Proyecto Final:

Análisis de Sobrevivencia Bayesiano

Christian Francisco Badillo Hernández

Tabla de contenidos

1 Introducción

El análisis de supervivencia es una rama de la estadística que se ocupa del tiempo hasta que ocurre un evento de interés. Este tipo de análisis es fundamental en diversas disciplinas, como la medicina, la ingeniería y las ciencias sociales. En este proyecto, nos enfocaremos en el análisis de supervivencia utilizando métodos bayesianos, que ofrecen una forma flexible y robusta para modelar datos censurados y realizar inferencias. Los datos utilizados son de las comorbilidades de los pacientes con Covid-19 (SARS-CoV-2) y su relación con la mortalidad. El objetivo principal es aplicar modelos de riesgos proporcionales tabto paramétricos como semiparamétricos, y comparar su rendimiento utilizando métodos como LOO-CV y WAIC, que se han convertido en estándares en la evaluación de modelos bayesianos.

1.1 Metodología.

Los datos se obtuvieron del sitio web de Datos Abiertos de la Dirección General de Epidemiología del Gobierno de México en específico del historial registrado para el año 2022. La base contiene información de distintas enfermedades respiratorias, comorbilidades con otras enfermedades crónicas y variables sociodemográficas de los pacientes y la unidad médica donde fueron atendidos.

Para el análisis se filtraron los casos con Covid-19 que fueron confirmados ya sea por un comite experto o con una prueba PSR positiva. Se eliminaron los casos que no contaban con la fecha de inicio de síntomas. El total de pacientes atendidos en un centro de salud por Covid-19 fueron $3,195,409$ casos de los cuales $26,108$ fueron casos mortales dando así una tasa de mortalidad de 0.0082 o 0.82%.

Dada la gran cantidad de datos, se decidio realizar un muestreo aleatorio con 3 estratificaciones: `sexo` (hombre, mujer), `tipo_de_paciente` (si el paciente fue hospitalizado) e `intubado` (si el paciente requirío intubación). Dado que fueron las variables con mayor desbalance en la proporción de cases mortales con respecto a los no mortales. Se obtuvo una muestra de $5,000$ pacientes, de los cuales $1,000$ fueron casos mortales y $4,000$ no mortales.

Incluso con el muestreo estratificado las estimaciones de los modelos para la variable `tipo_de_paciente` eran inestables (intervalo de confianza/credibilidad superior infinito o muy grande), por lo que se decidio eliminar esta variable del análisis. Las variables que se incluyeron en el análisis como regresoras fueron las siguientes:

- `sexo`: Sexo del paciente (hombre, mujer).
- `intubado`: Si el paciente requirió intubación (si, no).
- `neumonia`: Si el paciente presentó neumonía (si, no).
- `edad`: Edad del paciente (en años).
- `diabetes`: Si el paciente tiene diabetes (si, no).
- `epoc`: Si el paciente tiene enfermedad pulmonar obstructiva crónica (si, no).
- `asma`: Si el paciente tiene asma (si, no).
- `inmusupr`: Si el paciente tiene inmunosupresión (si, no).
- `hipertension`: Si el paciente tiene hipertensión (si, no).
- `otra_com`: Si el paciente tiene otra comorbilidad (si, no).
- `cardiovascular`: Si el paciente tiene enfermedad cardiovascular (si, no).

- `obesidad`: Si el paciente tiene obesidad (si, no).
- `renal_cronica`: Si el paciente tiene enfermedad renal crónica (si, no).
- `tabaquismo`: Si el paciente es fumador (si, no).
- `otro_caso`: Si el paciente tuvo contacto con un caso confirmado de Covid-19 (si, no).

Para el tiempo de inicio se tomo como referencia la fecha de inicio de síntomas y como evento el fallecimiento del paciente. En caso de que el paciente no falleciera se consideró como censura el tiempo hasta la primera semana de diciembre de 2022. Los casos a partir de la 2 semana de diciembre no fueron considerados. El tiempo se toma en días (345 el máximo), dado que no se pueden obtener tiempos de 0 días se reemplzaron con el valor 0.01 para el análisis.

Para el modelamiento se uso el software Stan (Sampling through adaptive neighborhoods), que es el lenguaje de programación probabilístico de estado del arte para la estimación de modelos bayesianos. El sofware utiliza el algoritmo de Monte Carlo Halmitoniano (HMC) junto con la implementación de NUTS (No U-Turn Sampler) para la estimación de los parámetros del modelo. El software Stan es ampliamente utilizado en la comunidad estadística y ha demostrado ser eficiente y preciso en la estimación de modelos complejos. La versión utilizada fue la más reciente (`2.36.0`) en su implementación `cmdstanr` para R.

Los modelos paramétricos usaron 1000 muestreos con 4 cadenas y un calentamiento (*warmup*) de 1000, y en el caso del modelo semiparamétrico se uso 2000 muestreos con 4 cadenas y un calentamiento de 1000, siguiendo la guía estandar del uso del sofware (Gelman et al., 2013). Los modelos escritos en Stan se presentan en los apéndices.

Para la comparación de los modelos se uso la validación cruzada Leave-One-Out (LOO-CV), que es un método robusto para estimar la capacidad predictiva de un modelo al dejar fuera una observación a la vez y evaluar su rendimiento en función de las observaciones restantes (Vehtari et al., 2015, 2016). Para su implementacion se uso la libreria `loo` de R, que permite calcular el LOO-CV y el WAIC (Watanabe-Akaike Information Criterion) de manera eficiente.

2 Modelos de Riesgos Proporcionales

3 Comparación de Modelos

4 Conclusiones

5 Referencias

Bartoš, F., Aust, F., & Haaf, J. M. (2022). Informed Bayesian survival analysis. *BMC Medical Research Methodology*, *22*(1). https://doi.org/10.1186/s12874-022-01676-9

Brooks, S., Gelman, A., Jones, G., & Meng, X.-L. (2011). *Handbook of Markov Chain Monte Carlo*. https://doi.org/10.1201/b10905

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian data analysis*. https://doi.org/10.1201/b16018

Ibrahim, J. G., Chen, M.-H., & Sinha, D. (2001). *Bayesian survival Analysis*. Springer Science & Business Media.

Muse, A. H., Ngesa, O., Mwalili, S., Alshanbari, H. M., & El-Bagoury, A.-A. H. (2022). A Flexible Bayesian Parametric Proportional Hazard Model: Simulation and Applications to Right-Censored Healthcare Data. *Journal of Healthcare Engineering*, *2022*, 1-28. https://doi.org/10.1155/2022/2051642

Stan Development Team. (2024). *Stan Reference Manual*. https://mc-stan.org

Vehtari, A., Gelman, A., & Gabry, J. (2015). Pareto smoothed importance sampling. *arXiv*. https://doi.org/10.48550/arxiv.1507.02646

Vehtari, A., Gelman, A., & Gabry, J. (2016). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, *27*(5), 1413-1432. https://doi.org/10.1007/s11222-016-9696-4

6 Apéndices

6.1 Modelo Paramétrico RP Exponencial

```
data {
    int<lower=1> N;                       // Número de observaciones
    int<lower=1> K;                       // Número de covariables
    vector[N] time;                       // Tiempos observados (evento o censura)
    array[N] int<lower=0,upper=1> event;  // Indicador de evento (1=evento, 0=censura)
    matrix[N, K] X;                       // Matriz de covariables (sin intercepto)
}
parameters {
    vector[K] beta;                // Coeficientes de las covariables
    real<lower=0> lambda;           // Parámetro de escala (> 0)
    real<lower=0> kappa;            // Parámetro de forma (> 0)
}
model {
    // Priors
    beta ~ normal(0, 2);           // Prior para coeficientes
    lambda ~ gamma(1, 4);          // Prior para la escala
    kappa ~ gamma(2, 0.1);         // Prior para la forma

    // Definiciones vectorizadas
    vector[N] linpred = X * beta;
    vector[N] log_hazard = log(kappa) + kappa * log(lambda) + linpred + (kappa - 1) * log(time);
    vector[N] cum_hazard = exp(kappa * log(lambda) + linpred) .* pow(time, kappa);

    // Log-verosimilitud
    vector[N] event_vector = to_vector(event);
    vector[N] event_complement = 1 - event_vector;

    target += dot_product(event, log_hazard) - sum(cum_hazard);
}
generated quantities {
    vector[N] log_lik;  // Log-verosimilitud individual

    vector[N] linpred = X * beta;
    vector[N] log_hazard = log(kappa) + kappa * log(lambda) + linpred + (kappa - 1) * log(time);
    vector[N] cum_hazard = exp(kappa * log(lambda) + linpred) .* pow(time, kappa);

    vector[N] event_vec = to_vector(event);

    log_lik = event_vec .* log_hazard - cum_hazard;
}
```

6.2 Modelo Paramétrico RP Weibull

```
data {
  int<lower=1> N;                        // Número de observaciones
  int<lower=1> K;                        // Número de covariables
  vector[N] time;                        // Tiempos observados
  array[N] int<lower=0, upper=1> event; // Indicador de evento
  matrix[N, K] X;                        // Matriz de diseño (sin intercepto)
}

parameters {
  real lambda;               // Parámetro de escala (log(gamma))
  real<lower=0> alpha;       // Parámetro de forma
  vector[K] beta;            // Coeficientes de las covariables
}

transformed parameters {
  vector[N] sigma;           // Parámetro de escala de Stan por observación

  for (i in 1:N) {
    // Conversión a la parametrización de Stan: sigma = exp(-(lambda + X*beta)/alpha)
    sigma[i] = exp(-(lambda + X[i] * beta)/alpha);
  }
}

model {
  // Priors
  lambda ~ normal(10, 5);     // Prior normal para lambda
  alpha ~ gamma(1, 1);        // Prior gamma para alpha
  beta ~ normal(0, 3);        // Prior para coeficientes

  // Likelihood
  for (i in 1:N) {
    if (event[i]) {
      target += weibull_lpdf(time[i] | alpha, sigma[i]);
    } else {
      target += weibull_lccdf(time[i] | alpha, sigma[i]);
    }
  }
}
generated quantities {
  real gamma = exp(lambda);  // Parámetro de escala en parametrización de supervivencia
  vector[N] log_lik;         // Log-verosimilitud punto a punto

  for (i in 1:N) {
    // Usamos el sigma ya calculado en transformed parameters
    real sigma_i = sigma[i];

    if (event[i]) {
      log_lik[i] = weibull_lpdf(time[i] | alpha, sigma_i);
    } else {
      log_lik[i] = weibull_lccdf(time[i] | alpha, sigma_i);
    }
  }
}
```

6.3 Modelo Semiparamétrico RP

```
data {
  int<lower=1> N;               // Número de observaciones
  int<lower=1> J;               // Número de intervalos
  int<lower=1> K;               // Número de covariables
  vector<lower=0>[N] time;      // Tiempos observados
  array[N] int<lower=0, upper =1> event;// Indicador de evento
  matrix[N, K] X;               // Matriz de diseño
  vector<lower=0>[J+1] cuts;    // Puntos de corte (debe incluir 0 y max(time))
}

transformed data {
  vector[J+1] sorted_cuts = sort_asc(cuts);  // Ordenar los cortes
}

parameters {
  vector[J] log_lambda;         // Log-riesgo base por intervalo
  vector[K] beta;               // Coeficientes de covariables
  cholesky_factor_corr[J] L;    // Factor de Cholesky para correlaciones
  real<lower=0> sigma;          // Desviación estándar de los log-lambdas
}

transformed parameters {
  vector[J] lambda = exp(log_lambda);
  matrix[J,J] Sigma = multiply_lower_tri_self_transpose(diag_pre_multiply(rep_vector(sigma,J), L));
}

model {
  // Priors
  log_lambda ~ multi_normal_cholesky(rep_vector(0,J), diag_pre_multiply(rep_vector(sigma,J), L));
  sigma ~ exponential(1);
  beta ~ normal(0, 3);
  L ~ lkj_corr_cholesky(2);     // Prior para matriz de correlación

  // Likelihood
  for (i in 1:N) {
    real lin_pred = X[i] * beta;
    real cum_haz = 0;
    int current_interval = 1;

    // Calcular riesgo acumulado hasta el tiempo t
    while (current_interval <= J && time[i] > sorted_cuts[current_interval + 1]) {
      real width = sorted_cuts[current_interval + 1] - sorted_cuts[current_interval];
      cum_haz += lambda[current_interval] * width;
      current_interval += 1;
    }

    // Último intervalo parcial
    if (current_interval <= J) {
      real width = time[i] - sorted_cuts[current_interval];
      cum_haz += lambda[current_interval] * width;
    }

    // Contribución a la verosimilitud
    if (event[i]) {
      target += log(lambda[current_interval]) + lin_pred - exp(lin_pred) * cum_haz;
    } else {
      target += -exp(lin_pred) * cum_haz;
    }
  }
}

generated quantities {
  corr_matrix[J] Omega = multiply_lower_tri_self_transpose(L);
  vector[N] log_lik;
```

```
  // Cálculo de log-verosimilitud punto a punto
  for (i in 1:N) {
    real lin_pred = X[i] * beta;
    real cum_haz = 0;
    int current_interval = 1;

    while (current_interval <= J && time[i] > sorted_cuts[current_interval + 1]) {
      real width = sorted_cuts[current_interval + 1] - sorted_cuts[current_interval];
      cum_haz += lambda[current_interval] * width;
      current_interval += 1;
    }

    if (current_interval <= J) {
      real width = time[i] - sorted_cuts[current_interval];
      cum_haz += lambda[current_interval] * width;
    }

    log_lik[i] = event[i]
      ? log(lambda[current_interval]) + lin_pred - exp(lin_pred) * cum_haz
      : -exp(lin_pred) * cum_haz;
  }
}
```