Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América

Dirección General de Estudios de Posgrado
Facultad de Ciencias Matemáticas
Unidad de Posgrado

Optimización de árboles de decisión para la estimacion de primas puras en pólizas de seguros de no vida

TESIS

Para optar el Grado Académico de Magíster en Estadística Matemática

AUTOR

Jorge Ivan YUPANQUI DURAN

ASESOR

Mg. Emma Norma CAMBILLO MOYANO

Lima, Perú

2026



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

https://creativecommons.org/licenses/by-nc-sa/4.0/

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

Yupanqui, J. (2026). *Optimización de árboles de decisión para la estimacion de primas puras en pólizas de seguros de no vida*. [Tesis de maestría, Universidad Nacional Mayor de San Marcos, Facultad de Ciencias Matemáticas, Unidad de Posgrado]

Metadatos complementarios

Datos de autor	
Nombres y apellidos	Jorge Ivan Yupanqui Duran
Tipo de documento de identidad	DNI
Número de documento de identidad	46145239
URL de ORCID	https://orcid.org/0009-0005-0859-5533
Datos de asesor	
Nombres y apellidos	Emma Norma Cambillo Moyano
Tipo de documento de identidad	DNI
Número de documento de identidad	15377390
URL de ORCID	https://orcid.org/0000-0003-3173-9425
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	Zoraida Judith Huamán Gutiérrez
Tipo de documento	DNI
Número de documento de identidad	09890094
Miembro del jurado 1	
Nombres y apellidos	Olga Lidia Solano Dávila
Tipo de documento	DNI
Número de documento de identidad	09118212
Miembro del jurado 2	
Nombres y apellidos	Antonio Bravo Quiroz
Tipo de documento	DNI
Número de documento de identidad	10130035
Miembro del jurado 3	
Nombres y apellidos	Ysela Dominga Agüero Palacios
Tipo de documento	DNI

Número de documento de identidad	09059647
Datos de investigación	
Línea de investigación	A.3.2.6. Análisis de Datos y Modelamiento de Problemas de la Sociedad (Empresa, Instituciones, Población locales, regionales y nacionales)
Grupo de investigación	No aplica
Agencia de financiamiento	Sin financiamiento
Ubicación geográfica de la investigación	Universidad Nacional Mayor de San Marcos, Facultad de Ciencias Matemáticas, Unidad de Posgrado País: Perú Departamento: Lima Provincia: Lima Distrito: Lima Urbanización: Ciudad Universitaria Calle: Av. Venezuela s/n cuadra 34 Latitud: -12.056423 Longitud: -77.084333
Año o rango de años en que se realizó la investigación	Junio 2025 - abril 2026
URL de disciplinas OCDE	Estadísticas, Probabilidad https://purl.org/pe-repo/ocde/ford#1.01.03 Ciencias de la computación https://purl.org/pe-repo/ocde/ford#1.02.01 Economía https://purl.org/pe-repo/ocde/ford#5.02.01

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
Universidad del Perú, DECANA DE AMÉRICA

FACULTAD DE CIENCIAS MATEMÁTICAS
VICEDECANATO DE INVESTIGACIÓN Y POSGRADO
UNIDAD DE POSGRADO

ACTA DE SUSTENTACIÓN DE TESIS PARA OPTAR EL GRADO ACADÉMICO

Siendo las 19:30 horas del día jueves 30 de abril de dos mil veintiséis, en el Aula 102 de la Facultad de Ciencias Matemáticas de la Universidad Nacional Mayor de San Marcos, el Jurado Examinador de tesis, presidido por la Dra. Zoraida Judith Huamán Gutiérrez e integrado por los siguientes miembros: Dra. Olga Lidia Solano Dávila (Jurado evaluador), Mg. Antonio Bravo Quiroz(Jurado Informante), Mg. Ysela Dominga Agüero Palacios (Jurado Evaluador) y la Mg. Emma Norma Cambillo Moyano como Miembro Asesora; se reunieron para la sustentación de la tesis titulada: **«OPTIMIZACION DE ÁRBOLES DE DECISIÓN PARA LA ESTIMACION DE PRIMAS PURAS EN PÓLIZAS DE SEGUROS DE NO VIDA»**, presentada por el Br. Jorge Ivan Yupanqui Duran, para optar el Grado Académico de Magíster en Estadística Matemática.

Concluida la exposición, los miembros del Jurado de Tesis procedieron a formular sus preguntas que fueron absueltas por el graduando; acto seguido se procedió a la evaluación correspondiente, según tabla adjunta, habiendo obtenido el Bachiller **JORGE IVAN YUPANQUI DURAN**, el calificativo de Diecisiete (17) Muy Bueno

Habiendo sido aprobada la sustentación de la Tesis, el Jurado Examinador recomienda que el Consejo de Facultad apruebe el otorgamiento del **Grado Académico de Magister en Estadística Matemática** al Bachiller **JORGE IVAN YUPANQUI DURAN**.

Siendo las 20:40 horas, se levantó la sesión, firmando para constancia la presente Acta:

Dra. Zoraida Judith Huamán Gutiérrez
Presidente

Dra. Olga Lidia Solano Dávila
Miembro

Mg. Antonio Bravo Quiroz
Miembro

Mg. Ysela Dominga Agüero Palacios
Miembro

Mg. Emma Norma Cambillo Moyano
Miembro Asesora

UNIVERSIDAD NACIONAL MAYOR DE
SAN MARCOS
Universidad del Perú, Decana de América

Anexo 1

CERTIFICADO DE SIMILITUD

Yo, Emma Norma Cambillo Moyano. en mi condición de asesor acreditado con el Dictamen de la UPG N° **000062-2025-UPG-VDIP-FCM/UNMSM/** Resolución Decanal **N° 001016-2025-D-FCM/UNMCM** de la tesis, cuyo título es "**Optimización de árboles de decisión para la estimación de primas puras en pólizas de seguros de no vida**", presentado por el bachiller **Jorge Ivan Yupanqui Duran** para optar el Grado Académico de Magister en **Estadística Matemática**.

CERTIFICO que se ha cumplido con lo establecido en la Directiva de Originalidad y de Similitud de Trabajos Académicos, de Investigación y Producción Intelectual. Según la revisión, análisis y evaluación mediante la plataforma PIDS, el documento evaluado cuenta con el **0** % de similitud, nivel PERMITIDO para continuar con los trámites correspondientes y para su **publicación en el repositorio institucional.**
Se emite el presente certificado en cumplimiento de lo establecido en las normas vigentes, como uno de los requisitos para la obtención del Grado Académico de Magister en **Estadística Matemática**.

Lima, 12 de Marzo de 2026

Firma del Asesor _____

Nombres y Apellidos del asesor: Emma Norma Cambillo Moyano

DNI: 15377390

La vigencia del presente certificado corresponde a un período de noventa (90) días desde la fecha de su emisión; siempre que no haya observaciones en el texto durante el proceso de revisión por el jurado informante o surja alguna contingencia.

DEDICATORIA

Dedico este logro a mis padres y hermanos, autores de mis primeros pasos y pilares de mi formación. Pero, sobre todo, a Gabriela, el amor de mi vida: por ser la luz que ilumina mi presente y la inspiración detrás de cada uno de mis éxitos.

Jorge Yupanqui Duran

AGRADECIMIENTOS

A mi asesora, Emma Cambillo, por su valiosa orientación, paciencia y por brindarme los conocimientos necesarios para culminar este proyecto con éxito.

A la Facultad de Ciencias Matemáticas y a la Escuela de Estadística, por la formación académica recibida y por ser el espacio donde desarrollé mi pasión por el análisis y el rigor científico.

RESUMEN

Esta tesis evalúa el impacto del remuestreo (sobremuestreo y submuestreo) y de la poda (pre-poda y post-poda) en árboles de decisión para estimar la prima pura en seguros de automóviles (No Vida), mediante la descomposición frecuencia-severidad. El análisis se realiza mediante un enfoque cuantitativo y un diseño no experimental, empleando datos administrativos de seguros de automóviles publicados por la Superintendência de Seguros Privados (SUSEP), correspondientes al mercado brasileño del año 2020. Se contrastan dos hipótesis nulas: H_{01} (el remuestreo no mejora las métricas del modelo) y H_{02} (la poda no incrementa las métricas del modelo). El estudio implementa un modelo de frecuencia mediante un árbol de regresión cuya partición se optimiza con devianza Binomial Negativa e incorpora offset de exposición; asimismo, implementa un modelo de severidad mediante un árbol de regresión cuya partición se optimiza con devianza Gamma.

La evaluación de desempeño se realiza utilizando métricas como devianza, pseudo D^2, MAE, RMSE y WAPE calibrado sobre la prima pura (PP) agregada por segmentos. Los resultados muestran que la pre-poda sin remuestreo logra el mejor equilibrio entre precisión, estabilidad e interpretabilidad. En severidad, esta configuración empata prácticamente con el árbol completo y supera a la post-poda; en frecuencia, ofrece un desempeño competitivo sin los sesgos de compresión de medias que la post-poda puede inducir. El sobremuestreo no mejora las métricas y distorsiona la estimación de la prima pura sin reducir el WAPE; el submuestreo presenta un deterioro en las métricas de devianza y errores en ambos componentes. Se observa una persistente subestimación en las colas de severidad, lo que sugiere explorar familias de distribuciones Tweedie/mixturas Gamma-Pareto y calibraciones por bandas. En síntesis, el modelo que incorpora la pre-poda sin remuestreo provee el mejor desempeño y ofrece lineamientos prácticos para la optimización de la tarificación, equilibrando exactitud e interpretabilidad.

Palabras clave: Prima pura; árboles de decisión; Binomial Negativa; Gamma; poda; remuestreo; seguros de No Vida.

ABSTRACT

This thesis evaluates the impact of resampling (oversampling and undersampling) and pruning (pre-pruning and post-pruning) on decision trees used to estimate the pure premium in automobile insurance (Non-Life), utilizing the frequency-severity decomposition. The analysis follows a quantitative approach and a non-experimental design using administrative automobile insurance data published by the Superintendência de Seguros Privados (SUSEP), corresponding to the Brazilian market in 2020. Two null hypotheses are tested: H_{01} (resampling does not improve model metrics) and H_{02} (pruning does not increase model metrics). The study implements a frequency model using a regression tree whose splits are optimized through Negative Binomial deviance and include an exposure offset; likewise, the severity model is implemented using a regression tree whose splits are optimized through Gamma deviance.

Performance evaluation is conducted using metrics such as deviance, pseudo D^2, MAE, RMSE, and calibrated WAPE on the pure premium (PP) aggregated by segments. The results show that pre-pruning without resampling achieves the best balance between precision, stability, and interpretability. In severity, this configuration practically ties with the full tree and outperforms post-pruning; in frequency, it offers competitive performance without the mean compression biases that post-pruning can induce. Oversampling does not improve metrics and distorts the pure premium estimation without reducing WAPE, while undersampling leads to a deterioration in deviance metrics and errors in both components. A persistent underestimation in the tails of the severity distribution was observed, suggesting the need to explore Tweedie distribution families, Gamma-Pareto mixtures, and band-based calibrations. In summary, the model incorporating pre-pruning without resampling provides the best performance and offers practical guidelines for pricing optimization, balancing accuracy and interpretability.

Keywords: Pure premium; decision trees; Negative Binomial; Gamma; pruning; resampling; Non-Life insurance.

ÍNDICE

CAPÍTULO I. INTRODUCCIÓN 1
1.1. PLANTEAMIENTO DEL PROBLEMA 3
1.1.1. IDENTIFICACIÓN Y DESCRIPCIÓN DEL PROBLEMA DE ESTUDIO 3
1.1.2. FORMULACIÓN DEL PROBLEMA 4
1.2. OBJETIVOS 5
1.2.1. OBJETIVO GENERAL 5
1.2.2. OBJETIVOS ESPECÍFICOS 5
1.3. JUSTIFICACIÓN Y ALCANCES 6
1.3.1. JUSTIFICACIÓN 6
1.3.2. ALCANCE 7
CAPÍTULO II. MARCO TEÓRICO 9
2.1. ANTECEDENTES DE LA INVESTIGACIÓN 9
2.2. BASES TEÓRICAS Y DEFINICIÓN DE TÉRMINOS 11
2.2.1. TARIFICACIÓN EN SEGUROS DE NO VIDA 11
2.2.2. ÁRBOLES DE DECISIÓN 13
2.2.3. FUNCIONES DE PÉRDIDA EN ÁRBOLES DE REGRESIÓN 19
2.2.4. MANEJO DEL DESEQUILIBRIO DE DATOS 26
2.2.5. PODA PARA EVITAR EL SOBREAJUSTE 29
2.2.6. EVALUACIÓN DE LOS MODELOS 31
2.3. HIPÓTESIS 36
CAPÍTULO III. DISEÑO METODOLÓGICO 37
3.1. TIPO DE INVESTIGACIÓN 37
3.2. NIVEL DE INVESTIGACIÓN 37
3.3. MÉTODOS DE TRABAJO 38
3.4. POBLACIÓN Y VARIABLES 39
3.5. TÉCNICAS E INSTRUMENTOS DE RECOLECCIÓN DE DATOS 40
3.6. TÉCNICAS E INSTRUMENTOS DE ANÁLISIS Y PROCESAMIENTO 40
3.7. ETAPAS DE INTERVENCIÓN DEL ESTUDIO 42
CAPÍTULO IV. ANÁLISIS 44
4.1. DESCRIPTIVOS Y EXPLORATORIOS 44
4.2. MODELO DE SEVERIDAD 51
4.3. MODELO DE FRECUENCIA 62
4.4. MODELO PRIMA PURA 74

CAPÍTULO V. RESULTADOS 77
CAPÍTULO VI. CONCLUSIONES 81
CAPÍTULO VII. RECOMENDACIONES 83
REFERENCIAS Y ANEXO 85

LISTA DE FIGURAS

Figura 1: Ejemplos de árboles de decisión / de regresión. 14
Figura 2: Predichos vs Reales Modelo Severidad (árbol Gamma; paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under); línea de referencia y = x). 55
Figura 3: Residuos vs Predichos Modelo Severidad (árbol Gamma; paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under); residuo = y − ŷ) 57
Figura 4: Curvas de aprendizaje Modelo Severidad (devianza media Gamma vs tamaño muestral; entrenamiento (azul) y validación/prueba (naranja); paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under)) 58
Figura 5: Validación cruzada Modelo Severidad (K = 3 folds con semillas fijas; métrica: devianza media Gamma por observación) 60
Figura 6: Importancia de Variables Modelo Severidad (criterio: reducción de devianza Gamma; top 10 predictores) 61
Figura 7: Predichos vs Reales Modelo Frecuencia (árbol Binomial Negativa; paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under); línea de referencia y = x) 66
Figura 8: Residuos vs Predichos Modelo Frecuencia (árbol Binomial Negativa; paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under); residuo = y − ŷ) 68
Figura 9: Curvas de aprendizaje Modelo Frecuencia (devianza media Binomial Negativa vs tamaño muestral; entrenamiento (azul) y validación/prueba (naranja); paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under)) 70
Figura 10: Validación cruzada Modelo Frecuencia (K = 3 folds con semillas fijas; métrica: devianza media Binomial Negativa por observación) 72
Figura 11: Importancia de Variables Modelo Frecuencia (criterio: reducción de devianza Binomial Negativa; top 10 predictores) 73

LISTA DE TABLAS

Tabla 1: Año de Vehículo - conteo de siniestros (Ng) y pérdida agregada (Lg) por rango de año del vehículo 45
Tabla 2: Tipo de Vehículo - conteo de siniestros (Ng) y pérdida agregada (Lg) por tipo del vehículo 46
Tabla 3: Marcas de Vehículo - conteo de siniestros (Ng) y pérdida agregada (Lg) por marcas según país del vehículo 47
Tabla 4: Región - conteo de siniestros (Ng) y pérdida agregada (Lg) por densidad poblacional acorde a la región del vehículo 48
Tabla 5: Edad - conteo de siniestros (Ng) y pérdida agregada (Lg) por edad del asegurado del vehículo 49
Tabla 6: Género - conteo de siniestros (Ng) y pérdida agregada (Lg) por género del asegurado del vehículo 50
Tabla 7: Exposición - conteo de siniestros (Ng) y pérdida agregada (Lg) por exposición del vehículo 50
Tabla 8: Métricas principales de Modelo Severidad (métricas: Devianza Gamma total (suma), Pseudo D^2, MAE, RMSE; configuraciones: completo/post/pre × sin remuestreo/sobremuestreo/submuestreo) 53

Tabla 9: Métricas principales Modelo Frecuencia (métricas: Devianza Binomial Negativa total, Pseudo D^2, MAE, RMSE; configuraciones: pre/completo/post × sin remuestreo/sobremuestreo/submuestreo) 64
Tabla 10: Resultados por Modelo Prima Pura 75

GLOSARIO

A. Términos Actuariales

- **Prima pura (PP) —** Pérdida esperada por unidad de exposición (sin recargos): $\pi(x) = \lambda(x)\mu(x)$. Por póliza–periodo*:* $PP_i = e_i\,\pi(x_i)$.
- **EPP (Expected Pure Premium) —** Prima pura agregada por segmento: EPP_g^{tot} — $\sum_{i\in g} P\,P_i$ (total esperado del segmento)
- **Exposición** $(\boldsymbol{e})$**—** Tiempo/fracción asegurada (p. ej., póliza-año); escala la frecuencia.
- **Póliza-año —** Unidad: una póliza expuesta un año (o fracción).
- **Siniestro** $(\boldsymbol{N})$**—** Evento cubierto reclamado; su conteo es la respuesta en frecuencia.
- **Frecuencia (observada) —** Siniestros por unidad de exposición: $F = N/e$.
- **Frecuencia (modelo) —** Tasa esperada por unidad de exposición (con exposición e_{it} observada): $\lambda(x) = \frac{\mathbb{E}[N_{it}|X_{it}=x,\,e_{it}]}{e_{it}}$, equivalentemente $\mathbb{E}[\,N_{it} \mid X_{it} = x, e_{it}\,] = e_{it}\,\lambda(x)$.
- **Severidad (observada) —** Monto medio por siniestro en una póliza–periodo: $Y_{it} := \frac{L_{it}}{N_{it}}, N_{it} > 0$ (a nivel agregado por grupo: L/N).
- **Severidad (modelo) —** Costo esperado por siniestro condicionado a ocurrencia: $\mu(x) = \mathbb{E}\,[Y_{it} \mid N_{it} > 0,\, X = x]$. (Nota: se reserva $\mu^{(S)}$ para severidad; en frecuencia, la media del conteo se denota $\mu^{(N)}$)
- **Pérdida agregada (** $\boldsymbol{L}$**) —** Monto total indemnizado por póliza–periodo (observado en la base). Cuando $N_{it} > 0$, la severidad media se define como $Y_{it} = L_{it}/N_{it}$.
- **Segmento / Grupo tarifario —** Conjunto definido por claves (p. ej., región, sexo, edad, año vehículo).
- **Calibración global (factor** $\boldsymbol{c_{cal}}$**)**: razón entre el total observado y el total predicho no calibrado; se utiliza para reanclar el nivel global de la prima pura estimada sin alterar relatividades.
- **WAPE calibrado (Error porcentual absoluto ponderado)** — Error porcentual absoluto ponderado a nivel agregado, calculado sobre totales por segmento después de aplicar la calibración global.
- **Cartera —** Conjunto de pólizas analizadas en el estudio.

B. Términos estadísticos / aprendizaje automático

- **Aprendizaje automático —** Capacidad de los algoritmos para aprender patrones a partir de los datos sin ser programados explícitamente para cada regla de decisión.
- **Árbol de decisión (regresión) —** Segmenta el espacio con cortes; cada hoja predice un valor constante.

- **Nodo / Hoja / Corte (*node / leaf / split*) —** El corte se define por un umbral (*threshold*).
- **Árbol completo (*full tree*) —** Árbol sin poda posterior durante el crecimiento.
- **Pre-poda —** Reglas que impiden cortes débiles (profundidad máx., mínimo de muestras/exposición, ganancia mínima).
- **Post-poda —** Sustituye subárboles por hojas si no mejoran en validación.
- **Sobreajuste / Subajuste (*Overfitting/ Underfitting)* —** Sobreajuste: situación en la que el modelo captura patrones espurios o variabilidad propia del conjunto de entrenamiento, reduciendo su capacidad de generalización. Subajuste: el modelo es muy simple y no capta los patrones importantes.
- **Curvas de aprendizaje (*learning curves*) —** Error en entrenamiento/validación al aumentar el tamaño muestral.
- **Validación cruzada (k pliegues) (*k-fold cross-validation*) —** Rotación de pliegues para estimar generalización.
- **Residuos —** Diferencias $r = y - \hat{y}$; diagnostican sesgos y heterocedasticidad.
- **Heterocedasticidad —** Varianza no constante de los errores o residuos condicionada al nivel de las covariables o de los valores ajustados.
- **Hiperparámetros —** Parámetros de diseño (p. ej., *max_depth*, *min_samples*, *min_gain_rel*).
- **Importancia de variables —** Suma de reducciones de devianza atribuibles a cada variable en los *splits*.
- **Devianza** $(\boldsymbol{D})$**—** Medida basada en verosimilitud; menor $D \Rightarrow$ mejor ajuste.
- **Pseudo-$\boldsymbol{D^2}$ —** Proporción de devianza explicada: $1 - D_{\text{modelo}}/D_{\text{nulo}}$.
- **MAE (Mean Absolute Error) —** Promedio de los errores absolutos.
- **RMSE (Root Mean Squared Error) —** Raíz de la media de errores cuadrados.
- **Sobredispersión (conteos) —** Exceso de variabilidad respecto a Poisson: $\mathrm{Var}(Y \mid X) > \mu^{(N)}$; sugiere un modelo Binomial Negativa.
- **Offset —** Término incorporado al predictor lineal con coeficiente fijo igual a uno. En modelos de frecuencia con enlace logarítmico, permite ajustar la media esperada del conteo por la exposición: $\log\, \mu^{(N)} = \log\, e + \eta(x)$, donde e representa la exposición de la póliza.
- **Estratificación por cuantiles** — Partición del objetivo en intervalos por cuantiles, por ejemplo deciles, para crear estratos con tamaños similares y evaluar o equilibrar el desempeño por grupos.
- **Sobremuestreo / Submuestreo (oversampling / undersampling)** — Aumentar casos (con reemplazo) o reducir casos (sin reemplazo) en entrenamiento para equilibrar la muestra.

CAPÍTULO I. INTRODUCCIÓN

La tarificación en seguros de No Vida busca estimar primas puras que reflejen con fidelidad la frecuencia y la severidad esperadas de los siniestros, garantizando precios adecuados. En el ramo de automóviles, esta tarea exige modelos capaces de capturar patrones heterogéneos del riesgo —combinando características del conductor, del vehículo y de la póliza— sin perder interpretabilidad, requisito clave en contextos regulatorios y de gestión. Tal como indican Diana et al. (2019), el aumento de la disponibilidad de datos y el carácter complejo de los problemas de clasificación y regresión en seguros No Vida hacen que los métodos de aprendizaje automático — que no dependen de supuestos paramétricos rígidos— sean cada vez más relevantes en tarificación. Entre ellos, los árboles de decisión destacan por su capacidad de segmentar no linealidades e interacciones mediante reglas simples y comprensibles, lo que facilita explicar cómo se forman los segmentos tarifarios y por qué una póliza se ubica en determinado perfil de riesgo.

No obstante, la aplicación de árboles en tarificación enfrenta dos desafíos operativos. Primero, el desequilibrio de datos: en muchas carteras la frecuencia de siniestros es baja, por lo que la variable de frecuencia N presenta una alta proporción de ceros; si no se controla, el aprendizaje puede concentrarse en la región dominante $N = 0$ y perder sensibilidad para capturar patrones asociados a $N > 0$. Segundo, el sobreajuste: los árboles tienden a crecer en exceso si no se controla su complejidad, perdiendo capacidad de generalización. A estos retos se suma la necesidad de alinear el criterio de partición del árbol con la naturaleza estadística de las variables de interés: conteos para frecuencia (Poisson o Binomial Negativa ante sobredispersión, con *offset* de exposición) y montos positivos para severidad (Gamma) (Frees, 2010; Hilbe, 2011). La elección entre Poisson y Binomial Negativa se sustenta en el diagnóstico de dispersión mediante el estadístico de Pearson $\hat{\phi}_P$ (Sección 2.2.3); en la aplicación empírica se observa sobredispersión, por lo que se adopta Binomial Negativa en la componente de Frecuencia. En este contexto, usar devianzas específicas como función de pérdida dentro del árbol —en lugar de métricas genéricas como MSE— permite que la partición responda a la distribución actuarial relevante (Cadena, 2010).

Esta tesis aborda ese espacio con el siguiente objetivo general: optimizar modelos de árboles de decisión para mejorar el desempeño en la estimación de primas puras en seguros de automóviles, atendiendo explícitamente al desequilibrio de datos y al control de complejidad. Para ello se plantean dos objetivos específicos: (i) evaluar el efecto de técnicas de remuestreo, específicamente submuestreo y sobremuestreo, en el aprendizaje

de eventos; y (ii) determinar la influencia de la poda, mediante pre-poda y post-poda por reducción de error, y de la sintonía de hiperparámetros en la generalización del modelo.

Metodológicamente, se adopta una descomposición frecuencia–severidad: se entrena un árbol para frecuencia (criterio de devianza Poisson o Binomial Negativa, según la dispersión observada y con *offset* de exposición) y un árbol para severidad (criterio de devianza Gamma), y luego se combina la predicción de ambos para obtener la prima pura a nivel de póliza. Para evitar sobreajuste, se aplican estrategias de poda, y la evaluación se realiza con devianza, pseudo-D^2, MAE, RMSE y Error Porcentual Agregado Ponderado (EPA ponderado/WAPE) a nivel de cartera, priorizando métricas coherentes con la práctica tarifaria.

El aporte de la investigación es principalmente metodológico y operativo: (a) un marco de entrenamiento de árboles explícitamente alineado con distribuciones actuariales para frecuencia y severidad de siniestros —Poisson o Binomial Negativa con *offset* de exposición para conteos y Gamma para montos positivos— mediante el uso de sus devianzas específicas como función de pérdida; (b) un protocolo de control de complejidad basado en poda y validación cruzada, enfocado en mantener parsimonia e interpretabilidad; y (c) una comparación estructurada de estrategias de remuestreo para contextos de baja frecuencia, articulada con métricas estándar del sector. Todo ello se plantea con énfasis en trazabilidad: decisiones y supuestos quedan documentados para su verificación por parte de áreas técnicas y de gobierno de datos.

La presente investigación se divide en siete capítulos: el Capítulo I (Planteamiento del problema) contextualiza la investigación, delimita el problema y formula la pregunta central; además, precisa el objetivo general, los objetivos específicos y los alcances del estudio. El Capítulo II (Marco teórico) fundamenta la tarificación en seguros de No Vida y revisa los métodos empleados, con énfasis en los árboles de decisión. Desarrolla las funciones de pérdida por devianza adecuadas a cada variable (Poisson/Binomial Negativa para frecuencia y Gamma para severidad), aborda el desequilibrio de datos y la poda, y fija la notación utilizada en el resto del documento. El Capítulo III (Diseño metodológico) define el tipo y nivel de investigación, describe la preparación de los datos (limpieza, codificación y tratamiento de atípicos), detalla las estrategias de remuestreo, los criterios de partición basados en devianza y los procedimientos de poda y validación para controlar la complejidad y garantizar la generalización. El Capítulo IV (Análisis) desarrolla los descriptivos y la construcción de los modelos de frecuencia y severidad, así como su integración para estimar la prima pura. El Capítulo V (Resultados) expone los experimentos y la evaluación comparativa, reportando el desempeño según las métricas definidas y

analizando los hallazgos desde la perspectiva actuarial. El Capítulo VI (Conclusiones) sintetiza los principales hallazgos, discute sus implicancias para la práctica de tarificación e identifica líneas de trabajo futuro. Finalmente, el Capítulo VII (Recomendaciones) formula orientaciones prácticas para la implementación de los modelos en entornos actuariales y académicos.

1.1. PLANTEAMIENTO DEL PROBLEMA

1.1.1. IDENTIFICACIÓN Y DESCRIPCIÓN DEL PROBLEMA DE ESTUDIO

El aprendizaje supervisado mediante árboles de decisión es un enfoque no paramétrico ampliamente utilizado en clasificación y regresión —incluidas las ciencias actuariales— por su simplicidad, interpretabilidad y capacidad para modelar relaciones complejas entre variables. La literatura clásica y revisiones modernas coinciden en que los árboles ofrecen un balance atractivo entre desempeño predictivo y claridad del modelo, lo que favorece su adopción en contextos donde la transparencia es esencial (Breiman et al., 1984; Hastie et al., 2009; Scikit-learn Developers, 2025).

Un árbol de decisión posee una estructura jerárquica compuesta por nodos internos (pruebas sobre variables de entrada), ramas (resultados de esas pruebas) y nodos hoja (salidas: clase en clasificación o valor en regresión). Este esquema tipo diagrama de flujo facilita visualizar cómo se toman decisiones y favorece la comunicación con audiencias no técnicas (Safavian & Landgrebe, 1991; Hastie et al., 2009).

En seguros de No Vida, múltiples trabajos han resaltado la utilidad del aprendizaje automático —incluidos los árboles— para tarificación y reservas. A pesar de su popularidad y ventajas, los árboles de decisión presentan desafíos que deben abordarse en la práctica. Según Zorman et al. (1997), el aprendizaje de árboles de decisión suele mostrar muy buenos resultados en diversos entornos teóricos; sin embargo, en la práctica a menudo es imposible contar con el número deseado de observaciones de entrenamiento representativas por varias razones.

Esto implica que la falta de datos suficientes o adecuados puede limitar la capacidad del árbol para generalizar. Uno de los problemas más comunes al entrenar árboles con datos reales en tarificación es el desbalance distributivo. En frecuencia, el conteo de siniestros por póliza–periodo N suele presentar exceso de ceros y masa concentrada en valores bajos (eventos raros). En severidad, los montos positivos suelen ser asimétricos y con colas superiores relevantes. Si no se atiende este desbalance, el entrenamiento puede

concentrarse en la región más densa (principalmente $N = 0$ y montos pequeños) y debilitar la capacidad del modelo para capturar patrones en regiones menos frecuentes pero actuarialmente importantes. Por ello, se consideran estrategias de remuestreo como submuestreo/sobremuestreo (o enfoques equivalentes por estratos/bins), aplicadas solo al conjunto de entrenamiento, con el objetivo de reforzar la señal en $N > 0$ y/o en colas de severidad, sin reinterpretar el problema como uno de clasificación (Cieslak & Chawla, 2008).

Otro desafío clave es el sobreajuste (overfitting) a los datos de entrenamiento. Cuando un árbol se ajusta en exceso puede depender de variables irrelevantes o de patrones espurios del conjunto de entrenamiento; aunque parezca óptimo en entrenamiento, su rendimiento se deteriora en datos nuevos. Por ello, es fundamental controlar el sobreajuste durante la inducción del árbol y verificar la generalización mediante validación cruzada. Mecanismos como la poda — pre-poda con restricciones estructurales y post-poda por reducción de error — resultan imprescindibles, pues eliminan ramas poco informativas y reducen la complejidad del árbol, conservando las divisiones basadas en atributos relevantes (Scikit-learn Developers, 2025).

En resumen, si los datos de partida no se equilibran adecuadamente y los modelos no se regulan para evitar el sobreajuste, los árboles resultantes pueden ser subóptimos: perder variables predictoras relevantes, presentar inestabilidad ante cambios en los datos y, en última instancia, exhibir menor capacidad predictiva sobre nuevos casos. Para garantizar modelos más robustos y trazables, es necesario combinar una preparación balanceada de los datos con un control riguroso de la complejidad mediante poda y validación.

1.1.2. FORMULACIÓN DEL PROBLEMA

Derivado de lo anterior, se plantea la siguiente cuestión central de investigación:

- ¿De qué manera pueden optimizarse los modelos de árboles de decisión — mediante técnicas de remuestreo y de poda— para incrementar la precisión en la estimación de la prima pura en pólizas de seguros de no vida (automóviles) del año 2020?

Para abordar adecuadamente esta pregunta general, se han definido las siguientes subpreguntas específicas, cada una orientada a un aspecto clave del problema de investigación:

- ¿En qué medida las técnicas de submuestreo y sobremuestreo mejoran el desempeño de los modelos de frecuencia (Binomial Negativa con *offset* de exposición) y de severidad (Gamma) y, en consecuencia, la precisión de la *prima pura*, respecto de un modelo base?
- ¿Cómo inciden las técnicas de *pre-poda* y de *post-poda* en el sobreajuste de los modelos de frecuencia (Binomial Negativa con *offset*) y severidad (Gamma) y en la precisión de la *prima pura*, en comparación con el modelo base?

Cada subpregunta refleja un objetivo particular del estudio y en conjunto permiten descomponer el problema general en partes manejables para su análisis. La primera se enfoca en los métodos de preparación y balanceo del conjunto de datos, mientras que la segunda se centra en los métodos de optimización del modelo una vez entrenado (poda). Respondidas en conjunto, estas cuestiones proporcionarán la base para responder la pregunta principal de investigación.

Para responder las preguntas planteadas, se ejecutan simulaciones de escenarios que combinan: (i) niveles de remuestreo (submuestreo y sobremuestreo) y (ii) configuraciones de poda.

1.2. OBJETIVOS

1.2.1. OBJETIVO GENERAL

Evaluar el efecto del remuestreo de datos y la poda de árboles de decisión en la precisión de los modelos de frecuencia (Binomial Negativa con *offset*) y severidad (Gamma) para sustentar una estimación robusta de la prima pura en pólizas de seguros de no vida (automóviles) del año 2020, respecto de un modelo base sin remuestreo ni poda.

1.2.2. OBJETIVOS ESPECÍFICOS

- Determinar si las técnicas de submuestreo y sobremuestreo mejoran la devianza específica (Binomial Negativa con *offset* de exposición para frecuencia; Gamma para severidad) y el pseudo-D^2 de ambos submodelos, y mejoran la precisión de la prima pura, respecto de un modelo base sin remuestreo, en pólizas de automóviles 2020.

- Evaluar si la pre-poda y la post-poda reducen el sobreajuste, y si mejoran la devianza específica (Binomial Negativa para frecuencia con *offset* de exposición; Gamma para severidad), el pseudo-D^2 y la precisión de la prima pura, respecto del árbol base sin poda, en pólizas de automóviles 2020.

1.3. JUSTIFICACIÓN Y ALCANCES

1.3.1. JUSTIFICACIÓN

El análisis de modelos no paramétricos como los árboles de decisión en el ámbito de los seguros de automóviles (No Vida) ofrece la oportunidad de mejorar significativamente la precisión en la estimación de primas puras (primas de riesgo “puras” sin recargos comerciales). Una estimación correcta de estas primas tiene un importante valor tanto actuarial como económico para los diversos actores clave que participan en el sector asegurador.

Desde la perspectiva de las aseguradoras, contar con modelos que predigan con mayor exactitud el coste esperado de los siniestros (primas puras) facilita una valoración más exacta del riesgo asociado a cada póliza. Esto redunda en la posibilidad de ajustar las primas que se cobran a los asegurados de manera más adecuada, evitando tanto la subestimación del riesgo (que podría llevar a pérdidas financieras) como la sobreestimación (que encarecería el producto y podría ahuyentar a clientes). Con valoraciones más precisas, las compañías aseguradoras pueden diseñar y ofrecer productos mejor calibrados al perfil de riesgo y a las necesidades de sus clientes, fortaleciendo así su competitividad en el mercado. Además, al calcular correctamente las primas, pueden asegurar la suficiencia técnica de primas y minimizar desviaciones o insuficiencias en las reservas destinadas a cubrir siniestros, contribuyendo a la solvencia y salud financiera de la entidad.

Desde la perspectiva de los asegurados (clientes), también existen beneficios claros. Una tarificación más precisa se traduce en primas más equitativas y ajustadas al riesgo individual que representa cada asegurado. En la práctica, esto implica que aquellos clientes con menor riesgo (por ejemplo, conductores con buen historial, bajos índices de siniestralidad, etc.) podrían acceder a primas más bajas o justas, mientras que los de mayor riesgo pagarían primas acordes a su probabilidad de siniestro. Esta personalización mejora la equidad del sistema y puede incrementar la satisfacción y confianza de los clientes en la aseguradora, al percibir que se les cobra un precio razonable por la cobertura recibida.

Adicionalmente, la transparencia derivada de usar modelos interpretables como los árboles de decisión favorece la comprensión de los factores que influyen en el precio del seguro, lo cual puede incrementar la confianza en la relación asegurador–asegurado.

En un plano más amplio, estimaciones más precisas de las primas puras contribuyen a la estabilidad del sector asegurador, y esta, a su vez, repercute en la estabilidad económica general. Un sector asegurador solvente y correctamente provisionado es capaz de cumplir sus compromisos de indemnización, generando confianza en los consumidores y en los mercados financieros. Asimismo, un acceso adecuado al seguro (con primas calculadas rigurosamente y, por tanto, precios competitivos pero suficientes) permite que tantos individuos como empresas operen con mayor seguridad, sabiendo que están protegidos ante riesgos. Esto reduce la incertidumbre económica y puede evitar costes imprevistos no cubiertos que tendrían que ser asumidos por el propio afectado o incluso por el Estado. En suma, la correcta gestión de primas y riesgos por parte de las aseguradoras disminuye la probabilidad de escenarios extremos (por ejemplo, insolvencias masivas de aseguradoras o necesidad de rescates) y reduce la necesidad de intervenciones gubernamentales, favoreciendo un uso más eficiente de los recursos públicos en otras áreas prioritarias.

En definitiva, este estudio se justifica porque propone métodos mejorados para el cálculo de tarifas de seguro basadas en evaluaciones de riesgo más precisas, lo cual beneficia a las aseguradoras (mediante una tarificación más ajustada al riesgo real y una mejor gestión del negocio), beneficia a los asegurados (mediante primas más justas y mayor transparencia) y beneficia al sistema económico en general (mediante un sector asegurador más sólido y equilibrado). Aportar rigor técnico a la estimación de primas, incorporando técnicas modernas de aprendizaje automático con las debidas precauciones, representa un avance tanto para la ciencia actuarial como para la práctica del negocio asegurador.

1.3.2. ALCANCE

Este estudio analiza cómo la combinación de técnicas de equilibrado de datos y poda de árboles de decisión mejora la precisión en la estimación de la prima pura en seguros de automóviles de No Vida. Se parte de datos administrativos de seguros de automóviles publicados por la Superintendência de Seguros Privados (SUSEP), correspondientes al mercado brasileño del año 2020 con información de pólizas, características de asegurados y siniestros. Los datos se someten a una limpieza rigurosa (tratamiento de atípicos y

ausentes, recodificación de variables categóricas, verificación de coherencia temporal y de negocio).

La exploración inicial permite detectar patrones, distribuciones y relaciones entre variables, así como cuantificar el desbalance de reclamaciones debido a la alta proporción de pólizas sin siniestros. Con base en este diagnóstico, se aplican estrategias de remuestreo para reequilibrar la información en entrenamiento: submuestreo de estratos densos y sobremuestreo de estratos escasos (por ejemplo, observaciones con $N > 0$ en frecuencia o montos en colas en severidad), exclusivamente en el conjunto de entrenamiento. Los conjuntos de validación externa y prueba se mantienen con la distribución original de la cartera, sin aplicar remuestreo, a fin de evaluar el desempeño del modelo en condiciones representativas de la población observada.

El modelado se aborda mediante árboles de regresión entrenados de forma independiente para frecuencia y severidad:

- Frecuencia (F): árbol con función de pérdida de devianza Binomial Negativa (NB) e inclusión de *offset* de exposición.
- Severidad (S): árbol con función de pérdida de devianza Gamma, condicional en la ocurrencia de siniestro.

Para controlar el sobreajuste se emplea pre-poda (profundidad máxima, tamaño mínimo de hoja y mínima exposición por hoja) y post-poda por reducción de error, eliminando ramas cuyo aporte a la reducción de devianza sea marginal.

Las predicciones óptimas de F y S se combinan hoja a hoja para obtener la prima pura $\pi = F \times S$. La calidad de los modelos se evalúa con validación cruzada k-fold y métricas complementarias: MAE, RMSE, devianza y pseudo-D^2 en frecuencia y severidad; y, sobre PP, WAPE (también denominado Error Porcentual Agregado Ponderado, EPA Ponderado). La evaluación se apoya en análisis gráficos (dispersión predicho-real, residuos, curvas de aprendizaje, importancia de variables basada en reducción de devianza, resultados de la CV).

Se evalúa, mediante evidencia estadística y visual, si la combinación de remuestreo y poda incrementa la capacidad predictiva frente a modelos de referencia que no utilizan estas técnicas. Los resultados brindarán lineamientos prácticos para actuarios y desarrolladores sobre cómo preprocesar bases desbalanceadas y cómo configurar árboles robustos, obteniendo primas más precisas, estables e interpretables y fortaleciendo la determinación de precios del seguro automotor de No Vida.

CAPÍTULO II. MARCO TEÓRICO

2.1. ANTECEDENTES DE LA INVESTIGACIÓN

La estimación de primas puras con árboles de decisión se ha consolidado en el sector asegurador por su interpretabilidad y su capacidad para integrar variables categóricas y numéricas. No obstante, su desempeño depende de dos frentes metodológicos decisivos: el tratamiento riguroso del desbalance distributivo (exceso de ceros en frecuencia y asimetría/colas en severidad) y el control de complejidad mediante poda. Ignorar cualquiera de estos aspectos puede comprometer la precisión y la utilidad del modelo en producción, especialmente cuando el aprendizaje se concentra en la región más densa de los datos (He & García, 2009).

Estudios Previos sobre el Desequilibrio de Datos

En seguros es habitual que los eventos de interés sean raros en relación con la cartera. En tarificación, el riesgo es que el ajuste quede dominado por la región $N = 0$ y por montos pequeños, afectando la estimación en $N > 0$ y en colas de severidad. Por ello, el desempeño debe evaluarse con métricas coherentes con conteos y montos (p. ej., devianza NB/Gamma, pseudo-D^2, MAE/RMSE y métricas agregadas sobre prima pura), más que con medidas de acierto binario.

Conscientes de este problema, la literatura sobre aprendizaje con datos desbalanceados propone estrategias tanto a nivel de datos (p. ej., sobremuestreo) como a nivel de algoritmo (p. ej., ponderación de pérdidas y boosting). Aunque varios estudios se formulan en términos de detección binaria (presencia/ausencia de evento), el principio subyacente es el mismo en tarificación: evitar que el ajuste quede dominado por la región más frecuente (p. ej., N = 0 y montos pequeños) y preservar información relevante para N > 0 y para colas de severidad. En seguros de automóviles, Hanafy & Ming (2021) reportan mejoras al reequilibrar la ocurrencia de reclamaciones poco frecuentes; en esta tesis, dicho enfoque se incorpora de forma coherente con un problema de regresión (conteos y montos), evaluando el desempeño con métricas propias de este marco (devianza NB/Gamma, pseudo-D² y métricas agregadas sobre prima pura).

De forma complementaria, en dominios como fraude en salud se observa que, cuando la ocurrencia del evento es extremadamente rara, los métodos estándar tienden a no capturar patrones en la zona de interés; por ello se recomiendan estrategias específicas de balanceo y validación para revelar estructuras que, de otro modo, pasarían desapercibidas

(Shamitha & Ilango, 2019). En términos generales, el desbalance es inherente a múltiples aplicaciones reales y abordarlo es una prioridad actual en aprendizaje automático (Chen et al., 2024).

Estudios Previos sobre la poda de árboles de decisión

En paralelo al manejo de los datos desequilibrados, la poda es crucial. Podar un árbol consiste en simplificar su estructura eliminando ramas con bajo aporte informativo, con el objetivo de mejorar la generalización y la interpretabilidad. Brunello et al. (2017) señalan que una poda adecuada no solo reduce el tamaño del árbol, sino que también incrementa la capacidad del modelo para generalizar. La evidencia empírica respalda esta afirmación: Windeatt y Ardeshir (2001) observaron que árboles podados resultan más compactos y con frecuencia ofrecen mejor rendimiento fuera de muestra que árboles no podados más complejos.

En el terreno de seguros, investigaciones recientes subrayan la importancia de contar con árboles bien optimizados, especialmente en tareas de tarificación (p. ej., pólizas de salud) donde modelos sencillos, pero correctamente ajustados permiten calcular primas con mayor precisión y evitan distorsiones por sobreajuste (Ridzuan et al., 2024). Ya desde trabajos clásicos se advertía que sacrificar un poco de exactitud por simplicidad puede resultar beneficioso: Bohanec & Bratko (1994) mostraron que la simplificación mediante poda vuelve el modelo más robusto sin pérdidas sustanciales en poder predictivo. En síntesis, optimizar los árboles mediante poda no es un paso cosmético, sino una estrategia esencial para mejorar precisión, estabilidad e interpretabilidad, asegurando que estimaciones como la prima pura sean lo más confiables posible.

A la luz de estos antecedentes, es pertinente repasar los fundamentos teóricos y las definiciones de términos clave que sustentan el estudio. A continuación, se abordan conceptos relacionados con la tarificación en seguros generales, los árboles de decisión y las métricas empleadas, antes de establecer las hipótesis de investigación.

2.2. BASES TEÓRICAS Y DEFINICIÓN DE TÉRMINOS

2.2.1. TARIFICACIÓN EN SEGUROS DE NO VIDA

En los seguros generales (No Vida), la tarificación técnica se basa en la estimación de la pérdida esperada asociada al riesgo cubierto, comúnmente denominada prima pura o prima de riesgo (Werner & Modlin, 2016; Ohlsson & Johansson, 2010). Dado el enfoque actuarial del problema, el punto de partida empírico es una Tabla de Experiencia (tabla de exposición y siniestros), construida a partir del seguimiento del grupo de riesgo durante la ventana de observación (Werner & Modlin, 2016). En sentido operativo, esta tabla puede considerarse análoga a una "tabla de vida" únicamente como analogía organizacional de exposición y ocurrencia de eventos (sin implicar análisis de supervivencia, censura ni funciones de riesgo/hazard).

Sea la base longitudinal un conjunto de observaciones por póliza–periodo $(i\,,t)$. Para cada observación se registra: exposición $e_{it} > 0$, número de siniestros N_{it}, montos individuales $Y_{itj} > 0$ para $j = 1, \dots, N_{it}$, y la pérdida agregada del periodo:

$$L_{it} = \sum_{j=1}^{N_{it}} Y_{itj} \text{ (con } L_{it} = 0 \text{ si } N_{it} = 0) \tag{1}$$

En la base empleada, la información monetaria está registrada a nivel póliza–periodo como pérdida agregada L_{it} y conteo N_{it}. Por ello, la severidad observada por fila se define operativamente como $Y_{it} = L_{it}/N_{it}$ para $N_{it} > 0$.

Agregando en el tiempo para la póliza i, se define:

$$e_i = \sum_t e_{it}\,, \qquad N_i = \sum_t N_{it}\,, \qquad L_i = \sum_t L_{it} \tag{2}$$

En esta tesis, aunque la notación se presenta en forma general como póliza–periodo $(i\,,t)$, la base empírica corresponde a un solo periodo (año 2020), por lo que cada registro es una póliza-año. En adelante, se suprime t y se trabaja con i como unidad de observación, entendiendo $e_i = e_{i,2020}$, $N_i = N_{i,2020}$ y $L_i = L_{i,2020}$.

Bajo esta estructura de experiencia, es habitual descomponer el riesgo en frecuencia y severidad (Su & Bai, 2020). En particular:

- **Frecuencia observada:** para la póliza i,

$$F_i = \frac{N_i}{e_i}, \qquad e_i > 0 \tag{3}$$

donde N_i es el número total de siniestros en el horizonte observado y e_i la exposición total (póliza-año u otra unidad).

- **Tasa esperada de siniestros (modelo):**

$$\lambda_i = \frac{\mathbb{E}[\, N_i \mid x_i \,]}{e_i} \tag{4}$$

donde x_i es el vector de características de la póliza i.

- **Severidad esperada (modelo):**

$$\mu_i = \mathbb{E}[\, Y_i \mid N_i > 0, X_i = x_i \,], \qquad Y_i := \frac{L_i}{N_i}(N_i > 0) \tag{5}$$

- **Costo agregado por póliza (horizonte completo):**

$$L_i = \sum_{j=1}^{N_i} Y_{ij} \tag{6}$$

- **Prima pura por póliza:**

$$PP_i = \mathbb{E}[\, L_i \mid x_i \,] \tag{7}$$

Además, la prima pura por unidad de exposición se define como: $\pi(x_i) = \frac{PP_i}{e_i}$. Cabe precisar, la Tabla de Experiencia muestra que, para cualquier grupo homogéneo de riesgo g con exposición total E_g, siniestros totales N_g y pérdida total L_g, la prima pura observada por unidad de exposición satisface la identidad:

$$\frac{L_g}{E_g} = \left(\frac{N_g}{E_g}\right)\left(\frac{L_g}{N_g}\right), \qquad \left(N_g > 0\right) \tag{8}$$

donde N_g/E_g es la frecuencia observada y L_g/N_g la severidad media observada. Esta identidad evidencia que la descomposición frecuencia–severidad surge directamente de los agregados de la experiencia (conteos y montos) bajo exposición.

Bajo el supuesto de independencia condicional entre frecuencia y severidad, es decir, N_i es independiente de $\{Y_{ij}\}_{j\geq 1}$ dado x_i, y asumiendo que las severidades son i.i.d. condicionalmente a x_i, la pérdida esperada por unidad de exposición factoriza como:

$$\pi(x_i) = \lambda_i \, \mu_i, \qquad \lambda_i = \frac{\mathbb{E}[N_i \mid X_i = x_i, e_i]}{e_i} \tag{9}$$

En otras palabras, la prima pura por unidad de exposición se estima, bajo independencia, como el producto entre frecuencia esperada y severidad esperada (Klugman et al., 2012).

Si existiera dependencia entre frecuencia y severidad, la factorización puede no ser válida. En ese caso, la prima pura por exposición debe expresarse en forma general como:

$$\frac{\pi_i}{e_i} = \frac{1}{e_i} \mathbb{E}\left[\sum_{j=1}^{Ni} Y_{ij} | x_i\right] \tag{10}$$

y la dependencia puede modelarse explícitamente (por ejemplo, mediante cópulas u otras estructuras de dependencia), lo que permite capturar correlación subyacente y refinar la estimación (Cherubini et al., 2004; Jiménez, 2021).

Nota práctica (offset de exposición): en modelos de frecuencia (conteos) es habitual incorporar la exposición como offset, por ejemplo:

$$\log\left(\mu_i^{(N)}\right) = \log(e_i) + \eta(x_i), \mu_i^{(N)} = \mathbb{E}[\, N_i \mid x_i \,] \tag{11}$$

lo que garantiza que la tasa $\lambda_i = \mu_i^{(N)}/e_i$ se modele de forma coherente con la unidad de exposición.

2.2.2. ÁRBOLES DE DECISIÓN

El árbol de decisión es un modelo predictivo jerárquico ampliamente utilizado en clasificación y regresión. En esencia, segmenta recursivamente el espacio de predictores en regiones cada vez más homogéneas respecto de la variable objetivo, mediante reglas simples basadas en los atributos de entrada. El resultado es una estructura ramificada (un "árbol invertido") compuesta por nodos conectados por ramas: cada nodo interno aplica una condición sobre un atributo y cada rama representa un resultado posible de esa condición que conduce a otro nodo; los nodos terminales o hojas no se subdividen y contienen la predicción final (clase en clasificación o valor estimado en regresión). El nodo superior, sin padres, es la raíz y corresponde a la primera partición del conjunto de datos (Hssina et al., 2014; Breiman et al., 1984).

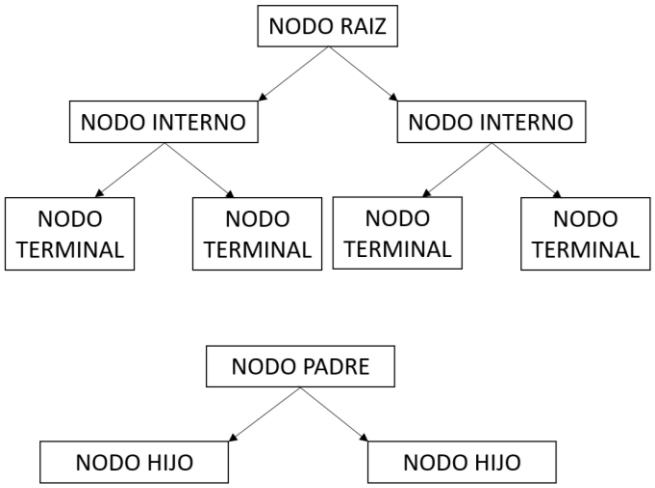


Figura 1: Ejemplos de árboles de decisión / de regresión.

Fuente: Adaptado de Zhao (2018).

En tarificación de No Vida, el punto de partida empírico es la Tabla de Experiencia construida por ventana de observación (2020) del grupo de riesgo. El árbol puede interpretarse como un mecanismo formal para “construir etapas/celdas” de dicha tabla: el nodo raíz representa la cartera completa; cada división sucesiva equivale a aplicar un filtro adicional (regla) que refina el grupo de riesgo; y cada hoja define un segmento final (celda de experiencia) sobre el cual se agregan exposición, siniestros y pérdidas. Esta correspondencia hace que la segmentación sea verificable directamente sobre la base de datos (por ejemplo, filtrando filas que cumplan las reglas del camino raíz–hoja) y permite calcular razones técnicas por segmento, como frecuencia y severidad observadas (Ohlsson & Johansson, 2010; Klugman et al., 2012).

La construcción de un árbol de decisión se realiza típicamente con algoritmos voraces (*greedy*) que particionan recursivamente el conjunto de datos. En el nodo raíz se selecciona el atributo (y umbral, si procede) que produce la mejor partición según un criterio de ganancia, generando subconjuntos más homogéneos; el proceso se repite en cada nodo hijo hasta cumplir un criterio de parada. En clasificación, la selección busca maximizar la pureza de los nodos hijos (p. ej., reduciendo impureza Gini o entropía); en regresión, se busca minimizar el error en los nodos resultantes (p. ej., reducción de MSE o MAE), repitiendo el procedimiento de forma recursiva hasta alcanzar límites predefinidos de complejidad (Zhao, 2018; Molnar, 2025; Kushwah et al., 2022).

Desde una perspectiva histórica, los métodos de árboles se desarrollaron en dos vertientes principales a partir de los años sesenta. En estadística, Morgan y Sonquist (1963) introdujeron AID (Automatic Interaction Detection), considerado el primer método tipo árbol

para regresión. Dieron paso a variantes de clasificación/segmentación como THAID (Morgan & Messenger, 1973) y CHAID (Kass, 1980), que incorporaron pruebas estadísticas para divisiones multirrama. Esta línea culminó con CART (Classification and Regression Trees) de Breiman et al. (1984), que formalizó el particionado binario recursivo, criterios de impureza y técnicas de poda para optimizar el tamaño del árbol. En aprendizaje automático, Hunt (1962) propuso un marco conceptual de inducción que influyó en Quinlan, quien desarrolló ID3 y posteriormente C4.5 (1993), con manejo de atributos continuos y datos faltantes; su versión comercial se difundió como C5.0 (Hssina et al., 2014). En conjunto, los árboles modernos integran aportes de ambas tradiciones (CART e ID3/C4.5).

Una virtud destacada de los árboles es su interpretabilidad. A diferencia de modelos de "caja negra", un árbol ofrece reglas explícitas: todo recorrido de la raíz a una hoja se interpreta como una secuencia de condiciones lógicas que justifican la predicción para un segmento. Esta transparencia es especialmente valiosa en seguros, donde se requiere explicabilidad de tarifas y factores de riesgo. Además, los árboles capturan naturalmente no linealidades y efectos de interacción. La literatura reciente muestra que métodos basados en árboles —y, en particular, sus ensambles— suelen ofrecer alto desempeño predictivo cuando existen patrones complejos, sin renunciar por completo a la interpretabilidad (Henckaerts et al., 2021).

Ahora bien, los árboles no están exentos de desafíos. Dos problemas comunes —especialmente en seguros— son: (1) el desequilibrio distributivo y (2) el sobreajuste. Un problema común en tarificación es el desequilibrio distributivo: alta proporción de pólizas con cero siniestros ($N = 0$) y, cuando hay siniestros, colas pesadas en los montos positivos. Este desbalance puede sesgar el ajuste de modelos de conteo (frecuencia) y afectar la estabilidad del modelo de severidad entrenado sobre observaciones positivas. Para mitigarlo, se emplean estrategias de remuestreo (sobremuestreo/submuestreo) orientadas a mejorar la representación de eventos raros (p. ej., pólizas con $N > 0$ o tasas altas), manteniendo métricas coherentes con regresión (devianza, MAE, WAPE calibrado), más que medidas de acierto binario. El sobreajuste, por su parte, aparece cuando el árbol crece en exceso y captura variabilidad idiosincrática del conjunto de entrenamiento, mostrando alta precisión aparente, pero rendimiento deficiente fuera de muestra. Para mitigarlo, se recurre a la poda (*pruning*) y a criterios de parada (p. ej., profundidad máxima, mínimo de muestras por hoja, ganancia mínima) (Kohavi & Quinlan, 2002).

En esta tesis, los árboles de decisión se emplearán como alternativa no paramétrica y flexible para estimar la prima pura de seguros vehiculares mediante un esquema de dos partes:

- un árbol de frecuencia para estimar la tasa esperada λ(x) (siniestros por unidad de exposición), y
- un árbol de severidad para estimar la severidad esperada μ(x) (costo esperado por siniestro).

Asumiendo independencia condicional entre frecuencia y severidad dados los factores (simplificación habitual), la prima pura por unidad de exposición se obtiene como:

$$\frac{\pi}{e} = \lambda(x)\mu(x) \tag{12}$$

En términos por póliza, $\pi = \mathbb{E}[N \mid \mathrm{x}] \cdot \mu(\mathrm{x})$. La motivación es aprovechar que los árboles segmentan el portafolio en grupos de riesgo homogéneo, reflejan umbrales y cambios abruptos en la siniestralidad, y capturan interacciones complejas manteniendo una representación interpretable. En suma, se espera mejorar la precisión de las primas puras y, a la vez, explicar qué factores y umbrales caracterizan los distintos perfiles de riesgo. La investigación evaluará su desempeño y las técnicas necesarias para abordar sus desafíos (desbalance y sobreajuste), con un diseño cuidadoso y validaciones rigurosas acordes al contexto actuarial.

FUNDAMENTOS MATEMÁTICOS DE ÁRBOLES DE REGRESIÓN

Un árbol de regresión aproxima la relación entre el vector de predictores x y la respuesta y mediante particiones recursivas del espacio de atributos. La función estimada es constante por tramos y puede expresarse como una suma de indicadores sobre regiones disjuntas:

$$\hat{f}(x) = \sum_{m=1}^{M} c_m \, 1(x \in R_m), \qquad R_m \cap R_{m'} = \emptyset \; (m \neq m'), \; \bigcup_{m=1}^{M} R_m = \mathcal{X} \tag{13}$$

En esta formulación, $\mathcal{X}$ denota el espacio de predictores, $\{R_m\}_{m=1}^{M}$ es una partición disjunta de $\mathcal{X}$ y 1(.) es la función indicadora. Cada hoja R_m asigna un valor constante c_m, de modo que todas las observaciones que caen en R_m reciben la misma predicción c_m. A mayor número de hojas M, más fina es la aproximación (mayor capacidad de ajuste), pero también crece el riesgo de sobreajuste; por ello, resulta imprescindible controlar la complejidad del árbol (Breiman et al., 1984; Hastie et al., 2009).

En el caso actuarial, se tiene una base de experiencia con unidad de observación póliza–periodo (i, t) (en esta tesis, $t = 2020$), covariables x_{it}, exposición $e_{it} > 0$, conteo N_{it} y

pérdida agregada L_{it}. Una hoja $R \subset X$ define el subconjunto de observaciones que cumplen las reglas del camino raíz–hoja:

$$I(R) = (i,t): x_{it} \in R \tag{14}$$

Sobre esa hoja se forman agregados segmentales:

$$E(R) = \sum_{(i,t)\in I(R)} e_{it}\,, N(R) = \sum_{(i,t)\in I(R)} N_{it}\,, L(R) = \sum_{(i,t)\in I(R)} L_{it} \tag{15}$$

Así, cada hoja corresponde a una celda final de la Tabla de Experiencia generada por etapas sucesivas de segmentación. En particular, cuando $N(R) > 0$, se obtiene la identidad:

$$\frac{L(R)}{E(R)} = \left(\frac{N(R)}{E(R)}\right)\left(\frac{L(R)}{N(R)}\right) \tag{16}$$

donde $N(R)/E(R)$ es la frecuencia observada del segmento y $L(R)/N(R)$ su severidad media. Por tanto, la expresión "constante por tramos" significa que el árbol asigna una misma tasa/media dentro de cada segmento final R_m, lo que facilita interpretación y trazabilidad de la segmentación tarifaria (Klugman et al., 2012; Ohlsson & Johansson, 2010). En esta tesis, m indexa hojas del árbol (regiones R_m, que pueden cambiar entre configuraciones pre/completo/post y over/under/none). En cambio, g indexa segmentos de evaluación definidos por una grilla fija para agregar pérdidas y primas y así comparar el WAPE entre modelos.

El crecimiento del árbol se realiza de forma voraz (greedy), eligiendo en cada nodo la división que maximiza una ganancia local (por ejemplo, incremento de pureza o reducción de una función de pérdida). En regresión, esta pérdida suele ser la suma de errores cuadrados o la devianza de una familia de verosimilitud (Breiman et al., 1984; Quinlan, 1986, 1993). Si R_j es un nodo con datos $\{(x_i, y_i): x_i \in R_j\}$, se evalúan divisiones univariadas de la forma:

$$R_j^L(v,t) = \{x \in R_j: x_v \le t\}, \qquad R_j^R(v,t) = \{x \in R_j: x_v > t\} \tag{17}$$

donde $v \in \{1, \dots, p\}$ es el índice de la variable candidata y t el umbral de corte. Para atributos continuos, los umbrales candidatos suelen tomarse como puntos medios entre valores ordenados de x_v; para atributos categóricos con muchos niveles, es habitual ordenarlos por la media del objetivo y probar cortes adyacentes. Este procedimiento reduce drásticamente el número de particiones a evaluar y, bajo pérdida cuadrática, es equivalente a buscar el mejor split binario (Breiman et al., 1984; Hastie et al., 2009). En la práctica, esta

heurística equilibra calidad y costo computacional, y evita el crecimiento combinatorio que supondría explorar particiones arbitrarias de niveles.

Dada una función de pérdida genérica $L(y, \hat{y})$, el valor óptimo asignado por una hoja R y la impureza del nodo se definen como:

$$\hat{c}_R = \arg\min_{c \in \mathbb{R}} \sum_{i: x_i \in R} L(y_i, c), \qquad L(R) = \sum_{i: x_i \in R} L(y_i, \hat{c}_R) \tag{18}$$

Bajo estas definiciones, el mejor split (v^*, t^*) en R_j es el que maximiza la reducción de impureza:

$$\Delta L(v, t; R_j) = L(R_j) - L(R_j^L(v, t)) - L(R_j^R(v, t)) \tag{19}$$

esto es, el que minimiza la pérdida agregada en los nodos hijos. El proceso se repite recursivamente hasta satisfacer un criterio de parada; típicamente, se detiene si (i) se alcanza una profundidad máxima, (ii) el tamaño mínimo de los nodos hijos sería insuficiente o (iii) la ganancia mínima de la división es demasiado pequeña para considerarse fiable (Breiman et al., 1984; Hastie et al., 2009). Estas restricciones constituyen la Pre-poda (poda anticipada) y actúan como regularización estructural del modelo.

En la implementación clásica de CART para regresión se utiliza la pérdida cuadrática $L(y, \hat{y}) = (y - \hat{y})^2$. En ese caso, el estimador constante por hoja es la media muestral en la región y la impureza es la suma de cuadrados residuales:

$$\hat{c}_R = \arg\min_{c} \sum_{i: x_i \in R} (y_i - c)^2 = \bar{y}_R \tag{20}$$

$$L(R) = \sum_{i: x_i \in R} (y_i - \bar{y}_R)^2 \tag{21}$$

El algoritmo recorre variables y umbrales candidatos (p. ej., puntos medios entre valores ordenados), calcula ΔL y elige el split con mayor reducción (Hastie et al., 2009).

Ahora bien, los árboles tienden a sobreajustar si se les permite crecer sin restricciones, ya que pueden capturar variabilidad idiosincrática del conjunto de entrenamiento. Para controlar la complejidad existen dos familias de técnicas complementarias. La Pre-poda detiene divisiones débiles durante el crecimiento mediante umbrales (profundidad, tamaño, ganancia mínima). La Post-poda aplica una simplificación a posteriori: una vez construido el árbol (con o sin restricciones laxas), se sustituyen subárboles por hojas si ello no empeora el desempeño en una validación separada. Un método ampliamente utilizado es

la poda por reducción de error (Reduced Error Pruning, REP), que compara, para cada nodo interno, el subárbol actual frente a una hoja única con constante recalculada y elige la opción que minimiza la pérdida en validación, procediendo de abajo hacia arriba hasta que no existen podas beneficiosas (Breiman et al., 1984; Elomaa & Kääriäinen, 2001; Kohavi & Quinlan, 2002). En términos de gobernanza del modelo, la poda aporta parsimonia, estabilidad y trazabilidad de las reglas, a costa —si se exagera— de perder sensibilidad en regiones escasas (por ejemplo, colas).

2.2.3. FUNCIONES DE PÉRDIDA EN ÁRBOLES DE REGRESIÓN

En los árboles de regresión clásicos, inaugurados por CART (Classification and Regression Trees), la calidad de una partición se evalúa mediante la suma de errores dentro del nodo. En particular, para respuestas continuas se utiliza el Error Cuadrático Medio (MSE). Este criterio garantiza que, en cada hoja, la predicción constante que minimiza el MSE sea la media muestral de la respuesta en esa región; de forma análoga, si se emplea el Error Absoluto Medio (MAE), el estimador constante óptimo es la mediana (Breiman et al., 1984; Hastie et al., 2009). Estas elecciones no son arbitrarias: ambas se derivan de principios de máxima verosimilitud bajo modelos de error distintos (gaussiano para MSE, Laplace para MAE).

Cuando la variable objetivo no es bien representada por un error cuadrático gaussiano —por ejemplo, conteos de siniestros o costos positivos y asimétricos— resulta más adecuado fundamentar el criterio de partición en la verosimilitud del modelo, reemplazando el MSE por la devianza. Sea ℓ_{sat} la log-verosimilitud del modelo saturado (el que ajusta exactamente los datos) y ℓ_{mod} la del modelo considerado; la devianza se define como:

$$D = 2(\ell_{\text{sat}} - \ell_{\text{mod}}) \tag{22}$$

La devianza es, así, una medida de discrepancia respecto del ajuste perfecto: minimizar D equivale a maximizar la verosimilitud dentro de la familia elegida (McCullagh & Nelder, 1989; Dunn & Smyth, 2018). Además, su carácter aditivo por observación facilita su uso en árboles, porque permite evaluar localmente el beneficio de un corte como suma de contribuciones individuales.

Sobre esta base se definen los árboles de regresión generalizados, que trasladan el principio de los GLM (modelos lineales generalizados) a la estructura de particionado

recursivo. En cada nodo, la división óptima es aquella que maximiza la reducción de devianza del padre frente a la suma de devianzas de los nodos hijos, esto es:

$$\Delta D = D_{padre} - (D_{izq} + D_{der}) \tag{23}$$

Esta ΔD es análoga a un test de razón de verosimilitudes local para la partición propuesta y permite al árbol adaptarse a la distribución de la respuesta (Ciampi et al., 1991; Loh, 2014). En coherencia con la devianza, la constante de hoja es, ahora, el estimador de máxima verosimilitud (MLE) del parámetro de interés (típicamente, la media μ) bajo la familia especificada, lo que asegura consistencia entre el criterio de corte y el valor de predicción en cada región.

En tarificación de No Vida, tres familias son especialmente pertinentes: Poisson para frecuencias (conteos por unidad de exposición u *offset*), Binomial Negativa cuando existe sobredispersión en los conteos ($\text{Var}(Y) = \mu_i^{(N)} + k{\mu_i^{(N)}}^2 con\ k > 0$), y Gamma para severidades (montos positivos). Con *offset* de exposición e_i, la media se formula como $\mu_i^{(N)} = e_i\, \lambda\,(x_i)$ con $\lambda(x_i) = \exp\{\eta(x_i)\}$, equivalente a $\log\, \mu_i^{(N)} = \log\, e_i + \eta(x_i)$; al evaluar devianzas se usa siempre $\mu_i^{(N)}$ que incorpora el *offset* (Dunn y Smyth, 2018; Hilbe, 2014).

Funciones de Pérdida Clásicas: MSE y MAE

A continuación, se presenta cómo las pérdidas clásicas de regresión (MSE y MAE) se obtienen como casos particulares de la devianza a partir de la definición general $D = 2(\ell_{\text{sat}} - \ell_{\text{mod}}) = 2[log\, L_{sat} - log\, L_{mod}]$ donde L_{sat} es la verosimilitud del *modelo saturado* (uno que ajusta exactamente cada observación) y L_{mod} la del modelo propuesto (Dobson & Barnett, 2018).

1) Caso Normal (gaussiano) ⇒MSE

Suponga $y_i \sim \mathcal{N}(\mu_i^{(N)}, \sigma^2)$ con varianza común σ^2. La log-verosimilitud del modelo es

$$\ell_{mod}(\mu_i^{(N)}, \sigma^2) = -\frac{1}{2}\sum_{i=1}^{n}[\frac{(y_i - \mu_i^{(N)})^2}{\sigma^2} + log(2\pi\sigma^2)] \tag{24}$$

En el modelo saturado se toma $\mu_i^{(N)} = y_i$ para todo i, con lo cual $\ell_{\text{sat}} = -\frac{1}{2}\sum_i \log\,(2\pi\sigma^2)$. Sustituyendo en la definición de devianza:

$$D = 2(\ell_{\text{sat}} - \ell_{\text{mod}}) = \frac{1}{\sigma^2}\sum_{i=1}^{n}\left(y_i - \mu_i^{(N)}\right)^2 \tag{25}$$

Para σ^2 fijo, minimizar la devianza equivale a minimizar la suma de errores cuadrados (SSE/RSS), y al dividir por n se obtiene el MSE. En otras palabras, el criterio MSE no es una construcción ad hoc, sino que corresponde al negativo de la log-verosimilitud gaussiana (salvo constantes y un factor de escala). Por ello, el estimador que minimiza MSE coincide con el MLE bajo supuestos normales (Dobson y Barnett, 2018).

2) Caso Laplace (doble exponencial) ⇒MAE

Para el modelo de errores Laplace (doble exponencial), con densidad

$$f(y_i \mid \mu_i^{(N)}, b) = \frac{1}{2b} exp(-\frac{\mid y_i - \mu_i^{(N)} \mid}{b}) \tag{26}$$

la log-verosimilitud del modelo es

$$\ell_{mod}\left(\mu_i^{(N)}, b\right) = -n\ log(2b) - \frac{1}{b}\sum_{i=1}^{n} \mid y_i - \mu_i^{(N)} \mid \tag{27}$$

En el modelo saturado, $\mu_i^{(N)} = y_i \Rightarrow \ell_{\text{sat}} = -n\log(2b)$. Por tanto,

$$D = 2(\ell_{\text{sat}} - \ell_{\text{mod}}) = \frac{2}{b}\sum_{i=1}^{n} \mid y_i - \mu_i^{(N)} \mid \tag{28}$$

para b fijo, minimizar la devianza equivale a minimizar la suma de errores absolutos (L1), y al escalar por n se obtiene el MAE. En particular, el minimizador de la suma de desviaciones absolutas en un nodo (con predicción constante) es la mediana muestral, y a nivel de regresión el caso central de la regresión cuantilíca $(\tau = 0{,}5)$ coincide con el estimador LAD (least absolute deviations). Esta conexión explica por qué el MAE es más robusto a outliers que el MSE: la pérdida crece linealmente con el error en lugar de cuadráticamente, de modo que observaciones extremas ejercen menor influencia en la estimación de la constante de hoja.

Modelos con Devianza: Poisson, Binomial Negativa, Gamma (frecuencia y severidad)

Definición general. La devianza se define como

$$D = 2(\ell_{\text{sat}} - \ell_{\text{mod}}) = -2\ln\Lambda \tag{29}$$

donde ℓ_{sat} es la log-verosimilitud del modelo saturado, ℓ_{mod} la del modelo considerado, y Λ la razón de verosimilitudes del modelo frente al saturado. Minimizar D equivale a maximizar la verosimilitud dentro de la familia especificada (Statsmodels Developers, 2025; Dobson & Barnett, 2018; Dunn & Smyth, 2018).

Cuando hay exposición $e_i > 0$ (frecuencia), trabajamos con medias que incorporan el offset:

$$\log \hat{\mu}_i^{(N)} = \log e_i + \eta(x_i), \qquad equiv. \qquad \hat{\mu}_i = e_i \, exp\{\eta(x_i)\} \tag{30}$$

Esta formulación asegura invariancia de unidades: reexpresar la exposición (por ejemplo, de "años" a "meses") se compensa agregando $\log e_i$ al predictor lineal, de modo que las contribuciones de devianza se calculan siempre con la media ajustada μ_i que incluye el offset.

Devianza en el Modelo Poisson

Log-verosimilitud: Si $\mathrm{y}_i \sim \mathrm{Poisson}\left(\mu_i^{(N)}\right)$, la log-verosimilitud individual es

$$\ell_i^{mod}\,(\,\mu_i \mid y_i\,) = y_i \log \mu_i \, - \mu_i^{(N)} - \log(y_i!) \tag{31}$$

En el modelo saturado, se toma $\mu_i^{(N)^{sat}} = y_i$, que maximiza la verosimilitud por observación:

$$\ell_i^{sat} \, = y_i \log y_i \, - y_i - \log(y_i!) \tag{32}$$

Devianza: Sustituyendo en $D \, = 2(\ell_{\text{sat}} - \ell_{\text{mod}})$ y simplificando términos constantes se obtiene, para n observaciones,

$$D_{\text{Poisson}} \;=\; 2\sum_{i=1}^{n}\left[\, y_i \ln\left(\frac{y_i}{\mu_i^{(N)}}\right) - \left(y_i - \mu_i^{(N)}\right)\right] \tag{33}$$

Por convención, $y_i \ln\left(\frac{y_i}{\mu_i^{(N)}}\right) = 0$ cuando $y_i = 0$ (límite continuo); si $\mu_i^{(N)} \to 0^+$ con $y_i > 0$, la contribución de devianza diverge, lo que refleja un grave desajuste. Valores pequeños de $D_{Poisson}$ indican buen ajuste; valores grandes señalan discrepancias sistemáticas (Dobson y Barnett, 2018; Dunn y Smyth, 2018).

Estimador óptimo $\hat{c}_R$ en un nodo:

En una hoja R de un árbol de frecuencias con modelo Poisson y exposición e_i (esto es, $\hat{\mu}_i^{(N)} = e_i \, c$ con enlace log y offset $\ln e_i$), el valor constante óptimo es la tasa promedio ponderada por exposición:

$$\hat{c}_R = \frac{\sum_{i \in R} y_i}{\sum_{i \in R} e_i} \tag{34}$$

Este resultado surge de minimizar la devianza Poisson en la hoja y es coherente con el MLE de una tasa con denominador e_i. Si todas las exposiciones son iguales ($e_i \equiv 1$), $\hat{c}_R$ coincide con la media muestral de y en el nodo (Henckaerts et al., 2021; Dunn & Smyth, 2018; Dobson & Barnett, 2018).

Devianza en el Binomial Negativa

Log-verosimilitud: Consideremos la parametrización Binomial Negativa, donde $Y_i \sim Binomial\,Negativa(\mu_i^{(N)}, k)$ con media $\mu_i^{(N)}$ y parámetro de dispersión $k > 0$ fija $\left(Var(Y_i) = \mu_i^{(N)} + k{\mu_i^{(N)}}^2\right)$. Su función de masa es:

$$P(Y_i = y_i) = \frac{\Gamma\left(y_i + \frac{1}{k}\right)}{y_i!\ \Gamma\left(\frac{1}{k}\right)} \left(\frac{1}{1 + k\mu_i^{(N)}}\right)^{\frac{1}{k}} \left(\frac{k\mu_i^{(N)}}{1 + k\mu_i^{(N)}}\right)^{y_i} \tag{35}$$

De aquí se obtiene la log-verosimilitud individual:

$$\ell_i^{mod}\left(\mu_i^{(N)}, k \mid y_i\right) = log\,\Gamma\left(y_i + \frac{1}{k}\right) - log\,\Gamma\left(\frac{1}{k}\right) - log(y_i!) + y_i \log\left(k\mu_i^{(N)}\right) - \left(y_i + \frac{1}{k}\right) log\left(1 + k\mu_i^{(N)}\right) \tag{36}$$

En el modelo saturado, ${\mu_i^{(N)}}^{sat} = y_i$, por lo que

$$\ell_i^{sat} = log\,\Gamma\left(y_i + \frac{1}{k}\right) - log\,\Gamma\left(\frac{1}{k}\right) - log(y_i!) + y_i \log(ky_i) - \left(y_i + \frac{1}{k}\right) log(1 + ky_i) \tag{37}$$

Devianza: Al calcular $D = 2(\ell_{\text{sat}} - \ell_{\text{mod}})$, los términos constantes se cancelan. La devianza total para n observaciones es

$$D_{\text{NegBin}} = 2\sum_{i=1}^{n}\left[y_i \log\left(\frac{y_i}{\hat{\mu}_i^{(N)}}\right) - \left(y_i + \frac{1}{k}\right)\log\left(\frac{y_i + \frac{1}{k}}{\hat{\mu}_i^{(N)} + \frac{1}{k}}\right)\right] \tag{38}$$

(con la convención $y_i \log(y_i/\hat{\mu}_i^{(N)}) = 0$ si $y_i = 0$).

Esta expresión generaliza la devianza Poisson incorporando el término de sobredispersión k (que aparece como $1/k$ en forma). Nótese que si $k \to 0$ (sin sobredispersión), la expresión tiende a la devianza Poisson. Como en Poisson, D_{NegBin} se interpreta como un estadístico de razón de verosimilitudes (Dobson & Barnett, 2018).

Estimador óptimo $\hat{c}_R$ en un nodo: Para Binomial Negativa con media $\mu_i^{(N)} = e_i\, c$ y dispersión $k > 0$ ($\text{var}(Y_i) = \mu_i^{(N)} + k{\mu_i^{(N)}}^2$), el óptimo por hoja $\hat{c}_R$ se obtiene resolviendo la ecuación de score (respecto de c):

$$\sum_{i \in R} \frac{e_i \left(y_i - \mu_i^{(N)}\right)}{\mu_i^{(N)} \left(1 + k\, \mu_i^{(N)}\right)} = 0\,, \qquad \mu_i^{(N)} = e_i\, c \qquad \ldots (39)$$

La solución no tiene forma cerrada cuando e_i varía; se resuelve por Newton–Raphson (o métodos cuasi-newtonianos), siendo habitual inicializar en la tasa Poisson ($\sum y / \sum e$) para acelerar la convergencia (Dunn & Smyth, 2018; Henckaerts et al., 2021; Statsmodels Developers, 2025).

Devianza en el Gamma

Log-verosimilitud: Tomemos $Y_i \sim Gamma(\alpha, \theta_i)$ parametrizada por su media $\mu_i^{(S)} = \alpha\, \theta_i$ (Es común reparametrizar por $\mu_i^{(S)}$ y ϕ donde $\phi = 1/\alpha$ es la dispersión) (Dunn & Smyth, 2018). Una forma conveniente de la densidad es

$$f\left(y_i \mid \mu_i^{(S)}, \phi\right) = \frac{1}{\Gamma\left(\frac{1}{\phi}\right)\left[\mu_i^{(S)}\phi\right]^{\frac{1}{\phi}}} y_i^{\left(\frac{1}{\phi}\right)-1} \exp\left(-\frac{y_i}{\mu_i^{(S)}\phi}\right) \qquad (40)$$

La log-verosimilitud del modelo es

$$\ell_i^{mod}\left(\mu_i^{(S)}, \phi \mid y_i\right) = \left(\frac{1}{\phi} - 1\right)\log y_i - \log\Gamma\left(\frac{1}{\phi}\right) - \frac{1}{\phi}\log\left(\mu_i^{(S)}\phi\right) - \frac{y_i}{\mu_i^{(S)}\phi} \qquad (41)$$

En el modelo saturado es ${\mu_i^{(S)}}^{sat} = y_i$, mientras que en el modelo propuesto $\mu_i^{(S)}$ es la media ajustada (Dunn & Smyth, 2018).

$$\ell_i^{sat} = \left(\frac{1}{\phi} - 1\right)\log y_i - \log\Gamma\left(\frac{1}{\phi}\right) - \frac{1}{\phi}\log(y_i\phi) - \frac{y_i}{y_i\phi} \qquad (42)$$

Devianza: Al aplicar la definición general y simplificar se obtiene la devianza Gamma total

$$D_{\text{Gamma}} = \sum_{i=1}^{n} d(y_i, \mu_i^{(S)})\,, \qquad d(\mathrm{y}, \mu^{(S)}) = 2\left[\frac{\mathrm{y} - \mu^{(S)}}{\mu^{(S)}} - \log\left(\frac{\mathrm{y}}{\mu^{(S)}}\right)\right] \qquad (43)$$

La devianza unitaria $d(\mathrm{y}, \mu^{(S)})$ es convexa en $\mu^{(S)} > 0$, no negativa y se anula si y solo si $\mathrm{y} = \mu^{(S)}$. Esta convexidad garantiza unicidad del MLE de la media por hoja (Dunn & Smyth, 2018).

Estimador óptimo $\hat{c}_R$ en un nodo:

Con devianza unitaria $d_\Gamma(y,\mu^{(S)}) = 2[\frac{y}{\mu^{(S)}} - 1 - \ln(\frac{y}{\mu^{(S)}})]$, el estimador de hoja depende de cómo esté estructurado el dato:

Nivel reclamo (cada y_i es un monto positivo): La constante óptima es la media muestral del nodo: $\hat{c}_R = \frac{1}{n_R}\sum_{i\in R} y_i$.

Nivel póliza (pérdida media por siniestro): si se modela $y_i = L_i/N_i$ y se pondera con $w_i = N_i$, la constante óptima es $\hat{c}_R = \frac{\sum_{i\in R} L_i}{\sum_{i\in R} N_i}$, esto es, la pérdida media por siniestro del nodo.

Ambos casos se obtienen de minimizar la devianza Gamma y son estándar en árboles de severidad (Henckaerts et al., 2021; Dunn y Smyth, 2018). En la práctica, cuando se trabaja a nivel póliza con muchos ceros (sin reclamos), la componente de severidad se entrena sobre $y_i > 0$ y las ponderaciones o agregaciones se manejan de forma consistente en la integración con frecuencia.

Sobredispersión

Para decidir si la devianza Poisson es adecuada o si debe utilizarse la devianza Binomial Negativa en un árbol de regresión de frecuencias, conviene evaluar la sobredispersión mediante el estadístico de dispersión de Pearson usando las medias ajustadas $\hat{\mu}_i^{(N)}$ con offset. Limitarse a comparar media y varianza muestral global puede ser engañoso cuando existe heterogeneidad o exposición desigual.

Sea $Y = (y_1, \dots, y_n)$ un conjunto de conteos y $\hat{\mu}_i^{(N)}$las medias ajustadas. La media muestral es $\bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$ y la varianza muestral con divisor n es $s_0^2 = \frac{1}{n}\sum_{i=1}^n (y_i - \bar{y})^2$. En un modelo Poisson homogéneo (sin covariables ni offset) se espera equidispersión: $\mathbb{E}[Y] = \mathrm{Var}(Y)$ (Roback & Legler, 2021). En presencia de covariables/offset, es preferible usar el estadístico de Pearson:

$$\hat{\phi}_P = \frac{\chi_P^2}{n-p}, \qquad \chi_P^2 = \sum_{i=1}^n \frac{(y_i - \hat{\mu}_i^{(N)})^2}{\hat{\mu}_i^{(N)}} \tag{44}$$

donde p es el número de parámetros efectivos estimados (en una hoja, $p = 1$; a escala de árbol, p crece con el número de hojas). Importante: en un nodo Poisson con exposición e_i, debe usarse $\hat{\mu}_i^{(N)} = e_i\,\hat{c}_R$(con $\hat{c}_R = \frac{\sum y_i}{\sum e_i}$). Con datos realmente Poisson, $\hat{\phi}_P$ oscila alrededor de 1 incluso con decenas de miles de registros (Hilbe, 2014).

Hilbe (2014) propone umbrales operativos:

Si la base es pequeña o moderada, el modelo Poisson se mantiene mientras $\hat{\phi}_P \leq 1.25$;

valores mayores de indican sobredispersión clara (Hilbe, 2014).

Si la base es muy grande, basta una inflación del 5 % ($\hat{\phi}_P > 1.05$;) para descartar la devianza Poisson, porque los errores estándar ya quedarían subestimados (Hilbe, 2014).

En el marco de esta tesis, cuando $\hat{\phi}_P$ supera razonablemente estos umbrales, se prefiere la Binomial Negativa y se evalúan todos los modelos con una dispersión de referencia k estimada en entrenamiento, manteniéndola fija al comparar devianzas en prueba y en validación cruzada. Este procedimiento garantiza comparabilidad y evita que las diferencias en el ajuste se confundan con diferencias de parametrización.

2.2.4. MANEJO DEL DESEQUILIBRIO DE DATOS

En problemas actuariales es habitual que la probabilidad se concentre en pocas regiones del espacio de covariables. En frecuencia, esto se observa como tasas de siniestro muy bajas (gran proporción de ceros); en severidad, como montos positivos fuertemente asimétricos (colas superiores). Si no se corrige, este desequilibrio sesga el entrenamiento y debilita la capacidad del modelo para detectar patrones en regiones raras (Frees, 2018; Kulkarni et al., 2021).

En esta tesis se emplean dos técnicas de remuestreo físico por cuantiles del objetivo:

- Sobremuestreo (*oversampling*): duplicar con reemplazo observaciones de estratos (“bins”) escasos.
- Submuestreo (*undersampling*): recortar sin reemplazo observaciones en estratos (“bins”) densos.

El remuestreo se aplica únicamente dentro del proceso de entrenamiento y, cuando corresponde, en la validación interna utilizada para la post-poda. El conjunto de prueba se mantiene siempre con la distribución original de la cartera, sin remuestreo, a fin de evaluar el desempeño fuera de muestra en condiciones representativas.

Variable de estratificación por cuantiles

Sea (x_i, y_i) la observación $i = 1, \dots, n$, con $x_i \in \mathbb{R}^p$ y y_i el objetivo. Definimos una variable de estratificación T_i que determina los cuantiles sobre los que se remuestrean.

Frecuencia (Binomial Negativa con *offset*): Se estratifica por la tasa observada r_i, esto es, siniestros por unidad de exposición $e_i > 0$:

$$T_i = r_i = \frac{y_i}{e_i} \tag{45}$$

La tasa r_i es coherente con el modelo con offset $log\, e_i$: normaliza por exposición y alinea la estratificación con la media modelada $\mu_i = e_i\, \lambda\,(x_i)$ (McCullagh & Nelder, 1989; Cameron & Trivedi, 2013; Hilbe, 2011).

Severidad (Gamma, $y_i > 0$): Se estratifica por cuantiles del monto positivo sin transformar:

$$T_i = y_i, (y_i > 0) \tag{46}$$

Esta elección (que coincide con la implementación desarrollada en esta investigación) preserva el orden natural de las colas y simplifica la operación (Frees, 2009).

Con la variable T se definen m cortes por cuantiles:

$$\tau_j = F_T^{-1}(\alpha_j), \qquad \alpha_j = \frac{j}{m}, \qquad j = 0, \dots, m \tag{47}$$

donde F_T^{-1} es la función cuantil de T. A partir de los cortes se forman los estratos (bins):

$$Bj = \{i: \tau_{j-1} < T_i \leq \tau_j\}, \qquad n_j = |B_j|, \qquad j = 1, \dots, m \tag{48}$$

El tamaño medio por bin es

$$\bar{n} = \frac{1}{m}\sum_{j=1}^{m} n_j \tag{49}$$

Conexión con el ajuste: el remuestreo por cuantiles repondera implícitamente las regiones del objetivo que más interesan (eventos raros y colas). En Binomial Negativa / Gamma, donde se minimiza devianza (suma de contribuciones por observación), este reequilibrio cambia la influencia relativa de cada región durante la estimación (McCullagh & Nelder, 1989; Cameron & Trivedi, 2013).

Sobremuestreo

El sobremuestreo duplica con reemplazo observaciones escasas para reforzar regiones raras. (Chawla, Bowyer, Hall & Kegelmeyer, 2002). En la estratificación por cuantiles, se utiliza una regla multiplicativa por bin. Para cada bin Bj, se fija un tamaño objetivo:

$$n_j^* = min\left(\left\lceil \left(1 + \lambda^{(O)}\right) n_j \right\rceil, n_{max}\right), \qquad \lambda \geq 0 \tag{50}$$

donde $\lambda^{(O)}$ es la intensidad del sobremuestreo (por ejemplo, $\lambda^{(O)} = 0.05$ añade ≈5 %) y n_{max} es una cota de seguridad por bin.

El bin aumentado se construye como:

$$B_j' = B_j \cup Resample(B_j, n_j^* - n_j) \tag{51}$$

es decir, se añaden $n_j^* - n_j$ índices del propio bin seleccionados con reemplazo. El conjunto balanceado final es

$$\mathcal{D}' = \bigcup_{j=1}^{m} \{(x_i, y_i) : i \in B_j'\} \tag{52}$$

Presenta la ventaja de reforzar las regiones escasas (colas) sin generar datos sintéticos. No obstante, una duplicación excesiva —por una intensidad alta (λ) o un límite $n_{\max}$ demasiado amplio— puede inducir sobreajuste y elevar el costo computacional, como se discute en la literatura (Chawla et al., 2002; Kulkarni et al., 2021).

Submuestreo

El submuestreo es la técnica inversa: reduce el número de observaciones en estratos densos con el fin de balancear el conjunto en términos de información, disminuyendo el sesgo del entrenamiento hacia la región más frecuente (Kulkarni et al., 2021). En esta investigación, reduce el número de observaciones en estratos demasiado densos para disminuir el sesgo.

Se define un objetivo de recorte por estrato - *bin* B_j:

$$n_j^\star = \begin{cases} n_j, & j \in \{1, m\} \text{ (colas preservadas)} \\ \min(n_j, \lfloor (1-\rho)\,\bar{n} \rfloor), & \text{en otro caso,} \end{cases} \quad 0 \le \rho < 1 \tag{53}$$

donde ρ controla la intensidad del recorte (p.ej., $\rho = 0.2$ intenta llevar los bins centrales a ≈$0.8\,\bar{n}$).

El bin recortado se obtiene por muestreo sin reemplazo:

$$B_j' = \text{Sample}(B_j, n_j^\star) \tag{54}$$

Así, un submuestreo acelera el entrenamiento y reduce varianza en zonas muy densas sin destruir la información crítica de colas.

Es eficiente computacionalmente y presenta menor riesgo de sobreajuste; no obstante, puede producir pérdida de información si el recorte es demasiado agresivo en *bins* que no pertenecen a las colas (Kulkarni et al., 2021).

2.2.5. PODA PARA EVITAR EL SOBREAJUSTE

Los árboles de decisión tienden a sobreajustar si crecen sin restricciones: reducen la pérdida en entrenamiento, incluso casi a cero, a costa de capturar variabilidad idiosincrática del conjunto de entrenamiento, lo que deteriora la generalización (Scikit-learn Developers, 2025; Mausam, 2011). La poda controla la complejidad con dos enfoques complementarios: (i) la pre-poda detiene divisiones débiles durante el crecimiento y (ii) la post-poda sustituye, a posteriori, subárboles por hojas cuando ello no empeora la pérdida en validación (Song & Lu, 2015).

Pre-Poda (Poda Anticipada)

La pre-poda frena el crecimiento cuando la partición no compensa (mejora marginal) o no es estable (hijos muy pequeños). Sea un nodo padre con regiones hijas R_L y R_R. Con devianza como pérdida (NB en frecuencia; Gamma en severidad), la ganancia del split es

$$\Delta D \;=\; D(\text{padre}) - [D(R_L) + D(R_R)] \tag{55}$$

donde D es la devianza de la familia (p. ej., NB o Gamma). Se acepta la división si ΔD supera ciertos umbrales y los hijos cumplen requisitos mínimos (Therneau & Atkinson, 2023; Scikit-learn Developers, 2025).

Reglas de parada habituales

Profundidad máxima

La profundidad de un nodo se define como la distancia (número de bifurcaciones) desde la raíz hasta ese nodo.

$$Si\ profundidad(t)\ \geq\ D_{max}\ \Rightarrow\ t\ es\ hoja. \tag{56}$$

Limitar la profundidad máxima D_{max} es una forma directa de acotar la complejidad del modelo y prevenir el sobreajuste (Scikit-learn Developers, 2025).

Tamaño mínimo de nodo

Otra restricción común es exigir un mínimo de observaciones para subdividir un nodo (*min samples*).

$$Si\ \mid Dt \mid < N_{min} \Rightarrow t\ es\ hoja. \tag{57}$$

Se impone típicamente que cada hoja contenga al menos un número N_{min} de observaciones. Valores altos de N_{min} hacen que el árbol no cree hojas con pocos datos, evitando así reglas basadas en muestras escasas (que tienden a sobreajustar) (Therneau & Atkinson, 2023; Scikit-learn Developers, 2025).

Ganancia de devianza mínima (absoluta/relativa)

El algoritmo de construcción del árbol evalúa en cada nodo potenciales divisiones y escoge la que mayor mejora produzca en la métrica de impureza o pérdida.

$$\Delta D_{NB\ o\ Gamma} \geq max\{\min_gain, \min_gain_rel \cdot D_{NB\ o\ Gamma}(padre)\} \tag{58}$$

Se acepta una división sólo si ΔD supera un umbral. Dado que las diferencias de devianza se comportan asintóticamente como pruebas χ^2, también puede usarse un contraste de significancia como criterio de parada (Ritschard, 2006).

Post-Poda (Poda Posterior)

En la post-poda se construye primero el árbol completo (o al menos sin restricciones fuertes) y luego se podan las ramas irrelevantes, conservando aquellas que mejoran la validación (Song & Lu, 2015).

Poda por reducción de error (Reduced Error Pruning, REP):

Sea T un árbol entrenado sobre un conjunto de entrenamiento S. Denota $L(T)$ el conjunto de hojas. Sea $L(y, c)$ la pérdida por observación (en esta tesis, la contribución de devianza de NB o de Gamma). La constante de hoja (c_l) se obtiene como:

$$c_l = arg\min_{c \in \mathbb{R}} \sum_{(x,y) \in S_l} L(y, c) \tag{59}$$

donde c es el dominio apropiado del parámetro de hoja (tasa ≥ 0 para NB; media > 0 para Gamma). Para un conjunto de validación de poda E, la pérdida del árbol es

$$E(T; E) = \sum_{l \in L(T)} \sum_{(x,y) \in E_l} L(y, c_l) \tag{60}$$

Sea n un nodo interno con subárbol T_n. El árbol podado T^{T_n} reemplaza T_n por una hoja en n con valor

$$c_n = arg\min_{c \in C} \sum_{(x,y) \in S_n} L(y, c) \tag{61}$$

La regla de poda por reducción de error (REP) poda T_n si no aumenta la pérdida en validación:

$$\sum_{(x,y) \in E_n} L(y, c_n) \leq \sum_{l \in L(T_n)} \sum_{(x,y) \in E_l} L(y, c_l) \tag{62}$$

procediendo de abajo hacia arriba hasta que no haya podas beneficiosas. REP es simple, estable y consistente con el principio de mantener únicamente ramas que aportan mejora verificable fuera de muestra (Elomaa & Kääriäinen, 2001).

2.2.6. EVALUACIÓN DE LOS MODELOS

Devianza Binomial Negativa

Para $y_i \in \mathbb{N}_0$, $\mu_i^{(N)} > 0$ y parámetro de dispersión $k > 0$, la devianza NB usada es

$$D_{\text{NegBin}} = 2\sum_{i=1}^{n}\left[y_i \log\left(\frac{y_i}{\hat{\mu}_i^{(N)}}\right) - \left(y_i + \frac{1}{k}\right)\log\left(\frac{y_i + \frac{1}{k}}{\hat{\mu}_i^{(N)} + \frac{1}{k}}\right)\right] \tag{63}$$

con la convención $y_i \log\left(\frac{y_i}{\hat{\mu}_i^{(N)}}\right) = 0$ si $y_i = 0$ Esta forma surge de la razón de verosimilitudes entre el modelo saturado $\left(\hat{\mu}_i^{(N),sat} = y_i\right)$ y el modelo ajustado $\left(\hat{\mu}_i^{(N)}\right)$, donde $\hat{\mu}_i^{(N)}$ es la media ajustada provista por el modelo NB (en esta tesis, $\hat{\mu}_i^{(N)} = \hat{r}\, e_i$). En esta tesis, $\hat{\mu}_i^{(N)} = \hat{r} e_i$ incorpora el offset de exposición e_i y la devianza se evalúa con el mismo k de referencia estimado en entrenamiento, para garantizar comparabilidad entre configuraciones (Dunn & Smyth, 2018).

Interpretación. Valores bajos de D_{NegBin} indican mejor ajuste a la distribución de conteos sobredispersada. El uso del mismo k_{ref} (procedente de entrenamiento) evita “ventajas artificiales” al comparar modelos en prueba (Dunn & Smyth, 2018).

Devianza Gamma

Para montos positivos $y_i > 0$, la devianza Gamma (unidad de devianza sumada en toda la muestra) es

$$D_{\Gamma}\left(y,\mu^{(S)}\right) \;=\; 2\sum_{i}\left[\frac{y_i-\mu_i^{(S)}}{\mu_i^{(S)}}-\log\left(\frac{y_i}{\mu_i^{(S)}}\right)\right] \tag{64}$$

Esta expresión también proviene de la razón de verosimilitudes saturado vs. ajustado dentro de la familia Gamma, y es aditiva por observación, lo que la hace apta para decisiones locales en árboles (Dunn & Smyth, 2018).

Interpretación. Devianzas menores indican mejor capacidad del árbol para reproducir la media local de severidad en cada hoja, respetando la asimetría y heterocedasticidad típicas de montos positivos.

Pseudo D²

Se reporta el pseudo D^2 como proporción de devianza explicada respecto de un modelo nulo:

$$D^2 \;=\; 1-\frac{D_{\text{modelo}}}{D_{\text{nulo}}} \tag{65}$$

El índice puede ser negativo si el modelo es peor que el nulo (Dunn & Smyth, 2018). Para que el cociente sea interpretable, el nulo debe definirse de forma coherente con la familia y con la presencia de offset.

En frecuencia (NB), se reporta únicamente el pseudo D^2coherente con offset (con la misma dispersión k_{ref}estimada en entrenamiento, para asegurar comparabilidad).

- Pseudo D^2NB (offset): el nulo usa $\mu_i^{(\mathrm{N}),\,\text{null}} = r_0\, e_i$, con $r_0 = \frac{\sum y_{\text{train}}}{\sum e_{\text{train}}}$. Este nulo respeta la exposición y sirve de base actuarial.

En severidad Gamma: se toma como nulo la media de entrenamiento: $\mu^{(\mathrm{S}),\,\text{null}} = \bar{y}_{\text{train}}$.

Error Absoluto Medio (MAE)

El MAE es un indicador que mide el tamaño promedio de los errores de predicción, sin considerar su signo. Se define como:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i-\hat{y}_i| \tag{66}$$

Es robusto frente a colas moderadas y resulta fácilmente interpretable (Hyndman & Athanasopoulos, 2021).

Error Cuadrático Medio (RMSE).

El RMSE penaliza fuertemente los errores grandes:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{67}$$

En presencia de colas pesadas, es habitual observar RMSE ≫ MAE; este patrón confirma unos pocos errores muy grandes que inflan el cuadrático (Hyndman & Athanasopoulos, 2021; Hyndman & Koehler, 2006).

Predichos vs. Reales

El diagrama de dispersión (x: y_i, y: $\hat{y}_i$) diagnostica sesgos sistemáticos (p. ej., subestimación en colas altas si los puntos quedan debajo de la diagonal) y la precisión global (James, Witten, Hastie, & Tibshirani, 2021). En frecuencia, la gran masa en cero produce un "bloque" en el origen; en severidad, la cola alta debería mostrar mayor dispersión vertical por heterocedasticidad.

Residuos vs. Predichos

Definiendo $re_i = y_i - \hat{y}_i$ el gráfico re_i vs. $\hat{y}_i$ permite identificar patrones sistemáticos y heterocedasticidad. En frecuencia, la discreción genera bandas paralelas (una por cada entero) con pendiente aparente −1; en severidad Gamma aparece el patrón de "abanico" (varianza creciente con la media). Ambos son compatibles con las familias utilizadas (James et al., 2021).

Curvas de aprendizaje

Las curvas muestran el error (típicamente devianza media por observación) en entrenamiento y validación al crecer el tamaño muestral. Permiten distinguir alto sesgo (errores altos en ambos) de alta varianza (entrenamiento bajo, validación alta) y evaluar si más datos mejoran la generalización (James et al., 2021). En esta investigación, estas curvas se reportan por familia y combinación de poda/remuestreo.

Validación cruzada

La validación cruzada K-fold divide los datos en K pliegues, entrena con $K-1$ y valida en el restante, rotando hasta usar todos. Es un estándar para estimar error predictivo y seleccionar hiperparámetros (Arlot & Celisse, 2010; James et al., 2021). En esta tesis:

- **Frecuencia (NB):** se estratifica por conteo (0/1/2/≥3) para estabilizar la mezcla.
- **Severidad (Gamma):** se preserva la condición $y > 0$ y se mantiene la mezcla de montos por pliegue.

Todas las devianzas en CV se informan como medias por observación para hacer comparables pliegues de distinto tamaño.

Importancia de Variables

En árboles, la importancia se mide como la suma de reducciones de pérdida (aquí, reducciones de devianza ΔD) atribuibles a cada variable en todos los splits donde interviene (Breiman, 2001; Louppe, 2014). Variables con mayor $\sum \Delta D$ se consideran más influyentes en la explicación.

Prima Pura y WAPE (Weighted Absolute Percentage Error) calibrado por segmentos

Se busca calcular el WAPE que mide el error absoluto total como fracción del total real. Para ello, se debe construir la Prima Pura en base a los modelos de Binomial Negativa y Gamma.

Prima Pura (PP)

En adelante, $\pi(x)$ denota la pérdida esperada por unidad de exposición, mientras que PP_i representa la pérdida esperada por póliza–periodo, dada por $PP_i = e_i\ \pi(x_i)$. Se combinan las predicciones de frecuencia esperada $\hat{\lambda}_i^{NB}$ (tasa por unidad de exposición, modelada con *offset*) y $\hat{\mu}_i^{\Gamma}$ es la severidad esperada por siniestro, condicionada a ocurrencia ($N_i > 0$); en la estimación se trabaja con severidades positivas.

$$\mathrm{PP}_i\ = e_i\ \hat{\lambda}_i^{\mathrm{NB}} \times \hat{\mu}_i^{\Gamma} \tag{68}$$

producto de la frecuencia (NB con offset) y la severidad (Gamma).

Prima pura agregada por segmento g: pérdida observada y prima pura esperada (predicha)

Se construyen segmentos con claves derivadas de las variables de los modelos, para cada segmento g. Nótese que g (segmentos de evaluación) es una partición fija para la comparación agregada, distinta de $\{R_m\}$(hojas del árbol), que puede variar entre configuraciones.

Observado (pérdida agregada):

$$L_g = N_{\mathrm{obs},g} \times \bar{\mathrm{sev}}_{\mathrm{obs},g} \tag{69}$$

donde $N_{obs,g}$ es el número observado de siniestros (agregado por segmento) y $s\bar{e}v_{obs,g}$ la severidad media observada. Para evitar ambigüedad con la sigla EPP (Expected Pure Premium), en la presente investigación se denota el observado como L_g (equivalente a $PP_{obs,g}$).

Predicho:

$$EPP^{tot}_{\text{pred},g} = \sum_{i \in g} \text{PP}_i \tag{70}$$

Calibración global y WAPE

Calibración escalar (corrige sesgo global):

$$c_{cal} = \frac{\sum_g L_g}{\sum_g EPP^{tot}_{\text{pred},g}}, \quad EPP^{tot}_{\text{pred},calibrada,g} = c_{cal}\, EPP^{tot}_{\text{pred},g} \tag{71}$$

de modo que el total predicho coincida con el total observado. Esta calibración no altera relatividades entre segmentos, sólo corrige el nivel global.

WAPE calibrado (Error porcentual absoluto ponderado) por segmentos:

$$\text{WAPE}_{\text{calibrado}} = \frac{\sum_g | \; EPP^{tot}_{\text{pred},calibrada,g} - L_g \; |}{\sum_g L_g} \tag{72}$$

El $\text{WAPE}_{\text{calibrado}}$ mide el error absoluto total tras el re-anclaje, de modo que separa el error de escala global (corregido con c_{cal}) del error de distribución entre segmentos (lo que realmente reporta el WAPE). En el capítulo de resultados, esta métrica se utiliza como base para comparar las configuraciones a nivel agregado.

2.3. HIPÓTESIS

Hipótesis General

Ho (Hipótesis nula general): La utilización combinada de técnicas de remuestreo y de poda no mejora la devianza específica (Binomial Negativa con *offset* de exposición para frecuencia; Gamma para severidad) ni el pseudo-D^2 de los submodelos, ni reduce el error de la prima pura (medido por el WAPE calibrado), respecto de un modelo base sin remuestreo ni poda, en pólizas de automóviles del año 2020.

Hipótesis Específicas

Se desarrollan las siguientes hipótesis específicas:

Ho1 (Hipótesis nula específica 1): El uso de técnicas de remuestreo (submuestreo/sobremuestreo) no mejora la devianza específica (Binomial Negativa con *offset* de exposición para frecuencia; Gamma para severidad) ni el pseudo-D^2 de los modelos de frecuencia y severidad, ni mejora la precisión de la prima pura (medida por el WAPE calibrado), respecto de un modelo base sin remuestreo, en pólizas de automóviles del año 2020.

Ho2 (Hipótesis nula específica 2): La aplicación de poda (pre-poda y post-poda) no reduce el sobreajuste ni mejora la devianza específica (Binomial Negativa con *offset* de exposición para frecuencia; Gamma para severidad), el pseudo-D^2 ni la precisión de la prima pura (medida por el WAPE calibrado), respecto del árbol base sin poda, en pólizas de automóviles del año 2020.

CAPÍTULO III. DISEÑO METODOLÓGICO

3.1. TIPO DE INVESTIGACIÓN

La investigación es de enfoque cuantitativo, aplicada, de alcance comparativo–predictivo y diseño no experimental, basada en datos observacionales de una cartera de seguros de automóviles correspondiente al año 2020. Se evalúa cómo cambian las métricas de desempeño de distintas estrategias de remuestreo (sin remuestreo, sobremuestreo y submuestreo) combinadas con Pre-poda y Post-poda por reducción de error (Reduced Error Pruning, REP) en dos modelos desarrollados: un árbol de regresión para conteos, optimizado mediante la reducción de devianza Binomial Negativa (NB), con incorporación del offset de exposición para modelar la frecuencia de siniestros, y un árbol de regresión para montos positivos de severidad, optimizado mediante la reducción de devianza Gamma. Se analiza la experiencia del año 2020 como una fotografía de la cartera, la estructura de la información es de tipo póliza–periodo: dentro del periodo observado se registran exposición, conteo de siniestros y pérdida agregada. Por ello, el término 'seguimiento' se entiende operativamente como el registro de dichas magnitudes durante la vigencia del periodo de observación.

3.2. NIVEL DE INVESTIGACIÓN

El estudio es aplicado y de alcance comparativo - predictivo. Se evalúa cómo cambian las métricas de desempeño al modificar, de manera controlada, tres elementos: (i) la política de remuestreo; (ii) los criterios de Pre-poda (profundidad máxima, tamaño mínimo de hoja y ganancia mínima en devianza para autorizar un corte); y (iii) la Post-poda REP, que compara cada subárbol con una hoja única sobre un conjunto de validación. La selección de configuraciones se guía por Devianza como métrica principal de cada familia (Binomial Negativa en frecuencia, Gamma en severidad) y por Pseudo D^2, MAE y RMSE como métricas complementarias, además de WAPE calibrado en la prima pura agregada por segmentos.

3.3. MÉTODOS DE TRABAJO

Dado que el enfoque es cuantitativo e hipotético-deductivo, se formulan hipótesis nulas y se contrastan mediante el entrenamiento de árboles de decisión que eligen sus cortes maximizando la reducción de devianza en cada paso.

Modelo de frecuencia (Binomial Negativa con offset)

Los datos de frecuencia son conteos por póliza con una exposición asociada (por ejemplo, tiempo asegurado). En el árbol, cada hoja aprende una tasa de siniestros por unidad de exposición, y la predicción por póliza se obtiene multiplicando la tasa de su hoja por su exposición. El modelo incorpora un parámetro de dispersión propio de la Binomial Negativa (que habilita varianzas mayores que Poisson), estimado en el conjunto de entrenamiento mediante un método de momentos que respeta el offset. La tasa de hoja se obtiene con un método de Newton con búsqueda de paso e inicialización estable, y luego se suaviza hacia la tasa promedio de entrenamiento con un peso configurable para ganar estabilidad cuando la exposición en la hoja es reducida. Para cortar un nodo, el algoritmo prueba umbrales sobre cada predictor y elige el que maximiza la reducción de devianza NB. En cortes sobre variables con faltantes, esas observaciones se envían sistemáticamente a la rama derecha a fin de preservar un tratamiento consistente.

Modelo de severidad (Gamma)

Los datos de severidad son montos positivos. En cada hoja, la predicción es la media local de los montos de la hoja. El criterio de corte es, nuevamente, la reducción de devianza (en este caso, la correspondiente a Gamma). Si la variable de corte presenta faltantes, el algoritmo prueba enviarlos a izquierda o derecha y adopta la opción que mejor reduce la devianza.

Selección de cortes y control de complejidad

Para cada predictor, el algoritmo ordena sus valores y evalúa de forma eficiente los puntos de corte. Si un predictor exhibe muchos valores distintos, se limita el número de umbrales evaluados para controlar el costo computacional. Se exige Pre-poda: no se autoriza un corte si los nodos resultantes serían demasiado pequeños, si la mejora en devianza es insuficiente o si ya se alcanzó la profundidad máxima.

Post-poda por reducción de error (REP) con devianza

La post-poda REP se aplica usando la misma pérdida del entrenamiento (devianza NB en frecuencia, Gamma en severidad). En cada nodo interno, se compara el subárbol con su sustitución por una hoja (constante MLE) sobre una validación de poda (25 %). Se poda si la devianza no empeora (o mejora), procediendo de abajo hacia arriba. El remuestreo respecto a la validación de poda sigue un esquema de aplicación: (i) solo entrenamiento (no se remuestrea la validación de poda), (ii) entrenamiento + validación de poda, o (iii) solo validación de poda (no se remuestrea el entrenamiento). Antes de podar o predecir en validación se alinean columnas para garantizar compatibilidad.

Reproducibilidad

Todas las decisiones —hiperparámetros, remuestreo, poda y particiones— se registran en consola. Se emplean semillas fijas para garantizar particiones pseudoaleatorias reproducibles y se imprime la lista de columnas efectivamente utilizadas por cada componente. Este registro asegura trazabilidad, auditoría y reproducibilidad del experimento.

3.4. POBLACIÓN Y VARIABLES

La población de estudio es la cartera completa de Siniestros de Automóviles con Pérdida Total reportada a la Superintendência de Seguros Privados (SUSEP) en 2020 (Brasil), con 7 053 495 registros. La unidad de análisis es la póliza-año (registro 2020) y el ámbito geográfico es Brasil.

Las métricas de evaluación son las siguientes. La Devianza —derivada de la verosimilitud— se utiliza como métrica principal; menor devianza implica mejor ajuste, empleando la forma correspondiente a cada familia (NB en frecuencia, Gamma en severidad). El Pseudo D^2 mide la mejora relativa frente a un modelo nulo (0 indica ausencia de mejora; 1, mejora perfecta). En frecuencia se reporta solo con offset, que predice la tasa promedio de entrenamiento multiplicada por la exposición de cada póliza. En severidad se reporta contra un nulo constante (la media del conjunto de prueba). Se incluyen MAE y RMSE para dimensionar el error en unidades naturales. La prima pura (PP) se obtiene como el producto de frecuencia esperada por severidad esperada por póliza y luego se agrega por segmentos. Sobre esos totales por grupo se aplica una calibración global por

modelo (un único factor multiplicativo) para que la suma total predicha coincida con la suma total observada. Finalmente, el desempeño se resume en un único WAPE calibrado por modelo, calculado a partir de los totales por segmento.

Los factores experimentales son dos. Primero, el remuestreo: en frecuencia se remuestrea por tasa (siniestros/exposición), con los tres modos (sin remuestreo, sobremuestreo y submuestreo); en severidad se remuestrea por cuantiles del monto positivo, con los mismos tres modos. Segundo, la complejidad y poda del árbol: Pre-poda (límites a profundidad, tamaño mínimo de hoja y ganancia mínima) y Post-poda REP, con definición explícita del escenario de aplicación del remuestreo en la validación de poda.

Los predictores incluyen atributos del asegurado, del vehículo y de la póliza (edad, año del vehículo, región, sexo, variables indicadoras de modelo y de tarifa, y la exposición). En frecuencia se crea además LOG_EXPOSICAO1 como predictor adicional. La variable EXPOSICAO1 no se utiliza directamente como variable de partición, dado que cumple el rol de offset en el modelo de frecuencia. Sin embargo, se incluye LOG_EXPOSICAO1 como predictor auxiliar para capturar heterogeneidad residual asociada a exposiciones parciales o agregadas, sin sustituir el papel actuarial del offset.

3.5. TÉCNICAS E INSTRUMENTOS DE RECOLECCIÓN DE DATOS

La fuente es SUSEP (Automóviles, 2020). Se generan dos archivos previos al modelado: modelo_frecuencia.csv (con frecuencia y exposición) y modelo_severidad.csv (con montos de severidad positivos). El script aplica controles de calidad: lectura tipada, exposición mínima (ajustada a un valor pequeño cuando es muy baja), creación de variables indicadoras y de la variable LOG_EXPOSICAO1, alineación de columnas (completando faltantes con 0 cuando corresponde) y eliminación de predictores constantes. Se utilizan semillas fijas y se imprime la lista completa de columnas finalmente usadas por cada componente.

3.6. TÉCNICAS E INSTRUMENTOS DE ANÁLISIS Y PROCESAMIENTO

El entorno de trabajo es Python, con NumPy y Pandas. Se utiliza train_test_split de scikit-learn únicamente para particionar en entrenamiento/validación/prueba. No se emplean librerías de árboles “caja negra”: los modelos se implementan desde cero.

Frecuencia (Binomial Negativa con offset)

El árbol optimiza en cada nodo la reducción de devianza NB. La dispersión se estima en entrenamiento con un método de momentos que incorpora el offset; para comparar modelos con justicia en prueba, se fija esa misma dispersión de referencia obtenida en entrenamiento. La tasa de hoja se estima por Newton con búsqueda de paso, con inicialización sensata y un suavizado hacia la tasa promedio. Los hiperparámetros (profundidad máxima y mínimo de observaciones por hoja) se fijan tras una búsqueda en una muestra de la base de datos y actúan como límites superiores al crecimiento del árbol. La predicción es exposición × tasa de hoja, con pisos y topes numéricos opcionales para evitar extremos. En cortes sobre variables con faltantes, las observaciones se dirigen a la rama derecha de modo consistente. Los candidatos de corte se generan ordenando valores y probando umbrales; si hay demasiados distintos, se reduce el número de evaluaciones para contener el costo computacional. La exposición (EXPOSICAO1) no participa como variable de corte; su logaritmo (LOG_EXPOSICAO1) se incluye como predictor.

Severidad (Gamma)

El árbol optimiza la reducción de devianza Gamma. La predicción de hoja es la media local de los montos (siempre positiva). Los hiperparámetros (profundidad y mínimo de observaciones) se eligen mediante búsqueda en muestra y actúan como límites superiores. Ante faltantes en la variable de corte, el algoritmo prueba enviarlos a izquierda o derecha y adopta la opción que mejor resultado produce.

Pre-poda y Post-poda

La Pre-poda impide cortes si el nodo resultante queda demasiado pequeño, si la mejora en devianza es insuficiente o si se alcanzó la profundidad máxima permitida. La Post-poda REP reserva un 25 % del entrenamiento para validación, compara el subárbol con una hoja y simplifica cuando no hay pérdida en esa validación. Los escenarios de aplicación del remuestreo son: remuestreo solo en entrenamiento, remuestreo en entrenamiento y validación interna, y remuestreo solo en validación interna de poda, según corresponda al diseño experimental. En todos los casos, el conjunto de prueba permanece sin remuestreo. En todos los casos, antes de podar o predecir se alinean las columnas. Importante: las métricas reportadas fuera de muestra se calculan siempre sobre el conjunto de prueba (20%) sin remuestreo, de modo que el desempeño se evalúa en la distribución original del portafolio; el remuestreo, cuando existe, se usa solo como mecanismo operativo para estabilizar el ajuste/selección interna (p. ej., poda).

Remuestreo (balanceo)

En frecuencia, el remuestreo se realiza por tasa y permite fijar una proporción objetivo de positivos, controlando duplicados; se aplica según la configuración del árbol (pre-poda, árbol completo o post-poda). En severidad, el remuestreo se realiza por cuantiles del monto positivo con el mismo esquema. La implementación trabaja por índices para evitar duplicar grandes estructuras tabulares.

Búsqueda y validación

Para contener el costo computacional, las curvas de aprendizaje y la validación K-fold se ejecutan al final del flujo principal de procesamiento. Asimismo, se adopta K=3 en la validación cruzada por restricciones computacionales (muestras muy grandes) y porque la variabilidad entre pliegues resultó baja (≈ 5–6 % en devianza media de severidad; ≈ 0,0003–0,0008 en frecuencia), suficiente para ordenar configuraciones. El flujo operativo usa una partición 80% en entrenamiento / 20% en prueba. Adicionalmente, para la post-poda REP, dentro del conjunto de entrenamiento se reserva un 25% como validación de poda, quedando el 75% restante como entrenamiento efectivo para el crecimiento del árbol.

Esquema de validación y métricas

La partición es 80 % entrenamiento y 20 % prueba. En frecuencia, la partición estratifica por presencia/ausencia de siniestros para estabilizar las comparaciones. En frecuencia, sobre el conjunto de prueba, se evalúa con la misma dispersión de referencia estimada en entrenamiento y se reportan devianza, Pseudo D^2 (contra el nulo con offset) y MAE/RMSE. En severidad, sobre el conjunto de prueba, el modelo nulo corresponde a la media del conjunto de prueba; se reportan devianza, Pseudo D^2, MAE y RMSE. Para prima pura y WAPE calibrado, se calcula la PP por póliza, se agrega por segmentos (región, sexo, tramos de edad y de año de vehículo) y se aplica un único factor de calibración por modelo para alinear totales; con la versión calibrada se computa el WAPE agregado del modelo.

3.7. ETAPAS DE INTERVENCIÓN DEL ESTUDIO

La Fase 1 (extracción y preparación) comprende la generación de los archivos de frecuencia y severidad, el control de exposición mínima, la creación de variables derivadas, la eliminación de predictores constantes y la alineación de columnas. La Fase 2 (modelado de frecuencia) aplica una partición 80/20 — estratificada por presencia de siniestros —, ejecuta el remuestreo según el escenario de aplicación definido, aplica pre-poda, realiza

post-poda con validación del 25 % (consistente con el escenario de aplicación del remuestreo) y evalúa con Devianza, Pseudo D^2 (offset y constante), MAE y RMSE. La Fase 3 (modelado de severidad) replica la lógica con partición 80/20, remuestreo por cuantiles del objetivo según el escenario de aplicación definido, pre-poda, post-poda (25 %) y evaluación con Devianza, Pseudo D^2, MAE y RMSE. La Fase 4 (síntesis de prima pura y evaluación agregada) calcula la PP en la muestra de prueba, agrega por segmentos y aplica una calibración global por modelo; se reporta el WAPE calibrado consolidado (un único valor) para cada una de las nueve configuraciones y, de apoyo, un ranking de segmentos por monto observado, junto con el factor de calibración aplicado. La Fase 5 (documentación y reproducibilidad) registra semillas, columnas efectivas, parámetros, escenarios de remuestreo y métricas; el script entrega resúmenes comparativos para las nueve configuraciones (tres configuraciones de poda × tres escenarios de remuestreo) en cada componente y la síntesis de PP y WAPE.

CAPÍTULO IV. ANÁLISIS

4.1. DESCRIPTIVOS Y EXPLORATORIOS

Una vez establecido que el punto de partida empírico de la tarificación en seguros de no vida es una tabla de experiencia (tabla de exposición y siniestros), análoga en sentido operativo a una 'tabla de vida' únicamente como analogía organizacional de exposición y ocurrencia de eventos (sin implicar análisis de supervivencia, censura ni funciones de riesgo/hazard), en tanto organiza el registro de exposición y siniestros durante la ventana de observación —en esta tesis, un único periodo anual— mediante tres magnitudes fundamentales: exposición, número de siniestros y pérdida agregada. En términos generales, la unidad de observación se denota como póliza–periodo (i, t), sin embargo, dado que la base empírica corresponde a un solo periodo (año 2020), cada registro es una póliza-año i. En adelante, se suprime t y se trabaja con i como unidad de observación, registrando la exposición e_i (vehículo-año), el número de siniestros N_i y la pérdida agregada L_i.

Para el análisis descriptivo, estas magnitudes se agregan por grupos g definidos por una variable explicativa (o por un conjunto de ellas), obteniendo los totales $E_g = \sum e_{it}$, $N_g = \sum N_{it}$ y $L_g = \sum L_{it}$. Bajo esta estructura, la prima pura observada por unidad de exposición del grupo g se expresa como:

$$\hat{\pi}_g = \frac{L_g}{E_g} = \left(\frac{N_g}{E_g}\right)\left(\frac{L_g}{N_g}\right) \tag{73}$$

donde N_g/E_g corresponde a la frecuencia observada, entendida como tasa de siniestros por unidad de exposición y L_g/N_g a la severidad media. Nótese que la severidad L_g/N_g se entiende condicional a $N_g > 0$; si en un segmento se observa $N_g = 0$, entonces $L_g = 0$ y la prima pura observada del segmento es $\hat{\pi}_g = 0$. En consecuencia, las Tablas 1–7 deben interpretarse como "celdas" (vistas marginales) de la tabla de experiencia, reportando N_g(conteo), L_g(pérdida agregada) y, cuando aplica, E_g(exposición).

Se dispone de datos administrativos de seguros de automóviles publicados por la Superintendência de Seguros Privados (SUSEP), correspondientes al mercado brasileño del año 2020, que abarcan información sobre la frecuencia de siniestros con pérdida total (eventos reportados por póliza) y la pérdida agregada (monto indemnizado total por póliza, L_i) y, cuando $N_i > 0$, la severidad media (monto medio por siniestro, $Y_i = L_i/N_i$). Antes de construir los modelos predictivos, se realiza un análisis exploratorio de las principales variables explicativas consideradas, con el fin de comprender su distribución e impacto

potencial en la siniestralidad, manteniendo coherencia con las definiciones actuariales previamente mencionadas.

Año del Vehículo

El año del modelo del vehículo asegurado (ANO_MODELO) se utiliza para identificar la antigüedad del vehículo, lo que influye en su valor y en el perfil de riesgo, ya que modelos más antiguos y más nuevos pueden presentar diferencias en tecnología, seguridad y comportamiento en siniestros. Para fines de presentación, se presentan los años en rangos:

Tabla 1: Año de Vehículo - conteo de siniestros (N_g) y pérdida agregada (L_g) por rango de año del vehículo

Año Inicial	Año Final	Antigüedad	Conteo de siniestros ($\mathbf{N_g}$)	Pérdida agregada R$ ($\mathbf{L_g}$)	Severidad Media R$ ($\mathbf{L_g/N_g}$)
2018	**2020**	Muy nuevos	10,005	698,548,900	69,819.98
2010	**2017**	Nuevos	28,639	1,207,892,000	42,176.47
2000	**2009**	Modernos	6,820	162,799,000	23,870.82
1980	**1999**	Antiguos	381	6,704,364	17,596.76
0	**0**	No informado	3	16,527	5,509.00

Se observa una tendencia clara: la severidad media aumenta conforme el vehículo es más reciente. Los vehículos muy nuevos (2018–2020) presentan, en promedio, siniestros más costosos, mientras que los vehículos antiguos muestran severidades medias menores. Este patrón es consistente con que vehículos recientes suelen tener mayor valor de reposición y costos de reparación más altos.

Tipo de Vehículo

El tipo y origen del vehículo (COD_TARIF) permite segmentar el vehículo asegurado según su tipo y origen, lo que permite segmentar el riesgo según las características particulares de cada clase.

1 – Vehículo nacional: Automóviles fabricados y comercializados en el país.

2 – Vehículo importado: Automóviles importados.

3 – Pick-up (nacional e importado): Camionetas que se utilizan tanto para uso personal como comercial, nacional e importado.

4 – Vehículo de Carga (nacional e importado): Camiones y otros vehículos destinados al transporte de mercancías.

5 – Motocicleta (nacional e importado): Vehículos de dos ruedas.

6 – Ómnibus (nacional e importado): Vehículos de transporte colectivo.

7 – Utilitario (nacional e importado): Vehículos multifuncionales con usos variados.

9 – Otros: Agrupa aquellos casos que no se clasifican en las categorías anteriores.

Tabla 2: Tipo de Vehículo - conteo de siniestros (N_g) y pérdida agregada (L_g) por tipo del vehículo

Tipo de Vehículo	Conteo de siniestros ($\mathbf{N_g}$)	Pérdida agregada R$ ($\mathbf{L_g}$)	Severidad Media R$ ($\mathbf{L_g/N_g}$)
Vehículo nacional	29,949	959,694,083	32,044.28
Pick-up (nacional e importado)	9,452	646,076,445	68,353.41
Vehículo importado	2,519	167,016,829	66,302.83
Motocicleta (nacional e importado)	1,334	29,838,809	22,367.92
Vehículo de Carga (nacional e importado)	1,132	116,016,384	102,487.97
Utilitario (nacional e importado)	812	45,861,989	56,480.28
Otros	609	110,132,714	180,841.89
Ómnibus (nacional e importado)	41	1,323,192	32,272.98

Del análisis por tipo se desprende que Vehículo de Carga y, especialmente, la categoría Otros presentan severidades medias elevadas, seguidas por pick-ups e importados.

Motocicletas presentan la severidad media más baja. Este resultado sugiere que el tipo de vehículo es un determinante relevante del costo medio del siniestro; por ello, para el modelado se emplean variables indicadoras por categoría, consolidando niveles de baja frecuencia cuando sea necesario para estabilidad.

Marcas de Vehículo

Las Marcas de Vehículo identifican al fabricante o la marca comercial bajo la cual se produce y comercializa el vehículo. Estas marcas no solo reflejan la reputación y la calidad del automóvil. Para una mejor presentación, se agrupan según su país de procedencia dado que se han trabajado con numerosas marcas.

Tabla 3: Marcas de Vehículo - conteo de siniestros (N_g) y pérdida agregada (L_g) por marcas según país del vehículo

Marcas según país	Conteo de siniestros $(\mathbf{N_g})$	Pérdida agregada R$ $(\mathbf{L_g})$	Severidad Media R$ $(\mathbf{L_g/N_g})$
Estados Unidos	12,005	464,347,245	38,679.49
Japón	9,056	450,023,682	49,693.43
Alemania	7,527	408,529,567	54,275.22
Italia	7,473	222,240,289	29,739.10
Francia	4,198	133,254,575	31,742.40
Corea del Sur	2,595	116,462,415	44,879.54
Otros	1,846	137,363,034	74,411.18
Suecia	525	96,848,792	184,473.89
Reino Unido	325	35,038,100	107,809.54
China	252	8,646,732	34,312.43
Brasil	32	2,885,398	90,168.69
Austria	8	140,074	17,509.25
India	6	180,542	30,090.33

Agrupar por país revela severidades medias elevadas en Suecia y Reino Unido; sin embargo, estos grupos presentan baja cantidad de eventos (N_g pequeño), por lo que las medias pueden ser inestables. El patrón es consistente con diferencias de precio del

vehículo y costos de repuestos asociados al origen. En el modelado, se incorporan variables indicadoras por grupo, cuidando la estabilidad mediante reagrupaciones cuando la frecuencia sea reducida.

Región

La región geográfica (REGIAO) identifica el lugar donde se encuentra el riesgo asegurado, y en la base de datos se registran 41 códigos distintos, cada uno correspondiente a una región específica del Brasil (por ejemplo, "RS - Met. Porto Alegre e Caxias do Sul", "SP - Met. de São Paulo", etc.).

Para efectos de presentación, las regiones se agrupan según la densidad poblacional. Es decir, mediante un diccionario de mapeo, se asigna a cada código de región una categoría de densidad (por ejemplo, "Alta", "Media" o "Baja").

Tabla 4: Región - conteo de siniestros (N_g) y pérdida agregada (L_g) por densidad poblacional acorde a la región del vehículo

Densidad	Conteo de siniestros ($\mathbf{N_g}$)	Pérdida agregada R$ ($\mathbf{L_g}$)	Severidad Media R$ ($\mathbf{L_g/N_g}$)
Alta	23,995	1,006,586,000	41,949.82
Media	19,194	892,588,300	46,503.51
Baja	2,552	169,548,700	66,437.58
Desconocido	107	7,236,952	67,635.07

Las regiones de baja densidad exhiben mayor severidad media pese a registrar menor conteo de eventos. Las regiones altamente densas concentran más eventos, pero con severidades medias más moderadas. Para el modelado se considerarán variables indicadoras por región (o por agrupación), evaluando su efecto en frecuencia y severidad.

Edad

La edad (IDADE) es fundamental para la modelización, ya que la edad del asegurado influye en el perfil de riesgo y en la probabilidad y costo de los siniestros. Se trabaja con seis rangos provistos en la base original.

Tabla 5: Edad - conteo de siniestros (N_g) y pérdida agregada (L_g) por edad del asegurado del vehículo

Edad	Conteo de siniestros ($\mathbf{N_g}$)	Pérdida agregada R$ ($\mathbf{L_g}$)	Severidad Media R$ ($\mathbf{L_g/N_g}$)
Entre 18 y 25 años	1,622	53,363,335	32,899.71
Entre 26 y 35 años	6,365	259,108,695	40,708.36
Entre 36 y 45 años	9,552	430,777,990	45,098.20
Entre 46 y 55 años	7,156	323,331,491	45,183.27
Mayor que 55 años	8,599	388,482,450	45,177.63
No Data	12,554	620,896,484	49,458.06

La severidad media aumenta desde el grupo 18–25 hacia rangos mayores y luego se estabiliza alrededor de 45 mil. El grupo “No Data” es numeroso y exhibe severidad elevada, por lo que se codifica explícitamente como categoría para no perder trazabilidad.

Género

El género (SEXO) se codifica como “F” (femenino) y “M” (masculino), además de valores faltantes. Esta variable puede reflejar diferencias de perfil de conducción y características del riesgo; no obstante, su interpretación debe distinguir entre conteos (N_g)y tasas por exposición (N_g/E_g), las cuales se abordarán en el modelado de frecuencia al incluir la exposición. Dado que la base empírica corresponde a un único periodo (2020), las cantidades por grupo $g$se obtienen agregando sobre pólizas-año: $E_g = \sum_{i \in g} e_i,\ N_g = \sum_{i \in g} N_i$y $L_g = \sum_{i \in g} L_i$.

Tabla 6: Género - conteo de siniestros (N_g) y pérdida agregada (L_g) por género del asegurado del vehículo

SEXO	Conteo de siniestros ($\mathbf{N_g}$)	Pérdida agregada R$ ($\mathbf{L_g}$)	Severidad Media R$ ($\mathbf{L_g/N_g}$)
M	23,008	1,064,368,000	46,260.78
F	12,749	531,749,900	41,709.15
No data	10,091	479,842,900	47,551.57

El grupo masculino presenta mayor conteo de eventos y una severidad media superior a la del grupo femenino. Sin embargo, el análisis causal del "riesgo" en términos actuariales requiere controlar por exposición; por ello, el contraste formal en frecuencia se realizará en el modelo de frecuencia mediante offset/ajuste por exposición. Para el modelado se crean variables indicadoras por categoría (SEXO_M, SEXO_F), manteniendo "No data" como nivel explícito.

En estas tablas, 'Frecuencia Total' corresponde al conteo de siniestros N_g; la frecuencia actuarial (tasa) es N_g/E_g y se aborda en el modelado de frecuencia controlando por exposición.

Exposición

La exposición (EXPOSICAO1) representa la masa temporal asegurada dentro de la ventana de observación (vehículo-año). Un valor menor que 1 indica fracción de año en vigor, mientras que valores mayores que 1 son esperables cuando la información se presenta agregada por segmentos (suma de múltiples pólizas/vehículos o periodos dentro del segmento).

Tabla 7: Exposición - conteo de siniestros (N_g) y pérdida agregada (L_g) por exposición del vehículo

Completitud	Exposición Total ($\mathbf{E_g}$)	Conteo de siniestros ($\mathbf{N_g}$)	Pérdida agregada R$ ($\mathbf{L_g}$)	Severidad Media R$ ($\mathbf{L_g/N_g}$)
Mayor o igual a 1	490,815.47	29,681	1,201,561,000	40,482.50
Menor a 1	3,732.45	16,167	874,399,000	54,085.42

Se observa que las pólizas con exposición parcial exhiben severidad media mayor. Dado que, actuarialmente, la frecuencia corresponde a una tasa de siniestros por unidad de exposición (N/E), la exposición e_{it}se incorpora en el modelo de conteos mediante un offset $\log(e_{it})$, de modo que $\mathbb{E}[N_{it} \mid X_{it}] = e_{it}\,\lambda(X_{it})$ y el árbol estima una tasa por unidad de exposición. Adicionalmente, se permite que $\log(e_{it})$ actúe como variable de segmentación (variable candidata de partición) cuando exista heterogeneidad residual asociada a la exposición (p. ej., exposiciones parciales o agregadas), con el fin de capturar desviaciones de proporcionalidad (elasticidades diferenciales) no absorbidas por el offset, trabajando en escala log para estabilizar y evitar cortes en la escala original de e.

Síntesis operativa del análisis exploratorio

En términos de severidad media, los vehículos más recientes y ciertas tipologías —en particular carga, pick-up, importados y utilitarios— exhiben costos medios superiores; las motocicletas se sitúan por debajo del promedio. La procedencia de la marca refuerza este patrón con medias elevadas en algunos orígenes europeos, aunque en ciertos grupos la baja cantidad de eventos aconseja reagrupación para estabilidad. Geográficamente, las regiones de baja densidad muestran severidades más altas pese a menor cantidad de eventos. En el perfil del asegurado, el tramo 18–25 años registra la menor severidad y a partir de 36+ las severidades se estabilizan; la ausencia de dato (edad/género) es numerosa y costosa, por lo que se mantiene como categoría explícita.

Finalmente, dado que toda la modelización se apoya en la tabla de experiencia, el paso siguiente consistirá en estimar, para cada segmento, las componentes actuariales de la prima pura: frecuencia (N/E)y severidad (L/N), y su combinación (L/E), usando un enfoque no paramétrico de segmentación mediante árboles.

4.2. MODELO DE SEVERIDAD

Como estimador de severidad se trabaja con la subtabla con siniestro ($N_{it} \geq 1$). Para cada observación se define la severidad como $Y_{it} = L_{it}/N_{it}$, de modo que el modelo de severidad estima

$$\mu(x) = \mathbb{E}\left[Y_{it} \mid N_{it} > 0,\, X = x\right] \tag{74}$$

En esta sección se modela exclusivamente la severidad $\mu(x)$, entendida como el costo medio del siniestro condicionado **a ocurrencia**, por lo que el conjunto de modelamiento se construye con observaciones con siniestro $(N_{it} \geq 1)$ y con monto positivo $Y > 0$.

Operativamente, esto corresponde a trabajar con la “subtabla” de la tabla de experiencia donde existe pérdida, manteniendo la lógica de agregación por segmentos: cada hoja del árbol define una celda R_m sobre la cual la severidad estimada es constante. Dado que la severidad se define condicionada a ocurrencia ($Y > 0$), esta sección trabaja con la subpoblación con siniestro, manteniendo la lógica de tabla de experiencia y agregación por celdas (hojas).

Modelo: árbol de decisión con devianza Gamma

Se estima un modelo no paramétrico de severidad mediante un árbol de decisión usando devianza Gamma como criterio de partición, adecuado para montos positivos, asimétricos y heterocedásticos. El árbol induce una partición del espacio de covariables en regiones disjuntas $\{R_m\}$(hojas), de modo que:

$$\hat{\mu}(x) = \sum_{m=1}^{M} \hat{\mu}_m \ \mathbf{1}x \in R_m \tag{75}$$

donde $\hat{\mu}_m$ es la severidad media estimada en la hoja R_m. Desde el punto de vista actuarial, cada hoja R_m equivale a una celda final de la tabla de experiencia para severidad (subpoblación con siniestro), sobre la cual se resume la pérdida y se obtiene el costo medio esperado.

Se comparan tres configuraciones del árbol: pre-poda, árbol completo (sin poda) y post-poda. Además, se evalúan tres esquemas de remuestreo (sin remuestreo, sobremuestreo y submuestreo). Para evitar ambigüedad, en adelante los términos pre/completo/post se reservan exclusivamente para referirse a la familia del árbol (pre-poda / completo / post-poda).

Métricas principales

Luego de calibrar el modelo de severidad con devianza Gamma y los hiperparámetros seleccionados, se obtienen las métricas de desempeño de la Tabla 8.

Tabla 8: Métricas principales de Modelo Severidad (métricas: Devianza Gamma total (suma), Pseudo D², MAE, RMSE; configuraciones: completo/post/pre × sin remuestreo/sobremuestreo/submuestreo)

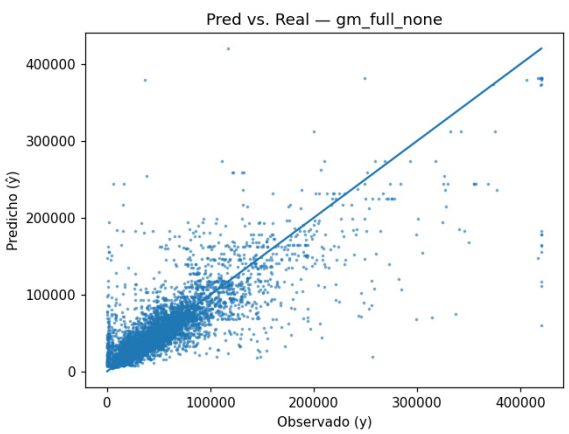
Modelo	Remuestreo	Devianza Gamma Total	Pseudo D²	MAE	RMSE
Árbol completo	Sin remuestreo	2,120	64.47%	11,946	28,459
Árbol completo	Sobremuestreo	2,130	64.30%	12,050	28,618
Árbol completo	Submuestreo	2,253	62.24%	12,637	29,930
Post-poda	Sin remuestreo	2,168	63.66%	12,198	28,790
Post-poda	Sobremuestreo	2,272	61.91%	13,286	31,053
Post-poda	Submuestreo	2,316	61.19%	12,607	29,157
Pre-poda	Sin remuestreo	2,123	64.42%	11,967	28,505
Pre-poda	Sobremuestreo	2,142	64.10%	12,051	28,687
Pre-poda	Submuestreo	2,241	62.43%	12,610	29,938

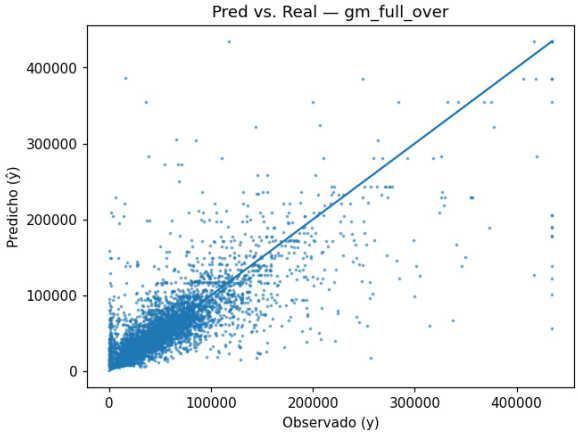
Nota: En esta tabla se reporta la devianza total $D_{\text{tot}} = \sum_{i=1}^{n} d_i$ (suma sobre las n observaciones del modelo de severidad). En las figuras y en la validación cruzada se reporta la devianza media por observación, $\bar{D} = D_{\text{tot}}/n$, para facilitar la comparación entre pliegues. Ambas medidas difieren solo por el factor $1/n$ cuando n es fijo.

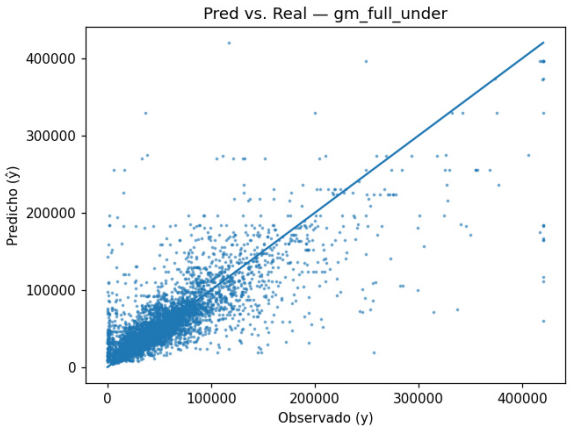
Las métricas de la Tabla 8 muestran un empate práctico entre pre-poda sin remuestreo y árbol completo sin remuestreo (diferencias <0,2 % en devianza/MAE/RMSE). En contraste, la post-poda deteriora el ajuste (mayor devianza y menor pseudo-D²), consistente con compresión de hojas y pérdida de sensibilidad en montos altos. El submuestreo empeora sistemáticamente (≈5–9 % en devianza; 2–5 % en RMSE) y el sobremuestreo no aporta mejoras robustas, especialmente cuando se combina con post-poda. En este problema, la política preferible es no remuestrear. Los valores de Pseudo D² cercanos a 64 % para Pre-poda y Completo sin remuestreo evidencian buen ajuste global pese a asimetría y heterocedasticidad; las caídas a 61–62 % en submuestreo y Post-poda con sobremuestreo reflejan pérdida de ajuste en colas.

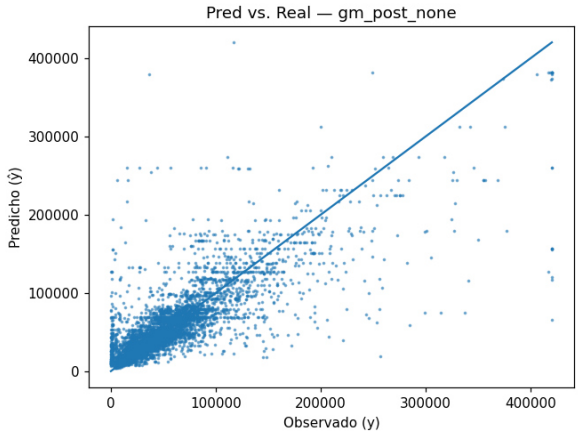
Predichos vs. Reales

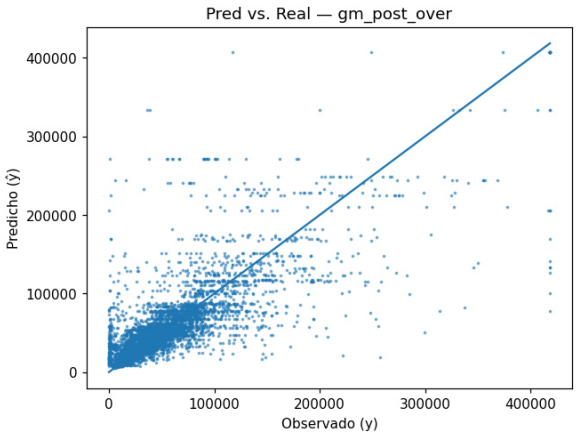
En los diagramas de dispersión Predicho (eje Y) vs. Real (eje X) se aprecia la relación entre observaciones y estimaciones. La línea de referencia $y = x$representa el ajuste ideal (predicción exacta).

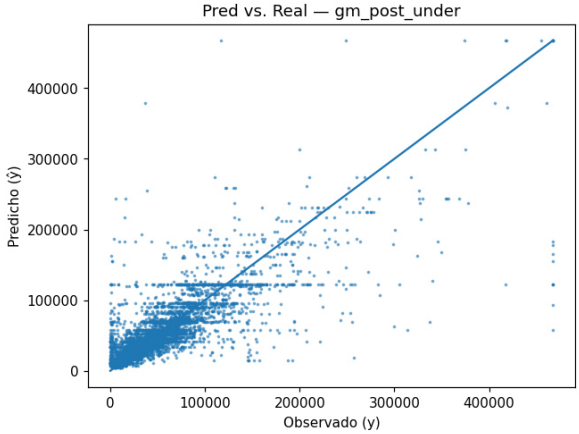












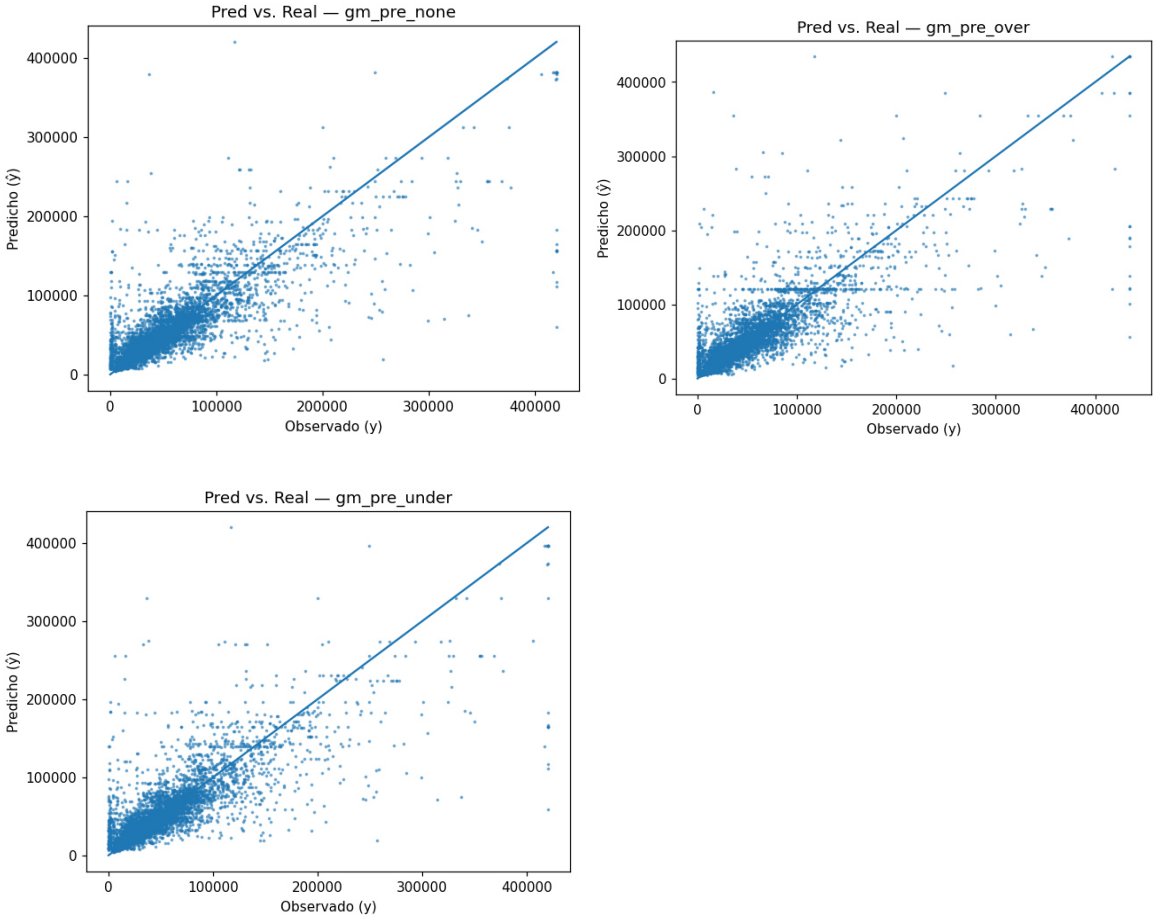
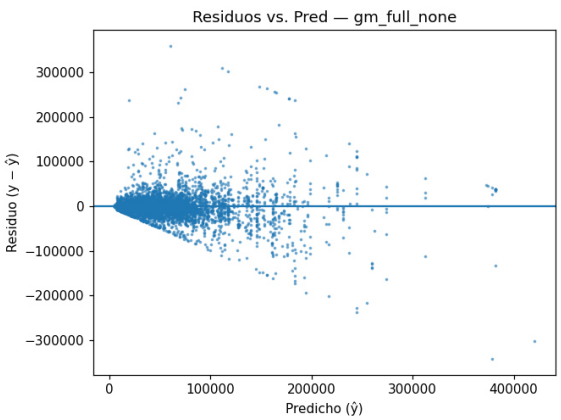


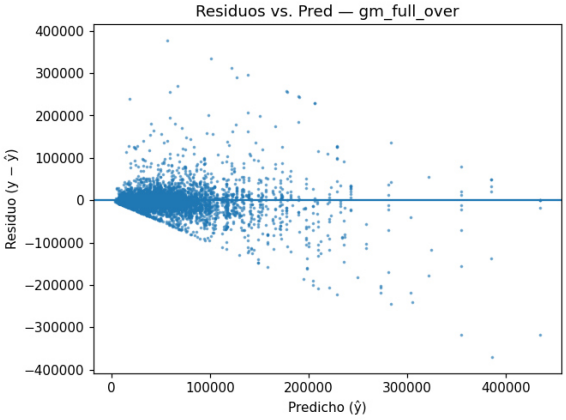
Figura 2: Predichos vs Reales Modelo Severidad (árbol Gamma; paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under); línea de referencia y = x)

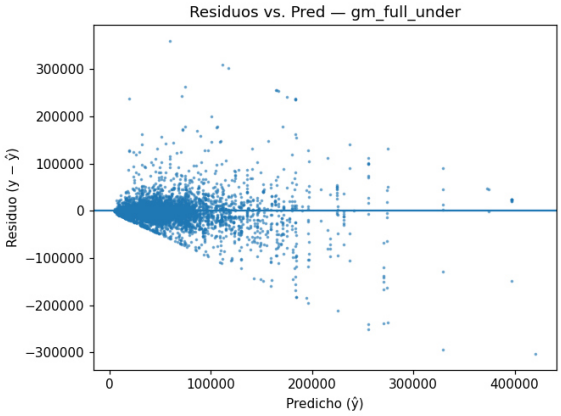
En montos bajos y medios, la nube se alinea con la recta $y = x$, señal de buena calibración del cuerpo. A partir de aproximadamente 300 000 se observa subestimación de extremos ($\hat{y} < y$). Pre-poda y Completo, ambos sin remuestreo, preservan el rango de $\hat{y}$; Post-poda lo comprime, reforzando el sesgo en la parte alta. El submuestreo incrementa la dispersión sin ganar resolución en cola; el sobremuestreo apenas cambia la geometría. El hecho de que RMSE supere ampliamente al MAE confirma la influencia de pocos errores muy grandes.

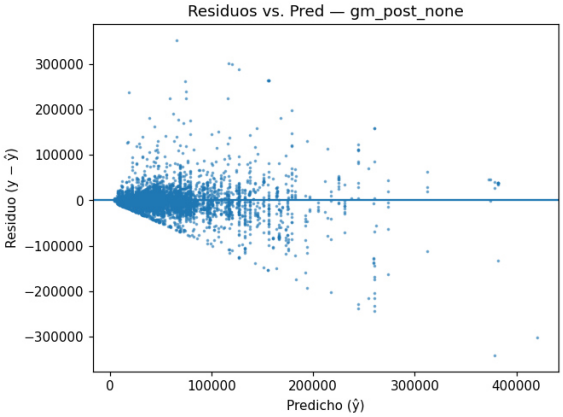
Residuos vs. Predichos

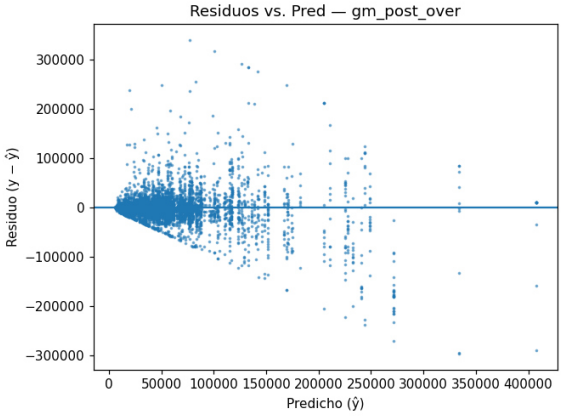
Las gráficas Residuos vs. Predichos (definidos como residuo $= y - \hat{y}$) permiten evaluar la distribución de errores a lo largo del rango de severidades predichas.

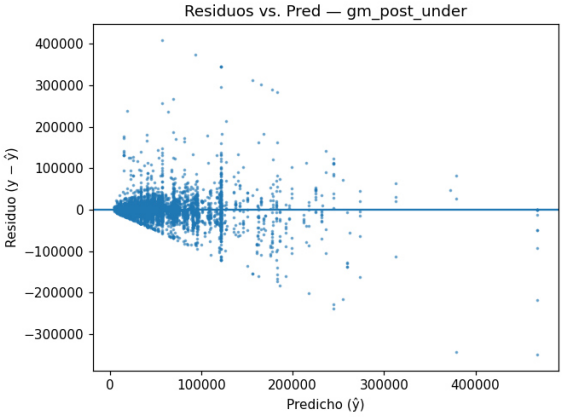

Residuos vs. Pred — gm_full_none
Residuo (y − ŷ)
Predicho (ŷ)

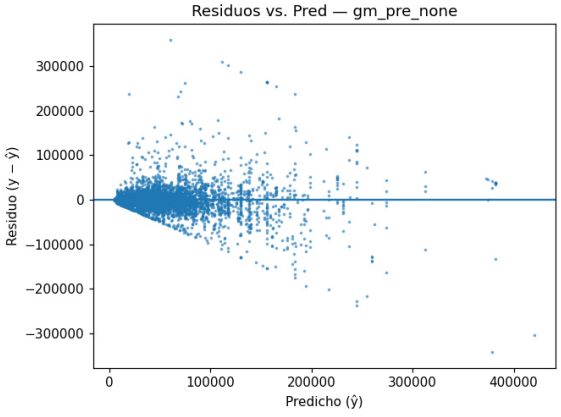

Residuos vs. Pred — gm_full_over
Residuo (y − ŷ)
Predicho (ŷ)

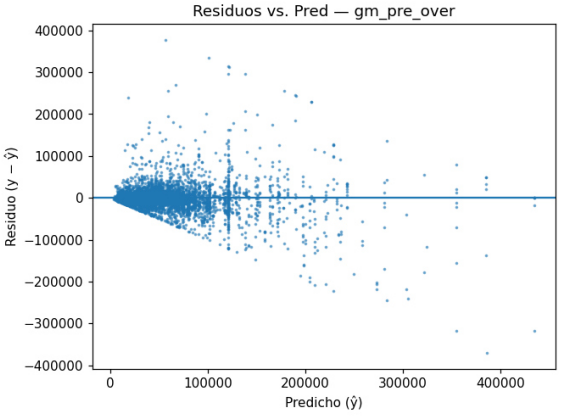

Residuos vs. Pred — gm_full_under
Residuo (y − ŷ)
Predicho (ŷ)


Residuos vs. Pred — gm_post_none
Residuo (y − ŷ)
Predicho (ŷ)


Residuos vs. Pred — gm_post_over
Residuo (y − ŷ)
Predicho (ŷ)


Residuos vs. Pred — gm_post_under
Residuo (y − ŷ)
Predicho (ŷ)


Residuos vs. Pred — gm_pre_none
Residuo (y − ŷ)
Predicho (ŷ)


Residuos vs. Pred — gm_pre_over
Residuo (y − ŷ)
Predicho (ŷ)

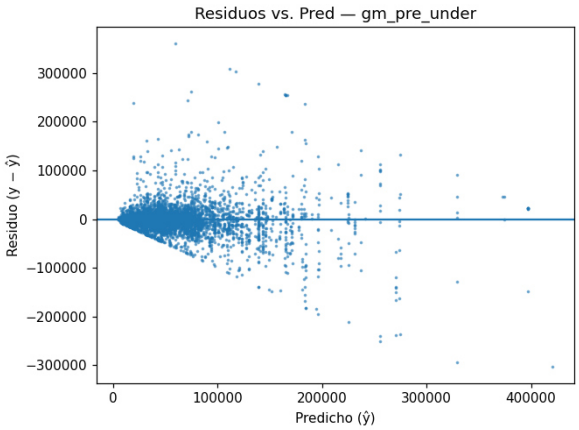
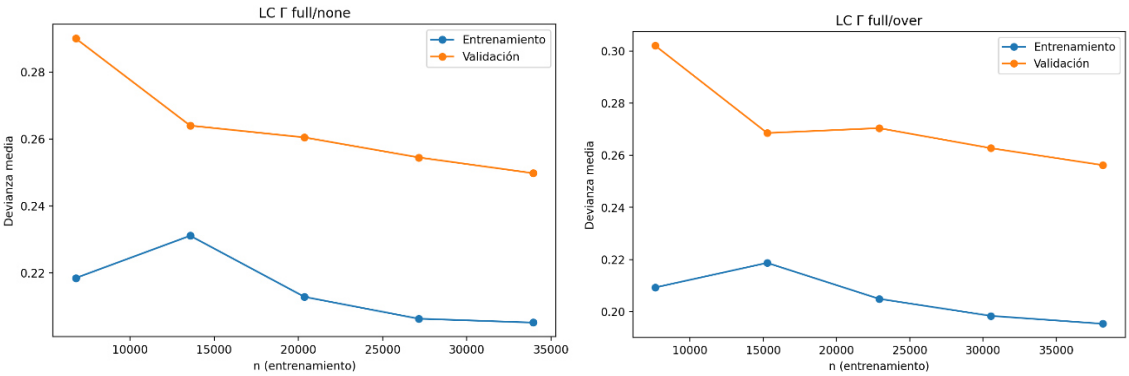


Figura 3: Residuos vs Predichos Modelo Severidad (árbol Gamma; paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under); residuo = y − ŷ)

Los residuos exhiben el patrón típico de severidad Gamma: varianza creciente con $\hat{y}$ y "bandas" por hoja (celdas) con medianas cercanas a cero en el cuerpo. Persisten residuos positivos grandes en la parte alta, consistentes con subestimación de extremos. La post-poda concentra más residuo positivo en cola y genera residuo negativo en zona media (efecto de recorte/compresión). El submuestreo ensancha la varianza; el sobremuestreo agrega cola negativa sin corregir la positiva.

Curvas de aprendizaje

Cada gráfico reporta la devianza media (más bajo es mejor) de entrenamiento (línea azul) y validación/prueba (línea naranja) a medida que aumenta el tamaño del conjunto de entrenamiento (*n*). Se comparan tres familias de árboles de decisión — pre-poda (pre), árbol completo (full) y post-poda (post)— bajo tres esquemas de remuestreo (sin remuestreo (none), sobremuestreo (over) y submuestreo (under).



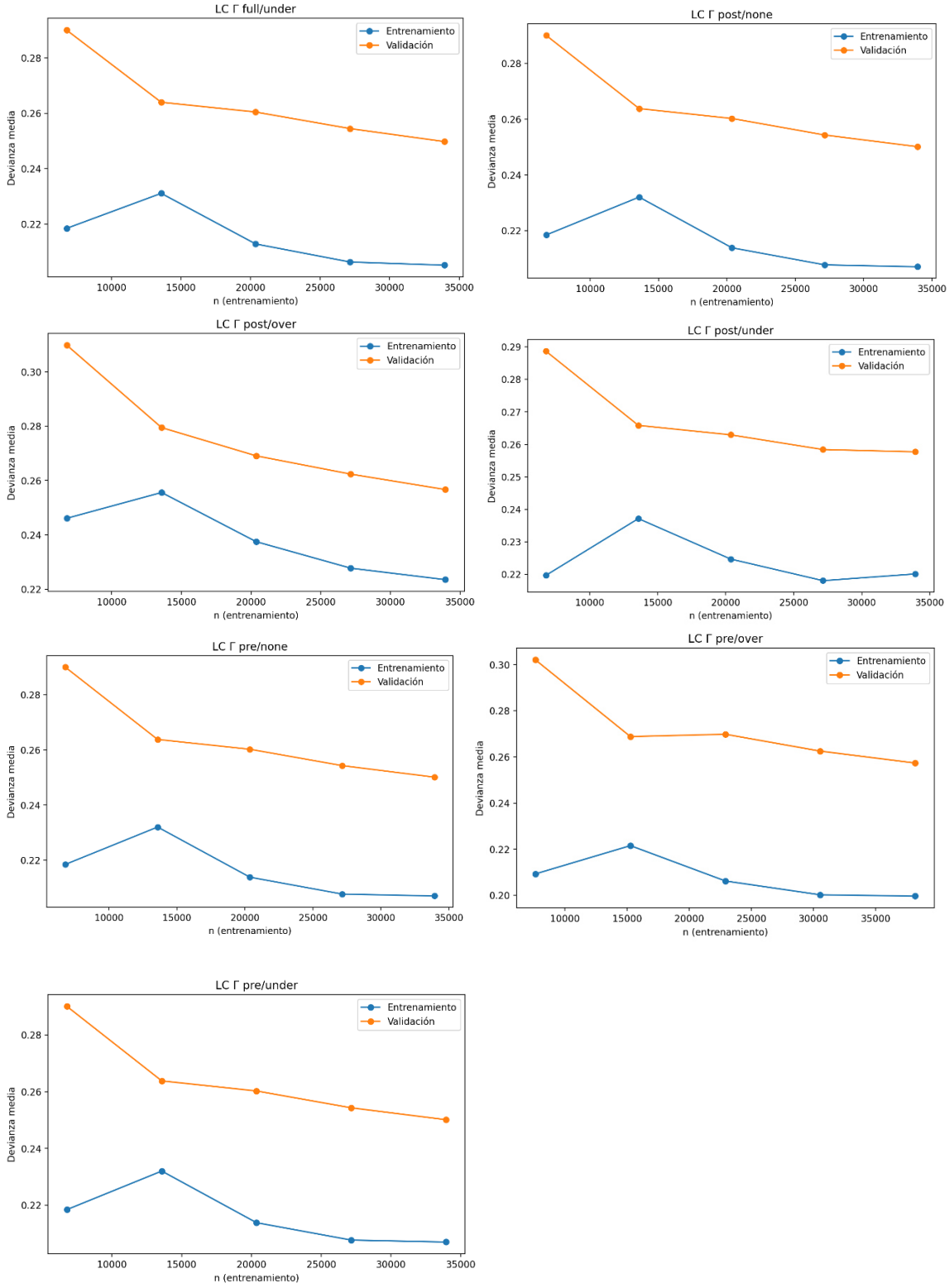


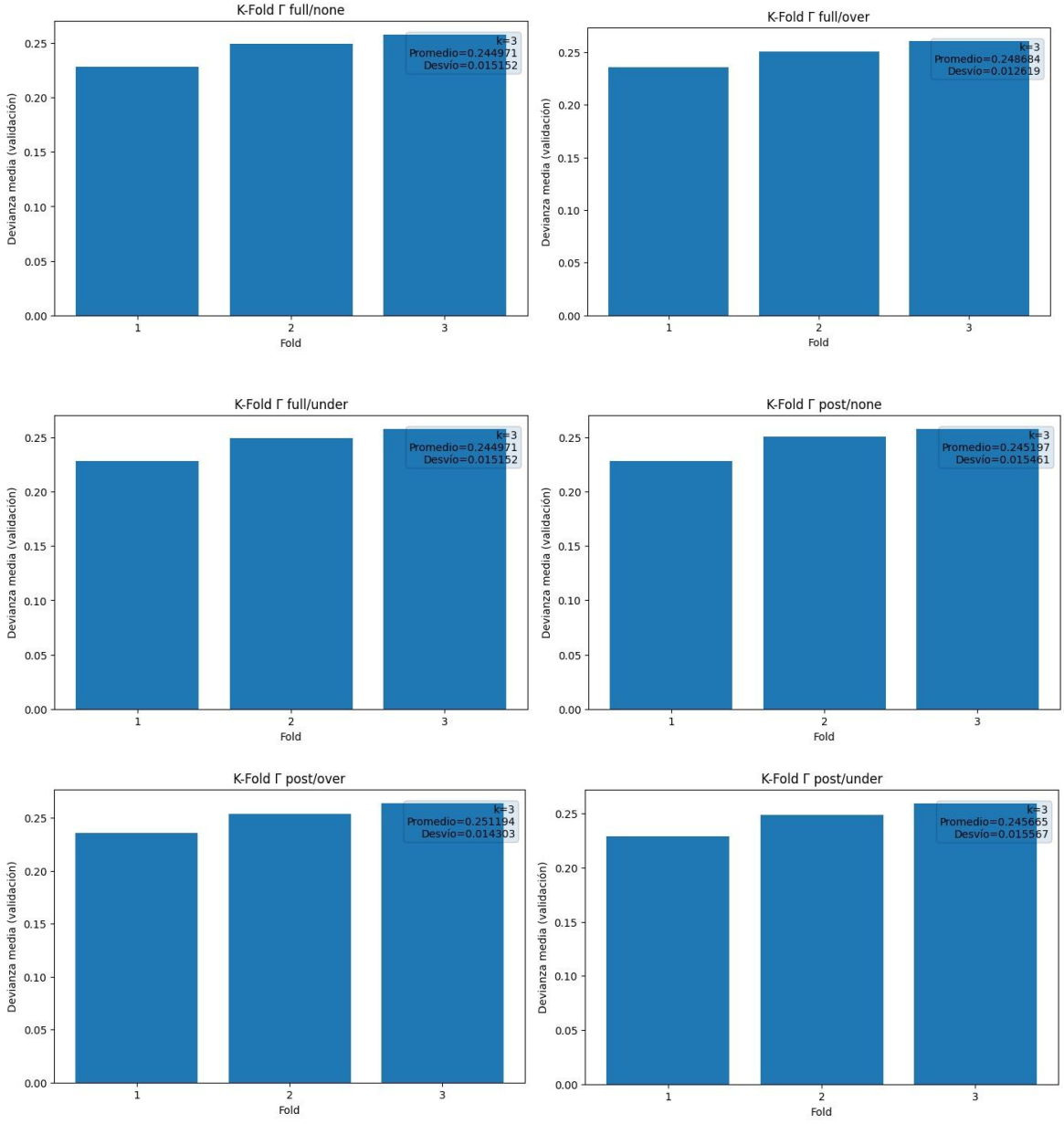
Figura 4: Curvas de aprendizaje Modelo Severidad (devianza media Gamma vs tamaño muestral; entrenamiento (azul) y validación/prueba (naranja); paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under))

La devianza de validación disminuye con el tamaño muestral (caída acumulada del orden de 15–20%) y la brecha entrenamiento–validación se reduce, indicando mitigación del

sobreajuste. Con muestras grandes, pre-poda y completo validan mejor que post-poda, y el esquema sin remuestreo es consistentemente igual o superior a sobremuestreo/ submuestreo.

Validación cruzada

Para comparar rigurosamente el desempeño y la estabilidad de los nueve modelos propuestos, se aplicó validación cruzada K-Fold con K = 3 (tres cortes). En cada fold se mantuvo la misma partición y semilla para asegurar comparabilidad entre configuraciones (*pre/completo (full)/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under)*).



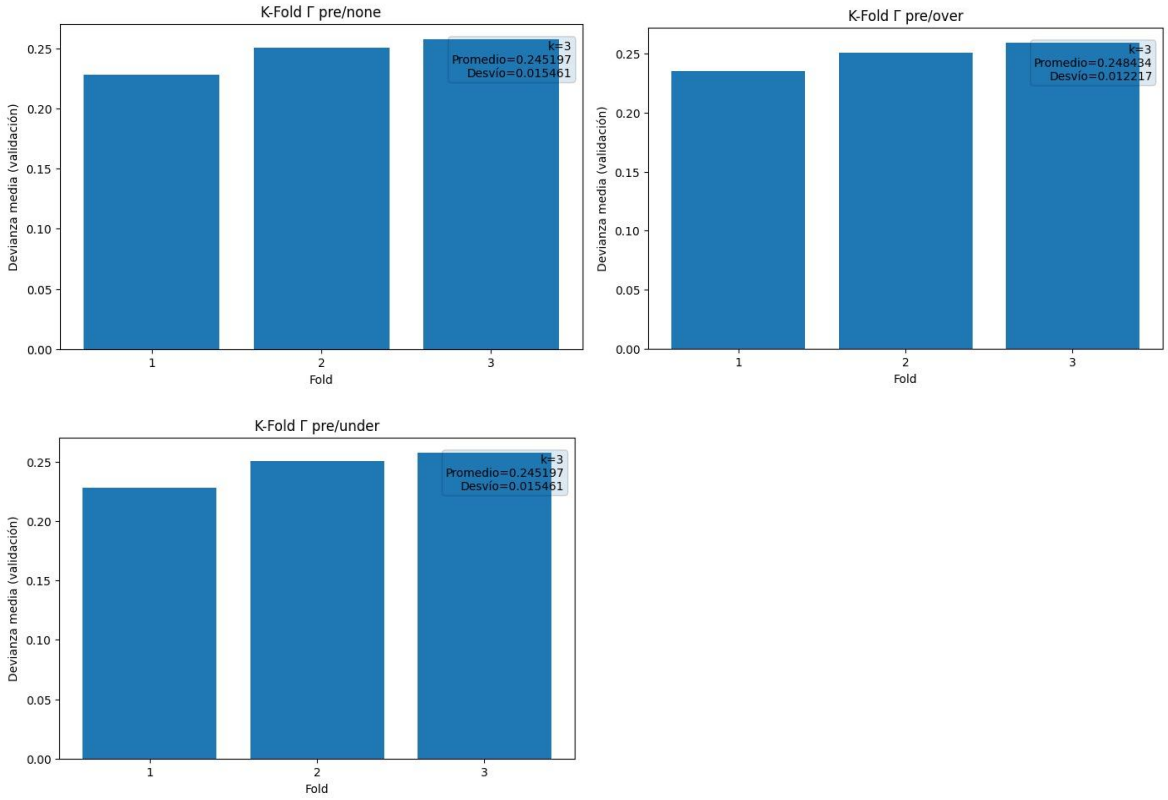
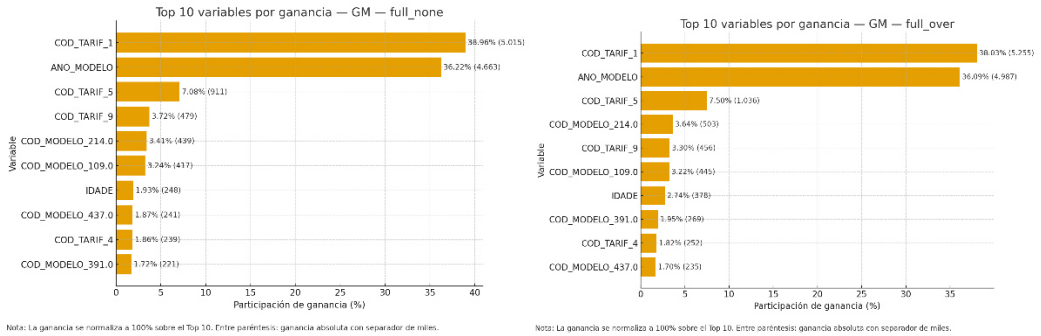


Figura 5: Validación cruzada Modelo Severidad (K = 3 folds con semillas fijas; métrica: devianza media Gamma por observación)

La variabilidad entre pliegues es baja, por lo que el orden de mérito es robusto. Cabe precisar que en la Tabla 8 se reporta devianza total, mientras que en la validación cruzada se reporta devianza media por observación, de modo que las magnitudes no son comparables, aunque sí lo es el ranking. Dado el empate práctico entre el árbol completo y la pre-poda sin remuestreo, la decisión se inclina por pre-poda sin remuestreo por parsimonia y mantenibilidad.

Importancia de Variables

En todos los escenarios de modelado se evaluó la relevancia de las variables predictoras (relevancia calculada típicamente por la reducción de devianza).



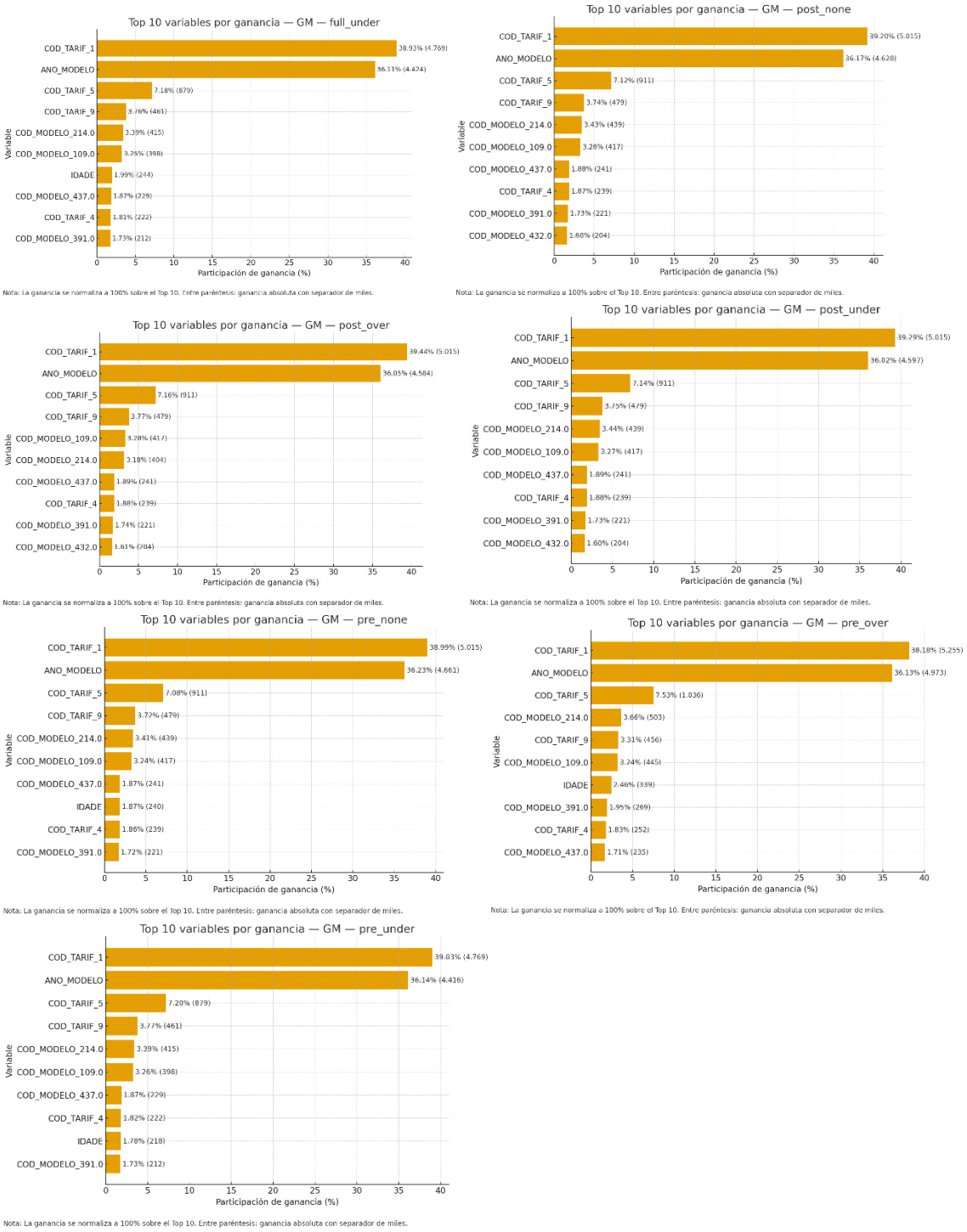


Figura 6: Importancia de Variables Modelo Severidad (criterio: reducción de devianza Gamma; top 10 predictores)

La relevancia es estable: COD_TARIF_3 y ANO_MODELO concentran aproximadamente 72–77% entre los predictores más relevantes, seguidos por COD_TARIF_1, COD_TARIF_5/9 y varias COD_MODELO*. Las variables de edad y las variables indicadoras poco frecuentes aparecen con importancia marginal. En términos actuariales, tipo de vehículo y año modelo explican la mayor parte del costo medio del siniestro, lo que sugiere posibilidad de simplificación sin pérdida material.

Se recomienda pre-poda sin remuestreo por precisión, estabilidad y mantenibilidad; el árbol completo sin remuestreo se conserva como modelo de referencia. El sesgo en cola puede abordarse posteriormente en la etapa de tarificación mediante análisis O/E por deciles, recargos/márgenes por percentiles altos o pruebas complementarias (p. ej., Tweedie, regresión cuantílica p50–p95 o mixturas Gamma–Pareto).

4.3. MODELO DE FRECUENCIA

Conforme al marco teórico, la prima pura por unidad de exposición se descompone en frecuencia y severidad. En esta sección se modela la frecuencia, entendida como la tasa esperada de ocurrencia de siniestros por unidad de exposición (vehículo-año). Sea N_{it} el número de siniestros observados en la póliza–periodo (i, t) y sea $e_{it} > 0$ su exposición. Entonces, para un perfil de covariables x, el objetivo es estimar la tasa

$$\lambda(x) = \frac{\mathbb{E}[N_{it} \mid X_{it} = x, e_{it}]}{e_{it}} \tag{76}$$

Aquí x denota el vector de variables de segmentación del árbol (covariables de riesgo) y, cuando se requiera, puede incluir $\log(e_{it})$ para capturar heterogeneidad residual; en todos los casos, la exposición se incorpora además mediante el offset $\log(e_{it})$, de modo que el árbol estima una tasa por unidad de exposición de modo que

$$\mathbb{E}[N_{it} \mid X_{it} = x, e_{it}] \approx e_{it}\, \lambda(x) \tag{77}$$

Operativamente, esto equivale a trabajar con una tabla de experiencia por segmentos: cada hoja del árbol define una región R_m(celda) en el espacio de covariables, dentro de la cual la tasa $\lambda(x)$se aproxima por un valor constante λ_m. En consecuencia, en cada hoja se acumulan la exposición y el conteo:

$$E(R_m) = \sum_{(i,t)\in R_m} e_{it}\,, N(R_m) = \sum_{(i,t)\in R_m} N_{it} \tag{78}$$

y se estima una tasa constante por hoja, por ejemplo,

$$\hat{\lambda}_m = \frac{N(R_m)}{E(R_m)} \tag{79}$$

En la implementación del modelo, e_{it}se incorpora mediante un ***offset*** $\log(e_{it})$, lo que asegura que el árbol estime una tasa por unidad de exposición y no un conteo absoluto.

Modelo: árbol de decisión con devianza Binomial Negativa y offset

Se estima un árbol de regresión para conteos usando devianza Binomial Negativa (NB), incorporando la exposición mediante un offset $\log(e_{it})$. Bajo esta especificación, el árbol aproxima una función por tramos para la tasa:

$$\hat{\lambda}(x) = \sum_{m=1}^{M} \hat{\lambda}_m \; 1\{x \in R_m\}, \text{y por tanto } \mathbb{E}[N \mid \widehat{X = x}, e] = e\,\hat{\lambda}(x) \qquad (80)$$

Desde el punto de vista actuarial, cada hoja R_m es una celda final de la tabla de experiencia para frecuencia: dentro de esa celda se acumula exposición $E(R_m)$ y conteo $N(R_m)$, y la tasa estimada $\hat{\lambda}_m$ resume el riesgo de ocurrencia por unidad de exposición.

Se comparan tres configuraciones del árbol — pre-poda, árbol completo (sin poda) y post-poda— combinadas con tres estrategias de remuestreo del entrenamiento: sin remuestreo, sobremuestreo y submuestreo

Sobredispersión de la frecuencia

El primer paso consistió en caracterizar la variable de frecuencia de siniestros en el conjunto de entrenamiento (N = 2 712 606). Se obtuvo un cociente varianza-media de $\hat{\phi}_P = 1{,}172$, lo que indica una sobredispersión ligera (≈ 17,2 % por encima de la equidispersión Poisson). Al estimar el parámetro de dispersión de la Binomial Negativa por momentos, se obtuvo $\hat{k} = 0{,}0788$.

Luego, se entrenaron dos árboles de regresión idénticos —mismos hiperparámetros y partición pseudoaleatoria reproducible mediante semillas fijas—, optimizados uno con devianza Poisson y otro con devianza NB. En el conjunto de prueba, el árbol NB registró una devianza de 46 957 frente a 80 280 de la Poisson, consiguiendo una mejora adicional de alrededor del 41,6 %.

Por tanto, se opta por desplegar el árbol de regresión con función de pérdida de Binomial Negativa. Esta decisión maximiza la precisión en la estimación de la frecuencia esperada, evita la infraestimación de la incertidumbre y refuerza la solidez metodológica de la tesis.

Métricas principales

Después de construir el modelo de frecuencia con devianza Binomial Negativa (NB), se observan las siguientes métricas principales:

Tabla 9: Métricas principales Modelo Frecuencia (métricas: Devianza Binomial Negativa total, Pseudo D², MAE, RMSE; configuraciones: pre/completo/post × sin remuestreo/sobremuestreo/submuestreo)

Modelo	Remuestreo	Devianza NB Total	Pseudo D² NB Offset	MAE	RMSE
Árbol completo	Sin remuestreo	65,771	17.25%	0.02267	0.12437
Árbol completo	Sobremuestreo	65,851	17.15%	0.02246	0.12436
Árbol completo	Submuestreo	68,019	14.42%	0.02871	0.12760
Post-poda	Sin remuestreo	59,665	24.93%	0.02406	0.12374
Post-poda	Sobremuestreo	59,784	24.78%	0.02389	0.12371
Post-poda	Submuestreo	61,919	22.09%	0.03188	0.12629
Pre-poda	Sin remuestreo	59,821	24.73%	0.02384	0.12345
Pre-poda	Sobremuestreo	60,108	24.37%	0.02413	0.12345
Pre-poda	Submuestreo	62,228	21.70%	0.03253	0.12552

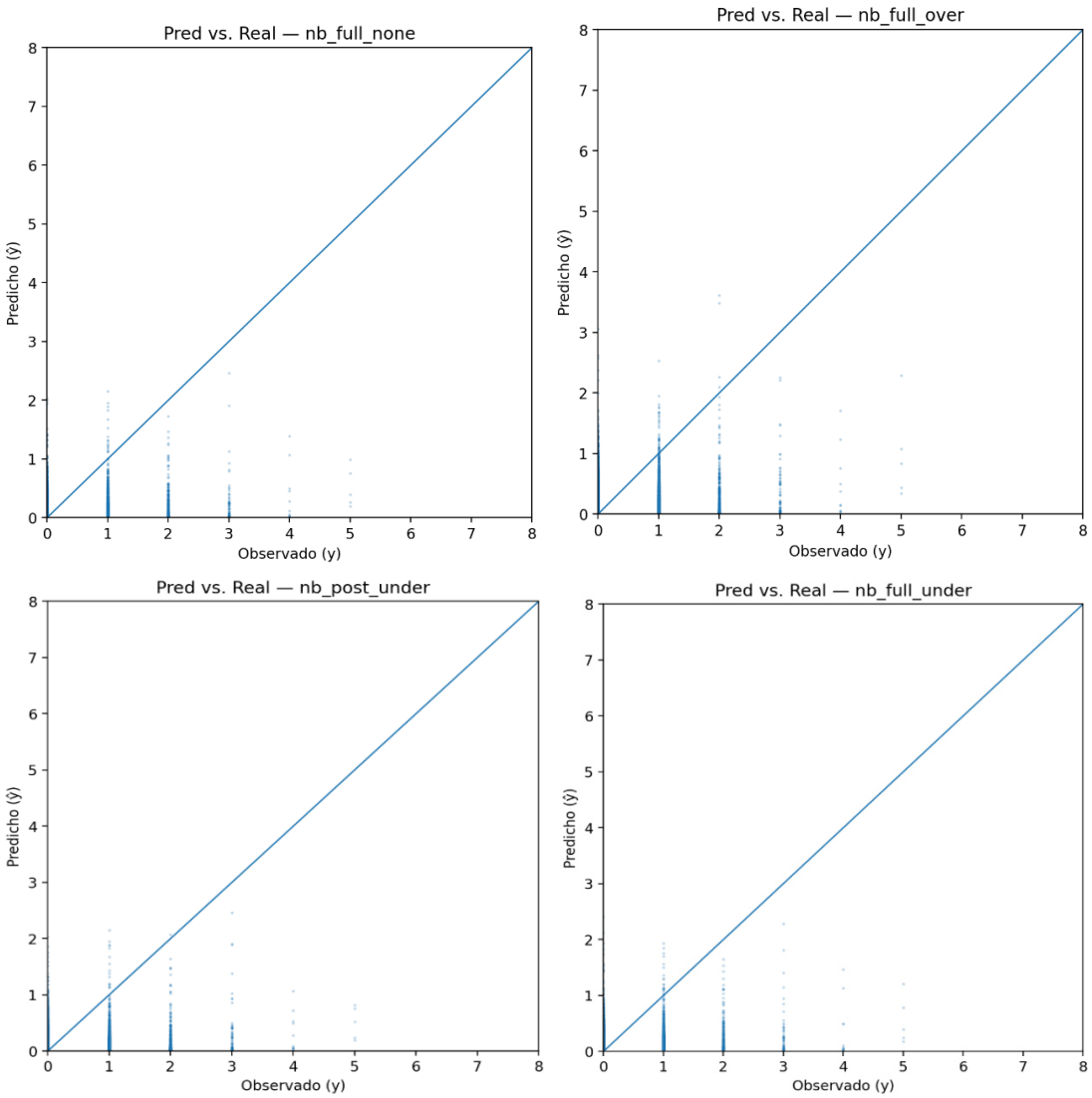
Nota: En esta tabla se reporta la devianza total $D_{\text{tot}} = \sum_{i=1}^{n} d_i$ (suma sobre las $n$observaciones del modelo de frecuencia). En las figuras y en la validación cruzada se reporta la devianza media por observación, $\bar{D} = D_{\text{tot}}/n$, para facilitar la comparación entre pliegues. Ambas medidas difieren solo por el factor $1/n$ cuando $n$es fijo.

La Pre-poda sin remuestreo ofrece el mejor equilibrio global (devianza, RMSE e interpretabilidad). La post-poda sin remuestreo puede registrar una devianza algo menor en una partición puntual, pero no sostiene ventaja en validación cruzada y puede comprimir medias de hoja. El submuestreo deteriora devianza y errores; el sobremuestreo produce cambios marginales sin mejorar la métrica global. Pseudo-D² en torno a 24–25 % es alto dada la masa en cero, confirmando la idoneidad de la NB.

Cabe precisar, en modelos de conteo basados en verosimilitud y con offset, los pseudo-R² "tipo devianza" suelen ser inferiores a los R² de OLS y no son comparables (Cameron & Trivedi, 2013; Hilbe, 2011). Como regla de orientación —derivada de la literatura de elección discreta para el índice de McFadden— valores en el entorno 0.2–0.4 se consideran muy buenos (McFadden, 1979; Train, 2009). Aunque aquí usamos el análogo por devianza, la guía cualitativa es útil: 24–25 % (Post-Sin remuestreo/Pre-Sin remuestreo) es razonablemente alto dada la alta masa en cero.

Predichos vs. Reales

En los diagramas de dispersión “Predichos vs Reales” se aprecia la relación entre los valores reales en el eje X y los valores predichos en el eje Y. La línea punteada representa la referencia “y = x” (predicción ideal).



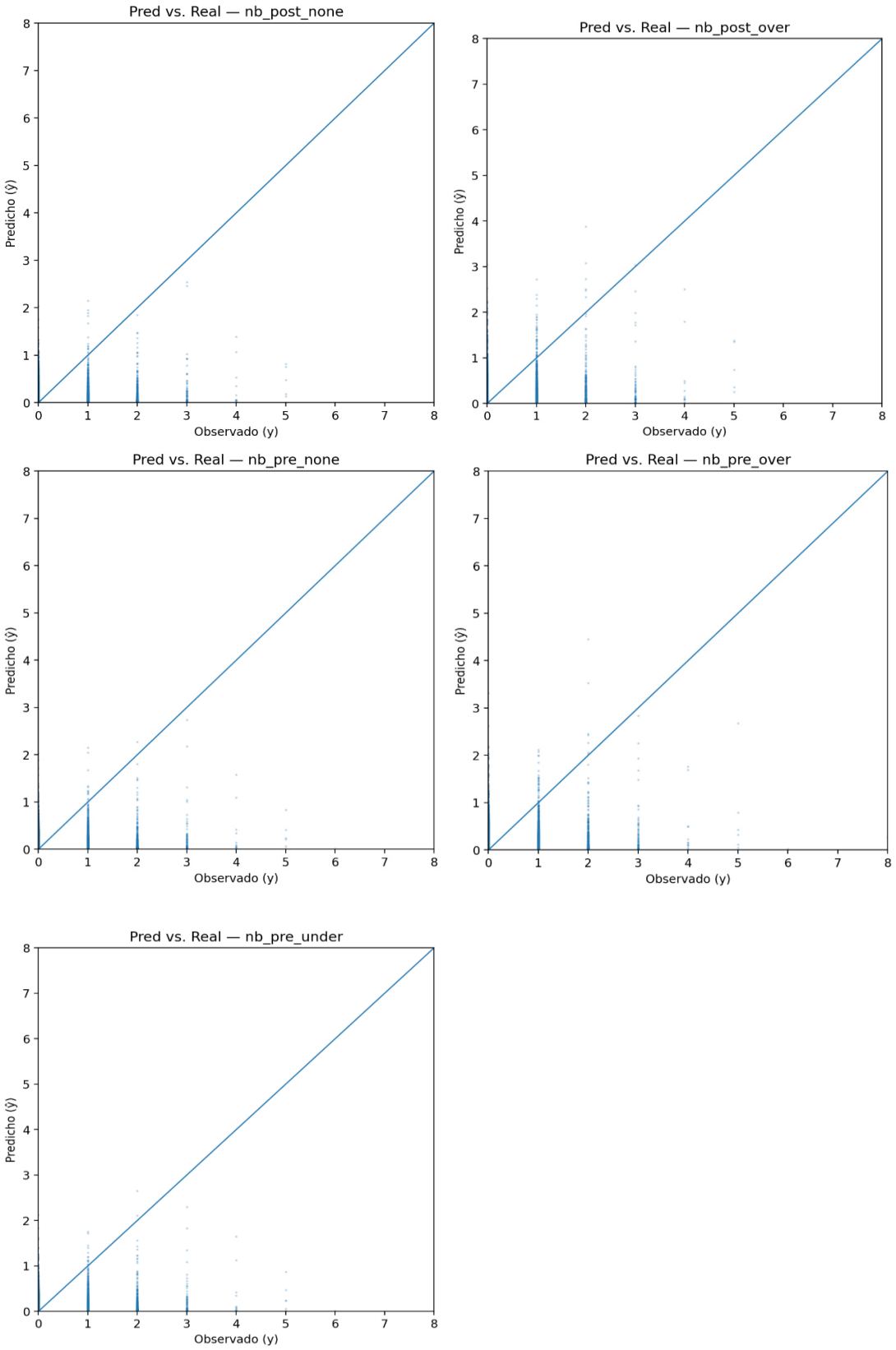
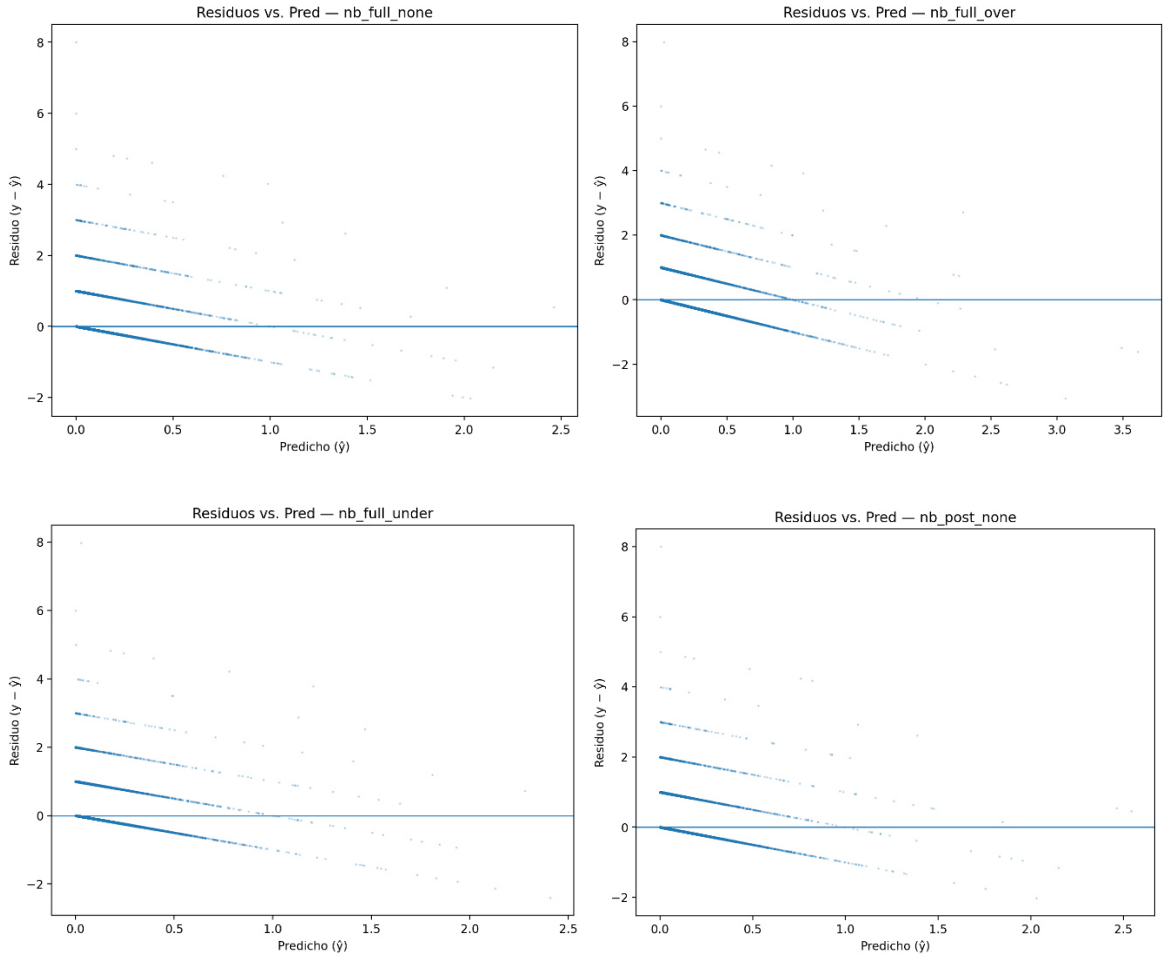


Figura 7: Predichos vs Reales Modelo Frecuencia (árbol Binomial Negativa; paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under); línea de referencia y = x)

La nube se concentra en $y \in \{0,1,2\}$ con mucha masa en cero. Se observa sobrepredicción de ceros (puntos por encima de $y = x$ en $y = 0$) y subpredicción en $y \geq 2$ (puntos por debajo de $y = x$), lo que produce encogimiento hacia la media. Aunque hay reales hasta 7–8, ŷ rara vez supera 2–3 (en sobremuestreo, hasta ~3–4,5). El sobremuestreo amplía el rango en $y \geq 2$ a costa de empeorar en $y = 0$; el submuestreo reduce $\hat{y}$ y acentúa la subpredicción en conteos medios/altos.

Residuos vs Predichos

Las gráficas de “Residuos vs. Predichos” (residuo = real − predicho) ayudan a evaluar la distribución de los errores:



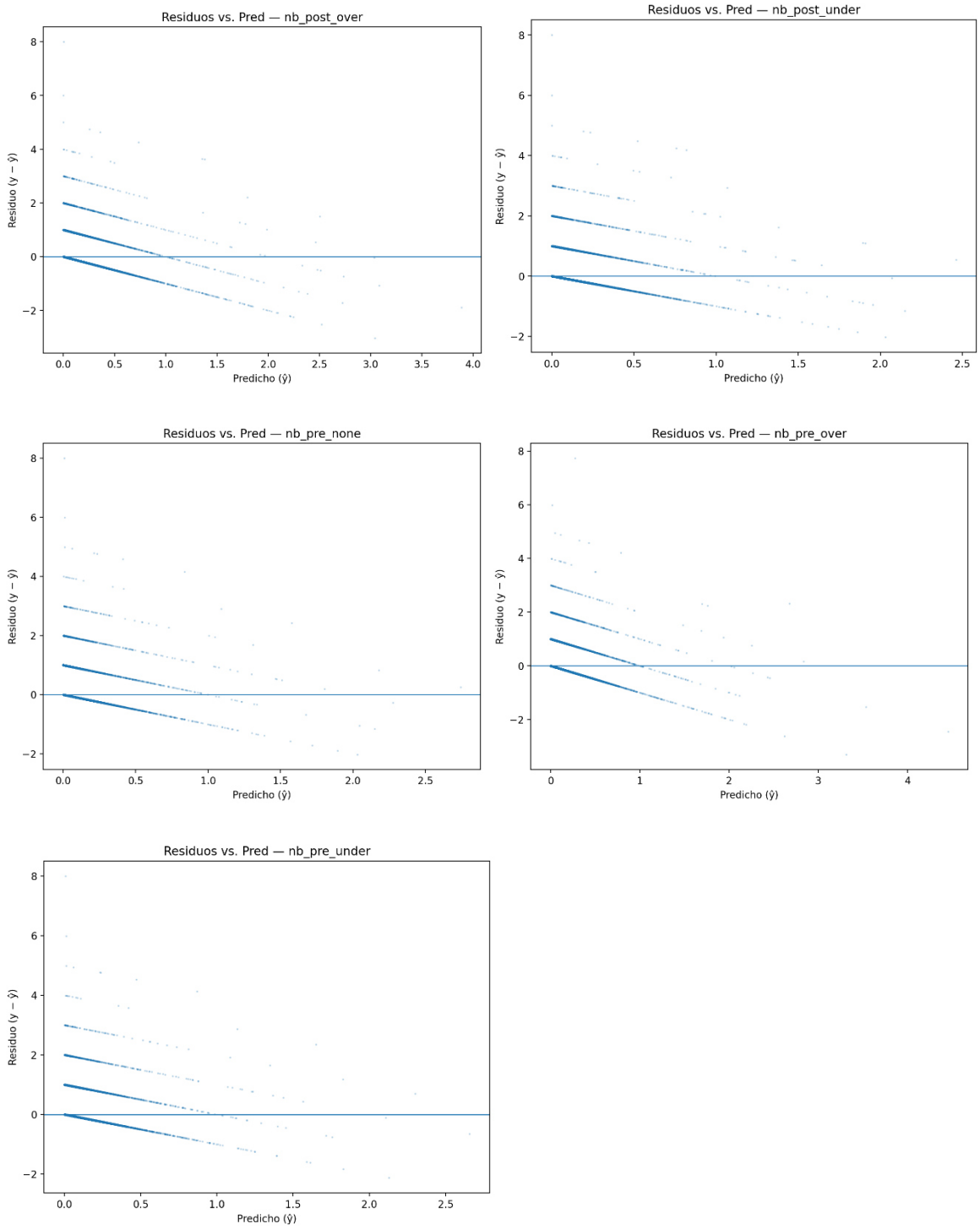
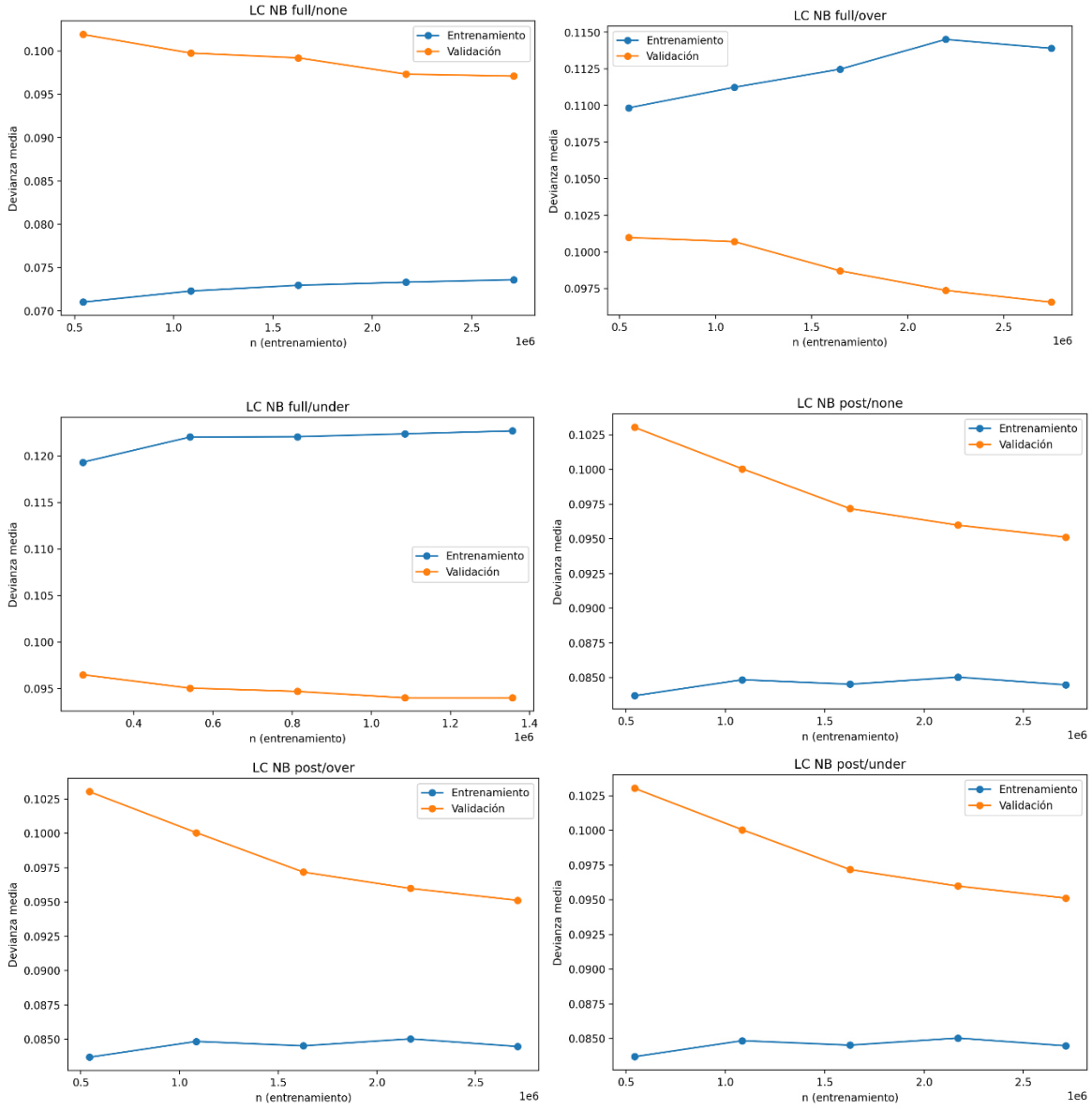


Figura 8: Residuos vs Predichos Modelo Frecuencia (árbol Binomial Negativa; paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under); residuo = y − ŷ)

Aparecen bandas de pendiente −1 por cada entero de y, con la de $y = 0$ como la más densa. El suavizado permanece próximo a cero en $\hat{y}$ bajos y desciende al aumentar $\hat{y}$, sobre todo en sobremuestreo, lo que indica sobreestimación en riesgos de media alta. Los outliers positivos se dan cuando $\hat{y}$ es pequeño e y es grande; lo cual es consistente con la geometría del residuo crudo en conteos y con la baja probabilidad de valores altos.

Curvas de aprendizaje

Cada gráfico reporta la devianza media (más bajo es mejor) de entrenamiento (línea azul) y validación/prueba (línea naranja) a medida que aumenta el tamaño del conjunto de entrenamiento (*n*). Se comparan tres familias de árboles de decisión —pre-poda (pre), árbol completo (full) y post-poda (post)— bajo tres esquemas de remuestreo (none, over, under).



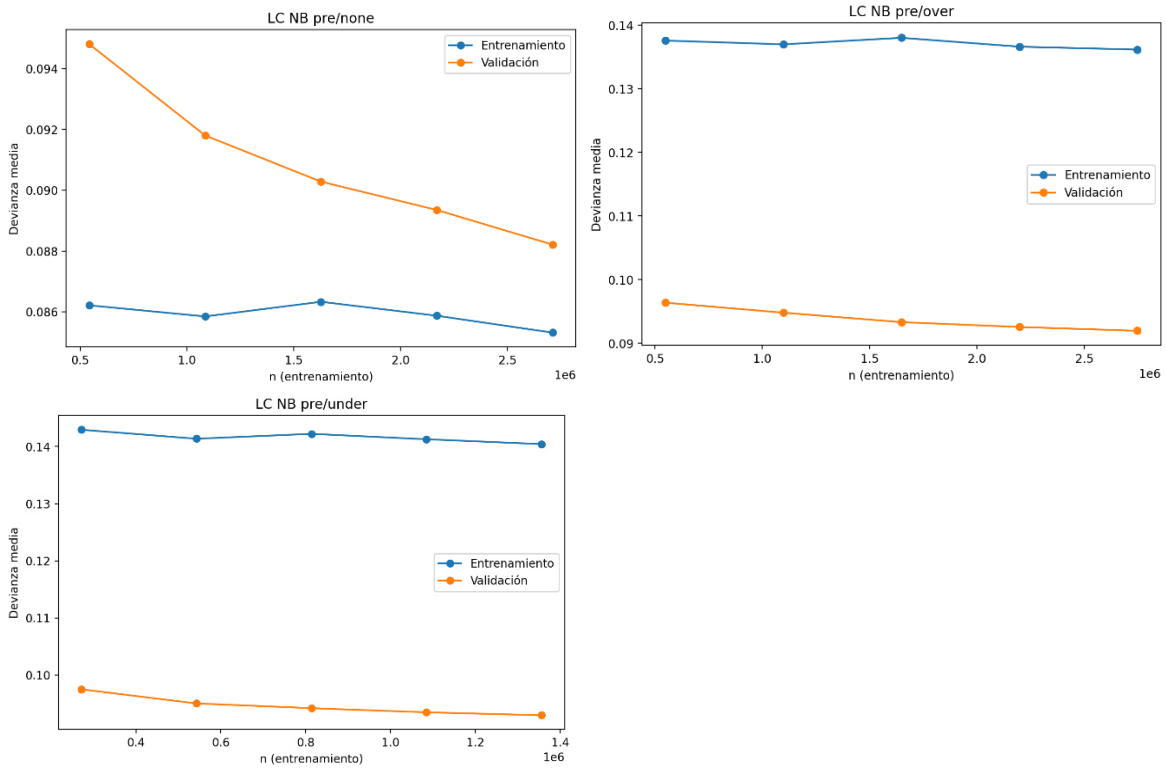
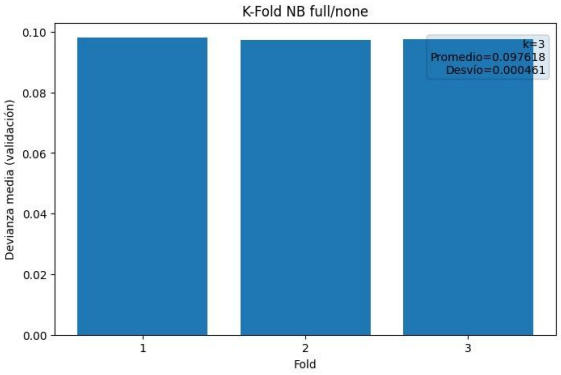


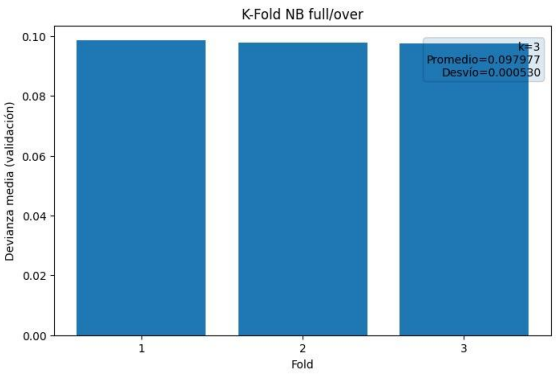
Figura 9: Curvas de aprendizaje Modelo Frecuencia (devianza media Binomial Negativa vs tamaño muestral; entrenamiento (azul) y validación/prueba (naranja); paneles: pre/completo/post × sin remuestreo (none)/sobremuestreo (over)/submuestreo (under))

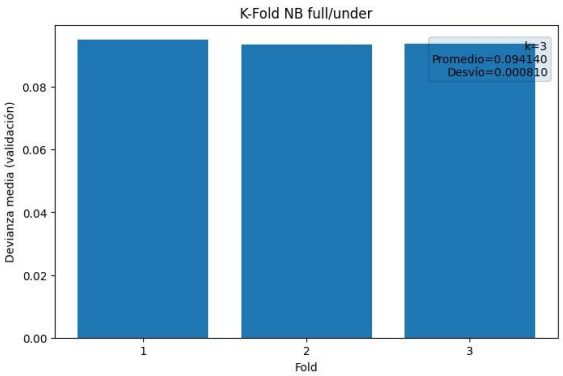
La devianza de validación disminuye con el tamaño muestral y no hay evidencia de sobreajuste severo. En sobremuestreo y submuestreo, puede ocurrir que la devianza de entrenamiento supere a la de validación por cambio de mezcla: el entrenamiento es artificialmente modificado mientras la validación mantiene su alta masa en cero. En el esquema sin remuestreo se observa el patrón clásico de brecha decreciente.

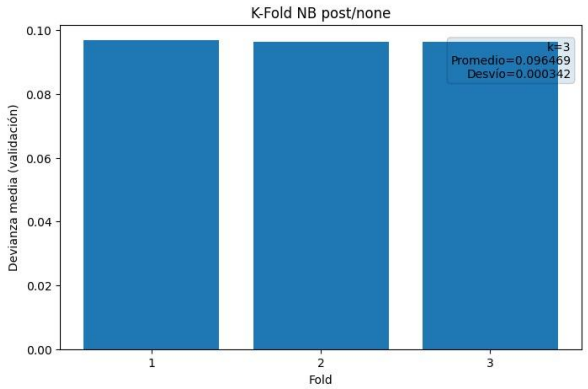
Validación cruzada

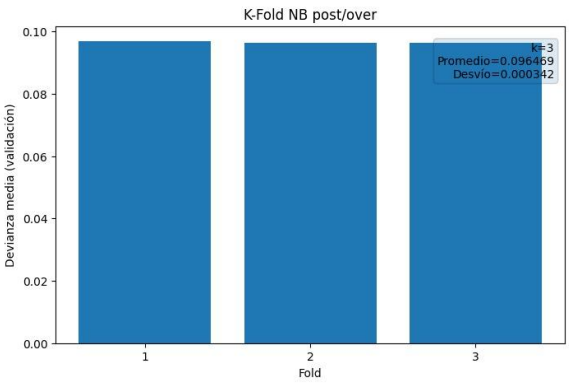
Para comparar rigurosamente el desempeño y la estabilidad de los nueve modelos propuestos, se aplicó validación cruzada K-Fold con K = 3 (tres cortes). En cada fold se mantuvo la misma partición y semilla para asegurar comparabilidad entre configuraciones (*pre/completo (full)/post × sin remuestreo (none)/ sobremuestreo (over)/ submuestreo (under)*).

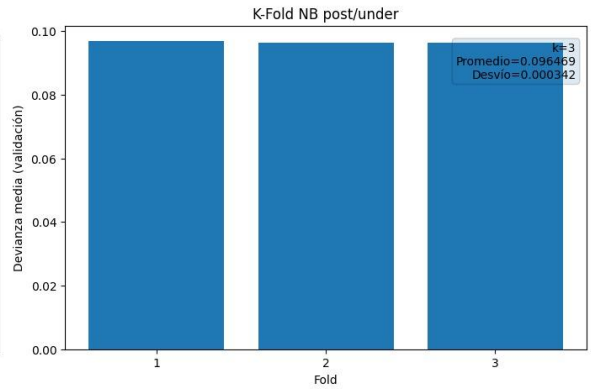

K-Fold NB full/none
k=3
Promedio=0.097618
Desvío=0.000461
Devianza media (validación)
Fold

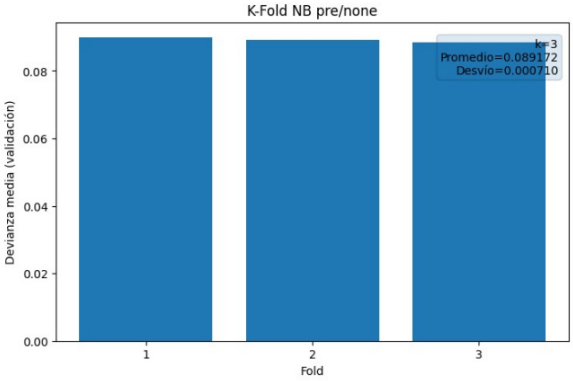

K-Fold NB full/over
k=3
Promedio=0.097977
Desvío=0.000530
Devianza media (validación)
Fold

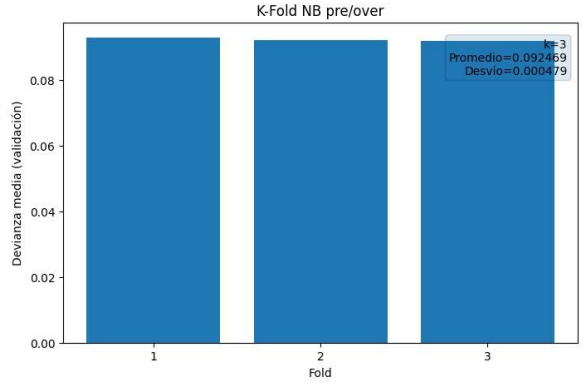

K-Fold NB full/under
k=3
Promedio=0.094140
Desvío=0.000810
Devianza media (validación)
Fold


K-Fold NB post/none
k=3
Promedio=0.096469
Desvío=0.000342
Devianza media (validación)
Fold


K-Fold NB post/over
k=3
Promedio=0.096469
Desvío=0.000342
Devianza media (validación)
Fold


K-Fold NB post/under
k=3
Promedio=0.096469
Desvío=0.000342
Devianza media (validación)
Fold


K-Fold NB pre/none
k=3
Promedio=0.089172
Desvío=0.000710
Devianza media (validación)
Fold


K-Fold NB pre/over
k=3
Promedio=0.092469
Desvío=0.000479
Devianza media (validación)
Fold

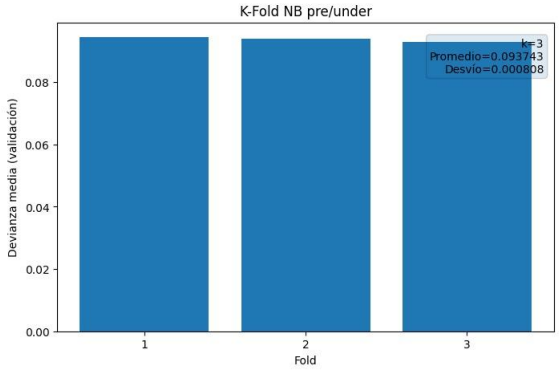
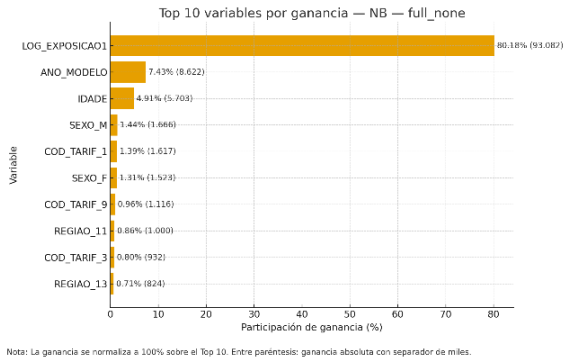


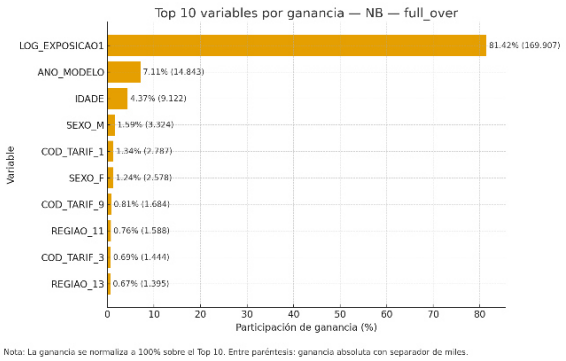
Figura 10: Validación cruzada Modelo Frecuencia (K = 3 folds con semillas fijas; métrica: devianza media Binomial Negativa por observación)

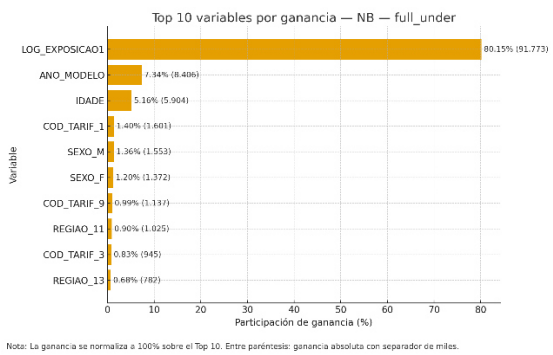
La variabilidad entre pliegues es baja, lo que robustece el orden de mérito. La política sin remuestreo se ubica consistentemente entre las mejores devianzas; el submuestreo empeora y el sobremuestreo no muestra mejoras robustas. La ventaja marginal de post-poda sin remuestreo en un conjunto puntual no se sostiene al promediar pliegues.

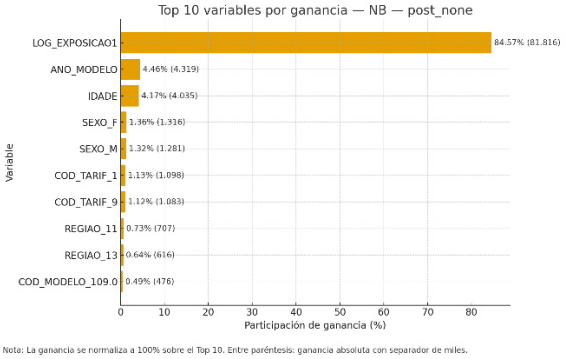
Importancia de Variables

En todos los escenarios de modelado se evaluó la importancia de las variables predictoras (importancia calculada típicamente por la reducción de devianza atribuible a cada variable en el árbol). Los resultados son coherentes a través de las distintas configuraciones:









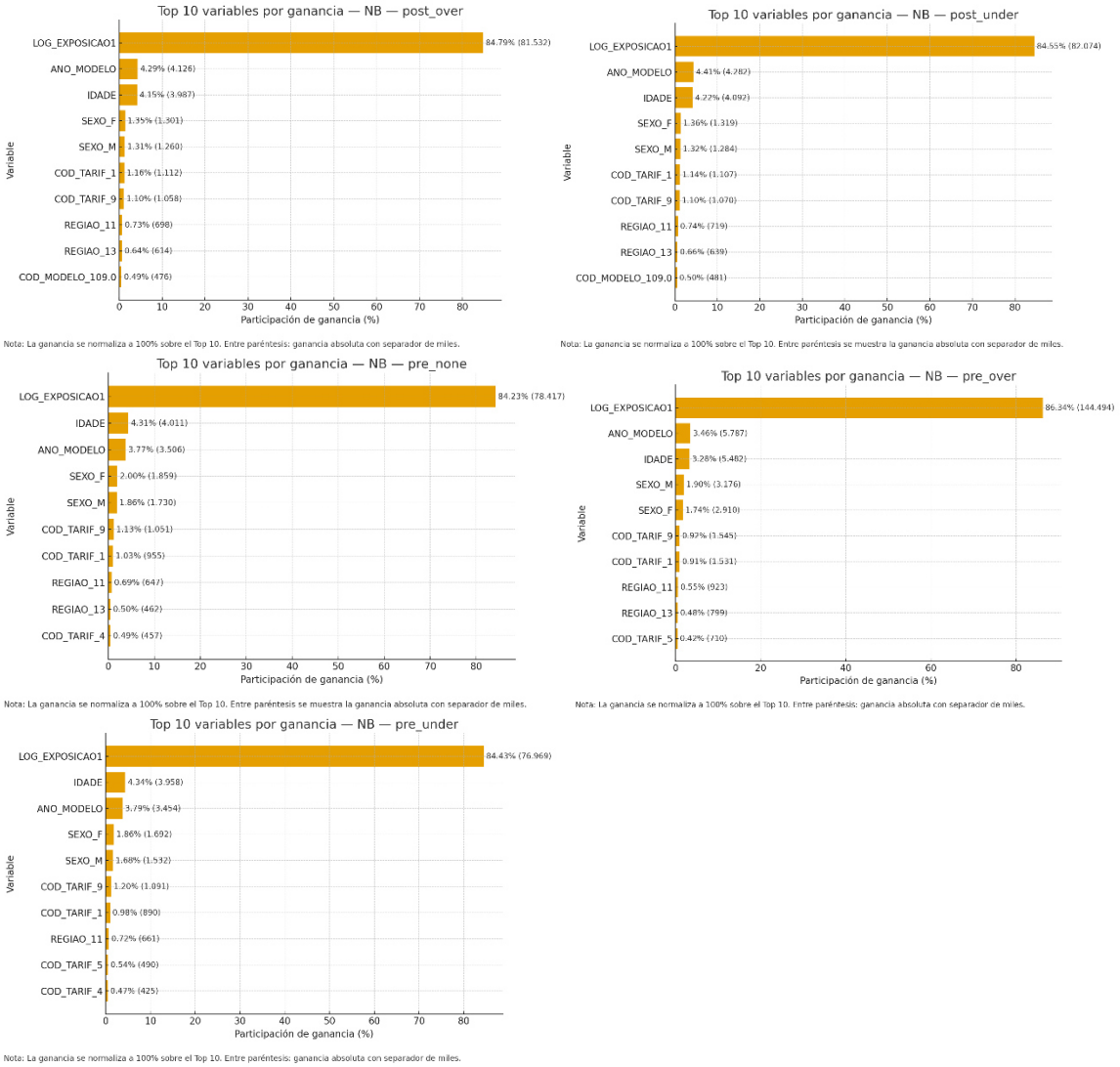


Figura 11: Importancia de Variables Modelo Frecuencia (criterio: reducción de devianza Binomial Negativa; top 10 predictores)

La variable LOG_EXPOSICAO1 domina (80–86% entre las más relevantes), lo que sugiere la presencia de heterogeneidad residual asociada a exposiciones parciales o agregadas. Esta variable actúa como predictor auxiliar y no sustituye el uso de EXPOSICAO1 como offset en el modelo de frecuencia. Le siguen ANO_MODELO e IDADE (3–5% cada una), y luego SEXO y algunas categorías de COD_TARIF/REGIAO. La relevancia de LOG_EXPOSICAO1 es consistente con su función auxiliar para capturar heterogeneidad asociada a la exposición, manteniendo EXPOSICAO1 como offset actuarial del modelo de frecuencia.

Se adopta pre-poda sin remuestreo por equilibrio entre desempeño, estabilidad e interpretabilidad, y se recomienda evitar el submuestreo. El sobremuestreo puede reservarse para objetivos secundarios (mayor sensibilidad en conteos 2–3), con la salvedad de que no mejora la métrica global. La robustez del modelo se complementará

con análisis O/E (observado/esperado) por deciles de riesgo y seguimiento específico de la distribución de ceros.

4.4. MODELO PRIMA PURA

Conforme al marco teórico, la prima pura o prima de riesgo se define como la pérdida esperada por unidad de exposición, y en términos agregados por segmento g se expresa como:

$$\hat{\pi}_g = \frac{L_g}{E_g} = \left(\frac{N_g}{E_g}\right)\left(\frac{L_g}{N_g}\right) \tag{81}$$

donde E_g es la exposición total, N_g el número total de siniestros y L_g la pérdida total del segmento. Esta identidad muestra que la prima pura se obtiene multiplicando la frecuencia (tasa de siniestros por unidad de exposición) por la severidad media (costo medio condicionado a ocurrencia).

En las secciones previas se estimaron, con árboles de decisión no paramétricos, los dos componentes:

- **Frecuencia:** modelo Binomial Negativa con offset $\log(e)$, que estima $\hat{\lambda}(x)$ y por tanto $\mathbb{E}[\widehat{N \mid x}, e] = e\,\hat{\lambda}(x)$.
- **Severidad:** modelo Gamma sobre severidades positivas $Y_{it} = L_{it}/N_{it}$ (definidas cuando $N_{it} > 0$), que estima $\mu(x) = \mathbb{E}[Y_{it} \mid N_{it} > 0, X = x]$.

Dado que los árboles inducen particiones en regiones disjuntas (hojas) $\{R_m\}$, cada hoja actúa como una celda final de la tabla de experiencia. En cada hoja R_m, la prima pura estimada queda constante por tramos y se obtiene como:

$$\hat{\pi}(x) = \hat{\lambda}(x)\,\hat{\mu}(x) = \sum_{m=1}^{M} (\hat{\lambda}_m\,\hat{\mu}_m)\,1\{x \in R_m\} \tag{82}$$

Por tanto, el modelo de prima pura preserva la lógica actuarial: el árbol define segmentos (celdas) y dentro de cada uno se estiman tasas y costos medios que, al multiplicarse, producen la pérdida esperada por unidad de exposición.

Evaluación agregada por segmentos: WAPE calibrado (Error porcentual absoluto ponderado)

Para evaluar el desempeño a nivel agregado, se utiliza el WAPE calibrado, calculado sobre totales por segmento. La evaluación se realiza comparando, para cada segmento g, el total observado de pérdida L_g frente al total esperado $\hat{L}_g$ derivado del modelo (integrando frecuencia y severidad), y luego aplicando un factor de calibración global c_{cal} que re-ancle el nivel total de prima (neutralidad de tarifa). Así, el WAPE calibrado se interpreta principalmente como una medida de distribución/relatividades entre segmentos, no de escala.

Tabla 10: Resultados por Modelo Prima Pura

Modelo	Remuestreo	PP Media (Sin Calibrar) (R$/póliza–año (incluye e_i))	WAPE calibrado
Árbol completo	Sin remuestreo	465	28.68%
Árbol completo	Sobremuestreo	790	28.53%
Árbol completo	Submuestreo	477	28.69%
Post-poda	Sin remuestreo	559	27.81%
Post-poda	Sobremuestreo	998	28.30%
Post-poda	Submuestreo	567	28.08%
Pre-poda	Sin remuestreo	538	27.61%
Pre-poda	Sobremuestreo	995	28.45%
Pre-poda	Submuestreo	554	27.69%

Nota de tabla: El WAPE se computa después de aplicar el factor de calibración c_{cal}, calculado sobre los totales por segmento. La segmentación de agregación utilizada para los totales (segmentos g de evaluación) es la misma en los nueve modelos, garantizando comparabilidad del WAPE. Nótese que las hojas del árbol (R_m) pueden diferir entre configuraciones (pre/completo/post y remuestreo). La 'PP media' se informa previa a la calibración para evidenciar desplazamientos de nivel entre configuraciones.

Interpretación de resultados

La mejor configuración es pre-poda sin remuestreo, con WAPE = 27.61%. Le siguen pre-poda con submuestreo (27.69%) y post-poda sin remuestreo (27.81%). Las tres variantes completo (full) quedan rezagadas (28.53–28.69%), lo que sugiere que una ligera ventaja en severidad en ciertos casos no compensa un peor desempeño relativo en frecuencia al integrarse en prima pura.

El sobremuestreo incrementa fuertemente el nivel de PP media (≈ +70–85% sin calibración) sin traducirse en mejoras del WAPE tras calibrar. Esto es coherente con que, una vez aplicada la calibración global c_{cal}, el WAPE mide principalmente la capacidad del modelo para asignar correctamente la pérdida entre segmentos (relatividades), y no el nivel total. El submuestreo también desplaza el nivel, pero no entrega mejoras consistentes en precisión agregada.

Cabe precisar que, al ser un producto, $\hat{\pi}(x) = \hat{\lambda}(x)\,\hat{\mu}(x)$, cualquier desplazamiento de nivel o sesgo en frecuencia (por ejemplo, elasticidad de exposición distinta de 1 o compresión por poda) y cualquier sesgo en severidad (particularmente en cola alta) tienden a amplificarse al multiplicarse, deteriorando el desempeño agregado incluso si una de las componentes es competitiva por separado.

Recomendación operativa

Se recomienda pre-poda sin remuestreo por su mejor desempeño agregado (menor WAPE calibrado) y parsimonia. Como alternativas cercanas, pre-poda con submuestreo y post-poda sin remuestreo pueden considerarse si se priorizan matices específicos. Debe evitarse el sobremuestreo, dada su tendencia a distorsionar el nivel sin mejorar las relatividades (WAPE).

Asimismo, dada la subestimación persistente en cola alta observada en severidad, resulta prudente complementar el despliegue con controles y ajustes operativos: análisis O/E por deciles (o percentiles) de prima pura, recargos por perfiles extremos y, si se requiere mayor robustez en colas, evaluar extensiones como Tweedie o mixturas Gamma–Pareto para extremos.

En síntesis, el Capítulo IV replica la lógica de la tabla de experiencia: el árbol define celdas (hojas) y, dentro de cada celda, se estiman E, N y Lmediante $\hat{\lambda}$ y $\hat{\mu}$, de manera que la prima pura resulta consistente con $L/E = (N/E)(L/N)$.

CAPÍTULO V. RESULTADOS

El análisis empírico se estructuró en tres componentes coherentes con la descomposición actuarial de la prima pura expuesta en el Capítulo II: (i) Severidad $\mu(x) = \mathbb{E}[Y \mid Y > 0, X = x]$, (ii) Frecuencia $\lambda(x)$, definida como la tasa esperada por unidad de exposición: $\lambda(x) = \frac{\mathbb{E}[N_{it} | X_{it}=x, e_{it}]}{e_{it}}$ de modo que $\mathbb{E}[N_{it} \mid X_{it} = x, e_{it}] = e_{it}\lambda(x)$, incorporando la exposición mediante offset $\log(e_{it})$. Operativamente, en cada hoja R_m se estima $\hat{\lambda}_m = N(R_m)/E(R_m)$ y (iii) la integración multiplicativa $\pi(x) = \lambda(x)\mu(x)$ para obtener la prima pura por unidad de exposición. En términos operativos, cada árbol induce una partición del espacio de covariables en regiones disjuntas (hojas) que actúan como celdas finales de la tabla de experiencia: dentro de cada hoja se sintetizan las magnitudes (E, N, L) y se aproximan por constantes por tramos las cantidades de interés (tasa de frecuencia, severidad media y prima pura).

Para cada componente se contrastaron configuraciones de complejidad del árbol —árbol completo, pre-poda y post-poda— combinadas con políticas de muestreo —sin remuestreo, sobremuestreo y submuestreo—. El desempeño se evaluó mediante devianza, RMSE, MAE y Pseudo-D^2; en la integración se empleó adicionalmente WAPE. Salvo indicación expresa, la devianza se reporta como media por observación, lo que facilita comparaciones entre pliegues y escenarios dentro de una misma familia; no obstante, sus niveles no son comparables entre familias (Gamma vs Binomial Negativa) debido a sus distintas funciones de enlace y definiciones de devianza. Las estimaciones se obtuvieron sobre un conjunto de validación independiente y mediante validación cruzada de tres pliegues con semillas fijas, garantizando replicabilidad.

Severidad (Gamma)

En severidad, el árbol completo sin remuestreo se adoptó como referencia inicial. Frente a este estándar, la pre-poda sin remuestreo alcanzó resultados prácticamente equivalentes, con diferencias menores a 0.2% en devianza, MAE y RMSE, magnitudes dentro de la variabilidad observada entre pliegues. Este resultado indica que aumentar complejidad estructural no genera mejoras robustas fuera de muestra, mientras que la pre-poda reduce varianza y favorece mantenibilidad. En contraste, la post-poda deterioró el ajuste en validación, coherente con la compresión de hojas (celdas) que reduce resolución para montos elevados, elevando devianza y RMSE.

Las políticas de remuestreo confirmaron este patrón: las variantes sin remuestreo fueron sistemáticamente superiores; el submuestreo incrementó la pérdida y los errores al eliminar

observaciones relevantes del cuerpo de la distribución, mientras que el sobremuestreo no aportó mejoras consistentes y, en algunos casos, incrementó variabilidad al replicar registros poco representativos. Los valores de Pseudo-D^2 cercanos a 64% para árbol completo y pre-poda sin remuestreo muestran una mejora sustantiva respecto del modelo nulo pese a asimetría y heterocedasticidad; en cambio, configuraciones con remuestreo o post-poda descienden hacia 61–62%, consistente con pérdida de ajuste en colas. Adicionalmente, el hecho de que RMSE exceda ampliamente a MAE confirma la influencia de colas pesadas: pocos errores muy grandes inflan el error cuadrático sin afectar en igual proporción el error absoluto medio.

La evidencia gráfica respalda estas conclusiones. En predicho vs observado se aprecia buena calibración en montos bajos y medios, junto con subestimación sistemática en tramos altos (puntos bajo la diagonal), acentuada con post-poda. Los residuos vs predichos exhiben el abanico característico de la Gamma (varianza creciente con la media) y bandas por hoja con medianas próximas a cero en el cuerpo; sin embargo, persisten residuos positivos grandes en la parte superior, indicio de extremos no plenamente capturados. Las curvas de aprendizaje muestran descenso sostenido de la devianza de validación con el tamaño muestral y reducción de la brecha entrenamiento–validación, sugiriendo mejora de generalización sin señales de sobreajuste severo. En tamaños muestrales grandes, árbol completo y pre-poda sin remuestreo se mantienen en el grupo superior, mientras que post-poda pierde rendimiento, especialmente combinada con sobremuestreo. La validación cruzada refuerza la robustez: los escenarios líderes se agrupan con desviaciones estándar acotadas y la pre-poda sin remuestreo resulta preferible por parsimonia.

El análisis de importancia de variables confirma que la tarifa/clase del vehículo y el año modelo explican la mayor parte de la reducción de devianza (≈ 75% en conjunto), lo que habilita simplificación del set de predictores sin pérdida material. Para preservar suficiencia en extremos, se recomienda complementar la implementación con controles operativos de cola (p. ej., O/E por deciles, recargos o márgenes en perfiles de alto riesgo) y, si la evidencia lo justifica, explorar familias Tweedie, mixturas Gamma–Pareto o enfoques cuantílicos en percentiles altos.

Frecuencia (Binomial Negativa con offset)

En frecuencia, la pre-poda sin remuestreo ofrece el mejor equilibrio global entre desempeño, estabilidad e interpretabilidad. La post-poda sin remuestreo puede registrar una devianza menor en una partición puntual, pero dicha ventaja no se sostiene al promediar pliegues y tiende a comprimir medias de hoja, afectando la diferenciación del

riesgo. El árbol completo no aporta ventajas adicionales y sí incrementa complejidad. El submuestreo deteriora sistemáticamente las métricas; el sobremuestreo produce mejoras marginales localizadas sin corregir la subpredicción en conteos altos.

La variable de respuesta exhibe alta masa en cero y pocos conteos elevados; en consecuencia, el modelo tiende a sobrepredecir ceros y subpredecir valores ≥ 2(encogimiento hacia la media), comportamiento esperable en problemas discretos dominados por ceros. El sobremuestreo amplía el rango predicho en conteos 2–3, pero a costa de empeorar calibración en cero; el submuestreo produce el efecto inverso. En residuos vs predichos aparecen las bandas asociadas a cada entero de conteo; el suavizado se mantiene cerca de cero para $\hat{y}$bajos y decrece conforme aumenta $\hat{y}$, señal de sobreestimación en riesgos de mayor expectativa, más marcada bajo sobremuestreo. Las curvas de aprendizaje son descendentes y estables; no se observan señales de sobreajuste severo y las variantes sin remuestreo presentan la convergencia más limpia. La validación cruzada confirma la jerarquía: los escenarios sin remuestreo se ubican consistentemente entre los mejores promedios, mientras que remuestreo y árboles más complejos se rezagan.

En importancia de variables, el logaritmo de la exposición —además del offset— resulta dominante (≈ 80–86%), seguido por año modelo y edad; sexo y variables territoriales presentan contribución secundaria. La persistencia de sobrepredicción en cero y subpredicción en conteos ≥ 3 sugiere, como extensión, evaluar modelos con exceso de ceros (inflacionados o tipo hurdle) o calibraciones por bandas para mejorar sensibilidad en extremos sin perder parsimonia.

Prima pura (integración Frecuencia × Severidad)

La integración de ambos submodelos permitió estimar la prima pura, $\hat{\pi}(x) = \hat{\lambda}(x)\,\hat{\mu}(x)$, consistente con la identidad actuarial agregada $L/E = (N/E)(L/N)$ sobre celdas/segmentos. Para la evaluación agregada se utilizó WAPE después de una calibración global de nivel que reancla la suma total de primas puras predichas al total observado sin alterar relatividades entre riesgos. En términos operativos, la prima pura por póliza–periodo se obtiene multiplicando la exposición e_i por la tasa de frecuencia esperada y por la severidad esperada. Con calibración global: $\widehat{PP}_i^{\,cal} = c_{cal}\, e_i\, \hat{\lambda}(x_i)\, \hat{\mu}(x_i)$. Este procedimiento separa el error de escala del error distributivo y garantiza neutralidad tarifaria a nivel agregado.

Los resultados confirman que la combinación pre-poda sin remuestreo alcanza el mejor WAPE, coherente con el equilibrio obtenido en los modelos individuales. Alternativas como

pre-poda con submuestreo y post-poda sin remuestreo exhiben desempeños cercanos pero inferiores. En cambio, las variantes con sobremuestreo elevan sustancialmente el nivel de prima pura media sin traducirse en reducciones apreciables del WAPE tras calibración, mientras que el submuestreo desplaza el nivel sin ganar precisión. Dado que la prima pura es un producto, cualquier desplazamiento de nivel en frecuencia o sesgo de cola en severidad tiende a amplificarse en el error agregado, por lo que la estrategia recomendada prioriza estabilidad y parsimonia antes que complejidad adicional.

CAPÍTULO VI. CONCLUSIONES

Los resultados evidencian que la arquitectura bifásica propuesta —modelo de Frecuencia con Binomial Negativa (incorporando la exposición mediante offset) y modelo de Severidad con Gamma— integrada multiplicativamente para estimar la prima pura y reanclada mediante un factor de calibración global c_{cal}, alcanza un desempeño consistente fuera de muestra. En coherencia con el enfoque actuarial del Capítulo II, la estimación reproduce la identidad de la tabla de experiencia: la prima pura se obtiene como $\pi(x) = \lambda(x)\mu(x)$, equivalente a $L/E = (N/E)(L/N)$ sobre celdas (hojas) del árbol. En este marco, la configuración pre-poda sin remuestreo emerge como preferente por su equilibrio entre precisión, estabilidad e interpretabilidad en severidad, frecuencia e integración. La calibración global c_{cal} garantiza neutralidad tarifaria agregada (coherencia entre prima total estimada y siniestralidad observada), evitando confundir mejoras distributivas con un simple reanclaje de nivel.

En términos de complejidad, se identifica un punto de “codo” a partir del cual expandir el árbol no produce ganancias robustas en métricas fuera de muestra. En Severidad, la pre-poda sin remuestreo y el árbol completo sin remuestreo muestran desempeño prácticamente equivalente en promedio, con diferencias dentro de la variabilidad entre pliegues; por tanto, el incremento de complejidad no se traduce en mejoras sostenidas. En Frecuencia, la post-poda puede exhibir ventajas puntuales en devianza, pero a costa de comprimir medias de hoja y perder sensibilidad en colas; en conjunto, no se observa una superioridad estable de la post-poda respecto de alternativas más parsimoniosas. Así, la pre-poda se consolida como la opción más trazable y mantenible, sin sacrificio material de desempeño.

Respecto del remuestreo, no se encuentra evidencia de mejoras persistentes. En Severidad, el submuestreo incrementa devianza y RMSE al remover información relevante del cuerpo; en Frecuencia, también deteriora métricas pese a la alta masa en cero. El sobremuestreo, por su parte, desplaza el nivel de prima pura no calibrada, pero no reduce de manera consistente el error distributivo tras calibración (WAPE). En consecuencia, el remuestreo no se considera una práctica recomendable en el presente contexto.

Aun cuando los modelos exhiben buen ajuste en la media y en el cuerpo de la distribución, persiste infraestimación en extremos: conteos altos (frecuencia) y montos severos muy elevados (severidad). Ni el aumento de complejidad ni las estrategias de remuestreo eliminan completamente este sesgo, lo que sugiere una limitación estructural del particionado por hojas para capturar colas pesadas. El sistema, una vez calibrado,

reproduce adecuadamente la siniestralidad agregada, pero requiere salvaguardas específicas para perfiles con potencial extremo a fin de preservar suficiencia técnica y estabilidad tarifaria en operación.

La validez externa queda condicionada al período y a la composición de la cartera analizada. La extrapolación a otros periodos o portafolios dependerá de la estabilidad de las relaciones subyacentes y de la ausencia de cambios estructurales (regulatorios, macroeconómicos o en patrones de siniestralidad). Para mitigar estos riesgos, resulta pertinente establecer monitoreo continuo de deriva de datos y desempeño, además de pruebas de estrés bajo escenarios alternativos (aumento de frecuencia, incremento de severidad en cola o cambios abruptos de exposición) para evaluar resiliencia del esquema de tarificación.

Se emplean dos esquemas de validación: (i) una partición 80/20 estratificada por la presencia de siniestros (indicador binario) para garantizar representatividad del evento; y (ii) validación cruzada K-fold (K=3) construida mediante barajado pseudoaleatorio reproducible con semilla fija, con estratificación en Frecuencia (NB) por categorías de conteo (0/1/2/≥3) para estabilizar la mezcla. Se considera devianza específica por familia (Binomial Negativa con offset para frecuencia; Gamma para severidad), pseudo-D^2 y, en la integración, WAPE de prima pura.

Respecto a las hipótesis, la hipótesis nula 01 (remuestreo: “no mejora”) no se rechaza. El submuestreo deteriora métricas de forma sistemática y el sobremuestreo no exhibe mejoras sostenidas en devianza ni pseudo-D^2, tampoco reduce el WAPE de prima pura y puede distorsionar el nivel agregado antes de la calibración. La hipótesis nula 02 (poda: “no mejora”) presenta evidencia heterogénea. La pre-poda aporta beneficios operativos y estadísticos: en frecuencia mejora el equilibrio entre ajuste y generalización, y en severidad mantiene desempeño equivalente al árbol completo con mayor parsimonia. En contraste, la post-poda no ofrece mejoras consistentes y puede degradar el desempeño al comprimir medias de hoja, afectando extremos.

En síntesis, para la hipótesis general (remuestreo + poda: “no mejora”) se concluye un rechazo parcial: las ganancias se explican principalmente por la poda (en particular pre-poda), mientras que el remuestreo no aporta mejoras robustas ni en los componentes ni en la prima pura integrada. Esta evidencia respalda adoptar como base operativa modelos sin remuestreo y con pre-poda, integrados multiplicativamente y calibrados con c.

CAPÍTULO VII. RECOMENDACIONES

En atención a la evidencia empírica y manteniendo coherencia con la descomposición actuarial de la prima pura $\pi(x) = \lambda(x)\mu(x)$ (frecuencia por severidad) sobre celdas de la tabla de experiencia, se recomienda adoptar en producción la pre-poda sin remuestreo como configuración base para los modelos de Severidad (Gamma) y Frecuencia (Binomial Negativa con offset), integrados multiplicativamente para la estimación de la Prima Pura. Asimismo, se sugiere conservar el árbol completo sin remuestreo como referencia interna, auditoría y análisis de sensibilidad, dada su utilidad para contrastar cambios de estructura sin interferir con la operación cotidiana.

Debe evitarse el sobremuestreo, dado que tiende a desplazar el nivel agregado de la prima pura sin traducirse en mejoras sistemáticas del WAPE una vez aplicada la calibración, y el submuestreo, debido a que deteriora de forma consistente devianza y errores al remover información relevante del cuerpo de la distribución. Solo en escenarios excepcionales —por restricciones severas de cómputo, ventanas de actualización muy reducidas o limitaciones operativas temporales— podría considerarse pre-poda con submuestreo, pero de manera acotada, documentada y con monitoreo reforzado.

La calibración global de nivel debe aplicarse sistemáticamente para garantizar neutralidad actuarial agregada sin alterar las relatividades entre riesgos. En particular, el factor c_{cal} (razón entre total observado y total predicho no calibrado) debe estimarse con una segmentación consistente y aplicarse como re-anclaje operativo del portafolio. Esta práctica debe complementarse con seguimiento local mediante razones Observado/Esperado (O/E) por segmentos relevantes (tarifa, región y bandas de riesgo) y por deciles de prima pura, con criterios explícitos de intervención y documentación formal cuando se detecten desviaciones persistentes.

Para administrar el riesgo en extremos y corregir la subestimación en colas altas, se recomienda implementar márgenes o recargos focalizados en pólizas de alto potencial de pérdida, junto con límites operativos y reglas de revisión de suscripción para combinaciones inusuales de características (p. ej., usos atípicos, categorías raras o condiciones que concentren severidad alta). Adicionalmente, es pertinente evaluar esquemas de calibración local en cola (p. ej., calibración monotónica/isotónica por bandas superiores), diseñados para corregir sesgos en extremos sin distorsionar el comportamiento del cuerpo de la distribución.

Con el fin de preservar parsimonia, estabilidad y trazabilidad, el conjunto de predictores debe mantenerse reducido y consistente en el tiempo. En severidad, se recomienda priorizar variables estructurales con alta contribución (p. ej., clase tarifaria y año modelo), y agrupar categorías raras para mejorar robustez. En frecuencia, además del offset de exposición, se recomienda conservar $\log(\text{exposición})$como predictor cuando capture elasticidades diferenciales no absorbidas por el offset. Variables de aporte marginal deben eliminarse para reducir varianza y costo de mantenimiento. Toda modificación (variables, hiperparámetros, reglas de agrupación y ventanas temporales) debe registrarse en una bitácora técnica, incluyendo motivación, evidencia, impacto y fecha de despliegue.

Finalmente, se sugiere profundizar líneas de investigación aplicada orientadas a: (i) reforzar la modelación de colas (familias Tweedie, mixturas Gamma–Pareto, métodos de valores extremos y regresión cuantílica); (ii) evaluar modelos con exceso de ceros en frecuencia (inflacionados de ceros o tipo hurdle) o calibraciones por bandas para mejorar sensibilidad en conteos ≥ 3; y (iii) explorar restricciones monotónicas y funciones de pérdida específicas en esquemas de gradiente potenciado, cuando se requiera mayor precisión con control de complejidad. La validación en nuevos periodos y contextos, idealmente incorporando variables operativas, geoespaciales y de comportamiento, permitirá evaluar la transferibilidad del enfoque y fortalecer su robustez frente a cambios en el entorno de riesgo.

REFERENCIAS Y ANEXO

Referencias:

Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys, 4*, 40–79. https://doi.org/10.1214/09-SS054

Bohanec, M., & Bratko, I. (1994). Trading accuracy for simplicity in decision trees. *Machine Learning, 15*(3), 223–250. https://doi.org/10.1007/BF00993345

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth. https://doi.org/10.1201/9781315139470

Brunello, A., Marzano, E., Montanari, A., & Sciavicco, G. (2017). Decision-tree pruning via multi-objective evolutionary computation. *International Journal of Machine Learning and Computing, 7*(6), 167–175. https://doi.org/10.18178/ijmlc.2017.7.6.641

Cadena, M. (2010). *An efficient algorithm for premium calcul*. En *International Congress of Actuaries 2010 (ASTIN track)* [Conference paper]. International Actuarial Association. https://www.actuaries.org/app/uploads/2025/07/ICA2010_ASTIN_25_final-paper_Cadena.pdf

Cameron, A. C., & Trivedi, P. K. (2013). *Regression analysis of count data* (2.ª ed.). Cambridge University Press.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research, 16*, 321–357. https://doi.org/10.1613/jair.953

Chen, W., Yang, K., Yu, Z., & Guo, P. (2024). A survey on imbalanced learning: Latest research, applications and future directions. *Artificial Intelligence Review, 57*, Artículo 137. https://doi.org/10.1007/s10462-024-10759-6

Cherubini, U., Luciano, E., & Vecchiato, W. (2004). *Copula methods in finance*. Wiley.

Ciampi, A., Thiffault, J., Nakache, J.-P., & Asselain, B. (1991). RECPAM: A computer program for recursive partition and amalgamation for censored survival data and other situations frequently occurring in biometry. *Computers and Biomedical Research, 24*(5), 429–445. https://doi.org/10.1016/0010-4809(91)90016-B

Cieslak, D. A., & Chawla, N. V. (2008). Start globally, optimize locally, predict globally: Improving performance on imbalanced data. En *Proceedings of the IEEE International Conference on Data Mining* (pp. 143–152). IEEE. https://doi.org/10.1109/ICDM.2008.87

Diana, A., Griffin, J. E., Oberoi, J., & Yao, J. (2019). *Machine-learning methods for insurance applications: A survey*. Society of Actuaries.

https://www.soa.org/493479/globalassets/assets/files/resources/research-report/2019/machine-learning-methods.pdf

Dobson, A. J., & Barnett, A. G. (2018). *An introduction to generalized linear models* (4.ª ed.). Chapman & Hall/CRC.

Dunn, P. K., & Smyth, G. K. (2018). *Generalized linear models with examples in R*. Springer. https://doi.org/10.1007/978-1-4419-0118-7

Elomaa, T., & Kääriäinen, M. (2001). An analysis of reduced error pruning. *Journal of Artificial Intelligence Research, 15*, 163–187.

Frees, E. W. (2009). *Regression modeling with actuarial and financial applications*. Cambridge University Press.

Frees, E. W. (2018). *Loss data analytics*. Open Actuarial Textbooks. Recuperado el 29 de octubre de 2025, de https://arxiv.org/pdf/1808.06718.pdf

Hanafy, M., & Ming, R. (2021). Improving imbalanced data classification in auto insurance by data-level approaches. *International Journal of Advanced Computer Science and Applications, 12*(6), 493–499. https://doi.org/10.14569/IJACSA.2021.0120656

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning* (2.ª ed.). Springer. https://doi.org/10.1007/978-0-387-84858-7

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering, 21*(9), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

Henckaerts, R., Côté, M.-P., Antonio, K., & Verbelen, R. (2021). Boosting insights in insurance tariff plans with tree-based machine learning methods. *North American Actuarial Journal, 25*(2), 255–285. https://doi.org/10.1080/10920277.2020.1745656

Hilbe, J. M. (2011). *Negative binomial regression* (2.ª ed.). Cambridge University Press.

Hilbe, J. M. (2014). *Modeling count data*. Cambridge University Press. https://doi.org/10.1017/CBO9781139236065

Hssina, B., Merbouha, A., Ezzikouri, H., & Erritali, M. (2014). A comparative study of decision tree ID3 and C4.5. *International Journal of Advanced Computer Science and Applications, 4*(Supl. 2), 13–19. https://doi.org/10.14569/SpecialIssue.2014.040203

Hunt, E. B. (1962). *Concept formation: An information processing problem*. Wiley.

Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: Principles and practice* (3.ª ed.). OTexts. Recuperado el 29 de octubre de 2025, de https://otexts.com/fpp3/

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting, 22*(4), 679–688. https://doi.org/10.1016/j.ijforecast.2006.03.001

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning* (2.ª ed.). Springer. https://doi.org/10.1007/978-1-0716-1418-1

Jiménez, R. (2021). *Estructuras de dependencia en modelos colectivos de riesgo mediante cópulas* [Trabajo fin de máster]. Universidad Carlos III de Madrid. Recuperado el 29 de octubre de 2025, de https://hdl.handle.net/10016/33128

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. *Journal of Big Data, 6*, Artículo 27. https://doi.org/10.1186/s40537-019-0192-5

Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society: Series C (Applied Statistics), 29*(2), 119–127. https://doi.org/10.2307/2986296

Klugman, S., Panjer, H., & Willmot, G. (2012). *Loss models: From data to decisions* (4.ª ed.). Wiley.

Kohavi, R., & Quinlan, J. R. (2002). Decision tree discovery. En W. Kloesgen & J. M. Zytkow (Eds.), *Handbook of data mining and knowledge discovery* (pp. 267–276). Oxford University Press.

Kulkarni, A., Batarseh, F. A., & Chong, D. (2021). Foundations of data imbalance and solutions for a data democracy. En *Data democracy: Data sharing and privacy* (pp. 85–106). Springer. Recuperado el 29 de octubre de 2025, de https://arxiv.org/pdf/2108.00071

Kushwah, J. S., Kumar, A., Patel, S., Soni, R., Gawande, A., & Gupta, S. (2022). Comparative study of regressor and classifier with decision tree using modern tools. En *Proceedings of the First International Conference on Design and Materials (ICDM 2021)* (pp. 32–39).

Loh, W. Y. (2014). Fifty years of classification and regression trees. *International Statistical Review, 82*(3), 329–348. https://doi.org/10.1111/insr.12016

Louppe, G. (2014). *Understanding random forests: From theory to practice* [Tesis doctoral]. Université de Liège. https://arxiv.org/abs/1407.7502

Mausam. (2011). Supervised learning (cont'd) – Decision trees (Lecture 21). University of Washington. Recuperado el 29 de octubre de 2025, de https://courses.cs.washington.edu/courses/csep573/11wi/lectures/21-dtree.pdf

McCullagh, P., & Nelder, J. A. (1989). *Generalized linear models* (2.ª ed.). Chapman & Hall.

McFadden, D. (1979). Quantitative methods for analyzing travel behavior of individuals: Some recent developments. En D. A. Hensher & P. R. Stopher (Eds.), *Behavioural travel modelling* (pp. 279–318).

Molnar, C. (2025). Decision tree. En *Interpretable machine learning* (3.ª ed.). Recuperado el 29 de octubre de 2025, de https://christophm.github.io/interpretable-ml-book/tree.html

Morgan, J. N., & Messenger, R. C. (1973). *THAID: A sequential analysis program for the analysis of nominal scale dependent variables*. Institute for Social Research, University of Michigan.

Morgan, J. N., & Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association, 58*(302), 415–434.

Ohlsson, E., & Johansson, B. (2010). *Non-life insurance pricing with generalized linear models*. Springer. https://doi.org/10.1007/978-3-642-10791-7

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*(1), 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann.

Riddle, P. (1998). Reduced error pruning. En *Decision tree learning (Course 706_98)*. University of Auckland. Recuperado el 29 de octubre de 2025, de https://www.cs.auckland.ac.nz/~pat/706_98/ln/node90.html

Ridzuan, A. N. A. (2024). Health insurance premium pricing using machine learning methods. *Journal of Advanced Research in Applied Sciences and Engineering Technology, 41*(1), 134–141. https://doi.org/10.37934/ARASET.41.1.134141

Ritschard, G. (2006). Computing and using the deviance with classification trees. En A. Rizzi & M. Vichi (Eds.), *Compstat 2006 – Proceedings in computational statistics* (pp. 55–66). Physica Verlag. https://doi.org/10.1007/978-3-7908-1709-6_5

Roback, P., & Legler, J. (2021). Poisson regression. En *Beyond multiple linear regression: Applied generalized linear models and multilevel models in R*. Recuperado el 29 de octubre de 2025, de https://bookdown.org/roback/bookdown-BeyondMLR/ch-poissonreg.html

Safavian, S. R., & Landgrebe, D. A. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics, 21*(3), 660–674. https://doi.org/10.1109/21.97458

scikit-learn Developers. (2025). *1.10. Decision trees — scikit-learn 1.7.2 documentation*. Recuperado el 29 de octubre de 2025, de https://scikit-learn.org/stable/modules/tree.html

Shamitha, S. K., & Ilango, V. (2019). A hybrid approach for health insurance fraud detection on highly imbalanced dataset. *International Journal of Engineering and Advanced Technology, 8*(6S), 1091–1095. https://doi.org/10.35940/IJRTE.F1210.0886S19

Song, Y., & Lu, Y. (2015). Decision tree methods: Applications for classification and prediction. *Shanghai Archives of Psychiatry, 27*(2), 130–135. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/

Statsmodels Developers. (2025). *Statsmodels.genmod.families.family* [Código fuente]. Recuperado el 29 de octubre de 2025, de https://www.statsmodels.org/dev/_modules/statsmodels/genmod/families/family.html

Su, X., & Bai, M. (2020). Stochastic gradient boosting frequency-severity model of insurance claims. *PLOS ONE, 15*(8), e0238000. https://doi.org/10.1371/journal.pone.0238000

Superintendência de Seguros Privados (SUSEP). (s. f.). *Auto Seguros – Portal de estadísticas*. Recuperado el 29 de octubre de 2025, de https://www2.susep.gov.br/menuestatistica/autoseg/principal.aspx

Therneau, T. M., & Atkinson, E. J. (2023). *An introduction to recursive partitioning using the rpart routines* (Vignette). Mayo Foundation for Medical Education and Research. Recuperado el 29 de octubre de 2025, de https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf

Train, K. (2009). *Discrete choice methods with simulation* (2.ª ed.). Cambridge University Press.

Werner, G., & Modlin, C. (2016). *Basic ratemaking* (5th ed.). Casualty Actuarial Society.

Windeatt, T., & Ardeshir, G. (2001). An empirical comparison of pruning methods for ensemble classifiers. En *Proceedings of the 4th International Symposium on Intelligent Data Analysis* (pp. 208–217). Springer. https://doi.org/10.1007/3-540-44816-0_21

Zhao, X. (2018). *Prediction of panel and streaming data using wavelet transform based decision trees* [Tesis doctoral]. University of Leeds. Recuperado el 29 de octubre de 2025, de https://etheses.whiterose.ac.uk/22431/

Zorman, M., Štiglic, M. M., Kokol, P., & Malčić, I. (1997). The limitations of decision trees and automatic learning in real world medical decision making. *Journal of Medical Systems, 21*(6), 403–415. https://doi.org/10.1023/A:1022876330390

Anexo:

Código

```
#-*- coding: utf-8-*-
# ================================================================
# 0) Test GPU (Windows + CUDA 12.5, RTX 50xx) — CuPy con PTX
# ================================================================
import os, time, math, warnings, logging, gc
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)

# Pandas: evitar copias innecesarias
import pandas as pd
try:
    pd.set_option('mode.copy_on_write', True)
except Exception:
    pass

# Forzar 1 hilo BLAS (portátiles/Jupyter)
os.environ.setdefault("OMP_NUM_THREADS", "1")
os.environ.setdefault("OPENBLAS_NUM_THREADS", "1")
os.environ.setdefault("MKL_NUM_THREADS", "1")

def _set_nvrtc_arch_from_device():
    try:
        import cupy as cp
        props = cp.cuda.runtime.getDeviceProperties(0)
        major, minor = props['major'], props['minor']
        candidates = [f"compute_{major}{minor}", "compute_90", "compute_89", "compute_86"]
        if "CUPY_NVRTC_ARCH" not in os.environ:
            os.environ["CUPY_NVRTC_ARCH"] = candidates[0]
    except Exception:
```

```
        pass


def test_gpu_windows_cuda125():
    os.environ.setdefault("CUPY_NVRTC_PATH", r"C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.5\bin\nvrtc64_125_0.dll")
    os.environ.setdefault("CUPY_COMPILE_WITH_PTX", "1")
    if "CUPY_NVRTC_ARCH" not in os.environ:
        _set_nvrtc_arch_from_device()
    try:
        import cupy as cp
        name = cp.cuda.runtime.getDeviceProperties(0)['name']
        print("GPU:", name)
        print("NVRTC_PATH:", os.environ.get("CUPY_NVRTC_PATH"))
        print("PTX mode:", os.environ.get("CUPY_COMPILE_WITH_PTX"))
        print("NVRTC_ARCH:", os.environ.get("CUPY_NVRTC_ARCH"))
        t0 = time.perf_counter()
        x = cp.random.rand(2048, 2048, dtype=cp.float32)
        y = cp.random.rand(2048, 2048, dtype=cp.float32)
        z = x @ y
        cp.cuda.Stream.null.synchronize()
        t1 = time.perf_counter()
        print("OK (suma):", float(z.sum()))
        print("Tiempo GPU (s):", round(t1- t0, 3))
        print("Memoria pool CuPy (MB):", round(cp.get_default_memory_pool().used_bytes()/1e6, 1))
    except Exception as e:
        print("[GPU TEST] Falla:", repr(e))
        print("Sugerencia: revisa CUDA 12.5 o cambia CUPY_NVRTC_ARCH a 'compute_89'.")


# ================================================================
# 1) Utils XPU (NumPy/CuPy), devianzas NB/Poisson y estimadores
# ================================================================
logger = logging.getLogger("modelo")
```

```
if not logger.handlers:
    _h = logging.StreamHandler()
    _h.setFormatter(logging.Formatter("[%(levelname)s] %(message)s"))
    logger.addHandler(_h)
logger.setLevel(logging.INFO)

import numpy as np
from typing import Optional, Dict, Any, List, Tuple
from sklearn.model_selection import train_test_split

def seed_everything(seed: int = 42):
    import random
    random.seed(seed); np.random.seed(seed)
    try:
        import cupy as _cp
        _cp.random.seed(seed)
    except Exception:
        pass

def _try_import_cupy():
    try:
        import cupy as cp
        gpu_ok = cp.cuda.runtime.getDeviceCount() > 0
        return cp if gpu_ok else None
    except Exception:
        return None

def xp_module(use_gpu: bool):
    if use_gpu:
        cp = _try_import_cupy()
        if cp is not None:
            return cp
```

```
    return np

def xp_to_numpy(x):
    try:
        import cupy as cp
        if isinstance(x, cp.ndarray):
            return cp.asnumpy(x)
    except Exception:
        pass
    return np.asarray(x)

def to_xp_array(a, xp):
    if xp is np:
        try:
            import cupy as cp
            if isinstance(a, cp.ndarray): return cp.asnumpy(a)
        except Exception:
            pass
        if isinstance(a, (pd.Series, pd.DataFrame)): return a.to_numpy()
        return np.asarray(a)
    # xp es cupy
    try:
        import cudf  # opcional
        if isinstance(a, (cudf.Series, cudf.DataFrame)):
            return a.to_cupy() if hasattr(a, "to_cupy") else a.values
    except Exception:
        pass
    if isinstance(a, (pd.Series, pd.DataFrame)): a = a.to_numpy()
    return xp.asarray(a)

def xp_scalar(x):
    try: return float(x.item())
```

```
    except Exception: pass
    try: return float(np.asarray(x).item())
    except Exception: return float(x)

POISSON_THETA = 1e6

def dev_poisson_xp(y, mu, xp=np, eps=1e-12, dtype=None):
    if dtype is None: dtype = xp.float64
    y  = to_xp_array(y, xp).astype(dtype)
    mu = xp.maximum(to_xp_array(mu, xp).astype(dtype), xp.asarray(eps, dtype=dtype))
    with (np.errstate if xp is np else xp.errstate)(divide='ignore', invalid='ignore'):
        term = xp.where(y == 0, mu, y*(xp.log(xp.maximum(y, eps))- xp.log(mu))- y + mu)
    return 2.0 * xp_scalar(term.sum())

def dev_nb_xp(y, mu, theta, xp=np, eps=1e-12, dtype=None):
    if theta >= POISSON_THETA: return dev_poisson_xp(y, mu, xp=xp, eps=eps, dtype=dtype)
    if dtype is None: dtype = xp.float64
    y  = to_xp_array(y, xp).astype(dtype)
    mu = xp.maximum(to_xp_array(mu, xp).astype(dtype), xp.asarray(eps, dtype=dtype))
    th = xp.asarray(theta, dtype=dtype)
    with (np.errstate if xp is np else xp.errstate)(divide='ignore', invalid='ignore'):
        t1 = xp.where(y>0, y*(xp.log(xp.maximum(y, eps))- xp.log(mu)), 0.0)
        t2 = (y + th) * (xp.log1p(y/th)- xp.log1p(mu/th))
    return 2.0 * xp_scalar((t1- t2).sum())
# Nota: aquí theta es el parámetro "size" de NB2 (theta = 1/k en la notación teórica);
#       Var(Y)=mu + mu^2/theta  <=>  Var(Y)=mu + k*mu^2.
def dev_nb_xp(y, mu, theta, xp=np, eps=1e-12, dtype=None):
def estimate_theta_offset_moments(y, expo, r: Optional[float]=None, eps: float=1e-12):
"""
Estima θ (NB2) por momentos...
Var(Y)=μ+μ^2/θ.  (Equivalente a Var(Y)=μ+kμ^2 con k = 1/θ).
"""
```

```
    y = np.asarray(y, float); expo = np.asarray(expo, float)
    if r is None: r = float(y.sum()) / max(float(expo.sum()), eps)
    mu = r * np.maximum(expo, 1e-10)
    num = float(np.sum(mu**2))
    den = float(np.sum((y- mu)**2- mu))
    if not np.isfinite(den) or den <= max(1e-8 * max(1.0, float(np.sum(mu))), 0.0): return
POISSON_THETA
    th = num/den
    return float(th) if np.isfinite(th) and th > 0 else POISSON_THETA


def r_hat_nb_xp(y, expo, theta, xp=np, tol=1e-8, maxit=50, r_init: Optional[float]=None,
dtype=None):
    if dtype is None: dtype = xp.float64
    y = to_xp_array(y, xp).astype(dtype)
    expo = to_xp_array(expo, xp).astype(dtype)
    if r_init is None:
        r = xp_scalar(y.sum()) / max(xp_scalar(expo.sum()), 1e-12)
    else:
        r = float(max(r_init, 1e-12))
    if theta >= POISSON_THETA or xp_scalar(y.sum()) <= 0.0: return r
    expo_pos = xp.maximum(expo, 1e-10)
    def _loss(rr): return dev_nb_xp(y, rr*expo_pos, theta, xp=xp, dtype=dtype)
    f_prev = _loss(r)
    for _ in range(maxit):
        denom = r*expo_pos + theta
        with (np.errstate if xp is np else xp.errstate)(divide='ignore', invalid='ignore'):
            g  = 2.0 * xp_scalar((-(y/r) + (y+theta)*(expo_pos/denom)).sum())
            g2 = 2.0 * xp_scalar(((y/(r**2))- (y+theta)*(expo_pos**2)/(denom**2)).sum())
        if abs(g) <= tol * max(1.0, int(y.size)): return max(r, 1e-12)
        g_s, g2_s = r*g, r*r*g2 + r*g
        lam = 1e-4
        denom_s = (g2_s if abs(g2_s) > 1e-10 else 0.0) + lam * max(1.0, abs(g_s)) or 1e-12
        step_s = float(np.clip(g_s / denom_s,-0.5, 0.5))
```

```
        s, s_try = math.log(max(r, 1e-12)), None
        s_try = s- step_s
        r_try = math.exp(s_try)
        ok = False
        for _bt in range(6):
            if not (np.isfinite(r_try)) or r_try <= 0: r_try = max(r*0.5, 1e-12); s_try = math.log(r_try)
            f_try = _loss(r_try)
            if not np.isfinite(f_try): s_try = 0.5*s + 0.5*s_try; r_try = math.exp(s_try); continue
            t = s_try- s
            if f_try <= f_prev + 1e-4 * g_s * t: ok = True; break
            s_try = 0.5*s + 0.5*s_try; r_try = math.exp(s_try)
        if ok:
            if abs(r_try- r) <= tol*max(r,1.0): return max(r_try, 1e-12)
            r, f_prev = r_try, f_try
        else:
            return max(r, 1e-12)
    return max(r, 1e-12)

# ================================================================
# 2) Balanceos RAM-friendly
# ================================================================
def balance_data_rate(
    X: pd.DataFrame, y: pd.Series, expo_col: str,
    method='none', n_bins=10, factor=0.10, max_per_bin=None, random_state=42,
    pos_target: Optional[float]=None, target_col_name="__y__", cap_dup:int=3,
    cap_total_rows: Optional[int]=None
)-> Tuple[pd.DataFrame, pd.Series]:
    if method == 'none':
        return X.reset_index(drop=True), y.reset_index(drop=True)

    rng = np.random.default_rng(random_state)
    n = len(X)
```

```
idx_all = np.arange(n, dtype=np.int64)

yv = pd.to_numeric(y, errors='coerce').astype(float).to_numpy()
ev = np.maximum(pd.to_numeric(X[expo_col], errors='coerce').astype(float).to_numpy(), 1e-8)
rate = np.nan_to_num(yv/ev, nan=0.0, posinf=0.0, neginf=0.0)

bins = pd.qcut(rate, q=n_bins, duplicates='drop')
idx_groups = (pd.DataFrame({'__bin__': bins, '__i__': idx_all})
              .groupby('__bin__', observed=True)['__i__'].apply(np.array))

idx_parts = []
for arr in idx_groups:
    idx_b = np.asarray(arr, dtype=np.int64)
    nb = idx_b.size
    if nb == 0: continue
    if method == 'over':
        target_n = int(np.ceil(nb*(1+factor)))
        if max_per_bin is not None:
            target_n = min(target_n, int(max_per_bin))
        if target_n <= nb:
            idx_parts.append(idx_b)
        else:
            add = rng.choice(idx_b, size=target_n-nb, replace=True)
            idx_parts.append(np.concatenate([idx_b, add]))
    elif method == 'under':
        target_n = int(np.floor(nb*(1-factor)))
        if target_n > 0:
            take = rng.choice(idx_b, size=target_n, replace=False)
            idx_parts.append(take)

idx_sel = np.concatenate(idx_parts) if idx_parts else idx_all.copy()
```

```
    if pos_target is not None:
        n_pos_sel = int(np.sum(yv[idx_sel] > 0))
        n_neg_sel = idx_sel.size- n_pos_sel
        needed_pos = int(np.ceil((pos_target/(1-pos_target)) * max(n_neg_sel,0)))- n_pos_sel
        if cap_total_rows is not None:
            needed_pos = min(needed_pos, max(0, int(cap_total_rows)- idx_sel.size))
        if needed_pos > 0:
            pos_pool = idx_all[yv > 0]
            if pos_pool.size > 0:
                addp = rng.choice(pos_pool, size=needed_pos, replace=True)
                idx_sel = np.concatenate([idx_sel, addp])

    if cap_dup is not None and cap_dup > 0:
        counts = np.bincount(idx_sel, minlength=n)
        counts = np.minimum(counts, int(cap_dup))
        idx_sel = np.repeat(idx_all, counts)

    if cap_total_rows is not None and idx_sel.size > int(cap_total_rows):
        idx_sel = np.random.default_rng(random_state).choice(idx_sel, size=int(cap_total_rows),
replace=False)

    Xb = X.iloc[idx_sel].reset_index(drop=True).astype('float32', copy=False)
    yb = y.iloc[idx_sel].reset_index(drop=True).astype('float32', copy=False)
    return Xb, yb

def balance_data_gamma(
    X: pd.DataFrame, y: pd.Series,
    method='none', n_bins=10, factor=0.10, random_state=42,
    cap_total_rows: Optional[int]=None
)-> Tuple[pd.DataFrame, pd.Series]:
    if method == 'none':
        return X.reset_index(drop=True), y.reset_index(drop=True)
```

```
    df = pd.concat([X.reset_index(drop=True), y.reset_index(drop=True).rename('__y__')], axis=1)
    bins = pd.qcut(df['__y__'], q=n_bins, duplicates='drop')
    groups = pd.DataFrame({'__bin__': bins}).groupby('__bin__', observed=True).indices
    parts = []
    for _, idx in groups.items():
        idx = np.asarray(idx, int)
        n = idx.size
        base = df.iloc[idx]
        if method == 'over':
            target_n = int(np.ceil(n*(1+factor)))
            parts.append(base if target_n <= n else base.sample(n=target_n, replace=True,
random_state=random_state))
        elif method == 'under':
            target_n = int(np.floor(n*(1-factor)))
            if target_n > 0:
                parts.append(base.sample(n=target_n, replace=False, random_state=random_state))
    dfb = pd.concat(parts, axis=0, ignore_index=True) if parts else df
    if cap_total_rows is not None and len(dfb) > int(cap_total_rows):
        dfb = dfb.sample(n=int(cap_total_rows), random_state=random_state).reset_index(drop=True)
    return dfb.drop(columns=['__y__']).reset_index(drop=True).astype('float32', copy=False),
dfb['__y__'].astype('float32').reset_index(drop=True)


# ============================================================
# 3) Árbol NB con pre-poda (min_expo / min_gain_rel) + REP opcional
# ============================================================
class NBNode:
    __slots__ = ("r","left","right","feat","feat_name","val","gain")
    def __init__(self, r=None, left=None, right=None, feat=None, feat_name=None, val=None,
gain=0.0):
        self.r, self.left, self.right = r, left, right
        self.feat, self.feat_name, self.val, self.gain = feat, feat_name, val, gain


class NBTreeOffset:
```

```
    def __init__(self, X: pd.DataFrame, y: pd.Series, expo_col='EXPOSICAO1',
                 max_depth=30, min_samples=20, max_splits_per_feature=128,
                 min_gain=1e-9, min_gain_rel: float = 0.0, min_expo: float = 0.0,
                 alpha_prior=50.0, rate_cap=np.inf, r0_prior=None,
                 verbose: bool=True):
        self.columns = list(X.columns); self.expo_col = expo_col
        self.verbose = bool(verbose)
        self._X = X.to_numpy(copy=False).astype(np.float32, copy=False)
        self._y = y.to_numpy(dtype=np.float32, copy=False)
        self._expo = X[expo_col].to_numpy(dtype=np.float32, copy=False)
        self._feat_idx = [i for i,c in enumerate(self.columns) if c != expo_col]

        self.max_depth=int(max_depth)
        self.min_samples=int(min_samples)
        self.max_splits_per_feature=int(max_splits_per_feature)
        self.min_gain=float(min_gain)
        self.min_gain_rel=float(min_gain_rel)
        self.min_expo=float(min_expo)

        self.alpha_prior=float(alpha_prior); self.rate_cap=float(rate_cap)
        self.r0 = float(r0_prior) if r0_prior is not None else
(float(self._y.sum())/max(float(self._expo.sum()),1e-12))
        self.theta = estimate_theta_offset_moments(self._y, self._expo, r=self.r0)
        self.root=None

    def _leaf_rate(self, idx):
        y = self._y[idx]; e = self._expo[idx]
        r_mle = r_hat_nb_xp(y, e, self.theta, xp=np)
        expo_sum = float(e.sum()); expo_sum = max(expo_sum, 1e-8)
        w = self.alpha_prior / (self.alpha_prior + expo_sum)
        r = (1.0- w) * float(r_mle) + w * self.r0
        return min(r, self.rate_cap)
```

```
def _dev_node(self, y, expo, r_init=None):
    r = r_hat_nb_xp(y, expo, self.theta, xp=np, r_init=r_init)
    mu = r * np.maximum(expo, 1e-10)
    return dev_nb_xp(y, mu, self.theta, xp=np), r

def _best_split(self, idx)-> Tuple[float, Optional[int], Optional[float], Any, Any]:
    ms = self.min_samples
    y = self._y[idx]; e = self._expo[idx]
    expo_parent = float(np.sum(e))
    if expo_parent < max(1e-10, 2.0*self.min_expo):
        return-np.inf, None, None, None, None

    D_parent, _ = self._dev_node(y, e)
    gain_thr_abs = max(self.min_gain, self.min_gain_rel * float(D_parent))

    best_gain, best_f, best_v, best_L, best_R =-np.inf, None, None, None, None
    for f in self._feat_idx:
        col = self._X[idx, f]
        ok = ~np.isnan(col); n_ok = int(ok.sum())
        if n_ok < 2*ms: continue
        iok = idx[ok]; ina = idx[~ok]; n_na = int(ina.size)
        col_ok = col[ok]
        if np.nanmax(col_ok)- np.nanmin(col_ok) <= 0: continue
        y_ok = self._y[iok]; e_ok = self._expo[iok]

        ord_idx = np.argsort(col_ok, kind='mergesort')
        col_s   = col_ok[ord_idx]
        y_s     = y_ok[ord_idx]
        e_s     = e_ok[ord_idx]
        y_c     = np.cumsum(y_s, dtype=float)
        e_c     = np.cumsum(e_s, dtype=float)
```

```
iok_s   = iok[ord_idx]

u = np.unique(col_s)
if u.size > self.max_splits_per_feature:
    m = self.max_splits_per_feature
    idxs = np.linspace(1, u.size-2, m).round().astype(int)
    idxs = np.unique(idxs)
    thr = u[idxs]
else:
    thr = u
k = np.searchsorted(col_s, thr, side='right').astype(int)

left_ok  = k
right_ok = (int(y_s.size)- k) + n_na
valid = (left_ok >= ms) & (right_ok >= ms)
if not valid.any():
    continue

ks = k[valid].astype(int, copy=False)
for kk in ks:
    if kk <= 0 or kk >= int(y_s.size):
        continue
    t = float((col_s[kk-1] + col_s[kk]) * 0.5)

    L = iok_s[:kk]
    R = iok_s[kk:]
    if n_na > 0:
        R = np.concatenate([R, ina])

    # Guardas nuevas
    expo_L = float(self._expo[L].sum())
    expo_R = float(self._expo[R].sum())
```

```
            if (expo_L < self.min_expo) or (expo_R < self.min_expo):
                continue

            rL0 = float(y_c[kk-1] / max(e_c[kk-1], 1e-12)) if kk > 0 else self.r0
            eR  = (e_c[-1]- e_c[kk-1]) + (float(self._expo[ina].sum()) if n_na > 0 else 0.0)
            yR  =  y_c[-1]- y_c[kk-1]
            rR0 = float(yR / max(eR, 1e-12))

            DL, _ = self._dev_node(self._y[L], self._expo[L], r_init=rL0)
            DR, _ = self._dev_node(self._y[R], self._expo[R], r_init=rR0)
            gain = float(D_parent- (DL + DR))
            if gain > best_gain and gain >= gain_thr_abs:
                best_gain, best_f, best_v, best_L, best_R = gain, f, t, L, R
        return best_gain, best_f, best_v, best_L, best_R

    def _build(self, idx, depth):
        idx = np.asarray(idx, np.int32)
        expo_here = float(self._expo[idx].sum())
        if depth >= self.max_depth or idx.size < self.min_samples or expo_here < max(1e-10,
2.0*self.min_expo):
            return NBNode(r=float(self._leaf_rate(idx)))
        gain, f, v, iL, iR = self._best_split(idx)
        if (f is None) or (gain <= self.min_gain):
            return NBNode(r=float(self._leaf_rate(idx)))
        return NBNode(
            feat=int(f), feat_name=self.columns[int(f)], val=float(v), gain=float(gain),
            left = self._build(iL, depth+1),
            right= self._build(iR, depth+1)
        )

    def fit(self):
        n = int(self._y.size)
```

```
        idx = np.arange(n, dtype=np.int32)
        self.root = self._build(idx, 0)
        if self.verbose: logger.info(f"[NB] θ={self.theta:.4f} (r0={self.r0:.6g})")
        return self

    def _predict_traverse(self, node: NBNode, Xv: np.ndarray, expo: np.ndarray, idxs: np.ndarray, mu:
np.ndarray):
        if node.r is not None:
            mu[idxs] = np.minimum(node.r * np.maximum(expo[idxs], 1e-10),
self.rate_cap*np.maximum(expo[idxs],1e-10)); return
        col = Xv[idxs, node.feat]
        mask_left = (~np.isnan(col)) & (col <= node.val)
        self._predict_traverse(node.left , Xv, expo, idxs[mask_left], mu)
        self._predict_traverse(node.right, Xv, expo, idxs[~mask_left], mu)

    def predict(self, X_new: pd.DataFrame)-> np.ndarray:
        X_new = X_new.copy()
        for c in self.columns:
            if c not in X_new.columns: X_new[c] = 0.0
        X_new = X_new[self.columns]
        expo = np.maximum(pd.to_numeric(X_new[self.expo_col],
errors='coerce').fillna(0.0).to_numpy(float), 1e-10)
        Xv = X_new.to_numpy(copy=False).astype(np.float32, copy=False)
        mu = np.empty(Xv.shape[0], dtype=float)
        self._predict_traverse(self.root, Xv, expo, np.arange(Xv.shape[0], dtype=np.int32), mu)
        return mu

def reduced_error_pruning_offset(nodo: NBNode, X_val: pd.DataFrame, y_val: pd.Series,
                                 theta_val: float, expo_col='EXPOSICAO1',
                                 alpha_prior=50.0, r0_prior=None, rate_cap=np.inf, tol: float = 0.0):
    if nodo is None or len(y_val) == 0 or (nodo.left is None and nodo.right is None): return nodo
    f, t = nodo.feat, nodo.val
    col = X_val.iloc[:, f].to_numpy()
```

```
    mL = (~np.isnan(col)) & (col <= t); mR = ~mL

    if nodo.left  is not None: nodo.left  = reduced_error_pruning_offset(nodo.left , X_val[mL], y_val[mL], theta_val, expo_col, alpha_prior, r0_prior, rate_cap, tol)

    if nodo.right is not None: nodo.right = reduced_error_pruning_offset(nodo.right, X_val[mR], y_val[mR], theta_val, expo_col, alpha_prior, r0_prior, rate_cap, tol)


    tmp = NBTreeOffset(X_val, y_val, expo_col=expo_col); tmp.root = nodo

    y_hat_sub = tmp.predict(X_val)

    D_sub = dev_nb_xp(y_val.values, y_hat_sub, theta_val, xp=np)

    expo_val = np.maximum(pd.to_numeric(X_val[expo_col], errors='coerce').fillna(0.01).to_numpy(float), 1e-10)

    r_mle = r_hat_nb_xp(y_val.values, expo_val, theta_val, xp=np)

    w = alpha_prior / (alpha_prior + float(np.sum(expo_val)))

    r_shrunk = (1.0- w)*float(r_mle) + w*float(0.0 if r0_prior is None else r0_prior)

    mu_leaf = np.minimum(r_shrunk * expo_val, rate_cap * expo_val)

    D_leaf = dev_nb_xp(y_val.values, mu_leaf, theta_val, xp=np)

    return NBNode(r=float(r_shrunk)) if D_leaf <= D_sub + max(tol, 1e-6*abs(D_sub)) else nodo


# ============================================================

# 4) Árbol Gamma + REP

# ============================================================

class GammaNode:

    __slots__ = ("pred","left","right","feat","val","gain","nan_left")

    def __init__(self, pred=None, left=None, right=None, feat=None, val=None, gain=0.0, nan_left=True):

        self.pred, self.left, self.right = pred, left, right

        self.feat, self.val, self.gain   = feat, val, gain

        self.nan_left = bool(nan_left)


def gamma_deviance(y, yhat, eps=1e-12):

    y = np.asarray(y, float); yhat = np.maximum(np.asarray(yhat, float), eps)

    if np.any(y<=0): raise ValueError("Gamma deviance requiere y>0.")

    v = 2.0 * ((y- yhat)/yhat- np.log(y/yhat))
```

```
    return float(np.sum(v))


class GammaTree:
    def __init__(self, X: pd.DataFrame, y: pd.Series, max_depth=30, min_samples=20,
max_splits_per_feature=64, min_gain=1e-9):
        self.columns = list(X.columns)
        self._X = X.to_numpy(copy=False).astype(np.float32, copy=False)
        self._y = y.to_numpy(dtype=np.float32, copy=False)
        if np.any(self._y<=0): raise ValueError("GammaTree requiere y>0.")
        self._logy = np.log(self._y.astype(np.float64)).astype(np.float32)
        self.max_depth=int(max_depth); self.min_samples=int(min_samples)
        self.max_splits_per_feature=int(max_splits_per_feature); self.min_gain=float(min_gain)
        self.root=None

    @staticmethod
    def _leaf_dev(sum_y, sum_log_y, n, eps=1e-12):
        if n<=0: return np.inf
        mu = max(sum_y/max(n,1.0), eps)
        return 2.0*(n*np.log(mu)- sum_log_y)


    def _best_split(self, idx):
        idx = np.asarray(idx, np.int32)
        y_node = self._y[idx]; base = self._leaf_dev(float(y_node.sum()), float(self._logy[idx].sum()),
int(y_node.size))
        best_gain, best_f, best_v, best_L, best_R, best_nan_left =-np.inf, None, None, None, None,
True
        n_features = self._X.shape[1]; ms = self.min_samples
        for f in range(n_features):
            col_full = self._X[idx, f]; m_ok = ~np.isnan(col_full); n_ok = int(m_ok.sum())
            if n_ok < 2*ms: continue
            iok = idx[m_ok]; ina = idx[~m_ok]; n_na = int(ina.size)
            col_ok = col_full[m_ok]
            if np.nanmin(col_ok) == np.nanmax(col_ok): continue
```

```
        y_ok = self._y[iok]; logy_ok = self._logy[iok]

        ord_idx = np.argsort(col_ok, kind='mergesort')
        col_s   = col_ok[ord_idx]
        y_s     = y_ok[ord_idx]
        log_s   = logy_ok[ord_idx]
        py   = np.cumsum(y_s, dtype=float)
        plog = np.cumsum(log_s, dtype=float)
        ty, tlog = float(py[-1]), float(plog[-1])

        u = np.unique(col_s)
        if u.size > self.max_splits_per_feature:
            m=self.max_splits_per_feature
            idxs = np.linspace(1, u.size-2, m).round().astype(int); idxs = np.unique(idxs)
            cand = u[idxs]
        else:
            cand = u
        k = np.searchsorted(col_s, cand, side='right').astype(int)

        # NaN-> izquierda y derecha
        nL_A = k + n_na; nR_A = (int(y_s.size)- k)
        dev_A = np.full(k.shape, np.inf, float)
        ok_A = (nL_A>=ms) & (nR_A>=ms)
        if ok_A.any():
            pref_y = np.where(k>0, py[k-1], 0.0); pref_log = np.where(k>0, plog[k-1], 0.0)
            dev_L = 2.0*(nL_A*np.log(np.maximum((pref_y + (self._y[ina].sum() if n_na>0 else 0.0)) /
np.maximum(nL_A,1), 1e-12))
                        - (pref_log + (self._logy[ina].sum() if n_na>0 else 0.0)))
            dev_R = 2.0*((nR_A)*np.log(np.maximum((ty- pref_y) / np.maximum(nR_A,1), 1e-12))-
(tlog- pref_log))
            dev_A = np.where(ok_A, dev_L + dev_R, np.inf)

        nL_B = k; nR_B = (int(y_s.size)- k) + n_na
```

```
            dev_B = np.full(k.shape, np.inf, float)
            ok_B = (nL_B>=ms) & (nR_B>=ms)
            if ok_B.any():
                pref_y = np.where(k>0, py[k-1], 0.0); pref_log = np.where(k>0, plog[k-1], 0.0)
                dev_L = 2.0*(nL_B*np.log(np.maximum((pref_y)/np.maximum(nL_B,1), 1e-12))-
(pref_log))
                dev_R = 2.0*(nR_B*np.log(np.maximum(((ty- pref_y) + (self._y[ina].sum() if n_na>0 else
0.0)) / np.maximum(nR_B,1), 1e-12))
                        - ((tlog- pref_log) + (self._logy[ina].sum() if n_na>0 else 0.0)))
                dev_B = np.where(ok_B, dev_L + dev_R, np.inf)

            use_A = dev_A <= dev_B
            dev_c = np.where(use_A, dev_A, dev_B)
            j = int(np.argmin(dev_c))
            if not np.isfinite(dev_c[j]): continue
            gain = base- float(dev_c[j])
            if gain > best_gain:
                best_gain = gain; best_f = f; best_v = float((cand[j] + np.nextafter(cand[j], np.inf)) * 0.5)
                kstar = int(k[j])
                iok_s = iok[ord_idx]
                L = iok_s[:kstar]; R = iok_s[kstar:]
                if use_A[j] and n_na>0:  # NaN izquierda
                    L = np.concatenate([L, ina]); nan_left=True
                else:
                    nan_left=False
                    if n_na>0: R = np.concatenate([R, ina])
                best_L, best_R, best_nan_left = L, R, nan_left
        return best_gain, best_f, best_v, best_L, best_R, best_nan_left


    def _build(self, idx, depth):
        idx = np.asarray(idx, np.int32)
        if depth >= self.max_depth or idx.size < self.min_samples:
            return GammaNode(pred=float(self._y[idx].mean()))
```

```
        gain, f, v, L, R, nan_left = self._best_split(idx)
        if (f is None) or (gain <= self.min_gain): return GammaNode(pred=float(self._y[idx].mean()))
        return GammaNode(feat=f, val=float(v), gain=float(gain), nan_left=nan_left,
                         left=self._build(L, depth+1), right=self._build(R, depth+1))

    def fit(self):
        self.root = self._build(np.arange(self._y.size, dtype=np.int32), 0)
        return self

    def predict(self, X: pd.DataFrame)-> np.ndarray:
        if list(X.columns) != self.columns:
            X = X.copy(); missing=[c for c in self.columns if c not in X.columns]
            for c in missing: X[c]=0.0
            X = X[self.columns]
        xv = X.to_numpy(copy=False).astype(np.float32, copy=False)
        out = np.empty(xv.shape[0], float)
        root = self.root
        for i in range(xv.shape[0]):
            nd = root; row = xv[i]
            while nd.pred is None:
                x = row[nd.feat]
                if x != x: nd = nd.left if nd.nan_left else nd.right
                else:      nd = nd.left if x <= nd.val else nd.right
            out[i] = nd.pred
        return out

def gamma_reduced_prune(node: GammaNode, X_val: pd.DataFrame, y_val: pd.Series, tol=0.0):
    if len(y_val)==0 or (node.left is None and node.right is None): return node
    f, v = node.feat, node.val
    col = X_val.iloc[:, f].to_numpy()
    mL = np.where(np.isnan(col), node.nan_left, col <= v)
    mR = ~mL
```

```
    if node.left  is not None: node.left  = gamma_reduced_prune(node.left , X_val[mL], y_val[mL], tol)
    if node.right is not None: node.right = gamma_reduced_prune(node.right, X_val[mR], y_val[mR],
tol)
    gt_tmp = type("Tmp", (), {"root":node, "columns":list(X_val.columns)})()
    GammaTree.predict = GammaTree.predict
    y_hat_sub = GammaTree.predict(gt_tmp, X_val)
    D_sub = gamma_deviance(y_val.values, y_hat_sub)
    leaf_pred = float(y_val.mean()); D_leaf = gamma_deviance(y_val.values, np.full_like(y_val.values,
leaf_pred, float))
    return GammaNode(pred=leaf_pred) if D_leaf <= D_sub + max(tol, 1e-6*abs(D_sub)) else node


# ================================================================
# 5) Helpers de downcast (ahorro RAM)
# ================================================================
def downcast_onehots_inplace(X: pd.DataFrame,
onehot_prefixes=('SEXO_','REGIAO_','COD_TARIF_','COD_MODELO_')):
    for c in X.columns:
        if any(c.startswith(p) for p in onehot_prefixes):
            s = pd.to_numeric(X[c], errors='coerce').fillna(0)
            if s.isin([0,1]).all():
                X[c] = s.astype('uint8', copy=False)


# ================================================================
# 6) HP fijos por modo/rs (según tu búsqueda)
# ================================================================
def _align_cols(df: pd.DataFrame, cols: List[str])-> pd.DataFrame:
    if list(df.columns) == list(cols):
        return df
    Xv = df.copy()
    for c in cols:
        if c not in Xv.columns:
            Xv[c] = 0.0
    return Xv[cols]
```

```
def _seed_for(mode: str, rs: str, base: int = 42)-> int:
   offs_mode = {'pre': 100, 'full': 200, 'post': 300}[mode]
   offs_rs   = {'none': 0, 'under': 7, 'over': 13}.get(rs, 0)
   return int(base + offs_mode + offs_rs)


NB_HP = {
   ('pre','none') : dict(depth=12, min_samples=50, max_splits=128, min_gain_rel=0.0,
min_expo=0.0),
   ('pre','over') : dict(depth=12, min_samples=50, max_splits=128, min_gain_rel=0.0,
min_expo=0.0),
   ('pre','under'): dict(depth=12, min_samples=50, max_splits=128, min_gain_rel=0.0,
min_expo=0.0),


   ('full','none') : dict(depth=40, min_samples=50, max_splits=128, min_gain_rel=0.0,
min_expo=10.0),
   ('full','over') : dict(depth=40, min_samples=50, max_splits=128, min_gain_rel=0.0,
min_expo=10.0),
   ('full','under'): dict(depth=40, min_samples=50, max_splits=128, min_gain_rel=0.0,
min_expo=10.0),


   ('post','none') : dict(depth=20, min_samples=50, max_splits=128, min_gain_rel=0.0,
min_expo=0.0, tol=0.0, prune_frac=0.25),
   ('post','over') : dict(depth=20, min_samples=50, max_splits=128, min_gain_rel=0.0,
min_expo=0.0, tol=0.0, prune_frac=0.25),
   ('post','under'): dict(depth=20, min_samples=50, max_splits=128, min_gain_rel=0.0,
min_expo=0.0, tol=0.0, prune_frac=0.25),
}


GM_HP = {
   ('pre','none') : dict(depth=30, min_samples=10,  max_splits=64),
   ('pre','over') : dict(depth=30, min_samples=10,  max_splits=64),
   ('pre','under'): dict(depth=30, min_samples=10,  max_splits=64),


   ('full','none') : dict(depth=40, min_samples=10,  max_splits=64),
```

```
    ('full','over') : dict(depth=40, min_samples=10,  max_splits=64),
    ('full','under'): dict(depth=40, min_samples=10,  max_splits=64),

    ('post','none') : dict(depth=30, min_samples=10,  max_splits=64, tol=0.0, prune_frac=0.25),
    ('post','over') : dict(depth=30, min_samples=20,  max_splits=64, tol=0.0, prune_frac=0.25),
    ('post','under'): dict(depth=20, min_samples=10,  max_splits=64, tol=0.0, prune_frac=0.25),
}

# ================================================================
# 7) Pipelines NB / Gamma (REP solo en post)
# ================================================================
def run_nb_pipeline_explicit(
    csv_path: str,
    columnas_seleccionadas: List[str],
    target_col='FREQ_SIN3',
    expo_col='EXPOSICAO1',
    test_size=0.20,
    random_state=42,
    resampling_opts=('none','under','over'),
    resample_bins=8,
    resample_factor=0.05,
    alpha_prior=50.0
):
    seed_everything(random_state)
    try: df = pd.read_csv(csv_path, delimiter=';')
    except UnicodeDecodeError: df = pd.read_csv(csv_path, delimiter=';', encoding='latin-1')
    if target_col not in df.columns or expo_col not in df.columns:
        raise KeyError("Faltan columnas target/expo en el CSV.")

    # Expo + feature log
    df[expo_col] = pd.to_numeric(df[expo_col], errors='coerce').fillna(0.01).clip(lower=0.01)
    df['LOG_EXPOSICAO1'] = np.log(df[expo_col].astype(float) + 1e-6)
```

```
cols_pred_eff = list(dict.fromkeys(columnas_seleccionadas + ['LOG_EXPOSICAO1']))
missing = [c for c in cols_pred_eff if c not in df.columns]
if missing:
    df = pd.concat([df, pd.DataFrame(0.0, index=df.index, columns=missing)], axis=1)

print("\n[NB] Lista completa de columnas usadas ({}):".format(len(cols_pred_eff)))
print(cols_pred_eff)

X = df[cols_pred_eff].copy().astype('float32', copy=False)
y = pd.to_numeric(df[target_col], errors='coerce').fillna(0).astype('float32')
downcast_onehots_inplace(X)

nun = X.nunique(dropna=True); mask = (nun > 1); mask[expo_col] = True
dropped = [c for c in X.columns if not mask[c]]
if dropped:
    print(f"[NB] Columnas constantes eliminadas: {len(dropped)} (ej: {dropped[:10]})")
X = X.loc[:, mask].copy()

X_tr, X_te, y_tr, y_te = train_test_split(
    X, y, test_size=test_size, random_state=random_state, shuffle=True,
    stratify=(y>0).astype(int)
)

#---------- NULOS y THETA de REFERENCIA (definidos sobre TRAIN)----------
theta_ref = estimate_theta_offset_moments(y_tr.values, X_tr[expo_col].to_numpy(float))

# (offset) mu_null_off_i = r0 * e_i  con r0 = sum(y_tr)/sum(e_tr)
r_null = float(np.sum(y_tr.values)) / float(max(np.sum(X_tr[expo_col].to_numpy(float)), 1e-12))
mu_null_off = r_null * np.maximum(X_te[expo_col].to_numpy(float), 1e-10)
```

```
    # Positividad target para remuestreo
    orig_pos = float((y_tr.values > 0).mean())
    pos_target = min(max(orig_pos*2.0, orig_pos), 0.03)

    results, info = {}, {}
for mode in ['pre', 'full', 'post']:
    for rs in resampling_opts:
        hp = NB_HP[(mode, rs)]
        rs_seed = _seed_for(mode, rs, base=random_state)

        #--------- (A) Partición REP SOLO para post (antes de entrenar)---------
        if mode == 'post':
            Xp, Xv, yp, yv = train_test_split(
                X_tr, y_tr,
                test_size=hp.get('prune_frac', 0.25),
                random_state=rs_seed,
                shuffle=True
            )
            X_train_base, y_train_base = Xp, yp       # entrenamiento real
            X_prune, y_prune = Xv, yv                 # validación REP (NO tocar)
        else:
            X_train_base, y_train_base = X_tr, y_tr
            X_prune, y_prune = None, None

        #--------- (B) Remuestreo SIEMPRE sobre entrenamiento (si rs != none)---------
        if rs != 'none':
            n_tr = len(X_train_base)
            max_per_bin = int(np.ceil(n_tr / resample_bins) * 1.2)
            cap_total = int(n_tr * 1.3)
            cap_dup_nb = 3 if rs == 'over' else 1
            orig_pos = float((y_train_base.values > 0).mean())
            pos_target = min(max(orig_pos * 2.0, orig_pos), 0.03)
```

```
    Xb, yb = balance_data_rate(
        X_train_base, y_train_base, expo_col,
        method=rs, n_bins=resample_bins,
        factor=resample_factor, random_state=rs_seed,
        pos_target=pos_target, cap_dup=cap_dup_nb,
        max_per_bin=max_per_bin, cap_total_rows=cap_total
    )
else:
    Xb, yb = X_train_base, y_train_base

#--------- (C) Entrenar árbol con el entrenamiento (posiblemente remuestreado)---------
tree = NBTreeOffset(
    Xb, yb, expo_col=expo_col,
    max_depth=hp['depth'], min_samples=hp['min_samples'],
    max_splits_per_feature=hp['max_splits'],
    min_gain_rel=hp.get('min_gain_rel', 0.0),
    min_expo=hp.get('min_expo', 0.0),
    alpha_prior=alpha_prior, r0_prior=r_null, verbose=False
).fit()

#--------- (D) Post-poda REP usando X_prune SIN remuestreo---------
if mode == 'post':
    Xv_aligned = _align_cols(X_prune, tree.columns)
    theta_pr = estimate_theta_offset_moments(
        y_prune.values, Xv_aligned[expo_col].to_numpy(float)
    )
    tree.root = reduced_error_pruning_offset(
        tree.root, Xv_aligned, y_prune, theta_pr,
        expo_col=expo_col, alpha_prior=alpha_prior,
        r0_prior=r_null, rate_cap=np.inf, tol=hp.get('tol', 0.0)
    )
```

```
            if rs!='none':
               n_v = len(Xv)
               max_per_bin_v = int(np.ceil(n_v / resample_bins) * 1.2)
               cap_total_v   = int(n_v * 1.3)
               cap_dup_nb_v  = 3 if rs == 'over' else 1
               Xv, yv = balance_data_rate(
                  Xv, yv, expo_col, method=rs, n_bins=resample_bins,
                  factor=resample_factor, random_state=rs_seed+1,
                  pos_target=pos_target, cap_dup=cap_dup_nb_v,
                  max_per_bin=max_per_bin_v, cap_total_rows=cap_total_v
               )
            Xv_aligned = _align_cols(Xv, tree.columns)
            theta_pr = estimate_theta_offset_moments(yv.values,
Xv_aligned[expo_col].to_numpy(float))
            yhat_b = tree.predict(Xv_aligned); Dval_b = dev_nb_xp(yv.values, yhat_b, theta_pr, xp=np)
            tree.root = reduced_error_pruning_offset(
               tree.root, Xv_aligned, yv, theta_pr,
               expo_col=expo_col, alpha_prior=alpha_prior, r0_prior=r_null,
               rate_cap=np.inf, tol=hp.get('tol', 0.0)
            )
            yhat_a = tree.predict(Xv_aligned); Dval_a = dev_nb_xp(yv.values, yhat_a, theta_pr, xp=np)
            logger.info(f"[NB {mode}/{rs}] REP Dev(val) {Dval_b:.1f}->{Dval_a:.1f}")


         tree._X = None; tree._y = None; tree._expo = None; gc.collect()


         #--------- Métricas en TEST (Pseudo D² vs nulos definidos en TRAIN)---------
         y_hat = tree.predict(X_te)
         dev_model      = float(dev_nb_xp(y_te.values, y_hat,       theta_ref, xp=np))
         dev_null_off   = float(dev_nb_xp(y_te.values, mu_null_off, theta_ref, xp=np))   # offset
(r0*e_i)
         dev_null_const = float(dev_nb_xp(y_te.values, mu_te_const,
if dev_null_off  > 0 else float('nan')
         pseudo_const   = 1.0- (dev_model / dev_null_const) if dev_null_const> 0 else float('nan')
```

```
            mae  = float(np.mean(np.abs(y_te.values- y_hat)))
            rmse = float(np.sqrt(np.mean((y_te.values- y_hat)**2)))

            results[mode][rs] = {
               'Devianza NB': dev_model,
               'Pseudo D² NB (offset)': float(pseudo_off),
               'r_null': float(r_null),
               'Dev NB null (offset)': float(dev_null_off),

            }
            info[mode][rs] = {'model':tree, 'y_pred':y_hat, 'theta_ref': theta_ref, 'r_null': r_null}

    rows=[]
    for mode in results:
        for rs in results[mode]:
            rows.append({'mode':mode,'resampling':rs, **results[mode][rs]})
    summary_df = pd.DataFrame(rows).sort_values(['mode','Devianza NB']).reset_index(drop=True)
    print(f"\n[NB] Resumen {summary_df.shape[0]} modelos (incluye Pseudo D² offset definidos en
TRAIN):")
    print(summary_df)
    return info, results, X_tr, y_tr, X_te, y_te


def run_gamma_pipeline_explicit(
    csv_path: str,
    cols_pred: List[str],
    target_col='severidad_promedio3',
    test_size=0.20,
    random_state=42,
    resampling_opts=('none','under','over'),
    resample_bins=8, resample_factor=0.05
):
    seed_everything(random_state)
```

```
    try: df = pd.read_csv(csv_path, delimiter=';')
    except UnicodeDecodeError: df = pd.read_csv(csv_path, delimiter=';', encoding='latin-1')
    if target_col not in df.columns: raise KeyError("No se encuentra la columna de severidad en el CSV.")
    df = df[pd.to_numeric(df[target_col], errors='coerce') > 0].dropna(subset=[target_col]).copy()

    missing = [c for c in cols_pred if c not in df.columns]
    if missing:
        df = pd.concat([df, pd.DataFrame(0.0, index=df.index, columns=missing)], axis=1)

    print("\n[Γ] Lista completa de columnas usadas ({}):".format(len(cols_pred)))
    print(cols_pred)

    X = df[cols_pred].copy().astype('float32', copy=False)
    y = pd.to_numeric(df[target_col], errors='coerce').astype('float32')
    downcast_onehots_inplace(X)

    nun = X.nunique(dropna=True); mask = nun>1
    dropped = [c for c in X.columns if not mask[c]]
    if dropped:
        print(f"[Γ] Columnas constantes eliminadas: {len(dropped)} (ej: {dropped[:10]})")
    X = X.loc[:, mask].copy()

    X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=test_size, random_state=random_state, shuffle=True)

    #---------- NULO Gamma: media de----------
mu_te = float(np.mean(y_te.values))
mu_null_te = np.full(y_te.shape, mu_te, float)
dev_null_te = gamma_deviance(y_te.values, mu_null_te)
    results, info = {}, {}
    for mode in ['pre','full','post']:
        results[mode] = {}; info[mode] = {}
```

```
for rs in resampling_opts:
    if mode in ('pre','full') and rs!='none':
        cap_total = int(len(X_tr) * 1.3)
        Xb, yb = balance_data_gamma(X_tr, y_tr, method=rs, n_bins=resample_bins,
                                    factor=resample_factor, random_state=random_state,
                                    cap_total_rows=cap_total)
    else:
        Xb, yb = X_tr, y_tr

    hp = GM_HP[(mode, rs)]
    gt = GammaTree(
        Xb, yb, max_depth=hp['depth'], min_samples=hp['min_samples'],
        max_splits_per_feature=hp['max_splits']
    ).fit()

    if mode == 'post':
        rs_seed = _seed_for(mode, rs, base=random_state)
        Xp, Xv, yp, yv = train_test_split(Xb, yb, test_size=hp.get('prune_frac', 0.25),
                                          random_state=rs_seed, shuffle=True)
        if rs!='none':
            cap_total_v = int(len(Xv) * 1.3)
            Xv, yv = balance_data_gamma(
                Xv, yv, method=rs, n_bins=resample_bins,
                factor=resample_factor, random_state=rs_seed+1,
                cap_total_rows=cap_total_v
            )
        yhat_b = gt.predict(Xv); Dval_b = gamma_deviance(yv.values, yhat_b)
        gt.root = gamma_reduced_prune(gt.root, Xv, yv, tol=hp.get('tol', 0.0))
        yhat_a = gt.predict(Xv); Dval_a = gamma_deviance(yv.values, yhat_a)
        logger.info(f"[Γ {mode}/{rs}] REP Dev(val) {Dval_b:.1f}->{Dval_a:.1f}")

    gt._X = None; gt._y = None; gt._logy = None; gc.collect()
```

```
            #--------- Métricas en TEST (Pseudo D² con nulo)---------
            y_hat = gt.predict(X_te)
            dev = gamma_deviance(y_te.values, y_hat)
            pseudo = float(1.0- dev/dev_null_te) if dev_null_te>0 else float('nan')
            mae = float(np.mean(np.abs(y_te.values- y_hat)))
            rmse= float(np.sqrt(np.mean((y_te.values- y_hat)**2)))

            results[mode][rs] = {
               'Devianza Gamma': float(dev),
               'Pseudo D² Gamma (null = ybar_test)': pseudo,
               'MAE': mae, 'RMSE': rmse
            }
            info[mode][rs] = {'model':gt, 'columns': list(gt.columns), 'y_pred':y_hat}

    rows=[]
    for mode in results:
       for rs in results[mode]:
          rows.append({'mode':mode, 'resampling':rs, **results[mode][rs]})
    summary_df = pd.DataFrame(rows).sort_values(['mode','Devianza
Gamma']).reset_index(drop=True)
    print(f"\n[Γ] Resumen {summary_df.shape[0]} modelos (Pseudo D² con nulo de TRAIN):")
    print(summary_df)
    return info, results, X_tr, y_tr, X_te, y_te

# ================================================================
# 8) PP General (9 modelos) + WAPE por segmentos
# ================================================================
def _collapse_one_hot(df, prefix):
    cols = [c for c in df.columns if c.startswith(prefix)]
    if not cols: return pd.Series(['UNK']*len(df), index=df.index)
    arr = df[cols].to_numpy()
```

```
    if arr.size == 0: return pd.Series(['UNK']*len(df), index=df.index)
    sums = arr.sum(axis=1)
    idx = arr.argmax(axis=1)
    labels = [c.split('_',1)[1] if '_' in c else c for c in cols]
    out = np.where(sums>0, [labels[int(i)] for i in idx], 'UNK')
    return pd.Series(out, index=df.index)

def _safe_cut(df, col, bins, labels):
    if col in df.columns:
        ser = pd.to_numeric(df[col], errors='coerce')
        return pd.cut(ser, bins=bins, labels=labels)
    return pd.Series(['UNK']*len(df), index=df.index)

def _group_keys(df):
    out = pd.DataFrame(index=df.index)
    out['region']   = _collapse_one_hot(df, 'REGIAO_')
    out['sexo']     = np.where(df.get('SEXO_M',0)==1, 'M', np.where(df.get('SEXO_F',0)==1, 'F', 'UNK'))
    out['edad_bin'] = _safe_cut(df, 'IDADE', bins=[-1,24,34,44,54,64,200], labels=['<=24','25-34','35-
44','45-54','55-64','65+'])
    out['ano_bin']  = _safe_cut(df, 'ANO_MODELO', bins=[-1,2005,2010,2015,2020,2100],
labels=['<=2005','06-10','11-15','16-20','21+'])
    return [out[c] for c in out.columns]

def compute_pp_all_models(
    nb_info: Dict[str, Dict[str, Any]],
    gamma_info: Dict[str, Dict[str, Any]],
    X_universo: pd.DataFrame,
    modes=('pre','full','post'),
    ress=('none','under','over')
)-> Tuple[Dict[Tuple[str,str], Dict[str, Any]], pd.DataFrame]:
    """
    Combina NB y Gamma por (mode, rs) → 9 PP distintos sobre el MISMO universo (X_universo de
NB).
```

```
    """
    pp_info_all, rows = {}, []
    for mode in modes:
        for rs in ress:
            if (mode not in nb_info) or (rs not in nb_info[mode]): continue
            if (mode not in gamma_info) or (rs not in gamma_info[mode]): continue

            nb_blk = nb_info[mode][rs]
            gm_blk = gamma_info[mode][rs]

            mu_nb = np.asarray(nb_blk['y_pred'], float)
            assert len(mu_nb) == len(X_universo), f"X_universo debe alinear con y_pred NB para {mode}/{rs}."

            gm_cols = gm_blk['model'].columns if hasattr(gm_blk['model'], 'columns') else list(X_universo.columns)
            Xg = X_universo.reindex(columns=gm_cols, fill_value=0.0)
            sev = np.asarray(gm_blk['model'].predict(Xg), float)

            pp = mu_nb * sev
            df_out = pd.DataFrame({'mu_nb':mu_nb,'sev_gamma':sev,'PP':pp})
            if 'EXPOSICAO1' in X_universo.columns:
                df_out['EXPOSICAO1'] = pd.to_numeric(X_universo['EXPOSICAO1'], errors='coerce')

            key = (mode, rs)
            pp_info_all[key] = {'PP':pp, 'df':df_out, 'PP_sum':float(pp.sum()), 'PP_mean':float(pp.mean()), 'n':int(len(pp))}
            rows.append({'mode':mode, 'resampling':rs, 'PP_sum':float(pp.sum()), 'PP_mean':float(pp.mean()), 'n':int(len(pp))})
    return pp_info_all, pd.DataFrame(rows).sort_values(['mode','resampling']).reset_index(drop=True)

def evaluar_wape_calibrado_all(
    pp_info_all: Dict[Tuple[str,str], Dict[str, Any]],
```

```
    X_te: pd.DataFrame, y_te: pd.Series,
    severidad_csv: str, sev_col='severidad_promedio3', eps=1e-12
)-> Tuple[pd.DataFrame, Dict[Tuple[str,str], pd.DataFrame]]:
    # Observado por grupo (frecuencia y severidad real media)
    df_freq = X_te.copy()
    df_freq['N_obs'] = pd.to_numeric(pd.Series(y_te), errors='coerce').fillna(0.0).astype(float)
    N_g = df_freq.groupby(_group_keys(df_freq), dropna=False)['N_obs'].sum().rename('N_obs')

    try: sevd = pd.read_csv(severidad_csv, delimiter=';')
    except UnicodeDecodeError: sevd = pd.read_csv(severidad_csv, delimiter=';', encoding='latin-1')
    if sev_col not in sevd.columns: raise KeyError("Columna de severidad no encontrada.")
    sevd = sevd[pd.to_numeric(sevd[sev_col], errors='coerce') > 0].copy()
    sev_g = sevd.groupby(_group_keys(sevd), dropna=False)[sev_col].mean().rename('sev_obs')

    obs_g = N_g.to_frame().join(sev_g, how='left')
    obs_g['sev_obs'] = obs_g['sev_obs'].fillna(sev_g.mean())
    obs_g['EPP_obs'] = obs_g['N_obs'] * obs_g['sev_obs']

    resumen, cmp_tables = [], {}
    for key, blk in pp_info_all.items():
        mode, rs = key
        df_univ = X_te.copy()
        df_univ['EPP_pred'] = np.asarray(blk['PP'], float)
        pred_g = df_univ.groupby(_group_keys(df_univ), dropna=False)['EPP_pred'].sum()
        cmp = obs_g.join(pred_g, how='outer').fillna(0.0).reset_index()
        sum_obs, sum_pred = float(cmp['EPP_obs'].sum()), float(cmp['EPP_pred'].sum())
        k = (sum_obs/sum_pred) if sum_pred>0 else 1.0
        cmp['EPP_pred_cal'] = cmp['EPP_pred']*k
        cmp['abs_err_cal'] = np.abs(cmp['EPP_pred_cal']- cmp['EPP_obs'])
        WAPE_cal = cmp['abs_err_cal'].sum() / (sum_obs + eps)
        resumen.append({'mode':mode, 'resampling':rs, 'WAPE_cal':float(WAPE_cal), 'k_cal':float(k)})
        cmp['calib_ratio'] = np.where(cmp['EPP_obs']>0, cmp['EPP_pred_cal']/cmp['EPP_obs'], np.nan)
```

```
        cmp_tables[key] = cmp.sort_values('EPP_obs', ascending=False).reset_index(drop=True)
    return
pd.DataFrame(resumen).sort_values(['WAPE_cal','mode','resampling']).reset_index(drop=True),
cmp_tables


# ============================================================

# 9) Rutas y LISTAS COMPLETAS DE COLUMNAS (reutilizadas)

# ============================================================

FREQ_CSV = r"C:/Users/USUARIO/Desktop/Uni Tesis/Modelo
Final/Preprocesamiento/modelo_frecuencia.csv"

SEV_CSV  = r"C:/Users/USUARIO/Desktop/Uni Tesis/Modelo
Final/Preprocesamiento/modelo_severidad.csv"


COD_TARIF_COLS = [

    'COD_TARIF_1','COD_TARIF_2','COD_TARIF_3','COD_TARIF_4','COD_TARIF_5','COD_TARIF_6','COD
_TARIF_7','COD_TARIF_9'

]


# (LISTA GRANDE) COD_MODELO_COLS y REGIAO_COLS — deja tus listas tal cual:

COD_MODELO_COLS = [

    'COD_MODELO_1.0','COD_MODELO_10.0','COD_MODELO_100.0','COD_MODELO_101.0','COD_
MODELO_102.0','COD_MODELO_103.0',

    'COD_MODELO_104.0','COD_MODELO_105.0','COD_MODELO_106.0','COD_MODELO_107.0','CO
D_MODELO_108.0','COD_MODELO_109.0',

    'COD_MODELO_11.0','COD_MODELO_110.0','COD_MODELO_111.0','COD_MODELO_112.0','COD
_MODELO_113.0','COD_MODELO_114.0',

    'COD_MODELO_117.0','COD_MODELO_118.0','COD_MODELO_119.0','COD_MODELO_12.0','COD
_MODELO_120.0','COD_MODELO_122.0',

    'COD_MODELO_123.0','COD_MODELO_124.0','COD_MODELO_125.0','COD_MODELO_126.0','CO
D_MODELO_127.0','COD_MODELO_128.0',

    'COD_MODELO_129.0','COD_MODELO_13.0','COD_MODELO_130.0','COD_MODELO_131.0','COD
_MODELO_132.0','COD_MODELO_133.0',

    'COD_MODELO_134.0','COD_MODELO_135.0','COD_MODELO_136.0','COD_MODELO_137.0','CO
D_MODELO_138.0','COD_MODELO_139.0',

    'COD_MODELO_14.0','COD_MODELO_140.0','COD_MODELO_141.0','COD_MODELO_142.0','COD
_MODELO_143.0','COD_MODELO_144.0',
```

'COD_MODELO_146.0','COD_MODELO_147.0','COD_MODELO_148.0','COD_MODELO_149.0','COD_MODELO_15.0','COD_MODELO_150.0',

'COD_MODELO_151.0','COD_MODELO_152.0','COD_MODELO_153.0','COD_MODELO_154.0','COD_MODELO_155.0','COD_MODELO_156.0',

'COD_MODELO_157.0','COD_MODELO_158.0','COD_MODELO_159.0','COD_MODELO_16.0','COD_MODELO_160.0','COD_MODELO_161.0',

'COD_MODELO_162.0','COD_MODELO_163.0','COD_MODELO_164.0','COD_MODELO_166.0','COD_MODELO_168.0','COD_MODELO_169.0',

'COD_MODELO_17.0','COD_MODELO_170.0','COD_MODELO_171.0','COD_MODELO_173.0','COD_MODELO_174.0','COD_MODELO_175.0',

'COD_MODELO_176.0','COD_MODELO_177.0','COD_MODELO_178.0','COD_MODELO_179.0','COD_MODELO_18.0','COD_MODELO_180.0',

'COD_MODELO_181.0','COD_MODELO_182.0','COD_MODELO_183.0','COD_MODELO_184.0','COD_MODELO_185.0','COD_MODELO_186.0',

'COD_MODELO_187.0','COD_MODELO_188.0','COD_MODELO_189.0','COD_MODELO_19.0','COD_MODELO_190.0','COD_MODELO_191.0',

'COD_MODELO_192.0','COD_MODELO_193.0','COD_MODELO_194.0','COD_MODELO_195.0','COD_MODELO_196.0','COD_MODELO_198.0',

'COD_MODELO_199.0','COD_MODELO_20.0','COD_MODELO_200.0','COD_MODELO_201.0','COD_MODELO_202.0','COD_MODELO_203.0',

'COD_MODELO_204.0','COD_MODELO_205.0','COD_MODELO_206.0','COD_MODELO_209.0','COD_MODELO_210.0','COD_MODELO_211.0',

'COD_MODELO_212.0','COD_MODELO_213.0','COD_MODELO_214.0','COD_MODELO_215.0','COD_MODELO_217.0','COD_MODELO_22.0',

'COD_MODELO_220.0','COD_MODELO_222.0','COD_MODELO_223.0','COD_MODELO_224.0','COD_MODELO_225.0','COD_MODELO_226.0',

'COD_MODELO_227.0','COD_MODELO_228.0','COD_MODELO_229.0','COD_MODELO_23.0','COD_MODELO_230.0','COD_MODELO_231.0',

'COD_MODELO_232.0','COD_MODELO_233.0','COD_MODELO_235.0','COD_MODELO_236.0','COD_MODELO_237.0','COD_MODELO_238.0',

'COD_MODELO_24.0','COD_MODELO_240.0','COD_MODELO_241.0','COD_MODELO_242.0','COD_MODELO_243.0','COD_MODELO_244.0',

'COD_MODELO_245.0','COD_MODELO_246.0','COD_MODELO_247.0','COD_MODELO_248.0','COD_MODELO_249.0','COD_MODELO_25.0',

'COD_MODELO_250.0','COD_MODELO_251.0','COD_MODELO_252.0','COD_MODELO_253.0','COD_MODELO_255.0','COD_MODELO_256.0',

'COD_MODELO_257.0','COD_MODELO_258.0','COD_MODELO_259.0','COD_MODELO_26.0','COD_MODELO_260.0','COD_MODELO_261.0',

'COD_MODELO_262.0','COD_MODELO_263.0','COD_MODELO_264.0','COD_MODELO_265.0','COD_MODELO_266.0','COD_MODELO_267.0',

'COD_MODELO_268.0','COD_MODELO_269.0','COD_MODELO_27.0','COD_MODELO_270.0','COD_MODELO_271.0','COD_MODELO_272.0',

'COD_MODELO_273.0','COD_MODELO_274.0','COD_MODELO_275.0','COD_MODELO_276.0','COD_MODELO_277.0','COD_MODELO_278.0',

'COD_MODELO_279.0','COD_MODELO_28.0','COD_MODELO_280.0','COD_MODELO_281.0','COD_MODELO_282.0','COD_MODELO_283.0',

'COD_MODELO_286.0','COD_MODELO_287.0','COD_MODELO_289.0','COD_MODELO_29.0','COD_MODELO_290.0','COD_MODELO_291.0',

'COD_MODELO_292.0','COD_MODELO_293.0','COD_MODELO_294.0','COD_MODELO_295.0','COD_MODELO_296.0','COD_MODELO_297.0',

'COD_MODELO_298.0','COD_MODELO_3.0','COD_MODELO_30.0','COD_MODELO_300.0','COD_MODELO_301.0','COD_MODELO_302.0',

'COD_MODELO_303.0','COD_MODELO_305.0','COD_MODELO_306.0','COD_MODELO_307.0','COD_MODELO_308.0','COD_MODELO_31.0',

'COD_MODELO_310.0','COD_MODELO_311.0','COD_MODELO_312.0','COD_MODELO_313.0','COD_MODELO_314.0','COD_MODELO_315.0',

'COD_MODELO_316.0','COD_MODELO_317.0','COD_MODELO_318.0','COD_MODELO_319.0','COD_MODELO_32.0','COD_MODELO_321.0',

'COD_MODELO_322.0','COD_MODELO_323.0','COD_MODELO_324.0','COD_MODELO_325.0','COD_MODELO_326.0','COD_MODELO_327.0',

'COD_MODELO_328.0','COD_MODELO_329.0','COD_MODELO_33.0','COD_MODELO_330.0','COD_MODELO_333.0','COD_MODELO_334.0',

'COD_MODELO_335.0','COD_MODELO_336.0','COD_MODELO_338.0','COD_MODELO_339.0','COD_MODELO_34.0','COD_MODELO_340.0',

'COD_MODELO_342.0','COD_MODELO_343.0','COD_MODELO_344.0','COD_MODELO_345.0','COD_MODELO_346.0','COD_MODELO_347.0',

'COD_MODELO_348.0','COD_MODELO_349.0','COD_MODELO_35.0','COD_MODELO_350.0','COD_MODELO_351.0','COD_MODELO_352.0',

'COD_MODELO_353.0','COD_MODELO_354.0','COD_MODELO_355.0','COD_MODELO_356.0','COD_MODELO_357.0','COD_MODELO_358.0',

'COD_MODELO_359.0','COD_MODELO_36.0','COD_MODELO_360.0','COD_MODELO_362.0','COD_MODELO_365.0','COD_MODELO_366.0',

'COD_MODELO_367.0','COD_MODELO_368.0','COD_MODELO_37.0','COD_MODELO_370.0','COD_MODELO_374.0','COD_MODELO_375.0',

'COD_MODELO_376.0','COD_MODELO_377.0','COD_MODELO_378.0','COD_MODELO_379.0','COD_MODELO_380.0','COD_MODELO_381.0',

'COD_MODELO_382.0','COD_MODELO_383.0','COD_MODELO_384.0','COD_MODELO_385.0','COD_MODELO_386.0','COD_MODELO_387.0',

'COD_MODELO_388.0','COD_MODELO_389.0','COD_MODELO_390.0','COD_MODELO_391.0','COD_MODELO_392.0','COD_MODELO_393.0',

'COD_MODELO_394.0','COD_MODELO_395.0','COD_MODELO_396.0','COD_MODELO_397.0','COD_MODELO_398.0','COD_MODELO_399.0',

'COD_MODELO_4.0','COD_MODELO_40.0','COD_MODELO_400.0','COD_MODELO_401.0','COD_MODELO_402.0','COD_MODELO_403.0',

'COD_MODELO_404.0','COD_MODELO_405.0','COD_MODELO_406.0','COD_MODELO_407.0','COD_MODELO_408.0','COD_MODELO_409.0',

'COD_MODELO_41.0','COD_MODELO_410.0','COD_MODELO_411.0','COD_MODELO_412.0','COD_MODELO_413.0','COD_MODELO_414.0',

'COD_MODELO_415.0','COD_MODELO_416.0','COD_MODELO_417.0','COD_MODELO_418.0','COD_MODELO_419.0','COD_MODELO_42.0',

'COD_MODELO_420.0','COD_MODELO_421.0','COD_MODELO_422.0','COD_MODELO_423.0','COD_MODELO_424.0','COD_MODELO_425.0',

'COD_MODELO_426.0','COD_MODELO_427.0','COD_MODELO_428.0','COD_MODELO_429.0','COD_MODELO_430.0','COD_MODELO_431.0',

'COD_MODELO_432.0','COD_MODELO_433.0','COD_MODELO_434.0','COD_MODELO_435.0','COD_MODELO_436.0','COD_MODELO_437.0',

'COD_MODELO_438.0','COD_MODELO_439.0','COD_MODELO_44.0','COD_MODELO_441.0','COD_MODELO_442.0','COD_MODELO_443.0',

'COD_MODELO_444.0','COD_MODELO_445.0','COD_MODELO_446.0','COD_MODELO_447.0','COD_MODELO_448.0','COD_MODELO_449.0',

'COD_MODELO_45.0','COD_MODELO_450.0','COD_MODELO_451.0','COD_MODELO_452.0','COD_MODELO_454.0','COD_MODELO_455.0',

'COD_MODELO_456.0','COD_MODELO_457.0','COD_MODELO_458.0','COD_MODELO_459.0','COD_MODELO_46.0','COD_MODELO_460.0',

'COD_MODELO_461.0','COD_MODELO_463.0','COD_MODELO_466.0','COD_MODELO_467.0','COD_MODELO_469.0','COD_MODELO_470.0',

'COD_MODELO_471.0','COD_MODELO_472.0','COD_MODELO_473.0','COD_MODELO_474.0','COD_MODELO_475.0','COD_MODELO_476.0',

'COD_MODELO_477.0','COD_MODELO_478.0','COD_MODELO_479.0','COD_MODELO_48.0','COD_MODELO_480.0','COD_MODELO_481.0',

'COD_MODELO_482.0','COD_MODELO_484.0','COD_MODELO_485.0','COD_MODELO_486.0','COD_MODELO_487.0','COD_MODELO_488.0',

'COD_MODELO_489.0','COD_MODELO_49.0','COD_MODELO_490.0','COD_MODELO_491.0','COD_MODELO_492.0','COD_MODELO_493.0',

'COD_MODELO_494.0','COD_MODELO_495.0','COD_MODELO_496.0','COD_MODELO_497.0','COD_MODELO_498.0','COD_MODELO_499.0',

'COD_MODELO_50.0','COD_MODELO_500.0','COD_MODELO_501.0','COD_MODELO_502.0','COD_MODELO_503.0','COD_MODELO_504.0',

'COD_MODELO_505.0','COD_MODELO_506.0','COD_MODELO_507.0','COD_MODELO_510.0','COD_MODELO_511.0','COD_MODELO_512.0',

'COD_MODELO_513.0','COD_MODELO_514.0','COD_MODELO_515.0','COD_MODELO_516.0','COD_MODELO_517.0','COD_MODELO_518.0',

'COD_MODELO_519.0','COD_MODELO_520.0','COD_MODELO_521.0','COD_MODELO_522.0','COD_MODELO_523.0','COD_MODELO_524.0',

'COD_MODELO_525.0','COD_MODELO_526.0','COD_MODELO_527.0','COD_MODELO_528.0','COD_MODELO_529.0','COD_MODELO_531.0',

'COD_MODELO_532.0','COD_MODELO_533.0','COD_MODELO_534.0','COD_MODELO_535.0','COD_MODELO_536.0','COD_MODELO_537.0',

'COD_MODELO_538.0','COD_MODELO_539.0','COD_MODELO_54.0','COD_MODELO_540.0','COD_MODELO_541.0','COD_MODELO_542.0',

'COD_MODELO_543.0','COD_MODELO_544.0','COD_MODELO_545.0','COD_MODELO_546.0','COD_MODELO_547.0','COD_MODELO_548.0',

'COD_MODELO_549.0','COD_MODELO_55.0','COD_MODELO_550.0','COD_MODELO_551.0','COD_MODELO_552.0','COD_MODELO_553.0',

'COD_MODELO_554.0','COD_MODELO_555.0','COD_MODELO_556.0','COD_MODELO_557.0','COD_MODELO_558.0','COD_MODELO_559.0',

'COD_MODELO_56.0','COD_MODELO_560.0','COD_MODELO_561.0','COD_MODELO_562.0','COD_MODELO_57.0','COD_MODELO_59.0',

'COD_MODELO_6.0','COD_MODELO_61.0','COD_MODELO_62.0','COD_MODELO_63.0','COD_MODELO_64.0','COD_MODELO_65.0',

'COD_MODELO_66.0','COD_MODELO_67.0','COD_MODELO_68.0','COD_MODELO_69.0','COD_MODELO_7.0','COD_MODELO_70.0',

'COD_MODELO_71.0','COD_MODELO_72.0','COD_MODELO_73.0','COD_MODELO_74.0','COD_MODELO_77.0','COD_MODELO_78.0',

'COD_MODELO_79.0','COD_MODELO_8.0','COD_MODELO_80.0','COD_MODELO_82.0','COD_MODELO_83.0','COD_MODELO_84.0',

'COD_MODELO_85.0','COD_MODELO_86.0','COD_MODELO_87.0','COD_MODELO_88.0','COD_MODELO_89.0','COD_MODELO_9.0',

'COD_MODELO_90.0','COD_MODELO_91.0','COD_MODELO_92.0','COD_MODELO_93.0','COD_MODELO_94.0','COD_MODELO_95.0',

'COD_MODELO_96.0','COD_MODELO_97.0','COD_MODELO_98.0','COD_MODELO_99.0','COD_MODELO_nan'

]

```
REGIAO_COLS = [
    'REGIAO_.','REGIAO_0','REGIAO_00','REGIAO_01','REGIAO_02','REGIAO_03','REGIAO_04','REGIAO
_05',
    'REGIAO_06','REGIAO_07','REGIAO_08','REGIAO_09','REGIAO_1','REGIAO_10','REGIAO_11','REGI
AO_12',
    'REGIAO_13','REGIAO_14','REGIAO_15','REGIAO_16','REGIAO_17','REGIAO_18','REGIAO_19','REGI
AO_2',
    'REGIAO_20','REGIAO_21','REGIAO_22','REGIAO_23','REGIAO_24','REGIAO_25','REGIAO_26','REGI
AO_27',
    'REGIAO_28','REGIAO_29','REGIAO_3','REGIAO_30','REGIAO_31','REGIAO_32','REGIAO_33','REGI
AO_34',
    'REGIAO_35','REGIAO_36','REGIAO_37','REGIAO_38','REGIAO_39','REGIAO_4','REGIAO_40','REGI
AO_41',
    'REGIAO_5','REGIAO_6','REGIAO_7','REGIAO_8','REGIAO_9','REGIAO_99'
]

#-------- Lista COMPLETA para Frecuencia (NB)--------
columnas_seleccionadas = [
    'ANO_MODELO', 'IDADE', 'EXPOSICAO1', 'SEXO_F', 'SEXO_M',
    *COD_TARIF_COLS, *COD_MODELO_COLS, *REGIAO_COLS
]
#-------- Lista COMPLETA para Severidad (Gamma)--------
cols_pred = [
    'ANO_MODELO','IDADE','SEXO_F','SEXO_M',
    *COD_TARIF_COLS, *COD_MODELO_COLS, *REGIAO_COLS
]

# ============================================================
# 10) Ejecución: NB → Gamma → PP (9 modelos) → WAPE_cal
# ============================================================
if __name__ == "__main__":
    # 0) (opcional) Test GPU
    # test_gpu_windows_cuda125()
```

```
# 1) Frecuencia NB
nb_info, nb_results, X_tr, y_tr, X_te, y_te = run_nb_pipeline_explicit(
    FREQ_CSV, columnas_seleccionadas,
    target_col='FREQ_SIN3', expo_col='EXPOSICAO1',
    test_size=0.20, random_state=42,
    resampling_opts=('none','under','over'),
    resample_bins=8, resample_factor=0.05,
    alpha_prior=50.0
)

# 2) Severidad Gamma
gamma_info, gamma_results, GX_tr, Gy_tr, GX_te, Gy_te = run_gamma_pipeline_explicit(
    SEV_CSV, cols_pred,
    target_col='severidad_promedio3',
    test_size=0.20, random_state=42,
    resampling_opts=('none','under','over'),
    resample_bins=8, resample_factor=0.05
)

# 3) Prima Pura para los 9 modelos (mismo universo: X_te de NB)
pp_all, pp_summary = compute_pp_all_models(nb_info, gamma_info, X_universo=X_te)
print("\n[PP] Resumen PP_sum/PP_mean por modo y resampling:")
print(pp_summary)

# 4) WAPE_cal por segmentos para los 9 modelos
resumen_all, cmp_tables_all = evaluar_wape_calibrado_all(
    pp_all, X_te=X_te, y_te=y_te,
    severidad_csv=SEV_CSV, sev_col='severidad_promedio3'
)
print("\n[PP] Ranking global por WAPE_cal (menor es mejor):")
print(resumen_all)
```

```
    # 5) Top 15 segmentos para el mejor modelo (modo,rs) por EPP_obs
    best_mode = resumen_all.iloc[0]['mode']; best_rs = resumen_all.iloc[0]['resampling']
    print(f"\nTop 15 segmentos por EPP_obs (mejor modelo = {best_mode}/{best_rs}):")
    cols_show =
['region','sexo','edad_bin','ano_bin','EPP_obs','EPP_pred','EPP_pred_cal','calib_ratio']
    print(cmp_tables_all[(best_mode,best_rs)].head(15)[cols_show])


#-*- coding: utf-8-*-
# ================================================================
# 11) Diagnósticos completos (NB/Gamma) con opción GPU (CuPy) *segura*
#    - scatter/residuos (full data), importancia, LC (5 cortes), K-Fold
#    - Deviance NB/Gamma memory-safe (CPU) + opción GPU SIN cp.errstate
#    - Hotfix estimate_theta_offset_moments(..., r=...)
#    - Conmutador use_gpu en DIAG_CFG + kill-switch NB_FORCE_CPU
# ================================================================
import os, gc, math, warnings, logging
import numpy as np
import pandas as pd
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)

logger = logging.getLogger(__name__)
if not logger.handlers:
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

# Si el usuario decide forzar CPU en este kernel:
if os.environ.get("NB_FORCE_CPU") == "1":
    # Limpia variables heredadas que suelen causar ruido
    os.environ.pop("CUPY_NVRTC_PATH", None)
    os.environ.pop("CUPY_NVRTC_ARCH", None)

# ========================
```

```
# Ajustes globales seguros
# ========================
FLOATX_DEFAULT    = np.float32       # dtype preferido para devianzas (memoria)
NB_BLOCK_DEFAULT  = 300_000          # bloque NB (CPU)
GM_BLOCK_DEFAULT  = 300_000          # bloque Gamma (CPU)
GPU_BLOCK_DEFAULT = 2_000_000        # tamaño sugerido por bloque si se hace chunking en GPU

# ========================
# Utilidades GPU/CuPy
# ========================
def _init_gpu_env_for_cupy():
    """
    No fija NVRTC_PATH ni ARCH para evitar conflictos en kernels ya cargados.
    Solo deja activado PTX por defecto; CuPy elegirá lo que pueda.
    """
    os.environ.setdefault("CUPY_COMPILE_WITH_PTX", "1")

def _get_xp(use_gpu: bool = True):
    """
    Devuelve cupy si hay GPU y se solicita; si no, numpy.
    Respeta NB_FORCE_CPU=1. Hace una sonda mínima de cuBLAS/elementwise;
    si falla, vuelve a NumPy sin romper.
    """
    if (not use_gpu) or os.environ.get("NB_FORCE_CPU") == "1":
        return np
    try:
        _init_gpu_env_for_cupy()
        import cupy as cp
        if cp.cuda.runtime.getDeviceCount() <= 0:
            return np
        # Sonda mínima: operaciones básicas en GPU
        try:
```

```
            a = cp.arange(8, dtype=cp.float32)
            b = (a * a).sum()
            cp.cuda.Stream.null.synchronize()
            _ = float(cp.asnumpy(b))
            return cp
        except Exception as e:
            logger.warning(f"[GPU probe] {e!r}; usaré CPU.")
            return np
    except Exception as e:
        logger.warning(f"[GPU import] {e!r}; usaré CPU.")
        return np

def _is_cupy(xp):
    return (xp is not np)

def _xp_asarray(a, xp, dtype=None):
    if _is_cupy(xp):
        return xp.asarray(a, dtype=dtype) if dtype is not None else xp.asarray(a)
    return np.asarray(a, dtype=dtype) if dtype is not None else np.asarray(a)

def _xp_to_numpy(a):
    try:
        import cupy as cp
        if isinstance(a, cp.ndarray):
            return cp.asnumpy(a)
    except Exception:
        pass
    return np.asarray(a)

#---------------------------- Utils base----------------------------
def _ensure_dir(p):
    os.makedirs(p, exist_ok=True)
```

```
    return p

def _sample_idx(n, k, rng):
    if k is None or k >= n:
        return np.arange(n, dtype=int)
    return rng.choice(n, size=int(k), replace=False)

def _maybe_show(show: bool):
    if show:
        import matplotlib.pyplot as plt
        plt.show()

def _savefig_multi(fig, base_path: str, formats=('png','svg','pdf'), dpi=200, show=False):
    for ext in formats:
        fig.savefig(f"{base_path}.{ext}", bbox_inches="tight", dpi=dpi if ext=='png' else None)
    _maybe_show(show)
    import matplotlib.pyplot as plt
    plt.close(fig)

def _align_cols(df: pd.DataFrame, cols):
    """Alinea columnas (crea faltantes como 0.0 y reordena)."""
    df = df.copy()
    for c in cols:
        if c not in df.columns:
            df[c] = 0.0
    extra = [c for c in df.columns if c not in cols]
    if extra:
        df = df.drop(columns=extra)
    return df.loc[:, list(cols)]

def _seed_for(mode: str, rs: str, base: int = 42)-> int:
    return abs(hash((mode, rs, base))) % (2**31- 1)
```

```
#---------------------------- Split simple----------------------------
def train_test_split(X, y, test_size=0.25, random_state=42, shuffle=True):
    rng = np.random.default_rng(random_state)
    n = len(X)
    idx = np.arange(n, dtype=int)
    if shuffle:
        rng.shuffle(idx)
    n_test = int(round(n * float(test_size)))
    te = idx[:n_test]
    tr = idx[n_test:]
    return X.iloc[tr].copy(), X.iloc[te].copy(), y.iloc[tr].copy(), y.iloc[te].copy()

#---------------------------- Métricas de devianza (CPU/GPU, memory-safe)----------------------------
def _safe_log(x, eps=1e-12):
    return np.log(np.maximum(x, eps))

def _dev_nb_block_cpu(y, mu, th, start, end, eps=1e-12):
    yy = y[start:end]; mm = mu[start:end]
    t1 = np.where(yy > 0, yy * (np.log(np.maximum(yy, eps))- np.log(np.maximum(mm, eps))), 0.0)
    y_th = yy + th
    t2 = y_th * (np.log1p(yy / th)- np.log1p(mm / th))
    return float(2.0 * np.sum(t1- t2))

def dev_nb_xp(y, mu, theta, xp=np, dtype=None, block_size=None, gpu_block_size=None)-> float:
    """
    Deviance NB2 con memoria controlada.
   - CPU: procesa en bloques grandes para ahorrar RAM.
   - GPU (CuPy): intenta full-array; si falta VRAM, por bloques en VRAM.
    *No usa cp.errstate* para compatibilidad amplia de CuPy.
    """
    if dtype is None:
```

```
        dtype = FLOATX_DEFAULT
    if block_size is None:
        block_size = NB_BLOCK_DEFAULT
    if gpu_block_size is None:
        gpu_block_size = GPU_BLOCK_DEFAULT


    th = float(max(theta, 1e-9))
    if not _is_cupy(xp):
        #------- CPU (NumPy)-------
        try:
            y = np.asarray(y, dtype=dtype)
            mu = np.asarray(mu, dtype=dtype)
            n = y.size
            if n == 0:
                return 0.0
            if n <= block_size:
                with np.errstate(divide='ignore', invalid='ignore'):
                    t1 = np.where(y > 0, y * (_safe_log(y)- _safe_log(mu)), 0.0)
                    t2 = (y + th) * (np.log1p(y / th)- np.log1p(mu / th))
                    return float(2.0 * np.sum(t1- t2))
            D = 0.0
            for s in range(0, n, block_size):
                e = min(s + block_size, n)
                D += _dev_nb_block_cpu(y, mu, th, s, e)
            return float(D)
        except MemoryError:
            if dtype != np.float32 or block_size > 50_000:
                logger.warning("[dev_nb_xp CPU] MemoryError → reintento con float32 y bloques más
pequeños.")
                return dev_nb_xp(y, mu, theta, xp=np, dtype=np.float32, block_size=50_000)
            raise
    else:
```

```
#------- GPU (CuPy)-------
cp = xp  # alias
try:
    y_gpu = cp.asarray(y, dtype=cp.float32 if dtype==np.float32 else cp.float64)
    mu_gpu = cp.asarray(mu, dtype=cp.float32 if dtype==np.float32 else cp.float64)
    t1 = cp.where(y_gpu > 0,
                  y_gpu * (cp.log(cp.maximum(y_gpu, 1e-12))- cp.log(cp.maximum(mu_gpu, 1e-12))),
                  0.0)
    t2 = (y_gpu + th) * (cp.log1p(y_gpu / th)- cp.log1p(mu_gpu / th))
    D = 2.0 * (t1- t2).sum()
    cp.cuda.Stream.null.synchronize()
    return float(_xp_to_numpy(D))
except Exception as e:
    # Fallback por bloques en GPU (menor VRAM o kernels deshabilitados)
    if "OutOfMemoryError" not in repr(e) and "MemoryError" not in repr(e):
        logger.warning(f"[dev_nb_xp GPU] {repr(e)} → intento por bloques.")
    try:
        n = len(y)
        D_total = 0.0
        th_cp = float(th)
        for s in range(0, n, gpu_block_size):
            e = min(s + gpu_block_size, n)
            yg = cp.asarray(y[s:e], dtype=cp.float32 if dtype==np.float32 else cp.float64)
            mg = cp.asarray(mu[s:e], dtype=cp.float32 if dtype==np.float32 else cp.float64)
            t1 = cp.where(yg > 0,
                          yg * (cp.log(cp.maximum(yg, 1e-12))- cp.log(cp.maximum(mg, 1e-12))),
                          0.0)
            t2 = (yg + th_cp) * (cp.log1p(yg / th_cp)- cp.log1p(mg / th_cp))
            D = 2.0 * (t1- t2).sum()
            D_total += float(_xp_to_numpy(D))
            del yg, mg, D, t1, t2
            cp.get_default_memory_pool().free_all_blocks()
```

```
            cp.cuda.Stream.null.synchronize()
            return float(D_total)
        except Exception as e2:
            logger.warning(f"[dev_nb_xp GPU] Fallback a CPU por: {repr(e2)}")
            return dev_nb_xp(y, mu, theta, xp=np, dtype=dtype, block_size=block_size)


def _dev_gamma_block_cpu(y, mu, start, end, eps=1e-12):
    yy = y[start:end]; mm = mu[start:end]
    yy = np.maximum(yy, eps); mm = np.maximum(mm, eps)
    return float(2.0 * np.sum((yy- mm) / mm- np.log(yy / mm)))


def gamma_deviance(y, mu, xp=np, dtype=None, block_size=None, gpu_block_size=None)-> float:
    """Deviance Gamma (ϕ=1) con soporte CPU/GPU y procesamiento por bloques. Sin
cp.errstate."""
    if dtype is None:
        dtype = FLOATX_DEFAULT
    if block_size is None:
        block_size = GM_BLOCK_DEFAULT
    if gpu_block_size is None:
        gpu_block_size = GPU_BLOCK_DEFAULT


    if not _is_cupy(xp):
        #------- CPU-------
        try:
            y = np.asarray(y, dtype=dtype); mu = np.asarray(mu, dtype=dtype)
            n = y.size
            if n == 0:
                return 0.0
            if n <= block_size:
                y = np.maximum(y, 1e-12); mu = np.maximum(mu, 1e-12)
                D = 2.0 * np.sum((y- mu) / mu- np.log(y / mu))
                return float(D)
```

```
            D = 0.0
            for s in range(0, n, block_size):
                e = min(s + block_size, n)
                D += _dev_gamma_block_cpu(y, mu, s, e)
            return float(D)
        except MemoryError:
            if dtype != np.float32 or block_size > 50_000:
                logger.warning("[gamma_deviance CPU] MemoryError → reintento con float32 y bloques
pequeños.")
                return gamma_deviance(y, mu, xp=np, dtype=np.float32, block_size=50_000)
            raise
    else:
        #------- GPU (CuPy)-------
        cp = xp
        try:
            y_gpu = cp.asarray(y, dtype=cp.float32 if dtype==np.float32 else cp.float64)
            mu_gpu = cp.asarray(mu, dtype=cp.float32 if dtype==np.float32 else cp.float64)
            y_gpu = cp.maximum(y_gpu, 1e-12); mu_gpu = cp.maximum(mu_gpu, 1e-12)
            D = 2.0 * ((y_gpu- mu_gpu) / mu_gpu- cp.log(y_gpu / mu_gpu)).sum()
            cp.cuda.Stream.null.synchronize()
            return float(_xp_to_numpy(D))
        except Exception as e:
            # Fallback por bloques en GPU
            if "OutOfMemoryError" not in repr(e) and "MemoryError" not in repr(e):
                logger.warning(f"[gamma_deviance GPU] {repr(e)} → intento por bloques.")
            try:
                n = len(y)
                D_total = 0.0
                for s in range(0, n, gpu_block_size):
                    e2 = min(s + gpu_block_size, n)
                    yg = cp.asarray(y[s:e2], dtype=cp.float32 if dtype==np.float32 else cp.float64)
                    mg = cp.asarray(mu[s:e2], dtype=cp.float32 if dtype==np.float32 else cp.float64)
```

```
                yg = cp.maximum(yg, 1e-12); mg = cp.maximum(mg, 1e-12)
                D = 2.0 * ((yg- mg) / mg- cp.log(yg / mg)).sum()
                D_total += float(_xp_to_numpy(D))
                del yg, mg, D
                cp.get_default_memory_pool().free_all_blocks()
            cp.cuda.Stream.null.synchronize()
            return float(D_total)
        except Exception as e2:
            logger.warning(f"[gamma_deviance GPU] Fallback a CPU por: {repr(e2)}")
            return gamma_deviance(y, mu, xp=np, dtype=dtype, block_size=block_size)

def estimate_theta_offset_moments(y, expo, r=None, dtype=None)-> float:
    """
    Hotfix — acepta r explícito.
    Estima θ (NB2) por momentos con medias heterogéneas: Var(Y)=μ+μ^2/θ.
    μ_i = r * expo_i, con r:
      - si se pasa explícito, se usa ese,
     - si es None, r = sum(y)/sum(expo).
    """
    if dtype is None:
        dtype = FLOATX_DEFAULT
    y = np.asarray(y, dtype=dtype)
    expo = np.asarray(expo, dtype=dtype)
    expo = np.maximum(expo, 1e-12)

    if r is None or not np.isfinite(r) or r <= 0:
        r = float(np.sum(y)) / float(np.sum(expo))

    mu = r * expo
    num = float(np.sum(mu * mu))
    den = float(np.sum((y- mu) * (y- mu)- y))
    if den <= 1e-12 or not np.isfinite(den):
```

```
        return 10.0
    th = num / den
    th = max(min(float(th), 1e6), 1e-6)
    return th

#---------------------------- Balanceo (fallbacks simples)----------------------------
def balance_data_rate(X: pd.DataFrame, y: pd.Series, expo_col: str,
                      method: str, n_bins=8, factor=0.05, random_state=42,
                      pos_target=0.03, cap_dup=3, max_per_bin=None, cap_total_rows=None):
    rng = np.random.default_rng(random_state)
    y_arr = y.to_numpy()
    pos_idx = np.where(y_arr > 0)[0]
    neg_idx = np.where(y_arr <= 0)[0]
    n_pos = len(pos_idx)
    if method == 'under':
        n_neg_target = int(max(n_pos * (1.0- pos_target) / max(pos_target, 1e-6), 0))
        n_neg_target = min(n_neg_target, len(neg_idx))
        sel_neg = rng.choice(neg_idx, size=n_neg_target, replace=False)
        idx = np.concatenate([pos_idx, sel_neg])
        rng.shuffle(idx)
        Xb, yb = X.iloc[idx].copy(), y.iloc[idx].copy()
    elif method == 'over':
        Xb = X.copy(); yb = y.copy()
        if n_pos > 0:
            needed = int(max((len(yb) * pos_target) / max(1- pos_target, 1e-6)- n_pos, 0))
            if needed > 0:
                reps = min(cap_dup, max(1, needed // max(n_pos, 1)))
                dup_idx = np.repeat(pos_idx, reps)
                Xb = pd.concat([Xb, X.iloc[dup_idx]], ignore_index=True)
                yb = pd.concat([yb, y.iloc[dup_idx]], ignore_index=True)
    else:
        Xb, yb = X, y
```

```
    if cap_total_rows is not None and len(yb) > int(cap_total_rows):
        take = int(cap_total_rows)
        idx2 = rng.choice(len(yb), size=take, replace=False)
        Xb = Xb.iloc[idx2].copy(); yb = yb.iloc[idx2].copy()
    return Xb, yb


def balance_data_gamma(X: pd.DataFrame, y: pd.Series,
                       method: str, n_bins=8, factor=0.05,
                       random_state=42, cap_total_rows=None):
    rng = np.random.default_rng(random_state)
    if method == 'under':
        if cap_total_rows is None or len(y) <= cap_total_rows:
            return X, y
        q = pd.qcut(y.rank(method='first'), q=n_bins, labels=False, duplicates='drop')
        Xb_list, yb_list = [], []
        per_bin = int(math.floor(cap_total_rows / max(q.max()+1, 1)))
        for b in range(q.max()+1):
            idx_b = np.where(q == b)[0]
            if len(idx_b) <= per_bin:
                take_idx = idx_b
            else:
                take_idx = rng.choice(idx_b, size=per_bin, replace=False)
            Xb_list.append(X.iloc[take_idx]); yb_list.append(y.iloc[take_idx])
        Xb = pd.concat(Xb_list, ignore_index=True)
        yb = pd.concat(yb_list, ignore_index=True)
        return Xb, yb
    elif method == 'over':
        q = pd.qcut(y.rank(method='first'), q=n_bins, labels=False, duplicates='drop')
        tail_idx = np.where(q == q.max())[0]
        reps = max(1, int(round(factor * n_bins)))
        if len(tail_idx) > 0 and reps > 0:
```

```
            dup_idx = np.repeat(tail_idx, reps)
            Xb = pd.concat([X, X.iloc[dup_idx]], ignore_index=True)
            yb = pd.concat([y, y.iloc[dup_idx]], ignore_index=True)
            if cap_total_rows is not None and len(yb) > cap_total_rows:
                sel = rng.choice(len(yb), size=int(cap_total_rows), replace=False)
                Xb = Xb.iloc[sel].copy(); yb = yb.iloc[sel].copy()
            return Xb, yb
        return X, y
    else:
        return X, y

#---------------------------- Plots de dispersión/residuos----------------------------
def diag_scatter_and_residuals(y_true, y_pred, name, out_dir,
                               max_points=None, q_clip=None,
                               random_state=42, show=False,
                               equal_axes=True, dpi=200, save_formats=('png','svg','pdf')):
    import matplotlib.pyplot as plt
    _ensure_dir(out_dir)
    y_true = np.asarray(y_true, float)
    y_pred = np.asarray(y_pred, float)

    rng = np.random.default_rng(random_state)
    idx = _sample_idx(len(y_true), max_points, rng)  # None => todos
    yt, yp = y_true[idx], y_pred[idx]

    if q_clip is not None:
        hi = np.quantile(np.concatenate([yt, yp]), q_clip)
        yt = np.clip(yt, None, hi); yp = np.clip(yp, None, hi)

    # Pred vs Real
    fig1, ax1 = plt.subplots(figsize=(8, 6))
    ax1.scatter(yt, yp, s=1, alpha=0.20)
```

```
    lim_max = float(max(np.nanmax(yt), np.nanmax(yp), 1e-12))
    ax1.plot([0.0, lim_max], [0.0, lim_max], linewidth=1)
    ax1.set_xlabel("Observado (y)"); ax1.set_ylabel("Predicho (ŷ)")
    ax1.set_title(f"Pred vs. Real — {name}")
    if equal_axes:
        ax1.set_xlim(0.0, lim_max); ax1.set_ylim(0.0, lim_max); ax1.set_aspect('equal', 'box')
    _savefig_multi(fig1, os.path.join(out_dir, f"{name}_scatter"), formats=save_formats, dpi=dpi,
show=show)


    # Residuos vs Pred
    r = yt- yp
    fig2, ax2 = plt.subplots(figsize=(8, 6))
    ax2.scatter(yp, r, s=1, alpha=0.20)
    ax2.axhline(0.0, linewidth=1)
    ax2.set_xlabel("Predicho (ŷ)"); ax2.set_ylabel("Residuo (y − ŷ)")
    ax2.set_title(f"Residuos vs. Pred — {name}")
    _savefig_multi(fig2, os.path.join(out_dir, f"{name}_resid"), formats=save_formats, dpi=dpi,
show=show)


#---------------------------- Importancia de variables----------------------------
def nb_feature_importance(tree)-> pd.DataFrame:
    cols = list(tree.columns)
    gains = np.zeros(len(cols), float)
    splits = np.zeros(len(cols), int)
    stack = [tree.root]
    while stack:
        nd = stack.pop()
        if nd is None:
            continue
        if getattr(nd, "r", None) is None:  # nodo interno
            gains[int(nd.feat)] += float(getattr(nd, "gain", 0.0))
            splits[int(nd.feat)] += 1
            stack.append(nd.left); stack.append(nd.right)
```

```
    total = float(gains.sum()) if gains.sum() > 0 else 1.0
    df = pd.DataFrame({'feature': cols, 'gain': gains, 'gain_pct': gains/total, 'splits': splits})
    return df.sort_values('gain', ascending=False).reset_index(drop=True)

def gamma_feature_importance(tree)-> pd.DataFrame:
    cols = list(tree.columns)
    gains = np.zeros(len(cols), float)
    splits = np.zeros(len(cols), int)
    stack = [tree.root]
    while stack:
        nd = stack.pop()
        if nd is None:
            continue
        if getattr(nd, "pred", None) is None:  # nodo interno
            gains[int(nd.feat)] += float(getattr(nd, "gain", 0.0))
            splits[int(nd.feat)] += 1
            stack.append(nd.left); stack.append(nd.right)
    total = float(gains.sum()) if gains.sum() > 0 else 1.0
    df = pd.DataFrame({'feature': cols, 'gain': gains, 'gain_pct': gains/total, 'splits': splits})
    return df.sort_values('gain', ascending=False).reset_index(drop=True)

def plot_topk_importance(df_imp: pd.DataFrame, title: str, out_path_base: str, topk: int = 10,
                         show=False, dpi=200, save_formats=('png','svg','pdf')):
    import matplotlib.pyplot as plt
    top = df_imp.head(topk).iloc[::-1]
    fig, ax = plt.subplots(figsize=(10, 4 + 0.3*len(top)))
    ax.barh(top['feature'].astype(str), top['gain'])
    ax.set_xlabel("Ganancia total de devianza")
    ax.set_title(title)
    fig.tight_layout()
    _savefig_multi(fig, out_path_base, formats=save_formats, dpi=dpi, show=show)
```

```
#---------------------------- Plots LC y K-Fold----------------------------
def plot_learning_curve(df: pd.DataFrame, title: str, out_path_base: str,
                        show=False, dpi=200, save_formats=('png','svg','pdf')):
    import matplotlib.pyplot as plt
    fig, ax = plt.subplots(figsize=(8, 5))
    x = df['n_train'].values
    ax.plot(x, df['Dev_tr_mean'].values, marker='o', label='Entrenamiento')
    ax.plot(x, df['Dev_val_mean'].values, marker='o', label='Validación')
    ax.set_xlabel("n (entrenamiento)"); ax.set_ylabel("Devianza media")
    ax.set_title(title); ax.legend()
    _savefig_multi(fig, out_path_base, formats=save_formats, dpi=dpi, show=show)

def plot_kfold(df_cv: pd.DataFrame, summary: dict, title: str, out_path_base: str,
               show=False, dpi=200, save_formats=('png','svg','pdf')):
    import matplotlib.pyplot as plt
    fig, ax = plt.subplots(figsize=(8, 5))
    ax.bar(df_cv['fold'].astype(str), df_cv['Dev_val_mean'].values)
    ax.set_xlabel("Fold"); ax.set_ylabel("Devianza media (validación)")
    ax.set_title(title)
    txt = (f"k={summary.get('k', len(df_cv))}\n"
           f"Promedio={summary['Dev_val_mean_avg']:.6f}\n"
           f"Desvío={summary['Dev_val_mean_std']:.6f}")
    ax.text(0.98, 0.95, txt, transform=ax.transAxes,
            ha='right', va='top', bbox=dict(boxstyle='round', alpha=0.15))
    _savefig_multi(fig, out_path_base, formats=save_formats, dpi=dpi, show=show)

#---------------------------- Helpers de backoff (memoria)----------------------------
def _fit_nb_with_backoff(Xs, ys, hp, expo_col, alpha_prior, mode, rs, random_state):
    for attempt in range(3):
        try:
            nb = NBTreeOffset(
                Xs, ys, expo_col=expo_col,
```

```
                max_depth=hp['depth'], min_samples=hp['min_samples'],
                max_splits_per_feature=hp['max_splits'],
                min_gain_rel=hp.get('min_gain_rel', 0.0),
                min_expo=hp.get('min_expo', 0.0),
                alpha_prior=alpha_prior, r0_prior=None, verbose=False
            ).fit()
            return nb
        except MemoryError:
            if len(Xs) < 50_000:
                raise
            new_n = int(len(Xs) * 0.8)
            logger.warning(f"[NB fit] MemoryError ({mode}/{rs}). Backoff → {new_n} filas.")
            rng = np.random.default_rng(_seed_for(mode, rs, random_state))
            take = rng.choice(len(Xs), size=new_n, replace=False)
            Xs = Xs.iloc[take].copy(); ys = ys.iloc[take].copy()
            gc.collect()
    nb = NBTreeOffset(
        Xs, ys, expo_col=expo_col,
        max_depth=hp['depth'], min_samples=hp['min_samples'],
        max_splits_per_feature=hp['max_splits'],
        min_gain_rel=hp.get('min_gain_rel', 0.0),
        min_expo=hp.get('min_expo', 0.0),
        alpha_prior=alpha_prior, r0_prior=None, verbose=False
    ).fit()
    return nb

def _fit_gm_with_backoff(Xs, ys, hp, mode, rs, random_state):
    for attempt in range(3):
        try:
            gt = GammaTree(
                Xs, ys, max_depth=hp['depth'], min_samples=hp['min_samples'],
                max_splits_per_feature=hp['max_splits']
```

```
            ).fit()
            return gt
        except MemoryError:
            if len(Xs) < 50_000:
                raise
            new_n = int(len(Xs) * 0.8)
            logger.warning(f"[Gamma fit] MemoryError ({mode}/{rs}). Backoff → {new_n} filas.")
            rng = np.random.default_rng(_seed_for(mode, rs, random_state))
            take = rng.choice(len(Xs), size=new_n, replace=False)
            Xs = Xs.iloc[take].copy(); ys = ys.iloc[take].copy()
            gc.collect()
    gt = GammaTree(
        Xs, ys, max_depth=hp['depth'], min_samples=hp['min_samples'],
        max_splits_per_feature=hp['max_splits']
    ).fit()
    return gt


#---------------------------- Learning Curves (5 cortes)----------------------------
def learning_curve_nb_light(X_tr: pd.DataFrame, y_tr: pd.Series,
                            X_val: pd.DataFrame, y_val: pd.Series,
                            model_cols, mode: str, rs: str,
                            expo_col='EXPOSICAO1', alpha_prior=50.0,
                            fracs=(0.20, 0.40, 0.60, 0.80, 1.00),
                            cap=None, random_state=42,
                            resample_bins=8, resample_factor=0.05,
                            dev_xp=np)-> pd.DataFrame:
    try:
        NB_HP  # noqa
        NBTreeOffset  # noqa
    except NameError:
        raise RuntimeError("Se requieren NB_HP y NBTreeOffset definidos antes para LC NB.")
    try:
```

```
    reduced_error_pruning_offset  # noqa
except NameError:
    def reduced_error_pruning_offset(root, *args, **kwargs):
        return root

rng = np.random.default_rng(random_state)
results = []

theta_ref = estimate_theta_offset_moments(y_tr.values, X_tr[expo_col].to_numpy(float))
r_null = float(np.sum(y_tr.values)) / max(float(np.sum(X_tr[expo_col].to_numpy(float))), 1e-12)
mu_null_off = r_null * np.maximum(X_val[expo_col].to_numpy(float), 1e-10)
D0_val = dev_nb_xp(y_val.values, mu_null_off, theta_ref, xp=dev_xp)

for frac in fracs:
    n_target = max(1, int(round(len(X_tr)*frac)))
    if cap is not None:
        n_target = min(n_target, int(cap))
    idx = rng.choice(len(X_tr), size=n_target, replace=False)
    Xs = _align_cols(X_tr.iloc[idx], list(model_cols)).copy()
    ys = y_tr.iloc[idx].copy()

    if mode in ('pre','full') and rs != 'none':
        n_tr = len(Xs)
        max_per_bin = int(np.ceil(n_tr / resample_bins) * 1.2)
        cap_total   = int(n_tr * 1.3) if cap is None else min(int(n_tr * 1.3), int(cap))
        cap_dup_nb  = 3 if rs == 'over' else 1
        orig_pos = float((ys.values > 0).mean())
        pos_target = min(max(orig_pos*2.0, orig_pos), 0.03)
        Xs, ys = balance_data_rate(
            Xs, ys, expo_col, method=rs, n_bins=resample_bins, factor=resample_factor,
            random_state=random_state, pos_target=pos_target, cap_dup=cap_dup_nb,
            max_per_bin=max_per_bin, cap_total_rows=cap_total
```

```
        )

    hp = NB_HP[(mode, rs)]
    nb = _fit_nb_with_backoff(Xs, ys, hp, expo_col, alpha_prior, mode, rs, random_state)

    if mode == 'post':
        rs_seed = _seed_for(mode, rs, base=random_state)
        Xp, Xv, yp, yv = train_test_split(Xs, ys, test_size=hp.get('prune_frac', 0.25),
                                          random_state=rs_seed, shuffle=True)
        theta_pr = estimate_theta_offset_moments(yv.values, Xv[expo_col].to_numpy(float))
        nb.root = reduced_error_pruning_offset(
            nb.root, Xv, yv, theta_pr, expo_col=expo_col, alpha_prior=alpha_prior,
            r0_prior=None, rate_cap=np.inf, tol=hp.get('tol', 0.0)
        )

    yhat_tr = nb.predict(Xs)
    D_tr = dev_nb_xp(ys.values, yhat_tr, theta_ref, xp=dev_xp)
    X_val_al = _align_cols(X_val, list(nb.columns))
    yhat_val = nb.predict(X_val_al)
    D_val = dev_nb_xp(y_val.values, yhat_val, theta_ref, xp=dev_xp)

    results.append({
        'frac': float(frac), 'n_train': int(len(Xs)),
        'Dev_tr_mean': float(D_tr)/max(len(ys),1),
        'Dev_val_mean': float(D_val)/max(len(y_val),1),
        'PseudoD2_val_off': float(1.0- (D_val/D0_val)) if D0_val > 0 else np.nan
    })
    nb._X = nb._y = nb._expo = None
    gc.collect()

return pd.DataFrame(results).sort_values('n_train').reset_index(drop=True)
```

```
def learning_curve_gamma_light(X_tr: pd.DataFrame, y_tr: pd.Series,
                               X_val: pd.DataFrame, y_val: pd.Series,
                               model_cols, mode: str, rs: str,
                               fracs=(0.20, 0.40, 0.60, 0.80, 1.00),
                               cap=None, random_state=42,
                               resample_bins=8, resample_factor=0.05,
                               dev_xp=np)-> pd.DataFrame:
    try:
        GM_HP  # noqa
        GammaTree  # noqa
    except NameError:
        raise RuntimeError("Se requieren GM_HP y GammaTree definidos antes para LC Gamma.")
    try:
        gamma_reduced_prune  # noqa
    except NameError:
        def gamma_reduced_prune(root, *args, **kwargs):
            return root

    rng = np.random.default_rng(random_state)
    results = []

    mu0 = float(np.mean(y_val.values))
    D0_val = gamma_deviance(y_val.values, np.full(y_val.shape, mu0, float), xp=dev_xp)

    for frac in fracs:
        n_target = max(1, int(round(len(X_tr)*frac)))
        if cap is not None:
            n_target = min(n_target, int(cap))
        idx = rng.choice(len(X_tr), size=n_target, replace=False)
        Xs = _align_cols(X_tr.iloc[idx], list(model_cols)).copy()
        ys = y_tr.iloc[idx].copy()
```

```
if mode in ('pre','full') and rs != 'none':
    cap_total = int(len(Xs)*1.3) if cap is None else min(int(len(Xs)*1.3), int(cap))
    Xs, ys = balance_data_gamma(
        Xs, ys, method=rs, n_bins=resample_bins,
        factor=resample_factor, random_state=random_state,
        cap_total_rows=cap_total
    )

hp = GM_HP[(mode, rs)]
gt = _fit_gm_with_backoff(Xs, ys, hp, mode, rs, random_state)

if mode == 'post':
    rs_seed = _seed_for(mode, rs, base=random_state)
    Xp, Xv, yp, yv = train_test_split(Xs, ys, test_size=hp.get('prune_frac', 0.25),
                                      random_state=rs_seed, shuffle=True)
    gt.root = gamma_reduced_prune(gt.root, Xv, yv, tol=hp.get('tol', 0.0))

yhat_tr = gt.predict(Xs)
D_tr = gamma_deviance(ys.values, yhat_tr, xp=dev_xp)
X_val_al = _align_cols(X_val, list(gt.columns))
yhat_val = gt.predict(X_val_al)
D_val = gamma_deviance(y_val.values, yhat_val, xp=dev_xp)

results.append({
    'frac': float(frac), 'n_train': int(len(Xs)),
    'Dev_tr_mean': float(D_tr)/max(len(ys),1),
    'Dev_val_mean': float(D_val)/max(len(y_val),1),
    'PseudoD2_val': float(1.0- (D_val/D0_val)) if D0_val > 0 else np.nan
})
gt._X = gt._y = gt._logy = None
gc.collect()
```

```
    return pd.DataFrame(results).sort_values('n_train').reset_index(drop=True)

#---------------------------- K-Fold----------------------------
def kfold_cv_nb_light(X: pd.DataFrame, y: pd.Series, model_cols, mode: str, rs: str,
                      expo_col='EXPOSICAO1', k=3, cap=None, random_state=42,
                      resample_bins=8, resample_factor=0.05, dev_xp=np):
    try:
        NB_HP  # noqa
        NBTreeOffset  # noqa
    except NameError:
        raise RuntimeError("Se requieren NB_HP y NBTreeOffset definidos antes para K-Fold NB.")

    rng = np.random.default_rng(random_state)
    idx = np.arange(len(X), dtype=int)
    rng.shuffle(idx)
    folds = np.array_split(idx, int(k))

    rows = []
    for i in range(int(k)):
        val_idx = folds[i]
        tr_idx = np.concatenate([folds[j] for j in range(int(k)) if j != i])
        if cap is not None and len(tr_idx) > int(cap):
            tr_idx = rng.choice(tr_idx, size=int(cap), replace=False)

        Xtr = _align_cols(X.iloc[tr_idx], list(model_cols)).copy()
        ytr = y.iloc[tr_idx].copy()

        if mode in ('pre','full') and rs != 'none':
            n_tr = len(Xtr)
            max_per_bin = int(np.ceil(n_tr / resample_bins) * 1.2)
            cap_total   = int(n_tr * 1.3) if cap is None else min(int(n_tr * 1.3), int(cap))
            cap_dup_nb  = 3 if rs == 'over' else 1
```

```
            orig_pos = float((ytr.values > 0).mean())
            pos_target = min(max(orig_pos*2.0, orig_pos), 0.03)
            Xtr, ytr = balance_data_rate(
                Xtr, ytr, expo_col, method=rs, n_bins=resample_bins, factor=resample_factor,
                random_state=random_state, pos_target=pos_target, cap_dup=cap_dup_nb,
                max_per_bin=max_per_bin, cap_total_rows=cap_total
            )

        hp = NB_HP[(mode, rs)]
        nb = _fit_nb_with_backoff(Xtr, ytr, hp, expo_col, 50.0, mode, rs, random_state)

        Xval = _align_cols(X.iloc[val_idx], list(nb.columns))
        yval = y.iloc[val_idx].values
        theta_val = estimate_theta_offset_moments(yval, X.iloc[val_idx][expo_col].to_numpy(float))
        yhat_val = nb.predict(Xval)
        D_val = dev_nb_xp(yval, yhat_val, theta_val, xp=dev_xp)

        rows.append({'fold': i+1, 'n_train': int(len(Xtr)), 'n_val': int(len(val_idx)),
                     'Dev_val_mean': float(D_val)/max(len(val_idx),1)})

        nb._X = nb._y = nb._expo = None
        gc.collect()

    df = pd.DataFrame(rows)
    summary = {
        'k': int(k),
        'Dev_val_mean_avg': float(df['Dev_val_mean'].mean()),
        'Dev_val_mean_std': float(df['Dev_val_mean'].std(ddof=1)) if len(df)>1 else 0.0
    }
    return df, summary

def kfold_cv_gamma_light(X: pd.DataFrame, y: pd.Series, model_cols, mode: str, rs: str,
```

```
                k=3, cap=None, random_state=42,
                resample_bins=8, resample_factor=0.05, dev_xp=np):
try:
    GM_HP  # noqa
    GammaTree  # noqa
except NameError:
    raise RuntimeError("Se requieren GM_HP y GammaTree definidos antes para K-Fold Gamma.")

rng = np.random.default_rng(random_state)
idx = np.arange(len(X), dtype=int)
rng.shuffle(idx)
folds = np.array_split(idx, int(k))

rows = []
for i in range(int(k)):
    val_idx = folds[i]
    tr_idx = np.concatenate([folds[j] for j in range(int(k)) if j != i])
    if cap is not None and len(tr_idx) > int(cap):
        tr_idx = rng.choice(tr_idx, size=int(cap), replace=False)

    Xtr = _align_cols(X.iloc[tr_idx], list(model_cols)).copy()
    ytr = y.iloc[tr_idx].copy()

    if mode in ('pre','full') and rs != 'none':
        cap_total = int(len(Xtr)*1.3) if cap is None else min(int(len(Xtr)*1.3), int(cap))
        Xtr, ytr = balance_data_gamma(
            Xtr, ytr, method=rs, n_bins=resample_bins,
            factor=resample_factor, random_state=random_state,
            cap_total_rows=cap_total
        )

    hp = GM_HP[(mode, rs)]
```

```
        gt = _fit_gm_with_backoff(Xtr, ytr, hp, mode, rs, random_state)

        Xval = _align_cols(X.iloc[val_idx], list(gt.columns))
        yval = y.iloc[val_idx].values
        yhat_val = gt.predict(Xval)
        D_val = gamma_deviance(yval, yhat_val, xp=dev_xp)

        rows.append({'fold': i+1, 'n_train': int(len(Xtr)), 'n_val': int(len(val_idx)),
                     'Dev_val_mean': float(D_val)/max(len(val_idx),1)})

        gt._X = gt._y = gt._logy = None
        gc.collect()

    df = pd.DataFrame(rows)
    summary = {
        'k': int(k),
        'Dev_val_mean_avg': float(df['Dev_val_mean'].mean()),
        'Dev_val_mean_std': float(df['Dev_val_mean'].std(ddof=1)) if len(df)>1 else 0.0
    }
    return df, summary

#---------------------------- Orquestador----------------------------
def run_diagnostics_all_models(
    nb_info, gamma_info,
    X_tr, y_tr, X_te, y_te,
    GX_tr, Gy_tr, GX_te, Gy_te,
    cfg: dict
):
    #--- Selección NumPy/CuPy para devianzas grandes---
    use_gpu = bool(cfg.get('use_gpu', True))
    DEV_XP = _get_xp(use_gpu=use_gpu)
```

```
#--- Monkey-patch (mismas firmas) para asegurar versiones memory-safe---
g = globals()
g['dev_nb_xp'] = dev_nb_xp
g['gamma_deviance'] = gamma_deviance
g['estimate_theta_offset_moments'] = estimate_theta_offset_moments
#----------------------------------------------------------------------

modes = ('pre','full','post')
ress  = ('none','under','over')
out_root = _ensure_dir(cfg.get('out_dir', 'diagnostics'))
out_nb = _ensure_dir(os.path.join(out_root, 'nb'))
out_gm = _ensure_dir(os.path.join(out_root, 'gm'))

max_points = cfg.get('max_plot_points', None)   # None => todos
q_clip = cfg.get('q_clip', None)               # None => sin recorte
rs_seed = int(cfg.get('random_state', 42))
show_plots = bool(cfg.get('show_plots', True))
topk = int(cfg.get('topk_importance', 10))
save_formats = tuple(cfg.get('save_formats', ('png','svg','pdf')))
dpi = int(cfg.get('dpi', 200))
lc_cap = cfg.get('lc_cap', None)
cv_cap = cfg.get('cv_cap', None)
# >>> 5 cortes por defecto 20%→100% (uniformes)
lc_fracs = tuple(cfg.get('lc_fracs', tuple(np.linspace(0.20, 1.00, 5))))

for mode in modes:
    for rs in ress:
        if (mode not in nb_info) or (rs not in nb_info[mode]): continue
        if (mode not in gamma_info) or (rs not in gamma_info[mode]): continue

        #---------- NB----------
        nb_blk   = nb_info[mode][rs]
```

```
nb_model = nb_blk['model']
yhat_nb  = np.asarray(nb_blk['y_pred'], float)
base_nb  = f"nb_{mode}_{rs}"

diag_scatter_and_residuals(
   y_true=y_te.values, y_pred=yhat_nb, name=base_nb, out_dir=out_nb,
   max_points=max_points, q_clip=q_clip, random_state=rs_seed, show=show_plots,
   dpi=dpi, save_formats=save_formats
)
imp_nb = nb_feature_importance(nb_model)
imp_nb.to_csv(os.path.join(out_nb, f"{base_nb}_importance_full.csv"), index=False)
plot_topk_importance(imp_nb, title=f"Importancia (Top-{topk}) NB {mode}/{rs}",
                out_path_base=os.path.join(out_nb, f"{base_nb}_importance_top{topk}"),
                topk=topk, show=show_plots, dpi=dpi, save_formats=save_formats)

#---------- Gamma----------
gm_blk   = gamma_info[mode][rs]
gm_model = gm_blk['model']
yhat_gm  = np.asarray(gm_blk['y_pred'], float)
base_gm  = f"gm_{mode}_{rs}"

diag_scatter_and_residuals(
   y_true=Gy_te.values, y_pred=yhat_gm, name=base_gm, out_dir=out_gm,
   max_points=max_points, q_clip=q_clip, random_state=rs_seed, show=show_plots,
   dpi=dpi, save_formats=save_formats
)
imp_gm = gamma_feature_importance(gm_model)
imp_gm.to_csv(os.path.join(out_gm, f"{base_gm}_importance_full.csv"), index=False)
plot_topk_importance(imp_gm, title=f"Importancia (Top-{topk}) Γ {mode}/{rs}",
                out_path_base=os.path.join(out_gm, f"{base_gm}_importance_top{topk}"),
                topk=topk, show=show_plots, dpi=dpi, save_formats=save_formats)
```

```
#---------- Curvas de aprendizaje (usa DEV_XP para devianzas)----------
if cfg.get('lc_do', True):
    lc_nb = learning_curve_nb_light(
        X_tr=_align_cols(X_tr, list(nb_model.columns)),
        y_tr=y_tr,
        X_val=_align_cols(X_te, list(nb_model.columns)),
        y_val=y_te,
        model_cols=list(nb_model.columns),
        mode=mode, rs=rs, expo_col='EXPOSICAO1',
        alpha_prior=50.0, fracs=lc_fracs,
        cap=lc_cap, random_state=rs_seed,
        dev_xp=DEV_XP
    )
    lc_nb.to_csv(os.path.join(out_nb, f"{base_nb}_learning_curve.csv"), index=False)
    plot_learning_curve(lc_nb, title=f"LC NB {mode}/{rs}",
                        out_path_base=os.path.join(out_nb, f"{base_nb}_learning_curve"),
                        show=show_plots, dpi=dpi, save_formats=save_formats)

    lc_gm = learning_curve_gamma_light(
        X_tr=_align_cols(GX_tr, list(gm_model.columns)),
        y_tr=Gy_tr,
        X_val=_align_cols(GX_te, list(gm_model.columns)),
        y_val=Gy_te,
        model_cols=list(gm_model.columns),
        mode=mode, rs=rs, fracs=lc_fracs,
        cap=lc_cap, random_state=rs_seed,
        dev_xp=DEV_XP
    )
    lc_gm.to_csv(os.path.join(out_gm, f"{base_gm}_learning_curve.csv"), index=False)
    plot_learning_curve(lc_gm, title=f"LC Γ {mode}/{rs}",
                        out_path_base=os.path.join(out_gm, f"{base_gm}_learning_curve"),
                        show=show_plots, dpi=dpi, save_formats=save_formats)
```

```
        #---------- Validación cruzada (usa DEV_XP para devianzas)----------
        if cfg.get('cv_do', True):
            cv_nb_df, cv_nb_sum = kfold_cv_nb_light(
                X=_align_cols(X_tr, list(nb_model.columns)), y=y_tr,
                model_cols=list(nb_model.columns), mode=mode, rs=rs, expo_col='EXPOSICAO1',
                k=int(cfg.get('cv_k', 3)), cap=cv_cap, random_state=rs_seed,
                dev_xp=DEV_XP
            )
            cv_nb_df.to_csv(os.path.join(out_nb, f"{base_nb}_cv.csv"), index=False)
            with open(os.path.join(out_nb, f"{base_nb}_cv_summary.txt"), 'w', encoding='utf-8') as f:
                f.write(str(cv_nb_sum))
            plot_kfold(cv_nb_df, cv_nb_sum, title=f"K-Fold NB {mode}/{rs}",
                       out_path_base=os.path.join(out_nb, f"{base_nb}_cv"),
                       show=show_plots, dpi=dpi, save_formats=save_formats)

            cv_gm_df, cv_gm_sum = kfold_cv_gamma_light(
                X=_align_cols(GX_tr, list(gm_model.columns)), y=Gy_tr,
                model_cols=list(gm_model.columns), mode=mode, rs=rs,
                k=int(cfg.get('cv_k', 3)), cap=cv_cap, random_state=rs_seed,
                dev_xp=DEV_XP
            )
            cv_gm_df.to_csv(os.path.join(out_gm, f"{base_gm}_cv.csv"), index=False)
            with open(os.path.join(out_gm, f"{base_gm}_cv_summary.txt"), 'w', encoding='utf-8') as f:
                f.write(str(cv_gm_sum))
            plot_kfold(cv_gm_df, cv_gm_sum, title=f"K-Fold Γ {mode}/{rs}",
                       out_path_base=os.path.join(out_gm, f"{base_gm}_cv"),
                       show=show_plots, dpi=dpi, save_formats=save_formats)

        gc.collect()

# ============================================================
```

```
# 12) Ejecución de diagnósticos por los 9 modelos (full data)
#     Nota: NB_HP / GM_HP, NBTreeOffset / GammaTree, nb_info/gamma_info
#           y splits deben existir en el entorno (del script principal).
# ================================================================
if __name__ == "__main__":
    # Ejemplo de configuración (ajusta según tu entorno)
    DIAG_CFG = {
        'out_dir': "diagnostics",
        'max_plot_points': None,        # None => usa TODOS los puntos
        'q_clip': None,                 # None => sin recorte superior
        'lc_do': True,                  # curvas de aprendizaje
        # >>> 5 cortes uniformes 20%→100%
        'lc_fracs': tuple(np.linspace(0.20, 1.00, 5)),
        # >>> Control de tamaño de muestra (ajusta según RAM/VRAM)
        'lc_cap': None,                 # p.ej., 500_000 para limitar filas por punto LC
        'cv_do': True,                  # K-fold
        'cv_k': 3,
        'cv_cap': None,                 # p.ej., 600_000 para limitar filas por fold
        'topk_importance': 10,
        'random_state': 42,
        'show_plots': True,             # muestra los plots en Jupyter
        'save_formats': ('png','svg','pdf'),
        'dpi': 200,
        # >>> Por compatibilidad en este kernel: dejar False.
        #     Cuando actualices CUDA/CuPy y reinicies kernel, puedes pasar a True.
        'use_gpu': False
    }

    # Requisitos previos en el entorno:
    #- nb_info, gamma_info (dict por (mode,rs) con ['model','y_pred'])
    #- X_tr, y_tr, X_te, y_te, GX_tr, Gy_tr, GX_te, Gy_te
    #- NB_HP / GM_HP y clases NBTreeOffset / GammaTree para LC/K-Fold
```

```
run_diagnostics_all_models(
    nb_info, gamma_info,
    X_tr=X_tr, y_tr=y_tr, X_te=X_te, y_te=y_te,
    GX_tr=GX_tr, Gy_tr=Gy_tr, GX_te=GX_te, Gy_te=Gy_te,
    cfg=DIAG_CFG
```