Implementación de red neuronal y comparación de técnicas de reducción de dimensionalidad

Luis Herasme
Área de ingenierías
Instituto Tecnológico de Santo Domingo
Santo Domingo, República Dominicana
1088668@est.intec.edu.do

***Abstract*—En este trabajo se presentan la implementación de una red neuronal de una capa oculta con una sola neurona en la salida para la clasificación de pacientes, diferenciando entre pacientes con enfermedad cardiaca y pacientes sanos. Además, se realizan diferentes técnicas de reducción de dimensionalidad y se comparan con redes neuronales, es decir, se trata de reproducir el funcionamiento de esas técnicas de reducción de dimensionalidad con redes neuronales diseñadas para realizar la misma tarea.**

***Index Terms*—Redes neuronales, Algoritmos de aprendizaje, Principal Component Analysis, Linear Discriminant Analysis, Autoencoders**

I. Introducción

Este documento se podría dividir en dos partes, la primera trata sobre la implementación de una red neuronal y la segunda sobre la implementación de técnicas de reducción de dimensionalidad y el uso redes neuronales para realizar la misma tarea.

En la primera parte se presenta una red neuronal que contiene una capa oculta y una sola neurona en su salida, se evalúa su desempeño a partir de su capacidad de clasificar un paciente correctamente entre paciente con enfermedad cardiaca y paciente sano. Para evaluar la red lo primero que se hizo fue probar la red con todos los features disponibles, es decir: edad, altura, peso, sexo, presión arterial sistólica, presión arterial diastólica, colesterol, glucosa, tabaquismo, ingesta de alcohol, actividad física y luego sé probo con solo 3 features para poder compararla con el algoritmo anteriormente implementado (Bayes multivariable), los features utilizados para este segundo experimento fueron: edad, peso y estatura.

Luego en la segunda parte se implementaron dos técnicas de reducción de dimensionalidad, Análisis de componente principal (PCA) y Análisis de discriminante lineal (LDA), estas técnicas de reducción de dimensionalidad fueron comparadas con redes neuronales diseñadas para realizar la misma tarea, para PCA se utilizó un Autoencoder con la herramienta de MATLAB nntool, y para LDA con la misma herramienta se diseñó una red neuronal para realizar clasificación (Ambas redes serán explicadas con mayor detalle más adelante).

A. Objetivos

El objetivo de la implementación de la red neuronal es evaluar el desempeño general de la red, sin embargo también se realiza una comparación con el algoritmo anteriormente implementado (Bayes Multivariable).

Para la segunda parte el objetivo es comparar el desempeño de las técnicas tradicionales de reducción de dimensionalidad con técnicas equivalentes utilizando redes neuronales.

B. Origen y separación de los datos

El conjunto de datos utilizado para llevar a cabo los experimentos se obtuvo de Kaggle [3]. Este conjunto de datos cuenta con 70,000 entradas u observaciones, que fueron reorganizadas de forma aleatoria como paso previo a la separación entre conjunto de entrenamiento y conjunto de validación. 50,000 entradas fueron utilizadas para el entrenamiento (aprox. 70%), 10,000 fueron utilizadas para la validación y otras 10,000 como conjunto de pruebas.

II. Descripción de los algoritmos

A. Red neuronal

Antes de ir al algoritmo primero veamos como son obtenidas las ecuaciones que contiene. Lo primero que debemos tener presente es que estamos haciendo una clasificación de dos clases por lo tanto podemos asumir que sigue una distribución de Bernoulli:

$$r^t|x^t \sim \text{Bernoulli}(y^t)$$

$$P(X = r^t) = (y^t)^{r^t}(1 - y^t)^{1-r^t}$$

Ahora lo que queremos hacer maximizar nuestra verosimilitud variando los parámetros $\boldsymbol{W}$ y $\boldsymbol{v}$:

$$l(\boldsymbol{W}, \boldsymbol{v}|\mathcal{X}) = \prod_t (y^t)^{r^t}(1 - y^t)^{1-r^t}$$

Maximizar dicha función es equivalente a minimizar una función de error descrita de la siguiente forma $E = -\ln l$:

$$E\left(\boldsymbol{W}, \boldsymbol{v}|\mathcal{X}\right) = -\ln\left[l(\boldsymbol{W}, \boldsymbol{v}|\mathcal{X})\right]$$

Cuando aplicamos el logaritmo a la verosimilitud (Log-Likelihood) y expandimos, nos quedamos con la siguiente expresión:

$$E\left(\boldsymbol{W}, \boldsymbol{v}|\mathcal{X}\right) = -\sum_t r^t \ln y^t + \left(1 - r^t\right) \ln\left(1 - y^t\right)$$

Para minimizar hay que tener cuenta que y^t depende de muchas funciones donde se encuentran los parámetros $\boldsymbol{W}$ y $\boldsymbol{v}$ de nuestra red:

$$\begin{aligned} y^t &= \text{sigmoid}\left(\sum_{h=0}^{H} v_h z_h^t\right) = \text{sigmoid}\left(\boldsymbol{v}^T \boldsymbol{z}^t\right) \\ &\qquad \text{con } z_0 = 1 \\ z_h^t &= \text{sigmoid}\left(\sum_{j=0}^{H} w_{hj} x_j^t\right) = \text{sigmoid}\left(\boldsymbol{w}_h^T \boldsymbol{x}^t\right) \\ &\qquad \text{con } x_0 = 1 \end{aligned} \tag{1}$$

Donde la funciona sigmoide (Función logística) está definida de la siguiente forma:

$$\text{sigmoid}(x) = \sigma(x) = \frac{1}{1+e^{-x}};\ \frac{d\sigma}{dx} = \sigma(1-\sigma)$$

Para calcular el gradiente necesitamos utilizar la regla de la cadena, ya que v_h y w_{hj} están dentro de varias funciones anidadas:

$$\frac{\partial E}{\partial v_h} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial v_h}$$

Podemos calcular cada una de estas derivadas parciales por separado y luego multiplicarlas de la siguiente forma:

$$\begin{aligned} \frac{\partial E^t}{\partial y^t} &= \frac{\partial}{\partial y^t}\left(r^t \ln y^t + \left(1-r^t\right)\ln\left(1-y^t\right)\right) \\ &= \frac{r^t}{y^t} - \frac{1-r^t}{1-y^t} \\ \frac{\partial y^t}{\partial v_h} &= \frac{\partial}{\partial v_h}\left(\text{sigmoid}\left(\sum_{i=0}^{H} v_i z_i^t\right)\right) = \\ y^t(1-y^t)\frac{\partial}{\partial v_h}\left(\sum_{i=0}^{H} v_i z_i^t\right) &= \begin{cases} y^t(1-y^t)z_i^t & i = h \\ 0 & i \neq h \end{cases} \\ \frac{\partial y^t}{\partial v_h} &= y^t(1-y^t)z_h^t \end{aligned} \tag{2}$$

Por lo tanto combinando las derivadas parciales nos quedaremos con la siguiente expresión:

$$\frac{\partial E}{\partial v_h} = -\sum_t \underbrace{\left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}\right)}_{\frac{\partial E^t}{\partial y^t}} \overbrace{y^t(1-y^t)z_h^t}^{\frac{\partial y^t}{\partial v_h}}$$

Esta expresión aún se puede simplificarse bastante de la siguiente forma:

$$\begin{aligned} \frac{\partial E}{\partial v_h} &= -\sum_t \left(\frac{r^t(1-y^t) - y^t(1-r^t)}{\cancel{y^t(1-y^t)}}\right)\cancel{y^t(1-y^t)}z_h^t \\ \frac{\partial E}{\partial v_h} &= -\sum_t \left(r^t - \cancel{y^t r^t} - y^t + \cancel{y^t r^t}\right) z_h^t \\ \frac{\partial E}{\partial v_h} &= -\sum_t \left(r^t - y^t\right) z_h^t \end{aligned} \tag{3}$$

Ya que tenemos el cambio del error con respecto a v_h podemos multiplicar este cambio por una constate $\eta = 0.0003$, cuyo valor fue establecido tras prueba y error.

$$\Delta v_h = -\eta\frac{\partial E}{\partial v_h} = \eta\sum_t \left(r^t - y^t\right) z_h^t \tag{4}$$

Ahora para calcular w_{hj} vamos a tener que calcular más derivadas parciales, ya que hay más funciones anidadas para alcanzar a w_{hj} desde y^t, estas son:

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z_h}\frac{\partial z_h}{\partial w_{hj}}$$

La primera derivada parcial $\frac{\partial E^t}{\partial y^t}$ se mantiene, sin embargo ahora debemos derivar a y^t con respecto a z_h para luego así poder derivar a z_h con respecto a w_{hj}. Podemos encontrar estas derivadas de la siguiente forma:

$$\begin{aligned} \frac{\partial y^t}{\partial z_h^t} &= \frac{\partial}{\partial z_h^t}\left(\text{sigmoid}\left(\sum_{i=0}^{H} v_i z_i^t\right)\right) = \\ y^t(1-y^t)\frac{\partial}{\partial z_h^t}\left(\sum_{i=0}^{H} v_i z_i^t\right) &= \begin{cases} y^t(1-y^t)v_h^t & i = h \\ 0 & i \neq h \end{cases} \\ \frac{\partial y^t}{\partial z_h^t} &= y^t(1-y^t)v_h^t \\ \frac{\partial z_h^t}{\partial w_{hj}} &= \frac{\partial}{\partial w_{hj}}\left(\text{sigmoid}\left(\sum_{j=0}^{H} w_{hj} x_j^t\right)\right) = \\ z_h^t(1-z_h^t)\frac{\partial}{\partial w_{hj}}\left(\sum_{j=0}^{H} w_{hj} x_j^t\right) &= \begin{cases} z_h^t(1-z_h^t)x_j^t & j = h \\ 0 & j \neq h \end{cases} \\ \frac{\partial z_h^t}{\partial w_{hj}} &= z_h^t(1-z_h^t)x_j^t \end{aligned} \tag{5}$$

Entonces al combinar las derivadas parciales nos queda la siguiente expresión:

$$\frac{\partial E}{\partial w_{hj}} = -\sum_t \underbrace{\left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}\right)}_{\frac{\partial E^t}{\partial y^t}} \overbrace{y^t(1-y^t)v_h^t}^{\frac{\partial y^t}{\partial z_h^t}} \underbrace{z_h^t(1-z_h^t)x_j^t}_{\frac{\partial z_h^t}{\partial w_{hj}}}$$

Esta expresión se puede simplificar de forma similar a la expresión en la ecuación de $\frac{\partial E}{\partial v_h}$, por lo tanto $\frac{\partial E}{\partial w_{hj}}$ nos queda de la siguiente forma:

$$\frac{\partial E}{\partial w_{hj}} = -\sum_t (r^t - y^t)v_h^t z_h^t(1-z_h^t)x_j^t$$

Entonces añadiendo la tasa de aprendizaje η, el cambio de w_{hj} nos queda como:

$$\Delta w_{hj} = -\eta\frac{\partial E}{\partial w_{hj}} = \eta\sum_t \left(r^t - y^t\right) v_h^t z_h^t(1-z_h)x_j^t$$

Otro parámetro que nos falta especificar es la cantidad de neuronas en la capa oculta, donde utilizamos $H = 10$, en ambos experimentos, es decir el experimento realizado con todos los features y el realizado con los features de Bayes Multivariable.

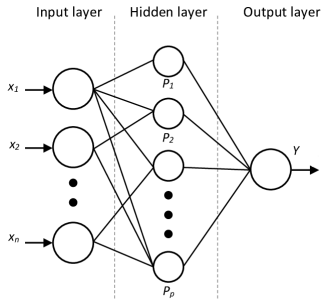

Fig. 1. Modelo general de una red neuronal de una capa oculta y una sola neurona en la salida [4].

Observando la figura 1, $\boldsymbol{W}$ que es la matriz que nos lleva de la capa de entrada a la capa oculta, tendrá dimensiones $(d + 1 \times H)$ y $\boldsymbol{v}$ que es la matriz que nos lleva de la capa oculta a la capa de salida tendrá dimensiones $(H + 1 \times 1)$.

Ahora tenemos todo lo necesario para presentar como el algoritmo utiliza estos gradientes para actualizar estas matrices y minimizar el error:

Algorithm 1: Algoritmo de backpropagation para entrenar un una red neuronal con una capa oculta y con una neurona en la salida.

Inicializar todo v_h y w_{hj} a rand(-0.01, 0.01);
repeat
  forall $(\boldsymbol{x}^t, r^t) \in \mathcal{X}$, *en un orden aleatorio* **do**
    $\boldsymbol{z}^t \leftarrow \text{sigmoid}(\boldsymbol{w}_h^t \boldsymbol{x}^t)$;
    $y^t \leftarrow \text{sigmoid}(\boldsymbol{v}^T \boldsymbol{z}^t)$;
    $\Delta \boldsymbol{v} \leftarrow \eta(r^t - y^t)\boldsymbol{z}$;
    for $h = 1, .., H$ **do**
      $\Delta \boldsymbol{w}_h \leftarrow \eta(r^t - y^t) v_h z_h^t (1 - z_h^t) \boldsymbol{x}^t$;
    end
    $\boldsymbol{v} \leftarrow \boldsymbol{v} + \Delta \boldsymbol{v}$;
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta \boldsymbol{w}$;
  end
until *convergencia*;

Para detener el algoritmo se observó el cambio de error porcentual en las últimas 5 épocas, si es menor a un 0.02%, entonces se asume que hubo convergencia. Para calcular dicho error primero se debe realizar una predicción, estas fueron realizadas de la siguiente forma:

$$\begin{aligned} \text{predecir}(x^t | \boldsymbol{W}, \boldsymbol{v}) &= \text{sigmoid}(\boldsymbol{v}^T \boldsymbol{z}^t), \\ \text{Donde, } \boldsymbol{z}^t &= \text{sigmoid}(\boldsymbol{W}^T \boldsymbol{x}^t) \end{aligned} \tag{6}$$

$$\hat{y} = \begin{cases} 1 & \text{predecir}(x^t | \boldsymbol{W}, \boldsymbol{v}) > 0.5 \\ 0 & \text{predecir}(x^t | \boldsymbol{W}, \boldsymbol{v}) \leq 0.5 \end{cases}$$

Entonces se calcula el error porcentual observando cuantas predicciones incorrectas fueron realizadas, sobre el total de observaciones de la siguiente forma:

$$\%\ \text{Error} = \frac{1}{N} \sum_{t=1}^{N} \boldsymbol{I}\left(\hat{y} \neq r^t\right)$$

Donde $\boldsymbol{I}$ es la función indicadora que resulta en 1 si su argumento es verdadero y 0 si su argumento es falso. La expresión nos dará el porcentaje de aciertos de nuestro algoritmo. También se utilizó el error cuadrático promedio, descrito de la siguiente forma:

$$\text{MSE} = \frac{1}{N} \sum_{t=1}^{N} \left[r^t - \text{predecir}(x^t | \boldsymbol{W}, \boldsymbol{v})\right]^2$$

Posteriormente en la sección de resultados, se presentará la curva de aprendizaje donde se observa el error cuadrático promedio y el error porcentual.

B. Análisis de componente principal

PCA es un método de reducción de dimensionalidad no supervisado y de proyección, por lo tanto la idea es encontrar una matriz de transformación $\boldsymbol{W}$ que nos convierta de una dimensionalidad d (cantidad de features original) a una dimensionalidad k donde $k < d$.

$$z = W^T x$$

La idea de PCA es mantener la máxima cantidad de varianza. Se puede demostrar que si $z_1 = w_1^T x$ donde $Cov(x) = \Sigma$, entonces:

$$Var(z_1) = w_1^t \Sigma w_1$$

La idea es encontrar w_1 de forma tal que se maximice la varianza $Var(z_1)$. Como maximizar la varianza $Var(z_1)$ es lo mismo que maximizar $w_1^t \Sigma w_1$, ya que, $Var(z_1) = w_1^t \Sigma w_1$, podemos maximizar $w_1^t \Sigma w_1$ y debemos añadir un término $\alpha(w_1^t w_1 - 1)$ para que w_1 no vaya a infinito, ya que esto también maximiza la varianza.

$$\max_{w_1} w_1^t \Sigma w_1 - \alpha(w_1^t w_1 - 1)$$

Aplicando derivada respecto a w_1, e igualando a cero:

$$\begin{aligned} 2\Sigma w_1 - 2\alpha w_1 &= 0 \\ \Sigma w_1 &= \alpha w_1 \end{aligned} \tag{7}$$

Evidentemente esto solo se cumple si w_1 es un vector propio de Σ y α el valor propio correspondiente.

Sabemos que α es un valor propio, este debe ser el máximo si se quiere conservar la varianza máxima, lo podemos comprobar al multiplicar ambos lados por w_1^t, tenemos:

$$w_1^t \Sigma w_1 = Var(z_1) = w_1^t \alpha w_1$$

Entonces $Var(z_1)$ será máximo cuando α sea máximo. Esta explicación se puede expandir para crear una matriz W, que utilizando la misma idea, incluya los vectores propios con los mayores valores propios.

$$z = W^T(x - m)$$

Por lo tanto W tiene k columnas las que son los k vectores propios de Σ, con mayores valores propios.

Si deseamos conocer cuanta varianza preservamos podemos utilizar la proporción de varianza, que se define de la siguiente forma: cuando los valores propios λ_i están organizados en orden descendiente, la proporción de varianza preservada si utilizamos k componentes principales es:

$$\text{PoV} = \frac{\lambda_1 + \lambda_2 + \ldots \lambda_k}{\lambda_1 + \lambda_2 + \ldots \lambda_k + \cdots + \lambda_d}$$

En la sección de resultados luego de aplicar PCA, se mostrara una gráfica con el PoV.

C. Análisis discriminante lineal

LDA es un método de proyección como PCA, sin embargo es un método supervisado, es decir se utilizan las etiquetas de las observaciones para determinar $\boldsymbol{W}$.

$$z = W^t x$$

Para encontrar a W, debemos primero determinar la matriz de dispersión entre clases S_B y la matriz de dispersión dentro de la clase S_W. Para encontrar a S_W realizamos los siguientes pasos:

$$S_i = \sum_t r_i^t (\boldsymbol{x}^t - \boldsymbol{m}_i)(\boldsymbol{x}^t - \boldsymbol{m}_i)^T$$

$$S_W = \sum_{i=1}^{K} \boldsymbol{S}_i$$

Para encontrar a S_B o la dispersión entre clases realizamos lo siguiente:

$$\boldsymbol{m} = \frac{1}{K} \sum_{i=1}^{K} \boldsymbol{m}_i$$

$$S_B = \sum_t N_i (\boldsymbol{m} - \boldsymbol{m}_i)(\boldsymbol{m} - \boldsymbol{m}_i)^T$$

Luego de aplicarle la proyección sobre W a los datos, matriz de dispersión entre clases será $W^T S_B W$ y la matriz de dispersión dentro de la clase será $W^T S_W W$. Como queremos la dispersión entre clases grande y la dispersión dentro de la clase pequeña, podemos determinar W como la matriz que maximiza:

$$J(W) = \frac{|W^T S_B W|}{|W^T S_W W|}$$

Como se puede ver $J(W)$ aumenta a medida que la dispersión dentro de la clase S_W disminuye y la dispersión entre clases S_B aumenta. La solución a este problema de optimización es $S_W^{-1} S_B$. Entonces W está compuesto por los vectores propios $S_W^{-1} S_B$ cuyo valor propio correspondiente es distinto a cero.

$$W = \underset{\text{VectoresPropios} \neq 0}{\text{ValoresPropios}} \left(S_W^{-1} S_B \right)$$

Es importante recalcar que LDA reduce como máximo a K-1 features, donde K es la cantidad de clases.

III. Descripción de los experimentos

A. Red neuronal

Lo primero que se hizo fue probar esta red con todos los features disponibles exceptuando "cardio", es decir: edad, altura, peso, sexo, presión arterial sistólica, presión arterial diastólica, colesterol, glucosa, tabaquismo, ingesta de alcohol, actividad física y luego sé probo con solo 3 features para realizar la comparación con el algoritmo anteriormente implementado (Bayes multivariable), los features utilizados para este segundo experimento fueron: edad, peso y estatura.

B. Análisis discriminante lineal

En este experimento cambiamos el objetivo, ya no es "cardio" sino "nivel de colesterol" y como features se aplicaron todos a excepción de "cardio" y "nivel de colesterol".

Luego de aplicar LDA, se redujo la cantidad de features a dos. Entonces para emular el comportamiento de LDA se diseñó una red neuronal que tiene una capa oculta con dos neuronas y que tiene como objetivo el mismo que el usado con LDA es decir "nivel de colesterol".

El algoritmo utilizado para comparar LDA con NN_{LDA} (el LDA realizado por la red neuronal) es regresión logística de múltiples clases. Debido a que este algoritmo realiza softmax en su salida que es una generalización de la función logística a múltiples dimensiones; en la salida de la red neuronal coloque una función de activación logística.

Esta red neuronal se ve de la siguiente forma:

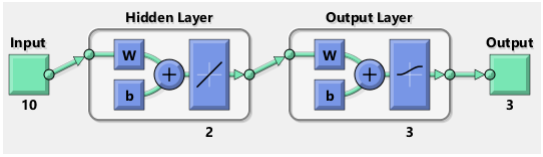

Fig. 2. Red neuronal para replicar LDA.

La salida de la red neuronal es de 3 neuronas, ya que para poder hacer la regresión logística de múltiples clases tuve que modificar la etiqueta "nivel de colesterol", la cual estaba descrita como [1], [2] y [3], para indicar nivel de colesterol normal, alto y muy alto, sin embargo se cambió a [0 0 1], [0 1 0] y [1 0 0], para poder aplicar el algoritmo.

El algoritmo para aplicar la regresión logística de múltiples clases fue tomado de [1], y es el siguiente:

For $i = 1, \ldots, K$
  For $j = 0, \ldots, d$
    $w_{ij} \leftarrow \text{rand}(-0.01, 0.01)$
Repeat
  For $i = 1, \ldots, K$
    For $j = 0, \ldots, d$
      $\Delta w_{ij} \leftarrow 0$
  For $t = 1, \ldots, N$
    For $i = 1, \ldots, K$
      $o_i \leftarrow 0$
      For $j = 0, \ldots, d$
        $o_i \leftarrow o_i + w_{ij} x_j^t$
    For $i = 1, \ldots, K$
      $y_i \leftarrow \exp(o_i) / \sum_k \exp(o_k)$
    For $i = 1, \ldots, K$
      For $j = 0, \ldots, d$
        $\Delta w_{ij} \leftarrow \Delta w_{ij} + (r_i^t - y_i) x_j^t$
  For $i = 1, \ldots, K$
    For $j = 0, \ldots, d$
      $w_{ij} \leftarrow w_{ij} + \eta \Delta w_{ij}$
Until convergence

Fig. 3. Algoritmo de discriminación logística implementando descenso de gradiente para el caso con clases $K > 2$. [1]

Luego de aplicar LDA y el LDA con la red neuronal llamémosle (NN_{LDA}), se compararon los dos conjuntos de datos reducidos utilizando la regresión logística de múltiples clases, la idea es evaluar la capacidad de estos conjuntos de datos reducidos de representar la clase "nivel de colesterol".

C. Análisis de componente principal

Antes de aplicar PCA, se estandarizaron los datos, por dos razones principalmente:

- Features como la edad (en días) van a tener una varianza muy alta y no necesariamente son más importantes que otros features como el nivel de glucosa (1, 2, 3), género (1, 2) o la presión arterial sistólica y diastólica.
- Una vez haga la reducción de dimensionalidad y luego el aumento de dimensionalidad será más fácil calcular las distancias, ya que se puede utilizar la misma función de distancia para todos los features.

Luego de aplicar PCA sobre todas las características exceptuando a "cardio", se tomaron los vectores propios necesarios para el 90% de la varianza, esto resulto en que PCA redujo finalmente a 9 features.

Se comparó PCA con usar un autoencoder al cual se le colocó 9 features en su capa oculta (Para emular a PCA) y en la salida se colocó la misma entrada. La red neuronal resultante fue la siguiente:

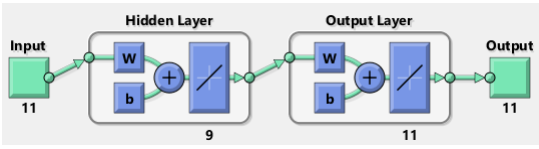

Fig. 4. Autoencoder para replicar PCA.

Luego de que se tiene la matriz de transformación de este autoencoder de la capa de entrada a la capa oculta y la matriz de transformación obtenida con PCA, podemos evaluar ambos algoritmos viendo que tanto ambos difieren de los datos originales, aumentar la dimensionalidad con PCA lo que se hizo fue multiplicar los datos reducidos por la transpuesta de su matriz de transformación, ya que al ser una matriz ortogonal es equivalente a encontrar su inversa, sin embargo para el caso del autoencoder lo que se hizo fue tomar su matriz que va de la capa oculta a la capa de salida.

Luego de hacer la reducción y el aumento de dimensionalidad con PCA y con el autoencoder se realizó la comparación de los datos obtenidos con ambos procedimientos con los datos originales, para esto se calculó la distancia euclidiana de cada dato original a su dato correspondiente en la versión recuperada, luego se sumaron todas estas distancias y se dividieron entre la cantidad total de datos.

$$D_{\text{conjuntos}} = \frac{1}{N} \sum_{t}^{N} \sqrt{\left(\sum_{i}^{d} \textbf{original}_i^t - \textbf{recuperada}_i^t \right)^2}$$

Esta comparación de conjunto se realiza con el conjunto original y el recuperado, es decir se reducirá la dimensionalidad del conjunto de datos luego se aumentara y esta versión recuperada se comparara con el conjunto de datos original.

IV. Resultados

A. Análisis discriminante lineal

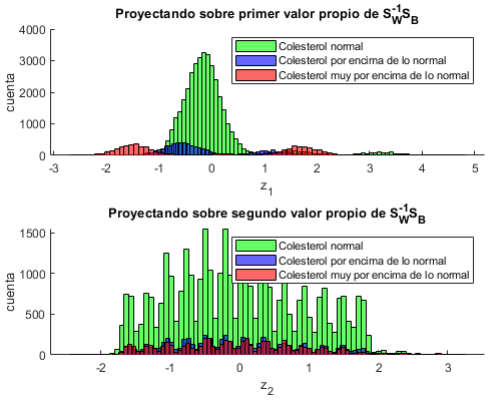

Fig. 5. Proyección realizada por LDA.

Al aplicar LDA solo quedaron dos valores propios distintos a cero en la matriz $S_W^{-1} S_B$ por lo tanto, la data queda proyectada solo sobre los dos vectores propios correspondientes a esos dos valores propios como se pudo ver en la figura anterior y en la siguiente figura:

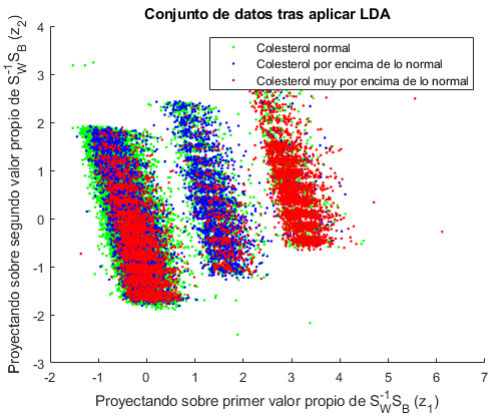

Fig. 6. Proyección realizada por LDA.

Luego se procedió a entrenar la red neuronal para replicar el comportamiento de LDA:

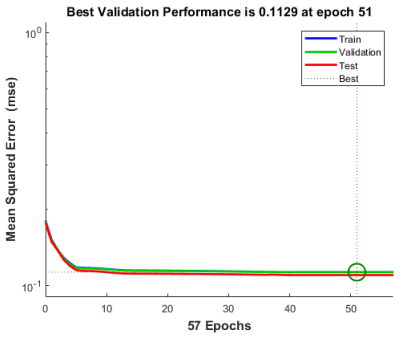

Fig. 7. Entrenamiento de la red neuronal para hacer LDA, NN_{LDA}.

Al extraer la matriz de transformación de la capa de entrada a la capa oculta y utilizar esta matriz para proyectar la data, la data nos queda de la siguiente forma:

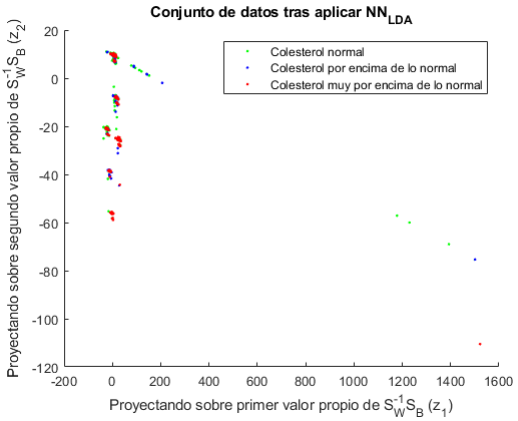

Fig. 8. Proyección realizada por NN_{LDA}.

Luego de que se tienen los dos conjuntos de datos se procedió a entrenar un algoritmo de aprendizaje (regresión logística de múltiples clases), sobre la data con la dimensionalidad reducida por medio de LDA y posteriormente por NN_{LDA}:

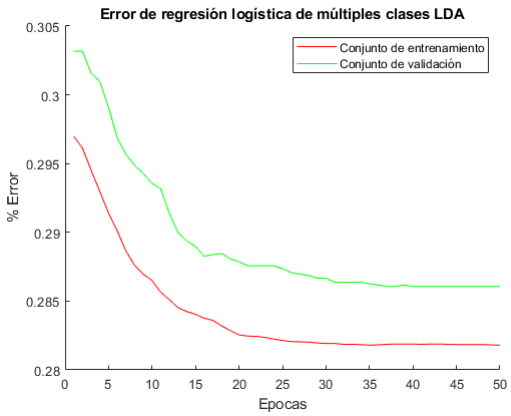

Fig. 9. Curva de aprendizaje LDA.

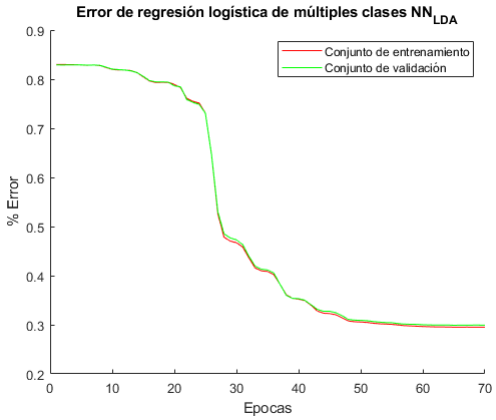

Fig. 10. Curva de aprendizaje NN_{LDA}.

Como ya se puede ver en las curvas de aprendizaje, el algoritmo es capaz de aprender más sobre la data con dimensionalidad reducida proporcionada por LDA que la proporcionada por NN_{LDA}, sin embargo esta diferencia es mínima.

Los resultados de utilizar ambos algoritmo fueron los siguientes:

TABLE I
RESUMEN DE RESULTADOS LDA

Conjunto	**Error de clasificación: LDA**	**Error de clasificación: NN_{LDA}**
Conjunto de entrenamiento	28.18%	29.51%
Conjunto de validación	28.61%	29.93%

Sus matrices de confusión fueron las siguientes:

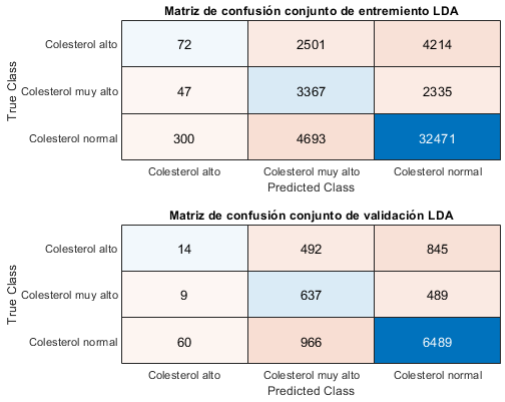

Fig. 11. Matriz de confusión LDA.

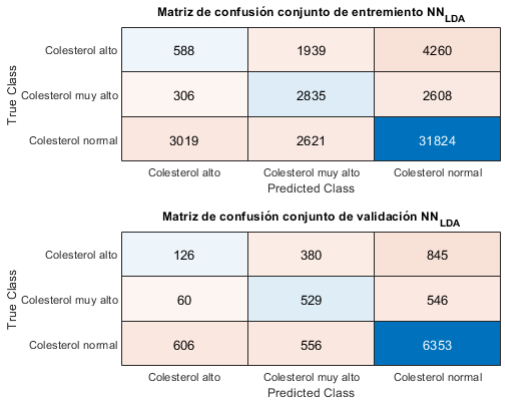

Fig. 12. Matriz de confusión NN_{LDA}.

B. Análisis de componente principal

Al aplicar PCA, quedaron finalmente 9 componentes para preservar el 90% de la varianza como se puede ver en la siguiente figura:

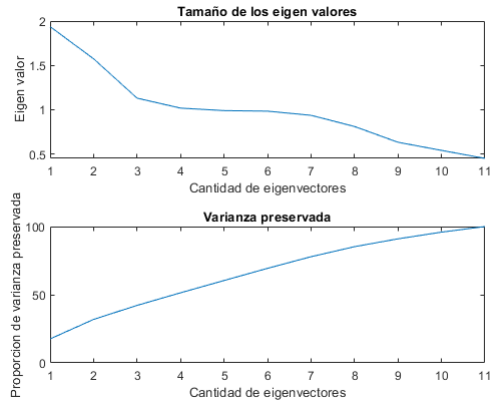

Fig. 13. Proporción de varianza preservada.

Luego se procedió a entrenar la red neuronal con 9 neuronas en su capa oculta y la misma entrada en la salida para replicar el comportamiento de PCA:

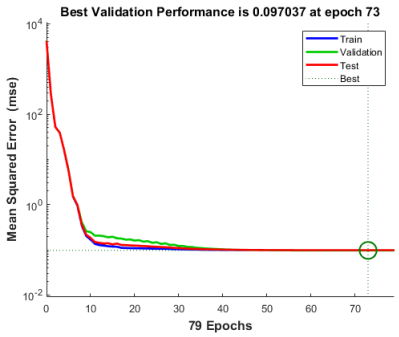

Fig. 14. Entrenamiento de la red neuronal para replicar PCA.

TABLE II
RESUMEN DE RESULTADOS PCA

Conjunto	PCA: Distancia entre conjunto orginal y conjunto recuperado	NN_{PCA}: Distancia entre conjunto orginal y conjunto recuperado
Conjunto de entrenamiento	1.2453	3.8676
Conjunto de validación	1.2513	3.7324

Recordando que esta distancia entre conjuntos la definí de la siguiente forma:

$$D_{\text{conjuntos}} = \frac{1}{N}\sum_{t}^{N}\sqrt{\left(\sum_{i}^{d}\textbf{original}_i^t - \textbf{recuperada}_i^t\right)^2}$$

C. Red neuronal

Como se mencionó anteriormente se corrieron dos experimentos, uno utilizando todos los features y otro solamente utilizando la altura, edad y peso para poder comparar con el algoritmo realizado anteriormente (Bayes multivariable), en la siguiente gráfica se puede ver la curva de aprendizaje al utilizar todos los features

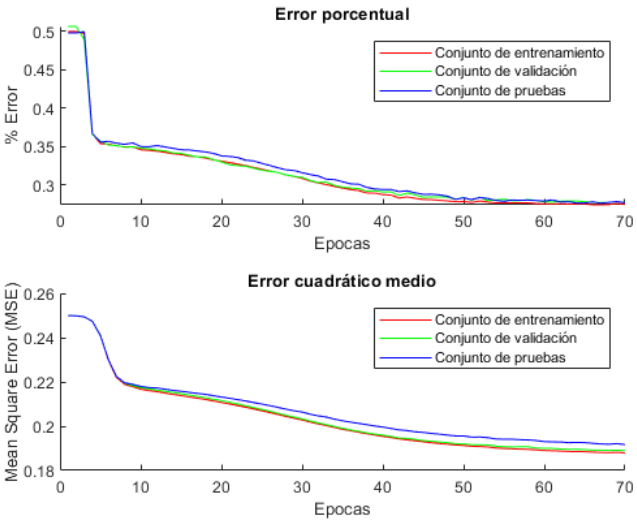

Fig. 15. Curva de aprendizaje de la red neuronal utilizando todos los features.

Como se puede ver en la gráfica pasaron 70 épocas para llegar a la convergencia y el error de entrenamiento final fue de 27.47%, el error de validación fue de 27.74% y el error de pruebas fue de 27.68%.

Para el segundo experimento donde solo fueron utilizados los features de Bayes multivariable, la curva de aprendizaje fue la siguiente.

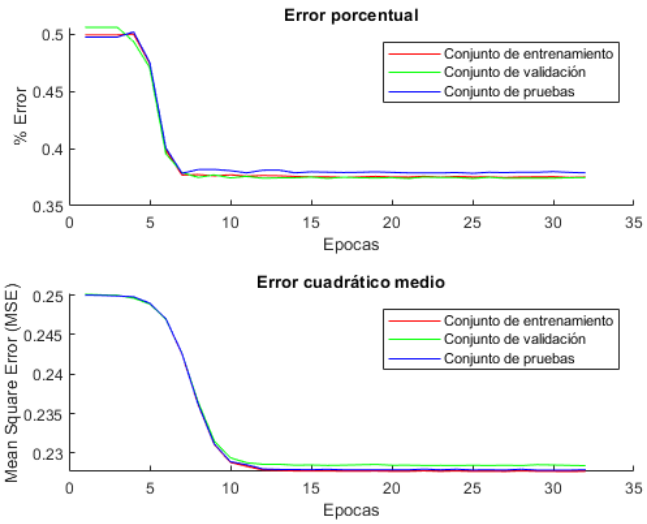

Fig. 16. Curva de aprendizaje de la red neuronal utilizando edad, peso y altura.

Como se puede ver en la gráfica pasaron 32 épocas para llegar a la convergencia y el error de entrenamiento final fue de 37.47%, el error de validación fue de 37.52% y el error de pruebas fue de 37.88%. Las matrices de dispersión correspondiente son las siguientes:

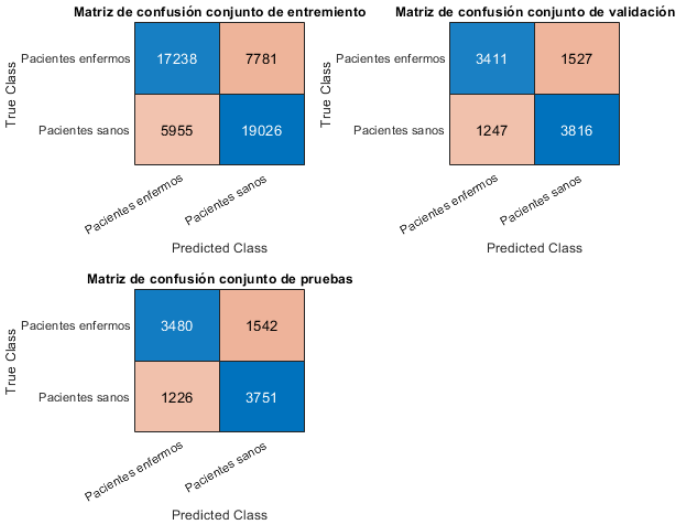

Fig. 17. Matriz de confusión de la red utilizando todos los features.

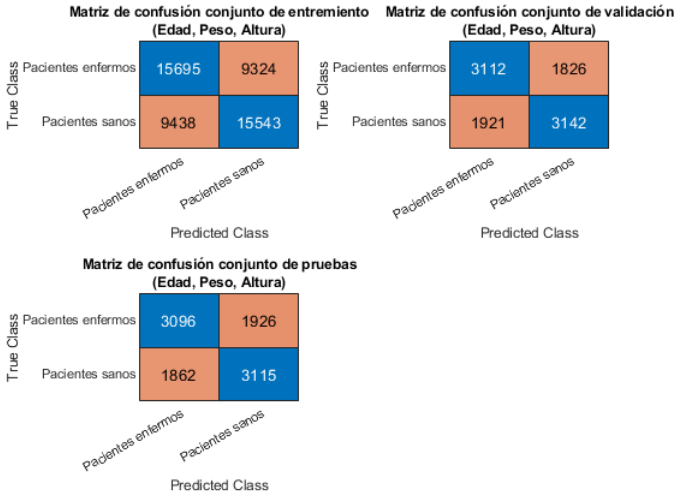

Fig. 18. Matriz de confusión de la red utilizando solo edad, peso y altura.

TABLE III
RESUMEN DE TIEMPOS

Modo	Tiempo de desarrollo	Tiempo de entrenamiento	Tiempo de ejecución (ms)
Todas las caracteristicas	∼ 6 horas	34.9095 s	Entrenamiento: 54.5ms Validacion: 10.9ms Pruebas: 11ms
Caracteristicas usadas en Bayes (Altura, Peso, Edad)	Mismo algoritmo diferentes parametros	9.6357 s	Entrenamiento: 127.2 ms Validacion: 26 ms Pruebas: 12.7 ms

TABLE IV
RESUMEN DE RESULTADOS DE ERROR

Modo	Error entrenamiento	Error validacion	Error pruebas
Todas las caracteristicas	%Error: 27.47% MSE: 0.1879	%Error: 27.74% MSE: 0.1891	%Error: 27.68% MSE: 0.1917
Caracteristicas usadas en Bayes (Altura, Peso, Edad)	%Error: 37.47% MSE: 0.2277	%Error: 37.52% MSE: 0.2284	%Error: 37.88% MSE: 0.2279

V. ANÁLISIS Y CONCLUSIONES

A. *Red neuronal*

Como se puede ver en los porcentajes de error, las curvas de aprendizaje y en las matrices de confusión de las figuras 17 y 18, cuando utilizamos todos los features el rendimiento del algoritmo aumenta considerablemente. Además si compramos el rendimiento solamente utilizando los features edad peso y altura el porcentaje de error es bástente similar al obtenido con Bayes multivariable, es ese algoritmo se alcanzó un porcentaje de aciertos de un 62.27% mientras que aquí en el conjunto de pruebas se alcanzó un error de un 37.88% que es equivalente a un porcentaje de aciertos de 62.12%.

Aunque cuando en la red neuronal se probaron todos los features el porcentaje de aciertos ascendió a un 72%, como se puede ver en la tabla 4. En Bayes multivariable no se llegó a

probar con todos los features por lo tanto no podemos hacer una comparación en este aspecto.

Otros aspectos importantes que creo que debo recalcar de esta implementación es que se utiliza $H = 10$, este valor fue establecido de forma arbitraria y $\eta = 0.0003$, cuyo valor fue establecido tras prueba y error. Además para detener el algoritmo se observó el cambio de error porcentual en las últimas 5 épocas, si es menor a un 0.02%, entonces se asume que hubo convergencia, estos valores de 5 y 0.02% también fue obtenidos tras prueba y error.

B. Análisis de componente principal

En la comparación de PCA con el Autoencoder como se puede ver en la tabla 2, los resultados son bastante similares y para tener una idea de que significan estos números sustituí a W, por una matriz de vectores aleatorios del mismo tamaño, y realice el aumento y disminución de dimensionalidad y los valores rondaban por +30, lo que es un orden de magnitud superior a los valores observados en la tabla 2 (que rondan entre 1 y 4). Sin embargo aunque ambos son buenos se ve que el PCA tienen una menor distancia al conjunto original por lo tanto se podría considerar que representa mejor la data original.

C. Análisis discriminante lineal

En el caso de LDA como se puede ver en la tabla 1, los resultados son bastante similares, se puede ver que la reducción de dimensionalidad realizada con una red neuronal representa muy bien la clase “nivel de colesterol”, ya que su porcentaje de error al realizar una regresión logística de múltiples clases, fue de 29.51% mientras que al realizar la regresión logística con LDA el porcentaje de error fue de 28.18%, es decir la diferencia en el rendimiento de ambos algoritmos es mínima.

REFERENCES

[1] E. Alpaydin, Introduction to machine learning, 3rd ed. Cambridge, Massachusetts: The MIT Press, 2014, pp. 267-310.

[2] Mathworks, ”MATLABDocumentation”, Mathworks.com, 2021. [Online]. Available: https://www.mathworks.com/help/matlab/. [Accessed: 9- Jun- 2021].

[3] S. Ulianova, ”Cardiovascular Disease dataset”, Kaggle.com, 2021. [Online]. Available: https://www.kaggle.com/sulianova/cardiovascular-disease-dataset. [Accessed: 11- Jun- 2021].

[4] Brooks, H., Tucker, N. (2015). Electrospinning predictions using artificial neural networks. Polymer, 58, 22-29. doi: 10.1016/j.polymer.2014.12.046