Proyecto de semestre - Entrega final

UNIVERSIDAD
DE ANTIOQUIA
1 8 0 3

Materia
Introducción a la inteligencia artificial

Docente
Raúl Ramos Pollan

Estudiantes
Hernán Aguilar Cruz

Jhonier Jiménez Acevedo

Oriana Mejía Cardona

Universidad de Antioquia

Medellín
2023

Introducción

Descripción del problema

El Taiwan Economic Journal, una empresa de información financiera de Taiwan, recogió datos desde el año 1999 hasta el 2009. La intención es identificar si la compañía puede entrar en quiebra teniendo en cuenta las regulaciones comerciales de la bolsa de valores de Taiwan.

Dataset

El dataset utilizado es Company Bankrupcy Prediction (https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction) el cual tiene 6.819 instancias o filas, y 96 columnas o atributos.

Métricas de desempeño

Para el desarrollo del proyecto se plantean las siguientes métricas de Machine Learning:

- Logistic Regression
- Random Forest Classifier
- SVC
- KNN Confusion matrix
- Naive Bayes

Exploración descriptiva del dataset

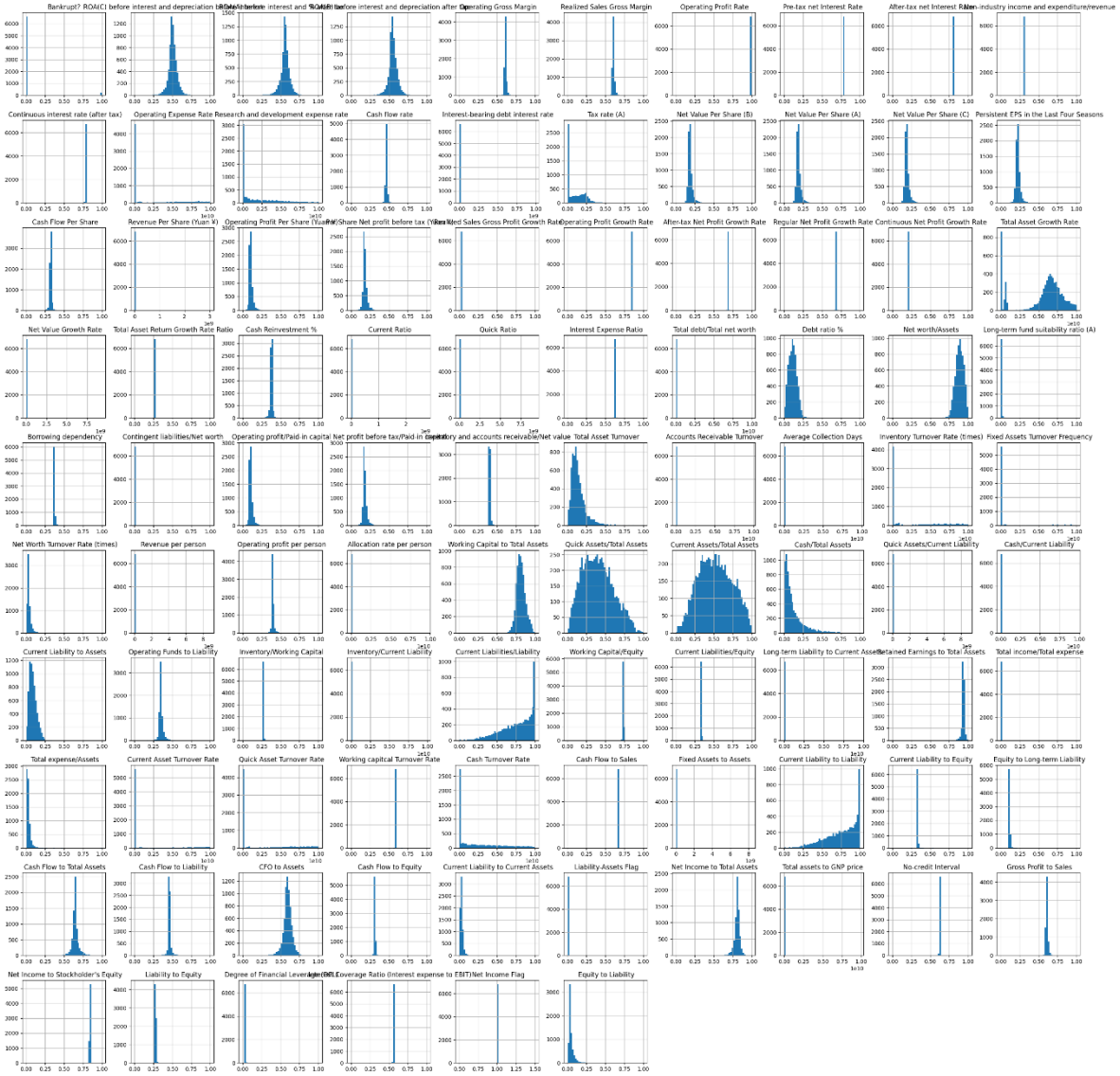
En el archivo 01- Simulación de datos.ipynb se muestra el dataset original (6819 x 96) que se carga al notebook con el nombre de *data.csv*, este no cuenta con columnas categóricas ni valores nulos, por lo que es necesario simularlos; primero es necesario convertir variables numéricas a categóricas, lo cual se logra con la función *modify_column(),* la cual toma columnas con valores menores a 1 y clasifica los valores en *Low, Medium y High.* De esta forma se cumple con el requisito de tener al menos 10% de columnas categóricas.

```python
def modify_column(data, column_name):
    if column_name not in data.columns:
        print(f"{column_name} is not a valid column name.")
        return data
    else:
        col_idx = data.columns.get_loc(column_name)
        for i in range(len(data)):
            if data.iloc[i, col_idx] < 0.33:
                data.iloc[i, col_idx] = "LOW"
            elif data.iloc[i, col_idx] < 0.66:
                data.iloc[i, col_idx] = "MEDIUM"
            else:
                data.iloc[i, col_idx] = "HIGH"
        return data
```

En segundo lugar, el dataset no cuenta con valores nulos por lo que se lleva a cabo una eliminación de datos en 3 columnas.

```
for i in range(0, len(data), 5) :
    data.at[i, ' Operating Gross Margin'] = np.nan
    data.at[i, ' Cash Turnover Rate'] = np.nan
    data.at[i, ' Current Liability to Equity'] = np.nan
```

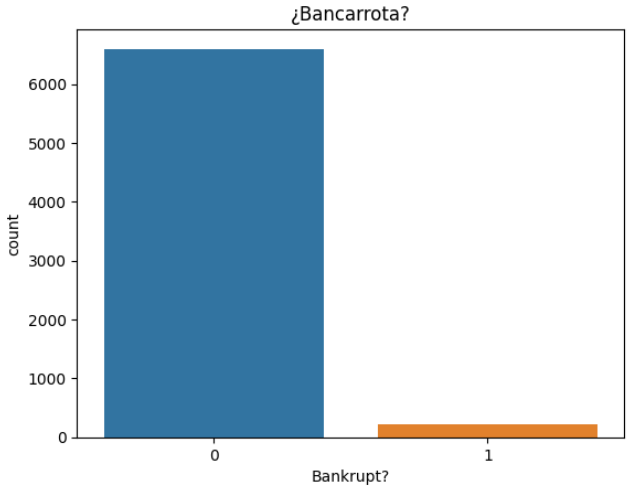
En este notebook también se puede apreciar la distribución de los datos:



En el archivo 02 – Exploración de datos.ipynb se realiza la exploración de los datos del nuevo dataset llamado *simulated_data.csv*, para identificar patrones, graficarlos y conocer los datos.

- **Tamaño del dataset:** El tamaño del dataset es de (6819, 96).
- **Valores nulos:** Las columnas *Operating Gross Margin, Cash Turnover Rate y Current Liability to Equity* tienen 1364 valores nulos.

- **Variable objetivo:** Se grafica la variable objetivo para ver su distribución y notamos que este desequilibrado, debido a que el valor 0 (o sea, que la empresa no entra en bancarrota) es la más presente.



- **Tipos de datos:** Los tipos de datos son es su mayoría float64, hay también algunos int64 y object que serian los datos que se convirtieron a nulos y categóricos.
- **Inspección de datos numéricos:** En esta inspección se calcula la desviación estándar, valor mínimo, máximos y percentiles de las columnas numéricas y promedio.
- **Inspección de variables categóricas:** A las variables que se convirtieron en categóricas se les realiza un conteo de las filas en cada categoría (Low, Medium y High).

Iteraciones

Las iteraciones se realizan en base al archivo *processed_data.csv* que es el archivo que se genera después de haber realizado el preprocesamiento te los datos en el notebook 03 – Preprocesado de los datos.ipynb.

En el archivo 03 – Preprocesamiento de los datos.ipynb se utiliza el dataset llamado *simulated_data.csv* en el cual se realiza un preprocesamiento de los datos que consiste en:

- **Llenar datos nulos:** Como hay 3 columnas con datos nulos, se utiliza la media para llenar esos datos utilizando la función fillna().

```
data[' Operating Gross Margin'] = data[' Operating Gross Margin'].fillna(data[' Operating Gross Margin'].mean())
data[' Cash Turnover Rate'] = data[' Cash Turnover Rate'].fillna(data[' Cash Turnover Rate'].mean())
data[' Current Liability to Equity'] = data[' Current Liability to Equity'].fillna(data[' Current Liability to Equity'].mean())
```

- **Convertir variables categóricas a numéricas:** Utilizamos la estrategia One Hot Encoding para crear una columna para cada valor distinto que existe (Low, Medium, High) y cada registro marcha un 1 en la columna a la que pertenece ese registro y deja un 0 en los demás. La función que realiza este registro es get_dummies().

```
df1 = pd.get_dummies(data = data, columns = [' ROA(C) before interest and depreciation before interest'], prefix = "is_ROAC")
df2 = pd.get_dummies(data = data, columns = [' ROA(A) before interest and % after tax'], prefix = "is_ROAA")
df3 = pd.get_dummies(data = data, columns = [' ROA(B) before interest and depreciation after tax'], prefix = "is_ROAB")
df4 = pd.get_dummies(data = data, columns = [' Non-industry income and expenditure/revenue'], prefix = "is_NIIER")
df5 = pd.get_dummies(data = data, columns = [' Inventory Turnover Rate (times)'], prefix = "is_ITR")
df6 = pd.get_dummies(data = data, columns = [' Working Capital/Equity'], prefix = "is_WCE")
df7 = pd.get_dummies(data = data, columns = [' Cash Flow to Sales'], prefix = "is_CFS")
df8 = pd.get_dummies(data = data, columns = [' Net Income to Total Assets'], prefix = "is_NITA")
```

Teniendo en cuenta lo anterior, comenzamos con las diferentes iteraciones:

Iteración 1: Modelos sin SMOTE.

Se separan los datos en X y Y, donde X contiene todas las columnas que no sean la variable objetivo o variable a predecir, y Y es la que contiene la variable a predecir que en este caso es bankrupt?. Se dividen los datos en train y test, y se realiza una división mediante 4 folds del método de Stratified K Fold, con el fin de que las pruebas se realicen conservando la estratificación de los datos. Y se empiezan a utilizar modelos sin la técnica de oversampling SMOTE.

```
[ ]  X = data.drop('bankrupt?', axis=1).values
     Y = data['bankrupt?'].values
     print (X.shape , Y.shape)

     (6819, 111) (6819,)

[ ]  X_train, X_test, y_train, y_test = train_test_split(X, Y,stratify=Y,test_size=0.30)
```

Visualizando el archivo 04 – Modelos sin SMOTE.ipynb Una vez hecho esto, se prueban los siguientes modelos:

Gradient boosting tree

En la ejecución del modelo de GBT se da un rango de entre 20 y 300 estimators con el objetivo de obtener un resultado robusto al momento de obtener los resultados del accuracy del modelo. Una vez hecho esto, se recorren la cantidad de árboles ingresada. Para cada árbol entonces, se utiliza la librería GradientBoostingClassifier con el número de estimadores, en este caso de árboles, luego se toman los datos de training y test y se toman de estos los datos: accuracy_score, precision_score, recall_score y f1_score, además agregando la eficiencia, la desviación estandar y el intervalo de confianza.

	número de arboles	eficiencia de entrenamiento	desviacion estandar entrenamiento	eficiencia de prueba	Intervalo de confianza (prueba)	accuracy real	precision_score	recall_score	f1_score
0	20.0	0.982052	0.001069	0.968573	0.001825	0.968573	0.558166	0.174764	0.960618
1	50.0	0.989036	0.000799	0.970459	0.002166	0.970459	0.587662	0.298920	0.965792
2	100.0	0.995251	0.001134	0.969412	0.002809	0.969412	0.552919	0.292341	0.964837
3	200.0	0.999441	0.000342	0.970040	0.001715	0.970040	0.567913	0.304993	0.965618
4	300.0	1.000000	0.000000	0.970250	0.001910	0.970250	0.565711	0.331309	0.966439

Podemos ver entonces, que la eficiencia de entrenamiento va aumentando de manera considerable, cada vez que se añaden más árboles al modelo. Así mismo, vemos que la eficiencia, el accuracy y el f1_score también suben. Los resultados son considerablemente buenos teniendo como base que se esperaba al menos un 90% de accuracy para los modelos a realizar.

Support Vector Machine

Para esta prueba se utiliza la librería SVC con el kernel "rbf", gamma en 0.01 y el parámetro en 0.01, luego se toman los datos de training y test, de estos los datos se toma: accuracy_score, precision_score, recall_score y f1_score, además agregando la eficiencia, la desviación estandar y el intervalo de confianza.

	kernel	gamma	param_reg	error de entrenamiento	error de prueba	% de vectores de soporte	accuracy real	Intervalo de confianza (prueba)	precision_score	recall_score	f1_score
0	rbf	0.01	0.01	0.967737	0.967737	54.333484	0.967737	0.000008	0.0	0.0	0.983604

Podemos ver que el accuracy real da de un 96%, un 6% más alto de lo esperado según nuestros retos de la entrega 1, esto significaría que el modelo realizado podría ser puesto en producción ya que tiene un resultado esperado alto.

Iteración 2: Modelos con SMOTE.

Ahora procedemos a realizar modelos utilizando la técnica de SMOTE (Synthetic Minority Over-sampling Technique) para abordar el desequilibrio de clases en los datos. SMOTE es una técnica de sobremuestreo que genera instancias sintéticas de la clase minoritaria para equilibrar la distribución de clases.
Estos modelos se pueden visualizar en el archivo 05 – Modelos con SMOTE.ipynb.

Validación cruzada de k iteraciones o k-fold cross validation

Recordemos que la validación cruzada es una técnica utilizada para evaluar el rendimiento de un modelo de aprendizaje automático de manera más confiable.
En este caso, utilizamos una validación cruzada con 5 iteraciones (n_splits=5). Esto significa que el conjunto de datos se divide en 5 partes o "pliegues" de tamaño aproximadamente igual. Luego, ejecutamos el algoritmo 5 veces, utilizando cada vez 4 partes como conjunto de entrenamiento y 1 parte como conjunto de prueba.
Los conjuntos de entrenamiento los almacenamos en las variables X_train_sm y y_train_sm, mientras que los conjuntos de prueba en X_val_sm y y_val_sm. Y finalmente, convertimos los conjuntos de datos en matrices NumPy utilizando el método values.
Cada uno de los modelos a desarrollar recibirá como parámetros estas matrices anteriormente descritas para así dar control al desequilibrio de clases; lo que nos permitirá tener una mejor evaluación de estos.

Modelos

Regresión Logística

En esta sección realizamos una regresión logística agregando la técnica de SMOTE para abordar el desequilibrio de clases en los datos.
En cada iteración, entrenamos un modelo utilizando el conjunto de entrenamiento y se evalúa en el conjunto de validación. Recordemos que aquí los conjuntos son un objeto de validación cruzada estratificada que divide los datos en k pliegues y garantiza que se mantenga la proporción de clases en cada pliegue.
Utilizamos medidas de evaluación como la precisión, la recuperación (recall), la puntuación F1 y el área bajo la curva (AUC) para evaluar el rendimiento del modelo en cada división.
Finalmente, calculamos el promedio de estas métricas para obtener una evaluación general del modelo de Regresión Logística con SMOTE.
Estos fueron los resultados:

```
Regresión Logística con SMOTE resultados:

accuracy: 0.6219622299015262
precision: 0.6677820992436787
recall: 0.4801082543978349
f1: 0.5581182623228066
```

```
rand_log_reg.best_params_
```

```
{'solver': 'newton-cg', 'penalty': 'l2', 'class_weight': 'balanced', 'C': 10}
```

Potenciación del gradiente

En esta sección utilizamos un modelo de Gradient Boosting Classifier (GBC) utilizando nuevamente la técnica de sobre muestreo SMOTE para abordar el desequilibrio de clases en los datos. Realizamos búsquedas aleatorias de hiper parámetros para encontrar la mejor configuración del modelo GBC. Luego, realizamos una validación cruzada estratificada para evaluar el rendimiento del modelo.
Finalmente, se calculan y se imprimen las métricas de evaluación promedio, como la precisión, la exhaustividad (recall) y la exactitud (accuracy) del modelo GBC.
Estos fueron los resultados:

```
accuracy: 0.9889044731793142
precision: 0.985031638802148
recall: 0.9929634641407308
f1: 0.9889744343455653
```

```
rand_gbc.best_params_
```

```
{'random_state': 42, 'n_estimators': 300, 'max_features': 'log2'}
```

Potenciación extrema del gradiente

En esta sección entrenamos un modelo de Extreme Gradient Boosting (XGBoost) enviando como parámetro el mismo SMOTE. Utilizamos también diferentes combinaciones de hiper parámetros para el modelo XGBoost mediante una búsqueda aleatoria y realizamos una validación cruzada estratificada para evaluar el rendimiento del modelo.
Finalmente, calculamos el promedio de estas métricas para obtener una evaluación general del modelo de Extreme Gradient Boosting con SMOTE.
Estos fueron los resultados:

```
--------------------------------------------------------------------------------
Extreme Gradient Boosting (con SMOTE) resultados:

accuracy: 0.9955345060893098
precision: 0.9946380697050937
recall: 0.9964817320703654
f1: 0.9955555555555555
--------------------------------------------------------------------------------
```

```
rand_xgb.best_params_
```

```
{'max_depth': 3, 'lambda': 2, 'eval_metric': 'logloss', 'eta': 0.1, 'alpha': 1}
```

Bosques Aleatorios

Con este código realizamos un modelo de Random Forest Classifier (RFC) o bosque aleatorio; utilizando la técnica de sobre muestreo SMOTE. También realizamos una búsqueda aleatoria de hiper parámetros para encontrar la mejor configuración del modelo RFC. A continuación, se realiza una validación cruzada estratificada para evaluar el rendimiento del modelo. Finalmente, calculamos e imprimimos las métricas de evaluación promedio, como la precisión, la exhaustividad (recall) y la exactitud (accuracy) del modelo RFC.
Estos fueron los resultados:

```
[ ]  label = ['Fin.Stable', 'Fin.Unstable']
     smote_prediction_rfc = best_est_rfc.predict(X_val_sm)
     print(classification_report(y_val_sm, smote_prediction_rfc, target_names=label))
```

```
                precision    recall  f1-score   support

   Fin.Stable       1.00      0.97      0.98       923
 Fin.Unstable       0.97      1.00      0.98       924

     accuracy                           0.98      1847
    macro avg       0.98      0.98      0.98      1847
 weighted avg       0.98      0.98      0.98      1847
```

CatBoostClassifier

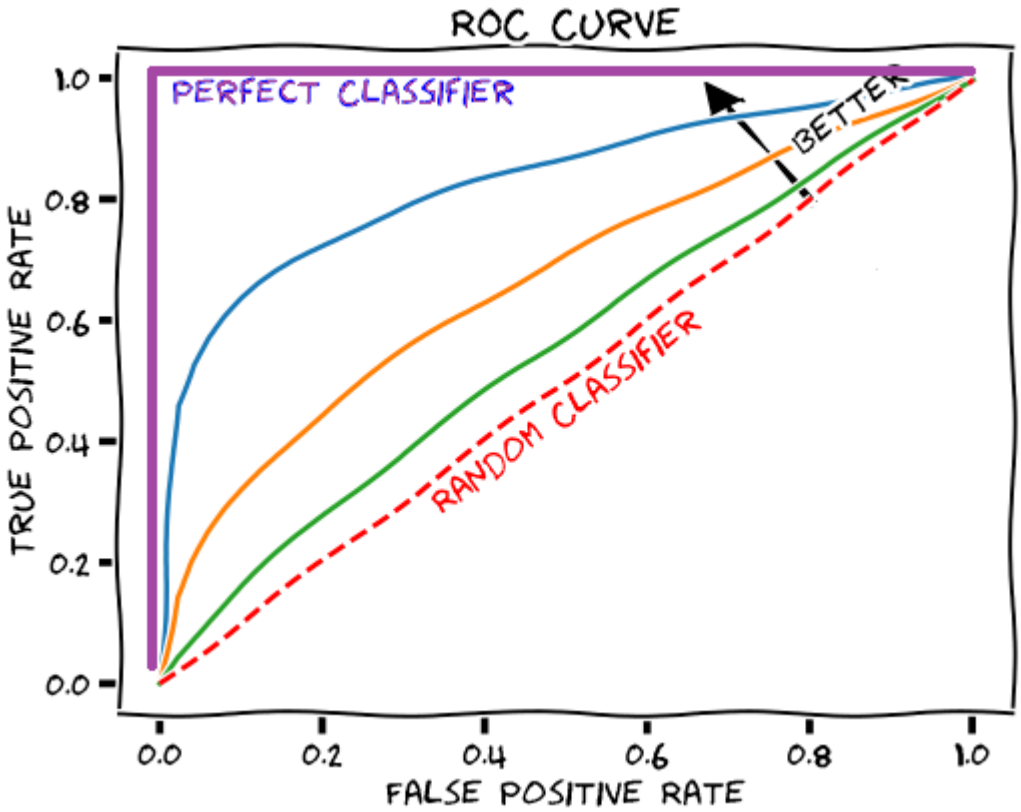
Finalmente realizamos un modelo de CatBoostClassifier (CBT) utilizando la técnica de sobremuestreo SMOTE para abordar el desequilibrio de clases en los datos. Se realiza una búsqueda aleatoria de hiperparámetros para encontrar la mejor configuración del modelo CBT. A continuación, se realiza una validación cruzada estratificada para evaluar el rendimiento del modelo.
Finalmente, se calculan y se imprimen las métricas de evaluación promedio, como la precisión, la exhaustividad (recall) y la medida F1 del modelo CBT
Estos fueron los resultados:

```
precision: 0.9859761114421085
recall: 0.9921515561569689
f1: 0.9890308826238676
```

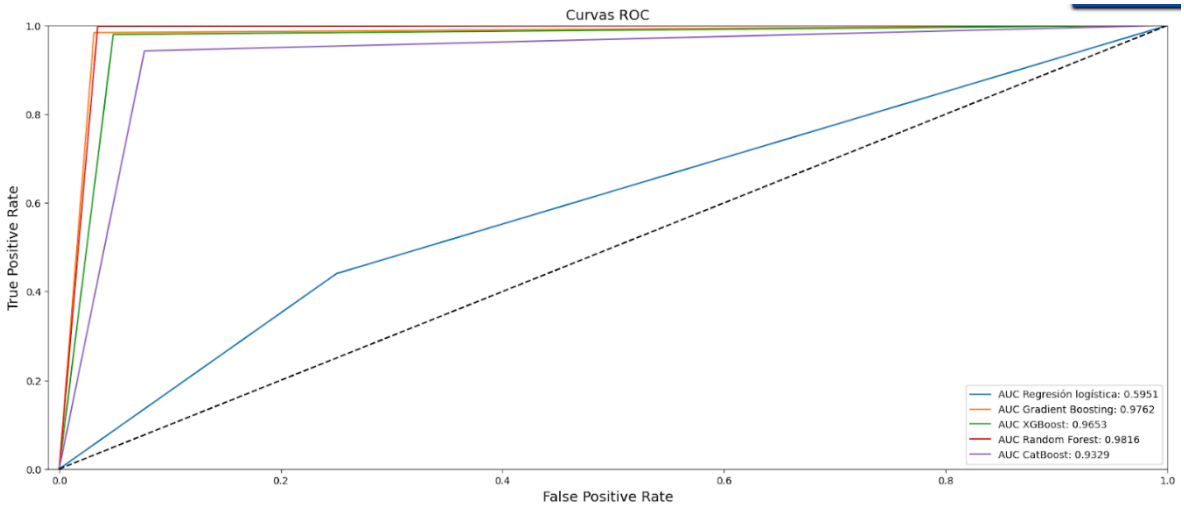
```
rand_cbt.best_params_
```

```
{'random_seed': 42,
 'learning_rate': 0.01,
 'iterations': 100,
 'eval_metric': 'F1',
 'auto_class_weights': 'SqrtBalanced'}
```

Comparación de la efectividad de los modelos



Tal como nos explica en la imagen, la curva ROC nos permite evaluar el área bajo la curva y encontrar el modelo con los mejores resultados. Que para nuestro caso es el Random Forest.



Retos

Los retos planteados en el transcurso del proyecto fueron propuestos en la entrega 1, donde se esperaba que los modelos tuviesen un accuracy del 90% o superior. Esto anterior, ya que consideramos que se trata de un caso de mucha importancia como lo es la bancarrota en empresas, y queríamos que los modelos que fuesen a ser creados tuviesen resultados altamente confiables, y de no ser de esta manera, entonces se consideraría que no valdría la pena poner los modelos en producción ya que las empresas no estarían interesadas en utilizarlos.

Este porcentaje de 90% no fue elegido al azar, sino más bien fue un promedio de los resultados que obtenían las personas que aportaban código en el reto de Kaggle con sus modelos, a su vez proponiéndonos el reto de dejar los modelos con un accuracy alto.

Conclusiones

- Poder encontrar un dataset que cumpliese con las especificaciones del curso no fue tarea fácil, ya que son solo unos cuantos que cumplen el tamaño en la cantidad de filas y columnas, y si nos vamos a especificaciones sobre nulos y columnas categóricas entonces las opciones son más bajas. Por suerte, y conocimiento de materias anteriores de los integrantes, se tenía la página de UCI – Machine Learning que nos permitió encontrar un dataset con los requerimientos, y ya partiendo del mismo, buscamos un reto de Kaggle que lo estuviese utilizando para realizar el trabajo de una manera más cómoda.
- La inteligencia artificial es un mundo cada vez más grande, y que lo escuchamos día tras día ya sea en televisión, internet, por un amigo, etc. Entender sobre esta rama que se ha crecido de manera exponencial en los últimos años hace parte de los deberes que tenemos como ingenieros, afrontándonos a una industria cada vez más competitiva.
- El uso de la ténica de oversampling SMOTE fue de gran ayuda para lograr entender los motivos por los que esta se utiliza. Como se vió a en los notebooks realizados, nuestra característica objetivo estaba altamente desbalanceada, y esto podría afectar en el resultado del accuracy de nuestros modelos, por lo que utilizarla hace parte de la creación de un modelo robusto, mejorando así la tasa de acierto de una predicción.
- Para utilizar datasets, sea el que sea, es importante entender el contenido del mismo, saber el tamaño, cuál es el objetivo o variable, si cuenta con una cantidad alta de valores nulos o si no cuenta con ninguno, si se necesita de técnicas de procesamiento de datos o si por otro lado no necesita nada y está listo para ser utilizado. En nuestro caso se hicieron adecuaciones para poder trabajar con el dataset, por lo que, si estas modificaciones no hubiesen sido hechas, probablemente el resultado no hubiese sido el esperado.
- Si bien en la entrega 1 se mencionaron algunos modelos que no fueron utilizados en esta entrega final, se cumple con el desempeño deseable en producción que fue estimado con un 90% de accuracy, por lo que el objetivo se cumplió en el desarrollo de la última entrega.