Instituto Tecnológico y de Estudios Superiores de Monterrey

Inteligencia Artificial Avanzada para la Ciencia de Datos (Gpo 101)

Reporte final: Titanic

Equipo 5

Jorge Eduardo De León Reyna - A00829759
David Esquer Ramos - A01114940
Francisco Mestizo Hernández - A01731549
Adrián Emmanuel Faz Mercado - A01570770

Septiembre 13, 2023

1. Introducción

Una vez que se finalizó con la limpieza de los datos, es posible comenzar con la decisión del modelo que se utilizará. El problema que se presenta es generar un modelo que permita predecir si un tripulante del Titanic sobrevivió o no en base a diferentes características, tales como el sexo, edad, la tarifa que pagó, la clase a la que pertenece, si cuenta con familiares dentro del barco, entre otros. Se trata de un problema de **clasificación binaria,** pues se busca generar un modelo que permita clasificar entre 2 posibles salidas, si el tripulante sobrevivió o si no lo hizo.

Los algoritmos de clasificación están especializados en aprender patrones en sets de datos que se encuentran previamente categorizados. Una vez que el modelo está entrenado y detecta estos patrones, se le pueden dar nuevos sets de datos que no estás categorizados y el modelo predecirá a qué categoría pertenecen.

Por esto, es posible utilizar un algoritmo de clasificación para predecir si un pasajero del Titanic sobrevivió o no basado en sus características. Algunos de los modelos de clasificación que existen son la regresión logística, Support Vector Machines, árboles de decisión, Random Forests, Extreme-Gradient Boosting (Wolff, 2022).

1.1 Set de datos

El set de datos utilizado es el del titanic, obtenido de la plataforma de Kaggle. Para consultarlo a más detalle se puede consultar en el anexo 1. El set tiene 891 registros y cuenta con las siguientes variables: Passenger ID, Survival, Ticket class (PClass), Sex, Age, Number of siblings / spouses aboard the Titanic (sibsp), Number of parents / children aboard the Titanic (parch), Ticket number, Passenger fare, Cabin number, Port of Embarkation. Estos datos presentan únicamente dos posibles resultados: 1, indicando que la persona sobrevivió, y 0, señalando que no lo logró.

Las variables utilizadas buscan representar rasgos físicos de la persona como su sexo y su edad y variables que se relacionen directamente con el Titanic, como el dinero que pagó por su boleto, el puerto en el que se embarcó y la clase en la que viajaba. Por lo tanto, las variables utilizadas fueron Survival, Ticket class (PClass), Sex, Age, Number of siblings /

spouses aboard the Titanic (sibsp), Number of parents / children aboard the Titanic (parch), Passenger fare y Port of Embarkation.

Debido a que el set de datos no está listo para utilizarse, se realizó una limpieza de datos que puede ser consultada en el anexo 2. En esta limpieza se utilizó la técnica de "one hot encoding" para las variables que eran categóricas. Además, existían algunos registros que les faltaban datos como la edad. Basados en su título y su clase se determinó un aproximado de la edad que podrían haber tenido.

Algunos de los datos que estaban en el set de datos no resultaban relevantes para el modelo. Había dos registros que no tenían información sobre donde embarcaron. Debido a que sería muy complicado hacer cálculos para determinar dónde embarcaron, se eliminaron. Para eliminar el ruido generado por los datos para el modelo, también se eliminaron los registros que tenían outliers en alguna columna.

Para facilidad de desarrollo en las siguientes fases del proyecto, se generó un nuevo set de datos que contiene el set de datos limpio, solo con las columnas a utilizar y sin datos faltantes. Este set de datos se puede encontrar en el anexo 3.

1.1.1 Separación de los datos

El set de datos obtenido de Kaggle se encuentra separado en dos documentos diferentes. El primero, *train.csv* contiene los datos categorizados con los pasajeros que sobreviven y los que no. El segundo, *test.csv* contiene registros diferentes, pero no se encuentran categorizados. El objetivo de esto es entrenar el modelo con los datos contenidos en *train.csv* y realizar predicciones sobre los registros de *test.csv* sin conocer los resultados que se deberían obtener, ya que la plataforma evalúa si las predicciones son correctas o no.

La limpieza de datos se realizó sobre estos dos documentos, pero para el desarrollo del modelo solamente se utilizarán los del archivo *train_clean.csv* (registros de *train.csv* después de la limpieza).

Para entrenar el modelo, se realizó la separación de los datos de entrenamiento en 2 grupos más, uno para entrenar el modelo y el otro para validarlo de manera iterativa y con diferentes parámetros. Cuando el desempeño del subset de datos para validación sea adecuado, se procederá a usar el de prueba original, y así obtener una estimación final de cómo el modelo se desempeñará en datos completamente nuevos.

Para la división, se designó el 80% de los datos de entrenamiento para entrenar el modelo, y el 20% para realizar la validación del mismo.

Datos de entrenamiento *train_clean.csv*	Datos de entrenamiento (80%)
	Datos para validación (20%)
Datos de prueba *test_clean.csv*	Datos de prueba

2. Métricas

Para la selección del mejor modelo, se utilizará el criterio de "**accuracy**". Esta métrica mide la proporción de predicciones correctas que hizo un modelo en relación con el total de predicciones.

Para calcularla, se utilizan los valores obtenidos de la matriz de confusión, estos valores son los siguientes:

Verdaderos positivos (VP)	Número de casos positivos (1) que el modelo predijo correctamente como positivos.
Verdaderos negativos (VN)	Número de casos negativos (0) que el modelo predijo correctamente como negativos.
Falsos positivos (FP)	Número de casos negativos (0) que el modelo predijo erróneamente como positivos.
Falsos negativos (FN)	Número de casos positivos (0) que el modelo predijo erróneamente como negativos.

La fórmula matemática para obtener la "accuracy" o exactitud del modelo es la siguiente:

$$Accuracy = \frac{VP+VN}{VP+VN+FN+FP}$$

Básicamente, consiste en dividir la cantidad total de predicciones correctas entre el número total de predicciones. La elección de esta métrica se debe a que se busca tratar de

predecir la mayor cantidad de registros correctamente. Esta métrica se utilizará tanto para el subconjunto de entrenamiento como para el subconjunto de prueba.

3. Definición de los modelos

Existen muchos modelos para hacer predicciones de clasificación. Es por esto que se realizarán pruebas con tres modelos diferentes, con hiper parámetros diferentes para cada modelo y así decidir el modelo que tenga un mejor desempeño con este set de datos.

Para decidir el modelo de clasificación que se utilizará, es importante conocer los datos que utiliza y los resultados que se pueden obtener de ellos. Por esto se listan a continuación las características principales de cada modelo.

- **Regresión logística:** Este modelo puede recibir parámetros categóricos o numéricos, pero el resultado que dará siempre será categórico. El resultado será la probabilidad de que el elemento pertenezca a una categoría, en una escala del 0 al 1.
- **Random Forest:** Es una expansión de los árboles de decisión. Los Random Forest generan promedios para los datos y así generan las conexiones. Resuelven el problema de los árboles de decisión a forzar que las variables sean categóricas.
- **Extreme-Gradient Boosting (XGBoost):** Modelo que se basa en la idea de construir árboles de decisión de manera secuencial, en donde cada árbol intenta corregir los errores de árboles anteriores. Además, cuenta con herramientas para la regularización, lo que ayuda a prevenir el sobreajuste.

Como se puede observar, cada algoritmo muestra eficiencia variada dependiendo del escenario en el que se aplique. Además, cada uno tiene sus propias restricciones en cuanto a las entradas que acepta o el número de clasificaciones de salida que maneja. Para determinar el modelo que mejor se ajusta a los datos, se realizarán pruebas con estos tres algoritmos y se utilizará la métrica de accuracy en los datos de entrenamiento y validación para elegir el más adecuado.

4. Pruebas de modelos

Se comienza con la prueba de los tres modelos seleccionados para resolver el problema. Para seleccionar el modelo que se utilizará para resolver el problema, se harán pruebas con diferentes hiper parámetros de cada uno, para que se pueda asegurar que el modelo elegido sea verdaderamente el más efectivo y no uno que por azar dio buenos resultados pero que no modele correctamente los datos.

A continuación se describen los hiper parámetros que se modificarán de cada modelo. Para que el proceso sea más efectivo se utilizará un grid search que seleccionará la mejor combinación de hiper parámetros.

Hiper parámetros para el random forest	
n_estimators	Indica la cantidad de árboles de decisión que se utilizarán.
max_depth	Marca la profundidad máxima que puede tener cada árbol de decisión.
min_sample_split	Determina cuántas muestras se necesitan como mínimo para partir un nodo.
min_sample_leaf	Especifica la cantidad mínima de muestras que debe haber en un nodo final o nodo hoja.
bootstrap	Indica si se usa el *bootstraping* para crear los árboles de decisión individuales.
criterion	Se usa para escoger el método que tendrán los árboles de decisión para hacer las divisiones en cada nodo (gini o por entropía).

Hiper parámetros para la regresión logística

penalty	Indica la regularización que se aplica al modelo, puede ser Lasso (L1), Ridge (L2) o las dos.
C	También es un hiper parámetro de regularización que indica qué tan fuerte es la regularización que se aplica al modelo.
l1_ratio	Es un parámetro que se usa cuando se tiene un *penalty* de los dos métodos. Controla la proporción para la mezcla del L1 y L2.
max_iter	Debido a que el modelo se resuelve por un método iterativo, este parámetro define cuántas iteraciones se realizan para que el modelo se ajuste a los datos.

Hiper parámetros para la XGBoost	
alpha	Marca la regularización Lasso (L1) para el modelo, dependiendo de la penalización que se de por la función de pérdida.
lambda	Marca la regularización Ridge (L2) para el modelo, dependiendo de la penalización que se de por la función de pérdida.
gamma	Parámetro de regularización que controla la mínima reducción en la pérdida para que se sigan haciendo divisiones en el nodo.
max_depth	Indica la profundidad máxima que puede tener cada árbol de decisión.
min_child_weight	Especifica el peso mínimo que debe tener cada hijo.

Al utilizar el grid search con estos hiper parámetros, se obtendrán tres modelos diferentes con los hiper parámetros que mejor se ajusten a los datos. Para consultar los rangos asignados para cada hiper parámetro se recomienda ver el código del anexo 4.

5. Análisis y comparación de los modelos

Para cada uno de los modelos obtenidos, se entrenaron con los datos de entrenamiento, se realizaron pruebas con los datos reservados para pruebas y finalmente se obtuvo la matriz de confusión y el valor de la exactitud (“accuracy”) para analizar el desempeño de cada uno.

Los resultados obtenidos de cada modelo fueron:

1. Logistic Regression (0.865)
2. Gradient Boosting Classifier (0.8426)
3. Extreme Gradient Boosting Classifier (0.831)

Sin embargo, se decidió que para tener un mejor desempeño, en donde se busque tener una mayor exactitud y precisión con los resultados, sería una buena idea usar una combinación en donde se tomen en cuenta todos los modelos, es decir, un Voting Classifier.

A pesar de que se exploró la idea de juntar los modelos en el Voting Classifier, no se encontraron mejoras significativas a los resultados, por lo que se descartó la idea. Para ver más a fondo los resultados obtenidos por el Voting Classifier se recomienda ver el reporte que se encuentra en el anexo 5.

Por lo tanto, el mejor modelo que se puede utilizar para predecir si una persona sobrevive o no en el Titanic, se obtendrá con la regresión logística, ya que no presenta overfitting sobre los datos y tiene un accuracy por arriba del 85%.

6. Optimización de hiper parámetros y regularización

Para obtener los mejores resultados con el modelo de regresión lineal, se deben modificar sus hiperparámetros y realizar una regularización.

Para el ajuste óptimo de los hiper parámetros del modelo utilizado se hizo uso de la librería *Optuna*. Esta librería tiene como objetivo explorar distintas opciones de hiper paramétros a través de algoritmos y estrategias más eficientes que si se hiciera de manera “manual”.

6.1 Hiperparametros del modelo

Se pueden modificar diferentes parámetros para que el modelo se ajuste mejor a los datos. Los hiperparámetros que se modificaron fueron los siguientes:

- **Solver (solucionador):** es el algoritmo utilizado para resolver el problema de optimización asociado con la regresión logística. Los solucionadores comunes incluyen "liblinear", "lbfgs", "newton-cg", "sag", y "saga". Cada uno de estos solucionadores utiliza un enfoque diferente para encontrar los coeficientes óptimos del modelo.
- **Fit_intercept (Ajuste de intercepción):** controla si se debe ajustar o no la intercepción (también conocida como sesgo) en el modelo de regresión logística. Si se establece en "True", el modelo ajustará la intercepción; si se establece en "False", el modelo no ajustará la intercepción y se considerará que pasa por el origen.
- **Class_weight (Peso de clases):** se utiliza para manejar el desequilibrio de clases en un problema de clasificación. Si las clases en el conjunto de datos no están balanceadas (una clase tiene muchas más muestras que otra), puedes usar este parámetro para asignar pesos diferentes a las clases.
- **Tol (Tolerancia)**: controla cuándo se considera que el modelo ha convergido durante el proceso de entrenamiento. Indica la diferencia mínima aceptable entre las iteraciones sucesivas para considerar que el modelo ha convergido. Si la diferencia entre las iteraciones cae por debajo de esta tolerancia, el modelo se considera convergente y el entrenamiento se detiene.
- **max_iter (Número máximo de iteraciones)**: para encontrar un modelo que se ajuste a los datos es necesario correr varias iteraciones. Pero el tener muchas iteraciones puede generar un overfitting, que el código sea muy tardado de correr o que se siga corriendo sin obtener mejores resultados. Por esto es importante cambiar este hiperparámetro, para que el modelo se ajuste bien a los datos y deje de ajustarse cuando no se vea una mejora

6.2 Hiperparametros de regularización

La regularización del modelo permite cambiar algunos de los hiper parámetros para evitar tener overfitting. Esto hace que el modelo generalice mejor y tenga buenas

predicciones con datos que no ha visto antes. Para la regresión lineal se utilizaron los siguientes hiper parámetros para realizar la regularización:

- **Penalty (Penalización)**: controla la regularización aplicada al modelo de regresión logística. La regularización es una técnica que se utiliza para evitar el sobreajuste al penalizar los coeficientes de las características menos importantes. Puede tomar dos valores principales: "l1" Utiliza la norma L1 para penalizar los coeficientes, lo que puede llevar a la selección automática de características al reducir algunos coeficientes a cero y "l2" el cual utiliza la norma L2 para penalizar los coeficientes, lo que ayuda a evitar coeficientes muy grandes y reduce el impacto de características menos importantes.
- **Parámetro C:** controla la regularización del modelo. La regularización evita el sobreajuste ajustando la fuerza de penalización en la función de costo. Un valor pequeño de "C" aumenta la regularización, haciendo que el modelo sea más conservador y generalice mejor, mientras que un valor grande de "C" disminuye la regularización, permitiendo un ajuste más cercano a los datos de entrenamiento, lo que podría resultar en sobreajuste.

7. Refinamiento del modelo

Para obtener la mejor combinación de hiperparámetros, se corrieron 10,000 pruebas utilizando *Optuna* y se seleccionó el modelo que tuvo mejores resultados. Antes de correr las pruebas, se tenía un modelo que daba los siguientes resultados:

```
Logistic Regression sin optimización de hiperparámetros
Logistic Regression Training : 0.846
Logistic Regression Testing : 0.775
```

Se puede ver que tiene un buen desempeño en training y en testing ya que el accuracy está por arriba de 70. De todas formas, los valores se encuentran un poco separados lo cual podría indicar que el modelo presenta un nivel de overfitting a los datos de entrenamiento.

Con *Optuna*, se obtuvieron los siguientes hiper parámetros que hacen el modelo lo más óptimo posible:

C	0.0126412354368
Penalty	l2
Solver	'saga'
Fit_intercept	False
Class_weight	Balanced
Tol	2.61438480
Max_iter	1200

8. Resultados y conclusiones

Al entrenar un modelo lineal con los hiperparámetros optimizados, se obtienen los un modelo refinado y regularizado, con los siguientes resultados:

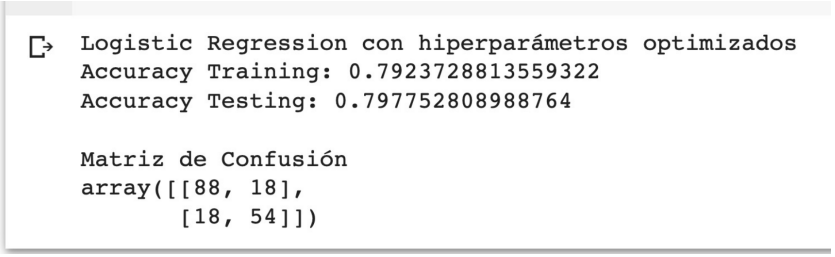

Figura 1.1 Resultados del modelo refinado y regularizado

Los resultados para training y testing se encuentran muy cercanos, alrededor de 79% de accuracy al predecir los resultados, que es un poco peor que el modelo anterior. Pero esto no presenta un problema ya que con el refinamiento y regularización se puede asegurar que el modelo será más versátil porque ya no muestra indicios de overfitting, lo que hace más probable el obtener buenas predicciones para datos nuevos o distintos a los de entrenamiento, aunque tenga un rendimiento un poco menor.

Igualmente, se comprueba esta mejoría del modelo viendo una curva ROC del desempeño antes y después del refinamiento. Si la curva se encuentra muy alejada de la diagonal central, tiene un valor diagnóstico perfecto, mientras que si se acerca mucho, es altamente probable que el modelo confunda las clases y no haga una clasificación correcta.

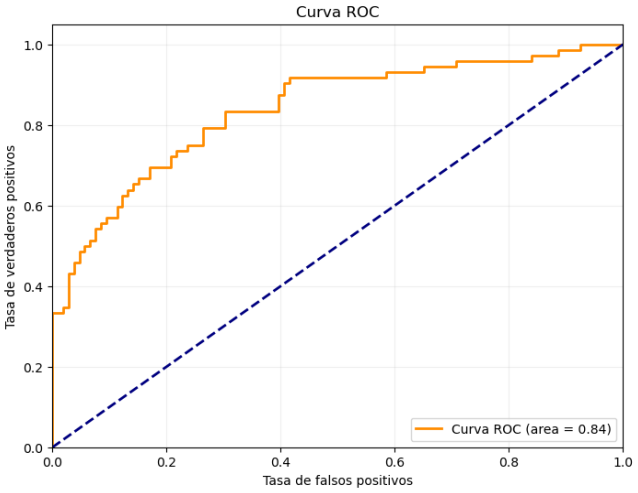

Figura 1.2 Curva ROC para el modelo sin refinamiento

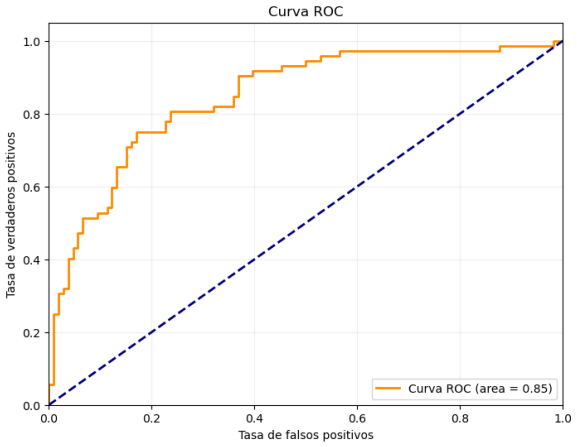

Figura 1.3 Curva ROC para el modelo después del refinamiento

En la figura 1.2 se indica que el área bajo la curva es de 0.84 para el modelo sin refinamiento, mientras que en la figura 1.3 el área bajo la curva es de 0.85. Por lo tanto, existe una pequeña mejoría del modelo.

9. Interfaz

Con el modelo final, se generó una interfaz web donde el usuario puede introducir sus propios datos y recibir una predicción si esa persona sobreviviría o no. Además, se incluyó una página especial para el administrador, donde puede incluir nuevos datos y reentrenar el modelo. Para consultar la interfaz final, ir al anexo 6.

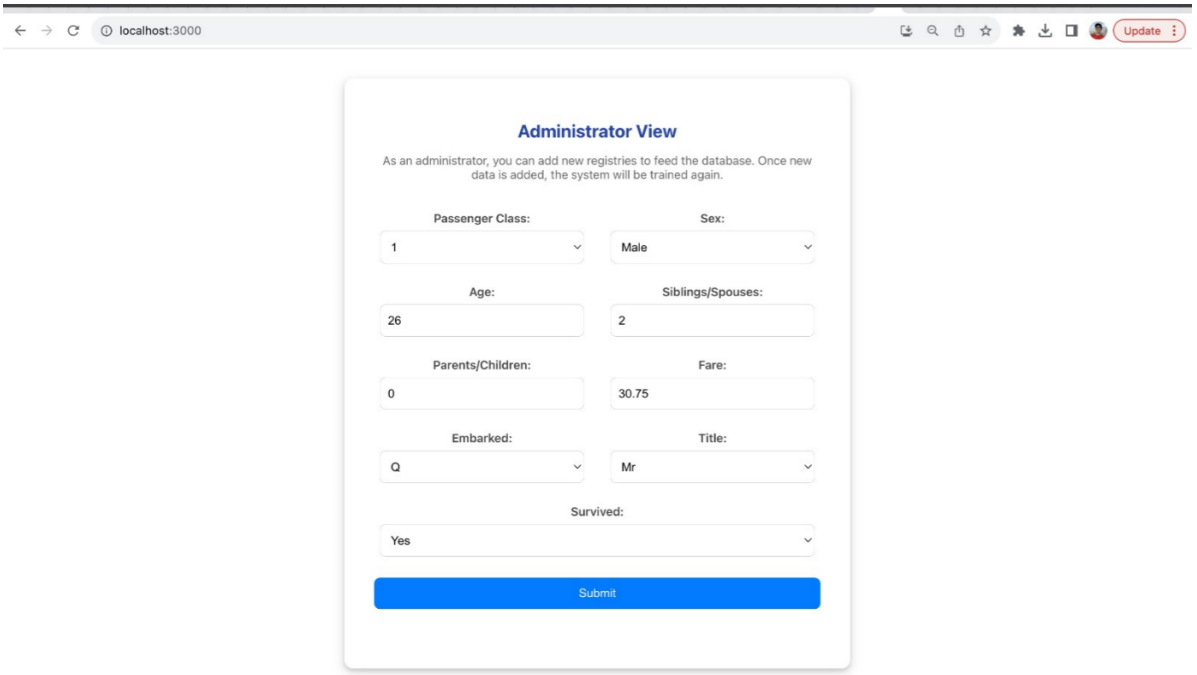

Figura 1.4 Diseño final para la interfaz

Anexos

1. Set de datos de la plataforma de Kaggle. https://www.kaggle.com/competitions/titanic/data?select=train.csv
2. Reporte de limpieza de datos. https://github.com/imjdl03/Reto-3006C-equipo5/tree/main/final/Limpieza/Reporte_Limpieza.pdf
3. Datos generados después de la limpieza https://github.com/imjdl03/Reto-3006C-equipo5/tree/main/final/Data
4. Pruebas de modelos con hiper parámetros https://github.com/imjdl03/Reto-3006C-equipo5/blob/main/final/Modelo/pruebas_de_modelacion.ipynb
5. Reporte de refinamiento del modelo https://github.com/imjdl03/Reto-3006C-equipo5/tree/main/final/Refinamiento/Reporte_Refinamiento
6. Interfaz final para la solución https://github.com/FranciscoMest02/InterfazTitanic

Bibliografía

Wolff, Rachel. (2022). *5 Types of Classification Algorithms in Machine Learning*. Estados Unidos: Monkey Learn. https://monkeylearn.com/blog/classification-algorithms/