# Reporte de la prueba técnica de Bancolombia para el puesto de científico de datos

Candidato: Felipe Díaz Jaramillo

Fecha: 25 de Noviembre 2024

**Resumen**

Describimos en detalle la construcción exitosa *end-to-end* de un modelo de *machine learning* basado en el método *Random Forest* para predecir la aceptación de opciones de pago hechas por Bancolombia a clientes en mora. El modelo exhibe un desempeño razonable, obteniendo un *F1-score* de 0.90 con los datos usados para su entrenamiento, y un *F1-score* de 0.62 al hacer las predicciones requeridas evaluadas en Kaggle. Usamos un subconjunto de los datos disponibles en el desafío de Kaggle de la prueba (link).

## Contenido

# 1 Introducción y planteamiento del problema

Este reporte documenta los pasos que se siguieron en la solución de la prueba técnica de Bancolombia para el puesto de científico de datos. El problema asignado es un problema de clasificación para predecir si clientes en mora con el banco aceptarán o no opciones de pago. Un cliente en mora es un cliente que no le paga sus obligaciones al banco luego de adquirir un producto. Bancolombia les ofrece a estos clientes diferentes opciones de pago para estas obligaciones morosas. El objetivo principal de este proyecto es, dados unos datos, predecir si clientes con mora van a aceptar o no una de estas opciones de pago alternativas.

Este reporte consta de tres secciones (además de esta), en las que se describe de forma detallada el razonamiento y los pasos que se siguieron para la solución del problema. En la segunda sección se detalla la exploración, selección y liempieza de datos. En particular, se explica por qué se escogieron ciertos datos para solucionar el problema y qué pasos se siguieron para hacer que estos datos sean útiles para construir un modelo de *machine learning* basado en el método *Random Forest* que se desarrolla en detalle en la tercera sección, en dóbnde se detalla la sintonización de los datos del modelo y su desempeño. Finalmente, en la cuarta sección se describe el procedimiento que se siguió para procesar los datos dados para hacer las predicciones requeridas. Al final de cada sección se incluye un apartado donde se describe cómo se podría mejorar cada etapa del proyecto.

# 2 Exploración, selección y liempieza de datos

En esta sección procedemos a cargar todos los datos proporcionados en Kaggle para verlos y analizarlos, en un principio de forma superficial. Generamos un diccionario en Python que aquí denotamos con $\mathcal{D}$ el cual llamamos `data` y cuyos valores son DataFrames de la librería pandas que contienen los archivos .cvs proporcionados. Los valores de $\mathcal{D}$ contienen información de diferentes clientes de distinta índole. Por ejemplo, en el valor $\mathcal{D}_{\text{pt}} \equiv \mathcal{D}[\text{`pivot\_test'}]$, encontramos datos sobre los clientes que han tenido mora con el banco (por lo menos hasta justo antes de enero del 2024), incluyendo las opciones de pago que el banco les ha ofrecido, si han aceptado estas opciones en el pasado, la identificación de las obligaciones con mora, entre otros. Similarmente, el valor $\mathcal{D}_{\text{cu}} \equiv \mathcal{D}[\text{`customers'}]$ contiene datos de clientes (no sólo aquellos que se encuentran en mora), como su estado marital, datos familiares, entre otros. Los demás valores de $\mathcal{D}$, si bien contienen datos relevantes, no los mencionaremos en el resto de este informe ya que no los usamos para resolver el problema. En lo que sigue, nos enfocaremos y procesaremos los datos $\mathcal{D}_{\text{pt}}$ ya que ofrecen la solución más práctica al problema, al contener todos los datos hitóricos de los clientes en mora del banco hasta justo antes de enero del 2024.

La decisión de enfocarnos en $\mathcal{D}_{\text{pt}}$ viene con un costo. Una vez cargamos los datos a ser analizados a un DataFrame llamado `data_oot`, el cual denotaremos $\mathcal{D}_{\text{oot}}$, es posible observar que en $\mathcal{D}_{\text{pt}}$ no hay datos de todos los clientes contenidos en los datos objetivo $\mathcal{D}_{\text{oot}}$. Efectivamente, es posible que nuevos clientes hayan entrado en mora, viejos clientes hayan salido de ella, o los clientes que ya tenían mora volvieron a entrar en mora con otras obligaciones. Esto, a su vez,

implica que la información contenida en $\mathcal{D}_{\text{pt}}$ no es suficiente para hacer todas las predicciones necesarias. Por esta razón, para hacer predicciones significativas, hay dos opciones que podemos tomar:

- Generar datos de manera artificial para los clientes y productos sobre los que no tenemos información de mora, basándonos en la información que sí tenemos.
- Usar otra fuente de datos, por ejemplo $\mathcal{D}_{\text{cu}}$, para hallar correlaciones entre esos datos y la probabilidad de aceptación de opciones de pago, usando los clientes que están tanto en los datos históricos de mora $\mathcal{D}_{\text{pt}}$ como en la información adicional $\mathcal{D}_{\text{cu}}$.

Si bien la segunda opción es más probable que resulte en mejores resultados, por cuestiones de recursos y tiempo vamos a optar por la más pragmática, primera opción.

Habiendo tomado la decisión de proceder únicamente con $\mathcal{D}_{\text{pt}}$, es necesario analizar los datos para garantizar que el modelo arroje buenos resultados. El primer paso en este proceso es buscar valores faltantes (`NaN`) en los datos. Una vez identificados, procedemos a reemplazarlos por valores según el tipo de característica que representan. Con un set de datos de este tamaño, optamos por llenar los datos numéricos faltantes con la media de las columnas de las características consideradas. Para variables no numéricas o categóricas, podemos usar la moda de los datos o podemos crear categorías nuevas cómo `OTRO`, `NO APLICA (N.A)`, etc. Los detalles de las nuevas categorías se puden encontrar en el Notebook entregado.

Finalmente, es necesario tener en cuenta que, si bien cada cliente tiene un N.I.T asociado, un mismo cliente puede estar en mora con diferentes obligaciones. En este proyecto trataremos a cada trío de identificadores con un único identificador

$$\texttt{ID} \sim \texttt{('nit\_enmascarado', 'num\_oblig\_orig\_enmascarado', 'num\_oblig\_enmascarado')}\ ,$$

dónde el símbolo $\sim$ significa "representa". Esto lo hacemos añadiendo nuevas columnas a $\mathcal{D}_{\text{pt}}$. Es importante recalcar que ID, de la forma en que es generado, es de tipo `str`, lo que dificulta hacer cálculos de manera eficiente en librerías como NumPy. Para remediar esto, construimos un identificador numérico `ID_prod_sum` definido como

$$\begin{aligned} \texttt{ID\_prod} &= (\texttt{nit\_enmascarado} \times \texttt{num\_oblig\_orig\_enmascarado}\ , \\ \texttt{ID\_prod\_sum} &= \texttt{ID\_prod} + \texttt{nit\_enmascarado} + \texttt{num\_oblig\_orig\_enmascarado}\ . \end{aligned}$$

Estas definiciones de los identificadores, por primero principios, no garantizan que generen un identificador numérico único para cada `ID`. Sin embargo, en el código se corroboró de manera explícita que así fuera.

## 2.1 Para el futuro de los datos

Idealmente, la exploración y análisis superficial de datos se debería hacer de una forma más exhaustiva para buscar (y posiblemente encontrar) patrones en los datos que puedan ayudar en la elección de método para construir un modelo. También se debería explotar las demás fuentes de información que no son $\mathcal{D}_{\text{pt}}$ con el fin de tener un set de datos más completo.

# 3 Construcción, entrenamiento y evaluación del modelo

Con los datos escogidos y procesados a nuestra disposición, en esta sección desarrollamos un modelo de *machine learning* usando el método *Random Forest*. Este método se basa en el uso de **árboles de decisión**, los cuales toman decisiones a partir de preguntas sobre las características de los datos. El nombre *Random Forest* (bosque aleatorio) se refiere a que el método consiste en tomar subconjuntos de los datos de entrada de manera aleatoria para hacerlos pasar por varios árboles de decisión. Luego las salidas de los árboles de decisión son combinadas para generar una predicción. Este método fue escogido ya que nos enfrentamos a un problema de clasificación (si clientes aceptarán o no opciones de pago), y *Random Forest* es un método robusto que lidia bien con no linealidades y con variables categóricas, las cuales aparecen frecuentemente en los datos que tenemos.

Antes de construir el modelo es necesario primero identificar qué características (*features*) de $\mathcal{D}_{\text{pt}}$ usaremos en su entrenamiento para no encontrar problemas una vez usemos el modelo para hacer las predicciones que se piden. Para simplificar el proceso (y para evitar *overfitting*), seleccionamos las características categóricas que consideramos relevantes y no modificamos las características numéricas. Luego, debemos asegurarnos de que las características escogidas puedan ser usadas tanto en el entrenamiento del modelo como en las predicciones deseadas. Teniendo esto en cuenta, escogemos las características categóricas que se muestran en el Notebook y filtramos $\mathcal{D}_{\text{pt}}$ de forma que tengamos características que no entren en conflicto con el uso del modelo en los clientes que debemos predecir. Vale la pena comentar que en las últimas columnas de $\mathcal{D}_{\text{pt}}$ se encuentran datos que se relacionan directamente con la variable respuesta que queremos predecir y por lo tanto deben ser eliminados.

Como mencionamos antes, $\mathcal{D}_{\text{pt}}$ contiene diferentes características categóricas. Para poder usarlas en la construcción del modelo, hay que codificarlas en características numéricas. En este proyecto usamos *One-Hot encoding* para codificar las características categóricas en características numéricas. Matemáticamente, podemos hacer la siguiente caricatura de *One-Hot encoding*, donde representamos la columna del DataFrame con $n$ características categóricas distintas $a_i$, $i = 1, \dots n$ como un vector columna[1] que se transforma en una matriz que representa un conjunto nuevo de columnas luego de la codificación:

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix} \rightarrow \begin{array}{c} \begin{matrix} a_1 & a_2 & \dots & a_n \end{matrix} \\ \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ 0 & 0 & \dots & 0 \end{pmatrix} \end{array} .$$

El efecto de esta codificación es que el número de columnas de $\mathcal{D}_{\text{pt}}$ crece. Estas columnas nuevas contienen valores binarios y cada valor categórico genera una columna nueva donde 1 indica que `ID` presenta uno de los valores categóricos codificado, y 0 que no. Luego de codificar los

[1] En general no tienen que ser distintas. En ese caso, no se tendría una matriz identidad, sino otra distribución de unos y ceros.

datos de forma apropiada, construimos el modelo. Usamos el 80 por ciento de los datos de $\mathcal{D}_{\text{pt}}$ (post-procesamiento) para entrenar el modelo y 20 por ciento para testearlo.

### 3.1 Optimización de parámetros y evaluación del modelo

Sintonizamos los parámetros del modelo usando métodos de optimización del *F1-score*. El código usado esá comentado en el Notebook y no es recomendable correrlo ya que puede tardar bastante tiempo. Los parámetros que optimizamos fueron la cantidad de estimadores (árboles de decisicón), cantidad de niveles (*depth*), y la cantidad de muestras. Al optimizar los parámetros del modelo, obtuvimos un *F1-score* de 0.90 sobre los datos para testear.

Un valor de 0.90 en el *F1-score* es, en principio, un signo de que el modelo va por un buen camino. Sin embargo, para ver si el uso del modelo es justificado, el *F1-score* del modelo debe ser comparado con alguna métrica base. Para este proyecto se escogió comparar el desempeño del modelo con una base de *Majority Class*. Esta consiste en simplemente hacer predicciones basadas en el valor más frecuente encontrado en los datos que se usaron para el entrenamiento del modelo. El *F1-score* base en este caso fue de 0.69. Por ello podemos concluir que el modelo presenta una ventaja significativa con respecto a la métrica base, y por lo tanto la implementación del modelo es justificada.

### 3.2 Para el futuro del modelo

Quizás el punto más importante a mejorar en la construcción del modelo es un entendimiento más riguroso de las categorías disponibles y escogidas. Las características escogidas en este proyecto fueron escogidas desde la ignorancia de la industria del autor. Además, algunas veces es díficil entender las descripciones de las variables sin un contexto más completo. También es relevante explorar de manera más exhaustiva las relaciones entre la variable respuesta (la decisión de aceptación) y las demás características.

Con más recursos (tanto de tiempo como computacionales), sería interesante sintonizar los parámetros del modelo de una mejor manera, y también se deberían explorar otras formas de codificar características categóricas y su influencia en los resultados. Finalmente, sería uan buena idea desarrollar otros modelos que permitan extraer información relevante de sets de datos como $\mathcal{D}_{\text{cu}}$ para ajustar mejor este modelo.

## 4 Implementación del modelo e inferencia

Finalmente procedemos a la implementación del modelo para hacer las predicciones requeridas. Para lograr esto, es necesario primero procesa los datos objetivo $\mathcal{D}_{\text{oot}}$ de forma que podamos usarlos en el modelo. El primer paso es generar un DataFrame que contenga los `ID's` contenidos en $\mathcal{D}_{\text{oot}}$, que a su vez también tenga la información de mora histórica de $\mathcal{D}_{\text{pt}}$. Esto lo hacemos uniendo los datos de $\mathcal{D}_{\text{pt}}$ con los de $\mathcal{D}_{\text{oot}}$. En esta fusión de los dos DataFrames, seguimos pasos

para obtener un DataFrame que objetivo para hacer predicciones que denotamos como $\mathcal{D}_{\text{pred}}$, como se ilustra de forma detallada en el Notebook. Este nuevo DataFrame contiene los datos históricos de mora de algunos `ID` en $\mathcal{D}_{\text{oot}}$, los cuales están completos, y datos de los demás `ID` en $\mathcal{D}_{\text{oot}}$, los cuales no tienen historial de mora, y por consiguiente incompletos, es decir con `NaN`.

Para hacer predicciones debemos tratar con los `NaN`. La forma en la se escogió tratar con este problema fue reemplazando los `NaN` numéricos por valores aleatorios normalmente distribuidos entre el valorr máximo y mínimo de los datos que disponibles. Similarmente, los datos categóricos faltantes los reemplazamos por valores aleatorios. Finalmente, teniendo un DataFrame completo, alimentamos al modelo con $\mathcal{D}_{\text{pred}}$ (post-procesamiento) y obtuvimos las predicciones requeridas. Para determinar la estabilidad del modelo y de la forma de tratar con los valores faltantes, se hicieron las predicciones repetidas veces, obteniendo siempre resultados consistentes. Los resultados fueron enviados a Kaggle, y se encontró un $F1 - score$ de 0.62 tres entregas distintas. La disminución del desempeño del modelo tiene sentido, teniendo en cuenta que muchos de los datos usados fueron generados de forma aleatoria.

## 4.1 Para el futuro de la implementación

Si bien generar los datos faltantes de manera aleatoria genera predicciones correctas, se debe encontrar una forma más elegante de abordar el problema para asegurar mejores resultados. El siguiente paso más evidente es depurar el modelo usando modelos auxiliares para hallar correlaciones entre $\mathcal{D}_{\text{pt}}$ y más fuentes de información como $\mathcal{D}_{\text{cu}}$.