Uso de ambientes reproducibles para la validación de modelos de aprendizaje estadístico

Fabricio Machado, Mauro Loprete

Uso de ambientes reproducibles para la validación de modelos de aprendizaje estadístico. Una implementación en Docker.

Fabricio Machado, Mauro Loprete

Keywords— Reproducibilidad, Portabilidad, Análisis discriminante, Validación cruzada, MLOps

Resumen

Este trabajo presenta la implementación de un modelo de aprendizaje estadístico apoyado en la herramienta Docker. Se combinan técnicas de clustering y análisis discriminante para clasificar individuos en grupos sociales y se recurre al uso de ambientes reproducibles en Docker para validar dicha clasificación y garantizar su replicabilidad. La relevancia del trabajo no se limita únicamente a los resultados del modelo y sus aportes a la ciencia social, sino que incluye también la experiencia acumulada en el uso de herramientas modernas y sus potencialidades para el diseño de flujos de análisis más robustos y trazables.

Resumen

Abstract en inglés

Introducción

El uso de técnicas de aprendizaje estadístico se ha extendido a múltiples campos de investigación. Estos métodos requieren una adecuada fundamentación de las decisiones adoptadas para su implementación y, además, pueden ser computacionalmente muy exigentes. Por este motivo, se han desarrollado herramientas complementarias a los softwares habituales que permiten afrontar estos desafíos y, al mismo tiempo, fomentan la reproducibilidad. En este contexto, la reproducibilidad no debe entenderse como un fin en sí mismo, sino como parte del proceso de trabajo a incorporar desde el inicio, lo que facilita la verificación de resultados, mejora la colaboración y aumenta la credibilidad del trabajo ante la comunidad académica.

Utilizamos entornos reproducibles en Docker para implementar y validar un modelo de aprendizaje estadístico en el marco de una investigación en ciencias económicas. En este documento, sistematizamos dicha implementación, destacando los métodos utilizados y los resultados obtenidos; además, introducimos los conceptos básicos para trabajar con Docker y presentamos una guía práctica para su uso.

Docker es una herramienta de software que permite ejecutar aplicaciones de manera aislada, consistente y reproducible, reduciendo los problemas habituales asociados a versiones, dependencias y diferencias entre entornos. Facilita que un programa y todos los componentes necesarios para su funcionamiento se configuren una sola vez y puedan ejecutarse de forma prácticamente idéntica en distintos contextos: desde un equipo personal hasta servidores locales o infraestructuras en la nube. Gracias a ello, simplifica las tareas de prueba y despliegue, mejora la colaboración entre desarrolladores y favorece la automatización de entornos complejos.

Usar Docker en investigación puede ser muy útil porque nos permite congelar un entorno de trabajo completo (sistema, librerías, versiones de R, Stata, Python, LaTeX, etc.) y volver a él siempre que lo necesitemos, sin depender de la computadora específica ni de cómo esté configurada. Eso hace que los proyectos sean reproducibles en el tiempo (volver a correr los archivos de replicación de un paper años después de realizados) y más fáciles de compartir con otras personas (coautores, tribunales, replicadores) sin dedicar horas a instalar dependencias o resolver conflictos de versiones.

En un proyecto típico podemos combinamos varios lenguajes y herramientas (R para procesar datos y gráficos, Stata para partes del análisis, Python para algún script auxiliar, LaTeX/Quarto para el documento, etc.). Además, solemos tener distintas máquinas involucradas: portátil personal, PC del trabajo, servidor de la facultad o la nube. Docker encaja bien con esa forma de trabajar porque permite definir un entorno único para el proyecto y ejecutarlo igual en todos esos lugares. En vez de escribir instrucciones largas del tipo “instalar tal versión de R, tal paquete, tal versión de Stata”, definimos el entorno una vez, lo empaquetamos y cualquiera puede levantarlo y correr el proyecto completo.

Docker es compatible con prácticamente todos los softwares y herramientas que usamos para investigar: se puede trabajar con R y RStudio Server, Python y Jupyter, Stata (en servidores donde tengas licencia), y sistemas de compilación de documentos como LaTeX o Quarto. En la práctica, lo que hace Docker es ejecutar estos programas dentro de un entorno Linux aislado, pero nosotros seguimos trabajando desde nuestro navegador o editor habitual (como RStudio, Positron o VS Code), conectándonos al contenedor. Esto permite crear, por ejemplo, un contenedor que tenga R + todos los paquetes necesarios + TeX para compilar el paper, otro con Stata y sus do-files, y así garantizar que los análisis y el documento se puedan reconstruir exactamente igual en cualquier máquina que tenga Docker instalado.

¿Qué aporta este documento? - Ejemplo de aplicación práctica en un modelo estadístico. - Guía para el uso de Docker en investigación. - Sistematización de los beneficios de usar esta herramienta. - Acercamiento de la herramienta a la comunidad IESTA.

La reproducibilidad es clave para verificar y validar nuestros resultados una vez finalizada una investigación. Pero también una forma de trabajo que podemos adoptar desde el inicio: documentar los pasos, fijar versiones de software, automatizar scripts y organizar el código y los datos de manera sistemática. Optar por este camino tiene beneficios importantes: facilita la revisión interna, mejora la colaboración entre coautores, reduce errores difíciles de rastrear y aumenta la credibilidad del trabajo ante la comunidad académica, que puede evaluar, replicar o extender nuestros resultados con mayor confianza.

La combinación entre los métodos y datos elegidos implicó un alto requerimiento computacional que motivó el acercamiento inicial a los entornos reporducibles.

Modelo de aprendizaje estadístico

Partimos de un análisis sobre la desigualdad económica cuyo objetivo es clasificar a los individuos en grupos sociales a partir de variables relevantes para estudiar su participación en el ingreso laboral (Machado, 2023). En una primera etapa aplicamos **análisis de clúster**, que nos permite detectar grupos relativamente homogéneos entre los perceptores de ingresos laborales en un año base, utilizando variables que resumen las características de los trabajadores y de sus puestos de trabajo. En una segunda etapa empleamos **análisis discriminante** para construir, a partir de esos clústeres, una regla de clasificación que pueda aplicarse a otros años y así seguir la evolución de los grupos y su participación en el ingreso laboral. Dado que esta regla se estima en un conjunto de datos y luego se aplica sobre información externa, utilizamos técnicas de **validación cruzada** para evaluar su desempeño y reducir el riesgo de sobreajuste.

A continuación se detallan los métodos que forman parte del proceso de implementación en Docker que se seguirá en las secciones posteriores, el cual incluye los desarrollos del análisis discriminante y de la validación cruzada. El paso previo de análisis de clúster puede consultarse en el Anexo 1.

Datos

La fuente de datos utilizada es la Encuesta Continua de Hogares (ECH) elaborada por el Instituto Nacional de Estadística (INE). Se emplea información de personas y hogares para los años 2006, 2011, 2019 y 2021, seleccionados por representar momentos distintos de la dinámica económica reciente (salida de la crisis, auge económico, período previo a la pandemia y pospandemia). En cada año se consideran las personas ocupadas de 20 años o más de todo el país con ingresos laborales, definiéndose ocupación como haber trabajado al menos una hora en la semana de referencia. Se excluye a quienes trabajaron en programas públicos de empleo, a los miembros no remunerados del hogar y a los individuos con datos faltantes en alguna de las variables empleadas en el análisis de clúster. En este último caso, la ausencia de datos impide computar dicho análisis; en los demás casos, los registros se apartan debido a su escasa magnitud relativa y a que no se cuenta con evidencia de que constituyan grupos clave para diferenciar a los trabajadores. En particular, los no remunerados no participan de la masa salarial, que es el objeto central del estudio.

Las muestras finales incluyen 92.735 individuos en 2006, 53.210 en 2011, 43.565 en 2019 y 9.007 en 2021. Una vez expandidas mediante el ponderador anual, las estimaciones poblacionales ascienden a 1.128.540 individuos en 2006, 1.411.419 en 2011, 1.441.540 en 2019 y 1.396.668 en 2021. La notoria reducción en el número de observaciones para 2021 se atribuye a los cambios en la metodología de la ECH introducidos en 2020 (INE, 2020). Para el análisis se seleccionan variables que recogen información sobre las tareas realizadas en el trabajo, el sector de actividad, la categoría de ocupación, la informalidad, las horas trabajadas, el nivel educativo, la propiedad de la vivienda y el cobro de rentas.

Análisis discriminante

Esta aplicación pretende llegar a una misma clasificación de grupos para todos los años seleccionados, manteniendo las características distintivas de cada uno. A partir de un conjunto de datos con la pertenencia a los grupos conocida (año base donde se aplicó el análisis de cluster), se deriva una regla para asignar las observaciones a los grupos haciendo mínima la probabilidad de clasificar incorrectamente. Se clasifican los datos de 2006, 2019 y 2021 para formar la misma cantidad de grupos y con la misma caracterización resultante del análisis de cluster realizado en 2011, que fue seleccionado como año base por presentar la clasificación más estable (Ver Anexo 1).

Entre las distintas funciones discriminantes se eligió el discriminante logístico dado que todas las variables son cualitativas (Peña, 2002). Dado que se espera encontrar más de dos grupos, suponemos que la variable que indica las subpoblaciones proviene de una distribución multinomial. Se presentan a continuación las probabilidades logarítmicas entre grupos (James et al., 2021):

$$log(\frac{Pr(Y=k|X=x)}{Pr(Y=K|X=x)}) = \beta_{k0} + \beta_{k1}x_1 + ... + \beta_{kp}x_p$$

En nuestra especificación se predefinieron cuatro grupos ($K = 4$), por lo que dejando fuera al de referencia nos quedan tres funciones de enlace para $k = 1, 2, 3$ ($k = 1, ..., K - 1$). Se denotan como $X = (X_1, ..., X_p)$ el conjunto de variables utilizadas, siendo p la cantidad de categorías de todas las variables, dejando siempre una categoría libre por la trampa de las dummys ($p = 23$).

Se estiman 69 parámetros $(K-1)(p)$ que representan la probabilidad de pertenecer al los grupos $k = 1, 2, 3$ en relación con el grupo de referencia según cada una de las características $X_1, ..., X_p$.[1]

Para revisar la significación conjunta del modelo se plantea, $H_0)\hat{\beta}_0 = \hat{\beta}_1 = ... = \hat{\beta}_{23} = 0$, contra $H_a)$ algún $\hat{\beta} \neq 0$, para esto se utiliza el estadístico de razón de verosimilitud que para esta aplicación sigue una distribución χ^2_{69} $(\chi^2_{(K-1)(p)})$.

Con la información obtenida del año base se pretende discriminar a los individuos de los demás años. Los individuos se asignan a un grupo o a otro comparando las probabilidades a priori, con esta regla de clasificación el grupo más probable es aquel donde su verosimilitud es más alta. Para evaluar el modelo se utiliza una medida de precisión definida como la cantidad de datos bien clasificados sobre el total. Valores altos de la misma implican un buen desempeño del modelo

Validación cruzada

Como es de interés clasificar datos externos, para evitar problemas de sobreidentificación se utiliza el método de **validación cruzada** $k-folds$ (James et. al., 2021). Se dividen los datos aleatoriamente en k pliegues de un tamaño similar, se estima la precisión tomando uno de los pliegues para el testeo y los restantes $k-1$ para el entrenamiento, el procedimiento se repite k veces haciendo variar el pliegue de testeo de tal forma que todos los datos hayan sido utilizados como testeo y entrenamiento. Como resultado se tienen k estimaciones de la tasa de error y se tomara el promedio del mismo como una buena medida del ajuste sobre el total de los datos.

El trabajo fue principalmente realizado en R, desde el procesamiento de datos con la librería tidytable y la implementación de los métodos con la librería **nnet** (v7.3-19; Venables y Ripley, 2002). Adicionalmente se crearon las funciones en R necesarias para la aplicación de la validación cruzada. Se explotó la independencia de los modelos de validación cruzada k-folds para realizarlos mediante computación en paralelo con la librería parallel.

Implementación

Dado el volumen de datos y la intensidad computacional requerida, se optó por el uso de Docker para garantizar la reproducibilidad y portabilidad del entorno de desarrollo. Docker nos permitió encapsular todas las dependencias del proyecto dentro de un contenedor, asegurando que el proceso pueda ejecutarse de manera consistente en diferentes entornos, ya sea en servidores en la nube, en un servidor de cómputo local o en computadoras personales con diferentes sistemas operativos (Matthias y Kane, 2015).

El ambiente de trabajo se configuró mediante un **Dockerfile** que define todas las herramientas y bibliotecas necesarias para la ejecución del modelo. Este archivo incluye desde la instalación de R y sus paquetes relevantes, hasta la configuración de las dependencias del sistema operativo. De esta forma, cualquier usuario puede reproducir el ambiente exacto simplemente construyendo y ejecutando el contenedor.

Para optimizar el rendimiento y facilitar la integración del modelo en entornos de producción, se aplicaron principios de Machine Learning Operations (MLOps). **MLOps** es un enfoque que combina prácticas de desarrollo de software (DevOps) con metodologías específicas de Machine Learning, con el objetivo de automatizar, escalar y gestionar los ciclos de vida de los modelos de aprendizaje automático de manera eficiente.

Dentro de este marco, el procesamiento en paralelo jugó un papel crucial. Se utilizó el paquete **parallel** (R Core Team, 2023) de R para distribuir el procesamiento de datos y la validación cruzada del modelo en múltiples núcleos de CPU, lo que permitió acelerar significativamente los tiempos de ejecución y manejar grandes volúmenes de datos de manera más efectiva. La integración de estas prácticas asegura que el modelo no solo sea reproducible, sino también escalable y adaptable a diferentes entornos.

Docker funciona como una capa de abstracción a nivel del sistema operativo, que permite empaquetar aplicaciones y sus dependencias en contenedores ligeros y portables. De este modo, el software se ejecuta de forma consistente en distintos entornos (equipo local, servidores o nube). Docker se ejecuta mediante comandos en la terminal (CLI) o a través de interfaces gráficas (Docker Desktop) y consta de varios componentes clave que facilitan la creación, distribución y ejecución de contenedores.

Los principales componentes de Docker son:

[1] Los parámetros del modelo se estiman bajo el método de máxima verosimilitud con técnicas numéricas de optimización, en concreto el método Brodyen-Fletcher-Goldfarb-Shanno.

- `Dockerfile`: archivo de texto con instrucciones para construir la imagen.
- Imagen: plantilla estática y versionada del entorno completo.
- Contenedor: instancia en ejecución de la imagen.
- Registro: repositorio para almacenar y compartir imágenes (Docker Hub, GHCR).

A continuación se presenta en la 1 un diagrama simplificado del ciclo de vida de Docker, que ilustra los conceptos clave y el flujo típico de trabajo con contenedores:

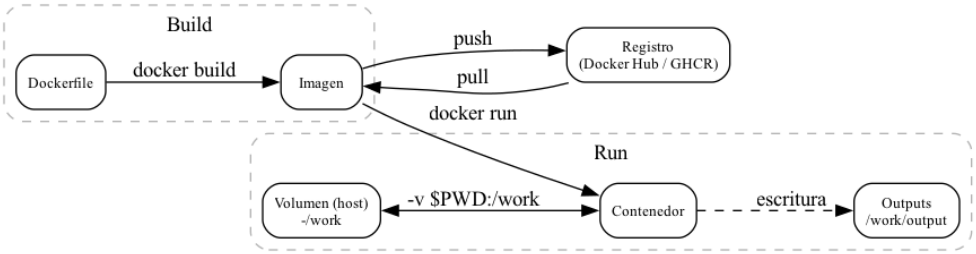

Figura 1: Ciclo de vida simplificado de Docker.

Para crear un contenedor se define un `Dockerfile` con la imagen base y los pasos para instalar dependencias, copiar código y configurar el entorno. El resultado es una imagen; un contenedor es la instancia en ejecución de esa imagen.

En términos prácticos, resulta útil retener algunas ideas esenciales. Una imagen es la plantilla inmutable del entorno (sistema, librerías y, cuando corresponde, parte del código) identificada por `repo:tag` y por un `digest` (`sha256:`) que garantiza identidad exacta entre máquinas. Un contenedor es el proceso que ejecuta esa imagen de forma aislada; toda escritura dentro del contenedor es temporal, salvo que se utilice un volumen montado desde el equipo anfitrión, que es donde deben guardarse datos y resultados. El Dockerfile es la receta para construir la imagen con instrucciones como `FROM`, `RUN` o `COPY`, y el registro es el repositorio donde se versionan y comparten imágenes con coautores y replicadores. También es recomendable limitar recursos (CPU y memoria) para obtener ejecuciones controladas y comparables.

Desde el punto de vista operativo, Docker utiliza un servicio del sistema que crea y administra contenedores, apoyándose en el kernel para aislar procesos y recursos, y en un sistema de archivos por capas (copy-on-write) que ahorra espacio y acelera las compilaciones. El ciclo de trabajo es simple y repetible: se construye la imagen, se ejecuta montando datos y carpeta de resultados, se observan los registros o se ingresa al contenedor cuando es necesario, se persisten los artefactos en el volumen del equipo anfitrión y, finalmente, se limpian recursos no utilizados. La idea central es que cualquier cambio que ocurra dentro del contenedor es efímero; por ello, los resultados deben escribirse siempre en una ruta montada desde el host (por ejemplo, `/output`).

La diferencia respecto a otros sistemas de virtualización como máquinas virtuales es que Docker comparte el kernel del sistema anfitrión, lo que reduce el uso de recursos y acelera el arranque. Esta característica lo vuelve especialmente adecuado para desarrollar, desplegar y escalar análisis reproducibles en investigación. Las máquinas virtuales, en cambio, cargan un sistema operativo completo, consumen más CPU, memoria y almacenamiento, y requieren herramientas de administración específicas. Para la mayoría de los flujos de investigación orientados a reproducibilidad y portabilidad, los contenedores ofrecen una solución más ágil y suficiente.

Para sostener la reproducibilidad a lo largo del tiempo, se recomienda etiquetar imágenes con fecha o versión (y, cuando sea crítico, referenciarlas por `digest`), congelar paquetes (R: `renv`; Python: `requirements.txt`) y guardar `sessionInfo()` o `pip freeze` junto a los resultados. Los datos y las credenciales no deben incorporarse en la imagen: se montan a través de un volumen con `-v` y se suministran por variables de entorno o archivos `.env`. También es conveniente fijar semillas aleatorias, normalizar configuraciones regionales y zonas horarias y ejecutar como usuario no privilegiado (o con `--user $(id -u):$(id -g)`). Documentar los comandos exactos de `docker build` y `docker run` (y los límites de recursos) en un README o Makefile mejora la trazabilidad.

El bloque de código 1 contiene lo esencial para inspeccionar, ejecutar, observar y limpiar:

A modo de cierre, es útil recordar algunas pautas para evitar errores frecuentes: utilizar siempre una imagen etiquetada y el Dockerfile versionado; fijar dependencias y semillas; montar datos y escribir resultados en `/work/output`; documentar el comando de ejecución y los recursos asignados; y conservar registros y metadatos de sesión junto a los outputs. Si los resultados no aparecen, suele deberse a que no se montó `-v "$PWD":/work` o no se escribió en la ruta

Código 1

```
# Inspección
docker images # Lista todas las imagenes locales
docker ps -a # Lista todos los contenedores (también los detenidos)

# Ejecutar con límites y variables
docker run --rm -it -v "$PWD":/work -w /work --cpus=4 -m 8g \
  --env-file .env ghcr.io/fabriciomch/cv:latest /bin/bash

# Logs y acceso
docker logs -f <nombre_o_id> # Seguir logs en tiempo real
docker exec -it <nombre_o_id> /bin/bash # Acceder a la shell del contenedor

# Copiar artefactos (scripts, resultados, etc.)
docker cp <nombre_o_id>:/work/output/resultados.csv ./output/

# Limpieza
docker system prune -f # Limpia volumenes, redes y contenedores detenidos
docker image prune -f # Limpia imagenes no usadas
```

montada; si faltan paquetes, deben instalarse en la imagen base o en un paso de inicialización; si surgen problemas de permisos, ejecutar con `--user $(id -u):$(id -g)`; y si hay limitaciones de memoria, aumentar `-m` o reducir el tamaño de los lotes.

Ejemplo

Como ejemplo concreto, a continuación se muestra un Dockerfile (ver bloque de código 2) para obtener el resultado del análisis detallado en la sección **PONER REF**. Usa como base `eddelbuettel/r2u:22.04` (Ubuntu + R con binarios precompilados), instala los paquetes mencionados en **poner ref** (`nnet`, `vroom`, `here`, `tidytable`, `magrittr`), crea una estructura de carpetas para el proyecto e incluye los datos (`/home/ubuntu/model/{data,R,output}`) a utilizar, instala paquetes de R, copia archivos y define un comando por defecto que ejecuta el script principal.

Una vez creada la imagen a partir de este `Dockerfile`, los contenedores ejecutarán el script `main.R`, que implementa el modelo de aprendizaje estadístico con validación cruzada y guarda los resultados en la carpeta `output/`. El comando por defecto puede ser sobrescrito al ejecutar el contenedor (mediante `docker run`) si se desea correr otro script o ingresar a una shell interactiva (`docker run imagen -it bash`) dentro del contenedor.

Al construir esta imagen, los comandos de ejecución deben montar el proyecto en la misma ruta definida como `WORKDIR` para que los scripts y outputs queden en el lugar esperado. Como puede verse en el bloque de código 3.

Este Dockerfile tiene un ejemplo básico para que devuelva el accuracy del modelo, definido en la sección **poner ref** como la forma correcta de validar los resultados. En otros casos puede ser util guardar otras métricas, modelos intermedios o datos procesados en la carpeta `output/`, que está montada desde el equipo anfitrión.

La imagen ya se encuentra construida y publicada en base a la ultima versión del código en el repositorio de GitHub y mediante GitHub Actions se actualiza automáticamente al crear nuevas etiquetas.

Ejecución

Para trabajar con la imagen pública de este proyecto alojada en GHCR (`ghcr.io/fabriciomch/cv`) no es necesario modificar nada: basta con descargarla, ejecutarla montando la carpeta del proyecto y, si se desea mantener una variante local, construir una imagen propia a partir de un Dockerfile. De manera esquemática, un flujo típico de trabajo puede resumirse del siguiente modo.

Código 2

```
FROM eddelbuettel/r2u:22.04  # Ubuntu 22.04

# Estructura de proyecto
RUN mkdir -p /home/ubuntu/model/{data,R,output}
WORKDIR /home/ubuntu/model

# Paquetes de R del sistema (binarios)
RUN apt-get update && apt-get install -y \
    r-cran-nnet \
    r-cran-vroom \
    r-cran-here \
    r-cran-tidytable \
    r-cran-magrittr \
  && rm -rf /var/lib/apt/lists/*

# Copiar archivos (opcional; preferible montar con -v en ejecución)
COPY R/* R/
COPY data/ data/
COPY main.R main.R

# Variables de entorno de ejemplo
ARG RUN_MODE=prod
ENV RUN_MODE=${RUN_MODE}

# Comando por defecto (En este script se ejecutan los modelos con CV)
CMD ["Rscript", "main.R"]
```

Pasos de uso (imagen GHCR):

1. Obtener la imagen
   - Para imágenes públicas (como en este caso), ejecutar:

     ```
     docker pull ghcr.io/fabriciomch/cv:latest
     ```

   - Si fuera privada, autenticarse previamente en GitHub Container Registry (GHCR) con un personal access token (PAT) asociado a la cuenta de GitHub.
2. Vincular el proyecto local
   - Desde la carpeta del proyecto (donde residen código y datos), ejecutar:

     ```
     docker run --rm -it -v "$PWD":/work -w /work ghcr.io/fabriciomch/cv:latest /bin/bash
     ```

3. Ejecutar el script de análisis
   - Por ejemplo:

     ```
     docker run --rm -v "$PWD":/work -w /work ghcr.io/fabriciomch/cv:latest Rscript main.R
     ```

4. (Opcional) Congelar una variante propia
   - Clonar el repositorio, adaptar el Dockerfile si es necesario y construir:

     ```
     docker build -t cv_multinom --build-arg test_arg=TRUE .
     ```

Código 3

```
# Construir la imagen local
docker build -t cv_multinom .

# Abrir una shell en la ruta del proyecto dentro del contenedor
docker run --rm -it \
  -v "$PWD":/home/ubuntu/model -w /home/ubuntu/model \
  cv_multinom /bin/bash

# Ejecutar el script principal y escribir resultados en ./output
docker run --rm \
  -v "$PWD":/home/ubuntu/model -w /home/ubuntu/model \
  cv_multinom Rscript main.R
```

Para la ejecución cotidiana y colaborativa, resulta útil acompañar el flujo de trabajo con herramientas auxiliares. La integración continua (GitHub/GitLab CI) permite construir y publicar imágenes automáticamente al crear etiquetas; un Makefile facilita encapsular comandos repetitivos como `docker run` o `docker build`; y servicios como Amazon EC2 o AWS Batch resultan apropiados para ejecutar los mismos contenedores cuando se requiere mayor capacidad de cómputo.

Para obtener resultados de forma consistente:

- Descargar o actualizar la imagen y verificar la etiqueta/versión utilizada.
- Ejecutar el contenedor montando el proyecto y escribir los outputs en la carpeta compartida.
- Registrar logs, versiones de paquetes y metadatos de sesión junto a los resultados.

Para ilustrar la simplicidad de uso, el bloque de código 4 muestra los comandos mínimos para descargar y ejecutar la imagen pública del proyecto, montando la carpeta `output` para guardar los resultados generados por el script principal.

Código 4

```
docker pull ghcr.io/fabriciomch/cv:main
docker run -v ${PWD}/output/:/home/ubuntu/model/output ghcr.io/fabriciomch/cv:main
```

Este comando descarga la imagen pública y ejecuta el contenedor, montando la carpeta `output` del proyecto local para guardar los resultados generados por el script principal. Al tener en cuenta las buenas prácticas mencionadas, se asegura que los análisis sean reproducibles y portables entre distintos entornos y colaboradores.

Desarrollo en la nube con GitHub Codespaces

Una vez que se cuenta con la imagen Docker, existen múltiples formas de utilizarla para ejecutar el análisis de forma reproducible tanto en entornos locales (utilizando Docker Desktop o Docker Engine) como en la nube como GitHub Codespaces.

GitHub Codespaces es un servicio que permite abrir un entorno de desarrollo completo en la nube, basado en Visual Studio Code, directamente desde un repositorio de GitHub. Este entorno puede configurarse para utilizar contenedores Docker como base, lo que facilita la ejecución de análisis reproducibles sin necesidad de instalar software adicional en el equipo local.

A continuación se describen los pasos básicos para utilizar la imagen Docker del proyecto en un entorno de GitHub Codespaces (Figura 2). Una vez abierto el Codespace, se puede trabajar directamente con el código. El repositorio incluye un editor de texto listo para ejecutar el contenedor Docker (Figura 3). Esto puede ser especialmente útil para colaborar con otros investigadores, docentes, alumnos, ya que todos pueden acceder al mismo entorno sin preocuparse por las diferencias en la configuración local de sus equipos.

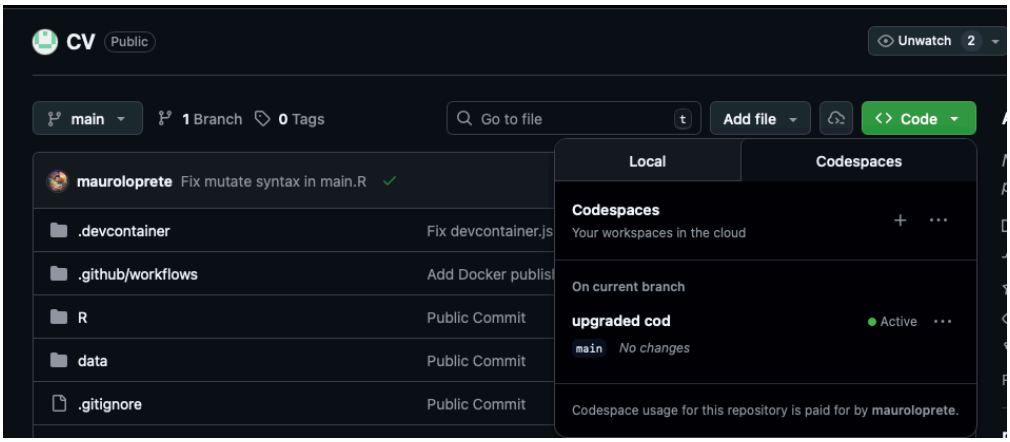

Figura 2: Apertura de un GitHub Codespace desde la interfaz del repositorio.

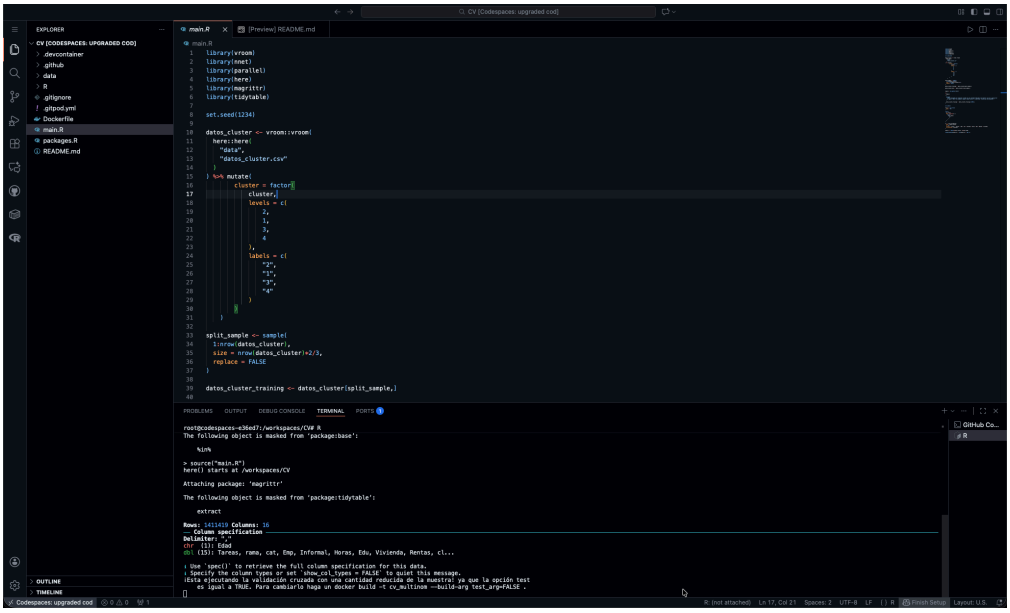

Figura 3: Entorno de desarrollo en la nube provisto por GitHub Codespaces.

GitHub Codespaces puede entenderse, en este contexto, como un complemento a los flujos basados en contenedores: permite abrir un entorno de desarrollo efímero en la nube, definido a partir del propio repositorio (por ejemplo mediante un archivo de configuración `.devcontainer.json`), en el que se puede inspeccionar el código, ajustar scripts y lanzar contenedores Docker con el mismo entorno que se utiliza localmente.

Esto resulta especialmente útil para trabajo colaborativo, docencia o talleres, ya que cualquier integrante del equipo puede acceder a un entorno listo para usar sin instalar herramientas en su equipo personal. No obstante, es importante tener en cuenta que GitHub Codespaces presenta ciertas limitaciones en cuanto a recursos (CPU, memoria y almacenamiento) y tiempo de ejecución, por lo que puede no resultar adecuado para análisis muy intensivos o de larga duración. Servicios de cómputo en la nube como Amazon EC2 u otras plataformas de infraestructura como servicio suelen ser más apropiados cuando se necesita alquilar capacidad de procesamiento para ejecutar cargas pesadas en segundo plano (por ejemplo, corridas extensivas de validación cruzada, simulaciones o entrenamientos de modelos de mayor porte). En estos casos, la práctica recomendable es mantener la lógica del análisis y la definición del entorno en una imagen Docker versionada, y desplegar esa misma imagen tanto en el entorno local como en instancias de cómputo en la nube, de modo que la decisión sobre el 'dónde' ejecutar (equipo propio o nube) no afecte al 'cómo' se ejecuta el análisis.

Comentarios finales

La implementación del modelo discriminante alcanzó una **precisión del 94.35 %**, demostrando un excelente desempeño en la clasificación de los individuos en los grupos sociales definidos. Además, este trabajo nos permitió aprender y aplicar herramientas modernas como Docker, Gitpod y Amazon EC2, lo que facilitó la reproducibilidad y escalabilidad del proyecto.

Hemos comprobado que Docker puede ser una herramienta valiosa en la investigación en Estadística y Ciencias Sociales, ya que permite encapsular ambientes de desarrollo completos, asegurando que los experimentos puedan replicarse de manera consistente en diferentes entornos. Sus principales fortalezas son la reproducibilidad, el aislamiento de dependencias, la portabilidad y la eficiencia computacional. Particularmente es beneficioso incorporar esta herramienta cuando se trabaja con combinaciones de datos y métodos que requieren gran exigencia computacional y se detectan espacios para paralelizar los procesos.

Recomendamos el uso de Docker y su complementación con otras herramientas modernas.

La experiencia documentada muestra que la adopción de contenedores Docker aporta beneficios tangibles para proyectos de métodos cuantitativos: reduce fricciones en la instalación de dependencias, estabiliza versiones de software, facilita la colaboración entre equipos y mejora la trazabilidad de los resultados. Al encapsular el entorno de ejecución, los análisis pueden replicarse en el tiempo y compartirse con terceros sin la carga de reconstruir manualmente las condiciones de trabajo. Esto se vuelve especialmente valioso cuando se combinan múltiples lenguajes y herramientas (por ejemplo, R para preprocesamiento y modelado, Python para tareas auxiliares, y servicios adicionales para visualización o documentación) o cuando se requiere ejecutar los mismos scripts en infraestructuras heterogéneas (estaciones de trabajo, servidores institucionales o plataformas en la nube).

Desde el punto de vista metodológico, la estrategia combinada de agrupamiento y análisis discriminante permite caracterizar perfiles de individuos y trasladar esa caracterización a diferentes períodos, con una métrica de desempeño clara y validada mediante procedimientos estándar (k-folds). La estimación de un discriminante logístico sobre variables cualitativas resulta apropiada bajo supuestos razonables y ofrece una regla de clasificación interpretable.

En síntesis, el uso de ambientes reproducibles con Docker constituye una práctica recomendable para fortalecer la solidez empírica y la portabilidad de los análisis en contextos académicos, a la vez que habilita escalabilidad y mantenimiento a lo largo del ciclo de vida de los proyectos.

Anexo

Análisis de cluster

Mediante esta técnica se pretende clasificar individuos en grupos. Para esto es necesario definir una métrica que cuantifique la similitud entre los individuos de la población. La variedad de métricas tienen distinta sensibilidad a los datos atípicos, así como difieren en el tipo de variables que pueden analizar entre cuantitativas y categóricas, nominales u ordinales.

Dado que se trabajará con variables categóricas la métrica elegida es el coeficiente de similitud de Jaccard, definido como el tamaño de la intersección dividido por el tamaño de la unión de los conjuntos de muestras:

$$J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|} = \frac{|A \bigcap B|}{|A| + |B| - |A \bigcup B|}$$

En su aplicación particular a este problema se requiere utilizarlo para considerar similares a quienes compartan categorías de las variables. De esta forma luego del análisis individuos que compartan un conjunto de categorías estarán asignados a un mismo grupo. Analizando posteriormente las características de los individuos pertenecientes a cada grupo, se podrá llegar a una caracterización a partir de la cual compararlos.

Existen métodos de clusterización jerárquicos y no jerárquicos. Su diferencia fundamental es que en los métodos jerárquicos la unión se realiza entre clusters y una vez que los individuos pertenecen a un cluster lo hacen hasta el final ya que se basan en proximidades locales, mientras que los métodos no jerárquicos o de partición dependen de definir de antemano la cantidad de grupos, asignando así para cada grupo un centroide y su asociación con los individuos será con base en medidas de proximidad. Como los métodos jerárquicos no son recomendados para grandes conjuntos de datos se optará por uno no jerárquico (Guénoche et al., 1991).

Los métodos no jerárquicos se caracterizan por comenzar con una primera solución de clasificación, y luego de forma iterativa aplicar diferentes combinaciones para obtener particiones más homogéneas.

El método a utilizar es K-Medoides o partición en torno a los medoides (PAM), donde cada cluster está representado por el individuo mediano candidato. Un medoide es un objeto representativo del conjunto de datos, tal que su diferencia promedio con los demás objetos es mínima. Con la cantidad k de grupos definida previamente se asignan k medoides dentro de los n individuos, los demás individuos son asignados al grupo del medoide más cercano, la asignación del medoide va cambiando minimizando la distancia entre los puntos de los grupos y su medoide designado. Como el arranque es aleatorio se recomienda aplicar más de una vez el método para llegar a conclusiones válidas.

Dado el tamaño de los datos a utilizar se optará por la implementación CLARA (Kaufman y Rousseeuw, 1990) de K-Medoides, que utiliza el enfoque de muestreo. Considera una muestra fija de los datos y aplica en ella el algoritmo PAM encontrando el conjunto óptimo de medoides para la muestra. Se repite el proceso de muestreo y agrupación un número predefinido de veces para minimizar el sesgo del muestreo. La cantidad de muestras se definirá probando un valor creciente de las mismas y verificando a partir de cual los resultados se estabilizan.

Como se mencionó anteriormente, este tipo de métodos depende de predefinir la cantidad de grupos. Una alternativa para esto es maximizar la silueta promedio. La silueta es un estadístico que determina cuán similar es un individuo respecto al grupo al que pertenece en comparación a los demás grupos. Presenta un rango acotado en valor absoluto menor a la unidad. Un valor alto indica que el individuo es muy similar al grupo, y un valor bajo indica que el individuo es muy diferente al grupo. En términos funcionales puede expresarse como:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

donde $a(i)$ es la distancia promedio de la observación i a los objetos de su grupo y $b(i)$ es la distancia promedio de la observación i a los objetos de los demás grupos (distancia intra y entre grupos). Una referencia a considerar a nivel agregado es la silueta promedio para una cantidad de grupos dada, la cual se calcula como la media de las siluetas de todos los grupos. Para obtener una recomendación sobre la cantidad de grupos óptima en este tipo de algoritmos se puede comparar la silueta promedio de cada configuración. Aquella que obtenga un valor más alto será una buena recomendación de la cantidad de grupos.

Dado que 2021 tiene dos relevamientos de datos semestrales a diferencia de los damás años que son anuales, se decidió no tomarlo como año base para aplicar el análisis de cluster. Los datos de 2006 fueron descartados para este análisis por la constatación de datos faltantes en la variable referente al tamaño de la empresa, considerada importante

para diferenciar los grupos. Los análisis de cluster realizados en los años 2011 y 2019 dan como resultado una mejor clasificación de grupos en el año 2011, donde se llega a un mayor valor de silueta promedio y no hay mayores problemas de similaridad dentro de grupos particulares. Se eligen por lo tanto los datos del año 2011 para realizar el análisis de cluster y servir de año base para los análisis posteriores.

Referencias:

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: Springer.
- Matthias, K., & Kane, S. P. (2015). Docker: Up & Running: Shipping Reliable Containers in Production. O'Reilly Media, Inc.
- Peña, D. (2002). Análisis de datos multivariantes (Vol. 24). Madrid: McGraw-hill.
- R Core Team (2023). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. <https://www.R-project.org/>.
- Venables WN, Ripley BD (2002). Modern Applied Statistics with S, Fourth edition. Springer, New York. https://www.stats.ox.ac.uk/pub/MASS4/.
- ARCA-DPSS (s.f.). Manual de Ciencia Abierta — Capítulo Docker. Disponible en: https://arca-dpss.github.io/manual-open-science/docker-chapter.html