Modelo predictivo de AKI postoperatoria en cirugía no cardíaca (UDAKI)

Comprensión del Negocio

Nuestra propuesta se centra en desarrollar un modelo predictivo de AKI (Acute Kidney Injury) postoperatoria en pacientes programados para cirugía no cardíaca, utilizando técnicas de machine learning.

El objetivo principal es desarrollar una aplicación basada en el modelo generado con inteligencia artificial que, alimentado con datos de un paciente, genere como resultado su probabilidad de desarrollar lesión renal aguda postoperatoria (AKI).

Este modelo busca servir como una herramienta de apoyo en la toma de decisiones clínicas, permitiendo una identificación temprana de pacientes en riesgo y facilitando intervenciones preventivas.

Objetivos específicos

1. Desarrollar modelos de regresión logística, Random Forest, gradiente extremo (XG Boost) y Redes neuronales recurrentes para predecir la probabilidad de presentar AKI en los primeros 7 días del post operatorio.
2. Comparar el rendimiento de los modelos para predecir la aparición AKI post operatoria.
3. Construir una aplicación con una interfaz con el mejor modelo.

Comprensión de los Datos

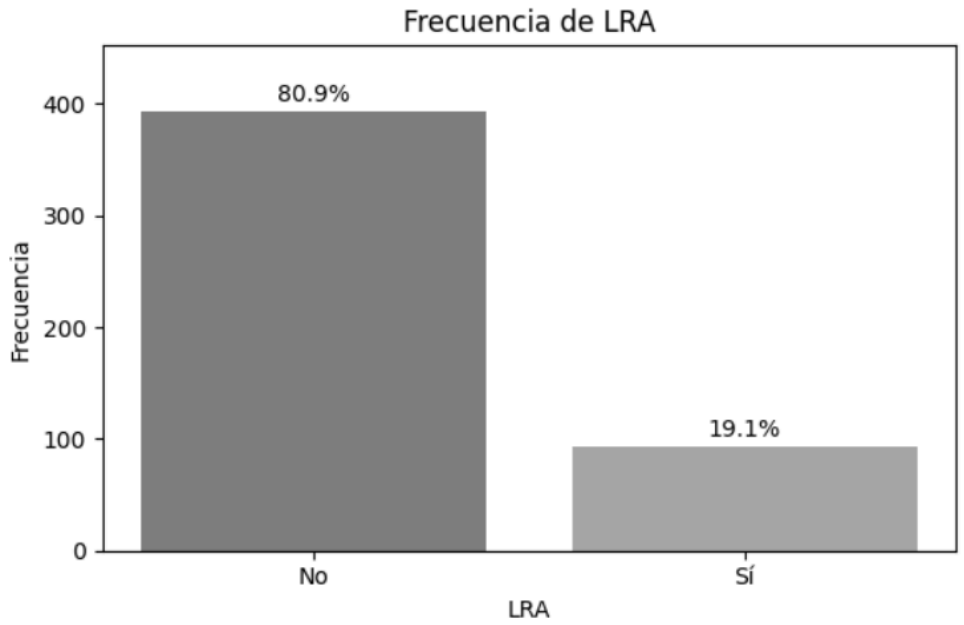
El conjunto de datos para el desarrollo de los modelos se obtiene de un estudio de cohorte retrospectivo realizado por estudiantes de la especialización de anestesiología y reanimación de la universidad de Antioquia , en proceso de publicación, en pacientes mayores de 18 años atendidos entre del 1 de enero de 2020 al 31 de diciembre de 2022 en los hospitales Alma Máter de Antioquia y Pablo Tobón Uribe de la ciudad de Medellín, Colombia, que fueron llevados a cirugía no cardiaca y que tenían una medición de creatinina sérica basal hasta 30 días antes del procedimiento y un control dentro de 48 horas hasta 7 días postquirúrgicos.

La información fue obtenida mediante la revisión de las historias clínicas por parte de dos investigadores, a partir de bases de datos aportadas por ambas instituciones hospitalarias. La información se registró en la herramienta Redcap, posteriormente se convirtieron y unieron en una base de datos de Excel.

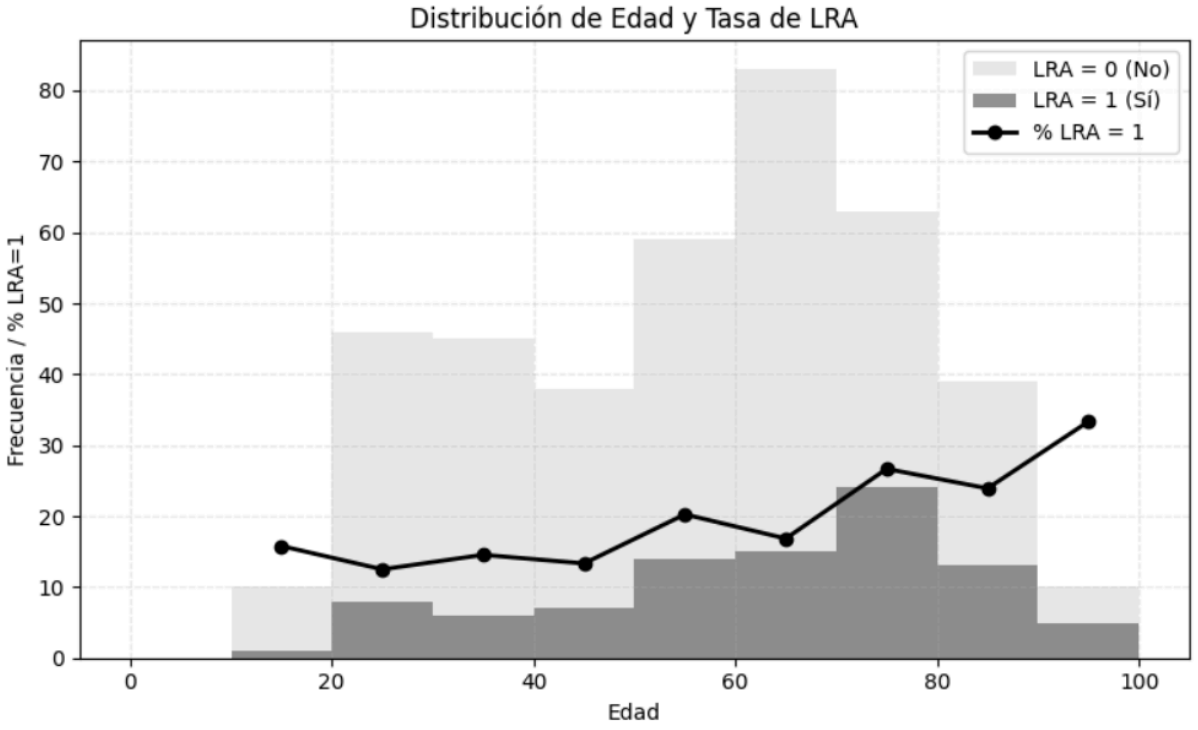
El conjunto de datos consta de 500 pacientes, las variables fueron: edad, sexo, índice de masa corporal, antecedentes patológicos como diabetes mellitus, hipertensión arterial, insuficiencia cardiaca crónica, neoplasia, infarto agudo al miocardio, cirrosis hepática, así como diagnósticos asociados como sepsis o choque séptico, rabdomiólisis, crisis hipertensiva, falla cardiaca descompensada además de uso de contraste endovenoso prequirúrgico, proteinuria preoperatoria, uso de antagonistas de los receptores de angiotensina II (ARA II) e inhibidores de la enzima convertidora de angiotensina (IECA), clasificación de american society of anesthesiologists (ASA), prioridad y tipo de la cirugía, técnica anestésica, ingreso a unidad de ciudadanos intensivos (UCI), sangrado intraoperatorio mayor o igual a 500 cc, transfusión de hemoderivados en el perioperatorio, uso de vasopresores en el intraoperatorio, lesión renal aguda post operatoria (LRA POP) según criterios KDIGO, estadios de la lesión renal estratificados en 1,2 y 3, días de estancia hospitalaria y muerte a los 28 días por cualquier causa o relacionada con LRA.
La media de la edad fue 57.4 años, de los cuales el 54% fueron hombres. Los antecedentes personales patológicos más comunes fueron hipertensión arterial (40%), seguido de neoplasia (maligna o del sistema nervioso central) (23%) y diabetes mellitus tipo 2 (15%).

	Count	Mean	STD	Min	25%	50%	75%	Max
Edad	500	57.364	20.217344	18	41	60	73.25	95
Talla corporal	480	163.9396	9.628078	130	157.75	165	170	192
Peso corporal	482	67.19917	14.796111	29	58	65	75	111
Creatinina POP	498	1.063715	0.94555	0.24	0.63	0.79	1.0875	9.23
Estadio 1 por creatinina	500	0.1	0.3003	0	0	0	0	1
Estadio 1 por GU	500	0	0	0	0	0	0	0
Estadio 2 por creatinina	500	0.038	0.191388	0	0	0	0	1
Estadio 2 por GU	500	0.002	0.044721	0	0	0	0	1
Estadio 3 por creatinina	500	0.03	0.170758	0	0	0	0	1
Estadio 3 por GU	500	0.028	0.165138	0	0	0	0	1
Estadio 3 por TRR	500	0.028	0.165138	0	0	0	0	1
LRA	500	0.194	0.395825	0	0	0	0	1
Estancia hospitalaria	499	20.78357	17.085459	1	9	15	28	131

La variable lesión renal aguda, es nuestra variable objetivo



tiene 94 casos positivos y casos positivos y 401 negativos, lo que de entrada plantea un conjunto de datos desbalanceado para la variable objetivo.



Al revisar las variables se encontraron los siguientes problemas:

- El conjunto de datos tenía las variables Record ID, Hospital y Número de historia clínica del paciente, que pudieran facilitar la identificación de los pacientes
- La variable sangrado mayor a 500 cc faltan 119 datos, la imputación de esta variable es difícil porque el sangrado durante una cirugía está relacionado a factores del paciente como al tipo de cirugía. No se puede imputar por la moda, ni por vecinos cercanos.
- En las variables estadios, Unchecked significa NO ocurrencia del evento y Checked significa sí. En la variable LRA POP (choice=No presentó), Checked significa que no presento AKI, Unchecked significa que, si presento AKI, esta forma de registrar el valor dificulta la interpretación
- La variable proteinuria preoperatoria tiene 322 datos indeterminados

```
# Eliminar las columnas Record ID, Hospital de practicas y Numero de historia clinica
# Estas columnas se eliminaron del archico de excel antes de subir el archivo para mantener la privacidad de los pacientes
# df = df.drop(['Record ID', 'Hospital de practicas', 'Numero de historia clinica'], axis=1, inplace=True)

```

UCI POP inmediato	1
Sangrado mayor 500cc	119
Transfusion hemoderivados	1
Uso vasopresores	1
Creatinina POP	2
dias de estancia hospitalaria	1

estadio 2 por creatinina	estadio 2 por GU	estadio 3 por creatinina	estadio 3 por GU	estadio 3 por TRR	LRA POP (choice=No presento)
Unchecked	Unchecked	Unchecked	Unchecked	Unchecked	Checked
Unchecked	Unchecked	Unchecked	Unchecked	Unchecked	Checked

proteinuria preoperatoria	
Indeterminado	386
No	53
Yes	37
Si	23

PROCESAMIENTO DE DATOS

Datos faltantes:

Variable	Datos faltantes
Talla corporal	20
Peso corporal	18
IAM	1
Cirrosis hepática	6
Falla cardíaca descompensada	2
Proteinuria preoperatoria	322
IECA	1
ARAII	1
Sangrado > 500 cc	119
Transfusión de hemoderivados	1
Uso de vasopresores	1
Creatinina pop	2
Estancia hospitalaria	1
Muerte a los 28 días	89
Variables categóricas	Pocos

Manejo de los datos faltantes

Muerte a los 28 días y relacionada con LRA: la variable mortalidad es una variable de resultado, dado que el objetivo del modelo es predecir la lesión renal aguda (LRA pop), y esta variable tiene gran número de datos faltantes, se decide eliminar.

```
df = df.drop(columns=['muerte por cualquier causa a 28 dias', 'muerte relacionada con LRA a 28 dias'])
```

La variable sangrado mayor a 500 cc faltan 119 datos, la imputación de esta variable es difícil porque el sangrado durante una cirugía está relacionado a factores del paciente como al tipo de cirugía; no se puede imputar por la moda, ni por vecinos cercanos. Adicionalmente la información pronóstica de esta variable está contenida en las variables transfusión y el uso de vasopresores, ya que los pacientes que se transfunden son los que sangran más de 500 cc durante la cirugía y están con soporte vasopresor, por esta razón se elimina la variable sangrada.

```
# Eliminar la variable sangrado mayor a 500 cc
df = df.drop(['Sangrado mayor 500cc'], axis=1)
```

Las variables talla corporal, peso corporal y Creatinina POP¨reemplazar los valores nulos por la mediana.

```
# En las variables talla corporal y peso corporal reemplazar los valores nulos por la mediana
df['talla corporal'] = df['talla corporal'].fillna(df['talla corporal'].median())
df['Peso corporal'] = df['Peso corporal'].fillna(df['Peso corporal'].median())
df['Creatinina POP'] = df['Creatinina POP'].fillna(df['Creatinina POP'].median())
```

La variable proteinuria preoperatoria con 322 registros indeterminados, se decide eliminar ya que tampoco es imputable fácilmente

La variable crisis hipertensiva solo tiene 3 registros positivos por lo que se elimina

```
# Eliminar la variable 'crisis hipertensiva'
df = df.drop('crisis hipertensiva ', axis=1)

```

La variable creatinina pop se utiliza para realizar el diagnóstico de lesión renal aguda pop, es decir su significado está ya dentro de esta variable, no aporta al modelo por lo que se elimina

```
# eliminar la variable 'Creatinina POP'
if 'Creatinina POP' in df.columns:
    df = df.drop('Creatinina POP', axis=1)

```

Las variables categóricas se eliminaron los valores nulos, ya que eran pocos registros

```
# eliminar los valores nulos en catCols
df.dropna(subset=catCols, inplace=True)

# Verify that null values have been removed
for col in catCols:
    print(col, np.sum(pd.isnull(df[col])))

```

El conjunto de datos final quedó con 39 columnas y 486 registros

```
df.shape
```

```
(486, 39)
```

Transformaciones aplicadas

En las variables estadios, el valor Unchecked significa NO ocurrencia del evento y Checked significa si, realizamos de una vez una dumificacion manual, cambiando Unchecked por 0 y Checked por 1.

.	estadio 1 por creatinina	estadio 1 por GU	estadio 2 por creatinina	estadio 2 por GU	estadio 3 por creatinina	estadio 3 por GU	estadio 3 por TRR
..	0.0	0.0	0.0	0.0	0.0	0.0	0.0
..	0.0	0.0	0.0	0.0	0.0	0.0	0.0

En la variable LRA POP (choice=No presentó), Checked significa que no presento AKI y Unchecked significa que, si presento AKI, se cambian los valores para mejorar la comprensión Nombre de la variable: LRA Checked por 0 y Unchecked Por 1

estadio 1 por creatinina	estadio 1 por GU	estadio 2 por creatinina	estadio 2 por GU	estadio 3 por creatinina	estadio 3 por GU	estadio 3 por TRR	LRA
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

El diagnóstico de AKI se realiza por criterios de aumento de la creatinina o por disminución del volumen urinario (gasto urinario). La severidad de la lesión renal aguda está evidenciada en los estadios 1,2 y 3. Se decide agrupar las variables estadios por creatinina y estadio por gasto urinario en una sola variable, ya que indican el mismo nivel de severidad

```
# se agrupan los estadios
import numpy as np
salida = salida.assign(
    aki_1=np.where((salida['estadio 1 por GU'] == 1) | (salida['estadio 1 por creatinina'] == 1), 1, 0),
    aki_2=np.where((salida['estadio 2 por GU'] == 1) | (salida['estadio 2 por creatinina'] == 1), 1, 0),
    aki_3=np.where(
        (salida['estadio 3 por creatinina'] == 1) | (salida['estadio 3 por GU'] == 1) | (salida['estadio 3 por TRR'] == 1),
        1,
        0,
    ),
)
```

LRA	dias de estancia hospitalaria	TRR hasta 7 dias POP	aki_1	aki_2	aki_3
0	22	No	0	0	0
0	60	No	0	0	0
1	18	No	0	1	0
0	8	No	0	0	0
0	16	No	0	0	0

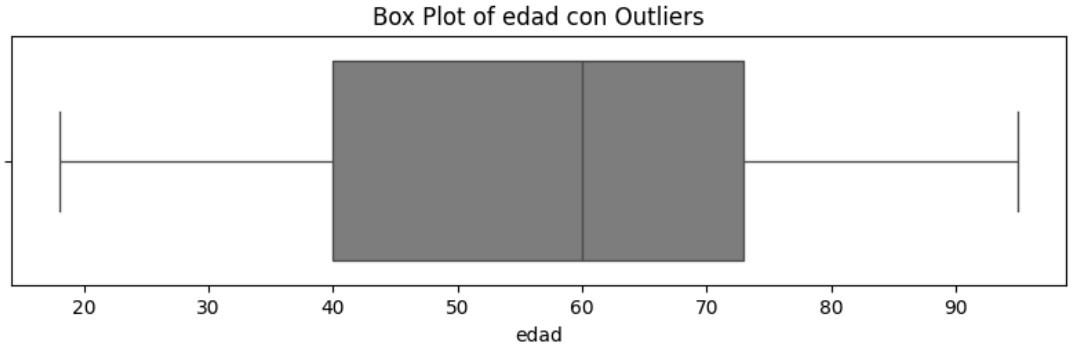
Valores atípicos

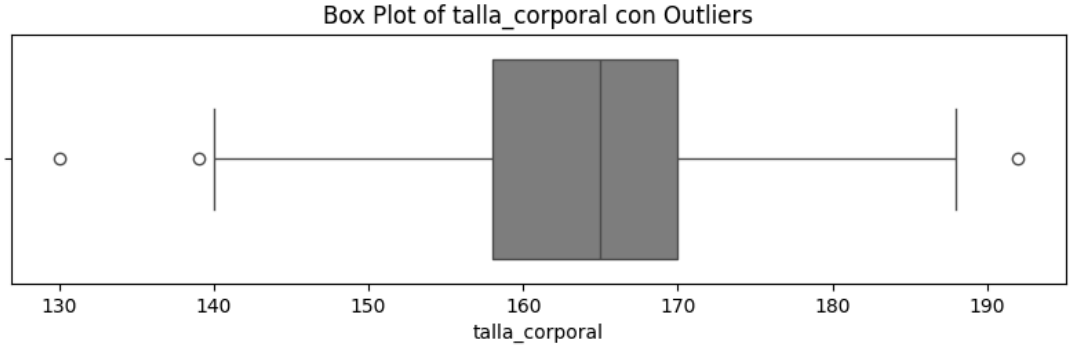
Se realizó por el método de vecinos cercanos. Al revisar los valores correspondientes en las variables evaluadas; edad, peso y talla, se encontró que eran valores posibles

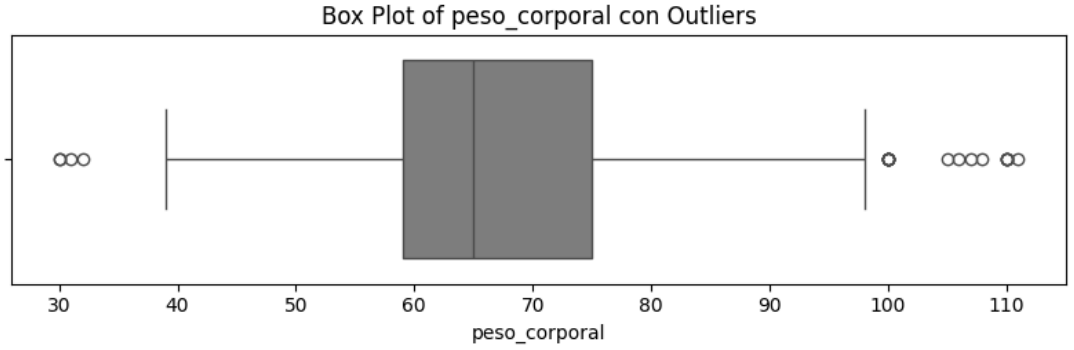
Ejemplo, un paciente puede tener 93 años de edad, puede parecer atípico, pero no es un dato incorrecto.

Similarmente un paciente con pérdida de peso severa, puede pesar 30 kilos, con referencia a la población si es un valor atípico, pero si es un valor posible para un paciente

	edad	sexo	talla_corporal	peso_corporal
72	93	Femenino	140	50
427	41	Masculino	185	83
249	80	Masculino	170	98
159	62	Femenino	150	80
241	89	Femenino	158	30
131	72	Masculino	180	84







-

Normalización

A las variables numéricas se les aplicó Min MaX scaler

```
scaler = MinMaxScaler()

# Fit and transform the numeric features
numeric_cols = ['edad', 'talla_corporal', 'peso_corporal']
entradas[numeric_cols] = scaler.fit_transform(entradas[numeric_cols])

# Display the scaled data
print(entradas.head(2))

```

	edad	talla_corporal	peso_corporal
0	0.974026	0.161290	0.246914
1	0.363636	0.467742	0.320988

2 rows × 42 columns

A las variables categóricas se les realizó Get Dummies

```
# Dumificar las variables categoricas
entradas=pd.get_dummies(entradas, drop_first=1)
entradas.info()
```

sexo_Masculino	imc_mayor_a_30_No	imc_mayor_a_30_Si	dm2_Yes	hta_Yes	_ic_cronica_Yes	iam_Yes	...	t
False	True	False	False	True	False	False	...	
False	True	False	False	False	False	False	...	

Para tratar de balancear los datos se utilizaron dos técnicas de sobremuestreo:

```
oversampler = RandomOverSampler()

# Se realiza el sobremuestreo para las variables de entrada y salida
X_resampled, y_resampled = oversampler.fit_resample(aki_final_train.drop('lra', axis=1), aki_final_train['lra'])

# Se crean dos dataframes con las variables de entrada y otro con la variable de salida a partir del proceso anterior
df1 = pd.DataFrame(X_resampled, columns=aki_final_train.columns.drop('lra'))
df2 = pd.DataFrame(y_resampled)

# Se concatenan los dataframes horizontalmente
df_balanced = pd.concat([df1, df2], axis=1)

df_aki=df_balanced
```

Esta técnica, aunque aumentó el número de datos no corrigió el desbalance:

```
print(df_balanced.shape)
print(aki_final_train.shape)
```

```
(704, 34)
(437, 34)
```

Se utilizó la técnica SMOTE, que se centra en la clase minoritaria

```
# Separar features (X) and target (y)
X = aki_final_train.drop('lra', axis=1)
y = aki_final_train['lra']

# Initialize LabelEncoder
encoder = LabelEncoder()

# Iterate through categorical columns and encode them
for col in X.select_dtypes(include=['object', 'category']).columns:
    X[col] = encoder.fit_transform(X[col])

# Aplicar SMOTE
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Verificar el balance
print("Distribución después de SMOTE:", y_resampled.value_counts())
```

El resultado fue mejor

```
Distribución después de SMOTE: lra
0    352
1    352
Name: count, dtype: int64
```

Este conjunto de datos con mejor balance, se guardó para posteriores análisis

```
# Guardar el conjunto de datos con sobremuesreo
# Combinar X_resampled y y_resampled en un nuevo DataFrame
balanced_data = pd.concat([X_resampled, y_resampled], axis=1)

# Guardar en un archivo CSV
balanced_data.to_csv('aki_final_train_balanced.csv', index=False)

print("Dataset balanceado guardado correctamente!")
```

Modelado

Teniendo como base la revisión de la literatura los modelos propuestos son:
Lineales: regresión Logística
No lineales: Random Forest, árbol de decisiones de aumento de gradiente (GBDT), aumento de gradiente extremo (XGBoost) y Red Neuronal Recurrente

Modelo	Descripción	Ventajas	Desventajas

Regresión Logística	Estima la probabilidad de un evento basándose en una o más entradas y se utiliza más comúnmente en problemas de clasificación	Cálculo de clasificación más rápido y con menor consumo de tiempo. Puntuaciones de probabilidad de la muestra de observación intuitivas. No se ve afectado por la multicolinealidad y se puede combinar con la regularización L2 para resolver el problema. Bajo coste computacional, fácil de entender e implementar.	El rendimiento computacional se degrada cuando el espacio de características es amplio. Fácil de subajuste, generalmente poco preciso. No maneja bien un gran número de características de clase. Se requiere conversión para características no lineales.
Random forest	método de aprendizaje integrado que contiene múltiples árboles de decisión para la clasificación	Los datos de alta dimensión (con muchas características) se pueden calcular sin reducción de dimensionalidad ni selección de características. Se puede evaluar la importancia de las características. Se puede evaluar la interacción entre diferentes características. No se sobreajustan fácilmente. El entrenamiento es más rápido y es fácil usar métodos paralelos. Se puede equilibrar el error de conjuntos de datos desequilibrados. Se puede mantener la precisión si falta una gran parte de las características.	Sobreajuste en algunos problemas de clasificación o regresión ruidosos. Para datos con atributos con diferentes valores, los atributos con más divisiones de valores tendrán un mayor impacto en el bosque aleatorio.
Gradient boosting decision tree	Los árboles potenciados utilizan modelos aditivos y algoritmos de avance gradual para implementar el proceso de optimización del aprendizaje	Se puede obtener una alta precisión con un tiempo de ajuste relativamente corto. Manejo flexible de diversos tipos de datos, incluyendo valores continuos y discretos, para una amplia gama de usos. Se pueden utilizar algunas funciones de pérdida robustas, que son más robustas a los valores atípicos.	Las dependencias entre aprendices débiles dificultan el entrenamiento de datos en paralelo

Extreme gradient boosting	El aumento de gradiente extremo (XGBoost) también es un tipo de algoritmo de integración, como un modelo de árbol reforzado, que consiste en una combinación de varios modelos de árbol para formar un clasificador muy robusto. Además, el modelo de árbol utilizado es el modelo de árbol de regresión CART	Es rápido y eficaz en el procesamiento de conjuntos de datos a gran escala y no requiere grandes cantidades de recursos de hardware, como memoria En comparación con los modelos de aprendizaje profundo, el efecto es similar sin necesidad de ajustar los parámetros. XGBoost implementa internamente un modelo de árbol potenciado, que puede gestionar automáticamente los valores faltantes	En comparación con el modelo de aprendizaje profundo, no puede modelar la ubicación espaciotemporal ni capturar datos de alta dimensión, como imágenes, voz y texto. El aprendizaje profundo es mucho más preciso que XGBoost cuando cuenta con una gran cantidad de datos de entrenamiento y puede encontrar un modelo de aprendizaje profundo adecuado
Artificial neural networks	Las redes neuronales artificiales son, en general, similares a redes compuestas por neuronas, que son unidades individuales conectadas entre sí, y cada unidad tiene una cantidad numérica de entradas y salidas, que pueden presentarse en forma de números reales o funciones combinatorias lineales.	Alta precisión de clasificación Alta capacidad de procesamiento de distribución paralela Almacenamiento distribuido y alta capacidad de aprendizaje Gran robustez y tolerancia a fallos ante nervios ruidosos Capacidad para aproximar completamente relaciones no lineales complejas con función de memoria asociativa	Las redes neuronales requieren una gran cantidad de parámetros, como la topología de la red, los valores iniciales de los pesos y los umbrales. La imposibilidad de observar el proceso de aprendizaje interno y la dificultad para interpretar los resultados pueden afectar la credibilidad y la aceptabilidad de los resultados. El tiempo de estudio es demasiado largo y puede que ni siquiera se logre el objetivo del aprendizaje.

Regresión logística

Se realizo una regresión logística para predecir las variables LRA, aki 1, aki 2 y aki 3. El propósito era predecir la ocurrencia de lesión renal aguda y determinar su severidad

```
# Se crean los datasets de entrenamiento y prueba para las variables de entrada y salida
import sklearn.model_selection as model_selection
from sklearn.model_selection import train_test_split

y_multi_target = salidas[['lra', 'aki_1', 'aki_2', 'aki_3']]

X_train, X_test, y_train, y_test = train_test_split(
                                       X,
                                       y_multi_target,  # Use the multi-target y
                                       train_size   = 0.8,
                                       random_state = 123,
                                       shuffle      = True
                                   )
```

El desempeño del modelo se evaluó con una matriz de confusión

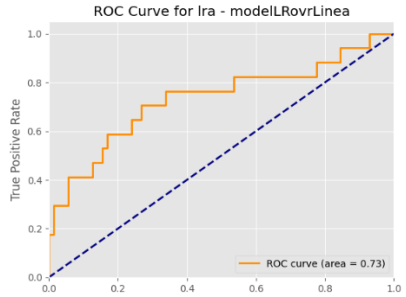
0 1

Predicted label

Classification Report for lra:

	precision	recall	f1-score	support
0	0.90	0.75	0.82	71
1	0.38	0.65	0.48	17
accuracy			0.73	88
macro avg	0.64	0.70	0.65	88
weighted avg	0.80	0.73	0.75	88

También se realizó una curva ROC



El desempeño del modelo no fue bueno para predecir la clase de interes

Las métricas fueron:

Accuracy (Exactitud): 0.74 (74% de predicciones correctas en total).

Clase 0 (Mayoritaria):

Precisión (0.90): Cuando el modelo predice "0", acierta el 90% de las veces.

Recall (0.76): Identifica correctamente el 76% de los casos reales de "0".

F1-Score (0.82): Combinación armónica de precisión y recall (buen equilibrio).

Clase 1 (Minoritaria):

Precisión (0.39): Solo el 39% de las predicciones de "1" son correctas (muchos falsos positivos).

Recall (0.65): Captura el 65% de los casos reales de "1" (pero a costa de baja precisión).

F1-Score (0.49): Bajo debido al trade-off entre precisión y recall.

Problema

Alto desbalance de clases: La clase "1" tiene solo 17 instancias vs. 71 de la clase "0".

Overfitting hacia la clase mayoritaria: El modelo prioriza aprender "0" porque es más frecuente.

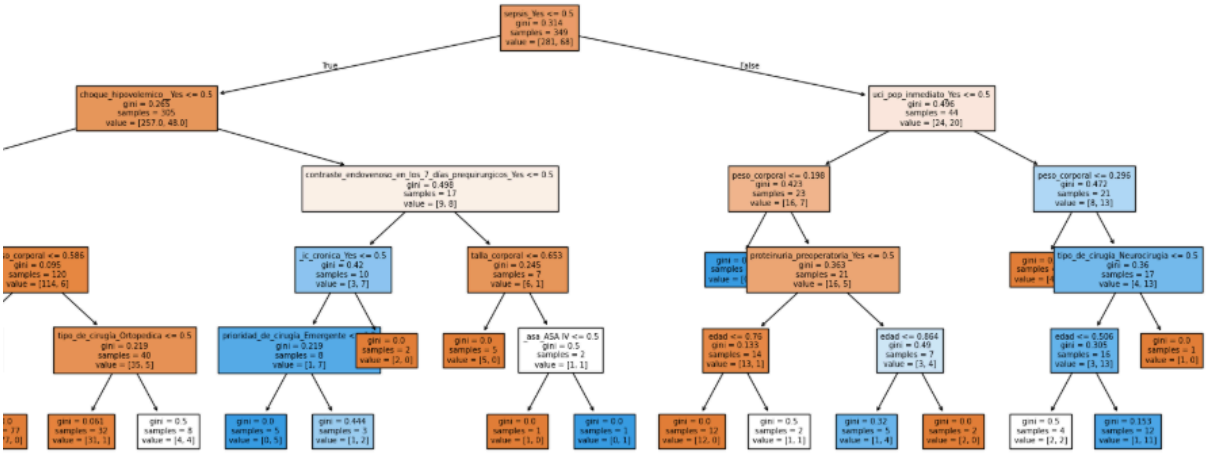
Baja precisión para la clase minoritaria: Predice muchos falsos positivos (ej: clasifica como "1" cuando en realidad es "0").

Arboles de decision

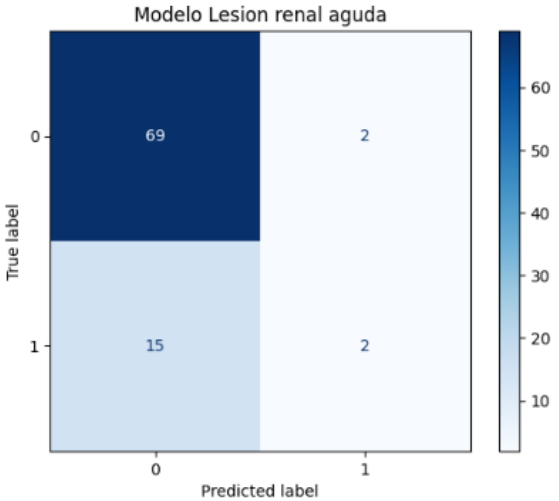
Se crearon modelos para cada una de las variables objetivos

```
# Creación del modelo
modelDtree_lra = DecisionTreeClassifier(
            max_depth         = 5,
            criterion         = 'gini',
            random_state      = 123
          )
modelDtree_aki1 = DecisionTreeClassifier(
            max_depth         = 5,
            criterion         = 'gini',
            random_state      = 123
          )
modelDtree_aki2 = DecisionTreeClassifier(
            max_depth         = 5,
            criterion         = 'gini',
            random_state      = 123
          )
modelDtree_aki3 = DecisionTreeClassifier(
            max_depth         = 5,
            criterion         = 'gini',
            random_state      = 123
          )

```



Se realizo matriz de confusión para determinar el desempeño de los modelos



Para la clase 0 (no LRA), el modelo tiene una precisión alta (0.82), mientras que para la clase 1 (LRA), la precisión es más baja (0.50)

Recall: mide la proporción de verdaderos positivos sobre el total de positivos reales. El modelo tiene un recall muy alto para la clase 0 (0.97), pero bastante bajo para la clase 1 (0.12).

F1-Score: es la media armónica de precisión y recall. Para la clase 0, el F1-Score es alto (0.89), pero para la clase 1 es bajo (0.19)

El modelo predice la clase 0 (no LRA) con alta precisión y recall, pero tiene dificultades para predecir la clase 1 (LRA) con precisión y recall bajo

Se realizaron optimación de hiperparámetros:

Podando el arbol

```python
# Post pruning (cost complexity pruning) por validación cruzada
# sebusca encontar el mejor valor para ccp_alpha
# El árbol se crece al máximo posible antes de aplicar el pruning
modelTreeC= DecisionTreeClassifier(
                            max_depth        = None,
                            min_samples_split = 2,
                            min_samples_leaf  = 1,
                            random_state      = 123
                        )

# Definir los parámetros para la búsqueda en la cuadrícula
parameters = {'ccp_alpha': np.logspace(-6, 2, 9)} # Valores entre 0.000001 y 100 en una escala logarítmica


# Definimos las metricas sobre las que queremos evaluar
metricas = 'f1'

# definicion de la variable con el número de pliegues
CV = 10

# Búsqueda por validación cruzada
grid_TreeC = GridSearchCV(
        estimator =  modelTreeC,
        param_grid = parameters,
        scoring    = metricas,
        cv         = CV,
        refit      = True,
        return_train_score = True
      )

grid_TreeC.fit(X, y_lra)
```

Pero no hubo mejoría en el desempeño:

Train accuracy = 100% vs Test accuracy = 32.6%: El modelo memorizó los datos de entrenamiento y no generaliza.

Árbol demasiado complejo (poda insuficiente con ccp_alpha=0.000001).

Rendimiento pésimo en test:

32.6% de accuracy es peor que un modelo aleatorio (50% en problema binario balanceado).

Alta desviación estándar (11.2%) → Inconsistencia entre folds.

Para finalizar se realizó un análisis de importancia de predictores:

Importancia de los predictores en el modelo

	predictor	importancia
0	edad	0.109836
1	talla_corporal	0.107219
2	peso_corporal	0.097387
14	sepsis_Yes	0.067686
17	choque_hipovolemico__Yes	0.052311
3	sexo_Masculino	0.049598
33	tipo_de_cirugía_Neurocirugia	0.046767
40	transfusion_hemoderivados_Yes	0.034757
10	erc_Yes	0.032719
28	prioridad_de_cirugía_Emergente	0.031985

Random Forest

Ventajas Random Forest

Reduce overfitting: Al promediar múltiples árboles, suaviza el ruido.

Más estable: Pequeños cambios en los datos no afectan significativamente el modelo.

Maneja mejores datos desbalanceados (con class_weight='balanced').

No requiere normalización de variables

Se creo un modelo Grid Search basado en out-of-bag score

El Out-of-Bag (OOB) Score es una métrica de evaluación interna que estima el rendimiento de un modelo de Random Forest sin necesidad de un conjunto de prueba separado.

Cada árbol del bosque se entrena con un subconjunto aleatorio de los datos de entrenamiento (muestreo con reemplazo).

Al hacer este muestreo, algunas muestras no son seleccionadas (aproximadamente 37% quedan fuera en cada árbol).

Estas muestras excluidas se llaman "Out-of-Bag" (OOB).

```python
# Grid de hiperparámetros evaluados lesion renal aguda
param_grid = ParameterGrid(
                {'n_estimators': [150, 200, 250],
                 'max_features': [5, 7, 9],
                 'max_depth'   : [None, 3, 10, 20],
                 'criterion'   : ['gini', 'entropy']
                }
            )

# Loop para ajustar un modelo con cada combinación de hiperparámetros
resultados = {'params': [], 'oob_accuracy': []}

for params in param_grid:

    modelo = RandomForestClassifier(
                oob_score    = True,
                n_jobs       = -1,
                random_state = 123,
                ** params
             )

    modelo.fit(X, y_lra)


    resultados['params'].append(params)
    resultados['oob_accuracy'].append(modelo.oob_score_)
    print(f"Modelo: {params} \u2713")

# Resultados
resultados = pd.DataFrame(resultados)
resultados = pd.concat([resultados, resultados['params'].apply(pd.Series)], axis=1)
resultados = resultados.sort_values('oob_accuracy', ascending=False)
resultados = resultados.drop(columns = 'params')
```

El resultado de los hiperparámetros:

OOB Accuracy: 81.2%

Hiperparámetros óptimos:

criterion='entropy'

max_features=9 (de ~39 features, solo 9 se consideran por árbol, limita overfitting).

n_estimators=250 (número alto de árboles mejora estabilidad).

max_depth=None (profundidad ilimitada, pero controlada por max_features).

Las métricas para la predicción de lesión renal aguda fueron:

```
           precision    recall  f1-score   support

        0       0.82      0.98      0.89       352
        1       0.56      0.12      0.19        85

  accuracy                           0.81       437
 macro avg       0.69      0.55      0.54       437
ighted avg       0.77      0.81      0.76       437
```

Clase 0 (No LRA)

Recall = 98%: El modelo identifica casi todos los casos negativos (especifidad alta).

Precisión = 82%: Cuando predice "No LRA", acierta el 82% de las veces.

Clase 1 (LRA)

Recall = 12%: Solo detecta 12 de cada 100 pacientes con LRA real.

Precisión = 56%: Cuando predice "LRA", solo acierta el 56% de las veces.

Accuracy engañoso (81%)

El alto accuracy se debe a que el modelo prioriza la clase mayoritaria (No LRA).

F1-Score de la clase 1 = 19%: Confirma que el modelo es inútil para predecir LRA.

Se realizo un Grid search con validación cruzada:

```
#Definicion del modelo
modelrf = RandomForestClassifier(random_state = 123)

# Grid de hiperparámetros evaluados

grid_param = {'n_estimators': [100, 120],
                          'max_features': [5, 7, 9, 11],
                          'max_depth'   : [3, 5, 10, 15, 20]
                          'criterion'   : ['gini', 'entropy']
                         }

# definicion de la variable con el número de pliegues
CV = 10

# las metricas sobre las se evaluará el modelo
scoring = 'f1'

# Búsqueda por grid search con validación cruzada
grid_rf = GridSearchCV(
                        estimator  = modelrf,
                        param_grid = grid_param,
                        scoring    = scoring,
                        cv = CV,
                        n_jobs     = - 1,
                        refit      = True,
                        verbose    = 4,
                        return_train_score = True
                       )

grid_rf.fit(X, y_lra)
```

Pero sin mejora en las métricas

XG Boost

Algoritmo de ensamble por boosting que combina múltiples modelos débiles (generalmente árboles de decisión) de forma secuencial, donde cada nuevo modelo corrige los errores del anterior.

XGBoost optimiza una función de pérdida (ej: para clasificación binaria, usa log-loss o entropía cruzada binaria) que penaliza las predicciones incorrectas.

```python
#Creación del modelo

#Armando un simple árbol de decisión
modelGrdB = GradientBoostingClassifier(n_estimators=2
                                       , max_features = 'sqrt'
                                       , subsample= 1 #Usa todo el dataset, no genera ningun muestreo. 1 es el valor por default
                                       , learning_rate = 0.01
                                       , random_state=123)


#Entrenamiento del modelo
modelGrdB.fit(X_train, y_train_lra)
```

```
GradientBoostingClassifier(learning_rate=0.01, max_features='sqrt',
                           n_estimators=2, random_state=123, subsample=1)
```

Hiperparametros para el desbalanceo

```python
from xgboost import XGBClassifier

# Calcular el ratio de desbalanceo
ratio = len(y_train_lra[y_train_lra == 0]) / len(y_train_lra[y_train_lra == 1])  # 352/85 = 4.14

# Modelo con peso ajustado
xgb = XGBClassifier(
    scale_pos_weight=ratio,  # Peso para la clase positiva (LRA)
    n_estimators=150,
    learning_rate=0.05,      # Tasa de aprendizaje baja para evitar overfitting
    max_depth=4,             # Profundidad controlada
    subsample=0.8,           # Muestreo aleatorio del 80% de datos por árbol
    colsample_bytree=0.8,    # Muestreo aleatorio del 80% de features por árbol
    eval_metric='logloss',   # Métrica para clasificación binaria
    random_state=42
)
xgb.fit(X_train, y_train_lra)
```

El resultado:

	precision	recall	f1-score	support
0	0.84	0.80	0.82	71
1	0.30	0.35	0.32	17
accuracy			0.72	88
macro avg	0.57	0.58	0.57	88
weighted avg	0.73	0.72	0.72	88

Precisión 0.30: Solo 30% de los predichos como LRA son verdaderos

Recall 0.35: Solo detecta 35% de los casos reales de LRA (65% no se diagnostican

Accuracy (72%): Engañoso, pues refleja el acierto en la clase mayoritaria (No LRA).

El modelo falla en el 65% de los pacientes con LRA

Optimización de los parámetros:

```python
# Obtener los resultados del grid search
results_grid_GrdB = pd.DataFrame(grid_GrdB.cv_results_)

# Seleccionar las columnas deseadas
columns_grid_GrdB = ['param_learning_rate'] + \
                    ['param_max_depth'] + \
                    ['param_max_features'] + \
                    ['param_n_estimators'] + \
                    ['param_subsample'] + \
                    ['mean_test_score', 'std_test_score']  + \
                    [f'split{i}_test_score' for i in range(CV)]

# Filtrar y mostrar los resultados
results_grid_GrdB_filtered = results_grid_GrdB[columns_grid_GrdB]

# Crear la columna scoreWithStd: f1 / std
results_grid_GrdB_filtered['scoreWithStd'] = results_grid_GrdB_filtered.apply(
    lambda row: row['mean_test_score'] / row['std_test_score'] if row['std_test_score'] != 0 else 0,
    axis=1
)

# Encuentra el índice del máximo valor en la columna scoreWithStd
indice_max_scoreWithStd = results_grid_GrdB_filtered['scoreWithStd'].idxmax()

# Mostrar los scores promedios por cada parámetro
results_grid_GrdB_filtered[['param_learning_rate','param_max_depth','param_max_features','param_n_estimators', 'mean_test_score', 'std_test_score', 'scoreWithStd']].head(5)
```

Las métricas:

	precision	recall	f1-score	support
0	0.82	1.00	0.90	71
1	1.00	0.06	0.11	17
accuracy			0.82	88
macro avg	0.91	0.53	0.50	88
weighted avg	0.85	0.82	0.75	88

La exactitud general es del 82%, el recall para la clase 1 (LRA) es extremadamente bajo (6%), lo que significa que el modelo está fallando peligrosamente en detectar pacientes con LRA.

Accuracy engañoso (82%): Se debe a que el modelo está simplemente detectando la clase mayoritaria

Recall: solo detecta el 6% de los casos reales de LRA (94% de falsos negativos)

Conclusión

Después de realizar la construcción y evaluación de los diferentes modelos para nosotros es claro que el desbalance de las clases en la variable objetivo está determinando que la predicción sea de la clase mayoritaria.

Debido a este desbalance, cuando se distribuye los datos entre los diferentes estadios de aki, quedan muy pocas instancias para poder hacer alguna predicción, por esta razón los próximos análisis se centrarán solo en la variable LRA (lesión renal aguda).

La solución al pobre desempeño de los modelos no está en la optimización de hiperparámetros como queda demostrado por las diferentes aproximaciones utilizadas; sino en utilizar técnicas que mejoren el balance de los datos.
Ya se cuenta con un conjunto de datos balanceado mediante la técnica SMOTE para el próximo análisis .