SEDESOL 2018

Mónica Zamudio López

Título del proyecto: USO DE DATOS MASIVOS PARA LA EFICIENCIA DEL ESTADO Y LA INTEGRACIÓN REGIONAL

Clave: ATN/OC 15822-RG

Puesto: Científico de Datos Junior

Resultados, conclusiones y recomendaciones

Entregable número: 3

Acrónimo del proyecto:	Estimación de Ingreso
Nombre completo del proyecto:	USO DE DATOS MASIVOS PARA LA EFICIENCIA DEL ESTADO Y LA INTEGRACIÓN REGIONAL
Referencia:	ATN/OC 15822-RG
URL del Proyecto:	http://www.plataformapreventiva.gob.mx

Tipo de Entregable:	Reporte (R)
Fecha de Entrega Contractual:	Diciembre - 2018
Fecha de Entrega:	Enero 2019
Número de Páginas:	35
Keywords:	estimación ingreso ciencia datos modelos gráficos
Autor:	Mónica Zamudio López, Laboratorio de Datos, SEDESOL

Resumen

La Secretaría de Desarrollo Social (SEDESOL) es una entidad del gobierno mexicano destinada al apoyo de la población para el mejoramiento de sus condiciones de vida.

Un problema importante para SEDESOL es la correcta distribución de sus recursos por lo que es importante contar con una metodología que permita la generación de una focalización correcta para así poder ayudar a aquellos que realmente están en condiciones vulnerables.

En este reporte se detalla el proceso de ajuste, evaluación e implementación de un modelo para priorizar las verificaciones domiciliarias y así corregir el reportaje incorrecto en los levantamientos socioeconómicos

Índice general

Lista de Acrónimos

CUIS Cuestionario Único de Información Socioeconómica

SIFODE Sistema de Focalización de Desarrollo

SEDESOL Secretaría de Desarrollo Social

ENIGH Encuesta Nacional de Ingresos y Gastos de los Hogares

PEA Población Económicamente Activa

LGDS Ley General de Desarrollo Social

INEGI Instituto Nacional de Estadística y Geografía

1. Introducción

Este proyecto tenía como objetivo principal corregir el reportaje incorrecto (sub y sobrerreportaje de variables) en los levantamientos socieconómicos. El mecanismo principal de corrección son las verificaciones domiciliarias, por lo que el proceso de análisis aquí descrito busca en primera instancia priorizar estas verificaciones, tomando en cuenta que verificar añade un costo importante al costo total del cuestionario.

Dado que contamos con datos de programas como PROSPERA, que verifican el 100 % de sus cuestionarios, podemos utilizar las respuestas verificadas y abordar el problema como un problema de datos faltantes, modelado con Redes Bayesianas. A partir de la estructura obtenida para la gráfica, construimos un clasificador para transformar el resultado de un modelo probabilístico en un clasificador binario para detonar una verificación.

Este clasificador fue una adición al aplicativo CUIS APP, utilizado para los levantamientos socioeconómicos, lo cual implicó reproducir el modelo en un dispositivo móvil de una forma que fuera independiente a su estatus de conectividad a internet.

En este reporte se detallan los resultados finales del análisis y modelado, así como la parte del proceso de implementación implementada en R.

2. Consideraciones finales para ajustar el modelo

El modelo: recapitulación y avances

Fuentes de datos

Como se mencionó en [2], la fuente principal de datos para el ajuste del modelo fue la Encuesta de Características Socioeconómicas de los Hogares, o ENCASEH. Esta encuesta contiene los módulos definidos en el CUIS como base, y puede ser complementada con módulos adicionales de preguntas que se consideren necesarias para la focalización, según el objetivo del programa en cuestión. Para nosotros fue de particular interés utilizar los datos de PROSPERA, que incluyen el módulo de verificaciones domiciliarias. Así, podemos considerar el problema de reportaje incorrecto como un problema de datos faltantes según el marco de análisis planteado por [4], utilizando el módulo de verificaciones domiciliarias como etiquetas[1]. La idea del enfoque de datos faltantes es entonces explotar la distribución conjunta entre las respuestas a los cuestionarios y las variables verificadas para poder imputar datos dado un conjunto de evidencia.

Ahora, es plausible pensar que la distribución de las características de los hogares cambie conforme nos encontremos en municipios con distintos perfiles de carencias o marginación. Eso nos llevó a incluir una segunda fuente de en el modelo: los datos de marginación a nivel municipal, publicados por el CONAPO. Estos datos consisten en un índice de marginación, que es esencialmente la primera componente resultante de aplicar el método de componentes principales a este conjunto de variables[2]:

- Población de 15 años o más analfabeta

[1] Aquí queda implícito el primer supuesto importante que hacemos en la modelación: que no hay error en las verificaciones. Es decir, que las variables verificadas siempre reflejan las verdaderas características de los hogares.
[2] Todas las variables se miden en porcentaje de la población que satisface la característica mencionada.

- Población de 15 años o más sin primaria terminada
- Viviendas particulares habitadas sin drenaje ni servicio sanitario
- Viviendas particulares habitadas sin energía eléctrica
- Viviendas particulares habitadas sin agua entubada
- Viviendas particulares habitadas con piso de tierra
- Viviendas particulares habitadas con algún nivel de hacinamiento
- Población en localidades con menos de cinco mil habitantes
- Población ocupada con ingreso de hasta dos salarios mínimos

Dado que los levantamientos contienen referencias al domicilio en el que se encuentra la vivienda, podemos utilizar esa información para filtrar en tiempo real los valores correspondientes al municipio que contiene a ese domicilio. Sin embargo, utilizar dos conjuntos distintos de variables (el conjunto con las variables a nivel municipio y el conjunto original) añade ciertas dificultades para la comparación entre modelos, de las que hablaremos más adelante.

El proceso de modelado

Recordemos que estamos ajustando una red bayesiana a los datos. Una red bayesiana en un grupo de variables $V = (X_1, X_2, \ldots, X_n)$ es una gráfica dirigida G, con V como su conjunto de vértices. La gráfica G representa las distribuciones cuya conjunta se puede escribir como $p(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} p(x_i|pa(x_i))$, donde $pa(x_i)$ es el conjunto de variables que son padres de x_i en la gráfica G.

Notemos cómo esta definición induce una propiedad de Markov en la red bayesiana: los nodos representan variables aleatorias y las aristas relaciones de probabilidad, por lo que la estructura de la red define una factorización de la función de distribución conjunta en un conjunto de distribuciones *locales* de probabilidad, una para cada variable[3]. Debido a la correspondencia entre independencia condicional y separación gráfica, es posible aprender primero la estructura de la gráfica, y dada una estructura, estimar las distribuciones locales una por una, sin las complicaciones inducidas por la dimensionalidad de los datos.

[3]Para más detalle sobre esto, ver [5].

El método

Independientemente del algoritmo que se escoja para encontrar la estructura de la gráfica, el método convencional para ajustar una red bayesiana a un conjunto de datos es el siguiente:

1. Aprender la estructura de la gráfica a través de un algoritmo apropiado.
2. Estimar los parámetros de las distribuciones locales para cada nodo, utilizando una distribución multinomial para variables categóricas y una distribución normal multivariada para variables continuas[4]

El algoritmo

Para ajustar la red bayesiana en R, escogimos el paquete `bnlearn`: un paquete que incluye distintos tipos de algoritmos para el aprendizaje de la estructura, con un conjunto bastante grande de métricas de evaluación a escoger.

Los algoritmos que aprenden la estructura de una gráfica se dividen en dos grandes categorías: basados en restricciones y basados en puntuación (constraint-based y score-based, respectivamente). Hasta ahora, no existe evidencia concluyente que apunte a la superioridad de algún tipo sobre el otro (ver [3]), por lo que decidimos utilizar `hill climbing`: un algoritmo *greedy* basado en puntuación que pertenece a la familia de algoritmos de búsqueda local.

Parámetros relevantes

Por su naturaleza, el algoritmo permite especificar ciertos parámetros de búsqueda que pueden ser optimizados para obtener mejores resultados:

- **blacklist/whitelist**: conjunto de aristas a excluir/incluir del modelo. En cualquiera de los dos casos, especificar este conjunto permite que el algoritmo descarte estructuras irrelevantes. En nuestro caso, descartamos todas las aristas de variables reportadas a variables verificadas, y también todas las aristas hacia variables a nivel municipal que no partieran de otra variable a

[4] En este caso, los modelos son mayormente conocidos como Redes Bayesianas *Gaussianas*, y las variables aleatorias se relacionan entre sí a partir de restricciones lineales.

nivel municipal.

- **score**: la medida de ”bondad de ajuste” a ser utilizada para la evaluación de las posibles estructuras. Existen muchas medidas posibles de evaluación, pero todas tienen que asignar el mismo puntaje a las estructuras que tengan asociada la misma distribución de probabilidad conjunta. En este caso, se utilizó siempre el AIC, o Criterio de Información de Akaike.
- **restarts**: número de ”recomienzos” que hace el algoritmo para evitar caer en un mínimo local.
- **k**: coeficiente de penalización por el número de parámetros estimados. Es importante incluir algún grado de penalización para prevenir el sobreajuste.

3. Evaluación del modelo

Recordemos que el problema que buscamos resolver a través de este modelo es el de priorizar correctamente las verificaciones domiciliarias, reduciendo así la proporción de hogares que son sujetos a un proceso de focalización incorrecta. Dada la deriva temporal a la que está sujeto el proceso de focalización -los programas que operan en cierta región, así como los criterios que utilizan para focalizar, están cambiando constantemente- simplificamos el problema al definir que nuestro objetivo es, simplemente, asignar correctamente las verificaciones. Definimos como una verificación correctamente asignada a aquella en la que se tuvo que corregir el valor de al menos una variable. Así, podemos utilizar la distribución posterior resultante de nuestro modelo bayesiano para definir un clasificador que indique una verificación a realizar en caso de que haya una probabilidad de tener cambios suficientemente alta.

Podemos entonces comparar entre conjuntos de variables como comparamos entre clasificadores. Sin embargo, dado un conjunto de variables, necesitamos escoger el valor de k que resulte en la estructura más adecuada. Definimos entonces nuestro proceso de evaluación de modelos en dos pasos.

Comparación de estructuras

Lo primero que necesitamos asegurar es que, dado un conjunto de variables, tenemos la mejor estructura posible que el algoritmo sea capaz de encontrar. Necesitamos escoger entonces una medida de bondad de ajuste para la distribución conjunta inducida por la estructura de la gráfica.

Dado que escogimos un modelo bayesiano, resulta natural pensar en la verosimilitud -o alguna transformación monótona de esa función- para evaluar modelos. En nuestro caso, decidimos utilizar la devianza de prueba por la facilidad de interpretarla de forma análoga a la suma de cuadrados residua-

les, en el caso de regresión lineal por mínimos cuadrados[1]. La devianza de prueba para un modelo M_0, que estima $\hat{\mu} := E[Y|\hat{\theta}_0]$ basada en las observaciones y está definida por:

$$D(y, \hat{\mu}) = 2(log(p(y|\hat{\theta}_s)) - log(p(y|\hat{\theta}_0)))$$

Donde $\hat{\theta}_0$ denota los valores de los parámetros del modelo a evaluar M_0, y $\hat{\theta}_s$ denota los valores de los parámetros del modelo **saturado**. Esta métrica nos permite, dado un conjunto de variables, escoger el valor de k que nos dé la mejor estructura posible para minimizar la devianza.

Después de un poco de exploración con las restricciones, decidimos considerar los siguientes seis modelos:

1. **Modelo sin restricciones**: modelo que incluye solamente las variables reportadas y verificadas, sin ninguna restricción para el algoritmo de búsqueda de estructura.
2. **Modelo base**: modelo que incluye solamente las variables reportadas y verificadas, con la restricción de que no exista una arista de una variable reportada a una verificada.
3. **Modelo base + variables municipales**: modelo base con las variables a nivel municipal que componen el índice de marginación de CONAPO.
4. **Modelo base + GM**: modelo base con la variable de grado de marginación: una discretización del índice de marginación a través del método Dalenius.
5. **Modelo base + variables municipales + GM**: modelo base con la variable de grado de marginación y con todas las variables municipales que lo componen.
6. **Modelo base + 3 variables municipales**: modelo base con tres variables a nivel municipal: analfabetismo, falta de drenaje/excusado y hacinamiento.

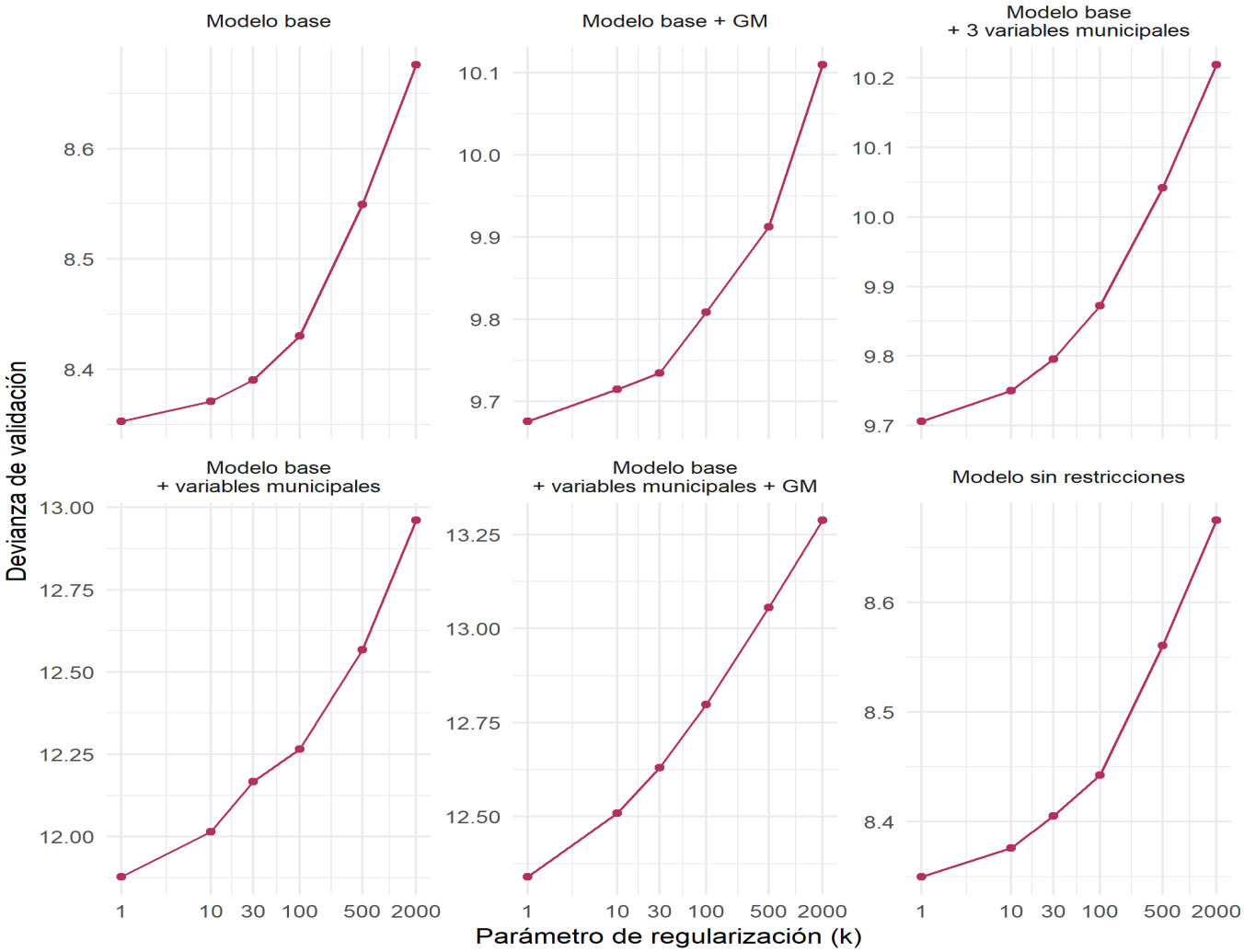
Para efectos analíticos, discretizamos todas las variables a nivel municipal con estas tres categorías:$(0 - 10, 11 - 40, 41 - 100)$. La estructura resultante en cada modelo se encuentra adjunta en el anexo.

Utilizando la métrica propuesta, comparamos cada uno de los modelos para escoger el mejor valor

[1]Es decir, que una devianza de validación en el modelo "perfecto.º saturado es igual a cero, y la devianza de validación incrementa conforme el ajuste del modelo empeora

del parámetro de regularización. Los resultados preliminares son los siguientes:

Figura 3.1: Devianza de validación para todos los modelos considerados.
Dado que la razón de verosimilitud tiene un denominador distinto (estamos considerando distintos conjuntos de variables), no es comparable entre gráficas.



Comparando distintos conjuntos de variables

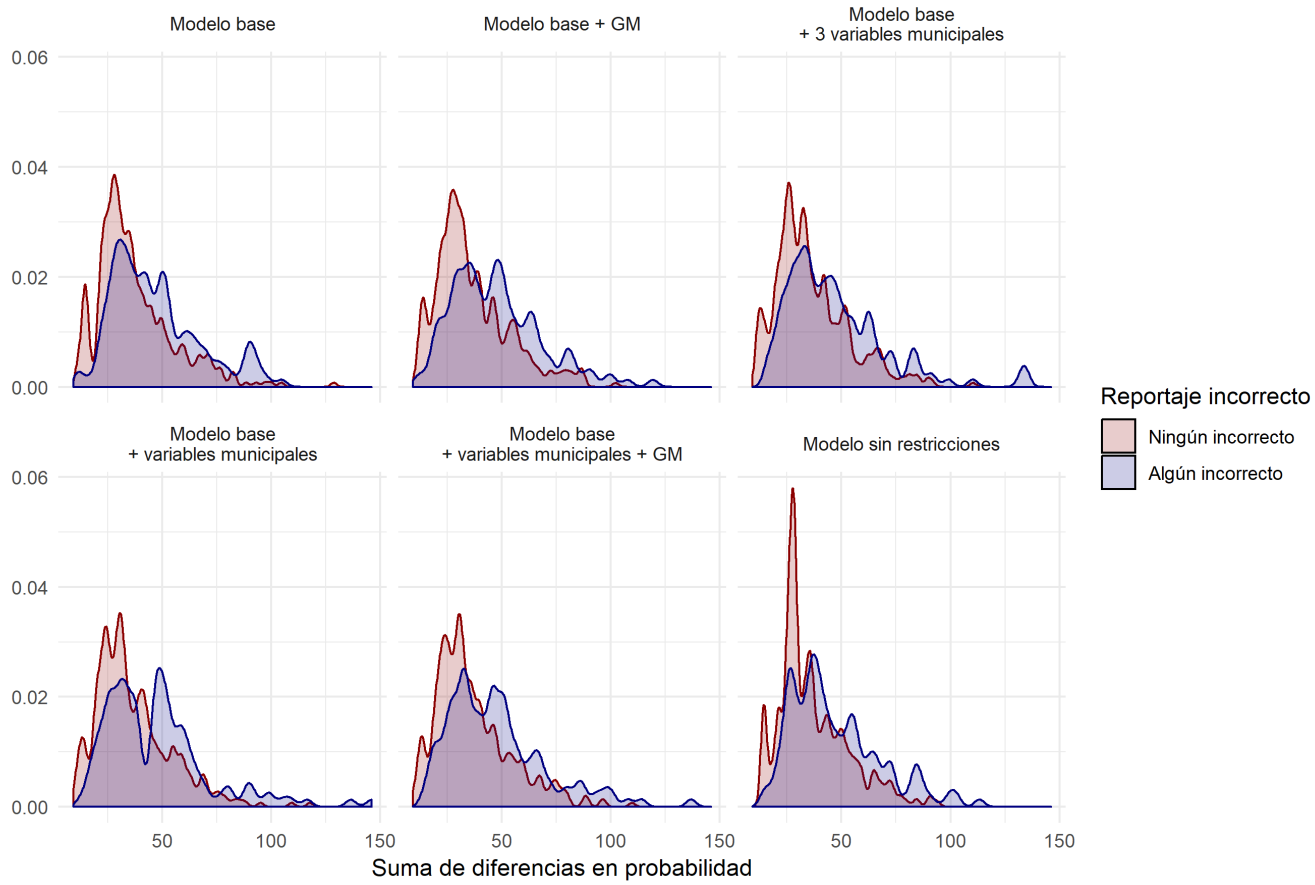
Una vez que escogimos la mejor estructura posible dado un conjunto de variables, es necesario construir el clasificador que mencionamos anteriormente. Necesitamos entonces hacer inferencia probabilística para obtener un estadístico que nos ayude con la tarea de clasificación.

Existen varios algoritmos para hacer inferencia en Redes Bayesianas. Nosotros consideramos dos: *logic sampling* -parte de la familia de técnicas de muestreo por importancia- y *belief propagation*,

un algoritmo de paso de mensaje. La ventaja del segundo sobre el primero es que permite el cálculo de las distribuciones marginales de manera más eficiente. Además, las técnicas de muestreo por importancia pueden llevar a descartar buena parte de los elementos de la muestra, resultando así en un tiempo mayor de convergencia. Sin embargo, los algoritmos de muestreo por importancia que están implementados en R tienen menos carga de dependencias, lo cual puede resultar útil en la implementación como explicaremos más adelante.

Una vez que calculamos la distribución posterior de las variables ”verdaderas” dadas las respuestas al cuestionario, podemos hacer una comparación simple y construir un estadístico que podamos interpretar como la probabilidad de tener algún incorrecto. Para esto, lo que hicimos fue restar las probabilidades de las respuestas (codificadas como variables binarias) y sumar esas diferencias. En la siguiente figura podemos observar la distribución de este estadístico, según el verdadero estado de reportaje en el cuestionario.

Figura 3.2: Suma de probabilidades según la distribución a posteriori



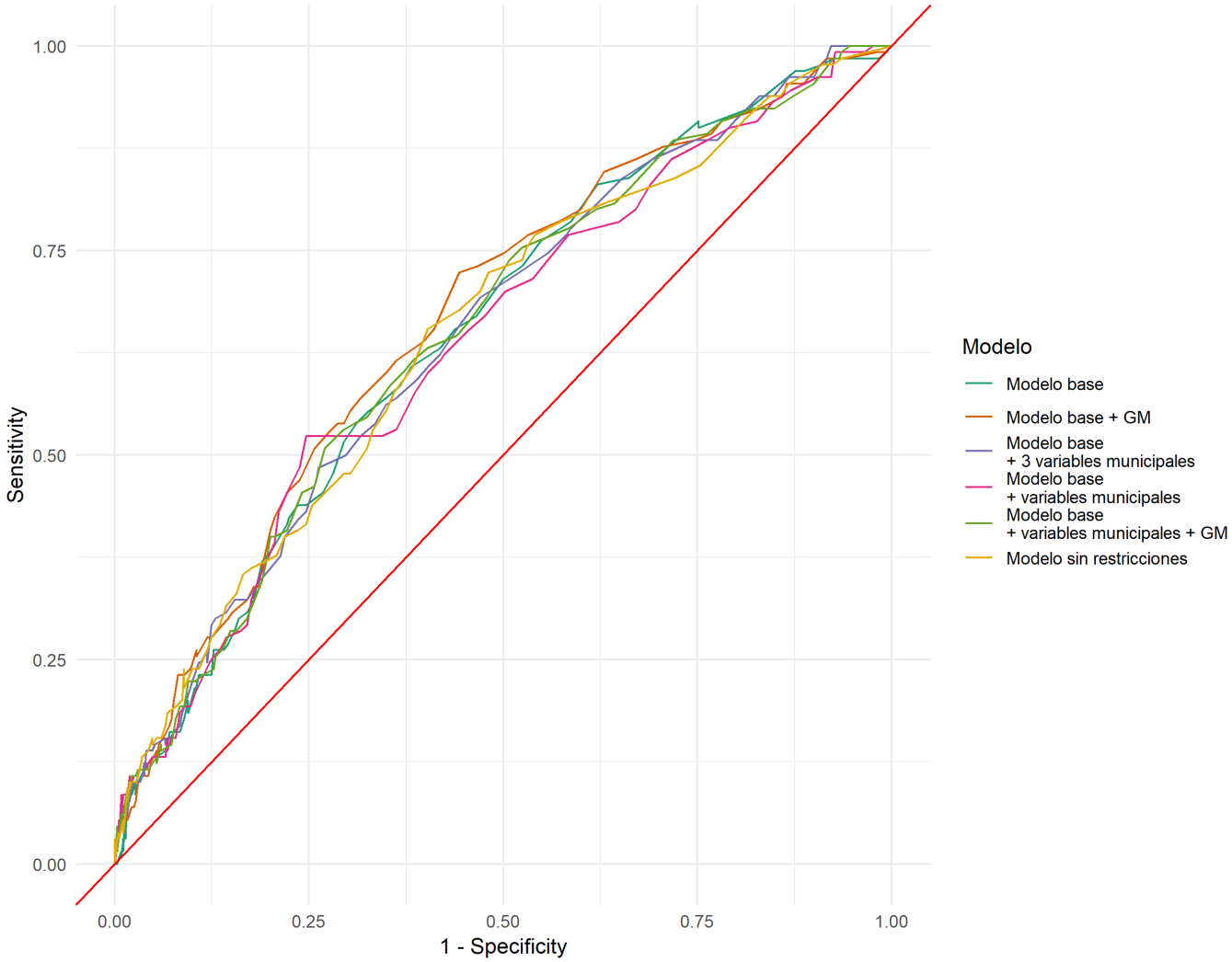
Comparando las distribuciones que induce cada modelo, notemos que si bien las que tienen algún error están más sesgadas a la derecha, en general no parece fácil establecer las diferencias entre estas y las distribuciones sin error. Esto implica que necesitamos explorar más opciones para construir un mejor clasificador.

Una vez obtenida la suma, comparamos varios puntos de corte para construir un clasificador binario, y evaluamos el clasificador utilizando el área bajo la curva ROC. La curva ROC (Receiver Operating Characteristic, por su origen naval) es una herramienta para evaluar la capacidad de un clasificador binario de discriminar conforme va variando el umbral de clasificación. Esta curva resulta de graficar la Tasa de Falsos Positivos (TFP) en el eje x, y la tasa de verdaderos positivos (TVP) en el eje Y. Estas dos métricas se definen como sigue:

$$TFP := \frac{FP}{N}$$
$$TVP := \frac{VP}{P}$$

Donde (F, V) son Falso y Verdadero, y (N, P) son Negativo y Positivo, respectivamente. Cabe mencionar que a la TVP comúnmente se le llama *sensitividad*, y que la TFP puede reescribirse como $1 - TVN$, donde TVN también es conocida como *especificidad*. Ambas métricas son relativamente estándar en el campo de la ciencia de datos, y compararlas gráficamente tiene la intención de tomar en cuenta el intercambio que se hace entre falsos positivos y falsos negativos en la construcción de un clasificador.

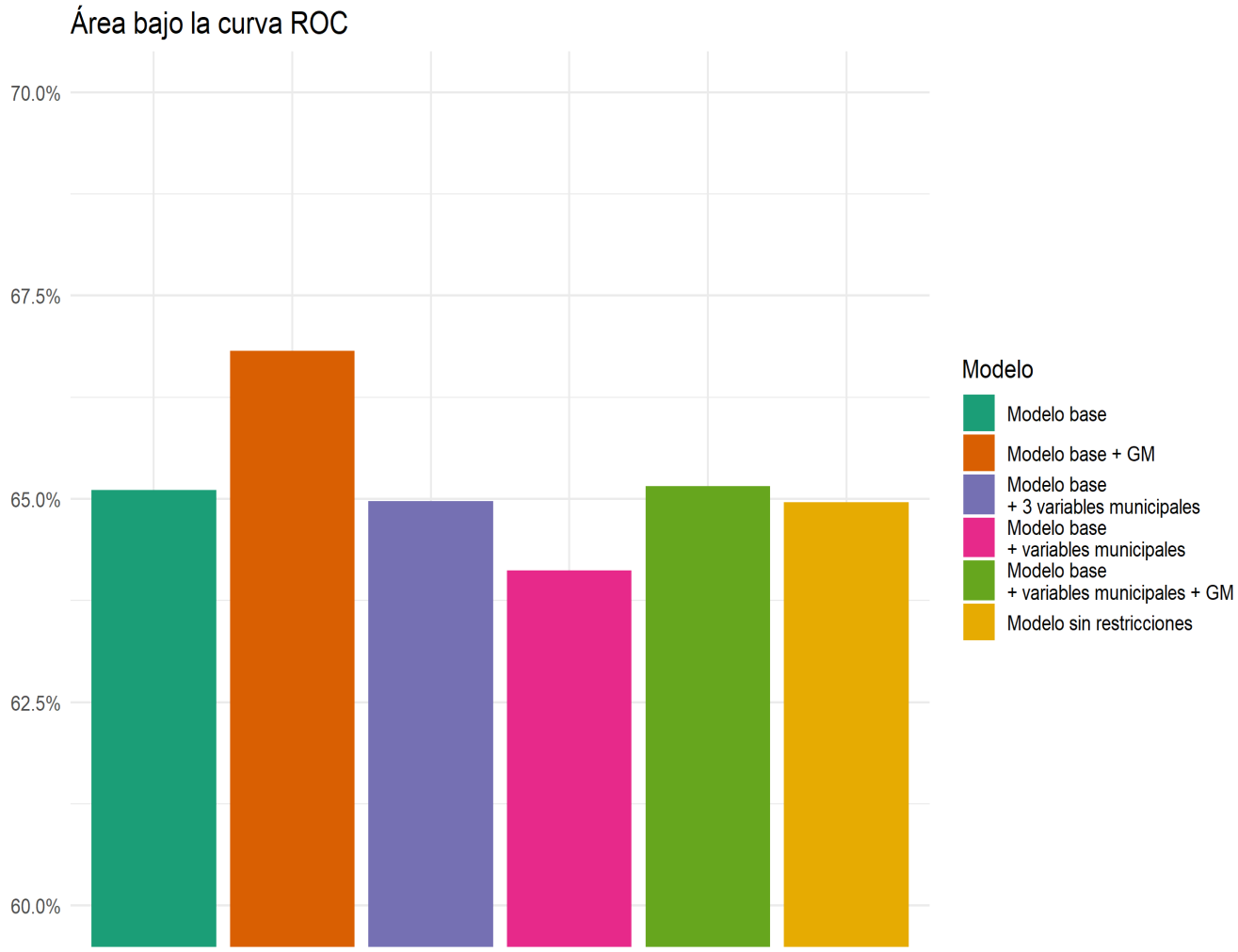
Figura 3.3: Curva ROC



Intuitivamente, la curva ROC de un clasificador perfecto sería una L reflejada sobre el eje horizontal, y la curva ROC de un clasificador construido tirando una moneda sería la línea roja $y = x$.

Ahora, para poder evaluar la efectividad general de cada clasificador, tomando en cuenta todos los

puntos de corte, se utiliza generalmente el área bajo la curva ROC (AUC, por sus siglas en inglés). En este caso utilizamos la Regla del Trapecio para calcular el AUC a partir de la colección de puntos obtenida.



Notemos que hemos limitado el rango del eje y para efectos de comparabilidad entre modelos, pero el resultado del AUC varía relativamente poco entre ellos.

Los resultados indican que el Modelo Base con grado de marginación induce un clasificador ligeramente mejor que los otros. Además, si comparamos las estructuras de los modelos, el grado de marginación sí parece estar resumiendo gran parte de la información que tienen el resto de las variables. Decidimos utilizar este modelo para la implementación del piloto, pero tenemos una serie de recomendaciones para seguir esta exploración:

- Explorar más conjuntos de variables a nivel municipal, y otro tipo de discretizaciones.

- Reentrenar el modelo con los resultados del piloto. Intuitivamente, tiene sentido pensar que las tasas de reportaje incorrecto sean mayores en programas que no hacen verificaciones domiciliarias a todos sus cuestionarios.
- Explorar los patrones de sub y sobrerreporte obtenidos del piloto, buscando reducir la dimensionalidad del problema.

4. Implementación en el piloto

Dados los requerimientos de la arquitectura a nivel aplicación (ver [1]), buscamos que el modelo sea capaz de calcular la distribución a posteriori con una latencia muy baja y de forma independiente a su ubicación en red. Dentro de los dos algoritmos considerados para la inferencia que se mencionaron anteriormente, el que genera menor latencia es el de propagación de creencias (belief propagation). Sin embargo, este tiene una carga de dependencias considerable que presentan potenciales retos de compilación. La latencia de los algoritmos de muestreo es suficientemente baja, por lo que el modelo se implementó en los dispositivos móviles con logic sampling.

Como se explica también en [1], el aplicativo móvil guarda los datos de entrada en un archivo CSV, que es leído por el script de R que carga el binario del modelo, calcula la suma de probabilidades de incorrectos a posteriori y manda al STDOUT una variable binaria que indica si es o no necesaria una verificación. Para fines de reentrenamiento del modelo, todos los cuestionarios serán verificados en este piloto.

5. Conclusión

En este proyecto se planteó el problema del reportaje incorrecto como un problema de datos faltantes, modelado a partir de redes bayesianas. Para esto, se tomaron en cuenta dos conjuntos de variables: las variables de los cuestionarios ENCASEH, aplicados por el programa PROSPERA, y las variables que componen el índice de marginación de CONAPO, discretizadas de acuerdo a una partición heurística.

Se consideraron distintos conjuntos de variables en el modelado, evaluando primero la estructura resultante a través de la devianza de validación, y después construyendo un clasificador a partir de la distribución a posteriori para las variables verificadas. El clasificador fue evaluado con el área bajo la curva ROC.

Una vez elegida la especificación para el modelo, se construyó el código que recibe las respuestas por parte del aplicativo móvil, calcula la distribución a posteriori de las variables dada la evidencia y genera la clasificación que indica si hay o no que verificar ese cuestionario. En el caso de la implementación de este piloto, se decidió verificar todas las observaciones para fines del reentrenamiento del modelo.

Bibliografía

[1] Edgar Cabrera. Integración y api de descarga de información. Entregable 2.

[2] Mónica Zamudio López. Resultados de los modelos de subreportaje. Entregable 2.

[3] José Manuel Gutiérrez Marco Scutari, Catharina Elisabeth Graafland. Who learns better bayesian network structures: Constraint-based, score-based or hybrid algorithms?

[4] Karthika Mohan, Judea Pearl, and Tian Jin. Missing data as a causal inference problem, 2013.

[5] Marco Scutari. Learning bayesian newtorks with the bnlearn r package.

A. Apéndice

A.1. Estructuras ajustadas de cada modelo

Figura A.1: Modelo sin restricciones

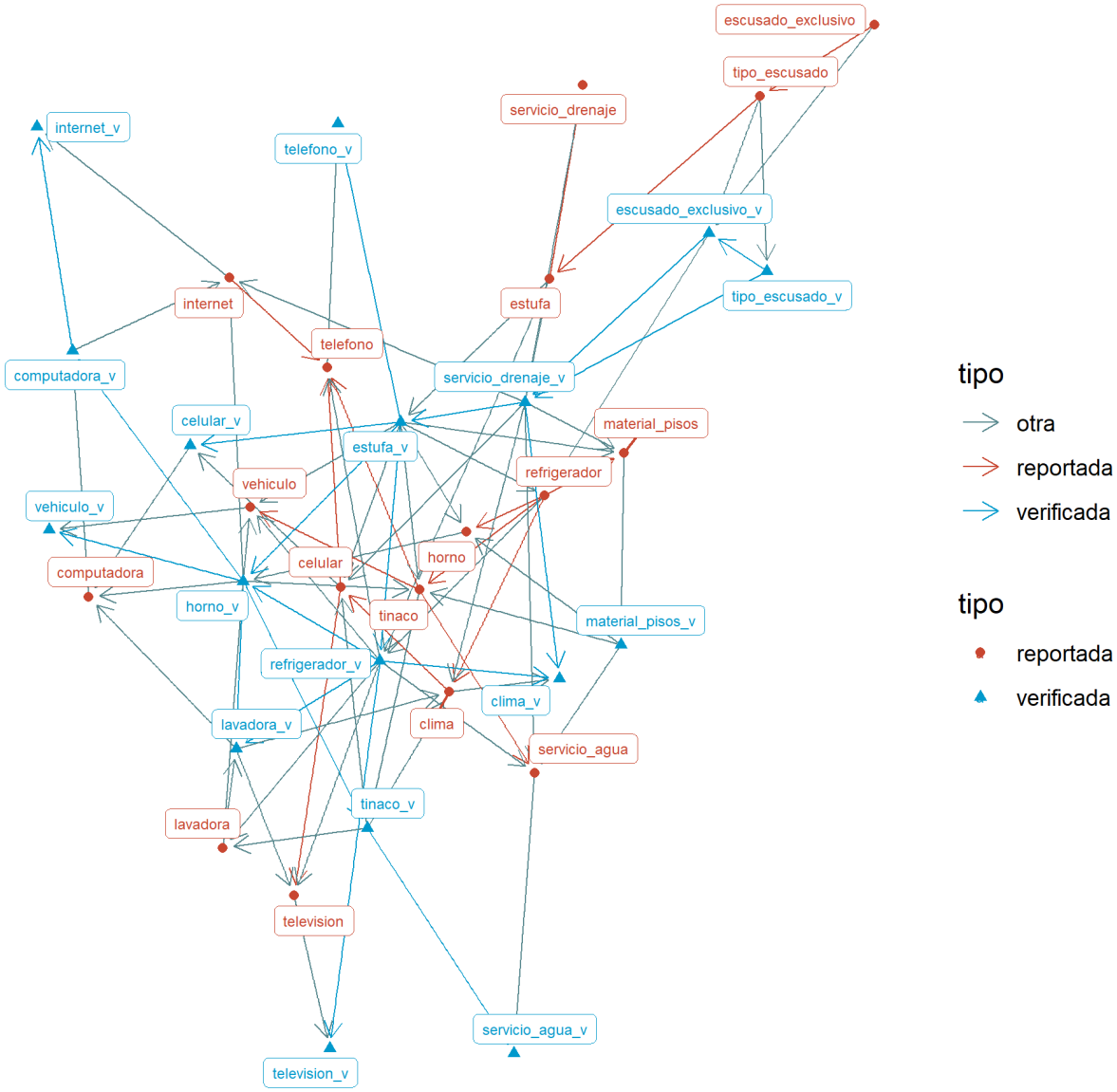


Figura A.2: Modelo base + variables municipales + grado de marginación

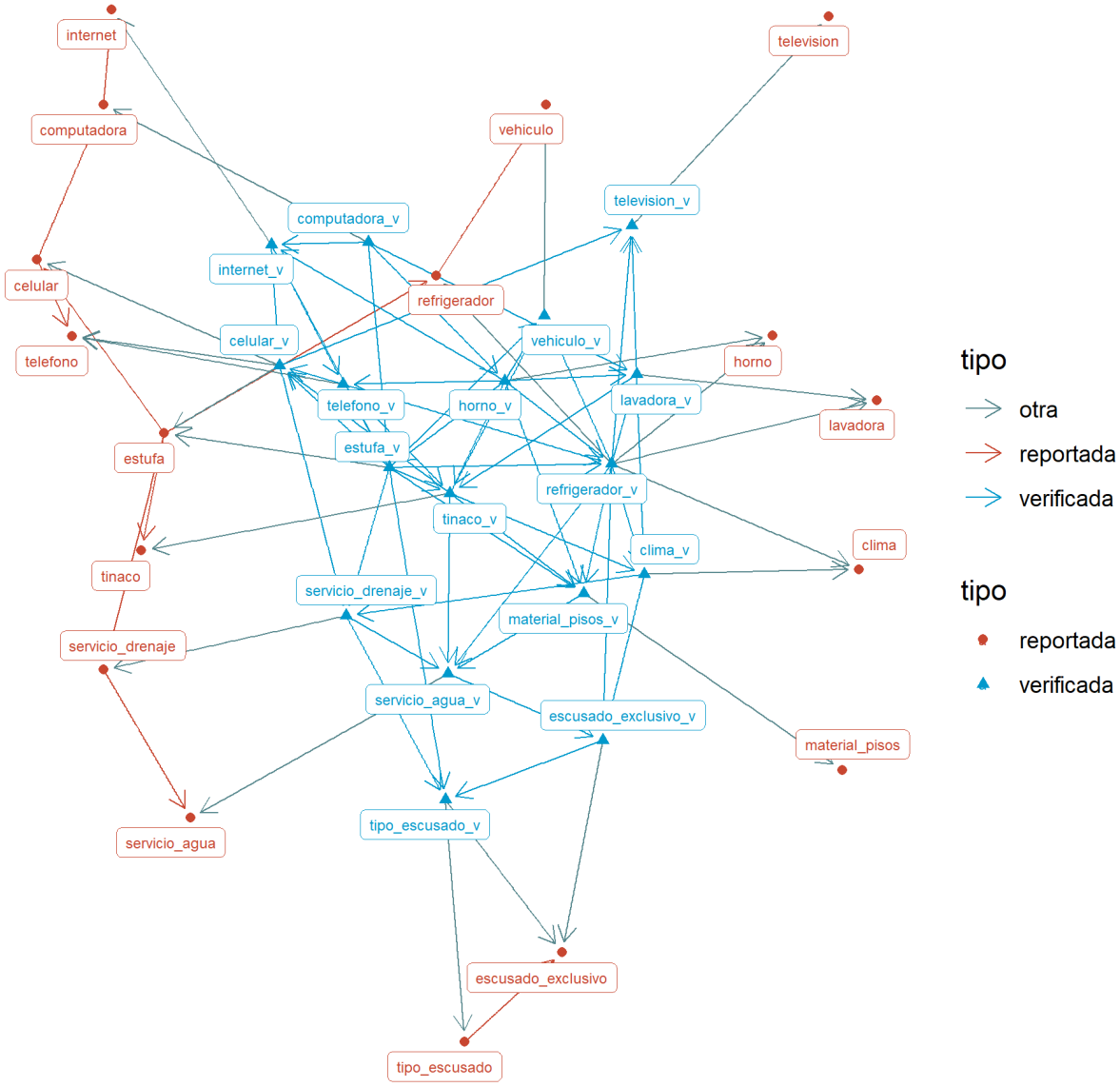


Figura A.3: Modelo base + grado de marginación

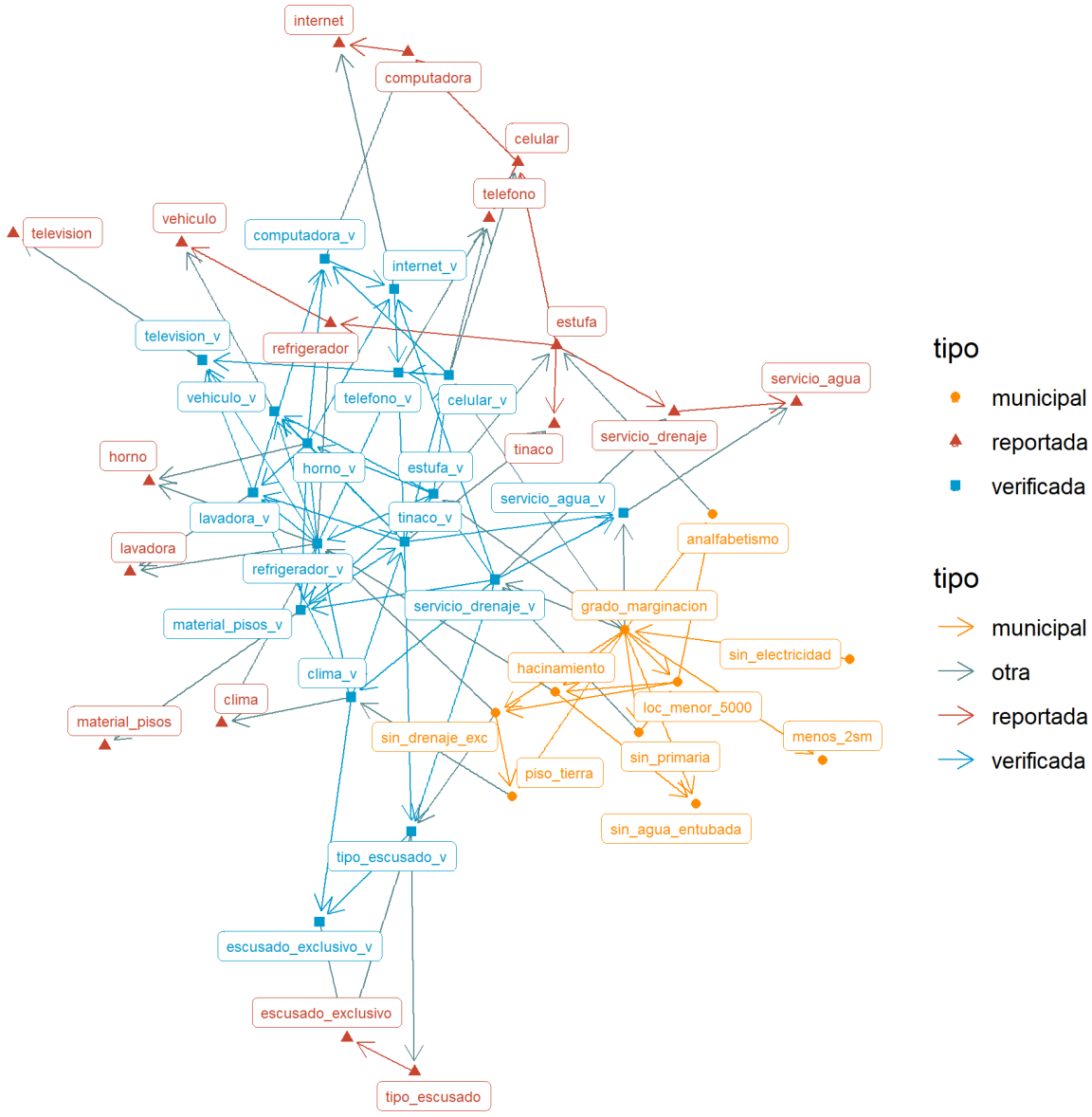


Figura A.4: Modelo base + variables municipales

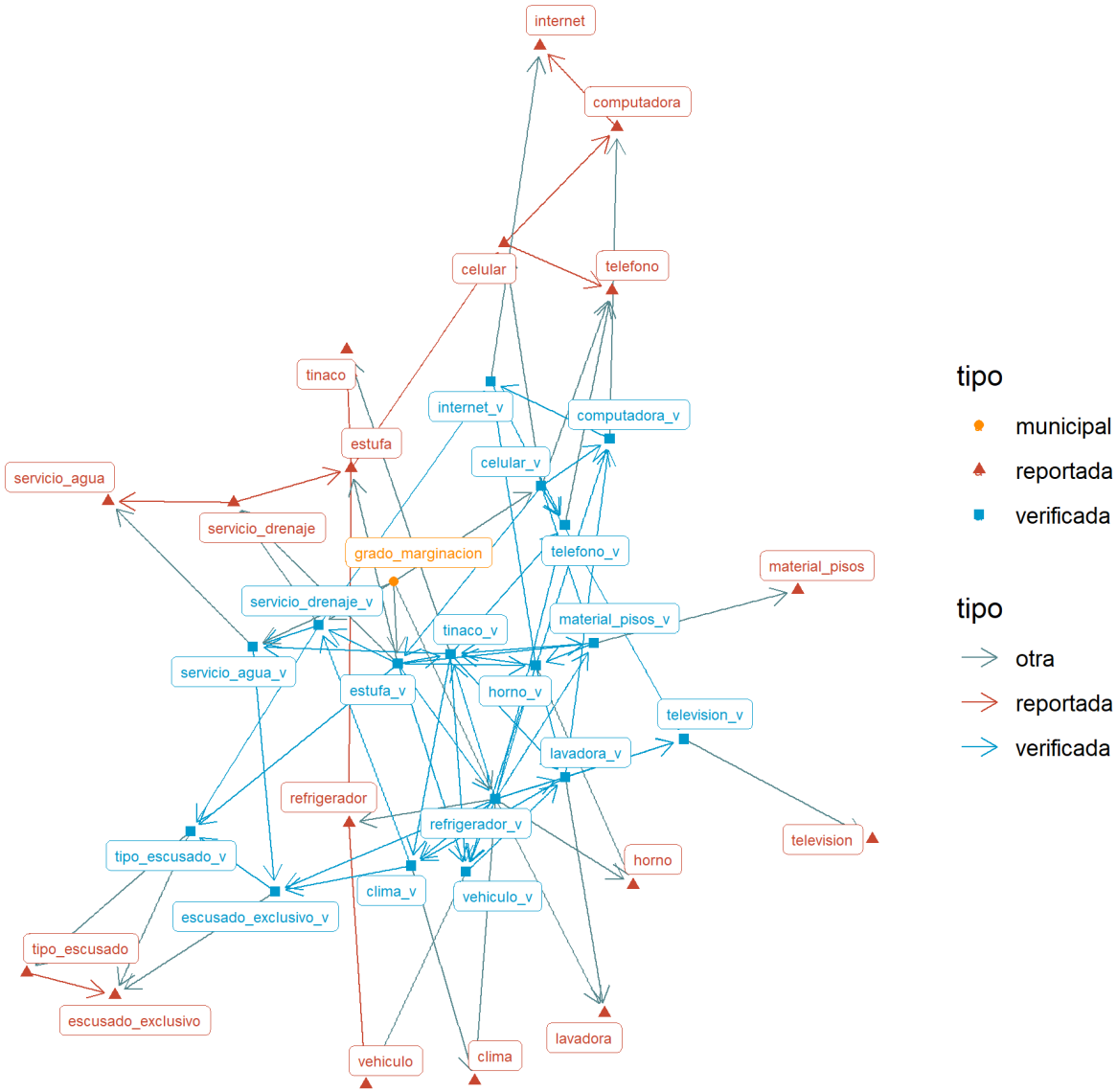
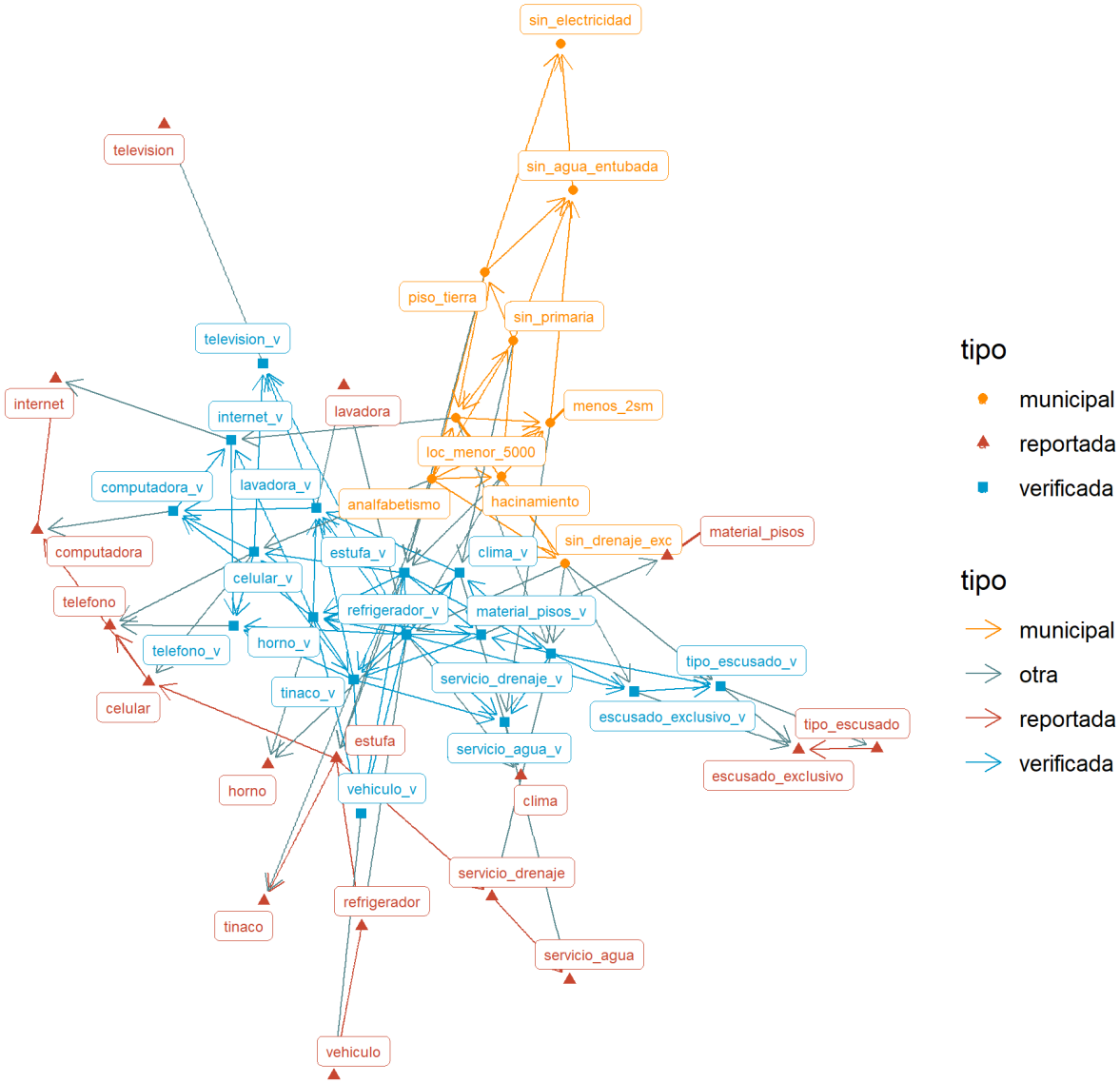


Figura A.5: Modelo base + 3 variables municipales



A.2. Código

Listing A.1: Funciones auxiliares para el ajuste e inferencia en Redes Bayesianas

```
#!/usr/bin/env Rscript
# Function to plot a graphical model

graficar_red <- function(p_dag){
  cbbPalette <- c("reportada" = "#C84630",
                  "verificada" = "#0099CC",
                  "otra" = "#55868C",
                  "municipal" = "darkorange")

  graf_1 <- tidygraph::as_tbl_graph(bnlearn::as.graphNEL(p_dag))
  graf_1 <- graf_1 %>%
            tidygraph::activate(nodes) %>%
            dplyr::mutate(tipo = ifelse(str_detect(name, "_v"), "verificada",
                                ifelse(name %in% municipales, "municipal",
                                       "reportada"))) %>%
            tidygraph::activate(edges) %>%
            dplyr::mutate(tipo = ifelse(.N()$tipo[from] == "verificada" &
                                        .N()$tipo[to] == "verificada", "verificada",
                                        ifelse(.N()$tipo[from] == 'municipal' &
                                               .N()$tipo[to] == "municipal", 'municipal',
                                               ifelse(.N()$tipo[from] == 'reportada' &
                                                      .N()$tipo[to] == "reportada", 'reportada',
                                                      'otra'))))

  gg <- ggraph(graf_1, "fr") +
        ggraph::geom_edge_link(aes(colour = tipo),
                               arrow = arrow(length = unit(0.1, "inches")),
                               edge_width = 0.3,
                               end_cap = circle(1.5, 'mm')) +
        ggraph::geom_node_label(aes(label = name, colour = tipo),
                                repel = TRUE,
                                label.size = 0.10,
                                size = 2) +
        ggraph::geom_node_point(aes(colour = tipo, shape = tipo)) +
        ggraph::theme_graph(base_family = "sans") +
        ggraph::scale_edge_color_manual(values = cbbPalette) +
        ggplot2::scale_colour_manual(values = cbbPalette) +
        ggplot2::scale_fill_manual(values = cbbPalette)
```

```
  gg
}

# Function to evaluate a graphical model using the validation deviance score

eval_deviance <- function(data,
                          blacklist,
                          evidence_vars,
                          model_name,
                          k_vec = c(1, 10, 30, 100, 500, 2000)){

  input_data <- data %>%
                dplyr::select(one_of(c(evidence_vars, verificadas)))

  loss_k <- sapply(k_vec, function(k){
    print(k)
    evaluate_p_dag <- bnlearn::bn.cv(method = "hold-out",
                                     data = input_data,
                                     bn = "hc", fit = "bayes",
                                     k = 5, m = floor(0.20 * nrow(data)),
                                     algorithm.args = list(score = "aic",
                                                           k = k,
                                                           blacklist = blacklist,
                                                           restart = 5))

    loss(evaluate_p_dag)
  })

  data.frame(k = k_vec, devianza = loss_k, modelo = model_name)
}

# AIC/BIC scoring for BN models

score_dag <- function(dag, data, model_name){
  data.frame(model_name = model_name,
             aic = bnlearn::score(x = dag,
                                  data = data,
                                  type = 'aic'),
             bic = bnlearn::score(x = dag,
                                  data = data,
                                  type = 'bic'))
}
```

```
eval_model <- function(train_data, evidence_vars, model_name, blacklist){

  input_data <- train_data  %>%
                dplyr::select(one_of(c(evidence_vars, verificadas)))

  purrr::map_df(k_vec, function(k) eval_deviance(train_data = data.frame(input_data),
                                                 test_data = data.frame(test_data),
                                                 blacklist = blacklist,
                                                 evidence_vars = evidence_vars,
                                                 model_name = model_name))
}


###################### Evaluation by imputation of probabilities ##########################

# Fit a BN with the structure specified by p_dag
fit_bn <- function(data, p_dag){
  fit_p <- bnlearn::bn.fit(p_dag, data = data,
                           method = "bayes", iss = 2)
  as.grain(fit_p)

}

# Predict values for unobserved variables, given some observed evidence
imputar <- function(data, i, fit_grain, evidence_vars, vars_to_predict){

  # Define evidence to input into the model
  observed_nodes <- lapply(as.list(data[i, evidence_vars]),
                       function(x)(as.character(x)))
  fit_evidence <- setEvidence(fit_grain,
                              nodes = evidence_vars,
                              states = observed_nodes)

  # Get predictions
  query_result <- querygrain(fit_evidence, nodes = vars_to_predict)  %>%
                  lapply(., function(dist) { dist [['1']]})
  query_result$id <- i

  # Join predictions, evidence and true labels into one resulting dataframe
  predictions_df <- query_result  %>%
```

```
                    tibble::as_data_frame() %>%
                    tidyr::gather(variable, prob_tiene, -id) %>%
                    dplyr::mutate(variable = gsub('_v$', '', variable))
  observed_df <- data_frame(variable = evidence_vars,
                       reportada = unlist(observed_nodes))
  labels_df <- data[i, vars_to_predict] %>%
               tidyr::gather(variable, verificada) %>%
               dplyr::mutate(variable = gsub('_v$', '', variable))
  result_df <- predictions_df %>%
               dplyr::left_join(observed_df, by = "variable") %>%
               dplyr::left_join(labels_df, by = "variable") %>%
               dplyr::mutate(prob_tiene = round(100 * prob_tiene, 1),
                             diferencia = abs(prob_tiene - 100*as.numeric(reportada)))
  result_df
}


get_roc <- function(prediction_data, c){
    prediction_data %>%
    dplyr::mutate(incorrecta = reportada != verificada) %>%
    dplyr::group_by(id, model) %>%
    dplyr::summarise_at(vars(incorrecta, diferencia), funs(sum)) %>%
    dplyr::ungroup() %>%
    dplyr::mutate(cutoff = c,
                  positive = incorrecta > 0,
                  selected = diferencia > cutoff,
                  true_positive = selected & positive,
                  false_positive = selected & !positive,
                  true_negative = !selected & !positive,
                  false_negative = !selected & positive)#  %>%
     dplyr::group_by(cutoff, model) %>%
     dplyr::summarise(TPR = sum(true_positive)/sum(positive),
                      FPR = sum(false_positive)/sum(!positive),
                      TNR = sum(true_negative)/sum(!positive),
                      FNR = sum(false_negative)/sum(positive)) %>%
     dplyr::ungroup()
}


aprox_auc <- function(x, y){
  x_lead <- dplyr::lead(x)
  y_lead <- dplyr::lead(y)
```

```
  delta_x <- x - x_lead
  f_x_avg <- (y + y_lead)/2

  sum(delta_x*f_x_avg, na.rm = T)
}


eval_at_k <- function(train_data,
                      test_data,
                      evidence_vars,
                      blacklist,
                      model_name, k,
                      vars_to_predict = verificadas){

  print(model_name)

  input_data <- train_data  %>%
                dplyr::select(one_of(c(evidence_vars, verificadas)))  %>%
                data.frame()

  test_data <- data.frame(test_data)

  # Get the best structure for the data and fit the probability distributions accordingly
  p_dag <- bnlearn::hc(x = input_data,
                       score = 'aic',
                       restart = 5,
                       k = k,
                       blacklist = blacklist)
  fit_grain <- fit_bn(data = input_data, p_dag = p_dag)

  # Given that structure, predict values for a sample
  set.seed(140693)
  sample_i <- sample(1:nrow(test_data), size = 500)
  prediction_data <- map_df(sample_i, function(i) imputar(data = test_data,
                                                          i = i,
                                                          fit_grain = fit_grain,
                                                          evidence_vars = evidence_vars,
                                                          vars_to_predict = vars_to_predict))
  # Get TPR and FPR for those predictions and labels
  # roc_data <- map_df(cutoff_vec,
  #                    function(cutoff) get_roc(prediction_data = prediction_data, c = cutoff))
```

```
  # Compute AUC score
  # data.frame(auc = aprox_auc(roc_data, n = length(cutoff_vec)),
  #            k = k,
  #            model = model_name)

  prediction_data %>%
    dplyr::mutate(k = k,
                  model = model_name)
}
```

Listing A.2: Código de exploración, análisis y ajuste del modelo

```
# Cargamos todas las librerias necesarias, generamos la conexion con la base de datos
library(igraph)
library(DBI)
library(dbrsocial)
library(dotenv)
library(readr)
library(tidyverse)
library(ggrepel)
library(ggraph)
library(tidygraph)
library(bnlearn)
library(gRain)

dotenv::load_dot_env()
source('/graph_models_funs.R')

con <- prev_connect()

# Definimos las variables de interes
reportadas <- c('estufa', 'refrigerador', 'horno', 'lavadora',
                'television', 'telefono', 'celular', 'computadora',
                'clima', 'vehiculo', 'internet', 'servicio_agua',
                'escusado_exclusivo', 'tipo_escusado', 'servicio_drenaje',
                'material_pisos', 'tinaco')
verificadas <- glue::glue('{reportadas}_v') %>% as.character()
ids <- c('ent_fed', 'munici', 'hogar_id')

# Filtrar variables relevantes para usar menos RAM
familias <- dbrsocial::large_table(con, clean, encaseh_familias) %>%
  dplyr::filter(data_date == '2016-a') %>%
  dplyr::select(one_of(c(ids, reportadas, verificadas))) %>%
  dplyr::collect() %>%
  dplyr::mutate(cve_ent = stringr::str_pad(ent_fed, width = 2,
                                           side = "left", pad = '0'),
                cve_mun = stringr::str_pad(munici, width = 3,
                                           side = "left", pad = '0'),
                cve_muni = paste0(cve_ent, cve_mun))

municipales_cut <- c('analfabetismo', 'sin_primaria', 'sin_drenaje_exc',
                     'sin_electricidad', 'sin_agua_entubada', 'hacinamiento',
```

```
                    'piso_tierra', 'loc_menor_5000', 'menos_2sm')
municipales <- c(municipales_cut, 'grado_marginacion')

municipios <- dbrsocial::large_table(con, raw, conapo_marginacion) %>%
  dplyr::filter(anio == 2015) %>%
  dplyr::collect() %>%
  dplyr::mutate(cve_muni = stringr::str_pad(cve_mun, width = 5,
                                            side = 'left', pad = '0')) %>%
  dplyr::mutate_at(vars(one_of(municipales_cut)),
                   .funs = function(x) cut(as.numeric(x),
                                           breaks = c(0, 10, 40, 100),
                                           include.lowest = TRUE)) %>%
  dplyr::select(cve_muni, one_of(municipales))

familias <- dplyr::left_join(familias, municipios)
rm(municipios)

discon_db(con)
dim(familias)

# Creamos el dataset de trabajo que vamos a utilizar, dividimos en entrenamiento y prueba
familias_base <- familias %>%
  dplyr::mutate_at(vars(starts_with('servicio_agua')), function(x) if_else(x == 7, 1, 0)) %>%
  dplyr::mutate_at(vars(starts_with('servicio_drenaje')), function(x) if_else(x == 5, 1, 0)) %>%
  dplyr::mutate_at(vars(starts_with('material_pisos')), function(x) if_else(x == 2, 1, 0)) %>%
  dplyr::mutate_at(vars(starts_with('tipo_escusado')), function(x) if_else(x == 1, 1, 0)) %>%
  tidyr::replace_na(list(escusado_exclusivo = 0, escusado_exclusivo_v = 0)) %>%
  dplyr::select(hogar_id, one_of(c(reportadas, verificadas, municipales)))

set.seed(140693)
familias_base <- familias_base %>%
                   dplyr::sample_n(400000)

familias_entrena <- familias_base %>%
                      dplyr::mutate(id = row_number()) %>%
                      dplyr::sample_frac(size = 0.7)

familias_valida <- familias_base %>%
                     dplyr::mutate(id = row_number()) %>%
                     dplyr::filter(!(id %in% familias_entrena$id)) %>%
                     dplyr::select(one_of(c(reportadas, verificadas, municipales))) %>%
                     dplyr::mutate_all(as.factor)
```

```
familias_entrena <- familias_entrena %>%
                      dplyr::select(one_of(c(reportadas, verificadas, municipales))) %>%
                      dplyr::mutate_all(as.factor)


#### Modelo sin restricciones
p_dag <- bnlearn::hc(x = data.frame(familias_entrena %>% select(-one_of(municipales))),
                     score = 'aic', k = 100,
                     restart = 20, perturb = 5)

graficar_red(p_dag)
ggsave('p_dag.png')

### Modelo base
blacklist_1 <- expand.grid(from = reportadas, to = verificadas)

p_dag_1 <- bnlearn::hc(x = data.frame(familias_entrena %>% select(-one_of(municipales))),
                       score = 'aic', k = 100,
                       restart = 20, perturb = 5,
                       blacklist = blacklist_1)

graficar_red(p_dag_1)
ggsave('p_dag_1.png')

### Modelo base con variables municipales y GM
blacklist_mun1 <- expand.grid(from = c(reportadas, verificadas),
                              to = municipales) %>%
                    dplyr::bind_rows(blacklist_1) %>%
                    dplyr::mutate_all(as.factor)

p_dag_mun1 <- bnlearn::hc(x = data.frame(familias_entrena),
                          score = 'aic', k = 100,
                          restart = 20, perturb = 5,
                          blacklist = blacklist_mun1)


graficar_red(p_dag_mun1)
ggsave('p_dag_mun1.png')

### Modelo base con GM
blacklist_mun2 <- blacklist_mun1 %>%
```

```
                    dplyr::filter(!(from %in% municipales_cut),
                                  !(to %in% municipales_cut))

p_dag_mun2 <- bnlearn::hc(x = data.frame(familias_entrena %>%
                                         dplyr::select(-one_of(municipales_cut))),
                          score = 'aic', k = 100,
                          restart = 20, perturb = 5,
                          blacklist = blacklist_mun2)


graficar_red(p_dag_mun2)
ggsave('p_dag_mun2.png')

### Modelo base + variables municipales
blacklist_mun3 <- blacklist_mun1 %>%
                    dplyr::filter(from != 'grado_marginacion',
                                  to != 'grado_marginacion')

p_dag_mun3 <- bnlearn::hc(x = data.frame(familias_entrena %>%
                                         dplyr::select(-grado_marginacion)),
                          score = 'aic', k = 100,
                          restart = 20, perturb = 5,
                          blacklist = blacklist_mun3)


graficar_red(p_dag_mun3)
ggsave('p_dag_mun3.png')

### Modelo base + 3 variables municipales
vars_mun4 <- c(reportadas, verificadas, c("analfabetismo", "sin_drenaje_exc", "hacinamiento"))

blacklist_mun4 <- blacklist_mun1 %>%
                  dplyr::filter(from %in% vars_mun4,
                                to %in% vars_mun4)


p_dag_mun4 <- bnlearn::hc(x = data.frame(familias_entrena %>%
                                         dplyr::select(one_of(vars_mun4))),
                          score = 'aic', k = 100,
                          restart = 20, perturb = 5,
                          blacklist = blacklist_mun4)
```

```
graficar_red(p_dag_mun4)
ggsave('p_dag_mun4.png')

### Evaluamos la devianza de validacion
model_list = list(modelo_base = list(evidence_vars = reportadas,
#                                    blacklist = NULL,
                                     name = 'Modelo sin restricciones'),
                  modelo_r1 = list(evidence_vars = reportadas,
#                                  blacklist = blacklist_1,
                                   name = 'Modelo base'),
                  modelo_mun1 = list(evidence_vars = c(reportadas, municipales),
#                                    blacklist = blacklist_mun1,
                                     name = 'Modelo base\n+ variables municipales + GM'),
                  modelo_mun2 = list(evidence_vars = c(reportadas, 'grado_marginacion'),
#                                    blacklist = blacklist_mun2,
                                     name = 'Modelo base + GM'),
                  modelo_mun3 = list(evidence_vars = c(reportadas, municipales_cut),
#                                    blacklist = blacklist_mun3,
                                     name = 'Modelo base\n+ variables municipales'),
                  modelo_mun4 = list(evidence_vars = c(reportadas, c('analfabetismo', 'sin_drenaje_exc', 'hacinam
#                                    blacklist = blacklist_mun4,
                                     name = 'Modelo base\n+ 3 variables municipales'))
set.seed(140693)

results_deviance <- purrr::map_df(names(model_list),
                      function(model_name) eval_deviance(data = data.frame(train_data),
                                                         blacklist = model_list[[model_name]]$blacklist,
                                                         evidence_vars = model_list[[model_name]]$evidence_vars,
                                                         model_name = model_list[[model_name]]$name))

ggplot(results_deviance, aes(x = k, y = devianza)) +
  ggplot2::geom_point(color = 'maroon') +
  ggplot2::geom_line(color = 'maroon') +
  ggplot2::scale_x_log10(breaks = c(1, 10, 30, 100, 500, 2000)) +
  ggplot2::facet_wrap(vars(modelo),
                      scales = 'free_y',
                      ncol = 3) +
  ggplot2::labs(x = 'Parametro de regularizacion (k)',
                y = 'Devianza de validacion') +
  ggplot2::theme_minimal()
```

```
ggsave('devianza_validacion.png')


### Generamos las predicciones de nuestro clasificador a partir de la suma de probabilidades
results_prediction <- purrr::map_df(names(model_list),
                            function(model_name) eval_at_k(train_data = familias_entrena,
                                                           test_data = familias_valida,
                                                           evidence_vars = model_list[[model_name]]$evidence_var
                                                           blacklist = model_list[[model_name]]$blacklist,
                                                           model_name = model_list[[model_name]]$name,
                                                           k = 10))


cutoff_vec <- seq(0, 200, length.out = 200)
results_selections <- map_df(cutoff_vec, function(c) get_roc(results_prediction, c))
ggplot(results_selections, aes(diferencia,
                               color = positive,
                               fill = positive)) +
geom_density(alpha=0.2) +
scale_color_manual(values=c('darkred', 'navyblue')) +
scale_fill_manual(values=c('darkred', 'navyblue'),
                  breaks=c(FALSE, TRUE),
                  labels=c('Ningun incorrecto', 'Algun incorrecto'),
                  name='Reportaje incorrecto') +
guides(colour='none') +
labs(x = 'Suma de diferencias en probabilidad', y = '') +
facet_wrap(.~model) +
theme_minimal()

ggsave('suma_probabilidades.png',
       height = 6, width = 9)


### Calculamos los valores para generar la curva ROC
results_roc  <- results_selections %>%
                dplyr::group_by(cutoff, model) %>%
                dplyr::summarise(TPR = sum(true_positive)/sum(positive),
                                 FPR = sum(false_positive)/sum(!positive),
                                 TNR = sum(true_negative)/sum(!positive),
                                 FNR = sum(false_negative)/sum(positive)) %>%
                dplyr::ungroup()
```

```
ggplot(results_roc, aes(x = 1-TNR, y = TPR,
                        group = model, colour = model)) +
  geom_line() +
  geom_abline(slope = 1, color = 'red') +
  ylab("Sensitivity") +
  xlab("1 - Specificity") +
  scale_colour_brewer(name = 'Modelo', palette = 'Dark2') +
  scale_x_continuous(limits = c(0,1)) +
  scale_y_continuous(limits = c(0,1)) +
  theme_minimal()
  ggsave('results_roc.png', height = 7, width = 9)


### Utilizamos la Regla del Trapecio para calcular el AUC
results_auc <- results_roc %>%
               dplyr::group_by(model) %>%
               dplyr::summarise(auc = aprox_auc(1-TNR, TPR))

results_auc %>%
  ggplot(aes(model, auc, fill = model)) +
  geom_bar(stat='identity') +
  scale_fill_brewer(name = 'Modelo', palette = 'Dark2') +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(labels = scales::percent,
                     limits = c(0.6, 0.7),
                     oob = rescale_none) +
  labs(title = "Area bajo la curva ROC",
       x = '',
       y = '') +
  theme_minimal()

  ggsave('results_auc.png', height = 6, width = 9)
```

USO DE DATOS MASIVOS PARA LA EFICIENCIA DEL ESTADO Y LA INTEGRACIÓN REGIONAL

23 de enero de 2019

FJR-3-ATN/OC 15822-RG

SEDESOL 2018