Segunda entrega del proyecto final

Juan Lucas Ibarra Sierra – 201813900

Juan Camilo Garcia Ruiz - 201729554

1. Preparación de datos

Iniciamos el proceso de preparación de los datos en la anterior entrega del proyecto. El diseño que realizamos estuvo dividido en 3 etapas:

a) Corrección de los datos: En esta fase se realizó el ajuste de datos que tuvieran valores incorrectos y se limpiaron valores atípicos que pudieran alterar la construcción del modelo.

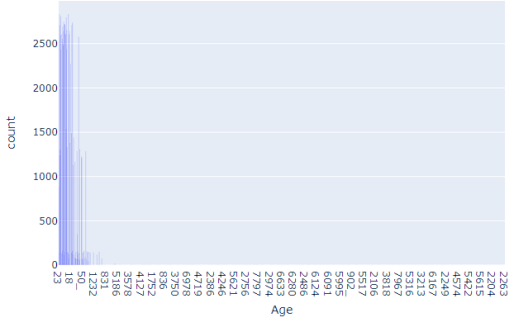
Para nuestro caso específico utilizamos la siguiente metodología:

Al ver que hay varios datos para un mismo cliente los datos anómalos o faltantes eran buscados en instancias anteriores para ese mismo cliente. Un ejemplo de esto es:

Customer_ID	Age	Monthly_Inhand_Salary
CUS_0xd40	23	1824.843333
CUS_0xd40	23	NaN
CUS_0xd40	-500	NaN
CUS_0xd40	23	1824.843333

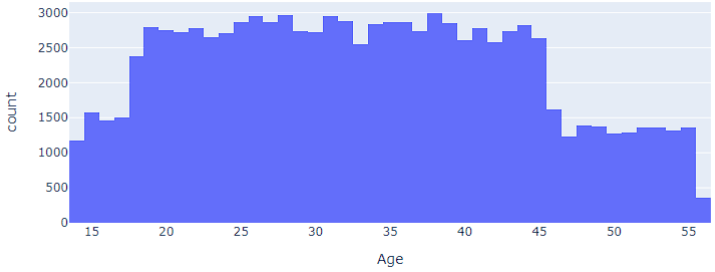
Podemos ver que el cliente con ID CUS_0xd40 hay un dato con una edad de **–500** o un Monthly_Inhand_Salary **faltante**. Sin embargo, vemos que hay otras entradas que si tienen el valor correcto. Es por esto por lo que, para la limpieza de datos, se buscaba a ese cliente especifico y se reemplazaba el valor anómalo/faltante con la moda para esa característica de ese cliente. En este caso la edad de –500 se remplaza por **23** y los Monthly_Inhand_Salay faltantes se reemplazaron por **1824,643333**.

Variable Age: En esta variable encontramos dos errores de imputación. El primero de ellos es que algunas edades tenían el símbolo "_" al final de la edad. El segundo de ellos era de edades mal imputadas con valores como 4808 como se puede ver en la imagen de las distribuciones.



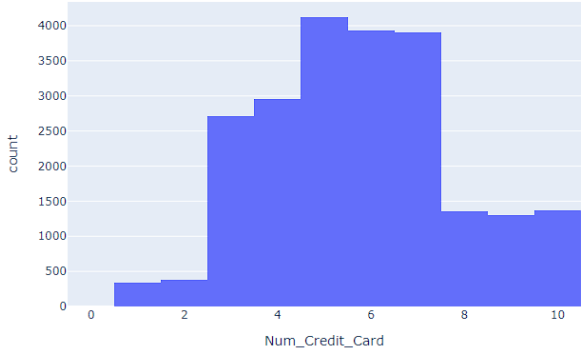
La transformación que se le realizo fue eliminar el símbolo "_" de los datos y después filtrar aquellos que estuvieran entre 0 y 90 que son las edades que

tienen sentido conforme a las reglas de negocios. A los que estaban fuera de este rango se les aplico la metodología explicada al inicio. En la siguiente grafica se puede ver como quedo la distribución de los datos corregidos con valores entre 14 y 56 años.



Variable Num_Bank_Accounts: Encontramos que algunos clientes tenían números de cuenta negativas y valores que no concuerdan con el negocio como 1798 cuentas. Se filtro por valores cercanos al percentil 75 (7 cuentas) y realizamos la limpieza.

Variable Num_Credit_Card: Encontramos valores atípicos como un usuario con 1499 tarjetas. De nuevo filtramos por valores cercanos al percentil 75 (7 tarjetas) y realizamos la limpieza.

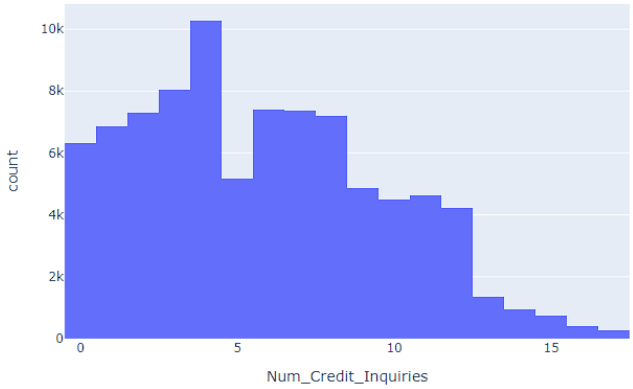


Variable Interest_Rate: Encontramos valores atípicos como tasas del 5797% lo cual no corresponde a la lógica del negocio. Filtramos tasas desde el 0% al 50% y realizamos la limpieza.

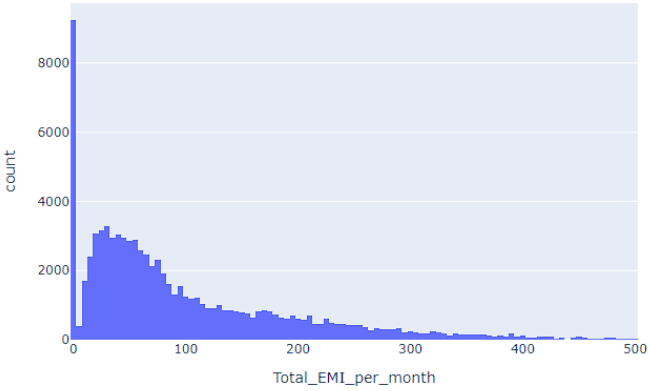
Variable Delay_from_due_date: Encontramos valores negativos que no tienen sentido con el negocio. Se eliminaron estos valores ya que eran muy

pocos y al ser un rango tan amplio era muy complicado imputarlos como se habia hecho para valores anteriores.

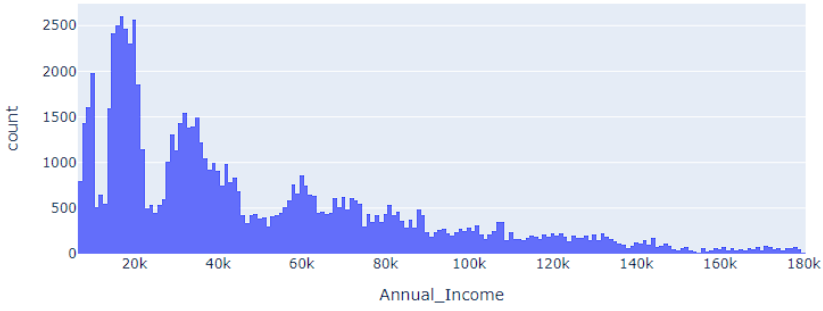
Variable Num_Credit_Inquiries: Encontramos valores atípicos como 2597 lo cual no cumple con las reglas de negocio. Se filtraron los datos que estuvieran entre 0 y 20 y se realizó la limpieza.



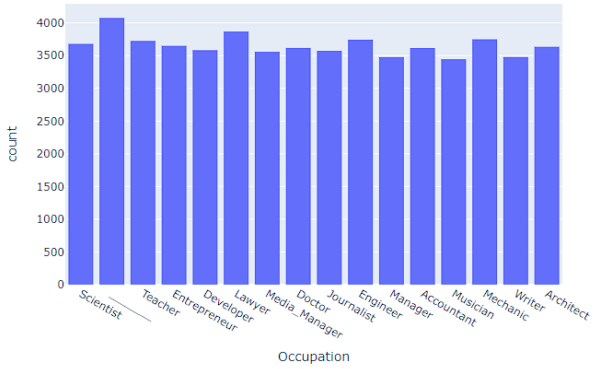
Variable Total_EMI_per_month: Encontramos valores atípicos como 82331, lo cual no cumple las reglas de negocio. Filtramos con valores cercanos al percentil 90 (422.3) y filtramos los valores entre 0 y 500.



Variable Annual_Income: De nuevo encontramos valores con el símbolo "_" y valores atípicos fuera del contexto de negocio. Se eliminaron estos valores atípicos tomando el percentil 75 y se eliminó el símbolo incorrecto.



Variable Occupation: Se imputan los datos con valor "____" con la metodología mencionada arriba. Tenemos que las profesiones están balanceadas.



b) Selección de variables: En esta etapa se hizo el correspondiente análisis de los features y se tomó la decisión de cuáles serían las variables
Utilizadas para entrenar el modelo. A continuación, se muestran las variables que fueron eliminadas y porque:
   - **Customer_ID, Name, SSN, ID:** Son datos propios y **únicos** del usuario por lo que no aportan información al modelo.
   - **Type_Of_Loan:** Tiene una alta cardinalidad y no es posible realizar una agrupación de las categorías. Tiene 2011 valores únicos.

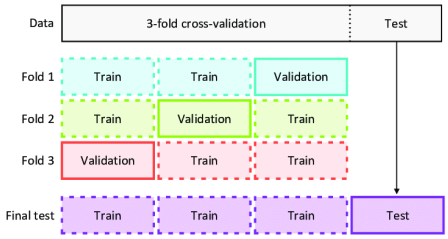
c) Conversión de variables categóricas: Después de filtrar las variables se realizó un proceso de conversión de las variables categóricas a numéricas.
   - **Month:** Se reemplazo el mes por su número respectivo en orden (Enero: 1, Febrero: 2, …)
   - **Credit_History_Age**: Se tenía el tiempo en palabras, ej: 3 years and 5 months. Se paso a númerica con unidad en meses.
   - **Occupation, Payment_of_Min_Amount, Payment_Behaviour:** Se procesaron con un OneHotEncoder que crea columnas con 0 y 1 por cada uno de los valores de la categoría.
   - **Credit_Mix:** Se le asigno un valor a cada categoría ya que este valor es ordinal (Bad: 0, Standard: 1, Good: 2)

Nota: Los datos no fueron normalizados para nuestros modelos más complejos (Gradient Boosting y Random Forest). Solo se normalizo para KNN ya que este si es sensible a la escala de los datos.

2. Estrategia de validación y selección de modelo

Al finalizar la preparación de los datos nos dispusimos a construir nuestra estrategia de validación. Tras la limpieza obtuvimos un total de 77,514 entradas y un total de 43 columnas para entrenar el modelo. Dividimos el dataset en una proporción 70% entrenamiento y 30% de prueba. Siguiendo esto obtuvimos 54,259 entradas para entrenamiento y 23,255 entradas para prueba, lo cual nos parece una cantidad adecuada para realizar un modelo correcto. Además de esto para evitar sesgos en el entrenamiento utilizamos un **3-Fold Cross-Validation** al momento de encontrar los hiperparametros de nuestros modelos con el fin de entregar un mejor resultado. Para

esto tomaremos del 70% un aproximado del 33% (17905 entradas) en cada una de las iteraciones con el fin de hacer las respectivas validaciones.



La etapa de experimentación consistirá en 2 etapas:

- 1. Construcción del modelo: Se construirán 3 modelos, uno básico que será el KNN, y dos complejos que serán un Random Forest y un Gradient Boosting
- 2. Ajuste de hiperparametros: Para cada uno de los modelos se realizar un ajuste de hiperparametros donde se aplica el Cross Validation y se compararan resultados contra el modelo original. La métrica para seleccionar los mejores parámetros es el F1-score.

Una vez finalizada esta etapa se procederá a realizar la evaluación de los modelos, y con la métrica del F1-score se procederá a seleccionar el mejor para llevarlo a producción.

3. Construcción del modelo

Para todos los modelos la búsqueda se hizo por medio de Grid Search.

- Modelo 1. KNN:

  Se construyo un modelo inicial utilizando 5 vecinos. Tras esto se realizó un ajuste de hiperparametros evaluando con 3, 5 y 10 vecinos respectivamente.

```
knn = KNeighborsClassifier(n_neighbors=5)
```

  Se obtuvo que el mejor modelo se obtiene con n_neighbors = 3.

  Calculando las métricas en el conjunto de train se obtiene:

```
Classification Report
              precision    recall  f1-score   support

        Good       0.81      0.84      0.82      8748
        Poor       0.86      0.89      0.88     16267
    Standard       0.90      0.87      0.88     29244

    accuracy                           0.87     54259
   macro avg       0.86      0.87      0.86     54259
weighted avg       0.87      0.87      0.87     54259
```

  Observamos que tiene un muy buen rendimiento en **train.** El desempeño es homogeneo entre clases. Se debe validar en test si es consistente para ver si no hay evidencias de overfitting.

- Modelo 2. Random Forest:

  Se construyo el modelo inicial utilizando un máximo de 15 features y una profundidad máxima de 12. Tras esto se realizó un ajuste de hiperparametros

evaluando 10, 15, 20, y 25 máximo de features y 7, 12, 15 y 20 de profundidad máxima.

```
rf = RandomForestClassifier(max_features=15 , max_depth=12)
```

Se obtuvo que el mejor modelo se obtiene con max_depth = 20, max_features = 25

Calculando las métricas en el conjunto de train se obtiene:

Classification Report

	precision	recall	f1-score	support
Good	0.68	0.76	0.72	8748
Poor	0.81	0.77	0.79	16267
Standard	0.83	0.82	0.83	29244
accuracy			0.80	54259
macro avg	0.77	0.78	0.78	54259
weighted avg	0.80	0.80	0.80	54259

Observamos que este modelo se equivoca mucho en las clases “Good” y “Poor” y se desempeña bien en “Standard”. Si observamos la cantidad de los datos por clase podemos ver que en las dos que peor se desempeña son las que menos datos tiene. Esta puede ser una de las razones por la cual es peor prediciendo para estas clases.

- Modelo 3: Gradient Boosting:

  Se construyo el modelo inicial con una tasa de aprendizaje de 0.75, una profundidad máxima de 12 y el número de estimadores en 50. En la fase de ajuste de hiperparametros mantuvimos el número de estimadores y probamos

con 0.75 y 1 de tasa de aprendizaje, y con 5, 10 y 12 de máximo de profundidad.

```
gb = GradientBoostingClassifier(learning_rate=0.75 , max_depth=12, n_estimators=50)
```

Con el modelo base se obtuvo el mejor resultado.

Calculando las métricas en el conjunto de train se obtiene:

Classification Report

	precision	recall	f1-score	support
Good	0.80	0.80	0.80	8748
Poor	0.83	0.82	0.82	16267
Standard	0.86	0.86	0.86	29244
accuracy			0.84	54259
macro avg	0.83	0.83	0.83	54259
weighted avg	0.84	0.84	0.84	54259

Observamos que tiene un muy buen rendimiento en **train.** El desempeño es homogéneo entre clases. Se debe validar en test si es consistente para ver si no hay evidencias de overfitting.

4. Evaluación del modelo

Para la evaluación y comparación de modelos utilizaremos su desempeño en el dataset de prueba.

KNN

Classification Report

	precision	recall	f1-score	support
Good	0.63	0.64	0.64	3801
Poor	0.76	0.76	0.76	7026
Standard	0.78	0.77	0.77	12428
accuracy			0.75	23255
macro avg	0.72	0.72	0.72	23255
weighted avg	0.75	0.75	0.75	23255

Random Forest

Classification Report

	precision	recall	f1-score	support
Good	0.75	0.75	0.75	3801
Poor	0.79	0.81	0.80	7026
Standard	0.83	0.81	0.82	12428
accuracy			0.80	23255
macro avg	0.79	0.79	0.79	23255
weighted avg	0.80	0.80	0.80	23255

Gradient Boosting

Classification Report

	precision	recall	f1-score	support
Good	0.76	0.72	0.74	3801
Poor	0.79	0.79	0.79	7026
Standard	0.81	0.82	0.81	12428
accuracy			0.79	23255
macro avg	0.78	0.78	0.78	23255
weighted avg	0.79	0.79	0.79	23255

Como podemos observar, el KNN es el modelo que tiene peor desempeño. En cuanto al Random Forest y Gradient Boosting tienen un desempeño muy similar; sin embargo, el RF se desempeña un poco mejor en datos que todavía no habían sido vistos por los modelos por lo que es el más apto para producción. Adicional a esto, identificamos síntomas de overfitting para el KNN y el GB ya que se desempeña muy bien en train pero no sucede lo mismo en test. Esto se puede reducir buscando otros hiperparametros o modificando el número de folds del cross validation.

Analizando más a fondo la precisión y el recall observamos el modelo se desempeña casi igual. Y se tiene un accuracy del 80% por lo que el 80% de las veces nuestro modelo va a predecir la clase correctamente.

Para mejorar los modelos se pueden ajustar otros hiperparametros que podrían mejorar el desempeño de este ya que para esta ocasión solo se ajustaron 2 por modelo. También se podría mejorar el desempeño en las clases minoritarias como lo son Good y Poor ya que ahí es donde el modelo tiene el peor rendimiento realizando un over-sampling.

5. Conclusiones

- Como vimos en la etapa de limpieza, la calidad de los datos es muy baja. Hay muchos datos faltantes y demasiados datos con valores no permitidos. Los datos deben estar en mejores condiciones para que no se hagan tantos supuestos en la limpieza y se pueda obtener un mejor desempeño con datos reales.
- También se podrían reducir los tipos de préstamo ya que actualmente son demasiados, pero es una característica importante para predecir. Se podría pensar en características adicionales como si alguna vez fue reportado en una central de riesgo.
- Por último, sería ideal tener más datos de las clases Good y Poor porque se cuenta con muy pocos datos, lo que genera un desbalanceo. Esto hace que nuestro modelo no se desempeñe de la mejor forma en estas clases. Consideramos que el modelo obtenido es un buen modelo para poner en producción ya que las métricas dan resultados favorables, sin embargo, se debería validar con el negocio **para ver si el porcentaje de error es aceptable para ellos**. Este modelo podría ponerse en producción por medio de un API REST que se conecte a una aplicación web.
- El modelo seleccionado tiene un rendimiento del 75% en general en su comportamiento. Consideramos que es un buen modelo para iniciar la segmentación de los clientes, más sin embargo creemos que mejorando la forma en la que se imputan y guardan los datos de los clientes puede mejorar aún más la calidad del modelo.
- La salida a producción de este modelo se realizará por medio de una página web, donde los consultores después de verificar la veracidad de los datos podrán cargar archivos CSV o Excel con multitud de clientes y estos serán categorizados automáticamente por el sistema.