Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Acatlán

DETECCIÓN DEL SÍNDROME METABÓLICO A TRAVÉS DE MODELOS PREDICTIVOS

Tesis

Que para obtener el título de:

Licenciado en Actuaría

Presenta:

Gustavo Mercado Enriquez

Asesor:

Act. Iván Domínguez Medina

Santa Cruz Acatlán, Naucalpan de Juárez, Estado de México,
2021.

Agradecimientos

A mi madre, por su apoyo incondicional en cada proyecto que me he planteado y por su entrañable amor. A mi padre, por ser un ejemplo de superación para mí, esto me pone un escalón más cerca de ti. Muchas gracias a los dos por esta increíble familia que me han dado.

A mi hermana, por existir y por compartir cada momento de nuestras vidas juntos. Muchas gracias por motivarme a terminar mi tesis, estoy muy orgulloso de ti.

A mi abuelita, tíos y primos que siempre han estado presentes alentándome, celebrando mis logros y apoyándome cuando lo necesito, muchas gracias por todo. Estoy seguro están disfrutando de este logro tanto como yo.

A todos mis amigos con quienes compartí no solo las clases y los tiempos libres, sino también las noches de desvelo y los fines de semana estudiando y preparando exámenes o tareas. Sin su apoyo no hubiera podido concluir mis estudios. Especial agradecimiento a Raziel, a Gaby y a Espe por su incondicional apoyo.

A Mich, por ayudarme a superarme y por hacerme compañía durante muchos fines de semana mientras hacía mi tesis. ¡Lo logramos!

A mi asesor de tesis, Iván, por su amistad y por todo el apoyo brindado, invaluable para completar esta etapa de mi vida académica y para comenzar mi carrera profesional en SAS.

Al Dr. Miguel Murguía, por el tiempo y por sus valiosas aportaciones para el desarrollo de este trabajo de tesis.

A mis sinodales, la Act. Luz María Lavín Alanís, el Dr. Arturo Erdely Ruíz, el Act. Omar Alejandro Tapia Martínez y el Act. Espartaco Martínez Tolentino, por su valioso tiempo dedicado a la revisión de este trabajo.

Y, por último, aunque no menos importante, a mi querida Universidad Nacional Autónoma de México y a todos mis profesores, que a lo largo de mi trayectoria por la ENP 9 y la FES Acatlán, me han dado todas las herramientas necesarias para poder desempeñarme en esta profesión que tanto me apasiona.

Índice general

Índice de figuras

Índice de Tablas

Introducción

Desde sus orígenes, el ser humano ha intentado comprender todos los fenómenos que ocurren a su alrededor. Como resultado de su intento por explicar la realidad, comenzaron a surgir las primeras teorías que, con el paso del tiempo, se expandieron hasta constituir las diferentes ciencias que hoy se conocen.

Una de las principales amenazas, que ha estado presente en todo momento a lo largo de la historia de la humanidad, es la enfermedad. En un inicio, las primeras civilizaciones buscaron refugio en la religión y el misticismo para intentar curar los padecimientos que les afligían, a la par, descubrieron hierbas y prácticas que les permitían tratar las enfermedades a través de remedios naturales. Hoy en día la ciencia médica ha podido conocer los diversos factores que influyen en la mayoría de las enfermedades y, una vez identificados dichos factores, ha sido más fácil encontrar los tratamientos adecuados para curarlas y reconocer las acciones que pueden ayudar a prevenirlas.

En México, de acuerdo con cifras publicadas por el Instituto Nacional de Estadística y Geografía (INEGI), los padecimientos que más muertes provocan en la población mexicana son la diabetes mellitus, las enfermedades cardiovasculares y los tumores malignos [1]. Existe una forma de identificar a personas con riesgo de padecer cualquiera de los primeros dos padecimientos antes mencionados y es a través de la detección de un conjunto de alteraciones metabólicas conocidas como síndrome metabólico [7] (ver Sección 1.1.).

A través de la Unidad de Biomedicina (UBIMED), la FES Iztacala implementó una campaña para la detección del síndrome metabólico en los jóvenes de nuevo ingreso a la facultad. Esta campaña se llevó a cabo en las jornadas de salud durante la semana de bienvenida y consistía en la toma de medidas clínicas y antropométricas y de una muestra de sangre para hacer una química sanguínea. Una vez teniendo los resultados se determinaba si el alumno tenía (o no) síndrome metabólico (ver Sección 1.1.2.) y se canalizaba a la clínica para que fueran atendidos por un médico y por un nutriólogo. Sin embargo, el elevado costo económico asociado a esta campaña (derivado principalmente de las químicas sanguíneas) y la baja respuesta por parte de los alumnos (no iban con los especialistas y/o no seguían sus indicaciones) hizo que no se pudiera seguir implementando esta campaña.

Este trabajo propone una herramienta, a través del uso de los modelos predictivos, que ayude a los médicos a diagnosticar, de una forma más sencilla, rápida y económica, la presencia del síndrome metabólico en jóvenes de nuevo ingreso a la FES Iztacala. Esta

herramienta podría ser extrapolada a otras facultades, unidades de salud, etc. y así ayudar a las estrátegias de salud pública para tratar de mitigar la alta prevalencia de diabetes y enfermedades cardiovasculares en adultos en México.

Si bien ya existen algunos otros trabajos en los que se utilizan modelos predictivos para determinar si un paciente tiene (o no) síndrome metabólico (por ejemplo: Worachartcheewan et al (2015) [2], Karimi-Alavijeh et al (2016) [3] y Gutiérrez-Esparza et al (2020) [4]), estos tienen como objetivo pronosticar directamente la presencia del síndrome y usan como variables predictoras medidas clínicas y antropométricas, información de estilo de vida y valores de química sanguínea. En el presente trabajo se toma un enfoque diferente, el cual toma las medidas clínicas y antropométricas para pronosticar los valores de química sanguínea, colesterol HDL y triglicéridos, que se necesitan para diagnosticar el síndrome metabólico (ver Sección 1.1.2.) y de esta forma tratar de disminuir tanto el costo económico como el tiempo que toma hacer la química sanguínea.

Por lo anteriormente expuesto, se plantearon los siguientes objetivos.

Objetivos

- **Objetivo general:**
  Desarrollar modelos predictivos que ayuden al diagnóstico oportuno del síndrome metabólico y que disminuya el costo económico de la práctica médica.

- **Objetivo particular:**
  Construir 4 modelos predictivos, usando únicamente medidas clínicas y antropométricas, que permitan identificar si existe posibilidad de tener valores de triglicéridos y colesterol HDL anormales, para posteriormente hacer una estimación de los mismos, en jóvenes mexicanos de entre 18 y 24 años.

1 | Marco Teórico

En este capítulo se explicarán los fundamentos teóricos necesarios para poder comprender, a grandes rasgos, las características del Síndrome Metabólico. Además, se abordará la teoría que sustenta a los modelos predictivos que se utilizarán en el presente trabajo.

1.1. Síndrome Metabólico

Se denomina *Síndrome Metabólico* al conjunto de alteraciones metabólicas constituido por la obesidad de distribución central (es decir, obesidad abdominal), la disminución de las concentraciones del colesterol unido a las lipoproteínas de alta densidad (HDL), la elevación de las concentraciones de triglicéridos, el aumento de la presión arterial y la hiperglucemia (en otras palabras, altos niveles de azúcar en la sangre) [5]. Estas alteraciones metabólicas de manera aislada en una persona no son de extrema gravedad, pero combinadas (tres o más de ellas, ver Sección 1.1.2.) constituyen un problema serio de salud [6].

El Síndrome Metabólico no es una enfermedad nueva; en 1923 el Dr. Eskil Kylin, de Goteburgo (Suecia), publicó un trabajo de título muy parecido a la descripción actual del Síndrome Metabólico: *Síndrome de hipertensión, hiperglucemia e hiperuricemia*, concentrando la atención en un grupo de enfermos que, además de tener la tensión arterial elevada, también presentaban intolerancia a la glucosa o diabetes del adulto y otras alteraciones metabólicas. De esta manera, Kylin y posteriormente Marañón (considerado como el fundador de la endocrinología moderna en España), quien denomina a este conjunto como prediabetes, avanzaban hacia una hipótesis básica sobre la patogenia del síndrome metabólico [7].

1.1.1. Principales complicaciones derivadas del Síndrome Metabólico

El criterio de la Federación Internacional de Diabetes (IDF) es que el síndrome metabólico es una herramienta clínica de gran capacidad predictiva para el médico que le permite la identificación de pacientes en riesgo para padecer diabetes mellitus tipo 2 o alguna enfermedad cardiovascular [7]; de hecho, de acuerdo con la Federación Mexicana de Diabetes, la gravedad que se confiere en términos clínicos al síndrome metabólico se debe en gran medida a que es capaz de aumentar en más de 10 veces el riesgo de desarrollar diabetes y en 3.5 veces el riesgo de muerte por causa cardiovascular [6].

1.1.1.1. Diabetes

Acorde con la Organización Mundial de la Salud (WHO, por sus siglas en inglés), la diabetes es una enfermedad crónica que aparece cuando el páncreas no produce insulina suficiente o cuando el organismo no utiliza eficazmente la insulina que produce [8]. La insulina es una hormona que regula el azúcar en la sangre. El efecto de la diabetes no controlada es la hiperglucemia (aumento del azúcar en la sangre), que con el tiempo daña gravemente muchos órganos y sistemas, especialmente los nervios y los vasos sanguíneos.

En particular, la diabetes de tipo 2 (también llamada no insulinodependiente o de inicio en la edad adulta) se debe a una utilización ineficaz de la insulina. Este tipo representa la mayoría de los casos mundiales y se debe en gran medida a un peso corporal excesivo y a la inactividad física.

Es importante mencionar que, hasta hace poco, este tipo de diabetes solo se observaba en adultos, pero en la actualidad también se está manifestando en niños.

Con el tiempo, la diabetes puede dañar el corazón, los vasos sanguíneos, ojos, riñones y nervios. Algunas de las consecuencias frecuentes de la diabetes son:

- Los adultos con diabetes tienen un riesgo 2 a 3 veces mayor de infarto al miocardio y accidente cerebrovascular.
- La neuropatía periférica (la que afecta a las extremidades, es decir, manos y pies) combinada con la reducción del flujo sanguíneo incrementan el riesgo de úlceras, infección y, en última instancia, amputación.
- La retinopatía diabética es una causa importante de ceguera y es la consecuencia del daño de los pequeños vasos sanguíneos de la retina que se va acumulando a lo largo del tiempo. El 2.6 % de los casos mundiales de ceguera es consecuencia de la diabetes.
- La diabetes se encuentra entre las principales causas de insuficiencia renal.

1.1.1.2. Enfermedad Cardiovascular

De acuerdo con la Organización Mundial de la Salud, las enfermedades cardiovasculares (ECV) son un grupo de desórdenes del corazón y de los vasos sanguíneos [9], entre los que se incluyen:

- La cardiopatía reumática: lesiones del músculo cardiaco y de las válvulas cardíacas debidas a la fiebre reumática, una enfermedad causada por bacterias denominadas estreptococos.
- Las trombosis venosas profundas y embolias pulmonares: coágulos de sangre (trombos) en las venas de las piernas, que pueden desprenderse (émbolos) y alojarse en los vasos del corazón y los pulmones.
- Las cardiopatías congénitas: malformaciones del corazón presentes desde el nacimiento.

- La cardiopatía coronaria: enfermedad de los vasos sanguíneos que irrigan el músculo cardiaco.
- Las arteriopatías periféricas: enfermedades de los vasos sanguíneos que irrigan los miembros superiores e inferiores.
- Las enfermedades cerebrovasculares: enfermedades de los vasos sanguíneos que irrigan el cerebro.

Los ataques al corazón y los accidentes vasculares cerebrales suelen ser fenómenos agudos que se deben sobre todo a obstrucciones que impiden que la sangre fluya hacia el corazón o el cerebro. La causa más frecuente es la formación de depósitos de grasa en las paredes de los vasos sanguíneos que irrigan el corazón o el cerebro. Los ataques cardíacos y accidentes cerebrovasculares suelen tener su causa en la presencia de una combinación de factores de riesgo, entre los cuales están la hipertensión arterial, la diabetes y la hiperlipidemia[1], en otras palabras, el Síndrome Metabólico.

Algunas cifras alarmantes respecto a las enfermedades cardiovasculares son:

- Las ECV son la principal causa de muerte en todo el mundo. Cada año mueren más personas por ECV que por cualquier otra causa.
- Más de tres cuartas partes de las defunciones por ECV se producen en los países de ingresos bajos y medios.
- De 2017 a 2030, casi 23,6 millones de personas morirán por alguna ECV, principalmente por cardiopatías y accidentes cerebrovasculares. Se prevé que estas enfermedades sigan siendo la principal causa de muerte.

1.1.2. Criterio usado para diagnosticar el Síndrome Metabólico

A través de los años se han publicado diferentes guías o criterios para el diagnóstico del Síndrome Metabólico como los de la Organización Mundial de la Salud que toma como punto de partida la resistencia a la insulina, pero resulta compleja su medición, por lo que fue difícil de adaptar en la práctica clínica rutinaria [11].

Algunos otros criterios fueron propuestos por la Asociación Americana del Corazón (AHA, por sus siglas en inglés) y la Federación Internacional de Diabetes (IDF, por sus siglas en inglés). Sin embargo, el criterio más utilizado mundialmente hoy en día es el NCEP-ATP III, el cual fue propuesto en el Tercer Reporte del Programa de Educación sobre el Colesterol del Panel de Expertos en Diagnóstico, Evaluación y Tratamiento de la Hipercolesterolemia en Adultos (Third Report of the National Cholesterol Education Program - NCEP- Expert Panel on Detection, Evaluation, and Treatment of High Blood Cholesterol in Adults -ATP III-) en el 2001, además, considera por igual todos los componentes del Síndrome Metabólico y se

[1]Hiperlipidemia: Afección caracterizada por niveles elevados de partículas de grasa (lípidos) en la sangre. Dos ejemplos de lípidos son el colesterol y los triglicéridos [10].

propone que la presencia de tres de los cinco factores establecía el diagnóstico, lo que hizo que fuera aceptado por su sencillez [11] [12].

En la Tabla 1.1 se presentan los diferentes grupos de criterios antes mencionados para el diagnóstico del Síndrome Metabólico.

Tabla 1.1: Resumen de los criterios diagnósticos del síndrome metabólico.

Criterios diagnósticos	OMS	EGIR	IDF	NCEP-ATP III
Resistencia a la insulina	Disminución de la captación de glucosa en condiciones euglicémicas, con hiperinsulinemia	Insulinemia >25% de los valores en ayunas en no diabéticos	No lo considera	No lo considera
Glicemia en ayunas (mmol/L)	≥ 6,1	≥ 6,1	≥ 5,6	≥ 5,6
PTG (mmol/L)	≥ 7,8	No lo considera	No lo considera	Alterada previamente
Diabetes mellitus	Diagnóstico previo	No lo considera	Diagnóstico previo	Diagnóstico previo
Triglicéridos (mmol/L)	≥ 1,695	≥ 2,00 o tratamiento	≥ 1,70 o tratamiento	≥ 1,70 o tratamiento
HDL-C (mmol/L)	M ≤ 0,9 F ≤1,0	< 1,0 o tratamiento	M <1,04, F <1,29 o tratamiento	M <1,04, F <1,29 o tratamiento
Presión arterial (mmHg)	≥140/90	≥140/90 o diagnóstico previo de hipertenso	≥130/85 o diagnóstico previo de hipertenso	≥130/85 o diagnóstico previo de hipertenso
Diámetro cintura abdominal (cm)	M>90 F > 85	M≥94 F ≥ 80	Variable según grupo étnico	M>102 F > 88
IMC (kg/m2)	>30	No lo considera	No lo considera	No lo considera

OMS: Organización Mundial de la Salud, EGIR: Grupo Europeo para el Estudio de la Resistencia a la Insulina, IDF: Federación Internacional de Diabetes, NCEP-ATP III: Tercer Reporte del Programa de Educación sobre el Colesterol, el Panel de Expertos en Diagnóstico, Evaluación y Tratamiento de la Hipercolesterolemia en Adultos M-sexo masculino; F-sexo femenino; HDL-C: colesterol transportado por lipoproteínas de alta densidad; IMC-índice de masa corporal; Tratamiento-se refiere a presentar valores normales de los lípidos en el momento del estudio pero está bajo tratamiento por diagnóstico previo.

Los criterios utilizados por el Dr. Miguel Murguía Romero, investigador de la Unidad de Biomedicina (UBIMED) de la Facultad de Estudios Superiores Iztacala, toman en consideración cinco factores de riesgo y se diagnostica Síndrome Metabólico si están presentes 3 o más de los 5 factores analizados. Los factores de riesgo considerados son los siguientes:

1. Glucosa en sangre elevada.
2. Niveles bajos de colesterol HDL en la sangre.
3. Niveles altos de triglicéridos en la sangre.
4. Circunferencia de la cintura por arriba del estándar.
5. Presión arterial alta.

En la Tabla 1.2 se muestran los diferentes puntos de corte[2] considerados para cada uno de los factores de riesgo antes mencionados [3].

[2]Los puntos de corte fueron tomados de [13].

[3]Unidades de medida: miligramos por decilitro (mg/dL), centímetros (cm) y milímetros de mercurio ($mmHg$).

Tabla 1.2: Valores de referencia de parámetros clínicos para diagnosticar Síndrome Metabólico.

Parámetro	**Punto de corte**
Colesterol de alta densidad	$<50\ mg/dL$ para mujeres
	$<40\ mg/dL$ para hombres
Circunferencia de Cintura	$\geq 80\ cm$ para mujeres
	$\geq 90\ cm$ para hombres
Triglicéridos	$\geq 150\ mg/dL$
Presión arterial diastólica	$\geq 85\ mmHg$
Presión arterial sistólica	$\geq 130\ mmHg$
Glucosa	$\geq 100\ mg/dL$

Para identificar formalmente si una persona tiene Síndrome Metabólico o no, es indispensable medir los niveles sanguíneos de colesterol HDL, triglicéridos y glucosa en ayunas, lo que implica tomar una muestra de sangre, por lo tanto, si se implementa una estrategia de salud pública para detectar el síndrome, implicaría un costo económico no bajo.

1.1.3. El Síndrome Metabólico en México

Acerca de la prevalencia del síndrome metabólico en el país, se encuentra un artículo emitido por la Unidad de Investigación en Epidemiología Clínica del Hospital de Especialidades Centro Médico Nacional Siglo XXI del Instituto Mexicano del Seguro Social y que data del 2009, en el cual se informa que la prevalencia de este síndrome, sólo en la Ciudad de México, es de 31.9 % con los criterios del NCEP ATP III y de 54.4 % con los criterios de la IDF [14].

Otro artículo, éste realizado por la Facultad de Medicina de Colima en el 2017, donde se investigó la frecuencia del síndrome metabólico en los derechohabientes mayores de 20 años de los servicios de consulta externa del Hospital General de Zona No. 1 del Instituto Mexicano del Seguro Social de Colima; se encontró información parecida a la expuesta en el artículo anteriormente mencionado, la frecuencia global de síndrome metabólico fue del 52.3 % (56 % mujeres y 46.4 % hombres) con los criterios de la IDF [15].

Estos son los datos estadísticos más recientes que se tienen en México acerca del síndrome metabólico. Se puede encontrar más información estadística en los resultados de la Encuesta Nacional de Salud y Nutrición de 2018.

1.1.3.1. Encuesta Nacional de Salud y Nutrición

La Encuesta Nacional de Salud y Nutrición (ENSANUT) es un proyecto del Instituto Nacional de Salud Pública y la Secretaría de Salud Federal que permite conocer cuál es el estado de salud y las condiciones nutricionales de los diversos grupos que forman la población mexicana.

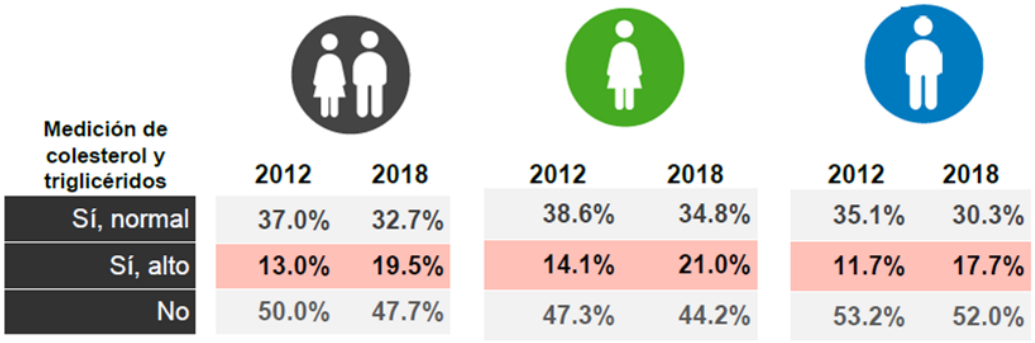
Las primeras Encuestas Nacionales de Nutrición (ENN) se efectuaron en 1988 y 1999. Para 2006 se integraron componentes de salud en aquella encuesta, lo que dio origen a la primera

Encuesta Nacional de Salud y Nutrición (ENSANUT), diseñada para llevarse a cabo cada 6 años.

No obstante, ante el acelerado incremento en el número de niños, adolescentes y adultos, tanto mujeres como hombres, con sobrepeso y obesidad, y de la aparición de enfermedades relacionadas con la nutrición, como diabetes, anemia e hipertensión, se decidió realizar la Encuesta Nacional de Salud y Nutrición de Medio Camino (ENSANUT-MC) en 2016, con el objetivo de dar seguimiento e identificar oportunamente el estado de salud y nutrición de la población y reforzar o ajustar las acciones necesarias para frenarlos [16].

El Instituto Nacional de Estadística y Geografía (INEGI), y el Instituto Nacional de Salud Pública (INSP), llevaron a cabo el levantamiento de la ENSANUT 2018, con el propósito de conocer el estado de salud y nutrición de la población mexicana, actualizar las estimaciones de la ENSANUT-MC 2016, evaluar los resultados de los programas y políticas de salud implementados durante la Administración Federal 2012-2018, y entregar información de utilidad para el desarrollo de políticas públicas acordes con el Plan Nacional de Salud 2018-2024 [17].

A continuación, se presentan algunas figuras y conclusiones publicadas por el INEGI como parte del informe de los resultados obtenidos de la ENSANUT 2018 [18]. Se comienza analizando las mediciones de colesterol y triglicéridos, así como los porcentajes de sobrepeso y obesidad en la población mexicana, los cuales, conforme a lo planteado en la Sección 1.1.2, son tres de los criterios que se toman en cuenta para poder diagnosticar Síndrome Metabólico a una persona.



Medición de colesterol y triglicéridos	2012	2018	2012	2018	2012	2018
Sí, normal	37.0%	32.7%	38.6%	34.8%	35.1%	30.3%
Sí, alto	13.0%	19.5%	14.1%	21.0%	11.7%	17.7%
No	50.0%	47.7%	47.3%	44.2%	53.2%	52.0%

Figura 1.1: Distribución porcentual de la población de 20 y más años de edad según condición de reporte de medición de colesterol y triglicéridos y su resultado, por sexo (2012 *vs* 2018) (INEGI, 2019).

De la Figura 1.1 se puede observar un aumento considerable en el porcentaje de las personas que reportan que la medición de colesterol y triglicéridos en su sangre es alta. También se identifica que casi la mitad de la población mexicana aún no cuenta con una medición de estos valores y es precisamente en este sector de la población que se deben de enfocar los esfuerzos para poder facilitarles una medición oportuna y así desarrollar estrategias que les permitan disminuir el riesgo de padecer algunas de las complicaciones mencionadas en la Sección 1.1.1.

En la Figura 1.2 se puede observar la comparación de la distribución porcentual de la población de 20 años y más con obesidad y sobrepeso, por sexo.

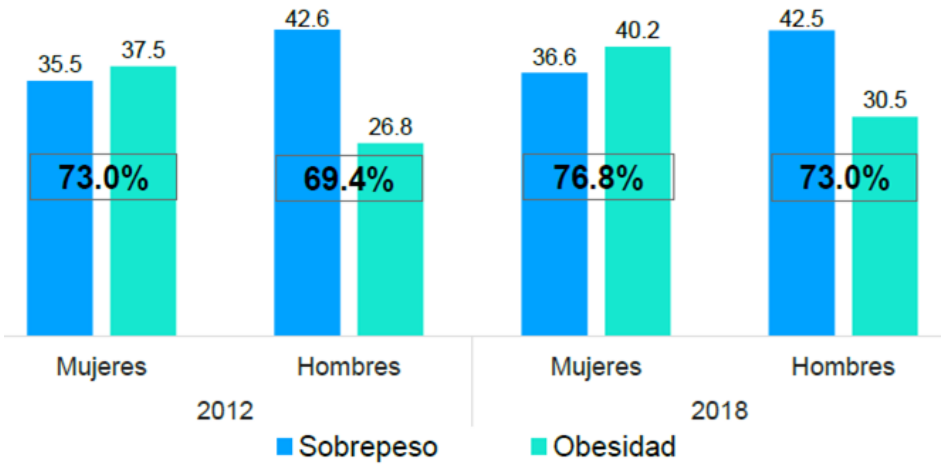


Figura 1.2: Porcentaje de población de 20 años y más de edad con sobrepeso y obesidad, por sexo (2012 *vs* 2018) (INEGI, 2019).

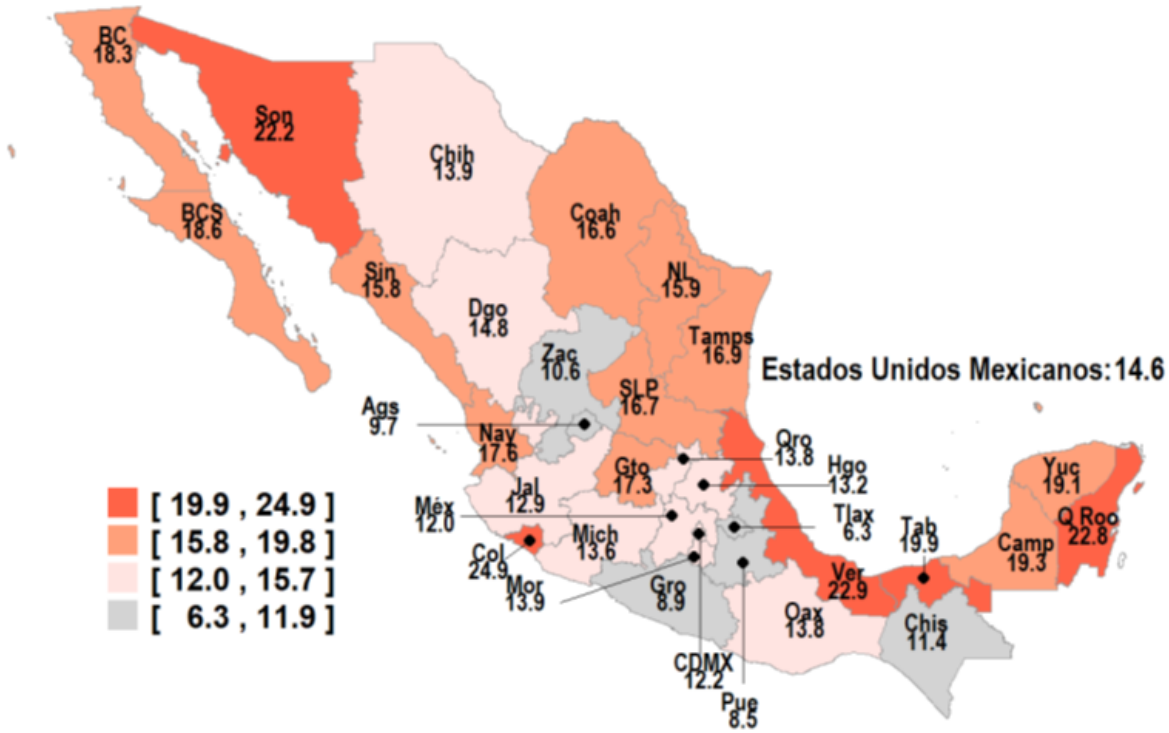


Figura 1.3: Porcentaje de población de 12 a 19 años de edad con obesidad, por entidad federativa (2018) (INEGI, 2019).

El INEGI informó que a nivel nacional, en 2018, el porcentaje de adultos de 20 años y más con sobrepeso y obesidad es de 75.2 % (39.1 % sobrepeso y 36.1 % obesidad), porcentaje que en 2012 fue de 71.3 por ciento. Al hacer un un análisis a nivel entidad de los porcentajes de la población de 12 a 19 años de edad con obesidad, de la Figura 1.3 se puede identificar que los estados con los porcentajes más altos son: Veracruz (22.9 %), Quintana Roo (22.8 %), Colima (24.9 %), Sonora (22.2 %) y Tabasco (19.9 %).

El síndrome metabólico agrupa varios factores de riesgo cardiovascular, sin embargo, de la fisiopatogenia del síndrome, la obesidad parece ser uno de los factores desencadenantes más importantes entre las otras alteraciones metabólicas que lo caracterizan. En otras palabras, en la mayoría de los casos la expresión del síndrome metabólico ocurre en individuos obesos y, además, en muchos casos, la expresión del síndrome metabólico es en buena medida una comorbilidad de la obesidad [19]. Por lo tanto, de la Figura 1.3, se identifica que aproximadamente el 15 % de los adolescentes mexicanos están en riesgo de padecer Síndrome Metabólico, por lo que la detección oportuna de este síndrome es imperativa para disminuir la probabilidad de que desarrollen en un futuro enfermedades como la diabetes y las ECV.

Diabetes en México

En 2018 murieron 101 mil 257 mexicanos por la diabetes mellitus. Es la segunda causa de muerte en México, solo por debajo de las ECV, de acuerdo con estadísticas del INEGI [20].

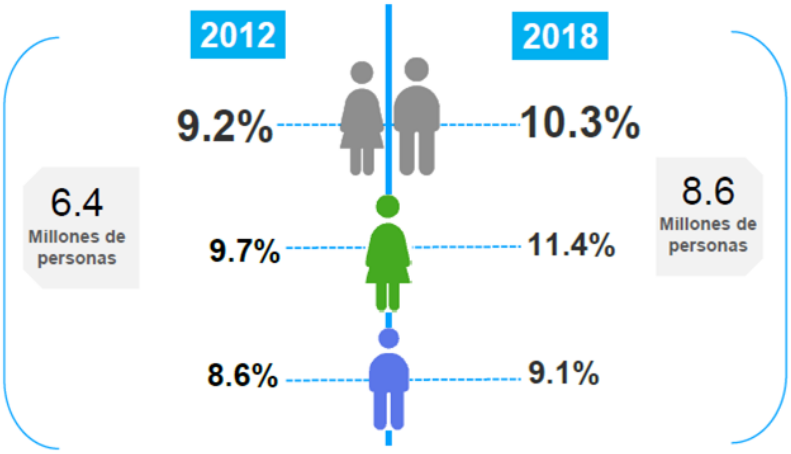


Figura 1.4: Porcentaje de la población de 20 años y más con diagnóstico médico previo de diabetes, por sexo (2012 *vs* 2018) (INEGI, 2019).

La diabetes ha aumentado (Figura 1.4) en paralelo a cambios sociales y culturales, entre los que se encuentran el envejecimiento de la población, aumento urbanístico, reducción de la actividad física, aumento en el consumo de azúcar y menor consumo de frutas y verduras [21].

De la Figura 1.5 se identifica que los estados con porcentajes más altos de personas con 20 años y más que han sido diagnosticadas con diabetes son: Campeche (14 %), Tamaulipas

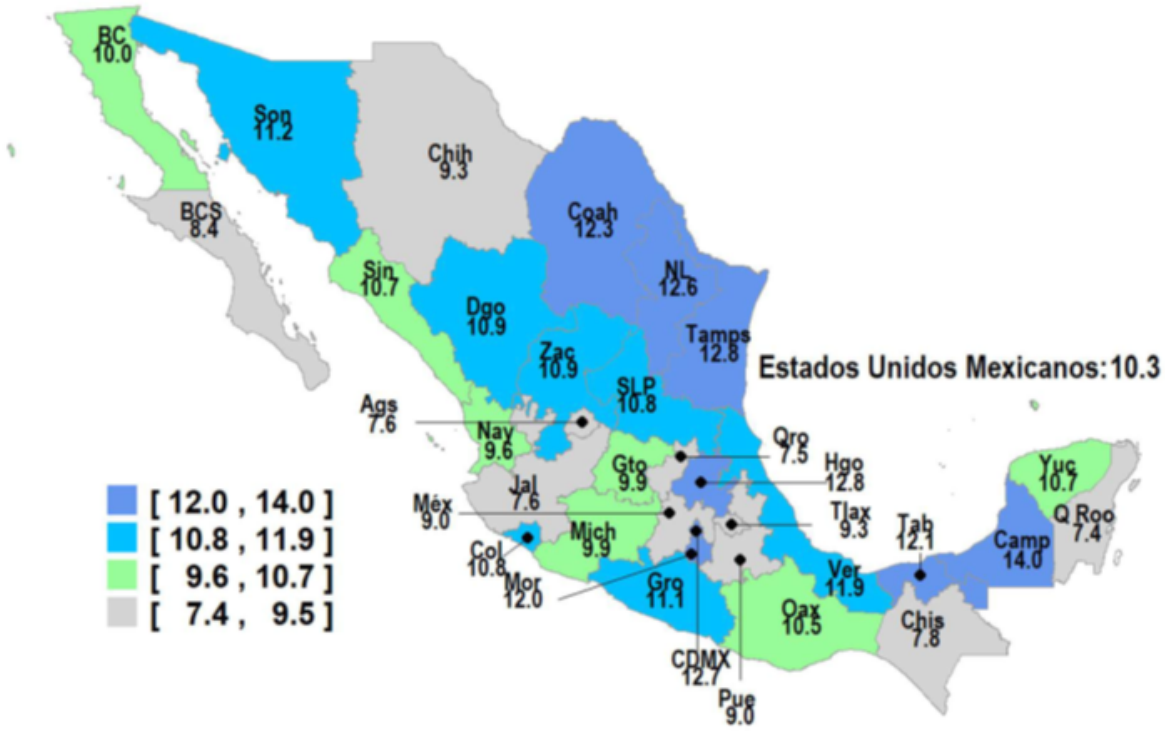


Figura 1.5: Porcentaje de la población de 20 años y más con diagnóstico médico previo de diabetes, por entidad federativa (2018) (INEGI, 2019).

(12.8 %), Hidalgo (12.8 %), CDMX (12.7 %) y Nuevo León (12.6 %); a nivel nacional se tiene un porcentaje de 10.3.

Hipertensión en México

En un comunicado de prensa, el Instituto Mexicano del Seguro Social (IMSS) informó que uno de cada tres mexicanos mayores de edad padece hipertensión arterial, una enfermedad crónica degenerativa cardiovascular que registra 7 millones de casos y provoca más de 50 mil muertes al año[4], por lo que el IMSS recomienda realizarse revisiones preventivas periódicas para detectarla oportunamente; se estima que hasta 80 por ciento de la población vive con esta enfermedad silenciosa, que no da síntomas hasta que la enfermedad esta avanzada o cuando si tienen complicaciones [22].

En la Figura 1.6 se puede ver el cambio porcentual de la población de 20 años y más que ha sido diagnosticada con hipertensión, compara lo que se obtuvo de la ENSANUT 2012 y de la ENSANUT 2018. Se puede observar que, aunque el cambio porcentual no es tan grande (1.8 %) la diferencia en personas lo hace alarmante (5.9 millones de personas).

De la Figura 1.7 identificamos que las entidades con porcentajes más altos de personas

[4]Estadísticas acotadas a los derechohabientes del IMSS

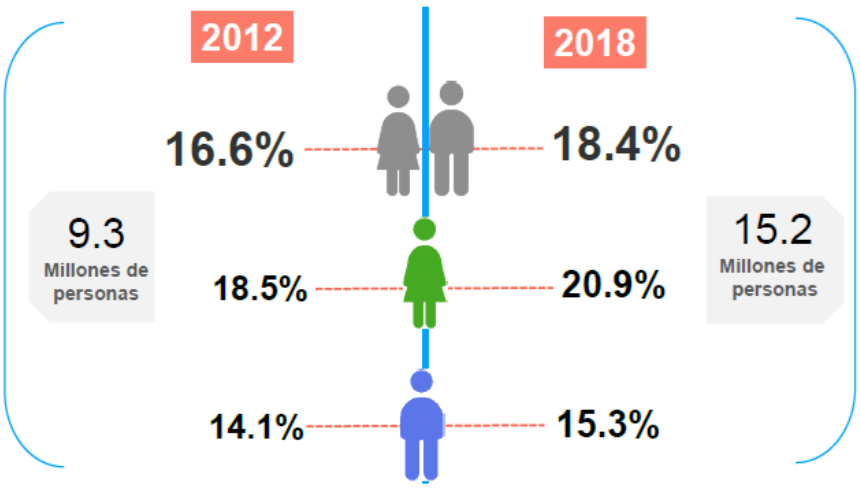


Figura 1.6: Porcentaje de la población de 20 años y más con diagnóstico médico previo de hipertensión, por sexo (2012 *vs* 2018) (INEGI, 2019).

diagnosticadas con hipertensión son: Campeche (21.7%), Sonora (24.6%), Veracruz (23.6%), Chihuahua (22.6%) y Coahuila (22.4%). A nivel nacional el 18.4% de la población ha sido diagnosticada con hipertensión.

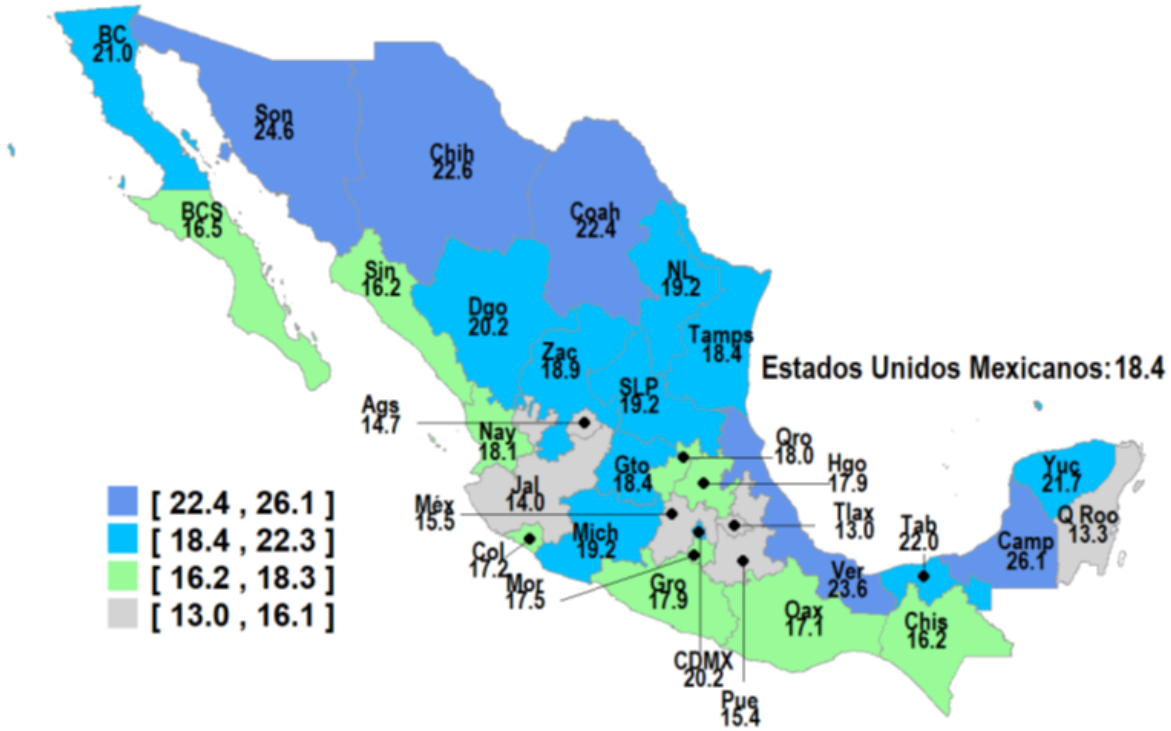


Figura 1.7: Porcentaje de población de 20 y más años de edad con diagnóstico médico previo de hipertensión, por entidad federativa (2018) (INEGI, 2019).

Un detalle que no se debe de olvidar al analizar los porcentajes presentados en las Figuras 1.4, 1.6, 1.5 y 1.7 es que únicamente representan el porcentaje de las personas de 20 años y más que ya han sido diagnosticadas con diabetes e hipertensión, respectivamente, por lo que no es el porcentaje real de las personas que padecen estas enfermedades ya que hay gente que no ha sido diagnosticada, pero que tiene el padecimiento.

Es importante mencionar, además del impacto en la salud de las personas que tiene el Síndrome Metabólico, las consecuencias que representan estas enfermedades para el Estado; a nivel macroeconómico, las enfermedades cardiovasculares (ECV) suponen una pesada carga para las economías de los países de ingresos bajos y medios, entre los cuales se encuentra México. Se calcula que, debido a la muerte prematura de muchas personas, las enfermedades no transmisibles, en particular las ECV y la diabetes, pueden reducir el Producto Interno Bruto (PIB)[5] hasta en un 6.77 % en los países de ingresos bajos y medios con un crecimiento económico rápido [9].

1.2. Ciclo de vida de la analítica

La analítica es un campo incluyente y multidisiplinario que utiliza las matemáticas, la estadística y los modelos de machine learning (ver Sección 1.3.) para el descubrimiento de patrones y conocimientos significativos de los datos, con el añadido de la comunicación, interpretación e implementación de los resultados obtenidos.

El ciclo de vida de la analítica (Figura 1.8) es un proceso iterativo e interactivo, en el cual debe participar personal con diferentes formaciones profesionales y habilidades en distintas etapas. A continuación, los pasos iterativos más detallados [23]:

- **Análisis del problema:** se centra en entender la necesidad, el alcance, las condiciones y la meta relacionados con el problema que se desea resolver, lo que llevará a la selección de una o más técnicas de modelado.
- **Preparación de los datos:** dependiendo de los métodos de análisis propuestos, este paso implica el uso de técnicas especializadas para localizar, acceder a, depurar y preparar los datos para lograr resultados óptimos.
- **Análisis exploratorio de los datos:** consiste en familiarizarse con los datos para identificar variables, tendencias y relaciones relevantes a través del uso de técnicas de estadística descriptiva.
- **Transformar y seleccionar datos:** se enfoca en la constucción del conjunto de datos final sobre los cuales se aplicarán los modelos, entre estas tareas se incluye la selección de variables y registros, así como su transformación, en caso de que sea sea necesario.

[5] El PIB es un indicador económico que refleja el valor monetario de todos los bienes y servicios finales producidos por un país o región en un determinado periodo de tiempo, normalmente un año. Se utiliza para medir la riqueza de un país.

La tabla resultante de este proceso es comúnmente conocida como ABT (Analytical Base Table).

- **Construcción del modelo:** generalmente existen diversas técnicas aplicables al mismo tipo de problema, en esta etapa se seleccionan y aplican varias técnicas de modelado y se calibran sus parámetros en los valores óptimos.
- **Validación del modelo:** se evalúa el rendiemiento de los modelos y se selecciona el que satisface de mejor manera los objetivos planteados en la primera fase.
- **Implementación del modelo:** una vez seleccionado el modelo campeón se pone en producción, es decir, el modelo se aplica a nuevos datos para generar insights predictivos.
- **Monitoreo de resultados:** se monitorea el desempeño predictivo del modelo para garantizar que esté actualizado y entregue resultados válidos. Si se degrada el desempeño del modelo, se debe de decidir si se tiene que recalibrar el modelo o ponerlo a competir contra modelos challenger.

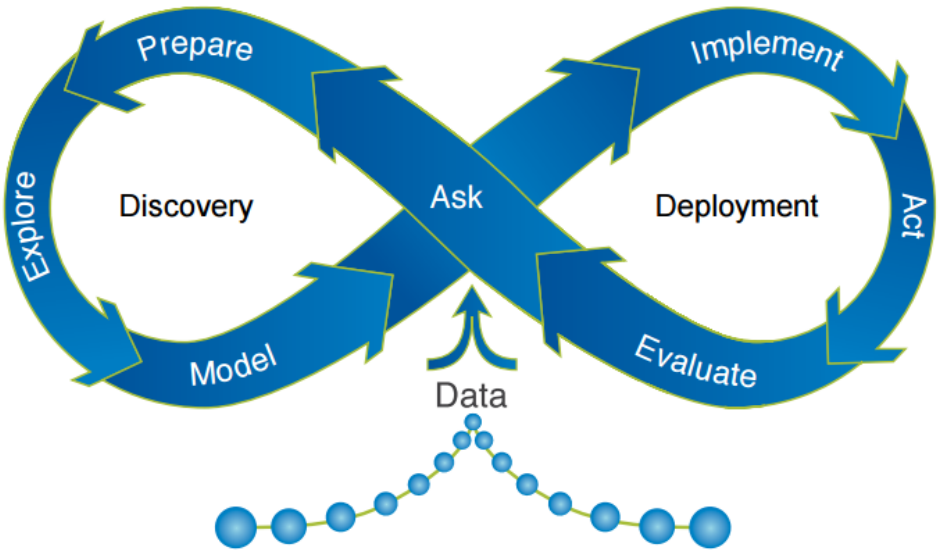


Figura 1.8: Ciclo de vida analítico (SAS Institute, 2020).

1.3. Machine Learning

La definición de la palabra “aprendizaje”, acorde a la Real Academia Española (RAE), es “acción y efecto de aprender” y menciona que para la psicología es la “adquisición por la práctica de una conducta duradera” [24]; por otro lado, la RAE define el verbo “aprender” como la acción de “adquirir el conocimiento de algo por medio del estudio o de la experiencia” [25]. De lo anterior se puede llegar a la conclusión de que el aprendizaje de máquina (del

inglés, machine learning) consiste en la acción, por parte de las computadoras, de adquirir conocimiento relacionado a algún fenómeno. Actualmente se considera que el aprendizaje de máquina es una rama de la Inteligencia Artificial[6] y que tiene como objetivo el desarrollo e implementación de modelos analíticos que le permita a las computadoras aprender de los datos para la toma decisiones.

El aprendizaje de máquina, también conocido como aprendizaje automático o aprendizaje automatizado, esta enfocado en la implementación de modelos matemáticos como algoritmos que le permitan a la computadora identificar patrones e información no trivial dentro de grandes volúmenes de datos. Estos algoritmos son conocidos como "algoritmos de aprendizaje" y le permiten a la computadora aprender de los datos sin necesidad de programar cada paso del camino.

El proceso básico del aprendizaje automático es proporcionar datos de entrenamiento a un algoritmo de aprendizaje. El algoritmo de aprendizaje genera un nuevo conjunto de reglas, basado en inferencias de los datos. En esencia, esto genera un nuevo algoritmo, denominado formalmente el modelo de aprendizaje automático. Al usar diferentes datos de entrenamiento, el mismo algoritmo de aprendizaje podría usarse para generar diferentes modelos. Por ejemplo, el mismo tipo de algoritmo de aprendizaje podría usarse para enseñarle a la computadora cómo traducir idiomas o predecir el mercado de valores [27].

El aprendizaje automatizado no es un campo del conocimiento nuevo, muchos de los algoritmos de aprendizaje utilizados en la actualidad se basan en investigaciones que tienen décadas de antigüedad, como, por ejemplo, las redes neuronales; fue en 1943 cuando el neurofisiólogo Warren McCulloch y el matemático Walter Pitts escribieron un artículo sobre cómo podrían funcionar las neuronas en el cerebro y para ilustrar su teoría, modelaron una red neuronal simple utilizando circuitos eléctricos. A medida que las computadoras se hicieron más avanzadas en la década de los 50's, finalmente fue posible aplicar la primera red neuronal a un problema del mundo real cuando en 1959 Bernard Widrow y Marcian Hoff, de Stanford, utilizaron una red neuronal para crear un filtro adaptativo que elimina los ecos en las líneas telefónicas conocido como MADELINE (Multiple ADAptive LINear Elements) [28].

Desde ese entonces, el término "Machine Learning" ha crecido en popularidad (Figura 1.9) al igual que el uso de este tipo del algoritmos en diferentes tipos de industrias. Este aumento en el uso actual del aprendizaje automático está vinculado a la evolución en dos áreas importantes:

- **Poder de cómputo:** las computadoras poderosas y la capacidad de conectar el poder de procesamiento remoto a través de Internet (cloud computing) hacen posible que las técnicas de aprendizaje automático procesen enormes cantidades de datos y modelos cada vez más robustos.

- **Disponibilidad de datos:** el aumento en el uso de medios digitales, dispositivos mobiles,

[6] La Inteligencia Artificial se refiere a la teoría y el desarrollo de sistemas informáticos capaces de realizar tareas que normalmente requieren inteligencia humana, como la percepción visual, el reconocimiento de voz, la toma de decisiones y la traducción entre idiomas [26].

redes sociales, internet, etc. ha provocado una generación masiva de información, ver Figura 1.10 (un zettabyte equivale a 1000 millones de terabytes [29]).

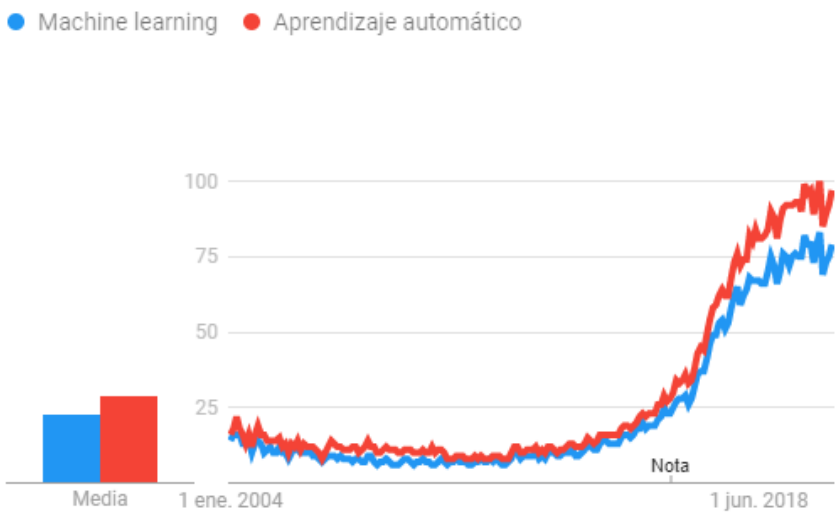


Figura 1.9: Tendencia de las búsquedas en Google de los términos “Machine Learning” y “Aprendizaje Automático” a lo largo del tiempo mostrando la popularidad de los términos (Google Trends, 2020).

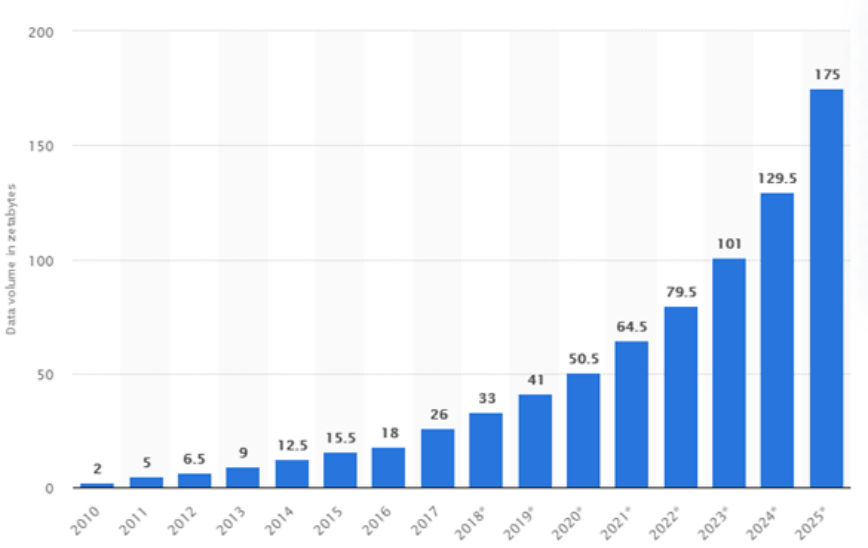


Figura 1.10: Volumen de datos creados en todo el mundo desde 2010 hasta 2025 (en zettabytes) (Statista, 2019).

A continuación, se explican los diferentes métodos de aprendizaje en los que se puede clasificar los modelos de Machine Learning.

1.3.1. Aprendizaje no supervisado

Los algoritmos de aprendizaje automático no supervisados detectan patrones, previamente desconocidos, dentro de un conjunto de datos no etiquetados, es decir, sin referencia a resultados conocidos, estos patrones permiten descubrir la estructura subyacente de los datos.

Algunas aplicaciones de los algoritmos de aprendizaje automático no supervisados incluyen:

- **Clustering:** busca segmentar automáticamente el conjunto de datos de tal manera que los miembros del mismo grupo sean similares entre si y distintos con respecto a los miembros de los demás grupos.
- **Detección de anomalías:** permite descubrir automáticamente datos inusuales dentro del conjunto de datos, es decir, datos que no obedezcan cabalmente el comportamiento (distribución) del resto de conjunto de datos.
- **Reglas de asociaciones:** identifican conjuntos de elementos que con frecuencia ocurren juntos dentro del conjunto de datos.
- **Selección de variables:** se usan comúnmente para el preprocesamiento de datos, y sirven para reducir el número de características en un conjunto de datos (reducción de dimensionalidad) o descomponer el conjunto de datos en múltiples componentes.

Los patrones descubiertos usando modelos de aprendizaje automático no supervisados también pueden ser útiles al momento implementar modelos de aprendizaje automático supervisados. Por ejemplo, un modelo de detección de fraude que utiliza el puntaje resultante de un modelo de detección de anomalías como una característica adicional para el monitoreo. Otro ejemplo, se puede usar una técnica no supervisada para realizar la segmentación de los datos, luego usar el segmento al que pertenece cada elemento como una característica adicional en el modelo de aprendizaje supervisado [30] [31].

1.3.2. Aprendizaje supervisado

Se llama *aprendizaje supervisado* porque el proceso a través del cual un algoritmo aprende, a partir del conjunto de datos de entrenamiento, puede considerarse como tener un profesor que supervise todo el proceso. El conjunto de datos de entrenamiento se conforma de variables de entrada ($X = x_1, ..., x_n$) con su correspondiente variable de salida (Y). Durante el entrenamiento, el algoritmo buscará patrones en los datos de entrada que se asocien con los resultados deseados, hace predicciones iterativas sobre los datos de entrenamiento y es corregido por el profesor. El aprendizaje se detiene cuando el algoritmo alcanza un nivel aceptable de rendimiento [32] [33].

En su forma más básica, un algoritmo de aprendizaje supervisado se puede escribir simplemente como:

$$Y = f(x_1, ..., x_n)$$

Donde Y es el valor pronosticado, el cual está determinado por una función de mapeo que asigna una clase a un valor de entrada X. El modelo de aprendizaje automático crea la función utilizada para conectar las características de entrada a una salida con base en lo aprendido durante el entrenamiento [33].

El aprendizaje supervisado se puede dividir en dos subcategorías: clasificación y regresión.

1.3.2.1. Modelos de clasificación

Se utiliza un modelo supervisado de clasificación cuando la variable de salida es una categoría, como por ejemplo, si una persona tiene síndrome metabólico o no (1 y 0, respectivamente). En este caso se tendría:

$$y = f(x_1, ..., x_n) \qquad\qquad y \in \{0, 1\}$$

Durante el entrenamiento, un algoritmo de clasificación recibe observaciones de datos con una categoría asignada, con el objetivo de que el algoritmo después reciba nuevas observaciones y les asigne una clase o categoría en la que se ajuste según los datos de entrenamiento proporcionados [33].

Los problemas de clasificación se pueden resolver con una gran cantidad de algoritmos. El algoritmo que se elija utilizar depende de los datos y la situación. A continuación, se muestran algunos algoritmos de clasificación:

- Clasificadores lineales, en inglés conocidos como *Linear Classifiers*.
- Árboles de decisión, en inglés conocidos como *Decision Trees*.
- Bosques aleatorios, en inglés conocidos como *Random Forest*.
- Redes neuronales, en inglés conocidas como *Neural Networks*.
- Máquinas de soporte vectorial, en inglés conocidas como *Support Vector Machines* (SVM).

1.3.2.2. Modelos de regresión

Se utiliza un modelo supervisado de regresión cuando la variable de salida es un número real, como por ejemplo, la cantidad de triglicéridos en la sangre de una persona. En este caso se tendría:

$$y = f(x_1, ..., x_n) \qquad\qquad y \in [60, 1500]$$

A continuación, se muestran algunos algoritmos de regresión [33] [34]:

- Regresión lineal, en inglés conocida como *Linear Regression*.
- Regresión polinomial, en inglés conocida como *Polynomial Regression*.
- Árbol de regresión, en inglés conocido como *Decision Tree Regression*.

- Regresión basada en redes neuronales, en inglés conocida como *Neural Network Regression*.

- Bosques alaetorios de regresión, en inglés conocidos como *Random Forest Regression*.

1.3.3. Cómo seleccionar el algoritmo de Machine Learning a utilizar

Una pregunta típica cuando uno se encuentra frente a la amplia variedad de algoritmos de machine learning es: "¿qué algoritmo debo utilizar?". La respuesta a esta pregunta varía dependiendo de varios factores, que incluyen:

- El tamaño, la calidad y la naturaleza de los datos.

- El tiempo y poder computacional disponible.

- La urgencia de la tarea.

- Qué se desea hacer con los datos.

A continuación, se presenta una guía [35] que ayuda a identificar el tipo de algoritmo que se debe utilizar, cabe mencionar que no es el enfoque único pero ayuda como un buen punto de partida a la hora de trabajar con modelos de machine learning.

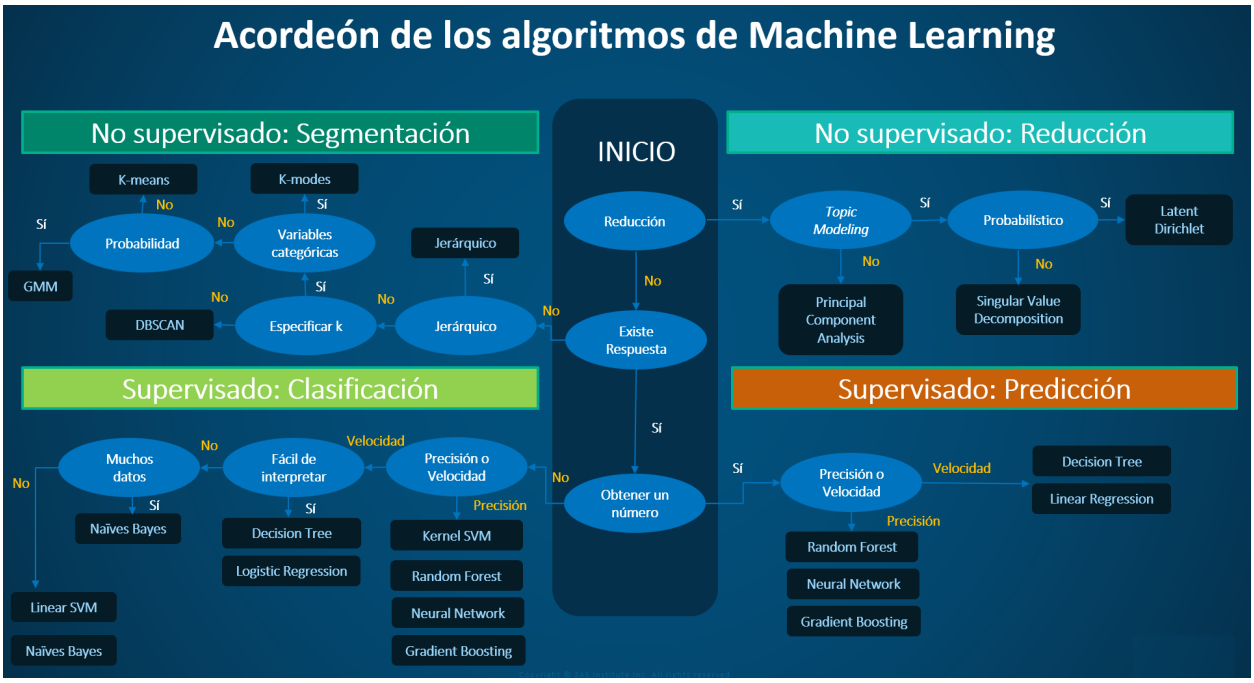


Figura 1.11: Acordeón de los algoritmos de Machine Learning (SAS Institute, 2017).

Cómo usar el acordeón de los algoritmos de Machine Learning

Lea las etiquetas en el caminito y algoritmo en el Figura 1.11 como “Si <etiqueta>, entonces use <algoritmo>”. Por ejemplo:

- Si se desea realizar una reducción de dimensión, entonces utilizar el análisis de componentes principales.
- Si se necesita una predicción numérica rápidamente, entonces usar árboles de decisión o regresión lineal.
- Si se necesita un segmentación jerárquica, entonces utilizar la agrupación en clústeres jerárquica.

Es importante recordar que estas guías están destinadas a ser recomendaciones prácticas (reglas basadas en experiencia), por lo que algunas de las recomendaciones no son exactas, la única forma seguro de haber encontrado el mejor algoritmo es habiendo probado todos o probar una combinación de más de una rama (presentadas en la Figura 1.11) .

Para cumplir con los Objetivos planteados se identifica la necesidad de construir 4 modelos de aprendizaje supervisado: dos modelos de clasificación para identificar si el paciente tiene (o no) los valores de triglicéridos y colesterol alterados; y dos modelos de regresión para estimar los valores de triglicéridos y colesterol que tendría el paciente. Haciendo uso del acordeón presentado en la Figura 1.11, se debe de decidir si lo que se busca es velocidad o precisión, en este caso, por trarse de datos médicos lo más importante es la precisión de los modelos pero también se busca que los modelos no sean tan complejos para poder transmitir los resultados a un gremio cuya especialidad no son los modelos de machine learning, es por ello que se decidió hacer uso de los algoritmos basados en árboles (en particular: decision tree, random forest, gradient boosting y XGBoost) los cuales, además, aplican para ambos objetivos, es decir, para los modelos de regresión y los modelos de clasificación.

1.4. Algoritmos basados en árboles

Los algoritmos de machine learning basados en árboles engloban a un conjunto de técnicas supervisadas que consiguen segmentar el espacio de los predictores en regiones simples, dentro de las cuales es más sencillo manejar las interacciones y predecir la variable de respuesta [36]; estos modelos utilizan una serie de reglas del tipo “Si, entonces” para generar predicciones a partir de uno o más árboles de decisión. Todos los modelos basados en árboles se pueden utilizar para resolver problemas de clasificación (predicción de valores categóricos) y de regresión (predicción de valores numéricos) [37].

A continuación, se hace una introducción a la parte teórica de 4 modelos de machine learning basados en árboles:

- **Decision trees**: son la base de todos los modelos basados en árboles.

- **Random forest**: es un método de ensamble que construye muchos árboles de decisión en paralelo.
- **Gradient boosting**: es un método de ensamble que construye muchos árboles de decisión secuencialmente.
- **XGBoost**: es una implementación del modelo de gradient boosting diseñada para la mejorar la velocidad y el rendimiento.

Los algoritmos basados en árboles son de los modelos de aprendizaje supervisado más utilizados para todo tipo de problemas de Machine Learning ya que, a diferencia de los modelos lineales, mapean bastante bien las relaciones no lineales y, además, son modelos que facilitan su interpretación y pueden llegar a tener una muy alta precisión y estabilidad [38].

Un método de ensamble es un proceso en el que se crean múltiples modelos diversos para predecir un resultado, ya sea mediante el uso de muchos algoritmos de modelado diferentes o el uso de diferentes conjuntos de datos de entrenamiento. El ensamble luego agrega la predicción de cada modelo base y da como resultado una predicción final única para los nuevos datos analizados. La motivación para usar ensambles es mejorar la generalización de la predicción. Siempre que los modelos base sean diversos e independientes, el error de predicción del modelo disminuye cuando se utilizan ensambles. El enfoque busca la sabiduría de las multitudes al hacer una predicción. Aunque el ensamble tiene varios modelos base dentro del modelo, actúa y funciona como un solo modelo [39].

1.4.1. Árboles de decisión

El árbol de decisión es uno de los algoritmos de aprendizaje supervisado más populares y utilizados para resolver toda clase de problemas de Machine Learning, esto debido a la tolerancia que tienen a los valores nulos, su interpretabilidad, el manejo de variables predictivas redundantes e irrelevantes, su bajo costo computacional, entre otras [40]. Además, tienen la ventaja de que pueden manejar variables de entrada y salida tanto categóricas como continuas por lo que pueden ser adaptados para resolver problemas de clasificación y de regresión.

Ventajas

1. Son fáciles de interpretar, aún cuando las relaciones entre predictores son complejas. Su estructura se asemejan a la forma intuitiva en que clasificamos y predecimos las personas, además, no se requieren conocimientos estadísticos para comprenderlos.
2. Pueden manejar tanto predictores cuantitativos como cualitativos sin tener que crear variables “dummy” .
3. No es necesario que los datos cumplan ningún tipo de distribución específica o supuestos.
4. No se ven muy influenciados por los valores atípicos.
5. Son capaces de seleccionar predictores de forma automática.

6. Son muy útiles en la exploración de datos, permiten identificar de forma rápida y eficiente las variables más importantes [41].

Desventajas

1. La capacidad predictiva de los modelos de regresión y clasificación basados en un único árbol es bastante inferior a la conseguida con otros modelos debido a su tendencia a sobreajustarse[7]. Sin embargo, existen técnicas más complejas que, haciendo uso de la combinación de múltiples árboles (random forest, boosting, etc.), consiguen disminuir en gran medida este problema.

2. Cuando tratan con variables continuas, pierden parte de su información al categorizarlas en el momento de la división de los nodos. Por esta razón, suelen ser modelos que consiguen mejores resultados en clasificación que en regresión [41].

1.4.1.1. Árboles de clasificación vs Árboles de regresión

Ambos árboles funcionan de manera similar, sin embargo, es importante mencionar las principales diferencias y similitudes entre los árboles de clasificación y regresión [38]:

- Los árboles de regresión se utilizan cuando la variable dependiente es continua. Los árboles de clasificación se utilizan cuando la variable dependiente es categórica.

- En el caso del árbol de regresión, el valor obtenido es la respuesta media de las observaciones de los datos de entrenamiento que caen en esa región (nodos terminales). Por lo tanto, si una nueva observación de datos cae en cierta región, su predicción sería la media de esa región.

- En el caso del árbol de clasificación, el valor (clase) obtenido es la moda de las observaciones de los datos de entrenamiento que caen en esa región (nodos terminales). Por lo tanto, si una nueva observación de datos cae en cierta región, su predicción sería la moda de esa región.

- Ambos árboles dividen el espacio de predicción (variables independientes) en regiones distintas y no superpuestas.

- Ambos árboles siguen un proceso conocido como división binaria recursiva. Este proceso va “de arriba hacia abajo” porque comienza desde la parte superior del árbol (root node), cuando todas las observaciones están disponibles en una sola región, y divide sucesivamente el espacio de predicción en dos nuevas ramas hacia abajo del árbol. Además, es un algoritmo que se enfoca en buscar la mejor variable disponible para hacer la división. Este proceso de división continúa hasta que se alcanza un criterio de parada definido. Por ejemplo, se le puede especificar al algoritmo que se detenga una vez que el número de observaciones por nodo sea inferior a 50.

[7] El sobreajuste (overfitting) ocurre cuando un modelo aprende los detalles y el ruido en los datos de entrenamiento de tal manera que impacta negativamente el rendimiento del modelo en nuevos datos, es decir, no generaliza bien.

- En ambos casos, el proceso de división da como resultado árboles completamente desarrollados hasta que se alcanza el criterio de parada. Pero, es probable que el árbol completamente desarrollado sobreajuste los datos, lo que lleva a una precisión deficiente al momento de calificar nuevos datos. Para hacer frente al sobreajuste se puede utilizar una técnica conocida como poda.

1.4.1.2. Terminología de los algoritmos basados en árboles

Antes de comenzar a utilizar algoritmos basados en árboles es importante definir algunos conceptos básicos de los árboles de decisión[8] [42]:

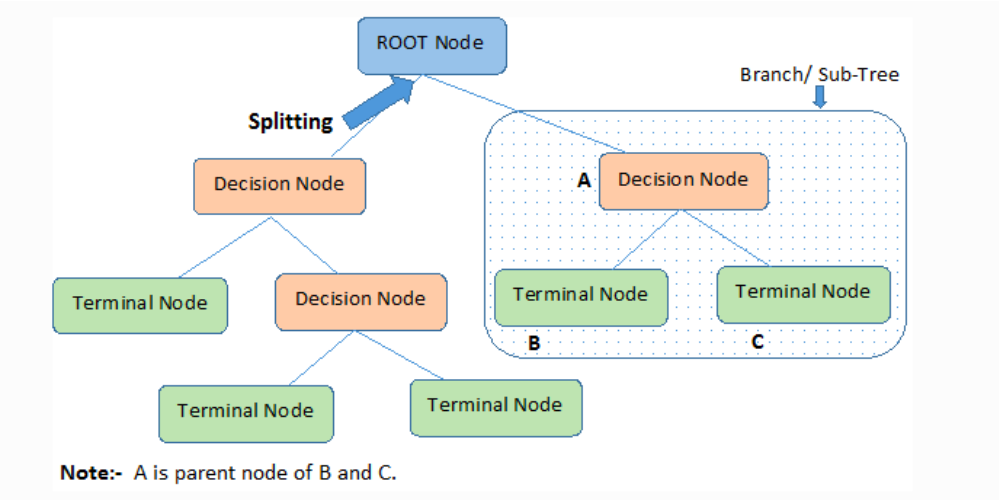


Figura 1.12: Conceptos básicos de los árboles de decisión (Analytics Vidhya, 2016).

1. **Root node**: es el nodo inicial, en el cual esta contenida toda la población (muestra) para posteriormente ser dividido en dos o más conjuntos.
2. **Splitting**: se refiere al proceso de dividir un nodo en dos o más subnodos.
3. **Decision node**: cuando un subnodo se divide en otros subnodos, este se denomina nodo de decisión.
4. **Leaf node / Terminal node**: los nodos que no se dividen se denominan nodos terminales, mejor conocidos como hojas.
5. **Pruning**: se refiere al proceso de eliminar subnodos de un nodo de decisión. Se puede considerar como el proceso opuesto al "splitting", en español se le conoce como poda del árbol.
6. **Branch / Sub-tree**: se refiere a una subsección del árbol, en español se le conoce como rama o sub-árbol.

[8]Se mencionan los conceptos en inglés ya que la sintaxis del lenguaje de programación Python esta en ese idioma.

7. **Parent and Child node**: un nodo que se divide en subnodos se denomina nodo padre, a su vez, los subnodos se denominan nodos hijos.

A continuación, se desarrolla el algoritmo matemático detrás del proceso de ”splitting” de los árboles de decisión, el cual explica cómo se llega del nodo inicial a las hojas.

1.4.1.3. Planteamiento matemático

Dados los vectores (observaciones) de entrenamiento $x_i \in \mathbb{R}^n$, con $i = 1, 2, ..., l$ y un vector de etiquetas (label vector) $y \in \mathbb{R}^l$, un árbol de decisión divide de forma recursiva el espacio de características (variables input) de tal forma que las observaciones con las mismas etiquetas o valores objetivo similares se agrupen [43].

Sean los datos dentro del nodo m representados por Q_m con N_m observaciones. Cada posible división $\theta = (j, t_m)$ que consta de la característica j y umbral t_m, particiona los datos en los subconjuntos:

$$\begin{aligned} Q_m^{left}(\theta) &= \{(x, y) | x_j <= t_m\} \\ Q_m^{right}(\theta) &= Q_m \setminus Q_m^{left}(\theta) \end{aligned} \tag{1.1}$$

Lo que sigue es medir la calidad de las posibles divisiones del nodo m usando una función de impureza o función de pérdida $H()$, cuya elección depende de la tarea a resolver (clasificación o regresión):

$$G(Q_m, \theta) = \frac{N_m^{left}}{N_m} H(Q_m^{left}(\theta)) + \frac{N_m^{right}}{N_m} H(Q_m^{right}(\theta)) \tag{1.2}$$

Se seleccionan los parámetros que minimizan la impureza:

$$\theta^* = \text{argmin}_\theta \, G(Q_m, \theta) \tag{1.3}$$

Este proceso de hace de forma recursiva para todos los subconjuntos $Q_m^{left}(\theta^*)$ y $Q_m^{right}(\theta^*)$ hasta que se alcance la profundidad máxima permitida, $N_m < \text{mín}_{samples}$ o $N_m = 1$.

Criterio de clasificación

Si el objetivo es una clasificación que toma valores $0, 1, ..., K-1$, para el nodo m:

$$p_{mk} = \frac{1}{N_m} \sum_{y \in Q_m} I(y = k) \tag{1.4}$$

es la proporción de observaciones de clase k en el nodo m. Si se trata un nodo terminal, entonces la probabilidad predicha por el árbol para esta región se calcula en p_{mk} [43].

Las medidas más comunes de impureza para las tareas de clasificación son las siguientes:

- Gini:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

- Entropía:

$$H(Q_m) = -\sum_k p_{mk} \log(p_{mk})$$

- Clasificación erronea:

$$H(Q_m) = 1 - \max(p_{mk})$$

Criterio de regresión

Si el objetivo es un valor continuo entonces, para el nodo m, los criterios que se utilizan para minimizar la función de pérdida que determina la ubicación de las siguientes divisiones son el error cuadrático medio (MSE - Mean Squared Error) y el error absoluto medio (MAE - Mean Absolute Error). Para determinar el valor predicho, el MSE utiliza la media, $\bar{y}_m$, del nodo terminal mientras que el MAE utiliza la mediana, $median(y)_m$, del nodo terminal [43].

- Error cuadrático medio:

$$\begin{aligned} \bar{y}_m &= \frac{1}{N_m} \sum_{y \in Q_m} y \\ H(Q_m) &= \frac{1}{N_m} \sum_{y \in Q_m} (y - \bar{y}_m)^2 \end{aligned} \tag{1.5}$$

- Error absoluto medio:

$$\begin{aligned} median(y)_m &= \underset{y \in Q_m}{\text{median}}(y) \\ H(Q_m) &= \frac{1}{N_m} \sum_{y \in Q_m} |y - median(y)_m| \end{aligned} \tag{1.6}$$

1.4.1.4. Parámetros de los algoritmos basados en árboles

Además de conocer la lógica detrás del algoritmo de aprendizaje de los árboles de decisión, es importante comprender el papel de los parámetros utilizados en el modelo. Los parámetros utilizados para construir un árbol de decisión en Python se explican con más detalle a continuación [44]:

- **max_depth**: se refiere a la profundidad vertical máxima de un árbol y sirve como criterio de parada para el algoritmo[9].

- **min_samples_leaf**: se refiere a la cantidad mínima de observaciones contenidas en un nodo terminal (u hoja) y sirve como criterio de parada para el algoritmo.

- **n_estimators**: se refiere al número de árboles que serán considerados dentro de los ensambles.

[9]Este parámetro puede ser entendido como el máximo número de decisiones (divisiones) que se toman para llegar del nodo inicial a una hoja.

Los árboles de decisión tienen la deficiencia, por la forma en la que se contruyen, de tener un sesgo si dominan algunas clases. Por lo tanto, se recomienda equilibrar el conjunto de datos antes de ajustarlo al árbol de decisiones [41], para ello se pueden utilizar los siguientes parámetros con lo cuales el algoritmo se encarga de equilibrar a la case minoritaria durante el entrenamiento a través de la asignación de pesos a las clases, a este proceso se le conoce como compensación por penalización [45].

- **class_weight** : se utiliza con los árboles de decisión y con el random forest.
- **scale_pos_weight** : se utiliza con el XGBoost[10].

Los siguientes son los parámetros con los que el algoritmo del árbol de decisión decide cómo y dónde dividir (splitting) [44]:

- **max_features**: se refiere a la cantidad de las variables de entrada que serán consideradas al momento de buscar la mejor división.
- **splitter**: se refiere a la estrategia utilizada para elegir la mejor división (split) en cada nodo. Las estrategias admitidas son "best" para elegir la mejor división y "random" para elegir la mejor división aleatoria.
- **criterion**: se refiere a la función para medir la calidad de una división. Los criterios admitidos son Gini y Entropía.

Para seleccionar los mejores hiperparámetros para los modelos que se van a desarrollar en el presente trabajo se utilizará un método de optimización de hiperparámetros (en inglés conocido como *hyperparameter tuning*) llamado "RandomizedSearchCV", disponible en la librería Scikit-learn de Pyhon. Este método lo que hace es buscar objetivamente los valores de los hiperparámetro del modelo para elegir aquellos que den como resultado un modelo con el mejor rendimiento en un conjunto de datos determinado [46].

Para evitar que el modelo se sobreajuste, este método implementa un procedimiento llamado "cross-validation" (CV). En el enfoque básico, llamado "k-fold CV", el conjunto de entrenamiento se divide en *k* conjuntos más pequeños de tal manera que el modelo sea entrenado usando *k-1* subconjuntos (folds) y que se evaluado con el subconjunto restante, esto se lleva a cabo *k* veces, y las métricas de precisión se obtienen como el promedio de los valores reportados en las iteraciones [47].

1.4.2. Ensambles

Un ensamble parte de la premisa de que "dos cabezas piensan mejor que una" y lo que busca es que muchas predicciones débiles se unan para generar una predicción fuerte, de manera más formal, un ensamble es un grupo de modelos base entrenados por diferentes métodos o algoritmos, combinados para producir una predicción final [48]. Su objetivo es mejorar mejorar la generalización y robustez que se obtendría con un solo modelo. Para el caso de los árboles de

[10]No existe un parámetro similar para el gradient boosting

decisión, si bien existen métodos que ayudan a mejorar el rendimiento predictivo de un árbol, un solo árbol generalmente no producirá predicciones sólidas por sí solo. Para mejorar el poder predictivo del modelo, se pueden construir muchos árboles y combinar las predicciones finales.

Dos de los algoritmos de ensamble más populares son el bosque aleatorio (Random Forest) y el aumento de gradiente (Gradient Boosting) que son bastante potentes y se usan comúnmente en las diferentes aplicaciones del aprendizaje automático.

1.4.2.1. Bagging y Random Forest

En los algoritmos de ensamble, los métodos de "bagging" (bootstrap aggregating) forman una clase de algoritmos que construyen varios modelos usando diferentes subconjuntos aleatorios del conjunto de entrenamiento original y luego agregan sus predicciones individuales para formar una predicción final. Estos métodos se utilizan como una forma de reducir la varianza de un estimador base (por ejemplo, un árbol de decisión), al introducir la aleatorización en su procedimiento de construcción y luego hacer un ensamble a partir de los modelos resultantes.

En muchos casos, los métodos de bagging constituyen una forma muy sencilla de mejorar con respecto a un solo modelo, sin que sea necesario adaptar el algoritmo base subyacente. Dado que proporcionan una forma de reducir el sobreajuste, los métodos de bagging funcionan mejor con modelos sólidos y complejos (por ejemplo, árboles de decisión completamente desarrollados).

Existen diferentes métodos de "bagging", cuando los estimadores base se construyen en subconjuntos de muestras y características (variables), entonces el modelo resultante se conoce como Random Forest [49]. En los bosques aleatorios, cada árbol del ensamble se construye a partir de una muestra extraída con reemplazo (i.e., bootstrap sample) del conjunto de entrenamiento. Además, al dividir cada nodo durante la construcción de un árbol, la mejor división se encuentra entre todas las variables de entrada o un subconjunto aleatorio de tamaño "max_features" (ver Sección 1.4.1.4.).

El propósito de estas dos fuentes de aleatoriedad es disminuir la varianza del bosque aleatorio. De hecho, los árboles de decisión individuales suelen presentar una gran variación y tienden a sobreajustarse. La aleatoriedad inyectada en los bosques produce árboles de decisión con errores de predicción algo desacoplados. Al tomar un promedio de esas predicciones[11], algunos errores pueden anularse, por lo que, en general, se obtiene un mejor modelo [50].

Ventajas

1. Bueno para manejar relaciones complejas y no lineales en los datos.
2. Maneja bien los conjuntos de datos de alta dimensionalidad (muchos atributos).
3. Se pueden entrenar rápidamente dado que los árboles subyacentes no dependen unos de otros y se pueden entrenar en paralelo.

[11] La implementación de Scikit-learn combina clasificadores promediando su predicción probabilística.

Desventajas

1. La principal desventaja es que, a diferencia de los árboles de decisión, los random forests son difíciles de interpretar dado que la decisión final del modelo considera las decisiones tomadas (predicciones) por los n árboles que componen al bosque aleatorio.

2. Si existe una proporción alta de atributos irrelevantes, el random forest puede verse perjudicado, sobre todo a medida que se reduce el número de atributos evaluados [36].

1.4.2.2. Boosting y Gradient Boosting

El boosting es un método de ensamble de árboles de decisión que construye árboles pequeños consecutivos, a menudo de solo un nodo, con cada árbol enfocado en corregir el error neto del árbol anterior. En otras palabras, lo que hace este método es dividir (split) el primer árbol usando el atributo más predictivo y luego actualiza las ponderaciones de los atributos para asegurarse de que el árbol subsecuente se divida usando el atributo que le permita clasificar correctamente las observaciones que fueron mal clasificadas en el árbol anterior. El siguiente árbol se enfocará en clasificar correctamente los errores de ese árbol, y así sucesivamente [38].

El Gradient Boosting es la extensión más popular del boosting y utiliza el algoritmo de descenso de gradiente para la optimización. El descenso de gradiente es un algoritmo de optimización que se utiliza para minimizar alguna función moviéndose iterativamente en la dirección del descenso más pronunciado (negativo del gradiente) [51], en el caso del Gradient Boosting el objetivo es ir minimizando el error iteración a iteración.

Ventajas

1. Bueno para manejar relaciones complejas y no lineales en los datos.

2. Son buenos para modelar con datos con clases desequilibradas.

3. Son potentes y precisos, en algunos casos incluso más que los Random Forest.

Desventajas

1. La principal desventaja es que, a diferencia de los árboles de decisión, los gradient boosting son difíciles de interpretar dado que la decisión final del modelo considera las decisiones tomadas (predicciones) por los n árboles contruidos por el modelo.

2. Más lento de entrenar, ya que los árboles deben construirse secuencialmente.

3. Más difícil de ajustar los hiperparámetros.

1.4.2.3. Planteamiento matemático

El algoritmo "Gradient Tree Boosting" o "Gradient Boosted Decision Trees" (GBDT) es una generalización del boosting que permite la optimización arbitraria de funciones de pérdida diferenciables [52]. GBDT es un procedimiento preciso y eficaz que se puede utilizar tanto para problemas de regresión como de clasificación.

A continuación, se muestra primero el algoritmo GBDT para tareas de regresión y, posteriormente, para tareas de clasificación [53].

Regresión

Los regresores GBRT son modelos aditivos cuya predicción y_i para un vector de entrada x_i es de la siguiente forma:

$$\hat{y}_i = F_M(x_i) = \sum_{m=1}^{M} h_m(x_i) \tag{1.7}$$

donde h_m son árboles de regresión de tamaño fijo y son llamados "aprendices débiles". La constante M corresponde al número de aprendices débiles que se consideran para el ensamble.

Al igual que otros algoritmos de boosting, un GBRT se construye de la siguiente manera:

$$F_m(x) = F_{m-1}(x) + h_m(x) \tag{1.8}$$

donde el árbol recién agregado, h_m, se ajusta con el fin de minimizar la suma de pérdidas, L_m, dada por el ensamble anterior, F_{m-1}:

$$h_m = \arg\min_h L_m = \arg\min_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i)) \tag{1.9}$$

donde $l(y_i, F(x_i))$ está definido por la función de pérdida (detalladas más adelante).

El modelo inicial F_0 se elige como la constante que minimiza la función de pérdida, por ejemplo, para una pérdida por mínimos cuadrados, F_0 es la media empírica de los valores objetivo.

Usando una aproximación de Taylor de primer orden[12], el valor de l se puede aproximar de la siguiente manera:

$$l(y_i, F_{m-1}(x_i) + h_m(x_i)) \approx l(y_i, F_{m-1}(x_i)) + h_m(x_i) \left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)} \right]_{F=F_{m-1}} \tag{1.10}$$

donde $\left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)} \right]_{F=F_{m-1}}$ es la derivada de la pérdida con respecto a su segundo parámetro, evaluado en $F_{m-1}(x)$. Lo denotaremos por g_i.

[12]Una aproximación de Taylor de primer orden dice que $l(z) \approx l(a) + (z-a)\frac{\partial l(a)}{\partial a}$. Aquí z corresponde a $F_{m-1}(x_i) + h_m(x_i)$, y a corresponde a $F_{m-1}(x_i)$. (Ver Teorema de Taylor [54].

Eliminando los términos constantes, tenemos:

$$h_m \approx \arg\min_h \sum_{i=1}^{n} h(x_i) g_i \tag{1.11}$$

Esto se minimiza si $h(x_i)$ se ajusta para predecir un valor que es proporcional al gradiente negativo ($-g_i$). Por lo tanto, en cada iteración, el estimador h_m se ajusta para predecir los gradientes negativos de las observaciones. Los gradientes se actualizan en cada iteración. Esto puede considerarse como un descenso de gradiente en un espacio funcional.

Clasificación

El gradient boosting para clasificación es muy similar al caso de regresión. Sin embargo, la suma de los árboles $F_M(x_i) = \sum_m h_m(x_i)$ no es homogéneo para una predicción: no puede ser una clase, ya que los árboles predicen valores continuos.

El mapeo del valor $F_M(x_i)$ a una clase o probabilidad depende de la pérdida. Para la desviación (o pérdida logarítmica), la probabilidad de que x_i pertenezca a la clase positiva se modela como $p(y_i = 1|x_i) = \sigma(F_M(x_i))$, donde σ es la función sigmoidea.

Se debe de tener en cuenta que incluso para una tarea de clasificación, el subestimador h_m sigue siendo un regresor, no un clasificador. Esto se debe a que los subestimadores están entrenados para predecir gradientes (negativos), que siempre son cantidades continuas.

Funciones de pérdida

Las siguientes son algunas funciones de pérdida que son compatibles con el algoritmo de gradient boosting implementado en Python y se pueden especificar mediante el parámetro“loss” [55] [56]:

- Regresión:
  - Error cuadrático medio (MSE), también denominada regularización L2. El modelo inicial F_0 esta dado por la media de los valores objetivo.
  - Error absoluto medio (MAE), también denominada regularización L1. El modelo inicial F_0 esta dado por la mediana de los valores objetivo.
- Clasificación:
  - Binomial deviance: es la función de pérdida del logaritmo de la verosimilitud (log-likelihood, en inglés) de la binomial negativa para clasificación binaria. El modelo inicial viene dado por el logaritmo de los momios de probabilidad (odds-ratio, en inglés).
  - Multinomial deviance: es la función de pérdida del logaritmo de la verosimilitud (log-likelihood, en inglés) de la multinomial negativa para clasificación multiclase, con n clases mutuamente excluyentes. El modelo inicial esta dado por la probabilidad inicial de cada clase.

Tasa de aprendizaje

Se puede generar una estrategia de regularización simple que escale la contribución de cada aprendiz débil en un factor constante ν [57]:

$$F_m(x) = F_{m-1}(x) + \nu h_m(x) \tag{1.12}$$

El parámetro ν también se denomina tasa de aprendizaje porque escala la longitud del paso del algoritmo de descenso de gradiente; se puede configurar a través del parámetro "learning_rate" en Python.

El parámetro"learning_rate" interactúa fuertemente con el parámetro " n_estimators". Valores más pequeños de "learning_rate" requieren un mayor número de estudiantes débiles para mantener un error de entrenamiento constante. La evidencia empírica sugiere que los valores pequeños de "learning_rate" mejoran error del ensamble.

1.4.2.4. XGBoost

El nombre XGBoost se refiere al objetivo de impulsar al límite los recursos computacionales para el cálculo de algoritmos de Gradient Boosting, es decir, un Extreme Gradient Boosting.

Al igual que el modelo de Gradient Boosting, el XGBoost crea nuevos modelos que predicen los residuos (o errores) de modelos anteriores y luego se suman para hacer la predicción final. Se llama aumento de gradiente (gradient boosting) porque utiliza un algoritmo de descenso de gradiente para minimizar la pérdida al agregar nuevos modelos. La diferencia radica en la implementación del algoritmo, ya que fue diseñado para mejorar la eficiencia del tiempo de cómputo y de los recursos de memoria. El objetivo del diseño es aprovechar al máximo los recursos disponibles para entrenar el modelo [58] [59].

Ventajas

1. Son rápidos y precisos, en algunos casos incluso más que los Gradient Boosting.
2. Estructura de bloques para apoyar la paralelización de la construcción de árboles.

Desventajas

1. La principal desventaja es que, a diferencia de los árboles de decisión, los XGBoost son difíciles de interpretar dado que la decisión final del modelo considera las decisiones tomadas (predicciones) por los n árboles contruidos por el modelo.
2. Más difícil de ajustar los hiperparámetros.

1.4.3. Medidas de precisión

Una vez que se tienen construidos los modelos seleccionados para resolver el caso de uso que se esta trabajando, se tiene que evaluar el desempeño de cada uno de estos modelos.

Existen diferentes métricas que analizan la precisión de los modelos desde diferentes ángulos y dependiento del tipo de problema que se esta resolviendo, es decir, si es de clasificación o de regresión.

1.4.3.1. La Matriz de Confusión y sus métricas

Es una herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado de clasificación. Como su nombre lo indica, es una matríz *nxn* en la que las filas representan las clases reales y las columnas representan las clases predichas por el modelo.

Sirve para mostrar de forma explícita cuando una clase es confundida con otra. Por eso, permite identificar los distintos tipos de error en lo que se puede caer al momento de hacer la clasificación (falsos positivos y falsos negativos).

La siguiente es la matriz de confusión cuando n=2, es decir, cuando se tiene un problema de clasificación binaria (1 y 0):

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 1.13: Matriz de Confusión (RPubs, 2017).

En donde:

- **VP**: es la cantidad de casos *positivos* que fueron *clasificados correctamente* como positivos por el modelo.
- **VN**: es la cantidad de casos *negativos* que fueron *clasificados correctamente* como negativos por el modelo.
- **FN**: es la cantidad de casos *positivos* que fueron *clasificados incorrectamente* como negativos por el modelo.
- **FP**: es la cantidad de casos *negativos* que fueron *clasificados incorrectamente* como positivos por el modelo.

A partir de estas cuatro variables se construyen otro conjunto de métricas que permiten evaluar el desempeño de los modelos [60] [61] [62]:

Exactitud

La exactitud, o "accuracy" (en inglés), mide el porcentaje de casos en los que el modelo ha acertado. Se calcula con la siguiente fórmula:

$$Exactitud = \frac{VP + VN}{VP + VN + FN + FP} \tag{1.13}$$

Precisión

La precisión, o "precision" (en inglés), informa la calidad del modelo ya que mide el porcentaje de las predicciones positivas correctas. Se calcula con la siguiente fórmula:

$$Precisión = \frac{VP}{VP + FP} \tag{1.14}$$

Sensibilidad

La sensibilidad, o "sensitivity" (en inglés), también conocida como exhaustividad, o "recall" (en inglés), es la tasa de verdaderos positivos, es decir, mide el porcentaje de casos positivos detectados con el modelo. Se calcula con la siguiente fórmula:

$$Sensibilidad = \frac{VP}{VP + FN} \tag{1.15}$$

Especificidad

La especificidad, o "specificity" (en inglés), es la tasa de verdaderos negativos, es decir, mide el porcentaje de casos negativos detectados por el modelo. Se calcula con la siguiente fórmula:

$$Especificidad = \frac{VN}{FP + VN} \tag{1.16}$$

Tasa de error

La tasa de error, o "misclassification rate" (en inglés), mide el porcentaje de los casos que fueron clasificados incorrectamente. Se calcula con la siguiente fórmula:

$$Tasa\ de\ Error = \frac{FP + FN}{VP + VN + FN + FP} \tag{1.17}$$

Conforme a estas métricas, se puede analizar el performance de un modelo, en particular, utilizando la precisión y la sensibilidad (recall) se tienen cuatro casos posibles:

- Alta *precisión* y alto *recall*: el modelo maneja perfectamente esa clase.
- Alta *precisión* y bajo *recall*: el modelo no detecta la clase muy bien, pero cuando lo hace es altamente confiable.

- Baja *precisión* y alto *recall*: el modelo detecta bien la clase, pero también incluye muestras de la otra clase.
- Baja *precisión* y bajo *recall*: el modelo no logra clasificar la clase correctamente.

Además de estas métricas, intimamente ligadas con la matriz de confusión, existen otras métricas resultantes de la combinación de algunas de las anteriores, y que sirven para comparar el rendimiento de los modelos de una manera más sencilla, por ejemplo, el valor F1 (F1 score) y la curva ROC, explicadas a continuación.

1.4.3.2. F1 score

El valor F1, o "F1 Score" (en inglés), se utiliza para combinar la precisión y la sensibilidad en un sólo valor calculando la media armónica entre la precisión y la sensibilidad [63].

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{1.18}$$

En realidad, es un caso particular de la función F beta cuando $\beta = 1$ [64].

$$F_\beta = (1 + \beta^2) \times \frac{Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{1.19}$$

El valor F_β alcanza su mejor valor en 1 y su peor puntuación en 0 [65]. Es importante que al elegir la β para la función F beta, ésta debe ser más alta cuando se le dé más peso a la sensibilidad sobre la precisión. Por ejemplo, el valor F1 se preocupa por igual por la sensibilidad y la precisión, mientras que con el valor F2, la sensibilidad es dos veces más importante que la precisión.

1.4.3.3. Curva ROC

La curva ROC (acrónimo de Receiver Operating Characteristic, o Característica Operativa del Receptor) es una representación gráfica de la tasa de verdaderos positivos (sensibilidad) frente a la tasa de falsos positivos (1 - especificidad) para todos los umbrales de corte posibles, es decir, cada punto[13] de la curva ROC representa un umbral de corte diferente [66].

Existen dos casos particulares que son importantes de mencionar:

1. La curva ROC de un modelo de clasificación perfecto (Figura 1.14).
2. La curva ROC de un modelo de clasificación aleatorio (Figura 1.15), la cual divide el espacio en dos para poder identificar rapidamente el desempeño de los modelos: las curvas ROC en el área con la esquina superior izquierda indican buenos niveles de desempeño, mientras que las curvas ROC en la otra área con la esquina inferior derecha indican bajos niveles de desempeño.

[13] Los puntos están conectados para formar la curva

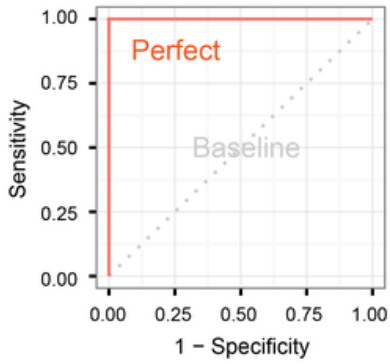


Figura 1.14: Curva ROC para un modelo perfecto (Saito, T. y Rehmsmeier, M., 2015).

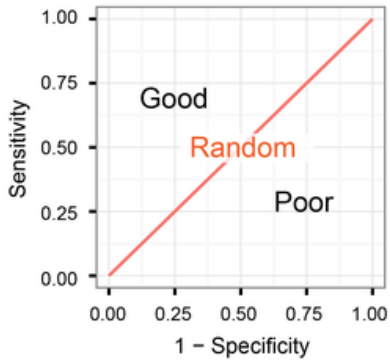


Figura 1.15: Curva ROC para un modelo aleatorio (Saito, T. y Rehmsmeier, M., 2015).

El área debajo de la curva ROC

El área debajo de la curva ROC (ROC AUC) es una medida popular de la precisión de los modelos de clasificación (Figura 1.16). En general, valores altos de AUC indican un mejor rendimiento del modelo. Los posibles valores de AUC para un buen modelo oscilan entre 0.5 (sin capacidad de clasificación) y el 1.0 (capacidad de clasificación perfecta).

Los puntajes AUC son convenientes para comparar múltiples modelos de clasificación. No obstante, también es importante verificar las curvas reales, especialmente al evaluar el modelo final. Incluso cuando dos curvas ROC tienen los mismos valores de AUC, las curvas reales pueden ser bastante diferentes [67].

Para el caso que se analiza en el presente trabajo, se utilizarán las curvas ROC y su correspondiente AUC para identificar los mejores modelos y los mejores puntos de corte, además, se hará uso de la matriz de confusión para identificar cuál es el modelo que maximiza la sensibilidad y declararlo como modelo campeón debido a que se trata de un problema de clasificación binaria donde se le da un mayor peso a la clase positiva, es decir, a aquellos pacientes que tengan los triglicéridos o el colesterol alterados, en otras palabras, se busca minimizar la cantidad de falsos negativos.

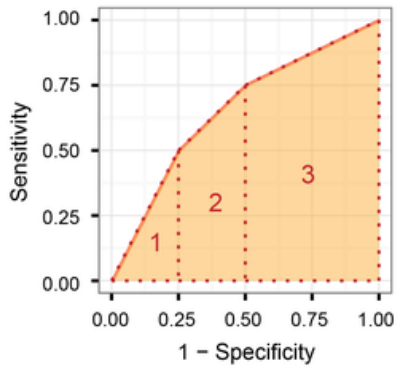


Figura 1.16: Área debajo de la curva ROC (Saito, T., Rehmsmeier, M., 2015).

Por otro lado, la tarea de los modelos de regresión es de predecir valores continuos aprendiendo de otras variables independientes correlacionadas con la variable objetivo. Existen diversas métricas que evaluan los resultados de las predicciones hechas por los modelos, algunas de las más utilizadas son las siguientes.

1.4.3.4. Error Cuadrático Medio

El error cuadrático medio o MSE (Mean Squared Error, en inglés) es el promedio de la diferencia al cuadrado entre el valor objetivo (y) y el valor predicho ($\hat{y}$) por el modelo de regresión. Al elevar al cuadrado las diferencias, se penaliza incluso un pequeño error, lo cual conduce a una sobreestimación de lo malo que es el modelo [68]. Se calcula con la siguiente fórmula:

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2 \tag{1.20}$$

Raíz del Error Cuadrático Medio

La raíz del error cuadrático medio o RMSE (Root Mean Squared Error, en inglés), como su nombre lo indica, es la raiz cuadrada del MSE, y, al igual que el MSE, penaliza los grandes errorres, lo que lo hace muy útil cuando no se desean diferencias muy grandes entre los valores reales y predichos [68] [69]. Se calcula con la siguiente fórmula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2} \tag{1.21}$$

1.4.3.5. Coeficiente de Determinación

El coeficiente de determinación, denominado R^2, representa la proporción de varianza (de y) que ha sido explicada por las variables independientes en el modelo. Proporciona una medida de bondad de ajuste y, por lo tanto, indica qué tan bien el modelo va a predecir nuevas

observaciones [70]. Se calcula con la siguiente fórmula:

$$R^2 = 1 - \frac{\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}{\sum_{j=1}^{n}(y_j - \bar{y}_j)^2} \tag{1.22}$$

En donde:

$$\bar{y} = \frac{1}{n}\sum_{j=1}^{n} y_j \tag{1.23}$$

La R^2 es una métrica que no se ve afectada por la escala de la variable modelada [68], por lo que siempre estará en el intervalo $(-\infty, 1]$. Es importante notar que un modelo que siempre prediga el valor esperado de y, sin importar las variables de entrada, obtendría una R^2 de 0.

Para el caso que se analiza en este trabajo, se tomarán en cuenta el RMSE para identificar qué tan cercanas son las predicciones con respecto a los valores reales y, además, se utilizará la R^2 para medir el desempeño de los modelos. Se considerarán estos dos estadísticos en conjunto para seleccionar los modelos campeones que predigan los valores de triglicéridos y colesterol de los pacientes analizados.

1.5. Lenguajes de programación para Machine Learning

Los modelos de Machine Learning, como su nombre lo indica, lo que hacen es enseñarle a la computadora a adquirir conocimiento relacionado con algún fenómeno. Hasta el momento ya se han mencionado los álgoritmos matemáticos para enseñarle, sin embargo, hace falta un componente muy importante: el lenguaje para comunicarse con la computadora.

Un lenguaje de programación es un lenguaje formal que, mediante una serie de instrucciones, le permite a un programador escribir un conjunto de órdenes, acciones consecutivas, datos y algoritmos para, de esa forma, crear programas que controlen el comportamiento físico y lógico de una máquina. Este lenguaje está conformado de símbolos, palabras cláves, reglas semánticas y sintácticas que permiten el entendimiento entre un programador y una computadora [71].

Para este trabajo se decidió utilizar Python, el cual es el lenguaje que lidera el sector. Más del 60 % de los desarrolladores de machine learning lo utilizan y lo priorizan para llevar a cabo sus tareas. Es un lenguaje fácil de aprender, escalable y, además, es software libre. Python tiene muchos paquetes de visualización y librerías muy útiles como Numpy, Pandas, Matplotlib, Seaborn y Scikit-learn, los cuales permiten que sean muy sencillas las tareas de modelado [72].

- **Numpy**: es una librería de álgebra lineal para Python con poderosas estructuras de datos para el cálculo eficiente de matrices y arreglos multidimensionales.
- **Pandas**: es la librería de Python más popular que proporciona un rendimiento altamente optimizado para el análisis de datos.

- **Matplotlib**: es una librería de visualización para Python, se utiliza para crear gráficos básicos como gráficos de líneas, gráficos de barras, histogramas y muchos más.
- **Seaborn**: es una librería de visualización de datos de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.
- **Scikit-learn**: es una librería de Python para la minería y el análisis de datos. Implementa una amplia gama de algoritmos de aprendizaje automático como algoritmos de clasificación, regresión y agrupación.

Otros lenguajes de programación usados para las tareas de machine learning son R, SAS, Java y Julia (por mencionar algunos).

2 | Análisis exploratorio

Antes de comenzar con el análisis exploratorio de los datos y siguiendo la metodología presentada en la Sección 1.2 se hará el planteamiento inicial del problema y se analizará la información con la que se cuenta para realizar los modelos predictivos.

2.1. Análisis del caso a resolver

De lo mencionado en la Sección 1.1.2, se diagnostica síndrome metabólico a una persona si están presentes tres de los siguientes cinco factores de riesgo:

1. Glucosa en sangre elevada
2. Niveles bajos de colesterol HDL en la sangre
3. Niveles altos de triglicéridos en la sangre
4. Circunferencia de la cintura grande
5. Presión arterial alta

Los factores 1, 4 y 5 pueden ser obtenidos de una manera más sencilla que los factores 2 y 3 pues para estos últimos se necesita realizar una química sanguínea. Con base en esto y en lo planteado en los Objetivos del proyecto se identifica la necesidad de resolver los siguientes cuestionamientos:

* ¿El paciente tiene niveles bajos de colesterol HDL en la sangre?
* ¿Cuál es el nivel de colesterol HDL en la sangre del paciente?
* ¿El paciente tiene niveles altos de triglicéridos en la sangre?
* ¿Cuál es el nivel de triglicéridos en la sangre del paciente?
* ¿El paciente tiene Síndrome Metabólico?

Para poder dar respuesta al primer y tercer cuestionamiento se tienen que desarrollar modelos de aprendizaje supervisado de clasificación (respuesta binaria: Si o No)[1]; para dar respuesta al segundo y tercer cuestionamiento se tienen que desarrollar modelos de aprendizaje supervisado de regresión (estimar un valor); y por último se realizará una evaluación de los cinco factores de riesgo considerados para dar un diagnóstico final.

[1]Ver la construcción de las variables objetivo en la Sección A.1.5.

2.2. Preparación de los datos

Para la realización del presente trabajo se ocupará una tabla de datos proporcionada por el Dr. Miguel Murguía (UBIMED - FES Iztacala), en las que se recaban las medidas invasivas, provenientes de un estudio de química sanguínea, y las no invasivas o antropométricas, obtenidas de una báscula digital, de 1,765 alumnos universitarios de nuevo ingreso.

Para cada uno de los 1,765 alumnos universitarios considerados en este estudio se tienen los valores de las medidas descritas en la Tabla 2.1[2].

Tabla 2.1: Medidas de química sanguínea y antropométicas.

Variable	Descripción	Unidad de medición
ID_EVENTO	Identificador del paciente	Código de 6 dígitos
SEXO	Sexo	M=Masculino , F=Femenino
EDAD	Edad	Años cumplidos
SMet	Síndrome Metabólico	0=F , 1=T
PAD	Presión arterial diastólica	$mmHg$
PAS	Presión arterial sistólica	$mmHg$
GLU	Glucosa	mg/dL
HDL	Colesterol de alta densidad	mg/dL
TRI	Triglicéridos	mg/dL
CC	Circunferencia de Cintura	cm
PESO	Peso	kg
TALLA	Talla	cm
IMC	Índicel de masa corporal	kg/m^2
C_TOTAL	Colesterol	mg/dL
HOMA	Índice de resistencia a la insulina	$\mu UI/mL$
INS	Insulina	$\mu UI/mL$
CC_AGUA	Composición corporal: agua	kg
CC_GRAS	Composición corporal: grasa	kg
CC_HUES	Composición corporal: hueso	kg
CC_MUSC	Composición corporal: músculo	kg
P_AGUA	Composición corporal: % agua	%
P_GRAS	Composición corporal: % grasa	%
P_HUES	Composición corporal: % hueso	%
P_MUSC	Composición corporal: % músculo	%
VFA	Área de Grasa Visceral	cm^2

Se identifica que, aunque se cuenta con la variable SMet que indica si el paciente tiene Síndrome Metabólico o no, no se tienen las variables objetivo necesarias para los modelos de aprendizaje supervisado de clasificación, por lo que tendrán que ser construidas. Para los

[2]Unidades de medida: milímetros de mercurio ($mmHg$), miligramos por decilitro (mg/dL) y microunidades internacionales por mililitro ($\mu UI/mL$).

modelos de aprendizaje supervisados de regresión se considerarán las variables GLU y TRI como las variables objetivo[3].

Es importante mencionar que para cumplir con los Objetivos planteados, únicamente se tomarán en cuenta las medidas antropométricas al momento de generar los modelos predictivos, es decir, no se tomarán como variables predictoras a todas aquellas variables que se obtengan directamente de una química sanguínea.

2.3. Análisis exploratorio de los datos

El análisis exploratorio de los datos es un paso crucial antes de pasar a la generación de modelos de Machine Learning, proporciona el contexto necesario para desarrollar un modelo apropiado e interpretar los resultados correctamente.

2.3.1. Librerías utilizadas

```
In [1]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt

        # Para poder visualizar todas las columnas:
        pd.set_option('display.max_columns',100)

        # Para ver unicamente 3 decimales:
        pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

2.3.2. Acceso a los datos

Se importan los datos que resultaron del proceso de preparación de datos mostrado en la sección anterior:

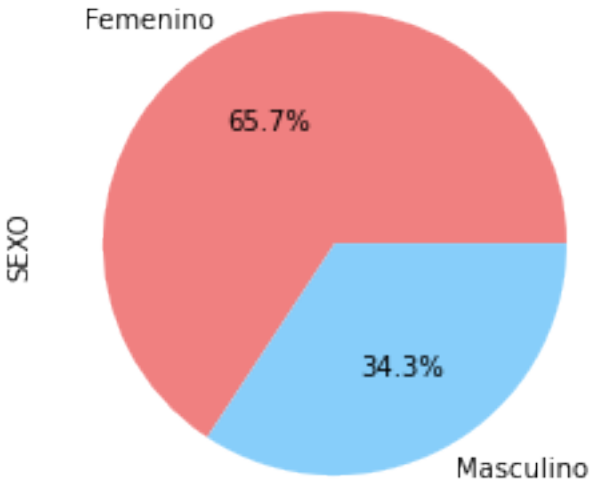
```
In [2]: data = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_prep.csv')
```

2.3.3. Población analizada

Se comienza analizando el perfil de los alumnos de nuevo ingreso de la FES Iztacala:

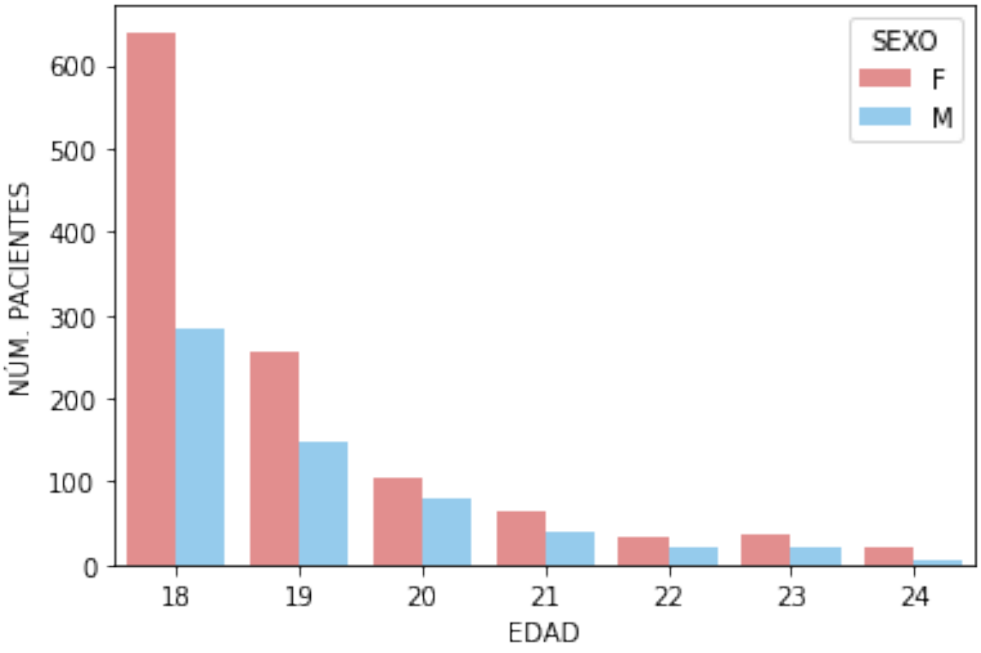
```
In [3]: data['SEXO'].value_counts().plot(kind='pie', autopct='%1.1f%%',
                                         labels=['Femenino','Masculino'],
                                         colors=['lightcoral','lightskyblue']);
```

[3]Para ver el proceso de preparación de los datos referirse al Apéndice A.1.



La variable “SEXO” indica el género (Máculino, Femenino) de los alumnos y con base en la gráfica anterior se concluye que la mayoría de los alumnos de nuevo ingreso en la FES Iztacala son mujeres. Dado que no se tiene la cantidad suficiente de datos para tener una muestra balanceada para la variable “SEXO” (50 y 50 de ambos géneros), se decide quitarla del análisis para evitar un sesgo al momento de entrenar los modelos.

```
In [4]: data["CONTEO"] = 1;
        sns.barplot(x="EDAD", y="CONTEO", hue="SEXO", data=data, estimator=sum,
                    palette=['lightcoral','lightskyblue']).set_ylabel("NÚM. PACIENTES");
```



Del gráfico anterior se deduce que la mayoría de los alumnos de nuevo ingreso a la FES Iztacala tienen entre 18 y 19 años de edad, la distribución de esta variable hace sentido con

lo que se conoce empíricamente de los alumnos de nuevo ingreso a la universidad.

Debido a la cantidad de información con la que se cuenta no sería viable tener una muestra balanceada de observaciones por cada grupo de edad por lo que tampoco será considerada como una variable predictiva.

2.3.4. Análisis de las variables numéricas

Ahora se hace el cálculo de los principales estadísticos de las variables que contienen las medidas antropométricas de los pacientes y se hace un análisis de la distribución de las mismas:

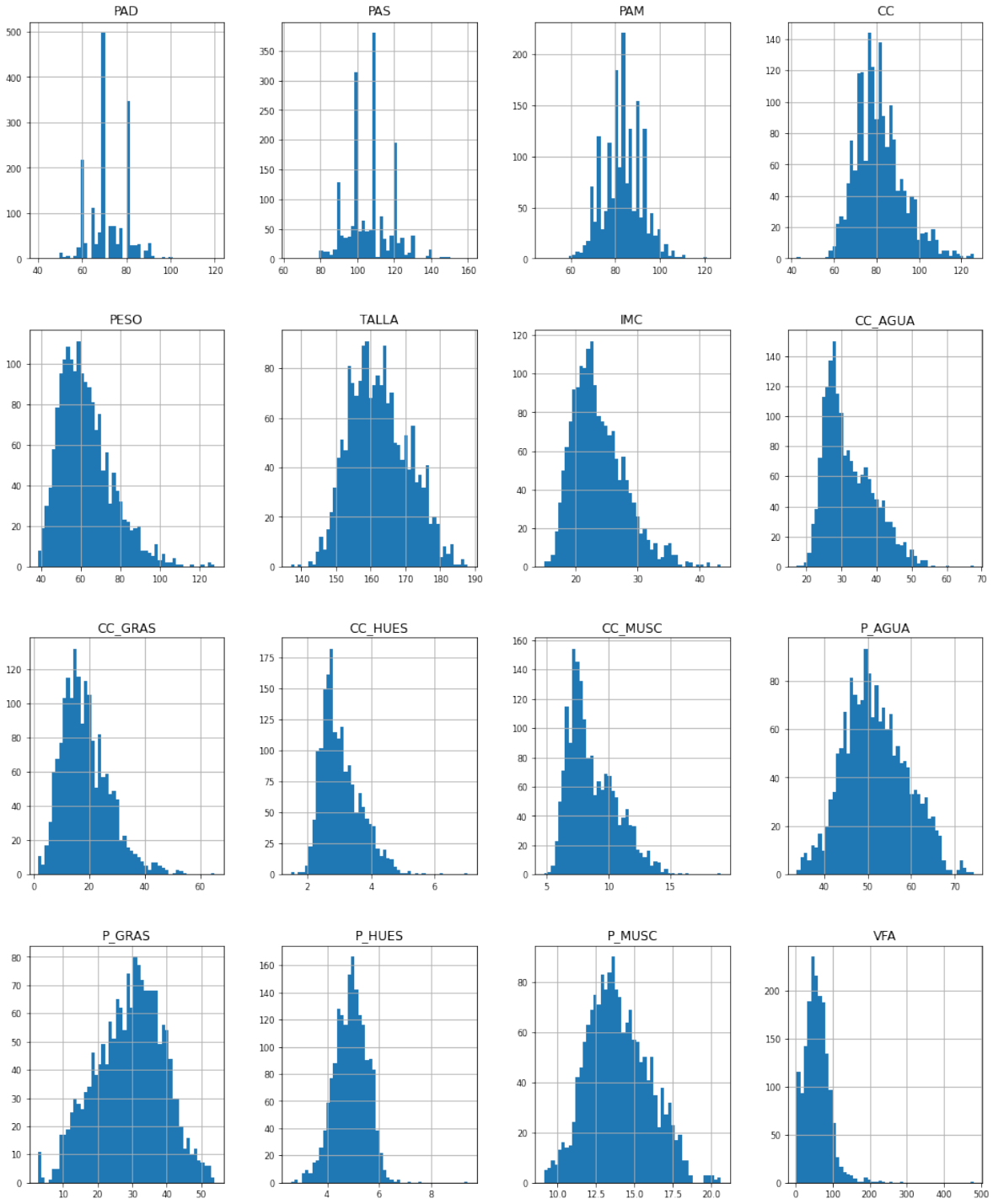
```
In [5]: var_explanatory= [x for x in data.columns if x not in ['ID_EVENTO','SMet','HDL','TRI',
                                                               'TARGET_TRI','TARGET_HDL',
                                                               'CONTEO','EDAD','SEXO']]
        data[var_explanatory].describe()
```

Out[5]:

	PAD	PAS	PAM	CC	PESO	TALLA	IMC	CC_AGUA
count	1757.000	1757.000	1757.000	1757.000	1757.000	1757.000	1757.000	1757.000
mean	71.783	106.933	83.500	81.360	62.719	162.372	23.707	32.214
std	8.727	11.743	8.854	11.194	13.342	8.262	4.237	7.029
min	40.000	64.000	48.000	42.500	38.370	137.000	14.889	17.300
25%	66.000	100.000	77.333	73.500	52.930	156.000	20.624	26.900
50%	70.000	110.000	83.333	80.000	60.400	162.000	22.917	30.400
75%	80.000	112.000	90.000	87.500	70.040	168.000	26.203	36.900
max	120.000	160.000	127.333	126.000	127.900	188.000	43.415	67.800

	CC_GRAS	CC_HUES	CC_MUSC	P_AGUA	P_GRAS	P_HUES	P_MUSC	VFA
count	1757.000	1757.000	1757.000	1757.000	1757.000	1757.000	1757.000	1757.000
mean	18.741	3.049	8.715	51.799	29.288	4.911	14.002	60.008
std	8.680	0.626	1.954	7.219	9.766	0.640	1.990	34.406
min	1.500	1.470	4.800	33.779	2.703	2.603	9.167	5.000
25%	12.600	2.590	7.200	46.562	22.492	4.488	12.564	38.200
50%	17.400	2.910	8.200	51.236	30.035	4.936	13.809	56.800
75%	23.800	3.420	10.000	56.708	36.353	5.358	15.345	78.300
max	65.300	7.060	19.100	74.403	53.931	9.431	20.650	479.600

```
In [6]: data[var_explanatory].hist(figsize=(16, 20), bins=50, xlabelsize=8,
                                   ylabelsize=8);
```



Las distribuciones anteriores son consistentes con los resultados obtenidos en la ENSANUT 2018, en donde se muestra el alto porcentaje de jovenes y adultos con problemas de obesidad y sobrepeso.

2.3.5. Selección de variables

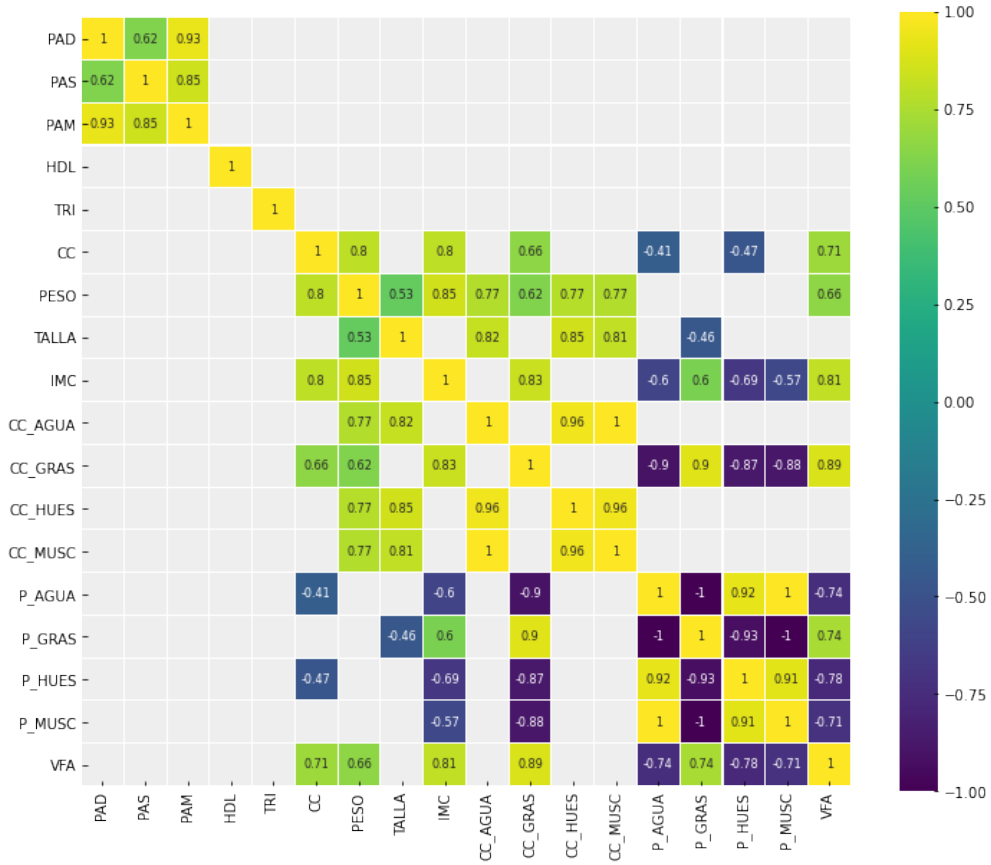
Un paso importante antes de comenzar con la construcción de los modelos predictivos es la selección de variables, es decir, la selección de aquellas medidas que ayudarán a explicar el fenómeno a ser modelado. A pesar de que los árboles de decisión no hacen suposiciones sobre las variables de entrada, estos utilizan medidas de impureza (como Gini o Entropía) para identificar los cortes óptimos para mejorar la clasificación, es importante realizar una selección de variables a través del coeficiente de correlación entre las variables para ayudar al modelo proveyéndole de variables que no tengan problemas de multicolinealidad.

2.3.5.1. Coeficiente de correlación

El Coeficiente de Correlación es utilizado en estadística para medir la fuerza y la dirección de la asociación entre dos variables continuas. Hay dos coeficientes de correlación que se usan frecuentemente: el de Pearson (paramétrico) y el de Spearman (no paramétrico, se utiliza en aquellos casos donde las variables examinadas no cumplen criterios de normalidad o cuando las variables son ordinales).

A continuación, se presenta un gráfico que ayuda a identificar la correlación (utilizando Spearman) entre las variables numéricas consideradas:

```
In [7]: with plt.style.context('bmh'):

            corr = data.drop(columns=['ID_EVENTO','SMet','CONTEO',
                                      'TARGET_TRI','TARGET_HDL',
                                      'EDAD'], axis=1).corr(method='spearman')
            plt.figure(figsize=(12, 10))

            sns.heatmap(corr[(corr >= 0.5) | (corr <= -0.4)],
                        cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
                        annot=True, annot_kws={"size": 8},
                        square=True);

        plt.show()
```



De los coeficientes de correlacion identificados en la gráfica anterior y usando un poco de conocimiento empírico relacionado con el Síndrome Metabólico y, en particular, con los triglicéridos y el colesterol HDL, de deciden utilizar únicamente las siguientes variables:

- **Presión arterial media (PAM)**: con esta variable se puede resumir la información de la presión arterial diastólica y sistólica.
- **Circunferencia de cintura (CC)**, **índice de masa corporal (IMC)**, **composición corporal: grasa (CC_GRAS)** y **área de grasa viseral (VFA)**: como se explicó anteriormente, en la mayoría de los casos la expresión del síndrome metabólico ocurre en individuos obesos y, además, en muchos casos, la expresión del síndrome metabólico es en buena medida una comorbilidad de la obesidad [19]. Es por ello que se decide incluir este conjunto de variables que aportan más información relacionada al nivel de obesidad de los pacientes pues parece ser uno de los factores desencadenantes más importantes del síndrome.
- **Talla (TALLA)**, **composición corporal: agua (CC_AGUA)** y **composición corporal: músculo (CC_MUSC)**: se decide incluir estas variables ya que, a pesar de tener correlaciones positivas con algunas de las variables anteriores, estas ayudan a explicar y dimensionar los valores que se tengan. Por ejemplo, se tiene una correlación positiva alta entre talla y peso (a mayor altura, mayor peso), sin embargo, no es lo mismo medir 180 cm y pesar 80 kg a tener el mismo peso pero midiendo 140 cm, de igual forma ayuda saber cuál es la composición de ese peso, es decir, saber si el peso que se tiene esta dado por la cantidad de músculo o la cantidad de grasa que tiene el paciente.

3 | Construcción de los Modelos

En este capítulo se mostrará la construcción de los modelos de aprendizaje supervisado basados en árboles, los cuales resuelven lo planteado en la Sección 2.1.

3.1. Modelos predictivos para detectar niveles bajos de colesterol HDL

3.1.1. Modelos supervisados para clasificación

En esta sección se llevará a cabo la construcción de los modelos supervisados de Machine Learning que calculen la probabilidad de que el paciente tenga su nivel de colesterol HDL por debajo de los 40 mg/dL (evento) para posteriormente hacer la clasificación de Alterado (1) y No Alterado (0), si esta por debajo o por arriba del umbral, respectivamente.

3.1.1.1. Librerías utilizadas

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib
        import matplotlib.pyplot as plt

        from collections import Counter
        from imblearn.combine import SMOTETomek
        from sklearn.model_selection import train_test_split

        from sklearn.model_selection import RandomizedSearchCV,GridSearchCV

        from sklearn import tree
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import GradientBoostingClassifier
        from xgboost import XGBClassifier

        from sklearn.metrics import plot_confusion_matrix, confusion_matrix
        from sklearn.metrics import precision_score, recall_score
        from sklearn.metrics import fbeta_score, make_scorer
        from sklearn.metrics import roc_auc_score, roc_curve

        #conda install python-graphviz
        import graphviz
```

3.1.1.2. Datos utilizados

Se importan los datos que resultaron del proceso de preparación de datos.

```
In [2]: data = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_prep.csv')
```

Se seleccionan las variables que serán utilizadas como variables dependientes.

```
In [3]: var_predictivas = ['CC','VFA','CC_GRAS','CC_MUSC','CC_AGUA','PAM','TALLA','IMC']
        X = data[var_predictivas]
        y = data['TARGET_HDL']
```

3.1.1.3. Separación de datos

En el modelado predictivo, la estrategia estándar para una evaluación honesta del rendimiento de los modelos se da a través de la separación de los datos en dos subconjuntos. Una parte de la fuente de datos se utiliza para ajustar los modelos: el conjunto de datos de entrenamiento. El resto de la fuente de datos se utiliza para dar una estimación honesta final de la generalización de los modelos: el conjunto de datos de validación. La selección de qué datos van en cada uno de los subconjuntos antes mencionados se hace de manera aleatoria.

Para el presente trabajo el conjunto de entrenamiento representa el 70 % de los datos originales y el conjunto de validación se compone del 30 % restante.

```
In [4]: print('Original dataset shape %s' % Counter(y))

Original dataset shape Counter({0: 1449, 1: 308})
```

```
In [5]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,
        random_state=13719)
        print('Train dataset shape %s' % Counter(y_train))
        print('Test dataset shape %s' % Counter(y_test))

Train dataset shape Counter({0: 1019, 1: 210})
Test dataset shape Counter({0: 430, 1: 98})
```

Dado que la muestra de datos para el entrenamiento no está balanceada (17.09 % de Eventos vs 82.91 % de No-Eventos)[1], se utilizará una técnica que consiste en aplicar simultáneamente un algoritmo de subsampling[2] y otro de oversampling[3] al conjunto de datos. Se utilizará SMOTE (Synthetic Minority Oversampling Technique) para el oversampling, el cual busca puntos vecinos cercanos y agrega puntos en línea recta entre ellos. Y se utilizará Tomek para el undersampling, el cual quita elementos de la clase mayoritaria que sean "nearest neighbor" para delimitar mejor la zona limítrofe de las clases [77] [45]. El objetivo de aplicar estos métodos a la muestra de entrenamiento es ayudar a los modelos para que puedan aprender a diferenciar las clases de mejor manera y que tengan una buena generalización.

```
In [6]: smotetomek = SMOTETomek(sampling_strategy=0.5,random_state=13719)
        X_train, y_train = smotetomek.fit_resample(X_train, y_train)
        print('Resampled train dataset shape %s' % Counter(y_train))

Resampled train dataset shape Counter({0: 992, 1: 482})
```

[1] El conjunto de datos se compone de 17.53 % de Eventos y 82.47 % de No-Eventos y en la muestra de validación se tiene 18.56 % de Eventos y 81.44 % de No-Eventos.

[2] Subsampling: eliminar observaciones de manera aleatoria perteneciente a la clase mayoritaria [76].

[3] Oversampling: duplicar observaciones pertenecientes a la clase minoritaria [76].

3.1.1.4. Árbol de decisión

El primer modelo que se contruye es un Árbol de Decisión, de la librería Scikit-learn [44], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el ROC AUC como métrica para la medición del desempeño de los modelos.

```
In [7]: # Parámetros para el modelo:
        param_dict_tr= dict(criterion=['gini','entropy'], splitter=['best','random'],
                            max_depth=range(2,30), max_features=range(2,9),
                            min_samples_leaf=[0.05], class_weight=['balanced'])

        # Se crea el árbol de decisión para clasificación:
        tr_clf = DecisionTreeClassifier()

        # Autotuning del modelo:
        rsgrid_tr = RandomizedSearchCV(tr_clf, param_dict_tr, cv=20, n_iter=40, n_jobs=-1,
                                       scoring='roc_auc')
        rsgrid_tr.fit(X_train,y_train)
        tr_clf = rsgrid_tr.best_estimator_

        # Se entrena el árbol de decisión:
        tr_clf = tr_clf.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Árbol de Decisión con los cuales se maximiza el área debajo de la curva ROC.

```
In [8]: rsgrid_tr.best_estimator_

Out[8]: DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                               max_depth=22, max_features=8, min_samples_leaf=0.05)
```

3.1.1.5. Random Forest

El segundo modelo que se contruye es un Random Forest, de la librería Scikit-learn [73], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el ROC AUC como métrica para la medición del desempeño de los modelos.

```
In [9]: # Parámetros para el modelo:
        param_dict_rf= dict(n_estimators=range(50,150), criterion=['gini','entropy'],
                            max_depth=range(20,30), max_features=range(2,9),
                            min_samples_leaf=[0.05], class_weight=['balanced'])

        # Se crea el Random Forest para clasificación:
        rf_clf = RandomForestClassifier()

        #Autotuning del modelo:
        rsgrid_rf = RandomizedSearchCV(rf_clf, param_dict_rf, cv=20, n_iter=40, n_jobs=-1,
                                       scoring= 'roc_auc')
        rsgrid_rf.fit(X_train,y_train)
        rf_clf = rsgrid_rf.best_estimator_

        # Se entrena el Random Forest:
        rf_clf = rf_clf.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Random Forest con los cuales se maximiza el área debajo de la curva ROC.

```
In [10]: rsgrid_rf.best_estimator_

Out[10]: RandomForestClassifier(class_weight='balanced', criterion='entropy',  max_depth=29,
                                max_features=8, min_samples_leaf=0.05,  n_estimators=68)
```

3.1.1.6. Gradient Boosting

El tercer modelo que se contruye es un Gradient Boosting, de la librería Scikit-learn [74], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el ROC AUC como métrica para la medición del desempeño de los modelos.

```
In [11]: # Parámetros para el modelo:
         param_dict_gb= dict(criterion=['friedman_mse', 'mse', 'mae'],
                             loss=['deviance','exponential'],
                             learning_rate=[0.05,0.10,0.15,0.20,0.25,0.30,0.5,0.8,1],
                             max_depth=range(20,30), max_features=range(2,9),
                             min_samples_leaf=[0.05], n_estimators=range(50,150))

         # Se crea el Gradient Boosting para clasificación:
         gb_clf = GradientBoostingClassifier()

         #Autotuning del modelo:
         rsgrid_gb = RandomizedSearchCV(gb_clf, param_dict_gb, cv=20, n_iter=40, n_jobs=-1,
                                        scoring= 'roc_auc')
         rsgrid_gb.fit(X_train,y_train)
         gb_clf = rsgrid_gb.best_estimator_

         # Se entrena el Gradient Boosting:
         gb_clf = gb_clf.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Gradient Boosting con los cuales se maximiza el área debajo de la curva ROC.

```
In [12]: rsgrid_gb.best_estimator_


Out[12]: GradientBoostingClassifier(criterion='mse', learning_rate=0.5, max_depth=25,
                                    max_features=8, min_samples_leaf=0.05,  n_estimators=82)
```

3.1.1.7. XGBoost

El último modelo que se contruye es un XGBoost, de la librería XGBoost [75], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el ROC AUC como métrica para la medición del desempeño de los modelos.

```
In [13]: # Parámetros para el modelo:
         param_dict_xgb= dict(booster=['gbtree', 'dart'],
                              eta=[0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
                              max_depth=range(20,30),n_estimators= range(50,150),
                              scale_pos_weight=[1,2,2.5])

         # Se crea el XGBoost para clasificación:
         xgb_clf = XGBClassifier()

         #Autotuning del modelo:
         rsgrid_xgb = RandomizedSearchCV(xgb_clf, param_dict_xgb, cv=20, n_iter=40, n_jobs=-1,
                                         scoring= 'roc_auc')
         rsgrid_xgb.fit(X_train,y_train)
         xgb_clf = rsgrid_xgb.best_estimator_

         # Se entrena el XGBoost:
         xgb_clf = xgb_clf.fit(X_train,y_train)
```

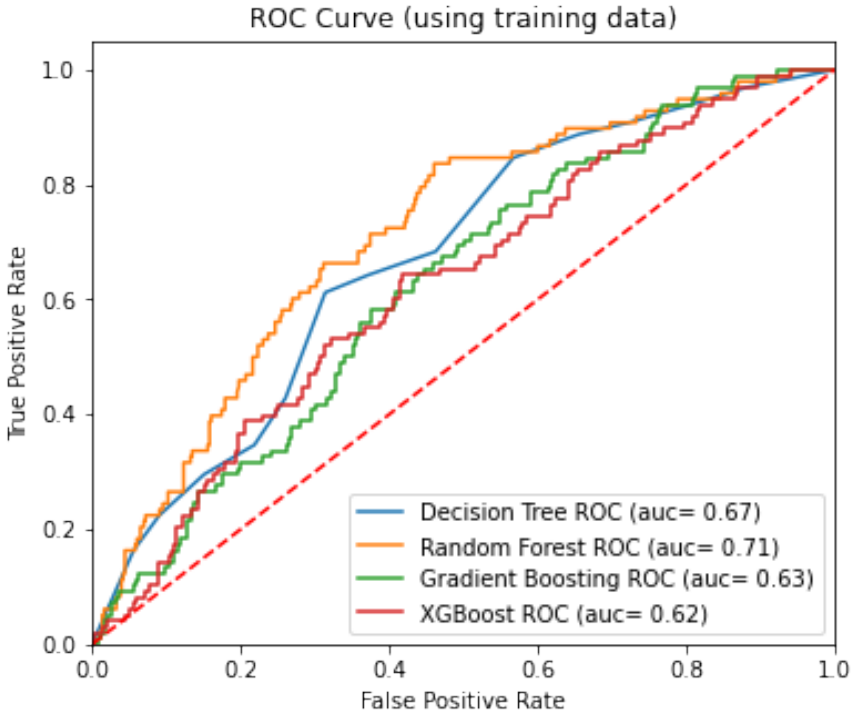
A continuación, se muestran los parámetros del XGBoost con los cuales se maximiza el área debajo de la curva ROC.

```
In [14]: rsgrid_xgb.best_estimator_

Out[14]: XGBClassifier(base_score=0.5, booster='dart', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eta=0.25, gamma=0,
              gpu_id=-1, importance_type='gain', interaction_constraints='',
              learning_rate=0.25, max_delta_step=0, max_depth=24,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=134, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=2, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

3.1.1.8. Estadísticos de precisión

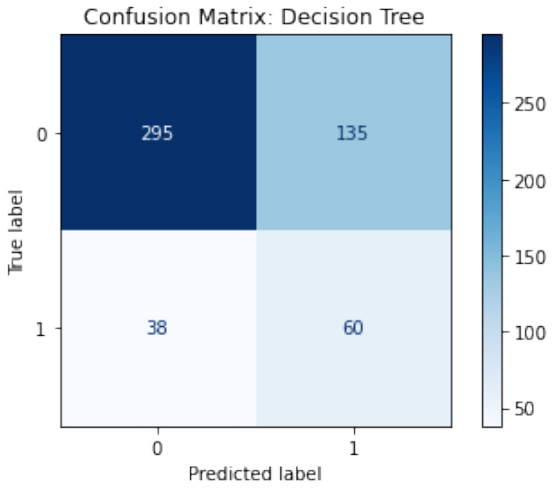
Para medir el desempeño de los modelos se utilizará el conjunto de datos de validación y se analizarán, en primer lugar, las curvas ROC, y del área debajo de las mismas, de cada uno de los modelos.

```
In [15]: plt.figure(figsize=(6,5))
         for m in modelos:
             modelo = m['modelo']
             fpr, tpr, thresholds = roc_curve(y_test, modelo.predict_proba(X_test)[:,1])
             auc = roc_auc_score(y_true=y_test, y_score=modelo.predict_proba(X_test)[:,1])
             plt.plot(fpr,tpr,label='%s ROC (auc= %0.2f)' % (m['label'], auc))
         plt.plot([0,1],[0,1],'r--')
         plt.xlim([0.0,1.0])
         plt.ylim([0.0,1.05])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('ROC Curve (using training data)')
         plt.legend(loc="lower right")
         plt.show()
```

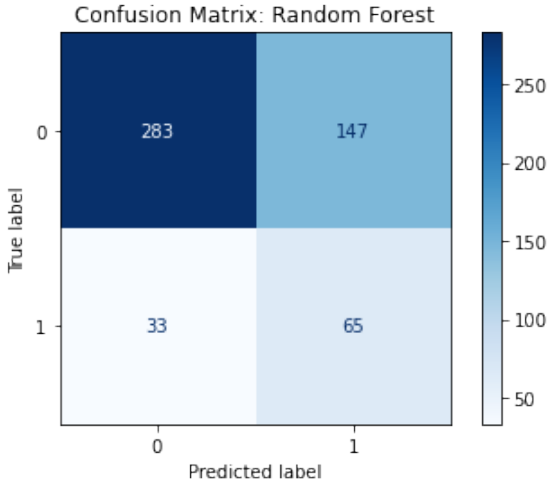


En segundo lugar, se analizará la matriz de confusión de cada uno de los modelos junto con las métrícas de Precisión, Sensibilidad y el valor F2.

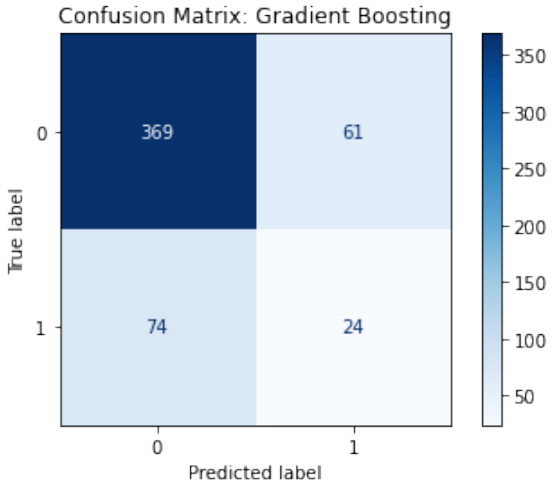
```
In [16]: modelos = [{'label':'Decision Tree','modelo':tr_clf},
                    {'label':'Random Forest','modelo':rf_clf},
                    {'label':'Gradient Boosting','modelo':gb_clf},
                    {'label':'XGBoost','modelo':xgb_clf}]

         for m in modelos:
             modelo = m['modelo']
             disp = plot_confusion_matrix(modelo, X_test, y_test, cmap=plt.cm.Blues)
             disp.ax_.set_title('Confusion Matrix: %s' % (m['label']))
             plt.show()

             precision = precision_score(y_pred=modelo.predict(X_test),y_true=y_test)
             recall = recall_score(y_pred=modelo.predict(X_test),y_true=y_test)
             f2 = fbeta_score(y_pred=modelo.predict(X_test),y_true=y_test, beta=2, average='weighted')

             print('Precisión = %0.2f' % (precision))
             print('Sensibilidad = %0.2f' % (recall))
             print('F2 score = %0.2f' % (f2))
```



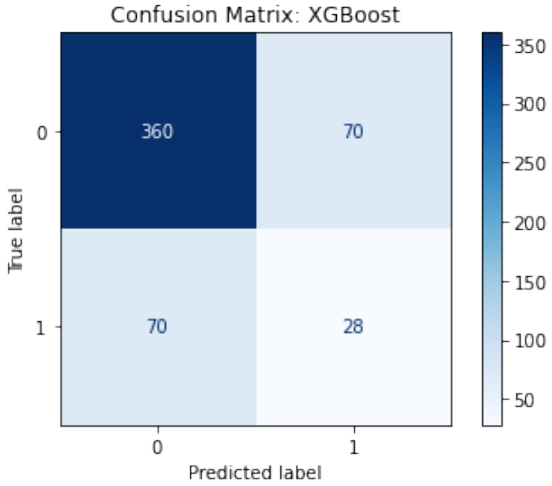

```
Precisión = 0.31
Sensibilidad = 0.61
F2 score = 0.68
```



```
Precisión = 0.31
Sensibilidad = 0.66
F2 score = 0.67
```




```
Precisión = 0.28
Sensibilidad = 0.24
F2 score = 0.74
```




```
Precisión = 0.29
Sensibilidad = 0.29
F2 score = 0.73
```

Al explorar las curvas ROC y el AUC de los modelos se puede observar claramente que el modelo que tiene un mejor desempeño es el Random Forest y, al analizar las matrices de confusión, se confirma como modelo campeón porque es el modelo que identifica la mayor cantidad del verdaderos positivos y, por ende, minimiza los falsos negativos.

Se busca minimizar la máximo la cantidad de falsos negativos ya que, al traducirlo a la práctica médica, lo que estaría haciendo el modelo es decirle al paciente que sus niveles de colesterol no están alterados, cuando si lo están, por lo que no recibiría ningún tipo de tratamiento para mitigar este padecimiento; por otro lado, los falsos negativos no afectan a los

pacientes, ya que, en este caso, se le diría al paciente que tiene que hacer dieta y ejercicio para corregir estos valores, lo cual es algo benéfico para cualquier persona (con y sin Síndrome Metabólico), y se le haría (ahora sí) una química sanguínea para que un médico determine los siguientes pasos.

```
In [17]: champion_model = rf_clf
```

3.1.1.9. Umbral de probabilidad óptimo

Una vez seleccionado el modelo campeón, lo siguiente es identificar cuál es el umbral (punto de corte) de probabilidad con el cual se mejora la clasificación hecha por el modelo, es decir, identificar cuál es el umbral con el cual, si la probabilidad esta por arriba de ese valor entonces la observación se clasifica como evento y si no, entonces se clasifica como no evento. Para llevar a cabo esta tarea, se calculan las métricas de precisión (precisión, sensibilidad, F-Score y AUC) usando diferentes puntos de corte. A continuación, se muestra una tabla con el top 10 de los valores ordenados por AUC, dentro de los cuales se utilizará juicio experto para seleccionar el umbral de probabilidad óptimo.

```
In [18]: fpr, tpr, thresholds = roc_curve(y_test, champion_model.predict_proba(X_test)[:,1])
         predicted_proba = champion_model.predict_proba(X_test)[:,1]

In [19]: precision_ls = []
         recall_ls = []
         fscore_ls = []
         auc_ls = []

         for thres in thresholds:
             y_pred = np.where(predicted_proba>thres,1,0)
             precision_ls.append(precision_score(y_true=y_test, y_pred=y_pred, zero_division=0))
             recall_ls.append(recall_score(y_true=y_test, y_pred=y_pred))
             fscore_ls.append(fbeta_score(y_pred=y_pred, y_true=y_test, beta=2, average='weighted'))
             auc_ls.append(roc_auc_score(y_true=y_test, y_score=y_pred))

         thresholding = pd.concat([pd.Series(thresholds), pd.Series(precision_ls),
                                   pd.Series(recall_ls),pd.Series(fscore_ls), pd.Series(auc_ls)], axis=1)

         thresholding.columns = ['thresholds', 'Precisión','Sensibilidad','Fscore','AUC']
         thresholding.sort_values(by='AUC', ascending=False, inplace=True)
         thresholding.head(n=10)
```

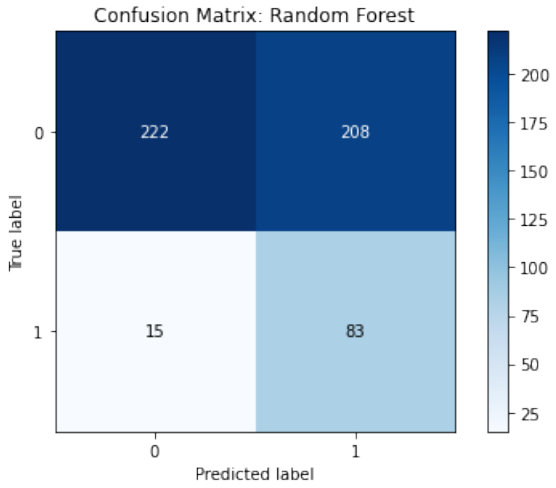
Out[19]:

	Thresholds	Precisión	Sensibilidad	F-score	AUC
111	0.442818	0.290323	0.826531	0.591813	0.683033
112	0.436317	0.287719	0.836735	0.583285	0.682321
116	0.430056	0.285223	0.846939	0.574697	0.681609
113	0.435283	0.286713	0.836735	0.581284	0.681158
117	0.427909	0.284247	0.846939	0.572686	0.680446
110	0.443284	0.288809	0.816327	0.592339	0.679093
114	0.433126	0.284722	0.836735	0.577278	0.678832
115	0.432086	0.283737	0.836735	0.575272	0.677670
108	0.445445	0.289377	0.806122	0.596822	0.677480
109	0.445046	0.288321	0.806122	0.594840	0.676317

Cabe recordar que la intención es minimizar la cantidad de falsos negativos, en otras palabras, se busca maximizar la Sensibilidad del modelo. De acuerdo con la tabla anterior, el umbral número 116 es el punto de corte que ayuda a tener una alta sensibilidad sin sacrificar demasiado la precisión y manteniendo al modelo con un buen desempeño (buen ROC AUC).

```
In [20]: umbral = thresholds[116]
```

```
In [22]: y_pred_final = np.where(predicted_proba>umbral,1,0)

         precision = precision_score(y_pred=y_pred_final, y_true=y_test)
         recall = recall_score(y_pred=y_pred_final, y_true=y_test)
         f2 = fbeta_score(y_pred=y_pred_final, y_true=y_test, beta=2, average='weighted')
         auc = roc_auc_score(y_true=y_test, y_score=y_pred_final)
         matriz = confusion_matrix(y_true=y_test, y_pred=y_pred_final)

         true_label = ['0','1']
         predicted_label = ['0','1']

         fig, ax = plt.subplots()

         im, cbar = heatmap(matriz, true_label, true_label, ax=ax,
                            cmap=plt.cm.Blues, cbarlabel="")
         texts = annotate_heatmap(im, valfmt="{x:.0f}")

         fig.tight_layout()
         plt.xlabel('Predicted label')
         plt.ylabel('True label')
         plt.title('Confusion Matrix: Random Forest')
         plt.show()

         print('Precisión = %0.2f' % (precision))
         print('Sensibilidad = %0.2f' % (recall))
         print('F2 score = %0.2f' % (f2))
         print('ROC AUC = %0.2f' % (auc))
```




```
Precisión = 0.29
Sensibilidad = 0.85
F2 score = 0.57
ROC AUC = 0.68
```

3.1.1.10. Datos para ROI

Para hacer el ejercicio de ROI se utilizará un subconjunto de los datos originales (el 10 %, elegido aleatoriamente), los cuales serán calificados por cada uno de los modelos campeones, en este caso, el modelo campeón para detectar si el colesterol HDL del paciente está alterado o no. La finalidad es, dentro de las conclusiones, determinar la relación costo-beneficio de usar este tipo de modelos para diagnosticar la prevalencia del Síndrome Metabólico en jóvenes mexicanos.

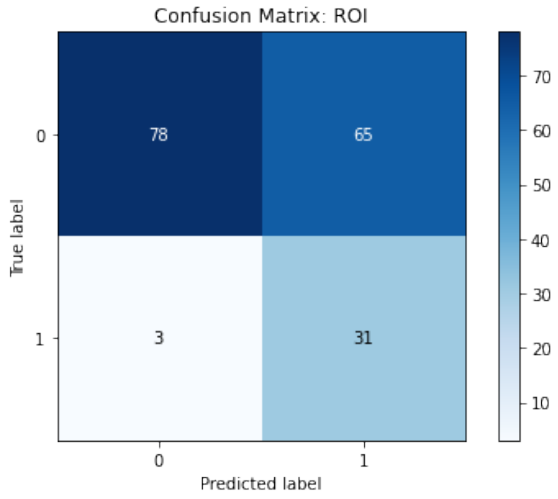
```
In [23]: datos_roi = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_muestra.csv')
         X_roi = datos_roi[var_predictivas]
         y_roi = datos_roi['TARGET_HDL']
```

Se calculan las probabilidades y, haciendo uso del umbral óptimo, se hace la clasificación binaria:

```
In [24]: predicted_proba = champion_model.predict_proba(X_roi)[:,1]
         predicted_class = np.where(predicted_proba>umbral,1,0)
```

Se calcula la matriz de confusión junto con los estadísticos de precisión:

```
In [25]: precision = precision_score(y_pred=predicted_class, y_true=y_roi)
         recall = recall_score(y_pred=predicted_class, y_true=y_roi)
         f2 = fbeta_score(y_pred=predicted_class, y_true=y_roi, beta=2, average='weighted')
         auc = roc_auc_score(y_true=y_roi, y_score=predicted_class)
         matriz = confusion_matrix(y_true=y_roi, y_pred=predicted_class)

         true_label = ['0','1']
         predicted_label = ['0','1']

         fig, ax = plt.subplots()

         im, cbar = heatmap(matriz, true_label, true_label, ax=ax, cmap=plt.cm.Blues, cbarlabel="")
         texts = annotate_heatmap(im, valfmt="{x:.0f}")

         fig.tight_layout()
         plt.xlabel('Predicted label')
         plt.ylabel('True label')
         plt.title('Confusion Matrix: ROI')
         plt.show()

         print('Precisión = %0.2f' % (precision))
         print('Sensibilidad = %0.2f' % (recall))
         print('F2 score = %0.2f' % (f2))
         print('ROC AUC = %0.2f' % (auc))
```




```
Precisión = 0.32
Sensibilidad = 0.91
F2 score = 0.61
ROC AUC = 0.73
```

Se guarda la clasificación hecha por el modelo para que sirva como insumo para el modelo de regresión:

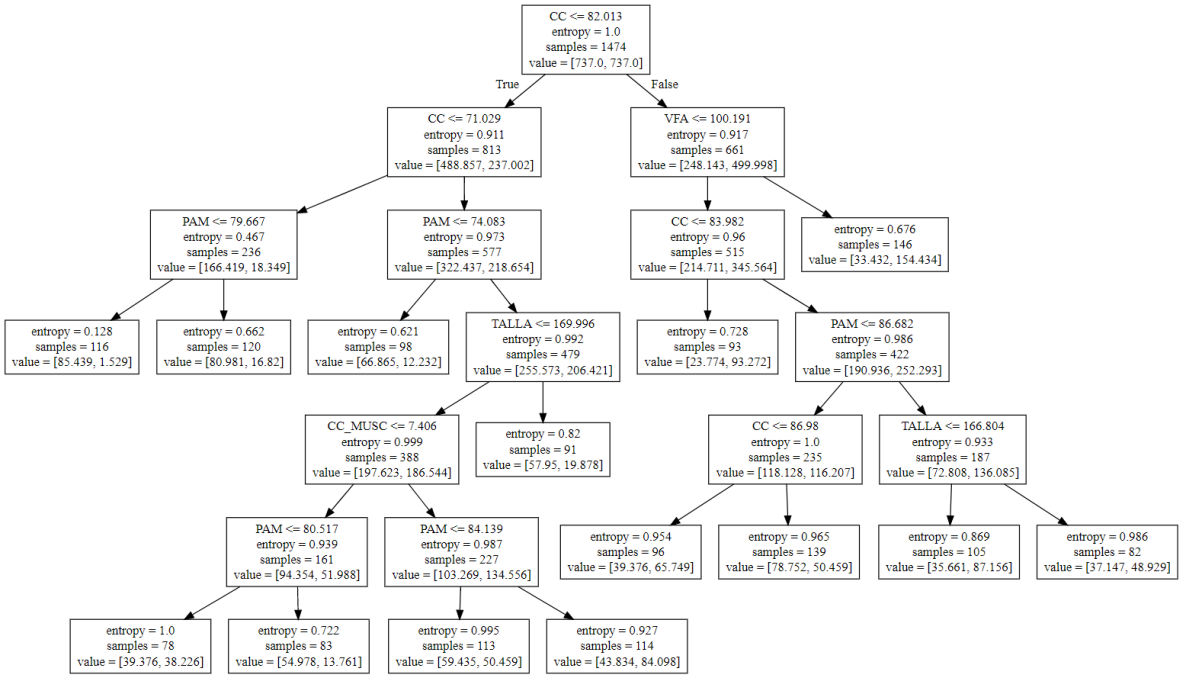
```
In [26]: datos_roi['HDL_Class'] = np.where(predicted_proba>umbral,1,0)

         datos_roi.to_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_roi.csv', index=False)
```

3.1.1.11. Interpretación del modelo campeón

El modelo seleccionado como campeón para hacer la clasificación de si los pacientes tienen (o no) su colesterol HDL por debajo de los 40 mg/dL es el random forest, el cual está constituido por 68 árboles de decisión (n_estimators), estos usan la entropía como función para medir la impureza y pueden utilizar un máximo de 8 variables para su construcción. Los criterios de parada para estos árboles son: que las hojas (nodos finales) tengan al menos el 5 % de las observaciones o que la profundidad máxima sea de 29 decisiones.

Por último, para ayudar a la interpretación del funcionamiento del modelo campeón, se muestra el árbol de decisión construido (68 árboles como este fueron construidos dentro de random forest).

```
In [27]: # Nombres:
         Xnom=data[var_predictivas]

         # Árbol:
         tree.export_graphviz(tr_clf, out_file="tree_clf_HDL.dot", feature_names=Xnom.columns)
         with open("tree_clf_HDL.dot") as f:
             dot_graph = f.read()
         graphviz.Source(dot_graph)
```



La forma de leer la representación gráfica del árbol de decisión es de arriba hacia abajo, dentro de cada recuadro se encuentra la decisión que se toma en ese nodo, el valor de la medida de impureza, el número de observaciones que hay en ese nodo y cuántas observaciones hay en cada una de las clases con lo cual se puede determinar a qué clase que se asignarían las observaciones de ese nodo.

Por ejemplo, en el nodo inicial del árbol anterior se tienen 1,474 observaciones en total, 737 eventos (482 × 1.53) y 737 no-eventos (992 × 0.74) (recordar que se esta utilizando el parámetro class_weight='balanced'). Si se tratara de un nodo final, entonces a todas las observaciones de ese nodo se les diagnosticaría HDL debajo del umbral. La decisión que se evalúa en ese nodo es si la circunferencia de cintura es menor o igual a 82.013 cm, si se cumple, entonces la observación pasa al nodo izquierdo, si no al nodo derecho, y el valor de la función de impureza (entropía) de esta decisión es de 1.

3.1.2. Modelos supervisados para regresión

En esta sección se llevará a cabo la construcción de los modelos supervisados de Machine Learning que realicen una predicción de cuál sería el valor del colesterol HDL de los pacientes.

3.1.2.1. Librerías utilizadas

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import RandomizedSearchCV,GridSearchCV

        from sklearn import tree
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.ensemble import GradientBoostingRegressor
        from xgboost import XGBRegressor

        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import mean_squared_error

        #conda install python-graphviz
        import graphviz
```

3.1.2.2. Datos utilizados

Se importan los datos que resultaron del proceso de preparación de datos.

```
In [2]: data = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_prep.csv')
```

Se seleccionan las variables que serán utilizadas como variables dependientes.

```
In [3]: var_predictivas = ['CC','VFA','CC_GRAS','CC_MUSC','CC_AGUA','PAM',
                          'TALLA','IMC','TARGET_HDL']
        X = data[var_predictivas]
        y = data['HDL']
```

3.1.2.3. Separación de datos

En el modelado predictivo, la estrategia estándar para una evaluación honesta del rendimiento de los modelos se da a través de la separación de los datos en dos subconjuntos. Una parte de la fuente de datos se utiliza para ajustar los modelos: el conjunto de datos de entrenamiento. El resto de la fuente de datos se utiliza para dar una estimación honesta final de la generalizacion de los modelos: el conjunto de datos de validación. La selección de

qué datos van en cada uno de los subconjuntos antes mencionados se hace de manera aleatoria.

Para el presente trabajo el conjunto de entrenamiento representa el 70 % de los datos originales y el conjunto de validación se compone del 30 % restante.

```
In [4]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=1)
```

3.1.2.4. Árbol de decisión

El primer modelo que se contruye es un Árbol de Decisión, de la librería Scikit-learn [78], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el coeficiente de determinación como métrica para la medición del desempeño de los modelos.

```
In [5]: # Parámetros para el modelo:
        param_dict_tr= dict(criterion=['mse','friedman_mse','mae'],
                            splitter=['best','random'], max_depth=range(2,30),
                            max_features=range(2,10), min_samples_leaf=[0.05])

        # Se crea el árbol de decisión para regresión:
        tr_reg = DecisionTreeRegressor()

        # Autotuning del modelo:
        rsgrid_tr = RandomizedSearchCV(tr_reg, param_dict_tr, cv=20, n_iter=40,
                                       n_jobs=-1, scoring='r2')
        rsgrid_tr.fit(X_train,y_train)
        tr_reg = rsgrid_tr.best_estimator_

        # Se entrena el árbol de decisión:
        tr_reg = tr_reg.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Árbol de Decisión con los cuales se maximiza el valor la R-cuadrada.

```
In [6]: rsgrid_tr.best_estimator_

Out[6]: DecisionTreeRegressor(criterion='friedman_mse', max_depth=19, max_features=8,
                              min_samples_leaf=0.05, splitter='random')
```

3.1.2.5. Random Forest

El segundo modelo que se contruye es un Random Forest, de la librería Scikit-learn [79], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el coeficiente de determinación como métrica para la medición del desempeño de los modelos.

```
In [7]: # Parámetros para el modelo:
        param_dict_rf= dict(n_estimators=range(50,150), criterion=['mse','mae'],
                            max_depth=range(20,30), max_features=range(2,10),
                            min_samples_leaf=[0.05])

        # Se crea el Random Forest para regresión:
        rf_reg = RandomForestRegressor()

        #Autotuning del modelo:
        rsgrid_rf = RandomizedSearchCV(rf_reg, param_dict_rf, cv=20, n_iter=40,
                                       n_jobs=-1, scoring='r2')
        rsgrid_rf.fit(X_train,y_train)
```

```
        rf_reg = rsgrid_rf.best_estimator_

        # Se entrena el Random Forest
        rf_reg = rf_reg.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Random Forest con los cuales se maximiza el valor la R-cuadrada.

```
In [8]: rsgrid_rf.best_estimator_

Out[8]: RandomForestRegressor(max_depth=22, max_features=9, min_samples_leaf=0.05, n_estimators=81)
```

3.1.2.6. Gradient Boosting

El tercer modelo que se contruye es un Gradient Boosting, de la librería Scikit-learn [80], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el coeficiente de determinación como métrica para la medición del desempeño de los modelos.

```
In [9]: # Parámetros para el modelo:
        param_dict_gb= dict(criterion=['friedman_mse','mse','mae'],
                            loss=['ls','lad','huber','quantile'],
                            learning_rate=[0.05,0.10,0.15,0.20,0.25,0.30,0.5,0.8,1],
                            max_depth=range(2,10), max_features=range(2,10),
                            min_samples_leaf=[0.05], n_estimators=range(50,150))

        # Se crea el Gradient Boosting para regresión:
        gb_reg = GradientBoostingRegressor()

        #Autotuning del modelo:
        rsgrid_gb = RandomizedSearchCV(gb_reg, param_dict_gb, cv=20, n_iter=40,
                                       n_jobs=-1, scoring='r2')
        rsgrid_gb.fit(X_train,y_train)
        gb_reg = rsgrid_gb.best_estimator_

        # Se entrena el Gradient Boosting
        gb_reg = gb_reg.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Gradient Boosting con los cuales se maximiza el valor la R-cuadrada.

```
In [10]: rsgrid_gb.best_estimator_

Out[10]: GradientBoostingRegressor(learning_rate=0.05, max_features=7,
                           min_samples_leaf=0.05, n_estimators=70)
```

3.1.2.7. XGBoost

El último modelo que se contruye es un XGBoost, de la librería XGBoost [75], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el coeficiente de determinación como métrica para la medición del desempeño de los modelos.

```
In [11]: # Parámetros para el modelo:
         param_dict_xgb= dict(booster=['gbtree', 'dart'],
                              eta=[0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
                              max_depth=range(20,30), n_estimators=range(50,150))

         # Se crea el XGBoost para regresión:
```

```
xg_reg = XGBRegressor()

#Autotuning del modelo:
rsgrid_xgb = RandomizedSearchCV(xg_reg, param_dict_xgb, cv=20, n_iter=40,
                                n_jobs=-1, scoring='r2')
rsgrid_xgb.fit(X_train,y_train)
xg_reg = rsgrid_xgb.best_estimator_

# Se entrena el XGBoost:
xg_reg = xg_reg.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del XGBoost con los cuales se maximiza el valor la R-cuadrada.

```
In [12]: rsgrid_xgb.best_estimator_
```

```
Out[12]: XGBRegressor(base_score=0.5, booster='dart', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, eta=0.1, gamma=0,
             gpu_id=-1, importance_type='gain', interaction_constraints='',
             learning_rate=0.100000001, max_delta_step=0, max_depth=20,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=69, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

3.1.2.8. Estadísticos de precisión

Para medir el desempeño de los modelos se utilizará el conjunto de datos de validación y se analizarán los estadísticos RMSE y el coeficiente de determinación de cada uno de los modelos.

```
In [13]: modelos = [{'label':'Decision Tree','modelo':tr_reg,},
                    {'label':'Random Forest','modelo':rf_reg,},
                    {'label':'Gradient Boosting','modelo':gb_reg,},
                    {'label':'XGBoost','modelo':xg_reg,}]

         for m in modelos:
             modelo = m['modelo']
             R2 =  modelo.score(X_test, y_test)
             y_pred = modelo.predict(X_test)
             print('%s R-squared: %0.2f' % (m['label'], R2))
             print("%s RMSE: %.2f" %
         (m['label'],mean_squared_error(y_test,y_pred,squared=False)))


         Decision Tree R-squared: 0.39
         Decision Tree RMSE: 8.04
         Random Forest R-squared: 0.40
         Random Forest RMSE: 7.98
         Gradient Boosting R-squared: 0.40
         Gradient Boosting RMSE: 7.99
         XGBoost R-squared: 0.26
         XGBoost RMSE: 8.90
```

Al analizar el coeficiente de determinación se puede observar que los primeros tres modelos son los que tienen un mejor desempeño y, dentro de estos tres modelos, los dos mejores son el Random Forest y el Gradient Boosting. Para determinar cual de los dos es el modelo campeón, se analiza el RMSE y se selecciona aquel que tiene el error más chico.

```
In [14]: champion_model = rf_reg
```

3.1.2.9. Valores pronosticados

A continuación, se muestran los valores del colesterol HDL (medidos a través de una química sanguínea) de los primeros 15 pacientes, pertenecientes al grupo utilizado para la validación del modelo, junto con los valores pronosticados por el modelo campeón.

```
In [15]: y_pred = pd.DataFrame(champion_model.predict(X_test),columns=['HDL_pred'])
         y_test.reset_index(inplace=True, drop=True)
         datos_pronosticados = pd.concat([y_test,y_pred], axis=1)
         datos_pronosticados.head(15)
```

Out[15]:

	HDL	HDL_pred
0	62	53.238732
1	46	51.856654
2	27	36.159174
3	49	51.586966
4	32	35.560056
5	54	54.362711
6	66	55.974980
7	46	50.021608
8	47	52.912237
9	51	51.726978
10	47	51.957330
11	69	48.110988
12	38	35.453361
13	48	56.671593
14	38	35.948586

3.1.2.10. Datos para ROI

Para hacer el ejercicio de ROI se utilizará un subconjunto de los datos originales (el 10 %, elegido aleatoriamente), los cuales serán calificados por cada uno de los modelos campeones, en este caso, el modelo campeón para pronosticar el valor de colesterol HDL que tendría el paciente. La finalidad es, dentro de las conclusiones, determinar la relación costo-beneficio de usar este tipo de modelos para diagnosticar la prevalencia del Síndrome Metabólico en jóvenes mexicanos.

```
In [16]: datos_roi = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_roi.csv')

In [17]: var_predictivas.remove('TARGET_HDL')
         var_predictivas.append('HDL_Class')

         X_roi = datos_roi[var_predictivas]
         X_roi= X_roi.rename(columns={'HDL_Class':'TARGET_HDL'})

         y_roi = datos_roi['HDL']
```

Se hace el pronóstico del valor de colesterol HDL que tendrían los pacientes:

```
In [18]: predicted_val = pd.DataFrame(champion_model.predict(X_roi),columns=['HDL_reg'])

         datos_pronosticados = pd.concat([y_roi,predicted_val], axis=1)
         datos_pronosticados.head(15)
```

Out[18]:

	HDL	HDL_reg
0	63	35.508384
1	37	35.563940
2	59	55.298851
3	36	52.461583
4	52	54.820179
5	46	35.041791

```
6    39  34.796173
7    35  34.999452
8    55  35.928964
9    47  36.122852
10   66  35.317724
11   42  50.122697
12   72  54.936378
13   36  35.430582
14   31  35.573788
```

Se calcula la raíz cuadrada del error cuadrático medio (RMSE), dado que una de las variables de entrada para este modelo (TARGET_HDL)es la variable de salida del modelo de clasificación se incrementa el error en el pronóstico.

```
In [19]: RMSE = mean_squared_error(y_pred=predicted_val, y_true=y_roi, squared=False)
         print("RMSE: %.2f" % (RMSE))

         RMSE: 11.63
```

Se guardan los valores pronosticados por el modelo.

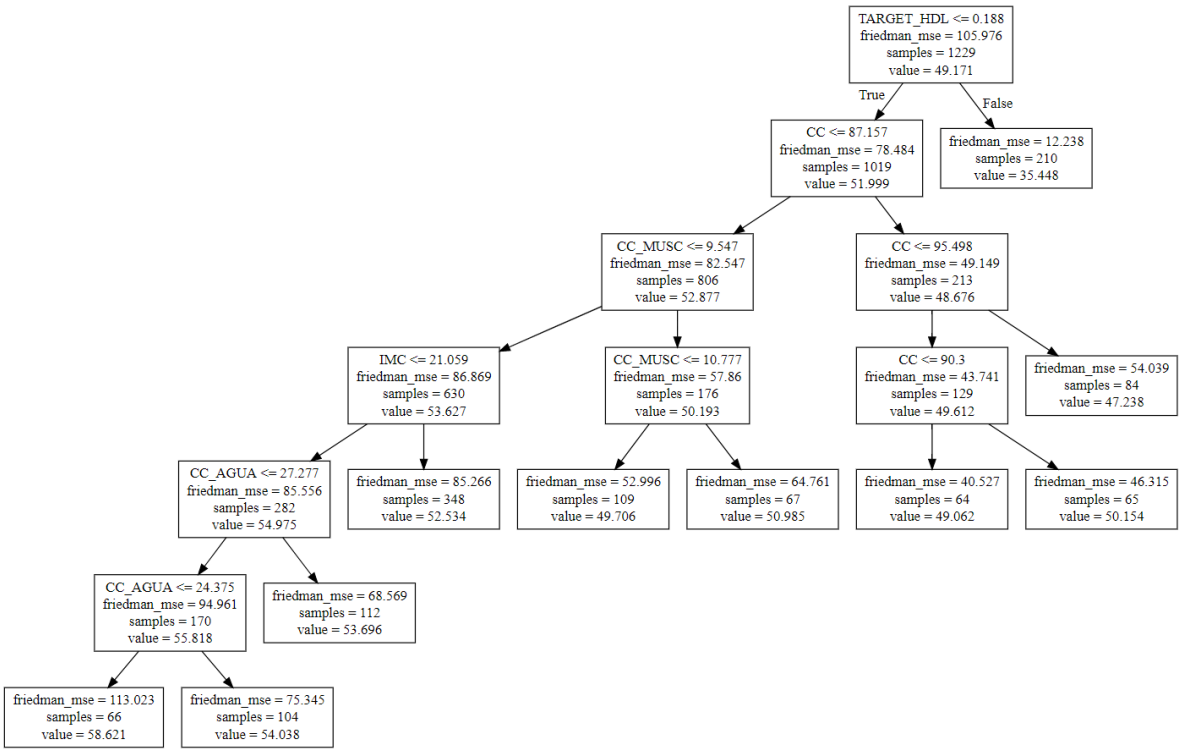
```
In [20]: datos_roi['HDL_reg'] = champion_model.predict(X_roi)

         datos_roi.to_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_roi.csv',
                          index=False)
```

3.1.2.11. Interpretación del modelo campeón

El modelo seleccionado como campeón para hacer la estimación de los valores de colesterol HDL de los pacientes es el random forest, el cual está constituido por 81 árboles de decisión (n_estimators), estos usan el error cuadrático medio como función para medir la impureza y pueden utilizar un máximo de 9 variables para su construcción. Los criterios de parada para estos árboles son: que las hojas (nodos finales) tengan al menos el 5 % de las observaciones o que la profundidad máxima sea de 22 decisiones.

Por último, para ayudar a la interpretación del funcionamiento del modelo campeón, se muestra el árbol de decisión construido (81 árboles como este fueron construidos dentro de random forest).

```
In [21]: # Nombres:
         var_predictivas.remove('HDL_Class')
         var_predictivas.append('TARGET_HDL')
         Xnom=data[var_predictivas]

         # Árbol:
         tree.export_graphviz(tr_reg, out_file="tree_reg_HDL.dot", feature_names=Xnom.columns)
         with open("tree_reg_HDL.dot") as f:
             dot_graph = f.read()
         graphviz.Source(dot_graph)
```



La forma de leer la representación gráfica del árbol de decisión es de arriba hacia abajo, dentro de cada recuadro se encuentra la decisión que se toma en ese nodo, el valor de la medida de impureza, el número de observaciones que hay en ese nodo y el valor que se le asignaría a las observaciones de ese nodo (la media de variable objetivo de las observaciones en el nodo).

Por ejemplo, en el nodo inicial del árbol anterior se tienen 1,229 observaciones en total. Si se tratara de un nodo final, entonces a todas las observaciones de ese nodo se les pronosticaría un valor de colesterol HDL de 49.171 mg/dL. La decisión que se evalúa en ese nodo es la clasificación hecha por el modelo de clasificación, i.e., si el colesterol HDL está por debajo del umbral, si se cumple, entonces la observación pasa al nodo derecho, si no al nodo izquierdo, y el valor de la función de impureza (friedman mse) de esta decisión es de 105.976.

3.2. Modelos predictivos para detectar niveles altos de triglicéridos

3.2.1. Modelos supervisados para clasificación

En esta sección se llevará a cabo la construcción de los modelos supervisados de Machine Learning que calculen la probabilidad de que el paciente tenga su nivel de triglicéridos por arriba de los 150 mg/dL (evento) para posteriormente hacer la clasificación de Alterado (1) y No Alterado (0), si esta por arriba o por debajo del umbral, respectivamente.

3.2.1.1. Librerías utilizadas

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib
        import matplotlib.pyplot as plt

        from collections import Counter
        from imblearn.combine import SMOTETomek
        from sklearn.model_selection import train_test_split

        from sklearn.model_selection import RandomizedSearchCV,GridSearchCV

        from sklearn import tree
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import GradientBoostingClassifier
        from xgboost import XGBClassifier

        from sklearn.metrics import plot_confusion_matrix, confusion_matrix
        from sklearn.metrics import precision_score, recall_score
        from sklearn.metrics import fbeta_score, make_scorer
        from sklearn.metrics import roc_auc_score, roc_curve

        #conda install python-graphviz
        import graphviz
```

3.2.1.2. Datos utilizados

Se importan los datos que resultaron del proceso de preparación de datos:

```
In [2]: data = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_prep.csv')
```

Se seleccionan las variables que serán utilizadas como variables dependientes (variables predictivas):

```
In [3]: var_predictivas = ['CC','VFA','CC_GRAS','CC_MUSC','CC_AGUA','PAM','TALLA','IMC']
        X = data[var_predictivas]
        y = data['TARGET_TRI']
```

3.2.1.3. Separación de datos

En el modelado predictivo, la estrategia estándar para una evaluación honesta del rendimiento de los modelos se da a través de la separación de los datos en dos subconjuntos. Una parte de la fuente de datos se utiliza para ajustar los modelos: el conjunto de datos

de entrenamiento. El resto de la fuente de datos se utiliza para dar una estimación honesta final de la generalización de los modelos: el conjunto de datos de validación. La selección de qué datos van en cada uno de los subconjuntos antes mencionados se hace de manera aleatoria.

Para el presente trabajo el conjunto de entrenamiento representa el 70 % de los datos originales y el conjunto de validación se compone del 30 % restante.

```
In [4]: print('Original dataset shape %s' % Counter(y))

Original dataset shape Counter({0: 1479, 1: 278})

In [5]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=13719)
        print('Train dataset shape %s' % Counter(y_train))
        print('Test dataset shape %s' % Counter(y_test))

Train dataset shape Counter({0: 1038, 1: 191})
Test dataset shape Counter({0: 441, 1: 87})
```

Dado que la muestra de datos para el entrenamiento no está balanceada (15.54 % de Eventos vs 84.46 % de No-Eventos)[4], se utilizará una técnica que consiste en aplicar simultáneamente un algoritmo de subsampling[5] y otro de oversampling[6] al conjunto de datos. Se utilizará SMOTE (Synthetic Minority Oversampling Technique) para el oversampling, el cual busca puntos vecinos cercanos y agrega puntos en línea recta entre ellos. Y se utilizará Tomek para el undersampling, el cual quita elementos de la clase mayoritaria que sean "nearest neighbor" para delimitar mejor la zona limítrofe de las clases [77] [45]. El objetivo de aplicar estos métodos a la muestra de entrenamiento es ayudar a los modelos para que puedan aprender a diferenciar las clases de mejor manera y que tengan una buena generalización.

```
In [6]: smotetomek = SMOTETomek(sampling_strategy=0.5,random_state=13719)
        X_train, y_train = smotetomek.fit_resample(X_train, y_train)
        print('Resampled train dataset shape %s' % Counter(y_train))

Resampled train dataset shape Counter({0: 1016, 1: 497})
```

3.2.1.4. Árbol de decisión

El primer modelo que se contruye es un Árbol de Decisión, de la librería Scikit-learn [44], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el ROC AUC como métrica para la medición del desempeño de los modelos.

```
In [7]: # Parámetros para el modelo:
        param_dict_tr= dict(criterion=['gini','entropy'], splitter=['best','random'],
                            max_depth=range(2,30), max_features=range(2,9),
                            min_samples_leaf=[0.05], class_weight=['balanced'])

        # Se crea el árbol de decisión para clasificación:
        tr_clf = DecisionTreeClassifier()

        # Autotuning del modelo:
```

[4]El conjunto de datos se compone de 15.82 % de Eventos y 84.18 % de No-Eventos y en la muestra de validación se tiene 16.48 % de Eventos y 83.52 % de No-Eventos.

[5]Subsampling: eliminar observaciones de manera aleatoria perteneciente a la clase mayoritaria [76].

[6]Oversampling: duplicar observaciones pertenecientes a la clase minoritaria [76].

```
        rsgrid_tr = RandomizedSearchCV(tr_clf, param_dict_tr, cv=20, n_iter=40, n_jobs=-1,
                                       scoring='roc_auc')
        rsgrid_tr.fit(X_train,y_train)
        tr_clf = rsgrid_tr.best_estimator_

        # Se entrena el árbol de decisión:
        tr_clf = tr_clf.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Árbol de Decisión con los cuales se maximiza el área debajo de la curva ROC.

```
In [8]: rsgrid_tr.best_estimator_


Out[8]: DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=20, max_features=5, min_samples_leaf=0.05)
```

3.2.1.5. Random Forest

El segundo modelo que se contruye es un Random Forest, de la librería Scikit-learn [73], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el ROC AUC como métrica para la medición del desempeño de los modelos.

```
In [9]: # Parámetros para el modelo:
        param_dict_rf= dict(n_estimators=range(50,150), criterion=['gini','entropy'],
                            max_depth=range(20,30), max_features=range(2,9),
                            min_samples_leaf=[0.05], class_weight=['balanced'])

        # Se crea el Random Forest para clasificación:
        rf_clf = RandomForestClassifier()

        #Autotuning del modelo:
        rsgrid_rf = RandomizedSearchCV(rf_clf, param_dict_rf, cv=20, n_iter=40, n_jobs=-1,
                                       scoring='roc_auc')
        rsgrid_rf.fit(X_train,y_train)
        rf_clf = rsgrid_rf.best_estimator_

        # Se entrena el Random Forest
        rf_clf = rf_clf.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Random Forest con los cuales se maximiza el área debajo de la curva ROC.

```
In [10]: rsgrid_rf.best_estimator_


Out[10]: RandomForestClassifier(class_weight='balanced', criterion='entropy',
                        max_depth=20, max_features=6, min_samples_leaf=0.05,
                        n_estimators=135)
```

3.2.1.6. Gradient Boosting

El tercer modelo que se contruye es un Gradient Boosting, de la librería Scikit-learn [74], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el ROC AUC como métrica para la medición del desempeño de los modelos.

```
In [11]: # Parámetros para el modelo:
         param_dict_gb= dict(criterion=['friedman_mse', 'mse', 'mae'],
                             loss=['deviance','exponential'],
                             learning_rate=[0.05,0.10,0.15,0.20,0.25,0.30,0.5,0.8,1],
                             max_depth=range(20,30), max_features=range(2,9),
                             min_samples_leaf=[0.05], n_estimators=range(50,150))

         # Se crea el Gradient Boosting para clasificación:
         gb_clf = GradientBoostingClassifier()

         #Autotuning del modelo:
         rsgrid_gb = RandomizedSearchCV(gb_clf, param_dict_gb, cv=20, n_iter=40, n_jobs=-1,
                                        scoring='roc_auc')
         rsgrid_gb.fit(X_train,y_train)
         gb_clf = rsgrid_gb.best_estimator_

         # Se entrena el Gradient Boosting
         gb_clf = gb_clf.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Gradient Boosting con los cuales se maximiza el área debajo de la curva ROC.

```
In [12]: rsgrid_gb.best_estimator_

Out[12]: GradientBoostingClassifier(learning_rate=1, max_depth=24, max_features=4,
                                    min_samples_leaf=0.05, n_estimators=145)
```

3.2.1.7. XGBoost

El último modelo que se contruye es un XGBoost, de la librería XGBoost [75], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el ROC AUC como métrica para la medición del desempeño de los modelos.

```
In [13]: # Parámetros para el modelo:
         param_dict_xgb= dict(booster=['gbtree', 'dart'],
                              eta=[0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
                              max_depth=range(20,30),n_estimators= range(50,150),
                              scale_pos_weight=[1,2,2.5])

         # Se crea el XGBoost para clasificación:
         xgb_clf = XGBClassifier()

         #Autotuning del modelo:
         rsgrid_xgb = RandomizedSearchCV(xgb_clf, param_dict_xgb, cv=20, n_iter=40, n_jobs=-1,
                                         scoring='roc_auc')
         rsgrid_xgb.fit(X_train,y_train)
         xgb_clf = rsgrid_xgb.best_estimator_

         # Se entrena el XGBoost:
         xgb_clf = xgb_clf.fit(X_train,y_train)
```

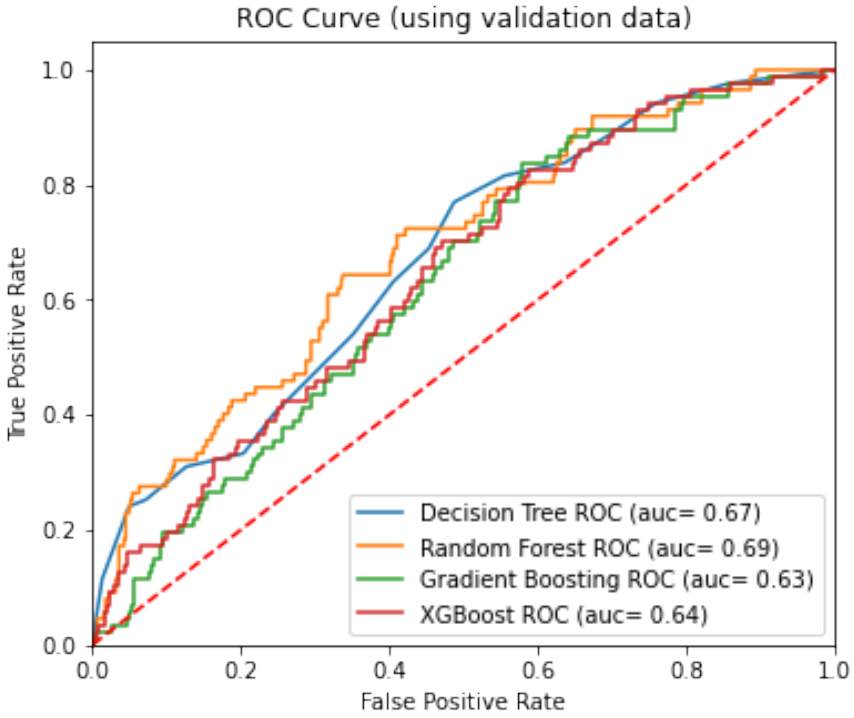
A continuación, se muestran los parámetros del XGBoost con los cuales se maximiza el área debajo de la curva ROC.

```
In [14]: rsgrid_xgb.best_estimator_

Out[14]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eta=0.25, gamma=0,
              gpu_id=-1, importance_type='gain', interaction_constraints='',
              learning_rate=0.25, max_delta_step=0, max_depth=29,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=143, n_jobs=0, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

3.2.1.8. Estadísticos de precisión

Para medir el desempeño de los modelos se utilizará el conjunto de datos de validación y se analizarán, en primer lugar, las curvas ROC, y del área debajo de las mismas, de cada uno de los modelos.

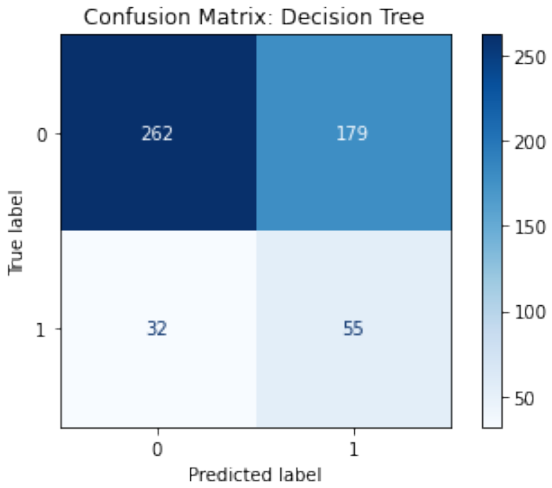
```
In [15]: plt.figure(figsize=(6,5))
         for m in modelos:
             modelo = m['modelo']
             fpr, tpr, thresholds = roc_curve(y_test, modelo.predict_proba(X_test)[:,1])
             auc = roc_auc_score(y_true=y_test, y_score=modelo.predict_proba(X_test)[:,1])
             plt.plot(fpr,tpr,label='%s ROC (auc= %0.2f)' % (m['label'], auc))
         plt.plot([0,1],[0,1],'r--')
         plt.xlim([0.0,1.0])
         plt.ylim([0.0,1.05])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('ROC Curve (using validation data)')
         plt.legend(loc="lower right")
         plt.show()
```



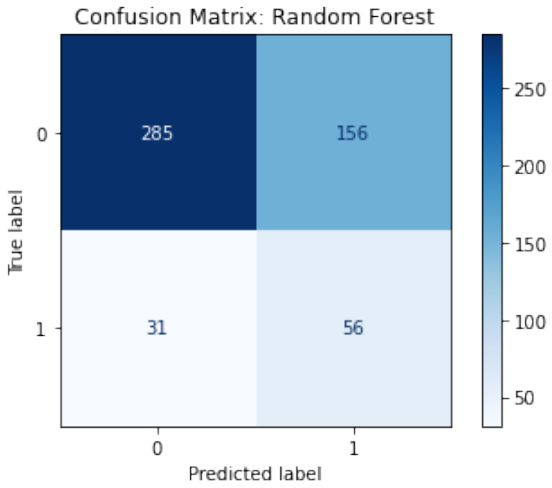
En segundo lugar, se analizará la matriz de confusión de cada uno de los modelos junto con las métrícas de Precisión, Sensibilidad y el valor F2.

```
In [16]: modelos = [{'label':'Decision Tree','modelo':tr_clf},
                   {'label':'Random Forest','modelo':rf_clf},
                   {'label':'Gradient Boosting','modelo':gb_clf},
                   {'label':'XGBoost','modelo':xgb_clf}]

         for m in modelos:
             modelo = m['modelo']
```

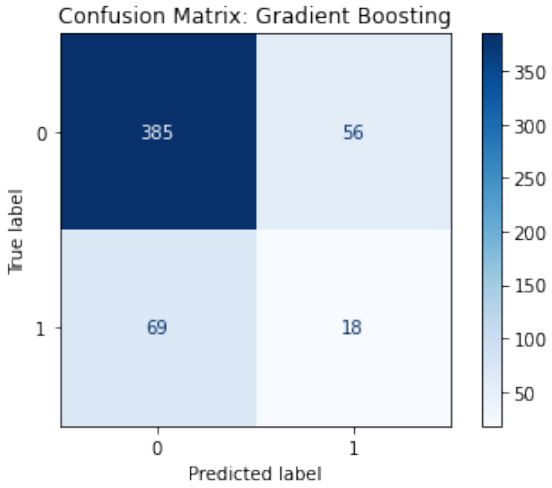
```
disp = plot_confusion_matrix(modelo, X_test, y_test, cmap=plt.cm.Blues)
disp.ax_.set_title('Confusion Matrix: %s' % (m['label']))
plt.show()

precision = precision_score(y_pred=modelo.predict(X_test),y_true=y_test)
recall = recall_score(y_pred=modelo.predict(X_test),y_true=y_test)
f2 = fbeta_score(y_pred=modelo.predict(X_test),y_true=y_test, beta=2, average='weighted')

print('Precisión = %0.2f' % (precision))
print('Sensibilidad = %0.2f' % (recall))
print('F2 score = %0.2f' % (f2))
```



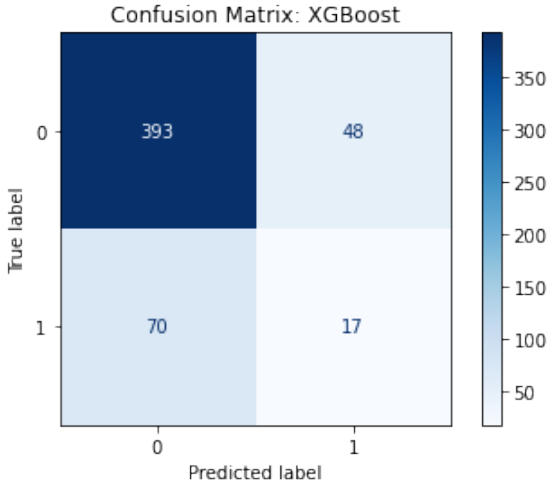

```
Precisión = 0.24
Sensibilidad = 0.63
F2 score = 0.61
```




```
Precisión = 0.26
Sensibilidad = 0.64
F2 score = 0.65
```




```
Precisión = 0.24
Sensibilidad = 0.21
F2 score = 0.76
```




```
Precisión = 0.26
Sensibilidad = 0.20
F2 score = 0.77
```

Al explorar las curvas ROC y el AUC de los modelos se puede observar que el modelo que tiene un mejor desempeño es el Random Forest y, al analizar las matrices de confusión, se confirma como modelo campeón porque es el modelo que identifica la mayor cantidad del verdaderos positivos y, por ende, minimiza los falsos negativos; se prefiere sobre el árbol de decisión porque tiene una menor cantidad de falsos positivos.

Se busca minimizar la máximo la cantidad de falsos negativos ya que, al traducirlo a la práctica médica, lo que estaría haciendo el modelo es decirle al paciente que sus niveles de triglicéridos no están alterados, cuando si lo están, por lo que no recibiría ningún tipo de tratamiento para mitigar este padecimiento; por otro lado, los falsos negativos no afectan a

los pacientes, ya que, en este caso, se le diría al paciente que tiene que hacer dieta y ejercicio para corregir estos valores, lo cual es algo benéfico para cualquier persona (con y sin Síndrome Metabólico), y se le haría (ahora sí) una química sanguínea para que un médico determine los siguientes pasos.

```
In [17]: champion_model = rf_clf
```

3.2.1.9. Umbral de probabilidad óptimo

Una vez seleccionado el modelo campeón, lo siguiente es identificar cuál es el umbral (punto de corte) de probabilidad con el cual se mejora la clasificación hecha por el modelo, es decir, identificar cuál es el umbral con el cual, si la probabilidad esta por arriba de ese valor entonces la observación se clasifica como evento y si no, entonces se clasifica como no evento. Para llevar a cabo esta tarea, se calculan las métricas de precisión (precisión, sensibilidad, F-Score y AUC) usando diferentes puntos de corte. A continuación, se muestra una tabla con el top 10 de los valores ordenados por AUC, dentro de los cuales se utilizará juicio experto para seleccionar el umbral de probabilidad óptimo.

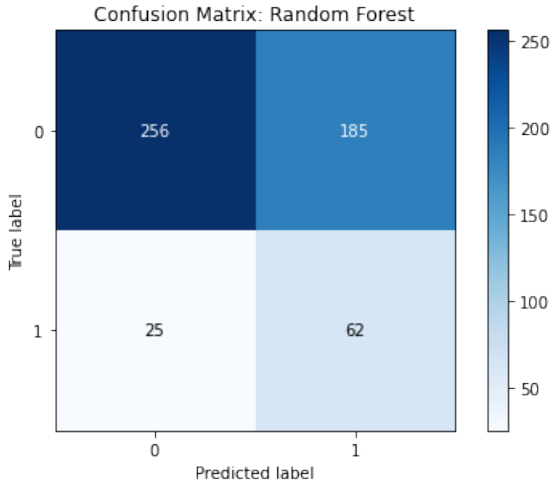
```
In [18]: fpr, tpr, thresholds = roc_curve(y_test, champion_model.predict_proba(X_test)[:,1])
         predicted_proba = champion_model.predict_proba(X_test)[:,1]

In [19]: precision_ls = []
         recall_ls = []
         fscore_ls = []
         auc_ls = []

         for thres in thresholds:
             y_pred = np.where(predicted_proba>thres,1,0)
             precision_ls.append(precision_score(y_true=y_test, y_pred=y_pred, zero_division=0))
             recall_ls.append(recall_score(y_true=y_test, y_pred=y_pred))
             fscore_ls.append(fbeta_score(y_pred=y_pred, y_true=y_test, beta=2,
         average='weighted'))
             auc_ls.append(roc_auc_score(y_true=y_test, y_score=y_pred))

         thresholding = pd.concat([pd.Series(thresholds), pd.Series(precision_ls),
                                   pd.Series(recall_ls),pd.Series(fscore_ls), pd.Series(auc_ls)],
                                  axis=1)

         thresholding.columns = ['thresholds', 'Precisión','Sensibilidad','Fscore','AUC']
         thresholding.sort_values(by='AUC', ascending=False, inplace=True)
         thresholding.head(n=10)
```

Out[19]:

	Thresholds	Precisión	Sensibilidad	F-Score	AUC
77	0.511790	0.270936	0.632184	0.667975	0.648292
86	0.466027	0.251012	0.712644	0.608631	0.646571
87	0.465900	0.250000	0.712644	0.606699	0.645437
85	0.467825	0.252066	0.701149	0.614835	0.645359
75	0.514275	0.268657	0.620690	0.668211	0.643678
76	0.514046	0.267327	0.620690	0.666347	0.642544
84	0.468936	0.250000	0.689655	0.615243	0.640746
73	0.520752	0.267677	0.609195	0.670295	0.640199
72	0.533325	0.270833	0.597701	0.677915	0.640120
74	0.519112	0.266332	0.609195	0.668436	0.639065

Cabe recordar que la intención es minimizar la cantidad de falsos negativos, en otras palabras, se busca maximizar la Sensibilidad del modelo. De acuerdo con la tabla anterior, el umbral número 86 es el punto de corte que ayuda a tener una alta sensibilidad sin sacrificar demasiado la precisión y manteniendo al modelo con un buen desempeño (buen ROC AUC).

```
In [20]: umbral = thresholds[86]

In [22]: y_pred_final = np.where(predicted_proba>umbral,1,0)

         precision = precision_score(y_pred=y_pred_final, y_true=y_test)
         recall = recall_score(y_pred=y_pred_final, y_true=y_test)
         f2 = fbeta_score(y_pred=y_pred_final, y_true=y_test, beta=2, average='weighted')
         auc = roc_auc_score(y_true=y_test, y_score=y_pred_final)
         matriz = confusion_matrix(y_true=y_test, y_pred=y_pred_final)

         true_label = ['0','1']
         predicted_label = ['0','1']

         fig, ax = plt.subplots()

         im, cbar = heatmap(matriz, true_label, true_label, ax=ax,
                            cmap=plt.cm.Blues, cbarlabel="")
         texts = annotate_heatmap(im, valfmt="{x:.0f}")

         fig.tight_layout()
         plt.xlabel('Predicted label')
         plt.ylabel('True label')
         plt.title('Confusion Matrix: Random Forest')
         plt.show()

         print('Precisión = %0.2f' % (precision))
         print('Sensibilidad = %0.2f' % (recall))
         print('F2 score = %0.2f' % (f2))
         print('ROC AUC = %0.2f' % (auc))
```




```
Precisión = 0.25
Sensibilidad = 0.71
F2 score = 0.61
ROC AUC = 0.65
```

3.2.1.10. Datos para ROI

Para hacer el ejercicio de ROI se utilizará un subconjunto de los datos originales (el 10 %, elegido aleatoriamente), los cuales serán calificados por cada uno de los modelos campeones, en este caso, el modelo campeón para detectar si los triglicéridos del paciente están alterados o no. La finalidad es, dentro de las conclusiones, determinar la relación costo-beneficio de usar

este tipo de modelos para diagnosticar la prevalencia del Síndrome Metabólico en jóvenes mexicanos.

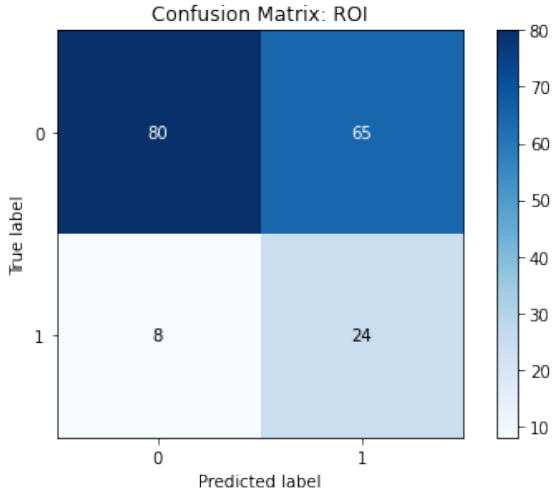
```
In [23]: datos_roi = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_roi.csv')
         X_roi = datos_roi[var_predictivas]
         y_roi = datos_roi['TARGET_TRI']
```

Se hacen calculan las probabilidades y, haciendo uso del umbral óptimo, se hace la clasificación binaria:

```
In [24]: predicted_proba = champion_model.predict_proba(X_roi)[:,1]
         predicted_class = np.where(predicted_proba>umbral,1,0)
```

Se calcula la matriz de confusión junto con los estadísticos de precisión:

```
In [25]: precision = precision_score(y_pred=predicted_class, y_true=y_roi)
         recall = recall_score(y_pred=predicted_class, y_true=y_roi)
         f2 = fbeta_score(y_pred=predicted_class, y_true=y_roi, beta=2, average='weighted')
         auc = roc_auc_score(y_true=y_roi, y_score=predicted_class)
         matriz = confusion_matrix(y_true=y_roi, y_pred=predicted_class)

         true_label = ['0','1']
         predicted_label = ['0','1']

         fig, ax = plt.subplots()

         im, cbar = heatmap(matriz, true_label, true_label, ax=ax,
                            cmap=plt.cm.Blues, cbarlabel="")
         texts = annotate_heatmap(im, valfmt="{x:.0f}")

         fig.tight_layout()
         plt.xlabel('Predicted label')
         plt.ylabel('True label')
         plt.title('Confusion Matrix: ROI')
         plt.show()

         print('Precisión = %0.2f' % (precision))
         print('Sensibilidad = %0.2f' % (recall))
         print('F2 score = %0.2f' % (f2))
         print('ROC AUC = %0.2f' % (auc))
```




```
Precisión = 0.27
Sensibilidad = 0.75
F2 score = 0.59
ROC AUC = 0.65
```

Se guarda la clasificación hecha por el modelo para que sirva como insumo para el modelo de regresión:

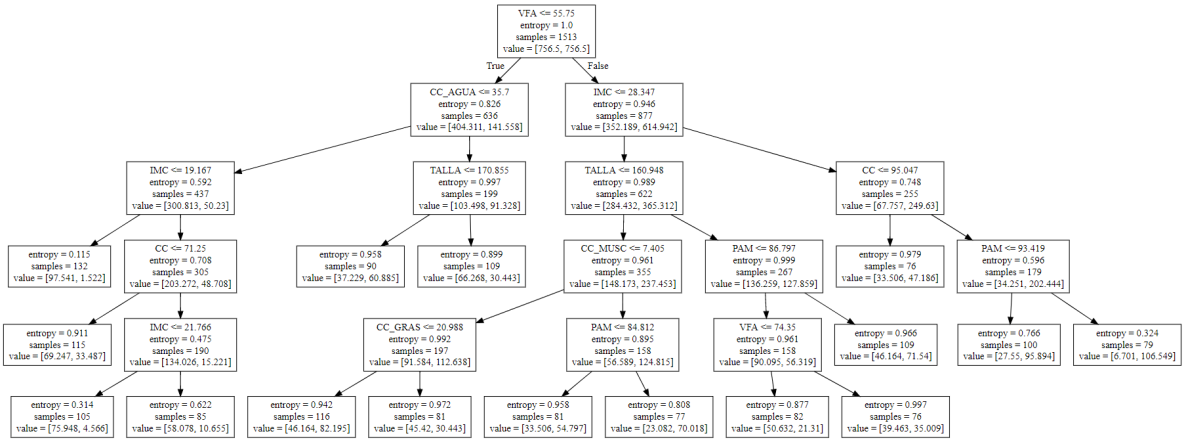
```
In [26]: datos_roi['TRI_Class'] = np.where(predicted_proba>umbral,1,0)

         datos_roi.to_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_roi.csv',
                          index=False)
```

3.2.1.11. Interpretación del modelo campeón

El modelo seleccionado como campeón para hacer la clasificación de si los pacientes tienen (o no) sus triglicéridos por arriba de los 150 mg/dL es el random forest, el cual está constituido por 135 árboles de decisión (n_estimators), estos usan la entropía como función para medir la impureza y pueden utilizar un máximo de 6 variables para su construcción. Los criterios de parada para estos árboles son: que las hojas (nodos finales) tengan al menos el 5 % de las observaciones o que la profundidad máxima sea de 20 decisiones.

Por último, para ayudar a la interpretación del funcionamiento del modelo campeón, se muestra el árbol de decisión construido (135 árboles como este fueron construidos dentro de random forest).

```
In [27]: # Nombres:
         Xnom=data[var_predictivas]

         # Árbol:
         tree.export_graphviz(tr_clf, out_file="tree_clf_TRI.dot", feature_names=Xnom.columns)
         with open("tree_clf_TRI.dot") as f:
             dot_graph = f.read()
         graphviz.Source(dot_graph)
```



La forma de leer la representación gráfica del árbol de decisión es de arriba hacia abajo, dentro de cada recuadro se encuentra la decisión que se toma en ese nodo, el valor de la medida de impureza, el número de observaciones que hay en ese nodo y cuántas observaciones hay en cada una de las clases con lo cual se puede determinar a qué clase que se asignarían las observaciones de ese nodo.

Por ejemplo, en el nodo inicial del árbol anterior se tienen 1,513 observaciones en total, 756.5 eventos (497 × 1.5221) y 756.5 no-eventos (1016 × 0.7446) (recordar que se esta

utilizando el parámetro class_weight='balanced'). Si se tratara de un nodo final, entonces a todas las observaciones de ese nodo se les diagnosticaría TRI arriba del umbral. La decisión que se evalúa en ese nodo es si el área de grasa viseral es menor o igual a 55.75 cm^2, si se cumple, entonces la observación pasa al nodo izquierdo, si no al nodo derecho, y el valor de la función de impureza (entropía) de esta decisión es de 1.

3.2.2. Modelos supervisados para regresión

En esta sección se llevará a cabo la construcción de los modelos supervisados de Machine Learning que realicen una predicción de cuál sería el valor de los triglicéridos de los pacientes.

3.2.2.1. Librerías utilizadas

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import RandomizedSearchCV,GridSearchCV

        from sklearn import tree
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.ensemble import GradientBoostingRegressor
        from xgboost import XGBRegressor

        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import mean_squared_error

        #conda install python-graphviz
        import graphviz
```

3.2.2.2. Datos utilizados

Se importan los datos que resultaron del proceso de preparación de datos.

```
In [2]: data = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_prep.csv')
```

Se seleccionan las variables que serán utilizadas como variables dependientes.

```
In [3]: var_predictivas = ['CC','VFA','CC_GRAS','CC_MUSC','CC_AGUA','PAM',
                          'TALLA','IMC','TARGET_TRI']
        X = data[var_predictivas]
        y = data['TRI']
```

3.2.2.3. Separación de datos

En el modelado predictivo, la estrategia estándar para una evaluación honesta del rendimiento de los modelos se da a través de la separación de los datos en dos subconjuntos. Una parte de la fuente de datos se utiliza para ajustar los modelos: el conjunto de datos de entrenamiento. El resto de la fuente de datos se utiliza para dar una estimación honesta final de la generalizacion de los modelos: el conjunto de datos de validación. La selección de qué datos van en cada uno de los subconjuntos antes mencionados se hace de manera aleatoria.

Para el presente trabajo el conjunto de entrenamiento representa el 70 % de los datos originales y el conjunto de validación se compone del 30 % restante.

```
In [4]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=1)
```

3.2.2.4. Árbol de decisión

El primer modelo que se contruye es un Árbol de Decisión, de la librería Scikit-learn [78], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el coeficiente de determinación como métrica para la medición del desempeño de los modelos.

```
In [5]: # Parámetros para el modelo:
        param_dict_tr= dict(criterion=['mse','friedman_mse','mae'],
                            splitter=['best','random'], max_depth=range(2,30),
                            max_features=range(2,10), min_samples_leaf=[0.05])

        # Se crea el árbol de decisión para regresión:
        tr_reg = DecisionTreeRegressor()

        # Autotuning del modelo:
        rsgrid_tr = RandomizedSearchCV(tr_reg, param_dict_tr, cv=20, n_iter=40,
                                       n_jobs=-1, scoring='r2')
        rsgrid_tr.fit(X_train,y_train)
        tr_reg = rsgrid_tr.best_estimator_

        # Se entrena el árbol de decisión:
        tr_reg = tr_reg.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Árbol de Decisión con los cuales se maximiza el valor la R-cuadrada.

```
In [6]: rsgrid_tr.best_estimator_

Out[6]: DecisionTreeRegressor(criterion='friedman_mse', max_depth=22, max_features=9,
                      min_samples_leaf=0.05, splitter='random')
```

3.2.2.5. Random Forest

El segundo modelo que se contruye es un Random Forest, de la librería Scikit-learn [79], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el coeficiente de determinación como métrica para la medición del desempeño de los modelos.

```
In [7]: # Parámetros para el modelo:
        param_dict_rf= dict(n_estimators=range(50,150), criterion=['mse','mae'],
                            max_depth=range(20,30), max_features=range(2,10),
                            min_samples_leaf=[0.05])

        # Se crea el Random Forest para regresión:
        rf_reg = RandomForestRegressor()

        #Autotuning del modelo:
        rsgrid_rf = RandomizedSearchCV(rf_reg, param_dict_rf, cv=20, n_iter=40,
                                       n_jobs=-1, scoring='r2')
        rsgrid_rf.fit(X_train,y_train)
        rf_reg = rsgrid_rf.best_estimator_

        # Se entrena el Random Forest
        rf_reg = rf_reg.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Random Forest con los cuales se maximiza el valor la R-cuadrada.

```
In [8]: rsgrid_rf.best_estimator_

Out[8]: RandomForestRegressor(max_depth=23, max_features=7, min_samples_leaf=0.05,
                              n_estimators=65)
```

3.2.2.6. Gradient Boosting

El tercer modelo que se contruye es un Gradient Boosting, de la librería Scikit-learn [80], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el coeficiente de determinación como métrica para la medición del desempeño de los modelos.

```
In [9]: # Parámetros para el modelo:
        param_dict_gb= dict(criterion=['friedman_mse','mse','mae'],
                            loss=['ls','lad','huber','quantile'],
                            learning_rate=[0.05,0.10,0.15,0.20,0.25,0.30,0.5,0.8,1],
                            max_depth=range(2,10), max_features=range(2,10),
                            min_samples_leaf=[0.05], n_estimators=range(50,150))

        # Se crea el Gradient Boosting para regresión:
        gb_reg = GradientBoostingRegressor()

        #Autotuning del modelo:
        rsgrid_gb = RandomizedSearchCV(gb_reg, param_dict_gb, cv=20, n_iter=40,
                                       n_jobs=-1, scoring='r2')
        rsgrid_gb.fit(X_train,y_train)
        gb_reg = rsgrid_gb.best_estimator_

        # Se entrena el Gradient Boosting
        gb_reg = gb_reg.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del Gradient Boosting con los cuales se maximiza el valor la R-cuadrada.

```
In [10]: rsgrid_gb.best_estimator_

Out[10]: GradientBoostingRegressor(criterion='mse', learning_rate=0.05, max_depth=8,
                                   max_features=8, min_samples_leaf=0.05,
                                   n_estimators=92)
```

3.2.2.7. XGBoost

El último modelo que se contruye es un XGBoost, de la librería XGBoost [75], además, se hace uso del método "RandomizedSearchCV" para encontrar los parámetros óptimos para el modelo, usando el coeficiente de determinación como métrica para la medición del desempeño de los modelos.

```
In [11]: # Parámetros para el modelo:
         param_dict_xgb= dict(booster=['gbtree', 'dart'],
                              eta=[0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
                              max_depth=range(20,30), n_estimators=range(50,150))

         # Se crea el XGBoost para regresión:
         xg_reg = XGBRegressor()

         #Autotuning del modelo:
```

```
rsgrid_xgb = RandomizedSearchCV(xg_reg, param_dict_xgb, cv=20, n_iter=40,
                                n_jobs=-1, scoring='r2')
rsgrid_xgb.fit(X_train,y_train)
xg_reg = rsgrid_xgb.best_estimator_

# Se entrena el XGBoost:
xg_reg = xg_reg.fit(X_train,y_train)
```

A continuación, se muestran los parámetros del XGBoost con los cuales se maximiza el valor la R-cuadrada.

```
In [12]: rsgrid_xgb.best_estimator_
```

```
Out[12]: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, eta=0.05, gamma=0,
             gpu_id=-1, importance_type='gain', interaction_constraints='',
             learning_rate=0.0500000007, max_delta_step=0, max_depth=26,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=84, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

3.2.2.8. Estadísticos de precisión

Para medir el desempeño de los modelos se utilizará el conjunto de datos de validación y se analizarán los estadísticos RMSE y el coeficiente de determinación de cada uno de los modelos.

```
In [13]: modelos = [{'label':'Decision Tree','modelo':tr_reg,},
                    {'label':'Random Forest','modelo':rf_reg,},
                    {'label':'Gradient Boosting','modelo':gb_reg,},
                    {'label':'XGBoost','modelo':xg_reg,}]

         for m in modelos:
             modelo = m['modelo']
             R2 =  modelo.score(X_test, y_test)
             y_pred = modelo.predict(X_test)
             print('%s R-squared: %0.2f' % (m['label'], R2))
             print("%s RMSE: %.2f" %
         (m['label'],mean_squared_error(y_test,y_pred,squared=False)))


         Decision Tree R-squared: 0.61
         Decision Tree RMSE: 36.97
         Random Forest R-squared: 0.61
         Random Forest RMSE: 37.00
         Gradient Boosting R-squared: 0.60
         Gradient Boosting RMSE: 37.68
         XGBoost R-squared: 0.41
         XGBoost RMSE: 45.43
```

Al analizar el coeficiente de determinación se puede observar que los primeros tres modelos son los que tienen un mejor desempeño y, dentro de estos tres modelos, los dos mejores son el Árbol de Decisión y el Random Forest. Para determinar cual de los dos es el modelo campeón, se analiza el RMSE y se selecciona aquel que tiene el error más chico, sin embargo, dado que la diferencia es tan pequeña en los dos, se decide optar por el Random Forest el cual, al ser un ensamble de árboles de decisión, es un modelo más robusto.

```
In [14]: champion_model = rf_reg
```

3.2.2.9. Valores pronosticados

A continuación, se muestran los valores de triglicéridos (medidos a través de una química sanguínea) de los primeros 15 pacientes, pertenecientes al grupo utilizado para la validación del modelo, junto con los valores pronosticados por el modelo campeón.

```
In [15]: y_pred = pd.DataFrame(champion_model.predict(X_test),columns=['TRI_pred'])
         y_test.reset_index(inplace=True, drop=True)
         datos_pronosticados = pd.concat([y_test,y_pred], axis=1)
         datos_pronosticados.head(15)
```

Out[15]:

	TRI	TRI_pred
0	80	86.011154
1	47	81.450798
2	71	93.136139
3	181	198.140109
4	107	106.339434
5	178	187.307481
6	96	79.264214
7	98	97.885221
8	65	90.044446
9	92	95.285338
10	54	81.260468
11	53	105.507304
12	304	200.428459
13	66	84.358221
14	112	97.752183

3.2.2.10. Datos para ROI

Para hacer el ejercicio de ROI se utilizará un subconjunto de los datos originales (el 10 %, elegido aleatoriamente), los cuales serán calificados por cada uno de los modelos campeones, en este caso, el modelo campeón para pronosticar el valor de triglicéridos que tendría el paciente. La finalidad es, dentro de las conclusiones, determinar la relación costo-beneficio de usar este tipo de modelos para diagnosticar la prevalencia del Síndrome Metabólico en jóvenes mexicanos.

```
In [16]: datos_roi = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_roi.csv')

In [17]: var_predictivas.remove('TARGET_TRI')
         var_predictivas.append('TRI_Class')

         X_roi = datos_roi[var_predictivas]
         X_roi= X_roi.rename(columns={'TRI_Class':'TARGET_TRI'})

         y_roi = datos_roi['TRI']
```

Se hacen el pronóstico del valor de triglicéridos que tendrían los pacientes:

```
In [18]: predicted_val = pd.DataFrame(champion_model.predict(X_roi),columns=['TRI_reg'])

         datos_pronosticados = pd.concat([y_roi,predicted_val], axis=1)
         datos_pronosticados.head(15)
```

Out[18]:

	TRI	TRI_reg
0	88	196.040679
1	185	195.403891
2	78	90.244379
3	91	78.907518
4	59	91.958599
5	206	208.833650

```
6   106  210.057706
7   111  210.822124
8   108  195.208948
9    68   83.597603
10  133  196.391109
11  169  195.208948
12   68   83.229014
13   84   91.508789
14  146  200.221081
```

Se calcula la raíz cuadrada del error cuadrático medio (RMSE), dado que una de las variables de entrada para este modelo (TARGET_TRI) es la variable de salida del modelo de clasificación se incrementa el error en el pronóstico.

```
In [19]: RMSE = mean_squared_error(y_pred=predicted_val, y_true=y_roi, squared=False)
         print("RMSE: %.2f" % (RMSE))

         RMSE: 76.81
```

Se guardan los valores pronosticados por el modelo.

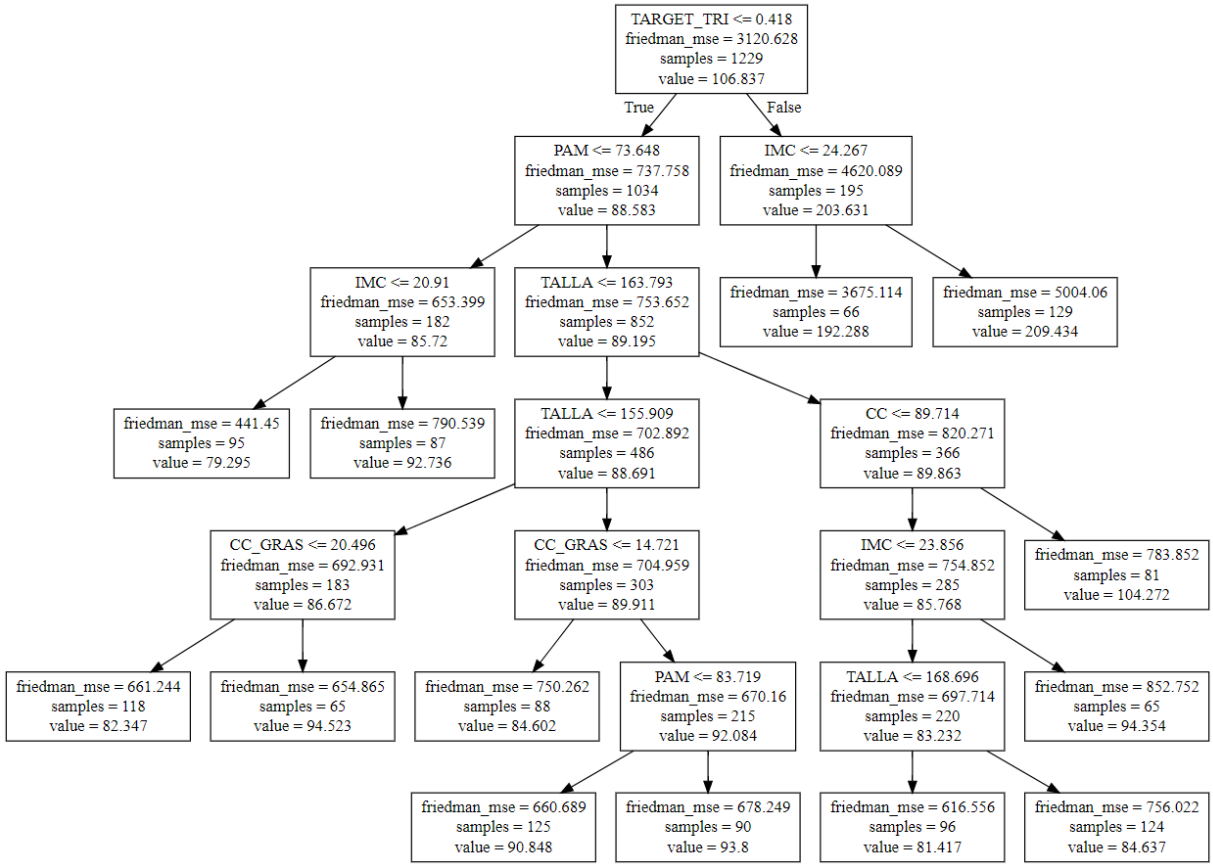
```
In [20]: datos_roi['TRI_reg'] = champion_model.predict(X_roi)

         datos_roi.to_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_roi.csv',
                          index=False)
```

3.2.2.11. Interpretación del modelo campeón

El modelo seleccionado como campeón para hacer la estimación de los valores de colesterol HDL de los pacientes es el random forest, el cual está constituido por 65 árboles de decisión (n_estimators), estos usan el error cuadrático medio como función para medir la impureza y pueden utilizar un máximo de 7 variables para su construcción. Los criterios de parada para estos árboles son: que las hojas (nodos finales) tengan al menos el 5 % de las observaciones o que la profundidad máxima sea de 23 decisiones.

Por último, para ayudar a la interpretación del funcionamiento del modelo campeón, se muestra el árbol de decisión construido (65 árboles como este fueron construidos dentro de random forest).

```
In [21]: # Nombres:
         var_predictivas.remove('TRI_Class')
         var_predictivas.append('TARGET_TRI')

         Xnom=data[var_predictivas]

         # Árbol:
         tree.export_graphviz(tr_reg, out_file="tree_reg_TRI.dot", feature_names=Xnom.columns)
         with open("tree_reg_TRI.dot") as f:
             dot_graph = f.read()
         graphviz.Source(dot_graph)
```



La forma de leer la representación gráfica del árbol de decisión es de arriba hacia abajo, dentro de cada recuadro se encuentra la decisión que se toma en ese nodo, el valor de la medida de impureza, el número de observaciones que hay en ese nodo y el valor que se le asignaría a las observaciones de ese nodo (la media de variable objetivo de las observaciones en el nodo).

Por ejemplo, en el nodo inicial del árbol anterior se tienen 1,229 observaciones en total. Si se tratara de un nodo final, entonces a todas las observaciones de ese nodo se les pronosticaría un valor de triglicéridos de 106.837 mg/dL. La decisión que se evalúa en ese nodo es la clasificación hecha por el modelo de clasificación, i.e., si los triglicéridos están por arriba del umbral, si se cumple, entonces la observación pasa al nodo derecho, si no al nodo izquierdo, y el valor de la función de impureza (friedman mse) de esta decisión es de 3,120.628.

4 | Análisis de costo-beneficio

Dentro de los Objetivos planetados al inicio de este proyecto, se encuentra la disminución del costo económico de la práctica médica. Para ilustrar el beneficio económico que se tendría gracias al uso de los modelos predictivos para ayudar a diagnosticar el Síndrome Metabólico se presenta a continuación el análisis del costo-beneficio, también conocido como análisis del retorno de la inversión (ROI, por las siglas en inglés de return on investment).

4.1. Diagnóstico del Síndrome Metabólico

Para llevar a cabo este análisis se tomó una muesta aleatoria estratificada (sobre la variable SMet) equivalente al 10 % de los datos[1], los cuales fueron calificados por los modelos de la siguiente forma:

1. Se evalúa la probabilidad de que el paciente tenga el colesterol HDL bajo y se le da una clasificación (ver Sección 3.1.1.10.).

2. Se toma en consideración el resultado del modelo de clasificación y, junto con las otras variables explicativas, se hace un pronóstico del valor de colesterol HDL que tendría el paciente (ver Sección 3.1.2.10.).

3. Se evalúa la probabilidad de que el paciente tenga los triglicéridos altos y se le da una clasificación (ver Sección 3.2.1.10.).

4. Se toma en consideración el resultado del modelo de clasificación y, junto con las otras variables explicativas, se hace un pronóstico del valor de triglicéridos que tendría el paciente (ver Sección 3.2.2.10.).

Posteriormente, haciendo uso de los factores de riesgo establecidos en la Tabla 1.2., se determina si el paciente tiene (o no) el Síndrome Metabólico (ver Sección A.3.3.). Es importante

[1]Debido a que se cuenta con muy pocas observaciones para poder separar el conjunto inicial de datos en tres (uno para el entrenamiento, otro para la validación y uno último para el ejercicio de ROI), se decidió hacer un muestreo aleatorio estratificado sobre el conjunto de datos completo. Al hacer esto se puede argumentar que el ejercicio de ROI esta sesgado, dado que no se esta ocupando un conjunto de datos totalmente ajeno a los datos utilizados para el aprendizaje de los modelos, sin embargo, debido a que la variable que se utilizó como estrato es diferente a las variables objetivo que se utilizaron para la construcción de los modelos y, también, a que se hizo el muestreo sobre los datos completos, es decir, puede haber observaciones que estaban en el conjunto de entrenamiento, pero también observaciones que estaban en el conjunto de validación, se minimiza un poco este sesgo.

mencionar que, aunque se hizo la estimación de los valores de triglicéridos y de colesterol HDL que tendrían los pacientes, se decidió utilizar el resultado de los modelos de clasificación al momento de hacer el diagnóstico, los cuales evalúan la probabilidad de que estos valores estén distorcionados y, gracias a la forma en que fueron construidos y a la identificación del umbral de probabilidad óptimo, se tiene una muy buena sensibilidad por parte de los modelos para identificar aquellos casos que sí tienen los triglicéridos altos y el colesterol HDL bajo.

Para analizar la efectividad de utilizar los modelos predictivos para la detección del Síndrome Metabólico se utilizó la matriz de confusión (Figura 4.1.), la cual reportó una sensibilidad del 91 % y una precisión del 33 %, en otras palabras, utilizando esta metodología se logró identificar a 9 de cada 10 pacientes (pertenecientes a esta muestra de la población analizada) que sí tienen el Síndrome Metabólico y, además, de cada 3 pacientes a quienes se les diagnosticó el síndrome 1 realmente lo tiene. Por otro lado, el porcentaje de casos negativos detectados correctamente por esta metodología fue del 79 %.

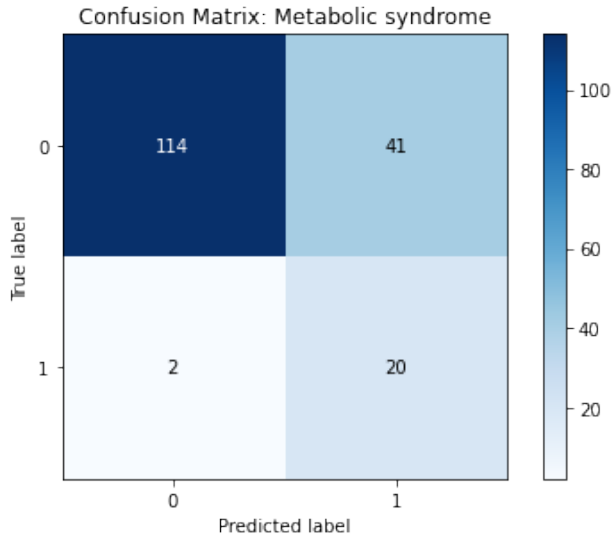


Figura 4.1: Matriz de confusión resultante del uso de modelos predictivos para ayudar a la detección del Síndrome Metabólico.

4.2. Comparación de costos

Actualmente, la manera de diagnosticar el Síndrome Metabólico en los pacientes implica tomar medidas antropométricas (circunferencia de cintura y presión arterial) junto con una muestra de sangre para mandarla al laboratorio y hacer la química sanguínea para determinar los valores de triglicéridos, colesterol HDL y glucosa. Éste es el método más preciso con el que lo médicos pueden saber si el paciente tiene el síndrome o no, sin embargo, la química sanguínea tiene un costo relacionado y un tiempo de espera de 24 horas en lo que se lleva a cabo.

La metodología propuesta en este trabajo implica tomar medidas antropométricas[2] (con la ayuda de una báscula inteligente), tomar la glucosa (usando un glucómetro) y el uso de una computadora con Python instalado para poder hacer las predicciones al instante.

Aunque en el método propuesto no es necesario realizar químicas sanguíneas, sí se necesitan considerar los siguientes materiales y sus costos:

- Paquete de glucómetro Accu-Check Instant con 150 tiras reactivas y 150 lancetas.
  Costo unitario: $1498.00 (fecha de consulta el 22 de octubre del 2020 en Link).

- Paquete de 150 Tiras reactivas Accu-Check Instant y 150 lancetas.
  Costo unitario: $926.00 (fecha de consulta el 22 de octubre del 2020 en Link).

Para hacer la comparación de los costos relacionados se tomaron en consideración los siguientes precios para las químicas sanguíneas:

1. Laboratorio Médico del Chopo: Química sanguínea integral de 45 elementos.
   Costo unitario: $809.10 (fecha de consulta el 22 de octubre del 2020 en Link).

2. Salud Digna: Paquete Completo Adulto.
   Costo unitario: $370.00 (fecha de consulta el 22 de octubre del 2020 en Link).

Tomando en cuenta la población que se consideró para llevar a cabo este análisis (177 pacientes elegidos de manera aleatoria de la muestra completa de datos), los costos[3] que se tendrían para detectar el síndrome utilizando las dos metodologías son los siguientes:

Tabla 4.1: Comparación de costos relacionados con la detección del Síndome Metabólico.

Método	Pacientes	Núm. Químicas sanguíneas	Costo total 1.	Costo total 2.
Tradicional	177	177	$143,210.70	$65,490.00
Propuesto	177	0	$2,424.00	$2,424.00
Ahorro			$140,786.70	$63,066.00

Por lo que, sin importar cual de los dos escenarios (el caro o el barato) se considere, el ahorro de dinero es impresionante (entre el 98 % y el 96 %, dependiendo del laboratorio). Sin embargo, no hay que perder de vista la efectividad para detectar el Síndrome Metabólico en los pacientes:

Tabla 4.2: Comparación de efectividad para la detección del Síndome Metabólico.

Método	SMet. VP	SMet. VN	SMet. FP	SMet. FN
Tradicional	22	145	0	0
Propuesto	20	114	41	2

[2]Se toman en cuenta no solo la circunferencia de cintura y la presión arterial, sino también la talla, el índice de masa corporal (IMC), el área de grasa viseral (VFA) y la composición corporal de agua, grasa y músculo.

[3]Para la metodología propuesta se tomará en cuenta la compra ambos paquetes presentados, aunque sobren tiras reactivas y lancetas.

Haciendo uso de la metodología tradicional se puede determinar correctamente si el paciente tiene Síndrome Metabólico (o no) para el 100 % de los casos, mientras que con la metodología propuesta se tendrían tanto falsos positivos (pacientes que erroneamente fueron clasificados con Síndrome Metabólico) como falsos negativos (pacientes que erroneamente no fueron clasificados con Sindróme Metabólico). Para minimizar la tasa de falsos negativos, el cual sería el error más peligroso para la práctica médica, se hicieron algunos procedimientos analíticos (como la selección del umbral de probabilidad óptimo para hacer la clasificación) para mejorar la sensibilidad de los modelos predictivos.

4.3. Confirmación del diagnóstico

Como ya se ha mencionado anteriormente los modelos predictivos de clasificación fueron construidos de tal manera que se minimizara la tasa de falsos negativos, en otras palabras, que se maximizara la sensibilidad de los modelos, la cual mide el porcentaje de casos positivos que si fueron detectados por los modelos. Al hacer esto, el costo que se tuvo fue un aumento en los falsos positivos.

Es importante recordar que el Síndrome Metabólico es una herramienta clínica de gran capacidad predictiva para los médicos que les permite identificar a pacientes en riesgo de padecer diabetes mellitus tipo 2 o alguna enfermedad cardiovascular [7] y que además, se esta trabajando con jóvenes de entre 18 y 24 años (la gran mayoría de entre 18 y 19 años) por lo que el tratamiento considerado para contrarrestar estos riesgos implica cambios en el estilo de vida de los pacientes como [81]:

- Hacer ejercicio con regularidad.
- Bajar de peso.
- Llevar una alimentación saludable.
- Dejar de fumar.
- Reducción o control de estrés.

En otras palabras, el tratamiento para el Síndrome Metabólico sería llevar una vida saludable, lo cual no implicaría riesgo alguno contra la salud de los pacientes que fueron clasificados erroneamente con la metodología propuesta. Sin embargo, es cierto que para algunos de los pacientes no es suficiente el cambio en el estilo de vida, por lo que el médico podría sugerir medicamentos que ayuden a controlar la presión arterial, los niveles de triglicéridos, los niveles de colesterol y/o los niveles de azúcar en sangre. Para identificar estos casos que requieren mayor atención médica se propone un enfoque híbrido que resulta de la combinación de las dos metodologías analizadas:

1. Pre-clasificación: se hace el diagnóstico utilizando la metodología que toma en cuenta los modelos predictivos.

2. Confirmación: a aquellos casos que salgan positivos, se les hace una química sanguínea para confirmar el diagnóstico.

Siguiendo este nuevo enfoque se tendrían los siguientes costos:

Tabla 4.3: Comparación de costos relacionados con la detección del Síndome Metabólico.

Método	Pacientes	Núm. Químicas sanguíneas	Costo total 1.	Costo total 2.
Tradicional	177	177	$143,210.70	$65,490.00
Híbrido	177	61	$50,853.10	$24,068.00
Ahorro			$90,357.60	$41,422.00

Por lo tanto, al seguir esta metodología para confirmar el diagnóstico, se tendría un ahorro de entre el 63 % y 64 % (dependiendo del laboratorio) y la efectivadad para detectar el síndrome metabólico sería la siguiente:

Tabla 4.4: Comparación de efectividad para la detección del Síndome Metabólico.

Método	SMet. VP	SMet. VN	SMet. FP	SMet. FN
Tradicional	22	145	0	0
Híbrido	20	145	0	2

Es importante mencionar que los costos presentados en esta sección son costos al menudeo, por lo que se podrían mejorar si se lograran conseguir mercancías y químicas sanguíneas a un precio de mayoreo considerando que se estaría analizando a una población mucho mayor que la utilizada para el análisis presentado. También se debe de tener en mente que conforme se vayan haciendo más análisis de química sanguínea y se vayan acumulando más pacientes en la información disponible para entrenar, los modelos cada vez van a ser más precisos para identificar correctamente los casos positivos y negativos, lo que implicaría hacer cada vez menos químicas sanguíneas para confirmar los diagnósticos. En cuanto a costos, una vez que se hayan adquirido los glucómetros necesarios (depende de cuántos consideren necesarios para hacer ágil la toma de medidas) el único costo que se tendría sería el costo de las tiras reactivas y las lancetas, una para cada alumno, lo cual no se compara con el costo de una química sanguínea por alumno.

5 | Conclusiones

Al inicio de este proyecto se plantearon los Objetivos generales y particulares; en esta sección se abordarán cada uno de ellos para mencionar los resultados y las conclusiones que se obtuvieron. El orden a seguir será de lo particular a lo general.

Este trabajo tenía como objetivo particular la construcción de 4 modelos predictivos (dos de clasificación y dos de regresión) que ayudaran a la detección del Síndrome Metabólico a través de la generación de predicciones para los valores de triglicéridos y colesterol HDL. Como parte del desarrollo se llevó a cabo la construcción de 4 modelos distintos, todos algoritmos basados en árboles, para cada uno de los casos que se analizaron, dando un total de 16 modelos construidos. Parte de la construcción de los modelos fue la identificación de los parámetros que ayudaran a maximizar el performance de los mismos (en inglés se conoce como hyperparameter optimization) y, para el caso de los modelos de clasificación, la identificación del umbral de probabilidad óptimo para hacer la clasificación, esto con la intención de mejorar la sensibilidad de los modelos, es decir, que se identificaran más casos positivos de manera correcta.

Los resultados obtenidos para los objetivos de particulares se presentan a continuación:

Modelos predictivos para detectar niveles bajos de colesterol HDL

Los random forest construidos para este primer caso fueron los modelos campeones. El random forest para la clasificación, después de identificar el umbral de probabilidad óptimo y usando los datos de validación, reportó la matriz de confusión de la Figura 5.1. Además, tuvo una precisión del 0.29 y una sensibilidad de 0.85, lo que indica que del total de los casos clasificados por el modelo como pacientes con colesterol HDL bajo únicamente el 29 % realmente tienen esta condición, esto parecería indicar que el modelo no está siendo tan preciso a la hora de detectar este patrón, sin embargo, ese 29 % de casos pronosticados correctamente como positivos representa el 85 % del total de casos positivos reales[1]. Por otro lado, la raíz cuadrada del error cuadrático medio reportada por el random forest para la regresión es de 7.98.

[1]El alto porcentaje de falsos positivos parecería indicar que no es un buen modelo, sin embargo, como se explicó en el Capítulo 4, los falsos positivos no representan un impacto muy grave para la práctica médica, lo más importante es minimizar los falsos negativos.

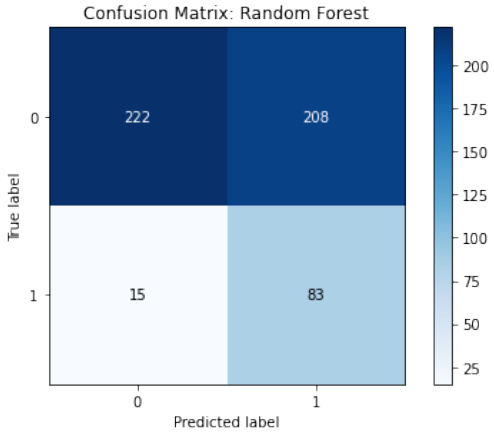


Figura 5.1: Matriz de confusión del modelo campeón para la detección de niveles bajos de colesterol HDL.

Modelos predictivos para detectar niveles altos de triglicéridos

Los random forest construidos para este segundo caso también fueron los modelos campeones. El random forest para la clasificación, después de identificar el umbral de probabilidad óptimo y usando los datos de validación, reportó la matriz de confusión de la Figura 5.2. Además, tuvo una precisión del 0.25 y una sensibilidad de 0.71, lo que indica que del total de los casos clasificados por el modelo como pacientes con triglicéridos altos únicamente el 25 % realmente tienen esta condición; ese 25 % de casos pronosticados correctamente como positivos representa el 71 % del total de casos positivos reales[2]. Por otro lado, la raíz cuadrada del error cuadrático medio reportada por el random forest para la regresión es de 37.00.

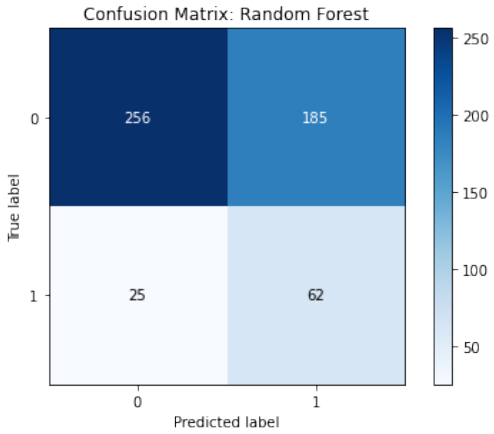


Figura 5.2: Matriz de confusión del modelo campeón para la detección de niveles altos de triglicéridos.

[2]El alto porcentaje de falsos positivos parecería indicar que no es un buen modelo, sin embargo, como se explicó en el Capítulo 4, los falsos positivos no representan un impacto muy grave para la práctica médica, lo más importante es minimizar los falsos negativos

Por último, el objetivo general de este trabajo es el uso de los modelos predictivos para ayudar a detectar el Síndrome Metabólico de tal manera que se optimice el costo económico asociado al diagnóstico médico. Para ello, como se muestra en el Capítulo 4, se utilizó un subconjunto de los datos sobre los cuales se midieron los factores de riesgo antropométricos y se hicieron los pronósticos para los valores de triglicéridos y colesterol HDL. El máximo ahorro económico se tiene cuando únicamente se utilizan las mediciones y los resultados de los modelos de clasificación para hacer el diagnóstico del Síndrome Metabólico. También se planteó un enfoque híbrido que utiliza los resultados obtenidos de la metodología propuesta para filtrar a aquellos pacientes que tienen un mayor riesgo de padecer el síndrome (aquellos que fueron diagnosticados como positivos) y, únicamente a ese grupo, hacerles la química sanguínea para confirmar el diagnóstico. El costo se puede ver en la Tabla 4.3. y el benificio en la Tabla 4.4.

Algunos aspectos que se tienen que considerar al analizar el desempeño de los modelos son:

- La cantidad de datos disponibles: con los datos disponibles para hacer el trabajo se tiene un muy buen punto de partida, pero conforme se vayan teniendo más pacientes (y su información) los modelos van a mejorar su desempeño.

- El tipo de datos disponible: para hacer las predicciones únicamente se cuenta con datos antropométricos (la mayoría provenientes de una báscula inteligente); para mejorar la capacidad predictiva de los modelos se pueden incorporar otros datos como, por ejemplo, datos de la dieta que tienen (cantidad de vegetales, carbohidratos, refrescos, etc.), información familiar (familiares con diabetes y/o hipertensión), por mencionar algunos.

- Población analizada: al tratarse de jóvenes se tiene la desventaja (para el modelo; es una ventaja para el paciente) que el organismo es joven también. Por ejemplo, un joven con obesidad puede no tener problemas de colesterol ni triglicéridos, ya que su organismo por ser joven está trabajado al 100 para mantenerse sano, mientras que un adulto con obesidad prácticamente es sinónimo de triglicéridos y colesterol alterados.

- Candidatos a este tipo de análisis: debido a que únicamente se entrenó con datos de jóvenes de entre 18 y 24 años no se tiene una muestra representativa de toda la población mexicana, por lo que los modelos resultantes (para que tengan un buen desempeño) no se deben de utilizar para analizar personas que no pertenezcan a este grupo demográfico.

En conclusión, se considera que se lograron los objetivos (general y particular) planteados y que, además, propuestas como éstas pueden ayudar a mejorar el estado de salud de los mexicanos y toman una mayor importancia si se toma en consideración el contexto actual de la pandemia por Covid-19, cuyas principales comorbilidades son la hipertensión, la obesidad y la diabetes[3], todas estas son enfermedades que se pueden detectar y mitigar de manera oportuna a través del diagnóstico del Síndrome Metabólico. Esta detección oportuna, además de los beneficios a la salud de las personas, implica también un ahorro muy importante para los gobiernos y las personas en cuanto a tratamientos y gastos hospitalarios.

[3]Acorde a los datos publicados en el tablero general CONACYT disponible en la página https://coronavirus.gob.mx/. Fecha de consulta el 25 de Octubre de 2020.

A | Código en Python

A.1. Preparación de Datos

En esta sección se muestra el proceso de adquisición de los datos (importados desde un archivo excel), la creación de las variables objetivo que se necesitan para generar los modelos y la preparación de los datos para generar el ABT necesario para la construcción de los modelos predictivos.

A.1.1. Librerías utilizadas

```
In [1]: import pandas as pd
        import numpy as np

        # Para poder visualizar todas las columnas:
        pd.set_option('display.max_columns',100)

        # Para ver unicamente 3 decimales:
        pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

A.1.2. Acceso a los datos

Se importan los datos desde un archivo .csv y se visualizan los primeros registros para corroborar que se importaron correctamente:

```
In [2]: data_inicial = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos.csv')
        print(data_inicial.head())
```

```
Out[2]:    ID_EVENTO SEXO  EDAD  SMet    PAD     PAS  GLU  HDL  TRI     CC   PESO  \
        0     623823    F    20     0 80.000 130.000   88   51   64 95.000 73.940
        1     623827    M    19     1 80.000 140.000   89   47  207 94.000 80.050
        2     623848    F    22     0 80.000 110.000   84   68   90 67.000 52.110
        3     623899    F    20     0 70.000 110.000   80   71   87 74.000 54.910
        4     623908    M    19     0 74.000 124.000  113   44  101 95.000 91.990

            TALLA    IMC  C_TOTAL  HOMA    INS  CC_AGUA  CC_GRAS  CC_HUES  CC_MUSC  \
        0 156.000 30.383       144 1.304  6.000   30.600   32.400    2.740    8.200
        1 166.500 28.876       161 3.428 15.600   39.700   26.100    3.550   10.700
        2 158.000 20.874       201 0.830  4.000   28.100   13.600    2.810    7.600
        3 157.000 22.277       183 0.751  3.800   32.000   11.800    2.810    8.300
        4 174.000 30.384       152 3.683 13.200   50.700   23.400    3.890   14.000

           P_AGUA  P_GRAS  P_HUES  P_MUSC     VFA
        0  41.385  43.819   3.706  11.090  92.300
        1  49.594  32.605   4.435  13.367 119.200
        2  53.924  26.099   5.392  14.585  56.200
        3  58.277  21.490   5.117  15.116  61.500
```

```
4  55.115  25.438   4.229  15.219  83.100
```

A.1.3. Filtrado de variables

Se seleccionan unicamente las medidas (variables) que no provienen de la química sanguínea, excepto las variables TRI y HDL, dado que la premisa es construir los modelos predictivos usando los valores de las medidas no invasivas.

```
In [3]: data = data_inicial[[x for x in data_inicial.columns if x not in
        ['C_TOTAL','HOMA','INS','GLU']]]
```

Se calcula la Presión Arterial Media (PAM) la cual ayuda a resumir la información de dos variables (PAD y PAS):

```
In [4]: data.insert(6,'PAM', (data.PAS + 2*data.PAD)/3)
```

A.1.4. Perfilamiento de los datos

La cantidad de pacientes que presentan síndrome metabólico (SMet=1) vs aquellos que no lo tienen (SMet=0):

```
In [5]: conteos = data['SMet'].value_counts()
        porcentaje = data['SMet'].value_counts(normalize=True).mul(100).round(2).astype(str) +
        '%'
        pd.DataFrame({'Conteo': conteos, 'Porcentaje': porcentaje})
```

```
Out[5]:    Conteo Porcentaje
        0    1550     87.82%
        1     215     12.18%
```

Se calculan los principales estadísticos a cada una de las variables analizadas:

```
In [6]: data.describe()
```

```
Out[6]:          ID_EVENTO     EDAD     SMet       PAD       PAS       PAM       HDL  \
        count     1765.000 1765.000 1765.000 1758.000 1758.000 1758.000 1765.000
        mean   1902077.852   18.999    0.122   71.776  106.935   83.496   48.965
        std     600394.926    1.438    0.327    8.729   11.740    8.853   10.310
        min     623823.000   18.000    0.000   40.000   64.000   48.000   17.000
        25%    1793376.000   18.000    0.000   66.000  100.000   77.333   42.000
        50%    2095692.000   18.000    0.000   70.000  110.000   83.333   48.000
        75%    2120448.000   19.000    0.000   80.000  112.000   90.000   55.000
        max    5642826.000   24.000    1.000  120.000  160.000  127.333  113.000

                    TRI        CC      PESO     TALLA       IMC  CC_AGUA  CC_GRAS  CC_HUES  \
        count  1765.000  1758.000  1765.000  1759.000  1759.000  1765.000  1765.000  1765.000
        mean    106.573    81.359    62.718   162.374    23.706    32.203    18.755     3.048
        std      56.874    11.191    13.330     8.261     4.235     7.021     8.684     0.625
        min      22.000    42.500    38.370   137.000    14.889    17.300     1.500     1.470
        25%      71.000    73.500    52.930   156.000    20.627    26.900    12.600     2.590
        50%      91.000    80.000    60.370   162.000    22.923    30.300    17.400     2.910
        75%     126.000    87.500    70.040   168.000    26.201    36.900    23.800     3.420
        max     698.000   126.000   127.900   188.000    43.415    67.800    65.300     7.060

                CC_MUSC    P_AGUA    P_GRAS    P_HUES    P_MUSC       VFA
        count  1765.000  1765.000  1765.000  1765.000  1765.000  1765.000
        mean      8.712    51.782    29.311     4.910    13.997    60.007
        std       1.952     7.220     9.768     0.640     1.991    34.360
        min       4.800    33.779     2.703     2.603     9.167     5.000
        25%       7.200    46.546    22.504     4.488    12.561    38.200
        50%       8.200    51.219    30.049     4.936    13.806    56.800
        75%      10.000    56.691    36.391     5.358    15.342    78.300
        max      19.100    74.403    53.931     9.431    20.650   479.600
```

Del procedimiento anterior se identifica que hay registros en los que se tienen valores nulos en alguna de las variables. A continuación, se filtrarán esos registros para ser analizados:

```
In [7]: nulos = data.isnull().values.any(axis=1)
        data_nulos = data[nulos]
        data_nulos
```

Out[7]:

	ID_EVENTO	SEXO	EDAD	SMet	PAD	PAS	PAM	HDL	TRI	CC
169	911364	F	18	0	nan	nan	nan	43	175	nan
216	1159105	F	19	0	nan	nan	nan	47	188	nan
509	1793514	F	23	0	nan	nan	nan	56	82	nan
631	1794686	M	18	0	nan	nan	nan	65	80	81.000
1520	2120717	F	18	0	nan	nan	nan	43	58	nan
1527	2120727	F	18	0	nan	nan	nan	54	57	nan
1529	2120729	F	18	0	nan	nan	nan	32	135	nan
1583	2120804	F	21	0	60.000	110.000	76.667	38	130	nan

	PESO	TALLA	IMC	CC_AGUA	CC_GRAS	CC_HUES	CC_MUSC	P_AGUA
169	80.710	nan	nan	31.300	37.900	2.910	8.600	38.781
216	51.950	nan	nan	24.800	18.000	2.350	6.800	47.738
509	73.790	nan	nan	29.000	34.100	2.990	7.700	39.301
631	67.130	171.000	22.957	39.800	12.600	3.730	11.000	59.288
1520	59.560	nan	nan	29.800	19.000	2.960	7.800	50.034
1527	48.170	nan	nan	26.500	12.000	2.670	7.000	55.013
1529	60.000	nan	nan	27.300	22.700	2.700	7.300	45.500
1583	59.470	158.000	23.822	29.100	19.800	2.770	7.800	48.932

	P_GRAS	P_HUES	P_MUSC	VFA
169	46.958	3.606	10.655	71.600
216	34.649	4.524	13.090	62.300
509	46.212	4.052	10.435	96.400
631	18.770	5.556	16.386	36.200
1520	31.901	4.970	13.096	53.900
1527	24.912	5.543	14.532	19.800
1529	37.833	4.500	12.167	70.800
1583	33.294	4.658	13.116	67.700

Se identifica que todos los pacientes en los que se tienen valores nulos no tienen síndrome metabólico y que este subconjunto de pacientes representa menos del 1 % de la población total analizada, por lo que eliminar aquellos registros con valores nulos no representa una pérdida significativa de información y de esta forma no se provoca un ningún sesgo (por más mínimo que sea) en el análisis al utilizar algún método de imputación de valores:

```
In [8]: data = data.dropna()
        conteos = data['SMet'].value_counts()
        porcentaje = data['SMet'].value_counts(normalize=True).mul(100).round(2).astype(str) +
        '%'
        pd.DataFrame({'Conteo': conteos, 'Porcentaje': porcentaje})
```

Out[8]:

	Conteo	Porcentaje
0	1542	87.76%
1	215	12.24%

Se calculan nuevamente los principales estadísticos a cada una de las variables consideradas:

```
In [9]: data.describe()
```

Out[9]:

	ID_EVENTO	EDAD	SMet	PAD	PAS	PAM	HDL
count	1757.000	1757.000	1757.000	1757.000	1757.000	1757.000	1757.000
mean	1902689.677	18.999	0.122	71.783	106.933	83.500	48.973
std	600932.976	1.437	0.328	8.727	11.743	8.854	10.311
min	623823.000	18.000	0.000	40.000	64.000	48.000	17.000

```
25%   1793376.000   18.000    0.000   66.000  100.000   77.333   42.000
50%   2095692.000   18.000    0.000   70.000  110.000   83.333   48.000
75%   2120446.000   19.000    0.000   80.000  112.000   90.000   55.000
max   5642826.000   24.000    1.000  120.000  160.000  127.333  113.000

            TRI       CC     PESO    TALLA      IMC  CC_AGUA  CC_GRAS  CC_HUES  \
count 1757.000 1757.000 1757.000 1757.000 1757.000 1757.000 1757.000 1757.000
mean   106.544   81.360   62.719  162.372   23.707   32.214   18.741    3.049
std     56.910   11.194   13.342    8.262    4.237    7.029    8.680    0.626
min     22.000   42.500   38.370  137.000   14.889   17.300    1.500    1.470
25%     71.000   73.500   52.930  156.000   20.624   26.900   12.600    2.590
50%     91.000   80.000   60.400  162.000   22.917   30.400   17.400    2.910
75%    126.000   87.500   70.040  168.000   26.203   36.900   23.800    3.420
max    698.000  126.000  127.900  188.000   43.415   67.800   65.300    7.060

       CC_MUSC   P_AGUA   P_GRAS   P_HUES   P_MUSC      VFA
count 1757.000 1757.000 1757.000 1757.000 1757.000 1757.000
mean     8.715   51.799   29.288    4.911   14.002   60.008
std      1.954    7.219    9.766    0.640    1.990   34.406
min      4.800   33.779    2.703    2.603    9.167    5.000
25%      7.200   46.562   22.492    4.488   12.564   38.200
50%      8.200   51.236   30.035    4.936   13.809   56.800
75%     10.000   56.708   36.353    5.358   15.345   78.300
max     19.100   74.403   53.931    9.431   20.650  479.600
```

A.1.5. Construcción de variables objetivo

Uno de los criterios para diagnosticar síndrome metabólico es que los triglicéridos estén por arriba de 150 mg/dL:

```
In [10]: data['TARGET_TRI'] = np.where(data['TRI']>150,1,0)
         conteos_tri = data['TARGET_TRI'].value_counts()
         porcentaje_tri =
         data['TARGET_TRI'].value_counts(normalize=True).mul(100).round(2).astype(str) + '%'
         pd.DataFrame({'Conteo': conteos_tri, 'Porcentaje': porcentaje_tri})

Out[10]:    Conteo Porcentaje
         0    1479     84.18%
         1     278     15.82%
```

Otro de los criterios para diagnosticar síndrome metabólico es que el colesterol HDL esté por debajo de 40 mg/dL en hombres y de 50 mg/dL en mujeres. Como no se cuenta con suficientes datos para hacer un modelo para cada segmento, se tomará como punto de corte para ambos sexos que el colesterol HDL esté por debajo de 40 mg/dL.

```
In [11]: data['TARGET_HDL'] = np.where(data['HDL']<40 ,1,0)
         conteos_hdl = data['TARGET_HDL'].value_counts()
         porcentaje_hdl =
         data['TARGET_HDL'].value_counts(normalize=True).mul(100).round(2).astype(str) + '%'
         pd.DataFrame({'Conteo': conteos_hdl, 'Porcentaje': porcentaje_hdl})

Out[11]:    Conteo Porcentaje
         0    1449     82.47%
         1     308     17.53%
```

Para finalizar, se guardarán los cambios hechos en esta sección en un archivo .csv para continuar con su análisis en las demás secciones.

```
In [12]: data.to_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_prep.csv',
         index=False)
```

A.2. Matriz de Confusión

A continuación, se muestra el código fuente de las funciones que fueron utilizadas para la construcción de las matrices de confusión presentadas en el desarrollo del trabajo.

A.2.1. Funciones para contruir la matriz de confusión

```
In [ ]: def heatmap(data, row_labels, col_labels, ax=None,
                    cbar_kw={}, cbarlabel="", **kwargs):
            """
            Create a heatmap from a numpy array and two lists of labels.

            Parameters
            ----------
            data
                A 2D numpy array of shape (N, M).
            row_labels
                A list or array of length N with the labels for the rows.
            col_labels
                A list or array of length M with the labels for the columns.
            ax
                A `matplotlib.axes.Axes` instance to which the heatmap is plotted.  If
                not provided, use current axes or create a new one.  Optional.
            cbar_kw
                A dictionary with arguments to `matplotlib.Figure.colorbar`.  Optional.
            cbarlabel
                The label for the colorbar.  Optional.
            **kwargs
                All other arguments are forwarded to `imshow`.
            """

            if not ax:
                ax = plt.gca()

            # Plot the heatmap
            im = ax.imshow(data, **kwargs)

            # Create colorbar
            cbar = ax.figure.colorbar(im, ax=ax, **cbar_kw)
            cbar.ax.set_ylabel(cbarlabel, rotation=-90, va="bottom")

            # We want to show all ticks...
            ax.set_xticks(np.arange(data.shape[1]))
            ax.set_yticks(np.arange(data.shape[0]))
            # ... and label them with the respective list entries.
            ax.set_xticklabels(col_labels)
            ax.set_yticklabels(row_labels)

            # Let the horizontal axes labeling appear on top.
            ax.tick_params(top=False, bottom=True,
                           labeltop=False, labelbottom=True)

            return im, cbar


        def annotate_heatmap(im, data=None, valfmt="{x:.2f}",
                             textcolors=["black", "white"],
                             threshold=None, **textkw):
            """
            A function to annotate a heatmap.

            Parameters
            ----------
            im
```

```
    The AxesImage to be labeled.
data
    Data used to annotate.  If None, the image's data is used.  Optional.
valfmt
    The format of the annotations inside the heatmap.  This should either
    use the string format method, e.g. "$ {x:.2f}", or be a
    `matplotlib.ticker.Formatter`.  Optional.
textcolors
    A list or array of two color specifications.  The first is used for
    values below a threshold, the second for those above.  Optional.
threshold
    Value in data units according to which the colors from textcolors are
    applied.  If None (the default) uses the middle of the colormap as
    separation.  Optional.
**kwargs
    All other arguments are forwarded to each call to `text` used to create
    the text labels.
"""

if not isinstance(data, (list, np.ndarray)):
    data = im.get_array()

# Normalize the threshold to the images color range.
if threshold is not None:
    threshold = im.norm(threshold)
else:
    threshold = im.norm(data.max())/2.

# Set default alignment to center, but allow it to be
# overwritten by textkw.
kw = dict(horizontalalignment="center",
          verticalalignment="center")
kw.update(textkw)

# Get the formatter in case a string is supplied
if isinstance(valfmt, str):
    valfmt = matplotlib.ticker.StrMethodFormatter(valfmt)

# Loop over the data and create a `Text` for each "pixel".
# Change the text's color depending on the data.
texts = []
for i in range(data.shape[0]):
    for j in range(data.shape[1]):
        kw.update(color=textcolors[int(im.norm(data[i, j]) > threshold)])
        text = im.axes.text(j, i, valfmt(data[i, j], None), **kw)
        texts.append(text)

return texts
```

A.3. Diagnóstico del Síndrome Metabólico

Para poder hacer el análisis del costo-beneficio, primero se tiene que hacer el diagnóstico del Síndrome Metabolico usando el resultado de los modelos predictivos contruidos.

A continuación, se analizan los 5 factores para determinar si los pacientes analizados tienen (o no) el síndrome.

A.3.1. Librerías utilizadas

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib
        import matplotlib.pyplot as plt

        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import precision_score, recall_score

        # Para poder visualizar todas las columnas:
        pd.set_option('display.max_columns',100)

        # Para ver unicamente 1 decimales:
        pd.set_option('display.float_format', lambda x: '%.1f' % x)
```

A.3.2. Datos utilizados

Se importan los datos con los resultados de la calificación de los modelos construidos en el capítulo anterior.

```
In [2]: data = pd.read_csv('C:/Users/gusta/OneDrive/Documentos/Tesis/Datos/Datos_roi.csv')
```

Se seleccionan las variables consideradas para diagnosticar el Síndrome Metabólico y las variables pronosticadas por los modelos para poder hacer el análisis de ROI.

```
In [3]: var = ['SEXO','CC','GLU','PAS','PAD','HDL','TRI','SMet','HDL_Class','TRI_Class']
        datos = data[var]
```

A.3.3. Diagnóstico

Se utilizan los parámetros establecidos en la Tabla 1.2 para complementar los resultados de los modelos.

```
In [4]: # Circunferencia de cintura alterado:
        datos['CC_Class'] = np.where(np.logical_or(np.logical_and(datos.SEXO == 'M', datos.CC >=90),
                                                   np.logical_and(datos.SEXO == 'F', datos.CC >=80)),
                                     1, 0)

        # Presión arterial alta:
        datos['Pres_Class'] = np.where(np.logical_and(datos.PAD >= 85, datos.PAS >= 130), 1, 0)

        # Glucosa alterada:
        datos['Glu_Class'] = np.where(datos.GLU >= 100, 1, 0)
```

Para determinar si el paciente tiene Síndrome Metabólico tienen que estar presentes 3 o más de los 5 factores analizados.

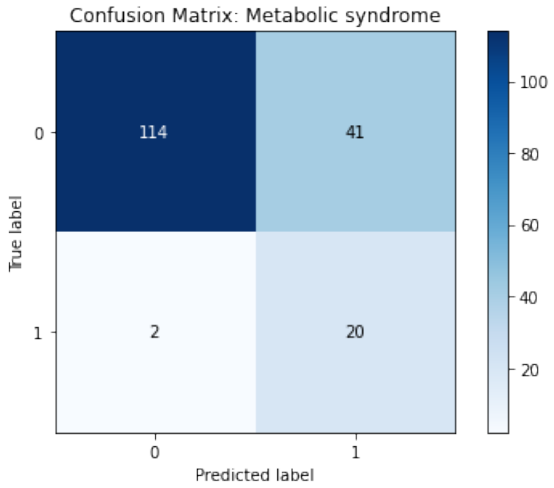
```
In [5]: datos['Sum_factores'] = datos['HDL_Class'] + datos['TRI_Class'] + datos['CC_Class'] +
                                datos['Pres_Class'] + datos['Glu_Class']

        datos['SMet_Class'] = np.where(datos.Sum_factores >= 3, 1, 0)
```

```
In [6]: datos
```

```
Out[6]:     SEXO   CC  GLU  PAS  PAD  HDL  TRI  SMet  HDL_Class  TRI_Class  CC_Class  \
        0      F 73.0   80  122   70   63   88     0          1          1         0
        1      M 83.0   69  100   80   37  185     0          1          1         0
        2      F 78.0   87   90   60   59   78     0          0          0         0
        3      M 71.0   97  100   80   36   91     0          0          0         0
        4      F 75.0   94   96   78   52   59     0          0          0         0
        ..   ...  ...  ...  ...  ...  ...  ...   ...        ...        ...       ...
        172    F 84.0   89   98   89   34  154     1          1          1         1
        173    M 93.0   89  112   70   36  246     1          1          1         1
        174    M 83.5   99  120   96   39  156     1          1          1         0
        175    F 84.1  102  110   70   49   90     1          1          0         1
        176    M 94.5   96  115   75   30  216     1          1          1         1

             Pres_Class  Glu_Class  Sum_factores  SMet_Class
        0             0          0             2           0
        1             0          0             2           0
        2             0          0             0           0
        3             0          0             0           0
        4             0          0             0           0
        ..          ...        ...           ...         ...
        172           0          0             3           1
        173           0          0             3           1
        174           0          0             2           0
        175           0          1             3           1
        176           0          0             3           1

        [177 rows x 15 columns]
```

Se construye la matriz de confusión.

```
In [8]: SMet = datos['SMet']
        SMet_Class = datos['SMet_Class']

        matriz = confusion_matrix(y_true=SMet, y_pred=SMet_Class)
        precision = precision_score(y_pred=SMet_Class, y_true=SMet)
        recall = recall_score(y_pred=SMet_Class, y_true=SMet)

        true_label = ['0','1']
        predicted_label = ['0','1']

        fig, ax = plt.subplots()

        im, cbar = heatmap(matriz, true_label, true_label, ax=ax,
                           cmap=plt.cm.Blues, cbarlabel="")
        texts = annotate_heatmap(im, valfmt="{x:.0f}")

        fig.tight_layout()
        plt.xlabel('Predicted label')
        plt.ylabel('True label')
        plt.title('Confusion Matrix: Metabolic syndrome')
        plt.show()

        print('Precisión = %0.2f' % (precision))
        print('Sensibilidad = %0.2f' % (recall))
```




```
Precisión = 0.33
Sensibilidad = 0.91
```

A.4. Repositorio en GitHub

Todos los códigos desarrollados en el presente trabajo se encuentran disponibles en un repositorio en GitHub y se puede acceder a ellos a través de la siguiente liga:

- https://github.com/GustavoMercado/MetabolicSyndrome_PredictiveModeling.

Referencias

[1] INEGI (INSTITUTO NACIONAL DE ESTADÍSTICA Y GEOGRAFÍA) (2015). *Mortalidad. ¿De qué mueren los mexicanos?*. Disponible en: http://cuentame.inegi.org.mx/poblacion/defunciones.aspx?tema=P. Fecha de consulta el 1 de agosto del 2020.

[2] WORACHARTCHEEWAN, A., SHOOMBUATONG, W., PIDETCHA, P., NOPNITHIPAT, W., PRACHAYASITTIKUL, V. y NANTASENAMAT, C. (2015). *Predicting Metabolic Syndrome Using the Random Forest Method*. Disponible en: https://www.hindawi.com/journals/tswj/2015/581501/. Fecha de consulta el 14 de agosto del 2021.

[3] KARIMI-ALAVIJEH, F., JALILI, S. y SADEGHI, M. (2016). *Predicting metabolic syndrome using decision tree and support vector machine methods.*. Disponible en: https://pubmed.ncbi.nlm.nih.gov/27752272/. Fecha de consulta el 14 de agosto del 2021.

[4] GUTIÉRREZ-ESPARZA, G.O., INFANTE VÁZQUEZ, O., VALLEJO, M. y HERNÁNDEZ-TORRUCO, J. (2020). *Predicting metabolic syndrome using decision tree and support vector machine methods.*. Disponible en: https://www.mdpi.com/2073-8994/12/4/581/htm. Fecha de consulta el 14 de agosto del 2021.

[5] ZIMMET, P., ALBERTI, K. G. y SERRANO, R. M. (2005). *Una nueva definición mundial del síndrome metabólico propuesta por la Federación Internacional de Diabetes: fundamento y resultados*. Revista Española de Cardiología 58(12): 1371-1376.

[6] FEDERACIÓN MEXICANA DE DIABETES (2016). *Síndrome Metabólico y su relación con la diabetes*. Disponible en: http://fmdiabetes.org/sindrome-metabolico-relacion-la-diabetes/. Fecha de consulta el 8 de agosto del 2021.

[7] WASSERMANN, A. y GROSSO, C. (2009). *Síndrome Metabólico. Definición*. Disponible en: http://www.fepreva.org/curso/4to_curso/bibliografia/volumen1/ut1_1_sindrome_metabolico_definicion_epidemiologia.pdf. Fecha de consulta el 1 de agosto del 2020.

[8] WHO (WORLD HEALTH ORGANIZATION) (2018). *Diabetes*. Disponible en: https://www.who.int/es/news-room/fact-sheets/detail/diabetes. Fecha de consulta el 1 de agosto del 2020.

[9] WHO (2017). *Enfermedades cardiovasculares*. Disponible en: https://www.who.int/es/news-room/fac-sheets/detail/cardiovascular-diseases-(cvds).Fecha de consulta el 1 de agosto del 2020.

[10] KREISBERG, R. y LEITER, L. (2011). *Hiperlipedemia*. Hormone Health Network. Disponible en: https://www.hormone.org/pacientes-y-cuidadores/hiperlipedemia. Fecha de consulta el 1 de agosto del 2020.

[11] FERNÁNDEZ, T. J. C. (2016). *Síndrome Metabólico y Riesgo Cardiovascular*. Revista CENIC. Ciencias Biológicas 47(2): 106-119.

[12] FERRI, F.F. (2019). *Ferri's Clinical Advisor. Metabolic Syndrome*. E-Book: 5 Books in 1. Edit. Elsevier-Health Sciences Division. Philadelphia, United States. 2136 pp.

[13] ALBERTI, K., ECKEL, R., GRUNDY, S., ZIMMET, P., CLEEMAN, J., DONATO, K., FRUCHART, J., JAMES, W., LORIA, C. y SMITH, S.J. (2009). *Harmonizing the Metabolic Syndrome*. Circulation 120(16): 1640-1645.

[14] WACHER, R.N. (2009). *Epidemiología del síndrome metabólico*. Gac Méd Méx 145(5): 384-391.

[15] TRUJILLO, H.B., TRUJILLO, M.E., TRUJILLO, M.M., BRIZUELA, A.C.A., GARCÍA, M.M.A., GONZÁLEZ, J.M.A., LÓPEZ, P.G.A., MINAKATA, N.J., RINCÓN, G.L.A., TINTOS, R.T., TORRES, V.R., VÁSQUEZ, C, y GUZMÁN, E.J. (2017). *Frecuencia del síndrome metabólico y factores de riesgo en adultos con y sin diabetes mellitus e hipertensión arterial*. Revista de Salud Pública 19(5): 609-616.

[16] CENTRO DE INVESTIGACIÓN EN EVALUACIÓN Y ENCUESTAS (2016). *Encuesta Nacional de Salud y Nutrición - MC 2016*. Disponible en: https://ensanut.insp.mx/encuestas/ensanut2016/index.php. Fecha de consulta el 2 de agosto del 2020.

[17] INEGI (2019). *Encuesta Nacional de Salud y Nutrición 2018. ENSANUT. Diseño conceptual*. Disponible en: https://ensanut.insp.mx/encuestas/ensanut2018/doctos/metodologia/ensanut_2018_diseno_conceptual.pdf. Fecha de consulta el 2 de agosto del 2020.

[18] INEGI (2019). *Encuesta Nacional de Salud y Nutrición 2018. ENSANUT. Presentación de resultados*. Disponible en: https://ensanut.insp.mx/encuestas/ensanut2018/doctos/informes/ensanut_2018_presentacion_resultados.pdf. Fecha de consulta el 2 de agosto del 2020.

[19] GARCÍA, E., DE LA LLATA, M., KAUFER, M., TUSIÉ, M.T., CALZADA, R., VÁZQUEZ, V., BARQUERA, S., CABALLERO, A.J. , OROZCO, L., VELÁSQUEZ, D., ROSAS, M., BARRIGUETE, A., ZACARÍAS, R. y SOTELO, J. (2008). *La obesidad y el síndrome metabólico como problema de salud pública: una reflexión*. Revista de Salud Pública de México 50(6): 530-547.

[20] INEGI (2019). *Principales causas de mortalidad por residencia habitual, grupos de edad y sexo del fallecido*. Disponible en: https://www.inegi.org.mx/sistemas/olap/registros/vitales/mortalidad/tabulados/pc.asp?t=14&c=11817. Fecha de consulta el 2 de agosto del 2020.

[21] GÓMEZ, P. F. J., ALMEDA, V. P. y GÓMEZ, S. M. A. (Sin fecha). *La Diabetes En México, Orígenes, Retos Y Soluciones*. Disponible en: https://www.asieslamedicina.org.mx/la-diabetes-en-mexico-origenes-retos-y-soluciones/?pdf=3126. Fecha de consulta el 2 de agosto del 2020.

[22] IMSS (INSTITUTO MEXICANO DEL SEGURO SOCIAL) (2017). *La Hipertensión Arterial de la población en México, una de las más altas del Mundo*. Disponible en: http://www.imss.gob.mx/prensa/archivo/201707/203. Fecha de consulta el 2 de agosto del 2020.

[23] SAS INSTITUTE (2019). *Analítica. Qué es y por qué es importante*. Disponible en: https://www.sas.com/es_mx/insights/analytics/what-is-analytics.html. Fecha de consulta el 3 de agosto del 2020.

[24] RAE (REAL ACADEMIA ESPAÑOLA) (2019). *Diccionario de la lengua española*. Disponible en: https://dle.rae.es/aprendizaje?m=form. Fecha de consulta el 3 de agosto del 2020.

[25] RAE (2019). *Diccionario de la lengua española*. Disponible en: https://dle.rae.es/aprender?m=form. Fecha de consulta el 3 de agosto del 2020.

[26] COPELAND, B.J. (2020). *Artificial Intelligence*. Disponible en: https://www.britannica.com/technology/artificial-intelligence. Fecha de consulta el 8 de agosto del 2021.

[27] INTERNET SOCIETY (2017). *Artificial Intelligence and Machine Learning: Policy Paper*. Disponible en: https://www.internetsociety.org/resources/doc/2017/artificial-intelligence-and-machine-learning-policy-paper/. Fecha de consulta el 3 de agosto del 2020.

[28] DEPARTMENT OF ENGINEERING: COMPUTER SCIENCE, STANFORD UNIVERSITY (s.f.). *Neural Networks History: The 1940's to the 1970's*. Disponible en: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html. Fecha de consulta el 3 de agosto del 2020.

[29] WIKIPEDIA (2021). *Zettabyte*. Disponible en: https://es.wikipedia.org/wiki/Zettabyte. Fecha de consulta el 17 de agosto del 2021.

[30] DATAROBOT (s.f.). *Unsupervised Machine Learning*. Disponible en: https://www.datarobot.com/wiki/unsupervised-machine-learning/. Fecha de consulta el 3 de agosto del 2020.

[31] IBM (2020). *Unsupervised Learning*. Disponible en: https://www.ibm.com/cloud/learn/unsupervised-learning. Fecha de consulta el 15 de agosto del 2021.

[32] BROWNLEE, J. (2016). *Supervised and Unsupervised Machine Learning Algorithms*. Disponible en: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/. Fecha de consulta el 4 de agosto del 2020.

[33] WILSON, A. (2019). *A Brief Introduction to Supervised Learning*. Disponible en: https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590. Fecha de consulta el 5 de agosto del 2020.

[34] YE, A. (2019). *5 Machine Learning Regression Algorithms You Need to Know*. Disponible en: https://towardsdatascience.com/5-regression-algorithms-you-need-to-know-theory-implementation-37993382122d. Fecha de consulta el 6 de agosto del 2020.

[35] SAS INSTITUTE (2017). *Which machine learning algorithm should I use?*. Disponible en: https://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/#prettyPhoto. Fecha de consulta el 7 de agosto del 2020.

[36] AMAT, J. (2017). *Árboles de predicción: random forest, gradient boosting y C5.0*. Disponible en: https://www.cienciadedatos.net/documentos/33_arboles_de_prediccion_bagging_random_forest_boosting#Introducción. Fecha de consulta el 8 de agosto del 2020.

[37] DATAIKU (2020). *Tree-Based Models: How They Work*. Disponible en: https://blog.dataiku.com/tree-based-models-how-they-work-in-plain-english. Fecha de consulta el 8 de agosto del 2020.

[38] ANALYTICS VIDHYA (2016). *Tree Based Algorithms: A Complete Tutorial from Scratch*. Disponible en: https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/#one. Fecha de consulta el 8 de agosto del 2020.

[39] KOTU, K. y DESHPANDE, B. (2019). *Ensemble Modeling*. Disponible en: https://www.sciencedirect.com/topics/computer-science/ensemble-modeling. Fecha de consulta el 8 de agosto del 2021.

[40] MITHRAKUMAR, M. (2019). *How to tune a Decision Tree?*. Disponible en: https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680. Fecha de consulta el 8 de agosto del 2020.

[41] SCIKIT LEARN (s.f.). *Decision Trees*. Disponible en: https://scikit-learn.org/stable/modules/tree.html#. Fecha de consulta el 8 de agosto del 2020.

[42] KDNUGGETS (2020). *Decision Tree Algorithm. Explained*. Disponible en: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html. Fecha de consulta el 15 de agosto del 2021.

[43] SCIKIT LEARN (s.f.). *Decision Trees. Mathematical formulation.*. Disponible en: https://scikit-learn.org/stable/modules/tree.html#mathematical-formulation. Fecha de consulta el 15 de agosto del 2021.

[44] SCIKIT LEARN (s.f.). *Decision Tree Classifier*. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. Fecha de consulta el 15 de agosto del 2020.

[45] BARRIOS, J. I. (2019). *Clases desbalanceadas en modelos de Machine Learning*. Disponible en: https://www.juanbarrios.com/clases-desbalanceadas/. Fecha de consulta el 22 de agosto del 2020.

[46] BROWNLEE, J. (2020). *Hyperparameter Optimization With Random Search and Grid Search*. Disponible en: https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/. Fecha de consulta el 15 de septiembre del 2020.

[47] SCIKIT LEARN (s.f.). *Cross-validation: evaluating estimator performance*. Disponible en: https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation. Fecha de consulta el 15 de septiembre del 2020.

[48] ELDER, J. (2018). *The Apparent Paradox of Complexity in Ensemble Modeling*. Disponible en: https://www.sciencedirect.com/topics/computer-science/ensemble-modeling. Fecha de consulta el 8 de agosto del 2021.

[49] SCIKIT LEARN (s.f.). *Ensamble methods. Bagging meta-estimator*. Disponible en: https://scikit-learn.org/stable/modules/ensemble.html#bagging-meta-estimator. Fecha de consulta el 15 de agosto del 2021.

[50] SCIKIT LEARN (s.f.). *Ensamble methods. Random Forests*. Disponible en: https://scikit-learn.org/stable/modules/ensemble.html#random-forests. Fecha de consulta el 15 de agosto del 2021.

[51] ML GLOSSARY (2019). *Gradient Descent*. Disponible en: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html?highlight=gradient. Fecha de consulta el 9 de agosto del 2020.

[52] SCIKIT LEARN (s.f.). *Ensamble methods. Gradient Tree Boosting*. Disponible en: https://scikit-learn.org/stable/modules/ensemble.html#gradient-tree-boosting. Fecha de consulta el 17 de agosto del 2021.

[53] SCIKIT LEARN (s.f.). *Ensamble methods. Gradient Tree Boosting. Mathematical formulation*. Disponible en: https://scikit-learn.org/stable/modules/ensemble.html#mathematical-formulation. Fecha de consulta el 17 de agosto del 2021.

[54] WIKIPEDIA (2020). *Teorema de Taylor*. Disponible en: https://es.wikipedia.org/wiki/Teorema_de_Taylor. Fecha de consulta el 17 de agosto del 2021.

[55] SCIKIT LEARN (s.f.). *Ensamble methods. Gradient Tree Boosting. Loss Functions*. Disponible en: https://scikit-learn.org/stable/modules/ensemble.html#loss-functions. Fecha de consulta el 17 de agosto del 2021.

[56] ANALYTICS VIDHYA (2019). *A Detailed Guide to 7 Loss Functions for Machine Learning Algorithms with Python Code*. Disponible en: https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/. Fecha de consulta el 17 de agosto del 2020.

[57] SCIKIT LEARN (s.f.). *Ensamble methods. Gradient Tree Boosting. Shrinkage via learning rate*. Disponible en: https://scikit-learn.org/stable/modules/ensemble.html#shrinkage-via-learning-rate. Fecha de consulta el 17 de agosto del 2021.

[58] BROWNLEE, J. (2020). *A Gentle Introduction to XGBoost for Applied Machine Learning*. Disponible en: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/. Fecha de consulta el 9 de agosto del 2020.

[59] BROWNLEE, J. (2020). *How to Configure XGBoost for Imbalanced Classification*. Disponible en: https://machinelearningmastery.com/xgboost-for-imbalanced-classification/. Fecha de consulta el 9 de agosto del 2020.

[60] RECUERO, P. (2018). *Machine Learning a tu alcance: La matriz de confusión*. Disponible en: https://empresas.blogthinkbig.com/ml-a-tu-alcance-matriz-confusion/. Fecha de consulta el 10 de agosto del 2020.

[61] ZELADA, C. (2017). *Evaluación de modelos de clasificación*. Disponible en: https://rpubs.com/chzelada/275494. Fecha de consulta el 10 de agosto del 2020.

[62] MARTINEZ, J. (Sin fecha). *Precision, Recall, F1, Accuracy en clasificación*. Disponible en: https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/. Fecha de consulta el 10 de agosto del 2020.

[63] BARRIOS, J. I. (2019). *La matriz de confusión y sus métricas*. Disponible en: https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/. Fecha de consulta el 10 de agosto del 2020.

[64] CZAKON, J. (2019). *F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose?*. Disponible en: https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc#2. Fecha de consulta el 10 de agosto del 2020.

[65] SCIKIT LEARN (s.f.). *Metrics and scoring: quantifying the quality of predictions*. Disponible en: https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-and-f-measures. Fecha de consulta el 10 de agosto del 2020.

[66] NCSS (s.f.). *One ROC Curve and Cutoff Analysis*. Disponible en: https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/One_ROC_Curve_and_Cutoff_Analysis.pdf. Fecha de consulta el 10 de agosto del 2020.

[67] SAITO, T. y REHMSMEIER, M. (2019). *Introduction to the ROC (Receiver Operating Characteristics) plot*. Disponible en: https://classeval.wordpress.com/introduction/introduction-to-the-roc-receiver-operating-characteristics-plot/. Fecha de consulta el 10 de agosto del 2020.

[68] MISHRA, D. (2019). *Regression: An Explanation of Regression Metrics And What Can Go Wrong*. Disponible en: https://towardsdatascience.com/regression-an-explanation-of-regression-metrics-and-what-can-go-wrong-a39a9793d914. Fecha de consulta el 11 de agosto del 2020.

[69] SWALIN, A. (2018). *Choosing the Right Metric for Evaluating Machine Learning Models*. Disponible en: https://medium.com/usf-msds/choosing-the-right-metric-for-machine-learning-models-part-1-a99d7d7414e4. Fecha de consulta el 11 de agosto del 2020.

[70] SCIKIT LEARN (s.f.). *Metrics and scoring: quantifying the quality of predictions*. Disponible en: https://scikit-learn.org/stable/modules/model_evaluation.html#r2-score. Fecha de consulta el 11 de agosto del 2020.

[71] ROCK CONTENT (2020). *¿Qué es un lenguaje de programación y qué tipos existen?*. Disponible en: https://rockcontent.com/es/blog/que-es-un-lenguaje-de-programacion/. Fecha de consulta el 5 de noviembre del 2020.

[72] GEEKS FOR GEEKS (2020). *Top 5 Programming Languages and their Libraries for Machine Learning in 2020*. Disponible en: https://www.geeksforgeeks.org/top-5-programming-languages-and-their-libraries-for-machine-learning-in-2020/. Fecha de consulta el 5 de noviembre del 2020.

[73] SCIKIT LEARN (s.f.). *Random Forest Classifier*. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Fecha de consulta el 15 de agosto del 2020.

[74] SCIKIT LEARN (s.f.). *Gradient Boosting Classifier*. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html. Fecha de consulta el 16 de agosto del 2020.

[75] XGBOOST (s.f.). *XGBoost Parameters*. Disponible en: https://xgboost.readthedocs.io/en/latest/parameter.html. Fecha de consulta el 16 de agosto del 2020.

[76] PYKES, K. (2020). *Oversampling and Undersampling. A technique for Imbalanced Classification*. Disponible en: https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf. Fecha de consulta el 21 de agosto del 2021.

[77] BROWNLEE, J. (2020). *SMOTE for Imbalanced Classification with Python*. Disponible en: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/. Fecha de consulta el 22 de agosto del 2020.

[78] SCIKIT LEARN (s.f.). *Decision Tree Regressor*. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html. Fecha de consulta el 29 de agosto del 2020.

[79] SCIKIT LEARN (s.f.). *Random Forest Regressor*. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. Fecha de consulta el 29 de agosto del 2020.

[80] SCIKIT LEARN (s.f.). *Gradient Boosting Regressor*. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html. Fecha de consulta el 30 de agosto del 2020.

[81] MAYO CLINIC (2019). *Síndrome Metabólico. Diagnóstico y tratamiento*. Disponible en: https://www.mayoclinic.org/es-es/diseases-conditions/metabolic-syndrome/diagnosis-treatment/drc-20351921. Fecha de consulta el 1 de agosto del 2020.