TRABAJO FINAL

PREDICCIÓN DEL MONTO PARA ACCEDER A UN SEGURO

CURSO:INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PROFESOR: RAÚL RAMOS POLLÁN

INTEGRANTES:

Allisson Rojas

Luis Felipe Sanchez Ramirez

Andres Camilo Zuñiga Morelo

FECHA:

12-11-2022

UNIVERSIDAD DE ANTIOQUIA

Medellín - Antioquia

INTRODUCCIÓN

Planteamiento del problema (Determinar el monto de un seguro)

Hoy en día es básico y casi fundamental contar con un seguro que permite garantizar la estabilidad económica de una persona o familia, al momento de acceder a un seguro sea de cualquier campo en el que se requiera se debe acceder a cierto papeleo, y muchos datos mediante los bienes que desea asegurar la persona. Por lo tanto se desea desarrollar un modelo de código que permita mediante inteligencia artificial cuantificar el monto al momento de reclamar algún tipo de seguro,ya que al tener una mejor valoración de los respectivos datos se podrá generar una mayor precisión al determinar el valor del respectivo seguro.

Dataset

Se toma un dataset que permite predecir las cantidades de las reclamaciones de seguros que corresponde al “Tabular Playground Series Mar-2021” que contiene el siguiente link:

https://www.kaggle.com/competitions/tabular-playground-series-mar-2021/data

además de contener los siguientes 3 archivos:

- **train.csv** - los datos de entrenamiento con la `target` columna
- **test.csv** - el conjunto de prueba; estará prediciendo `target` para cada fila en este archivo (la probabilidad del objetivo binario)
- **sample_submission.csv** : un archivo de envío de muestra en el formato correcto

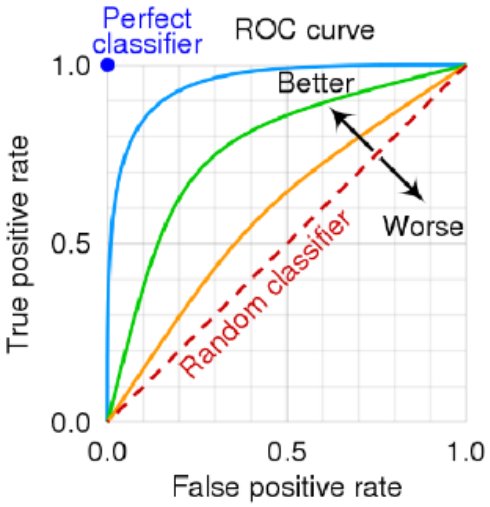
Métrica

Se ha tomado un conjunto de datos real y se genera usando un GTAN (modelo se basa en el sintetizador de datos de aprendizaje profundo basado en GAN que se presentó en la conferencia NeurIPS 2020 por el documento titulado Modelado de datos tabulares utilizando GAN condicional.)

Los datos obtenidos se dispusieron mediante letras que al recopilar se les pudo dar determinados datos numéricos.

para organizar estos datos se usa un clasificador también conocido como curva ROC , Este índice se puede interpretar como la probabilidad de que un clasificador

ordenará o puntuará una instancia positiva elegida aleatoriamente más alta que una negativa respecto a los datos



La relación del área bajo la curva ROC ha demostrado una relación con el Coeficiente de Gini,que se muestra en la siguiente fórmula

$$AUC = \frac{1}{2}(G_1 + 1),$$

donde:

$$G_1 = 1 - \sum_{k=1}^{n}(X_k - X_{k-1})(Y_k + Y_{k-1})$$

El conjunto de datos se les puede dar un respectivo valor en la curva ROC en el que será usado el área bajo la curva entre los valores de la probabilidad del target predicho y observa para determinar un valor diferente al que fuera el presentado en las tablas que equivale a un 0,5.

EXPLORACIÓN DESCRIPTIVA DEL DATASET

Al momento de comenzar con la exploración de variables, primero se debe acceder a la información repartida encontrada en **“train.csv,” “test.csv” y “sample_submission.csv” ,** al final se obtiene un dataset con todas las variables disponibles, este dataset tiene por nombre *“01_Exploración_de_datos”* luego se analizan algunas variables que requieran de análisis e interés. El archivo original

contaba con 500000 filas, de las cuales repartieron 300000 para train con su columna de resultados (target), y 200000 para test cuya columna target fue retirada.

Se consigue verificar si hay vectores con valor nulo en el grupo de datos, pero no existen este tipo de valores. Se consigue ver la proporción de valores 0 respecto a valores 1 en la columna target, que son graficados en la figura 1.

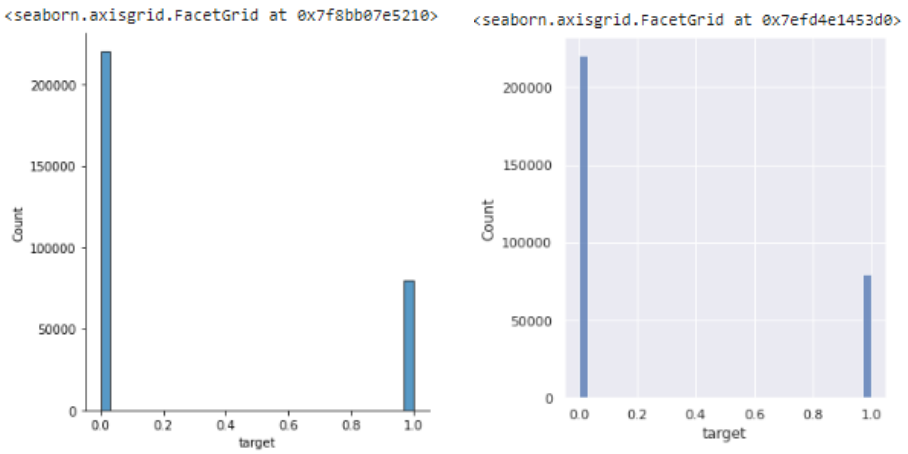


Figura 1. cantidad de valores 0 y 1 en los datos.

Se extrae del gran grupo de datos una matriz de 2000 muestras aleatorias que permite observar la correlación entre los datos que son dados como letras y los distintos datos numéricos de término flotante

id	cat0	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont2	cont3	cont4	cont5	cont6	cont7	cont8
496727	A	K	L	A	E	BI	C	AS	AO	A	...	0.633350	0.560301	0.827169	0.286365	0.604044	0.664543	0.32744
271954	A	L	M	A	F	BI	A	U	M	A	...	0.412224	0.309342	0.272437	0.681509	0.560443	0.380293	0.37784
214451	A	N	D	A	E	AB	A	AI	BK	A	...	0.356967	0.358495	0.214972	0.737280	0.489606	0.358737	0.36163
437273	A	I	A	A	E	BI	A	AV	AE	N	...	0.314126	0.601055	0.289341	0.735080	0.371766	0.351074	0.36212
150101	A	F	D	A	H	BI	A	AS	L	C	...	0.857233	0.846578	0.364537	0.781422	0.340745	0.883776	0.75141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
413479	A	I	A	A	E	BI	A	U	AE	A	...	0.272109	0.626213	0.235916	0.737046	0.470881	0.474993	0.44621
302988	B	O	F	C	E	BI	A	U	BM	A	...	0.721551	0.548196	0.809795	0.735205	0.676300	0.209965	0.38324
107975	A	N	A	D	F	BI	A	AH	AX	I	...	0.351010	0.341208	0.235602	0.791763	0.555594	0.346806	0.48161
163140	A	L	G	A	D	K	C	AK	BM	A	...	0.157727	0.265058	0.713893	0.089546	0.341776	0.311527	0.13211
22303	B	A	A	B	E	BI	A	N	S	A	...	0.363145	0.680288	0.615827	0.555436	0.252375	0.565532	0.53075

2000 rows × 31 columns

figura 2.cuadro que recopila un grupo de datos

Distribución de las variables numéricas

En la figura 3 se muestran las distribuciones de cada variable, algunas se pueden presentar

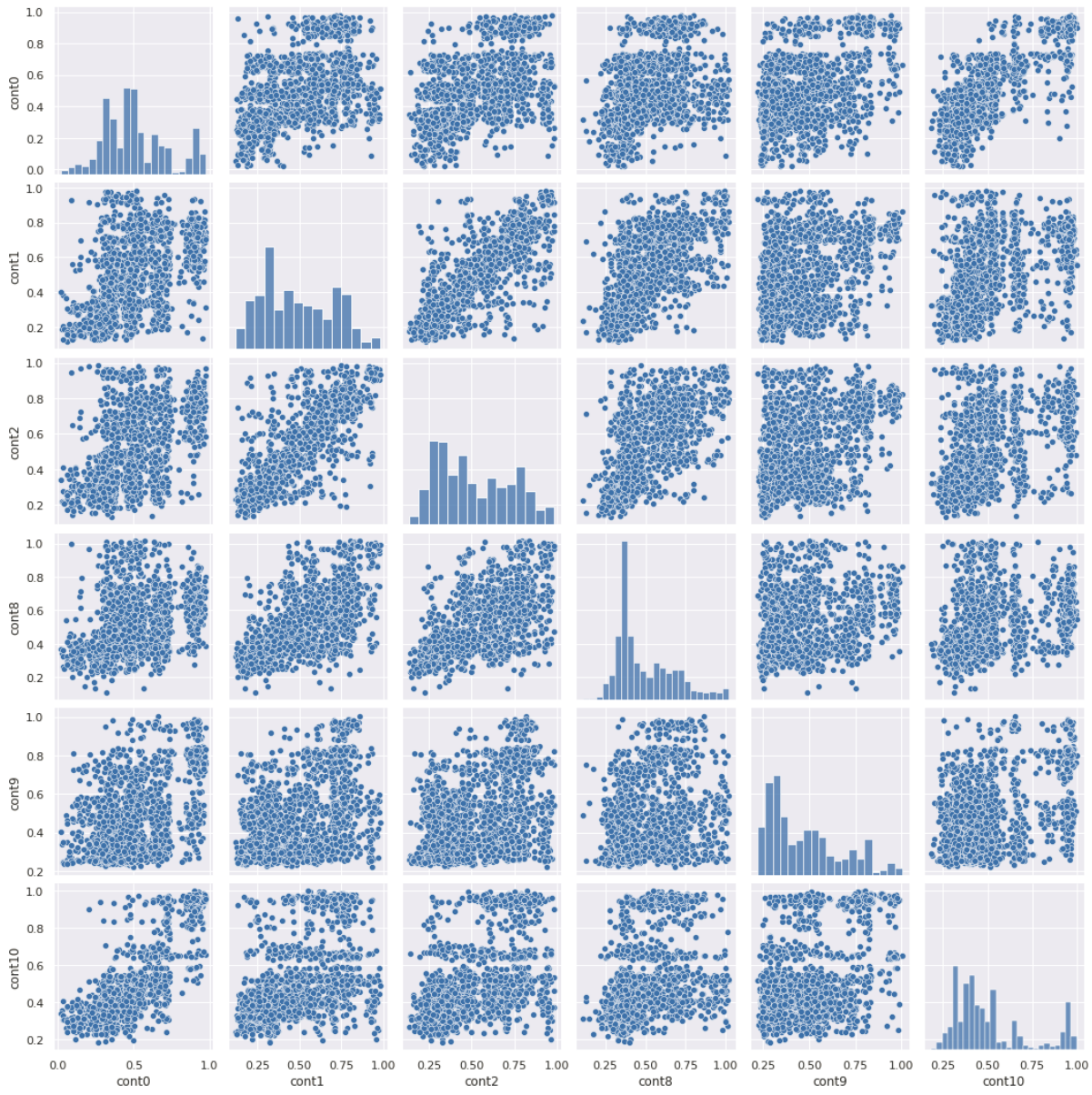


Figura 3. gráfica de algunas columnas

En estas columnas numéricas se puede observar un caos desordenado en general, excepto en, por ejemplo, la interacción que ocurre entre cont1, cont2 y cont8. Es muy posible que los valores de estas 3 columnas agrupen información común.

Distribución de las columnas categóricas

Se seleccionan algunas columnas categóricas para identificar los datos que pueden estar repetidos en todas sus filas.

```
from collections import Counter
print(Counter(train['cat0']))
print(Counter(train['cat1']))
print(Counter(train['cat2']))
print(Counter(train['cat3']))
print(Counter(train['cat4']))
print(Counter(train['cat5']))
print(Counter(train['cat6']))
print(Counter(train['cat7']))
```

```
Counter({'A': 223525, 'B': 76475})
Counter({'I': 90809, 'F': 43818, 'K': 41870, 'L': 31891, 'H': 17257, 'N': 13
Counter({'A': 168694, 'C': 38875, 'D': 22720, 'G': 18225, 'Q': 10901, 'F': 9
Counter({'A': 187251, 'B': 79951, 'C': 15957, 'D': 8676, 'E': 3318, 'F': 248
Counter({'E': 129385, 'F': 76678, 'G': 30754, 'D': 27919, 'H': 23388, 'J': 4
Counter({'BI': 238563, 'AB': 41639, 'BU': 6740, 'K': 2713, 'G': 683, 'BQ': 4
Counter({'A': 187896, 'C': 71427, 'E': 16581, 'G': 11198, 'I': 6648, 'M': 21
Counter({'AH': 45818, 'E': 39601, 'AS': 25326, 'J': 16135, 'AN': 16097, 'U':
```

Figura 4. Valoración cuantitativa de las columnas categóricas.

A partir de este conteo de variables repetidas, se busca la que mayor veces se repite.

```
from collections import Counter
col_cat = ['cat0', 'cat1', 'cat2', 'cat3', 'cat4', 'cat5', 'cat6', 'cat7', 'cat8',
        'cat9', 'cat10', 'cat11', 'cat12', 'cat13', 'cat14', 'cat15', 'cat16',
        'cat17', 'cat18']

result = Counter()
for x in col_cat:
    result = result + Counter(train[x])

result.most_common(1)
```

```
[('A', 1975085)]
```

Figura 5. código para encontrar cual letra se repite más

La que aparece más veces es la correspondiente a la letra A, por tanto es posible que sea la de mayor peso para la decisión de asignación al target de 0 o 1.

TRATAMIENTO DE DATOS

Se realiza una codificación de las columnas categóricas “cat#:”

Se cambian de letras a números usando la función “LabelEncoder” de la sección preprocessing de “scikit-learn”

```
for c in train.columns:
    if train[c].dtype=='object':
        lbl = LabelEncoder()
        lbl.fit(list(train[c].values) + list(test[c].values))
        train[c] = lbl.transform(train[c].values)
        test[c] = lbl.transform(test[c].values)
display(train.head())
```

id	cat0	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont2	cont3
0	0	8	0	1	1	33	0	44	54	0	...	0.759439	0.795549
1	0	8	0	0	4	33	8	48	3	5	...	0.386385	0.541366
2	0	10	0	0	4	33	0	30	38	9	...	0.343255	0.616352
3	0	10	0	2	4	33	0	50	3	5	...	0.831147	0.807807
4	0	8	6	1	4	33	2	32	54	0	...	0.338818	0.277308

Figura 6. código para intercambiar letras por números naturales.

Para facilitar el análisis de los datos se separa del dataframe que se llama “train” la columna que necesitamos que contiene los datos compuestos por 1 y 0 , el uno que determina que se acepta y el cero que no es aceptado se selecciona de “target”.

```
[ ]  target = train.pop('target')
     display(target.head)
```

Figura 7. código que permite seleccionar una columna

Procesado

A partir de la matriz de entrenamiento “train” y usando los resultados de “target,” que es la columna con la información que se necesita, se saquen 4 sub-variables: X_train, X_test, y_train, y_test.

Es decir, se van a sacar 2 matrices X y 2 vectores donde en cada matriz todas las filas son aleatorias con la condición que se corresponden los 'X' con los 'y' respectivos.

Se coloca en el código "validation split" con la función train_test_split de la sección model_selection de scikit-learn.

```
] X_train, X_test, y_train, y_test = train_test_split(train, target, train_size=0.60)

  display(X_test.head)
```

Figura 8. código para sacar 4 sub-variables.

La variable bosque será el bosque aleatorio de árboles de decisión a incorporar, con un total de 200 árboles que se pueden determinar de las respectivas filas de datos

Por defecto se calcula con 100 árboles o "estimadores".

Como hay tantas variables, se acota la profundidad de las ramas a 7, para que no se expandan hasta el infinito y gasten recursos del sistema.

n_jobs=-1 significa que se debe utilizar todos los procesadores en paralelo.

```
[ ] bosque = RandomForestClassifier(n_estimators=200,
                                    max_depth=7,
                                    n_jobs=-1)
```

Con `.fit` se incorporan las sub-variables de entrenamiento al `bosque`

```
[ ] bosque.fit(X_train, y_train)

    bosque
```

Figura 9. Clasificador de random forest

La función predict_proba toma la media de probabilidad de predicción y entrega un vector con los valores entre 0.00 y 0.99 para el caso de que sea 0 o 1 de cada renglón.

```
y_pred = bosque.predict_proba(X_test)
y_pred
```

```
array([[0.67798122, 0.32201878],
       [0.14444123, 0.85555877],
       [0.92836426, 0.07163574],
       ...,
       [0.81264301, 0.18735699],
       [0.90030828, 0.09969172],
       [0.81127914, 0.18872086]])
```

Figura 10. Asignación de variable y_pred.

RETOS Y CONSIDERACIONES DE DESPLIEGUE

Durante la realización del trabajo el principal reto fue la selección del clasificador, se eligió el bosque aleatorio de árboles de decisiones (random forest) porque se ajusta de forma adecuada al criterio del área debajo de la curva ROC, también fue importante cambiar las columnas categóricas a valores numéricos para que el 'RandomForestClassifier' leyera adecuadamente dichas variables.

Para implementar el clasificador se encontraron 2 opciones; una fue usando librerías de LightGBM, desarrollada por Microsoft, y la otra utilizando las librerías de Scikit-learn. Se optó por utilizar esta última ya que funciona óptimamente y es de código abierto.

Adicionalmente, y en general, es un gran reto el aprendizaje operacional de las plataformas GitHub, Kaggle, Google Colab, y también adquirir familiaridad con las librerías de pandas, numpy y sklearn. Integrar todas estas herramientas para resolver problemas que de otra manera probablemente la única opción fuese prueba y error, o directamente lanzar una moneda, demuestra la potencia de aplicar estos conocimientos a problemas reales.

CONCLUSIONES

Hay muchas maneras de solucionar este reto, a medida que fuimos aventurándonos a conocer más información nos dimos cuenta que hay varias formas, por ejemplo Random Forest, CatBoost, XGBoost, LightGBM entre otras, por la estructura que maneja

de tener 300.000 filas en train y 200.000 en test, con 19 características categóricas y 11 características continuas, para un total de 30, siendo un dataset sintético pero basado en uno real.

no tuvimos información de un hyperparameter entonces creemos que los resultados pueden ser mejores o al menos diferentes intentando otros modelos, ya que el que intentamos esta vez es Random Forest, pero con alguno de los otros y mayor conocimiento para sacar el hyperparameter, podría dar un resultado diferente por lo que puede mejorar el resultado, pero con la información que teníamos nos dio un muy buen resultado que podemos seguir mejorando a medida que interactuemos con diferentes formas.

WEBGRAFÍA

https://www.kaggle.com/competitions/tabular-playground-series-mar-2021/data

https://scikit-learn.org/stable/