Informe del proyecto final de Ingeniería de Software Avanzada

-

Calidad de datos y Big Data en Ingeniería de Software

Tutora:
Adriana Marotta

Estudiante:
Andrés Veiro

Introducción

El objetivo de este trabajo es llevar a cabo una lectura exhaustiva del artículo titulado "Handling incomplete heterogeneous data using VAEs"[1]. Este artículo, el último de una serie publicada por los autores, aborda el manejo de datos faltantes y heterogéneos utilizando la técnica del Variational Autoencoder (VAE). Los VAE pertenecen a las técnicas generativas no supervisadas de aprendizaje profundo basadas en autoencoders.

En general, los modelos generativos son capaces de aprender la distribución de los datos y la estructura latente de grandes conjuntos de datos, lo que les permite generar nuevos datos basados en entradas específicas. Específicamente, los VAE son eficientes en la captura de la estructura latente en grandes conjuntos de datos de alta dimensionalidad. Sin embargo, no pueden manejar directamente datos heterogéneos (continuos y discretos) ni datos incompletos.

Los autores proponen un enfoque que utiliza modelos probabilísticos para manejar los diferentes tipos de datos posibles, lo que permite realizar estimaciones precisas para la imputación de datos. Además, el modelo propuesto es semi-supervisado, teniendo en cuenta los diferentes tipos de datos y la presencia de valores faltantes.

Definición de conceptos

Los conceptos principales para entender la propuesta son el autoencoder y el VAE.

El método fue propuesto en 2006, como se menciona en la referencia [2]. Se trata de un método de aprendizaje automático no supervisado en el cual, dado un dato de entrada X, se obtiene una salida X' que es muy similar a X.

Desde el punto de vista de su estructura, los autoencoders constan de dos redes neuronales. La primera se denomina "Encoder" y toma la entrada X para generar una representación Z en un espacio latente. La segunda red neuronal se llama "Decoder" y utiliza la salida Z del "Encoder" para generar la salida X'. En la Figura 1 se muestra un ejemplo de esto.

El objetivo del autoencoder no es que la entrada y la salida sean idénticas. En cambio, se entrena la red para que aprenda la estructura de los datos y pueda generar nuevos datos basados en una entrada, siguiendo la estructura de los datos de entrenamiento.

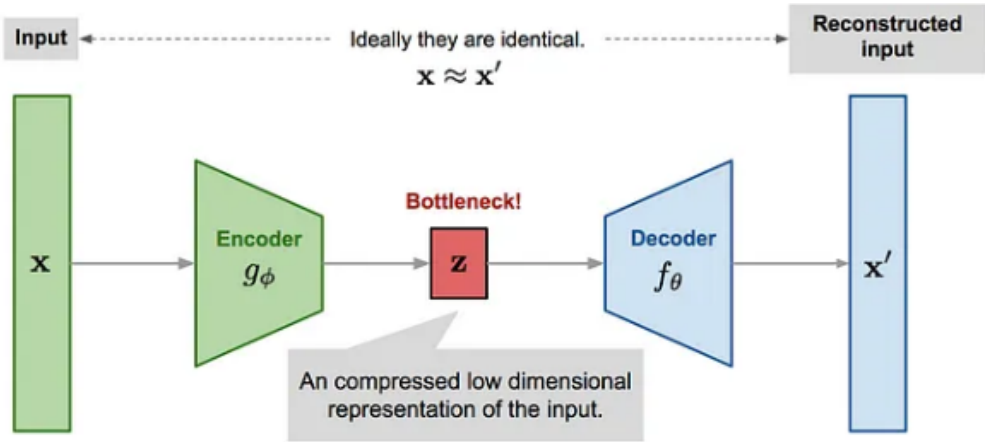

Figura 1: Ejemplo de autoencoder, tomada de [3].

Los autoencoders han evolucionado a medida que se han abordado algunos problemas que afectan a este tipo de algoritmos.

En 2013, como se menciona en la referencia [4], se introdujeron los VAE (Variational Autoencoders). Según la referencia [5], el problema que se busca resolver en los autoencoders es que el espacio latente tiende a formar agrupaciones basadas en las entradas. Por ejemplo, los números '1' y '7' pueden tener representaciones agrupadas, mientras que el número '9' puede estar separado en otra agrupación. Los espacios de separación entre las agrupaciones pueden generar datos que no forman parte del entrenamiento. Para poder generar datos a partir de los espacios intermedios, se propone un enfoque probabilístico. El espacio latente se representa mediante dos vectores de la misma dimensión que representan la media y la desviación estándar. Al considerarlos conjuntamente, se obtiene un vector de variables aleatorias con una distribución normal.

En la Figura 2 se muestra un ejemplo de la arquitectura de este modelo.

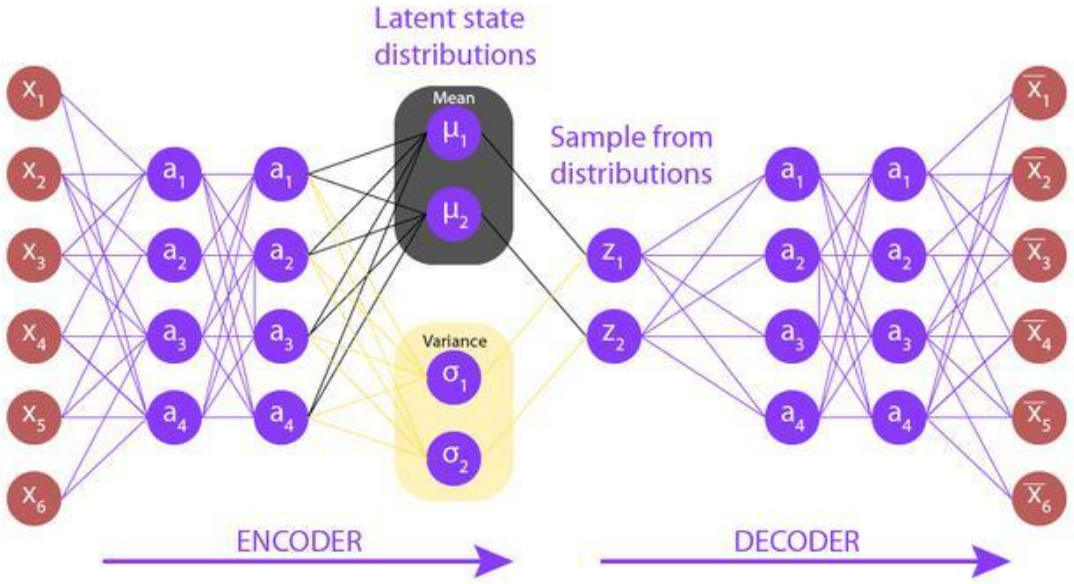

Figura 2: Ejemplo de VAE. Tomado de [6].

Análisis de propuesta del artículo

El artículo analizado destaca en su estado del arte que otros artículos relacionados con la utilización de modelos generativos buscan enriquecer las entradas o salidas de los datos. También explica que existen muchas variaciones de VAE y otros modelos generativos para resolver problemas en conjuntos de datos arbitrarios. Sin embargo, hasta el momento de la publicación del artículo, no se ha abordado de manera exhaustiva cómo entrenar modelos generativos con datos faltantes.

El modelo generativo propuesto es capaz de manejar datos reales continuos y discretos, así como datos nominales (categóricos y ordinales). Utiliza modelos de probabilidad parametrizados con una Red Neuronal Profunda (DNN) para capturar la estructura y las relaciones en los datos.
Una característica destacada del modelo es su capacidad para realizar imputación de datos en presencia de valores faltantes, asumiendo la hipótesis de datos faltantes completamente al azar (MCAR). Esto significa que el modelo puede inferir y completar los valores faltantes en función de los datos observados.
Además, se realiza la normalización de las entradas para evitar que una dimensión específica domine el entrenamiento. Esto asegura que todas las dimensiones de los datos tengan un impacto equilibrado en el modelo.
Para optimizar los parámetros de los modelos generativos, se utiliza una cota inferior de evidencia (ELBO). Esta cota se calcula utilizando los datos observados y permite ajustar los parámetros de manera eficiente durante el proceso de entrenamiento del modelo generativo.

Para entender mejor la propuesta, se realizaron tres diagramas. En la Figura 3 se muestra el "Encoder", en la Figura 4 se muestra el "Decoder", y en la Figura 5 se presentan ambos juntos. Estos diagramas utilizan las mismas dimensiones que se emplearon en el caso ejecutado en la sección de Análisis de código.

El "Encoder" de la Figura 3, recibe como entrada un vector "X" y establece en cero los atributos faltantes. Luego, genera un vector "X´" de la misma dimensión que el vector de entrada.
A continuación, los atributos de "X´" los normaliza en base al tipo de dato: continuo, nominal, etc. Generando un vector de mismo largo normalizado “X´´”.
Luego aplica una capa que tiene cómo salida una representación multinomial de dimensión 10. Desde el punto de vista de la implementación es una capa densa de la red neuronal con una salida softmax. En el artículo, la salida de esa capa lo llama “S”
Con “X´´”, “S” cómo entradas genera una representación de dimensión 2x2 aplicando un Modelo de Mezcla de Gaussianas [13]. Desde el punto de vista de la implementación son dos capas densas. Esto genera la salida del “Encoder” que en artículo la denomina espacio latente “Z(u,o)”, siendo la distribución de probabilidad comentada en la sección de conceptos de los VAE.

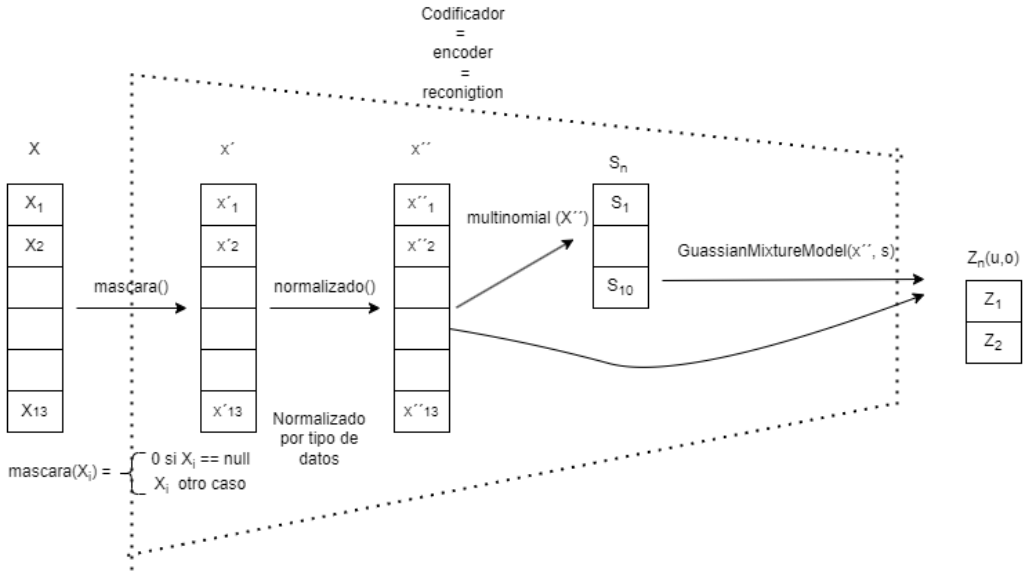

Figura 3: Encoder de “Handling incomplete heterogeneous data using VAEs” [1].

Por otro lado, en la Figura 4, el "Decoder" toma como entrada el espacio latente "Z" y la representación "S" para generar una representación intermedia utilizando una capa densa en la red neuronal. Esta capa tiene como salida un vector de longitud 5, denominado "Y".

En base a “Y” y “S” genera el vector “X” de largo 13, este vector es la salida del “Decoder”. Para generar la salida, utiliza distintas distribuciones de probabilidad en base a qué tipo de atributo de “X” quiere generar cómo se ve en la Figura 4.

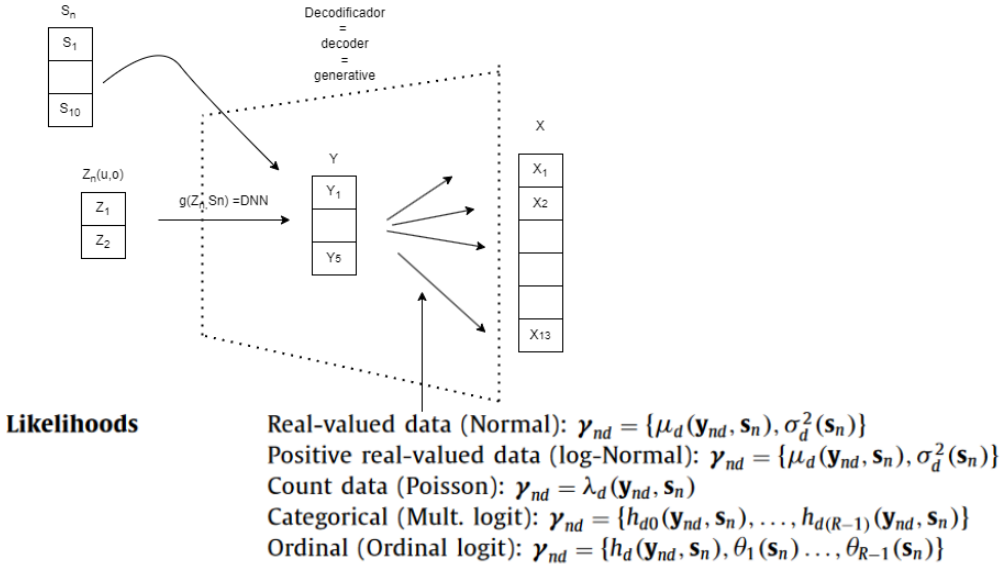

Figura 4: Decoder de “Handling incomplete heterogeneous data using VAEs” [1].

En la Figura 5, se observa cómo es la interacción entre el “Decoder” y “Encoder”. Desde el punto de vista del entrenamiento, todas las capas que se mencionaron se entrenan al mismo tiempo con los mismos datos.

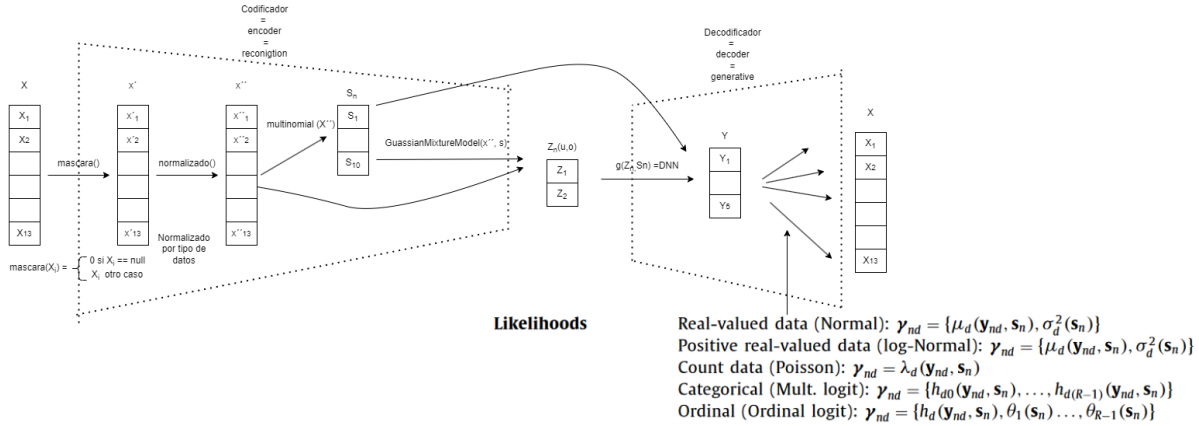

Figura 5: “Encoder” y “Decoder” de “Handling incomplete heterogeneous data using VAEs” [1].

Experimentos realizados

Se realizaron pruebas utilizando seis conjuntos de datos del repositorio UCI [7]. Para cada conjunto de datos, se generaron series de datos que representan los índices de los valores faltantes para diferentes porcentajes: 10%, 20%, 30%, 40% y 50%. En total, se llevaron a cabo 20 pruebas para evaluar el desempeño del modelo generativo propuesto en diferentes escenarios de datos faltantes.

Para comprender mejor los datos utilizados, se presenta el caso del conjunto de datos Wine [9], el cual también se utiliza en la siguiente sección de Análisis de código.
En la Figura 6 se muestra el tipo de dato de cada columna, mientras que en la Figura 7 se muestran los datos contenidos en el conjunto de datos. Estas figuras proporcionan información detallada sobre la estructura y contenido del conjunto de datos Wine.

	type	dim	nclass
2	real	1	
3	real	1	
4	real	1	
5	real	1	
6	real	1	
7	real	1	
8	real	1	
9	real	1	
10	real	1	
11	real	1	
12	real	1	
13	real	1	
14	cat	2	2

Figura 6: Tipos de datos de Wine [9]

1	7	0.27	0.36	20.7	0.045	45	170	1.001	3	0.45	8.8	6	2
2	6.3	0.3	0.34	1.6	0.049	14	132	0.994	3.3	0.49	9.5	6	2
3	8.1	0.28	0.4	6.9	0.05	30	97	0.9951	3.26	0.44	10.1	6	2
4	7.2	0.23	0.32	8.5	0.058	47	186	0.9956	3.19	0.4	9.9	6	2

Figura 7: Juego de datos Wine [9]

Los resultados obtenidos fueron comparados con otros algoritmos, incluyendo el método mean (promedio), MICE, GAIN y GLFM. A continuación, se explica brevemente cada uno de estos métodos.

El método MICE [11] reemplaza los valores faltantes con un valor inicial, cómo por ejemplo cero, y luego imputa los valores faltantes de un atributo seleccionado utilizando el conjunto de datos completo y alguna función relevante, como la regresión lineal. Este proceso se repite para cada atributo hasta que los resultados convergen.
El método GAIN [12] utiliza una Red Generativa Adversarial (GAN) para la imputación de datos faltantes. La GAN se entrena para generar datos que sean coherentes con los datos observados y se utiliza para imputar los valores faltantes.
Por último, General Latent Feature Modeling for Heterogeneous data (GLFM) [10] es un algoritmo que permite modelar datos heterogéneos en un espacio latente utilizando distribuciones probabilísticas. Luego, estas distribuciones se utilizan para generar datos que se utilizan en la imputación de valores faltantes.
Estos métodos se compararon con el enfoque propuesto en términos de su desempeño en la imputación de datos faltantes.

En el artículo se detallan los parámetros utilizados en la red neuronal del modelo. Estos parámetros se refieren a las dimensiones de las variables X, S y Y, que fueron determinadas a través de pruebas experimentales.

En los resultados, se destaca el caso del conjunto de datos "Wine numeric", donde el modelo propuesto muestra resultados similares a los obtenidos en la ejecución del código. En la Figura 8 se pueden observar los resultados, donde se resalta en amarillo el valor publicado en el artículo y que se compara en la siguiente sección de Análisis de código.

Table 3
Imputation error. Average and standard deviation of the imputation error for a 20% of missing data, evaluated exclusively over **numeric variables**.

Model	Adult	Breast	DefaultCredit	Letter	Spam	Wine
HI-VAE (no norm.)	0.210 ± 0.028	–	*Inf*	–	0.054 ± 0.018	0.165 ± 0.042
HI-VAE	0.106 ± 0.002	–	0.043 ± 0.001	–	0.052 ± 0.001	**0.074 ± 0.001**
Mean imputation	0.111 ± 0.002	–	0.056 ± 0.001	–	0.053 ± 0.001	0.103 ± 0.002
MICE	0.108 ± 0.002	–	**0.035 ± 0.002**	–	0.052 ± 0.003	0.074 ± 0.002
GLFM	**0.083 ± 0.001**	–	0.051 ± 0.005	–	0.052 ± 0.001	0.082 ± 0.004
GAIN	0.225 ± 0.192	–	0.044 ± 0.002	–	**0.049 ± 0.001**	0.086 ± 0.002

Table 4
Imputation error. Average and standard deviation of the imputation error for a 20% of missing data, evaluated exclusively over **nominal variables**.

Model	Adult	Breast	DefaultCredit	Letter	Spam	Wine
HI-VAE (no norm.)	0.406 ± 0.005	–	0.202 ± 0.003	–	0.166 ± 0.019	0.245 ± 0.017
HI-VAE	**0.304 ± 0.006**	0.112 ± 0.003	**0.158 ± 0.001**	**0.105 ± 0.002**	**0.111 ± 0.009**	0.016 ± 0.003
Mean imputation	0.405 ± 0.002	0.211 ± 0.006	0.2 ± 0.001	0.162 ± 0.002	0.393 ± 0.014	0.248 ± 0.014
MICE	0.601 ± 0.002	0.111 ± 0.002	0.163 ± 0.003	0.133 ± 0.0	0.168 ± 0.012	0.02 ± 0.004
GLFM	0.407 ± 0.003	**0.076 ± 0.003**	0.236 ± 0.012	0.161 ± 0.001	0.154 ± 0.02	**0.006 ± 0.001**
GAIN	0.66 ± 0.025	0.16 ± 0.009	0.211 ± 0.005	0.164 ± 0.001	0.276 ± 0.017	0.236 ± 0.014

Figura 8: Resultados obtenidos de experimentación de HI-VAE tomados de [1].

Análisis del código

En [1], se proporcionan enlaces al código original. Sin embargo, al seguir las instrucciones de instalación, se encontraron problemas de compatibilidad de versiones al intentar instalar las librerías requeridas, especialmente TensorFlow. Esto se debe a que el código no ha sido actualizado en los últimos 4 años.

Para poder hacer que el código funcione, se realizaron actualizaciones en todas las dependencias. En algunos casos, los cambios en las versiones causaron problemas de ejecución, principalmente debido a cambios en las ubicaciones de clases y métodos en las actualizaciones.

El código actualizado y funcional se encuentra disponible en [8]. Este código modificado y actualizado ejecuta con las versiones más recientes de las librerías y soluciona los problemas encontrados al intentar ejecutar el código original.

En la Figura 9 se muestra una captura del código ejecutado y el resultado obtenido para el dataset Wine. En este caso, se utilizó el 20% de los datos faltantes. El resultado obtenido coincide con los resultados publicados en el artículo.
Durante la ejecución, se obtuvo un error de 0.082, que representa el error en todos los atributos. En los resultados publicados en la Figura 8, se obtuvo un error de 0.074±0.001 para los atributos numéricos y un error de 0.016±0.003 para los atributos nominales, lo que suma un total de 0.090±0.003. La diferencia entre los resultados obtenidos y los publicados es despreciable y se atribuye a la naturaleza aleatoria de la solución.

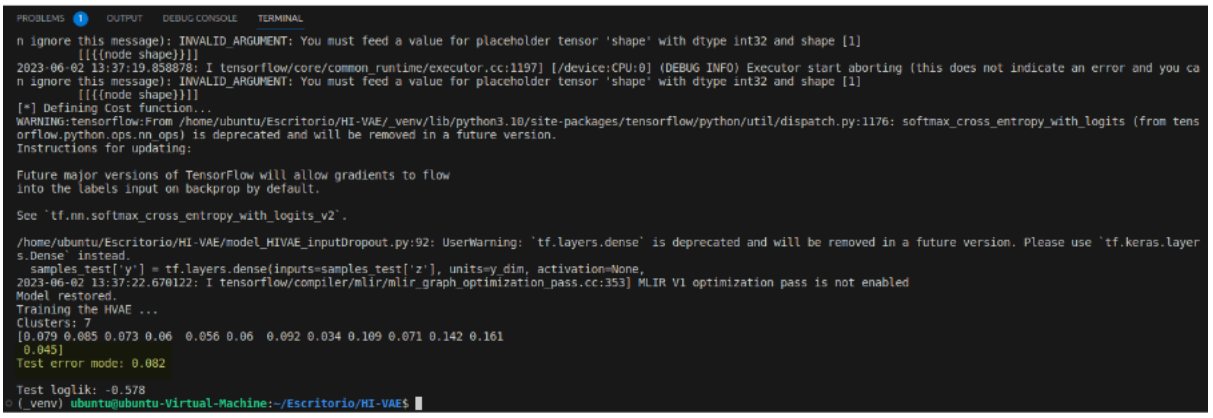

Figura 9: Ejecución de código para Wine con 20% de datos faltantes.

Conclusiones

El artículo cuenta con 137 citas, una cifra interesante aunque inferior a los 27,000 del artículo que introdujo los VAE. Esto indica que el artículo es relevante en el ámbito académico. El código original funcionó según lo esperado, pero se encontraron problemas de compatibilidad al intentar reproducir los resultados debido a las versiones de las librerías. Se destaca una actualización realizada por otros investigadores, pero no se profundizó en ella.

En general, se considera que el artículo es interesante y plantea un enfoque novedoso. Surgieron dudas sobre si se requiere un mayor conocimiento en estadística para aportar al desarrollo de este tipo de algoritmos.

El artículo no aborda la imputación de datos como un tema relacionado a alguna metodología de calidad de los datos.

La formulación formal del problema y la propuesta pueden resultar complejas, pero es posible seguirlas en su mayoría.
La propuesta presentada en el artículo es compleja debido a la naturaleza del problema que aborda. La implementación de las pruebas de concepto también es compleja, en parte debido a las diferentes pruebas mencionadas en la sección de experimentos.

En conclusión, este artículo es interesante a nivel académico y plantea un enfoque novedoso en el manejo de datos faltantes y heterogéneos. Representa una contribución significativa al campo de investigación.

Referencias

[1] A. Nazábal, P. M. Olmos, Z. Ghahramani, and I. Valera, ‘Handling incomplete heterogeneous data using VAEs’, Pattern Recognition, vol. 107, p. 107501, 2020.

[2] G. E. Hinton and R. R. Salakhutdinov, ‘Reducing the Dimensionality of Data with Neural Networks’, Science, vol. 313, no. 5786, pp. 504–507, 2006.

[3] ‘Mathematical Prerequisites For Understanding Autoencoders and Variational Autoencoders (VAEs): Beginner Friendly, Intermediate Exciting, and Expert Refreshing. | by Victor E. Irekponor | Analytics Vidhya | Medium’. [Online]. Available: https://medium.com/analytics-vidhya/mathematical-prerequisites-for-understanding-autoencoders-and-variational-autoencoders-vaes-8f854025390e.

[4] D. P. Kingma and M. Welling, ‘Auto-Encoding Variational Bayes’, arXiv [stat.ML]. 2022.

[5] ‘Variational AutoEncoder’. | by Fernando Sancho Caparrini. [Online]. Available: http://www.cs.us.es/~fsancho/?e=232.

[6] ‘Variational AutoEncoders - GeeksforGeeks’. [Online]. https://www.geeksforgeeks.org/variational-autoencoders/.

[7] ‘UCI Machine Learning Repository’. [Online]. Available: https://archive.ics.uci.edu/ml/index.php.

[8] Código fuente actualizado de HIVAE. 2023. Available: https://github.com/veiro/HI-VAE.

[9] ‘UC Irvine Machine Learning Repository’. [Online]. Available: https://archive-beta.ics.uci.edu/dataset/109/wine.

[10] I. Valera, M. F. Pradier, M. Lomeli and Z. Ghahramani,
"General Latent Feature Model for Heterogeneous Datasets", 2017.
Available on ArXiv: https://arxiv.org/abs/1706.03779.

[11] S. van Buuren and K. Groothuis-Oudshoorn, ‘mice: Multivariate Imputation by Chained Equations in R’, Journal of Statistical Software, vol. 45, no. 3, pp. 1–67, 2011.

[12] J. Yoon, J. Jordon, and M. van der Schaar, ‘GAIN: Missing Data Imputation using Generative Adversarial Nets’, arXiv [cs.LG]. 2018.

[13] S. van Buuren and K. Groothuis-Oudshoorn, ‘mice: Multivariate Imputation by Chained Equations in R’, Journal of Statistical Software, vol. 45, no. 3, pp. 1–67, 2011.