance de aprendizaje.

7. Cuando se aplique SVM, se recomienda tener en consideración el tipo de kernel que se vaya a usar; ya que esto afectara enormemente la capacidad de clasificación del algoritmo.

8. Se recomienda hacer uso de varios indicadores a la hora de realizar comparaciones ente algoritmos; ya que de esta manera se puede tener un panorama desde varias perspectivas de la performance de aprendizaje de los algoritmos. Asimismo se recomienda para futuros trabajos con el data set QSAR Biodegradation conseguir más instancias de la clase ***NRB***, así como también si se desea usar el algoritmo MLNN usando el método del gradiente descendente se debe hacer énfasis en el número de iteraciones, así como también se debe vigilar de cerca los valores obtenidos por el gradiente respecto al costo.

9. Respecto a los problemas para π, ρ con respecto a las clases ***NRB*** y ***RB*** para el algoritmo MLNN mostrados en las conclusiones (8); se podría mejorar los indicadores por medio de la realización de un análisis más minucioso a la distribución del data set, así como también un re – entrenamiento en donde se podrían agregar más neuronas y/o capas ocultas para incrementar la capacidad de complejidad del algoritmo.

10. Finalmente para trabajos futuros recomendamos aplicar el estudio sobre mas data sets; para así poder tener una referencia más global del impacto de PCA como herramienta de pre procesamiento para los data sets.

Bibliografía

Asch, V. Van. (2013). Macro-and micro-averaged evaluation measures [[BASIC DRAFT]], 1–27.

Awad, M., & Khan, L. (2004). An effective support vector machines (SVMs) performance using hierarchical clustering. *Tools with Artificial …*, 1–20. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1374251

Awad, M., & Khanna, R. (2015). *Efficient Learning Machines Theories, Concepts, and Applications for Engineers and System Designers* (1st ed.). Apress. Retrieved from http://www.apress.com/9781430259893?gtmf=s

Baker, K. (2013). Singular Value Decomposition Tutorial Contents, *2005*, 1–24.

Bell, J. (2014). *Machine Learning: Hands-on for Developers and Technical Professionals* (1st ed.). Indianapolis, Indiana: John Wilet & Sons, Inc.

Benali, a. (2013). Principal Component Analysis and Neural Networks for Predicting the Pile Capacity Using SPT. *International Journal of Engineering and Technology*, *5*(1), 162–169. http://doi.org/10.7763/IJET.2013.V5.533

Ben-Hur, A., & Weston, J. (2010). A user's guide to support vector machines. *Methods in Molecular Biology (Clifton, N.J.)*, *609*, 223–239. http://doi.org/10.1007/978-1-60327-241-4_13

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory* (pp. 144–152). http://doi.org/10.1.1.21.3818

Cesar, R. M., & da Fontoura Costa, L. (1997). An introduction to neural networks. *Neurocomputing*, *14*(1), 101–104. http://doi.org/10.1016/S0925-2312(96)00046-X

Chang, C., & Lin, C. (2011). LIBSVM : A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *2*, 1–39. http://doi.org/10.1145/1961189.1961199

Chellappa, R., & Theodoridis, S. (2014). *Signal Processing Theory and Machine Learning* (1st ed.). Academic Press.

Chih-Wei Hsu, Chih-Chung Chang, and C.-J. L. (2008). A Practical Guide to Support Vector Classification. *BJU International*, *101*(1), 1396–400. http://doi.org/10.1177/02632760022050997

Daw, N. D. (2003). *Reinforcement learning models of the dopamine system and their behavioral implications*. *Science*. Carnegie Mellon University. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.6376&rep=rep1&type=pdf

Dietterich, T. G., & Kong, E. B. (1995). Machine Learning Bias , Statistical Bias , and Statistical Variance of Decision Tree Algorithms. *Machine Learning*, *255*, 0–13. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.38.2702&rep=rep1&type=pdf

Farley, B., & Clark, W. (1954). Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory*, *4*(4), 76–84. http://doi.org/10.1109/TIT.1954.1057468

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874. http://doi.org/10.1016/j.patrec.2005.10.010

Flach, P. (2012). *Machine Learning The Art and Science of Algorithms that Make Sense of Data* (1st ed.). Cambridge University Press.

Fletcher, T. (2009). Appendix B: Tutorial Introduction to R. In *Knowledge Discovery with Support Vector Machines* (pp. 221–230). Hoboken, NJ, USA: John Wiley & Sons, Inc. http://doi.org/10.1002/9780470503065.app2

Geras, K. J. (2011). *Prediction Markets for Machine Learning*. *Artificial Intelligence*. University of Warsaw.

Hastie, T., & Tibshirani, R. (2015). *Statistical Learning with Sparsity The Lasso and Generalizations*. Chapman and Hall/CRC.

Heba, F. E., Darwish, A., Hassanien, A. E., & Abraham, A. (2010). Principle components analysis and Support Vector Machine based Intrusion Detection System. *2010 10th International Conference on Intelligent Systems Design and Applications*, 363–367. http://doi.org/10.1109/ISDA.2010.5687239

Hebb, D. O. (1949). The first stage of perception: growth of the assembly BT - The Organization of Behavior. *The Organization of Behavior*, (4), 60–78. http://doi.org/10.1016/0301-0082(84)90021-2

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*(6), 417–441. http://doi.org/10.1037/h0071325

Lee, M.-C., & To, C. (2010). Comparison of Support Vector Machine and Back Propagation Neural Network in Evaluating the Enterprise Financial Distress, *1*(3), 13. http://doi.org/10.5121/ijaia.2010.1303

Mansouri, K., Ringsted, T., Ballabio, D., Todeschini, R., & Consonni, V. (2013). Quantitative structure-activity relationship models for ready biodegradability of chemicals. *Journal of Chemical Information and Modeling*, *53*(4), 867–78. http://doi.org/10.1021/ci4000213

Mo, H. (2009). *Handbook of research on Artificial Immune Systems and Natural Computing: Applying Complex Adaptive Technologies* (1st ed.). New York: Medical Information Science Reference. http://doi.org/10.4018/978-1-60566-310-4.ch007

Ng, A. Y. (2013a). Backpropagation Algorithm - Ufldl. Retrieved December 2, 2015, from http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm

Ng, A. Y. (2013b). Neural Networks - Ufldl. Retrieved December 2, 2015, from http://ufldl.stanford.edu/wiki/index.php/Neural_Networks

Nielsen, M. A. (2015). Neural Networks and Deep Learning. Retrieved from http://neuralnetworksanddeeplearning.com

Özgür, A., Özgür, L., & Güngör, T. (2005). Text Categorization with Class-Based and Corpus-Based Keyword Selection. *Proceedings of the 20th International Conference on Computer and Information Sciences*, *3733*, 606–615. http://doi.org/10.1007/11569596

Patra, D., Das, M. K., & Pradhan, S. (2009). Integration of FCM, PCA and Neural Networks for classification of ECG Arrhythmias. *IAENG International Journal of Computer Science*, *36*(3).

Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, *2*(11), 559–572. http://doi.org/10.1080/14786440109462720

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. http://doi.org/10.1037/h0042519

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*. http://doi.org/10.1147/rd.33.0210

Schölkopf, B., Tsuda, K., & Vert, J. P. (2004). *Kernel Methods in Computational Biology*. *Academic Radiology* (Vol. 11).

Shalev-shwartz, S., & Ben-david, S. (2014). *Understanding Machine Learning: From Theory to Algorithms* (1st ed.). Cambridge University Press. http://doi.org/10.1017/CBO9781107298019

Shlens, J. (2005). A Tutorial on Principal Component Analysis, 1–13. Retrieved from papers3://publication/uuid/4D1DBE59-7625-4528-BAB6-E076486F0C77

Spackman, K. A. (1989). Signal detection theory: valuable tools for evaluating inductive learning. In *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 160–163). Retrieved from http://portal.acm.org/citation.cfm?id=102118.102172&coll=GUIDE&dl=GUIDE&CFID=8689407&CFTOKEN=49107713

Stephen, M. (2014). *Chapman & Hall/CRC Machine Learning & Pattern Recognition Series Chapman & Hall/CRC Machine Learning & Pattern Recognition Series M AC H I N E LEARNING An Algorithmic Perspective* (2nd ed.). CRC Press.

Toh, K., & Romay, M. G. (2014). *Extreme Learning Machines 2013: Algorithms and Applications*. (F. Sun, K.-A. Toh, M. G. Romay, & K. Mao, Eds.) (Vol. 16). Cham: Springer International Publishing. http://doi.org/10.1007/978-3-319-04741-6

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, *49*, 433–460.

Verleysen, M., François, D., Simon, G., & Wertz, V. (2003). On the effects of dimensionality on data analysis with neural networks. *Artificial Neural Nets Problem Solving Methods*, *2687*, 105–112. http://doi.org/10.1007/3-540-44869-1_14

Anexos

A. Entrenamiento de SVM sin aplicar PCA:

A continuación se muestran el resultado y/o iteraciones del entrenamiento de SVMLIB sin aplicar PCA al dataset:

---

optimization finished, #iter = 223
nu = 0.660348
obj = -13.058516, rho = -1.015570
nSV = 420, nBSV = 416
Total nSV = 420
Accuracy = 66.9826% (424/633) (classification)
Accuracy = 64.9289% (137/211) (classification)
*
optimization finished, #iter = 245
nu = 0.660348
obj = -51.977221, rho = -1.117811
nSV = 423, nBSV = 415
Total nSV = 423
Accuracy = 66.9826% (424/633) (classification)
Accuracy = 64.9289% (137/211) (classification)
*
optimization finished, #iter = 242
nu = 0.660348
obj = -190.590034, rho = -1.637228
nSV = 421, nBSV = 413
Total nSV = 421
Accuracy = 74.8815% (474/633) (classification)
Accuracy = 76.7773% (162/211) (classification)
*
optimization finished, #iter = 357
nu = 0.489340
obj = -531.355074, rho = -1.435193
nSV = 327, nBSV = 298
Total nSV = 327
Accuracy = 85.1501% (539/633) (classification)
Accuracy = 85.782% (181/211) (classification)
.*
optimization finished, #iter = 1129
nu = 0.298420
obj = -1150.240413, rho = -0.913271
nSV = 287, nBSV = 137
Total nSV = 287
Accuracy = 93.5229% (592/633) (classification)

Accuracy = 82.4645% (174/211) (classification)
..*
optimization finished, #iter = 1706
nu = 0.076800
obj = -836.263457, rho = -0.751159
nSV = 365, nBSV = 5
Total nSV = 365
Accuracy = 99.684% (631/633) (classification)
Accuracy = 79.1469% (167/211) (classification)
.*
optimization finished, #iter = 1221
nu = 0.009536
obj = -494.346676, rho = -0.525447
nSV = 559, nBSV = 1
Total nSV = 559
Accuracy = 99.842% (632/633) (classification)
Accuracy = 81.5166% (172/211) (classification)
.*.*
optimization finished, #iter = 1367
nu = 0.002815
obj = -456.132793, rho = -0.368261
nSV = 620, nBSV = 0
Total nSV = 620
Accuracy = 100% (633/633) (classification)
Accuracy = 72.0379% (152/211) (classification)
..*.*
optimization finished, #iter = 1928
nu = 0.000493
obj = -319.874893, rho = -0.333373
nSV = 625, nBSV = 0
Total nSV = 625
Accuracy = 100% (633/633) (classification)
Accuracy = 66.8246% (141/211) (classification)
..*.*
optimization finished, #iter = 2180
nu = 0.000110
obj = -283.965482, rho = -0.332874
nSV = 625, nBSV = 0
Total nSV = 625
Accuracy = 100% (633/633) (classification)
Accuracy = 66.3507% (140/211) (classification)
..*.*
optimization finished, #iter = 2081
nu = 0.000027
obj = -277.015278, rho = -0.333723
nSV = 627, nBSV = 0
Total nSV = 627
Accuracy = 100% (633/633) (classification)

Accuracy = 65.4028% (138/211) (classification)
C is: 2.000000, and y is: 0.001953
*
optimization finished, #iter = 357
nu = 0.489340
obj = -531.355074, rho = -1.435193
nSV = 327, nBSV = 298
Total nSV = 327
*
optimization finished, #iter = 158
nu = 0.554240
obj = -191.749754, rho = 1.364167
nSV = 127, nBSV = 104
Total nSV = 127
*
optimization finished, #iter = 132
nu = 0.534123
obj = -188.259469, rho = -0.927426
nSV = 124, nBSV = 104
Total nSV = 124
Train result without PCA
Accuracy = 85.1501% (539/633) (classification)
Crosv result without PCA
Accuracy = 89.0995% (188/211) (classification)
Test result without PCA
Accuracy = 84.8341% (179/211) (classification)
C = 2.000000, and y = 0.001953

B. Entrenamiento de la Red Neuronal sin PCA:

A continuación se muestran el resultado y/o iteraciones del entrenamiento de la red neuronal sin aplicar PCA al dataset:

```
Iteration    1 | Cost: 1.303568e+00
Iteration    2 | Cost: 1.271557e+00
Iteration    3 | Cost: 1.270748e+00
Iteration    4 | Cost: 1.270284e+00
Iteration    5 | Cost: 1.269729e+00
Iteration    6 | Cost: 1.269447e+00
Iteration    7 | Cost: 1.269213e+00
Iteration    8 | Cost: 1.269072e+00
Iteration    9 | Cost: 1.268117e+00
Iteration   10 | Cost: 1.267091e+00
Iteration   11 | Cost: 1.266587e+00
Iteration   12 | Cost: 1.264526e+00
Iteration   13 | Cost: 1.260772e+00
Iteration   14 | Cost: 1.257621e+00
Iteration   15 | Cost: 1.246711e+00
Iteration   16 | Cost: 1.243384e+00
Iteration   17 | Cost: 1.227699e+00
Iteration   18 | Cost: 1.226863e+00
Iteration   19 | Cost: 1.219371e+00
Iteration   20 | Cost: 1.217096e+00
Iteration   21 | Cost: 1.211933e+00
Iteration   22 | Cost: 1.206942e+00
Iteration   23 | Cost: 1.101664e+00
Iteration   24 | Cost: 1.095963e+00
Iteration   25 | Cost: 1.095806e+00
Iteration   26 | Cost: 1.090440e+00
Iteration   27 | Cost: 1.061674e+00
Iteration   28 | Cost: 1.061298e+00
Iteration   29 | Cost: 1.051658e+00
Iteration   30 | Cost: 1.039036e+00
Iteration   31 | Cost: 1.038872e+00
Iteration   32 | Cost: 1.037281e+00
Iteration   33 | Cost: 1.019536e+00
Iteration   34 | Cost: 1.011125e+00
Iteration   35 | Cost: 1.005799e+00
Iteration   36 | Cost: 1.001919e+00
Iteration   37 | Cost: 1.000248e+00
Iteration   38 | Cost: 9.919298e-01
Iteration   39 | Cost: 9.882796e-01
Iteration   40 | Cost: 9.864518e-01
Iteration   41 | Cost: 9.856636e-01
Iteration   42 | Cost: 9.758054e-01
```

Iteration 43 | Cost: 9.750955e-01
Iteration 44 | Cost: 9.721928e-01
Iteration 45 | Cost: 9.701108e-01
Iteration 46 | Cost: 9.670476e-01
Iteration 47 | Cost: 9.639624e-01
Iteration 48 | Cost: 9.593430e-01
Iteration 49 | Cost: 9.589197e-01
Iteration 50 | Cost: 9.486054e-01
Iteration 51 | Cost: 9.428560e-01
Iteration 52 | Cost: 9.201462e-01
Iteration 53 | Cost: 9.178638e-01
Iteration 54 | Cost: 9.136424e-01
Iteration 55 | Cost: 9.119820e-01
Iteration 56 | Cost: 9.100759e-01
Iteration 57 | Cost: 9.091427e-01
Iteration 58 | Cost: 9.086604e-01
Iteration 59 | Cost: 9.071326e-01
Iteration 60 | Cost: 9.050879e-01
Iteration 61 | Cost: 9.031758e-01
Iteration 62 | Cost: 9.023901e-01
Iteration 63 | Cost: 9.021744e-01
Iteration 64 | Cost: 8.993654e-01
Iteration 65 | Cost: 8.992616e-01
Iteration 66 | Cost: 8.989330e-01
Iteration 67 | Cost: 8.966277e-01
Iteration 68 | Cost: 8.960446e-01
Iteration 69 | Cost: 8.934691e-01
Iteration 70 | Cost: 8.924401e-01
Iteration 71 | Cost: 8.913383e-01
Iteration 72 | Cost: 8.907666e-01
Iteration 73 | Cost: 8.903550e-01
Iteration 74 | Cost: 8.880621e-01
Iteration 75 | Cost: 8.837956e-01
Iteration 76 | Cost: 8.797045e-01
Iteration 77 | Cost: 8.790105e-01
Iteration 78 | Cost: 8.772217e-01
Iteration 79 | Cost: 8.728961e-01
Iteration 80 | Cost: 8.672940e-01
Iteration 81 | Cost: 8.667845e-01
Iteration 82 | Cost: 8.652506e-01
Iteration 83 | Cost: 8.649939e-01
Iteration 84 | Cost: 8.646927e-01
Iteration 85 | Cost: 8.637814e-01
Iteration 86 | Cost: 8.627009e-01
Iteration 87 | Cost: 8.625061e-01
Iteration 88 | Cost: 8.623803e-01
Iteration 89 | Cost: 8.622992e-01
Iteration 90 | Cost: 8.622466e-01

Iteration 91 | Cost: 8.620215e-01
Iteration 92 | Cost: 8.609388e-01
Iteration 93 | Cost: 8.569922e-01
Iteration 94 | Cost: 8.551667e-01
Iteration 95 | Cost: 8.514984e-01
Iteration 96 | Cost: 8.508842e-01
Iteration 97 | Cost: 8.496490e-01
Iteration 98 | Cost: 8.492999e-01
Iteration 99 | Cost: 8.480223e-01
Iteration 100 | Cost: 8.479530e-01
Iteration 101 | Cost: 8.477667e-01
Iteration 102 | Cost: 8.477346e-01
Iteration 103 | Cost: 8.476203e-01
Iteration 104 | Cost: 8.475046e-01
Iteration 105 | Cost: 8.463492e-01
Iteration 106 | Cost: 8.455228e-01
Iteration 107 | Cost: 8.446019e-01
Iteration 108 | Cost: 8.434245e-01
Iteration 109 | Cost: 8.429515e-01
Iteration 110 | Cost: 8.427042e-01
Iteration 111 | Cost: 8.424669e-01
Iteration 112 | Cost: 8.424242e-01
Iteration 113 | Cost: 8.422920e-01
Iteration 114 | Cost: 8.422180e-01
Iteration 115 | Cost: 8.421498e-01
Iteration 116 | Cost: 8.420029e-01
Iteration 117 | Cost: 8.416026e-01
Iteration 118 | Cost: 8.414708e-01
Iteration 119 | Cost: 8.413894e-01
Iteration 120 | Cost: 8.352173e-01
Iteration 121 | Cost: 8.347014e-01
Iteration 122 | Cost: 8.321343e-01
Iteration 123 | Cost: 8.318239e-01
Iteration 124 | Cost: 8.295318e-01
Iteration 125 | Cost: 8.268272e-01
Iteration 126 | Cost: 8.256677e-01
Iteration 127 | Cost: 8.248947e-01
Iteration 128 | Cost: 8.248080e-01
Iteration 129 | Cost: 8.247438e-01
Iteration 130 | Cost: 8.247269e-01
Iteration 131 | Cost: 8.245701e-01
Iteration 132 | Cost: 8.244120e-01
Iteration 133 | Cost: 8.230264e-01
Iteration 134 | Cost: 8.223584e-01
Iteration 135 | Cost: 8.219328e-01
Iteration 136 | Cost: 8.210646e-01
Iteration 137 | Cost: 8.205255e-01
Iteration 138 | Cost: 8.137806e-01

Iteration 139 | Cost: 8.116976e-01
Iteration 140 | Cost: 8.114778e-01
Iteration 141 | Cost: 8.105850e-01
Iteration 142 | Cost: 8.103995e-01
Iteration 143 | Cost: 8.099261e-01
Iteration 144 | Cost: 8.098416e-01
Iteration 145 | Cost: 8.095563e-01
Iteration 146 | Cost: 8.087288e-01
Iteration 147 | Cost: 8.079132e-01
Iteration 148 | Cost: 8.077810e-01
Iteration 149 | Cost: 8.077468e-01
Iteration 150 | Cost: 8.075014e-01

C. Entrenamiento de SVM usando PCA en los datos:

A continuación se muestran el resultado y/o iteraciones del entrenamiento de SVMLIB aplicando PCA al dataset:

```
optimization finished, #iter = 221
nu = 0.695103
obj = -13.746947, rho = -1.000116
nSV = 440, nBSV = 440
Total nSV = 440
Accuracy = 65.2449% (413/633) (classification)
Accuracy = 67.7725% (143/211) (classification)
*
optimization finished, #iter = 225
nu = 0.695103
obj = -54.808644, rho = -1.002341
nSV = 442, nBSV = 439
Total nSV = 442
Accuracy = 65.2449% (413/633) (classification)
Accuracy = 67.7725% (143/211) (classification)
*
optimization finished, #iter = 227
nu = 0.695103
obj = -207.905765, rho = -1.017790
nSV = 442, nBSV = 438
Total nSV = 442
Accuracy = 65.2449% (413/633) (classification)
Accuracy = 68.2464% (144/211) (classification)
*
optimization finished, #iter = 251
nu = 0.515333
obj = -572.528327, rho = -1.833924
nSV = 336, nBSV = 318
Total nSV = 336
Accuracy = 83.2543% (527/633) (classification)
Accuracy = 85.3081% (180/211) (classification)
*
optimization finished, #iter = 506
nu = 0.353825
obj = -1538.893539, rho = -0.790392
nSV = 248, nBSV = 206
Total nSV = 248
Accuracy = 90.0474% (570/633) (classification)
Accuracy = 87.2038% (184/211) (classification)
..*.*
optimization finished, #iter = 2058
nu = 0.158079
```

obj = -2211.609849, rho = -0.929257
nSV = 230, nBSV = 56
Total nSV = 230
Accuracy = 97.6303% (618/633) (classification)
Accuracy = 84.3602% (178/211) (classification)
.*.*
optimization finished, #iter = 1435
nu = 0.018613
obj = -858.094419, rho = -0.716043
nSV = 374, nBSV = 3
Total nSV = 374
Accuracy = 99.842% (632/633) (classification)
Accuracy = 84.3602% (178/211) (classification)
.*.*
optimization finished, #iter = 1275
nu = 0.001942
obj = -314.649553, rho = -0.446043
nSV = 572, nBSV = 0
Total nSV = 572
Accuracy = 100% (633/633) (classification)
Accuracy = 82.4645% (174/211) (classification)
.*.*
optimization finished, #iter = 1364
nu = 0.000436
obj = -282.755069, rho = -0.313255
nSV = 627, nBSV = 0
Total nSV = 627
Accuracy = 100% (633/633) (classification)
Accuracy = 70.6161% (149/211) (classification)
..*
optimization finished, #iter = 1746
nu = 0.000109
obj = -283.141527, rho = -0.301450
nSV = 632, nBSV = 0
Total nSV = 632
Accuracy = 100% (633/633) (classification)
Accuracy = 67.7725% (143/211) (classification)
..*
optimization finished, #iter = 1647
nu = 0.000027
obj = -283.796255, rho = -0.304775
nSV = 633, nBSV = 0
Total nSV = 633
Accuracy = 100% (633/633) (classification)
Accuracy = 67.7725% (143/211) (classification)
C is: 8.000000, and y is: 0.007813
*
optimization finished, #iter = 506

nu = 0.353825
obj = -1538.893539, rho = -0.790392
nSV = 248, nBSV = 206
Total nSV = 248
.*
optimization finished, #iter = 310
nu = 0.344509
obj = -471.616848, rho = 1.288195
nSV = 93, nBSV = 58
Total nSV = 93
.
optimization finished, #iter = 285
nu = 0.325212
obj = -437.764491, rho = -1.207899
nSV = 83, nBSV = 54
Total nSV = 83
Train
Accuracy = 90.0474% (570/633) (classification)
Crosv
Accuracy = 91.9431% (194/211) (classification)
Test
Accuracy = 90.9953% (192/211) (classification)
C = 8.000000, and y = 0.007813

D. Entrenamiento de la Red Neuronal aplicando PCA a los datos:

A continuación se muestran el resultado y/o iteraciones del entrenamiento de la red neuronal aplicando PCA al dataset:

```
Iteration    1 | Cost: 1.320565e+00
Iteration    2 | Cost: 1.294817e+00
Iteration    3 | Cost: 1.292823e+00
Iteration    4 | Cost: 1.292347e+00
Iteration    5 | Cost: 1.290157e+00
Iteration    6 | Cost: 1.288659e+00
Iteration    7 | Cost: 1.284593e+00
Iteration    8 | Cost: 1.180808e+00
Iteration    9 | Cost: 1.177829e+00
Iteration   10 | Cost: 1.065105e+00
Iteration   11 | Cost: 1.008441e+00
Iteration   12 | Cost: 9.328612e-01
Iteration   13 | Cost: 9.041532e-01
Iteration   14 | Cost: 8.699311e-01
Iteration   15 | Cost: 8.350135e-01
Iteration   16 | Cost: 8.252438e-01
Iteration   17 | Cost: 8.201823e-01
Iteration   18 | Cost: 8.080596e-01
Iteration   19 | Cost: 8.010700e-01
Iteration   20 | Cost: 7.961571e-01
Iteration   21 | Cost: 7.915973e-01
Iteration   22 | Cost: 7.873179e-01
Iteration   23 | Cost: 7.862499e-01
Iteration   24 | Cost: 7.838702e-01
Iteration   25 | Cost: 7.824429e-01
Iteration   26 | Cost: 7.808636e-01
Iteration   27 | Cost: 7.775616e-01
Iteration   28 | Cost: 7.756729e-01
Iteration   29 | Cost: 7.730518e-01
Iteration   30 | Cost: 7.707552e-01
Iteration   31 | Cost: 7.698183e-01
Iteration   32 | Cost: 7.686310e-01
Iteration   33 | Cost: 7.681483e-01
Iteration   34 | Cost: 7.675371e-01
Iteration   35 | Cost: 7.664543e-01
Iteration   36 | Cost: 7.654752e-01
Iteration   37 | Cost: 7.632208e-01
Iteration   38 | Cost: 7.622895e-01
Iteration   39 | Cost: 7.609286e-01
Iteration   40 | Cost: 7.603074e-01
Iteration   41 | Cost: 7.599699e-01
Iteration   42 | Cost: 7.590483e-01
```

Iteration 43 | Cost: 7.587894e-01
Iteration 44 | Cost: 7.580826e-01
Iteration 45 | Cost: 7.572506e-01
Iteration 46 | Cost: 7.565515e-01
Iteration 47 | Cost: 7.555103e-01
Iteration 48 | Cost: 7.550871e-01
Iteration 49 | Cost: 7.546712e-01
Iteration 50 | Cost: 7.537382e-01
Iteration 51 | Cost: 7.533012e-01
Iteration 52 | Cost: 7.526808e-01
Iteration 53 | Cost: 7.525818e-01
Iteration 54 | Cost: 7.522207e-01
Iteration 55 | Cost: 7.519491e-01
Iteration 56 | Cost: 7.517851e-01
Iteration 57 | Cost: 7.514860e-01
Iteration 58 | Cost: 7.510882e-01
Iteration 59 | Cost: 7.508156e-01
Iteration 60 | Cost: 7.505869e-01
Iteration 61 | Cost: 7.503069e-01
Iteration 62 | Cost: 7.501960e-01
Iteration 63 | Cost: 7.500544e-01
Iteration 64 | Cost: 7.499430e-01
Iteration 65 | Cost: 7.498077e-01
Iteration 66 | Cost: 7.496461e-01
Iteration 67 | Cost: 7.496130e-01
Iteration 68 | Cost: 7.495392e-01
Iteration 69 | Cost: 7.494762e-01
Iteration 70 | Cost: 7.493964e-01
Iteration 71 | Cost: 7.491745e-01
Iteration 72 | Cost: 7.488596e-01
Iteration 73 | Cost: 7.486251e-01
Iteration 74 | Cost: 7.485108e-01
Iteration 75 | Cost: 7.484205e-01
Iteration 76 | Cost: 7.483032e-01
Iteration 77 | Cost: 7.481959e-01
Iteration 78 | Cost: 7.480595e-01
Iteration 79 | Cost: 7.480092e-01
Iteration 80 | Cost: 7.479714e-01
Iteration 81 | Cost: 7.479267e-01
Iteration 82 | Cost: 7.478674e-01
Iteration 83 | Cost: 7.478054e-01
Iteration 84 | Cost: 7.477755e-01
Iteration 85 | Cost: 7.477409e-01
Iteration 86 | Cost: 7.476618e-01
Iteration 87 | Cost: 7.475563e-01
Iteration 88 | Cost: 7.474476e-01
Iteration 89 | Cost: 7.473887e-01
Iteration 90 | Cost: 7.473783e-01

Iteration 91 | Cost: 7.473341e-01
Iteration 92 | Cost: 7.473217e-01
Iteration 93 | Cost: 7.473096e-01
Iteration 94 | Cost: 7.472929e-01
Iteration 95 | Cost: 7.472543e-01
Iteration 96 | Cost: 7.472177e-01
Iteration 97 | Cost: 7.471100e-01
Iteration 98 | Cost: 7.469511e-01
Iteration 99 | Cost: 7.469273e-01
Iteration 100 | Cost: 7.469146e-01
Iteration 101 | Cost: 7.468992e-01
Iteration 102 | Cost: 7.468806e-01
Iteration 103 | Cost: 7.468642e-01
Iteration 104 | Cost: 7.468067e-01
Iteration 105 | Cost: 7.467912e-01
Iteration 106 | Cost: 7.467736e-01
Iteration 107 | Cost: 7.467104e-01
Iteration 108 | Cost: 7.466907e-01
Iteration 109 | Cost: 7.466760e-01
Iteration 110 | Cost: 7.466534e-01
Iteration 111 | Cost: 7.466366e-01
Iteration 112 | Cost: 7.466216e-01
Iteration 113 | Cost: 7.466152e-01
Iteration 114 | Cost: 7.465993e-01
Iteration 115 | Cost: 7.465605e-01
Iteration 116 | Cost: 7.464586e-01
Iteration 117 | Cost: 7.463394e-01
Iteration 118 | Cost: 7.462316e-01
Iteration 119 | Cost: 7.461903e-01
Iteration 120 | Cost: 7.460871e-01
Iteration 121 | Cost: 7.460414e-01
Iteration 122 | Cost: 7.460177e-01
Iteration 123 | Cost: 7.459657e-01
Iteration 124 | Cost: 7.459089e-01
Iteration 125 | Cost: 7.458844e-01
Iteration 126 | Cost: 7.458494e-01
Iteration 127 | Cost: 7.458265e-01
Iteration 128 | Cost: 7.458110e-01
Iteration 129 | Cost: 7.457847e-01
Iteration 130 | Cost: 7.457304e-01
Iteration 131 | Cost: 7.457078e-01
Iteration 132 | Cost: 7.456524e-01
Iteration 133 | Cost: 7.455968e-01
Iteration 134 | Cost: 7.455315e-01
Iteration 135 | Cost: 7.454131e-01
Iteration 136 | Cost: 7.453489e-01
Iteration 137 | Cost: 7.451636e-01
Iteration 138 | Cost: 7.450404e-01

Iteration 139 | Cost: 7.449369e-01
Iteration 140 | Cost: 7.446765e-01
Iteration 141 | Cost: 7.445811e-01
Iteration 142 | Cost: 7.444376e-01
Iteration 143 | Cost: 7.443985e-01
Iteration 144 | Cost: 7.443119e-01
Iteration 145 | Cost: 7.440753e-01
Iteration 146 | Cost: 7.438735e-01
Iteration 147 | Cost: 7.437388e-01
Iteration 148 | Cost: 7.435905e-01
Iteration 149 | Cost: 7.435086e-01
Iteration 150 | Cost: 7.434114e-01

E. Curvas ROC de todos los algoritmos:

A continuación se muestran las curvas ROC para todos los algoritmos que han sido analizados.

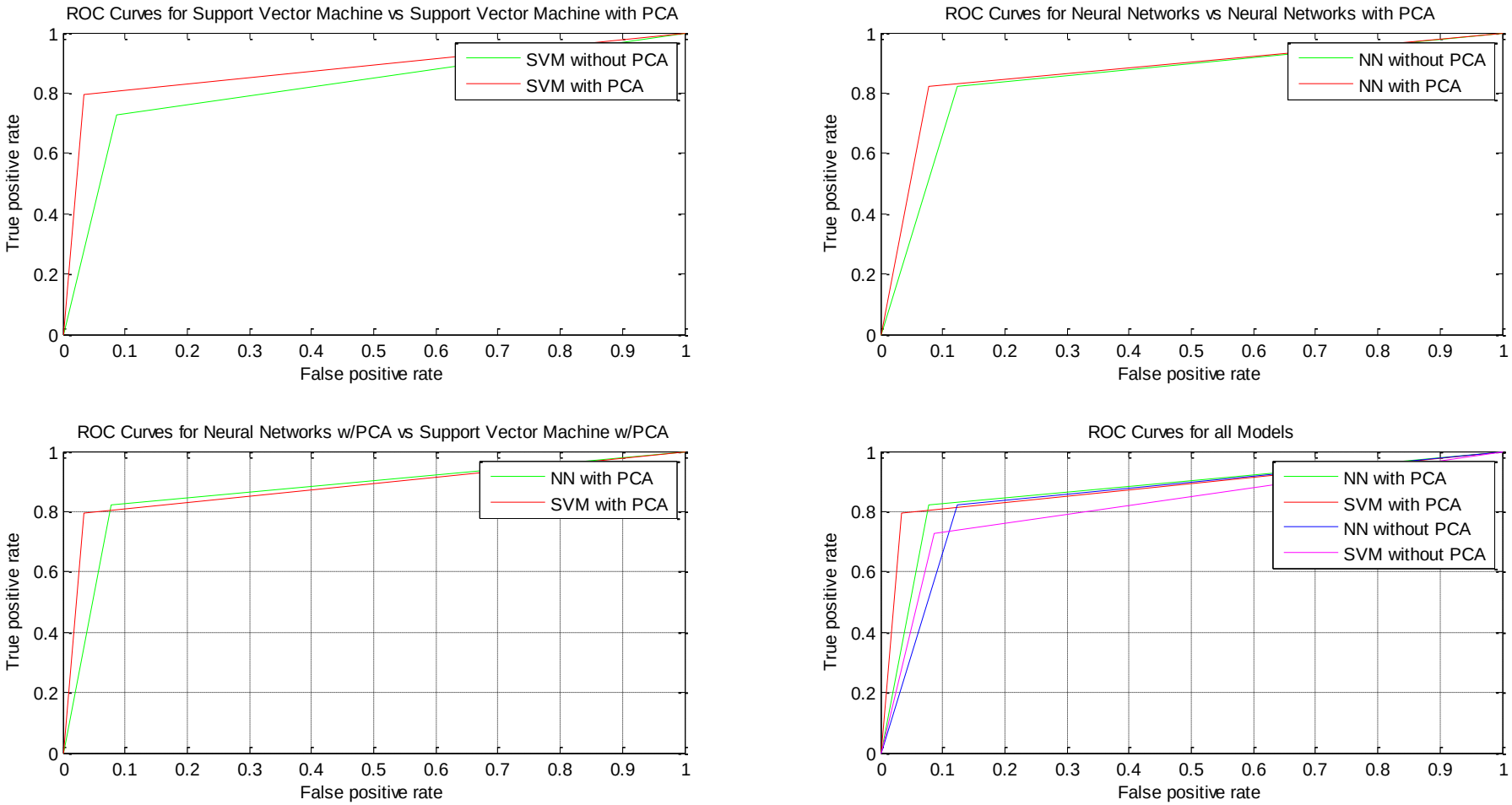


Figura N° 38: Curva ROC para todos los algoritmos

F. Pseudo código:

A continuación se muestra una lista de las funciones que se usaron para poder llevar a cabo la implementación de los algoritmos SVM, MLNN y PCA.

1) *norm:*

Calcula la media y desviación estándar del data set.

procedure *norm* (X)

$\langle m, n\rangle := Dim(X)$

$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$

$x_j^{(i)} := x_j^{(i)} - \mu_j$

$\sigma_j^2 = \frac{1}{m}\sum_{i}\left(x_j^{(i)}\right)^2$

$x_j^{(i)} := x_j^{(i)}/\sigma_j$

$X' := \langle x^{(1)}, \dots, x^{(n)}\rangle$

return X'

2) *projectDim*:

Proyecta el data set sobre un espacio dimensional *k*

procedure *projectDim* $(\langle X, U, k\rangle)$

$\langle m, n\rangle := Dim(X)$

for $i := 0$ ***to*** m

$Z(i, 1 \dots k) := U(1 \dots m, 1 \dots k)^T * X(i, 1 \dots n)^T$

return Z

3) *runPCA:*

Implementacion del algoritmo PCA

---

procedure *runPCA*(X, y)

Split (X, y) into $\langle (X_{tn}, y_{tn}'), (X_{cv}, y_{cv}'), (X_{ts}, y_{ts}') \rangle$

$X_{tn} := norm(X_{tn})$

$X_{cv} := norm(X_{cv})$

$X_{ts} := norm(X_{ts})$

Obtain the covariance matrix $\Sigma := \frac{1}{m} X_{tn}^T X_{tn}$

$\langle U, S, V \rangle := SVD(\Sigma)$

$\langle m, n \rangle := Dim(S)$

for $i := 1$ ***to*** m

$$VarLost := 1 - \frac{\sum_{i=1}^{k} S_{(i,i)}}{\sum_{i=1}^{n} S_{(i,i)}}$$

$$VarRet := \frac{\sum_{i=1}^{k} S_{(i,i)}}{\sum_{i=1}^{n} S_{(i,i)}}$$

if $(VarLost \leq 0.01 \wedge VarRet \geq 0.99)$

$k := i$

break

$U := U(1 \ldots k)$

$X_{tn}' := projectDim(X_{tn}', U, k)$

$X_{cv}' := projectDim(X_{cv}', U, k)$

$X_{cv}' := projectDim(X_{cv}', U, k)$

return $\langle (X_{tn}', y_{tn}'), (X_{cv}', y_{cv}'), (X_{ts}', y_{ts}') \rangle$

---

4) *searchSVMP:*

Busca los parámetros C, γ óptimos usando como medida el menor error cuadrático.

---

procedure *searchSVMP*$(C, \gamma, \langle (X_{tn}', y_{tn}'), (X_{cv}', y_{cv}') \rangle)$

$Error := \langle \ldots \rangle$

for $i := 1$ to $dim(C, \gamma)$

$model := svmtrain(y_{tn}', X_{tn}', C^{(i)}, \gamma^{(i)})$

$pred_{cv} := svmpredict(y_{cv}', X_{cv}', model)$

$Error^{(i)} := MSE(X_{cv}', pred_{cv}, y_{cv}')$

$[\sim, k] := \min(Error)$

$C' := 2^{C^{(k)}}$

$\gamma' := 2^{C^{(k)}}$

return $\langle C', \gamma' \rangle$

---

5) *runSVMLIB:*

Implementación del algoritmo SVM usando la librería LIBSVM (Chang & Lin, 2011).

procedure *runSVMLIB*$(\langle (X_{tn}', y_{tn}'), (X_{cv}', y_{cv}'), (X_{ts}', y_{ts}') \rangle)$

$C := \langle -5, \dots, 15 \rangle$
$\gamma := \langle -15, \dots, 5 \rangle$
$\langle C', \gamma' \rangle := searchSVMP(C, \gamma, \langle (X_{tn}', y_{tn}'), (X_{cv}', y_{cv}') \rangle)$
$model := svmtrain(y_{tn}', X_{tn}', C^{(i)}, \gamma^{(i)})$
$pred_{tn} := svmpredict(y_{tn}', X_{tn}', model)$
$pred_{cv} := svmpredict(y_{cv}', X_{cv}', model)$
$pred_{ts} := svmpredict(y_{ts}', X_{ts}', model)$
return $\langle pred_{tn}, pred_{cv}, pred_{ts} \rangle$

6) *searchLambda:*

Busca el parámetro λ óptimo usando como medida el menor error cuadrático.

procedure *searchLambda*$(\lambda, \langle (X_{tn}', y_{tn}'), (X_{cv}', y_{cv}') \rangle, \Theta)$

$Error := \langle \dots \rangle$
for $i := 1$ to $dim(\lambda)$
    $model := trainNetwork(\Theta, \lambda^{(i)} X_{tn}', y_{tn}')$
    $pred_{cv} := predNetwork(y_{cv}', X_{cv}', model)$
$Error^{(i)} := MSE(X_{cv}', pred_{cv}, y_{cv}')$
$[\sim, k] := \min(Error)$
return $\lambda^{(k)}$

7) *runNN:*

Implementación del algoritmo MLNN.

procedure *runNN* $(\langle (X_{tn}', y_{tn}'), (X_{cv}', y_{cv}'), (X_{ts}', y_{ts}') \rangle)$

$\Theta := \langle \theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n)} \rangle$
$\lambda := \langle 0, \dots, 10 \rangle$
$\lambda' := searchLambda(\lambda, \langle (X_{tn}', y_{tn}'), (X_{cv}', y_{cv}') \rangle, \Theta)$
$model := trainNetwork(\Theta, \lambda' X_{tn}', y_{tn}')$
$pred_{tn} := predNetwork(X_{tn}', y_{tn}', model)$
$pred_{cv} := predNetwork(X_{cv}', y_{cv}', model)$
$pred_{ts} := predNetwork(X_{cv}', y_{cv}', model)$
return $\langle pred_{tn}, pred_{cv}, pred_{ts} \rangle$

G. Diagrama de la metodología:

A continuación se muestra el diagrama de la implementación de la metodología que se aplicó.

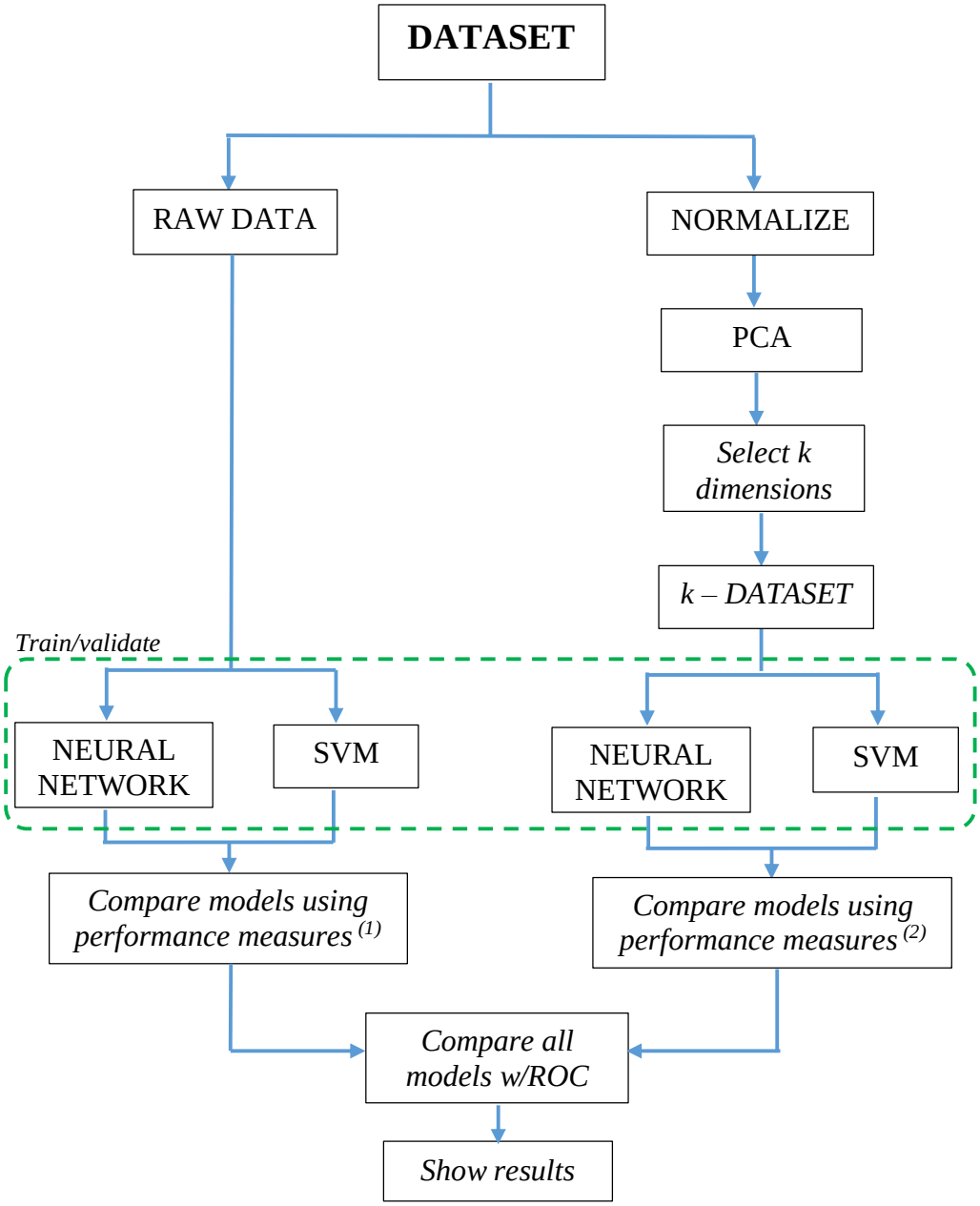


Figura N° 39: Diagrama de la Metodología aplicada