CT-GAN para imputación de datos

Andrés Veiro
Universidad de la República
Facultad de Ingeniería
Email: aveiro@fing.edu.uy

Sergio Nesmachnow
Universidad de la República
Facultad de Ingeniería
Email: sergion@fing.edu.uy

Jamal Toutouh El Alamin
Universidad de Malaga
Departamento de Lenguajes y
Ciencias de la Computación
Email: jamal@uma.es

***Abstract*—Se muestra que utilizar las técnicas de generación de datos sintéticos tabulares CTGAN mejora la imputación de datos con técnicas basadas en GANs.**

1. Introducción

En este trabajo se hace un repaso de la literatura relacionada al tema imputar datos con GANs. Se extiende la implementación hecha en [1] usando CTGAN [2]. Se comparan los resultados obtenidos con los resultados de los autores de [1].

2. Descripción del problema

Los datos faltantes son un problema al utilizar datos para tomar decisiones de negocio o para el uso de técnicas de aprendizaje automático. Para mitigar este problema hay muchas técnicas en la literatura. Las técnicas van desde lo básico, por ejemplo: borrar las tuplas que contienen datos vacíos. Pero también hay técnicas avanzadas que predicen el valor que debería llevar. Entre las técnicas avanzadas en los últimos años el uso de GANs ha demostrado buen rendimiento.

3. Marco teórico

En está sección se presentan los principales conceptos teóricos necesarios para poder entender las secciones siguientes.

3.1. Redes generativas adversarias

También conocido como GAN. El artículo que propone el método es del 2004 [3]. Es un caso de aprendizaje no supervisado. Se trata de dos redes profundas, la primera red es G (Generador) que intenta capturar la distribución de los datos y la segunda red es D (Discriminador) que estima la probabilidad que la muestra venga de los datos de entrenamiento en lugar de G. En la Figura 1 queda más clara la definición.

En la Figura 1 el conjunto de datos de imágenes reales es un conjunto de imágenes de rostros de estrellas de Hollywood.

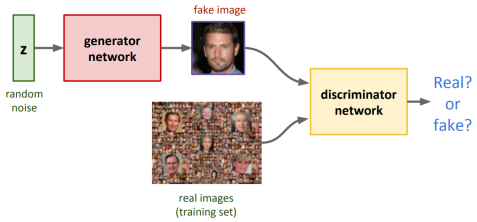

Figure 1. Modelo de arquitectura de GAN tomado de [4]

El modelo G tiene como entrada ruido aleatorio que lo llama Z, también llamado espacio latente. Genera en este caso una imagen que es un rostro de una estrella de Hollywood falso. El rostro falso y el conjunto de datos de entrenamiento son entrada del Discriminador D. Como salida de D se obtiene si el rostro es verdadero o falso.

El entrenamiento del Discriminador tiene como objetivo maximizar la probabilidad de que una entrada del conjunto de datos X sea clasificada como real. Pero también D minimiza la probabilidad de que una muestra G(Z) = X' generada por G sea clasificada como real.

Por otro lado, G es entrenado para burlar al Discriminador generando datos lo más cercano a la realidad. Entonces el objetivo es maximizar la probabilidad de que D clasifique una muestra falsa G(Z) = X' como verdadera.

Entrenar una GAN es un problema de optimización del tipo min-max. En la práctica se utiliza el logaritmo de la probabilidad como función de perdida:

$$min_G max_D V(D,G) = E[logD(x)] + E[log(1 - D(G(z)))]$$

La solución es que G capture la distribución de los datos de entrenamiento y D con probabilidad 0.5 diga que es verdadera para todos los casos.

En está primera versión G y D son redes neuronales de múltiple capa y se entrenan con backpropagation según [3].

La idea conceptual del entrenamiento es:

- Entrenar el Discriminador:
  - Entrenar Discriminador con datos reales:
    - Se toma una muestra de los datos reales X.

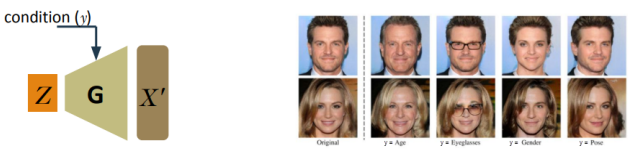

Figure 2. Ejemplo de Generador condicional, tomado de [4]

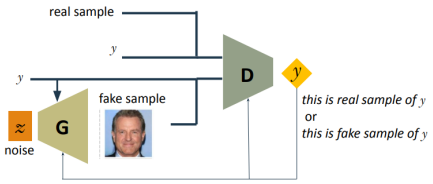

Figure 3. Modelo de arquitectura de CGAN, tomado de [4]

    - Calcular la función de perdida para la entrada X.
  - Entrenar Discriminador con datos producidos por el Generador:
    - Tomar una muestra de datos aleatorios del espacio latente Z.
    - Generar X' como salida de G para la entrada Z.
    - Calcular la función de perdida para D con la entrada X'.
  - Actualizar los pesos de D en base a las funciones de perdidas calculadas.
- Entrenar el Generador:
  - Tomar una muestra de datos aleatorios del espacio latente Z.
  - Generar X' como salida de G para la entrada Z.
  - Calcular la función de perdida para D con la entrada X'.
  - Actualizar los pesos de G en base a las funciones de perdidas calculadas.

Los detalles que explican el por qué este tipo de modelo es posible entrenar escapan el alcance de este documento. Se puede revisar el artículo [3] para obtener más información.

Un detalle interesante de las GANs es que en [5] hay un listado de variaciones de GANs que actualmente tiene 502 artículos donde cada uno explica una variación. En las siguientes secciones se comentan dos tipos de GANs interesantes: GAN condicional (CGAN) y GAN condicional tabular (CTGAN).

3.1.1. GAN Condicional. Uno de los problemas que tienen las GANs normales, vistas en la sección anterior, es generar datos con un objetivo específico. Siguiendo el ejemplo de generar rostros de famosos, visto en la Figura 1, se pretende generar rostros falsos de Hollywood con lentes. En la Figura 2 se puede seguir este ejemplo.

Para resolver este problema se presentan las GANs Condicionales. En la Figura 3 se puede ver un ejemplo de la arquitectura de CGAN.

El problema de optimización min-max explicado en la sección anterior se sigue manteniendo, pero cambian las entradas:

$$min_G max_D V(D,G) = E[logD(x,y)] + E[log(1 - D(G(z,y),y))]$$

Por más información de cómo funciona se puede ver en el artículo [6].

3.1.2. GANs Condicionales Tabulares. Otro problema de las GANs ordinarias es que, en el contexto de datos tabulares, no funcionan bien con datos discretos y continuos.

Los datos discretos pueden ser:

- Numéricos: Por ejemplo: edad o cantidades.
- Ordinales: Por ejemplo: días de la semana.
- Nominales: Por ejemplo: genero.

Los datos continuos son valores reales, por ejemplo, el precio.

CTGAN [2] es una propuesta para resolver los problemas de datos tabulares, la idea principal es hacer un preprocesamiento de las entradas antes de entrenar el modelo.

Los autores de [2] proponen normalizar los datos continuos y hacer un muestreo justo para los datos discretos.

Para realizar la normalización de los datos continuos utiliza un VGM (varational gaussian mixture model) que lo que hace es encontrar el valor K que mejor logra representar la distribución de la variable continua donde K es la cantidad de funciones Gausianas. En la Figura 4 se quiere normalizar el punto C_ij que corresponde a la propiedad C. La línea punteada azul es la distribución de la propiedad C. Utilizando VGM se aproxima con 3 Guasianas y se obtienen α y β los cuales serán la representación de C_ij.

Los datos discretos se pueden vectorizar usando la técnica one-hot-encoding. Está técnica crea vectores de largo igual a la cantidad de valores distintos que existan en la característica que se quiere codificar. Donde la j-esima entrada del vector se le asigna el valor 1 para representar el valor J y al resto de las entradas se le asgina el valor 0. Por ejemplo, los días de la semana se puede representar con un vector de largo 7, donde [1, 0, 0, 0, 0, 0, 0] es lunes y [0, 0, 0, 0, 0, 0, 1] es domingo. De todas formas, se pierde información de la distribución por lo que se agrega:

- Vector condicional: Usa la misma idea CGAN, la entrada del Generador y Discriminador recibe un vector condicional. En el caso de CTGAN se arma un vector con todas las columnas discretas en vectorizadas con la técnica one-hot-encoding. Pero solo se enciende un elemento que es el de la columna que se quiere generar. La columna elegida se realiza con el entrenamiento por muestreo.
- Entrenar por muestreo: La idea es respetar la frecuencia de las variables discretas en las entradas del

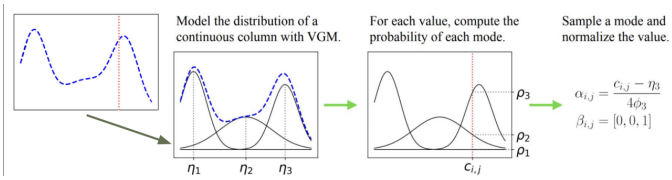

Figure 4. Ejemplo de aproximación con VGM, tomado de [7]

Generador. Se toma una columna de forma aleatoria, y se elige una categoría de esa columna en base a la frecuencia de esa categoría. Creando el vector condicional.

3.2. Dependencias entre datos faltantes

Los artículos que se usan como referencia en este trabajo parten de una de las siguientes hipótesis.

Se definen tres tipos de dependencia que explican por qué los datos faltan en relación con otras variables:

- Missing completely at random (MCAR): Sin dependencia de otras variables. Por ejemplo: Falla de equipo
- Missing at random (MAR): Depende de valores de otras columnas.
- Missing not at random (MNAR): Depende de valores que no forman parte de los datos. Por ejemplo: las personas de mayor ingreso no revelan cuánto dinero ganan. Por lo que ese valor no lo ingresan.

4. Estrategia de resolución

Los artículos que utilizan GANs para imputar datos faltantes utilizan los datos producidos por el Generador.

Los autores de [8] proponen utilizar una GAN formada por dos redes neuronales totalmente conectadas. Demuestran teóricamente que este método funciona en el caso de MCAR.

Desde el punto de vista de la arquitectura, hacen una variación en las entradas de cada red. Se define $X = (X_1, .., X_d)$ un vector, donde las componentes son discretas o continuas. Este vector es una tupla de los datos usados para el entrenamiento. También define M de la misma dimensión que X que toma valores entre $\{0, 1\}^d$; este vector funciona como máscara e indica que atributos van a ser imputados.

Define $\widetilde{X}$ como:

$$\widetilde{X} = \begin{cases} X_i, & \text{si } M_i = 1 \\ *, & \text{otro caso} \end{cases}$$

Donde $*$ representan los datos vacíos que se remplazan con ruido aleatorio. En particular, se define $Z = (Z_1, .., Z_d)$ en vector aleatorio y la componente $\widetilde{X}_i$ con valor $*$ es remplazada por Z_i.

Entonces $\widetilde{X}, M, Z$ son entradas del Generador por lo tanto se define $\vec{X} = G(\widetilde{X}, M, (1 - M) \odot Z)$ donde $\vec{X}$ es la estimación generada de X. De $\vec{X}$ son interesantes los valores donde $\widetilde{X}$ es *.

Como resultado final, el vector con los datos imputados es:

$$\hat{X} = M \odot \widetilde{X} + (1 - M) \odot \vec{X}$$

Por otro lado, el Discriminador intenta recuperar la verdadera M desde el vector completo $\hat{X}$ prediciendo la probabilidad de que cada $\hat{X}_i$ sea real o imputado. Entonces $\hat{M} = D(\hat{X})$.

El objetivo del par Generador - Discriminador es minimizar la distancia entre M y $\hat{M}$, por lo que el problema de optimización min-max queda de la siguiente forma:

$$min_G max_D V(D, G) = E[\sum_{i=1}^{n} (M_i \log(\hat{M}_i) + (1 - M_i) \log(1 - \hat{M}_i))]$$

Por otro lado, los autores de [1] se basan en la propuesta de [8] pero en lugar de utilizar una GAN ordinaria, utiliza CGANs (GAN Condicional). Éstas permiten un mayor control de los datos generados dado que condicionan la entrada del Generador y Discriminador con etiquetas de clases.

CGANS son particularmente efectivas para imputar datos bajo la condición de MNCAR con distribución no estándar y además son eficientes para conjuntos de gran escala según [1].

En [1] lo aplica en el dominio de los datos para entrenar algoritmos de ranking. Se explica que CGAN aplica para los casos MAR, este caso es más general en juegos de datos reales.

Teniendo en cuenta toda la explicación comentada anteriormente del artículo [8], en el artículo [1] supone que siempre hay información auxiliar que nunca es vacía. Define Y que puede ser por ejemplo una categoría. Este atributo influye en la probabilidad de que los otros atributos sean o no vacíos. Entonces $\tilde{X} = G(\widetilde{X}, M, (1 - M) \odot Z|Y)$. Esto impacta en $\hat{X}$ que es el vector imputado y también en la salida del Discriminador $\hat{M} = D(\hat{X}|Y)$.

Por último, tomando una función L de perdida cualquiera, el problema de optimización min-max se plantea de la siguiente forma:

$$min_G max_D V(D, G) = E[L(M, D(\hat{X}|Y))]$$

Demuestran teóricamente que hay garantías de imputación para el caso de MAR y comparan está nueva estrategia en conjuntos de datos reales mostrando que mejora la propuesta de [8].

4.0.1. Enfoque utilizado. En base a todo lo presentado, el presente trabajo propone agregar el preprocesamiento de datos que se realiza en CTGAN [2] y comparar los resultados que obtuvieron los autores de [1] usando el mismo juego de datos, haciendo las mismas pruebas y midiendo las mismas métricas.

Para hacer el preprocesamiento de datos, se utiliza la clase DataTransformer de CTGAN [1] que presenta los métodos para transformar los datos a partir de un dataframe en una representación VGM de los datos continuos y una representación one hot encoding para los datos discretos.

En el artículo [1] la representación utilizada de las entradas son los valores normalizados entre (0 ,1). Por lo tanto, cada tupla estaba formada por 5 elementos. En la transformación de CTGAN paso a estar formada por 72 elementos.

Siguiendo con la transformaciones, en el caso de [1] utiliza la GAN Condicional donde la condición es una tupla categórica que nunca puede ser vacía, para utilizar la fila Category los autores utilizaron one hot encoding que generaba vectores de 5 elementos. Utilizando la técnica de muestreo de CTGAN genero un vector de 31 valores.

5. Evaluación experimental

El código de este trabajo se encuentra en el repositorio público de github. [2]. Se partió del código publicado por [1] haciendo la modificación comentada por las transformaciones de CTGAN. Se debió modificar la estructura de la CGAN para recibir las nuevas dimensiones. También, se cambió el método para generar la máscara M. En el código de los autores de [1] se utiliza un vector aleatorio con distribución uniforme. Las entradas de la CGAN en [1] son $G(\tilde{X}, M, (1 - M) \odot Z)$ donde $\tilde{X}$ y M deben tener la misma dimensión. Si se usaba el mismo método se estaba enmáscarando parte de las representaciones hecha por DataTransformer de CTGAN, lo que no es correcto. Por lo tanto, se modificó para que enmascare toda la representación hecha por el DataTransformer de CTGAN para la propiedad que corresponde.

Siguiendo la evaluación de [1], se utilizan los mismos datos sintéticos que los autores utilizan para evaluar su método [3] El juego de datos tiene 640.000 tuplas, pero la clase DataTransformer de CTGAN generaba un error interno al utilizar más de 100.000 tuplas. Por lo que se utilizó un muestreo aleatorio.

El juego de datos está formado por cinco columnas de datos referentes a ranking de libros, que se utilizarán para probar la imputación de datos, donde cuatro de ellas son valores reales llamadas “V5”, “V6”, “V7”, “V9” y una columna categórica llamada “V8” la cual toma valores enteros del 0 al 27. Además, hay cuatro columnas adicionales llamadas “Category”, “QID”, “ProductID”, “QProductID” que son informativas. De ellas, se utiliza solo la columna “Category” que es categórica y tiene 5 valores posibles.

A continuación se cita traducido al español un párrafo de [1] que indica como es la GAN y el entrenamiento. “Se generan 10 imputaciones del conjunto de datos de ranking simulado. En TensorFlow, cada réplica de GAN se entrenó durante 50 épocas con un tamaño de minilote de 256.” “Se utiliza el optimizador de Adam (Kingma y Ba 2015) con una tasa de aprendizaje predeterminada de 0,001. No se utilizó dropout dado el tamaño moderado de los datos. Tanto el Generador como el Discriminador utilizaron una arquitectura estándar de capas totalmente conectadas con activación de leaky-relu.”

Siguiendo el marco de pruebas realizados por los autores de [1] se evalúa el método propuesto en un conjunto de datos de pruebas. Este conjunto de datos no se usó en el entrenamiento.

Se compara el dato producido por el Generador con el verdadero utilizando la medida Error Cuadrático Medio (RMSE). Se tomo está decisión dado que los autores de [1] comparan varios algoritmos usando RMSE.

Siguiendo la metodología de [1] se ejecutó el algoritmo con distintos niveles de datos faltantes. Se ejecutó para 5%, 10%, 20% y 30% de datos faltantes.

Method	5%	10%	20%	30%
Imp. CTGAN	0,31	0,30	0,29	0,29
Cond. Imp. GAN	0,81	0,86	0,89	0,91
GAIN	1,01	1,02	1,06	1,08
MICE	8,91	9,01	9,12	6,53
MissForest	6,53	6,51	7,01	7,01

Figure 5. Resultados de RMSE promedio comparando algoritmos, tomado y editado de [1]

5.1. Resultados obtenidos

Los resultados obtenidos son muy interesantes, disminuye tres veces el RMSE promedio comparado a los resultados de los autores de [1]. En la Figura 5 se observa el comportamiento del algoritmo de este trabajo en comparación con los resultados obtenidos por los autores de [1]. Las filas son los algoritmos con los valores obtenidos de RMSE promedio de las 10 ejecuciones de cada algoritmo de imputación. El algoritmo de este trabajo es “Imp. CTGAN”. Las columnas son los niveles de datos faltantes.

Para el algoritmo de este trabajo “Imp. CTGAN”, se observa que a medida que aumenta la probabilidad de datos faltantes disminuye el RMSE lo cual resulta contradictorio. Se reviso el algoritmo que genera la máscara M y se comprobó que el resultado de enmáscarar es correcto. Además, se ejecutaron las pruebas de los resultados presentados varias veces y se notó que en algunas ejecuciones los resultados variaban $\pm$ 0.01. Se observa en los otros resultados presentados por los autores de [1] que MICE también mejora por lo cual no parece ser un comportamiento aislado. Por último, la variación de resultados de “Imp. CTGAN” es de $\pm$ 0.02, la cual es muy pequeña. En base

1. https://github.com/sdv-dev/CTGAN/blob/master/ctgan/data_transformer.py

2. https://github.com/veiro/Extentend_Data_imputation_with_CTGAN/

3. https://raw.githubusercontent.com/gdeng96/cond-imp-gan-ranking/main/synthetic_ranking_data.csv

a todo lo anterior expuesto se considera que los resultados son válidos.

5.2. Proceso de desarrollo

Por último, como el desarrollo de los cambiós fue incremental se obtuvieron resultados intermedios a medida que se iban implementando.

En el desarrollo se utilizó una tasa de 5% de datos faltantes.

La primera versión consistió en hacer funcionar el código de [1]. Se obtuvo un RMSE promedio de 0.96. El código de esa versión está disponible en github [4].

En la segunda versión se aplicó la representación VGM a las columnas continuas y se mantuvo la columna "Category" en formato one-hot-encoding en la condición de la CGAN. Se obtuvo un cambió sustancial en el RMSE promedio de 0.29. El código de esa versión está disponible en github [5].

La tercera versión consistió en utilizar el muestreo de CTGAN en la condición. Se obtuvo un RMSE de 0.30. El código de esa versión está disponible en github [6]. En la revisión de los resultados se observó que no se estaba usando la columna "Category", entonces se implementó una nueva versión usando la columna "Category" como una columna más de la tupla.

Por último, la versión cuarta y final utiliza la representación VGM para las 6 columnas y el muestreo de CTGAN. El RMSE obtenido es 0.30. El código de esa versión está disponible en github [7].

6. Conclusión

Utilizar las estrategias propuestas por autores de CTGAN muestran una mejora a la hora de imputar datos utilizando las técnicas de [1]. También se mostró que la principal ventaja es utilizar la representación VGM para representar las variables continuas.

Referencias

[1] G. Deng, C. Han, and D. S. Matteson, "Extended missing data imputation via gans for ranking applications," 2020. [Online]. Available: https://arxiv.org/abs/2011.02089

[2] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," in *Advances in Neural Information Processing Systems*, 2019.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," Jun. 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[4] "Aprendizaje profundo para visión artificial - udelar," https://eva.fing.edu.uy/pluginfile.php/320521/mod_resource/content/2/DLVIS_2020_clase17.pdf, (Accessed on 10/11/2022).

[5] "The gan zoo," https://github.com/hindupuravinash/the-gan-zoo/blob/master/gans.tsv, (Accessed on 10/11/2022).

[6] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[7] "07.tabular data.pptx," https://eva.fing.edu.uy/pluginfile.php/361884/mod_resource/content/2/07.Tabular%20Data.pptx%20%281%29.pdf, (Accessed on 12/11/2022).

[8] J. Yoon, J. Jordon, and M. van der Schaar, "GAIN: Missing data imputation using generative adversarial nets," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 5689–5698. [Online]. Available: https://proceedings.mlr.press/v80/yoon18a.html

4. https://github.com/veiro/Extentend_Data_imputation_with_CTGAN/blob/bb82cf2fc2c6ef7fc88daf20a5223a0e830307b5/Conditional_GANs_for_Synthetic_Ranking_Data_v2.ipynb

5. https://github.com/veiro/Extentend_Data_imputation_with_CTGAN/blob/d6f4bd655020a93a9dbcbbf71e51adacc4fd9cf5/GANS_tarea_V2.ipynb

6. https://github.com/veiro/ADS2020/blob/4e97896da67d143a21b24894a43b1eff4f1687eb/GANS_tarea_V2.ipynb

7. https://github.com/veiro/Extentend_Data_imputation_with_CTGAN/blob/a8bc276dcced64bb49be7da0bd7bfc2bd37b94e5/GANS_tarea_V2.ipynb