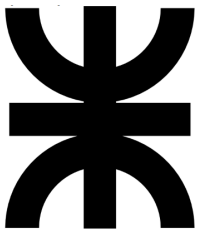

Universidad Tecnológica Nacional

Tesis de Maestría

Generación de datos sintéticos para selección de características con algoritmos genéticos

Autor:
Claudio Sebastian Castillo

Directores:
Matías Gerard y Leandro Vignolo

Tesis presentada en cumplimiento de los requisitos para el grado de Maestría en Minería de Datos

en

UTN
Seccional Paraná

Agosto, 2024

A mi corazón de 4/4; Morella, Sofía, Joquín y Manuel. A mi musa Verónica.

UNIVERSIDAD TECNOLÓGICA NACIONAL

Seccional Paraná

Maestría en Minería de Datos

Resumen

Generación de datos sintéticos para selección de características con algoritmos genéticos

Claudio Sebastian Castillo

La disponibilidad de datos muestrales afecta los procesos de selección de características, y resulta particularmente condicionante en escenarios de alta dimensionalidad y bajo número de muestras. En el caso de selección de características mediante Algoritmos Genéticos la falta de datos impacta negativamente en la función de aptitud, y de esa forma limita la eficacia del algoritmo. Por eso, la técnica de aumentación de datos mediante Autocodificadores Variacionales plantea una posible solución a este problema, ofreciendo distintas alternativas de implementación en el contexto de los Algoritmos Genéticos.

Introducción

En el campo del aprendizaje automático, la selección de características es una tarea crítica que puede determinar el éxito o fracaso de un modelo predictivo. La alta dimensionalidad y la complejidad inherente a los conjuntos de datos reales hacen que la selección de un subconjunto óptimo de características sea, muchas veces, un paso ineludible para el aprendizaje efectivo.

En este contexto, los Algoritmos Genéticos (AGs) se han consolidado como una herramienta poderosa para resolver problemas de optimización complejos, incluida la selección de características. Estos algoritmos, inspirados en la evolución natural, son capaces de explorar grandes espacios de búsqueda de manera efectiva, proporcionando soluciones cercanas al óptimo en una variedad de escenarios. Por ello, los AGs han sido ampliamente utilizados en problemas de selección, demostrando su eficacia en la identificación de subconjuntos relevantes de características en datos de alta dimensionalidad.

Sin embargo, la eficacia de los AGs depende de la disponibilidad de suficientes datos para evaluar las soluciones en competencia. En contextos donde los datos son limitados, los AGs pueden verse afectados en su capacidad discriminativa, produciendo soluciones subóptimas o inestables. Esta limitación es especialmente crítica en problemas de alta dimensionalidad y bajo número de muestras, donde la función objetivo que guía la búsqueda de soluciones puede degradarse significativamente.

Por esta razón, la investigación de estrategias que mitiguen las limitaciones impuestas por la escasez de datos se ha convertido en un área de interés creciente en el subcampo de la selección de características. Una de las técnicas emergentes en este ámbito es la aumentación de datos mediante Autoencodificadores Variacionales (AVs). Los AVs, como modelos generativos, tienen la capacidad de crear muestras sintéticas que mantienen las propiedades fundamentales de los datos originales, conviertiéndolos en una herramienta prometedora para mejorar la capacidad de los AGs en la selección de características.

El problema central de la tesis que aquí presentamos giró, precísamente, en la restricciones que la escacez de datos impone a los AGs, y cómo superarlas usando AVs. La pregunta que guió nuestro trabajo ha sido: ¿cómo puede la aumentación de datos mediante autoencodificadores variacionales mejorar el desempeño de los algoritmos genéticos en la selección de características?

Cabe destacar que este desafío y su eventual resolución son importantes por varias razones. La selección de características no solo condiciona la precisión de los modelos predictivos, sino también afecta la eficiencia computacional y la interpretabilidad de los resultados. En problemas de alta dimensionalidad, la capacidad de reducir el número de características relevantes sin perder información útil puede marcar la diferencia entre un modelo efectivo y uno ineficaz, entre uno interpretable y uno de caja negra. Por lo tanto, mejorar este proceso mediante la integración de técnicas de aumentación de datos puede tener un impacto significativo en diversas aplicaciones prácticas, desde la biología molecular hasta la ingeniería y las ciencias sociales.

La hipótesis que hemos llevado a prueba ha sido *que la aumentación de datos mediante AVs mejora la capacidad discriminativa de los AGs, permitiendo la identificación de subconjuntos de características más relevantes y estables en contextos de escasez muestral*. Para evaluarla, hemos propuesto un trabajo experimental que exploró la

integración de estas dos técnicas en un marco unificado. La idea de combinar la generación de datos sintéticos mediante AVs con la selección de características mediante AGs, estaba orientada a buscar combinaciones sinérgicas y eficaces entre modelos que mejorasen la selección de características. A estos fines, trabajamos con cuatro conjuntos de datos de referencia, representativos de distintos contextos y niveles de complejidad, para evaluar el desempeño de los modelos propuestos.

A lo largo de este documento, describiremos el proceso de investigación llevado a cabo, desde los estudios iniciales hasta los experimentos finales, pasando por el diseño y construcción de un modelo genérico de AV, y su adaptación a los datasets elegidos y la creación de una estructura combinada de AV + AG para la selección de características. Los resultados obtenidos en cada etapa se analizarán y discutirán en detalle, con el objetivo de identificar las ventajas y limitaciones de la propuesta, así como posibles áreas de mejora y futuros trabajos.

Al finalizar el documento, esperamos poder justificar la eficacia de la aumentación de datos mediante AVs en la selección de características, demostrando que esta técnica puede mejorar significativamente el desempeño de los AGs en contextos de escasez. Además, esperamos identificar las condiciones y contextos en los que esta técnica es más efectiva.

El documento está estructurado de la siguiente manera: en el capítulo 2 se presenta una revisión del estado del arte, destacando las principales contribuciones en el ámbito de la aumentación de datos y la selección de características, así como los resultados de modelos clásicos aplicados a los datasets seleccionados para este trabajo. El capítulo 3 describe el diseño y construcción de nuestro modelo de Autoencodificador Variacional, desde una revisión teórica hasta la implementación particular que hemos desarrollado. En el capítulo 4 se ofrece una breve descripción de los Algoritmos Genéticos, resaltando los aspectos más relevantes de esta técnica aplicada a la selección de características. En el capítulo 5 se presentan y analizan los resultados obtenidos a partir de los experimentos realizados, incluyendo la evaluación de los modelos propuestos. En el capítulo 6, se exponen las conclusiones de la investigación, así como posibles líneas futuras de trabajo.

Finalmente, en tiempos donde arrecian los debates sobre los riesgos de los modelos generativos, esperamos poder brindar a nuestros lectores un ejemplo funcional de su estructura, iluminar ciertos principios y evidenciar sus límites. Si nuestro trabajo permite resaltar la sobriedad de sus mecanismos internos, habremos complido con nuestro anhelo de hacer menos opaca su belleza y más clara sus posibilidades.

Índice

Lista de figuras

Lista de tablas

Capítulo 1

Marco específico de trabajo

En este capítulo revisaremos el desempeño de algoritmos clásicos en la solución de los dataset elegidos para nuestra investigación. Describiremos brevemente la composición de los conjuntos de datos, los algoritmos seleccionados para su tratamiento y los resultados obtenidos para cada uno. Luego, analizando y comparando dichos resultados, elegiremos un modelo para emplear como función de fitness en nuestro Algoritmo Genético. Finalmente, haremos un breve repaso de la aplicaciones de Autocodificaciones Variacionales al problema de la aumentación de datos, como antesala al capítulo siguiente.

El propósito de esta etapa del trabajo es doble. Por un lado, identificar los modelos más apropiados para servir de *función de fitness* en la implementación de nuestro algoritmo genético. Esto permitirá construir una implementación robusta, que cuenta con una función efectiva y computacionalmente conveniente para evaluar cada solución. Al mismo tiempo, disponer de métricas acerca del desempeño que logran distintas estrategias de clasificación, a partir de las cuales comparar el resultado de nuestras propias soluciones. Por ultimo, revisar ejemplos de aplicación de los AVs en escenarios de aumentación de datos para preparar nuestro trabajo del capítulo siguiente.

1.1. Datos elegidos en nuestro estudio

El conjunto de datos elegidos en este trabajo incluye:

1. *Madelon*: conjunto artificial de datos con 500 características, donde el objetivo es un XOR multidimensional con 5 características relevantes y 15 características resultantes de combinaciones lineales de aquellas (i.e. 20 características redundantes). Las otras 480 características fueron generadas aleatoreamente (no tienen poder predictivo). Madelon es un problema de clasificación de dos clases con variables de entrada binarias dispersas. Las dos clases están equilibradas, y los datos se dividen en conjuntos de entrenamiento y prueba. Fue creado para el desafío de Selección de Características NIPS_2003, y está disponible en el Repositorio UCI. Los datos están divididos en un conjunto de entrenamiento y un conjunto de testeo.

2. *Gisette:* es un dataset creado para trabajar el problema de reconocimiento de dígitos escritos a mano (Isabelle Guyon 2004). Este conjunto de datos forma parte de los cinco conjuntos utilizados en el desafío de selección de características de NIPS 2003. Tiene 13500 observaciones y 5000 atributos. El desafío radica en diferenciar los dígitos ‘4’ y ‘9’, que suelen ser fácilmente confundibles entre sí. Los dígitos han sido normalizados en tamaño y centrados en una imagen

fija de 28x28 píxeles. Además, se crearon características de orden superior como productos de estos píxeles para sumergir el problema en un espacio de características de mayor dimensión. También se añadieron características distractoras denominadas "sondas", que no tienen poder predictivo. El orden de las características y patrones fue aleatorizado. Los datos están divididos en un conjunto de entrenamiento y un conjunto de testeo.

3. *Leukemia*: El análisis de datos de expresión génica obtenidos de micro-datos de ADN se estudia en Golub (1999) para la clasificación de tipos de cáncer. Construyeron un conjunto de datos con 7129 mediciones de expresión génica en las clases ALL (leucemia linfocítica aguda) y AML (leucemia mielogénica aguda). El problema es distinguir entre estas dos variantes de leucemia (ALL y AML). Los datos se dividen originalmente en dos subconjuntos: un conjunto de entrenamiento y un conjunto de testeo.

4. *GCM*: El conjunto de datos GCM fue compilado en Ramaswamy (2001) y contiene los perfiles de expresión de 198 muestras de tumores que representan 14 clases comunes de cáncer humano. Aquí el enfoque estuvo en 190 muestras de tumores después de excluir 8 muestras de metástasis. Finalmente, cada matriz se estandarizó a una media de 0 y una varianza de 1. El conjunto de datos consta de un total de 190 instancias, con 16063 atributos (biomarcadores) cada una, y distribuidos en 14 clases desequilibradas. Los datos están divididos en un conjunto de entrenamiento y un conjunto de testeo.

1.2. El desempeño de algoritmos clásicos

Para disponer de métricas de base para la comparación de nuestra solución y, al mismo tiempo, evaluar el grado de complejidad que presentan los datos incluidos en nuestro estudio hemos seleccionado una serie de modelos ampliamente usados el campo del aprendizaje automático. A fin de estandarizar la implementación de estos algoritmos hemos empleado la librería `scikit-learn` que provee abstracciones convenientes para nuestro entorno de experimentación. Los modelos elegidos son:

Modelos lineales

Los modelos lineales son un conjunto de algoritmos que predicen la salida en función de una combinación lineal de características de entrada. Son particularmente útiles cuando se espera que haya una relación lineal entre variables.

- `LDA`: Análisis Discriminante Lineal, empleado para dimensiones reducidas y asumiendo distribuciones gaussianas.
- `QDA`: Análisis Discriminante Cuadrático, similar a LDA pero con covarianzas distintas por clase.
- `Ridge`: Regresión de Cresta, empleado para tratar con multicolinealidad mediante regularización L2.
- `SGD`: Descenso de Gradiente Estocástico, estrategia central del aprendizaje automático, empleado para optimizar modelos lineales.

Modelos basados en árboles

Los modelos basados en árboles implican la segmentación del espacio de características en regiones simples dentro de las cuales las predicciones son más o menos uniformes.

Son potentes y flexibles, capaces de capturar relaciones no lineales y complejas en los datos.

- `AdaBoost`: Estrategia que entrena modelos débiles secuencialmente, enfocándose en las instancias u observaciones previamente difíciles de clasificar.
- `Bagging`: Estrategia que combina predicciones de múltiples modelos para reducir la varianza.
- `Extra Trees Ensemble`: Estrategia que construye múltiples árboles con splits aleatorios de características y umbrales.
- `Gradient Boosting`: Estrategia que mejora modelos de forma secuencial minimizando el error residual.
- `Random Forest`: Estrategia basada en conjunto de árboles de decisión, cada uno entrenado con subconjuntos aleatorios de datos.
- `DTC`: Árbol de Decisión Clásico, modelo intuitivo que divide el espacio de características.
- `ETC`: Árboles Extremadamente Aleatorizados, variante de Random Forest con más aleatoriedad.

Modelos de Naive Bayes

Los modelos de Naive Bayes son clasificadores probabilísticos basados en el teorema de Bayes que presupone independencia entre las características. Son modelos de rápida ejecución y eficientes.

- `BNB`: Naive Bayes Bernoulli, se emplea para características de variables binarias.
- `GNB`: Naive Bayes Gaussiano, se emplea para distribución normal de los datos.

Modelos de vecinos más cercanos

KNN es un método de clasificación no paramétrico que clasifica una muestra basándose en cómo están clasificadas las muestras más cercanas en el espacio de características. Es simple y efectivo, particularmente para datos donde las relaciones entre características son complejas o desconocidas.

- `KNN`: K-Vecinos más Cercanos, clasifica según la mayoría de votos de los vecinos.

Modelos de redes neuronales

El Perceptrón Multicapa es un tipo de red neuronal que consiste en múltiples capas de neuronas con funciones de activación no lineales. Puede modelar relaciones complejas y no lineales entre entradas y salidas, y es altamente adaptable a la estructura de los datos.

- `MLP`: Perceptrón Multicapa, red neuronal con una o más capas ocultas.

Modelos de Máquinas de Vectores de Soporte

Las Máquinas de Soporte Vectorial son un conjunto de algoritmos supervisados que buscan la mejor frontera de decisión que puede separar diferentes clases en el espacio de características. Ofrecen alta precisión y son muy efectivos en espacios de alta dimensión y en casos donde el número de dimensiones supera al número de muestras.

- `LSVC`: Máquina de Vectores de Soporte Lineal, se emplea en espacios de alta dimensión.
- `NuSVC`: SVC con parámetro Nu, que controla el número de vectores de soporte.
- `SVC`: Máquina de Vectores de Soporte, se emplea para espacios de dimensiones intermedias y altas.

Finalmente, es preciso destacar que para el dataset GCM, que contiene 14 clases en la variable objetivo, hemos excluido modelos no compatibles o ineficientes para problemas de clasificación multiclases.

1.3. Configuración de los Modelos

Para evaluar los modelos clásicos hemos decidido su configuración a partir de la búsqueda de la mejor combinación de parámetros. A tal fin, hemos seleccionado aquellos parámetros más importantes en cada modelo y respecto de cada uno establecimos una búsqueda en grilla de sus respectivos valores. Hemos seleccionado para parámetros numéricos un mínimo de 3 valores y máximo de 20, y para no numéricos hemos decidido la configuración estándar según cada modelo. El espacio de búsqueda resultante para cada modelo puede verse en el siguiente link.

1.4. Resultados Obtenidos

En la siguiente tabla resumimos los resultos obtenidos del entrenamiento de los modelos en los dataset estudiados.

Models	Leukemia Train	Leukemia Test	Madelon Train	Madelon Test	Gisette Train	Gisette Test	GCM Train	GCM Test
LDA	0.93	0.85	0.82	0.6	1.0	0.96	-	-
QDA	1.0	0.5	1.0	0.66	1.0	0.7	-	-
Ridge	1.0	0.99	0.82	0.6	1.0	0.97	-	-
SGD	1.0	0.98	0.63	0.64	1.0	0.99	1.0	0.71
AdaBoost	1.0	0.91	0.89	0.84	1.0	0.99	-	-
Bagging	1.0	1.0	0.97	0.91	-	-	-	-
DTC	1.0	0.72	0.77	0.64	0.95	0.92	0.95	0.53
ETC	1.0	0.54	0.62	0.57	0.95	0.94	0.98	0.48
Ext.Trees.Ens.	1.0	1.0	1.0	0.71	0.99	0.99	1.0	0.57
Gradient Boost.	1.0	0.99	1.0	0.82	1.0	1.0	1.0	0.58
Random Forest	1.0	1.0	1.0	0.78	0.99	0.99	1.0	0.62
BNB	1.0	0.89	0.73	0.63	0.95	0.94	-	-
GNB	1.0	0.91	0.81	0.65	0.91	0.85	-	-
KNN	0.86	0.86	0.74	0.65	0.99	0.99	-	-
LSVC	1.0	0.99	0.78	0.62	1.0	0.99	1.0	0.62
NuSVC	1.0	1.0	1.0	0.61	1.0	0.99	0.99	0.58
SVC	1.0	1.0	1.0	0.61	1.0	0.99	1.0	0.58
MLP	1.0	0.96	1.0	0.58	1.0	0.99	1.0	0.68

Estos valores dan forma a la siguiente representación:

1.5. El uso de AVs como técnica de aumentación

La aplicación de AVs como técnica de aumentación de datos en el contexto de distintos problemas de aprendizaje automático es extendida fuera del campo de los AGs. Se aplica al tratamiento de imágenes (Fajardo et al. 2021; Ai et al. 2023; Khmaissia

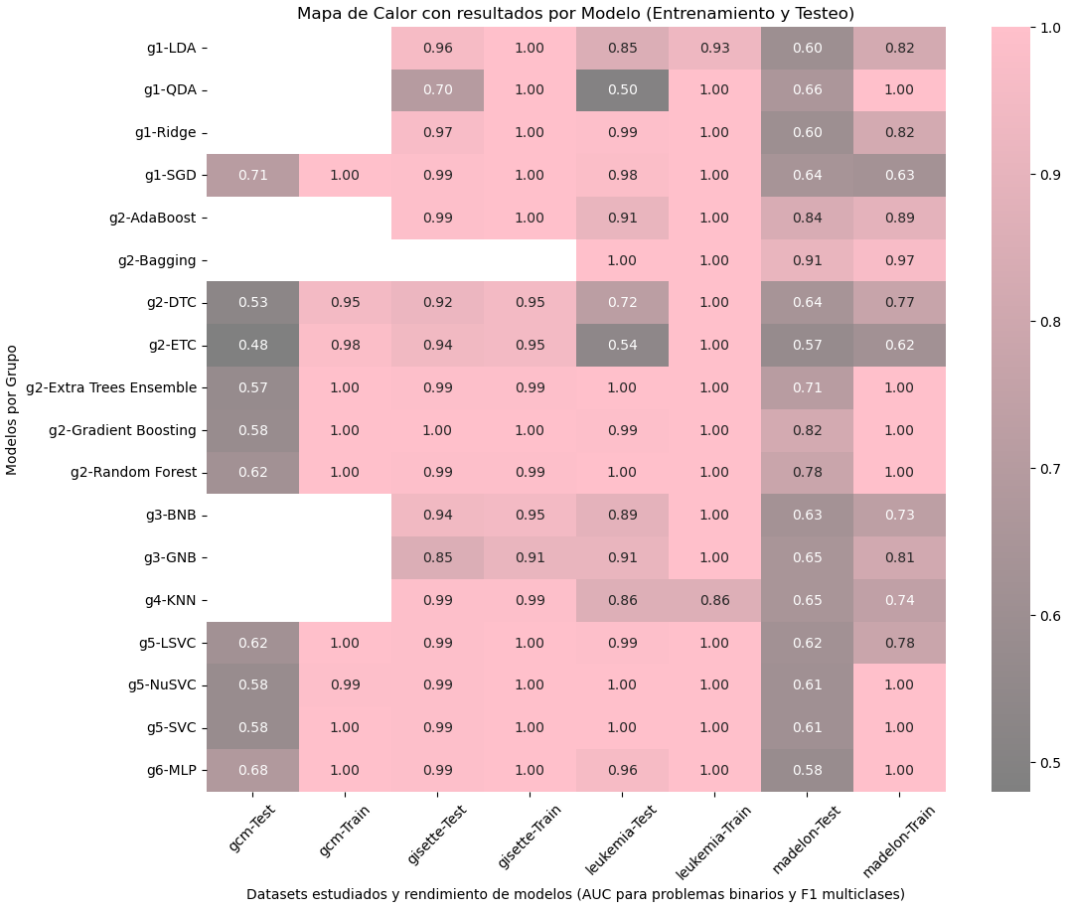

FIGURE 1.1: algoritmosclasicos

and Frigui 2023; Kwarciak and Wodzinski 2023), texto (Zhang et al. 2019) , habla (Blaauw and Bonada 2016; Latif et al. 2020) y música (Roberts et al. 2019), y distintos formatos de datos: tabulares (Leelarathna et al. 2023), longitudinales (Ramchandran et al. 2022) y grafos (Liu et al. 2018). En lo que sigue repasaremos las experiencias más afines a nuestro enfoque sobre el impacto de la aumentación en el aprendizaje y la selección de características.

En Fajardo et al. (2021) se investiga si los AVs y las redes generativas antagónicas (GAN) pueden aumentar datos desbalanceados via sobremuestreo de las clases minoritarias, y mejorar así el rendimiento de un clasificador. Para ello se crean versionanes desbalanceadas de reconocidos datos multiclases MNIST (Lecun 1998) y Fashion MNIST (Xiao, Rasul, and Vollgraf 2017), a los cuales, posteriormente, se los re-balancea agregándoles muestras sintéticas generadas por un AV condicionado por clase (AV Condicional). Para la tarea de clasificación se emplea un Perceptrón Multicapa (MLP), y se evalúa su desempeño promediando métricas de precisión, exhaustividad y F1 sobre distintos experimentos. La evaluación incluye la comparación de resultados del clasificador con datos aumentados por sobremuestreo aleatorio, mediante SMOTE (Blagus and Lusa 2013), GAN y AVs. El resultado muestra a los AVs -en su versión condicional- como el mejor modelo generativo para resolver el problema de datos desbalanceados mediante sobremuestreo de las clases minoritarias.

Ai et al. (2023) vuelve sobre los problemas planteados en Fajardo et al. (2021), proponiento una nueva metodología que superaría sus resultados. La propuesta en esta oportunidad plantea la aumentación de datos de la clase minoritaria condicionada a las características de la distribución que tienen los datos de la clase mayoritaria. El método se llama AV-Guiado-por-la-Mayoría (*Majority-Guided VAE* o MGVAE) y procura incorporar en la generación no solo información intraclase sino también interclases, con el fin de propagar la diversidad y riqueza de la mayoría en la minoría, y mitigar así riesgos de sobreajuste en los modelos. Este modelo se preentrena utilizando muestras de la clase mayoritaria, y luego se ajusta con datos de la clase minoritaria aplicando la regularización EWC (Kirkpatrick et al. 2017) para retener el aprendizaje de la etapa previa. Para evaluar la eficacia de MGVAE, se realizaron experimentos en varios conjuntos de datos de imágenes y tabulares, utilizando diversas métricas de evaluación como Precisión Balanceada (B-ACC), Precisión Específica Promedio por Clase (ACSA) y Media Geométrica (GM). Los resultados muestran que MGVAE supera a otros métodos de sobremuestreo en tareas de clasificación.

Un problema diferente es tratado en Khmaissia and Frigui (2023) donde se emplean AVs para aumentar datos en una tarea de clasificación con enfoque semi-supervisado. Aquí el desafío no pasa por el desbalance entre clases, sino en la búsqueda de mejorar el clasificador en regiones del espacio de características con bajo desempeño (ratios de error altos). Para eso, se mapea el espacio de características entrenando un modelo de WideResNet (Zagoruyko and Komodakis 2017) y luego se seleccionan las muestras mal clasificadas o con bajo nivel de confianza en la clasificación. Estas muestras se utilizan para entrenar un AV y generar datos sintéticos. Finalmente, las imágenes sintéticas se usan junto con las imágenes originales etiquetadas para entrenar un nuevo modelo de manera semi-supervisada. Se evalúan los resultados sobre STL10 y CIFAR-100 obteniendo mejoras en la clasificación de imágenes en comparación con los enfoques supervisados.

Finalmente, antecedente interesante es el presentado por Martins, Rocha, and Pereira (2022) pues pese a no estar directamente vinculado a la aumentación de datos, incluye la generación sintética de muestras mediante AV y la selección de características

por AG. En efecto, el artículo propone la generación de individuos y optimización de características orientados al diseño de proteínas (específicamente variantes de Luciferasa bacteriana *luxA*). Partiendo de muestras de ADN de proteinas obtenidas de un subconjunto de datos de la base InterPro (identificados bajo el código "IPR011251") se generan conjuntos de individuos combinando datos originales, datos muestreados de la capa latente -*encoder*- del AV (configurado como MSA-AV para procesar sequencias alineadas de ADN) y datos optimizados por aplicación del AG. En el caso del algoritmo genético se emplean dos tipos de optimización: de objetivo único y multiobjetivos, con funciones asociadas a la búsqueda de propiedades deseables en las muestras de ADN (solubilidad, síntesis, estabilidad y agregación de proteínas). El resultado de los experimentos realizados muestra que el diseño de proteínas guiado por la optimización mediante AG resultó en mejores soluciones que las obtenidas mediante muestreo directo, y que por su parte la optimización multiobjetivos permitió la selección de proteínas con el mejor conjunto de propiedades.

Los casos mencionados en el apartado nos ofrecen un conjunto de experiencias significativas a considerar al momento de resolver el problema planteado en este trabajo. Otras experiencias, como por ejemplo la configuración evolutiva de un AV (Wu, Cao, and Qi 2023), el ensamble de AVs (Leelarathna et al. 2023), por mencionar algunas novedosas, escapan al recorte que hemos fijado.

Capítulo 2

Autocodificadores Variacionales

En este capítulos presentamos la arquitectura del Autocodificador Variacional (AV) que empleamos para la generación de datos sintéticos. Exponemos brevemente sus fundamentos teóricos, los pasos que hemos seguidos en su implementación en este trabajo y las variaciones introducidas para su apropiada aplicación a los problemas abordados. En el capítulo siguiente nos enfocaremos en los Algoritmos Genéticos, sus fundamentos y características. Finalmente, el último capítulo expondremos los resultados obtenidos combinando ambas tecnologías para resolver problemas de selección de características.

2.1. Modelos generativos

Los modelos generativos (MG) son un amplio conjunto de algoritmos de aprendizaje automático que buscan modelar la distribución de probabilidad de datos observados $p_\theta(x)$. A diferencia de los modelos discriminantes (MD), cuyo objetivo es aprender un predictor a partir de los datos, en los modelos generativos el objetivo es *resolver un problema más general vinculado con el aprendizaje de la distribución de probabilidad conjunta de todas las variables*. Así, siguiendo a Kingma, podemos decir que *un modelo generativo simula la forma en que los datos son generados en el mundo real* (Kingma and Welling 2019). Dada estas propiedades, estos modelos permiten crear nuevos datos que se asemejan a los originales, y se aplican en tareas de generación de datos sintéticos, imputación de datos faltantes, reducción de dimensionalidad y selección de características, entre otros.

Los modelos generativos pueden tener como *inputs* diferentes tipos de dato, como imágenes, texto, audio, entre otros. Por ejemplo, las imágenes son un tipo de dato para los cuales los MG han demostrado gran efectividad. En este caso, cada dato de entrada x es una imagen que puede estar representada por un vector miles de elementos que corresponden a los valores de píxeles. El objetivo de un modelo generativo es aprender las dependencias (Doersch 2021) entre los píxeles (e.g. pixeles vecinos tienden a tener valores similares) y poder generar nuevas imágenes que se asemejen a las imágenes originales.

Podemos formalizar esta idea asumiendo que tenemos ejemplos de datos x, distribuidos según una distribución de probabilidad conjunta no conocida que queremos modelo $p_\theta(x)$ para que sea capaz de generar datos similares a los originales.

2.2. Autocodificadores

Los autocodificadores son un tipo de MG especializado en la representación de un espacio de características dado en un espacio de menor dimensión (de la Torre 2023). El objetivo de esta transformación es obtener una representación de baja dimensionalidad y la mayor fidelidad posible del espacio original. Para ello el modelo aprende a preservar la mayor cantidad de información relevante en un vector denso de menos dimensiones que las originales, y descarta -al mismo tiempo- lo irrelevante. Luego, a partir de esa información codificada, se busca reconstruir los datos observados según el espacio original.

Los autocodificadores se componen de dos partes: un *codificador* y un *decodificador*. El *codificador* es una función no lineal que opera sobre una observación x_i y la transforma en un vector de menor dimensión z, mientras que el *decodificador* opera a partir del vector z y lo transforma en una observación x_i', buscando que se asemeja a la observación original. Este vector de menor dimensión z es conocido como *espacio latente*.

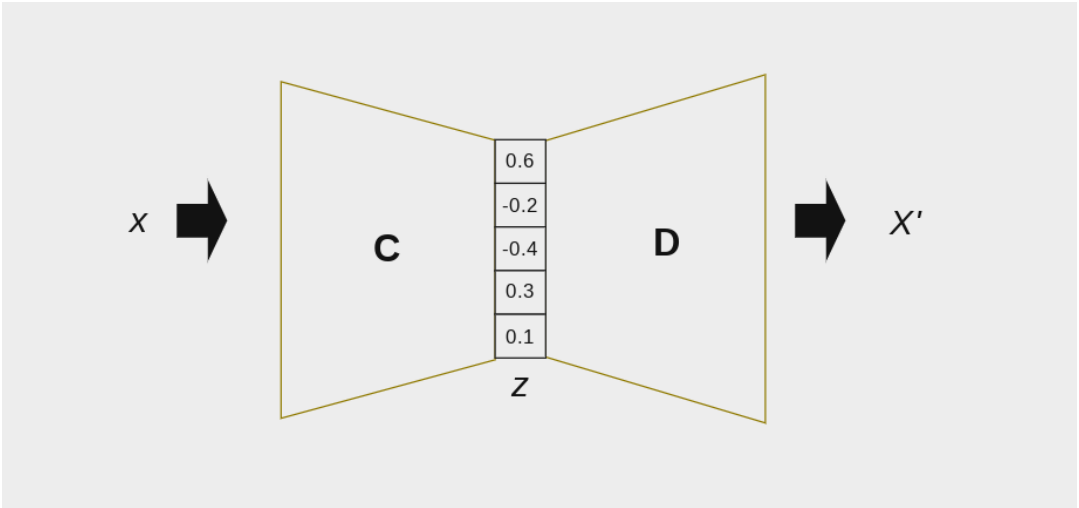

FIGURE 2.1: autocodificadores

En el proceso de aprendizaje de un autocodificador, la red modela la distribución de probabilidad de los datos de entrada x y aprende a mapearlos a un espacio latente z. Para ello, se busca minimizar la diferencia entre la observación original x_i y la reconstrucción x_i', diferencia que se denomina *error de reconstrucción*. Esta optimización se realiza a través de una *función de pérdida* que se define como la diferencia entre x_i y x_i', que permite la optimización simultánea del codificador y decodificador.

Formalmente, podemos establecer estas definiciones (de la Torre 2023):

- Sea x el espacio de características de los datos de entrada y z el espacio latente, ambos espacios son euclidianos, $x = \mathbb{R}^m$ y $z = \mathbb{R}^n$, donde $m > n$.
- Sea las siguientes funciones paramétricas $C_\theta : x \to z$ y $D_\phi : z \to x'$ que representan el codificador y decodificador respectivamente.
- Entonces para cada observación $x_i \in x$, el autocodificador busca minimizar la función de pérdida $L(x_i, D_\phi(E_\theta(x_i)))$. Ambas funciones E_θ y D_ϕ son redes neuronales profundas que se entrenan simultáneamente.

Para optimizar un autocodificador se requiere una función que permita medir la diferencia entre la observación original y la reconstrucción. Esta diferencia usualmente

se basa en la *distancia euclidia* entre x_i y x'_i, es decir, $||x_i - x'_i||^2$. La función de pérdida se define como la suma de todas las distancias a lo largo del conjunto de datos de entrenamiento. Tenemos entonces que:

$$L(\theta, \phi) = argmin_{\theta,\phi} \sum_{i=1}^{N} ||x_i - D_\phi(C_\theta(x_i))||^2$$

Donde $L(\theta, \phi)$ representa la función de pérdida que queremos minimizar: θ son los parámetros del codificador C y ϕ son los parámetros del decodificados D.

2.3. Autocodificadores y el problema de la generación de datos

En el proceso de aprendizaje antes descripto, la optimización no está sujeta a otra restricción mas que minimizar la diferencia entre la observación original y la reconstrucción, dando lugar a espacios latentes generalmente discontinuos. Esto sucede porque la red puede aprender a representar los datos de entrada de manera eficiente sin necesidad de aprender una representación continua. En la arquitectura del autocodificador no hay determinantes para que dos puntos cercanos en el espacio de características se mapeen a puntos cercanos en el espacio latente.

Esta discontinuidad en el espacio latente hace posible que ciertas regiones de este espacio no tengan relación significativa con el espacio de características. Durante el entrenamiento el modelo simplemente no ha tenido que reconstruir datos cuyas distribuciones coincidan con estas regiones. Esto es un problema en la generación de datos, ya que la red podrá generar representaciones alejadas de los datos originales. Regularmente lo que se busca en los MG, no es simplemente una generación de datos completemante igual o totalmente distintos a los orginales, sino cierta situación intermedia donde los nuevos datos introducen variaciones en características específicas.

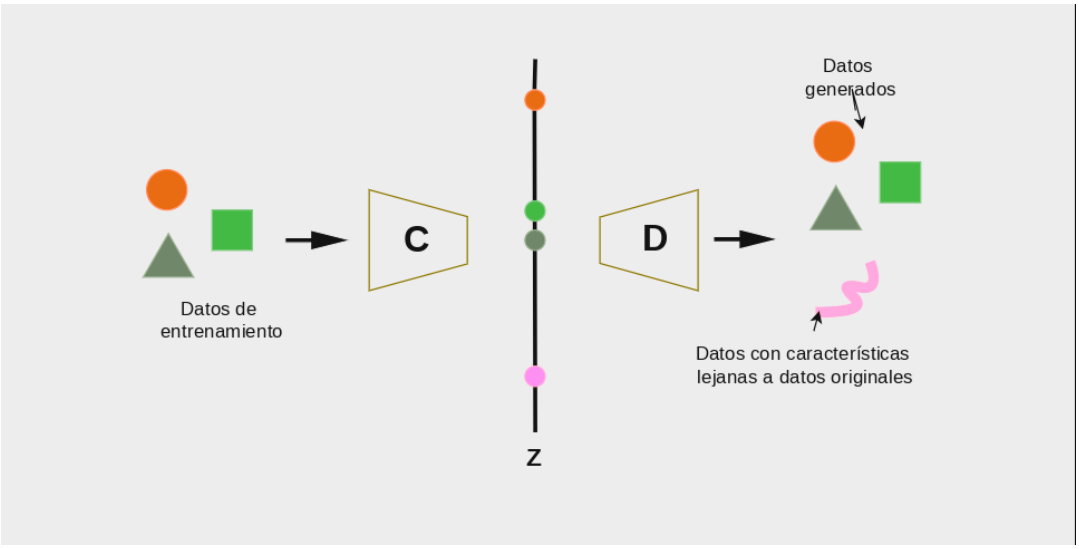

FIGURE 2.2: Discontinuidad del espacio latente

2.4. Autocodificadores Variacionales

Los Autocodificadores Variacionales (AVs) buscan resolver los problemas de discontinuidad y falta de regularidad en el espacio latente de los Autocodificadores. Comparten con éstos la arquitectura *codificador-decodificador*, pero introducen importantes

modificaciones en su diseño para crear un espacio latente continuo.

Estos modelos, a diferencia de los autocodificadores que realizan transformaciones determinísticas de los datos de entrada (codificándolos como vectores n-dimensionales), buscan modelar la distribución de probabilidad de dichos datos aproximando la distribución *a posteriori* de las variables latentes $p_\theta(z|x)$. Para ello, la codificación se produce mediante la generación de dos vectores (μ y σ) que conforman el espacio latente, a partir del cual se toman las muestras para la generación.

La red codificadora, también llamada *red de reconocimiento*, mapea los datos de entrada x a los vectores μ de medias y σ de desvíos estándar, que parametrizan una distribución de probabilidad en el espacio latente. Generalmente, esta distribución es una distribución simple, como la distribución normal multivariada. La red decodificadora, también llamada *red generativa*, toma muestras de esta distribución para generar un vector, y lo transforma según la distribución de probabilidad preexistente del espacio de características. De esta manera, se generan nuevas instancias que reflejan la probabilidad de los datos originales. Estas transformaciones implican que, incluso para el mismo dato observado (donde los parámetros de z son iguales), el dato de salida podrá ser diferente debido al proceso estocástico de reconstrucción.

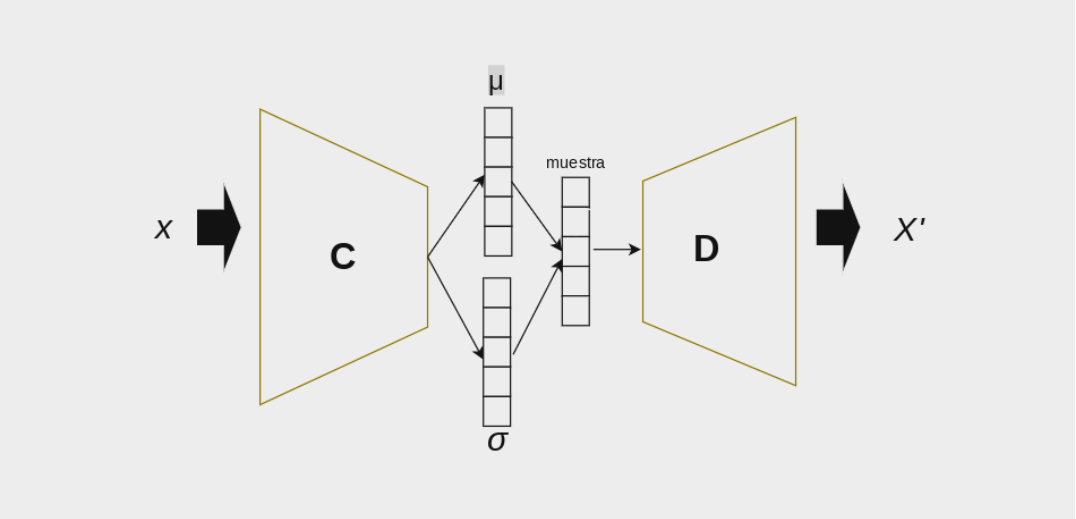

FIGURE 2.3: Autocoficadores Variacionales

Una forma de entender esta arquitectura sería relacionar los vectores que componen z como 'referecias', donde el vector de medias controla el *centro* en torno al cual se distribuirán los valores codificados de los datos de entrada, mientras que el vector de los desvíos traza el *área* que pueden asumir dichos valores en torno al *centro*.

Para indagar en estas intuiciones, veamos la solución que proponen los AV detenidamente, utilizando un enfoque formal. Así, dado un conjunto de datos de entrada $x = \{x_1, x_2, ..., x_N\}$, donde $x_i \in \mathbb{R}^m$, se asume que cada muestra es generada por un mismo proceso o sistema subyacente cuya distribución de probabilidad se desconoce. El modelo buscado procura aprender $p_\theta(x)$, donde θ son los parámetros de la función. Por las ventajas que ofrece el logaritmo[1] para el cálculo de la misma tendremos la siguiente expresión:

[1]El logaritmo convierte la probabilidad conjunta (que se calcula como el producto de las probabilidades condicionales) en una suma de logaritmos, facilitando el cálculo y evitando problemas de precisión numérica: $\log(ab) = \log(a) + \log(b)$.

$\log p_\theta(x) = \sum_{x_i \in x} \log p_\theta(x)$[2]

La forma más común de calcular el parámetro θ es a través del estimador de *máxima verosimilitud*, cuya función de optimización es: $\theta^* = \arg\max_\theta \log p_\theta(x)$, es decir, buscamos los parámetros θ que maximizan la log-verosimilitud asignada a los datos por el modelo.

En el contexto de los AVs, el objetivo es modelar la distribución de probabilidad de los datos observados x a través de una distribución de probabilidad conjunta de variables observadas y latentes: $p_\theta(x, z)$. Aplicando la regla de la cadena de probabilidad podemos factorizar la distribución conjunta de la siguiente manera: $p_\theta(x, z) = p_\theta(x|z)p_\theta(z)$. Aquí $p_\theta(x|z)$ es la probabilidad condicional de los datos observados dados los latentes, y $p_\theta(z)$ es la probabilidad *a priori*[3] de los latentes.

Para determinar la distribución marginal respecto de los datos observados, es preciso integrar sobre todos los elementos de z, dando como resultado la siguiente función:

$p_\theta(x) = \int p_\theta(x, z)dz$ [4]

Esta distribución marginal puede ser extremadamente compleja, y contener un número indeterminable de dependencias (Kingma and Welling 2019), volviendo el calculo de la verosimilitud de los datos observados intratable. Esta intratabilidad de $p_\theta(x)$ está determinada por la intratabilidad de la distribución *a posteriori* $p_\theta(z|x)$, cuya dimensionalidad y multi-modalidad pueden hacer difícil cualquier solución analítica o numérica eficiente. Dicho obstáculo impide la diferenciación y por lo tanto la optimización de los parámetros del modelo.

Para abordar este problema, se acude a la inferencia variacional que introduce una aproximación $q_\phi(z|x)$ a la verdadera distribución *a posteriori* $p_\theta(z|x)$. Generalmente se emplea la distribución normal multivariada para aproximar la distribución *a posteriori*, con media y varianza parametrizadas por la red neuronal[5]. Sin embargo, la elección de la distribución no necesariamente tiene que pasar por una distribución normal, el único requerimiento es que sea una distribución que permita la diferenciación y el cálculo de la divergencia entre ambas distribuciones (por ejemplo si X es binaria la distribución $p_\theta(x|z)$ puede ser una distribución Bernoulli).

Así, en lugar de maximizar directamente el logaritmo de la verosimilitud (*log-verosimilitud*), se maximiza una cota inferior conocida como *límite inferior de evidencia* (*ELBO* por sus siglas en ingles). La derivación procede de la siguiente manera:

1. Log-verosimilitud marginal (intratable):

   $\log p_\theta(x) = \log \left(\int p_\theta(x, z)\, dz \right)$

2. Aplicando inferencia variacional:

[2] Esta función se lee como la log-verosimilitud de los datos observados x bajo el modelo $p_\theta(x)$ y es igual a la suma de la log-verosimilitud de cada dato de entrada x_i.

[3] La expresión *a priori* alude a que no está condicionada por ningun dato observado.

[4] Aquí dz es el diferencial de z, por lo que la expresión indica la integración sobre todas las posibles configuraciones de la variable latente.

[5] En AVs, se suele asumir que z sigue una distribución normal multivariada: $p_\theta(z) = \mathcal{N}(z; 0, I)$, con media cero y matriz de covarianza identidad. La matriz de covarianza identidad es una matriz diagonal con unos en la diagonal y ceros en los demás lugares, y su empleo simplifica la implementación del modelo, permite que las variables latentes sean independientes (covarianza = 0) y varianza unitaria, evitando así cualquier complejidad vinculada a las dependencias entre dimensiones de z.

$$\log p_\theta(x) = \log\left(\int q_\phi(z|x)\frac{p_\theta(x,z)}{q_\phi(z|x)}\,dz\right)$$

3. Aplicando la desigualdad de Jensen[6]:

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}\left[\log\left(\frac{p_\theta(x,z)}{q_\phi(z|x)}\right)\right]$$

4. Descomponiendo la fracción dentro del logaritmo:

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z) + \log p_\theta(z) - \log q_\phi(z|x)\right]$$

5. El resultando es el límite inferior de evidencia:

$$\log p_\theta(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\mathrm{KL}}(q_\phi(z|x)\|p_\theta(z))$$

Donde:

- $\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)]$ es el valor esperado (*esperanza*[7]) de la log-verosimilitud bajo la aproximación variacional, y determina la precision de la reconstrucción de los datos de entrada (un valor alto de esta esperanza indica que el modelo es capaz de reconstruir los datos de entrada con alta precisión a partir de los parámetros generados por $q_\phi(z|x)$).
- $D_{\mathrm{KL}}(q_\phi(z|x)\|p_\theta(z))$ es la divergencia de Kullback-Leibler entre la distribución $q_\phi(z|x)$ y la distribución *a priori* de las variables latentes $p_\theta(z)$, y determina la regularización del espacio latente.

Maximizando esta cota inferior (*ELBO*), se optimizan simultáneamente los parámetros θ del modelo y los parámetros ϕ de la distribución empleada en la aproximación, permitiendo una inferencia eficiente y escalable en modelos con z de alta dimensionalidad [8].

El objetivo de aprendizaje del AV se da entonces por:

$$\mathcal{L}_{\theta,\phi}(x) = \text{máx}(\phi,\theta)\left(E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\mathrm{KL}}(q_\phi(z|x)\|p_\theta(z))\right),$$

Como puede apreciearse en la ecuación anterior la función de pérdida del AV se compone de dos términos: el primero es la esperanza de la log-verosimilitud bajo la aproximación variacional y el segundo es la divergencia de Kullback-Leibler relacionada a la reconstrucción de los datos y la regularización del espacio latente. Existe

[6] Nótese que ese límite es siempre menor o igual y esto se deriva de una de las propiedades de las funciones convexas. Esta propiedad, denominada *desigualdad de Jensen*, establece que el valor esperado de una función convexa es siempre mayor o igual a la función del valor esperado. Es decir, $\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$. En el caso de funciones cóncavas, la desigualdad se invierte: $\mathbb{E}[f(x)] \leq f(\mathbb{E}[x])$. En este caso, la función logaritmo es cóncava, por lo que la desigualdad se expresa como: $\log(\mathbb{E}[x]) \geq \mathbb{E}[\log(x)]$.

[7] La esperanza es un promedio ponderado de todos los posibles valores que puede tomar una variable aleatoria, donde los pesos son las probabilidades de esos valores. En un AV, donde consideramos una distribución aproximada $q_\phi(z|x)$ para el espacio latente, la expresión citada es la esperanza de la log-verosimilitud bajo esta distribución.Aunque teóricamente esto implica un promedio sobre todas las posibles muestras z de la distribución $q_\phi(z|x)$, en la práctica, esta esperanza se estima utilizando una única muestra durante el entrenamiento por razones de eficiencia computacional. Esta única muestra permite calcular directamente $\log p_\theta(x_i|z_i)$, proporcionando una aproximación a la esperanza teórica y determinando la precisión de la reconstrucción de los datos de entrada.

[8] En la teoría, cuando derivamos el objetivo de un AV, estamos maximizando la evidencia inferior variacional (ELBO), para que la aproximación sea lo más cercana posible a la verdadera distribución de los datos. LLevado el problema a una implementación práctica generalmente se emplean optimizadores (SGD, Adam, etc.) que minimizan una función de pérdida. Para convertir el problema de maximización del ELBO en un problema de minimización, simplemente negamos el ELBO, resultando que los términos de la ecuación se reescriben como suma de cantidades positivas. El error o pérdida de reconstrucción se mide, según los casos, mediante MSE o entropía cruzada.

entre ambos términos una relación de compromiso que permite al AV aprender una representación representativa de los datos de entrada y, al mismo tiempo, un espacio latente continuo y regularizado. Cuanto mayor sea la divergencia de Kullback-Leibler, más regularizado será el espacio latente y más suave será la distribución de probabilidad de los datos generados. Cuanto menor sea la divergencia de Kullback-Leibler, más se parecerá la distribución de probabilidad de los datos generados a la distribución de probabilidad de los datos de entrada, sin embargo, el espacio latente será menos regularizado y la generación de datos mas ruidosa.

2.5. Presentación de nuestro modelo de AV

El desarrollo del modelo de Autoencodificador Variacional (AV) empleado en esta investigación se organizó en dos pasos. El primero giró en torno al diseño y validación de la arquitectura del modelo, mientras que el segundo estuvo enfocado en la optimización de dicha arquitectura para la generación de datos sintéticos en cada uno de los dataset bajo estudio. A continuación describiremos brevemente este proceso y la configuración final de los modelos elegidos para los experimentos de aumentación.

2.5.1. Modelo Inicial

En la primera etapa, se centró el esfuerzo en el diseño de la arquitectura del AV. Este proceso comenzó con la creación de una versión exploratoria del modelo, cuya finalidad era establecer una base sobre la cual iterar en mejoras sucesivas. Se optó por una red con 2 capas ocultas en el encoder y en el decoder, siguiendo la estructura básica descripta precedentemente. Esto permitiría al modelo aprender representaciones latentes complejas.

El encoder incluyó dos capas lineales, cada una seguida de una activación ReLU, un diseño que sigue la lógica de la transformación no lineal en un espacio de menor dimensión para luego reconstruir la observación original a partir del espacio latente. Como se discutió en la primera parte, el encoder genera dos vectores, uno para la media y otro para la varianza logarítmica de la distribución latente, componentes críticos para el proceso de reparametrización que permite al modelo generar nuevas muestras en el espacio latente. El decoder, encargado de reconstruir los datos originales a partir del espacio latente, fue diseñado con una estructura simétrica a la del encoder, utilizando nuevamente activaciones ReLU y finalizando con una función Sigmoidea en la capa de salida. La función Sigmoidea condiciona la salida a un rango entre 0 y 1, lo que es particularmente útil para la normalización de los datos de entrada y salida.

La función de pérdida del modelo combinó la divergencia Kullback-Leibler (KLD) y error cuadrático medio (MSE). La KLD se utilizó para medir la diferencia entre la distribución aprendida por el modelo y una distribución normal estándar, mientras que el error cuadrático medio se empleó para evaluar el error de reconstrucción, es decir, qué tan bien el modelo era capaz de replicar los datos de entrada a partir del espacio latente.

Este modelo se probó en la generación de datos sintéticos en un dataset de clases binarias: Madelon, y un dataset multiclases: GCM. Para evaluar los datos generados se realizaron experimentos de clasificación utilizando un MLP, comparando los resultados obtenidos en el dataset original y en el dataset con muestras sintéticas.

Inicialmente, la arquitectura empleada resultó insuficiente para capturar la complejidad de los datos, lo que llevó a un modelo incapaz de representar con precisión las características latentes, produciendo datos sintéticos de baja calidad.

2.5.2. Segundo Modelo AV para clases binarias

En respuesta a los problemas identificados previamente, se diseñó un nuevo modelo con una arquitectura de tres capas lineales en el encoder y el decoder, cada una con activaciones ReLU seguidas de normalización por lotes. Esta técnica de normalización fue seleccionada debido a su capacidad para estabilizar y acelerar el proceso de entrenamiento, promoviendo la rápida convergencia y mejorando la precisión de la reconstrucción. Al mitigar el problemas de desplazamiento de covariables (*covariate shift*) durante el entrenamiento, la normalización por lotes estabiliza las activaciones intermedias de la red y permite que la información relevante sea conservada a lo largo de las capas. Dado que el modelo fue ajustado para capturar la estructura de los datos a través de capas lineales y normalización por lotes, mantuvimos la elección de ReLU como función de activación dada su eficiencia computacional.

El modelo resultante fue entrenado con un optimizador Adam y una tasa de aprendizaje en el rango de [1e-5, 1e-3]. Se experimentó con diferentes tamaños del espacio latente, evaluando el equilibrio entre la calidad de reconstrucción y la capacidad de generalización del modelo. Se empleó un termino de paciencia para detener el entrenamiento si no se observaba mejora en los datos de test durante 10 épocas consecutivas. Este mecanismo de corte temprano mejoró la eficiencia del entrenamiento y la capacidad de generalización del modelo.

Estos experimentos fueron clave para ajustar el AV a las necesidades específicas de los conjuntos de datos utilizados en la investigación, permitiendo una generación de datos sintéticos que no solo replicara los patrones de los datos originales, sino que también capturara la variabilidad inherente a estos.

Arquitectura del Autocodificador Variacional

La arquitectura del Autocodificador Variacional está compuesta por tres capas lineales (una capa de entrada y dos capas ocultas), cada una seguida de una normalización por lotes (Batch Normalization) y una activación ReLU. El proceso de codificación se realiza de la siguiente manera:

$h_1 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_1}x + b_1))$
$h_2 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_2}h_1 + b_2))$
$h_3 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_3}h_2 + b_3))$

Donde:

- $\mathbf{W_1}$es una matriz de pesos que transforma el vector de entrada x al espacio de características de dimensión H.
- $\mathbf{W_2}$transforma h_1 a un espacio de características de dimensión $H2$.
- $\mathbf{W_3}$ mantiene la dimensión $H2$ mientras transformah_2.
- b_1,b_2, y$b_3$son los sesgos correspondientes a cada capa.

Después de las tres capas, se generan los vectores latentes μ y $\log(\sigma^2)$ mediante dos capas lineales independientes que también aplican normalización por lotes.

Reparametrización

El vector latente z se obtiene mediante la técnica de reparametrización, donde se introduce ruido gaussiano para permitir la retropropagación del gradiente:

$z = \mu + \sigma \times \epsilon$

Donde ϵ es una variable aleatoria con distribución normal estándar, y σ se calcula a partir de $\log(\sigma^2)$.

Decodificador

El decodificador reconstruye el vector de entrada a partir del vector latente$z$utilizando una arquitectura de tres capas lineales, cada una seguida por una normalización por lotes y una activación ReLU:

$h_4 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_4}z + b_4))$
$h_5 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_5}h_4 + b_5))$
$\hat{x} = \text{BatchNorm}(\mathbf{W_6}h_5 + b_6)$

Donde:

- $\mathbf{W_4}$ transforma el vector latente z al espacio de características de dimensión $H2$.
- $\mathbf{W_5}$ transforma h_4 al espacio de características de dimensión H.
- $\mathbf{W_6}$ transforma h_5 de regreso al espacio de la dimensión original de la entrada D_{in}.
- b_4,b_5, y$b_6$son los sesgos correspondientes a cada capa.

Finalmente, la salida $\hat{x}$ es una aproximación reconstruida de la entrada original x.

2.5.3. Modelo AVC para datos multiclase

Para abordar el dataset GCM, que contiene 14 clases con distribuciones desiguales, se creo un Autocodificador Variacional Condicional (AVC) que combina la capacidad de generación de un AV tradicional con el condicionamiento explícito en las etiquetas de clase. El AVC propuesto permitió una modelización más precisa de los datos, al incorporar información de clase en el proceso de codificación y decodificación.

En escenarios donde los datasets están desbalanceados, los modelos generativos pueden tender a favorecer las clases mayoritarias, ignorando las minoritarias. Para abordar el desbalance de clases que presenta GCM, se implementó una estrategia de ponderación de clases dentro de la función de pérdida, penalizando de manera diferenciada los errores de reconstrucción en función de la clase, mejorando así la capacidad del modelo para representar adecuadamente las clases minoritarias.

Arquitectura del Autocodificador Variacional Condicional (AVC)

La arquitectura del Autocodificador Variacional Condicional (AVC) se basa en una modificación del Autocodificador Variacional tradicional para incorporar información adicional en forma de etiquetas. Esta información se concatena tanto en la fase de codificación como en la de decodificación, permitiendo que el modelo aprenda distribuciones condicionales.

Codificador

El codificador del AVC combina la entrada original con las etiquetas antes de ser procesada por una secuencia de capas lineales (una capa de entrada y dos capas ocultas), cada una seguida por una normalización por lotes (Batch Normalization) y una activación ReLU. El proceso de codificación se realiza de la siguiente manera:

$h_1 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_1}[x, y] + b_1))$
$h_2 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_2}h_1 + b_2))$
$h_3 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_3}h_2 + b_3))$

Donde:

- $[x, y]$ es la concatenación del vector de entrada x con las etiquetas y.
- $\mathbf{W_1}$ es una matriz de pesos que transforma el vector combinado $[x, y]$ al espacio de características de dimensión H.
- $\mathbf{W_2}$ transforma h_1 a un espacio de características de dimensión $H2$.
- $\mathbf{W_3}$ mantiene la dimensión $H2$ mientras transforma h_2.

- b_1, b_2, y b_3 son los sesgos correspondientes a cada capa.

Al igual que en el Autocodificador Variacional tradicional, se generan los vectores latentes μ y $\log(\sigma^2)$ mediante dos capas lineales independientes.

Reparametrización

El vector latente z se obtiene mediante la técnica de reparametrización, similar al Autocodificador Variacional tradicional:

$z = \mu + \sigma \times \epsilon$

Donde ϵ es una variable aleatoria con distribución normal estándar, y σ se calcula a partir de $\log(\sigma^2)$.

Decodificador

El decodificador del AVC reconstruye el vector de entrada a partir del vector latente z y las etiquetas y, utilizando una arquitectura de tres capas lineales, cada una seguida por una normalización por lotes y una activación ReLU:

$h_4 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_4}[z, y] + b_4))$
$h_5 = \text{ReLU}(\text{BatchNorm}(\mathbf{W_5}h_4 + b_5))$
$\hat{x} = \text{sigmoid}(\mathbf{W_6}h_5 + b_6)$

Donde:

- $[z, y]$ es la concatenación del vector latente z con las etiquetas y.
- $\mathbf{W_4}$ transforma el vector combinado $[z, y]$ al espacio de características de dimensión $ H2 + labels_length$.
- $\mathbf{W_5}$ transforma h_4 al espacio de características de dimensión H.
- $\mathbf{W_6}$ transforma h_5 de regreso al espacio de la dimensión original de la entrada $ D_{in}$.

- b_4, b_5, y b_6 son los sesgos correspondientes a cada capa.

Finalmente, la salida $\hat{x}$ es una aproximación reconstruida de la entrada original x, condicionada por las etiquetas y.

Elementos distintivos respecto a la arquitectura anterior:

- **Incorporación de etiquetas**: Tanto en el codificador como en el decodificador, se concatenan las etiquetas $ y$ con las entradas y el vector latente, respectivamente.
- **Dimensiones ajustadas**: Se han ajustado las dimensiones de las capas para acomodar las etiquetas, reflejadas en las matrices de pesos y las normalizaciones por lotes.
- **Capas adicionales en la fase de decodificación**: Se añaden capas y ajustes para manejar las etiquetas adicionales en el proceso de decodificación.

Esta arquitectura permite que el AVC capture relaciones condicionales más complejas entre las entradas y sus correspondientes etiquetas.

2.5.4. Optimización de Hiperparámetros

La búsqueda y ajuste de hiperparámetros para los modelos de AV y AVC ha sido un proceso crucial para optimizar la generación de datos sintéticos y, en última instancia, mejorar el rendimiento de nuestro Algoritmo Genético. A lo largo de esta etapa de investigación, se implementaron diversas estrategias para identificar las configuraciones óptimas de los modelos, así como para evitar el sobreajuste y garantizar la robustez de los resultados.

Uno de los primeros pasos fue ampliar la búsqueda de hiperparámetros, ajustando variables clave como las dimensiones latentes, las tasas de aprendizaje, y el número de neuronas en las capas ocultas. Un cambio significativo fue la implementación de un mecanismo de paciencia, configurado para detener el entrenamiento si no se observaba mejora en los datos de test durante 10 épocas consecutivas. Esta modificación tuvo un impacto directo en la calidad del modelo, ya que en experimentos iniciales, los modelos seguían entrenándose durante todas las épocas establecidas, lo que en muchos casos resultaba en un sobreajuste. La implementación de este corte temprano no solo mejoró la eficiencia del entrenamiento, sino que también contribuyó a reducir el error y mejorar la capacidad de generalización del modelo.

El análisis de los resultados obtenidos mediante estas configuraciones reveló que un MLP entrenado con datos sintéticos generados por un AV podía igualar o incluso superar el rendimiento de un MLP entrenado con datos reales en algunos casos. Este hallazgo es particularmente relevante, ya que sugiere que, bajo ciertas configuraciones, los datos sintéticos pueden ser tan útiles como los datos reales para el entrenamiento de modelos predictivos. Este fenómeno se observó de manera consistente en varios conjuntos de datos, como Leukemia, Madelon, y Gisette, donde la precisión y la exactitud del modelo entrenado con datos sintéticos alcanzaron o superaron las métricas obtenidas con los datos originales.

Un hallazgo interesante se refiere a las dimensiones latentes del modelo. A medida que se ampliaba la búsqueda de hiperparámetros, se descubrió que las mejores configuraciones para la variable latente no eran necesariamente las más grandes. De hecho, en muchos casos, valores para la dimensión latente entre 3 y 100 ofrecieron los mejores resultados. Esto, que inicialmente puede ser contraintuitivo, ya que se podría suponer que un mayor espacio latente permitiría capturar más complejidad en los datos; sugiere que un espacio latente excesivamente grande puede introducir ruido y hacer que el modelo pierda la capacidad de generalizar correctamente.

Las pruebas con diferentes arquitecturas también proporcionaron información valiosa. En el caso de leukemia, se exploraron modelos AV de dos, tres y cuatro capas ocultas, así como AVC con múltiples capas, pero no se observaron mejoras significativas al

aumentar la complejidad del modelo. En particular, se encontró que las configuraciones más simples (i.e. dos capas ocultas), ofrecían resultados tan buenos o incluso mejores que sus contrapartes más complejas. Esta observación refuerza la idea de que, en algunos casos, la simplicidad puede ser preferible y que el sobredimensionamiento de la arquitectura no necesariamente se traduce en mejores resultados.

Por otro lado, los experimentos realizados en el dataset GCM presentaron un desafío diferente. A pesar de la implementación de un AVC de tres capas ocultas, los resultados no mostraron mejoras sustanciales en comparación con un más simple de dos capas. Además, se observó una disminución en la capacidad del modelo para predecir correctamente clases con menor soporte en el conjunto de datos, lo que sugiere que la complejidad del modelo no fue capaz de capturar adecuadamente la variabilidad de las clases menos representadas. Este resultado subraya la dificultad inherente al trabajo con conjuntos de datos multiclase, especialmente cuando las clases tienen distribuciones subyacentes similares, están desbalanceadas o ambos.

En cuanto a la búsqueda de hiperparámetros, se utilizaron tanto Grid Search como Optimización Bayesiana (BO). Cada una de estas técnicas tiene sus fortalezas, y la elección entre ellas depende en gran medida del objetivo de la búsqueda. Grid Search, por ejemplo, permite un control total sobre el espacio de búsqueda, lo que es útil para responder preguntas específicas, como la configuración óptima de la dimensión latente. Sin embargo, la BO demostró ser particularmente eficiente en la exploración de un espacio de hiperparámetros más amplio y menos definido, logrando un equilibrio entre la exploración y la explotación que resultó especialmente útil en nuestra investigación dado el tamaño del espacio de búsqueda.

A pesar de los avances logrados, también se encontraron limitaciones. Por ejemplo, incrementar el tamaño de los datos sintéticos en leukemia no condujo a una mejora significativa en los resultados, lo que sugiere que, para ciertos datasets, los beneficios de aumentar los datos sintéticos son marginales una vez alcanzado un umbral de rendimiento. En el caso de GCM, los problemas de baja calidad en la reconstrucción de datos sintéticos por parte del AVC sugieren que simplemente aumentar el tamaño del dataset no compensa una arquitectura subóptima.

En efecto, uno de los experimentos más interesantes fue el relacionado con el aumento del tamaño del conjunto de datos sintéticos en el dataset GCM. Inicialmente, se logró incrementar las observaciones del conjunto de datos de entrenamiento a 3000 muestras balanceadas, con 214 observaciones por clase. Este aumento resultó en una mejora significativa en la performance del modelo, logrando igualar los resultados obtenidos con el clasificador MLP entrenado con datos reales. Sin embargo, al continuar incrementando la cantidad de datos sintéticos a 6000 muestras, se observó una degradación en el rendimiento. Esto sugiere la existencia de un umbral en la cantidad de datos sintéticos que, una vez superado, introduce ruido en el modelo en lugar de aportar valor. Este ruido puede estar relacionado con el solapamiento de las fronteras de decisión en las muestras generadas, lo que aumenta el error y disminuye la precisión del modelo.

Los resultados obtenidos en los experimentos reflejan que los beneficios de la aumentación de datos tienen un límite. Superado este umbral, la generación adicional de datos no solo deja de ser útil, sino que puede ser perjudicial, como se evidenció en nuestros experimentos. Este fenómeno destaca la importancia de una cuidadosa calibración en la cantidad de datos sintéticos generados, especialmente en conjuntos de datos con características complejas y altamente dimensionales como GCM.

Otro aspecto explorado fue la implementación de la pérdida L1 en lugar de MSE. Se realizaron pruebas para evaluar si la L1_loss podría ofrecer mejoras, pero los resultados no mostraron diferencias significativas en comparación con MSE. Este hallazgo sugiere que, al menos en este contexto específico, la L1_loss no proporciona un beneficio claro sobre el MSE para la tarea de generación de datos sintéticos.

Además, se experimentó con el uso de dropout como técnica de regularización. Se probaron tasas de dropout en un rango de 0.05 a 0.5 en distintas configuraciones de AVC, tanto con arquitecturas pequeñas (100-500 neuronas por capa) como grandes (1000-7000 neuronas por capa). Aunque algunos experimentos con una arquitectura más pequeña mostraron resultados interesantes en el clasificador MLP, en conjunto este grupo de experimentos se mantuvieron por debajo de los mejores experimentos previos. Esto sugiere que el dropout, si bien útil en otros contextos, no aporta beneficios en configuraciones ya optimizadas del AVC para este tipo de tareas.

Estos resultados llevaron a una reflexión sobre la falta de impacto positivo de ciertos ajustes, como la introducción de L1_loss y dropout. Es probable que la estabilidad y el buen rendimiento de las configuraciones ya validadas de AV y AVC, alcanzados a través de numerosos experimentos, limiten el potencial de mejora adicional mediante estos métodos. De hecho, en lugar de mejorar el rendimiento, estos cambios podrían estar degradando los resultados debido a la interferencia con una configuración ya optimizada.

En resumen, la búsqueda de hiperparámetros y el ajuste de la arquitectura del AV y AVC revelaron la importancia de un enfoque balanceado que evite tanto la simplicidad excesiva como la complejidad innecesaria. Los resultados obtenidos muestran que, bajo ciertas configuraciones, los datos sintéticos pueden igualar o superar la utilidad de los datos reales en la formación de modelos predictivos, aunque la eficiencia y la calidad de estos resultados dependen en gran medida de la cuidadosa calibración de los hiperparámetros y de la adecuada elección de la arquitectura del modelo.

Capítulo 3

Algoritmos Genéticos

En este capítulos presentamos una descripción general de los AGs, exponemos brevemente sus características y describimos la implementación realizada en nuestro trabajo. Específicamente, comentaremos su componentes y operadores básicos, y presentaremos un script genérico de un AG implementado con la librería `DEAP` de Python, que puede ser adaptado para la selección de características en problemas de alta dimensionalidad. En el capítulo siguiente nos enfocaremos finalmente en los resultados obtenidos combinando ambas tecnologías para resolver problemas de selección de características.

3.1. Elementos básicos de los Algoritmos Genéticos

Los Algoritmos Genéticos (AGs) son una clase de algoritmos de optimización inspirados en la evolución biológica y en la teoría de la selección natural. Los AGs se basan en el concepto de evolución de una población de soluciones potenciales a lo largo de múltiples generaciones, utilizando operadores genéticos como la selección, el cruce y la mutación para generar nuevas soluciones y mejorar la calidad de las mismas.

En el siguiente fragmento de código presentamos las operaciones elementales de un AG, que como dice Goldberg son *extraordinariamente sencillas*. La secuencia de operaciones se inicializa con un conjunto de soluciones, denominado población. El ciclo iterativo principal del Algoritmo Genético genera nuevas soluciones candidatas descendientes mediante cruce y mutación hasta que la población esté completa. En cada iteración, los individuos son evaluados mediante una función de aptitud que mide su calidad en relación al problema a resolver (aquí, la función de aptitud es simplemente la suma de los valores de los genes, en contextos reales, esta función se ajusta a las necesidades del problema pudiendo ser una función de costo, una métrica de desempeño, entre otras). Los individuos más aptos son seleccionados para reproducirse, lo que implica la aplicación de operadores genéticos para generar nuevos individuos. Este proceso se repite a lo largo de múltiples generaciones, permitiendo que la población evolucione y se adapte a las condiciones del problema.

```
import random

# Parámetros del Algoritmo Genético
num_individuals = 5
chromosome_length = 10
num_generations = 10
mutation_rate = 0.1
```

```
# Inicializar la población con individuos aleatorios
population = [[random.randint(0, 1) for _ in range(chromosome_length)] for _ in ran

# Ejecutar el Algoritmo Genético
for generation in range(num_generations):
    # Calcular aptitud
    fitness_values = [sum(ind) for ind in population]

    # Crear nueva población
    new_population = []
    while len(new_population) < num_individuals:
        # Selección de dos padres
        parent1 = random.choices(population, weights=fitness_values)[0]
        parent2 = random.choices(population, weights=fitness_values)[0]

        # Cruce de un punto
        point = random.randint(1, chromosome_length - 1)
        child1 = parent1[:point] + parent2[point:]
        child2 = parent2[:point] + parent1[point:]

        # Mutación
        for i in range(chromosome_length):
            if random.random() < mutation_rate:
                child1[i] = 1 - child1[i]
            if random.random() < mutation_rate:
                child2[i] = 1 - child2[i]

        new_population.append(child1)
        if len(new_population) < num_individuals:
            new_population.append(child2)

    # Reemplazar la población antigua con la nueva
    population = new_population

    # Mostrar la población actual y sus aptitudes
    print(f"Generación {generation + 1}:")
    for ind in population:
        print(f"Individuo: {ind}, Aptitud: {sum(ind)}")
    print()
```

A pesar de su extraordinaria simpleza -o quizas gracias a ella-, los AG constituyen algoritmos robustos, capaces de encontrar soluciones efectivas en una amplia variedad de problemas de optimización. Dicha robustez está determinada, como bien sostiene Goldberg (1989), por una serie de características distintivas, que fortalecen su configuración de búsqueda, a saber: a) operan sobre un espacio *codificado* del problema y no sobre el espacio en su representación original; b) realizan la exploración evaluando una *población de soluciones* y no soluciones individuales; c) tienen como guía una *función objetivo* (también llamada *función de aptitud*) que no requiere derivación u otras funciones de cálculo; y d) suponen *métodos probabilísticos de transición* (operadores estocásticos) y no reglas determinísticas. Estas características permiten a los

AG superar restricciones que tienen otros métodos de optimización, condicionados -por ejemplo- a espacios de búsqueda continuos, diferenciables o unimodales. Por ello, su aplicación se ha difundido notablemente, trascendiendo los problemas clásicos de optimización, aplicándose en distintas tareas (Vie, Kleinnijenhuis, and Farmer 2021) y a lo largo de diversas industrias (Jiao et al. 2023).

En lo que sigue expondremos brevemente cada una de estas características, destacando su relevancia en el contexto de los AGs y su aplicación en nuestra investigación.

3.2. a) Codificación del espacio de búsqueda

Como señalamos, los AGs se distinguen de otros algoritmos por su capacidad para operar en un espacio codificado del problema, en lugar de operar directamente el espacio en su representación original. Esto sucede gracias a la transformación de las soluciones potenciales en cadenas de datos, comúnmente conocidas como **cromosomas**, que luego son objeto de transformación mediante operadores genéticos como la mutación y el cruce. La capacidad de los AGs para operar con estas representaciones codificadas determina su adaptabilidad y eficacia en una amplia gama de problemas de optimización.

La codificación adecuada del problema es un paso inicial clave para el correcto desempeño del algoritmo. La elección de la codificación depende de la naturaleza del problema y de las características de las soluciones que se buscan optimizar. Por ejemplo, en problemas de optimización combinatoria, dado que las soluciones pueden representarse como permutaciones, una opción intuitiva en términos de codificación es la secuencias de números enteros. Por otro lado, en problemas de optimización continua, como la optimización de funciones matemáticas, las soluciones pueden representarse como vectores de números reales, lo que sugiere una codificación real-valuada. Así, la elección de la codificación adecuada en principio no tiene una respuesta única, antes bien admite múltiples alternativas confiriendo flexibilidad al diseño del AG.

Dada la importancia que tiene la codificación, es fácil advertir que así como una elección adecuada de la estrategia de codificación puede facilitar la convergencia del AG hacia soluciones óptimas, una elección inadecuada puede tener consecuencias negativas en su desempeño. En efecto, una codificación inapropiada puede llevar a una exploración ineficaz del espacio de soluciones o incluso a la generación de soluciones inviables. Así, una codificación que no preserve la viabilidad de las soluciones durante la evolución, puede resultar en la convergencia prematura del AG hacia soluciones subóptimas.

La traducción entre la representación interna codificada (genotipo) y la solución en el contexto del problema (fenotipo) es un componente importante del diseño de los AGs [1]. Este mapeo no solo permite interpretar las soluciones generadas por el algoritmo,

[1] El **genotipo** se refiere a la representación interna de una solución en el AG. Es la "cromosoma" o la estructura de datos que codifica la información genética de un individuo. En la mayoría de los casos, el genotipo se representa como una cadena de bits (0 y 1), pero también puede ser una cadena de números, caracteres, o cualquier otra estructura adecuada dependiendo del problema. Por otro lado, el **fenotipo** es la manifestación externa o la interpretación del genotipo en el contexto del problema. Es la forma en que se evalúa la solución codificada por el genotipo. El fenotipo corresponde a la solución real en el espacio de búsqueda y es lo que se evalúa mediante la función de aptitud para determinar la calidad de un individuo. Por ejemplo, en un problema de selección de características donde cada característica puede ser incluida o excluida. El genotipo podría ser una cadena binaria donde cada bit representa si una característica está seleccionada (1) o no (0). Genotipo: `1100101`. Aquí, el genotipo representa la selección de ciertas características en un conjunto de datos. Este

sino que también influye en la eficacia de los operadores genéticos. Ello así, por cuanto los operadores genéticos actúan directamente sobre la representación codificada, lo que puede afectar la exploración del espacio de soluciones y la convergencia del AG.

Una de las principales ventajas de operar en un espacio codificado del problema radica en la posibilidad de aplicar operadores genéticos de manera eficiente, lo que permite una exploración exhaustiva del espacio de soluciones. En efecto, los operadores genéticos -que veremos en breve- son diseñados específicamente para actuar directamente sobre la representación codificada, generando nuevas soluciones de manera efectiva.

Un proceso típico de codificación y decodificación en un AG incluye los siguientes pasos:

1. **Espacio Original**: Representación directa del problema, por ejemplo, valores continuos o categóricos.
2. **Codificación**: Traducción del espacio original a una forma binaria o simbólica.
3. **Operadores Genéticos**: Aplicación de mutación, cruce y selección en la representación codificada.
4. **Decodificación**: Traducción inversa de la solución codificada al espacio original para evaluación.

En el caso de nuestra investigación, dada la alta dimensionalidad de los datos y la complejidad de los modelos, la codificación adecuada de las soluciones fue un proceso fundamental para garantizar que los AGs puedan encontrar soluciones óptimas o cercanas al óptimo en tiempo razonable.

3.3. b) Búsqueda por *población de soluciones*

Una de las características distintivas de los AGs es su enfoque en la evaluación de una **población** de soluciones en cada iteración, en lugar de centrarse en una única solución. Esta población de soluciones, también conocida como población de **individuos**, permite a los AGs explorar simultáneamente múltiples regiones del espacio de búsqueda, aumentando así la probabilidad de encontrar soluciones óptimas o cercanas al óptimo.

Como vimos en el ejemplo de más arriba, la población inicial regularmente se genera de manera aleatoria, y cada individuo dentro de esta población representa una solución potencial al problema. A lo largo de las generaciones, los AGs aplican operadores genéticos como selección, cruce y mutación para producir nuevas generaciones de individuos, mejorando iterativamente la calidad de las soluciones.

Un esquema general del proceso de búsqueda por población en un AG se presenta a continuación:

genotipo incluye las características 1, 2, 4, y 7, mientras que excluye las características 3, 5, y 6. El **fenotipo** es la manifestación externa o la interpretación del genotipo en el contexto del problema. Es la forma en que se evalúa la solución codificada por el genotipo. El fenotipo corresponde a la solución real en el espacio de búsqueda y es lo que se evalúa mediante la función de aptitud para determinar la calidad de un individuo. Siguiendo el ejemplo del genotipo `1100101`, el fenotipo sería la selección efectiva de características en un conjunto de datos. Si tenemos 7 características disponibles, el fenotipo se traduciría en el subconjunto de características seleccionadas por el genotipo. Por ejemplo: Características Disponibles: `[X1, X2, X3, X4, X5, X6, X7]`, Fenotipo: `[X1, X2, X4, X7]` En este caso, el fenotipo es el subconjunto de datos que incluye solo las características seleccionadas (X1, X2, X4, X7). Este subconjunto se utilizará para entrenar un modelo, y su desempeño será evaluado para determinar la aptitud del individuo.

Esquema del proceso de búsqueda por población

1. **Población Inicial**: Generación aleatoria de un conjunto de individuos que representan soluciones potenciales.
2. **Evaluación de Población**: Cada individuo es evaluado según una función de aptitud para determinar su calidad.
3. **Operadores Genéticos**:
   - **Selección**: Elegir individuos más aptos para reproducirse.
   - **Cruce (Crossover)**: Combinar partes de dos individuos para crear uno nuevo.
   - **Mutación**: Alterar aleatoriamente un individuo para introducir variabilidad.
4. **Nueva Generación**: Creación de una nueva población basada en los individuos más aptos.
5. **Iteración**: Repetición del proceso a través de múltiples generaciones hasta alcanzar un criterio de terminación.

La diversidad genética dentro de la población es fundamental para la eficacia de los AGs, ya que permite a los algoritmos explorar de manera más exhaustiva el espacio de características y evitar la convergencia prematura hacia soluciones subóptimas. En efecto, consideremos una población homogénea donde todos los individuos son idénticos. En este caso, la capacidad del AG para explorar nuevas regiones del espacio de búsqueda se ve severamente limitada, lo que puede resultar en una convergencia temprana hacia soluciones subóptimas. Por el contrario, una población diversa, donde cada individuo representa una solución única, permite al AG explorar una variedad de soluciones y adaptarse a las condiciones cambiantes del problema.

A modo de ejemplo consideremos estas dos población de 5 individuos codificados como sigue:

Población A, con 5 individuos de longitud 5, de alta diversidad:

- Individuo 1: `11001`
- Individuo 2: `10110`
- Individuo 3: `01101`
- Individuo 4: `11100`
- Individuo 5: `00011`

Población B, con 5 individuos de longitud 5, de baja diversidad:

- Individuo 1: `11111`
- Individuo 2: `11111`
- Individuo 3: `11111`
- Individuo 4: `11111`
- Individuo 5: `11111`

Como podemos advertir en este ejemplo, cada individuo representa una solución potencial al problema, donde cada bit en la cadena codificada corresponde a una característica que puede ser seleccionada o excluida. En estas poblaciónes los individuos de A son distintos entre sí, lo que permite al AG explorar una variedad de soluciones y adaptarse a las condiciones cambiantes del problema, mientras que los individuos de B son idénticos, lo que limita la capacidad del AG para explorar nuevas regiones del espacio de búsqueda.

3.4. c) Función de aptitud y evaluación de soluciones

La función de aptitud es el núcleo que dirige el proceso evolutivo en los AGs, determinando qué soluciones sobreviven y se propagan a la siguiente generación. Su diseño y correcta implementación son esenciales para asegurar que el AG no solo converja hacia soluciones de alta calidad, sino que también lo haga de manera eficiente y efectiva, especialmente en problemas donde las evaluaciones de aptitud son costosas o complejas.

En el proceso evolutivo de los AGs, La función de aptitud se aplica al **fenotipo** de cada solución, es decir, a su manifestación en el contexto del problema a resolver, después de que el **genotipo** (la representación codificada de la solución) ha sido transformado. Esta evaluación cuantifica qué tan bien una solución potencial cumple con los objetivos del problema, asignándole un valor numérico que refleja su desempeño relativo en comparación con otras soluciones dentro de la población.

El diseño de la función de aptitud es un aspecto crítico del proceso de modelado en los AGs, ya que guía la dirección de la búsqueda evolutiva. Específicamente, la función de aptitud debe estar alineada con los objetivos del problema, reflejando correctamente las restricciones necesarias a satisfacer. En situaciones de optimización multiobjetivo, donde varios criterios deben ser optimizados simultáneamente, es común que funciones de aptitud individuales se combinen en una única métrica a través de técnicas como la suma ponderada de los valores de aptitud individuales. En el contexto de nuestra investigación, orientada a la selección de características, la función de aptitud combina la aptitud de un individuo en términos de precisión y el tamaño del conjunto de características seleccionadas (veremos un ejemplo en breve).

En línea con lo anterior, la evaluación precisa de las soluciones mediante la función de aptitud puede constituir un proceso sujeto a multiples restricciones. Aunque la asignación de valores de aptitud más bajos a soluciones subóptimas y más altos a soluciones superiores pueda parecer un criterio ineludible, en la práctica, este proceso requiere comunmente consideraciones adicionales. Por ejemplo, en problemas con restricciones, una solución que se acerque significativamente al óptimo global pero que infrinja requerimientos esenciales del problema debería recibir una calificación de aptitud inferior a una solución factible aunque menos cercana al óptimo con el fin de orientar la búsqueda hacia soluciones viables. Con esa lógica, en la optimización multiobjetivo es necesario establecer criterios para ponderar la proximidad al óptimo, especialmente cuando los distintos objetivos compiten entre sí.

Otro aspecto importante en la función de aptitud es la minimización del número de evaluaciones necesarias para alcanzar el óptimo o una solución lo suficientemente cercana a este. En muchos casos, cada evaluación de aptitud puede ser costosa en términos de tiempo y recursos computacionales, especialmente cuando la evaluación implica la simulación de modelos complejos o el entrenamiento de algoritmos de aprendizaje automático. Por ello, reducir el número de evaluaciones de aptitud es fundamental para mejorar la eficiencia del AG, sin sacrificar la calidad de las soluciones generadas. Este aspecto fue particularmente relevante en nuestra investigación, donde la evaluación de la función de aptitud implicaba el entrenamiento y validación de modelos de clasificación en conjuntos de datos de alta dimensionalidad. La técnica de paralelización y la evaluación diferencial de las soluciones fueron estrategias clave para reducir el tiempo de ejecución, aunque demandó una infraestructura computacional adecuada (más sobre esto en el próximo capítulo).

3.5. d) Operadores estocásticos y esquemas genéticos

Como hemos señalado los AGs emplean **métodos probabilísticos de transición** conformados por **operadores estocásticos**, que introducen aleatoriedad en el proceso evolutivo. Esto determina que las transformaciones dentro de un AG no siguen un camino determinista hacia la solución óptima; en su lugar, cada generación de soluciones es producto de un proceso estocástico controlado.

Los **operadores genéticos** fundamentales en este proceso son la **selección**, el **cruce (crossover)** y la **mutación**. Los mismos son responsables de la generación de nuevas soluciones, e inciden directamente en la evolución de los patrones genéticos que los AG tienden a preservar y reproducir. Patrones que se conocen como **esquemas** (1989).

Según explica Goldberg, los **esquemas** son estructuras genéticas que se repiten en la población y que influyen en la evolución de los individuos. Estos esquemas pueden ser de **orden bajo** (pocos genes) o de **orden alto** (más genes), y de **longitud de definición baja** (pocos bits) o de **longitud de definición alta** (más bits). En su operatoria, los AGs tienden a favorecer los esquemas de orden bajo y longitud de definición baja que muestran un rendimiento mejor que la media. Este fenómeno, conocido como **Teorema del Esquema**, proporciona una base para entender cómo la selección y los operadores estocásticos actúan en conjunto para guiar la evolución hacia soluciones óptimas. Veamos esto en detalle.

El operador de **selección** opera identificando y preservando los esquemas con aptitudes superiores a la media de la población. En términos probabilísticos, los esquemas con mejor aptitud tienen una mayor probabilidad de ser seleccionados y reproducidos en la siguiente generación. Esta selección basada en aptitud es clave para mantener y amplificar características beneficiosas dentro de la población. Dicho esto, la selección por sí sola no es suficiente para garantizar la exploración global del espacio de búsqueda, de ahí la importancia del cruce y la mutación.

El operador de **cruce** permite la recombinación de material genético entre dos o más soluciones. En un AG, la función principal del cruce es preservar y mejorar las características exitosas encontradas en los padres, mientras introduce suficiente variación para explorar nuevas áreas del espacio de búsqueda. Por ejemplo, en la representación binaria, un cruce de un punto dividirá dos soluciones en una posición elegida aleatoriamente y combinará segmentos de ambas para crear nuevos individuos. Este proceso asegura la transmisión de esquemas de orden bajo y longitud de definición baja, mientras introduce nuevas combinaciones genéticas que pueden llevar a soluciones más adaptativas.

Un ejemplo de cruce de un punto entre dos soluciones binarias sería:

- Padre 1: `110010`
- Padre 2: `101101`
- Punto de Cruce: `3`
- Hijo 1: `110101`
- Hijo 2: `101010`

En este caso, el cruce de un punto en la posición 3 divide los padres en dos segmentos y combina los segmentos para generar dos nuevos individuos. Este proceso de cruce permite la recombinación de material genético entre los padres, preservando y mejorando las características exitosas encontradas en ellos.

El operador de **mutación**, por su parte, introduce cambios aleatorios en las soluciones existentes, actuando como un mecanismo de perturbación que permite al AG escapar de óptimos locales y explorar más exhaustivamente el espacio de soluciones. La mutación puede variar desde simples alteraciones de bits en cadenas binarias hasta ajustes en representaciones continuas mediante la adición de ruido gaussiano. En términos del Teorema del Esquema, la mutación afecta la probabilidad de preservación de esquemas, especialmente aquellos de mayor orden, pero es crucial para asegurar que el AG mantenga la capacidad de descubrir nuevas áreas del espacio de búsqueda.

Un ejemplo de mutación en una solución binaria sería:

- Solución Original: `110010`
- Posición de Mutación: `4`
- Solución Mutada: `110110`

A esta altura ha de ser evidente que la preservación de ciertos patrones genéticos de aptitud superior es fundamental para la evolución de la población en un AG. La teoría de los esquemas, que se basa en el concepto de esquemas genéticos, proporciona un marco formal para entender cómo los operadores genéticos actúan en conjunto para guiar la evolución hacia soluciones óptimas.

Goldberg nos presenta, en relación a este punto, lo que se conoce como la **Ecuación del Esquema**, que es una herramienta teórica que permite predecir la evolución de los esquemas en una población a lo largo de múltiples generaciones. Esta ecuación tiene en cuenta factores como la aptitud de los esquemas, la probabilidad de cruce y mutación, la longitud de definición y el orden de los esquemas, y proporciona una guía para entender cómo los esquemas se propagan y se mantienen en la población.

La Ecuación del Esquema predice el número esperado de copias de un esquema H en la próxima generación $t+1$, dado su número de copias en la generación actual t. Se expresa de la siguiente manera:

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \cdot \left[1 - p_c \frac{\delta(H)}{l-1}\right] \cdot (1 - p_m)^{o(H)}$$

Donde: - $m(H,t)$ es el número de copias del esquema H en la generación t. - $f(H)$ es la aptitud promedio de los individuos que pertenecen al esquema H. - $\bar{f}$ es la aptitud promedio de la población total. - p_c es la probabilidad de cruce. - $\delta(H)$ es la longitud de definición del esquema H, que es la distancia entre el primer y el último gen fijo en el esquema. - l es la longitud del cromosoma. - p_m es la tasa de mutación. - $o(H)$ es el orden del esquema, es decir, el número de posiciones fijas en el esquema.

Consideremos un ejemplo con los siguientes parámetros:

- Longitud del cromosoma: $l = 6$
- Esquema $H = 1 * 0 * 1 *$ (donde $*$ puede ser 0 o 1)
- Población actual tiene 100 individuos.
- $m(H,t) = 20$ (es decir, 20 individuos coinciden con el esquema H).
- Aptitud promedio de la población $\bar{f} = 15$.
- Aptitud promedio de los individuos que coinciden con el esquema H, $f(H) = 18$.
- Probabilidad de cruce $p_c = 0.7$.
- Tasa de mutación $p_m = 0.01$.
- Longitud de definición del esquema $\delta(H) = 4$ (dado que las posiciones fijas son 1, 3 y 5, la distancia entre las posiciones es 4).
- Orden del esquema $o(H) = 3$ (el número de posiciones fijas es 3).

Aplicando estos valores a la Ecuación del Esquema:

1. **Factor de Selección**:

   $\frac{f(H)}{\bar{f}} = \frac{18}{15} = 1.2$

   Esto indica que los individuos que coinciden con el esquema H tienen una aptitud superior a la media y, por lo tanto, es más probable que sean seleccionados.

2. **Probabilidad de Conservación ante el Cruce**:

   $1 - p_c \frac{\delta(H)}{l-1} = 1 - 0.7 \cdot \frac{4}{6-1} = 1 - 0.7 \cdot 0.8 = 1 - 0.56 = 0.44$

   Hay un 44 % de probabilidad de que el esquema H se conserve tras el cruce.

3. **Probabilidad de Conservación ante la Mutación**:

   $(1 - p_m)^{o(H)} = (1 - 0.01)^3 = 0.99^3 \approx 0.9703$

   El esquema H tiene aproximadamente un 97 % de probabilidad de no ser destruido por la mutación.

4. **Cálculo Final**:

   $m(H, t+1) \geq 20 \cdot 1.2 \cdot 0.44 \cdot 0.9703 \approx 20 \cdot 0.5127 = 10.254$

   Por lo tanto, en la próxima generación, se espera que haya al menos 10 copias del esquema H en la población.

Este cálculo muestra cómo el esquema H, que tiene una aptitud superior a la media y ciertas características de proximidad posicional (es decir, una longitud de definición baja), es favorecido en la reproducción y es probable que se mantenga en la población.

Con los elementos vistos hasta aquí podemos pasar, en la parte final del presente capítulo, a la implementación de un Algoritmo Genético.

3.6. Implementación de un Algoritmo Genético para la selección de características

En esta sección se describe la implementación que hemos realizado de AG, usando la librería `DEAP` de Python, para la selección de características en problemas de alta dimensionalidad. La implementación se centra en la optimización simultánea de la precisión de un modelo de clasificación y la reducción del número de características seleccionadas, utilizando operadores genéticos clásicos como el cruce y la mutación, junto con técnicas avanzadas de evaluación y selección.

La configuración inicial del AG a lo largo de los distintos experimentos que formaron parte de esta investigación (y que revisaremos en detalle en el próximo capítulo) incluyó:

- **Población inicial**: individuos generados aleatoriamente, cada uno representado como una lista de bits de longitud igual al número de características.
- **Función de aptitud**: Maximización de la precisión del modelo de clasificación y minimización del número de características activas.
- **Operadores genéticos**: Selección por torneo, cruce de dos puntos y mutación por inversión de bits.

- **Parámetros del AG**: Probabilidad de mutación (e.g.`PROB_MUT = 0.1`), probabilidad de cruce (e.g. `PX = 0.75`), número máximo de generaciones (e.g.`GMAX = 15`).
- **Evaluación de características**: Análisis de la frecuencia de activación de las características a lo largo de las generaciones.
- **Criterio de terminación**: Convergencia o número máximo de generaciones alcanzado.
- **Análisis de resultados**: Selección de las características más relevantes basadas en su frecuencia de activación.

En cada experimento, la implementación del AG comienza con la definición de los componentes básicos del algoritmo. Se define una función de aptitud (`fitness`) orientada a maximizar, que evalúa cada individuo en función de dos criterios: la precisión (`accuracy`) del modelo de clasificación entrenado con las características seleccionadas, y la fracción de características activas. Esta función de aptitud está diseñada para balancear la necesidad de un modelo predictivo preciso con la simplicidad y la eficiencia del modelo, evitando el sobreajuste y facilitando la interpretación del modelo final.

Los individuos, representados como listas de bits, se construyen utilizando una función de construcción de genes que genera un bit aleatorio basado en una probabilidad definida (`p_indpb`). Estos individuos se agrupan en una población inicial, que luego se somete a un proceso evolutivo. Durante la evolución, los individuos se seleccionan mediante la técnica de torneo, donde aquellos con mejor aptitud tienen una mayor probabilidad de ser elegidos para reproducción. Los individuos seleccionados se cruzan utilizando un operador de cruce de dos puntos (`cxTwoPoint`), que intercambia segmentos de los cromosomas de los padres para generar descendientes con combinaciones genéticas novedosas. Posteriormente, se aplica un operador de mutación que invierte los bits en el cromosoma según la probabilidad de mutación definida, asegurando que el AG mantenga la capacidad de explorar nuevas regiones del espacio de búsqueda.

A lo largo de las generaciones, el AG monitoriza y registra diversas estadísticas de la población, como la aptitud promedio, la precisión y el número de genes activos. Estas métricas permiten evaluar el progreso del algoritmo y la convergencia hacia soluciones óptimas. Al final del proceso evolutivo, se realiza un análisis de las características seleccionadas, calculando la frecuencia de activación de cada característica a lo largo de las generaciones y seleccionando las más recurrentes como las más relevantes. Este enfoque permite identificar un subconjunto óptimo de características que no solo maximiza la precisión del modelo, sino que también minimiza su complejidad.

A continuación, y con fines ilustrativos, se presenta un script genérico de un Algoritmo Genético implementado con la librería `DEAP` de Python, que puede ser adaptado para la selección de características en problemas de alta dimensionalidad. Este script incluye la definición de los componentes básicos del AG que vimos antes, como la función de aptitud, los operadores genéticos y los parámetros del algoritmo, así como la configuración de la población inicial y la ejecución del proceso evolutivo a lo largo de 15 generaciones.

```
import random
import numpy as np
from deap import base, creator, tools
from sklearn.datasets import load_breast_cancer
```

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Parámetros del Algoritmo Genético
PROB_MUT = 0.1        # Probabilidad de mutación
PX = 0.75             # Probabilidad de cruce
GMAX = 15             # Número máximo de generaciones
POP_SIZE = 20         # Tamaño de la población

# Carga del conjunto de datos de ejemplo
data = load_breast_cancer()
X = data.data
y = data.target

# División del conjunto de datos en entrenamiento y prueba
Xtrain, Xtest, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Tamaños derivados
DAT_SIZE = Xtrain.shape[0]
IND_SIZE = Xtrain.shape[1]
PM = 1 / IND_SIZE    # Probabilidad de mutación por gen

# Definición de la función de fitness
def fitness(individual, Xtrain, Xtest, y_train, y_test):
    """Función de aptitud para evaluar la calidad de un individuo."""
    if not any(individual):
        return 0,  # Evita seleccionar individuos sin genes activos

    X_train = Xtrain[:, individual]
    X_test = Xtest[:, individual]

    model = MLPClassifier(hidden_layer_sizes=(5, 3), max_iter=1000, random_state=42)
    model.fit(X_train, y_train)

    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)

    # Minimización del número de características seleccionadas
    n_genes = np.sum(individual)
    alpha = 0.5  # Ponderación entre precisión y número de genes

    return alpha * accuracy + (1 - alpha) * (1 - n_genes / IND_SIZE),

# Configuración del entorno evolutivo
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", lambda: random.random() < PM)
```

```
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_b
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=PROB_MUT)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", fitness, Xtrain=Xtrain, Xtest=Xtest, y_train=y_train,

# Función principal del Algoritmo Genético
def main():
    # Inicialización de la población
    population = toolbox.population(n=POP_SIZE)

    # Evaluación inicial
    fitnesses = list(map(toolbox.evaluate, population))
    for ind, fit in zip(population, fitnesses):
        ind.fitness.values = fit

    # Bucle evolutivo
    for gen in range(GMAX):
        # Selección y reproducción
        offspring = toolbox.select(population, len(population))
        offspring = list(map(toolbox.clone, offspring))

        # Aplicación del cruce y mutación
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < PX:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < PROB_MUT:
                toolbox.mutate(mutant)
                del mutant.fitness.values

        # Evaluación de los nuevos individuos
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit

        # Reemplazo de la población
        population[:] = offspring

        # Recopilación de estadísticas
        fits = [ind.fitness.values[0] for ind in population]
        print(f"Generación {gen + 1} - Mejor fitness: {max(fits):.4f} - Promedio fi
```

```
    # Mejor individuo al finalizar
    best_ind = tools.selBest(population, 1)[0]
    print("\nMejor individuo encontrado: ", best_ind)
    print(f"Fitness: {best_ind.fitness.values[0]:.4f}")
    print(f"Número de características seleccionadas: {np.sum(best_ind)}")

if __name__ == "__main__":
    main()
```

Capítulo 4

Experimentos realizados y sus resultados

Aquí digo algo importante.

Capítulo 5

Próximos Pasos

Aquí digo algo importante.

References

Ai, Qingzhong, Pengyun Wang, Lirong He, Liangjian Wen, Lujia Pan, and Zenglin Xu. 2023. "Generative Oversampling for Imbalanced Data via Majority-Guided VAE." February 14, 2023. http://arxiv.org/abs/2302.10910.

Blaauw, Merlijn, and Jordi Bonada. 2016. "Modeling and Transforming Speech Using Variational Autoencoders." In *Interspeech 2016*, 1770–74. ISCA. https://doi.org/10.21437/Interspeech.2016-1183.

Blagus, Rok, and Lara Lusa. 2013. "SMOTE for High-Dimensional Class-Imbalanced Data." *BMC Bioinformatics* 14 (1): 106. https://doi.org/10.1186/1471-2105-14-106.

Doersch, Carl. 2021. "Tutorial on Variational Autoencoders." January 3, 2021. http://arxiv.org/abs/1606.05908.

Fajardo, Val Andrei, David Findlay, Charu Jaiswal, Xinshang Yin, Roshanak Houmanfar, Honglei Xie, Jiaxi Liang, Xichen She, and D. B. Emerson. 2021. "On Oversampling Imbalanced Data with Deep Conditional Generative Models." *Expert Systems with Applications* 169 (May): 114463. https://doi.org/10.1016/j.eswa.2020.114463.

Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York, NY, USA: Addison-Wesley.

Golub, T. R., D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, et al. 1999. "Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring." *Science* 286 (5439): 531–37. https://doi.org/10.1126/science.286.5439.531.

Isabelle Guyon, Steve Gunn. 2004. "Gisette." UCI Machine Learning Repository. https://doi.org/10.24432/C5HP5B.

Jiao, Ruwang, Bach Hoai Nguyen, Bing Xue, and Mengjie Zhang. 2023. "A Survey on Evolutionary Multiobjective Feature Selection in Classification: Approaches, Applications, and Challenges." *IEEE Transactions on Evolutionary Computation*, 1–1. https://doi.org/10.1109/TEVC.2023.3292527.

Khmaissia, Fadoua, and Hichem Frigui. 2023. "Confidence-Guided Data Augmentation for Improved Semi-Supervised Training." February 21, 2023. http://arxiv.org/abs/2209.08174.

Kingma, Diederik P., and Max Welling. 2019. "An Introduction to Variational Autoencoders." *Foundations and Trends® in Machine Learning* 12 (4): 307–92. https://doi.org/10.1561/2200000056.

Kirkpatrick, James, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, et al. 2017. "Overcoming Catastrophic Forgetting in Neural Networks." *Proceedings of the National Academy of Sciences* 114 (13): 3521–26. https://doi.org/10.1073/pnas.1611835114.

Kwarciak, Kamil, and Marek Wodzinski. 2023. "Deep Generative Networks for Heterogeneous Augmentation of Cranial Defects." August 9, 2023. http://arxiv.org/abs/2308.04883.

Latif, Siddique, Rajib Rana, Junaid Qadir, and Julien Epps. 2020. "Variational Autoencoders for Learning Latent Representations of Speech Emotion: A Preliminary

Study." July 27, 2020. https://doi.org/10.48550/arXiv.1712.08708.

Lecun, Yann. 1998. "Gradient-Based Learning Applied to Document Recognition." *PROCEEDINGS OF THE IEEE* 86 (11).

Leelarathna, Navindu, Andrei Margeloiu, Mateja Jamnik, and Nikola Simidjievski. 2023. "Enhancing Representation Learning on High-Dimensional, Small-Size Tabular Data: A Divide and Conquer Method with Ensembled VAEs." June 27, 2023. http://arxiv.org/abs/2306.15661.

Liu, Qi, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. 2018. "Constrained Graph Variational Autoencoders for Molecule Design." In *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/hash/b8a03c5c15fcfa8dae0b03351eb1742f-Abstract.html.

Martins, Miguel, Miguel Rocha, and Vítor Pereira. 2022. "Variational Autoencoders and Evolutionary Algorithms for Targeted Novel Enzyme Design." In *2022 IEEE Congress on Evolutionary Computation (CEC)*, 1–8. Padua, Italy: IEEE Press. https://doi.org/10.1109/CEC55065.2022.9870421.

Ramaswamy, Sridhar, Pablo Tamayo, Ryan Rifkin, Sayan Mukherjee, Chen-Hsiang Yeang, Michael Angelo, Christine Ladd, et al. 2001. "Multiclass Cancer Diagnosis Using Tumor Gene Expression Signatures." *Proceedings of the National Academy of Sciences* 98 (26): 15149–54. https://doi.org/10.1073/pnas.211566398.

Ramchandran, Siddharth, Gleb Tikhonov, Otto Lönnroth, Pekka Tiikkainen, and Harri Lähdesmäki. 2022. "Learning Conditional Variational Autoencoders with Missing Covariates." March 2, 2022. http://arxiv.org/abs/2203.01218.

Roberts, Adam, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. 2019. "A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music." November 11, 2019. http://arxiv.org/abs/1803.05428.

Torre, Jordi de la. 2023. "Autocodificadores Variacionales (VAE) Fundamentos Te\'oricos y Aplicaciones." February 18, 2023. https://doi.org/10.48550/arXiv.2302.09363.

Vie, Aymeric, Alissa M. Kleinnijenhuis, and Doyne J. Farmer. 2021. "Qualities, Challenges and Future of Genetic Algorithms: A Literature Review." September 13, 2021. http://arxiv.org/abs/2011.05277.

Wu, Zhangkai, Longbing Cao, and Lei Qi. 2023. "eVAE: Evolutionary Variational Autoencoder." January 1, 2023. https://doi.org/10.48550/arXiv.2301.00011.

Xiao, Han, Kashif Rasul, and Roland Vollgraf. 2017. "Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms." September 15, 2017. https://doi.org/10.48550/arXiv.1708.07747.

Zagoruyko, Sergey, and Nikos Komodakis. 2017. "Wide Residual Networks." June 14, 2017. https://doi.org/10.48550/arXiv.1605.07146.

Zhang, Yuchi, Yongliang Wang, Liping Zhang, Zhiqiang Zhang, and Kun Gai. 2019. "Improve Diverse Text Generation by Self Labeling Conditional Variational Auto Encoder." March 26, 2019. https://doi.org/10.48550/arXiv.1903.10842.

Apéndice A

Frequently Asked Questions

A.1. How do I change the colors of links?

Pass in `urlcolor:` in yaml. Or set these in the include-in-header file.

If you want to completely hide the links, you can use:

{`\hypersetup{allcolors=.}`}, or even better:

{`\hypersetup{hidelinks}`}.

If you want to have obvious links in the PDF but not the printed text, use:

{`\hypersetup{colorlinks=false}`}.