UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉCTRICA

ACELERACIÓN DEL ALGORITMO DE CLASIFICACIÓN K VECINOS MÁS CERCANOS A TRAVÉS DE LA COMPRESIÓN DE SET DE ENTRENAMIENTO

Br. Maria D. Dapena R.

Mérida, Junio, 2018

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIEÍA
ESCUELA DE INGENIERÍA ELÉCTRICA

ACELERACIÓN DEL ALGORITMO DE CLASIFICACIÓN K VECINOS MÁS CERCANOS A TRAVÉS DE LA COMPRESIÓN DE SET DE ENTRENAMIENTO

Trabajo de Grado presentado como requisito parcial para optar al tÍtulo de Ingeniero Electricista

Br. Maria D. Dapena R.

Tutor: Prof. José Luis Paredes

Mérida, Junio, 2018

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA ELÉTRICA

ACELERACIÓN DEL ALGORITMO DE CLASIFICACIÓN K VECINOS MÁS CERCANOS A TRAVÉS DE LA COMPRESIÓN DE SET DE ENTRENAMIENTO

Br. Maria D. Dapena R.

Trabajo de Grado presentado en cumplimiento parcial de los requisitos exigidos para optar al título de Ingeniero Electricista, aprobado en nombre de la Universidad de Los Andes por el siguiente Jurado.

Prof. José Luis Paredes

Prof. Francisco J. Viloria

Prof. Junior Amilcar Altamiranda

El futuro pertenece a quienes creen en la belleza de sus sueños

Eleanor Roosevelt

A mis padres y hermanos.

AGRADECIMIENTOS

En primer lugar, agradezco a Dios por guiarme siempre, por darme fuerza para levantarme cada vez que he caído y la constancia y determinación necesaria para alcanzar cada una de mis metas.

A la ilustre Universidad de Los Andes, a la escuela de Ingeniería Eléctrica y sus profesores, por darme la formación profesional y personal que todo estudiante debería aspirar. Gracias por abrirme las puertas para cumplir este sueño.

A mi tutor, el profesor José Luis Paredes por ser un guía durante toda la carrera, por todo el tiempo dedicado, por compartir su conocimiento dentro y fuera del salón de clases. Gracias por la oportunidad brindada, por enseñarme que antes de ser un gran profesional es importante ser una una excelente persona.

A la Universidad de Delaware por los servicios prestados para el desarrollo de este trabajo y al Profesor Gonzalo Arce por el tiempo dedicado.

Al Dr. Kilian Weinberger y el Dr. Lwarences Saul por facilitar el código para la implementación de *large margin nearest neighbor*.

A mis madre, padre, abuela, tía y hermanos por todo el amor y apoyo que siempre me han brindado.

A mis compañeros y amigos,gracias por compartir esta etapa de mi vida.

Maria D. Dapena R.

Maria D. Dapena R. Aceleración del algoritmo de clasificación K Vecinos más Cercanos a través de la compresión de set de entrenamiento. Universidad de Los Andes. Tutor: Prof. José Luis Paredes. Junio, 2018.

Resumen

El algoritmo de clasificación K vecinos más cercanos es posiblemente uno de los más antiguos y simples métodos de clasificación supervisada. Sin embargo, tiene un costo computacional muy alto ya que debe determinar la distancia Euclidiana entre la muestra a clasificar y todo el conjunto de entrenamiento. Existen tres técnicas para acelerar este algoritmo: reducción de cómputos, reducción del número de dimensión del conjunto de prueba y de entrenamiento y compresión del conjunto de entrenamiento. La última técnica presenta como ventaja un bajo uso de memoria, además de poder ser utilizado en grandes conjuntos de entrenamientos con pocas dimensiones. Se han realizado varios trabajos con el fin de realizar la compresión del conjunto de entrenamiento, unos proponen la selección de las muestras que mejor definan el conjunto de entrenamiento inicial eliminando así la redundancia, otros proponen comprimir los datos a algunos grupos centrales. En este trabajo se utiliza un enfoque estocástico de la compresión del conjunto de entrenamiento, para ello se desarrollaron los algoritmos necesarios y para evaluar su rendimiento, se probaron en cuatro bases de datos diferentes, tres de imágenes y una de audio. La técnica implementada permite obtener, dado un conjunto inicial de datos para el entrenamiento, un conjunto de entrenamiento mucho más pequeño de vectores optimizados. La compresión estocástica se basa en el método matemático del gradiente descendente, este método tiene propiedades atractivas como poder seleccionar la relación de compresión, permitiendo un conjunto de entrenamiento mucho más pequeño que aumenta la velocidad de clasificación drásticamente. De los resultados de las simulaciones realizadas con las bases de datos de prueba se obtuvo, en la mayoría de los casos, un error que está por debajo 5 % de muestras mal clasificadas, incluso para relaciones de compresión tan bajas como del 1 %. Adicionalmente, la combinación de este algoritmo junto con otros como *Large Margin Nearest Neighbor* para la reducción de dimensiones acelera al algoritmo por encima de los valores esperados, demostrando que el método es compatible con métodos ya diseñados.

Descriptores: Algoritmos de clasificación, K vecinos más cercanos, aceleración de algoritmos de clasificación, compresión del conjunto de entrenamiento.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE ALGORITMOS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Los algoritmos de clasificación son herramientas computacionales de gran utilidad debido a la gran cantidad de aplicaciones que poseen, pueden utilizarse para el reconocimiento de imágenes y audio para la clasificación de terrenos e incluso para realizar predicciones en el ámbito financiero. En la actualidad los volúmenes de datos que se manejan, además de la constante demanda de rápidas respuestas por parte de usuario, hacen necesario el desarrollo constante de técnicas innovadoras que mejoren el desempeño de los algoritmos de clasificación básicos. La clasificación es un proceso que divide un espacio en regiones mutuamente excluyentes de acuerdo a sus características del conjunto, a cada región se le denomina clase y a través de una regla, se asigna una nueva muestra a una de estas regiones [1]. Cuando se conoce de antemano las clases y sus características se trata de clasificación supervisada, donde las muestras son clasificadas a partir de un conjunto de entrenamiento que contiene las clases. Dentro de este tipo existen distintos algoritmos de clasificación donde destaca por su fácil entendimiento la regla de los K Vecinos más Cercano (K-NN, del inglés *K Nearest Neighbors*) [2].

K-NN es uno de los algoritmos de clasificación más sencillo, ya que basa su toma de decisiones en cálculos de distancia, así, una nueva muestra se clasifica de acuerdo a la clase dominante de sus K vecinos más cercanos. No obstante, para que el algoritmo tenga un buen desempeño es necesario contar con un conjunto de entrenamiento de gran tamaño, lo que vuelve la implementación de éste computacionalmente costosa en términos de memoria y cómputo. Con el fin de poder utilizar las ventajas que ofrece K-NN se han desarrollado diferentes métodos para su optimización [3].

Existen tres técnicas para optimizar el algoritmo K-NN. La primera consiste en reducir el número de cálculos de distancia a realizar, esto se realiza a través del diseño de estructuras de arboles [4]. Sin embargo, a pesar de acelerar el algoritmo, los métodos desarrollados hasta el momento dentro de esta técnica siguen almacenando todo el conjunto de entrenamiento manteniendo el uso de memoria como una desventaja. Una segunda técnica propone la

reducción de dimensión del conjunto de entrenamiento a través de métodos de reducción supervisada [5]. Por último, la tercera técnica consiste en la compresión del conjunto de entrenamiento reduciendo en número de muestras dentro de este conjunto [2].

Es lógico que disminuir el tamaño del conjunto de entrenamiento acelera el algoritmo de clasificación de manera inversamente proporcional a la compresión que se le aplique al conjunto, sin embargo, una simple compresión del conjunto de entrenamiento aumenta el error en las predicciones realizadas por K-NN. La compresión estocástica del conjunto de entrenamiento (SNC, del inglés *Stochastic Neighbor Compression*) [2] es un método propuesto bajo el enfoque de la tercera técnica. SNC Comprime el conjunto de entrenamiento a través del aprendizaje de una cantidad menor de vectores m sintetizados, los m vectores son seleccionados de un conjunto de entrenamiento inicial con n vectores ($n << m$), y luego optimizados de manera que el error se reduzca tanto como sea posible. Este algoritmo tiene sus bases en un enfoque estocástico de la regla de clasificación K-NN, lo cual permite aproximar el error de clasificación a una función de costo continua y derivable.

Este trabajo plantea como objetivo general el desarrollo de herramientas computacionales que permitan comprimir el conjunto de entrenamiento con el fin de acelerar el algoritmo de clasificación K-NN. Esto se logra a través de los siguientes objetivos específicos:

- Estudiar y comprender la regla de clasificación K-NN,así como, sus ventajas y limitaciones.
- Estudiar las técnicas existentes para la aceleración del algoritmo de clasificación: reducción de número de cálculos, reducción de dimensión y compresión del conjunto de entrenamiento.
- Estudiar los métodos propuestos para realizar la compresión del conjunto de entrenamiento: selección de los vectrores que mejor representen los datos [6], selección de grupos concentrados [7] y la compresión estocástica [2].
- Estudiar los métodos matemáticos y los fundamentos estocásticos del método seleccionado.
- Estudiar y seleccionar las bases de datos a utilizar.

- Aplicar pre-acondicionamiento a las bases de datos seleccionadas.
- Implementar, usando la herramienta de programación MATLAB, un algoritmo que permita realizar la compresión estocástica del conjunto de entrenamiento (SNC).
- Aplicar el algoritmo implementado en MATLAB sobre las diferentes bases de datos.
- Comprobar eficiencia de la compresión en términos de aceleración del algoritmo de clasificación.
- Verificar la eficiencia de la optimización en función del error en el algoritmo de clasificación.
- Analizar la velocidad de convergencia del método matemático seleccionado.

En este trabajo se busca corroborar, a través del diseño de algoritmos propios además de la revisión matemática del método, los resultados obtenidos en el artículo *Stochastic Neighbor Compression* [2], se usa esta referencia como guía en la implementación, desarrollo y prueba del algoritmo seleccionado.

Con el fin de cumplir los objetivos planteados, el presente trabajo se organizó en cuatro Capítulos. El **Capítulo 1** se presenta una breve revisión bibliográfica de los algoritmos de clasificación y su optimización, haciendo énfasis en el algoritmo K-NN. Adicionalmente, en este capítulo se describe los fundamentos estocásticos principales para comprensión de método desarrollado. En el **Capítulo 2** se desarrolla el método implementado en este trabajo, explicando sus fundamentos estocásticos, los trabajos anteriores que dieron origen al método y las bases matemáticas de mismo. En el **Capítulo 3** se describe cada una de las bases de datos seleccionadas, sus características, autores, donde encontrarlas, procesamientos previos a los realizados y los procesamientos aplicados sobre cada una de ellas en este trabajo. Finalmente, en el **Capítulo 4**, se analizan los resultados obtenidos en los experimentos realizados en las bases de datos con diferente relación de compresión, el porcentaje de muestras mal clasificadas en la etapa de pruebas, el tiempo de entrenamiento por iteración, la velocidad de convergencia del algoritmo, y por último la aceleración del algoritmo de clasificación.

CAPÍTULO 1
MARCO TEÓRICO

En la actualidad, los algoritmos de clasificación han adquirido importancia en diversos campos como el reconocimiento de imágenes, de audio, la predicción de comportamientos financieros entre otros. Es por esto que han surgido diversas técnicas para su mejora y optimización. En este capítulo se describirán los algoritmos de clasificación supervisada más conocidos, así como los métodos de optimización aplicados en éstos, los modelos de aceleración del algoritmo K Vecinos más cercanos (K-NN del ingles *K Nearest Neighbors*) y por último, los conceptos estadísticos necesarios para la comprensión del método desarrollado en este trabajo.

1.1 ALGORITMOS DE CLASIFICACIÓN

La clasificación, desde la perspectiva computacional, se define como la partición del espacio de características en regiones mutuamente excluyentes, de tal forma que cada región está asociada a una clase o grupo, y se utiliza para decidir a cúal de las clases disponibles pertenece una nueva muestra [1].

Dependiendo de la estructura de la información que se tenga los procesos de clasificación pueden ser supervisados o no. En la clasificación supervisada es necesario que el usuario caracterice las clases a partir de una muestra. Por otro lado, en la clasificación no supervisada no se necesita de interacción con el usuario, siendo el mismo algoritmo el encargado de caracterizar las clases de manera automática, agrupando los datos según sus características [8, pp.10]. Dado que la clasificación no supervisada está fuera de los alcances de este proyecto, no se profundizará más en este tema.

Los algoritmos de clasificación supervisada se utilizan cuando se conoce a priori las clases

y los representantes de las mismas. En este método se dispone de un conjunto de muestras llamado conjunto o *set* de entrenamiento, en el cual las muestras se agrupan en clases o categorías de acuerdo a sus propiedades. Básicamente, para clasificar una nueva muestra se compara sus características con las de las muestras del conjunto de entrenamiento y se le asigna una clase de acuerdo a una regla de clasificación [1].

Existen diversas técnicas inteligentes de clasificación supervisada tales como Redes Neuronales (NN del inglés *Neural Networks*), Máquinas de Soporte Vectorial (SVM del ingles *Support Vector Machine*), K Vecinos más Cercanos (K-NN del ingles *K Nearest Neighbors*), métodos Bayesianos entre otros. A continuación se describen brevemente algunas de estas técnicas.

1.1.1 Redes Neuronales

Una red neuronal (NN) es un sistema que imita el comportamiento del cerebro humano para ejecutar una determinada tarea o función de interés. La red es usualmente implementada por medio de elementos electrónicos o simulada mediante software en computadoras. Para tener un buen desempeño, las redes neuronales, utilizan una conexión masiva entre simples celdas de computación a las que se llaman "neuronas" o "unidades de procesamiento" [9]. Las redes neuronales pueden ser máquinas de aprendizaje supervisado o no supervisado dependiendo del algoritmo utilizado.

Según Haykin [9, pp.1-2] : Una red neuronal es una red de procesadores distribuidos masivamente, formada por simples unidades de procesamiento, que tiene la capacidad de almacenar conocimiento y utilizarla posteriormente. Es similar al cerebro en dos aspectos:

- El conocimiento es adquirido por la red del ambiente a través de un proceso de aprendizaje.
- La fuerza de conexión interneuronal, conocida como pesos sinápticos, es utilizada para almacenar el conocimiento.

1.1.2 Máquinas de soporte vectorial

Una Máquina de Soporte Vectorial (SVM) es básicamente una máquina de aprendizaje binario diseñada especialmente para problemas de clasificación, que también es utilizada

para resolver problemas de regresión lineal.

Su funcionamiento consiste en la construcción de un hiperplano, cuya dimensión está relacionada con la de la data. Se busca que el hiperplano tenga el mayor margen de separación posible con respecto a las dos clases del conjunto de entrenamiento.

Para la clasificación de una nueva muestra, se sustituye la misma en la ecuación de decisión ecuación (1.1), donde $\mathbf{x}$ es la muestra de entrada, $\mathbf{w}$ es un vector columna de ponderación relacionado con el hiperplano, b representa la desviación y el superíndice T indica la transposición. Si g(x) es mayor o igual a cero se asigna a la clase tomada como positiva, si por el contrario, es estrictamente menor, se le asigna la clase negativa [9, pp.269-270].

$$g(x) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b \tag{1.1}$$

En la figura 1.1 se presenta un ejemplo con muestras que poseen solo 2 dimensiones, y dos posibles hiperplanos HP1 y HP2. Siendo el hiperplano HP2 el mejor u óptimo, ya que además de separar las muestras, es el que ofrece mayor distancia con respecto a los datos.

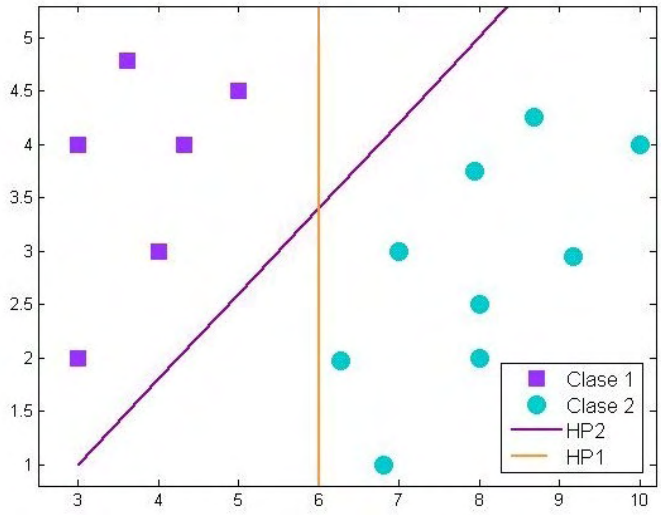


Figura 1.1. Hiperplanos de separación de clases usando Máquinas de Sopore Vectorial, tomado de [9].

1.1.3 K Vecinos más Cercanos

El algoritmo K Vecinos más Cercanos (K-NN) basa su funcionamiento en calcular la distancia de una nueva muestra a cada una de las muestras del conjunto de entrenamiento, y

asignarle la clase dominante entre sus K vecinos más cercanos. Como se observa en la figura 1.2 una nueva entrada (punto azul) puede ser clasificada de manera diferente dependiendo del valor de K, para K igual a 1 o 4 la entrada será clasificada como morada, en otro caso, si K es igual a 6 la entrada será clasificada como verde por ser esta la clase dominante en sus 7 vecinos más cercanos. Generalmente, se utiliza la distancia Euclidiana [1] para la asignación de clases, sin embargo, se han realizado estudios en los que se aplica la distancia de Mahalanobis [2] para disminuir el error de este algoritmo.

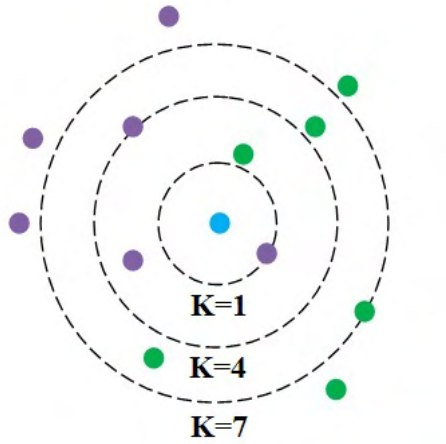


Figura 1.2. K Vecinos más Cercanos, tomado de [3].

La mayor desventaja del K-NN es su lenta velocidad de clasificación y su alto uso de memoria, al tener que calcular la distancia de la nueva muestra con respecto a cada una de las muestras del conjunto de entrenamiento y además tener que almacenar los datos del conjunto de entrenamiento. Limitación que se incrementa en la actualidad ya que se manejan grandes volúmenes de datos. A pesar de sus desventajas, la simplicidad en los procesos de toma de decisiones lo hace un algoritmo muy atractivo, es por esto que se han desarrollado diferentes técnicas para su optimización, la cuales se explicaran más adelante [3, pp.31-36].

1.2 OPTIMIZACIÓN DE ALGORTIMOS DE MÁQUINAS DE APRENDIZAJE SUPERVISADO

Cuando se trata de optimizar un algoritmo de aprendizaje es necesario conocer la función costo relacionada al algoritmo, que asigna una penalización cuando la predicción o clasi-

[1] Distancia Euclidiana u ordinaria es la distancia que se deduce a partir del teorema de Pitágoras $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

[2] La distancia de Mahalanobis se define el la sección 1.3.2

ficación no es realizada correctamente, si esta función es convexa o puede aproximarse a una función convexa, sin realizar demasiadas simplificaciones, se pueden aplicar métodos matemáticos de primer y segundo orden, para disminuir el error en las predicciones del algoritmo. Por otro lado, si realizar esta aproximación significa alterar demasiado la función encargada de realizar la predicción es necesario utilizar un nuevo método, como *Deep Learning* (DL) [10].

1.2.1 Funciones de costo convexas

Son varios los métodos de aprendizaje que están dentro de esta categoría, incluyendo K Vecinos más Cercanos y Máquinas de Soporte Vectorial. Los modelos aplicados para la optimización de este tipo de algoritmos se dividen en dos, éstos son: métodos de primer orden tales como método del gradiente, método estocástico del gradiente, métodos para reducir la varianza y métodos de segundo orden. Se considera de forma general el problema de optimización como el mostrado en la ecuación (1.2), donde n es el número de datos que se tienen, $\mathbf{x}_i$ es el elemento de entrada, $\mathbf{y}_i$ es el elemento de salida ya clasificado resultado de la predicción, p es la función de predicción asociada al algoritmo, l es la función costo, $\mathbf{w}$ es el vector de parametrización de la función costo, λ es un parámetro de ponderación mayor o igual a cero y finalmente, r es $\| \mathbf{w}_1 \|$ que es la primera solución al problema o alguna otra función convexa de regularización [10].

$$\min_{\mathbf{w} \in R^d} F(\mathbf{w}), \text{ donde } F(\mathbf{w}) = \frac{1}{n} \sum_{i=0}^{n} l(p(\mathbf{w}, \mathbf{x}_i), \mathbf{y}_i) + \lambda r(\mathbf{w}) \tag{1.2}$$

Métodos de primer orden: son llamados de esta forma ya que su solución solo necesita de la primera derivada. Dentro de los métodos de primer orden están:

- Método del gradiente descendente: también conocido como *Steepest Descent* (SD) conceptualmente es el método más rápido para minimizar una función convexa y suave. Para encontrar el mínimo de la función se toman pasos en direcciones proporcionales al negativo del gradiente o un aproximado del gradiente. Este algoritmo está diseñado especialmente para funciones multivariables. En esta solución se asume una respuesta inicial y a través de un proceso iterativo se actualiza la estimación utilizando la ecuación (1.3):

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k \nabla F(w_k) \ , \tag{1.3}$$

donde, F es la función definida en 1.2, $\mathbf{w}$ corresponde a la incógnita sobre la cual se itera, ∇ indica el gradiente de la función $F(\mathbf{w}_k)$ y $\alpha_k > 0$ es un parámetro que tiene el tamaño del paso en la iteración k. El funcionamiento del algoritmo está altamente relacionado con la elección de la secuencia α_k. Generalmente se utilizan líneas de búsqueda en cada iteración para encontrar el parámetro α_k ya que genera un alto rendimiento en gran variedad de problemas. Sin embargo, en el área de Máquinas de Aprendizaje implementar líneas de búsqueda es altamente costoso ya que para cada calculo de F es necesario recorrer todos los datos [10].

- Método estocástico del gradiente: el método estocástico del gradiente (SGD, del inglés *Stochastic gradient method*) inicialmente propuesto por Robbins y Monro [11], es utilizado cuando se desea minimizar funciones estocásticas. En el contexto de resolver sistemas de ecuaciones estocásticos este método es mucho más rápido que otros métodos, ya que por iteración trata $O(d)$ mientras que métodos como SD tardan $O(dn)$, donde d es la dimensión que pose los datos y n el número de datos.

  El nuevo valor es calculado según la ecuación (1.4):

$$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \alpha_k \nabla_{S_k} F(\mathbf{w}_k) \ , \tag{1.4}$$

  donde, las variables y parámetros son los mismos que los de la ecuación (1.3) y ∇_{S_k} representa la evaluación del gradiente en el vector $\mathbf{w}$ utilizando solo una porción de vectores contenidos en S_k, que es conocido como mini-batch y tiene elementos elegidos uniformemente y aleatoriamente entre $1, ...n$. Al igual que en SD es necesario calcular el tamaño del paso α_k. Al contrario que en SD, en este método la elección de la secuencia α_k no garantiza la convergencia a un mínimo de la función, solo garantiza la convergencia a un vecindario cercano al mínimo [10].

- Método de reducción de varianza: los argumentos utilizados es SGD parten de la suposición que la finalidad es resolver el problema 1.2 y que se puede muestrear indefinidamente el espacio de entradas y salidas. Sin embargo, en el ámbito de máquinas

de aprendizaje el conjunto $(x_i, y_i)_{i=1}^n$ esta generalmente fijado y n es realmente grande, por lo que existe suficientes razones para ver una distribución discreta y uniforme del conjunto como una buena aproximación a la distribución real P.

Una forma de mejorar el método SGD es explotando la estructura de la función F como la suma finita de n funciones más un simple termino convexo.

Otro enfoque para este método es utilizar la ecuación (1.4), pero para conseguir una convergencia más rápida se incrementa el tamaño de S_k durante el proceso de iteración [10].

Métodos de segundo orden: una de las áreas de optimización con más actividad en la actualidad está relacionada en cómo podrían ser utilizadas las derivadas de segundo orden para acelerar en entrenamiento en las máquinas de aprendizaje. La idea principal es calcular la iteración k de la función $F(\mathbf{w})$ alrededor de $\mathbf{w}_k$ a través del modelo cuadrático, mostrado en la ecuación(1.5), donde B_k es la matriz de curvatura que debe ser positiva definida y debe estar relacionada con la segunda derivada de $F(\mathbf{w})$, $\mathbf{w}$ es la variable sobre la cual se itera, F es la función definida en 1.2 y $\mathbf{s}$ se refiere al paso de la iteración [10].

$$m_k = F(\mathbf{w}_k) + \nabla F(\mathbf{w}_k)^\mathsf{T}\mathbf{s} + \frac{1}{2}\mathbf{s}^\mathsf{T}\mathbf{B}_k\mathbf{s} \tag{1.5}$$

Como los métodos utilizados en este trabajo se relacionan con los de primer orden no se profundizara más en estos métodos.

1.2.2 Deep Learning

Para una variedad de problemas de aprendizaje se ha demostrado que asumir funciones convexas implica demasiadas simplificaciones, trayendo como consecuencia un incremento en el error. El *Deep Learning* (DL) permite la optimización de algoritmos que no están relacionados con funciones convexas.

El DL representa una clase de algoritmos que busca construir abstracciones de alto nivel de la data a través del uso de grafos profundos (*deep graph*) con múltiples capas, que involucran transformadas lineales y no lineales secuenciales. Estructuralmente el algoritmo

tiene la forma de un grafo con subconjuntos de nodos arreglados secuencialmente, cada subconjunto de nodos es una capa. En los casos más sencillos las aristas de los nodos solo existen entre los nodos de una capa y la siguiente. A pesar de ser una estructura de datos, la clave es como la información se "alimenta" a través de ella. En el caso mas simple cada elemento de un vector de entrada es asignado a un nodo en la primera capa, conocida como capa de entrada (*input layer*), los valores en ésta son pasados a los nodos de la siguiente capa después de haber sido ponderados con los pesos correspondientes a cada arista, en un nodo dado un valor puede ser transformado aplicándole una función de activación (lineal o no lineal) antes que los valores sigan pasando a través de la red. La última capa es conocida como la capa de salida (*output layer*) que otorga la salida esperada. Las capas entre la entrada y la salida son conocidas como capas escondidas (*hidden layer*), cuantas más de éstas tenga la red, mayor nivel de profundidad tendra [10]. La figura 1.3 muestra un ejemplo de la estructura de la red, donde, los $\mathbf{x}_i$ representa los vectores de entrada, los $[\mathbf{W}_i]\,jk$ indica la ponderación de la entrada j en en nodo k, $\mathbf{h}_i$ indica la función que es aplicada en el nodo y $\mathbf{y}_i$ es la salida de la red.

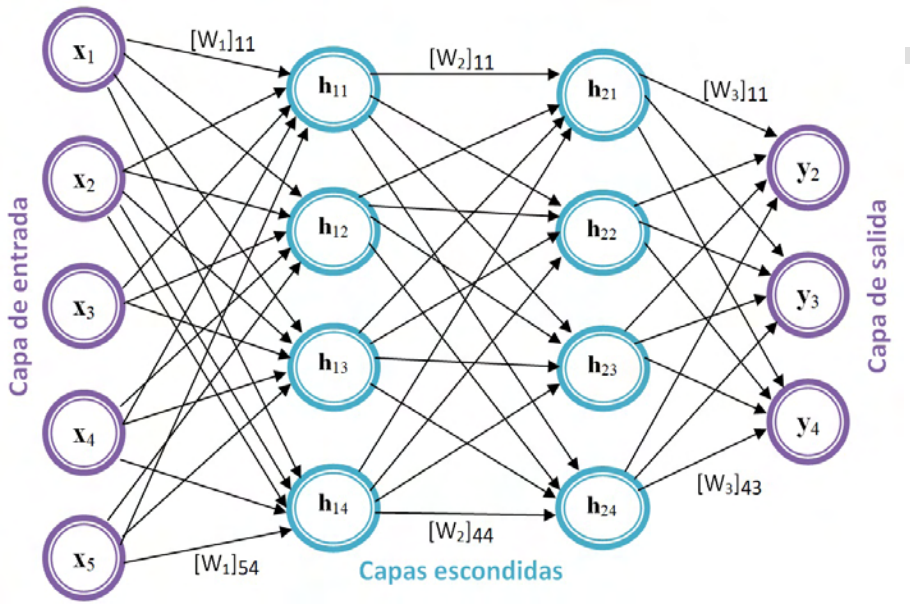


Figura 1.3. Estructura de un algoritmo de *Deep Learnig*, tomado de [10].

1.3 OPTIMIZACIÓN DEL ALGORITMO K-NN

Existen tres enfoques para la optimización del algoritmo K-NN. El primero consiste en reducir el número de cálculos de distancia, el segundo en la reducción de dimensión de los

datos y por último la compresión del conjunto de datos de entrenamiento.

1.3.1 Reducción del numéro del cálculos de distancia

La reducción de cálculos de distancia se implementa a través de estructuras de árboles bastante complicadas, como los *cover/ball trees* [4]. Un *cover/ball tree* es un refinamiento de una estructura de navegación (un árbol), que está relacionado con una variedad de otras estructuras de datos desarrolladas para organizar datos de baja dimensión.

Un *cover tree* de un conjunto de datos S está compuesto por diferentes niveles, cada uno está relacionado con un número entero i que disminuye a medida que se desciende en los niveles C. Todo nodo en el árbol está relacionado con un punto de S, como consecuencia cada punto del conjunto puede estar asociado con múltiples nodos del árbol, sin embargo, cada punto puede aparecer una sola vez por nivel. La primera aparición del dato es conocida como padre mientras que sus apariciones en las siguientes capas serán conocidas como su descendencia. Una característica importante de esta estructura consiste en que para todo punto p perteneciente a una capa C_{i-1} existe un q perteneciente a C_i tal que $d(p, q) < 2^i$, y el nodo en el nivel i asociado con q es un padre del nodo en el nivel *i-1* asociado con p.

La forma más sencilla de describir el árbol es asumir un número infinito de niveles donde C_∞ contiene únicamente las raíces de S, y $C_{-\infty} = S$.

Para encontrar el vecino más cercano de un punto p en un *cover tree*, se desciende a través de los niveles, haciendo un seguimiento de un subconjunto Q_i de nodos que probablemente contengan al vecino más cercano de p como un descendiente. El algoritmo construye iterativamente Q_{i-1} expandiendo Q_i hacia su descendencia en las siguientes capas y desechando los q que se alejan del vecino más cercano de p.

La desventaja de este método es que sigue siendo necesario almacenar en memoria todos los datos.

1.3.2 Reducción de dimensión

Este método reduce las dimensiones de los datos a través de mecanismos supervisados. Existen diversas formas de realizar la reducción, en este trabajo se hace referencia a dos de estos algoritmos por ser de especial interés, estos son: LMNN (del inglés *Large Margin Nearest*

Neighbor) y Análisis de Componentes Principales (PCA, del inglés *Principal Component Analysis*).

Large Margin Nearest Neighbor (LMNN): es un algoritmo de máquinas de aprendizaje con enfoque estadístico. Su función principal es aprender a calcular la distancia de Mahalanobis.

La distancia de Mahalanobis es una medida de distancia introducida por Mahalanobis en 1963, con la finalidad de determinar la similitud entre dos variables, tomando en cuenta la correlación entre ellas. En general está definida por la ecuación (1.6), donde si M es igual a la matriz identidad se obtiene la distancia Euclidiana.

En LMNN con el fin de aprender la distancia de Mahalanobis utiliza la ecuación (1.6), donde $\mathbf{x}_i$ y $\mathbf{x}_j$ son vectores columna, $\mathbf{M}$ representa la matriz de Mahalanobis, y puede ser descompuesta en términos de una matriz L.

$$d(\overrightarrow{\mathbf{x}_i}, \overrightarrow{\mathbf{x}_j}) = (\overrightarrow{\mathbf{x}_i} - \overrightarrow{\mathbf{x}_j})^{\mathsf{T}}\mathbf{M}(\overrightarrow{\mathbf{x}_i} - \overrightarrow{\mathbf{x}_j}), \text{ donde } \mathbf{M} = \mathbf{L}^{\mathsf{T}}\mathbf{L} \tag{1.6}$$

El efecto de aplicar la distancia de Mahalanobis en el algoritmo K-NN se muestra en la figura 1.4, donde la matriz de forma el espacio de manera que las muestras con la misma clase se acerquen y las de diferente clase se alejen. El objetivo principal de utilizar esta distancia es alejar los datos que puedan ocasionar una predicción errónea y acercar aquellos que producen un acierto.

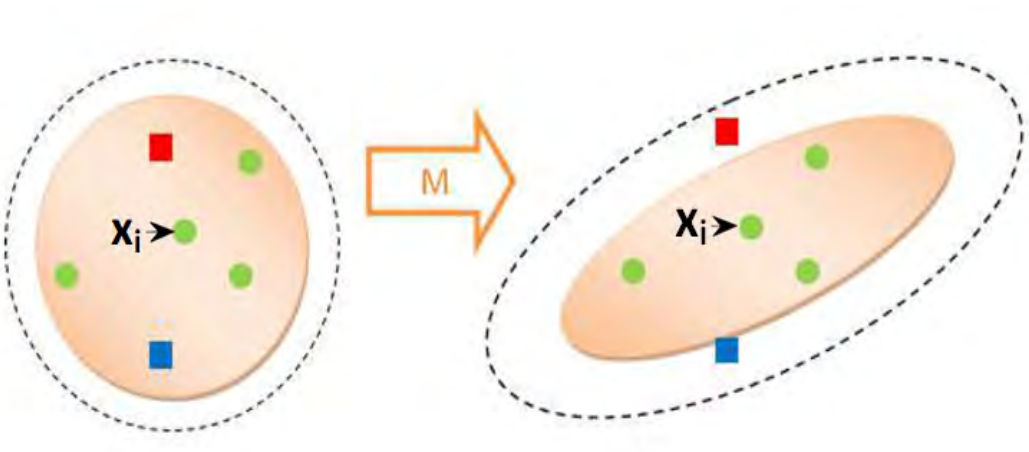


Figura 1.4. Efecto de aplicar la matriz de Mahalanobisel en el algorimo K-NN (LMNN), tomado de [5].

Si L es rectangular se puede realizar una reducción de dimensión a través de la transformación lineal $\mathbf{L}\overrightarrow{x_i}$, reduciendose así la dimensión de los datos resultantes a la menor dimensión de $\mathbf{L}$. El algoritmo LMNN propuesto por Weinberger y Saul [5] realiza estos cálculos, arrojando como resultado la matriz $\mathbf{L}$.

Análisis de Componentes Principales PCA: es una herramienta estadística que tiene aplicaciones en las máquinas de aprendizaje, ya que permite encontrar patrones en conjuntos de datos de elevada dimensión, es decir satos que están representados por muchas características.

En otras palabras, PCA busca la proyección en nuevo sistema de coordenadas según la cual los datos estén mejor representados. Esta técnica convierte un conjunto de observaciones de variables posiblemente correlacionadas en un conjunto de valores de variables sin correlación alguna, llamadas componentes principales.

El PCA construye una transformación lineal que escoge un nuevo sistema de coordenadas para el conjunto original de datos. La metodología que se aplica es sencilla, primero debe centrarse los datos extrayendo la media de cada variable, luego debe construirse la matriz de covarianza o de coeficientes de correlación a partir de los datos centrados. Como las matrices presenta simetría y sus vectores son linealmente independientes es posible encontrar los autovalores y autovectores de la misma. Para obtener la representación en el nuevo sistema de coordenadas deben organizarse los autovalores y los autovectores relacionados con ellos de manera descendente y finalmente se hace la transformación lineal mostrada en 1.7 donde $\mathbf{W}$ es la matriz de los autovectores y $\mathbf{X}$ es la matriz de datos con las muestras colocadas como columnas una a continuación de la otra. Para reducir la dimensión de los datos se pueden multiplicar únicamente por los autovectores que contengan la mayor cantidad de energía, asociados a los autovalores más grandes, los cuales están relacionados con las variables con mayor varianza, de esta manera se encontrará una representación de menor dimensión en el nuevo sistema de coordenadas.

$$\mathbf{X}_n \leftarrow \mathbf{W}^{\mathsf{T}} * \mathbf{X} \tag{1.7}$$

Es importante mencionar que después de haber eliminado componentes, se puede regresar

al sistema de coordenadas original donde se tenía la data a través de la relación $\mathbf{X} \leftarrow \mathbf{W} * \mathbf{X}_n$, debido a que $\mathbf{W}$ es ortogonal, su inversa es igual a la misma matriz transpuesta [12].

1.3.3 Compresión del conjunto de datos de entrenamiento

La compresión del conjunto de entrenamiento se logra a través de la reducción del número de datos de entrada. Existen diferentes enfoques para realizar esta tarea. Trabajos anteriores proponen submuestrear los datos respetando reglas audaces para remover datos redundantes, como los métodos propuestos por Hart [13] y Angiulli [6]. Algoritmos alternativos proponen reducir los datos a algunos grupos como lo propone Chang [14] o Kohonen [15], los grupos pueden ser optimizados con procedimientos de inicialización multifase métodos propuestos por Decaestecker [16], Liu y Nakagawa [7]. Otro algoritmo propuesto por Bermejo y Cabestany [17] aprende prototipos "suavizando" la regla para la toma de decisiones en el momento de realizar las pruebas, este método evita la necesidad de utilizar estructuras de árboles. Finalmente, existe un enfoque estocástico en el cual el conjunto de datos de entrenamiento está compuesto solo por algunas muestras sintetizadas de manera que se reduzca el error [2].

En el Capítulo 2 se describirá con detalle la compresión estocástica del conjunto de entrenamiento propuesto por Kusner, Tyree, Weinberger y Agrawal.

1.4 DISTRIBUCIÓN DE PROBABILIDAD

Una distribución de probabilidad de una variable continua o discreta se conoce como la regla de correspondencia entre cada uno de los valores que la variable pueda tomar y su probabilidad asociada. Dependiendo de la naturaleza de la variable, las distribuciones se consideran continuas o discretas, las cuales tienen tratamientos distintos. Las distribuciones de probabilidad se expresan en términos de una variable aleatoria, cuya función principal es realizar un mapeo de un conjunto de eventos en un espacio Ω a una línea recta. Es importante señalar que una distribución de probabilidad debe ser siempre una función no decreciente [18].

En este trabajo se utilizaran variables continuas razón por la cual se hará énfasis en las distribuciones continuas.

1.4.1 Distribución de probabilidad para variables aleatorias continuas

Se dice que una variable aleatoria Y es continua si su función de distribución $F_y(y)$ es continua para $-\infty < y < \infty$. Toda función $F_y(y)$ tiene asociada una función de densidad $f_y(y)$, siempre que la distribución de probabilidad sea derivable para la variable aleatoria Y. La función $f(y)$ es un modelo teórico para la distribución de frecuencia de una población de medias.

Una variable aleatoria puede ser caracterizada a través de su valor esperado ecuación (1.8) que representa el valor con mayor probabilidad de ser obtenido, y por su varianza ecuación (1.9) la cual representa el rango dinámico de la variable aleatoria alrededor de su valor esperado.

$$E_y(y) = \int_{-\infty}^{\infty} y * f(y)dy \tag{1.8}$$

$$V_y(y) = \int_{-\infty}^{\infty} (y - E(y))^2 f(y)dy \tag{1.9}$$

Existen distribuciones de probabilidad ya tabuladas que facilitan el estudio de los eventos, en este trabajo se utilizará la función de distribución de Gaussiana [18].

Distribución de Gaussiana: la distribución de Gaussiana o normal es la distribución más usada, conocida por su forma de campana. Se define según la ecuación (1.10), donde μ representa el valor esperado de la distribución y σ su varianza.

Como se observa en la figura 1.5 esta distribución es simétrica con respecto a su media.

$$f_y(y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}, -\infty < y < \infty \tag{1.10}$$

El uso de esta distribución puede justificarse debido al teorema central del límite que establece que la superposición de eventos aleatorios tiende a tener una distribución Gaussiana [18].

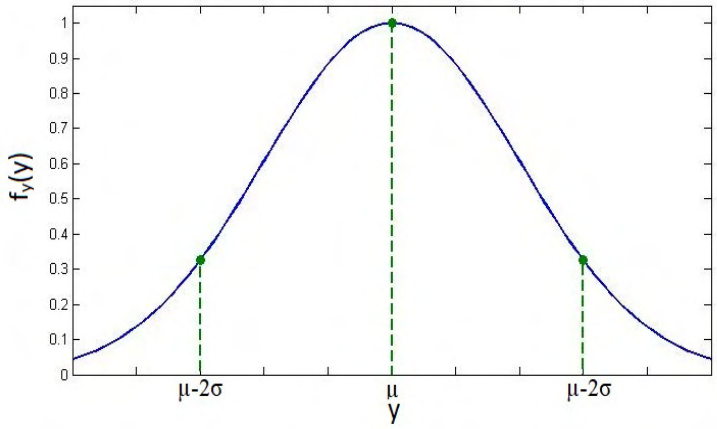


Figura 1.5. distribucíon de Gaussiana, tomado de [18].

1.4.2 Divergencia Kullback-Leibler

La divergencia de Kullback-Leibler KL es una medida no simétrica de similitud entre dos distribuciones de probabilidad. Esta divergencia entre dos funciones de P y Q viene dada por la ecuación (1.11), generalmente P representa la verdadera distribución de los datos o la distribución deseada, mientras que Q representa el modelo o descripción aproximado [19].

$$D_{KL} = \sum_i \left(P(i) ln \left(\frac{P(i)}{Q(i)} \right) \right) \tag{1.11}$$

CAPÍTULO 2
COMPRESIÓN DEL CONJUNTO DE ENTRENAMIENTO

En la literatura se encuentran diversos métodos de optimización de los algoritmos de clasificación, uno de ellos se enfoca en la compresión del conjunto de entrenamiento como un método para la optimización. En este Capítulo se describe detalladamente un método estocástico para realizar la compresión del conjunto de entrenamiento que basa su funcionamiento el método matemático de minimización del gradiente descendente.

2.1 COMPRESIÓN DEL CONJUNTO DE ENTRENAMIENTO

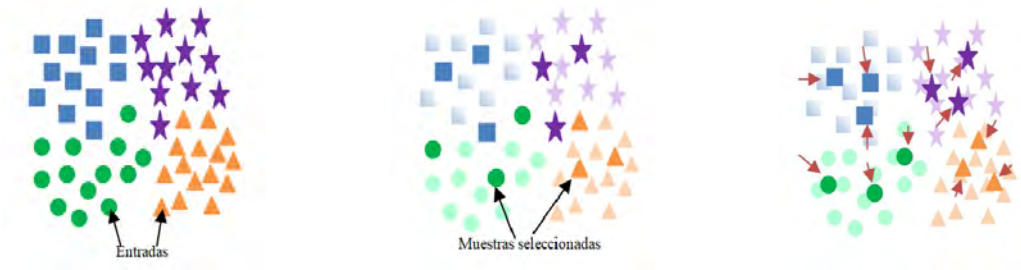
La forma más simple de implementar una compresión del conjunto de entrenamiento es submuestrear el conjunto, tal como se hace en el algoritmo propuesto por Hart [13], el cual usa la regla de vecinos mas cercanos condensados. Este método divide los datos con sus etiquetas correspondientes en dos grupos llamados **S** y **T**, las primeras muestras del conjunto de entrenamiento son colocadas en **S** y el resto se colocan en **T**. Posteriormente se clasifican los elementos de **T** usando a **S** como referencia, y cada vez que una muestra es clasificada erróneamente se agrega a **S**. En el método propuesto por Angiulli [6] las primeras muestras de **S** están ubicadas en los centroides de cada clase y las muestras de **T** se agregan iterativamente a **S**. Los métodos anteriores suponen seleccionar los vectores que mejor representan los datos eliminando redundancia. En la compresión estocástica del conjunto de entrenamiento (SNC) se toma inicialmente un porcentaje de los vectores de cada clase, después de seleccionados, estos vectores son optimizados hasta obtener aquellos que mejor representen los datos y no necesariamente deben estar dentro del conjunto de entrenamiento inicial [2]. El método desarrollado en este trabajo está basado en la compresión estocástica del conjunto de entrenamiento. Este método ofrece como ventaja poder comprimir el con-

junto aún más que los dos métodos anteriores debido a que la relación de compresión es independiente de la base de datos.

2.2 COMPRESIÓN ESTOCÁSTICA DEL CONJUNTO DE ENTRENAMIENTO

Este método propone, a partir de un conjunto de entrenamiento inicial obtener un conjunto mucho más pequeño y sintetizado de registros que minimiza el error estocástico en el entrenamiento del algoritmo K-NN. En este algoritmo se submuestrea de manera uniforme el conjunto de entrenamiento y posteriormente se optimiza las posiciones de los vectores para minimizar el error en el proceso de entrenamiento.

En la compresión estocástica del conjunto de entrenamiento (SNC, del inglés *Stochastic Neighbor Compression*) se utilizan distribuciones estocásticas para reducir un conjunto de entrenamiento de tamaño inicial n a uno con m vectores, con m mucho menor que n. El nuevo conjunto de entrenamiento $\mathbf{Z} = [\mathbf{z}_1, ..., \mathbf{z}_m]$ con etiquetas $\hat{y}_1 ... \hat{y}_m$ se obtiene de submuestrear uniformemente m vectores de $\mathbf{X}$ (conjunto de entrenamiento original), manteniendo sus etiquetas, las cuales se conservarán mientras los vectores de $\mathbf{Z}$ se optimizan. En la figura 2.1 se muestra un ejemplo de aplicar SNC sobre un conjunto de datos de dos dimensiones, donde, los datos de entrada se muestran en la figura 2.1(a), el efecto de submuestrear uniformemente en la figura 2.1(b) y los resultados después de la optimización en la figura 2.1(c).



(a) **(b)** **(c)**

Figura 2.1. Ilustración del algoritmo SNC, tomado de [2]. (a) Conjunto de datos de entrada. (b) Datos submuestreados uniformemente. (c) Datos después de la optimizació.

Se define la probabilidad de que un elemento de entrada $\mathbf{x}_i$ tome un vector $\mathbf{z}_j$ del conjunto de entrenamiento como su vecino más cercano de acuerdo a la ecuación (2.1), la cual se deriva

de la ecuación propuesta por Hinton y Roweis en *Stochastic Neighbor Embedding* (SNE) [20]. En el artículo, proponen una aproximación probabilística para la tarea de clasificación, con este fin una distribución Gaussiana es centrada en cada muestra de $\mathbf{X}$, y la función de densidad de esta distribución es utilizada para generar una distribución de probabilidad de todos los posibles vecinos de esta muestra. La finalidad principal de SNE es describir un espacio de muchas dimensiones en uno con pocas dimensiones.

$$p_{ij} = \frac{e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_j\|^2}}{\sum_{k=1}^{m} e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_k\|^2}} , \tag{2.1}$$

donde, γ representa la inversa de las desviación estándar o dispersión de los datos y $\| \mathbf{x}_i - \mathbf{z}_j \|$ denota la distancia Euclidiana del vector residuo, definida por $\sqrt{\sum_{k=1}^{d} (\mathbf{x}_i - \mathbf{z}_j)_k^2}$.

Como se observa en la ecuación (2.2) la probabilidad de que una muestra $\mathbf{x}_i$ se encuentre más cerca de una muestra en $\mathbf{Z}$ es una función de las distancias Euclidianas de la muestra $\mathbf{x}_i$ a la respectiva j-ésima muestra en $\mathbf{Z}$.

La ecuación (2.2) expresa la probabilidad de que un elemento $\mathbf{x}_i$ sea clasificado correctamente, donde γ representa la dispersión de los datos. Esta expresión es tomada de *Neighbourhood Components Analysis* (NCA) [21]. En esta ecuación se toma la probabilidad de que un elemento sea clasificado correctamente como la suma de todas las probabilidades en las que este elemento es comparado con uno de su misma clase. El propósito principal de NCA es aprender la distancia de Mahalanobis, a diferencia de otros métodos en éste no se realizan suposiciones sobre la forma en que las clases están distribuidas.

$$p_i = \sum_{j : y_i = y_j} p_{ij} \tag{2.2}$$

Idealmente, todo elemento debería ser clasificado correctamente, lo que implica que $p_i = 1$ para todo $\mathbf{x}_i \in \mathbf{X}$, no obstante, la distribución de probabilidad real está muy lejos de esta suposición. Como consecuencia, se utiliza la divergencia de Kullback-Leibler (KL) la cual compara de manera no simétrica el grado de similitud entre dos distribuciones de probabilidad, en este caso la distribución ideal es P y la real Q [21], como resultado se obtiene la ecuación (2.3).

$$KL(1||p_i) = -ln(p_i) \tag{2.3}$$

El objetivo principal de SNC es posicionar cada elemento de $\mathbf{Z}$ de manera que la mayor cantidad de entradas puedan ser clasificadas correctamente. En otras palabras, es necesario que $\mathbf{p}_i$ sea tan cercano a 1 como sea posible para todos los elementos $\mathbf{x}_i$ pertenecientes a $\mathbf{X}$, con este fin se define la función costo ecuación (2.4):

$$\mathcal{L}(\mathbf{Z}) = -\sum_{i=1}^{n} ln(p_i) \ , \tag{2.4}$$

como la suma de todas las divergencias KL ecuación (2.3) para todos los datos pertenecientes a $\mathbf{X}$.

Con el fin de alcanzar el objetivo SNC es necesario minimizar la función costo, para esto se utilizan métodos de primer orden con funciones de costo convexas. En especifico se utiliza el método del gradiente descendente (SD). Para poder expresar el gradiente de la función costo de manera más sencilla se definen las funciones $\mathbf{Q} \in \Re^{n \times m}$ y $\mathbf{P} \in \Re^{n \times m}$:

$$\mathbf{Q}_{ij} = (\delta_{y_i,y_j} - p_i) \tag{2.5}$$

$$\mathbf{P}_{ij} = \frac{p_{ij}}{p_i} \tag{2.6}$$

donde, δ_{y_i,y_j} representa la función Delta Dirác que toma el valor de 1 únicamente si $y_i = y_j$. Los procedimientos para obtener la derivada de la función costo ecuación (2.4) con respecto a un elemento $\mathbf{z}_j^d$ se muestran en el apéndice A ecuación (2.7):

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}}[\mathbf{z}_j^d] = \sum_{i=1}^{n} 2\gamma^2 x_i^d \mathbf{Q}_{ij}\mathbf{P}_{ij} - 2\gamma^2 z_j^d \mathbf{Q}_{ij}\mathbf{P}_{ij} \ , \tag{2.7}$$

donde, x_i^d representa el valor del vector $\mathbf{x}_i$ perteneciente a la matriz $\mathbf{X}$ en la dimensión d y z_j^d representa el valor del vector $\mathbf{z}_j$ perteneciente a la matriz $\mathbf{Z}$ en la dimensión d.

Ampliando esta derivada a todos los elementos de $\mathbf{Z}$ se obtiene la ecuación (2.8):

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} = -2\gamma^2(\mathbf{X}(\mathbf{Q} \circ \mathbf{P}) - \mathbf{Z}\Delta((\mathbf{Q} \circ \mathbf{P})^\top 1_n)) \ , \tag{2.8}$$

donde, $\circ$ indica el producto Hadamard[1], en el lenguaje de programación MATLAB este producto se denota como .*, $\mathbf{1}_n$ es un vector n-dimensional donde cada elemento del vector toma el valor de 1, y $\Delta(\mathbf{y})$ indica posicionar el vector $\mathbf{y}$ en la diagonal principal de una matriz de ceros (0) de dimensión $m \times m$ siendo m la dimensión del vector $\mathbf{y}$.

2.2.1 Distancia de Mahalanobis

Para agregar flexibilidad al método SNC se modifica ecuación (2.1) y ecuación (2.2) incluyendo la distancia de Mahalanobis a través de la matriz de transformación $\mathbf{A}$, tal como se propone en NCA [21], definiendo la probabilidad p_{ij} como:

$$p_{ij} = \frac{e^{-\|\mathbf{A}(\mathbf{x}_i - \mathbf{z}_j)\|^2}}{\sum_{k=1} e^{-\|\mathbf{A}(\mathbf{x}_i - \mathbf{z}_k)\|^2}} \tag{2.9}$$

La matriz $\mathbf{A}$ puede tener diferentes formas, por ejemplo, puede ser rectangular lo que permite una reducción de dimensión, diagonal, es decir $\mathbf{A} = \Delta$ con cada elemento de la diagonal definido apropiadamente para escalar las características contenidas en el vector o de la forma $\mathbf{A} = \gamma^2\mathbf{I}$, donde, $\mathbf{I}$ es la matriz identidad con dimensión d, lo cual permite la optimización del parámetro γ. Como consecuencia del uso de esta matriz es posible obtener el gradiente de la ecuación (2.4) con respecto a $\mathbf{A}$ dando origen a la ecuación (2.10). El proceso para obtener esta derivada se muestra en el Apéndice B.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = 2\mathbf{A}\sum_{i=1}^{n}\sum_{j=1}^{m}\mathbf{P}_{ij}\mathbf{Q}_{ij}\mathbf{v}_i j^{\mathsf{T}}\mathbf{v}_i j \ , \tag{2.10}$$

donde $\mathbf{v}_i j = ((\mathbf{x}_i - \mathbf{z}_j)$.

El gradiente con respecto a $\mathbf{Z}$ se modifica sustituyendo γ^2 por $\mathbf{A}^{\mathsf{T}}\mathbf{A}$, como se muestra en la ecuación (2.11).

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} = -2\mathbf{A}^{\mathsf{T}}\mathbf{A}(\mathbf{X}(\mathbf{Q}\circ\mathbf{P}) - \mathbf{Z}\Delta((\mathbf{Q}\circ\mathbf{P})^{\mathsf{T}}1_n)) \tag{2.11}$$

En este trabajo se utiliza $\mathbf{A} = \gamma^2\mathbf{I}$, debido a que es la opción computacionalmente menos costosa ya que al tener esta estructura se evita la multiplicación de los datos por la matriz

[1]Producto Hadamard: es un producto binario en el que se toman dos matices de las misma dimensión para generar otra matriz en la cual cada elemento i, j es el producto de los elemento i, j de las matrices originales

A. El valor de γ afecta sustancialmente los resultados. Debido a que el método que se utiliza supone funciones convexas es necesario primero fijar un valor inicial para γ a través de una búsqueda exhaustiva del valor óptimo. Además la previa optimización de este valor permite una convergencia más rápida de los valores de $\mathbf{Z}$, como se mostrará en el Capítulo 4.

2.2.2 Implementación

El Algoritmo 2.1 presenta el pseudocódigo utilizado para realizar la búsqueda exhaustiva de γ, de éste se selecciona el valor de γ que genere menor error en la etapa de pruebas (γ_b). Después de realizar el barrido de γ se ejecuta el pseudocódigo mostrado en el Algoritmo 2.2 para optimizar el valor de γ (γ_o). Finalmente, se optimiza $\mathbf{Z}$ con el Algoritmo 2.3, para ello se minimiza la ecuación (2.4) por medio de uso del método del gradiente descendente (SD).

Para la implementación del SD se utiliza una función de código abierto para MATLAB llamada *minimize* disponible en [22]. Esta función utiliza una combinación de métodos en cada iteración, para calcular la dirección de búsqueda se utiliza la versión de Pollack-Ribiere de SD, las líneas de búsqueda son calculadas con aproximaciones polinómicas cuadráticas y cúbicas, para detener las líneas de búsqueda se utiliza el criterio de Wolfe-Pow, por último, se utiliza el método de la relación de pendiente para calcular los pasos iniciales.

Algoritmo 2.1: Pseudocódigo para la búsqueda exhaustiva de γ

```
Entrada {X, y}; conjunto de datos de tamaño n;
Inicializar Z,submuestreando uniformemente las clases, extrayendo m vectores de X;
γ = 0;
for i=0:k do
    γ = γ + 0.1;
    Optimizar Z con el γ respectivo utilizando la ecuación (2.11);
end
Regresar Z
```

Algoritmo 2.2: Pseudocódigo optimización γ

```
Entrada {X, y}; conjunto de datos de tamaño n;
Inicializar Z,submuestreando uniformemente las clases, extrayendo m vectores de X;
γ = γ_b;
Optimizar γ con la ecuación (2.10);
Regresar γ optimizado (γ_o)
```

Algoritmo 2.3: Pseudocódigo optimización **Z**

Entrada $\{\mathbf{X}, \mathbf{y}\}$; conjunto de datos de tamaño n;

Inicializar **Z**,submuestreando uniformemente las clases, extrayendo m vectores de **X**;

$\gamma = \gamma_o$;

Optimizar **Z** con la ecuación (2.11);

Regresar **Z** optimizada

CAPÍTULO 3
BASES DE DATOS

En este trabajo se utilizaron cuatro bases de datos diferentes. La primera una base de datos usada para el reconocimiento de letras mayúsculas que contiene pocas dimensiones. La segunda y tercera base de datos están enfocadas al reconocimiento de audio y reconocimiento de dígitos escritos a mano, estas bases de datos presentaban más dimensiones que la primera, por lo cual fue necesario aplicar un procesamiento previo a los datos. Finalmente, la cuarta base de datos es utilizada para el reconocimiento facial, ésta contiene una elevada dimensión, razón por la cual se aplicaron varios procedimientos para disminuir sus dimensiones. En este capítulo se describe detalladamente cada una de las bases de datos, así como los procedimientos aplicados en ellas para modificar sus dimensiones.

3.1 *LETTERS*

Letters es el nombre que recibe la base de datos compuesta por letras mayúsculas del alfabeto en inglés, disponible en [23]. La tarea del algoritmo K-NN consiste en identificar cada letra basándose en las características de la fuente, que serán descritas más adelante. Las imágenes de los caracteres provienen de 20 diferentes fuentes, la cuales fueron expresadas en términos de 16 atributos numéricos.

Las 20 fuentes fueron diseñadas por el Dr. Allen C. Hershey, un físico matemático perteneciente al laboratorio de armas navales de los Estados Unidos. Los nombres de las fuentes utilizadas en esta base de datos son los siguientes: HASTR, HCART, HCITA, HCROM, HCSCR, HFROM, HGENG, HGGER, HGITA, HIITA, HIROM, HISYM, HIUMT, HIMETE, HMUSI, HSROM, HSSCR, HTITA, HTROM Y HUMAT. Estas fuentes representan cinco tipos de trazados diferentes (simple, doble, triple, complejo y gótico) y seis tipos de

letras (bloque, *script*, cursiva, inglesa, italiana y alemana).

Las imágenes originales fueron generadas como un vector de coordenadas con información de los puntos finales de los segmentos de las líneas. Estos vectores fueron escalados y modificados de manera que las letras se pudieran expresar en una matriz rectangular de píxeles encendidos/apagados.

Cada imagen fue escaneada píxel a píxel para extraer 16 atributos numéricos, los cuales representan las características estadísticas primitivas de la distribución de los píxeles. Los atributos finalmente fueron escalados linealmente a un rango de enteros entre 0 y 15, representados asi por cuatro bits de información. Este procedimiento fue realizado por Frey P.W. y Slate D.J. en [24]. Los atributos, antes de ser escalados, son:

1. Posición horizontal: es la ubicación del centro del rectángulo más pequeño que se pueda dibujar con dos píxeles seguidos encendidos recorriendo la imagen de izquierda a derecha.
2. Posición vertical: es la ubicación del centro del rectángulo más pequeño que se pueda dibujar con dos píxeles seguidos encendidos recorriendo la imagen de abajo hacia arriba.
3. El ancho en píxeles de la imagen.
4. La altura en pixeles de la imagen.
5. El número total de píxeles encendidos.
6. La media de horizontal de todos los píxeles encendidos con respecto al píxel central de la imagen dividido entre el ancho de la imagen.
7. La media de vertical de todos los píxeles encendidos con respecto al píxel central de la imagen dividido entre la altura de la imagen.
8. La media cuadrática de las distancias horizontales medidas como en 6.
9. La media cuadrática de las distancias verticales medidas como en 7.
10. El producto medio de las distancias verticales y horizontales de cada píxel medidos como en 6 y 7.

Figura 3.1. Letra F de la base de datos *Letters*.

11. El valor medio cuadrático de las distancias horizontales por la distancia vertical de cada píxel encendido.

12. El valor medio cuadrático de las distancias verticales por la distancia horizontal de cada píxel encendido.

13. El número medio de bordes (un píxel encendido seguido por la izquierda o derecha de uno apagado) encontrados al recorrer la imagen de izquierda a derecha en todas las posiciones verticales.

14. La suma de las posiciones verticales de los bordes encontrados de la misma forma que en 13.

15. El número medio de bordes (un píxel encendido seguido por arriba o por abajo de uno apagado) encontrados al recorrer la imagen de abajo hacia arriba en todas las posiciones horizontales.

16. La suma de las posiciones horizontales de los bordes encontrados de la misma forma que en 15.

En la figura 3.1 se muestra una de las letras F almacenadas en la base de datos y en la Tabla 3.1 se muestran los parámetros de dicha letra después de haber sido escalados.

La base de datos contiene 20000 muestras de las cuales las primeras 16000 son tomadas típicamente como el conjunto de entrenamiento. De cada una de las letras del alfabeto se tienen aproximadamente 770 muestras.

El algoritmo SNC se aplicó directamente sobre la base de datos sin necesidad de usar algún pre-acondicionamiento previo.

Tabla 3.1. Atributos para la letra F figura 3.1 después de ser escalados

Parámetros Letra F			
Parámetro	Valor	Parámetro	Valor
1	6	9	5
2	9	10	10
3	5	11	5
4	4	12	7
5	3	13	3
6	10	14	9
7	6	15	6
8	3	16	9

Figura 3.2. Flujo de procesos para la optimización de *Letters*.

3.2 *ISOLET*

ISOLET (*isolet letter speech recognition*) creada por Fanty M. y Cole R. [25], disponible en [26], es una base de datos compuesta por 6238 muestras de audio, en cada muestra se almacena una letra del alfabeto en inglés. Se tiene un total de 240 muestras de cada una de las 26 letras el alfabeto y cada muestra está compuesta por 617 características.

Todas las letras en inglés, exceptuando W, contienen un simple intervalo sonoro conocido como *SON* (Ej. el /iy/ en T). Este sonido siempre existe y proporciona el soporte para la medida de la características de la letra. La consonante previa al sonido de la vocal se conoce como fricativo[1] (*STOP* o *FRIC*). En algunos casos no existe *STOP*, en estos casos, los 200 ms previos al intervalo sonoro son tratados como un solo segmento sonoro para la extracción de las características.

Las características de la base de datos son descritas a continuación:

1. Los primeros 96 valores corresponden a la transformada discreta de Fourier (DFT, del inglés *Discrete Fourier Transform*), para ésto, la consonante fue dividida en 3

[1]Fricativo: que se pronuncia acercando determinados órganos de la boca de manera que el aire pasa rozando entre ellos

segmentos, de cada segmento se extrajeron linealmente 32 valores promedio desde 0 hasta 8 kHz. Las entradas fueron normalizadas localmente entre 1.0 y 0.0.

2. Los coeficiente de la DFT asociados con el SON, el cual fue dividido en 7 segmentos y de cada uno se extrajeron linealmente 32 valores promedio desde 0 a 4 kHz, dando un total de 224 valores.

3. De los coeficientes de la DFT siguientes al SON, se extrajeron 32 valores promedio linealmente desde 0 hasta 8 kHz.

4. Del segundo y quinto segmento del SON se extrajeron 64 valores linealmente desde 0 hasta 8 kHz. Estos valores no son promedios en el tiempo.

5. Los coeficiente de la DFT del centro del SON, con la diferencia espectral más grande (32 valores).

6. La relación de cruces por cero en segmentos de 11 a 18 ms antes del SON, en 11 segmentos de igual duración durante el SON y en segmentos de 11 a 18 ms después del SON. Esto da un total de 33 valores.

7. La amplitud antes, durante y después del SON representada de igual manera que los cruces por cero (33 valores).

8. La amplitud del filtrado representada de la misma manera que la amplitud (33 valores).

9. La diferencia espectral (33 valores).

10. Dentro del SON el centro de masa espectral entre 0 a 1000 Hz, medido en 10 segmentos iguales (10 valores).

11. Dentro del SON el centro de masa espectral entre 1500 a 3500 Hz, medido en 10 segmentos iguales (10 valores).

12. El tono medio (1 valor).

13. Duración de la consonante antes del SON (1 valor).

14. Representación de alta resolución de la amplitud del SON (5 Valores).

15. La variabilidad del tono en el tiempo antes del SON (1 valor).

16. El estado del segmento después del SON (3 valores).

17. La mayor diferencia espectral en los 100 ms después del SON (1 valor).

18. El número de tonos en la consonante previa al SON (1 valor).

19. El número de tonos antes de la consonante previa al SON (1 valor).

20. Un indicador binario de la presencia de un STOP o FRIC (donde 1 representa la presencia y 0 la ausencia de este) corresponde a un solo valor.

Para el tratamiento de esta base de datos se utilizó el criterio propuesto por Weinberger K.Q. y Saul L.K. en [5], en el cual se reducen las dimensiones originales de los datos a través de la proyección de ellos en sus 169 principales componentes aplicando PCA, estas componentes son suficientes para conservar alrededor del 95 % de la energía original de las muestras. En la figura 3.3 se presentan los primeros 170 auto valores obtenidos de la aplicación PCA, donde se observa que la energía se concentra en los primeros auto valores. Después, sobre estos datos se aplicó SNC. Para realizar la reducción de dimensión se hizo a través de la transformación lineal $\hat{\mathbf{x}} = \mathbf{W}_{95\,\%}\mathbf{x}$, donde, $\mathbf{W}_{95\,\%} = [\mathbf{w}_1 \vdots \mathbf{w}_2 \vdots \mathbf{w}_3 \vdots ... \vdots \mathbf{w}_L]$ donde L esta dado por la ecuación 3.1:

$$L = \min_{\mathbf{N_1}} \left(\frac{\sum_{i=1}^{N_1} \lambda_i}{\sum_{j=1}^{d} \lambda_j} \times 100 \leqslant 95\,\% \right) , \tag{3.1}$$

donde, λ es la magnitud del autovalor.

El procedimiento aplicado se muestra en la figura 3.4 y como se observa después de aplicar PCA sobre la base de datos se utilizó SNC.

3.3 MNIST

MNIST del inglés *Modified National Institute of Standards and Technology* es una base de datos de gran tamaño, compuesta por 60000 muestras de dígitos escritos a mano, disponible en [27]. MNIST [28] fue creada a partir de una base de datos anterior (NIST), MNIST está

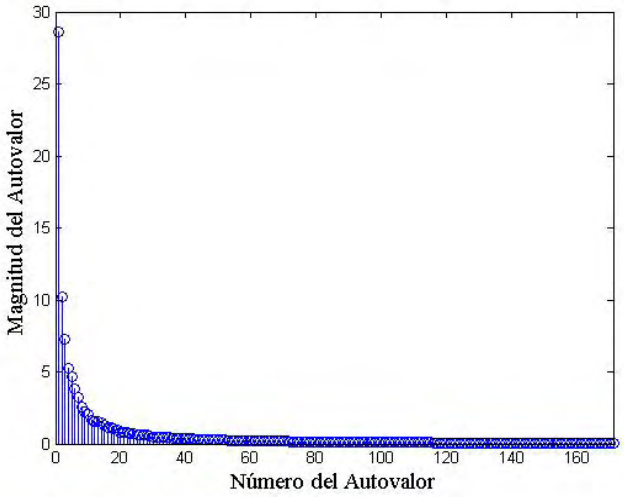
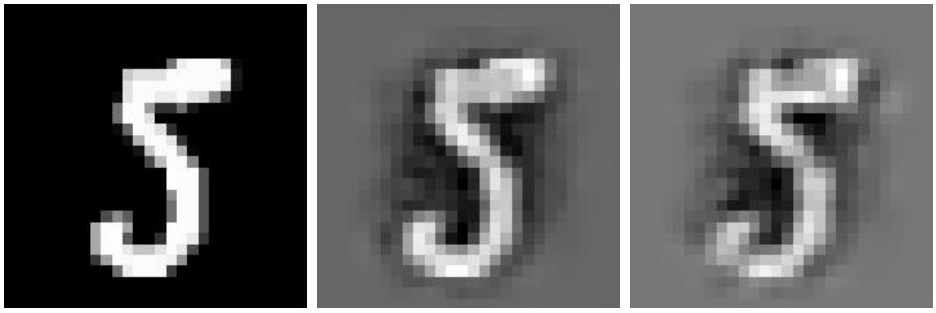


Figura 3.3. Autovalores de la base de datos *ISOLET*.

Figura 3.4. Flujo de procesos para la optimización de *ISOLET*.

compuesta por la mitad del conjunto de entrenamiento de NIST y la mitad del conjunto de pruebas de la misma base de datos. Los dígitos del 0 al 9 fueron escritos por estudiantes de bachillerato de los Estados Unidos y empelados de la oficina de censado del mismo país. Cada dígito se puede leer en una imagen de 28×28 píxeles, y se muestra en la base de datos como vectores de 784 valores, para formar este vector las columnas de píxeles de las imágenes son concatenadas.

Para el tratamiento de esta base de datos se utilizó el mismo criterio que en la base de datos *ISOLET*, esto es proyectando los datos en sus 153 principales componentes las cuales contienen el 95 % de la energía total de los datos. La figura 3.5 muestra como es la concentración de energía de los primeros autovalores. Para realizar la reducción de dimensión se hizo a través de la transformación lineal $\hat{\mathbf{x}} = \mathbf{W}_{95\,\%}\mathbf{x}$, donde, $\mathbf{W}_{95\,\%} = [\mathbf{w}_1 \vdots \mathbf{w}_2 \vdots \mathbf{w}_3 \vdots ... \vdots \mathbf{w}_L]$ donde L esta dado por la ecuación 3.1. En la figura 3.6 se observa los efectos de representar los datos solo por una parte de sus componentes principales donde, la figura 3.6(a) presenta el dígito original de la base de datos, la figura 3.6(b) el dígito representado por sus 153 componentes principales y en la 3.6(c) se presenta el dígito optimizado.



(a) (b) (c)

Figura 3.6. Dígito de la base de datos MNIST. (a) Dígito original. (b) Dígito después de la aplicación de PCA. (c) Dígito optimizado.

Figura 3.7. Flujo de procesos para la optimización de MNIST.

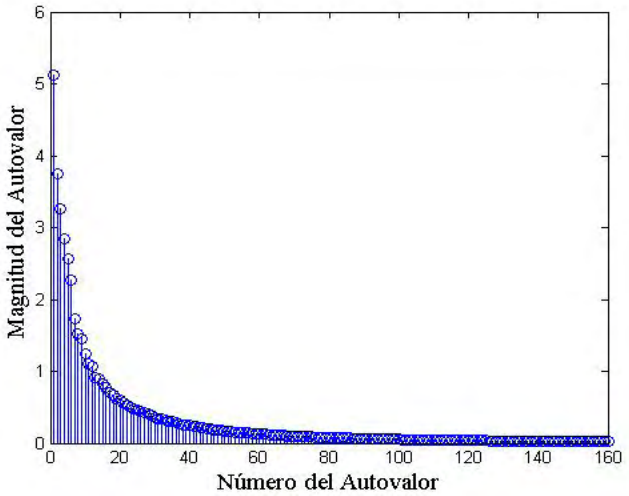


Figura 3.5. Autovalores de la base de datos MNIST.

El procedimiento aplicado sobre la base se datos de muestra en la figura 3.7,y como se observa en la misma después de aplicar PCA sobre la base de datos se aplicó SNC.

3.4 *YALEFACES*

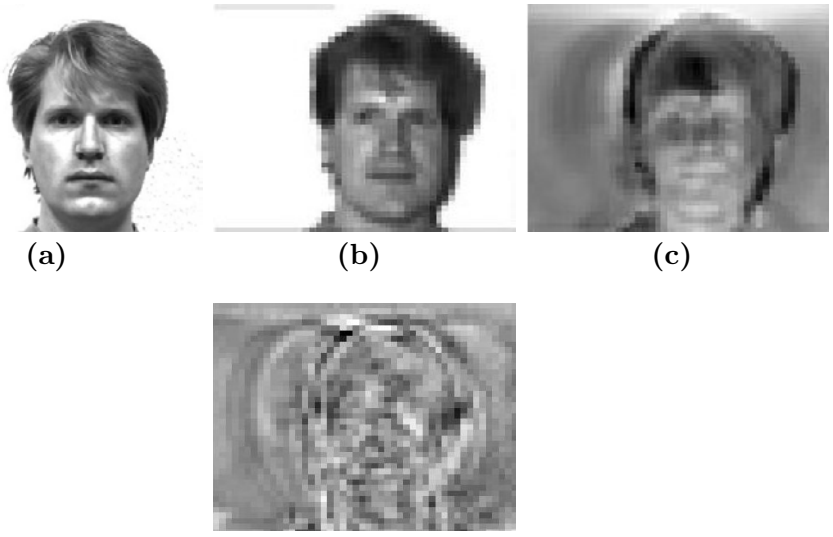
YaleFaces consiste en una colección de imágenes de 15 sujetos en escala de grises, disponible en [29]. Existen 11 imágenes por sujeto, cada una con diferentes expresiones faciales o ca-

Figura 3.8. Flujo de procesos para la optimización de *YaleFaces*.

racterísticas: frontal con luz en el centro, con lentes, feliz, iluminación desde la izquierda, sin lentes, normal, iluminación desde la derecha, triste, adormecido, sorprendido y parpadeando. Las imágenes originales son de 243×320 píxeles [30]. La base de datos cuenta con un total de 165 muestras de dimensión 77760. La tarea del algoritmo K-NN consiste en clasificar a cada uno de los sujetos.

Para el tratamiento de esta base de datos se siguió los procedimientos propuestos en LMNN [5] y en SNC [2]. Primero las imágenes fueron escaladas a 42×48 píxeles, para ésto se utilizó una herramienta de MATLAB (*imresize*). Después, para poder formar la matriz sobre la cual se aplicaría PCA se concatenaron las columnas de píxeles una a continuación de la otra, transformando cada imagen en un vector de entrada con 2.016 atributos. Luego se aplicó PCA para eliminar las primeras cinco componentes (las cuales están relacionadas con la mayor variación de iluminación en la imagen), posteriormente, aplicando el criterio explicado en [5], el cual especifica que a través del uso de LMNN en datos con dimesión mayor a 200 ($d \geqslant 200$) se reducen las dimensiones de los datos a 100. Este procedimiento se puede observar el la figura 3.8

En la figura 3.9 se observa el efecto de aplicar las etapas de la figura 3.8 en una imagen cualquiera de la base de datos, en la figura 3.9(a) se presenta la imagen original. En la figura 3.9(b) se observa los efectos de aplicar el escalamiento sobre ella, en esta etapa es posible ver como la foto pierde calidad, en la figura 3.9(c) se muestra el efecto de eliminar las primeras 5 componentes principales, en ésta se observa como se atenúa el contraste de la imagen y, aunque la imagen del sujeto no es clara se llega a observar la silueta del mismo. Finalmente, en la figura 3.9(d) se muestra el rostro obtenido después de aplicar SNC.



(a) (b) (c)

(d)

Figura 3.9. Foto del sujeto 1: (a) Imagen original, (b) Imagen escalada, (c) Imagen despues de extraer la 5 primeras componentes principales y (d) Imagen optimizada.

CAPÍTULO 4
SIMULACIÓN Y RESULTADOS

Recurriendo a la aplicación de los algoritmos propuestos anteriormente, este cuarto y último capítulo se enfoca en presentar los resultados obtenidos al implementar el método de optimización en cada base de datos. En la simulación y resultados se analizan una serie de características. Primero se estudia el desempeño de la optimización de los datos en la etapa de pruebas o inferencia a través del uso del algoritmo de clasificación K-NN utilizando como medida de desempeño el número de muestras mal clasificadas. Seguidamente se compara el tiempo de entrenamiento por iteración en las distintas relaciones de compresión seleccionadas. Luego, se analiza la convergencia y la importancia de optimizar el parámetro γ en la optimización del conjunto de entrenamiento. Finalmente, se comprueba el objetivo general de este trabajo, la aceleración del algoritmo de clasificación K-NN utilizado como medida de desempeño un coeficiente de aceleración.

4.1 DESEMPEÑO DEL ALGORITMO

En el Capítulo 2 se planteó la compresión del conjunto de entrenamiento, en donde se optimizó el conjunto de entrenamiento con el objetivo de que la función costo tendiera a cero. En esta sección se presentan los resultados del desempeño de los conjuntos de entrenamientos optimizados obtenidos como resultado de implementar el Algoritmo 2.3 en la etapa de inferencia o pruebas. Este algoritmo fue aplicado sobre las cuatro bases de datos descritas en el Capítulo 3. La tarea de aprendizaje fue especificada para cada una de las bases de datos, para *Letters* consintió en identificar cada una de las letras del alfabeto a partir de imágenes de las letras, para *ISOLET* se identificaron las letras del alfabeto a partir de grabaciones de audio donde se almacena la letra pronunciada, para MNIST se determinó el valor de los dígitos a partir de fotografías de los dígitos escritos a mano y para *YaleFaces* se identificó

Tabla 4.1. Características de las bases de datos utilizadas

Base de datos	n	Número de clases	$d(df)$
Letters	16000	26	16
ISOLET	6238	26	617(169)
MNIST	60000	10	784(153)
YaleFaces	165	15	77760(100)

a los individuos a partir de fotografías. En la Tabla 4.1 se muestran las estadísticas de las bases de datos, donde, n es el numero de muestras iniciales, d la dimensión inicial y df la dimensión después de aplicar la reducción de dimensión especificada en el Capítulo 3.

Con el fin de demostrar la efectividad del algoritmo y poder observar la variación de los resultados con respecto a la variación de la relación de compresión se seleccionaron 5 valores para la misma 1 %, 2 %, 4 %, 8 % y 16 %, del número de muestras iniciales (n).

Todos los experimentos realizados en este trabajo fueron ejecutados en un procesador Quad-Core AMD 2376 con 2.38 GHz de frecuencia de reloj y con 32 GB de memoria RAM.

Para realizar las pruebas se utilizó la regla del vecino más cercano (1-NN, del inglés *1-nearest neighbor*), utilizando como conjunto de pruebas el conjunto inicial de entrenamiento ($\mathbf{X}$). En esta sección se utilizó como medida de desempeño el porcentaje de muestras del conjunto de entrenamiento mal clasificadas dado por la ecuación (4.1). Los resultados obtenidos aplicando la optimización se compararon con los resultados que se obtuvieron al aplicar un simple submuestreó sobre la base de datos utilizando la misma regla de clasificación y medida de desempeño.

$$Error = \frac{número\ de\ muestras\ mal\ clasificadas}{n} \times 100 \tag{4.1}$$

En la figura 4.1 se muestran los errores en la etapa de pruebas utilizando como conjunto de entrenamiento el obtenido por el método propuesto (curva verde) y el conjunto de entrenamiento submuestreado sin optimizar (curva azul). En esta figura se observa que el efecto de simplemente submuestrear los datos arroja errores muy altos especialmente, en las bases de datos *Letters* y *ISOLET* y *YaleFaces*, probando que es necesaria una optimización si se desea submuestrear el conjunto de entrenamiento con el fin de acelerar el algoritmo de

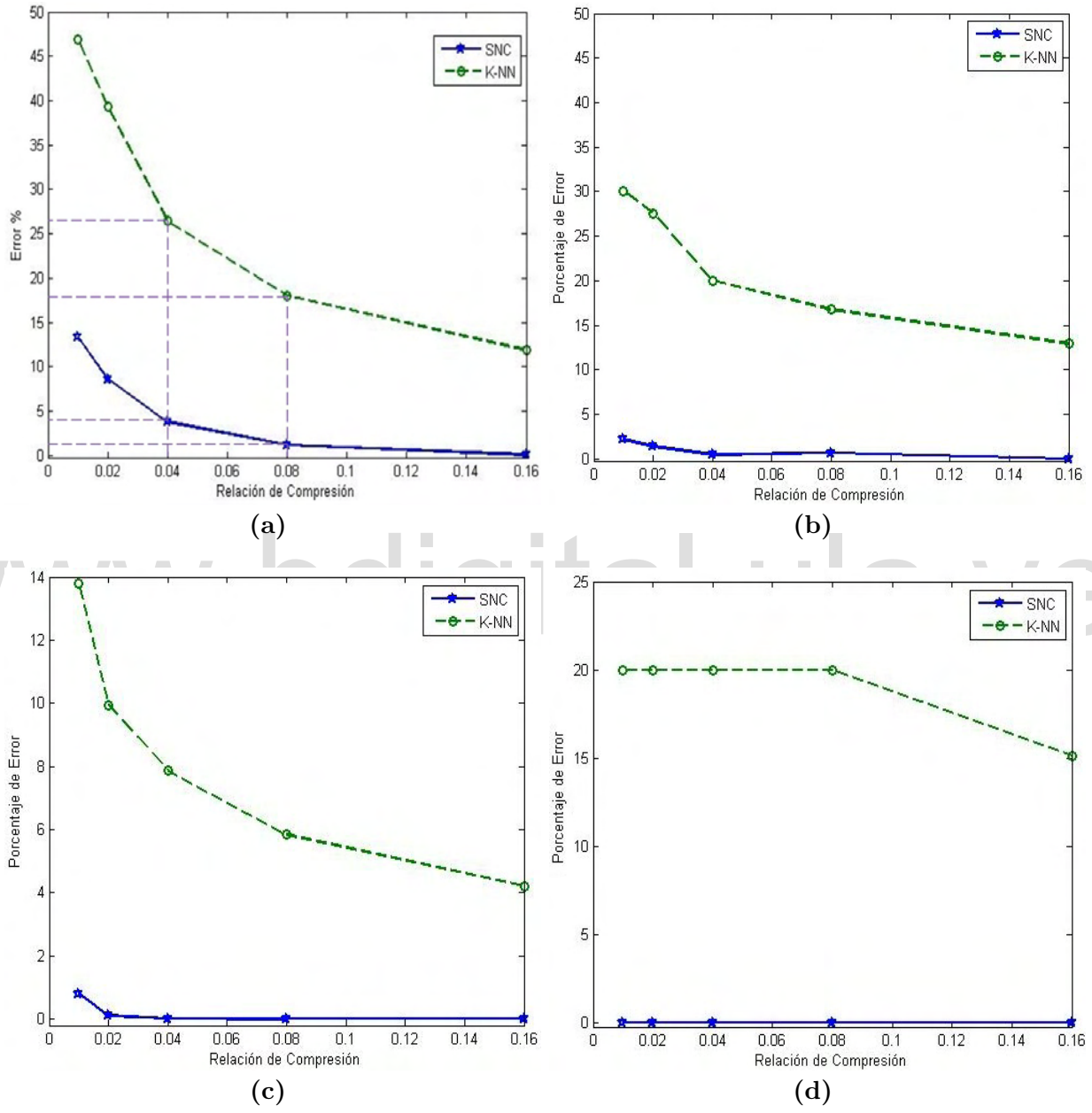


Figura 4.1. Error en la etapa de pruebas de 1-NN después de la compresión del conjunto de entrenamiento y depués de optimizar. (a) *Letters*. (b) *ISOLET*. (c) MNIST. (d) *YaleFaces*.

clasificación.

Del comportamiento de las curvas se puede concluir que al aumentar el tamaño del conjunto de entrenamiento el error disminuye, este comportamiento es el esperado debido a que cuanto más grande sea el conjunto de entrenamiento mayor es la probabilidad que una muestra encuentre su verdadera clase como vecino más cercano. En las bases de datos *Letters*, *ISOLET* y MNIST figuras 4.1(a), 4.1(b), 4.1(c) y 4.1(d), respectivamente, se observa que en el caso de aplicar solamente submuestreo de los datos (curva verde), que al duplicar el porcentaje de muestras tomadas, es decir, reducir la relación de compresión a la mitad, el error disminuye de forma notable. Por ejemplo, para *Letters* figura 4.1(a) cuando se pasa de una compresión del 4 % al 8 %, se obtuvo que el error disminuyó en un 8.41 %. En la base de datos *YaleFaces* figura 4.1(d) tiene un comportamiento distinto debido a que el número de muestras seleccionadas con compresión de 1 %, 2 %, 4 % y 8 % es el mismo, razón por la cual, para estas relaciones de compresión, se obtiene el mismo error.

Analizando la curva asociada a la optimización (curva azul) se presenta el mismo comportamiento, al aumentar la relación de compresión el error disminuye. En el caso de *Letters* la curva es mas suave, por ejemplo, al pasar de una relación del 4 % al 8 % el error disminuye un 2.56 % figura 4.1,(a) en las curvas de *ISOLET* y MNIST el cambio es casi imperceptible mientras que en *YaleFaces* no existen cambios al variar la relación de compresión, en esta base de datos, una vez se optimizó el conjunto de entrenamiento, el error fue de cero independientemente de la relación de compresión.

Los tiempos de ejecución por iteración en cada base de datos se muestran en la figura 4.2. Para las bases de datos *Letters*, *ISOLET* y MNIST el incremento en el tiempo de entrenamiento es lineal y directamente proporcional al incremento del conjunto de entrenamiento, en el caso de estas gráficas a medida que el tamaño del conjunto de entrenamiento se duplica el tiempo de entrenamiento también lo hace. Por ejemplo, en *Letters* cuando la relación de compresión varia de un 4 % a un 8 % el tiempo de entrenamiento aumenta de 3858 s (1 h 4 min) a 7743 s (2 h 9 min). Por otro lado, la figura 4.2(d) muestra que el tiempo de entrenamiento en compresiones del 1 %, 2 %, 4 % y 8 % es el mismo debido a que el tamaño del conjunto de entrenamiento no varió para estas compresiones.

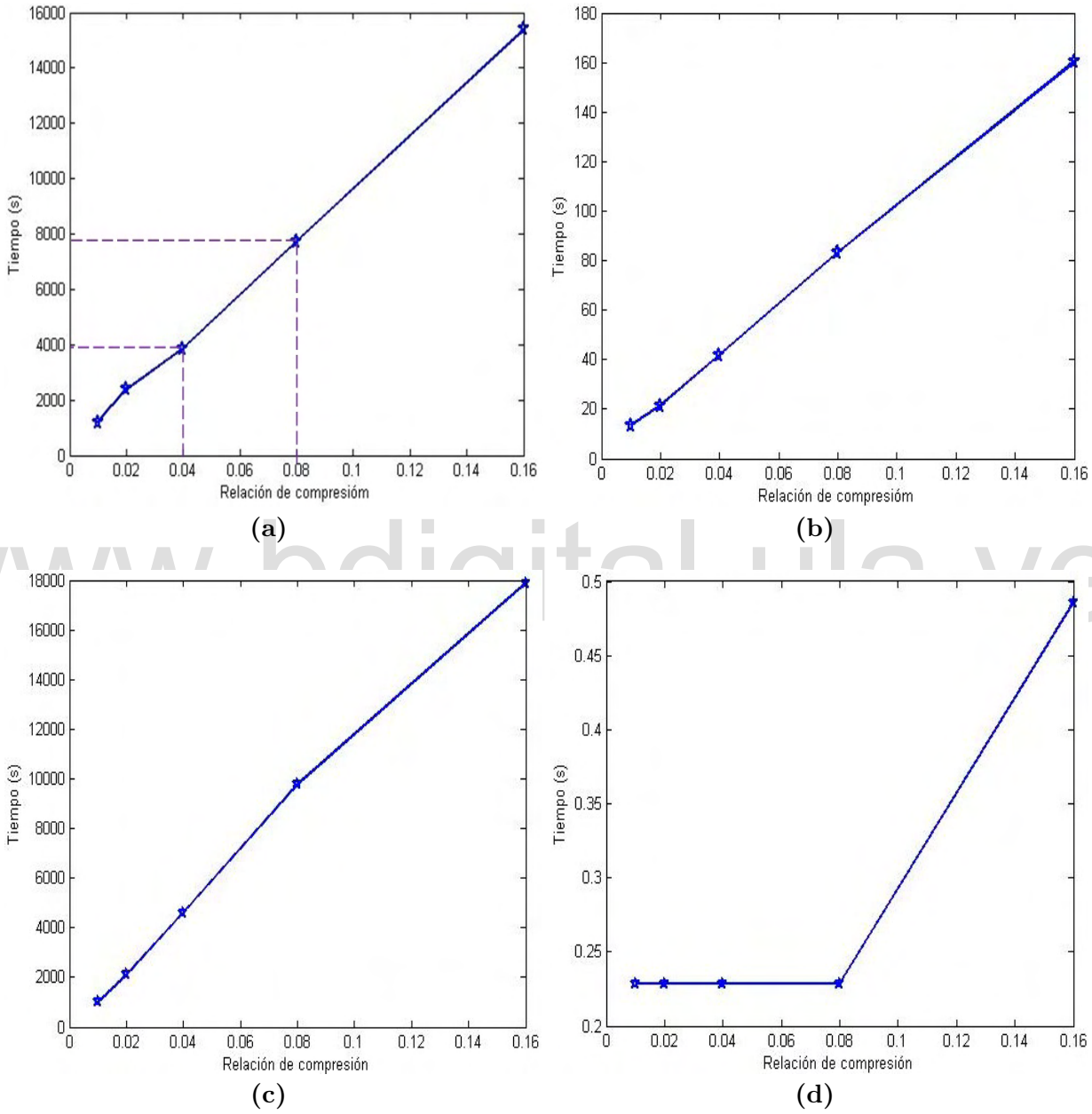


Figura 4.2. Tiempo de entrenamiento por iteración de SNC. (a) *Letters*. (b) *ISOLET*. (c) MNIST. (d) *YaleFaces*.

Comparando los resultados mostrados en la figura 4.1 y la figura 4.2 la diferencia del error obtenido variando la relación de compresión del 1 % al 16 % para las bases de datos *ISOLET*, MNIST y *YaleFaces* no es significativo, sin embargo, la variación del tiempo que existe para el entrenamiento de los conjuntos es bastante elevada, haciendo injustificable tomar relaciones de compresión mayores al 4 % en estas bases de datos. En el caso de *Letters* el error para las compresiones menores al 8 % esta por debajo del 5 %, por esta razón, a pesar que los tiempos de entrenamiento son sustancialmente grades, superando la hora, es necesario tomar una relación de compresión no mayor al 8 % para tener un error aceptable.

4.2 ANÁLISIS DECONVERGENCIA

Un método numérico converge cuando la diferencia entre el valor obtenido y el valor deseado es lo suficientemente pequeño. En el método propuesto se tomo como error para aplicar el criterio de convergencia la diferencia entre el porentaje de muestras mal clasificadas (error) que se tiene al utilizar el conjunto de entrenamiento obtenido en una iteración i en el algoritmo 1-NN y el que se obtiene al utilizar el conjunto de la siguiente iteración $(i+1)$. La rapidez de convergencia está relacionada con el número de iteraciones necesarias para que el error entre iteraciones sea lo suficientemente pequeño.

La figura 4.3 muestra la relación que existe entre el error en la etapa de pruebas y el número de iteraciones realizadas para la optimización del conjunto de entrenamiento.

En esta sección se define el codo de la curva como el número de iteraciones donde su incremento no significa una disminución sustancial del error. En la figura 4.3(a) se observa como en la base de datos *Letters*, para compresiones de 1 %, 2 % y 4 % realizar más de 10 iteraciones no mejora significativamente el resultado, obteniendo un avance aproximado del $1,5$ % al realizar 10 y 15 iteraciones, esta mejora no justifica el tiempo que tarda el algoritmo al realizar las cinco iteraciones adicionales, en el caso de 8 % y 16 % el codo de a curva se encuentra en siete iteraciones. La figura 4.3(b) muestra las curvas asociadas a la base de datos *ISOLET*, en ésta figura se observa que el codo para las relaciones de compresión está ubicado en 10, 7, 5, 7 y 4 iteraciones respectivamente. En *ISOLET* destaca el hecho de que la curva asociada a la relación de compresión del 4 % (curva roja) converge más rápidamente

que la asociada a la del 8 % (curva azul claro), e incluso después de la primera iteración, siempre tiene un error menor. Una explicación para este particular comportamiento puede deberse a los vectores iníciales seleccionados fueron mejores, o que el algoritmo SNC optimizó especialmente bien este conjunto de entrenamiento. La figura 4.3(c) presenta las curvas de la base de datos MNIST, para relaciones de compresión del 1 % y 2 %, el codo se presenta en la quinta iteración, mientras que en 4 %, 8 % y 16 % el codo se encuentra en la segunda iteración, por lo que es suficiente hacer sólo 2 iteraciones. Finalmente, la figura 4.3(d) muestra que para relaciones de compresión de 1 % al 8 % son necesarias tres iteraciones para llegar al codo de la curva, mientras que para 16 % basta con realizar una iteración.

Una forma de analizar las clases que se están confundiendo es a través de la matriz de confusión. La matriz de confusión es una herramienta que permite visualizar el desempeño de un algoritmo de clasificación, en cada columna presenta el número de predicciones de cada clase y las filas presenta las clases reales de las muestras. Idealmente esta matriz de confusión solo tendría valores en su diagonal principal. La figura 4.4 muestra la evolución de la matriz de confusión, como una serie de imágenes en escala de grises. Para generar estas matrices se utilizó el conjunto de entrenamiento resultado del algoritmo luego de la primera iteración, la matriz de confusión generada se muestra en la esquina superior izquierda de la figura, luego de 3 iteraciones (arriba en el medio), 6 iteraciones (esquina superior derecha), 9 iteraciones (esquina inferior izquierda), 12 iteraciones (abajo en el medio) y 15 iteraciones (equina inferior derecha). En esta imagen los píxeles o cuadros representan los elementos de la matriz, el blanco representa los valores más altos, el negro los valores más bajos los grises representan valores intermedios.

La figura 4.4(a) presenta las matrices generadas en la base de datos *Letters* con una relación de compresión del 2 %, en la primera matriz se observa que se cometen muchos errores en el proceso de clasificación, mientras que en la última se observa como la diagonal principal toma valores mayores al verse más clara y los elementos fuera de la diagonal tienden a ser nulos. La figura 4.4(b) presenta el mismo efecto en la base de datos *ISOLET* con una relación de compresión del 1 %. En ambas figuras se observa como los pixeles grises fuera de la diagonal principal van disminuyendo a medida que aumenta el número de iteraciones.

En la base de datos *ISOLET* la letra “F” tiende a confundirse con la letra “S”, esto se

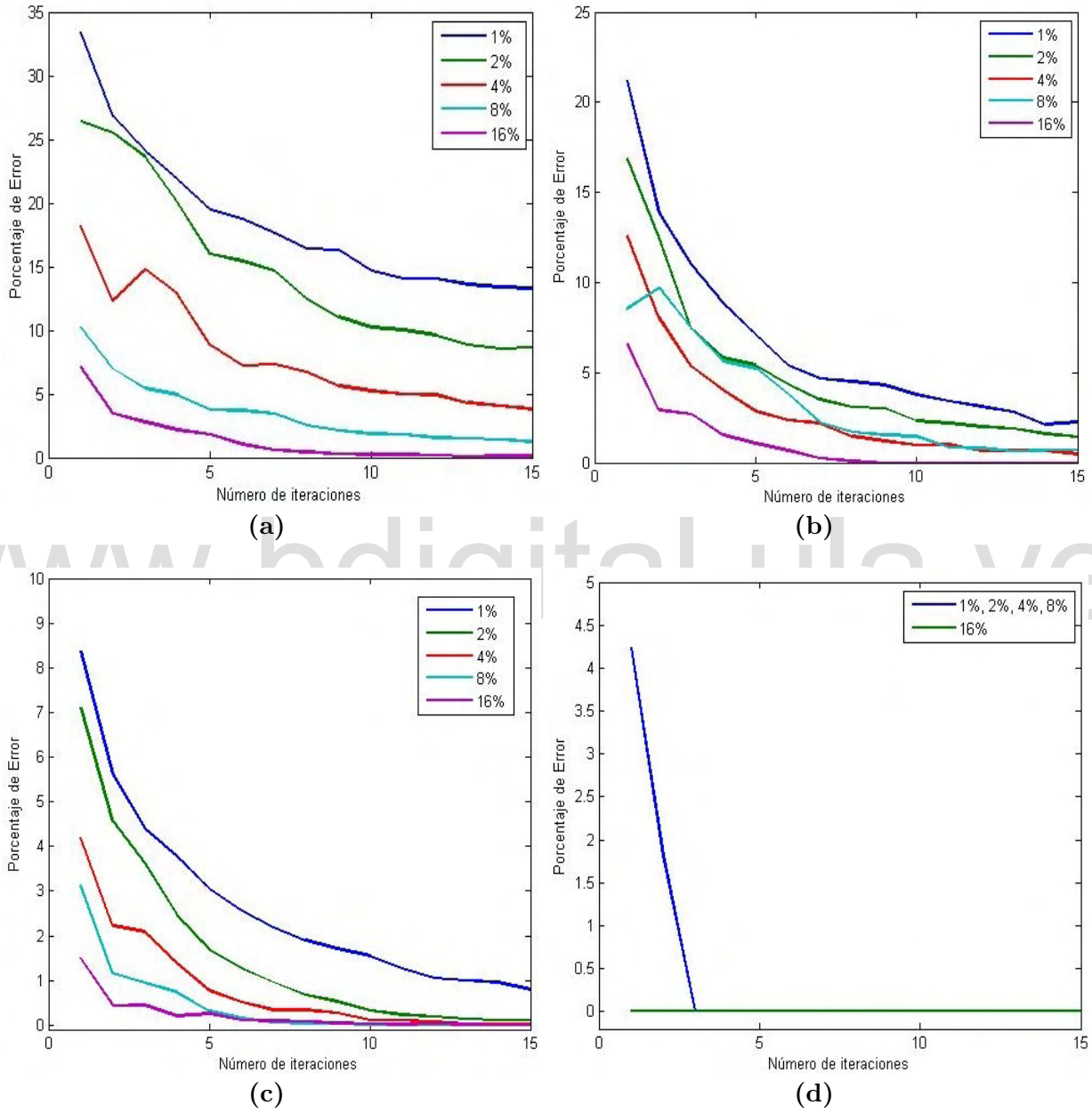
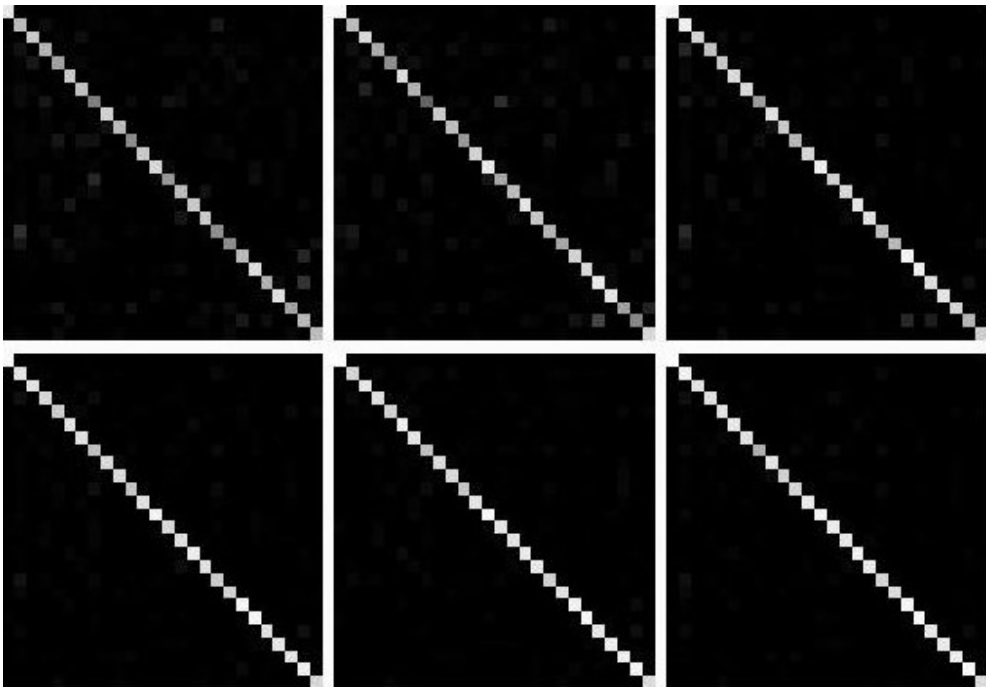
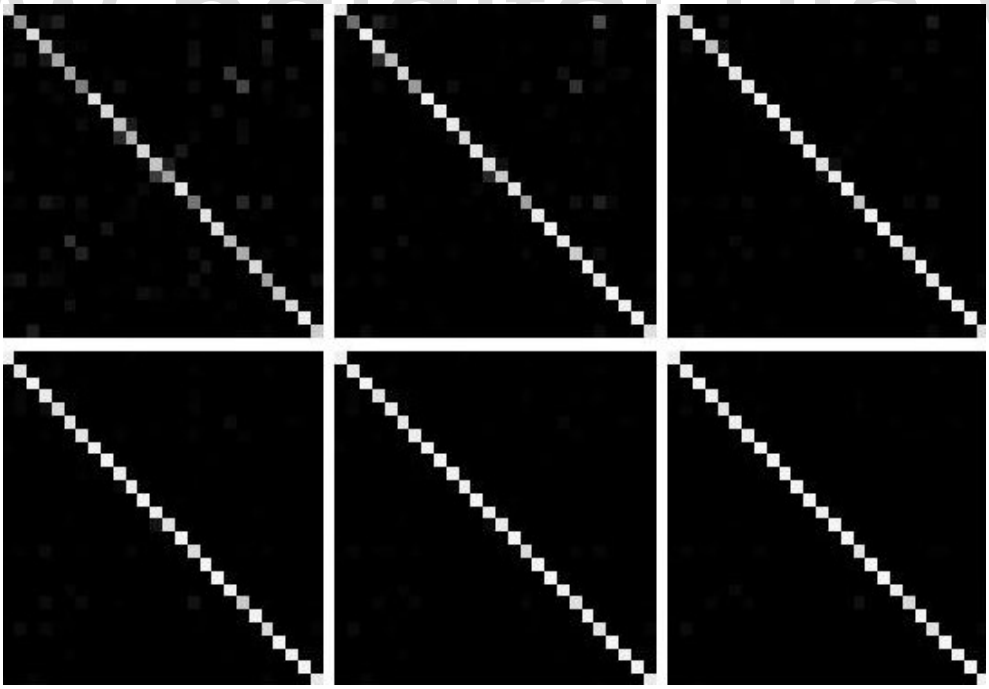


Figura 4.3. Error en función del número de iteraciones para las diferentes relaciones de compresión. (a) *Letters*. (b). *ISOLET*. (c) MNIST. (d) *YaleFaces*.



(a)



(b)

Figura 4.4. Matrices de confusión en dos bases de datos. (a) *Letters* con relación de compresión del 2 %. (b)*ISOLET* con relación de compresión del 1 %

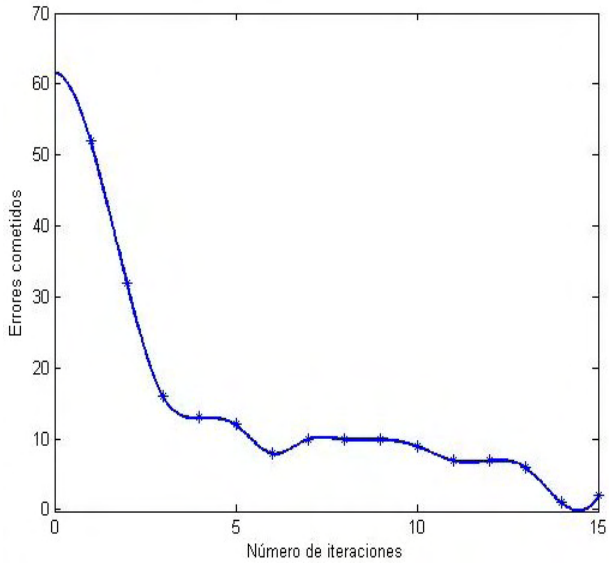


Figura 4.5. Número de veces que se confunde la letra "F" con la "S" en cada iteración en la base de datos *ISOLET* con una relación de compresión del 1 %

debe a que ambos sonidos en inglés son similares. En la figura 4.5 se muestra el número de veces que se confunde la letra F con la S para una relación de compresión del 1 % y como el número va disminuyendo a medida que aumenta el número de iteraciones. Sin embargo, se observa que hay puntos en que aumenta en vez de disminuir, esto se debe a que en el comportamiento normal de un método numérico a veces entre iteraciones aumenta el error.

4.2.1 Fijación del parámetro γ

En el Capítulo 2 se describió el parámetro γ como un hiperparámetro debido a que una mala selección de este parámetro puede aumentar la convexidad o forma de la función costo significativamente, por esta razón es necesario optimizar el valor de γ. En la figura 4.6 se muestra la importancia de la optimización de γ, la curva azul representa los errores al realizar una iteración para la optimización del conjunto de entrenamiento con un $\gamma = 0.1$ y la curva verde el error obtenido al realizar una iteración para la optimización del conjunto de entrenamiento con un γ optimizado previamente. Como se observa en estas gráficas, el valor de este parámetro es crucial para el correcto funcionamiento del algoritmo debido a que valores inadecuados pueden empeorar el desempeño del clasificador, convirtiendo el conjunto optimizado en una peor versión del conjunto de entrenamiento inicial.

En este trabajo se decidió optimizar primero el parámetro γ antes que el conjunto de entrenamiento, esta decisión se tomó después de realizar la optimización en paralelo y com-

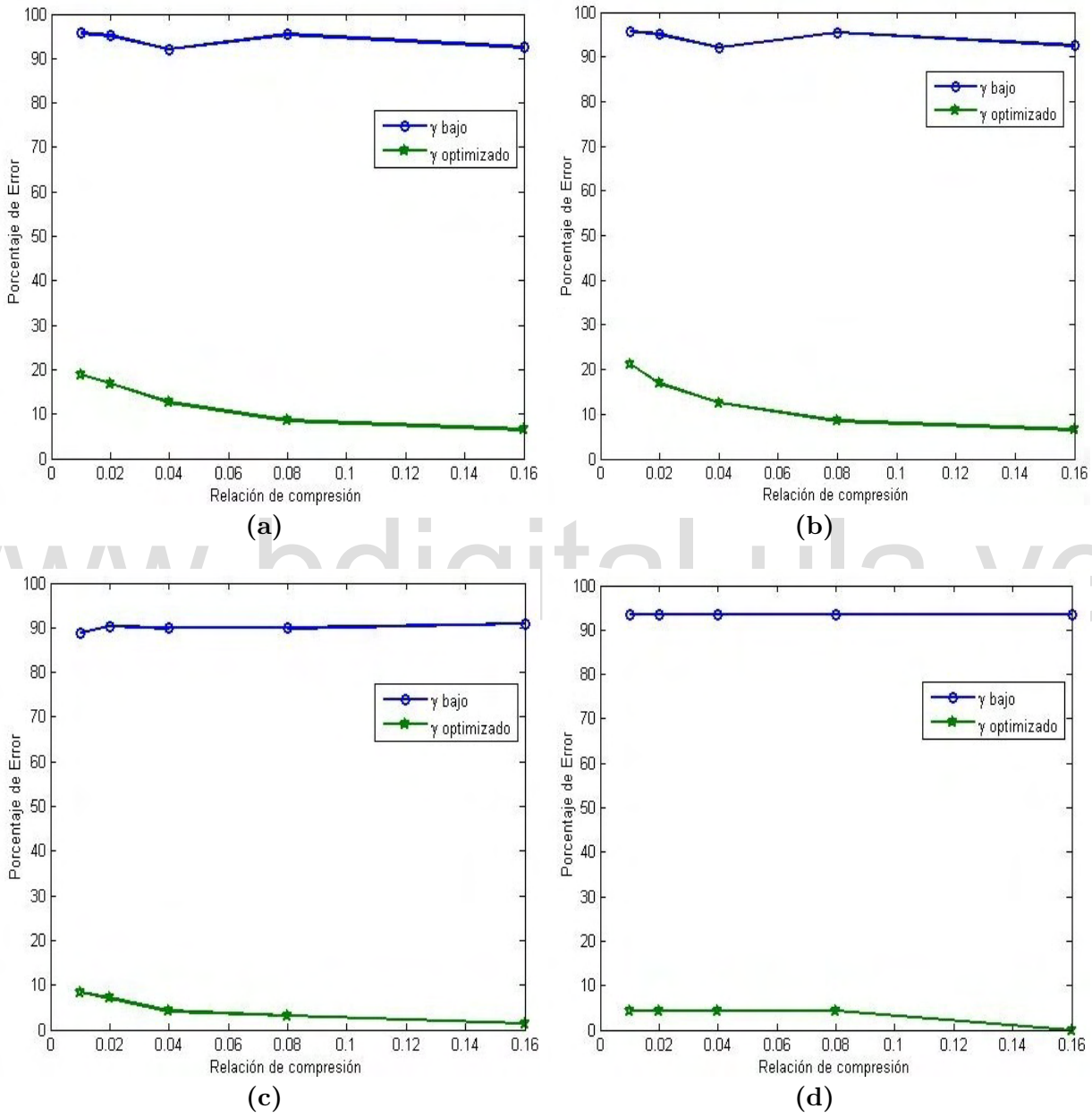
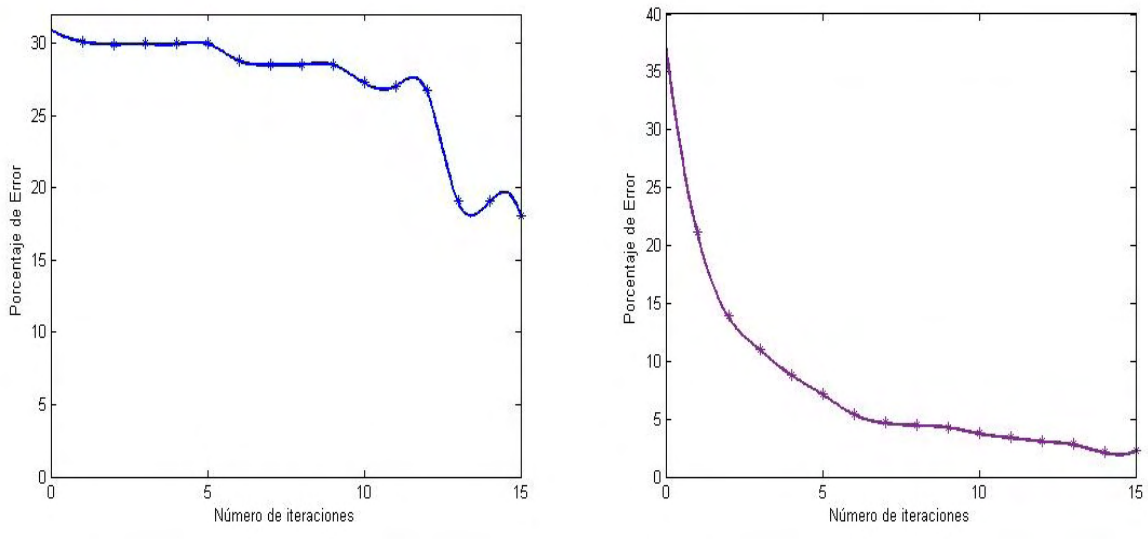


Figura 4.6. Comparacíon del error obtenido al realizar una iteración usando $\gamma = 0.1$ y el γ optimizado. (a) *Letters*. (b). *ISOLET*. (c) MNIST. (d) *YaleFaces*.



(a) **(b)**

Figura 4.7. Error en fución del número de iteraciones en *ISOLET* con relación de compresión de 1 %.(a) Optimización simultánea de γ y Z. (b) Optimización separada de γ y Z.

probar que la velocidad de convergencia era mucho más lenta. En la figura 4.7(a) se muestra este comportamiento en la base de datos *ISOLET* con una relación de compresión del 1 %, comparando con la figura 4.7(b) se observa como la convergencia es mucho más rápida al optimizar por separado a γ y el conjunto de entrenamiento **Z**.

En la figura 4.8 se presentan las curvas del error obtenido al aplicar 1-NN, utilizando como conjunto de entrenamiento el obtenido al realizar una iteración en método del gradiente descendente con *minimize* para diferentes valores de γ con una relación de compresión del 2 % en cada una de las bases de datos. En la figuras se muestra como el comportamiento de la curva de error no es convexo, como analogía se demuestra que la función costo tampoco lo es. Debido a que el método matemático seleccionado (SD) requiere funciones de costo convexas es necesario realizar una búsqueda exhaustiva de γ primero, para generar una curva convexa local que contenga el mínimo con menor valor, y luego sobre esta curva local se aplica el método matemático que resuelve el problema de minimización. Además se observa que valores de γ menores a 0.4 aumentan el error en el proceso de inferencia.

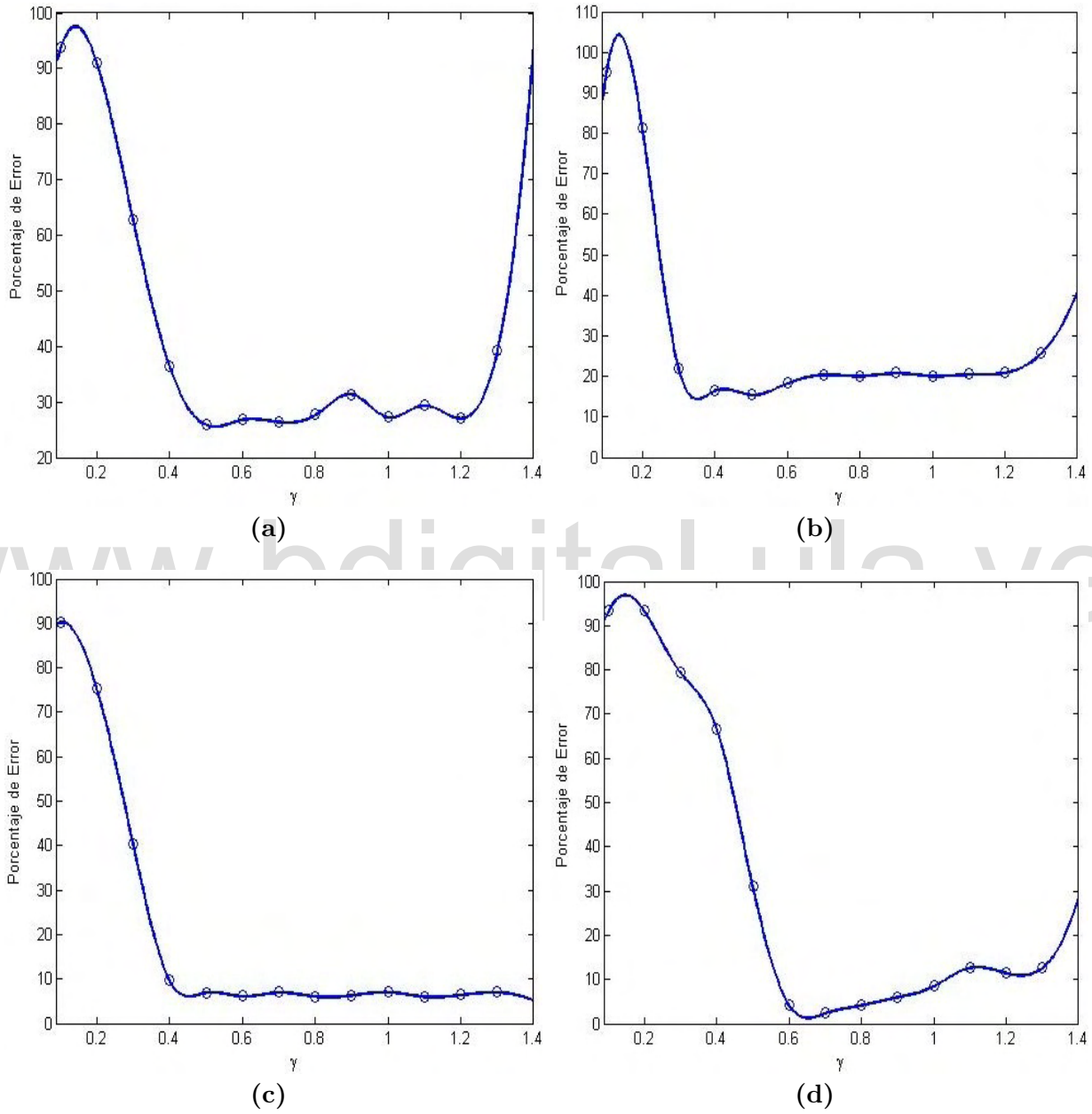


Figura 4.8. Error en función del valor de γ para la relación del 2%. (a) *Letters*. (b). *ISOLET*. (c) MNIST. (d) *YaleFaces*.

4.3 ACELERACIÓN DEL ALGORITMO DE CLASIFICACIÓN K-NN

El objetivo principal de este trabajo consiste en acelerar el tiempo de clasificación del algoritmo K-NN, con el fin de visualizar este objetivo se define como medida de desempeño el coeficiente de aceleración ecuación (4.2) definida como:

$$ca = \frac{\overline{t_1}}{\overline{t_2}} , \tag{4.2}$$

donde, ca es el coeficiente de aceleración, $\overline{t_1}$ es el tiempo promedio que le toma al algoritmo clasificar una muestra sin aplicar alguna compresión sobre el conjunto de entrenamiento y $\overline{t_2}$ es el tiempo promedio que le toma al algoritmo clasificar una muestra aplicando una compresión en el conjunto de entrenamiento.

La variable $\overline{t_1}$ está definida por la ecuación (4.3) definida como:

$$\overline{t_1} = \frac{\sum_{i=1}^{k} t_i}{k} , \tag{4.3}$$

donde t_i representa el tiempo necesario para clasificar la muestra i y k es el número de muestras que se seleccionó para calcular el promedio. En las bases de datos *Letter*, *ISOLET* y MNSIT se seleccionó $k = 100$ mientras que para *YaleFaces* $k = 25$.

La variable $\overline{t_2}$ se define en la ecuación (4.4) definida como:

$$\overline{t_2} = \frac{\sum_{i=1}^{n} t_j}{n} , \tag{4.4}$$

donde t_j es el tiempo que toma clasificar la muestra j y n es el número del conjunto de entrenamiento inicial.

El coeficiente de aceleración esperado es inversamente proporcional a la relación de compresión ecuación (4.5). Para la base de datos *Letters* figura 4.9(a) se observa que el comportamiento es casi ideal, con valores escasamente por debajo de los esperados. No obstante, las bases de datos *ISOLET*, MNIST y *YaleFaces* tienen coeficientes de aceleración mucho más altos debido a la reducción de dimensión que se aplicó sobre estas bases. En *ISOLET*

figura 4.9(b) los coeficientes están relacionados con la reducción de dimensión, un 27.39 % de las dimensiones originales, en MNIST figura 4.9(c) los coeficientes duplican los obtenidos en *ISOLET* a pesar de tener una reducción de dimensión similar en porcentajes (20.45 %), esto se debe al re-uso de la memoria caché en el equipo, debido a que el tamaño original de la base de datos es mucho mayor al de la base de datos *ISOLET*. Por último en *YaleFaces* figura 4.9(d) como ya se ha explicado previamente, no existe variación entre los conjuntos de entrenamiento comprimidos al 1 %, 2 %, 4 % y 8 % es por esto que, para estas compresiones, se observa un coeficiente de aceleración constante, sin embargo, el coeficiente de aceleración obtenido para la relación de compresión de 16 % es mucho mayor que en las bases de datos anteriores debido a que la reducción de dimensiones fue mucho mayor permitiendo un mejor uso de la memoria caché en este valor de compresión.

$$Coeficiente\ de\ aceleración\ esperado = \frac{1}{Relación\ de\ compresión} \tag{4.5}$$

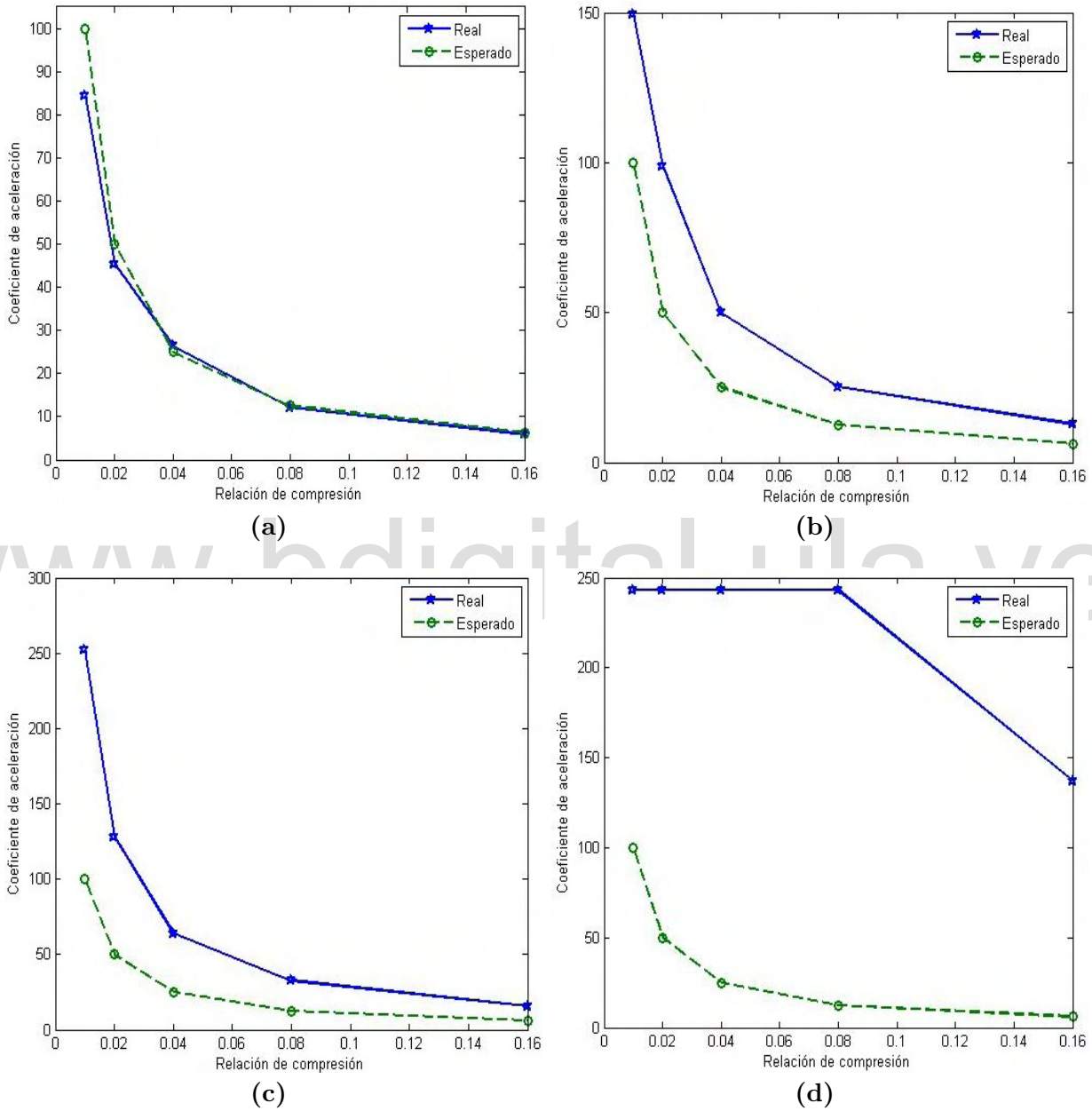


Figura 4.9. Coeficientes de aceleración. (a) *Letters*. (b). *ISOLET*. (c) MNIST. (d) *YaleFaces*.

CONCLUSIONES

- En la actualidad, los algoritmos de clasificación son una herramienta computacional muy utilizada debido a la diversidad de sus aplicaciones. Existen diferentes algoritmos capaces de realizar esta tarea, uno de los más utilizados debido a lo intuitivo de sus predicciones y la simplicidad de su implementación en el proceso de inferencia es el algoritmo K-NN. Como se explicó en este trabajo el algoritmo basa sus clasificaciones en simples cálculos de distancia. Como desventaja K-NN es un algoritmo lento y que requiere un gran uso de memoria.

- Existen tres técnicas con el fin de acelerar el algoritmo de clasificación K-NN. La primera reduce el cálculo de distancias a través de estructuras de arboles, este método si bien acelera el algoritmo tiene un mal rendimiento en términos de memoria. La segunda técnica consiste en reducir la dimensión de los datos, si bien esta metodología pareciese atractiva, este enfoque no es de gran utilidad cuando se manejan grandes volúmenes de baja dimensión. Por último, el método utilizado en este trabajo, comprime el conjunto de entrenamiento ofreciendo una aceleración al algoritmo de clasificación, reduciendo el uso de memoria y además es efectivo en bases de datos de baja dimensión.

- La compresión estocástica del conjunto de entrenamiento (SNC) es un método simple y efectivo basado en el método del gradiente descendente (SD) y en un enfoque estocástico del algoritmo K-NN, que acelera el algoritmo de clasificación mientras mantiene su precisión.

- SNC comprime el conjunto de entrenamiento a través del submuestreo uniforme de vectores de un conjunto inicial mucho más grande, los cuales son optimizados posteriormente.

- En SNC, basados en el hecho de que los tiempos de entrenamiento son proporcionales a la dimensión de los datos y al tamaño original del conjunto de entrenamiento, es

necesario analizar las características de los datos y de ser posible aplicar algún tipo de reducción de dimensión previo.

- El parámetro γ es de gran importancia en el proceso de optimización, este parámetro indica el grado de dispersión de los datos, valores bajos se relacionan con datos altamente concentrados alrededor de su media. Por esta razón, utilizar valores menores a 0.4 en la optimización empeora el desempeño del conjunto de entrenamiento en la etapa de clasificación.

- Para encontrar el valor de γ que permita una convergencia más rápida del algoritmo se debe realizar una optimización de este valor. Para realizar la optimización de γ se recurre a una matriz diagonal de la forma $\mathbf{A} = \gamma^2\mathbf{I}$, y se optimiza el parámetro dentro de esta matriz. Antes de optimizar, para logar resultados confiables y debido que SD funciona solo en funciones convexas, se debe realizar primero una búsqueda exhaustiva de γ para fijar su valor inicial.

- La convergencia del algoritmo está directamente relacionada con la relación de compresión que se utilice, como es de esperarse, y con las características de los datos iniciales. En general, a menor relación de compresión se necesita un menor número de iteraciones para llegar a la convergencia deseada. Sin embargo, un disminución en la relación de compresión significa un incremento inversamente proporcional en el tiempo requerido en el entrenamiento de los datos.

- El error que se obtuvo para las diferentes relaciones de compresión, después de optimizar, demuestra la eficiencia del algoritmo, arrojando una mejora de hasta el 34 %. El error obtenido en todas las bases de datos (exceptuando *Letters*) para una relación de compresión del 1 % está por debajo del 5 % siendo lo suficiente bajo para calificar el conjunto de entrenamiento como efectivo. En la base de datos *Letters* necesita una relación de compresión de 4 % para obtener un error menor al 5 %.

- Al reducir el conjunto de entrenamiento se obtuvo una aceleración en el tiempo de clasificación, la cual fue inversamente proporcional a la relación de compresión. La aceleración puede incrementarse si se realiza adicionalmente una reducción de las dimensiones de los datos, tal y como se demostró en el presente trabajo, donde se obtuvo una reducción de los tiempo de clasificación de hasta 250 veces del tiempo original.

- Basándose en las cuatro conclusiones anteriores, se establece que para la mayoría de las bases de datos utilizadas en este trabajo, la relación de compresión más efectiva es del 4 %. Por otro lado, la base de datos *YaleFaces* tiene un mejor desempeño utilizando una compresión del 16 % debido a que el error se hace cero en una sola iteración.

RECOMENDACIONES

La compresión del conjunto de entrenamiento desarrollada en este trabajo aborda el problema de aceleración con un enfoque estadístico y utiliza un modelo matemático ampliamente conocido. Los resultados obtenidos en este trabajo corroboran el buen funcionamiento de este algoritmo en diferentes bases de datos, esto permitió demostrar que el algoritmo se acelera de forma sustancial y manteniendo un rendimiento aceptable en los errores de clasificación . No obstante, estos resultados se presentaron en condiciones controladas. Como consecuencia, es necesario comprobar el funcionamiento del algoritmo en otras condiciones de trabajo, por esta razón se hace las siguientes recomendaciones:

- Estudiar el posible sobreentrenamiento de los datos. En este trabajo se optimizó el conjunto de entrenamiento de manera que la mayor cantidad de elementos, dentro de la base de datos, fueran clasificados correctamente. Es posible, entonces, que los conjuntos de entrenamiento esté muy bien definidos para clasificar elementos dentro de la misma base de datos. Sin embargo, es posible que al tratarse de nuevas entradas aumente el error. Es por eso que se recomienda estudiar el comportamiento de los conjuntos a la hora de clasificar nuevas entradas.

- Si bien el algoritmo presentó una aceleración significativa, utilizar combinaciones de SNC junto con estructuras de árboles, podría aumentar aún más esta aceleración.

- En este trabajo se utilizó como regla de clasificación 1-NN, es importante extender a SNC a vecindarios con $K > 1$, a través de la expansión de las funciones de probabilidad presentadas en este trabajo a vecindarios de mayor tamaño.

- A pesar de utilizar la matriz $\mathbf{A}$ para la optimización de γ, no se utiliza el concepto de la distancia de Mahalanobis. Una dirección interesante para incluir en trabajos posteriores sería comprobar si las distancias no Euclidianas podrián ser incluidas como parte de SNC. Por ejemplo tomar la norma ℓ_1 (suma de valores absolutos) como métrica de distancia.

- La mayor desventaja que se encontró en este método es que el tiempo de entrenamiento se veía altamente afectado a medida que se aumentaba el tamaño del conjunto de entrenamiento inicial. Una mejorar posterior, con el fin de acelerar los tiempo de entrenamiento, consiste en estudiar y realizar una paralelización del algoritmo.

REFERENCIAS

[1] R. Castro y M. Ruilova, "Árboles de clasificación (inteligencia artificial avanzada)," https://es.slideshare.net/techi322/algoritmos-de-clasificacin, 2008.

[2] M. Kusner, S. Tyree, L. Weinberger, y K. Agrawal, "Stochastic neighbor compression," *International Conference on Machine Learning*, vol. 32, pp. 622–630, junio 2014.

[3] H. Rajaguru y S. Prabhakar, *"KNN classifier and K-Means clustering for robust classification of epilepsy from EEG signals. A detailed analysis".* Alemania: Anchor Academic Publishin, 2017.

[4] A. Beygelzimer, K. Sham, y J. Langford, "Cover trees for nearest neighbor," *ICML 2006 - Proceedings of the 23rd International Conference on Machine Learning*, vol. 2006, pp. 97–104, 01 2000.

[5] K. Weinberger y L. Saul, "Distance metric learning for large margin nearest neighbor classification," *Journal of Machine Learning Research*, vol. 10, pp. 207–244, 2009.

[6] F. Angiulli, "Fast condensed nearest neighbor rule," *International Conference on Machine Learning*, pp. 25–32, 2005.

[7] C. Liu y M. Nakagawa, "Prototype learning algorithms for nearest neighbor classifier with application to handwritten character recognition," *International Conference on Document Analysis and Recognition*, pp. 378–381, 1999.

[8] G. Galves, "Desarrollo de herramientas computacionales de clasificación de superficies terrestres en imágenes satelitales usando representación poco densa de señales en diccionarios redundantes," Trabajo de Grado, Universidad de Los Andes, Diciembre 2016.

[9] S. Haykin, *Neural networks and learning machines*, 3ra edición. Estados Unidos de America: Pearson, 2008.

[10] F. Curtis y K. Scheinberg, "Optimization methods for supervised machine learning: from linear models to deep learning," https://pdfs.semanticscholar.org/b58e/a22b6673566f0574db46be7f0a9ae03cc81a.pdf, 2017.

[11] H. Robbins y S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22(3), pp. 400–407, Septiembre 1951.

[12] L. Smith, "A tutorial on principal components analysis," 2002.

[13] P. Hart, "The condenses nearest neidhbor rule," *IEEE Transactions on Information theory*, vol. 14, pp. 515–516, 1968.

[14] C. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Transactions on Computers*, vol. 100(11), pp. 1179–1184, 1974.

[15] T. Kohonen, "Improves versions of learning vector quantization," *International Joint Conference on Neural Network*, pp. 545–550, 1990.

[16] C. Decaestecker, "Finding prototypes for nearest neighbour classification by means of gradient descent and deterministic annealing," *Pattern Recognition*, vol. 30(2), pp. 281–288, 1997.

[17] S. Bermejo y J. Cabestany, "Adaptative soft k-nearest neighbor classifiers," *Pattern Recognition*, vol. 32, pp. 2077–2979, 1999.

[18] D. Wackerly, I. W. Mendenhall, y R. Scheaffer, *Estadística matemática con aplicaciones*, 7ma edición. Cengage learning, 2009, cap. Variables continuas y sus distribuciones de probabilidad.

[19] C. Manning y H. Schütze, *Fundations of statistical nature language processing*, 6ta edición. MIT, 2006.

[20] G. Hinton y S. Roweis, "Stochastic neighbor embedding," *Conference on Neural Information Processing Systems*, pp. 833–840, 2002, MIT Press.

[21] J. Goldberger, G. Hinton, S. Roweis, y R. Salakhutdinov, "Neighbourhood components analysis," *Conference on Neural Information Processing Systems*, pp. 513–520, 2004.

[22] R. Rasmussen, "Minimize," http://http://tinyurl.com/minimize-m, 2002.

[23] P. Frey y D. Slate, "Letters," https://archive.ics.uci.edu/ml/datasets/Letter+Recognition, 1991.

[24] P. Frey y D. Slate, "Letter recognition using holand-style adaptive classifiers," *Mahine Learning*, vol. 6, pp. 161–182, 1991.

[25] T. Cole y M. Fanty, "Spoken letter recognition," *Neural information processing systems conference*, 1990.

[26] T. Cole y M. Fanty, https://archive.ics.uci.edu/ml/datasets/ISOLET, 1990.

[27] Y. LeCun, C. Cortes, y C. Burges, "Mnist," http://yan.lecun.com/exdb/mnist, 1998.

[28] Y. LeCun, C. Cortes, y C. Burges, "Gradient-based learnig applied to document recognition," *Proceedings of IEEE*, vol. 86, pp. 2278–2324, 1998.

[29] A. Georghiades, P. Belhumeur, y D. Kriegman, "Yalefaces," http://cvc.cs.yale.edu/cvc/projects/yalefaces/yalefaces.html, 2001.

[30] A. Georghiades, P. Belhumeur, "From few to many: Illumination cone models for face recognition under variable lighting and pose," *IEEE Transactios on Pattern Analysis and Machine Learning Inteligence*, vol. 23, pp. 643–660, 2001.

APÉNDICE A
DERIVADA DE LA FUNCIÓN COSTO CON RESPECTO A LA MATRIZ Z

En este apéndice se muestra el procedimiento matemático aplicado para obtener la derivada de la función costo (ecuación (A.1)) con respecto a la matriz $\mathbf{Z}$

$$\mathcal{L}(\mathbf{Z}) = -\sum_{i=1}^{n} ln(p_i) \tag{A.1}$$

Derivando implícitamente la ecuación (A.1) con respecto a $\mathbf{Z}$ se obtiene:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} = -\sum_{i=1}^{n} \frac{1}{p_i} \frac{\partial p_i}{\partial \mathbf{Z}} \tag{A.2}$$

Donde p_i esta dado por la ecuación (A.3):

$$p_i = \sum_{j:y_i=y_j} \frac{e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_j\|^2}}{\sum_{k=1}^{2} e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_k\|^2}} \tag{A.3}$$

Se deriva la ecuación (A.3) con respecto a un elemento $\mathbf{z}_j^d$ de la matriz $\mathbf{Z}$, donde d denota la dirección sobre la cual se esta tomando la derivada.

$$\begin{aligned} \frac{\partial p_i}{\partial \mathbf{Z}}[\mathbf{z}_j^d] = & \frac{-\left(2\gamma^2(\mathbf{x}_i^d - \mathbf{z}_j^d) e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_j\|^2} \sum_{k=1}^{m} e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_k\|^2}\right) \delta_{y_i, y_j}}{\left(e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_k\|^2}\right)^2} - \dots \\ & \dots - \frac{-\left(2\gamma^2(\mathbf{x}_i^d - \mathbf{z}_j^d) e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_j\|^2} \sum_{j:y_i=y_j} e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_j\|^2}\right)}{\left(\sum_{k=1}^{m} e^{-\gamma^2 \|\mathbf{x}_i - \mathbf{z}_k\|^2}\right)^2} \end{aligned} \tag{A.4}$$

En la ecuación (A.4) δ_{y_i, y_j} es una función Delta Dirac que toma el valor de 1 únicamente si $y_i = y_j$, x_i^d representa el valor del vector $\mathbf{x}_i$ perteneciente a la matriz $\mathbf{X}$ en la dimensión d y z_j^d

representa el valor del vector $\mathbf{z}_j$ perteneciente a la matriz $\mathbf{Z}$ en la dimensión d. Simplificando y agrupando los elemento de de la ecuación (A.4) se obtiene:

$$\frac{\partial p_i}{\partial \mathbf{Z}}[\mathbf{z}_j^d] = *2\gamma^2 x_i^d p_{ij}(\delta_{y_i,y_j} - p_i) + 2\gamma^2 z_j^d p_{ij}(\delta_{y_i,y_j} - p_i) \tag{A.5}$$

Sustituyendo la ecuación (A.5) en la ecuación (A.2) se obtiene:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}}[\mathbf{z}_j^d] = \sum_{i=1}^{n} \left(\frac{2\gamma^2 x_i^d p_{ij}(\delta_{y_i,y_j} - p_i)}{p_i} - \frac{2\gamma^2 z_j^d p_{ij}(\delta_{y_i,y_j} - p_i)}{p_i} \right) \tag{A.6}$$

Para simplificar la expresión del gradiente es necesario definir las funciones $\mathbf{Q}_{ij}$ ecuación (A.7) y $\mathbf{P}_{ij}$ ecuación (A.8):

$$\mathbf{Q}_{ij} = (\delta_{y_i,y_j} - p_i) \tag{A.7}$$

$$\mathbf{P}_{ij} = \frac{p_{ij}}{p_i} \tag{A.8}$$

Utilizando las ecuación (A.7) y A.8 en la ecuación (A.6):

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}}[d, \mathbf{z}] = \sum_{i=1}^{n} 2\gamma^2 x_i^d \mathbf{Q}_{ij}\mathbf{P}_{ij} - 2\gamma^2 z_j^d \mathbf{Q}_{ij}\mathbf{P}_{ij} \tag{A.9}$$

Extrapolando los resultados de la ecuación (A.9) para la matriz $\mathbf{Z}$ se obtiene la ecuación (A.10):

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} = -2\gamma^2 \left(\mathbf{X}(\mathbf{Q} \circ \mathbf{P}) - \mathbf{Z}\Delta((\mathbf{Q} \circ \mathbf{P})^\intercal 1_n) \right) , \tag{A.10}$$

donde, $\circ$ denota el producto Hadamard, $\mathbf{1}_n$ es un vector n-dimensional donde cada elemento del vector toma el valor de 1, y $\Delta(\mathbf{y})$ indica posicionar el vector $\mathbf{y}$ en la diagonal principal de una matriz de ceros (0) de dimensión $m \times m$ siendo m la dimensión del vector $\mathbf{y}$.

APÉNDICE B
DERIVADA DE LA FUNCIÓN COSTO CON RESPECTO A LA MATRIZ A

En este apéndice se describe el procedimiento matemático aplicado para obtener la derivada de la función costo (ecuación (B.1)) con respecto a la matriz $\mathbf{A}$

$$\mathcal{L}(\mathbf{Z}) = -\sum_{i=1}^{n} ln(p_i) \tag{B.1}$$

Derivando implícitamente la ecuación (B.1) con respecto a $\mathbf{Z}$ se obtiene:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = -\sum_{i=1}^{n} \frac{1}{p_i} \frac{\partial p_i}{\partial \mathbf{A}} \tag{B.2}$$

Donde p_i esta dado por la ecuación (B.3):

$$p_i = \sum_{j:y_i=y_j} \frac{e^{-\|\mathbf{A}(\mathbf{x}_i-\mathbf{z}_j)\|^2}}{\sum_{k=1}^{2} e^{-\gamma^2\|\mathbf{x}_i-\mathbf{z}_k\|^2}} \ , \tag{B.3}$$

donde $\| \mathbf{A}(\mathbf{x}_i - \mathbf{z}_j) \|^2 = (\mathbf{x}_i - \mathbf{z}_j)^{\mathsf{T}}\mathbf{A}^{\mathbf{T}}\mathbf{A}(\mathbf{x}_i - \mathbf{z}_j)$ y debido a que $\mathbf{A} = \gamma^2\mathbf{I}$ se puede asumir sin alterar el resultado que $\mathbf{A}^{\mathbf{T}}\mathbf{A} = \mathbf{A}^2$. Sustituyendo en la ecuación se obtiene entonces $\| \mathbf{A}(\mathbf{x}_i - \mathbf{z}_j) \|^2 = \mathbf{A}^2(\mathbf{x}_i - \mathbf{z}_j)^{\mathsf{T}}(\mathbf{x}_i - \mathbf{z}_j)$, se sustituye este resultado en la ecuación (B.3) y derivando con respecto a $\mathbf{A}$ se obtiene:

$$\begin{aligned}\frac{\partial p_i}{\partial \mathbf{A}} = &\frac{-\left(2\mathbf{A}(\mathbf{x}_i - \mathbf{z}_j)^{\mathsf{T}}(\mathbf{x}_i - \mathbf{z}_j)\sum_{j=1}^{m} e^{-\|\mathbf{A}(\mathbf{x}_i-\mathbf{z}_j)\|^2}\sum_{k=1}^{m} e^{\|\mathbf{A}(\mathbf{x}_i-\mathbf{z}_k)\|^2}\right)\delta_{y_i,y_j}}{\left(\sum_{k=1}^{m} e^{-\gamma^2\|\mathbf{x}_i-\mathbf{z}_k\|^2}\right)^2} - \cdots \\ &\cdots - \frac{-\left(2\mathbf{A}(\mathbf{x}_i - \mathbf{z}_j)^{\mathsf{T}}(\mathbf{x}_i - \mathbf{z}_j)^2\sum_{j:y_i=y_j} e^{-\|\mathbf{A}(\mathbf{x}_i-\mathbf{z}_j)\|^2}\sum_{k=1}^{m} e^{\|\mathbf{A}(\mathbf{x}_i-\mathbf{z}_k)\|^2}\right)}{\left(\sum_{k=1}^{m} e^{\|\mathbf{A}(\mathbf{x}_i-\mathbf{z}_k)\|^2}\right)^2}\end{aligned} \tag{B.4}$$

Agrupando y reescribiendo la ecuación (B.4), donde δ_{y_i,y_j} es una función Delta Dirac que toma el valor de 1 únicamente si $y_i = y_j$, x_{di}, se obtiene:

$$\frac{\partial p_i}{\partial \mathbf{A}} = -2\mathbf{A}\sum_{j=1}^{m} p_{ij}(1-\delta_{y_i,y_j})(\mathbf{x}_i-\mathbf{z}_j)^{\mathsf{T}}(\mathbf{x}_i-\mathbf{z}_j) \tag{B.5}$$

Sustituyendo la ecuación (B.5) en la ecuación (B.2) se obtiene:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = -\sum_{i=1}^{n}\left(\frac{-2\mathbf{A}\sum_{j=1}^{m} p_{ij}(1-\delta_{y_i,y_j})(\mathbf{x}_i-\mathbf{z}_j)^{\mathsf{T}}(\mathbf{x}_i-\mathbf{z}_j)}{p_i}\right) \tag{B.6}$$

Utilizando las ecuación (A.8) y ecuación (A.7) del Apéndice A para simplificar la ecuación (B.6) se obtiene:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{A}} = 2\mathbf{A}\sum_{i=1}^{n}\sum_{j=1}^{m}\mathbf{P}_{ij}\mathbf{Q}_{ij}(\mathbf{x}_i-\mathbf{z}_j)^{\mathsf{T}}(\mathbf{x}_i-\mathbf{z}_j) \tag{B.7}$$

GLOSARIO DE TÉRMINOS

γ**:** Representa la dispersión de los datos del conjunto de entrenamiento en SNC.

X: Matrices se denotan con mayúscula y en negrita.

x: Vectores se denotan con minúscula y en negrita, y son representados como vectores columnas.

T**:** Indica el operador transpuesto aplicable a matrices o vectores.

d**:** dimensión del conjunto de entrenamiento.

m**:** tamaño del conjunto de entrenamiento comprimido.

n**:** tamaño del conjunto de entrenamiento inicial.

1-NN**:** Vecino más cercano (*1-Nearest Neighbor*).

DFT**:** Transformada Discreta de Fourier (*Discrete Fourier Transform*).

DL**:** Método de optimización de Aprendizaje Profundo (*Deep Learning*).

K-NN**:** K vecinos más cercanos (*K Nearest Neighbors*).

KL**:** Kullback-Leibler.

LMNN**:** Algoritmo estadístico de máquinas de aprendizaje para el aprendizaje de distancias (*Large Margin Nearest Neighbor*).

NCA**:** *Neighbourhood Components Analysis*.

NN**:** Redes Neuronales (*Neural Network*).

PCA**:** Análisis de Componentes Principales (*Principal Component Analysis*).

SD: Método de optimización del Gradiente Descendente (*Steepest Descent*).

SGD: Método de optimización Estocástica del Gradiente Descendente (*Stochastic Gradient Descent method*).

SNC: Compresión Estocástica de Conjunto del Entrenamiento (*Stochastic Neighbor Compression*).

SNE: *Stochastic Neighbor Embedding*.

SVM: Máquinas de Soporte Vectorial (*Support Vector Machine*).