Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Instituto de Ingeniería Matemática y Computacional
Introducción a la Ingeniería Matemática
IMT1001 2020-1 Profesor Elwin van't Wout

Clasificación y visualización de números y letras escritos a mano utilizando Machine Learning

Integrantes:

Vicente Agüero
vicenteaguero@uc.cl

Sebastián Ferrá
sferra@uc.cl

Antonia Reyes
antonia.reyes@uc.cl

Javier Takahashi
jtakahashi@uc.cl

Felipe Valdés
fvaldes17@uc.cl

Resumen

El problema de clasificación de números y letras en datasets de gran tamaño puede ser solucionado aplicando técnicas de aprendizaje supervisado. Podemos entrenar algoritmos que clasifiquen números y letras y que sean capaces de predecir nuevos datos con un alto accuracy. Además, se pueden utilizar métodos de reducción de dimensionalidad para visualizar en 2D datos que estén en altas dimensiones, por ejemplo con 784 dimensiones. Trabajar con grandes bases de datos implica la necesidad de entender la forma del dataset e identificar si tiene buenas propiedades. En caso de estar desbalanceado, se pueden aplicar técnicas de balanceo, como el subsampling y entrenar los algoritmos en nuevas bases de datos balanceadas. También, se pueden crear nuevos datasets con el objetivo de clasificar símbolos propios. Para entender como funcionan los algoritmos hay revisar como se comportan en otro tipo de datos y así poder comprender las diferentes utilidades según la tarea a la cual están diseñados.

Palabras claves: aprendizaje supervisado, dataset, reducción de dimensionalidad, clasificadores, entrenamiento, subsampling, balanceo, algoritmos.

Índice

1. Introducción

Cuando se tienen bases de datos lo suficientemente grandes, se pueden entrenar algoritmos de *Machine Learning* para poder predecir situaciones o datos futuros. Por ejemplo, a un banco le interesa tener almacenado una gran cantidad de datos de sus clientes, con estos se puede predecir quienes podrían obtener un crédito extra o se podría detectar un posible fraude. El aprendizaje automatizado supervisado es aquel donde ya se conocen las etiquetas de ciertas bases de datos a analizar para predecir datos futuros.

Sin embargo, no cualquier base de datos es útil cuando la meta es la predicción; para utilizar aprendizaje automático supervisado en la clasificación de datos es importante que la base de datos cumpla ciertos requisitos para una predicción correcta. Primero, se necesita que sea coherente, es decir, que las etiquetas estén relacionadas con la información contenida en el dato. Segundo, es importante que esté ordenada, que todos los datos presenten el mismo formato. Tercero, que exista una proporción similar en la cantidad de datos entre las clases. Por último, es fundamental que la información sea útil, que los datos tengan relación directa con lo que se quiere predecir. Por ejemplo, no es útil usar una base de datos sobre ventas de libros en China para predecir los choques de trenes con autos.

A partir de esto, la investigación del presente proyecto se basa en la visualización y clasificación de datos. Se utilizan imágenes digitalizadas en forma de matrices, para entrenar algoritmos de clasificación en aprendizaje supervisado. Para ello, se utilizaron tres bases de datos en la clasificación de letras y números: *Digits*, *MNIST* y *EMNIST*. La información de estas bases de datos está representada en matrices con una alta dimensionalidad, haciendo imposible para el humano poder visualizar la información en este formato. Es por esto que se utilizarán distintos métodos para lograr visualizar estos datos en bajas dimensiones y buscando minimizar la pérdida de información, para que le representación visual sea fiel a los datos originales.

Al momento de clasificar se pueden presentar diversos problemas. Diferentes tipos de información necesitarán diferentes tipos de clasificadores, donde es necesario considerar su complejidad, tiempo de ejecución y funcionalidad. Existen algoritmos basados en probabilidades, otros basados en similitud de los datos, pesos de la información, etc. Otro problema es que la base de datos no sea ideal, es decir, que presente irregularidades, como un desbalanceo en la proporción de los datos. Estos problemas se pueden identificar mediante parámetros como el *accuracy* o el análisis de la matriz de confusión. Algunos de estas soluciones están implementadas en el presente trabajo de investigación, en especial aquellas relacionadas con el contrarrestar el desbalanceo.

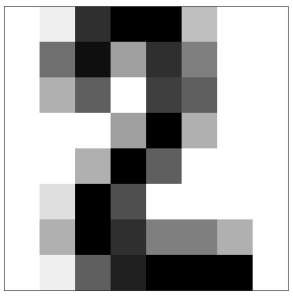
2. Metodología de la investigación

2.1. Bases de datos

La clasificación de números y letras se hizo a partir de tres bases de datos con diferentes tamaños y formatos. En un primer acercamiento se utilizó el dataset de Digits de Scikit-Learn, que es un dataset de un tamaño pequeño. Luego, se utilizaron los datasets MNIST y EMNIST desde la librería TensorFlow, estos superando las 60000 y 800000 instancias, respectivamente.

2.1.1. Digits

Digits es una base de datos proveniente de UCI [1] que es utilizada por la librería Scikit-Learn dentro de sus datasets predefinidos. Contiene 1797 imágenes escritas como matrices, las cuales tienen un tamaño de 8×8. Estas imágenes representan números desde el 0 hasta el 9 y la distribución de las cantidades de cada dígito es bastante similar, por ende es una base de datos balanceada.



$$\mapsto \begin{bmatrix} 0 & 1 & 13 & 16 & 16 & 4 & 0 & 0 \\ 0 & 9 & 15 & 6 & 13 & 8 & 0 & 0 \\ 0 & 5 & 10 & 0 & 12 & 10 & 0 & 0 \\ 0 & 0 & 0 & 6 & 16 & 5 & 0 & 0 \\ 0 & 0 & 5 & 16 & 10 & 0 & 0 & 0 \\ 0 & 2 & 16 & 11 & 0 & 0 & 0 & 0 \\ 0 & 5 & 16 & 13 & 8 & 8 & 5 & 0 \\ 0 & 1 & 10 & 14 & 16 & 16 & 16 & 0 \end{bmatrix}$$

Figura 1: Representación de una instancia del dataset en forma de matriz 8×8

La Figura 1 es un ejemplo de un elemento de la base de datos, las matrices se pueden mostrar como imágenes, donde cada elemento $a_{i,j}$ de la matriz representa que tan oscuro es el píxel que describe, desde 0 hasta 16. En la figura 2 se muestra como luce la base de datos en *Python*, en donde cada instancia es un vector de 64 dimensiones.

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	4.0	0.0	0.0	0.0	2.0	14.0	15.0	9.0	0.0	0.0
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	6.0	16.0	14.0	6.0	0.0	0.0
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	9.0	13.0	6.0	0.0	0.0
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	0.0	5.0	12.0	16.0	12.0	0.0	0.0
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	8.0	0.0	0.0	1.0	8.0	12.0	14.0	12.0	1.0	0.0

1797 rows × 64 columns

Figura 2: Base de Datos Digits visualizada con la librería Pandas en Python

2.1.2. MNIST

MNIST es una base de datos creada por el investigador francés Yann LeCunn [2], derivada de una base de datos más grande conocida como *NIST*. La base de datos *MNIST* en particular se puede obtener de la librería *TensorFlow*, distribuido por *Keras*. Contiene dígitos escritos a mano, específicamente 60000 ejemplos para entrenamiento y 10000 ejemplos de testeo. La representación de estos dígitos es a partir de matrices de tamaño 28×28. Estas matrices representan números desde el 0 hasta el 9 y la distribución de las cantidades de cada dígito es bastante similar, por ende es una base de datos balanceada.

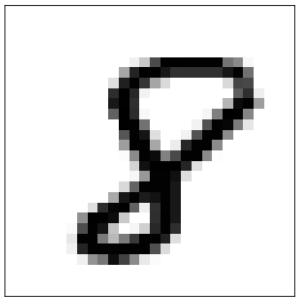


Figura 3: Ejemplo de una visualización como imagen de una instancia del dataset MNIST.

La Figura 3 es un ejemplo de un elemento de la base de datos, las matrices se pueden mostrar como imágenes, donde cada elemento $a_{i,j}$ de la matriz representa que tan oscuro es el píxel que describe, desde 0 hasta 255. En la figura 4 se muestra como luce la base de datos en *Python*, en donde cada instancia es un vector de 784 dimensiones.

	Label	0	1	2	3	4	5	6	7	8	...	774	775	776	777	778	779	780	781	782	783
0	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
59995	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59996	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59997	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59998	6	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
59999	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

60000 rows × 785 columns

Figura 4: Base de datos MNIST.

2.1.3. EMNIST

EMNIST es una base de datos de imágenes números y letras escritos a mano creada por investigadores del The MARCS Institute for Brain, Behaviour and Development en Australia [3]. Se puede acceder a esta base de datos desde la librería Tensorflow dentro de sus datasets del módulo `tensorflow_datasets`. Contiene más de 800.000 imágenes escritas como matrices, las cuales tienen un tamaño de 28×28. Estas imágenes pueden ser números desde el 0 hasta el 9 o una de las 26 letras del alfabeto inglés en sus versiones minúscula o mayúscula. La distribución de las cantidades de cada símbolo es bastante distinta, sobretodo al comparar entre números y letras, por ende es una base de datos desbalanceada.

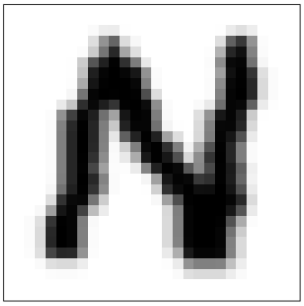


Figura 5: Ejemplo de una visualización como imagen de una instancia del dataset EMNIST

La Figura 5 es un ejemplo de un elemento de la base de datos, las matrices de estos elementos se pueden mostrar como imágenes, donde cada elemento $a_{i,j}$ de la matriz representa que tan oscuro es el píxel que describe, desde 0 hasta 255. En la figura 2 se muestra como luce la base de datos en *Python*, en donde cada instancia es un vector de 784 dimensiones con su respectivo label.

	image	label
0	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	5
1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	9
2	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	14
3	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	8
4	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	8
...	...	...
697927	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	1
697928	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	33
697929	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	6
697930	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	47
697931	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	12

697932 rows × 2 columns

	image	label
0	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	24
1	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	2
2	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	3
3	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	3
4	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	56
...	...	...
116318	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	3
116319	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	2
116320	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	4
116321	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	9
116322	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	7

116323 rows × 2 columns

Figura 6: Base de Datos EMNIST dividida entre datos para entrenamiento y de testeo

2.2. Reductores de dimensionalidad

Lo que se busca con cualquier reductor de dimensionalidad es poder graficar datos que estén como vectores de 4 o más coordenadas en un gráfico de 2 o 3 para poder visualizar y presentar los datos de una manera entendible, manteniendo las características iniciales lo más parecidas posible. A continuación mostraremos 5 reductores de dimensión con los que tratamos una base de datos y la parte teórica de cada uno.

2.2.1. Principal Component Analysis

Principal Component Analysis, o *PCA*, es un reductor lineal de dimensionalidad que utiliza principalmente el álgebra lineal para encontrar vectores de z coordenadas que expliquen o representen lo mismo o muy parecido a un vector de p coordenadas, con $z < p$. *PCA* condensa la información en vectores de menor dimensión pero que sigan siendo fieles representantes de los vectores originales, así entonces se puede mostrar un gráfico de 2 dimensiones de una base de datos con datos de 64 coordenadas [4].

Las componentes principales se calculan primero centralizando las variables, es decir a cada variable se le resta la media de las variables para que todas terminen con media 0. Luego, sea Z_1 la primera componente, se busca encontrar los ϕ tal que $Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + ... + \phi_{p1}X_p$, con $X_1...X_p$ un grupo de variables y estos ϕ son una combinación lineal normalizada de las variables que tiene mayor varianza. Esta combinación lineal es encontrada resolviendo un problema de optimización utilizando el cálculo de valores y vectores propios de la matriz de las covarianzas. Entonces por cada componente principal Z_i tenemos una combinación lineal de las variables originales. Luego de calculada la primera componente principal, las siguientes se calculan de la misma manera, agregando la condición de que deben ser ortogonales a las anteriores. Es importante mencionar que en la combinación lineal, los ϕ_{i1} representan la importancia de la variable en cada componente, o sea que nos dicen que tipo de información está recogiendo cada componente.
Es claro luego que cuando se calculen todas componentes, estará contenida toda la información y si utilizamos solo las 2 primeras componentes principales, perderemos información, esta cantidad de información perdida o la información contenida en una cantidad de componentes puede ser calculada utilizando

$$\frac{\sum_{i=1}^{n}(\sum_{j=1}^{p}\phi_{jm}x_{ij})^2}{\sum_{j=1}^{p}\sum_{i=1}^{n}x_{ij}^2}$$

Que representa la proporción varianza explicada por cada componente, considerando que las variables están normalizadas.

La cantidad de componentes principales a utilizar no puede ser sacada de una fórmula, sino que se debe considerar la finalidad con la que se utilizarán. Si se busca graficar una base de datos con datos de 64 coordenadas en el espacio, utilizaremos entonces 3 componentes principales, logrando así graficar asignándole a cada observación los valores de sus proyecciones sobre cada componente,

es decir el valor en la primera componente de una observación será su proyección sobre esta y así con el resto de las componentes; logrando hacer un gráfico con todas las observaciones dónde sus coordenadas en 2D, por ejemplo, serán (Proyección a la primera componente, Proyección a la segunda componente)[4].
La complejidad computacional del PCA es $\mathcal{O}(p^2n + p^3)$ con n puntos de datos y p características, donde se suman la complejidad del cálculo de la matriz de covarianzas y la complejidad del cálculo de vectores y valores propios. Con eso en mente, la cantidad de tiempo que se toma en aplicar este método a bases de datos como MNIST es mucho más que a una base de datos más simple como Digits. Aunque sean en tiempo polinomial el tamaño de MNIST y EMNIST es mucho más grande que Digits, lo que lo hizo impracticable al momento de comparar resultados.

2.2.2. t-Distributed Stochastic Neighbor

t-Distributed Stochastic Neighbor es un método de reducción de dimensionalidad no lineal que tiene por objetivo reducir al mínimo la divergencia de Kullback-Leible entre las distribuciones de probabilidad P, la cual representa las similitudes o afinidades entre pares en el espacio de alta dimensionalidad; y la distribución de probabilidad Q, representando las similitudes entre pares del espacio de baja dimensionalidad. Las similitudes en el espacio de alta dimensionalidad se obtiene en base a distancias euclidianas y siguen una distribución de probabilidad gaussiana. En cambio, la similitud en el espacio de baja dimensionalidad sigue una distribución de probabilidad t de Student y los puntos se determinan de manera tal que se cumpla la minimización de divergencia antes nombrada [5].

El método, a grandes rasgos, determina para cada par de objetos x_m y x_n en el espacio de alta dimensionalidad la distancia (comúnmente euclidiana) entre ambos, luego se define la probabilidad conjunta p_{mn}, que representa la afinidad entre los objetos x_m y x_n mediante la simetría de dos probabilidades condicionales como se muestra a continuación:

$$p_{m|n} = \frac{\exp(-d(x_m, x_n)^2/2\sigma_m^2)}{\sum_{k\neq m} \exp(-d(x_m, x_k)^2/2\sigma_m^2)} \qquad p_{m|m} = 0$$

$$p_{m,n} = \frac{p_{n|m} + p_{m|n}}{2N}$$

El valor de σ_m^2 se establece de manera tal que la perplejidad (definida como $2\mathrm{H}(p_m)$, donde $\mathrm{H}(p_m)$ es la entroía de Shannon de la distribución de probabilidad p_m) sea igual al valor establecido por el usuario (en el caso aplicado se utiliza el default igual a 30).

En el espacio de baja dimensionalidad, la afinidad q_{mn} entre dos puntos y_m y y_n se determina utilizando un kernel normalizado t de Student con un único grado de libertad, de la siguiente forma:

$$q_{mn} = \frac{(1 + ||y_m - y_n||^2)^{-1}}{\sum_{k\neq l}(1 + ||y_k - y_l||^2)^{-1}} \qquad q_{mm} = 0$$

La ubicación del punto y_m en el espacio de baja dimensionalidad se determina al minimizar la divergencia de Kullback-Leible entre las distribuciones conjuntas P y Q, como se muestra a continuación:

$$KL(P||Q) = \sum_{m \neq n} p_{mn} \log \frac{p_{mn}}{q_{mn}}$$

2.2.3. Truncated Singular Value Decomposition

Este método de reducción de dimensionalidad es lineal y similar al PCA. La diferencia es que en este método los datos no necesitan estar centrados, es decir, no se les resta la media.
Para utilizarlo se forma una matriz de m x n con las muestras y sus variables. Luego se aplica la descomposición en valores singulares, pero al ser esta una descomposición "truncada", como dice el nombre, se escogen k valores singulares, los cuales son (n_components) en la función de python. En otras palabras, no se ocupan las matrices enteras, sino solo una parte.
Si por ejemplo la matriz con los datos, llamada X, se descompone de la siguiente forma:

$$X = U\Sigma V^T$$

Escogemos una aproximación de X

$$X \approx X_k = U_k \Sigma_k V_k^T$$

Donde U_k es una matriz de m x k que tiene como columnas a los vectores propios de XX^T. Σ_k es una matriz de k x k diagonal con los valores singulares positivos ordenados por magnitud. V_k^T es una matriz de k x n que tiene como filas a los vectores propios de $X^T X$.
Entonces, lo que nos entrega este método son "mejores" ejes para proyectar la información, al igual que PCA. Un mejor eje significa uno que minimice la suma de las distancias entre el nuevo eje y la proyección de los datos en ese eje, con el método de mínimos cuadrados. La dirección de los nuevos ejes coincide con la de los vectores de la matriz V. Por otro lado, la varianza explicada por cada nuevo eje está dada por el valor singular correspondiente al vector. Además, los vectores de la matriz U explican la posición de cada punto en el nuevo eje que le corresponde [6].

2.2.4. Multidimensional Scaling

Multidimensional Scaling o *MDS* es un método de reducción de dimensionalidad que busca una representación de la base de datos en una dimensión baja, manteniendo las distancias que existían en los datos en la escala mayor. Con este método de reducción de la dimensionalidad de los datos podemos analizarlos y encontrar similitudes entre ellos. Los datos obtenidos por el método se pueden graficar, obteniendo una representación que es entendible para el ser humano, pues no es posible graficar correctamente datos en más de tres dimensiones. Este método se puede entender como una forma de llevar distancias desde una dimensión a otra, entendiéndose distancia como la diferencia entre dos variables o datos, como la distancia entre dos átomos o la diferencia entre un índice económico de un país versus el mismo índice económico de otro país.

$$\text{Stress}_D\left(x_1, x_2, \ldots, x_N\right) = \left(\sum_{i \neq j = 1,\ldots,N} \left(d_{ij} - \|x_i - x_j\|\right)^2\right)^{1/2}$$

En este caso b_{ij} son términos de la matriz B que describiremos a continuación. El algoritmo de *MDS* se basa en los siguientes cuatro pasos, primero se consigue la matriz de proximidad $D^{(2)} = [d_{ij}^2]$, en segundo lugar se debe aplicar centramiento doble de la siguiente manera, $B = -\frac{1}{2} J D^{(2)} J$ usando la matriz de centramiento $J = I - \frac{1}{n} 11'$, con en el número de objetos. Luego de esto se deben buscar la cantidad de vectores propios asociados a valores propios más grandes, según la dimensión que se quiera para el resultado, si busco graficar en 2 dimensiones entonces busco los 2 vectores propios asociados a los 2 valores propios más grandes. Por último, se encuentra $X = P_m \cdot \Lambda_m^{1/2}$, de las cuales P_m es la matriz de valores propios (de la matriz B encontrada antes) contenidos en la matriz diagonal Λ_m y m representa la cantidad que valores propios escogidos o la dimensión en la que se desea graficar[7]. Es importante mencionar que las distancias en este algoritmo deben ser euclidianas.

La complejidad de este método viene dada por $\mathcal{O} = (n^3)$ (n es la cantidad de datos)[8], debido a que debe calcular valores y vectores propios de una matriz igual de grande que la matriz de distancias D, eso si se utiliza el método de valores y vectores propios que corre en ese tiempo, puede demorar más si se utiliza un método menos eficiente. En la práctica *MDS* no es muy útil para reducir la dimensión de bases de datos muy grandes (más de 60,000 filas) ya que los tiempos que demoraría serían excesivamente extensos en una computadora doméstica, pero sigue siendo tiempo polinomial.

2.2.5. Isomap

Isometric mapping, o en su forma corta, *Isomap*, es un método de reducción de dimensionalidad no lineal, correspondiente a un *Manifold*, cuyo propósito es describir una estructura de $n-$dimensiones en un número menor de dimensiones k (con $n > k$) [9]. Para ello utiliza un sistema de "vecinos cercanos" basado en geometría geodésica, que preserva las distancias geodésicas en la $k-$dimensión. Es por esta preservación de distancias que *Isomap* es considerado una extensión del algoritmo *Multidimensional Scaling* (*MDS*) presentado en el apartado 2.2.4.

Para ejecutar el algoritmo de *ISOMAP*, en primer lugar, se crea un gráfico de vecindad para la base de datos. Sea G el gráfico obtenido y i, j puntos que representan un dato de la base de datos B, entonces utilizando un algoritmo de aproximación como $k-$ vecinos cercanos se obtienen las similitudes entre los diferentes puntos i,j, ubicando así cada punto en el gráfico G. De esta manera, G representa las relaciones existentes dadas por el vecindario. En segundo lugar, se estiman las distancias geodésicas entre todas las combinaciones de puntos M_i,M_j en el *Manifold* M a partir de las distancias más cortas dadas por el gráfico G entre los puntos G_i,G_j. Para ello se utiliza un algoritmo donde se conserva la distancia geodésica en caso que un par de puntos G_i,G_j estén bien definidos como puntos M_i,M_j. En caso contrario se toma a la distancia entre puntos G_i,G_j como infinito, de manera que iterando entre los diferentes puntos se puedan obtener distancias más cortas. El mínimo de esas distancias corresponde a la distancia entre los puntos M_i,M_j. En tercer lugar, se crea una matriz G_m que contiene a todas las distancias entre los puntos en G, luego se aplica el algoritmo *MDS* a G_m, construyendo una representación de los datos en un espacio Euclídeo $d-$dimensional K que preservan las distancias del *Manifold* M [10]. Para ello se utiliza la descomposición en valores propios de la matriz G_m.

Cabe destacar que la implementación del algoritmo de *Isomap* es de complejidad $\mathcal{O}(n^2)$, por lo que implementarlo por si solo no es recomendado. Por suerte la librería de *Scikit Learn* en su módulo *Manifold* posee implementado el algoritmo *Isomap* de manera mucho mas eficiente (una aproximación es *Landmark Isomap*). Al algoritmo de *Scikit Learn* se le entrega el número de vecinos cercanos y el número de componentes, de tal manera que pueda obtener la reducción de dimensionalidad a partir de estos datos. Sin embargo, también se le pueden entregar varios parámetros para hacer más personalizado el algoritmo y la reducción de dimensionalidad.

2.3. Clasificadores

2.3.1. Decision Tree

Decision Tree es un algoritmo de clasificación y regresión supervisado, al cual no se le entregan parámetros. El objetivo principal es poder predecir el valor de una variable objetivo a través de reglas simples extraídas de los datos entregados para el aprendizaje. Un ejemplo simple de esto es poder predecir si una persona entregada es o no presidente de un país latinoamericano. Para eso se le entrega un número de presidentes de distintos países entre Latinoamérica, Europa y Asia; el algoritmo identificará datos importantes que los diferencien y los separará en subgrupos (Ramas) hasta que haya una rama que solo contenga data parecida. La cantidad de ramas dependerá de la cantidad de datos que se entregue o contenga la base de datos.

Visto desde los fundamentos matemáticos del algoritmo, se entregan $x_i \in \mathbb{R}^n, i = 1, ..., m$ que son m datos puestos como un vector de $\mathbb{R}^n$ y estos se etiquetan con $labels\ y \in \mathbb{R}^m$, el árbol de decisión divide el espacio de forma recursiva tal que los vectores x_i con etiquetas iguales sean agrupados [11]. Un nodo es de donde salen 2 ramas nuevas, se toma la información en un nodo l representado por Q y para cada separación tendremos S con $S = (j, t_l)$ con j una característica y un límite t, entonces se busca que el nodo separe la información en 2 ramas tal que la rama de la izquierda $Q_{left}(S)$ contenga todos los datos o pares (x, y) tal que el valor de x sea menor al parámetro t_l y que la rama de la izquierda $Q_{right}(S)$ contenga todo lo que había en Q menos lo que se clasificó en $Q_{left}(S)$.

Para calcular la impureza de l se utiliza 'Impurity Function' $H()$, esta depende se se utiliza para regresiones o clasificaciones, las más utilizadas para clasificación son:

$$\text{GINI: } H(X_l) = \sum_k p_{lk} \cdot (1 - p_l k)$$

$$\text{Entropía: } H(X_l) = -\sum_k p_{lk} \cdot log(p_l k)$$

$$\text{Misclassification: } H(X_l) = 1 - max(1 - p_l k)$$

Donde X_l son los datos de entrenamiento en el nodo l.

Entonces obtenemos que $G(Q, S) = \frac{n_{left}}{N_l} H(Q_{left}(S)) + \frac{n_{right}}{N_l} H(Q_{right}(S))$ y se seleccionan los parámetros que minimicen la impuridad $S* = argmin_S G(Q, S)$ y luego se hace una recursión sobre estos 2 nuevos sets (Rama izquiera y derecha) hasta que se complete la profundidad máxima, $N_l < min_{samples}$ o $N_l = 1$.
En este artículo se utiliza Decision Tree para clasificación, así entonces los criterios de clasificación del método se muestran de la siguiente manera: El objetivo es un resultado de clasificación con valores 0,1,...,k-1, para el nodo m, que representen una región R_l con N_l observaciones, entonces se tiene que $p_{lk} = 1/N_l \sum_{x_i \in R_l} I(y_i = k)$ es la proporción de observaciones de la clase k en el nodo m [12]. Esta información está fundamentada sobre la documentación de la biblioteca SciKit-Learn.

2.3.2. Random Forest

Random Forest o en español, Bosques Aleatorios, es un algoritmo de clasificación correspondiente al aprendizaje automatizado supervisado. Un *Random Forest* está comprendido por árboles, de manera que mientras más árboles se tenga en el algoritmo, más robusto es. Estos árboles son justamente los árboles de decisión (Decision Tree) presentados en el apartado 2.3.1. El algoritmo de *Random Forest* consiste en dos partes que pueden ser resumidas por la siguiente situación; Supóngase que usted quiere comprar el mejor auto del mercado (no se tenga en cuenta el dinero a gastar), pero no sabe cual auto es el mejor del mercado. Para ello recurre a diferentes métodos de entrevistas, donde toma muestras de diferentes personas que hayan poseído más de un auto de gama alta. Así, usted crea una lista con todos los autos que le fueron recomendados, y a las personas las llama a votar entre los autos de esa lista, para así obtener el mejor auto del mercado. En esta situación se tiene implementado el algoritmo de *Random Forest*; En primera instancia recopila los datos a partir de las opiniones de personas (análogo a *Decision Tree Classifier*), para luego, en segunda instancia escoger a partir de los votos de las personas el mejor auto del mercado (algoritmo de *Random Forest*).

De manera formal, el algoritmo de *Random Forest* consiste en primer lugar en seleccionar muestras aleatorias de una base de datos B. Se debe tener en cuenta que cada muestra de la base de datos tenga una etiqueta, dado que el algoritmo corresponde a un clasificador de aprendizaje supervisado. En segundo lugar, se debe construir un árbol de decisión para cada muestra y obtener una predicción de cada árbol. Así, mientras más muestras se tenga, entonces más árboles se utilizan y por ende el bosque aleatorio es más denso y proporciona más información. En tercer lugar se debe votar por cada predicción resultante del paso anterior, de manera que la predicción con más votos corresponde a predicción a utilizar por el algoritmo [13].

Un gran problema del algoritmo es conjugar la precisión de la predicción con el *overfitting* – efecto obtenido al sobreentrenar un algoritmo – generado en la utilización de una gran cantidad de árboles de decisión. Es por ello que para efectos prácticos se utiliza la librería *Scikit Learn* [9], con su módulo *Ensemble methods*. La implementación del clasificador entonces viene dada en su totalidad por la librería, donde se pueden escoger parámetros que ayuden a mejorar la precisión de la predicción, pero que a su vez aumenten el tiempo de ejecución del algoritmo.

2.3.3. Gaussian Naive Bayes

Para describir de manera completa *Gaussian Naive Bayes*, primero se debe conocer sobre los métodos *Naive Bayes* en general, los cuales son algoritmos de aprendizaje supervisado basados en la aplicación del teorema de Bayes, además de suponer que se presenta independencia condicional entre cada par de características para cierto valor en una clase particular; que tienen por objetivo la predicción de clases a partir de características entregadas para el entrenamiento del algoritmo. La principal ventaja de estos métodos es que requiere una pequeña cantidad de datos de entrenamiento para dar paso a la clasificación, y una de sus principales aplicaciones es la clasificación de textos, junto con la extracción de frases clave.

Dado un vector de n características dependientes $(x_1, \dots, x_n)$ y una variable de clase y, el teorema de Bayes establece lo siguiente[14]:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \cdot P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

La fórmula se simplifica al utilizar la suposición de independencia condicional (1), quedando, para todo i, de la siguiente manera (2):

$$P(x_i \mid y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i \mid y) \tag{1}$$

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)} \tag{2}$$

Como $P(x_1, ..., x_n)$ se obtiene como constante de entrada gracias a la base de datos de entrenamiento, se puede utilizar la siguiente regla de clasificación:

$$P(y|x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \Rightarrow \hat{y} = arg\,\max_{y} P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Para estimar el valor de P(y) se utiliza la regla del máximo a posteriori (MAP).

Cuando se trabaja con datos continuos,la implementación del algoritmo *Gaussian Naive Bayes* es recurrente, la cual implica asumir que la probabilidad de las características presentan una distribución gaussiana de la siguiente manera:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \cdot \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Los parámetros σy y μ_y se estiman en base al cálculo de la máxima verosimilitud.

2.3.4. Logistic Regression

Este es un tipo de análisis de regresión para predecir una variable categórica, como en este caso lo son las distintas clases (número 1, letra a, letra F, etc). El método se basa en calcular la probabilidad de que pase un evento para asignarle la categoría final. Para el caso binario, donde 0 representa una

clase a clasificar y 1 representa a la otra (por ejemplo hombre o mujer, en riesgo o sin riesgo, sí o no, verdadero o falso), se hace una regresión lineal de la forma: $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$, donde β_i representa el peso de una característica X_i. El problema es que esta expresión puede arrojar un valor mayor a 1, por lo cual se utiliza una función que esté comprendida entre 0 y 1, en este caso la función logística (sigmoide):

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Entonces se reemplaza x por la regresión lineal:

$$\sigma(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n)}}$$

Entonces, reemplazando los valores y obteniendo la probabilidad, si el número da más cerca de 1 se etiquetará como 1, en cambio si da más cerca de 0 se etiquetará como 0[15].

2.3.5. Multilayer Perceptron

Es un algoritmo de aprendizaje supervisado que, entrenando en un dataset, aprende una función $f(a) : R^m \to R^k$, donde m es la dimensión del input y k la del output.

Dado un conjunto de n características X = $(x_1, ..., x_n)$, el algortimo presenta una capa de entrada (input), una capa de salida u objetivo (output) y capas no lineales ocultas, las cuales, con cada neurona, transforma los valores provenientes de las capas anteriores, generando una suma lineal ponderada $w_1 x_1 + w_2 x_2 + ... + w_n x_n$ con pesos w_i establecidos mediante en el entrenamiento previo. Luego, a los valores obtenidos en cada neurona se les aplica una función de activación no lineal (en el caso de estudio la función de unidad lineal rectificada es la utilizada, la cual entrega f(x) = máx$(0, x)$, la misma que se presenta por defecto) para dar paso a la siguiente capa oculta hasta llegar a la capa de salida, la cual transforma los valores obtenidos en valores de salida[16].

Con el fin de mejorar la predicción del algoritmo se implementan solucionadores para la optimización de los pesos w_i. En el caso implementado, el solucionador utilizado es el optimizador estocástico basado en gradiente propuesto por Diederik P. Kingma y Jimmy Ba.

2.4. Problemas con Desbalanceo

La base de datos de EMNIST está desbalanceada, por lo que aplicarle los métodos de clasificación directamente provocaría resultados que no son totalmente un reflejo de la capacidad de clasificación del algoritmo. Cuando un algoritmo es entrenado con datos desbalanceados, aquellas clases que estén sobre representadas tendrán más peso a la hora de clasificar.

El efecto que se produce es el siguiente: si tengo una población con 99 % datos A y 1 % datos B, si el algoritmo siempre predice que un dato nuevo es A, tendrá un *accuracy* de 0.99, sin embargo jamás podrá podrá predecir un dato B. Para evitar esto y que lo que esté pasando sea una real clasificación (no solo una maximización del *accuracy*) se utilizó la técnica de balanceo llamada *subsampling* [17]. La técnica *subsampling* consiste en disminuir los datos de las clases que estén sobrerepresentadas, de esta forma se hizo que todos los datos estuvieran en igual proporción. Sin embargo, esto también significó que nuestra base de datos de entrenamiento balanceada tuviese solo 1/7 de los datos en comparación con la desbalanceada.

Para que al comparar el desempeño entre la clasificación balanceada y la desbalanceada, se creó una base de datos de testeo balanceada también, pues esta también presentaba desbalanceo. Al estar el test desbalanceado significaría que no se podría identificar correctamente el impacto entre balanceo y desbalanceo.

2.5. Base de datos propia

Cada integrante del grupo proporcionó los 62 símbolos contenidos en *EMNIST*, es decir los dígitos y letras mayúsculas y minúsculas. Se entrenaron todos los métodos de clasificación con *EMNIST* luego de haberle aplicado un método de balanceo. La finalidad de esta sección es comparar las letras de cada uno. Además se incluyeron los mismos símbolos pero extraídos de *Word Office* con la fuente 'Arial Black'.

2.6. Resumen metodología

Los métodos de reducción de dimensionalidad fueron implementados para las bases de datos *Digits* y *MNIST*. Los métodos de clasificación fueron aplicados a *MNIST* y a *EMNIST*, para esta última, que es una base de datos desbalanceada, se implementaron los métodos en 3 situaciones diferentes, como se muestra en la siguiente tabla:

Situaciones	Train B	Train D	# de Train	Test B	Test D	# de Test
1	$False$	$True$	697932	$False$	$True$	116323
2	$True$	$False$	117490	$True$	$False$	19654
3	$False$	$True$	697932	$True$	$False$	19654

Tabla 1: Situaciones de entrenamiento y testeo.

Donde 'B' significa que ese grupo de datos está balanceado y 'D' significa que el grupo de datos está desbalanceado, además '#'es la cantidad de datos en total que se dispusieron en cada grupo de datos.

3. Resultados

3.1. Reductores de dimensionalidad

Los reductores de dimensionalidad tienen como finalidad principal poder visualizar bases de datos que en su estado original le es imposible al ser humano imaginar, entonces luego de haber aplicado los reductores a las bases de datos, buscando obtener gráficos en 2D, se debería poder tener un buen instrumento para ver y mostrar las bases de datos. Aquí se encuentran varios problemas, particularmente con la base de datos *MNIST*, debido a que es una base de datos muy grande (más de 60000 datos) algunos métodos de complejidad computacional elevada no lograban terminar su ejecución en un tiempo razonable. Los métodos en especifico que no terminaron de ejecutar en un tiempo razonable con *MNIST* son *t-SNE*, *Isomap* y *MDS*. Por otro lado con la base de datos *Digits*, todos los métodos lograron terminar de computar en un tiempo razonable (menos de 1 hora).

Los siguientes gráficos son los resultados obtenidos para cada reductor de dimensionalidad en los casos que se logró computar en un tiempo razonable, en estos gráficos cada diferente color representa un símbolo diferente, en el caso de *Digits* y *MNIST* serán solo dígitos del 0 al 9.

3.1.1. Tabla de tiempos

Poseer los tiempos de ejecución para los diferentes métodos de reducción es útil al momento de analizar su complejidad y posterior viabilidad en la utilización de base de datos. A continuación se presenta la tabla de tiempos de ejecución para la base de datos *Digits* con los cinco métodos de reducción utilizados.

	PCA	t-SNE	MDS	Isomap	T-SVD
Tiempo en segundos	0,016	10,151	185,751	2,513	0,009

Tabla 2: Tiempos de cómputo algoritmos de reducción de dimensión.

3.1.2. Principal Component Analysis

Resultados del método *PCA* aplicado a *Digits* y *MNIST*

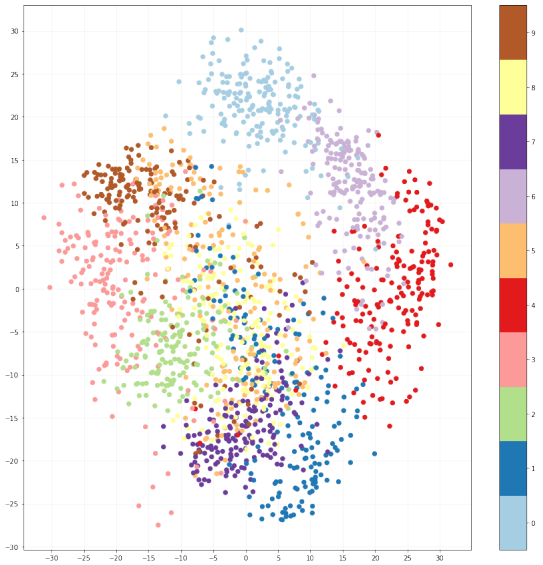


Figura 7: Gráfico de *PCA* aplicado a *Digits*

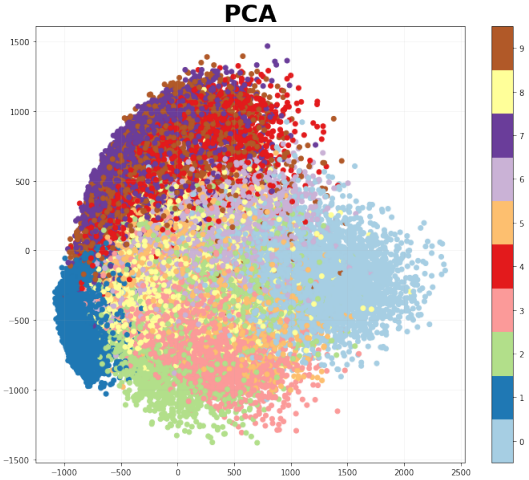


Figura 8: Gráfico de *PCA* aplicado a *MNIST*

3.1.3. t-Distributed Stochastic Neighbor

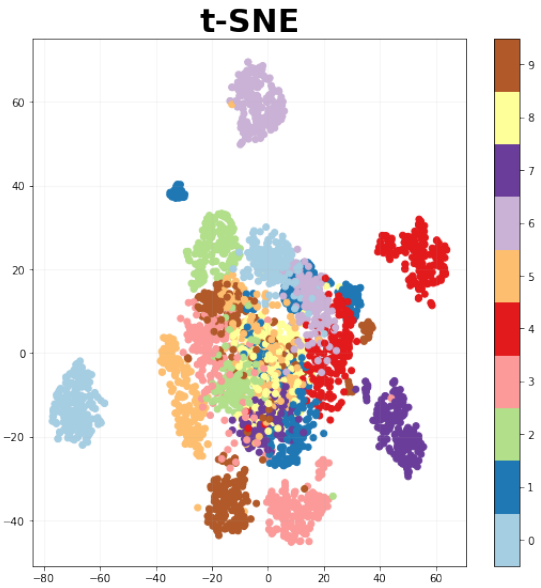


Figura 9: Gráfico de *t-SNE* aplicado a *Digits*

3.1.4. Truncated Singular Value Decomposition

Resultados del método *TSVD* aplicado a *Digits* y *MNIST*.

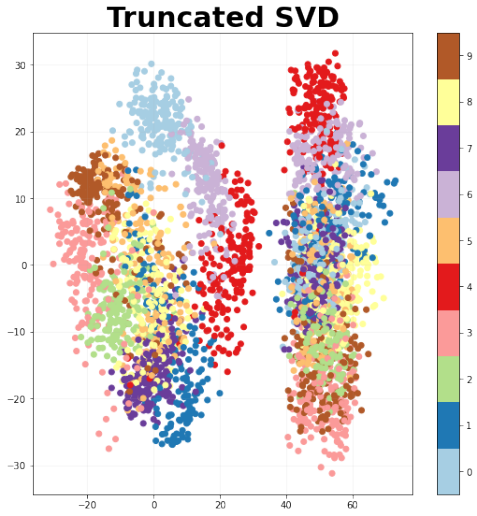


Figura 10: Gráfico de *TSVD* aplicado a *Digits*

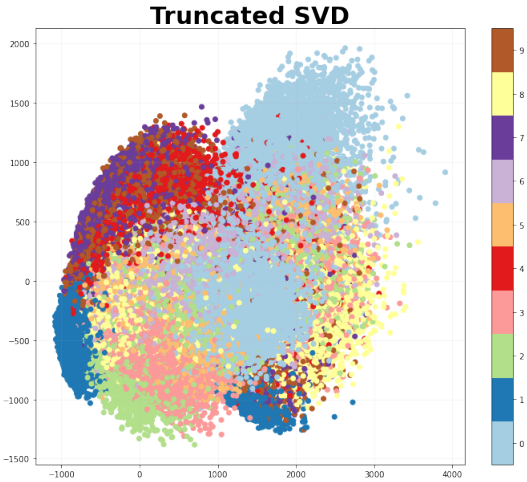


Figura 11: Gráfico de *TSVD* aplicado a *MNIST*

3.1.5. Isomap

Resultado del método *ISOMAP* aplicado a *Digits*

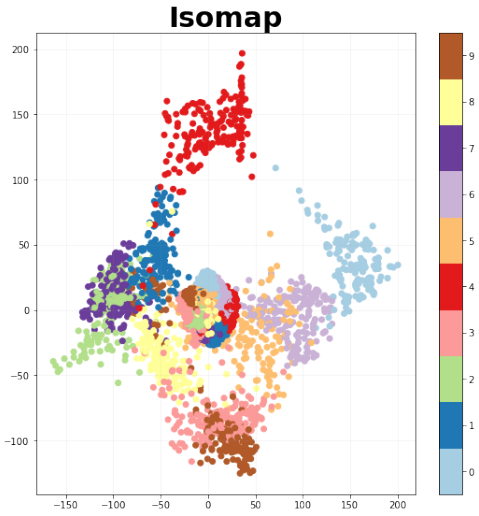


Figura 12: Gráfico de *ISOMAP* aplicado a *Digits*

3.1.6. Multidimensiona Scaling

Resultado del método *MDS* aplicado a *Digits*

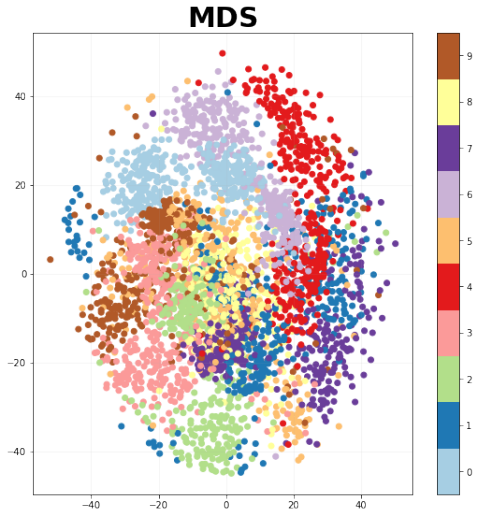


Figura 13: Gráfico de *MDS* aplicado a *Digits*

3.2. Clasificadores

Recordemos que lo que se busca al aplicar los métodos de clasificadores es poder comparar su rendimiento en 3 casos: cuando los datos para entrenar y para testear están desbalanceados, cuando están ambos balanceados luego de aplicar métodos de balanceo y por último en el caso de que se entrene el método con datos desbalanceados y se pruebe con datos balanceados. Se utilizaron estas combinaciones para que al comparar las matrices de confusión de los métodos, que tienen cantidad de datos diferentes, fuese claro la precisión de estas, es decir que si en ambas predijo un dato correctamente el mismo número de veces, podemos decir que la que tenía menos datos fue más precisa. Es fundamental tener claro que estos métodos fueron aplicados a la base de datos *EMNIST*.

Otra manera de evaluar es el rendimiento 'F1-Score', que es un promedio armónico de precisión y recall, esto nos entrega ratios que nos permiten juzgar el porcentaje de acierto en la predicción, si es 1 entonces acertó en todas las predicciones.

Las matrices de confusión también son útiles al momento de evaluar el rendimiento de un clasificador, estas entregan en cada entrada a_{ij} información de la siguiente manera, a_{ij} es la cantidad de veces que el símbolo i fue predicho como j.

3.2.1. Resumen de Accuracy MNIST y matrices de confusión

MNIST es una base de datos balanceada por lo tanto solo hay que tabular los 'F1-Score' de cada clasificador a aplicado a la base de datos.

Clasificador	F1-Score
Gaussian Naive Bayes	0,56
Decision Tree	0,88
Logistic Regression	0,93
Random Forest	0,97
Multilayer Perceptron	0,97

Tabla 3: Rendimiento clasificadores.

A continuación se muestran las matrices de confusión para cada clasificador.

	0	1	2	3	4	5	6	7	8	9
0	870	0	79	32	19	71	12	0	13	5
1	0	1079	25	39	2	25	12	15	72	7
2	3	2	266	6	5	1	3	2	3	3
3	5	1	91	353	4	20	1	10	7	6
4	2	0	5	2	168	3	1	5	3	1
5	5	0	2	3	7	44	7	1	11	0
6	31	10	269	51	63	40	895	5	12	1
7	1	0	4	8	7	2	0	280	4	13
8	35	38	271	409	210	586	26	39	648	18
9	28	5	20	107	497	100	1	671	201	955

Figura 14: Matriz de confusión de *Gaussian Naive Bayes* aplicado a *MNIST*

	0	1	2	3	4	5	6	7	8	9
0	914	1	14	6	6	14	17	4	9	15
1	0	1093	8	6	4	7	5	13	7	4
2	8	10	881	31	8	6	9	25	34	10
3	10	2	35	859	6	37	6	20	39	15
4	4	2	12	10	869	11	15	6	17	42
5	12	6	13	40	4	753	18	7	21	12
6	10	7	14	4	13	18	855	3	18	6
7	2	2	24	7	7	4	2	921	9	16
8	14	10	23	21	21	25	26	11	788	25
9	6	2	8	26	44	17	5	18	32	864

Figura 15: Matriz de confusión de *Decision Tree* aplicado a *MNIST*

-

	0	1	2	3	4	5	6	7	8	9
0	963	0	3	4	1	11	9	1	8	9
1	0	1112	10	1	1	2	3	6	7	7
2	0	4	926	21	7	1	7	24	6	0
3	3	2	15	916	3	33	3	5	23	11
4	1	0	6	1	910	11	7	7	6	25
5	3	1	4	26	0	776	16	1	26	6
6	4	3	15	3	9	11	910	0	10	0
7	4	2	8	9	7	6	2	951	10	22
8	2	11	42	22	10	35	1	3	869	7
9	0	0	3	7	34	6	0	30	9	922

Figura 16: Matriz de confusión de *Logistic Regression* aplicado a *MNIST*

	0	1	2	3	4	5	6	7	8	9
0	971	0	5	0	1	2	5	2	5	5
1	0	1122	0	0	0	0	3	2	0	4
2	0	4	1000	8	1	1	0	18	5	2
3	0	3	5	974	0	9	1	0	9	10
4	0	0	4	0	953	3	3	1	8	9
5	2	2	0	8	0	861	5	0	6	6
6	3	2	3	0	5	6	937	0	5	1
7	1	0	9	9	0	3	0	991	5	5
8	2	2	6	8	3	5	4	2	920	7
9	1	0	0	3	19	2	0	12	11	960

Figura 17: Matriz de confusión de *Random Forest* aplicado a *MNIST*

-

	0	1	2	3	4	5	6	7	8	9
0	965	0	6	1	1	1	6	1	7	4
1	0	1122	3	0	0	0	2	3	2	6
2	1	3	987	3	3	0	1	9	5	0
3	2	2	11	991	0	16	1	17	8	11
4	0	0	1	0	944	1	2	5	5	9
5	5	2	1	7	1	853	6	2	6	6
6	2	2	3	0	3	7	933	0	6	0
7	2	1	3	0	11	2	0	979	2	8
8	3	3	17	7	4	7	7	4	925	11
9	0	0	0	1	15	5	0	8	8	954

Figura 18: Matriz de confusión de *Multilayer Perceptron* aplicado a *MNIST*

3.2.2. Resumen de Accuracy EMNIST y matrices de confusión

A diferencia del caso anterior con *MNIST*, *EMNIST* es una base de datos desbalanceada por lo tanto se puede hacer diferentes análisis con Train y Test balanceados y desbalanceados, como se muestra a continuación.

En la siguiente tabla se detallan los datasets con los que se encontraron las 'Accuracy' para cada clasificador, la letra 'B' después de Train y Test significa que el set de datos de Train o Test está balanceado y la 'D' que está desbalanceado, además se incluyen la cantidad de datos con las que se entrenó y testeó en cada caso. Los datos se balancearon con un método de balanceo explicado previamente.

Accuracy	Train B	Train D	# de Train	Test B	Test D	# de Test
A1	$False$	$True$	697932	$False$	$True$	116323
A2	$True$	$False$	117490	$True$	$False$	19654
A3	$False$	$True$	697932	$True$	$False$	19654

Tabla 4: Situación de entrenamiento y testeo para cada cálculo de 'Accuracy'.

Clasificador	Accuracy 1	Accuracy 2	Accuracy 3
Decision Tree	0,67	0,48	0,52
Random Forest	0,83	0,70	0,66
Gaussian Naive Bayes	0,30	0,25	0,26
Logistic Regression	0,72	0,60	0,53
Multilayer Perceptron	0,72	0,56	0,51

Tabla 5:'Accuracy' para cada situación de entrenamiento y testeo con cada clasificador.

A partir de la siguiente página se muestran las matrices de confusión, en la leyenda de cada figura se especifica a que método pertenece y con qué tipo de base de datos.

1019	2	10	11	30	0	14	6	1	1
2	466	0	13	3	0	1	25	3	0
8	0	105	0	23	1	0	1	0	0
19	10	4	1285	0	6	3	11	3	11
36	0	39	5	3482	7	9	3	1	0
1	2	0	3	3	72	2	2	0	0
17	1	0	3	4	1	108	0	2	3
1	23	0	15	3	2	0	1115	3	2
1	0	0	0	1	0	0	5	129	29
0	1	0	12	0	1	1	1	20	139

Figura 19: Matriz de confusión Train y Test D de la 'a' a la 'j' Decision Tree

160	0	2	5	4	1	11	1	0	0
1	212	0	6	0	1	2	11	1	3
6	0	147	1	10	1	0	0	0	0
4	4	2	205	1	1	2	3	1	7
6	0	17	0	211	2	3	0	1	0
0	0	0	3	0	113	2	3	3	0
10	4	1	2	1	3	86	1	1	2
0	16	0	2	0	0	2	208	1	1
0	6	0	3	0	1	1	2	138	20
2	4	0	10	0	0	5	1	19	176

200	1	1	1	5	0	3	1	0	1
0	189	0	5	0	0	1	10	0	0
5	0	80	0	17	1	0	0	0	0
4	0	1	221	0	2	0	2	0	3
5	0	7	0	264	0	2	0	0	0
1	2	0	2	2	56	1	2	0	0
13	0	0	1	2	1	52	0	0	2
1	6	0	2	1	0	0	233	1	2
1	0	0	0	0	0	0	3	90	25
0	1	0	12	0	1	1	1	20	138

Figura 20: Matrices de confusión Train y test B // Train D Test B de la 'a' a la 'j' Desicion Tree

1374	0	0	11	16	0	0	2	0	0
0	635	0	8	1	0	0	15	0	0
7	0	7	0	29	0	0	0	0	0
12	0	0	1586	1	0	0	2	0	1
12	0	0	3	3963	0	0	2	0	0
0	0	0	2	0	14	0	1	0	0
24	1	0	2	2	0	99	1	0	0
1	6	0	7	1	0	0	1335	0	0
0	0	0	1	0	0	1	1	130	6
0	1	0	16	0	0	0	1	18	183

Figura 21: Matriz de confusión Train y Test D de la 'a' a la 'j' Random Forest

257	0	0	2	1	0	2	0	0	0
0	278	0	2	0	0	0	3	0	1
7	0	199	1	6	0	0	0	0	0
2	1	0	285	0	0	0	1	0	5
2	0	10	0	283	1	0	0	0	0
0	0	1	2	0	203	0	1	0	2
7	0	0	1	2	0	140	1	1	1
0	3	0	4	0	1	0	269	0	0
0	1	0	0	0	1	1	0	175	17
2	0	0	8	0	2	2	1	21	243

265	0	0	1	4	0	0	1	0	0
0	235	0	2	1	0	0	6	0	0
6	0	5	0	18	0	0	0	0	0
3	0	0	298	0	0	0	0	0	0
1	0	0	0	304	0	0	0	0	0
0	0	0	1	0	10	0	1	0	0
16	0	0	1	0	0	49	0	0	0
1	1	0	3	0	0	0	283	0	0
0	0	0	1	0	0	1	0	99	4
0	1	0	16	0	0	0	1	18	182

Figura 22: Matrices de confusión Train y test B // Train D Test B de la 'a' a la 'j' Random Forest

980	1	0	26	68	1	1	5	0	1
0	546	0	21	3	0	0	39	0	6
8	1	16	0	42	0	1	0	0	0
9	5	0	1424	2	0	1	16	4	7
48	3	1	13	3487	0	0	2	0	0
0	1	0	2	3	38	0	0	0	3
9	1	0	0	2	0	60	1	0	0
2	22	0	23	0	0	0	1126	6	2
1	0	0	8	0	2	1	5	86	11
0	4	0	33	0	0	2	4	9	159

Figura 23: Matriz de confusión Train y Test D de la 'a' a la 'j' Logistic Regression

184	0	0	5	5	1	3	1	0	0
0	258	0	3	1	0	0	12	1	5
4	1	187	0	15	0	1	0	0	0
0	3	0	261	0	0	0	3	1	11
11	0	21	0	234	0	0	0	0	0
0	0	0	2	0	181	1	1	4	3
4	0	0	0	0	0	93	0	1	0
2	10	0	5	0	0	0	216	9	1
0	0	0	3	0	4	0	1	124	11
0	5	0	14	0	2	4	2	14	214

184	0	0	4	11	0	0	1	0	0
0	212	0	7	3	0	0	14	0	3
6	1	11	0	32	0	1	0	0	0
1	1	0	263	1	0	0	4	1	2
4	0	0	1	268	0	0	0	0	0
0	1	0	2	2	28	0	0	0	2
8	0	0	0	1	0	30	0	0	0
2	4	0	6	0	0	0	233	2	0
0	0	0	4	0	1	0	3	66	6
0	4	0	33	0	0	2	4	9	158

Figura 24: Matrices de confusión Train y test B // Train D Test B de la 'a' a la 'j' Logistic Regression

492	5	54	10	90	0	6	0	6	1
1	187	1	19	0	0	0	2	213	23
3	1	218	1	7	0	0	0	4	0
20	9	5	867	6	0	6	1	213	96
54	2	796	6	1306	2	1	0	23	1
1	2	0	5	1	84	0	0	67	2
16	8	0	3	6	7	6	1	58	18
12	277	5	58	1	1	1	44	413	27
1	0	0	1	0	0	0	0	37	1
1	2	2	11	1	2	0	0	74	78

Figura 25: Matriz de confusión Train y Test D de la 'a' a la 'j' Gaussian Naive Bayes

36	3	0	3	116	0	1	0	0	0
0	88	0	9	4	0	0	1	59	1
1	1	97	1	81	0	0	0	4	0
2	2	0	176	7	0	0	2	17	7
1	0	15	0	245	0	0	0	1	0
1	2	0	5	5	37	0	0	42	1
7	5	0	0	9	3	2	0	15	2
0	74	0	7	7	0	0	17	60	0
0	0	0	1	2	0	0	0	20	0
1	2	1	18	4	2	0	0	57	37

96	1	4	3	18	0	1	0	1	0
0	71	1	7	0	0	0	1	77	9
1	1	160	1	6	0	0	0	4	0
2	3	1	164	2	0	0	0	28	19
7	0	75	0	90	0	1	0	1	0
1	2	0	4	1	70	0	0	52	2
12	5	0	1	3	6	3	0	29	6
3	66	1	10	1	0	0	7	88	3
0	0	0	1	0	0	0	0	30	1
1	2	2	11	1	2	0	0	74	77

Figura 26: Matrices de confusión Train y test B // Train D Test B de la 'a' a la 'j' Gaussian Naive Bayes

-

1047	1	2	5	61	0	0	1	0	0
0	571	0	13	0	0	0	55	0	0
8	0	104	0	98	0	0	0	1	0
74	34	0	1216	1	0	0	20	0	1
114	1	25	1	3563	2	0	2	0	0
0	0	0	1	0	121	0	8	0	0
4	1	0	1	1	1	70	0	0	0
4	22	0	16	0	0	0	1202	3	0
1	1	0	1	1	0	0	4	113	17
1	0	0	12	0	1	2	3	15	105

Figura 27: Matriz de confusión Train y Test D de la 'a' a la 'j' Multilayer Percerptron

175	0	7	3	8	0	0	1	1	0
1	229	0	9	0	0	0	6	1	1
8	1	267	0	15	0	0	0	0	0
0	10	1	250	0	0	0	1	1	7
22	0	50	1	218	1	1	0	0	0
0	0	0	2	0	50	1	3	0	0
5	1	0	1	0	0	81	0	0	3
1	16	0	4	0	0	0	231	2	0
0	1	1	2	0	0	1	1	96	18
1	1	0	7	0	0	3	0	18	151

203	0	1	0	8	0	0	0	0	0
0	222	0	4	0	0	0	14	0	0
7	0	76	0	74	0	0	0	0	0
13	8	0	225	0	0	0	4	0	0
7	0	3	0	275	1	0	0	0	0
0	0	0	1	0	97	0	8	0	0
3	0	0	1	0	1	38	0	0	0
2	4	0	6	0	0	0	258	1	0
0	1	0	1	1	0	0	2	86	7
1	0	0	12	0	1	2	3	15	104

Figura 28: Matrices de confusión Train y test B // Train D Test B de la 'a' a la 'j' Multilayer Perceptron

3.3. Base de datos propia

Ahora se presentan, tabulados, los porcentajes de acierto de cada integrante del grupo con los símbolos que proporcionó para la base de datos propia y los resultados de la fuente 'Arial Black'.

Clasificador	DTC	RFC	GNB	LGR	MLP	Promedio
Vicente Agüero	0,16	0,34	0,02	0,48	0,37	0,274
Antonia Reyes	0,1	0,16	0,06	0,19	0,21	0, 144
Javier Takahashi	0,16	0,31	0,18	0,42	0,27	0, 268
Felipe Valdés	0,05	0,02	0,02	0,16	0,08	0,066
Sebastián Ferrá	0,1	0,05	0,00	0,11	0,05	0,062
Arial Black (Word)	0,1	0,19	0,02	0,10	0,06	0,094

Tabla 6: Resultados de testeo con base de datos propia y entrenamiento *EMNIST* balanceada.

4. Discusión

4.1. Reductores de dimensionalidad

La utilización de las bases de datos *Digits* y *MNIST* en los métodos de reducción deja de lado a la base de datos *EMNIST* también analizada en la investigación. El motivo de no utilizar *EMNIST* es por dos sencillos motivos. El primero corresponde al tamaño de la base de datos, ya que posee más de 800000 imágenes representadas por matrices, lo que llevaría un tiempo de ejecución enorme (teniendo en cuenta que *MNIST*, siendo menos densa que *EMNIST*, no pudo ejecutar ciertos métodos). El segundo es por la composición de la base de datos, lo que contradice el objetivo principal de la reducción de dimensionalidad. Desde luego, *EMNIST* posee 62 tipos diferentes de clasificación, lo que conllevaría a 62 colores en la gráfica que deben ser muy diferentes para una mejor visualización, pero es evidente que con tal cantidad de colores existirían diferencias mínimas entre un color y otro, en consecuencia, no ayudaría en la visualización de la base de datos.

Junto a lo anterior, se vio que las complejidades de los diferentes métodos de reducción son en su totalidad correspondientes a tiempos polinomiales. Por ejemplo, *Isomap* posee una complejidad de $\mathcal{O}(n^2)$, en tanto la de *MDS* corresponde a $\mathcal{O}(n^3)$, lo que conlleva a que, en bases de datos grandes, la diferencia del tiempo de ejecución en comparación con una base de datos menor sea mucho mayor para *MDS* que para *Isomap*. En efecto, utilizar ciertos métodos de reducción conllevan un coste de oportunidad (en tiempo) diferente para cada caso específico, es por ese motivo que no se utilizaron muchos métodos de clasificación en la base de datos *MNIST*. Más aún, el tiempo de ejecución del método *MDS* con *Digits* presentado en la tabla 2 de la sección 3.1.1 superó los dos minutos, teniendo en consideración que *Digits* posee aproximadamente 33 veces una cantidad menor de datos que *MNIST*, y que *MDS* se ejecuta en $\mathcal{O}(n^3)$, entonces el tiempo de ejecución para *MNIST* sería considerablemente mayor a dos minutos.

El desempeño de cada método es bastante intuitivo de analizar. Los métodos de reducción generan en el plano una visualización de la base de datos, de manera que cada color represente a un tipo particular de dato. En el caso de las bases de datos utilizadas, la separación entre cada punto de un color específico en el plano señala la semejanza o diferencia visual entre un dígito y otro. Así, en la Figura 7 y Figura 8 se aprecia que el método *PCA* no es capaz de generar grupos particulares para puntos de colores iguales (tanto para la base de datos *Digits* como para *MNIST*), lo que lleva a la idea que existen dígitos que a este método se le parecen. Efectivamente, si se realiza un análisis más profundo de la Figura 7 se aprecia que el cero (dígito cero) está más alejado de los demás, por lo que en este método el cero es un buen ejemplo de un número que no se parece a otros. Sin embargo, de la misma figura se aprecia que el 8 se parece al 5 y al 9, dado que su representación no se encuentra agrupada, sino más bien mezclada. Otro dato importante a mencionar es la separación que realizó el *PCA* para los diferentes dígitos en la base de datos *MNIST*, donde prácticamente todos los puntos se encuentran mezclados.

El método *t-SNE* sigue la misma línea que la reducción realizada por el *PCA*, donde el dígito cero se encuentra bien aislado del resto (salvo algunos puntos que se encuentran mezclados en medio). No obstante se aprecia que la mayoría de los dígitos convergen al centro (se encuentran acumulados en su gran mayoría en ese lugar), lo que da un indicio de la calidad de reducción realizada por el *t-SNE*. El caso de *TSVD* es muy parecido, en *MNIST*, al método *PCA*, donde todos los valores

se encuentran acumulados a la izquierda. En tanto, para *Digits* se presentan como dos columnas con todos los dígitos mezclados. Esta similitud ya fue mencionada en la sección 2.2.3, debido a que ambos métodos son lineales. Finalmente *Isomap* y *MDS* realizaron una reducción con puntos mucho más aislados, donde se puede apreciar claramente grupos formados por colores iguales, que representan dígitos iguales. Aún así, cabe destacar que *Isomap* realiza una buena agrupación de los datos, pero persiste el problema de acumulación de datos en algún sector del plano.

4.2. Clasificadores en MNIST

Los resultados de la clasificación en la base de datos *MNIST* fueron en casi todos los casos bastante altos. En el método de *Random Forest* y en la red neuronal *Multilayer Perceptron*, los resultados obtenidos fueron de 0.97, lo que significa que de cada 100 números, solo se equivocan en 3. Los demás algoritmos tuvieron desempeños similares, sin embargo este no es el caso de *Gaussian Naive Bayes*. Esto se puede deber al funcionamiento probabilístico y de independencia del algoritmo en sí. Por ejemplo, si hay dos espacios en blanco más o menos similares, podemos decir que probablemente sea un 8, pues es el único número con dos círculos blancos en él. Sin embargo, *Gaussian Naive Bayes* interpreta esto de una forma diferente, por lo que las características de cada dato son analizadas de forma independiente. Al ver la matriz de confusión del *Gaussian Naive Bayes* nos podemos fijar que el número 8 básicamente lo confundió con casi todos los números. Esto pues, al compartir ciertas bases de sus estructuras con otros números, como el 3, interpreta de manera independiente esos pixeles en común y da un resultado equivocado.

4.3. Base de datos propia

La base de datos propia proporciona 62 imágenes para cada integrante. Al ejecutar los métodos de clasificación a las bases de datos se puede apreciar un bajo porcentaje de acierto en todos los métodos (nunca mayor a 50 %). En particular, la utilización de *Gaussian Naive Bayes* proporciona el menor porcentaje de acierto, en tanto, clasificadores como *Multilayer Perceptron* o *Logistic Regression* proporcionan los porcentajes más elevados. Además, los porcentajes más elevados se obtuvieron principalmente con letras de mayor grosor de escritura, así como aquellas con menor grosor obtenían menor acierto. Sin embargo, se puede apreciar que esto no siempre se cumple, dado que la letra proporcionada por 'Arial Black', siendo de un grosor similar a la de Vicente Agüero, posee un porcentaje de acierto muy bajo. Más aún, la principal diferencia entre estas letras corresponde al tipo de letra, de hecho, se debe tener en cuenta que la base de datos de testeo proporciona imágenes de letras y números escritos a mano, lo que da un indicio de como afecta esto al porcentaje de acierto.

4.4. Balanceo de datos

Se hicieron tres pruebas (A1, A2 y A3) para tener una perspectiva de cómo comparar cuanto afecta el desbalanceo en los métodos. Se llegó a la conclusión de que es más útil y preciso comparar A2 y A3 porque sus dataset de testeo están balanceados; en A1 hay más clases sobre representadas en el dataset de testeo, y al estar también desbalanceado el dataset de entrenamiento estos aciertos extra influyen el el porcentaje. En cambio como en A3 el testeo está balanceado, la tendencia del modelo a predecir las clases sobre representadas no afecta en el porcentaje de accuracy.

Para analizar la diferencia entre entrenar el método con una base de datos balanceada y otra desbalanceada es muy conveniente analizar las matrices de confusión.

Partiendo con las de Decision Tree; si se observa la diagonal de la matriz con Train desbalanceado y Test balanceado (Figura 20), hay mucha diferencia entre las predicciones acertadas dependiendo del símbolo al que representan, por ejemplo hay números pequeños como 56 y 52 y otros más grandes como 264 y 233. Esto se debe al desbalanceo de datos, ya que al modelo le resulta más adecuado predecir las clases que están sobre representadas para maximizar la accuracy.

Esta diferencia es menor en la matriz de confusión con Train y Test balanceado (Figura 20): notamos que los números que solían ser muy pequeños subieron a a 113 y 86. Los números mayores bajaron un poco (a 211 y 208), ahora la diferencia no es tan grande, por lo que el método está trabajando mejor y no solo prediciendo más de las clases sobre representadas.

En las matrices de Random Forest (Figura 22) se repite el mismo fenómeno, incluso más notorio. En la diagonal de la matriz de confusión con Train desbalanceado y Test balanceado los datos tienen una amplitud más grande, siendo los números de acierto máximo 304 y 298, mientras que los mínimos son de tan solo 5 y 10. En cambio en la diagonal de la matriz con Train y Test balanceado el rango es mucho menor; siendo los números de acierto máximo 285 y 283, mientras que los mínimos subieron a 140 y 175.

Siguiendo con clasificadores, las matrices de confusión para *Logistic Regression* (Figuras 23 y 24) en el caso de Train y Test Desbalanceado se observa, como es de esperarse, que haya bastante error: que clasifique mal muchos datos y que las diferencias entre las entradas de la matriz diagonal sean muy grandes. Estas diferencias logran achicarse cuando se balancean ambas (Train y Test), llegando a la primera matriz de la figura 24. Para el balanceo hay que hacer recortes de la cantidad de datos, por lo cual se observa esa baja general de los numero y quedan todos más parecidos. En la última matriz (matriz 2 Figura 24) vemos que sigue habiendo una gran diferencia entre los elementos de la diagonal principal, pero que se logra menos error en la clasificación. Esto se debe a que el programa aprende que en las diagonales existen esas diferencias, ya que es entrenado con un Train Desbalanceado, dónde hay muchos símbolos de un tipo y muy pocos de otro, entonces es esperable que se vea eso reflejado es un utilización para un Test balanceado.

Cuando analizamos las matrices de confusión del método *Gaussian Naive Bayes*, nos encontramos que la diferencia entre el Train y Test desbalanceados y balanceados (ambos) es muy pequeña, en la Figura 25 se nota muy claramente que son Train y Test Desbalanceados, hay números muy grandes como 277, muy lejos de la diagonal principal. Esto suele suceder cuando se entregan muchos datos

de un tipo y muy pocos de otro, clasifica 'asegurándose', muchos datos de los que se le entregan para Test como el dato que estaba desbalanceado hacia arriba en el Train. Es claro también que este método es menos útil que el resto con este tipo de bases de datos, es decir, funciona mucho peor si los datos están desbalanceados y aunque se le apliquen métodos de balanceo, no mejora mucho. Lo que sí es útil es que al balancear el train y el test se logra disminuir un poco las diferencias entre los componentes de la diagonal principal, pero siguen habiendo números muy pequeños como un 2 junto a un 245. Esas diferencias nos dicen que la base de datos sigue estando desbalanceada o que el clasificador funciona muy mal.

Para el último clasificador, *MLP*, se nota que al igual que en las anteriores, funciona igual de mal con datos desbalanceados. Siendo más específico, funciona muy bien para los datos que se encuentran en mayoría, pero luego cuando se utilizan Train y Test balanceados, la predicción mejora considerablemente. Se logran hacer los componentes de las diagonales principales más parecidos y se achican los errores, es decir, los números alejados de la diagonal principal. En el caso de Train Desbalanceado y Test Balanceado, se sigue notando que hay más error, reflejado en ese 74 alejado de la diagonal principal, pero sigue funcionando muy bien en comparación con, por ejemplo, *Gaussian Naive Bayes*.

4.5. Bases de datos con diferentes tipos de símbolos

Existe una gran cantidad de bases de datos en internet con una gran diversidad de símbolos [18], sin embargo, no todos los símbolos se comportarán de la misma manera en el proceso de clasificación. Dependiendo de los símbolos, los algoritmos necesarios para lograr clasificaciones eficientes son muy diferentes. Esto se debe a que la naturaleza del formato, la proporción, las similitudes y las variantes, van cambiando entre símbolos.

Los símbolos chinos son particularmente difíciles de clasificar, ya que presentan características que ponen retos a los algoritmos. Los carácteres chinos son aproximadamente 50000, sin embargo el 99.65 % del uso de estos se concentra en apenas 3375 carácteres comunes. Esto pues, los símbolos chinos están compuestos por diferentes piezas elementales comunes, que al juntarse en un patrón especifico forman un símbolo distinto. Además, al ver letras como la T o la Y en alfabeto español, vemos que son símbolos bastante simples, compuestos de 2 o 3 líneas. El promedio de líneas de un carácter en chino es de 16.03 trazos [19]. Veamos unos ejemplos de carácteres en chino:

故天将降大任于是人也，必先苦
其心志，劳其筋骨，饿其体肤，
空乏其身，行拂乱其所为，所以
动心忍性，曾益其所不能。

Figura 29: Página de texto en chino simplificado

A simple vista, los carácteres parecen bastante similiares, de hecho para los algoritmos también lo son. Esto pues, están compuestos de trazos en común, por lo que si un algoritmo detecta un trazo estará equivocándose al relacionarlo con cierto símbolo, pues son muchos otros símbolos que también contienen este trazo. Debido a esto, técnicas como *Image Segmentation* [20], que consisten en encontrar la información importante dentro de la imagen a través del reconocimiento de ciertas áreas, serán de mucha ayuda. Pues, para el reconocimiento de símbolos chinos será importante analizar el símbolo completo. Algortimos de segmentación de imagen útiles pueden ser redes neuronales como MobileNetV2 de Google [21], que funcionan a través de diferentes capas de cálculos que eliminar la información no importante y agregan la que sí lo es.

Si analizamos el rendimiento que tendrían los algoritmos utilzados en este trabajo según la experiencia en la literatura podemos observar que el resultado dependerá de múltiples factores. Algoritmos basados en probabilidades, por ejemplo *Gaussian Naive Bayes*, han demostrado ser bastante buenos a la hora de clasificar símbolos en chino, sin embargo requieren un gran dataset de entrenamiento para alcanzar un buen resultado. Los algoritmos que mejores resultados están dando en esta tarea son aquellos con estructura de redes neuronales. En este trabajo se utilizó la red neuronal *Multilayer Perceptron* que viene ya implementada en *Sckit-Learn*. Al venir esta ya preconfigurada, no nos garantiza un buen desempeño, esto pues las redes neuronales que están teniendo mayor éxito en el reconocimiento de símbolos en chino son por lo general adaptadas a estas tareas [22]. En librerías como *Tensorflow*, se puede acceder a diferentes funciones para diseñar redes neuronales capa por capa, de tal forma que se pueda configurar para desempeñar la tarea que se requiere resolver.

5. Conclusiones

Los reductores de dimensión, si bien son muy útiles para visualizar bases de datos y darles un acercamiento entendible, pierden sentido cuando hay muchas clases dentro de los datos, por ejemplo en *EMNIST* que hay 62 clases o símbolos diferentes, lo que significan 62 colores diferentes en el gráfico y termina siendo extremadamente difícil de entender. Además, es importante mencionar que esta base de datos era muy grande, por lo que se demoraba mucho en reducir las dimensiones. A partir de lo anterior también se puede concluir que para bases de datos muy grandes (mayores a 80000 datos), algunos reductores como *Isomap*, *t-SNE* y *MDS* demorarán mucho y no serán útiles para la finalidad que se busca, al menos de manera eficiente. Para optimizar, esto habría que buscar una manera de agilizar el cómputo de los reductores, tratando de reducir la complejidad del algoritmo.

En la clasificación de datos es fundamental que, para un buen funcionamiento del clasificador, la base de datos esté balanceada, dado que si hay un desbalance en la base de datos algunas clases se ignoran y otras se sobrerrepresentan. La sobrerrepresentación se refiere a que el algoritmo predice las clases mayoritarias para maximizar la accuracy, ya que estadísticamente hay más probabilidad de acertar.

Para balancear la base de datos y corregir este problema hay diversos métodos. El implementado en este caso fue el de *subsampling*, el cual toma solo una parte de cada clase hasta que todas tengan la misma cantidad. Sin embargo, al tener menos datos para el entrenamiento del método el accuracy baja. De todas formas se confirma que es mejor perder un poco de porcentaje de aciertos si es que el método funcionará de forma más integral. Por ejemplo, con el método Decision Tree el porcentaje de acierto con entrenamiento desbalanceado frente a un Test balanceado es 52 %, mientras que con entrenamiento balanceado y el mismo Test da un porcentaje de 48 %. Esto quiere decir que se perdió solo un 4 %, pero se ganaron más aciertos en las clases que solían estar subrepresentadas.

Por otro lado, respecto a los métodos de reducción de dimensionalidad, concluimos que el que mejor funcionó fue MDS debido a que es el que permite diferenciar grupos de manera más simple y es representación fiel de la información original. Además, los tiempos de ejecución de cada método no entregan información sobre qué tan bien funcionarán, ya que algunos que tienen rendimiento similar se demoran cantidades muy distintas de tiempos.

Respecto a la base de datos que fue creada nosotros mismos para probarla con los métodos, nos dimos cuenta de que el grosor de las líneas de los símbolos en la base de testeo afecta a la accuracy del método. Pero algo que afecta más es si la letra es manuscrita o imprenta, esto porque la base de datos utilizada para el entrenamiento solo contiene letras escritas a mano, entonces los métodos están más capacitados para reconocerlas frente a las imprentas.

Las letras que mejor reconocimiento tuvieron en promedio fueron las de Vicente y las de Javier, mientras que las más difíciles de reconocer fueron las de Sebastián y Felipe. Esto muy probablemente se explica por las razones anteriormente mencionadas, como el grosor de las líneas.

Referencias

[1] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, p. 2278–2324, 1998.

[3] G. Cohen, S. Afshar, J. Tapson, and A. V. Schaik, "Emnist: Extending mnist to handwritten letters," *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.

[4] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[5] L. van der Maaten and G. Hinton, "Viualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 11 2008.

[6] P. C. Hansen, "The truncatedsvd as a method for regularization," *Bit*, vol. 27, no. 4, p. 534–553, 1987.

[7] J. de Leeuw and W. Heiser, "13 theory of multidimensional scaling," *Handbook of statistics*, vol. 2, pp. 285–316, 1982.

[8] J. B. Kruskal, *Multidimensional scaling.* Sage, 1978, no. 11.

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

[10] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, p. 2319–2323, 2000. [Online]. Available: http://web.mit.edu/cocosci/Papers/sci_reprint.pdf

[11] L. Rokach and O. Maimon, "Decision trees," *The Data Mining and Knowledge Discovery Handbook*, vol. 6, pp. 165–192, 01 2005.

[12] S. L. Developers, "1.10. decision trees." [Online]. Available: https://scikit-learn.org/stable/modules/tree.html

[13] T. Yiu, "Understanding random forest," Aug 2019. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[14] S. L. Developers, "1.9. naive bayes." [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html

[15] S. Sekhar, "Math behind logistic regression algorithm," Aug 2019. [Online]. Available: https://medium.com/analytics-vidhya/logistic-regression-b35d2801a29c

[16] S. L. Developers, "1.17. neural network models (supervised)." [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.htmlmathematical-formulation

[17] A. M. Learning, "Clasificación con datos desbalanceados," Mar 2020. [Online]. Available: https://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/

[18] Lionbridge, "15 best ocr handwriting datasets for machine learning," Jan 2020. [Online]. Available: https://lionbridge.ai/datasets/15-best-ocr-handwriting-datasets/

[19] S. N. Srihari, X. Yang, and G. R. Ball, "Offline chinese handwriting recognition: an assessment of current technology," *Frontiers of Computer Science in China*, vol. 1, no. 2, p. 137–155, 2007.

[20] Y. Song and H. Yan, "Image segmentation techniques overview," *2017 Asia Modelling Symposium (AMS)*, 2017.

[21] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[22] C.-L. Liu, S. Jaeger, and M. Nakagawa, "Online recognition of chinese characters: the state-of-the-art," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, p. 198–213, 2004.