Modelo Machine Learning basado en Diagnóstico para el Otorgamiento de Crédito

Integrantes:

Dovale María
Feü Laura
Palacio Gabriel
Parada Laura

Comisión #25570
Data Science

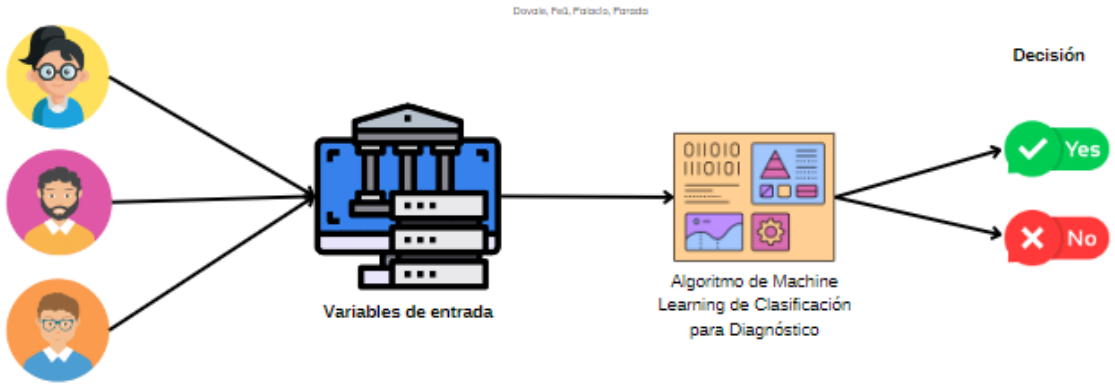
2022

Tabla de contenidos

Descripción del caso

Existe una problemática común en la industria bancaria en cuanto al otorgamiento de crédito para personas naturales, estos usuarios al solicitar un préstamo muchas veces deben esperar por tiempos prolongados para poder obtener una respuesta por parte del banco, en ocasiones la necesidad de conocer esta respuesta es de carácter urgente.

Con el fin de acelerar estos tiempos y semi-automatizar estos procesos permitiendo que sólo los casos especiales sean analizados por expertos en la industria bancaria, hemos decidido crear un modelo para el otorgamiento de créditos bancarios de acuerdo a un scoring basado en las características obtenidas de los datos que alimentará nuestro modelo.



De esta forma la Ciencia de Datos se vuelve una herramienta clave que facilita el proceso de toma de decisiones, en este caso una entidad bancaria que procesa grandes cantidades de información y cuya inmediatez en la toma de esas decisiones brindará mayores y mejores resultados en sus procesos y atención a sus clientes.

Tabla de versionado

Fecha	Rev.	Descripción del cambio	Realizado por
01/06/2022	01	EDA con varios set de datos	Laura Feu
10/06/2022	02	Creación del documento ejecutivo	Laura Parada
17/06/2022	03	Creación del PPT, elección del dataset final.	Maria Dovale y equipo
20/06/2022	04	Análisis Univariado	Equipo
21/06/2022	05	Análisis Bivariado	Equipo
27/06/2022	06	Análisis Multivariado	Equipo
01/07/2022	07	Actualización del PPT	Maria Dovale
04/07/2022	08	Actualización del documento ejecutivo	Laura Parada
06/07/2022	09	Creación del scoring y variable target	María Dovale
13/07/2022	10	Actualización del documento ejecutivo	Laura Parada
24/07/2022	11	Algoritmo elegido para Modelo Machine Learning	Equipo
24/07/2022	12	Actualización del documento ejecutivo	Laura Parada
01/08/2022	13	Actualización del PPT	María Dovale
06/08/2022	14	Optimización del modelo	Equipo
08/08/2022	15	Actualización del PPT	María Dovale
08/08/2022	16	Actualización del documento ejecutivo	Laura Parada
10/08/2022	17	Segundo modelo elegido para Modelo Machine Learning	Equipo
10/08/2022	18	Actualización del PPT	María Dovale
10/08/2022	19	Actualización del documento ejecutivo	Laura Parada
15/08/2022	20	Cross Validation	Laura Feu
15/08/2022	21	Actualización del PPT	Equipo
15/08/2022	22	Actualización del documento ejecutivo	Laura Parada

Objetivo del modelo

Crear un modelo de machine learning que genere un scoring (puntuación) para determinar si un usuario es candidato favorecido o no para recibir un crédito.

Objetivos específicos:

- Analizar varios datasets con información bancaria para elegir uno que cumpla con la mayoría de las variables de entrada más relevantes para los asesores comerciales
- Realizar un EDA al dataset seleccionado
- Realizar un análisis univariado, bivariado y multivariado
- Realizar una clasificación para el diagnóstico de otorgamiento de crédito a partir de datos de entrada
- Brindar la respuesta de otorgamiento del crédito como una variable decisora (YES/NO)

Descripción de los datos

Nuestro paquete de datasets fueron obtenidos del banco de dataset de la página kaggle mediante el uso de **keywords** relacionados al objetivo general, decir, usando palabras como: Machine Learning, Modelo Scoring, riesgo de crédito, otorgamiento de crédito, sistema financiero, de esta forma logramos encontrar 3 potenciales dataset para trabajar.

1. **Bank_df_1.csv (Se cambió el nombre porque fusionamos dos datasets)**
2. **Credit_risk_dataset.csv**
3. **Loan.csv**

Luego se procedió a realizar una exploración de cada uno de ellos para identificar cuál de los tres cumplía con la mayor cantidad de variables relevantes a la hora de hacer un análisis crediticio.

Tomamos cada dataset y vemos cómo está compuesto:

- Estructura del dataframe (rows and columns)
- Conteo del total de registros
- Nombres de las columnas
- Describe (para ver datos estadísticos)
- Tipo de dato
- Verificar si hay datos nulos
- Datos únicos

NÚMERO DATASET	DATASET 1	DATASET 2	DATASET 3
Nombre dataset	Bank_df_1.csv	Credit_risk_dataset.csv	Loan.csv
Datos	438557	32581	148670
Columnas	18	12	37

Posteriormente revisamos las variables más relevantes a la hora de solicitar un crédito y las comparamos con todas las que nos brinda cada dataset, de esta forma definimos con cuál trabajar teniendo en cuenta también la tabla anterior de cómo estaban compuestos y la cantidad de datos brindados.

Para hacer el siguiente análisis tuvimos en cuenta la siguiente bibliografía: Construcción de un modelo de scoring para el otorgamiento de crédito en una entidad financiera el cual sugiere las variables adecuadas a tener en cuenta al momento de tomar una decisión de otorgamiento de crédito.

DATASET 1	DATASET 2	DATASET 3	VARIABLE
X	X	✓	Oficina: ubicación de solicitud de préstamos
X	✓	X	Categoría: Calificación de cada cliente por su historial de crédito.
X	✓	✓	Monto: Volumen del préstamo concedido. 5 categorías
✓	✓	✓	Garantía: Personal/real
X	✓	✓	Reestructurado: 0/1
✓	✓	✓	Edad
✓	X	X	Ocupación
✓	X	X	Nivel educativo
✓	✓	X	Ingreso total
X	X	X	Estrato social
✓	X	X	Antigüedad laboral
✓	X	X	Estado civil
✓	X	✓	Género
✓	X	X	Persona a cargo
✓	✓	X	Tipo de vivienda
✓	X	X	Tipo de contrato
✓	✓	X	Antigüedad en la institución
12	**8**	**6**	**TOTAL**

La tabla comparativa anterior nos permitió establecer según las variables recomendadas por la bibliografía cuál de los tres, era el dataset más adecuado para realizar la clasificación y decisión de otorgamiento de crédito.

Además de la bibliografía mencionada previamente, contamos con la opinión de experto en banca que confirma que ese tipo de datos son los que se suelen obtener al momento en el cual una persona natural aplica para un crédito bancario.

Descripción de las variables del dataset

NOMBRE	DESCRIPCIÓN	COMENTARIOS
ID	Número de cliente	
CODE_GENDER	Género	
FLAG_OWN_CAR	¿Tiene auto?	

FLAG_OWN_REALTY	¿Tiene propiedades?	
CNT_CHILDREN	Cantidad de hijos	
AMT_INCOME_TOTAL	Ingresos anuales	
NAME_INCOME_TYPE	Categoría de Ingresos	
NAME_EDUCATION_TYPE	Nivel de Educación	
NAME_FAMILY_STATUS	Estado civil	Estado civil
NAME_HOUSING_TYPE	¿Dónde vive?	
DAYS_BIRTH	Fecha de nacimiento	Conteo regresivo del día actual hasta el día que nació (-1 significa ayer)
DAYS_EMPLOYED	Fecha de inicio del empleo	Cuenta regresiva desde hoy, Si es positivo, implica que la persona está desempleada
FLAG_MOBIL	¿Tiene teléfono celular?	
FLAG_WORK_PHONE	¿Tiene teléfono corporativo?	
FLAG_PHONE	¿Tiene teléfono?	
FLAG_EMAIL	¿Tiene e-mail?	
OCCUPATION_TYPE	Ocupación	
CNT_FAM_MEMBERS	Tamaño de la familia	

Consideraciones

Dado que algunos datos de nuestro dataset necesitaban algunas transformaciones, se agregaron columnas con dicha transformación, adicional a ello y teniendo en cuenta que no contábamos con una variable TARGET volvimos a conversar con nuestro experto en banca quien sugirió tener en cuenta las siguientes variables que señalamos en **naranja** para realizar un **scoring.** Cada banco tiene scoring diferentes, por los que nos brindó ejemplos de los puntajes que suelen brindarse en base a las mismas para saber si el crédito se otorga o no, lo cual observamos en la variable **APPROVED**

Transformación de variables

NOMBRE	DESCRIPCIÓN	COMENTARIOS
AGE	Edad	Edad calculada en base a DAYS_BIRTH
YEARS_EMPLOYED	Años empleado	Años calculados en base a DAYS_EMPLOYED

Variables importantes dentro del análisis

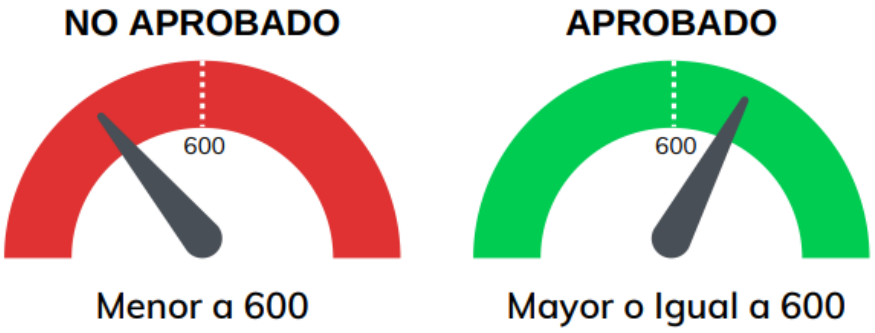
NOMBRE	DESCRIPCIÓN
FLAG_OWN_CAR	¿Tiene auto?
FLAG_OWN_REALTY	¿Tiene propiedades?
AMT_INCOME_TOTAL	Ingresos anuales
NAME_INCOME_TYPE	Categoría de Ingresos
NAME_EDUCATION_TYPE	Nivel de Educación
AGE	Edad
YEARS_EMPLOYED	Años empleado
TOTAL_SCORE	Puntaje total obtenido
APPROVED	¿Crédito aprobado?

Scoring

NOMBRE DE LA VARIABLE	VALUE	SCORE
FLAG_OWN_CAR	Y	100
FLAG_OWN_CAR	N	0
FLAG_OWN_REALTY	Y	100
FLAG_OWN_REALTY	N	0
AMT_INCOME_TOTAL	0	0
AMT_INCOME_TOTAL	< 100.000	100
AMT_INCOME_TOTAL	100.000 & 250.000	150
AMT_INCOME_TOTAL	> 250.000	200
NAME_INCOME_TYPE	Working	150
NAME_INCOME_TYPE	Commercial associate	100
NAME_INCOME_TYPE	State servant	100
NAME_INCOME_TYPE	Pensioner	100
NAME_INCOME_TYPE	Student	0
NAME_EDUCATION_TYPE	Secondary / secondary special	0
NAME_EDUCATION_TYPE	Higher education	50
NAME_EDUCATION_TYPE	Incomplete higher	0
NAME_EDUCATION_TYPE	Lower secondary	50

NAME_EDUCATION_TYPE	Academic degree	100
NAME_HOUSING_TYPE	House / apartment	100
NAME_HOUSING_TYPE	With parents	0
NAME_HOUSING_TYPE	Municipal apartment	50
NAME_HOUSING_TYPE	Rented apartment	50
NAME_HOUSING_TYPE	Office apartment	50
NAME_HOUSING_TYPE	Co-op apartment	50
DAYS_BIRTH	> 21 years	0
DAYS_BIRTH	21 & 60	100
DAYS_BIRTH	> 60	50
DAYS_EMPLOYED	< 1 year	0
DAYS_EMPLOYED	2 & 5 year	50
DAYS_EMPLOYED	5 year	100
DAYS_EMPLOYED	-1001	0

Como ejercicio académico hemos asignado para nuestro banco el siguiente criterio basado en el scoring de una de las entidades bancarias que más publicidad hemos encontrado en redes sociales que brinda tarjetas de crédito en Argentina Naranja X: que si el puntaje obtenido es menor a 600 el crédito no se aprueba, en cambio si es mayor o igual a 600 si se aprueba.



De esta manera cada uno de los clientes puede obtener un puntaje (score) basado en sus datos y así definir si el préstamo es aprobado o no lo cual se ve reflejado en la variable "**APPROVED**". Finalmente el dataset fue renombrado luego de los ajustes realizados y éste será el utilizado para hacer el entrenamiento del algoritmo.

Nombre Dataset	Cantidad de filas	Columnas
bank.csv	438557	22

Hallazgos encontrados por el EDA

Mediante EDA (Exploratory Data Analysis) vamos a organizar y preparar los datos para luego aplicar técnicas estadísticas, también detectaremos fallos en el diseño y recogida de datos, veremos si debemos hacer algún tipo de tratamiento de datos ausentes, así como identificar casos atípicos, entre otros.

Antes de iniciar cargamos todas aquellas librerías con las cuales vamos a trabajar:

1. Cargar librerías

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

2. Cargamos el dataset elegido con las respectivas consideraciones arriba descritas

```
#cargamos el dataset elegido
df = pd.read_csv('/content/bank.csv', sep = ";")
df.head()
```

3. Nos aseguramos que todos los datos NULL hayan quedado efectivamente borrados con una limpieza más

```
df_clean = df.mask(df == 'NULL').dropna()
df_clean
```

De los 438557 luego de la limpieza, nos quedamos con 304354 datos.

4. Limpieza de datos atípicos

Nos percatamos que en la variable CNT_CHILDREN hay datos atípicos que no agregan información de valor a nuestro estudio, procedemos a obviarlos

```
var=df_clean[df_clean["CNT_CHILDREN"]>=6].index
df_cleaner=df_clean.drop(var)
df_cleaner
```

df_cleaner es un dataset "más limpio" donde nos quedan 304332 filas habiendo realizado una limpieza por cantidad de hijos mayor a 6

5. Limpieza de datos duplicados

```
df_cleaner[~ df.duplicated()]
```

Comprobamos que la cantidad de valores en nuestro set de datos no cambia, por lo cual podemos asumir que no hay datos duplicados, comprobémoslo a continuación:

```
df_cleaner.duplicated().sum()
0
```

6. Realizamos un describe para tener una idea general de los datos que trae el dataframe hasta el momento

```
df_cleaner.describe(include='all').T
```

Nos percatamos que la variable AMT_INCOME_TOTAL existen datos atipicos (outliers) que desbalancean nuestra muestra fuertemente, eliminaremos aquellos que superan un salario de 300.000 USD anuales.

```
var=df_cleaner[df_cleaner["AMT_INCOME_TOTAL"]>=300000].index
new_df=df_cleaner.drop(var)
new_df
```

7. Dimensión del Dataset

```
print(new_df.shape)
(268801, 22)
```

8. Traemos los nombres de las columnas

```
columns_uno = new_df.columns
columns_uno
```

```
Index(['ID','CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REALTY','CN
T_CHILDREN','AMT_INCOME_TOTAL','NAME_INCOME_TYPE','NAME_EDUCAT
ION_TYPE','NAME_FAMILY_STATUS','NAME_HOUSING_TYPE','DAYS_BIRTH
','AGE','DAYS_EMPLOYED','YEARS_EMPLOYED','FLAG_MOBIL','FLAG_WO
RK_PHONE','FLAG_PHONE','FLAG_EMAIL','OCCUPATION_TYPE','CNT_FAM
_MEMBERS','TOTAL_SCORE','APPROVED'],dtype='object')
```

Como resultado final de nuestro data wrangling obtuvimos un nuevo dataframe "new_df" con el 61,3% de los datos iniciales con los que consideramos que podremos trabajar el modelo más eficientemente.

El procedimiento será hacer una tabla estilo dataframe con todas las columnas donde vamos a obtener:

- Cantidad de Datos
- Tipo de dato
- Valores únicos
- Valores nulos
- Promedio
- Máximo
- Mínimo
- Quartiles
- Sesgo y kurtosis

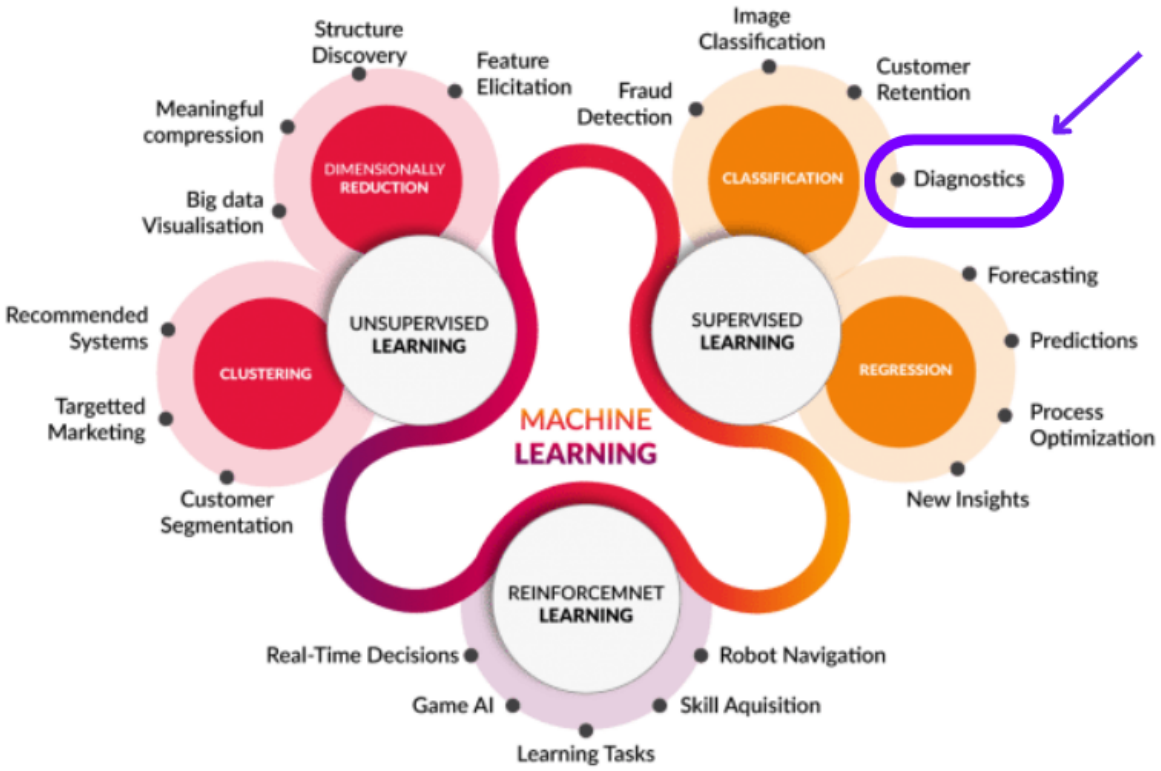
Con toda esta información podremos obtener cierta relación donde determinemos variables numéricas o categóricas, cuales tienen mucha variabilidad, promedio y medidas centrales.

```
from pandas.api.types import is_numeric_dtype

def info_univariado(new_df):
    df_info = pd.DataFrame(columns=['Cantidad', 'Tipo' , 'Missing', 'Unicos', 'Numeric'])
  for col in new_df:
    data_series = new_df[col]
                df_info.loc[col] = [data_series.count(), data_series.dtype, data_series.isnull().sum(), data_series.nunique(), is_numeric_dtype(data_series)]
      df_describe = new_df.describe(include='all').T[['top', 'mean', 'std', 'min', '25%', '50%', '75%', 'max']]
                df_stats = pd.DataFrame([new_df.skew(), new_df.kurtosis()], index=['sesgo', 'kurt']).T
  return pd.concat([df_info,pd.concat([df_describe, df_stats], axis=1)], axis=1).fillna('0')

df_uni_stats = info_univariado(new_df)
df_uni_stats
```

Algoritmo Elegido

Una vez analizado, limpiado y estructurados los datos se elegirá el algoritmo de Machine Learning. Se eliminarán los campos que no sirven para hacer la predicción y organizarlos adecuadamente para que el modelo no reciba información que no le es útil y que podría provocar predicciones de poca calidad o confianza.

El algoritmo elegido es de Aprendizaje Supervisado cuyo objetivo es realizar una Clasificación para el *Diagnóstico para el otorgamiento de créditos bancarios*.



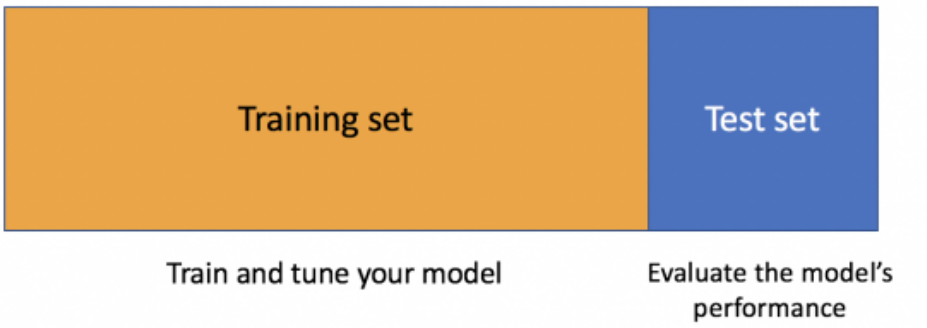
A partir de este momento, enseñaremos o entrenaremos al algoritmo teniendo en cuenta los datos que ya vienen etiquetados con la respuesta correcta, en este caso la variable TARGET.

Lo ideal en esta práctica es que cuanto mayor es el conjunto de datos más el algoritmo puede aprender sobre el tema; una vez concluído el entrenamiento, se le brindan nuevos datos, ya sin las etiquetas de las respuestas correctas, y el algoritmo de aprendizaje utiliza la experiencia pasada que adquirió durante la etapa de entrenamiento para predecir un resultado[1].

Validar un modelo de aprendizaje es esencial en la práctica y para poder evaluarlo correctamente, hay que realizar “split de datos”, es decir, separar nuestro dataset

[1] https://iaarbook.github.io/ML/
Definición de aprendizaje supervisado

original en “Datos de Entrenamiento”, que serán usados justamente para entrenar a nuestro modelo y en “Datos de Test o de Testing” que serán aquellos datos que utilizaremos para evaluar la performance de nuestro modelo.



Antes de iniciar cargamos todas aquellas librerías con las cuales vamos a trabajar:

1. Cargar librerías

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,
classification_report
import seaborn as sns
sns.set()
from sklearn.preprocessing import OneHotEncoder
```

2. Realiza un conteo de los valores que tenemos en la variable target

```
new_df.APPROVED.value_counts()
YES    221486
NO      47315
Name: APPROVED, dtype: int64
```

3. Separamos los datos en el 80% para entrenamiento y el 20% para test y separamos la variable target en el 80% para entrenamiento y el 20% para test.

```
# Separamos los datos en el 80% para entrenamiento y el 20%
para test
x_train = new_df[:-53760]
x_test = new_df[-53760:]
```

```
# Separamos la variable target en el 80% para entrenamiento y
el 20% para test
y_train = new_df.APPROVED[:-53760]
y_test = new_df.APPROVED[-53760:]
```

Entrenamiento

Preprocesamiento de variables categóricas a numéricas: En nuestro dataset tenemos variables como CODE_GENDER, FLAG_OWN_CAR, FLAG_OWN REALTY, NAME_INCOME_TYPE, NAME_EDUCATION_TYPE, NAME_HOUSING_TYPE como variables con string.

En algunos casos se eliminarán algunas de las variables y se realizará One Hot Encoding - Procesamiento.

1. Limpieza

Se eliminarán aquellas variables que no son relevantes para el TARGET como son: CODE_GENDER, NAME_FAMILY_STATUS, OCCUPATION_TYPE, DAYS_BIRTH, DAYS_EMPLOYED. Algunas de estas porque ya fueron transformadas y otras porque no son relevantes para nuestro TARGET.

```
x_train = x_train.drop('CODE_GENDER', axis=1)
x_train
```

```
x_train = x_train.drop('NAME_FAMILY_STATUS', axis=1)
x_train
```

```
x_train = x_train.drop('OCCUPATION_TYPE', axis=1)
x_train
```

```
x_train = x_train.drop('DAYS_BIRTH', axis=1)
x_train
```

```
x_train = x_train.drop('DAYS_EMPLOYED', axis=1)
x_train
```

2. Encoding

```
from sklearn.preprocessing import OneHotEncoder

#generate one hot encoder class
encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
#ponemos ignore para poder omitir cuando puede pasar que no exista los mismos valores
#  generarlo como. matrix para poder generar las columnas adecuadas
#entrenamiento para generar el preprocesamiento one hot encoding para las variables indicadas
encoder.fit(x_train[['FLAG_OWN_CAR',
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE']])
```

```
#transformo en train datasets en base a la clase generada
cat_encoding =
pd.DataFrame(encoder.transform(x_train[['FLAG_OWN_CAR',
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE']]),
columns=encoder.get_feature_names(['FLAG_OWN_CAR',
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE']))
cat_encoding
```

```
#joineo de las variables que deseo con categóricas encoding
df_all_train = pd.concat([x_train, cat_encoding], axis=1)
df_all_train
```

```
#Borrado de columnas innecesarias
df_all_train.drop(['FLAG_OWN_CAR',
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE'],axis=1, inplace=True)
```

```
x_train = df_all_train.loc[:,df_all_train.columns !=
'APPROVED']
```

```
#train datasets final
df_all_train
```

Acá queda listo nuestro dataset para iniciar el entrenamiento en el modelo elegido

Entrenamiento del Modelo

Se inició el primer intento con el modelo “Árboles de decisión”, los Árboles de Decisión son diagramas con construcciones lógicas, muy similares a los sistemas de predicción basados en reglas, que sirven para representar y categorizar una serie de condiciones que ocurren de forma sucesiva, para la resolución de un problema. Los Árboles de Decisión están compuestos por nodos internos, nodos terminales y ramas que emanan de los nodos interiores. Cada nodo interior en el árbol contiene una prueba de un atributo, y cada rama representa un valor distinto del atributo. Siguiendo las ramas desde el nodo raíz hacia abajo, cada ruta finalmente termina en un nodo terminal creando una segmentación de los datos.[2]

1. Spliteo de datos en target y features for test and train

```
#selección de x values sin el target
x_train = df_all_train.loc[:,df_all_train.columns != 'APPROVED']

#selección del target
y_train = df_all_train.APPROVED
```

```
#training del modelo
from sklearn import tree

model_default = tree.DecisionTreeClassifier(random_state = 5050)
# con hiperparametros default
model_default.fit(x_train, y_train)
```

```
DecisionTreeClassifier(random_state=5050)
```

[2] https://iaarbook.github.io/ML/
Definición de Árboles de Decisión

```
#modelo generado con algún hiper parámetro diferente
model_max_depth  =  tree.DecisionTreeClassifier(random_state  =
5050, max_depth=10)
model_max_depth.fit(x_train, y_train)
```

```
DecisionTreeClassifier(max_depth=10, random_state=5050)
```

Predicción

Ahora transformar los datos del test de la misma manera que el train

1. Limpieza

```
x_test = x_test.drop('CODE_GENDER', axis=1)
x_test
```

```
x_test = x_test.drop('NAME_FAMILY_STATUS', axis=1)
x_test
```

```
x_test = x_test.drop('OCCUPATION_TYPE', axis=1)
x_test
```

```
x_test = x_test.drop('DAYS_BIRTH', axis=1)
x_test
```

```
x_test = x_test.drop('DAYS_EMPLOYED', axis=1)
x_test
```

2. Encoding

```
x_test.count()
```

```
#one hot encoding
cat_encoding_test                                                =
pd.DataFrame(encoder.transform(x_test[['FLAG_OWN_CAR',
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE']]),
columns=encoder.get_feature_names(['FLAG_OWN_CAR',
```

```
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE']))
cat_encoding_test
```

```
#joineo de las variables que deseo con categóricas encoding
df_all_test = pd.concat([x_test, cat_encoding_test], axis=1)
df_all_test
```

```
#borrado de las mismas variables
df_all_test.drop(['FLAG_OWN_CAR',
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE'],axis=1, inplace=True)
```

```
df_all_test
```

```
#prediccion
y_pred_test = model_max_depth.predict(df_all_test)
y_pred_test
```

Métricas de desempeño

En este caso utilizaremos el Accuracy como métrica para determinar que un modelo es mucho mejor que otro.

```
y_test
```

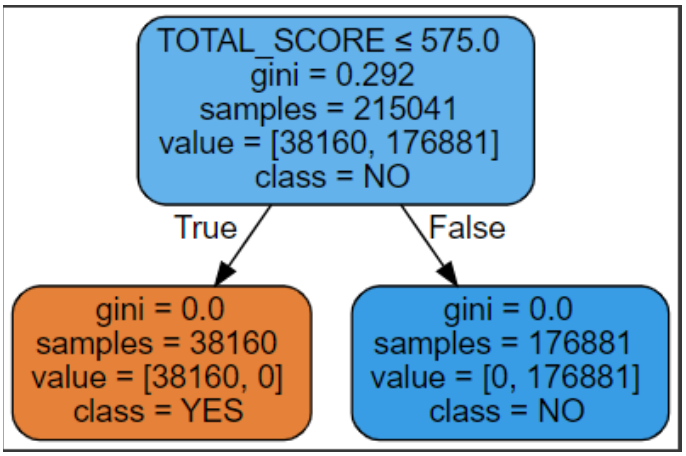
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

#accuracy
print('Accuracy score for test data is:', accuracy_score(y_test,y_pred_test))

confusion_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred_test))

confusion_matrix.index = ['NO APROBADO','APROBADO']
confusion_matrix.columns = ['PREDICCION NO APROBADO','PREDICCION APROBADO']
print(confusion_matrix)
```

```
Accuracy score for test data is: 1.0
             PREDICCION NO APROBADO  PREDICCION APROBADO
NO APROBADO                    9155                    0
APROBADO                          0                44605
```

```
import pydotplus #pip install pydotplus
from sklearn.tree import export_graphviz

def tree_graph_to_png(tree, feature_names, class_names, png_file_to_save):
    tree_str = export_graphviz(
        tree,
        out_file=None,
        feature_names=feature_names,
        class_names=class_names,
        rounded=True,
        filled=True
```

```
    )
    graph = pydotplus.graph_from_dot_data(tree_str)
    graph.write_png(png_file_to_save)

#lista de variables
list_features = [x for x in df_all_train.columns if x !=
'APPROVED']
tree_graph_to_png(model_max_depth, list_features,
df_all_train.APPROVED.unique().astype(str), 'model.png')
```

```
import graphviz
from sklearn.tree import DecisionTreeClassifier,
export_graphviz
list_features = [x for x in df_all_train.columns if x !=
'APPROVED']
data =
export_graphviz(model_default,out_file=None,feature_names=list
_features,class_names=df_all_train.APPROVED.unique().astype(st
r),
                        filled=True, rounded=True,
                        special_characters=True)
graph = graphviz.Source(data)
graph
```



Las métricas obtenidas en este primer intento fueron perfectas, lo cual nos resultó sospechoso, la razón es que el modelo estaba aprendiendo que si el score era mayor a 575.0 se le otorgaría el crédito, de lo contrario se le denegará.

En este primer intento obtuvimos un árbol de decisión de una sola rama, vemos que la variable por la cual estaba tomando la decisión era "TOTAL_SCORE" sin tener en cuenta ninguna variable más, esto nos condujo a métricas perfectas y nos percatamos que esta variable no debería ser parte del entrenamiento por lo que la eliminamos para volver a entrenar el modelo.

Iteraciones de Optimización 1 - Entrenamiento del Modelo

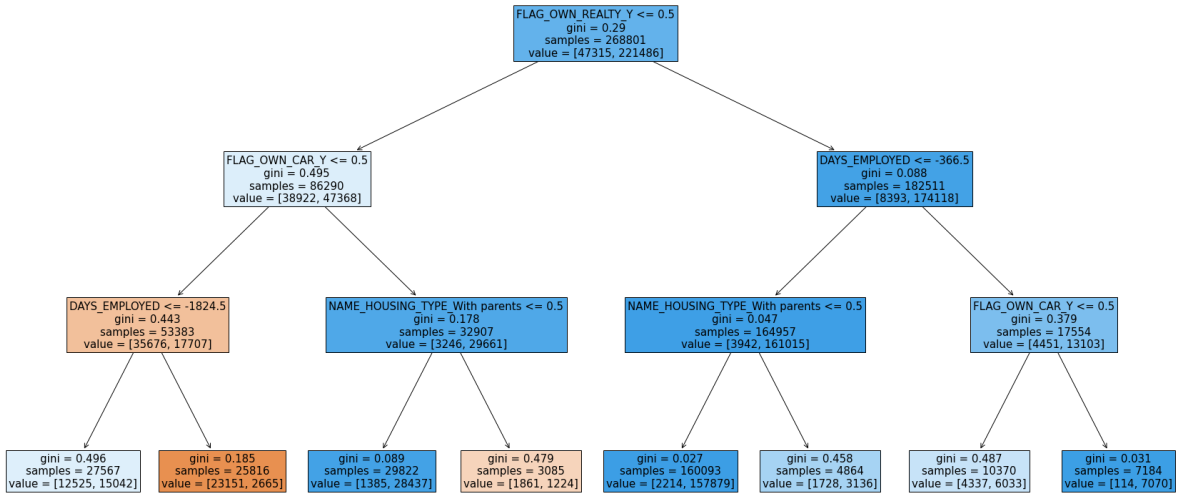
Como parte del proceso de aprendizaje entrenamos nuevamente nuestro modelo sin tener en cuenta la columna "SCORE" para que no aprenda de la misma, sino que use las otras variables para aprender, también logramos que tome todas las otras variables para entrenar el modelo y así lograr un árbol de decisión con múltiples ramas como veremos a continuación:

```
#Modificamos el hiper parámetro max_depth=3 para podar nuestro árbol
model = DecisionTreeClassifier(max_depth=3)
```

```
model.fit(X=explicativas, y=target)
```

```
DecisionTreeClassifier(max_depth=3)
```

```
plt.figure(figsize=(30,15))
plot_tree(decision_tree=model,
feature_names=explicativas.columns, filled=True, fontsize=15);
```



La primera variable que nuestro modelo tiene en cuenta es "FLAG_OWN_REALTY", si la misma es menor o igual a 0.5 la condición se cumple (la persona no tiene propiedad) y es verdadera por lo que siguiente variable que tiene en cuenta es "FLAG_OWN_CAR" si es falsa (la persona si tiene propiedad) la siguiente variable que toma es "DAYS_EMPLOYED". Evaluemos estas dos posibilidades:

- Evaluemos la variable "FLAG_OWN_CAR", si la persona no tiene auto, esta es verdadera por lo que se evaluaría a continuación "DAYS_EMPLOYED". Si

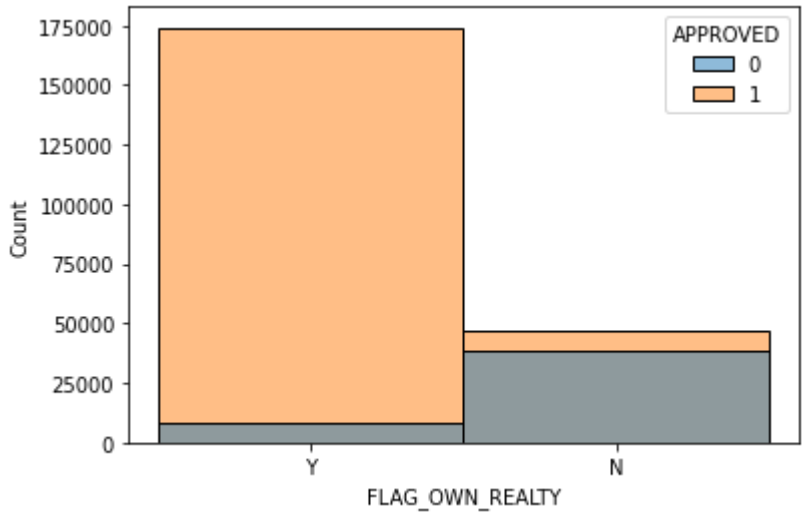
por el contrario, es falsa (si tiene auto) se evalúa a continuación "NAME_HOUSING_TYPE"

- Evaluemos la variable "DAYS_EMPLOYED", si los días que ha estado empleado son menores o iguales a -366.5 (más de 1 año) la condición se cumple, por ende, es verdadera y se procede a evaluar la variable "NAME_HOUSING_TYPE_With parents" de lo contrario se evalúa "FLAG_OWN_CAR_Y".

Interpretación del nodo raíz

Como vemos, el nodo raíz elegido por el modelo está formado por la variable "FLAG_OWN_REALTY".

```
sns.histplot(x=new_df.FLAG_OWN_REALTY, hue=new_df.APPROVED)
```



Quisimos graficar la influencia que ejerce la misma sobre nuestra variable target para validar su peso, para esto hacemos un análisis bivariado donde podemos comprobar a través de este histograma que aquellos que cuentan con propiedad tienen más probabilidad de que se les apruebe un crédito que aquellos que no.

Métricas de desempeño

Realizamos una suma entre la variable target y la de predicción (toma el true = 1 y el false = 0) por lo que tenemos una cantidad de 242609 aciertos que predijeron de forma correcta el modelo.

```
(dummy['APPROVED']==dummy['PREDICCION']).sum()
```

```
242609
```

Al hacer la suma entre la variable target y la de predicción toma el true = 1 y el false = 0 por lo que tenemos una cantidad de 242609 aciertos que predijeron de forma correcta el modelo.

```
#Dividimos entre el total de datos para obtener el porcentaje de acierto
(dummy['APPROVED']==dummy['PREDICCION']).sum()/268801
```

```
0.9025598863099468
```

Podemos confirmar que nuestro modelo tiene una precisión del 90.25%

Vemos que ahora si el resultado que nos marca el accuracy de nuestro modelo es basado en el aprendizaje de todas las variables del dataset, en lugar de tomar sólo una como nos ocurrió la primera vez.

Iteraciones de Optimización 2 - Entrenamiento del Modelo

En esta segunda iteración volvimos a entrenar nuestro modelo con las respectivas correcciones y en espera que mejore el accuracy.

```
#spliteo de datos en target y features for test and train

#seleccion de x values sin el target
x_train = x_train
x_train     =     df_all_train.loc[:,df_all_train.columns     !=
'APPROVED']

#seleccion del target
y_train = df_all_train.APPROVED
```

```
#training del modelo
from sklearn import tree

model_default   =   tree.DecisionTreeClassifier(random_state   =
5050) # con hiperparametros default
model_default.fit(x_train, y_train)
```

```
#modelo generado con algun hiperparametro diferente
model_max_depth  =  tree.DecisionTreeClassifier(random_state  =
5050, max_depth=10)
model_max_depth.fit(x_train, y_train)
```

Predicción

Ya tenemos entrenado el modelo , lo que debemos hacer es ahora transformar los datos del test de la misma manera que el train

Limpieza

```
x_test = x_test.drop('CODE_GENDER', axis=1)
x_test
```

```
x_test = x_test.drop('NAME_FAMILY_STATUS', axis=1)
x_test
```

```
x_test = x_test.drop('OCCUPATION_TYPE', axis=1)
x_test
```

```
x_test = x_test.drop('DAYS_BIRTH', axis=1)
x_test
```

```
x_test = x_test.drop('DAYS_EMPLOYED', axis=1)
x_test
```

Encoding

```
x_test.count()
```

```
#one hot encoding
cat_encoding_test                                                    =
pd.DataFrame(encoder.transform(x_test[['FLAG_OWN_CAR',
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE']]),
columns=encoder.get_feature_names(['FLAG_OWN_CAR',
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE']))
cat_encoding_test
```

```
#visualizamos este df para saber donde comienza el indice y
conocer si debemos resetarlo
x_test
```

```
#reseteo del index
x_test.reset_index(drop=True, inplace=True)
```

```
#comprobación
x_test
```

```
#joineo de las variables que deseo con categóricas encoding
df_all_test = pd.concat([x_test, cat_encoding_test], axis=1)
df_all_test
```

```
#borrado de las mismas variables
df_all_test.drop(['FLAG_OWN_CAR',
'FLAG_OWN_REALTY','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE',
'NAME_HOUSING_TYPE'],axis=1, inplace=True)
```

```
df_all_test
```

```
#prediccion
```

```
y_pred_test = model_max_depth.predict(df_all_test)
y_pred_test
array([0, 0, 0, ..., 1, 0, 1])
```

Métricas

En este caso utilizaremos el Accuracy como métrica para determinar que un modelo es mucho mejor que otro

```
y_test
```

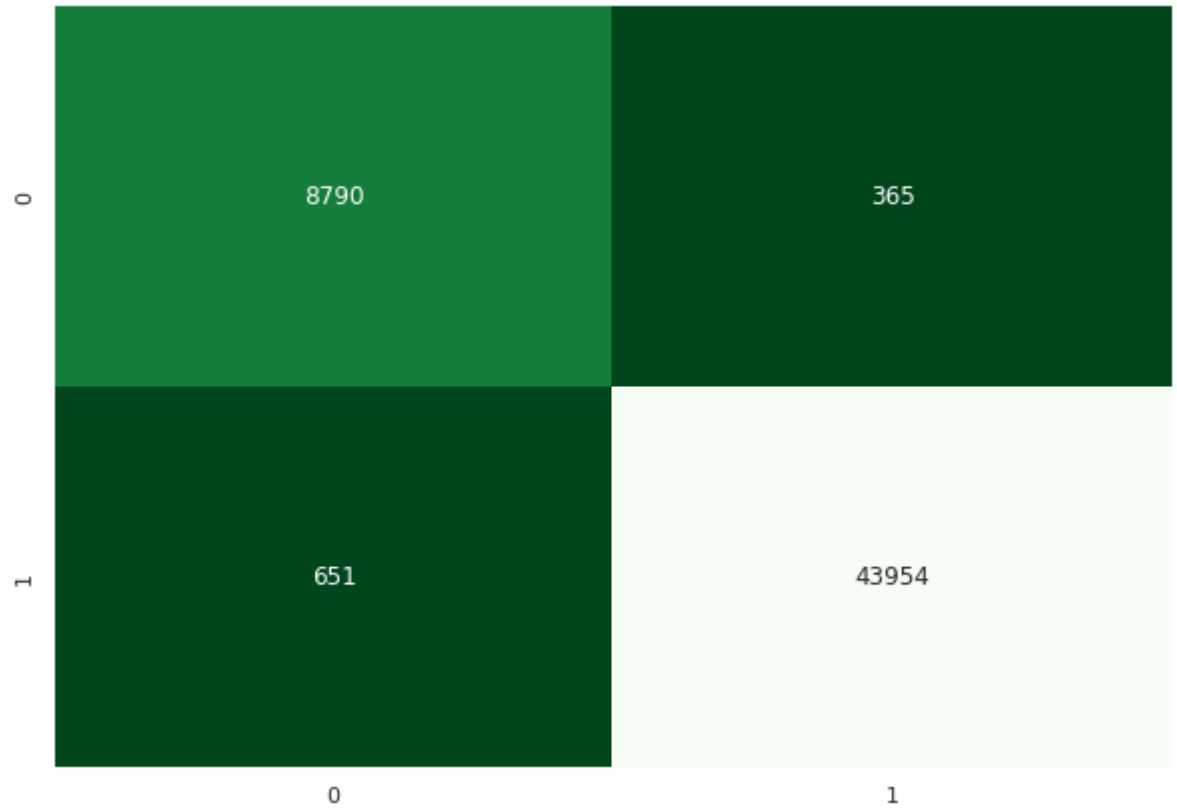
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

#accuracy
print('Accuracy score for test data is:', accuracy_score(y_test,y_pred_test))

confusion_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred_test))

confusion_matrix.index = ['NO APROBADO','APROBADO']
confusion_matrix.columns = ['PREDICCION NO APROBADO','PREDICCION APROBADO']
print(confusion_matrix)
```

```
Accuracy score for test data is: 0.9811011904761905
              PREDICCION NO APROBADO  PREDICCION APROBADO
NO APROBADO                     8790                  365
APROBADO                         651                43954
```

Conclusión

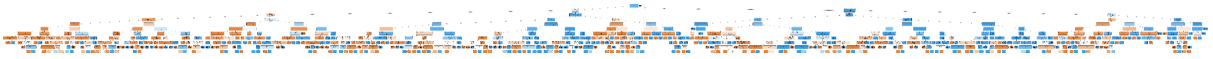
Podemos notar que tenemos un Accuracy de 98,11%. Podemos notar que la cantidad de falsos positivos es de 365 y los falsos negativos es de 651

```
#Matriz De Confusión
from sklearn.metrics import confusion_matrix, classification_report
cm = confusion_matrix (y_test, y_pred_test)
fig, ax = plt.subplots(figsize=(10,7))
sns.heatmap(cm, annot=True, cmap='Greens_r', fmt='2g', cbar=None)
```




```
import pydotplus #pip install pydotplus
from sklearn.tree import export_graphviz

def    tree_graph_to_png(tree,    feature_names,    class_names,
png_file_to_save):
    tree_str = export_graphviz(
        tree,
        out_file=None,
        feature_names=feature_names,
        class_names=class_names,
        rounded=True,
        filled=True
    )
    graph = pydotplus.graph_from_dot_data(tree_str)
    graph.write_png(png_file_to_save)

#lista de variables
list_features  =  [x  for  x  in  df_all_train.columns  if  x  !=
'APPROVED']
tree_graph_to_png(model_max_depth,                list_features,
df_all_train.APPROVED.unique().astype(str), 'model.png')
```

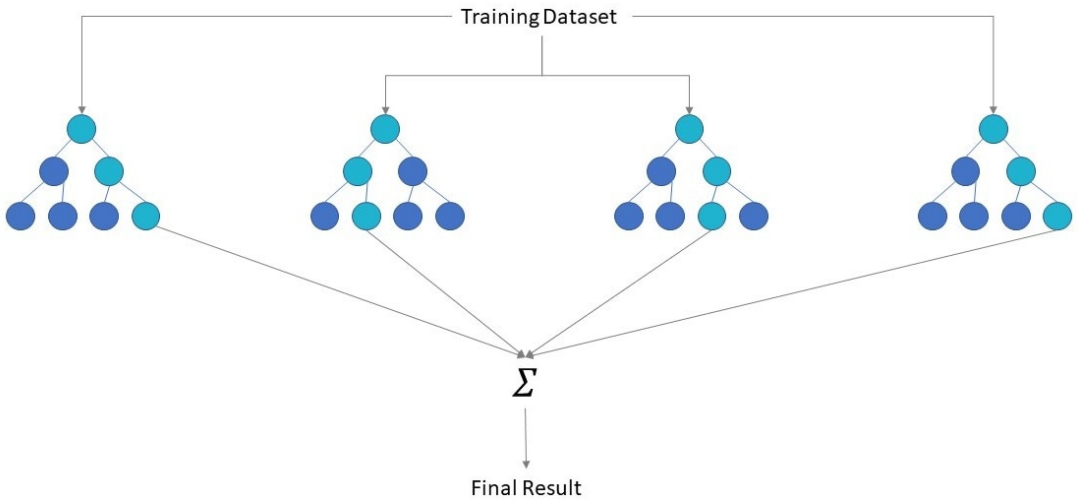
```
import graphviz
from sklearn.tree import DecisionTreeClassifier,
export_graphviz
list_features = [x for x in df_all_train.columns if x !=
'APPROVED']
data =
export_graphviz(model_default,out_file=None,feature_names=list
_features,class_names=df_all_train.APPROVED.unique().astype(st
r),
                filled=True, rounded=True,
                special_characters=True)
graph = graphviz.Source(data)
graph
```



Segundo algoritmo - Modelo Random Forest (Bosque aleatorio)

Random forest es un algoritmo de aprendizaje automático de uso común que combina la salida de múltiples árboles de decisión para llegar a un único resultado. Su facilidad de uso y flexibilidad han impulsado su adopción, ya que maneja problemas de clasificación y regresión.

Si bien los árboles de decisión son algoritmos de aprendizaje supervisado comunes, pueden ser propensos a problemas, como sesgos y sobre ajustes. Sin embargo, cuando varios árboles de decisión forman un conjunto en el algoritmo de bosque aleatorio, predicen resultados más precisos, especialmente cuando los árboles individuales no están correlacionados entre sí.[3]



Los requisitos previos para que el bosque aleatorio funcione bien son:

1. Es necesario que haya alguna señal real en nuestras funciones para que los modelos creados con esas funciones funcionen mejor que las conjeturas aleatorias.
2. Las predicciones, y por lo tanto los errores realizados por los árboles individuales deben tener bajas correlaciones entre sí.

El segundo algoritmo escogido para poder correr, entrenar el modelo y hacerlo más robusto lo vamos a correr con diferentes n_stimators de manera de poder notar las diferencias y hacer mejores conclusiones.

```
RF n_stimator = 1000
```

[3] https://www.ibm.com/cloud/learn/random-forest
Definición de Random Forest

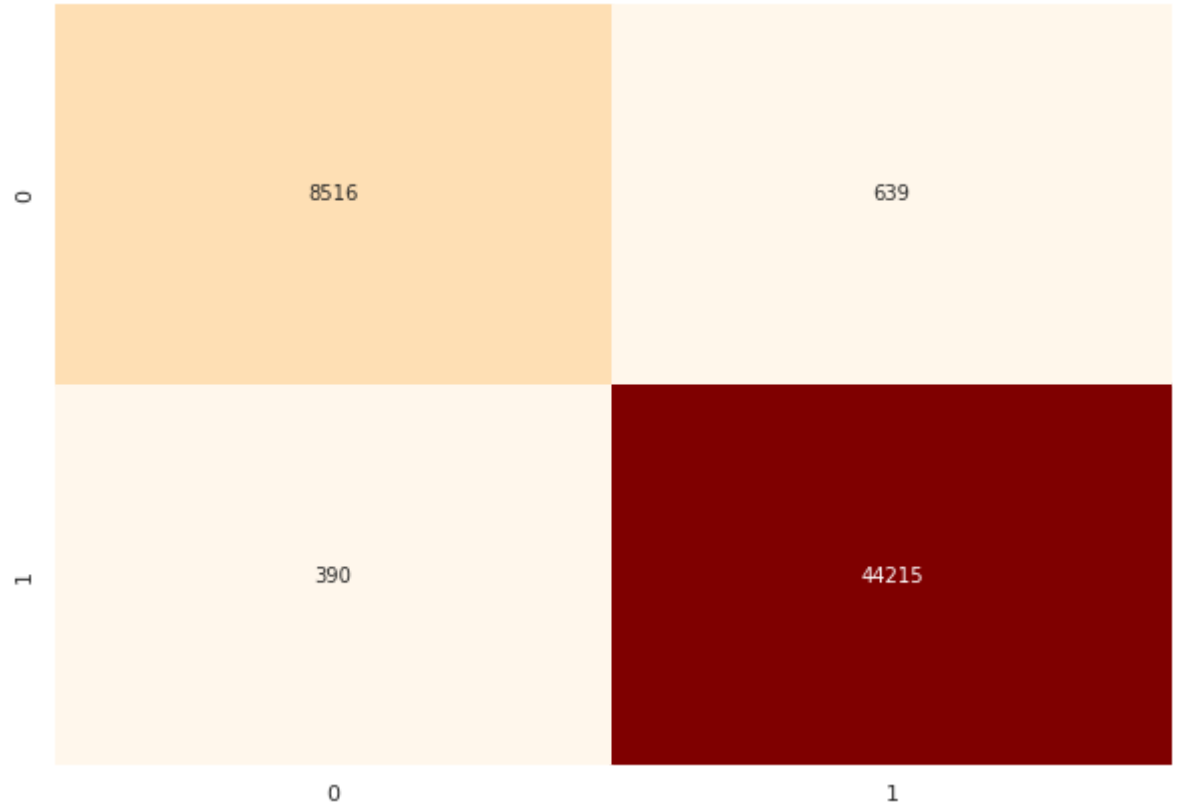
```
#Creamos un random forest! random_state = semilla con
n_stimators=1000
RFmodel = RandomForestClassifier(random_state=5050,
n_estimators=1000, class_weight="balanced",
max_features="log2")
RFmodel.fit(x_train, y_train)
RandomForestClassifier(class_weight='balanced',
max_features='log2',
                       n_estimators=1000, random_state=5050)
```

```
RFmodel.score(df_all_test, y_test) # n_estimators=1000
0.980859375
```

```
#Métricas de Desempeño n=1000
#En este caso ultilizaremos el Accuracy como metrica para
determinar que un modelo es mucho mejor que otro

y_predicted = RFmodel.predict(df_all_test)
y_predicted
array([0, 0, 0, ..., 1, 0, 1])
```

```
from sklearn.metrics import confusion_matrix
cm2 = pd.DataFrame(confusion_matrix(y_test, y_predicted))

cm2.index = ['NO APROBADO','APROBADO']
cm2.columns = ['PREDICCION NO APROBADO','PREDICCION APROBADO']
print(cm2)
from sklearn.metrics import accuracy_score

#accuracy
print('Accuracy score for test data is:',
accuracy_score(y_test, y_predicted))
            PREDICCION NO APROBADO  PREDICCION APROBADO
NO APROBADO                   8516                  639
APROBADO                       390                44215
Accuracy score for test data is: 0.980859375
```

```
#Matriz De Confusión con n_estimators=500
from sklearn.metrics import confusion_matrix,
classification_report
cm = confusion_matrix (y_test, y_predicted)
fig, ax = plt.subplots(figsize=(10,7))
sns.heatmap(cm, annot=True, cmap='OrRd', fmt='2g', cbar=None);
```



RF n_stimators = 500

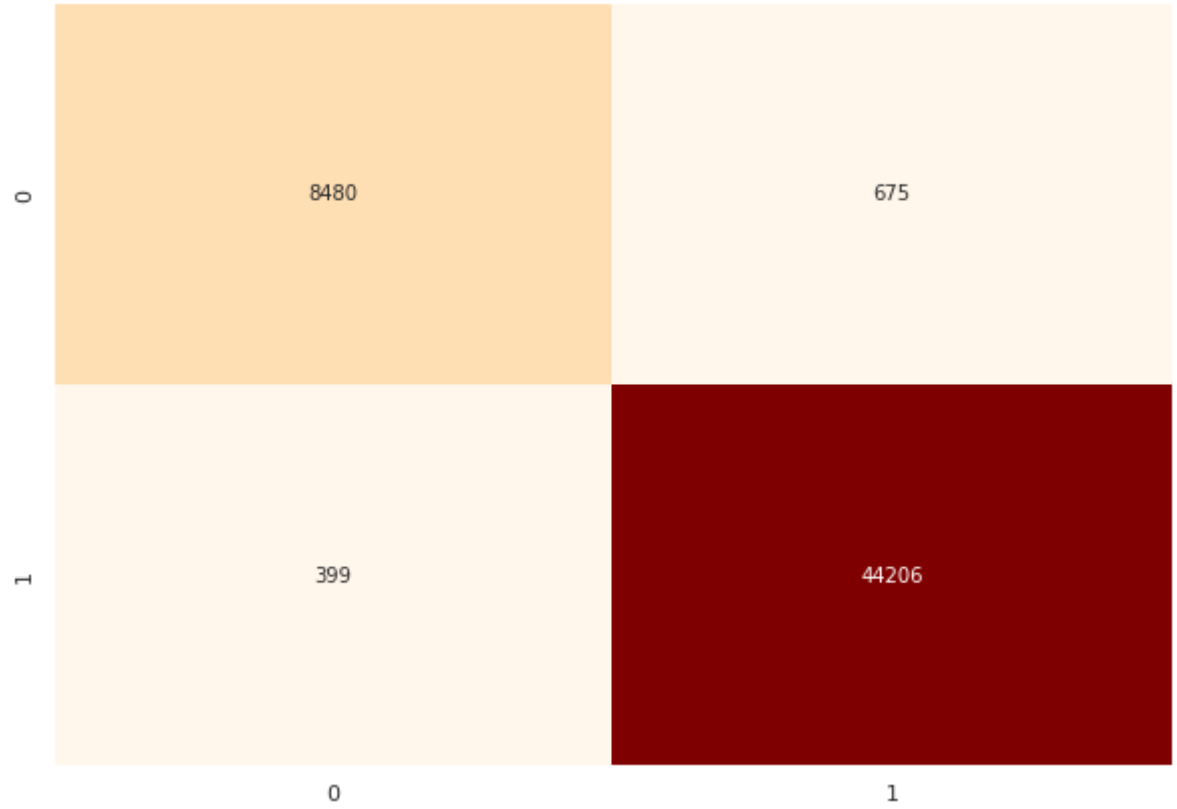
```
#Creamos un random forest! random_state = semilla
RFmodel = RandomForestClassifier(random_state=5050,
n_estimators=500, class_weight="balanced",
max_features="log2")
RFmodel.fit(x_train, y_train)
RandomForestClassifier(class_weight='balanced',
max_features='log2',
                       n_estimators=500, random_state=5050)
```

```
RFmodel.score(df_all_test, y_test) # n_estimators=500
0.9800223214285714
```

```
#Métricas de Desempeño n=500
#En este caso ultilizaremos el Accuracy como metrica para
determinar que un modelo es mucho mejor que otro

y_predicted = RFmodel.predict(df_all_test)
y_predicted
array([0, 0, 0, ..., 1, 0, 1])
```

```
from sklearn.metrics import confusion_matrix
cm2 = pd.DataFrame(confusion_matrix(y_test, y_predicted))

cm2.index = ['NO APROBADO','APROBADO']
cm2.columns = ['PREDICCION NO APROBADO','PREDICCION APROBADO']
print(cm2)
from sklearn.metrics import accuracy_score

#accuracy
print('Accuracy score for test data is:',
accuracy_score(y_test, y_predicted))
            PREDICCION NO APROBADO  PREDICCION APROBADO
NO APROBADO                   8480                  675
APROBADO                       399                44206
Accuracy score for test data is: 0.9800223214285714
```

```
#Matriz De Confusión con n_estimators=500
from sklearn.metrics import confusion_matrix,
classification_report
cm = confusion_matrix (y_test, y_predicted)
fig, ax = plt.subplots(figsize=(10,7))
sns.heatmap(cm, annot=True, cmap='OrRd', fmt='2g', cbar=None)
```



RF n_stimators = 50

```
#Creamos un random forest! random_state = semilla
RFmodel = RandomForestClassifier(random_state=5050,
n_estimators=50, class_weight="balanced", max_features="log2")
RFmodel.fit(x_train, y_train)
RandomForestClassifier(class_weight='balanced',
max_features='log2',
                       n_estimators=50, random_state=5050)
```

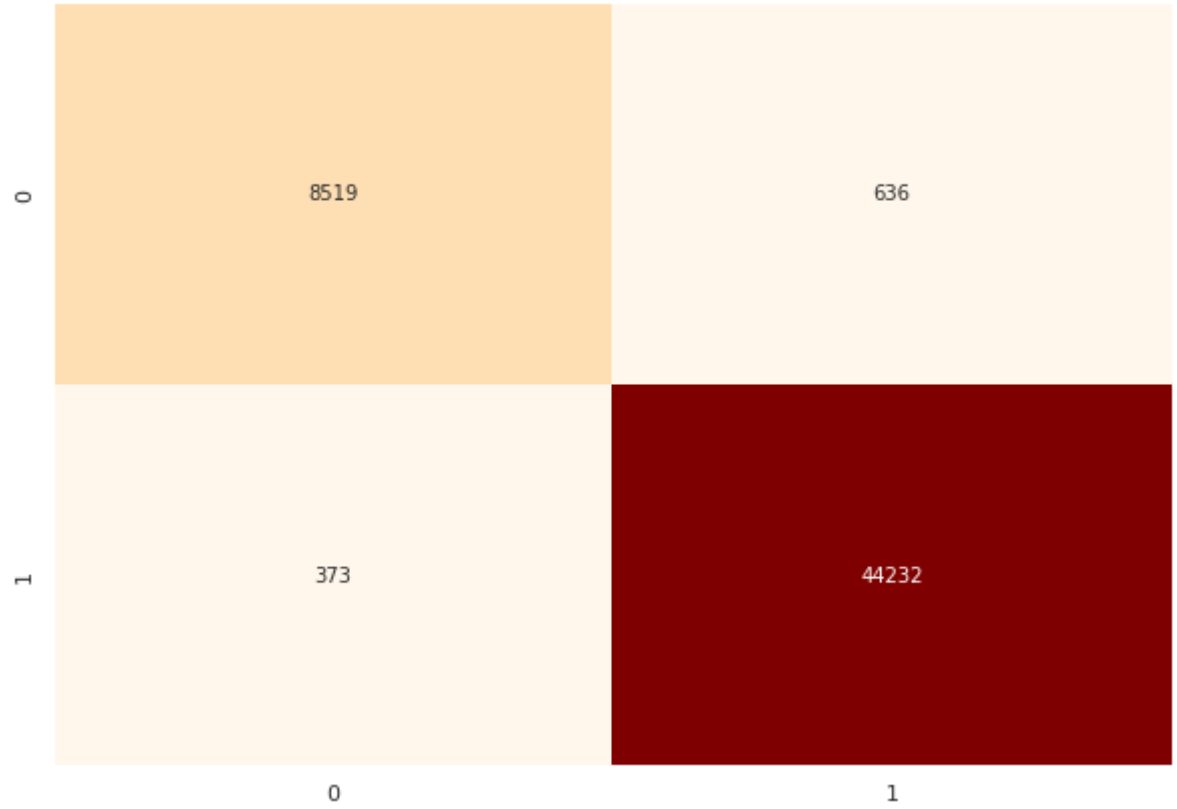
```
RFmodel.score(df_all_test, y_test) # n_estimators=50
0.9812313988095238
```

```
#Métricas de Desempeño n=50
#En este caso ultilizaremos el Accuracy como metrica para
determinar que un modelo es mucho mejor que otro
y_predicted = RFmodel.predict(df_all_test)
y_predicted
array([0, 0, 0, ..., 1, 0, 1])
```

```
from sklearn.metrics import confusion_matrix
cm2 = pd.DataFrame(confusion_matrix(y_test, y_predicted))

cm2.index = ['NO APROBADO','APROBADO']
cm2.columns = ['PREDICCION NO APROBADO','PREDICCION APROBADO']
print(cm2)
from sklearn.metrics import accuracy_score

#accuracy
print('Accuracy score for test data is:',
accuracy_score(y_test, y_predicted))
```

```
             PREDICCION NO APROBADO  PREDICCION APROBADO
NO APROBADO                    8519                  636
APROBADO                        373                44232
Accuracy score for test data is: 0.9812313988095238
```

```
#Matriz De Confusión con n_estimators=500
from sklearn.metrics import confusion_matrix,
classification_report
cm = confusion_matrix (y_test, y_predicted)
fig, ax = plt.subplots(figsize=(10,7))
sns.heatmap(cm, annot=True, cmap='OrRd', fmt='2g', cbar=None);
```



RF n_stimators = 5

```
#Creamos un random forest! random_state = semilla
RFmodel = RandomForestClassifier(random_state=5050,
n_estimators=5, class_weight="balanced", max_features="log2")
RFmodel.fit(x_train, y_train)
RandomForestClassifier(class_weight='balanced',
max_features='log2',
                       n_estimators=5, random_state=5050)
```

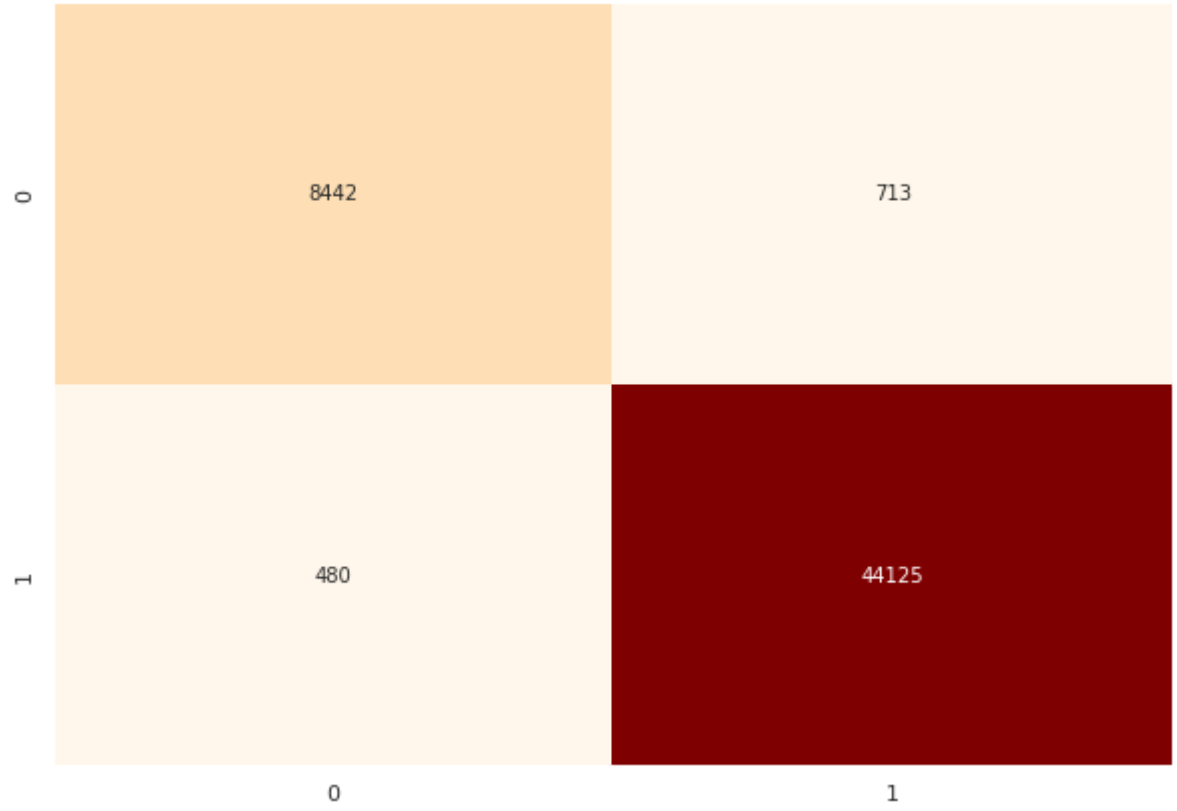
```
RFmodel.score(df_all_test, y_test) #n_estimators=5
0.9778087797619047
```

```
#Métricas de Desempeño n=5
#En este caso ultilizaremos el Accuracy como metrica para
determinar que un modelo es mucho mejor que otro
y_predicted = RFmodel.predict(df_all_test)
y_predicted
array([0, 0, 0, ..., 1, 0, 1])
```

```
from sklearn.metrics import confusion_matrix
cm2 = pd.DataFrame(confusion_matrix(y_test, y_predicted))

cm2.index = ['NO APROBADO','APROBADO']
cm2.columns = ['PREDICCION NO APROBADO','PREDICCION APROBADO']
print(cm2)
from sklearn.metrics import accuracy_score

#accuracy
print('Accuracy score for test data is:',
accuracy_score(y_test, y_predicted))
```

```
             PREDICCION NO APROBADO  PREDICCION APROBADO
NO APROBADO                    8442                  713
APROBADO                        480                44125
Accuracy score for test data is: 0.9778087797619047
```

```
#Matriz De Confusión con n_estimators=500
from sklearn.metrics import confusion_matrix,
classification_report
cm = confusion_matrix (y_test, y_predicted)
fig, ax = plt.subplots(figsize=(10,7))
sns.heatmap(cm, annot=True, cmap='OrRd', fmt='2g', cbar=None);
```



CONCLUSIÓN
Notamos que entre las 4 pruebas que se realizaron no obtuvimos mucha diferencia entre el margen 50 a 1000 pero notamos que el que se realizó con el número más alto dio el mejor resultado, por lo que decidimos quedarnos con el de 1000.

Conclusión comparando los dos algoritmos
Recordemos los resultados de los algoritmos que entrenamos.

<u>En el caso del "Árbol de decisión" fueron:</u>
Accuracy de 98,11%
Falsos Positivos: 365
Falsos Negativos: 651

<u>Y en el caso de del "Random Forest" tenemos:</u>
Accuracy de 98,08%
Falsos Positivos: 639
Falsos Negativos: 390

Teniendo esto en cuenta, la conclusión a la que llegamos en su momento es que, a priori el algoritmo que mejor se adecua al plan de negocio es el Árbol de Decisión, dado que los Falsos Positivos son menos que los del Random Forest. Ya que es mejor que el banco reciba reclamos por no poder sacar un crédito, que otorgar un préstamos a alguien que no pueda pagarlo a futuro.

Cross Validation

Una de las técnicas más empleadas para probar la eficacia de un modelo de Machine Learning es la “cross-validation” o validación cruzada. Este método también es un procedimiento de “re-sampling” (remuestreo) que permite evaluar un modelo incluso con datos limitados.

Para efectuar una “CV” (cross-validation), hace falta apartar de antemano una parte de los datos de la serie de datos de entrenamiento. Esos datos no se utilizarán para entrenar el modelo, sino más tarde para probarlo y validarlo.

A menudo en Machine Learning se usa la cross-validation para comparar los diferentes modelos y seleccionar el más adecuado para un problema específico. Esta técnica es a la vez fácil de comprender, fácil de implementar y tiene menos sesgos que los demás métodos.

La técnica del Train-Test Split

El enfoque Train-Test Split consiste en descomponer de manera aleatoria una serie de datos. Una parte servirá para el entrenamiento del modelo de Machine Learning, la otra permitirá probarlo para la validación.

Por lo general, se reserva entre un 70 % y 80 % de los datos de la serie para el entrenamiento. El 20-30 % restante se explotará en la cross-validation.

Esta técnica es eficaz, salvo si los datos están limitados. Entonces puede faltar alguna información contenida en los datos que no se utilizan para el entrenamiento y, por tanto, los resultados pueden tener un gran sesgo.

Sin embargo, si la serie de datos es amplia y la distribución es igual entre las dos muestras, este enfoque es totalmente adecuado. Se pueden separar los datos de manera manual o usar el método train_test split de scikit-learn.

El método K-Folds

La técnica K-Folds es fácil de comprender y es particularmente conocida. Respecto a otros enfoques de Cross-Validation, suele resultar un modelo menos sesgado.

Justamente, permite garantizar que todas las observaciones de la serie de datos original tengan la oportunidad de aparecer en la serie de entrenamiento y en la serie de prueba. En caso de datos de entrada limitados, resulta uno de los mejores enfoques.

Primero se empieza separando la serie de datos de manera aleatoria en K folds. El procedimiento tiene un parámetro único llamado “K” que hace referencia al número de grupos en el se dividirá la muestra.

El valor de K no debe ser ni demasiado bajo ni demasiado alto y, por lo general, se elige un valor comprendido entre 5 y 10 en función de la envergadura de la serie de datos. Por ejemplo, si K=10, la serie de datos se dividirá en 10 partes.

Un valor K más alto lleva a un modelo con menos sesgo, pero una varianza demasiado amplia puede llevar a un ajuste excesivo. Un valor más bajo es prácticamente lo mismo que utilizar el método Train-Test Split.

Después se ajusta el modelo utilizando los folds K-1 (K menos 1). El modelo se valida usando el K-fold restante. Las puntuaciones y los errores se deben anotar.

El proceso se repite hasta que cada K-fold sirva dentro de la serie de entrenamiento. La media de las puntuaciones registradas es la métrica de rendimiento del modelo.

Este proceso se puede realizar de manera manual o con ayuda de las funciones cross_val_score y cross_val-predict de la librería Python de Scikit-Learn. La función cross_val_score indica la puntuación de cada fold de prueba, mientras que la función cross_val_predict indica la puntuación predicha para cada observación de la serie de datos de entrada cuando formaba parte de la serie de prueba.

En el caso de que el modelo (estimador) sea un clasificador y la variable objetivo (y) binaria o multiclase, se utiliza por defecto la técnica «StratifiedKfold». Este método presenta pliegues estratificados, por ejemplo, manteniendo el porcentaje de muestras de cada clase en todos los folds. De este modo, los datos de los folds de entrenamiento y de prueba se distribuyen de manera equitativa.

En los demás casos, se usa por defecto la técnica K_Fold para dividir y entrenar el modelo. Los folds se pueden utilizar como iteradores o en un bucle para entrenar en un marco de datos de Pandas.[4]

Kfold

```
#Importo librería
from sklearn.model_selection import StratifiedKFold, cross_val_score,
train_test_split
```

```
#metodo para cross-validation
CV_scores = cross_val_score(model_default, x_train, y_train, cv=50)
```

[4] https://datascientest.com/es/cross-validation-definicion-e-importancia
Definición de Cross Validation

```
#lista de todos los entrenamiento de kfold
CV_scores
array([0.98000465, 0.98000465, 0.98651476, 0.98721228, 0.98604976,
       0.98232969, 0.97930714, 0.97395954, 0.97767961, 0.98953732,
       0.98116717, 0.98116717, 0.9769821 , 0.98139967, 0.98093467,
       0.97884213, 0.97814462, 0.98395722, 0.97581958, 0.97930714,
       0.98465473, 0.98139967, 0.98139967, 0.9711695 , 0.97581958,
       0.98070216, 0.98581725, 0.9830272 , 0.98628226, 0.98558475,
       0.97884213, 0.97814462, 0.97581958, 0.97326203, 0.98093467,
       0.98139967, 0.97070449, 0.97488956, 0.971402  , 0.98604976,
       0.97767961, 0.98023256, 0.98348837, 0.98186047, 0.97627907,
       0.98604651, 0.97511628, 0.98883721, 0.97604651, 0.98325581])
```

```
print("Cross validation score es  %.5f ± %0.2f" % (CV_scores.mean(),
CV_scores.std()))
Cross validation score es  0.98033 ± 0.00
```

Con esto podemos ver que el valor de nuestra validación cruzada es de un 0.976

Con este resultado confirmamos que el modelo Árbol de decisión en comparación con el modelo Random Forest tiene un mejor performance y que se ajusta mejor a los resultados esperados para este modelo de negocio, donde la agilidad y la seguridad en que un cliente cumple con los requisitos indicados para obtener un crédito..

Métricas finales del Modelo Optimizado

Con base en lo anterior nuestras métricas corresponden a las mismas obtenidas en la segunda iteración realizada de nuestro modelo por cuanto fueron las que mejor desempeño tuvieron:

```
y_test
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

#accuracy
print('Accuracy score for test data is:',
accuracy_score(y_test,y_pred_test))

confusion_matrix = pd.DataFrame(confusion_matrix(y_test,
y_pred_test))

confusion_matrix.index = ['NO APROBADO','APROBADO']
confusion_matrix.columns = ['PREDICCION NO
APROBADO','PREDICCION APROBADO']
print(confusion_matrix)
```

```
Accuracy score for test data is: 0.9811011904761905
             PREDICCION NO APROBADO  PREDICCION APROBADO
NO APROBADO                    8790                  365
APROBADO                        651                43954
```

Conclusión

Podemos notar que tenemos un Accuracy de 98,11%. Podemos notar que la cantidad de falsos positivos es de 365 y los falsos negativos es de 651.

Conclusiones

Si bien obtuvimos un modelo funcional y bastante básico, las posibilidades de aplicación y mejora son muchas para este modelo dentro del mercado (banca), entre esas posibilidades de alcance está la de poder asignar también el monto de crédito a dar al usuario y no limitarnos a una respuesta binaria, otro posible alcance del modelo sería el de segmentar a los usuario que obtuvieron crédito y de acuerdo a sus características poder aplicar modelos cross-sell/up-sell y los que no obtuvieron créditos derivarlos a otros productos que se ajustan a sus características/scoring y re-evaluarlos después de un tiempo determinado.

Como experiencia de esta cursada de Data Science entendimos la importancia de esta disciplina la cual es clave en el proceso de toma de decisiones hoy día donde los grandes volúmenes de información son materia bruta que podemos refinar y aprovechar en muchas industrias para brindar mayores y mejores soluciones en ventas, atención al cliente, optimización y personalización de resultados, etc.

A continuación tenemos una Presentación ejecutiva que resume lo anterior visto

BIBLIOGRAFÍA

- **Bank_df_1.csv**
  https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction

- **Credit_risk_dataset.csv**
  https://www.kaggle.com/datasets/laotse/credit-risk-dataset

- **Estado del arte**
  http://www.scielo.org.co/pdf/pece/n16/n16a10.pdf

- **Glosario**
  https://iaarbook.github.io/ML/
  https://www.ibm.com/cloud/learn/random-forest

- **Loan.csv**
  https://www.kaggle.com/datasets/ychope/loan-approval-dataset

- **Normas prudenciales del sistema financiero argentino**
  http://www.bcra.gov.ar/Pdfs/SistemasFinancierosYdePagos/mas1101.pdf