American Express - Predicción de Impagos usando Árbol de Decisión

(1) Huby Tuesta, Jean Paul and (2) Córdova Amaya, Efraín Antonio

Universidad de Ingenieria y Tecnología, Lima, Perú

I. Introducción

American Express desea detectar la probabilidad de impago de sus clientes, es decir, que al cierre de su facturación declaren que no pueden devolver el monto consumido [1]. Este proyecto cubrirá el desarrollo y discusión de un modelo predictivo que resuelva satsifactoriamente (con una precisión comprobada, al menos, aceptable) este problema computacional.

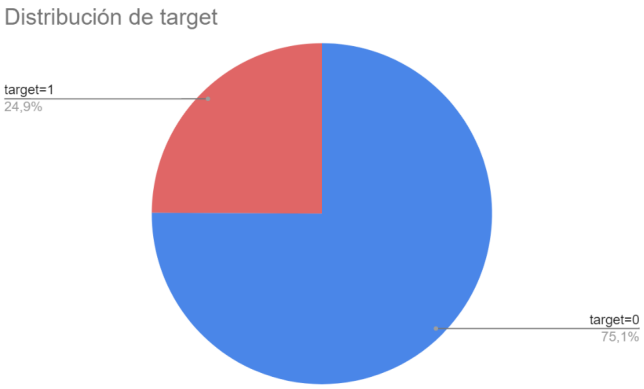

Figure 1. Resumen de la variable *target*, que representa el resultado esperado de la predicción. El modelo desarrollado debe entregar una variable resultado con una distribución cercana a esta (No incumplió = 75.1%, Incumplió = 24.9%).

Por el gran volumen de la base de clientes de la compañia, se utilizarán herramientas de *Big Data*. La velocidad de procesamiento por lotes de tiempo real proporcionada por *Spark* justifica su elección para las etapas de exploración y preprocesamiento de datos y entrenamiento del modelo [2], mientras que el *File System* de *Hadoop* permitirá alocar el proyecto de forma virtual mediante una arquitectura de 2 *workers* y 1 *master*, con el fin de distribuir la carga de recursos sin afectar significativamente la latencia.

Los árboles de decisión son algoritmos que permiten realizar una clasificación en base a una sucesión de comprobaciones binarias según los campos de cada registro [3]. Son modelos simples y rápidos de entrenar en comparación a otros algoritmos de clasificación, dos ventajas importantes debido a la gran cantidad de datos con las que el modelo debe manejarse. Adicionalmente, es un método soportado en *PySpark*, framework que también provee funcionalidades que nos permiten impactar positivamente en la eficiencia del entrenimiento.

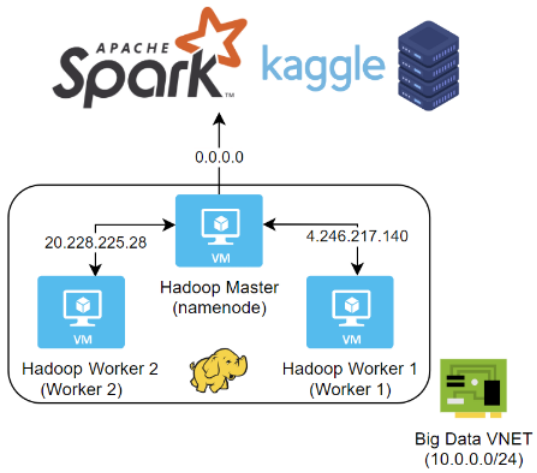

Figure 2. Diagrama de la arquitectura. El nodo maestro habilita cualquier entrada hacia el *namenode* a través del puerto 0.0.0.0 con la ruta de *hdfs*. Los nodos estan bajo una misma *subnet*.

II. Colección de datos

A. Descripción de la fuente

La colección fue recogida de la biblioteca *online* de Kaggle [4]. El tamaño total es de 50.31GB, distribuido en 4 archivos, de los cuales nos interesan, principalmente, los archivos de validación y entrenamiento: *test_data.csv* (33.82GB) y *train_data.csv* (16.39GB), respectivamente. El número de clientes (valores únicos en el campo *customer_id*) es de 458,913.

Las 190 variables de la colección estan anonimizadas debido a la sensibilidad de los datos. Se clasifican en 5 tipos: De delincuencia, de gastos, de pagos, de balance o de riesgos. Sólo 11 variables en toda la colección son *strings* y 3 variables presentan un rango con una longitud menor a 10, por lo que, finalmente, consideramos 14 variables categóricas.

B. Particiones en formato parquet

Los archivos de entrenamiento y validación serán particionados en el formato *parquet* usando las funciones de escritura de *Dask* utilizando el *engine* de *pyarrow* y un tamaño total de 10 particiones por archivo [5,6]. El proceso redujo el set de validación al 39% (13.2GB) y entrenamiento a 40% (6.6GB) con respecto a sus tamaños originales.

C. Preprocesamiento

La etapa de preprocesamiento tiene como finalidad asegurar que el *dataframe* que contiene los datos a ingresar en el modelo cumpla las siguientes propiedades, necesarias para soportar un árbol de decisión bajo la implementación de *PySpark*:

1) ***Valores válidos****:* Este proceso consiste en detectar las variables que presentan valores N.A y retirarlas del *dataframe*. Generalmente, este proceso es más fino, pues la variable que presenta estos valores puede ser estudiada y, en algunos casos, reinsertada en la colección con algo de procesamiento. Sin embargo, dada la anonimización de los datos, no conocemos mayor información de ningún campo más allá de su valor, por lo que se decidió simplemente deshacerse de estas variables. Sólo 69 de las 191 variables (incluyendo el resultado esperado de la predicción) no presentan ningún valor N.A.

```
Numero de variables totales: 191
Numero de variables sin NA: 69
CPU times: user 391 ms, sys: 136 ms, total: 527 ms
Wall time: 2min 8s
```

Figure 3. Resumen de variables N.A

2) ***Exclusividad de datos númericos****:* Este proceso consiste en detectar y procesar las variables categóricas presentes en el *dataframe* mediante el *StringIndexer*. Un *StringIndexer* mapea un set de *strings* a valores númericos seriales $(0..n)$ según la frecuencia del valor original [7]. De las 14 variables categóricas, sólo 2 fueron detectadas luego de la limpieza del paso anterior, para posteriormente ser procesadas. Esto es necesario para la implementación particular de *PySpark*.

```
ID Col: ['customer_ID']
Date Col: ['S_2']
Categorical Cols: ['D_63', 'B_31']
Total # of cols in df: 69
Total # of categorical cols: 2
Total # of other cols: 66
CPU times: user 721 ms, sys: 89.2 ms, total: 810 ms
Wall time: 4min 57s
```

Figure 4. Resumen de clasificación de variables

3) ***Agrupación de variables****:* Este proceso consiste en agrupar todas las variables que describen a cada registro de forma que tengan la forma [X,Y], siendo X el vector de características de la fila e Y el resultado esperado para dicha predicción (*target*). Para ello, *PySpark* proporciona la clase *VectorAssembler* [8], con la que juntaremos todas las variables excluyendo el *customer_id*.

III. Entrenamiento del Modelo y Evaluación

A. Justificación de modelo

La clasificación con árboles de decisión es eficiente cuando las relaciones entre sí son directas y claras. A pesar de que no podemos reconocer las variables, podemos esperar que sea el caso de nuestra colección ya que se trata de un registro real. Tienen la propiedad de distribuir el espacio de características de forma jerarqúica y tomar decisiones simples, lo cual es idóneo para un problema de clasificación binaria, como lo es la predicción de impagos.

La implementación de *PySpark* permite escalar horizontalmente la capacidad de procesamiento en la fase de entrenamiento y clasificación, lo que genera una reducción en los tiempos de ejecución al trabajar con grandes volúmenes de datos [8]. Además, este framework se integra con facilidad al ecosistema virtual de Big Data de *Hadoop* desarrollado para este proyecto.

B. Métrica de evaluación

Se uso el área bajo la curva ROC para medir la precisión del modelo. Esta curva muestra la tasa de verdaderos positivos en el eje Y y la tasa de falsos positivos en el eje X, y el área debajo de esta vária entre 0.5 (clasificacíon pobre) y 1 (clasificación perfecta) [9]. Bajo esta medida, el algoritmo consigió un valor AUC-ROC (precisión) de **0.79**, un valor que se considera aceptable [10].

C. Limitaciones del modelo

Como se mencionó en la etapa de preprocesamiento, el modelo sólo considera 69 de las 190 variables originales, un sesgo producido por la necesidad de no incluir valores N.A y la decisión de no tratar estos casos debido a la anonimización de las etiquetas de los campos en la colección. Esto impacta en la predicción, ya que el modelo tiene menos variables sobre las cuales establecer las reglas jerárquicas que componen su clasificación.

IV. Escalabilidad y Análisis de Rendimiento

A. Tiempo de entrenamiento

Al momento de entrenar el modelo se tomo los tiempos en un ambiente con *Hadoop*, y leyendo del *File System* local. Como se puede observar en la tabla de tiempos. El tiempo de *Hadoop* es ligeramente mayor que en local, esto se puede atribuir a que hay cierta latencia entre el servidor del *cluster* de *Hadoop* y el ambiente de *Kaggle* en donde se esta ejecutando los servicios.

Esto se puede atribuir al hecho que la manera en como se creo la sesión de *PySpark* fue con menor cantidad de recursos en el caso de *Hadoop*. En caso contrario, el *cluster* seria sobrecargado por sus recursos y se dentendria los servicios.

Enviroment	CPU	*Sys*	CPU + *Sys*	*Wall time*
Hadoop	332 ms	45 ms	377 ms	6 min
Local	195 ms	28.3 ms	223 ms	5 min

Tabla 1. Comparativa de tiempos de ejecución

B. Uso de CPU

Midiendo el uso de cpu cuando el *cluster* estaba siendo usado por PySpark tanto en el *worker* 1 5, como en el *worker* 2 6. Se tiene que el uso de CPU no es tan intensivo. Mayormente se notan picos de CPU pero estos no son tan elevados ni

tan consistentes. Teniendo un promedio de uso menor al 10% durante toda la duración de su uso.

Esto se puede intuir debido a que el manejo de *files* son operaciones mas pesadas por el dalo de IO que por el cálculo. Así que la mejor manera de medir el uso del *cluster* es mediante su interacción con la red que por el CPU.

Lo que se puede destacar, uso de tanto CPU como memoria RAM en el ambiente de *Kaggle* se reducía de manera considerable cuando se leí los *files* desde la conexión a *Hadoop*.

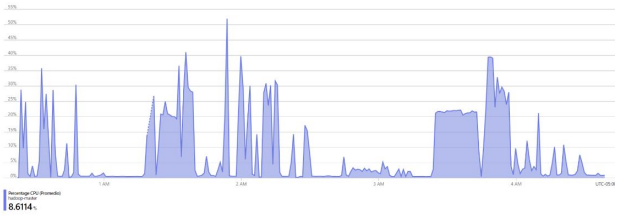

Figure 5. Gráfico de uso de CPU del Nodo *Worker* 1

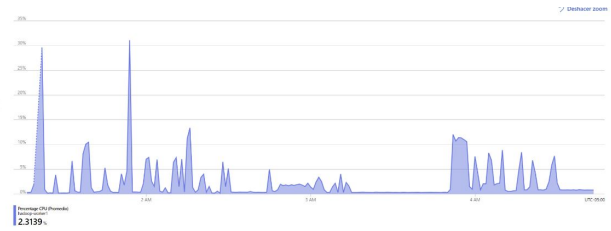

Figure 6. Gráfico de uso de CPU del Nodo *Worker* 2

C. Uso de red

Con respecto al uso de red, tenemos una gráfica caracterizada con picos, en donde la mayor transferencia de datos ocurre. Esos picos normalmente esta agrupados, indicado las partes de mayor uso por *PySpark*. En estos picos GBs de data son transferidos de los *VM* hacia el ambiente en *Kaggle*.

Esto se puede intuir que es cuando *Hadoop* tiene mayor uso. Por lo que nosotros podemos, lamentablemente esto puede incurrir un mayor costo y tiempo dependiendo de donde ocurre la comunicación.

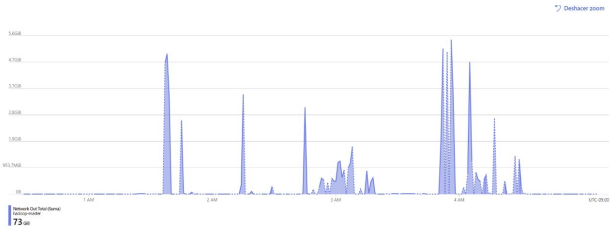

Figure 7. Gráfico de uso de red del Nodo *Worker* 1 y *NameNode*

V. Cierre y conclusiones

En el proyecto, se desarrollo un modelo de predicción de impagos basado en un árbol de decisión alocado en un ambiente de *Big Data* en *Hadoop*. Los resultados medidos

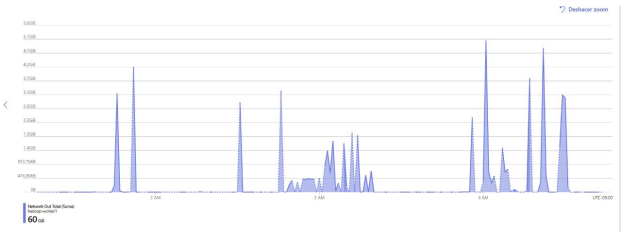

Figure 8. Gráfico de uso de red del Nodo *Worker* 2

con la métrica AUC-ROC determinaron una precisión del 79%. Los sets de entrenamiento y validación fueron particionados en el formato *parquet* con un ratio de compresión promedio de 39.5%, reduciendo el tamaño total de la colección de 50GB a 20GB en el ecosistema del proyecto.

Con respecto a hadoop, las configuraciones de ambiente, entorno y red pueden llegar a afectar notablemente el rendimiento. sin embargo, reduce el peso de los recursos de los ambientes de ejecución. Haciendo que sea una buena opción para cuando se quiera aumentar los recursos de manera horizontal.

En futuros proyectos, se pueden evaluar de forma más específica las variables que presentan valores N.A en la colección, con el fin de no descartarlas en el árbol de decisión e incluir una gama más extensa de reglas jerárquicas al modelo. Además, se pueden hacer pruebas de rendimiento en el ecosistema de *Big Data* para probar con un número mayor de *workers*, algo que no se realizó en el presente proyecto debido a la imposibilidad de hacerlo sin invertir económicamente según las herramientas utilizadas.

Referencias

[1] O'Sullivan A., Sheffrin S. (2003). Economics: Principles in Action. Upper Saddle River, New Jersey 07458: Pearson Prentice Hall. p. 261. ISBN 0-13-063085-3.

[2] Zaharia M., et al. (2010). Spark: Cluster Computing with Working Sets. USENIX Workshop on Hot Topics in Cloud Computing (HotCloud).

[3] Charbuty B., Abdulazeez A. (2021). Classification Based on Decision Tree Algorithm for Machine Learning. Journal of Applied Science and Technology Trends (vol. 2, no. 01, pp.20-28),

[4] American Express. (2022). Default Prediction Competition. https://www.kaggle.com/competitions/amex-default-prediction. Recuperado el 8 de julio del 2023.

[5] Dask Development Team. (2018). Dask DataFrame - Parquet. In Dask Documentation. https://docs.dask.org/en/stable/dataframe-parquet.html. Recuperado el 8 de julio del 2023.

[6] Apache Arrow (2021). https://arrow.apache.org/docs/python/index.html. Recuperado el 8 de julio del 2023.

[7] Apache Spark (2021). StringIndexer - Spark 3.2.0 Documentation. https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StringIndexer.html. Recuperado el 8 de julio del 2023.

[8] Apache Spark. (2021). DecisionTreeClassifier. https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.DecisionTreeClassifier.html. Recuperado el 8 de julio del 2023.

[9] Zou K., O'Malley A., Mauri L. (2007). Receiver-operating characteristic analysis for evaluating diagnostic tests and predictive models. Circulation, 6;115(5):654-7.

[10] Mandrekar, J. N. (2010). Receiver Operating Characteristic Curve in Diagnostic Test Assessment. Journal of Thoracic Oncology, 5(9), 1315-1316. ISSN 1556-0864. https://doi.org/10.1097/JTO.0b013e3181ec173d. Recuperado el 8 de julio del 2023.

[11] *Notebook* del proyecto (es necesario levantar los nodos). https://drive.google.com/file/d/1Q28168cvk5MLjokoa0LTFN7PgiV87Bfd/view?usp=sharing.