Tecnológico de Monterrey

Escuela de Ingeniería y Ciencias

Inteligencia Artificial Avanzada para la Ciencia de Datos

Aprendizaje Automatizado en el Titanic: Predicciones de Supervivencia con Clasificación Binaria

13/09/2023

Profesores:

Dr. Daniel Otero Fadul
Dr. Hugo Terashima Marín

—-

Equipo:

Pablo Gabriel Galeana Benítez - A01735281
Leonardo Ramírez Ramírez - A01351715
Miguel Chávez Silva - A01661109
Alessandro Arguelles Elias - A00829536
Luis Eduardo Aguilar Moreno - A01367545
Agustín Tapia - A01367639

Índice

Resumen

ABSTRACT

1. Introducción

La importancia y pertinencia de implementar modelos de Machine Learning (ML) en la resolución de problemas contemporáneos constituye el pilar fundamental en organizaciones y empresas arraigadas en la tecnología. El valor de estos modelos trasciende incluso la esfera tecnológica, emergiendo como una estrategia de alta valía en la predicción y clasificación de datos relevantes en campos diversos como medicina, metalurgia, aeroespacial y finanzas [2], [6]. Su influencia se extiende aún más, siendo un componente esencial dentro de líneas de investigación en genómica, astrofísica y biomedicina, por citar solo algunos ejemplos.

En términos generales, la esencia primordial para lograr resultados óptimos en los modelos de aprendizaje automático radica en la integridad de los datos: una base de datos depurada y rectificada es la base para asegurar pronósticos y clasificaciones precisas, de acuerdo con los objetivos establecidos. Una vez asegurada esta base sólida, los modelos pueden ser aplicados con éxito a diversas situaciones.
En cuestión de algunos modelos de ML ampliamente utilizados, los modelos de clasificación binaria resaltan por desplegar su potencia y capacidad para discernir patrones y relaciones entre los datos administrados. En el contexto de la extracción y validación de información basada en datos, por ejemplo, algunos modelos de clasificación binaria como *K-Nearest Neighbors, Support Vector Machine, Regresión Logística* y *Árboles de decisión* resaltan entre las técnicas más ampliamente utilizadas para el pronóstico de tiempo, detección de fraudes y spam, clasificación de imágenes, procesamiento sentimental, entre otros [7], [2].

Por ello, en el presente documento se desglosa la metodología seguida para la implementación de modelos de clasificación binaria en la predicción de supervivencia de tripulantes del Titanic pues, a pesar de existir un factor de suerte, presumiblemente existieron grupos con una mayor probabilidad de supervivencia que otros según ciertos atributos. La importancia del problema radica en la tarea de detectar variables significativas en la variable predictora: sobreviviente o no sobreviviente. A fin de generar un último modelo generalizable con predicciones acertadas.
A lo largo del documento se presenta la definición del problema 2, la metodología seguida para la limpieza y estructuración de la base de datos 3, el proceso estadístico para la elección de variables significativas 4, la aplicación de diferentes modelos de ML 5, el refinamiento del modelo final seleccionado 6 y resultados finales obtenidos

2. Definición del Problema

El hundimiento del Titanic fue un acontecimiento que impactó al mundo, incluso, hoy en día aún no finaliza las diferentes investigaciones que se tienen del suceso. Dentro de las investigaciones, mediante la implementación de ML, se quiere poder predecir la supervivencia de las personas con base en diferentes atributos que se tienen, por ejemplo: edad, título, si tenían hijos, si tenían pareja, entre otros. Se quiere poder realizar un modelo que pueda identificar si una persona fue subreviviente o no con base en las cualidades antes mencionadas. El objetivo es poder obtener el porcentaje más alto de precisión con la aplicación del modelo de clasificación binaria óptimo; es decir, encontrar las variables significativas y la mejor combinación de hiperparámetros según diferentes modelos para así, con base su rendimiento, obtener las predicciones más precisas.

3. Base de Datos

El *dataset* utilizado fue obtenido de Kaggle [5]. En el siguiente link: base de datos, se encuentra el *dataset* de tripulantes del Titanic, el cuál, como su nombre lo dice, es el conjunto de datos que se usa para poder entrenar a los diferentes modelos que se usaron, de esta forma una vez que el modelo está entrenada, se pueden introducir datos reales esperando una mayor precisión en el resultado.

El *dataset* está compuesto de 12 columnas, la primera es la columna es: **'PassengerId'** el cual es un valor único en cada registro, de esta forma se puede identificar a cada persona sin que hayan tripulantes repetidos; **'Survived'** es el resultado y está compuesto entre 1 y 0, donde 1 significa que sobrevivió y 0 que falleció; **'PClass'** el cuál es para indicar a cuál clase social pertenecía el tripulante (baja, media y alta); **'Name'** el cual viene compuesto por el nombre, el apellido y el título de la persona; **'Sexo'** si la persona era hombre y mujer; **'Age'** que indica la edad del tripulante; **'SibSp'** la cual es un número que indica si la persona tiene hermanos/hermanas o/y esposo/esposa; **'Parch'**, para los adultos indica con cuantos hijos viajaban, y para los niños indica si viajaba con uno o con ambos padres; **'Ticket'** que es el número de boleto que lo tocó al momento de comprarlo; **'Fare'** cuánto pagó por el boleto; **'Cabin'** que es el número de cabina que le fue asignada para el viaje; y finalmente **'Embarked'** que es para indicar en que puerto subió al Titanic.

Generalmente las bases de datos no están completas o suelen tener datos atípicos que si se entrena el modelo con esa base de datos sin limpiarla, podría afectar en el resultado final; es por eso que se analizaron los datos para posteriormente hacer una limpieza.

Primero se empezó por cambiar los nombres de algunos *atributos* para poder identificarlos e interpretarlos de manera más fácil, por ejemplo: el *atributo* **'SibSp'** se renombró como 'Siblings/Spouses'; **'Parch'** se le renombró como 'Parents/Children'; **'Pclass'** como 'Class'; **'Fare'** como 'Cost' y finalmente **'PassengerId'** como 'Id'. Además, a los valores de los *atributos* **'Cabin'** y **'Class'**, el tipo de valor se convirtió en *string* para poder facilitar el trabajo eventualmente. Después, se separó el atributo **'Name'**, como se mencionó anteriormente, el *atributo* viene compuesto por título,

nombre y apellidos, por sí sólo es infuncional pero al separar el *atributo* se podrían obtener *insights* interesantes a partir del título e incluso del apellido. Es por eso que se crearon 3 nuevos *atributos* a partir de este, uno nombrado 'Name' donde únicamente estaba el nombre, el siguiente fue 'Title' donde estaba el título de la persona y por último 'Last Name' el cual contiene los apellidos de las personas. Además, los nombres venían entre paréntesis, es por eso que también se creó una función que quitará los paréntesis para únicamente dejar los nombres.

Acto seguido, se acomodó el orden de los *atributos* a partir de pensamientos iniciales y la valuación que podría tener cada *atributo* para poder predecir el resultado. Inicialmente se dejaron *atributos* como título, nombre, edad, clase, si tenía pareja o hijos, y al final se pusieron *atributos* como el puerto en el que embarcó. A continuación, únicamente se obtuvo la sección de las cabinas, a partir de los datos solo se obtuvo la letra de las cabinas donde viajaban las personas; por ejemplo, si una persona viajaba en la cabina C134, dentro del *atributo* solo se almacenó "C" y para las personas que no tenían cabina, el valor se dejó en "-".

Después, en el *atributo* **'Class'** para poder facilitar el análisis de los datos, el valor se pasó a un valor ordinal; de esta forma, en lugar de tener un entero, se cambió por un *string* "First" para primera clase, "Second " para segunda y "Third" para la tercera clase. Una vez que se obtuvieron las variables ordinales, se estableció una jerarquía entre ellos a partir de la clase a la que pertenecían. También se jerarquizó a las personas de acuerdo a la cabina en la que estaban, siendo A de mayor jerarquía y "-" de menor jerarquía. De igual forma, para poder trabajar con variables que fueran más fáciles de interpretar, para el nombre de los puertos, en lugar de tener una sola letra, se cambió por el nombre completo del puerto.

Se observó que habían varios valores faltantes dentro del *atributo* **'Age'** por lo que se decidió llenarlos; para ello, a partir de la personas que sí tenían edad, se observó su título, para poder obtener un rango aproximado de edades a partir del título. Una vez obtenidas las edades, se calcularon variables estadísticas de tendencia central como la media, y medidas de dispersión como los cuartiles. Con el primer y tercer cuartil se pudo obtener un rango de edades de acuerdo al título. Una vez que se obtuvo el rango intercuartílico (IQR), se asignó aleatoriamente una edad dentro del rango para poder obtener datos más realistas asegurando el mismo equilibrio inicial en la distribución de edades, y que el modelo se pueda entrenar con información tal que la salida del modelo sea mejor.

Una vez hecho la asignación de edades, se decidió clasificarlas de acuerdo a su edad, para las personas menores a 18 años, fueron clasificadas como "Child", las personas entre 19 y 60 años como "Adult" y finalmente las personas mayores a 60 años como "Old". Por último, se quiso hacer una segunda división de clases de acuerdo al *atributo* de 'Class' y el precio que pagaron por el boleto. Básicamente, cada clase se dividió en 2 lo que finalmente nos dio 6 clases: Baja - Baja, Baja - Alta, Media - Baja, Media - Alta, Alta - Baja y Alta - Alta. Con esto se podrá comparar la clase de las personas incluyendo lo que pagaron por el boleto contra la supervivencia. Como se mencionó anteriormente, el tener una base de datos limpia, sin valores nulos ni atípicos, es el punto de partida la resolución de la problemática.

4. Elección de Variables

La elección de variables es un pilar fundamental previo a la implementación de cualquier tipo de modelo de ML. Este proceso debe de realizarse de manera meticulosa con el fin de no introducir atributos sin importancia destacable y como resultado terminen por sesgar los resultados. Para ello, se implementaron herramientas estadísticas fundamentales en la selección de variables que, a grandes rasgos, miden la influencia o correlación de cada una de estas con la variable a predecir.

4.1. ANOVA

El ANOVA se basa en comparar la variabilidad entre los grupos con la variabilidad dentro de los grupos. Si la variabilidad entre los grupos es suficientemente mayor que la variabilidad dentro de los grupos, se concluye que hay una diferencia significativa entre las medias de los grupos. Es por eso que se usó la ANOVA para poder identificar las variables que tienen repercusión en el resultado de si la persona sobrevivió o murió.

Feature	p-value
Age	0.029049
Cots	6.12E-15
Siblings/Spouses	0.292244
Parents/Children	0.014799

Tabla 1: ANOVA

La ANOVA se hizo con variables numéricas partiendo del supuesto es que al menos una de las variables a analizar sean categóricas; la variable categórica es 'Survived'. En la tabla 1 se pueden observar el p value que resulta de realizar una ANOVA a cada *feature*, se puede observar que 3 de 4 tiene un valor menor a 0.05, lo que significa que esas variables sí repercuten en la decisión de la supervivencia; mientras que 'Siblings/spouses' no.

4.2. Chi-cuadrada

Para poder realizar una prueba que pudiera determinar si las variables categóricas tuviera significancia dentro de la supervivencia, se usó una prueba Chi-cuadrada. La prueba Chi-cuadrada compara las frecuencias observadas de las categorías en la tabla de contingencia con las frecuencias que se esperarían si las dos variables fueran independientes. Se obtiene de:

$$\chi = \sum \left(\frac{(O - E)^2}{E} \right) \tag{1}$$

donde O son las frecuencias observadas y E son las frecuencias esperadas.

Variable	Chi2	P-value	Aceptación
Male	260.7170202	1.20E-58	Se acepta
Child	10.72343754	0.00105787	Se acepta
Adult	4.368773069	0.036603338	Se acepta
First Class	71.46583855	2.82E-17	Se acepta
Second Class	7.297192554	0.006906244	Se acepta
Cabin A	0.158031472	0.690975397	Se rechaza
Cabin B	25.72958003	3.93E-07	Se acepta
Cabin C	10.78338528	0.001024152	Se acepta
Cabin D	18.63149184	1.59E-05	Se acepta
Cabin E	17.24480374	3.29E-05	Se acepta
Cabin F	2.079597039	0.14927958	Se rechaza
Cabin G	0	1	Se rechaza
Clase Baja-Alta	1.796031508	0.180193008	Se rechaza
Clase Media-Baja	1.433576144	0.231181695	Se rechaza
Clase Media-Alta	0.657750272	0.417355307	Se rechaza
Clase Alta-Baja	18.63149184	1.59E-05	Se acepta
Clase Alta-Alta	48.37126495	3.53E-12	Se acepta
Southampton Port	20.87989839	4.89E-06	Se acepta
Cherbourg Port	24.34294029	8.06E-07	Se acepta
Queenstown Port	0	1	Se rechaza

Tabla 2: χ^2

En la tabla 2 se puede observar a las variables categóricas usadas para poder determinar si las variables eran significativas en el precio. El p-value es nuestro identificador, igual que la ANOVA, si el valor es menor que 0.05, se acepta la variable ya que sí tiene significancia dentro del precio. Los *features* que tienen "se rechaza" es que no son significativas dentro del precio.

5. Modelos de Machine Learning

Los modelos de ML, son una rama de la inteligencia artificial que utiliza algoritmos y técnicas estadísticas para permitir a las computadoras aprender y mejorar su rendimiento en tareas específicas a través de la experiencia y los datos [2]. Una de los aspectos más notables de los modelos de ML es su alta capacidad para extraer patrones aparentemente ocultos de grandes conjuntos de datos. Esto beneficia a la toma de decisiones ya que pueden ser decisiones más informadas; un claro ejemplo es como mediante diferentes modelos el diagnóstico de médicos ha mejorado, la detección de fraudes financieros, entre otros [2], [6], [7] El área de Machine Learning tiene infinidades de aplicaciones, por lo que la extracción de patrones tiene el potencial de transformar industrias, mejorar la atención médica,

optimizar la logística. Con base en el contexto y el alcance del problema, el objetivo es encontrar un modelo de ML de clasificación binaria cuyas predicciones puedan tener el mayor porcentaje de aciertos en cuestión de los sobrevivientes del Titanic.

Para la implementación de los modelos se siguió una metodología para poder potenciar el resultado obtenido.

1. Definición del problema: definiendo claramente el problema es posible abordar los objetivos a cumplir para poder y además analizar las posibles restricciones.

2. Recopilación de Datos: Se obtiene una base de datos que se ordena y limpia de los valores atípicos para el mejor funcionamiento del modelo.

3. Elección del modelo: Se eligen los modelos a implementar y entrenar para poder ajustar los parámetros.

4. Mediante otro conjunto de datos de validación se evalúa el rendimiento del modelo.

5. Por último, se ajustan los hiperparámetros del modelo para optimizar su rendimiento.

Una vez que se han obtenido ciertos resultados, es fundamental determinar cuál modelo se desempeña de manera óptima de acuerdo a los parámetros utilizados. Para lograr esta evaluación, existen diversas técnicas disponibles, entre las que destacan: la comparación de modelos, donde se consideran varios modelos y se analizan sus resultados y métricas de rendimiento en una comparativa exhaustiva; la validación cruzada, que implica la división de los datos en múltiples conjuntos de entrenamiento y prueba, calculando métricas promedio en cada iteración para una evaluación más robusta; y finalmente, las pruebas en datos de prueba independientes, que requieren un conjunto de datos de prueba separado para evaluar la capacidad del modelo en generalizar a datos no vistos. Estas técnicas son esenciales para la toma de decisiones informadas en el proceso de modelado de machine learning.

5.1. Modelos preliminares seleccionados

Los modelos de *Machine Learning* seleccionados para la resolución del problema fueron modelos de **aprendizaje supervisado** dentro de la categoría de modelos de clasificación binaria. Es importante definir los siguientes métricas:

Las métricas usadas fueron:

- **Accuracy**: mide la exactitud general del modelo en todas las clases. Sin embargo, puede no ser adecuado en conjuntos de datos desequilibrados. Es la proporción de predicciones correctas sobre el total de predicciones realizadas.

- **Target**: se refiere a la variable que se está tratando de predecir o clasificar en un modelo. Es la variable de salida o etiqueta.
- **Precision**: Es la proporción de verdaderos positivos respecto a todos los elementos que el modelo predijo como positivos.
- **F1-Score**: es una métrica que combina la precisión y el recall (sensibilidad) en un solo valor. Ayuda a equilibrar la compensación entre la precisión y la exhaustividad.
- **Recall**: es la proporción de verdaderos positivos respecto a todos los elementos reales que son positivos.
- **Matriz de Confusión**: es una tabla que muestra el desempeño del modelo al comparar las predicciones con los valores reales. Contiene cuatro valores: Verdaderos positivos (TP), Falsos positivos (FP), Verdaderos negativos (TN) y Falsos negativos (FN).

5.1.1. Arboles de decisión

Funcionamiento

Los árboles de decisión son una técnica de aprendizaje automático utilizada en problemas de clasificación y regresión. Funcionan de manera similar a un flujo de decisiones que se asemeja a un árbol invertido, donde cada nodo del árbol representa una pregunta o una prueba sobre una característica específica del conjunto de datos. Para lograr una clasificación, el modelo debe medir la impureza en los datos de cada nodo, para esto se utilizan funciones matemáticas, entropía e impureza de Gini. [4]

$$Entropy = -\sum_{i=0}^{n} p_i \cdot log(p_i)$$

Donde p_i es la frecuencia de la etiqueta en el nodo y n es el número de etiquetas únicas

$$Gini\ \ index = 1 - -\sum_{i=0}^{n} p_i^2$$

Hiperparámetros

Algunos de los hiperparámetros que pueden usarse para poder mejorar el modelo de Decision Tree, son:

```
{
'criterion': ['gini', 'entropy'],
'splitter': ['best', 'random'],
```

```
'max_depth': [None, 10, 20, 30],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'max_features': ['auto', 'sqrt', 'log2', None],
}
```

Los hiperparámetros se refieren a lo siguiente; Criterion: Criterio de selección de atributos, Splitter: Estrategia de partición, max_depth: Profundidad máxima del árbol, min_samples_split: Mínimo número de muestras para dividir un nodo, min_samples_leaf: Mínimo número de muestras en las hojas, max_features: Número máximo de características a considerar en cada división.

Dataset	Criterion			Deep				Precision
	Gini	Entropy	Log_Loss	5	6	7	8	
1		x				x		0.8379
			x			x		0.8379
	x				x			0.8379
2		x				x		0.8268
			x				x	0.8379
	x					x		0.8212
3		x		x				0.8491
			x	x				0.8435
	x			x				0.8491
4		x			x			0.8268
			x		x			0.8268
	x					x		0.8268

Tabla 3: Parámetros usados con su precisión correspondiente

5.1.2. Random Forest

Funcionamiento

El modelo de *Random Forest* es una técnica de conjunto (ensemble) utilizado para problemas de regresión y clasificación. El algoritmo se basa en la implementación de múltiples árboles de decisión para mejorar el rendimiento y robustez del modelo. Para cada árbol de decisión, Scikit-learn calcula la importancia de un nodo utilizando la Importancia de Gini, suponiendo sólo dos nodos hijos (árbol binario). [3]

$$ni_j = w_j + C_j - w_{left(j)}C_{left(j)} - w_{right(j)}C_{right(j)}$$

- ni_j= la importancia del nodo j
- w_j = número ponderado de muestras que llegan al nodo j

- C_j= el valor de impurez de j
- left_j = nodo hijo de la división izquierda en el nodo j
- right_j= nodo hijo de la división derecha del nodo j

La importancia de cada característica en un árbol de decisión se calcula como:

$$fi_i = \frac{\sum_{\text{jnodejsplitsonfeature}_i} ni_j}{\sum_{k \in \text{all nodes k}} ni_k}$$

- fi_i= la importancia de la característica i
- ni_j= la importancia del nodo j

La importancia final de la variable, a nivel del bosque aleatorio, es la media de todos los árboles. Se calcula la suma del valor de importancia de la característica en cada árbol y se divide por el número total de árboles:

$$RFfi_i = \frac{\sum_{j \in \text{all trees}} \text{norm} fi_{ij}}{T}$$

- REFfi= la importancia de la característica i calculada a partir de todos los árboles del modelo Random Forest
- normfi_{ij}= la importancia normalizada de la característica i en el árbol j
- T = número total de árboles

Hiperparámetros

Algunos de los hiperparámetros que pueden usarse para poder mejorar el modelo de *Random Forest*, son:

```
{
    'n_estimators': [100, 200, 300],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'bootstrap': [True, False],
```

```
    'random_state': [42]
}
```

Los hiperparámetros se refieren a lo siguiente; n_estimators: Número de árboles en el bosque, max_depth: Profundidad máxima de los árboles, max_depth: Profundidad máxima del árbol, min_samples_split: Mínimo número de muestras para dividir un nodo, min_samples_leaf: Mínimo número de muestras en las hojas, max_features: Número máximo de características a considerar en cada división.bootstrap: Si se realiza el muestreo con reemplazo (bootstrap) en la construcción de árboles, random_state: Semilla aleatoria para reproducibilidad

5.1.3. Regresión Lineal

Funcionamiento

Se usa para poder modelar la relación que hay entre una variable dependiente con una o más independientes. Inicialmente, se tiene que modelar la probabilidad de que la función sea 1, esto se puede hacer con la función sigmoide que es parecida a una función escalón pero la ventaja es que es continua y suave.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

Donde x resulta de: $x = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$. donde las Xs son las variables dependientes y las βs son los parámetros que se deben de ajustar para poder obtener un modelo más acertado. Entonces mediante la función sigmoide se puede decidir si la función es 1 o 0, ya que el resultado estará entre 1 y 0, por lo que se debe decidir un umbral en el cuál se va a 1.

Hiperparámetros

Algunos de los hiperparámetros que pueden usarse para poder mejorar el modelo, son:

```
{
'penalty': ['l1', 'l2', 'elasticnet', 'none'],
'C': np.logspace(-4, 4, 20),
'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
'max_iter': np.arange(100, 1000, 100),
'multi_class': ['auto', 'ovr', 'multinomial']
}
```

Los hiperparámetros que se muestran en la lista, *penalty* se refiere a la norma utilizada en la penalización; C se refiere a la inversa de la fuerza de regularización; *solver* se refiere a el algoritmo a utilizar en el problema de optimización; *mas_iter* es para dar el número máximo de iteraciones tomadas para que los *solvers* converjan en un punto; y por último, *multi_class* donde si la opción

elegida es 'ovr', entonces se ajusta un problema binario para cada etiqueta. Para 'multinomial' se minimiza la pérdida multinomial.

5.1.4. K Nearest Neighbors - KNN

Funcionamiento

El modelo *K Nearest Neighbor* es un método de clasificación y regresión especialmente útil cuando la relación entre los atributos o características de los datos no son lineales con el target o variable a predecir. El algoritmo k-*nearest neighbors* (k-NN) es un método de aprendizaje automático basado en instancias que se utiliza para la clasificación y la regresión.

Para poder determinar los vecinos más cercanos se debe seleccionar un origen, es decir, un punto a partir del cual se pueda definir qué es cercano. Y se calcula la distancia euclidiana en un espacio de 2 dimensiones, es decir, hay dos puntos $A(x, y)$ y $B(x, y)$, para poder determinar la distancia entre ambos, se hace mediante la fórmula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \tag{3}$$

Después se considera un parámetro k que es el número de vecinos más cercanos que se considerarán para hacer una predicción. Un valor de k pequeño, como 1 o 2, puede ser muy sensible al ruido en los datos, mientras que un valor de k muy grande puede suavizar demasiado el modelo.

Hiperparámetros

Algunos de los hiperparámetros que pueden usarse para poder mejorar el modelo, son:

```
{
'n_neighbors': np.arange(1, 50),
'weights': ['uniform', 'distance'],
'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
'p': [1, 2],
'metric': ['euclidean', 'manhattan', 'minkowski']
}
```

n_neighbors funciona para indicar el número de vecinos a considerar, es el k; *weigths* sirve para poder balancear las clases por si es que alguna influye más que otra dentro del modelo y no se desea que pase de esa forma; *algorithm* es el algoritmo utilizado para poder encontrar a los vecinos más cercanos; p que indica la potencia para la métrica de Minkowski y *metric* que situa el plano en el cuál se identifica la distancia entre los puntos.

5.2. Comparaciones

Las métricas de evaluación del rendimiento presentadas a continuación se realizaron utilizando diferentes sets de la base de datos original. Dicho procedimiento se implementó sin ninguna ayuda de funciones a fin de simular el funcionamiento de la validación cruzada por GridSearch.

Para encontrar la precisión se utilizó la función accuracy_score() de la librería de sk learn, para encontrar los valores de F1 score y Recall, se utilizó la función classification_report() y se tomaron los valores de weighted average. El término "weighted average"se refiere a la métrica promediada ponderada utilizada para calcular estadísticas de rendimiento como la precisión (accuracy), la recuperación (recall), la puntuación F1, y otras métricas en un problema de clasificación multiclase. Al comparar la precisión contra el recall existe un trade-off, a medida que aumentas la precisión (reduciendo los falsos positivos), generalmente el recall disminuye (aumentan los falsos negativos), y viceversa. Esto significa que, en general, mejorar una métrica puede empeorar la otra. En cuánto al F1 score, este valor busca encontrar un equilibrio entre la precisión y el recall. Es útil cuando es importante tener un buen rendimiento tanto en la identificación de ejemplos positivos como en la minimización de los falsos positivos. También es importante menciona la relación entre Accuracy contra F1 Score, en problemas de clasificación donde una clase es mucho más frecuente que la otra, el accuracy puede no indicar la veracidad del modelo, por ejemplo, un modelo que predice siempre la clase mayoritaria puede tener un alto accuracy pero un F1 score bajo.

Modelo Machine Learning	Precisión	f1-score	Recall
KNN	0.85	0.85	0.85
Decision Tree	0.8	0.8	0.8
Random Forest	0.85	0.85	0.85
Logistic Regression	0.84	0.84	0.84

Tabla 4: Pruebas en el data set 1 con splitter = best y class_weight = "balanced"

Modelo Machine Learning	Precisión	f1-score	Recall
KNN	0.83	0.83	0.82
Decision Tree	0.8	0.79	0.79
Random Forest	0.84	0.84	0.85
Logistic Regression	0.84	0.84	0.83

Tabla 5: Pruebas en el data set 2 con splitter = best y class_weight = "balanced"

Modelo Machine Learning	Precisión	f1-score	Recall
KNN	0.81	0.81	0.81
Decision Tree	0.81	0.8	0.81
Random Forest	0.86	0.83	0.85
Logistic Regression	0.85	0.84	0.85

Tabla 6: Pruebas en el data set 3 con splitter = best y class_weight = "balanced"

Modelo Machine Learning	Precisión	f1-score	Recall
KNN	0.87	0.87	0.86
Decision Tree	0.79	0.78	0.78
Random Forest	0.83	0.89	0.86
Logistic Regression	0.82	0.82	0.82

Tabla 7: Pruebas en el data set 4 con splitter = best y class_weight = "balanced"

En las tablas anteriores se muestran los resultados obtenidos para los diferentes sets.

6. Refinamiento

Seguido de la implementación de modelos, viene el refinamiento de cada uno de ellos. La idea primordial es lograr el mejor desempeño de cada uno de los modelo ajustando todo aquello que cambia el como aprenden. Para ello, es necesario definir estrategias que brinden un mejor entrenamiento o condiciones que modifiquen y amplíen la posibilidad de un mejor reconocimiento de patrones. En este caso, se utilizará un metodo conocido como *Random Search* para el refinamiento a través de la busqueda de los mejores configuraciones de cada modelo.

6.1. Random search

Random search es una técnica muy importante dentro del mundo de *Machine Learning*, ya que asiste en la definición de hiperparámetros de forma más eficiente. Los hiperparámetros son valores ajustables que definen el comportamiento del modelo y por lo tanto de su desempeño. Cada uno de los modelos tiene una cantidad determinada de hiperparametros y a la vez, cada uno puede tomar una cantidad diferente de valor o estructura. Idealmente, se busca encontrar la mejor combinación de estos valores, sin embargo, este proceso tiende a crecer muy rápido en cuanto a tiempo de computación ya que se forma un hiperespacio de combinaciones a probar. Con *Random Search* se puede encontrar la combinación óptima de valores para maximizar el rendimiento del modelo utilizando combinaciones aleatorias a través de un numero definido de iteraciones y solo se acepta la de mejor rendimiento.

El proceso de *Random Search* contiene los siguientes pasos:

1. Definición de hiperparámetros: se seleccionan los hiperparámetros que se desean ajustar y se especifican los rangos o valores posibles para cada uno.

2. Creación de la hiper-malla: se crea una hiper-malla que contiene todas las combinaciones posibles de valores de hiperparámetros.

3. Entrenamiento y evaluación: Se selecciona aleatoriamente una combinación de hiperparámetros en la hiper-malla, se entrena un modelo utilizando esos valores y se evalúa su rendimiento en un conjunto de validación o mediante validación cruzada. La métrica a optimizar puede ser definida en función del problema que se ataque.

4. Selección del mejor conjunto de Hiperparámetros: se identifica la combinación de hiperparámetros que proporciona el mejor rendimiento en función de la métrica elegida.

Se utilizó *Random Search* para refinar el modelo buscando la mejor combinación de hiperparametros sin gastar una cantidad exuberante de tiempo o de capacidad computacional en ello. Posteriormente se calculo el *accuracy* de los datos de entrenamiento y se hicieron predicciones con los datos de prueba para subir a Kaggle con el fin de también obtener el de estos.

Para el modelo de *Decision Tree* se encontraron los hiperparametros presentados en la tabla 8.

Hiperparámetro	Valor
min_samples_split	10
min_samples_leaf	4
max_depth	None
criterion	entropy
Accuracy entrenamiento	0.828
Accuracy prueba	0.772

Tabla 8: Hiperparámetros y *accuracy* de entrenamiento y prueba para *Desicion Tree*.

Por otro lado, para el modelo de *Gradient Boosting* se halló lo mostrado en la tabla 9

Parámetro	Valor
n_estimators	100
min_samples_split	5
min_samples_leaf	2
max_depth	3
learning_rate	0.2
Accuracy entrenamiento	0.845
Accuracy prueba	0.775

Tabla 9: Hiperparámetros y *accuracy* de entrenamiento y prueba para *Gradient Boosting*.

Se realizó el mismo proceso para el método *KN Neighbors*, los resultados obtenidos son los de la tabla 10

Parámetro	Valor
weights	distance
p	3
n_neighbors	14
Accuracy entrenamiento	0.846
Accuracy prueba	0.770

Tabla 10: Hiperparámetros y *accuracy* de entrenamiento y prueba para *KNN*.

Así mismo para *Logistic Regression*, cuyos resultados están en la tabla 11.

Parámetro	Valor
solver	newton-cg
penalty	l2
max_iter	1000
C	0.01
Accuracy entrenamiento	0.802
Accuracy prueba	0.789

Tabla 11: Hiperparámetros y *accuracy* de entrenamiento y prueba para *Logistic Regression*.

También se encontraron los mejores hiperparámetros y métrica de precisión para el modelo de *Random Forest* para la tabla 12.

Parámetro	Valor
criterion	gini
max_depth	20
min_samples_leaf	2
min_samples_split	10
n_estimators	50
Accuracy entrenamiento	0.834
Accuracy prueba	0.779

Tabla 12: Hiperparámetros y *accuracy* de entrenamiento y prueba para *Random Forest*.

Tabla 13 para el modelo *SVM*.

Parámetro	Valor
kernel	rbf
gamma	scale
C	10
Accuracy entrenamiento	0.846
Accuracy prueba	0.76

Tabla 13: Hiperparámetros y *accuracy* de entrenamiento y prueba para *SVM*.

Finalmente, también se utilizó para un modelo que recopila los resultados de los otros modelo más ligeros y se pone a votación para dar un resultado, un método de ensamble conocido como *Voting Classifier*.

Voting Classifier	
Accuracy entrenamiento	0.8338
Accuracy prueba	0.76
Parametros	
svm_params	{'kernel': 'rbf', 'gamma': 'scale', 'C': 1}
logistic_params	{'solver': 'saga', 'max_iter': 50, 'penalty': 'l1', 'C': 1}
tree_params	{'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 5}

Tabla 14: Hiperparámetros y *accuracy* de entrenamiento y prueba para *Voting Classifier*.

Resulta interesante que se tiene un comportamiento bastante similar en cada uno de los casos. Ni siquiera utilizando modelos más avanzados como lo son los de ensamble se lográ mejorar el *accuracy* de la prueba. Esto, de forma muy intuitiva apunta a una sola razón; existe un sobreajuste de los

modelos. Se puede definir que la complejidad no esta brindando un mejor desempeño a los modelos, por lo que la varianza comienza a subir y se nota en las tablas mostradas.

7. Resultados

Después de llevar a cabo el procedimiento correspondiente para encontrar un modelo capaz de realizar una clasificación binaria para determinar si una persona sobrevivió al accidente del Titanic, se han obtenido diversos resultados en cuanto a la exactitud en el conjunto de datos de evaluación. Estos resultados varían según el modelo utilizado y las variables seleccionadas.

Se observa que las predicciones de clasificación para el conjunto de validación durante el entrenamiento presentan una media superior al 0.80 para los distintos modelos, lo cual puede considerarse adecuado en este tipo de tareas. Esto indica que aproximadamente 8 de cada 10 personas se clasifican correctamente en sus respectivas categorías. Sin embargo, la métrica de exactitud para el subconjunto de datos de evaluación no alcanzó el nivel esperado. En nuestros resultados, la media de exactitud para todos los modelos es de 0.75. Este valor es inferior al obtenido en los datos de validación, lo que sugiere que los modelos están sobreajustados a los datos de entrenamiento. Este sobreajuste se puede confirmar al comparar la exactitud en la validación con la obtenida en la plataforma Kaggle.

Este fenómeno de sobreajuste indica que puede haber una mala generalización debido a la complejidad de los modelos en una tarea de clasificación binaria como la propuesta en este desafío. De todos los modelos creados, tanto los refinados como aquellos implementados sin la búsqueda de ajustar hiperparámetros, el modelo que sobresale por su alta exactitud en el subconjunto de datos de evaluación en Kaggle es un modelo de regresión logística con hiperparámetros preestablecidos por la función de la librería scikit-learn. Este modelo logró una exactitud de 0.78, que se encuentra mucho más cercana a la media de exactitudes en el conjunto de validación.

Si comparamos esta métrica con la predicción realizada por Kaggle utilizando *gender_submission.csv*, donde se establece un criterio de supervivencia basado únicamente en el género de la persona a predecir (que obtuvo una exactitud de 0.76), se aprecia una mejora en el rendimiento del modelo. Sin embargo, dado el nivel de complejidad del modelo y el trabajo de limpieza, análisis y refinamiento realizado, se esperaba una diferencia aún mayor en la exactitud obtenida.

8. Conclusiones

En conclusión, este trabajo ha destacado la relevancia de la implementación de modelos de Machine Learning en la resolución de problemas de clasificación binaria, donde la importancia de la integridad de los datos se revela como un componente esencial para lograr resultados óptimos en estos, ya que una base de datos depurada y rectificada sienta las bases para pronósticos y clasificaciones precisas.

A lo largo del trabajo, se ha delineado una metodología que abarca desde la definición del problema, hasta la obtención de resultados finales. Se ha destacado la importancia de la aplicación de métodos de limpieza y estructuración de la base de datos, así como la selección de variables relevantes mediante métodos estadísticos como el análisis ANOVA y la prueba χ^2.

Especialmente, se examinó la aplicación de varios modelos de clasificación binaria, como *K-Nearest Neighbors*, *Support Vector Machine*, *Logistic Regression* y *Decision Tree*, en la predicción de la supervivencia de los pasajeros del Titanic. A pesar de la aleatoriedad inherente en este tipo de eventos, se ha demostrado que ciertos atributos pueden influir en la probabilidad de supervivencia. Por ello, el hecho de detectar estas variables significativas se convierte en una tarea fundamental para generar un modelo generalizable con pronósticos acertados.

Finalmente, se encontró que el modelo de *Logistic Regression* sin el refinamiento de hiperparametros contó con un mejor desempeño en comparación con los demás tras la implementación del método de refinación. Lo anterior, se logra destacar de la mayor parte de lo encontrado durante el desarrollo de este proyecto. Los modelos más simples tuvieron resultados mejores en comparación de los complejos, indicando la sensibilidad de los datos a un posible sobreajuste tal como se menciono anteriormente.

Referencias

[1] Brain, G. (2022). Tensorflow: An open source machine learning framework for everyone. `https://www.tensorflow.org/`. Accessed: yyyy-mm-dd.

[2] Brunton, S. L. and Kutz, J. N. (2019). *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press.

[3] del artículo, A. (Año de publicacióna). The mathematics of decision trees, random forest, and feature importance in scikit-learn and spark. *Towards Data Science*.

[4] del artículo, A. (Año de publicaciónb). Understanding the mathematics behind the decision tree algorithm - part i.

[5] Kaggle (2023). Titanic: Machine learning from disaster. `https://www.kaggle.com/competitions/titanic`. Accessed: 2023.

[6] Kotsiantis, S. B., Zaharakis, I., Pintelas, P., et al. (2007). Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24.

[7] Kumari, R. and Srivastava, S. K. (2017). Machine learning: A review on binary classification. *International Journal of Computer Applications*, 160(7).

[8] Scikit-Learn Contributors (Año de acceso). Scikit-learn grid search documentation.