SplitGen: algoritmo evolutivo para división de datos en conjuntos pequeños

Lucila Chiarvetto Peralta[1][0009−0008−1347−8149] and Nélida Beatriz Brignole[2,3][0000−0002−4795−2872]

[1] Universidad Nacional de Tierra del Fuego (UNTDF)
Fuegia Basket 251, Ushuaia, Tierra del Fuego
lchiarvetto@untdf.edu.ar
https://untdf.edu.ar
[2] Laboratorio de Investigación y Desarrollo en Computación Científica (LIDeCC),
Universidad Nacional del Sur (DCIC-UNS),
Avda. Alem 1253, Bahía Blanca, Argentina
[3] Planta Piloto de Ing. Química (PLAPIQUI), CONICET-UNS,
Cno La Carrindanga Km 7, Bahía Blanca, Argentina.
dybrigno@criba.edu.ar

Resumen La decisión comúnmente adoptada en escenarios con abundantes datos, es la división aleatoria de los mismos. Sin embargo, cuando los datos escasean, esta decisión puede no ser la más apropiada. Se introduce SplitGen que es un algoritmo evolutivo diseñado para optimizar la división del conjunto de datos basado en un criterio de distancias. Su enfoque busca garantizar que los subconjuntos sean los más representativos posibles del conjunto original, mitigando el riesgo de introducir sesgos en la evaluación del modelo. La implementación se realizó en Python utilizando la librería DEAP. Se comparó su desempeño contra una búsqueda aleatoria, evaluando las soluciones mediante la pseudo distancia de Mahalanobis y la distancia de Wasserstein. Se observó un mejor desempeño del algoritmo genético, especialmente cuando el tamaño del conjunto de datos está entre 1000 y 10000 observaciones, y en distribuciones uniformes comparadas con las normales, sugiriendo una mayor dificultad en presencia de curtosis y valores atípicos.

Keywords: Bases de datos pequeñas · Algoritmos evolutivos · Distancia de Mahalanobis · Distancia de Wasserstein · Algoritmos genéticos

1. Introducción

El aprendizaje automático (ML, del inglés: *Machine Learning*) es un subcampo de la Inteligencia Artificial que estudia la capacidad de mejorar el rendimiento en función de la experiencia: un programa de computadora analiza datos, desarrolla un modelo a partir de estos y se emplea dicho modelo tanto como una teoría del mundo o como un programa capaz de solucionar problemas [1]. En el desarrollo de estos modelos, en situaciones donde los datos abundan, el mejor enfoque es dividir los datos de forma aleatoria en tres partes: entrenamiento, validación y testeo [2,3].

Los modelos en el aprendizaje automatizado se construyen sobre datos, que se segmentan para diversas tareas a lo largo del desarrollo del modelo (M). El conjunto de entrenamiento τ se usa para construir el M, y durante este proceso, el modelo pasa por diferentes estados que varían desde subajuste hasta sobreajuste. El conjunto de validación κ se utiliza para evaluar estos estados y seleccionar el modelo que muestra un ajuste óptimo, es decir, aquel donde el error de validación es mínimo. Este modelo se considera como el más adecuado en términos de su capacidad de generalizar a nuevos datos. Por último, el conjunto de testeo ψ ofrece una estimación imparcial del error de generalización del modelo seleccionado como el más adecuado.

El aprendizaje profundo (DL, del inglés: *Deep Learning*) tiene su origen en los primeros trabajos que intentaron emular algunos aspectos de la biología cerebral. Por esta razón, las redes entrenadas por métodos de DL a menudo se denominan redes neuronales (NN, del inglés: Neural Networks), aunque el parecido con las células y estructuras biológicas sea superficial. Grandes volúmenes de datos son necesarios para utilizar NN [4]. En consecuencia, si se trata de mejorar el desempeño de un sistema que tiene embebido un componente de DL [5], puede seguirse distintos enfoques. En particular, mejorar la inicialización del modelo, garantizando que en etapas tempranas el gradiente tenga propiedades benéficas o construyendo grandes espacios de escasez o raleza (del inglés: *sparsity*).

Las prácticas actuales para el desarrollo de una NN dependen fuertemente del supuesto de la disponibilidad de grandes volúmenes de datos. Sin embargo, en un contexto de bases de datos pequeñas, dividir los datos de esta forma no es recomendable. En primer lugar, debido a que las diferencias en las varianzas de los conjuntos de entrenamiento y testeo podrían tener un impacto en la evaluación de desempeño de los modelos.

Sea n la cantidad de observaciones de un conjunto de datos, cuando $n \to \infty$ es esperable que la $Var(y_\tau) \to Var(y)$ por la Ley de los Grandes Números. Pero cuando tenemos una cardinalidad limitada, la convergencia empírica puede no satisfacerse. Este argumento sugeriría la necesidad de considerar cómo es llevada a cabo la división de los datos en un contexto de bases de datos pequeña con el objetivo de no introducir un sesgo de selección. Por lo que realizarla al azar podría tener sus riesgos, lo que sugiere la necesidad de un instrumento que permita particionar los conjuntos de datos.

Se propone encontrar una partición del conjunto de datos que minimice las diferencias de las varianzas entre los subconjuntos: esto es un problema de opti-

mización combinatoria que tiene una complejidad NP-Completo. Esta categoría de complejidad puede ser resuelta por una máquina de Turing no determinista en tiempo polinomial [6]. Tal clase de problemas contiene numerosas instancias de búsqueda y optimización, para los que se desea saber si hay una cierta solución o si existe una mejor solución que las conocidas.

Los algoritmos evolutivos se basan en la Teoría de la Evolución Natural y buscan soluciones a través de la iteración aleatoria de un conjunto de estructuras. Se parte de una población de individuos en un entorno con recursos limitados, donde la competencia por esos recursos lleva a una selección natural (supervivencia del más apto), lo que incrementa la aptitud de la población.

Dada una función a minimizar o maximizar, podemos crear aleatoriamente un conjunto de soluciones candidatas, es decir, elementos del dominio de la función. Entonces, es posible aplicar una función de calidad a estos como una medida abstracta de aptitud. En el caso de que sea un problema de minimización cuanto menor sea este valor, será mejor la solución. Sobre la base de estos valores de aptitud, algunos de los mejores candidatos serán elegidos para recrear la próxima generación.

La generación siguiente se forma mediante la aplicación de operadores genéticos a los individuos actuales. Algunos de estos operadores son la cruza y la mutación. La cruza o recombinación es un operador que se aplica a dos o más candidatos seleccionados (los llamados padres), produciendo nuevos candidatos. En el caso de la mutación se aplica a un candidato y da como resultado un nuevo candidato con el que se incorpora nueva información. Por lo tanto, ejecutar estas operaciones conduce a la creación de un conjunto de nuevos candidatos: la descendencia.

Luego, estos nuevos individuos tendrán la oportunidad de competir según su aptitud (y posiblemente por su antigüedad) con los mayores por un lugar en la próxima generación. Este proceso de selección favorece a los mejores individuos. Este favoritismo puede determinarse dada su aptitud, calculada por una función específica (del inglés: *fitness*). Cada uno de estos ciclos de operación y selección se define como generación. Luego de un cierto número de generaciones se espera que el mejor individuo o campeón esté razonablemente próximo a la solución buscada.

En la sección siguiente se presenta un algoritmo genético que permite encontrar una división del conjunto de datos.

2. SplitGen: Un algoritmo genético para particionar el conjunto de datos

Siguiendo el criterio propuesto por [7], el nombre del algoritmo dependerá de forma que tomen los individuos, el algoritmo tendrá diferentes nombres. Se llamará algoritmo genético (GA: Genetic Algorithm) si los individuos tienen la forma de cadenas (secuencias de caracteres) sobre un alfabeto finito. En el alg. 1 se presenta el algoritmo genético propuesto para dividir un conjunto de datos en subconjuntos donde se toma como entrada el conjunto de datos y una partición

específica. La partición se representa como una lista de porcentajes que indican la proporción del tamaño de cada subconjunto. Esta partición es una secuencia de longitud arbitraria l, donde cada elemento e debe cumplir la condición $0 \leq e \leq 1$, y la suma total de la secuencia debe ser igual a uno ($\sum e_i = 1$). Un alelo es uno de los posibles valores que puede tener un gen. En el algoritmo propuesto, los alelos pueden tomar algún valor dentro del rango $[0, l-1]$ y cada posición de la lista representa un gen.

Dado que el conjunto de datos consta de observaciones m, el algoritmo propuesto codifica la división en una secuencia s de longitud m. Cada elemento de s toma un valor en el rango $[0, l-1]$, lo que determina a qué subconjunto pertenece cada observación del conjunto de datos, asegurando que se cumpla la cardinalidad indicada para cada subconjunto.

La población inicial se construye como una permutación aleatoria de la secuencia ordenada de índices que respeta la proporción indicada por l. Cada individuo está representado por una lista de longitud m. Para ilustrar el concepto supongamos un conjunto de datos de 10 observaciones será dividido en partes de 7 y 3 unidades, en la fig. 1 se puede observar una representación de la secuencia que se utilizará para construir toda la población. Cualquier permutación de esta secuencia será un individuo factible que representa una potencial solución. El concepto de fenotipo corresponde a una característica del individuo y el genotipo es una representación del fenotipo. En la fig. 2 se muestra representaciones de las características fenotípicas y genotípicas del ejemplo planteado.

El algoritmo incluye operadores de mutación y cruza. El operador elegido de cruce es el monopunto que escoge una posición aleatoria con una distribución uniforme para cada de ellas. De esta forma se une la parte anterior de un individuo con la parte posterior de otro.

Esta operación permite la creación de individuos que no satisfacen la proporción de división del conjunto de datos: permitiendo la creación de individuos no factibles. En la 3 se muestra un ejemplo correspondiente al problema planteado, dónde los padres A y B como resultado del punto de cruce elegido producen dos hijos A y B. Podría producirse un desbalance en la proporción de los conjuntos al momento de cruzarlos, lo que daría lugar a individuos que difieren de la cantidad de observaciones que debe tener cada conjunto. Estos individuos que no respetan el criterio planteado se definen como no factibles, por lo que deben ser reparados antes de evaluarlos.

Se debe reparar los individuos cuando no se mantienen las proporciones de las particiones definidas para una secuencia, es esencial para garantizar que la

Figura 1: Ejemplo de secuencia utilizada para construir la población.

Algoritmo 1 Esquema del algoritmo evolutivo propuesto

```
Datos de entrada: ConjuntoDeDatos, Particion, Distancia,
TamañoPoblacion (por defecto 30), CantGeneraciones (por
defecto 50), ProbabilidadCruce (por defecto 0.60)

Comienzo
    # Crea una población de soluciones aleatorias
    poblacion <- InicializarPoblacion(ConjuntoDeDatos, Particion)

    campeon <- Ninguno

    Para g en el rango(1,CantGeneraciones):
            # Selecciona un subconjunto de soluciones de
            # la población actual para reproducirse. Las
            # soluciones con mejores calificaciones tienen una mayor
            # probabilidad de ser seleccionadas.
            desendencia <- Clonar(poblacion)

            # Genera nuevas soluciones mediante la combinación de las
            # soluciones seleccionadas. Esto se puede hacer utilizando
            # operadores de cruce y mutación.
            para individuo_1, individuo_2 en desdendencia:
                Si numero aleatorio < ProbabilidadCruce:
                    cruzar(individuo_1, individuo_2)

            probMutacion = 1 - (0.95 * ( g/ CantGeneraciones)
            para individuo en desdendecia:
                Si numero aleatorio < probMutacion:
                    mutar(individuo)

            Descendencia <- Mejores individuos (descendencia)
            Si la aptitud del mejor individuo Población < a la
            aptitud del Campeón:
                # se ha encontrado un nuevo campeón
            Campeón <- Nueva mejor solución

            # Reemplazo total de la población con las nuevas
            # soluciones generadas
            Poblacion <- Desdendencia

    retornar el Campeón

Fin del algoritmo
```

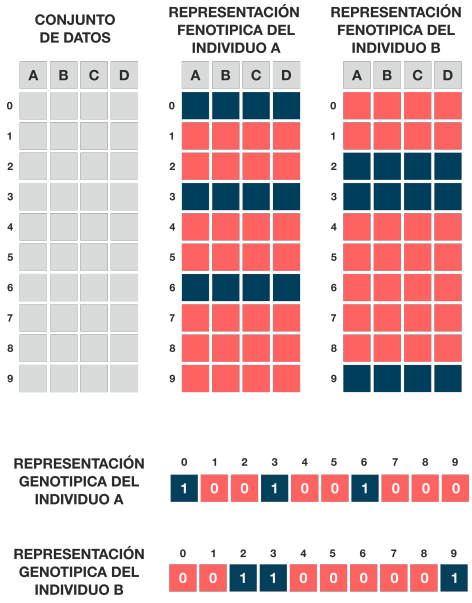


Figura 2: Representaciones fenotípica y genotípica de las características de los individuos.

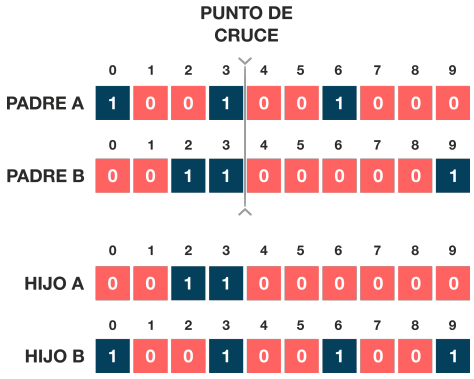


Figura 3: Cruce que produce individuos factibles

secuencia de índices resultante sea válida. Una vez que se detecta la necesidad, se identifica un índice de un conjunto con exceso de elementos y otro de un conjunto con déficit. Estos intercambios se llevan a cabo iterativamente hasta que se restaura el balance. Este proceso implica ajustar los elementos en las posiciones antes y después del punto de cruce, garantizando que la partición resultante cumpla con las restricciones de la partición especificada. Estos intercambios se realizan hasta que el balance es recuperado, garantizando una la secuencia válida y que cumpla con las restricciones de la partición especificada. En la fig. 4 se muestra como uno de los elementos es elegido para recuperar el balance.

Luego, es posible que el individuo sufra una mutación. En un algoritmo genético, la tasa de mutación juega un papel crucial para evitar el estancamiento en óptimos locales. En este enfoque, la probabilidad de mutación comienza de manera agresiva para permitir una exploración exhaustiva del espacio de búsqueda. Sin embargo, se reduce de forma adaptativa a medida que avanzan las generaciones. Este ajuste gradual permite que el algoritmo pase de una fase inicial de exploración intensiva a una etapa más enfocada en la explotación de las soluciones más prometedoras.

Si el individuo fue seleccionado para su mutación, el operador genético se resuelve mediante el intercambio del contenido de dos posiciones elegidas eventualmente. La probabilidad de mutar un gen es proporcional a la longitud del individuo, es de esperar no más de una mutación por cada uno de ellos.

El operador de selección utilizado se conoce como *tournament* o torneo. Aunque es posible que un torneo involucre a más de dos participantes, en el diseño actual se realiza con dos individuos específicamente: se compara y selecciona el mejor individuo entre los dos elegidos al azar para participar del torneo. Este proceso se repite la cantidad de veces necesaria hasta lograr una nueva población, reemplazando completamente la anterior. El individuo menos apto de cada generación perderá todos los torneos. Además, el algoritmo implementa elitismo, asegurando que el campeón, o mejor individuo, permanezca en la población en todo momento. Así, la mejor solución encontrada se mantiene de generación a generación.

Figura 4: Reparación de individuo no factible

El algoritmo cuenta con dos funciones de aptitud, la primera está basada en la distancia de Wassertein o de movimiento de tierra. Esta distancia se define como una función entre distribuciones de probabilidades en un espacio métrico M dado. Puede interpretarse como la cantidad mínima de "trabajo" requerido para transformar una pila de tierra en otra. "Trabajo" se mide como el peso de una distribución que debe moverse multiplicado por la distancia en la que tiene que ser desplazado.

Teóricamente, esta función se define sobre dos distribuciones u y v:

$$l_1(u, v) = \inf_{\pi \in \Gamma(u,v)} \sum_{(x_i, x_j)} |x_i - x_j| \cdot \pi(x_i, x_j) \tag{1}$$

donde:

- $l_1(u, v)$ representa la distancia de Wasserstein de orden 1 entre las distribuciones u y v.
- $\Gamma(u, v)$ es el conjunto de todas las medidas de acoplamiento (distribuciones conjuntas) que tienen u y v como marginales.
- La sumatoria se realiza sobre todos los pares i y j que representan la correspondencia entre los valores de las distribuciones u y v.
- x_i y x_j representan los valores discretos de las distribuciones u y v, respectivamente.
- $\pi(x_i, x_j)$ representa cuánta masa se transporta de x_i en u a x_j en v según la medida de acoplamiento π.

La restante función de aptitud es calculada con base en una definición de compromiso de la distancia de Mahalanobis propuesta por [**?**].

$$D_M(\mu_i, \mu_j) = (\mu_i - \mu_j)' \left[\frac{1}{2}(\Sigma_i + \Sigma_j)\right]^{-1} (\mu_i - \mu_j) \tag{2}$$

donde:

- $D_M(\mu_i, \mu_j)$ representa la distancia de Mahalanobis entre los vectores de medias μ_i y μ_i de las dos distribuciones u y v.
- μ_i y μ_j representan los vectores de medias de las dos distribuciones u y v.
- Σ_i, Σ_j representan las matrices de varianzas-covarianzas de las dos distribuciones dadas por:
  - $\Sigma_i = \frac{1}{n_i} (X_i - 1_{n_i} \cdot \mu_i)' (X_i - 1_{n_i} \cdot \mu_i)$
  - $\Sigma_j = \frac{1}{n_j} \left(X_j - 1_{n_j} \cdot \mu_j\right)' \left(X_j - 1_{n_j} \cdot \mu_j\right)$
  - n_i y n_j son el número de observaciones en las distribuciones i y j, respectivamente. Representan el tamaño de las muestras en cada conjunto de datos.
  - X_i y X_j son las matrices de datos para las distribuciones.
  - 1_{n_i} y 1_{n_i} son vectores columna de unos de dimensiones $n_i \times 1$ y $n_j \times 1$, respectivamente.

- $\left[\frac{1}{2}(\Sigma_i + \Sigma_j)\right]^{-1}$ es la inversa de la suma promediada de las matrices de varianza-covarianza.

En lugar de los parámetros poblacionales, en la implementación se emplearon los estadísticos muestrales. Para este cálculo, se tomaron en consideración las medias de cada uno de los atributos y la matriz de varianza-covarianza que relaciona los valores de la distribución conjunta de las variables del conjunto de observaciones.

Con base en estas funciones, la función de aptitud es calculada siguiendo:

$$\text{aptitud}(D_m, I) = \sum_{i=1}^{p(I)} \text{distancia}(\text{D}, \text{D}_{[partidion_i]}) \tag{3}$$

Dónde:

- D: es el conjunto de datos de longitud m
- I es una secuencia de longitud m que representa al individuo
- $p(I)$ se define como el $max(i_j)$ presente en I
- $\text{D}_{[particion_i]}$: se define como el subconjunto de datos indicados por el individuo que pertenecen a la partición i
- distancia: función de distancia definida sobre dos conjuntos de datos. En la implementación realizada esta función puede ser la ec. 1 ó ec. 2

3. Resultados

El algoritmo fue implementado en el lenguaje Python usando la librería DEAP (acrónimo de *Distributed Evolutionary Algorithms in Python*) en su versión v.1.0. Para evaluar el desempeño del algoritmo se diseñó un conjunto de experimentos comparándolo con una búsqueda aleatoria.

La estrategia de la búsqueda aleatoria (RS: *Random Search*) se basa en la creación de candidatos potenciales al barajar de manera aleatoria una secuencia de índices predefinida. Cada candidato se evalúa con la misma función de aptitud que la utilizada en su contraparte, ya sea con la pseudo distancia de Mahalanobis o la distancia de Wasserstein. La RS se repite múltiples veces, y en cada iteración, se compara la aptitud del candidato actual con la mejor aptitud previamente registrada. Si el candidato actual supera a la mejor solución encontrada hasta el momento, se actualiza el *campeón*. Al final del proceso de búsqueda, se almacena la secuencia del *campeón*, junto con su aptitud, que representa la solución óptima encontrada. En el alg. 2 se muestra el esquema del algoritmo utilizo para la comparación.

Se llevó a cabo una evaluación utilizando dos procesos generadores de datos (DGP, del inglés: *Data Generating Process*) distintos, cada una con características específicas de curtosis. El primer DGP se basa en la generación de números aleatorios con una distribución uniforme, mientras que la segunda emplea una distribución normal. En este último caso, tanto la media como la varianza se

Algoritmo 2 Esquema del algoritmo de búsqueda aleatoria

```
Datos de entrada: ConjuntoDeDatos, Particion, Distancia, CantRepeticiones

Datos de salida: Campeón

Comienzo

    # Crea una población de soluciones aleatorias
    poblacion <- InicializarPoblacion(ConjuntoDeDatos, Particion)

    Campeón <- Ninguno
    aptitud_campeon <- +Infinito

    Para g en el rango(1,CantGeneraciones):

        #Genera un nuevo estado aleatorio
        candidato <- generar_particion_aleatoria(ConjuntoDatos, Particion)

        # Evalúa el nuevo estado generado
        Si(Distancia= "Mahalanobis")
            entonces aptitud_candidato <- aptitudMahalanobis(Candidate,
                                          ConjuntoDatos)
            sino aptitud_candidato <- aptitudWassertein(Candidate,
                                          ConjuntoDatos)

        #Si el nuevo candidato es mejor que el campeón actual,
        #actualiza con la nueva partición generada

        Si(aptitud_candidato < aptitud_campeon):
            entonces aptitud_campeon <- aptitud_candidato
                    Campeón <- candidato

    retornar Campeón

Fin del algoritmo
```

Tabla 1: Partición en k partes realizada con la pseudo distancia de Mahalanobis

Cantidad de observaciones	Uniforme (DC %)	Normal (DC %)
100	56.04 % (+/- 9.48 %)	56.90 % (+/- 10.44 %)
1000	68.27 % (+/- 13.78 %)	70.08 % (+/-14.20 %)
10000	51.01 % (+/- 24.89 %)	48.37 % (+/- 24.68 %)
100000	31.48 % (+/- 31.39 %)	28.66 % (+/- 24.53 %)

Tabla 2: Partición en k partes realizada con la distancia de Wassertein

Cantidad de observaciones	Uniforme (DC %)	Normal (DC %)
100	25.43 % (+/- 5.67 %)	20.95 % (+/- 5.57 %)
1000	28.20 % (+/-6.45 %)	23.47 % (+/- 5.41 %)
10000	17.78 % (+/-8.38 %)	15.92 % (+/- 4.32 %)
100000	11.22 % (+/-7.80 %)	9.62 % (+/- 5.99 %)

generan de manera aleatoria para cada columna del conjunto de datos, siendo estas columnas los atributos evaluados.

Para la evaluación de la propuesta se define la diferencia de campeones porcentual (DC (%)) con el objetivo de ofrecer una medida cuantitativa de la eficacia de GA con RS. Dado que ambos métodos son de naturaleza estocástica, es relevante tener un punto de referencia para evaluar el rendimiento del algoritmo genético. DC (%) permite determinar si el método genético encuentra soluciones significativamente mejores que un método aleatorio, y en qué magnitud. El algoritmo genético fue evaluado con el valor de sus hiperparámetros por defecto.

$$\text{DC } (\%) = \frac{\text{Aptitud del campeón RS} - \text{Aptitud del campeón GA}}{\text{Aptitud del campeón GA}} \times 100 \quad (4)$$

Si DC (%) toma un valor cercano a 0, entonces ambos métodos tienen un rendimiento similar en términos de función de aptitud, sugiriendo que el algoritmo genético no ofrece una ventaja significativa. Pero si esta métrica es positiva, implica que la RS ha encontrado una solución con mayor de función de aptitud lo cual supone una peor solución. Por último, si la DC (%) es negativa, indica que la RS ha encontrado una mejor solución que el GA.

Para cada tamaño de conjunto de datos y distribución se generaron 36 conjuntos. En las Tab. 1 y Tab. 2 se presentan los resultados para las distancias de Mahalanobis y Wassertein respectivamente, para la división de datos que se haría para una validación cruzada en k partes tomando valores del conjunto 3,5,7,9. En las Fig. 5a y Fig. 5b se muestran gráficos de barras de las tablas reportadas.

En el caso de la Tab. 1 se observan los resultados de la evaluación de las mejores soluciones reportadas utilizando la función de compromiso de distancia de Mahalanobis. Mientras que la Tab. 2 se observan los resultados de la evaluación

Tabla 3: Partición en tres partes realizada con la pseudo distancia de Mahalanobis

Cantidad de observaciones	Uniforme (DC %)	Normal (DC %)
100	59,78 % (+/- 8 ,55 %)	60,17 % (+/- 12,72 %)
1.000	84,74 % (+/- 4,39 %)	80,24 % (+/- 3,30 %)
10.000	70,16 % (+/- 19,27 %)	82,31 % (+/- 9,40 %)
100.000	53,77 % (+/- 28,55 %)	62,16 % (+/- 23,62 %)

Tabla 4: Partición en tres partes realizada con la distancia de Wassertein

Cantidad de observaciones	Uniforme (DC %)	Normal (DC %)
100	26,41 % (+/- 2,60 %)	21,96 % (+/- 3,20 %)
1.000	32,58 % (+/- 4,61 %)	26,89 % (+/- 3,60 %)
10.000	21,89 % (+/- 6,31 %)	16,09 % (+/- 8,47 %)
100.000	15,70 % (+/- 7,09 %)	12,09 % (+/- 7,64 %)

de las mejores soluciones reportadas utilizando la distancia de Wassertein. Esto se evidencia claramente en las Fig. 5a y Fig. 5b.

En términos generales, se observa un mejor desempeño de los datos uniformemente distribuidos en comparación con los normales, lo que sugiere que la curtosis presente en el conjunto de datos supone alguna dificultad en resolver el problema: la presencia de valores atípicos supone un obstáculo. A su vez, en ambos casos se observa que entre 100 y 1000 observaciones parece encontrarse el punto de máximo beneficio de la utilización del método de división de los datos propuesto.

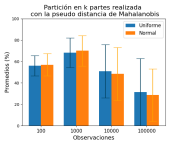
En las Tab. 3 y Tab. 4 se presentan los resultados para las distancia de Mahalanobis y Wassertein respectivamente, para la división de datos que se haría para una validación cruzada (en tres partes y proporciones de 70 %, 15 % y 15 % respectivamente). También muestra que el punto de mayor beneficio se encuentra entre las 1000 y 10000 observaciones para ambas distancias.

En la Tab. 3 se observan los resultados de la evaluación de las mejores soluciones reportadas utilizando la función de compromiso de distancia de Mahalanobis. Por otro lado, en la Tab. 4 se observan los resultados de la evaluación la aptitud de las mejores soluciones reportadas utilizando la distancia de Wassertein.

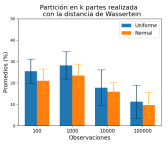
La distancia de Mahalanobis parece tener un mejor desempeño cuando la división de datos es para validación cruzada, tanto en la distribución normal como uniforme. Esto parece mostrar cierta resistencia a los valores atípicos. Es decir, que de utilizar validación cruzada se recomienda esta distancia.

La distancia de Wassertein tiene un mejor desempeño cuando la distribución de los datos es uniforme. Esto sugiere que en presencia de valores atípicos tendrá un desempeño menor que el esperado, independientemente del mecanismo de división que se emplee.

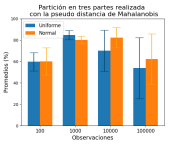
Figura 5: Promedios



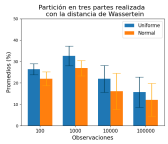
(a) Promedios de las distribuciones uniforme y normal según diferentes cantidades de observaciones, basado en la Tab. 1



(b) Promedios de las distribuciones uniforme y normal según diferentes cantidades de observaciones, basado en la Tab. 2



(c) Promedios de las distribuciones uniforme y normal según diferentes cantidades de observaciones, basado en la Tab. 3



(d) Promedios de las distribuciones uniforme y normal según diferentes cantidades de observaciones, basado en la Tab. 4

4. Conclusiones y trabajos futuros

Futuras investigaciones evaluarán la transformación de variables categóricas y su impacto en la generalización de modelos. Se explorará la adecuación de distintas métricas, incluyendo la distancia de Hamming y el coeficiente de similaridad de Gower para datos mixtos.

Se propone rediseñar el algoritmo para el ajuste autónomo de los hiperparámetros para mejorar la adaptación a variados problemas estadísticos, examinando en particular cómo la asimetría y otras características estadísticas afectan los resultados.

Se observaron diferencias significativas en la distribución de los datos cuando la división es aleatoria versus controlada, analizadas mediante las distancias de Mahalanobis y Wasserstein. Estas métricas reflejaron variaciones según la curtosis de los datos, sugiriendo cautela al generalizar sus impactos en la construcción de modelos de ML. Se identificó el rango óptimo de observaciones entre 10^3 y 10^4 para maximizar los beneficios de la división controlada de datos.

En este estudio solamente hemos considerado operadores de cruzamiento cuya aplicabilidad es general. Sería interesante comparar su desempeño con operadores diseñados especialmente para permutaciones.

Por otro lado, se explorarán otras operaciones de cruce que no requieran un procedimiento de reparación, ya que este supone desafíos significativos. La necesidad de intercambiar elementos para mantener las proporciones adecuadas puede introducir complejidad adicional y afectar la eficiencia del algoritmo. Además, se estudiarán los efectos de estos intercambios en el rendimiento del modelo y en la calidad de las particiones resultantes. Al investigar métodos de cruce alternativos, se busca simplificar el proceso y mejorar la robustez del modelo, garantizando al mismo tiempo que las particiones resultantes cumplan con las restricciones especificadas de manera más directa y eficiente.

Una tarea de regresión asigna un valor numérico a una entrada numérica. El DL en datos tabulares enfrenta desafíos, especialmente por la falta de evalua-

ciones comparativas estandarizadas (benchmarks) para problemas de regresión. A diferencia de la clasificación en aprendizaje de pocas oportunidades, donde ya existen benchmarks [9], esto no se ha abordado tanto en la regresión con datos tabulares. En [10], se sugiere un conjunto de datos limitado: de tamaño medio (más de 3000 observaciones) con atributos numéricos y heterogéneos. Por ello, futuros trabajos propondrán diseñar una batería de pruebas y evaluar el algoritmo sobre estas.

Referencias

1. P. Russell, Stuart y Norvig, Artificial Intelligence: A Modern Approach, 4ta Ed. Pearson, 2010.
2. I. Goodfellow, Y. Bengio, y A. Courville, Deep Learning, ser. Serie de Computación Adaptativa y Aprendizaje Automático. MIT Press, 2016, [En línea] Disponible en `http://www.deeplearningbook.org`.
3. T. Hastie, R. Tibshirani, y J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, ser. Springer Series in Statistics. Springer New York, 2013. [En línea] Disponible en `https://books.google.com.ar/books?id=yPfZBwAAQBAJ`
4. T. Brown et al., "Language models are few-shot learners," in Advances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [En línea] Disponible en `https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`
5. T. Dozat, "Incorporating nesterov momentum into adam," en Proceedings of the 33nd International Conference on Machine Learning (ICML), 2016, pp. 1480–1489.
6. S. Arora and B. Barak, Computational Complexity: A Modern Approach. Cambridge University Press, 2009. [En línea] Disponible en `https://books.google.com.ar/books?id=8Wjqvsoo48MC`
7. A. E. Eiben y J. E. Smith, Introduction to Evolutionary Computing, 2nd ed., ser. Natural Computing Series. Berlin: Springer Publishing Company, Incorporated, 2015, vol. 53.
8. C. M. Cuadras, Nuevos métodos de análisis multivariante. CMC Edicions Barcelona, Spain, 2019.
9. T. Hospedales, A. Antoniou, P. Micaelli, y A. Storkey, "Meta-learning in neural networks: A survey," IEEE transactions on pattern analysis and machine intelligence, vol. 44, no. 9, pp. 5149–5169, 2021.
10. L. Grinsztajn, E. Oyallon, y G. Varoquaux, "Why do tree-based models still outperform deep learning on tabular data?" 2022. [En línea] Disponible en `https://arxiv.org/abs/2207.08815`